A Language for Provenance Access Control [*]

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

Provenance is a directed acyclic graph that explains how a resource came to be in its current form. Traditional access control does not support provenance graphs. We cannot achieve all the benefits of access control if the relationships between the data and their sources are not protected. In this paper, we propose a language that complements and extends existing access control languages to support provenance. This language also provides access to data based on integrity criteria. We have also built a prototype to show that this language can be implemented effectively using Semantic Web technologies.

Categories and Subject Descriptors: D.4.6 [Operating Systems]: Operating Systems —Access Control

General Terms: Languages, Security.

Keywords: Access Control, Provenance, RDF, SPARQL, Regular Expressions, XML.

1. INTRODUCTION

Provenance is the lineage, pedigree and filiation of a resource (or data item) and is essential for various domains including healthcare and intelligence. Provenance can be used to drill down to the source of a medical record or an intelligence report, to track the activities of a doctor or a field agent, and to provide an audit trail that can be used later for validation and verification tasks. Existing access control specifications that define policies for resources do not easily support provenance [3]. Despite the current drawback of provenance to divulge sensitive information, the security of provenance has not been given a high priority in the research community. It is clear that the protection of provenance is required by laws and regulations to avoid the disclosure of sensitive information [10]. For example, in a security intelligence agency, the improper disclosure of the source or the ownership of a piece of classified information may result in great and irreversible losses. Also, many compliance regulations require proper archives and audit logs for electronic records, e.g. HIPAA mandates that we properly log accesses and updates to the histories of medical records [10].

In order to define an access control policy for provenance, it is imperative that we identify the parts of the provenance graph that we want to protect. Therefore, we must have a clear definition of the users, their actions and the resources to be protected. Provenance takes the form of a directed acyclic graph (DAG) that establishes causal relationships between data items [15]. Traditional access control models focus on individual data items whereas in provenance we are concerned with protecting both, the data items and their relationships [3]. In this paper we refer to both, data items and their relationships as resources, to be protected. In order to protect a resource we need to first identify it in the provenance graph. This identification is one of the major distinguishing factors between a provenance access control model and existing access control models. The provenance graph structure not only poses challenges to access control models but also to querying languages [11]. The various paths in a provenance graph from a resource to all its sources are important in proving the validity of that resource. Furthermore, these paths contain the pertinent information needed to verify the integrity of the data and establish trust between a user and the data; however, we do not want to divulge any exclusive information in the path which could be used by an adversary to gain advantages, for example in military intelligence.

We need appropriate access control mechanisms for provenance that prevent the improper disclosure of any sensitive information along a path in the provenance graph. We need to extend the traditional access control definition that protects a single data item to one where we now want to protect any resources along a path of arbitrary length. In this paper, we propose a policy language that extends the definition of traditional access control languages to allow specification of policies over data items and their relationships in a provenance graph. This language will allow a policy author to write policies that specify who accesses these resources. The language provides natural support for traditional access control policies over data items. We motivate this idea with the following example. Consider a medical example where we may want to give access to everything in a patient's record that was updated by processes controlled only by the pa-

[*] This work was partially supported by Air Force Office of Scientific Research MURI Grant FA9550-08-1-0265, National Institutes of Health Grant 1R01LM009989, National Science Foundation Grants Career-0845803, CNS-0964350, and CNS-1016343.

CODASPY'11, February 21–23, 2011, San Antonio, Texas, USA.
Copyright 2011 ACM 978-1-4503-0465-8/11/02 ...$10.00.

tient's physician and surgeon. For this example, the system would evaluate two policies. The first policy would check if the user has access to the medical record. This policy would be applied over all the medical records in the system with the traditional access control policies in place. The second policy would check if the patient's medical record has indeed only been modified by the patient's physician and surgeon. This second policy would be applied over the provenance graph associated with the given medical record. This example not only shows how existing access control policies can be integrated in our language, but also how traditional access control can be used to allow access to provenance.

The traditional definition of access control policies is extended in our policy language to include relationships over data items in the provenance graph by making use of regular expressions. The use of an existing access control language to build policies over the provenance graph would require enumerating all the possible paths that we want to protect in the graph as separate policies. The use of regular expressions in our language not only solves this problem, since many paths can be specified using the same regular expression, but also allows the same policy to be applied to multiple provenance graphs. In contrast to the first example these regular expressions can be used to first verify the quality of the data items and second, act as a "pseudo" access control mechanism for giving data access to the user. Again, we use the following example to motivate this idea. Consider a military scenario where access to an intelligence report can only be given to a user if the report was created by a particular field agent belonging to a specific agency in a particular country. In this example, the system would evaluate the regular expression in the policy over the provenance graph for the given intelligence report to check if that report was indeed created by the specified field agent belonging to the given agency in the specified country. If such a path exists in the provenance graph only then access is granted to the querying user for the report. This example emphasizes how provenance can be used to first determine integrity of the data in order to guarantee high quality information before access is given to the actual data items.

Our main contribution in this paper is the definition of an access control policy language for provenance. This language retains the properties of traditional access control to gain access to data. Furthermore, the language provides an additional advantage whereby provenance not only acts as an access control mechanism but also as an integrity mechanism for giving access to the data. We also build a prototype using Semantic Web technologies that allows a user to query for data and provenance based on access control polices defined using our policy language.

The rest of the paper is organized as follows. Section 2 gives a basic idea of access control and shows the drawbacks of using existing access control models for provenance. Section 3 provides a formalism to represent access control policies for provenance. In Section 4 we present existing storage mechanisms for provenance and we show how we can use regular expressions to support our policy language using Semantic Web technologies. In section 5 we show how the complexity changes from that of protecting single resources to that of protecting resources over relationships in a provenance graph. Section 6 presents an architecture which incorporates our policy language in an access control prototype system. Section 7 reviews previous work in the area of access control for provenance. In closing, in Section 8 we provide our conclusions and future work.

2. ACCESS CONTROL

An access control system has three levels of abstraction [23]:

1. Policy. This is a high level requirement that specifies how access is managed and who, under what conditions, may access a resource.

2. Mechanism. This implements the regulations established by a policy.

3. Model. This is a formal representation of a policy. The model allows the verification of the security properties provided by the system.

An access control policy authorizes a set of *users* to perform a set of *actions* on a set of *resources* within an *environment*. Unless authorized through one or more access control policies, users have no access to any resource of the system. There are many access control policies defined in the literature. These can be grouped into three main classes [23], which differ by the constraints they place on the sets of *users*, *actions* and *objects* (access control models often refer to *resources* as *objects*). These classes are (1) RBAC, which restricts access based on roles; (2) DAC, which controls access based on the identity of the user; and (3) MAC, which controls access based on mandated regulations determined by a central authority. There are two major concerns with these policies. The first is the number of user to object assignments and the second is that these policies are defined over a single resource.

Role-Based Access Control (RBAC) models have enjoyed popularity by simplifying the management of security policies. These models depend on the definition of roles as an intermediary between users and permissions (which is a combination of actions and objects). The core model defines two assignments: a user-assignment that associates users to roles and a permission-assignment that associates roles with permissions. In [8], the authors argue that there is a direct relationship between the cost of administration and the number of mappings that must be managed. The drawbacks with using RBAC include, (i) each time a user does not have access to an object through an existing role, a new role is needed; and (ii) as the policies become more fine-grained, a role is needed for each combination of the different resources in the provenance graph [22]. Similar drawbacks apply to the DAC and MAC access control models since they both use mapping functions to associate users with objects.

Clearly, applying these traditional access control policies for fine-grained access control in provenance would result in prohibitive management costs. Moreover, their usage in provenance would be an arduous task for the administrator. In Section 5, we provide an analysis, which shows that the number of resources in a provenance graph is exponential in the number of nodes in the graph. We address these drawbacks in this paper and provide an implementation of a prototype mechanism, which shows that we can greatly reduce these mappings.

In summary, the general expectations of an access control language for provenance are (i) to be able to define policies over a directed acyclic graph; (ii) to support fine-grained access control on any component of the graph; and (iii) to seamlessly integrate existing organizational policies.

3. POLICY LANGUAGE

A generalized language that extends existing access control languages such as XACML [14] was proposed in [17]. We extend the syntax of this XML-based policy language in order to incorporate regular expressions in a policy. The existing provenance language in [17] was developed as a generalized model of access control for provenance, but did not address resources with arbitrary path lengths within the provenance graph. Therefore, this language suffers from the fact that a resource must be identified before hand, rather than be given as a string which is matched against the graph at execution time. An example of our adaptation of the language in [17] is given in Figure 1, which now allows the policy to be written using the regular expression syntax. We place an emphasis on the target, effect and condition elements given in [17], but make slight modifications to their meanings to incorporate regular expressions on a provenance graph. Since our focus in this paper is on specifying a policy for access control in provenance, we provide only the relevant XML elements in this paper. The interested reader can find other interesting elements of the language, such as obligation and originator preference, in [17].

```
<policy ID="1" >
   <target>
     <subject>anyuser</subject>
     <record>Doc1_2</record>
     <restriction>
        Doc1_2 [WasGeneratedBy] process AND
        process [WasControlledBy] physician|surgeon
     </restriction>
     <scope>non-transferable</scope>
  </target>
  <condition>purpose == research</condition>
  <effect>Permit</effect>
</policy>
```

Figure 1: Policy language

The description of each element in Figure 1 is as follows: The **subject** element can be the name of a user or any collection of users, e.g. physician or surgeon, or a special user collection *anyuser* which represents all users. The **record** element is the name of a resource. The **restriction** element is an (optional) element which refines the applicability established by the subject or record. The **scope** element is an (optional) element which is used to indicate whether the target applies only to the record or its entire ancestry. The **condition** element is an (optional) element that describes under what conditions access is to be given or denied to a user. The **effect** element indicates the policy author's intended consequence for a true evaluation of a policy.

The scope element is useful, in particular, when we want to protect the record only if it is along a specified path in the provenance graph. This is achieved by using the predefined value "non-transferable". This element can also be used when we need to protect a path in the provenance graph if a particular record is along that path. This is achieved by the predefined value "transferable". The condition element is necessary when we want to specify system or context parameters for giving access, e.g. permitting access to provenance when it is being used for research. It is important that we keep the number of policies to a minimum by combining them using regular expressions. This will improve the effectiveness of an access control system that protects the sensitive information from unauthorized users. It was also pointed out in [17] that when the policy size is not small, detecting abnormal policies is essentially a SAT problem. The reason is that the effects of different semantics for the predicates used in the condition and restriction elements may cause incorrect policy specifications, which may generate conflicting or redundant policies.

We achieve fine-grained access control by allowing a record value to be any (indivisible) part of a provenance graph. The regular expressions in the "restriction" element allow us to define policies over paths of arbitrary length in a provenance graph that apply to a subject or record. Also, since XML is an open and extensible language, our policy language is both customizable and readily supports integration of other policies.

3.1 The Grammar

In this section, we define a grammar for each of the tags in the language we propose.

```
<exp> ::= <char>+ ("." <char>+)?
<char> ::= [a-z] | [A-Z] | "_" | "-" |
<reg> ::= "*" | "+" | "?"
<bool> ::= " AND " | " OR " | "|"
<op> ::=  " == " | " <= " | " >= " | " < " | " > "
<num> ::= ([0-9])+
<sp>  ::= "[" <exp> "]"
```

We now define the set of strings accepted by each element in our language.

```
subject =  <char>+ | <num>

record =  <exp>

restriction =  (<exp><num>?)+ (<op> | <sp><reg>?)
                (<exp><num>?)+
              (<bool> (<exp><num>?)+
              (<op> | <sp><reg>?)? (<exp><num>?)+)*

scope =  <char>+

condition =  (<exp><num>?)+ (<op> | <sp><reg>?)
             (<exp><num>?)+
           (<bool> (<exp><num>?)+
           (<op> | <sp><reg>?)? (<exp><num>?)+)*

effect =  <char>+ | <num>
```

The grammar defined above allows us to evaluate the policy for correctness and secondly, allows a parser to unambiguously translate the policy into a form that can be used by the appropriate layer in our architecture.

4. DATA REPRESENTATION

We require a suitable data representation for storing provenance. Such a data representation must naturally support the directed graph structure of provenance and also allow path queries of arbitrary length. The Open Provenance Model (OPM) [15] does not specify protocols for storing or querying provenance information; but it does specify properties that any data model should have. One such property includes allowing provenance information to be shared among systems. Provenance data can be stored in the relational database model, the XML data model or the RDF data model [12]. Each of these in their current form has drawbacks with respect to provenance [11]. A relational model suffers from the fact that it needs expensive joins on relations (tables) for storing edges or paths. Also, current SQL languages that support transitive queries are complex and awkward to write. XML supports path queries, but the current query languages XQuery and XPath only support a tree structure. RDF naturally supports a graph structure, but the current W3C Recommendation for SPARQL (the standard query language for RDF) lacks many features needed for path queries. There are recent works on extending SPARQL with path expressions and variables. These include SPARQL Query 1.1 [9] which is now part of a proposal put forward by the W3C recently. The SPARQL 1.1 query language includes new features such as aggregates, subqueries, property paths, negation and regular expressions, but this is still a W3C Working draft as of this writing.

In the case of access control in provenance we may have two different sets of access control policies, one for traditional access control and one for provenance access control. This may result in the management of two different sets of policies if both, the traditional data items and provenance are placed in the same data store. If we allow this scenario, all requests from a user would be evaluated against both, the policies for the traditional access control and the policies for provenance. This would be the case even when the user is only working with the traditional data, and is not requesting the provenance information. In general, the lineage or ancestry of a data item may involve many sources and processes that influence a resource. Recording all these sources and paths may result in very large databases. Therefore, provenance may grow much faster than the actual data items and may be better served by a separate database. To this end, we will use a separate data store for provenance in our design of an architecture and prototype for provenance.

4.1 Graph Data Model

Of the many data models in the literature, we model our prototype based on a RDF data representation for provenance. This data model meets the specification of the OPM recommendation. RDF allows the integration of multiple databases describing the different pieces of the lineage of a resource; and naturally supports the directed structure of provenance.

The RDF terminology $\mathcal{T}$ is the union of three pairwise disjoint infinite sets of terms: the set $\mathcal{U}$ of URI references, the set $\mathcal{L}$ of literals (itself partitioned into two sets, the set $\mathcal{L}_p$ of plain literals and the set $\mathcal{L}_t$ of typed literals), and the set $\mathcal{B}$ of blanks. The set $\mathcal{U} \cup \mathcal{L}$ of names is called the vocabulary.

Definition 1. *(RDF Triple) A RDF triple* (s, p, o) *is an element of* $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times \mathcal{T}$ *. A RDF graph is a finite set of RDF triples.*

A RDF triple can be viewed as an arc from s to o, where p is used to label the arc. This is represented as $s \xrightarrow{p} o$. Our provenance graph is constructed from a set of these RDF triples. RDF is intended to make assertions about a resource. This includes making multiple assertions about the same two resources; for example, a heart surgery h was controlled by a surgeon s, and the inverse relation: s performed a heart surgery h. This would be modeled as a directed loop in a RDF graph. In order to preserve the properties of a provenance graph, we need to place restrictions on the assertions made in a RDF graph. That is, we require a directed acyclic RDF graph to retain the causal dependencies among the nodes as needed in provenance.

Definition 2. *(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.*

We define a resource in a provenance graph recursively as follows.

- The sets V_G and E_G are resources.
- ϵ is a resource.
- The set of provenance graphs are closed under intersection, union and set difference. Let H_1 and H_2 be two provenance graphs, then $H_1 \cup H_2$, $H_1 \cap H_2$ and $H_1 - H_2$ are resources, such that if $t \in H_1 \cup H_2$ then $t \in H_1$ or $t \in H_2$; if $t \in H_1 \cap H_2$ then $t \in H_1$ and $t \in H_2$; or if $t \in H_1 - H_2$ then $t \in H_1$ and $t \notin H_2$.

4.2 Provenance Vocabulary

We define the nodes in the provenance graph using the nomenclature in [15]. This nomenclature defines three entities: artifacts, processes and agents. These entities form the nodes in V_G in our provenance graph G. An artifact is an immutable piece of state, which may have a physical embodiment in a physical object, or a digital representation in a computer system [15]. A process is an action or series of actions performed on or caused by artifacts and resulting in new artifacts [15]. An agent is a contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, affecting its execution [15]. In RDF representation, an artifact, a process and an agent could be represented as,

```
<opm:Agent> <rdf:type> <opm:Entity>
<opm:Artifact> <rdf:type> <opm:Entity>
<opm:Process> <rdf:type> <opm:Entity>
```

The property rdf:type is used to indicate the class of a resource and the prefix opm: is reserved for the entities and relationships in the OPM nomenclature in [15].

Let $\mathcal{V}_G$ be the set of names appearing in a provenance graph G and $\mathcal{V}_G^P \subseteq \mathcal{V}_G$ be a set of names on the arcs in G. The label on each $e \in \mathcal{V}_G^P$ defines a relationship between the entities in G and also allows us to navigate across the different nodes by a single hop. A list of predicate names in $\mathcal{V}_G^P$ describing the causal relationships among the nodes in G are 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>
```

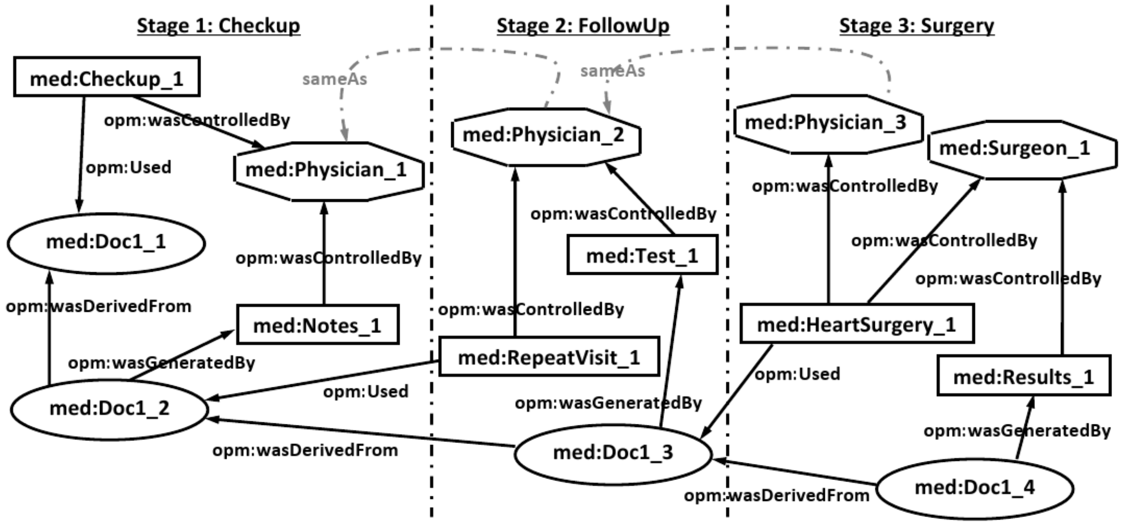


Figure 2: Provenance Graph

These predicates are the ones defined in [15] and they form the edges in our edge set, E_G, in our provenance graph G.

Definition 3. *(Path) A path in a RDF graph is a sequence of RDF triples, where the object of each triple in the sequence coincides with the subject of its successor triple in the sequence.*

Definition 4. *(Provenance Path) In G, a provenance path (s ρ o) is a path* $s(\xrightarrow{\rho})o$ *that is defined over the provenance vocabulary* $\mathcal{V}_G^P$ *using regular expressions.*

Definition 5. *(Regular Expressions) Let Σ be an alphabet of terms in* $\mathcal{U} \cap \mathcal{V}_G^P$*, then 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.

Our intention is to define paths between two nodes by edges equipped with * for paths of arbitrary length, including length 0 or + for paths that have at least length 1. Therefore, for two nodes x, y and predicate name p, $x(\xrightarrow{p})^*y$ and $x(\xrightarrow{p})^+y$ are paths in G.

The provenance graph in Figure 2 shows a workflow which updates a fictitious record for a patient who went though three medical stages at a hospital. In the first phase, the physician performed a checkup on the patient. At checkup, the physician consulted the history in the patient's record, med:Doc1_1 and performed the task of recording notes about the patient. At the end of the checkup, the physician then updated the patient's record, which resulted in a newer version, med:Doc1_2. In the second phase, the patient returned for a follow-up visit at the physician's request. During this visit, the physician consulted with the patient's record for a review of the patient's history and then performed a series of tests on the patient. At the end of this visit, the physician then updated the patient's record, which results in a newer version, med:Doc1_3. In the third phase, the patient returned to undergo heart surgery. This was ordered by the patient's physician and carried out by a resident surgeon. Before the surgeon started the surgery, a careful review of the patient's record was performed by both the patient's physician and surgeon. During the surgery process, the surgeon performed the task of recording the results at each stage of the heart surgery process. At the end of the surgery, the patient's record was updated by the surgeon, which resulted in a newer version, med:Doc1_4. The number in the suffix at the end of each process, agent and artifact is only meant to show an implicit condition whereby a larger number means that the provenance entity is at a later stage in the workflow process. The sameAs annotations on the light shaded arrows are meant to illustrate that the reference to physician is meant to be the same person in all the three phases. We use Figure 2 as a running example through the rest of the paper.

4.3 Path Queries

SPARQL is a RDF query language that is based around graph pattern matching [19].

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 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 does not support paths of arbitrary length [7]; therefore, extensions are needed to answer the queries over the provenance graph. Many approaches to supporting paths of arbitrary length have been proposed in the literature, which include [7, 2, 13]. A W3C working draft for extending SPARQL to support property paths can be found in [9].

We use the following basic SELECT query structure to map a regular expression that is part of a policy or part of a user provenance query into a query over the provenance graph.

$$\text{SELECT } \vec{B} \text{ WHERE P,}$$

where P is a graph pattern and $\vec{B}$ is a tuple of variables appearing in P.

Using regular expressions as part of the SELECT query above we can answer our policy example that gives access to everything in a patient's record that was updated by processes controlled only by the patient's physician or surgeon. This would evaluate the following regular expression query on the provenance graph:

```
Select ?x
{
 med:Doc1_4 gleen:OnPath("([opm:WasDerivedFrom]*/
             [opm:WasGeneratedBy]/
             [opm:WasTriggeredBy]*/
             [opm:WasControlledBy])" ?x).
}
```

The gleen:OnPath function [7] is used to determine the set of nodes on the provenance path between med:Doc1_4 and ?x. We can think of s, ρ and o for the provenance path, $s(\xrightarrow{\rho})o$, as placeholders for med:Doc1_4, the expression given to the gleen:OnPath function and the variable ?x respectively.

4.4 Query Templates

We can use the set of names in $\mathcal{V}_G$ to answer common queries about provenance such as why-provenance, where-provenance and how-provenance [16]. To anticipate the varying number of queries a user could ask, we create templates which are parameterized for a specific type of user query. This simplifies the construction of queries by allowing us to map a user query to a suitable template. This in turn allows us to build an interface through which a user could interact with the system, as well as create an abstraction layer which hides the details of the graph from the user.

Example 1. ***(Why Query)***

```
med:Doc1_2 gleen:OnPath("([opm:WasDerivedFrom] |
      [opm:WasGeneratedBy] | [opm:WasTriggeredBy] |
      [opm:WasControlledBy] | [Used])*" ?x).
```

This allows us to specify all the resources reachable from med:Doc1_2 by issuing a query against the provenance graph. This query explains why med:Doc1_2 came to be in its current form. Figure 3(c) shows the part of the graph from Figure 2 that would result from executing this why-provenance query.

Example 2. ***(How Query)***

```
compute leaf-set for med:Doc1_3
for each XXX in leaf-set
    computeFreq(XXX)
```

The modified Gleen API [7] allows us to compute the leaf-set given the starting resource med:Doc1_3. Each leaf in the leaf-set {med:Physician_1, med:Physician_2, med:doc1_1} is the end of a unique path from med:Doc1_3 to that leaf. We then compute the frequency of each leaf in the leaf-set. This query would return the following polynomials:

```
med:Physician_2^1, med:Physician1_1^1, med:Doc1_1^1
```

A how-provenance query returns a polynomial representation of the structure of the proof explaining how the resource was derived. This normally involves counting the number of ways a provenance entity influences a resource.

Example 3. ***(Where Query)***

```
med:Doc1_4 gleen:OnPath("([opm:WasDerivedFrom] |
      [opm:WasGeneratedBy])" ?x).
```

This query would return the following triples:

```
(med:Doc1_4, opm:WasDerivedFrom, med:Doc1_3)
(med:Doc1_4, opm:WasGeneratedBy, med:Results_1)
```

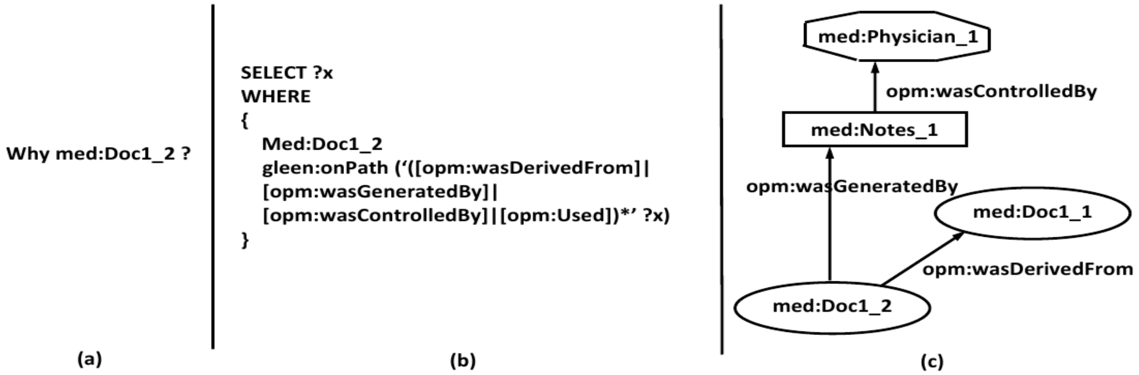
A where query would be useful if we need to pinpoint where in the process a possible risk could occur as a result of performing a surgery on the patient. For example, a where-provenance could be used to identify at which phase in the flow any medication administered to the patient had a negative interaction with the ones the patient is already taking. By using this query, we could compare the information in med:Doc1_3 with those in med:Doc1_4 (which incorporates the recording of events during the surgery operation).

The Open Provenance Model [15] in general allows us to extend $\mathcal{V}_G$ to support annotations on the nodes and edges in our provenance graph. These annotations allow us to capture additional information relevant to provenance such as time and location that pertain to execution. The annotations are not part of the vocabulary provided by OPM. The idea of not providing annotations as part of the predicate vocabulary is to allow a user the flexibility of creating his/her own vocabulary for the nodes and edges. The annotations themselves can be added as RDF triples since RDF allows us to make assertions about any node in a RDF graph. This allows us to capture more contextual information about resources, which would allow us to model the provenance information to capture the semantics of the domain. While a particular causal relation, such as process P_2 was triggered by process P_1, may imply that P_1 occurs before P_2 on a single logical clock, it does not tell us the exact physical time both processes occur. Such additional information plays a critical role in the intelligence domain. These additional annotations allow us to build more templates, which give our prototype the ability to respond to queries like: when was a resource generated, what was a resource based on, which location a resource was created or modified at, etc. We show a simple example of a when query below,

Example 4. ***(When Query)***

```
Select ?x
{
  med:Doc1_4 med:modifiedOn ?x.
}
```

This query would return the timestamp value as a binding for the variable ?x, if the graph pattern in the where clause successfully matches a triple in the extended annotated provenance graph.


Why med:Doc1_2 ?

```
SELECT ?x
WHERE
{
   Med:Doc1_2
   gleen:onPath ('([opm:wasDerivedFrom]|
   [opm:wasGeneratedBy]|
   [opm:wasControlledBy]|[opm:Used])*' ?x)
}
```

Figure 3: Why Query

5. GRAPH ANALYSIS

In this section, we will evaluate the impact of querying over a provenance graph with many subgraphs as resources. We will first address the complexity of protecting the resources in a provenance directed acyclic graph (digraph), then we will examine the case where two digraphs overlap, which may conflict with each other.

5.1 Analysis of Digraphs

We now provide a simple analysis addressing the concerns from Section 2 of traditional access control policies. We use the convention that a permission is a unique pair of (*action*, *resource*). Given n resources, m users and a set of only two actions (read, write), we have a maximum of $2 \times n$ possible permissions. This gives $m \times (2 \times n) = c_1 n$ mappings. To analyze RBAC, we assume the case where there is at least one role with two or more users assigned to it, from a possible set of r roles. Therefore, we have $r \times (2 \times n) = c_2 n$ mappings and we also assume that $c_2 \leq c_1$.

We continue our analysis by considering the varying number of relationships among the resources in a provenance graph. We assume that we have n nodes in our graph G. The first case is when the provenance paths are of length 0. This is similar to the case of access control policies over single resources. Next we consider the case where the provenance paths are of length 1. This is equivalent to counting the number of edges in E_G. We use the notion that a resource is a set of triples in G, and therefore; a resource is a directed acyclic graph (or digraph) from among all the allowed digraphs that can be formed from G. In general, the total number of ways of constructing a digraph from n nodes is given recursively as

$$a_n = \sum_{k=1}^{n} (-1)^{k-1} \left(\begin{array}{c} n \\ k \end{array} \right) 2^{k(n-k)} a_{n-k} \qquad (5.1)$$

in [20, 21]. Given n nodes in a provenance graph G, a_n would represent the upper limit of resources to be protected in G. The work done in [20] shows that the number of ways of constructing a directed acyclic graph is exponential in the size of n single resources.

In general, a node in a digraph can have both, an in-degree and an out-degree. OPM restricts the relationships we can have among the nodes in a provenance graph (see [15] for a formal definition of a provenance graph). This restriction is on the dependency relationships involving agents; in simple terms the only relation involving an agent is a directed edge from a process to an agent. That is, agents in a provenance graph can only have an in-degree. Although, this restriction limits the maximum number of resources to be protected (as given in equation 5.1) by a factor, the upper bound for the maximum number of digraphs is still exponential. The OPM specification for a provenance graph describes how to trace an artifact or process back to their direct source (or cause), which could be a process, an artifact or an agent, using the edges in the graph. It does not however provide a standard arc name which explains the causes or sources for an agent in the graph. Therefore, a more useful definition of provenance according to OPM in the context of our analysis, would describe how an artifact or process came to be in their current form. This definition is still consistent with the ones in the literature. Hence, even in the cases where we only consider $n^{'}$ artifacts and processes in our provenance graph, where $2 \leq n^{'} \leq n$, the number of digraphs is still exponential in $n^{'}$.

A traditional access control policy would first require identifying a provenance path and then, expressing a policy for each of the resources on this path. The regular expressions presented in Section 4 allow us to specify a pattern for resources that need to be protected with an access control policy. Since a regular expression pattern can match many paths (each of arbitrary length), we can replace all policies that protect a resource on any of these paths with one policy.

5.2 Composition of Digraphs

Access control systems normally contain policies that are used to handle situations where two policies have opposite values for the *effect* element of a policy. This happens when one policy has a permit (or +ve authorization) effect whenever it evaluates to true, while another policy has a deny (or -ve authorization) whenever it evaluates to true and both of these policies protect the same digraph. The conflict could be as a result of two policies overlapping with each other to form a common digraph or when a digraph associated with a -ve authorization overlaps with a digraph that results from the execution of a user's query.

Different conflict resolution policies [23] have been proposed to resolve conflicts that result from opposite access authorizations on a resource. These policies include Denials-take-precedence, Most-specific-takes-precedence and Most-specific-along-a-path-takes-precedence.

There are three possibilities that could occur when two digraphs overlap with each other. We will discuss these possibilities when the Denials-take-precedence conflict resolution policy is applied.

1. $G1 \subseteq G2$: The digraph $G1$ is associated with a policy that denies viewing its contents and the digraph $G2$ is associated with a policy that permits viewing of its contents. In this situation, the system would have the effect of permitting viewing of the digraph $G2 - G1$.

2. $G1 \supseteq G2$: The digraph $G1$ is associated with a policy that denies viewing its contents and the digraph $G2$ is associated with a policy that permits viewing of its contents. In this situation, the user would be denied from viewing the contents of both, $G1$ and $G2$.

3. $G1 \cap G2$: The digraph $G1$ is associated with a policy that denies viewing its contents and the digraph $G2$ is associated with a policy that permits viewing its contents. In this situation, the system would have the effect of denying access to digraphs $G1$ and $G1 \cap G2$.

These three cases also apply when a user's query execution returns the digraph, $G2$, and the *effect* of the policy for $G1$ is "deny".

6. ARCHITECTURE

Our system architecture assumes that the available information is divided into two parts: the actual data and provenance. Both, the data and provenance are represented as RDF graphs. The reader should note that we do not make any assumptions about how the actual information is stored. A user may have stored data and provenance in two different triple stores or in the same store. Access control policies are defined in our XML-based language for both, the data and the provenance. These policies define access for users on resources in the data graph and on agents, artifacts, processes and paths in the provenance graph. A user application can submit a query for access to the data and its associated provenance or vice versa. In this discussion we first present the various modules in our prototype implementation. We then give an example of a scenario where the user already has access to the data item and is requesting additional information from the provenance. The same logic applies when we want to give high quality information to a user, where we would first verify the information against the provenance store before allowing access to the data item.

6.1 Modules in our Architecture

We now present a detailed description of the different layers in our architecture followed by an example.

User Interface Layer
The User Interface Layer is an abstraction layer that allows a user to interact with the system. A user can pose either a data query or a provenance query to this layer. This layer determines whether the query should be evaluated against the data or provenance. Our interface hides the use of regular expression queries (i.e., the actual internal representation of a provenance query) from a user by providing a simple question-answer mechanism. This mechanism allows the user to pose standard provenance queries such as why a data item was created, where in the provenance graph it was generated, how the data item was generated and when and what location it was created, etc. We show an example of a provenance query in Figure 3(a) that a user would pose to the system. This layer also returns results after they have been examined against the access control policies.

Access Control Policy Layer
The Access Control Policy Layer is responsible for ensuring that the querying user is authorized to use the system. It also enforces the access control policies against the user query and results to make sure that no sensitive information is released to unauthorized users. This layer also resolves any conflicts that resulted from executing the policies over the data stores. An example of a provenance policy that can be used in this layer is given in Figure 1.

Policy Parser Layer
The Policy Parser Layer is a program that takes as input a policy set and parses each policy to extract the information in each element. The parser verifies that the structure of the policy conforms to a predefined XML schema. Further, the parser also validates the value of each element in a policy using the grammar specified in Section 3.1.

Regular Expression-Query Translator
The Regular Expression-Query Translator takes a valid regular expression string and builds a corresponding graph pattern from these strings. This module works in two ways. First it associates a provenance query from a user to a corresponding template query, by invoking the necessary parameters associated with the user's provenance query, for example, Figure 3(a) shows a user query and the corresponding translation in Figure 3(b).

Data Controller
The Data Controller is a suite of software programs that store and manage access to data. The data could be stored in any format such as in a relational database, in XML files or in a RDF store. The controller accepts requests for information from the access control policy layer if a policy allows the requesting user access to a data item. This layer then executes the request over the stored data and returns results back to the access control policy layer where it is re-evaluated based on the access control policies.

Provenance Controller
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 sub-graph back to the access control layer where it is re-examined using the access control policies.

We show an example of how a user query and a policy query are executed in our prototype system. The user query given in Figure 3(a) is submitted to the User Inter-

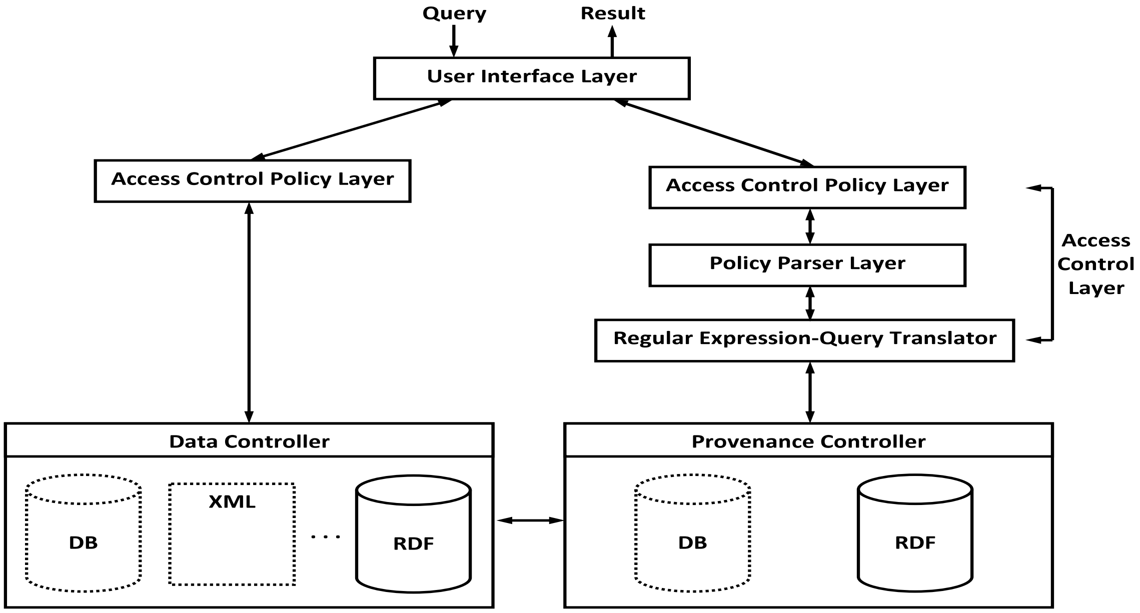
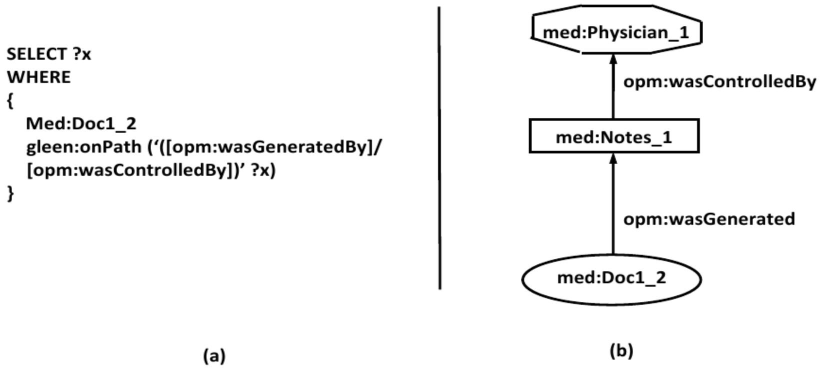


Figure 4: Architecture



```
SELECT ?x
WHERE
{
   Med:Doc1_2
   gleen:onPath ('([opm:wasGeneratedBy]/
   [opm:wasControlledBy])' ?x)
}
```

Figure 5: A resource protected by a policy

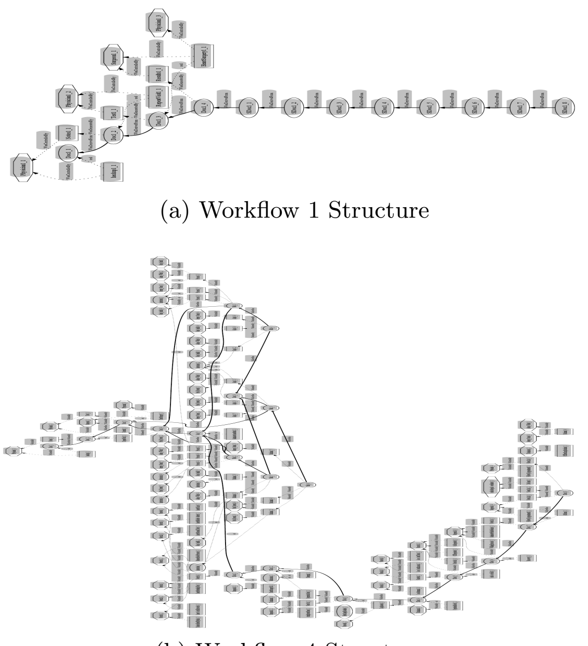
face Layer. This query asks for a complete explanation of why Doc1_2 came to be in existence. Doc1_2 is an internal node in the example provenance graph. This means that the user would have had access for the actual patient record in the traditional database before submitting a query about its provenance. Our Regular Expression-Query Translator in the Access Control Layer would transform this query into the query shown in Figure 3(b). The result of executing this query against the provenance graph shown in Figure 2 returns the results shown in Figure 3(c). This result is passed back to the Access Control Policy Layer. This layer also passes the policy given in Figure 1 to the Policy Parser Layer that parses the policy against a XML schema and the grammar given in Section 3.1. If the policy is well constructed, it is passed to the Regular Expression-Query Translator Layer that constructs the query given in Figure 5(a). This query is also evaluated against the provenance graph in Figure 2. The result of this query execution would return the digraph shown in Figure 5(b). This digraph represents the resource that the policy is protecting and is returned back to the Access Control Layer. The Access Control Layer would then compare the resource from Figure 3(c) with the digraph in Figure 5(b). Since the digraph in Figure 3(c) contains the digraph in Figure 5(b), the Access Control Policy Layer would need to execute the *effect* that is given in the policy. Since in this case, the *effect* is Permit, the results in Figure 3(c) are passed to the User Interface Layer which in turn will return the results to the user. For the second case where we want to verify the integrity of the data, the process will be the same as described above, except that the user query would be about a leaf node stored in the traditional database and this leaf node is the last node of an ancestral chain in provenance.

6.2 Prototype

To implement the layers in our architecture we use various open-source tools. To implement the Access Control Layer, we use the policy files written in XML 1.0, and Java 1.6 to write the logic that enforces the policies. 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 was also pro-

Table 1: Workflows

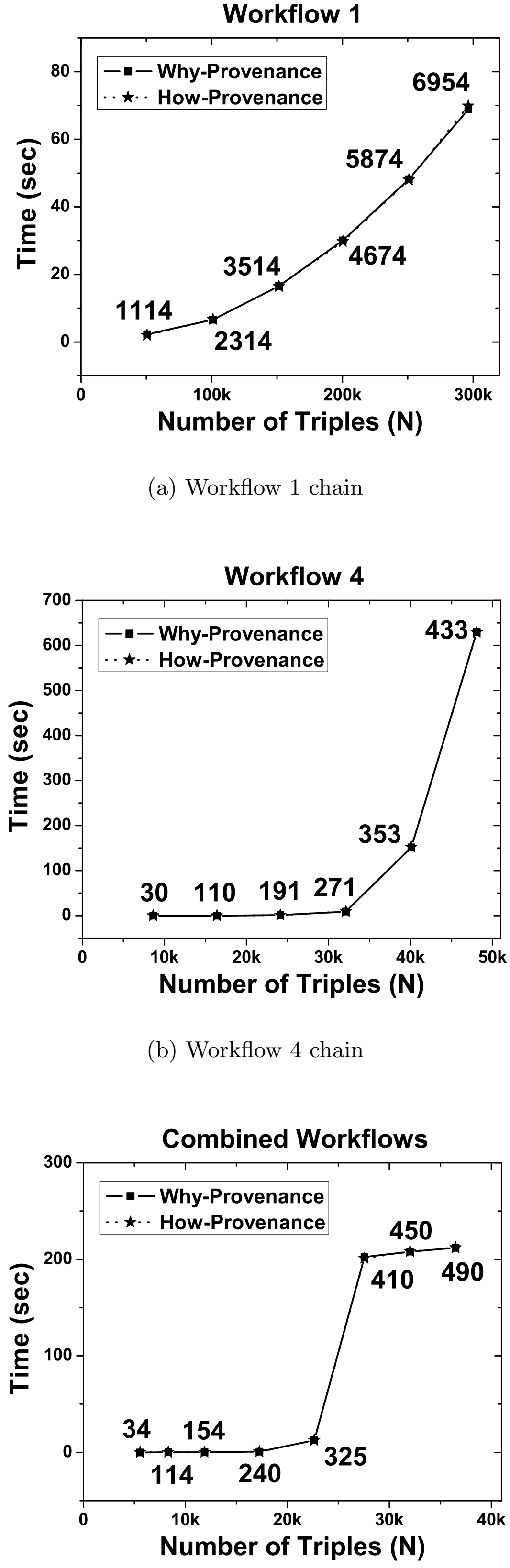
Workflow #	1	2	3	4
Diameter(longest path)	13	13	13	13
no. of Artifacts	4	13	24	22
no. of Processes	6	41	23	58
no. of Agents	4	25	26	46
Annotated Graph(triples)	5347	7214	6743	8533



(a) Workflow 1 Structure

(b) Workflow 4 Structure

Figure 6: Workflow Topologies



(a) Workflow 1 chain

(b) Workflow 4 chain

(c) Composite chain

Figure 7: Query Performance

grammed to apply the grammar in Section 3.1 against the values in the elements in the policies stored in the policy file. To implement the Regular Expression-Query Translator, we use the Gleen[1] regular expression library. This library extends SPARQL to support querying over a RDF graph using regular expressions [7]. To create the provenance store, we use the OPM toolbox[2]. This toolbox allows us to programmatically build workflows that use the OPM vocabulary and also allows us to generate RDF graphs corresponding to the workflow (with some tweaking to generate the RDF graphs for this prototype). There are other tools which support automatic provenance generation such as Taverna [18], but they are not as easy to use as the OPM toolbox. We use the OPM vocabulary which is based on RDF rather than existing vocabularies which have support for a more expressive representation of provenance, for example the vocabulary specification in [25]. Our aim in this paper is to demonstrate a general way of navigating a provenance graph, rather than capturing the semantics of the domain associated with the provenance paths.

We use synthetic data to build in-memory models, using the Jena API[3][5]. This tool allows us to add annotations

[1] `http://sig.biostr.washington.edu/projects/ontviews/gleen/index.html`

[2] `http://openprovenance.org/`

[3] `http://jena.sourceforge.net/`

to the existing RDF triples generated from executing the provenance workflows. We then issue different provenance queries, such as why, where, how, when and who against each of the provenance graphs in the in-memory Jena model. We have used in-memory models to simply show the feasibility of our prototype; however, in a real world scenario our prototype could query the provenance graphs stored in any disk-based storage.

6.3 Experiments

We generated four template workflow structures, each consisting of a varying number of provenance entities and RDF triples (as annotations). The composition of each workflow is shown in Table 1. Each workflow template has a different topology, with workflow 1 being the simplest and workflow 4 being the most intricate; Figure 6(a) and Figure 6(b) show the topology of workflow 1 and workflow 4 respectively.

We conducted three different experiments with our prototype using only workflow 1, only workflow 4 and a composite workflow consisting of all four workflows. For our experiments we varied the size of the in-memory models as shown in Figure 7. To vary the size of the experiment with workflow 1, we daisy chain a set of workflow 1's and record the time to perform each query. The results are shown in Figure 7(a). We similarly daisy chain a set of workflow 4's for our experiment involving only workflow 4 with the results shown in Figure 7(b). For our final experiment, we created one composite workflow, which is formed by daisy chaining six sets of workflow 1, followed by six sets of workflow 2, followed by six sets of workflow 3, followed by six sets of workflow 4. We then daisy chain these composite blocks and show the results of this final experiment in Figure 7(c).

Each point of the graph in Figure 7(a), Figure 7(b) and Figure 7(c) is labeled with the number of nodes in it's ancestry chain starting with the given node. This approximates the maximum number of hops needed to create a new digraph from the original provenance graph involving the starting resource. The execution times vary for each query template as well. A why-provenance query retrieves the transitive closure of the edges that justifies the existence of the resource, and so its execution time varies as the number of triples in its transitive closure grows. The structure of this query is given in Example 1. The how-provenance is like the why-provenance except that its execution time accounts for an additional step in computing the polynomials and therefore its execution time differs from the why-provenance execution time by 100-500 milliseconds. This difference is too small to be seen in Figure 7, therefore the how and why provenance query timings look very similar. The structure of this query is given in Example 2. As we increase the complexity of the workflows (from workflow 1 to workflow 4), the query execution time also increases (as shown in Figure 7). The other queries show almost constant execution times, ranging from 1-2 milliseconds. This is not surprising since these queries usually retrieve provenance information in the locality of the resource. For example, the when query just returns the RDF triple whose subject is the resource and whose predicate associates a time value with the resource and the where-provenance query finds the entities whose contents create the resource (i.e where the resource was copied from).

Our experiments were conducted on a IBM computer with 8 X 2.5GHz processors and 32GB RAM. For each pair of query and Jena model, we use the average execution time for the longest diameter in the graph. For a very simple topology (e.g. Figure 6(a)), our prototype is most efficient for both, finding the provenance resources (which involves single resources and their relationships) that an access control system is protecting and for finding the provenance resources a querying user is requesting.

7. RELATED WORK

A lot of research has been devoted to the study of access control in provenance. These include the work in [3], which emphasizes the need for a separate security model for provenance. This work also points out that existing access control models do not support the directed acyclic graph of provenance. The authors in reference [22] discuss the shortcomings of RBAC and instead propose ABAC which supports a fine-grained access control based on attributes rather than roles. In reference [24], the authors present an access control method for provenance over a directed acyclic graph. They build their access control model over a relational database which controls access to nodes and edges. They apply a grouping strategy to the provenance graph to create resources that need to be protected. We want to extend our access control model to support RDF triple stores in addition to relational databases. We support the idea of grouping by defining dynamic paths that are evaluated at query time based on incorporating regular expressions in our policies. In reference [6], the authors propose a grouping of provenance into blocks, and then applying a labeling strategy over these blocks. They also provide a language, SELinks, to encode their security policies. Reference [17] addresses the issues with existing access control models in provenance by proposing a general language. This language supports fine-grained policies and personal preferences and obligations, as well as decision aggregation from different applicable policies. We adapt the language given in [17] with support for regular expressions. Our language also incorporates other features of a general access control language such as support for fine-grained access control over the indivisible parts of a provenance graph, and integration of existing access control policies.

Research has also focused on general access control languages that are based on XML, logic and algebra. XACML [14] is an OASIS standard for an access control language that is based on XML. This language is very flexible and expressive. The work in [17] builds on XACML features to create a general access control language for provenance. Our language extends the XML-based policies in [17] for reasons such as, it is easy to write policies in XML and XML also provides a schema that can be used to verify the policies. Logic-based languages [1] offer features such as decidability and a formal proof of security policies. The work given in [4] shows how policies possibly expressed in different languages can be formulated in algebra. The algebra offers a formal semantics such as in logic-based languages.

8. CONCLUSION

In this paper we propose regular expressions as an extension to traditional access control policy specifications to protect not only traditional data items but also their relationships from unauthorized users. We presented our policy language, its XML-based structure and associated grammar for specifying policies over a provenance graph. We implemented a prototype based on our architecture that uses Semantic Web technologies (RDF, SPARQL) in order to evaluate the effectiveness of our policy language. We are exploring many directions for future research. We discuss some of them. (i) Our current research has focused on in-memory graphs. We plan to expand our experiments to include very large provenance graphs that use disk-based storage. (ii) The policies we have examined so far are those based on access control. We plan to investigate other types of policies including disclosure policies and release policies. This is necessary to further sanitize the resources that are being accessed. (iii) The applications we have been considering are in the areas of healthcare and intelligence. We plan to apply our language to other applications, especially in the area of e-science.

9. REFERENCES

[1] Abadi, M., "Logic in access control" *Proceedings of the 18th Annual Symposium on Logic in Computer Science (LICS03)*, pp. 228–233. Citeseer, 2003.

[2] Alkhateeb, F. and Baget, J.F. and Euzenat, J., "Extending SPARQL with regular expression patterns (for querying RDF)" *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 7, No. 2, pp. 57–73, Elsevier, 2009.

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

[4] Bonatti, P. and De Capitani di Vimercati, S. and Samarati, P., "An algebra for composing access control policies" *ACM Transactions on Information and System Security (TISSEC)*, Vol. 5, No. 1, pp. 1–35, ACM, 2002.

[5] Carroll, J.J. and Dickinson, I. and Dollin, C. and Reynolds, D. and Seaborne, A. and Wilkinson, K., "Jena: implementing the semantic web recommendations" *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pp. 74–83, ACM, 2004.

[6] Corcoran, B.J. and Swamy, N. and Hicks, M., "Combining provenance and security policies in a web-based document management system" *On-line Proceedings of the Workshop on Principles of Provenance (PrOPr)*, Citeseer, 2007.

[7] Detwiler, L.T. and Suciu, D. and Brinkley, J.F., "Regular paths in SparQL: querying the NCI thesaurus" *AMIA Annual Symposium Proceedings*, Vol. 2008, pp. 161, American Medical Informatics Association, 2008.

[8] Ferraiolo, D. and Kuhn, D.R. and Chandramouli, R., "Role-based access control" *Artech House Publishers*, 2003.

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

[10] Hasan, R. and Sion, R. and Winslett, M., "Introducing secure provenance: problems and challenges" *Proceedings of the 2007 ACM workshop on Storage security and survivability*, pp. 18, ACM, 2007.

[11] Holland, D.A. and Braun, U. and Maclean, D. and Muniswamy-Reddy, K.K. and Seltzer, M., "Choosing a data model and query language for provenance" *Int. Provenance and Annotation Workshop*, Citeseer, 2008.

[12] Klyne, G. and Carroll, J.J. and McBride, B., "Resource description framework (RDF): Concepts and abstract syntax" *Changes*, 2004.

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

[14] Lorch, M. and Proctor, S. and Lepro, R. and Kafura, D. and Shah, S., "First experiences using XACML for access control in distributed systems" *Proceedings of the 2003 ACM workshop on XML security*, pp. 25–37, ACM, 2003.

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

[16] Moreau, L., "The Foundations for Provenance on the Web" *Foundations and Trends in Web Science*, Citeseer, 2009.

[17] Ni, Q. and Xu, S. and Bertino, E. and Sandhu, R. and Han, W., "An access control language for a general provenance model" *Secure Data Management*, pp. 68–88, Springer, 2009.

[18] Oinn, T. and Addis, M. and Ferris, J. and Marvin, D. and Greenwood, M. and Carver, T. and Pocock, M.R. and Wipat, A. and Li, P., "Taverna: a tool for the composition and enactment of bioinformatics workflows" *Bioinformatics*, Oxford Univ Press, 2004.

[19] Prud'hommeaux, E. and Seaborne, A. and others, "SPARQL query language for RDF" *W3C working draft*, Vol. 20, 2006.

[20] Robinson, R., "Counting unlabeled acyclic digraphs" *Combinatorial mathematics V*, pp. 28–43, Springer, 1977.

[21] Robinson, R.W., "Counting Labeled Acyclic Digraphs" *Proceedings of the Third Ann Arbor Conference on Graph Theory Held at the University of Michigan*, pp. 239-275, 1971.

[22] Rosenthal, A. and Seligman, L. and Chapman, A. and Blaustein, B., "Scalable access controls for lineage" *First workshop on on Theory and practice of provenance*, pp. 1–10, USENIX Association, 2009.

[23] Samarati, P. and de Vimercati, S., "Access control: Policies, models, and mechanisms" *Foundations of Security Analysis and Design*, pp. 137–196, Springer, 2001.

[24] Syalim, A. and Hori, Y. and Sakurai, K., "Grouping Provenance Information to Improve Efficiency of Access Control" *Advances in Information Security and Assurance*, pp. 51–59, Springer, 2009.

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