SECURED DATA PROVENANCE USING
SEMANTIC WEB TECHNOLOGIES

by

Tyrone Cadenhead

APPROVED BY SUPERVISORY COMMITTEE:

____________________________________________
Dr. Bhavani Thuraisingham, Chair

____________________________________________
Dr. Murat Kantarcioglu, Co-Chair

____________________________________________
Dr. Latifur Khan

____________________________________________
Dr. Weili Wu
Dedicated to my family.
SECURED DATA PROVENANCE USING
SEMANTIC WEB TECHNOLOGIES

by

TYRONE CADENHEAD, B.Sc., M.S., M.A.

DISSERTATION
Presented to the Faculty of
The University of Texas at Dallas
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS
August, 2011
ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Dr. Bhavani Thuraisingham, for attracting me to this area of research, and helping me to set a foundation where I can continue to do future research in a new and challenging topic on the Semantic Web. Her patience and guidance enable me to publish and to reach this milestone in my career. Secondly, I would like to express my profound thanks and appreciation to my co-advisor Dr. Murat Kantarcioglu for his patience, thorough, and invaluable help for the past three years, from the development of the idea for this dissertation to its conclusion. Under his guidance, I have explored many topics and broadened my horizons.

Special thanks also to the other members of my dissertation committee, Dr. Latifur Khan and Dr. Weili Wu for their time to serve on my committee. I would like to thank Dr. Jinlin Chen at Queens College (CUNY) for his guidance, mentorship, recommendations and for being a catalyst for me to undertake this challenge.

I would like to thank my wife Tayana for her continued love, patience, understanding and continued support during my Ph.D. study, and my entire endeavor to become more educated. Through her devotion, she has taught me how to endure and to always see the cup as being half-full. I would also like to thank my mother for being there and for her continued prayers as I struggled through the process. Finally, I thank my sisters for their support and friendship in times of frustration.

August, 2011
Provenance is the lineage of a resource (or data item) and is essential for various domains, including healthcare, intelligence, E-science, legal and industry. The ongoing mutual relationships among entities in these domains rely on sharing quality information, which has created a critical need for provenance in these domains. However, revealing complete provenance raises security concerns, as provenance contains both sensitive and non-sensitive information. The main challenge of protecting provenance lies in its directed graph structure. This structure captures the history of data items and their causal relationships. Furthermore, current security policies such as access control and redaction were mainly developed for protecting single data items or data records; and therefore cannot extend to protecting the lineage of a data item. Therefore, a security model for provenance is missing.

In this thesis, we present a security framework for provenance, which extends the definition of traditional policies to allow specification of policies over a provenance document. The policies are extended in two dimensions. First, we extend the policies to adapt to changing environments with the use of key Semantic Web technologies. Second, we extend
the definition of the policies to include the notion of a provenance path or lineage.

The main contributions of this thesis are as follows:

- Flexible policies, which are independent of our implementation. A domain user may use any policy language that describes the security policies of the domain. A suitable parser will produce a correct low-level policy.

- Definition of an access control policy language for provenance. We can now protect provenance not only as comprising of single data items, but also as paths in a connected directed graph.

- Perform Redaction over a Provenance graph. This is accomplished using the application of a graph grammar technique to circumvent parts of a provenance graph.

- Semantic Web-based inference attacks. A risk-based inference approach is taken to decide suitable tradeoffs for hiding and sharing provenance.
TABLE OF CONTENTS

ACKNOWLEDGMENTS v

ABSTRACT vi

LIST OF TABLES xiii

LIST OF FIGURES xiv

CHAPTER 1 INTRODUCTION 1

1.1 Motivation ......................................................... 2
1.2 Proposed Solution ................................................... 4
    1.2.1 Overview of Provenance Management System Architecture ........ 4
1.3 Contributions ..................................................... 6
1.4 Thesis Outline ..................................................... 7

CHAPTER 2 LITERATURE REVIEW 8

2.1 Scalability and Security of Provenance .......................... 9
2.2 Access Control Languages and Provenance ........................ 11
2.3 Graph Operations and Provenance ................................ 12

CHAPTER 3 BACKGROUND INFORMATION 14

3.1 Access Control ..................................................... 14
3.2 RBAC .............................................................. 15
3.3 RDF .............................................................. 15
3.4 SPARQL .......................................................... 16
3.5 OWL .............................................................. 17
5.3 Policy Language ............................................. 44
  5.3.1 The Grammar ............................................ 47
5.4 Solution Based on Regular Expression Queries .................. 47
5.5 Data Representation ........................................ 49
  5.5.1 Graph Data Model ....................................... 50
  5.5.2 Provenance Vocabulary .................................. 51
  5.5.3 Path Queries ............................................ 53
5.6 Graph Analysis ............................................. 54
  5.6.1 Analysis of Digraphs .................................... 54
  5.6.2 Composition of Digraphs ................................. 55
5.7 Access Control Policy Architecture ............................ 56
  5.7.1 Modules in Access Control Policy Architecture .......... 57
5.8 Use Case: Medical Example .................................. 59
  5.8.1 Query Templates ........................................ 62
  5.8.2 Additional Templates .................................... 63
  5.8.3 Access Control Example .................................. 64
  5.8.4 Prototype ............................................... 65
5.9 Summary ................................................... 68

CHAPTER 6 TRANSFORMING PROVENANCE USING REDACTION 69
6.1 Graph Grammar ............................................. 70
  6.1.1 Graph Data Models ...................................... 71
  6.1.2 Graph Rewriting ......................................... 72
  6.1.3 Graph Operations ........................................ 73
  6.1.4 An Example Graph Transformation Step ................... 78
    6.1.4.1 Valid Provenance Graph ............................ 80
  6.1.5 Discussion ............................................... 83
6.2 Redaction Policy Architecture .................................................. 84
  6.2.1 Modules in Redaction Policy Architecture ............................... 84
  6.2.2 Experiments ........................................................................ 86
6.3 Summary ................................................................................. 91

CHAPTER 7 INFEERENCE .................................................................. 92
  7.1 Model ....................................................................................... 93
    7.1.1 The User’s System ............................................................. 96
    7.1.2 The Internal Knowledge Base System ................................. 96
      7.1.2.1 The Controller ............................................................... 97
    7.1.3 Adding Provenance ........................................................... 98
  7.2 Semantic Framework for Inferences .......................................... 99
    7.2.1 Inference Strategies ............................................................ 100
      7.2.1.1 Inference by Deductive Reasoning ................................. 100
      7.2.1.2 Inference by Inductive Reasoning ................................. 102
      7.2.1.3 Abduction .................................................................... 103
      7.2.1.4 Inference by Analogical Reasoning ............................... 104
      7.2.1.5 Inference by Heuristic Reasoning ................................. 105
      7.2.1.6 Inference by Semantic Association ............................... 105
      7.2.1.7 Inferred Existence ......................................................... 106
    7.2.2 Ontologies ........................................................................... 106
    7.2.3 Rules .................................................................................. 107
    7.2.4 Query Logs ........................................................................ 108

CHAPTER 8 UNIFYING FRAMEWORK ................................................. 110
  8.1 Design of Our Framework ....................................................... 110
  8.2 Inference Controller ............................................................... 114
LIST OF TABLES

4.1 Memory Exception - After Adding 1000 Individuals + 16(Rules) . . . . . . . 26
4.2 \textit{ALCQ} Rule Engine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
5.1 RDF Annotations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
6.1 Query Comparison in Milliseconds . . . . . . . . . . . . . . . . . . . . . . . . . . 87
LIST OF FIGURES

1.1 Unified Provenance Management System ........................................ 5
4.1 A Unified and Flexible RBAC ...................................................... 23
4.2 Distributed Knowledge Bases ....................................................... 26
4.3 Architecture ............................................................................ 28
4.4 A Trace (for a patient Bob). ......................................................... 34
4.5 Scaling Individuals ................................................................. 39
4.6 Scaling TBox and Rules ........................................................... 40
5.1 Policy Language ...................................................................... 45
5.2 Access Control Policy Layer ....................................................... 57
5.3 Provenance Graph ..................................................................... 60
5.4 Why Query ............................................................................... 62
5.5 A resource protected by a policy .................................................. 64
5.6 Why-Provenance Queries .......................................................... 66
5.7 How-Provenance Queries ............................................................ 67
6.1 Edge Contraction .................................................................... 75
6.2 Vertex Contraction .................................................................... 76
6.3 Path Contraction ........................................................................ 76
6.4 Node Relabeling ....................................................................... 77
6.5 Graph Transformation Step ........................................................ 79
6.6 Graph Transformations .............................................................. 81
6.7 Redaction Policy Layer .............................................................. 85

xiv
6.8 Comparison of Redaction Time and Utility vs. Graph Size ............... 89
6.9 Experimental Comparison of Complexity .................................. 90

7.1 Processing a Query ............................................................. 93

8.1 Architecture Design .......................................................... 111
8.2 Interface Module ............................................................... 117
8.3 Policy Modules ................................................................. 118
8.4 Parsing Process ................................................................. 120
8.5 Background Generator ......................................................... 131
8.6 Partial Result page (obtained from yellowpage.com) ....................... 132
8.7 Partial Result page (obtained from whitepage.com) ....................... 133
8.8 Patient Attributes .............................................................. 133
8.9 Search page (obtained from ratemd.com) .................................. 134
8.10 Single Result page (obtained from ratemd.com) .......................... 135
8.11 Multi-Result page (obtained from ratemd.com) .......................... 135
8.12 Physician Attributes ......................................................... 136
8.13 Hospital Result page (obtained from yellowpages.com) ............... 136
A framework for evaluating the utility and security aspects of provenance is a critical need in multiple domains. For instance, in healthcare, the provenance associated with a patient’s medical record can be used for post-diagnosis and for verifying regulatory and healthcare guidelines. Also, the provenance plays a critical part in an emergency room, since it is used to provide step-by-step transparent details of the entire patient’s history. On one side of the coin, the risks of unintentional disclosure of sensitive contents of an electronic patient record (EPR) document can be severe and costly [66]. On the other side of the coin, we need to share provenance data to verify the quality of information exchanged among mutual parties. In e-science, for example, the provenance is used to verify experiments and processes, validate data quality and associate trust values with scientific results [125, 59]. Also, in intelligence, the provenance can be used to determine the quality of information that decisions are based on [35]. Therefore, the more provenance that is associated with the information, the more useful that information is to the strategic decisions of a party.

Different security criteria have been formulated to ensure that documents are not improperly released. These include access control and redaction policies, anonymization and sanitization techniques and cryptographic techniques, etc. In this thesis, we focus on two approaches: access control and redaction policies. On the one hand, access control encompasses a list of techniques for protecting a document. These include policies that specify contextual conditions in the form of rules and those that hide the real data by distortion (e.g., cryptography, scrambling and noise techniques). Therefore, analyzing access control policies and mechanisms are an important topic with respect to the protection of sensitive provenance information. On the other hand, redaction encourages the sharing and continued transactions among organizations and their stakeholders, but the emphasis is on removing the sensitive and proprietary components of a document before releasing it. In
other words, redaction is used to eliminate the threats associated with releasing proprietary or competitive information, trade secrets, financial records, etc.

1.1 Motivation

The primary motivation of this thesis is to provide a unifying framework for securing provenance. We formulate this motivation in order to address different criteria of current information systems:

1. *Sharing Meaningful Information.* Any useful system should provide meaningful information that is relevant, accurate and timely. A user poses queries to gather relevant information for the day-to-day activities. The information may be in same format or different formats, and the activities may vary from domain to domain. For example, in intelligence, the information is for strategic planning; in industry, it is for gaining market share; and in healthcare, it is for providing treatments to patients. These queries could be answered with the use of traditional systems, which are already equipped with their own security mechanisms. However, for these domains, the information retrieved may be both incorrect or outdated.

2. *Verifying the Shared Information.* Each domain should be able to share meaningful information among different domains as well as within a domain. In order to share meaningful information, a provenance information system should be in place to provide metadata describing the accuracy and currency of the domain information that is retrieved and shared. However, domains often collect and store sensitive information. For example: In intelligence, the FBI may collect, classify, and preserve records related to criminal identification and tracking for the official use by authorized officials of the government. In industry, a company may collect customers’ information and transactions in order to target prospective customers effectively, tailor to their individual needs and improve customer satisfaction and retention. In healthcare, a hospital may keep a log of all activities, including patient visits to a hospital; diagnoses and treatments for diseases; and processes performed by healthcare professionals.
3. **Secured Provenance Framework.** There is a need to provide a framework for addressing the above two criteria. This framework will use Semantic Web technologies to integrate the different provenance information retrieved from the different domains, therefore unifying and solving any interoperability among the information formats provided by the systems. In addition, the framework will address the concerns about the security of provenance information for these systems.

We will motivate this thesis with a typical distributed healthcare system for hospitals, but any discussion in this thesis can be applied equally to other domains. The hospitals are divided into several autonomous wards and ancillary units, which are visited by the patients for treatments and examinations during hospitalization in accordance with their illness. The treatment process is dynamic: the new information about a patient results in new treatments. The patient may visit different wards in the same physical hospital or different wards in different hospitals (in case the tests can only be done at specific hospital sites). Also, each hospital is running different healthcare applications. Therefore, the data (a single patient’s healthcare history), the workflow (procedures carried out on that patient), and the logs (a recording of meaningful procedural events) are distributed among several heterogeneous and autonomous information systems.

This healthcare system is suitable for exploring why we need a secure provenance framework. It contains heterogeneous information and applications, which will motivate our choice of a Semantic Web framework. Our reference to information here and throughout this thesis is for both the traditional data and its provenance, unless otherwise noted. This system contains many hospitals and large distributed systems, which will motivate our need to handle the scalability of our framework. Accurate and current information is critical for emergency operations in a hospital, which will motivate our need for a framework with reasonable performances.

The provenance information contains both the entities and the relationships between them. The entities are the agents, processes, algorithms, inputs and outputs to tasks; these entities are similar to the single data items in traditional systems. Our framework should be able to handle the single-data-item view of provenance, as well as the more involved case
of when the relationships between the entities are a part of the provenance. We will address each of these cases in turn for this thesis. Chapter 4 will explore the simple case, where both the traditional data and provenance are treated as single data items. In the case where we consider the relationships among the provenance entities, we will extend the framework in Chapter 4 to handle new policies that are based on graph queries and graph transformation systems in Chapters 5 and 6. We will also consider further extensions to our framework in Chapter 7.

1.2 Proposed Solution

We develop two broad approaches that support provenance in order to protect the sensitive components and still share it. Our first approach is based on a representation of the provenance document. When provenance is both viewed and represented as a directed labeled graph (e.g. RDF, OWL), a rich set of graph-related techniques can be applied to a provenance graph so that access control policies and redaction policies can be used to protect provenance. The second approach is based on a utility measure of provenance. This second approach is a new formal model for determining when we should release provenance. We need to release provenance to enable the advancement of science; the inputs to a new scientific discovery must be verifiable and accurate. The accuracy of information as inputs to a decision making for intelligence must be established before actions are taken based on the available information. This formal model is used to find a comparable tradeoff for releasing both data and its associated provenance and hiding provenance in case of intolerable losses. We propose a special application of this model, which is to build an inference controller that emulates human-like reasoning for the purpose of countering known inference attacks. A discussion to this formal model is presented in Chapter 7.

1.2.1 Overview of Provenance Management System Architecture

Our general architecture consists of a provenance management system, whose role is to ensure that we provide high quality information to a querying user, while ensuring the
confidentiality of the underlying data. This provenance management system is domain neutral; for example, it is designed to protect any domain data which could be in different data formats, describing both the data and its provenance. Therefore, we can build domain independent modules, which are closely integrated to make policy decisions related to the context of input queries and the querying user.

The User Interface Layer hides the actual internal representation of a query and policy from a user. This allows a user to submit both high-level queries and specification of policies, without any knowledge of the underlying logic used to implement the solution. This layer also allows a user to retrieve information, irrespective of the underlying data representation. The Query Processing module translates a user query to one that is compatible with the underlying data representation. The Policy Layer module interacts with the query processing module, and is responsible for ensuring that a query conforms to a set of high-level policies. Any high level policy language can be used to write the security policies, as long as there is a compatible parser that translates these policies to a low-level
representation. The **Redaction** module is used to circumvent any sensitive contents from the result, before it is returned to the user.

It is also important to note that the query processing module, the policy layer module and the redaction module can all be used to enforce the security policies. The query processing module can be used to rewrite a user query according to a set of security policies. The policy layer can be used to provide advanced security features, and is usually the main component for enforcing the policies. The redaction module can be used to remove (circumvent or filter) sensitive information from the results before returning a final answer to the user.

Our solution is based on building a unified and effective policy management framework that enforces the following policies: access control policies, which is the subject of Chapters 4 and 5; redaction policies, which is the subject of Chapter 6; and inference policies, which is the subject of Chapter 7.

### 1.3 Contributions

The main contributions of this thesis are as follows:

- **Flexible policies**, which are independent of our implementation. A domain user may use any policy language that describes the security policies of the domain. A suitable parser will produce a correct low-level policy.

- **Definition of an access control policy language for provenance.** We can now protect provenance not only as comprising of single data items, but also as paths in a connected directed graph.

- **Perform Redaction over a Provenance graph.** This is accomplished using the application of a graph grammar technique to circumvent parts of a provenance graph.

- **Semantic Web-based inference attacks.** A risk-based inference approach is taken to decide suitable tradeoffs for hiding and sharing provenance.
1.4 Thesis Outline

- Section 2 reviews the existing literature.

- Section 3 presents briefly the background information required to follow the theory we use in this thesis.

- Section 4 discusses how we create flexible and scalable access control policies, by extending role-based access control (RBAC) using key Semantic Web technologies. We also implement a prototype, which shows that we can scale and reason over a set of access control policies efficiently.

- Section 5 provides a 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 we can write one policy which is a pattern for several policies, thus contracting the policy set. We also build a prototype using Semantic Web technologies that allows a user to query for data and provenance based on access control policies defined using our policy language.

- Section 6 discusses the application of a graph grammar technique, which can be used 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 7 gives an overview of our inference architecture, which uses a risk-based model to determine whether provenance can be released.

- Section 8 details a unified framework, which integrates the various parts of the system into an automatic framework for provenance. This framework can be used to execute various policies, including access control and redaction policies.

- Section 9 discusses our conclusions and future work.
CHAPTER 2
LITERATURE REVIEW

There are many opportunities and challenges for provenance in the research community [41, 55, 95]. Some opportunities are as follows: (1) information management infrastructure to efficiently and effectively manage the growing volume of provenance information; (2) provenance analytics and visualization for mining and extracting knowledge from provenance data, which has been largely unexplored; and (3) interoperability of different provenance systems and tools to aid in the integration of provenance. Also, recently there have been a series of challenges for provenance [97, 95]. The first provenance challenge aimed to establish an understanding of the capabilities of available provenance-related systems [99]. The second challenge aimed to allow disparate groups to gain a better understanding of the similarities, differences, core concepts and common issues across systems. The third challenge aimed at exchanging provenance information encoded in the Open Provenance Model (OPM) and provide additional profiles. The fourth challenge was to apply OPM to scenarios and demonstrate novel functionality that can only be achieved by the presence of an interoperable solution for provenance. In addition, some of the approaches use Semantic Web technologies to answer the first provenance challenge queries [56].

We think that a very important challenge for provenance is building a scalable and secured framework for provenance. In fact, we found some existing works in the literature relating to the scalability and security of provenance [61, 132, 65, 23]. Furthermore, recording complete provenance would result in extremely large amounts of documentation, which presents problems for recording, querying, management and storage of provenance information. Also, despite the importance of security issues in provenance, it has not been fully explored in the research community [65] especially with respect to access control [102].

The research community has raised many concerns about securing provenance information [22, 18, 23, 102]. In [22], it is suggested that provenance needs its own security model
and in [23], it is suggested that existing access control models do not operate over provenance, which takes the form of a directed graph. It was then suggested that we need a generalized access control language model for protecting provenance in [102]. Also, some approaches recognize that we need a system that handles both the security and provenance together [18, 132, 23, 119]. Other works suggest that we need scalable access control mechanisms for provenance [125, 114, 124].

We think that there are enough opportunities for research in the area of securing provenance to warrant the writing of this thesis. These include providing scalable systems for handling provenance information; exploring the potentials for different languages which support provenance; and representing and storing provenance as a directed graph to support visualization and graph operations over provenance. We explore these opportunities in this thesis. We will continue our survey by focusing on these aspects separately, but the underlying theme is to support them in a unifying secured framework for provenance. For the rest of this section, we will provide other related works that are relevant to the work in this thesis. In Section 2.1, we will present the relevant works related to the scalability of our secured provenance framework. In Section 2.2, we will present the relevant works related to our support for an access control language for provenance. Finally, in Section 2.3, we will provide the relevant related works on graph operations, which form the underlying theory behind the redaction module in our framework.

2.1 Scalability and Security of Provenance

We found related works in the area of access control dealing with reasoning and scalability issues [36, 143, 53, 83, 89, 92]. In [36], an approach for supporting RBAC in a Semantic Web environment using DL is discussed. This work combines RBAC with other context aware access control mechanisms (e.g., ABAC) to add flexibility to RBAC, which suffers from using static assignments. In [36], a high level OWL-DL ontology is used to express RBAC, which is combined with a domain specific ontology about the application domain.

Another approach is that done by Zhao et. al. [143]. They describe a formalism of
RBAC using the DL language $\mathcal{ALCQ}$. In their work, they show how to use DL to model policy constraints. Tim Finin et. al. [53] show different approaches to support RBAC in OWL, by investigating the use of OWL in unifying parallel approaches to policy needs in real world domains. Rules were also used in [53] for enforcing policies written in N3.

There is also work on access control in other policy languages such as XACML (the AOSIS standard for access control), where [83] addresses reasoning aspects of access control and [89] focuses on the scalability aspects.

We found existing works on partitioning knowledge bases and combining query answers over many knowledge bases. In [62], an approach is discussed for partitioning a large OWL ABox with respect to a TBox so that specific kinds of reasoning can be performed separately on each partition and the results combined in order to achieve complete answers. In [128], the authors examine the problem of speeding up and scaling the inferencing process for OWL knowledge bases, that employ rule based reasoners. They propose a data partitioning approach for the problem, in which the input data is partitioned into smaller chunks that are then processed independently. Also, in [47], the focus is on addressing aspects related to answering partial queries and then combining the results. Other works in this area include [107, 126].

We also found ongoing works addressing distributed reasoning in a Semantic web environment. These include [75, 104, 120, 135]. In [75], the focus is on RDF reasoning and Map-reduce, while in [104] the focus is on using a divide-conquer-swap strategy for processing large amounts of RDF data. In [120], a distributed reasoning approach is presented whereby the result is based on the combination local reasoning chunks. This approach uses a tableau-based distributed reasoning procedure. The work in [135] is a comparison of different Semantic Web Languages for Policy Representation and Reasoning in multi-agent and distributed systems.

Levandoski and Mokbel [85] store RDF data into relational databases, and use clustering and partitioning to reduce the number of joins and improve querying time. Database approaches to reasoning, however, require all instances of the data in the entire knowledge base (KB). Furthermore, works along this line could be used for supporting provenance in
a relational database (see Section 5.5).

Our main purpose for doing the above survey is to add scalability to our framework. In particular, we would like to (i) build a scalable Semantic Web framework that is capable of performing inferencing over both traditional data and provenance data that are represented in an OWL knowledge base; (ii) identify existing techniques for performing inferencing over large ABoxes; (iii) identify ways to efficiently answer queries over large distributed provenance knowledge bases; and (iv) explore ideas on how to apply access control mechanisms to a Semantic Web framework.

2.2 Access Control Languages and Provenance

We found recent works related to access control in provenance. These include the work in [23], 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 [114] discuss the shortcomings of RBAC and instead propose ABAC which supports a fine-grained access control based on attributes rather than roles. In reference [131], 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. In reference [37], the authors propose a grouping of provenance into blocks, and then apply a labeling strategy over these blocks. They also provide a language, SELinks, to encode their security policies.

Reference [102] 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.

Research has also focused on general access control languages that are based on XML [24], logic and algebra. XACML [90] is an OASIS standard for an access control language that is based on XML. This language is very flexible and expressive. The work in [102] builds on XACML features to create a general access control language for provenance.
Logic-based languages [2] offer features such as decidability and a formal proof of security policies. The work given in [21] 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.

Our purpose of doing the above survey are many fold: (i) We want to extend our access control model to support RDF triple stores in addition to relational databases. We also want to support the idea of grouping by defining dynamic paths that are evaluated at query time based on incorporating regular expressions in our policies. (ii) We would like to extend the language given in [102] with support for regular expressions. Our language should also incorporate 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. (iii) Finally, we want our language to extend the XML-based policies in [102] for reasons, such as it is easy to write policies in XML [24] and XML also provides a schema that can be used to verify the policies.

2.3 Graph Operations and Provenance

Previous works on using graph transformation approaches to model security aspects of a system include references [38, 80, 81]. In [38], an extension of the double pushout (DPO) rewriting, called Sesqui-pushout (SqPO), 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 [80], the authors used the formal properties of graph transformation to detect and resolve inconsistencies within the specification of access control policies. In [81], the authors proposed a methodology to integrate the specification of access control policies into UML. This provides a graph-based formal semantics for access control, which can allow software engineers to reason about the coherence of access control policies. References [102, 132, 23] focus on some unique features of provenance with respect to the security of provenance itself. For example, it is recognized that we need suitable access control for provenance, which has distinct characteristics from traditional data. While [102] prescribes a generalized access control model for provenance, the flow of information between various sources and the causal
relationships among entities are not immediately obvious in this work. We recognize that we can extend the work in [102] to make obvious these causal relationships, which we have done in [29]. In [132], the authors try to address basic security issues related to a provenance system in the context of a Service Oriented Architecture (SOA) view as well as the security issues of scaling such system. Some of these issues involve the aggregation and combination of processes in a SOA view to achieve the right granularity of access control. Our work is also motivated by [23, 98, 144] where the focus is on representing provenance as a directed graph structure. In [23], it is suggested that provenance needs its own security model, while in [98, 144], an abstract graph model is proposed and a corresponding vocabulary for describing the relationships between nodes in a provenance graph is also presented.

We also found previous works related to the efficiency of a graph rewriting system in [46, 49, 19]. 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 [19].

Our aim in this section is to explore the relevant related works in order to gain better insights into performing graph operations over the directed graph structure of provenance.
CHAPTER 3
BACKGROUND INFORMATION

In this chapter, we will present a brief background on access control and the Semantic Web. A large part of this thesis is devoted to providing a mechanism for extending the traditional definition of access control policies. As a result, Sections 3.1 will cover access control models in general, and Section 3.2 will explore one of these models in detail. In particular, Section 3.2 will discuss role-based access control (RBAC), which is a good starting point for explaining the components of an access control model. The rest of this chapter will give an overview of key Semantic Web technologies used in this thesis.

3.1 Access Control

Access control models include role-based access control (RBAC) [51], mandatory based access control (MAC) [87], discretionary access control (DAC) [94], lattice-based access control (LBAC) [42, 117], temporal access control [14], attribute based access control [36], to name a few. These can be grouped into three main classes [116, 118], 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.

Mandatory access control (MAC) usually enforces access control by attaching security labels to users (subjects) and objects. DAC enforces access control to an object on the basis of permissions and denials configured by the object’s owner. Mandatory policy is particularly applicable to military environments, whereas discretionary policy typically applies to commercial, industrial and educational environments. These two access control models can be simulated by RBAC [106]; therefore we give RBAC an entire section for discussion.
3.2 RBAC

This model [52, 51] generally comprises of loosely coupled components: (i) a user is usually a human or an autonomous agent; (ii) a role is a collection of permissions needed to perform a certain job function; (iii) a permission is an access mode that can be exercised on an object; and (iv) a session relates a user to roles.

- $PA: Roles \rightarrow Permissions$ the permission assignment function, that assigns to roles the permissions needed to complete their jobs;

- $UA: Users \rightarrow Roles$ the user assignment function, that assigns users to roles;

- $user: Sessions \rightarrow Users$, that assigns each session to a single user;

- $role: Sessions \rightarrow 2^{Roles}$, that assigns each session to a set of roles; and

- $RH \subseteq Roles \times Roles$, a partially ordered role hierarchy (written $\geq$).

3.3 RDF

The Resource Description Framework (RDF) [78] is a standard for describing resources on the Semantic Web. It provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource, Identifiers, or URIs), and describing resources in terms of simple properties and property values.

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

**Definition 3.3.1 (RDF Triple)** A RDF triple $(s, p, o)$ is an element of $(U \cup B) \times U \times T$. 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 $s \xrightarrow{p} o$. We also refer to the ordered triple $(s, p, o)$ as the subject, predicate and object of a triple.

RDF has a formal semantics which provides a dependable basis for reasoning about the meaning of a RDF graph. This reasoning is usually called entailment. Entailment rules state which implicit information can be inferred from explicit information. In general, it is not assumed that complete information about any resource is available in a RDF query. A query language should be aware of this and tolerate incomplete or contradicting information.

### 3.4 SPARQL

SPARQL (Simple Protocol And RDF Query Language) [110] is a powerful query language. It is a key Semantic Web technology and was standardized by the RDF Data Access Working Group of the World Wide Web Consortium. SPARQL syntax is similar to SQL, but it has the advantage whereby it enables queries to span multiple disparate data sources that consist of heterogeneous and semi-structured data.

SPARQL is based around graph pattern matching [110].

**Definition 3.4.1** (Graph pattern) a SPARQL graph pattern expression is defined recursively as follows:

1. A triple pattern is a graph pattern.

2. If $P_1$ and $P_2$ are graph patterns, then expressions $(P_1 \ \text{AND} \ P_2)$, $(P_1 \ \text{OPT} \ P_2)$, and $(P_1 \ \text{UNION} \ P_2)$ are graph patterns.

3. If $P$ is a graph pattern, $V$ a set of variables and $X \in U \cup V$ then $(X \ \text{GRAPH} \ P)$ is a graph pattern.

4. If $P$ is a graph pattern and $R$ is a built-in SPARQL condition, then the expression $(P \ \text{FILTER} \ R)$ is a graph pattern.
3.5 OWL

The Web Ontology Language (OWL) [93] is an ontology language that has more expressive power and reasoning capabilities than RDF and RDF Schema (RDF-S). It has additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sub-languages: OWL Lite, OWL DL, and OWL Full. These are designed for use by specific communities of implementers and users. The formal semantics in OWL is based on description logics, which is a decidable fragment of first order logics.

3.6 Description Logics

Description Logics is a family of knowledge representation (KR) formalisms that represent the knowledge of an application domain [9]. It defines the concepts of the domain (i.e., its terminology) as sets of objects called classes, and it uses these concepts to specify properties of objects and individuals occurring in the domain. A description logic is characterized by a set of constructors that allow one to build complex concepts and roles from atomic ones.

3.6.1 $\mathcal{ALCQ}$

A DL language $\mathcal{ALCQ}$ consists of a countable set of individuals $Ind$, a countable set of atomic concepts $CS$, a countable set of roles $RS$ and the concepts built on $CS$ and $RS$ as follows:

$C, D := A \mid \neg A \mid C \cap D \mid C \cup D \mid \exists R.C \mid \forall R.C \mid (\leq nR.C) \mid (\geq nR.C)$,

where $A \in CS, R \in RS, C$ and $D$ are concepts and $n$ is a natural number. Also, individuals are denoted by $a, b, c, \ldots$.

This language includes only concepts in negation normal form. The complement of a concept $\neg (C)$ is inductively defined, as usual, by using the law of double negation, de Morgan laws and the dualities for quantifiers. Moreover, the constants $\top$ and $\bot$ abbreviate $A \cup \neg A$ and $A \cap \neg A$, respectively, for some $A \in CS$.

An interpretation $\mathcal{I}$ consists of a non-empty domain, $\Delta^\mathcal{I}$, and a mapping, $^\mathcal{I}$, that assigns
• to each individual $a \in Ind$ an element $a^I \in \Delta^I$

• to each atomic concept $A \in CS$ a set $A^I \subseteq \Delta^I$

• to each role $R \in RS$ a relation $R^I \subseteq \Delta^I \times \Delta^I$

The interpretation $I$ extends then on concepts as follows:

$\neg A^I = \Delta^I \setminus A^I$

$(C \sqcup D)^I = C^I \cup D^I$

$(C \sqcap D)^I = C^I \cap D^I$

$(\exists R.C)^I = \{x \in \Delta^I \mid \exists y ((x,y) \in R^I \land y \in C^I)\}$

$(\forall R.C)^I = \{x \in \Delta^I \mid \forall y ((x,y) \in R^I \implies y \in C^I)\}$

$(\leq n.R.C)^I = \{x \in \Delta^I \mid \#\{y \mid ((x,y) \in R^I \land y \in C^I)\} \leq n\}$

$(\geq n.R.C)^I = \{x \in \Delta^I \mid \#\{y \mid ((x,y) \in R^I \land y \in C^I)\} \geq n\}$

We can define the notion of a knowledge base and its models. An $\mathcal{ALCQ}$ knowledge base $KB$ is the union of

1. a finite terminological set (TBox) of inclusion axioms that have the form $\top \subseteq C$, where $C$ is called inclusion concept, and

2. a finite assertional set (ABox) of assertions of the form $a:C$ (concept assertion) or $(a,b):R$ (role assertion) where $R$ is called assertional role and $C$ is called assertional concept.

We denote the set of individuals that appear in $KB$ by $Ind(KB)$. An interpretation $I$ is a model of

• an inclusion axiom $\top \subseteq C$ ($I \models \top \subseteq C$) if $C^I = \Delta^I$

• a concept assertion $a : C$ ($I \models a : C$) if $a^I \in C^I$

• a role assertion $a, b : R$ ($I \models (a, b) : R$) if $(a^I, b^I) \in R^I$

Let $K$ be the $\mathcal{ALCQ}$-knowledge base of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. An interpretation $I$ is a model of $K$ if $I \models \phi$, for every $\phi \in \mathcal{T} \cup \mathcal{A}$. A knowledge base $K$ is consistent if it has a
model. Moreover, for $\varphi$ an inclusion axiom or an assertion, we say that $K \models \varphi$ (in words, $K$ entails $\varphi$) if for every model $I$ of $K$, $I \models \varphi$ also holds.

The consistency problem for $\mathcal{ALCQ}$ is ExpTime-complete. The entailment problem is reducible to the consistency problem as follows:

Let $K$ be an $\mathcal{ALCQ}$ knowledge base and $d$ be an individual not belonging to $\text{Ind}(K)$. Then,

- $K \models \top \subseteq C$ iff $K \cup \{d : \neg C\}$ is inconsistent and

- $K \models a : C$ iff $K \cup \{a : \neg C\}$ is inconsistent.

This shows that an entailment can be decided in ExpTime. Moreover, the inconsistency problem is reducible to the entailment problem and so, deciding an entailment is an ExpTime-complete problem too.

### 3.6.2 Inferencing

The basic inference problem for DL is checking a knowledge base consistency. A knowledge base $K$ is consistent if it has a model. The additional inference problems are

- **Concept Satisfiability.** A concept $C$ is satisfiable relative to $K$ if there is a model $\mathcal{I}$ of $K$ such that $C^\mathcal{I} \neq \emptyset$.

- **Concept Subsumption.** A concept $C$ is subsumed by concept $D$ relative to $K$ if, for every model $\mathcal{I}$ of $K$, $C^\mathcal{I} \subseteq D^\mathcal{I}$.

- **Concept Instantiation.** An individual $i$ is an instance of concept $C$ relative to $K$ if, for every model $\mathcal{I}$ of $K$, $a^\mathcal{I} \in C^\mathcal{I}$.

All these reasoning problems can be reduced to KB consistency. For example, concept $C$ is satisfiable w.r.t. the knowledge base $K$ if $K \cup C(a)$ is consistent where $a$ is an individual not occurring in $K$. 
3.7 SWRL

The Semantic Web rule language (SWRL) [70] extends the set of OWL axioms to include horn-like rules, and it extends the Horn-like rules to be combined with an OWL knowledge base.

**Definition 3.7.1 (Horn Clause)** A Horn clause $C$ is an expression of the form $D_0 \leftarrow D_1 \cap \cdots \cap D_n$, where each $D_i$ is an atom. The atom $D_0$ is called the head, and the set $D_1, \ldots, D_n$ is called the body. Variables that occur in the body at most once and do not occur in the head are called unbound variables; all other variables are called bounded.

The proposed rules are of the form of an implication between an antecedent (body) and a consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, the conditions specified in the consequent must also hold. Both the antecedent (body) and consequent (head) consist of zero or more atoms. An empty antecedent is treated as trivially true (i.e., satisfied by every interpretation), so the consequent must also be satisfied by every interpretation. An empty consequent is treated as trivially false (i.e., not satisfied by any interpretation), so the antecedent must not be satisfied by any interpretation.

Multiple atoms are treated as a conjunction, and both the head and body can contain conjunction of such atoms. Note that rules with conjunctive consequents could easily be transformed (via Lloyd-Topor transformations) into multiple rules each with an atomic consequent. Atoms in these rules can be of the form $C(x)$, $P(x, y)$, $\text{SameAs}((x, y))$ or $\text{differentFrom}(x, y)$ where $C$ is an OWL description, $P$ is an OWL property, and $x, y$ are either variables, OWL individuals or OWL data values.
In this chapter, we discuss the drawbacks of using the current access control policies in a heterogeneous environment. These heterogeneous environments can contain either relational data or semistructured data in the form of a tree (e.g., XML) or a graph (e.g., the World Wide Web, RDF). We will motivate this chapter by focusing mainly on role-based access control (RBAC) systems, but the discussion applies equally well to the other access control models. We will first identify the key drawbacks of access control over provenance by concentrating on the simple case: with single data items within a provenance graph. Then we discuss the reasons why we need flexible policies, which are both dynamic and interoperable. We then present a Semantic Web approach for overcoming these challenges.

4.1 Motivation

There is a need for a scalable access control model that simplifies the management of security policies and handles the heterogeneity inherent in an information system with both, traditional data and provenance. We now support this claim with a motivating example taken from the medical domain, where both provenance is recorded [77] and heterogeneity is present [76, 138].

1. Bob’s history shows that he only visits his primary physician Sam to receive healthcare. Therefore, only Sam is preassigned a role to access Bob’s record when Bob is under his care. One day Bob requires emergency care, but Sam is off duty. Kelly is on duty in the emergency room, but Kelly is not pre-assigned to view/update Bob’s record. Therefore, Bob cannot get immediate treatment.

2. Kelly is eventually assigned to a role, which allows access to Bob’s record. However,
Kelly needs to collaborate with other specialists who are in different wards. To expedite the care given to Bob, the information given to Kelly must be coherent and unambiguous.

4.1.1 Contributions

The main contributions of this chapter are:

1. A flexible RBAC using existing semantic technologies.
2. Scalable, support for large instances.
3. Efficiently and accurately reason about access rights.

4.2 Unified and Flexible Policies

One of our overall goals is to provide a general access control model which can support multiple domains. To achieve this, we need an access control policy that can unify existing and disparate access control policies. Role-Based Access Control (RBAC) models have enjoyed popularity by simplifying the management of security policies using roles and also simulating existing access control policies [51]. Therefore, RBAC is a good choice for unifying the different policies in a heterogeneous environment.

In Figure 4.1, we show that we need a stronger notion of simplification of policies than that provided by RBAC. Large organizations are often distributed across different geographical locations. Each department may also have its own jargon and set of terms that are common to that department. We can extend RBAC [51] with ontologies with the use of Semantic Web technologies [13], which will enable RBAC to support these departments. Furthermore, we can integrate different ontologies by importing them into one ontology or we could use a framework for combining them.

Semantic Web technologies has been increasingly used to build rules, which enforce access control [111, 53, 143, 142, 83]. Though a lot of the previous works were about enforcing access control over traditional relational databases, the migration of relational databases to
Figure 4.1. A Unified and Flexible RBAC

the Semantic Web environment is also on the way [15, 40, 34]. Also, with the increasing use of the Semantic Web for e-commerce and web services, we need access control models and mechanisms for this new environment [74, 3, 135, 139, 4, 109]. Therefore, we hope that the work in this chapter plays a part in this shift to a new environment, which supports and enforces access control.

4.3 Flexible RBAC

In the case of Bob in the emergency room, a RBAC model that has a temporal dimension could have given Kelly access to Bob’s record temporarily. These models are well covered in the literature [73, 14]. A temporal access control model (TRBAC) can be summarized
as one that supports periodic role enabling and disabling. TRBAC is generally expressed by means of role triggers, which are active rules that are automatically executed when a specified action occurs. The triggers are basically used to constrain the set of roles that a particular user can activate at a given time instant.

Other models that extend RBAC are also given in [141, 6, 50, 54, 84]; these models are an attempt to overcome the static nature of traditional access control models. They cater to environments where access to resources is required based on some context, like attributes, location, organization structure and coalition.

4.4 Interoperability

In the case of information being shared across different wards (sometimes departments), a framework that allows a global naming scheme and a standard syntax for describing things would resolve the ambiguities. Within a Semantic Web framework, an access control mechanism could respond to a changing vocabulary, and therefore adapt to different environments. We can achieve interoperability by representing the information about RBAC and the domain using the Resource Description Framework (RDF) [78], or the Web Ontology Language (OWL) [93], both knowledge representation languages. RDF is a standard for describing resources on the Semantic Web. It provides a common framework for expressing information so it can be exchanged between applications without loss of meaning. RDF uses Web identifiers (called Uniform Resource, Identifiers, or URIs) for identifying and describing resources in terms of simple properties and property values. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics.

4.5 Supporting Inferences in RBAC

We also support inferencing with respect to RBAC in a Semantic Web environment. We represent RBAC and the domain ontology in OWL ontologies [93]; in particular we are concerned with the description logic (DL) $\mathcal{ALCQ}$ [9] for our knowledge bases. These knowl-
edge bases consist of a TBox and ABox. A TBox is also called the terminology box, which stores the ontology. The TBox organizes a domain by a concept hierarchy and relationships between the concepts. An ABox is the assertional box, which contains instances based on the TBox. Reasoning can be performed by different DL tools, such as Pellet [127] or FaCT++ [136].

The RBAC policies can be specified in terms of rules in a Semantic Web environment. There are different mechanisms for encoding these policies; we consider two of them here, but will choose one of them for addressing some scalability aspects of RBAC. A rule can be encoded using description logics (DL), as shown in [143, 142, 83]. These rules can be used to encode the constraints in the binary relationships of RBAC. Another approach for enforcing the RBAC constraints and policy rules is with the Semantic Web Rule Language (SWRL) [70]. This is more expressive than using DL alone, but with this expressivity there is a complexity tradeoff. SWRL rules are normally written as DL-safe rules in order to be decidable [101]; therefore each variable of a rule is required to occur in a non-DL-atom in the rule body.

Our solution is based on the more expressive rule language, SWRL, which is normally preferred for its ability to support complex relationships between properties, as well as expressing access control with semantic information [123, 86]. The expressiveness of SWRL would allow the formulation of policies like

\[
\text{studentOf}(\text{Mary}, ?x) \land \text{colleagueOf}(?x, ?z) \land \text{isProhibittedReadWith}(?x, \text{note1}) \rightarrow \text{isProhibittedReadWith}(?z, \text{note1}).
\]

This policy states that a student Mary uploaded a note and specifies that if her teachers are prohibited from reading this note, then so too are their colleagues.

We came across one major challenge in using SWRL in our knowledge base, which was due to the availability of memory on my machine. Our observation was that as we increase the number of rules in the knowledge base, the longer it takes to perform reasoning tasks (i.e., determining the consistency of the knowledge base). Table 4.1 shows the performance of when we add all the individual assertions and rules into one knowledge base, which resulted in a memory exception.
Table 4.1. Memory Exception - After Adding 1000 Individuals + 16 Rules

<table>
<thead>
<tr>
<th>Individuals</th>
<th>Inference Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (ms)</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>152</td>
</tr>
<tr>
<td>336</td>
<td>211</td>
</tr>
<tr>
<td>560</td>
<td>276</td>
</tr>
<tr>
<td>784</td>
<td>448</td>
</tr>
<tr>
<td>1008</td>
<td>552</td>
</tr>
</tbody>
</table>

4.5.1 Distributed Reasoning

A solution that we implemented for handling the memory constraint problem was to partition the knowledge base [27]. At first this appeared as a very naive solution: Arbitrary partitioning would not return the same answers to an access control query as that obtained by querying one knowledge base in memory. An access control query is formulated to answer questions like: Does Kelly currently have access to Bob’s record? What is the duration of such access? Is there a conflict between two roles?
After examining the current model for RBAC, we apply the basic tenet of RBAC; this states that a subject (therefore a user or autonomous agent) must assume a role before any access is allowed to a resource. Further, a role is associated with a permission, which is a combination of actions and objects (see Figure 4.2).

Next, we consider the impact of an access control query for Bob’s record in relation to another patient’s record. For example, if Bob is under eighteen years of age, would an access query for Bob’s record depend on another query for Michelle’s record in the case where Michelle is Bob’s guardian? Here we are concerned with a written consent for a minor. Another situation would be where we need to access Michelle’s record in the case where Michelle is related to Bob by blood (therefore, a biological parent).

For the first situation, a policy would have to be in place to ensure that Michelle gives consent to share the contents of Bob’s record with Kelly (or other hospital personnel or third party). This policy would have to be evaluated before another policy could determine that Kelly (or any other hospital personnel) has access to Bob’s record. The second situation is similar; Michelle would give consent to release her information to Kelly, who is in the emergency room. The sequential execution of these policies would indicate that we could create a control environment where one policy is evaluated and a note is made of the decision, then the second policy is evaluated based on the effect of the first policy. A third situation would be when Bob and Michelle are unrelated to each other. In this case, there are no causal dependencies between the execution of policies to consider.

These situations led us to consider a distributed approach to the memory problem. We now describe this approach. We partition the knowledge base (KB) in memory, which we call $KB_{global}$, into $n$ smaller knowledge bases.

$$KB_{global} = (T, A_{global})$$ and we distribute the ABox, $A_{global}$, over each $KB_i$, such that $A_{global} = A_1 \cup A_2 \cup \ldots \cup A_n$. $KB_i = (T, A_i)$.

The TBox remains the same in each knowledge base, but the ABox is distributed. The rational is that, in the general case, the number of instances in a knowledge base grows much faster than the TBox. This is normally observed in relational databases, where the schema is usually fixed, but the instances (or data) increase with the number of daily transactions.
This situation is also observed in DL knowledge bases [62].

4.6 Overview of Our Approach

We now present an overview of our approach, which we use to address the memory constraint problem. We take both the size of the ABox and the number of rules into consideration in order to devise a suitable approach for handling both the scalability issues and efficiency issues we observe in Table 4.1. This approach involves three stages as depicted in Figure 4.3. Note that Figure 4.3 describes an instance related to the medical domain, but Figure 4.3 can be applied to different domains as well. We now define these three stages:

1. We partition the global KB, $KB_{global}$, into $n$ smaller KBs, as shown in Figure 4.3(a). This reduces the size of a partition in memory. We then store each partition onto disk. In order to ensure acceptable query performances, we index each assertion in a partition. This is done by indexing a RDF triple by its subject, predicate and object. This stage is normally performed before we answer the access queries; and often lead to significantly low overhead at query time. There is one important restriction when performing this stage: each partition must be small enough to fit into the memory on the machine at query time.

2. Instead of loading the SWRL rules into a knowledge base with all the instances of
1. Adding these rules to $KB_{inf}$ reduces the impact of performing inferencing at query time. This is due to the fact that we are only retrieving a small subset of the entire triple set into memory. Also, a reasoner will only have to deal with a smaller set of symbols in the ABox, even in the presence of a large rule set.

2. Finally, at query time we only perform an access query for one patient. This is mainly due to the fact that an access decision for $Bob$ is either independent of Michelle’s or we can perform the access queries in sequence (therefore, at different temporal intervals). We retrieve the relevant assertions from disk, one step at a time, and update our inference KB, $KB_{inf}$ (see Figure 4.3(c)). Observe that once this loading is performed, the rules in $KB_{inf}$ could cause new facts to be added to $KB_{inf}$. A major assumption we make is that the cascading effect of firing the rules will eventually halt, indicating that all the access control policy rules are satisfied. Our goal will be to show that a controlled sequence for applying the rules produces acceptable performance times. Another observation is that some rules will function only to add special facts (or metadata about the activities in $KB_{inf}$). Querying for these metadata provides feedback about conflicts, such as a role is both enabled and disabled within the same session.

**Discussion**

When there is an access request for a specific patient, stage two and three are executed in sequence. This sequence ensures that $KB_{inf}$ will only contain the relevant facts for one patient. The main advantages of the architecture in Figure 4.3 can be summarized as follows:

- We can instantiate as many iterations over step two and three. At each iteration $i$ we start with a new knowledge base $KB_{inf(i)}$. This would handle cases where we need to make decisions about patients concurrently.

- There are performance improvements. We now discuss two of them.
1. The partitions, $KB_1, \ldots, KB_n$, are materialized. We perform consistency tests over $KB_i$ before putting it on disk. The effect of doing this is to avoid doing these inferencing tests at querying time. The inference tests are usually expensive [48, 31] and OWL reasoning is exponential in time and memory in the worst case [9].

2. $KB_{inf}$ stores SWRL rules. The effect of this is that the rules are applied to a small number of individuals in the knowledge base.

4.7 Scalable RBAC

A global knowledge base normally resides in memory and offers real-time reasoning and querying, but is not scalable as we stream more data to it. The main reason is that the instances in the ABox grow as we scale our implementation, while the size of the TBox remains relatively the same. We would like to handle a large number of instances, without sacrificing efficiency. To achieve this we partition a knowledge base into a set of smaller knowledge bases. Each smaller knowledge base has the same TBox as the original knowledge base, but a subset of the ABox. What we would like is to be able to do arbitrary partitioning whenever any of the partitions become too large. We can achieve this by using the same argument we use to partition $KB_{global}$. Therefore, $KB_{patient}$ could be further partitioned into $KB_{patient_1}, KB_{patient_2} \ldots KB_{patient_n}$.

4.8 Extending RBAC

Our system consists of the standard RBAC modules, Users, Roles, Sessions, Permissions, and Objects, plus some domain specific modules. The RBAC module could be described in terms of an access control vocabulary or ontology, while the domain modules would reflect the descriptions and vocabulary of the domain. This separation of RBAC from the domain would make our approach more flexible; therefore, we could support any domain by leveraging from the benefits of the Semantic Web.

In this section, we will first define the modules in one domain, but this definition is not limited to the healthcare domain. Next, we will define the binary constraints in our example
domain. These binary relations extend those that are already in place for RBAC. We would also like to point out that our example domain could extend any other access control model or variant of the access control model we use for this chapter.

**Definition 4.8.1 (Domain Modules).** The set \( \mathcal{D} \) consists of disjoint modules, where

- RBAC defines Users, Roles, Sessions, Permissions, and Objects;
- the hospital extends Users to employees; Roles to the organizational structure; Objects to Records (plus other resources like equipments, etc.); and
- the hospital defines Patients (plus other stakeholders like suppliers, etc.).

**Definition 4.8.2 (Mapping Function).** The set \( \mathcal{M} \) consists of unique atomic properties (binary relations) connecting two domain modules in \( \mathcal{D} \) so that we have:

- RBAC assignments: the mappings user-role, role-user, role-permission, permission-role, user-session, role-role and role-session;
- Hospital extensions: the mappings patient-user, user-patient and patient-session; and
- Patient-Record constraint: the one-to-one mappings patient-record and record-patient, where user \( \in \) Users, role \( \in \) Roles, permission \( \in \) Permissions, session \( \in \) Sessions, patient \( \in \) Patients and record \( \in \) Records.

### 4.9 Connecting RBAC Partitions

In order to ensure that we get the desired behavior using our partition approach as that of using one knowledge base, we need to ensure that we can hop from one module to the next. This flow will enable us to gather the pieces we need in order to make an access decision.

In this section, we first present the notion of a home partition. The purpose of defining a home partition is so that we have a unique partition to locate a RDF triple. In the
global approach, all the triples are in $KB_{global}$. However, without a home partition in the distributed approach, we have no predetermined method for retrieving a triple effectively. We take advantage of the fact that a subject in a RDF triple is a uriref, which identify a unique resource. Therefore, we use this unique name to get to a home partition. Next, we define the notion of a link. The idea of this link is to enable us to navigate the partitions. We can think of a link as a way of linking two RDF graphs; therefore a link connects two home partitions, where the subject of triple $t_i$ is in $KB_i$ and its object is the subject of another triple $t_{i+1}$ in $KB_{i+1}$.

**Definition 4.9.1 (Home Partition).** We define a home partition $KB_i$ for all the triples, where $i = 1, 2, \ldots n$, such that

- the TBox, $\mathcal{T}$, is in $KB_i$; and
- for all assertions of form $C(x)$ and $R(x, Y)$, both have the same home, $KB_i$, and $C$ is a concept, $R$ is an atomic property, $x$ is restricted to individuals in $\mathfrak{D}$ and $Y$ is either an individual or a literal (for object or dataType property respectively). In particular, the home is determined by $x$, the domain of $R$.

**Definition 4.9.2 (P-link).** A P-link is a directed arc that allows navigation from one materialized KB to the next. An atomic property $\rho \in RS$, the set of properties in $\mathcal{ALCQ}$, is a P-link if $\rho \in \mathfrak{M}$. Also, a P-link has a home partition.

4.10 A Query-Retrieval Process

We now present an example in which we show how we carry out steps two and three in Figure 4.3. This example will illustrate the controlled sequence of applying the rules in order to ensure good performance. We first present a notion of a policy query, which is encoded with the controlled sequence.

**Policy Query**

Let $KS$ be a set of partitions, a policy (or access) query $q$ against $KS$ is a tuple $(s, \alpha, K, \Psi, o)$,
where $s$ is of the form $[t_1, t_2]$, $\alpha$ is an individual, $K$ is an ordered set of partitions, $\Psi$ is a set of access policy rules and $o$ is the output of a positive query.

- $K$ represents a flow and is of the form $\langle KB_1 \prec \ldots \prec KB_m \rangle$ such that $KB_i \prec KB_{i+1}$ means $KB_i$ precedes $KB_{i+1}$, and the query process starts from $KB_1$ and ends in $KB_m$. Note $i < m$.

- $KB_i$ and $KB_{i+1}$ are connected by a special P-link.

- $\Psi$ is a set of SWRL rules of the form: $H_1 \land \ldots \land H_{m'} \leftarrow B_1 \land \ldots \land B_{n'}$ where $B_i, H_j, 1 \leq i \leq n', 1 \leq j \leq m'$ are atoms of the following form $C(i)$ or $P(i, j)$.

**Example of a Policy Query**

A policy query for a patient Bob in session $[t_1, t_2]$ would be

$([t_1, t_2], Bob, \langle KB_{patient} \prec KB_{user} \prec KB_{role} \rangle, \Psi, o)$.

**Example of a SWRL Rule**

A rule in $\Psi$ would be:

\[
\text{Patient}(?x_1) \land \text{patUser}(?x_1, ?x_2) \land \text{patSess}(?x_1, ?x_4) \land \text{patRec}(?x_1, ?x_3) \land \\
\text{UserRole}(?x_2, ?x_5) \land \text{userSess}(?x_2, ?x_4) \land \text{roleSess}(?x_5, ?x_4) \land \\
\text{rolePerm}(?x_5, ?x_6) \rightarrow \text{canAccess}(?x_2, ?x_3) \land \text{grantPerm}(?x_2, ?x_6),
\]

which means that a user on duty, who plays the appropriate role (e.g. patient’s physician) will be granted access to the patient’s record within the specified session (e.g. the patient’s session).

**Example of a Trace**

A policy query is decomposed into several queries, so that $g_i$ is evaluated over $KB_i$. Figure 4.4 outlines a trace of a policy query for a patient Bob entering the hospital at interval $[t_1, t_2]$. At each stage of the query, we are retrieving a set of results (on the right) for the individual (possibly many) and session on the left. In the diagram, we assume Sam is Bob’s physician.

**Output of the Trace**

The output of a trace is $o = o_1 \cup \ldots \cup o_3$. 
Figure 4.4. A Trace (for a patient Bob).

1. The first output, $o_1$ is determined by issuing $q_1$ against $KB_{patient}$. We add $o_1$ is added to $KB_{inf}$.

2. The second output, $o_1$ is determined by issuing $q_2$ against $KB_{doctor}$. We add $o_2$ is added to $KB_{inf}$.

3. The third output, $o_1$ is determined by issuing $q_3$ against $KB_{role}$. We add $o_3$ is added to $KB_{inf}$.

**Comment**

Queries are also used to retrieve facts from the knowledge base $KB_{inf}$; these include facts inferred by the SWRL rules in $KB_{inf}$. Under the open world assumption (OWA) of OWL and SWRL, a negative response to a query is not treated as failure to prove the query. This is because a knowledge base is assumed to be incomplete; that is, OWL assumes monotonic reasoning. However, under the closed world assumption (CWA), what is not known to be true is believed to be false. This is the common reasoning over a relational database, which uses default reasoning.
Under OWA the Semantic Web approach does not quite give the output we desire; instead, we would like the output returned under the closed world assumption. Many access control mechanisms make use of policy resolution techniques that handle situations where the effect of a policy is neither true (i.e., a permit) nor false (i.e., a deny). In these situations a default effect is assumed, which could be a denial-takes-precedence effect. Therefore, in the partitioned approach we presented for RBAC, a policy query is positive if \( o_i \neq \emptyset \) \( \forall i \) and false otherwise. This ensures that a user does not have the ability to perform unnecessary and potentially harmful actions merely as a side effect of granting access using incomplete knowledge from a set of KBs.

4.11 Optimization and Heuristics

In the case where everything fits into memory, inferencing is fast, but not scalable. In our distributed approach, we address the scalability aspects by using a combination of disk and memory. The distributed approach incurs overhead in creating the partitions and combining the relevant facts to answer an access query.

We identify two places where we need to apply indexing. The first place is in a partition on disk and the second place is in a table that stores the home partitions.

1. We index each triple \( t = (s \ p \ o) \) in order to have faster retrieval at query time. We employ the services of LARQ*, which is an extension of the SPARQL implementation of Jena using Apache Lucene. Using this indexing API allows us to find a triple by its subject (\( s \)), its predicate (\( p \)) or its object (\( o \)), without the cost of a linear search over all the triples in a partition.

2. We keep a lookup table which is an index of the location of a partition on disk. This time we employ the services of Lucene [57]. When we are ready to process an access request, we use this index to locate the home partition for a RDF triple. Using this lookup table, we are able to locate a triple in at most linear time with respect to the number of partitions.

*http://jena.sourceforge.net/ARQ/lucene-arq.html
We apply two heuristics in order to have greater performances.

1. A concept $C$ could be of the form $\geq nR.C$ or $\leq nR.C$. These concepts are used in RBAC to enforce minimum and maximum cardinality on roles. We limit $n$ for concepts $C \subseteq \geq nR.C$ and $C \subseteq \leq nR.C$.

2. For $n_u$ users and $n_r$ roles, the combination of users and roles is at most $n_u \times n_r$. A user, $a$, could theoretically have $n_r$ assertions of form $R(a, r_i), i = 1 \ldots n_r$. In these situations, arbitrary partitioning may seem too naive. In practice, however, this is less likely to be the case, since there are different competences among users and roles are based on the user’s ability to perform a job.

4.12 Correctness

For a $\mathcal{ALCQ}$ knowledge base $KB$, complex concepts are built from atomic concepts in $CS$, and properties in $RS$. Table 4.2 displays the predicate logic translation of these concepts [9, sec. 2.2].

Let $F$ infer assertions from $KB_i$, such that $F(T, A_i) = Inf_i$, $Q$ be a query over a set of triples and $S$ be a subset of $\mathcal{ALCQ}$ KBs. For RBAC system discussed in this chapter,
our partitioning based reasoning scheme correctly infers all the necessary triples needed for enforcing security policy.

**Theorem 4.12.1** \( Q(F(KB_{global})) \equiv F(\bigcup_{i \in S} Q_{S_i}(F(KB_i))) \)

**Proof.** Our goal is to prove that our partitioning scheme correctly infers all the relevant triples associated with a given session, user, role and permission. Basically, we argue that in order to correctly infer all the triples associated with the \( KB_{global} \), we can just do reasoning using each partition and combine the selected results of the local reasoners.†

To prove this claim, we will use the rule engine given in Table 4.2. First, we argue that the information needed to use the rules given in Table 4.2 is already captured by the TBox and the local ABox instances. To prove this we will examine all the rules given above and argue that correct application of those rules could be done without combining instances in different partitions.

Please note that the first rule in group 1 could be correctly applied by just using the TBox. The second and third rules in group 1 could be correctly applied by using the triple \((x R y)\) given in some local partition and TBox since the definition of \( R \) in TBox precisely specifies the domain and the range of the relation. The fourth rule in group 1 is just the negation of an atomic concept.

For rules in group 2, for correct reasoning, we need to find out triples of the form \((x R y_i), \forall i\). Since our partitioning puts all the triples with subject \( x \) to the same partition, all needed triples for correct inference will be in the same partition.

For the rules in group 3, we need to have all concepts \( D_i \) to be present at the time of reasoning. Clearly some of the concepts could be in different partitions. The way our system works is we query all the materialized results for each partition (i.e., \( F(KB_i) \)) related to RBAC query and get the \( D_i \) concepts needed. Using these \( D_i \) concepts and a TBox, we infer all \( C(x) \) and associated triples in memory (i.e., in \( KB_{inf} \)).

In our partition approach, we evaluate each policy query \((s, \alpha, K, \Psi, o)\), against the ordered set \( K \) of materialized KBs.

†Please note that, all the local reasoners share the same TBox
Therefore, \( Q(K) = Q((KB_1 \prec \ldots \prec KB_m)) \equiv Q((KB_{\text{global}})) \).

### 4.13 Experimental Evaluation

Our experiments were conducted on a Dell Inspiron 2.4GHz with 8GB RAM. We use various open sources to build a prototype. For keeping track of the partitions, we use a Lucene index [57]. To query a partition, we use LARQ and SPARQL [110]. We use Java 1.5 as the main programming language for writing the logic in our code and pellet [127] as the main reasoner. We use synthetic data to build in-memory models, using the Jena API\(^3\) [32]. The individuals and their properties in our knowledge bases were created randomly. We use Protégé [79] to build our TBox, and Jena to build our ABoxes and programmatically extend the TBox. Each user and patient have on average 30 object and data type properties. We use various information sources, such as WebMD\(^5\), pubmed\(^6\), and related literature to investigate the healthcare domain, in order to gain insight into the healthcare domain.

For our base line, we compare our performances to those in [83], which evaluates reasoning for a similar policy language, XACML. We use a naive approach to generate assertions for an in-memory KB (see Table 4.1). We perform various runs, each time with different combinations of individuals from our toy hospital domain. The time to build a set of materialized KBs ranges from one to three hours. We scale one variable in the set \{(D)octors, (N)urses, (P)atients (R)ules, TBox\} at a time, while keeping the other variables constant. We record the time for performing inferencing over \( KB_{\text{inf}} \). The time to retrieve the assertions and validate the policies in \( KB_{\text{inf}} \) are presented in Figure 4.5 and Figure 4.6. The performance of our inference knowledge base, \( KB_{\text{inf}} \), displays fluctuations as we simulate various activities. This is due to the fact that the index of any of the most accessed KBs will cache previous results, and this has unpredictable behavior. The results in Figure 4.5(a) and 4.5(b) show that we can achieve almost constant time for determining a policy decision even in the presence of large instances. The results in Figure 4.6(a) show that we can

---

\(^3\)http://jena.sourceforge.net/
\(^5\)http://www.webmd.com/
\(^6\)pubmed.gov
Figure 4.5. Scaling Individuals

(a) Scaling Doctors using constants: 4200(N), 2100(P), 16(R)

(b) Scaling Patients using constants: 3862(D), 3940(N), 16(R)
(a) Scaling Rules using constants: 320(D), 640(N), 1280(P)

(b) Scaling TBox using constants: 320(D), 640(N), 1280(P), 16(R)

Figure 4.6. Scaling TBox and Rules
support large number rules. The constant run-time in Figure 4.6(a) is due to our approach of only supporting one instance of a patient per access query. The results in Figure 4.6(b) show that scaling our TBox does have some performance limitations. This could be because the DL reasoner must perform more expensive tests each time the TBox size increases.

4.13.1 Discussion

Our implementation is quite scalable with respect to the ABox size and number of SWRL rules. However, it does not scale as well with the TBox. We expect the TBox size to be fairly constant in practice. Our implementation performs fairly well in comparison to our chosen baseline.

4.14 Summary

In this chapter, we tell part of the story where traditional access control does not extend over RDF graph data. We proposed our approach to handle policies on the Semantic Web. Most of the work in this chapter is published in [27]. In the rest of this thesis, we will continue to address concerns about traditional policies. In particular, we will turn our attention to the case where provenance takes the form of a directed graph. Moreover, we will continue to use RDF graphs to represent and store provenance, since a RDF graph data model can be restricted to capture the unique features of provenance, such as a directed acyclic structure and causality among entities.
CHAPTER 5
A LANGUAGE FOR PROVENANCE ACCESS CONTROL

In the previous chapter, we discuss the design of a scalable and efficient access control model for securing both data and its provenance. We assume that both the traditional data and provenance were represented in a RDF format, and therefore we could take advantage of an integrated Semantic web environment to represent and reason over RBAC policies. Besides the scalability and efficiency of an access control mechanism for managing a provenance system, there are other concerns we need to address before we arrive at a unified framework for provenance.

Traditional access control models focus on individual data items, whereas in provenance we are concerned with protecting both, the data items and their relationships [23]. 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).

Our main contribution in this chapter is the definition of an access control policy language for provenance. With this language, we can treat provenance not only as comprising of single data items, but also as paths in a connected directed graph. This language also 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 policies defined using our policy language.
5.1 Challenges

The major challenges we face in implementing an access control policy for provenance are related to the definition of a provenance resource. This identification is one of the major distinguishing factors between a provenance access control model and existing access control models. 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 [98]. The provenance graph structure not only poses challenges to access control models but also to querying languages [68].

5.2 Drawbacks of Current Access Control Mechanisms

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 [116], 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) discretionary access control (DAC), which controls access based on the identity of the user; and (3) mandatory access control (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 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 [51], 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 [114].

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 5.6, 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 chapter and provide an implementation of a prototype mechanism, which shows that we can greatly reduce these mappings.

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 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 a fine-grained access control on any component of the graph; and (iii) to seamlessly integrate existing organizational policies.

### 5.3 Policy Language

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 provide an adaptation of the access control language for provenance given in [102].
We extend the syntax of this XML policy language in order to incorporate regular expressions in the policy. This existing provenance language in [102] 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, it now suffers from the fact that a resource must be identified beforehand, 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 [102] is given in Figure 5.1, 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 [102], but make slight modifications to their meanings to incorporate regular expressions on a provenance graph. Since our focus in this chapter is on specifying a policy for access control in provenance, we provide only the relevant XML elements in this chapter. The interested reader can find other interesting elements of the language, such as obligation and originator preference, in [102].

The description of each element in Figure 5.1 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 the 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 [102] 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.
5.3.1 The Grammar

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

\[
\begin{align*}
\text{<exp>} & ::= \text{<char>}+ (\text{.} \text{<char>})? \\
\text{<char>} & ::= [a-z] | [A-Z] | \_ | - \\
\text{<reg>} & ::= \* | \+ | ? \\
\text{<bool>} & ::= \text{AND} | \text{OR} | \text{||} \\
\text{<op>} & ::= == | <= | >= | < | > \\
\text{<num>} & ::= ([0-9])+ \\
\text{<sp>} & ::= [\[ \text{<exp>} \]]
\end{align*}
\]

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

\[
\begin{align*}
\text{subject} & = \text{<char>}+ | \text{<num>} \\
\text{record} & = \text{<exp>} \\
\text{restriction} & = (\text{<exp><num>})+ (\text{<op>} | \text{<sp><reg>}) \\
\text{scope} & = \text{<char>}+ \\
\text{condition} & = (\text{<exp><num>})+ (\text{<op>} | \text{<sp><reg>}) \\
\text{effect} & = \text{<char>}+ | \text{<num>}
\end{align*}
\]

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.

5.4 Solution Based on Regular Expression Queries

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.

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 patient’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.

In contrast to the previous 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. Consider a military example 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.
5.5 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) [98] 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 [78]. Each of these in their current form has drawbacks with respect to provenance [68]. 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 [64] 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.

5.5.1 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. This data model has been successfully applied for provenance capture and representation [145, 45].

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.

We can view each RDF triple $(s, p, o)$ 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 5.5.1 (Provenance Graph)** 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 $s$.

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.
- $\epsilon$ 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 $t \notin H_2$.

5.5.2 Provenance Vocabulary

We define the nodes in the provenance graph using the nomenclature in [98]. 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 [98]. A process is an action or series of actions performed on or caused by artifacts and resulting in new artifacts [98]. An agent is a contextual entity acting as a catalyst of a process, enabling, facilitating, controlling and affecting its execution [98]. 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 [98].
Let $\mathcal{V}_G$ be the set of names appearing in a provenance graph $G$ and $\mathcal{V}^P_G \subseteq \mathcal{V}_G$ be a set of names on the arcs in $G$. The label on each $e \in \mathcal{V}^P_G$ 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 $\mathcal{V}^P_G$ describing the causal relationships among the nodes in $G$ are as follows:

$$\text{<opm:Process> <opm:WasControlledBy> <opm:Agent>}$$
$$\text{<opm:Process> <opm:Used> <opm:Artifact>}$$
$$\text{<opm:Artifact> <opm:WasDerivedFrom> <opm:Artifact>}$$
$$\text{<opm:Artifact> <opm:WasGeneratedBy> <opm:Process>}$$
$$\text{<opm:Process> <opm:WasTriggeredBy> <opm:Process>}$$

These predicates are the ones defined in [98] and they form the edges in our edge set, $E_G$, in our provenance graph $G$.

Definition 5.5.2 (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 5.5.3 (Provenance Path) In $G$, a provenance path $(s \rho o)$ is a path $s(\xrightarrow{\rho})o$ that is defined over the provenance vocabulary $\mathcal{V}^P_G$ using regular expressions.

Definition 5.5.4 (Regular Expressions) Let $\Sigma$ be an alphabet of terms in $\mathcal{U} \cap \mathcal{V}^P_G$, 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:
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 $p$, $x(p \rightarrow)^* y$ and $x(p \rightarrow)^+ y$ are paths in $G$.

### 5.5.3 Path Queries

SPARQL is based around graph pattern matching [110].

**Definition 5.5.5 (Graph pattern)** A SPARQL graph pattern expression is defined recursively as follows:

1. A triple pattern is a graph pattern.

2. If $P_1$ and $P_2$ are graph patterns, then expressions $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ UNION } P_2)$ are graph patterns.

3. If $P$ is a graph pattern and $R$ is a built-in SPARQL condition, then the expression $(P \text{ FILTER } R)$ is a graph pattern.

4. If $P$ is a graph pattern, $V$ is a set of variables and $X \in U \cup V$ then $(X \text{ GRAPH } P)$ is a graph pattern.

The current W3C recommendation for SPARQL does not support paths of arbitrary length [44]; 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 [44, 5, 82]. A W3C working draft for extending SPARQL to support property paths can be found in [64].

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.

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

where $P$ is a graph pattern and $\vec{B}$ is a tuple of variables appearing in $P$. 
5.6 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.

5.6.1 Analysis of Digraphs

We now provide a simple analysis addressing the concerns from Section 5.2 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 in \([113, 112]\) is given recursively as

\[
a_n = \sum_{k=1}^{n} (-1)^{k-1} \binom{n}{k} 2^{k(n-k)} a_{n-k}. \tag{5.1}
\]

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 \([113]\) 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 [98] 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 5.5 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.

### 5.6.2 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 [116] 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. $G_1 \subseteq G_2$: The digraph $G_1$ is associated with a policy that denies viewing its contents 
and the digraph $G_2$ 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 
$G_2 - G_1$.

2. $G_1 \supseteq G_2$: The digraph $G_1$ is associated with a policy that denies viewing its contents 
and the digraph $G_2$ 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, $G_1$ and $G_2$.

3. $G_1 \cap G_2$: The digraph $G_1$ is associated with a policy that denies viewing its contents 
and the digraph $G_2$ is associated with a policy that permits viewing of its contents. In 
this situation, the system would have the effect of denying access to digraphs $G_1$ and 
$G_1 \cap G_2$.

These three cases also apply when a user’s query execution returns the digraph, $G_2$, and 
the effect of the policy for $G_1$ is “deny”.

5.7 Access Control Policy 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
Figure 5.2. Access Control Policy Layer

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. 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.

5.7.1 Modules in Access Control Policy Architecture

We now present a detailed description of the different layers in Figure 5.2 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 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 5.4(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 5.1.

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 5.3.1.

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 5.4(a) shows a user query and the corresponding translation in Figure 5.4(b).

Data Controller
The Data Controller is a suite of software programs that stores and manages 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.

5.8 Use Case: Medical Example

In this section we provide examples of provenance queries. These queries can be used to identify resources for a policy or identify the answer for a user query. The examples in this section are based on the provenance graph in Figure 5.3.

The provenance graph in Figure 5.3 shows a workflow which updates a fictitious record
Figure 5.3. Provenance Graph

for a patient who went through 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 operation, 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.

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 5.3 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>

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 5.3 as a running example through the rest of this thesis.

Table 5.1. RDF Annotations

<table>
<thead>
<tr>
<th>Entity</th>
<th>RDF Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>PerformedOn</td>
</tr>
<tr>
<td>Agent</td>
<td>Name, Sex, Age and Zip Code</td>
</tr>
<tr>
<td>Artifact</td>
<td>UpdatedOn</td>
</tr>
</tbody>
</table>

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 5.1 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,
5.8.1 Query Templates

![Diagram](image)

Figure 5.4. Why Query

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 [95]. 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 5.8.1 (Why Query)

```prolog
med:Doc1_3 gleen:OnPath("([opm:WasDerivedFrom] | [opm:WasGeneratedBy] | [opm:WasTriggeredBy] | [opm:WasControlledBy] | [opm:Used])") ?x.
```

This allows us to specify all the resources reachable from med:Doc1_3 by issuing a query against the provenance graph. This query explains why med:Doc1_3 came to be in its
current form. Figure 5.4 shows a part of the graph in Figure 5.3 that would result from executing a why-provenance query.

**Example 5.8.2 (Where Query)**

```prolog
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).

### 5.8.2 Additional Templates

The Open Provenance Model [98] in general allows us to extend $V_G$ 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 5.8.3 (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.

### 5.8.3 Access Control Example

![Diagram of access control example](image)

Figure 5.5. A resource protected by a policy

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 5.4(a) is submitted to the User Interface 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 5.4(b). The result of executing this query against the provenance graph shown in Figure 5.3 returns the results shown in Figure 5.4(c). This result is passed back to the Access Control Policy Layer. This layer also passes the policy given in Figure 5.1 to the Policy Parser Layer that parses the policy against a XML schema and the grammar given in Section 5.3.1. If the policy is well constructed, it is passed to the Regular Expression-Query Translator Layer that constructs the query given in Figure 5.5(a). This query is also evaluated against the provenance graph in Figure 5.3. The result of this query execution would return the digraph shown in Figure 5.5(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 5.4(c) with the digraph in Figure 5.5(b). Since the digraph in Figure 5.4(c) contains the digraph in Figure 5.5(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 5.4(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.

5.8.4 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 programmed to apply the grammar in Section 5.3.1 against the values in the elements in the policies stored in the policy file. To implement the
Regular Expression-Query Translator, we use the Glen\(^*\) regular expression library. This library extends SPARQL to support querying over a RDF graph using regular expressions [44]. To create the provenance store, we use the OPM toolbox\(^\dagger\). This toolbox allows us to programatically 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 [103], 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 [144]. Our aim in this chapter 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

![Why-Provenance](image)

**Figure 5.6. Why-Provenance Queries**

Jena\(^\ddagger\) [32]. This tool allows us to add annotations 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 graph in the Jena model. The graphs vary in size as shown in Figure 5.6 and Figure 5.7. The execution times vary


\(^\dagger\)Available at [http://openprovenance.org/](http://openprovenance.org/)

\(^\ddagger\)[http://jena.sourceforge.net/](http://jena.sourceforge.net/)
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 main impact on the how-provenance execution time is the length of each ancestry chain starting from the resource. 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 times a provenance entity occurs at the beginning of an ancestry-chain for the resource. 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 Dell Inspiron 2.4GHz with 8GB RAM. For each pair of query and Jena model, we use the average execution time for the longest diameter in the graph. Each point of the graphs in Figure 5.6 and Figure 5.7 is labeled with the longest diameter, which is the path with the most edges starting from the resource. This approximates the maximum number of hops needed to create a new digraph from the original.

Figure 5.7. How-Provenance Queries
provenance graph involving the starting resource. Our prototype is 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.

5.9 Summary

In this chapter, we tell another part of the story, where traditional access control policies do not extend over provenance data. We proposed a language that can be used to express access control over provenance, which takes the form of a directed graph. In the rest of this thesis, we will continue to address concerns about security and provenance. Next, we will focus on another kind of policy, which governs the release of provenance. We refer to this release policy as redaction, which is used to specify the sensitive information that must be removed before the provenance is shared.
CHAPTER 6
TRANSFORMING PROVENANCE USING REDACTION

So far we have mainly focused on access control policies, which are mainly used to control access to a document. In this chapter we explore other policies (namely redaction) that enable the sharing of provenance.

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. An access control 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 [102]. This language was further extended to show how to effectively apply access control over provenance graphs by extending SPARQL queries with regular expressions in Chapter 5. However, these policies do not specify any formal graph models for applying policies over a provenance graph nor focus on the need for sharing provenance. We now address these shortcomings by writing the redaction policies as graph operations over a provenance graph [30], which has a formal foundation based on a graph grammar [115].

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*. 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 [66]. Such risks may include litigation proceedings related to non-compliance of HIPAA regulations [66].

Commercially available redaction tools have been so far applied over single resources

*http://www.redact-it.com/*
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 chapter, we apply a graph transformation technique (generally called graph grammar [115]) 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 [78], 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.

Our main contribution in this chapter 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.

### 6.1 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.

6.1.1 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 [78] and the OPM provenance model [98]. In addition, directed graphs
are a natural representation of provenance [23, 98, 144]. 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 6.1.1 (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, \(\ell = (\ell_V, \ell_E)\) is a set of labels, \(\mu : V \rightarrow \ell_V\)
is a function assigning labels to nodes, and \(\nu : E \rightarrow \ell_E\) is a function assigning labels to
edges. In addition, the sets \(\ell_V\) and \(\ell_E\) are disjoint.

Definition 6.1.2 (RDF Graph) Recall that a RDF graph is a finite collection of RDF
triples. A RDF graph used in this chapter restricts Definition 6.1.1 as follows:

1. \(\ell_V \subset (U \cup B \cup L)\)

2. \(\ell_E \subset U\)

3. A RDF triple \((s, p, o)\) is a directed labeled edge \(p\) in \(G_\ell\) with endpoints \(s\) and \(o\).

Definition 6.1.3 (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.

6.1.2 Graph Rewriting

A graph rewriting system is well suited for performing transformations over a graph. Furthermore, 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 [115, 39].

For the purpose of this thesis we define graph rewriting as follows,

**Definition 6.1.4 (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 6.1.1;
- $q$ 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 [110]. In addition, we use one of the features in the latest extension of SPARQL [64], namely regular expressions, to identify paths of arbitrary length in a provenance graph. A description of SPARQL can also be found in Section 5.5.3.

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.

6.1.3 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
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 $V \setminus \{u, v\}$ to itself, and otherwise maps it to a new vertex $w$. The contraction of $e$ results in a new graph $G' = (V', E')$, where $V' = (V \setminus \{u, v\}) \cup \{w\}$, $E' = (E \setminus \{e\})$, 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 6.1.3, we delete these edges.

Figure 6.1 is an example of an edge contraction for our use case (see Figure 5.3). 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 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 6.1.3. A vertex contraction of the left hand side of Figure 6.2 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’s 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 5.3 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. Figure 6.3 shows an example of a path contraction.

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 5.3.
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 5.3 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 [130]) Figure 6.4 shows an example of node relabeling.

Figure 6.4. Node Relabeling
We motivate this idea of node relabeling with the following RDF triples taken from our use case.

\[
\text{<med:Physician_1_1> <med:Sex> "M"}
\]
\[
\text{<med:Physician_1_1> <med:Age> "35"}
\]
\[
\text{<med:Physician_1_1> <med:Zip> "76543"}
\]

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

\[
\text{<med:Physician_1_1> <med:Sex> "X"}
\]
\[
\text{<med:Physician_1_1> <med:Age> "30-40"}
\]
\[
\text{<med:Physician_1_1> <med:Zip> "765XX"}
\]

6.1.4 An Example Graph Transformation Step

We show the general steps of the medical procedure only for one patient in Figure 5.3 for clarity. However, in reality Figure 5.3 would be a subgraph of a much larger graph that describes provenance for n patients. We now motivate the transformation step over Figure 5.3 with an example.

**Example 6.1.1** 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. From Figure 5.3, 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.

$$
\{ \text{med:Doc1}_4 \text{gleen:OnPath} \left( \left( \text{opm:WasDerivedFrom} + \text{opm:WasGeneratedBy}/\text{opm:WasControlledBy} \right) \right) \\
\text{UNION} \{ \text{med:HeartSurgery1}_1 \text{gleen:OnPath} \left( \left( \text{opm:Used} \right| \text{opm:WasControlledBy} \right) \ast \right) \}
\}
$$

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:WasControlledBy> <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> 
```
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 govern the particulars relating to how information is shared based on the hospital procedures or an even bigger set of regulatory guidelines (e.g., HIPAA). Figure 6.5(a) is the first production rule applied to $G_q$ and Figure 6.5(b) and Figure 6.5(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 6.5(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:Doc1_4> <opm:WasDerivedFrom> \_A1
<med:Doc1_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
```

### 6.1.4.1 Valid Provenance Graph

A graph rewriting system should be capable of specifying under what conditions a graph manipulation operation is valid. The embedding instructions normally contain a fair amount
Figure 6.6. Graph Transformations
of information and are usually very flexible. Therefore, allowing the policy designer to specify the embeddings may become error-proned. The OPM nomenclature places restrictions on the set of admissible RDF graphs, which we call valid OPM graphs. These restrictions serve to control a graph transformation process (also a graph rewriting process) by ruling out transformations leading to non-admissible graphs.

Let there be a rule in Figure 6.6(a) that replaces a subgraph of phase 1 in Figure 5.3 with a null (or empty) graph. Figures 6.6(b)-(d) show the effects of carrying out a graph transformation step using an embedding instruction. Figures 6.6(b) is the result of performing a transformation using the rule in Figure 6.6(a) and the following embedding instruction:

\[
<\text{med:Doc}_n_3> \ <\text{opm:wasDerivedFrom}> \ <\text{med:Doc}_n_1>  
<\text{med:RepeatVisit}_n_1> \ <\text{opm:Used}> \ <\text{med:Doc}_n_1>
\]

Figures 6.6(c) is the result of performing a transformation using the rule in Figure 6.6(a) but with an empty embedding instruction.

Figure 6.6(d) is the result of performing a transformation using the rule in Figure 6.6(a) and the following embedding instruction:

\[
<\text{med:Doc}_n_3> \ <\text{opm:wasDerivedFrom}> \ <\text{med:Doc}_n_1>  
<\text{med:RepeatVisit}_n_1> \ <\text{opm:wasControlledBy}> \ <\text{med:Checkup}_n_1>
\]

The only provenance graph of interest to us is the one in Figure 6.6(b). This is a valid OPM graph under the transformation of the rule in Figure 6.6(a). Figure 6.6(b) conforms to the OPM nomenclature convention, and each causal dependency in Figure 6.6(b) existed in Figure 5.3. Figure 6.6(c) is a valid OPM graph, but the causal relationships are not preserved, for example there is a causal relationship between med:Doc_n_3 and med:Doc_n_1 in Figure 5.3, which is absent in Figure 6.6(c). Figure 6.6(d) is not a valid OPM graph since the RDF triple

\[
<\text{med:RepeatVisit}_n_1> \ <\text{opm:wasControlledBy}> \ <\text{med:Checkup}_n_1>
\]

does not conform to the OPM nomenclature convention. In addition, there is no causal relationship between med:RepeatVisit_n_1 and med:Checkup_n_1 in Figure 5.3.
6.1.5 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 5.3, the contextual information associated with that physician (such as age, zip code and sex) could reidentify the physician. Another drawback in releasing informa-
tion 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 5.3, 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.

6.2 Redaction Policy Architecture

Figure 6.7 gives an overview of our redaction policy architecture.

6.2.1 Modules in Redaction Policy 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 [24], 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 6.5(a) using our default policy language for patient 1.

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

The \textbf{Policy Parser Layer}, \textbf{Redaction Policy Layer}, \textbf{Regular Expression-Order Query Translator}, \textbf{Data Controller} and the \textbf{Provenance Controller} all have the same default behavior like the modules in Section 5.7.1, except that the \textbf{Access Control Policy Layer} is replaced with the \textbf{Redaction Policy Layer}. The \textbf{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.

There is a similarity between the access control policy architecture in Figure 5.2 and the redaction policy architecture in Figure 6.7. This redundancy allows us to use the redaction policy layer without installing the access control policy layer. Sometimes we may want to hide information using only access control policies and other times hide information using redaction policies. Our design architecture provides a high level policy user with this option. Another advantage of this redundancy is that we can extend the modules in a policy layer without changing the functionality of the other policy layer.

\subsection*{6.2.2 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 (\textit{OO}); (2) lowest to highest utility (\textit{LHO}); and (3) highest to lowest utility (\textit{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 6.1.2.

\textbf{Algorithm 1} \textsc{Redact}\((G_q, RS)\)

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

\textbf{Algorithm 2} \textsc{Sort}\((G_q, RS)\)

\begin{verbatim}
1: SL = new List()
2: \textbf{for all} \textit{r} ∈ RS \textbf{do}
3: \textbf{if} \textit{r.se} ∈ \textit{G}_q \textbf{then}
4: \textbf{if} \textit{G}_q ∣ \textit{r} \textbf{then}
5: SL.add(\textit{r})
6: \textbf{end if}
7: \textbf{end if}
8: \textbf{end for}
9: \textbf{return} SL
\end{verbatim}

<table>
<thead>
<tr>
<th>\textit{c}_q</th>
<th>Order</th>
<th>\text{Redact}</th>
<th>\text{Rule}</th>
<th>\text{Emb}</th>
<th>\text{Sort}</th>
<th>\text{Diff}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\text{HLO}</td>
<td>17304</td>
<td>19</td>
<td>3</td>
<td>17241</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>\text{LHO}</td>
<td>41012</td>
<td>1853</td>
<td>7</td>
<td>39137</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>\text{HLO}</td>
<td>35270</td>
<td>28</td>
<td>2</td>
<td>35187</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>\text{LHO}</td>
<td>9044</td>
<td>2904</td>
<td>7</td>
<td>9100</td>
<td>27</td>
</tr>
</tbody>
</table>

Table 6.1 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 observation from Table 6.1 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 6.8 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 6.8 show the actual provenance graph size. Figure 6.8(a) compares the redaction time for our three utility conventions as the input graph to Algorithm 1 increases in size. The inset to Figure 6.8(a) shows that $OO$ 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 $HLO$ as shown in Figure 6.8(b).

Figure 6.9 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 $HLO$ while it picks a rule that alters the most triples in $G_q$ for $LHO$. Algorithm 1 picks any rule for $OO$.

At each transformation step, Algorithm 1 transforms $G_q$ by using rule $p$ at line 5. Rule $p$ is determined by applying either $LHO$ or $HLO$ 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, $p.e.$
Figure 6.8. Comparison of Redaction Time and Utility vs. Graph Size
Figure 6.9. Experimental Comparison of Complexity
6.3 Summary

In this chapter, we tell another part of the story for this thesis. 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 redaction policy architecture and on Semantic Web technologies (RDF, SPARQL) in order to evaluate the effectiveness of our graph rewriting system. In the next chapter, we plan on exploring other policy mechanisms for provenance: These include the strategies we use to decide when to protect provenance and when to release it.
In this chapter, we provide a short introduction to a formal model, which decides a comparable tradeoff for releasing both data and its associated provenance while hiding provenance in the case of intolerable losses. All of our security mechanisms explored so far for provenance suffers from what is known as an inference attack (see also Section 6.1.5). A user may employ different inference strategies in order to arrive at new information using prior knowledge and the answers to queries provided by our framework. When the newly information inferred is sensitive, we say the confidentiality of the system is compromised. To prevent the release of the confidential information, a first step is to identify the possible inference strategies available to the user.

An inference controller is a software device that guards against known inference attacks [133, 134]. We will explore different mechanisms for handling these attacks. An inference controller could be an autonomous module which lies between the interface layer and the data layers in our architecture (see Section 5.7 and Section 6.2). The Semantic Web offers many technologies that we can use to perform inferences. These include reasoners that support forward and backward chaining, classification and subsumption reasoning. For example, we can build more sophisticated SWRL rules and DL constraints on top of those given in Chapter 4. These new rules and constraints will allow us to detect the implicit knowledge in a knowledge base before any information is released from it. We can also leverage from existing Semantic Web frameworks that allow us to plug in any suitable reasoners. A suitable reasoner is one that implements any of the inference strategies supported by our inference controller.

So far we have made clear that provenance data contain relationships that can be used to infer highly sensitive data. For example, the provenance data may have an ancestry chain involving nodes of type "Secret Agent" or some top-secret algorithm. Revealing complete
provenance, including the chain, may reveal the identity of the sensitive information. In this chapter we proposed a new approach that takes both the utility of provenance and the risks of disclosing provenance into account, when we are adding provenance to the query answers [26].

7.1 Model

In this section, we present a formal presentation of our model, which is adapted from [8]. This model captures the essence of provenance and privacy risk. We first give a diagrammatic overview of the query process in Figure 7.1. This is followed by a formal description of our model and then a discussion of the modules in our system, which is composed of an external user system and an internal knowledge base system. Finally, we present a discussion on how to incorporate provenance into the model by means of utility functions. Figure 7.1 shows a query as input and a response (or answer) as output. The User System describes the process by which a user combines a subset, $C_1$, of the entire external collection available

![Figure 7.1. Processing a Query](image-url)
as background information with the history of previous answers to formulate a query. The Internal System describes the process that ensures a query adheres to the security policies. The security policies are used to ensure that we do not inadvertently release any information that can be classified as sensitive, private or confidential. We now present the steps composing our model:

1. \( \exists B_i \ i = \{1, 2, 3, \ldots, n\} \) a set of most likely backgrounds. Each \( B_i \) has a corresponding probability \( Pr_i \) that captures the likelihood of an attacker having background information \( B_i \in B \), the set of all probable backgrounds.

2. \( O_j \in O \) a set of outcomes, \( j = \{1, 2, 3, \ldots, m\} \).
   For any given query \( Q_j \in Q \) all possible query sets.
   \( Ans : Q \rightarrow K \subseteq 2^O \).

3. \( I : K \times B \rightarrow 2^O \), an inference function.

4. \( L : 2^O \rightarrow \mathbb{R} \), a loss function.
   \( U : 2^O \rightarrow \mathbb{R} \), a utility function.
   \( E[L(O)] = \sum_{i=1}^{n} L(I(O), B_i) \times Pr(B = B_i) \)
   \( E[L(Ans(Q_j))] \geq E[L(\emptyset)] \)

\( Prov(K) \) the provenance for \( K \).
\( P_i \) a privacy label, \( l = \{public, private\} \), \( P_i < P_{i+1} \ \forall l \).
\( \ell : O \rightarrow P \), maps \( O_i \) to labels.

In step one, we consider the case where a user has some background information associated with a likelihood of occurring. First, this is motivated by the fact that if no information is revealed (\( Ans(Q_j) = \emptyset \)), or if only complete noisy data is available after sanitization, then
there is no utility. In this case, the privacy of the data is not compromised. Second, if everything is known \((B_i \supseteq KB)\), then in the worst case we may have total loss of privacy. The first of these is not desirable to achieve any utility in the responses given to a user. The second is an extreme, which may be too pessimistic. In step two, we take into consideration that when a user queries the knowledge base system, the possibility exists that a user learns a subset of the knowledge base (that is, a user learns \(K\)). In step three, we note that a user could use the background information from step one and what is learnt in step two to draw inferences. Step four highlights the fact that what is learnt at step two may have an associated loss. A background could take on any of the possible subset of backgrounds from step one. Our model shows the expected loss associated with answering the query as opposed to not answering the query.

We need to estimate the parameters to our model to make it useful. We assume these are provided to the system before a query is answered. That is, \(L, Pr, U\) and \(B\) are given by an expert. This is usual when assigning values to unknowns in a system. For \(I\), we use a semantic web framework, which allows us to infer triples from the ones in a knowledge base. The function \(\ell\) maps a set of outputs to a privacy label, which is not limited to the ones we presented here. This is useful in determining whether we can infer private data. This function is used to determine which set of outputs to release to the user and which set we should not. In addition, if we could infer some private information using the labels, the private information that is inferred could be used as a guidance for defining the loss function.

Let \(D, P\) be the set of triples in the data and provenance collections respectively. Using the privacy labels from our model, we define a partition on \(D\) (\(P\)) as a public set and a private set denoted as \(D_V\) (\(P_V\)) and \(D_H\) (\(P_H\)) respectively. The set of labels is not limited to the ones presented here. The public set is visible to the user and the private is hidden. We assume a user combines \(B_i\) with each \(Ans(Q_j)\) to deduce new data, not explicit in \(D_V \cup P_V\).
7.1.1 The User’s System

A user may use the query history with an auxiliary source of information to build a reservoir of background knowledge. From this reservoir, a user can formulate specific queries that exploit the relationships between the background knowledge and the knowledge from the internal knowledge base system. It is not a trivial process to estimate the knowledge in this reservoir, since, for example, the sheer size of the web alone is intractable for the knowledge base system. For the User System in Figure 7.1, we assume that the user constrains the background over time as he/she builds a profile of the data from the internal knowledge base system (i.e. the user builds the User History collection). From this profile, the user adapts the background collection \( C_i \) for the next query. From this background collection and the answer given for the query, the user applies some inferencing techniques to make sense of the answer. It is this inferencing technique that makes it possible to learn some of the private data from the knowledge base system.

7.1.2 The Internal Knowledge Base System

The internal knowledge base system is composed of an interface module and a controller module (which implements an inference controller). We will speak of a controller module whenever it is clear that we are talking about the implementation of an inference controller. The interface module parses the query and also keeps a log of all the queries seen so far. We now describe the Interface module as follows:

1. Assign the user a privacy \( P_u \).
   This could be based on the context, attributes, authentication or credentials, etc., of the user.

2. Add \( Q_j \) to the query log.

3. Parse the query \( Q_j \).
   This is to identify and remove triple patterns with labels above \( P_u \) from the query
string. This ensures that we do not match the basic graph patterns (BGP’s) in the query string against the data in $D_V$.

7.1.2.1 The Controller

The Controller has the responsibility of fulfilling the request of the user. It retrieves the requested information by querying for a subset of $D_V$ and it also determines the provenance associated with $Ans(Q_J)$ as a subset of the data in $P_V$. The Controller then applies the privacy policies to the current results. Finally, it decides if the query can be answered according to the rules governing the release of data. The controller is described as follows:

1. $KB' = Ans(Q_J) \cup Prov(Ans(Q_J))$.
   If any answers are available from fulfilling the request of $Q_J$ from the set $D_V \cup P_V$, it is added to an empty knowledge base and we continue to the next step. Otherwise, there are no answers for $Q_J$. This case is most likely when it is not possible to release answers after satisfying the constraints from the privacy policies.

2. $KB' = KB' \cup H_Q$.
   The query history is derived from all the queries answered so far (e.g. $(Q_1, \ldots Q_{j-1}) \vdash H_Q$). $KB'$ is updated from the history.

3. $R(KB') \rightarrow KB' + Inf(KB')$,
   $R$ is a decidable RDFS/OWL reasoner.
   The reasoner performs inferencing over the knowledge base to determine newly inferred triples, different from those already in the knowledge base.

4. $\ell:Inf(KB') \rightarrow P$.
   $A = \{x | x \in KB', \ell(x) = P_i, P_i > P_u\}$.
   This determines all labels which are higher than that of the user.

5. $Ans(Q_J) = Ans(Q_J) \cup Prov(Ans(Q_J)) - A$;
   $H_Q = H_Q \cup Ans(Q_J)$.
   The answer set is then adjusted and the history reflects the new changes.
7.1.3 Adding Provenance

We now discuss when to provide provenance as part of the answer set given to the user. This process is very subjective and is based on the user’s utility function, as we show in the following two examples.

**Example 7.1.1** Assume that we are playing a game with the user, where we accept monetary payments for the services of the knowledge base system. Each time the system answers the user’s query, the user makes a payment commensurate with the utility of the answers.

**Example 7.1.2** Assume we are trying to catch a terrorist. By answering queries, we can facilitate the sharing of information, and this sharing of information can be used to capture a terrorist. Therefore, by answering the query, we help some tasks to get done or executed, which results in the capture of a terrorist.

We adapt our model to allow provenance to be part of the answer set for a query. We release provenance data to a user whenever $E[L(O)] + U(O) \geq E[L(\emptyset)]$. That is, we answer a query based on whether the addition of provenance improves the user’s utility value of the answers. In other words, when $U(I(Prov(Ans(Q,J)))) > \emptyset$.

Our model decides to answer a query as follows:

1. If $E[L(Prov(Ans(Q,J)) \cup Ans(Q,J))] + U(Prov(Ans(Q,J)) \cup Ans(Q,J)) \geq E[L(\emptyset)]$, release $Prov(Ans(Q,J)) \cup Ans(Q,J)$.

2. If $E[L(Ans(Q,J))] + U(Ans(Q,J)) \geq E[L(\emptyset)]$, release $Ans(Q,J)$.

3. Else release $\emptyset$.

In step one, we release provenance data if the user’s utility increases by adding provenance data. In step two, we release only the answers, since the utility resulting from integrating the provenance has not improved. If neither step one nor step two produces an answer, it means that our knowledge base system has no way of answering the user’s query without revealing some private information.
We now discuss the inference function in our model. This function takes two inputs, the answer for the current query and the background information, and infers new knowledge. We do not know the particular background information available to a user when a query is issued. However, we can conclude that the user has a large pool of data to rely on: The amount of information on the web grows daily and search engines make it easier to find information about any topic. Furthermore, there are efforts to represent information as RDF [7, 16, 17, 1], which increases the amount of linked data available to a user. Therefore, it is becoming increasingly easier for a user to link information in an answer set with any type of background information. We will study the behavior of a user under the assumption that the background knowledge can be encoded as rules. In addition, we limit our analysis to a healthcare domain, which was discussed in Section 5.8. The workflow in Section 5.8 is annotated with RDF triples to indicate contextual information about the patients and other entities; the annotations are a subset of the semistructured information on the Web.

We estimate the inferencing capabilities of the user by allowing the Controller in our internal knowledge base system to mimic some basic inferencing strategies. We use a combination of ontology reasoning, rule-base reasoning and heuristics to estimate the user’s background information. For estimating the background knowledge, we make use of the query logs and adapt the rules to anticipate certain types of inferences. The answers provided for a query is composed of different data from different databases in the knowledge base system as shown in Figure 7.1. The Controller combines data from different data stores when formulating responses. Important sources of information is the semistructured data available on the Web (e.g., Census data). This information can be cleaned and imported, thus allowing the controller to engage in limited record linking (see [137] for a background on record linkage methods). Although we cannot know the exact background of a user, we could use heuristics to limit the amount of inferencing by the user. Our controller also tracks the history of the interactions with the user, for example we are tracking the previous queries and previously released information.
In the rest of this section, we discuss various techniques for drawing inferences over the responses for a user’s query. We then discuss the use of ontologies, which can be used to classify data, followed by a discussion on basic inferences by rules, and finally we discuss how we can incorporate query logs as part of the inferencing process.

### 7.2.1 Inference Strategies

A user may employ different inference strategies in order to arrive at new information from the available background information and answers to all previous queries. When the newly inferred information is private, we say the confidentiality of the system is compromised. To prevent the release of the confidential information, a first step is to identify the possible inference strategies available to the user [134, 133].

#### 7.2.1.1 Inference by Deductive Reasoning

Deductive reasoning is the process of deriving the consequences of what is assumed. Given the truth of the assumptions, a valid deduction guarantees the truth of the conclusion [72]. In other words, deduction is the process of drawing inferences about a specific case on the basis of a general principle.

**Example 7.2.1** Consider the use case in Figure 5.3.

*If it is true that whenever med:HeartSurgery_1 is controlled by med:Surgeon_1, then the patient has a critical disease. If we know that John’s record was used by a process med:HeartSurgery_1 that was controlled by med:Surgeon_1, we can conclude that John has a critical disease.*

Let $r$ be an encoding of the rule: If it is true that whenever med:HeartSurgery_1 is controlled by med:Surgeon_1 then the patient has a critical disease, where $r$ is a prior knowledge. In order to prevent a user from learning that John has a critical disease, we would add $r$ to our knowledge base.

Assume $KB’$ contains the following triples:
Encode \( r \) as a DL constraint

\[
\text{HeartSurgery} \sqcap \exists \text{wasControlledBy}. \text{Surgeon} \sqsubseteq \text{Critical}
\]

Encode \( r \) as a SWRL rule

\[
(?p \text{wasControlledBy}\ ?s) \land \text{Surgeon}(?s) \land \text{HeartSurgery}(?p) \rightarrow \text{Critical}(?p)
\]

Forms of reasoning:

1. If A then B; A; therefore B. (modus ponens)
2. If A then B; not B; therefore not A. (modus tollens)
3. If A then B; If B then C; therefore If A then C. (Hypothetical Syllogism)
4. Either A or B; Not A; therefore B. (Disjunctive Syllogism)
5. If A then B; B; therefore A. (affirming the consequent premise)
6. If A then B; not A; therefore not B. (denying the antecedent premise)

Invalid Inferences:

- Affirming the Consequent
  
  (1) \( P \rightarrow Q \)
  
  (2) \( Q \)
  
  (3) Therefore: \( P \)

- Denying the Antecedant
  
  (1) \( P \rightarrow Q \)
  
  (2) \( \neg P \)
  
  (3) Therefore: \( \neg Q \)
7.2.1.2 Inference by Inductive Reasoning

Induction is the process of inferring probable conditional relevance as a result of observing multiple antecedents and consequents. An inductive statement requires empirical evidence for it to be true. In this strategy, well-formed rules are utilized to infer hypothesis from the examples observed. This allows inferring $\beta$ entails $\alpha$ from multiple instantiations of $\alpha$ and $\beta$ at the same time.

Example 7.2.2 Consider the following three observations:

Premise 1: Bob had a check up at day 1 and a follow up at day 4, Bob went into surgery.
Premise 2: Jack had a check up at day 1 and a follow up at day 5, Jack went into surgery.
Premise 3: Joe had a check up at day 1 and a follow up at day 4, Joe went into surgery.

Conclusion: Patient with follow up visit three days after a check up will undergo surgery.

Assume that a hospital has the following procedure: Patients diagnosed with critical heart complications are required to follow up with their physician within three days of being tested. After which time the physician should prepare the patient for heart surgery. Let $KB'$ contains the following triples:

```
<med:CheckUp_{Bob}_1> <opm:Used> <med:Doc_{Bob}_1>
<med:CheckUp_{Bob}_1> <med:TimeStamp> "1"
<med:RepeatVisit_{Bob}_1> <opm:Used> <med:Doc_{Bob}_1>
<med:RepeatVisit_{Bob}_1> <med:TimeStamp> "4"
<med:CheckUp_{Jack}_1> <opm:Used> <med:Doc_{Jack}_1>
<med:CheckUp_{Jack}_1> <med:TimeStamp> "1"
<med:RepeatVisit_{Jack}_1> <opm:Used> <med:Doc_{Jack}_1>
<med:RepeatVisit_{Jack}_1> <med:TimeStamp> "4"
<med:CheckUp_{Joe}_1> <opm:Used> <med:Doc_{Joe}_1>
<med:CheckUp_{Joe}_1> <med:TimeStamp> "1"
<med:RepeatVisit_{Joe}_1> <opm:Used> <med:Doc_{Joe}_1>
<med:RepeatVisit_{Joe}_1> <med:TimeStamp> "4"
```

In order to prevent the user from learning this rule, we could limit the amount of cases the user can observe. For example, we could enforce the following constraint in the knowledge base.
(CheckUp ⊓ ≥3Used) ⊓ (RepeatVisit ⊓ ≥3Used) ⊑ Critical

### 7.2.1.3 Abduction

This allows inferring \( a \) as an explanation of \( b \). Because of this, abduction allows the precondition \( a \) to be inferred from the consequence \( b \). Deduction and abduction thus differ in the direction in which a rule like “\( a \) entails \( b \)” is used for inference. As such abduction is formally equivalent to the logical fallacy affirming the consequent or *Post hoc ergo propter hoc*, because there are multiple possible explanations for \( b \). Abduction is typically defined as inference to the best explanation (e.g. [88, 105]). Given \( \alpha, \beta \) and the rule \( R1 : \alpha \vdash \beta \); then deduction is using the rule and its preconditions to make a conclusion \( (\alpha \land R1 \Rightarrow \beta) \); induction is learning \( R1 \) after seeing numerous examples of \( \alpha \) and \( \beta \); and abduction is using the postcondition and the rule to assume that the precondition could explain the postcondition \( (\beta \land R1 \Rightarrow \alpha) \). More formally, abduction is the search for assumptions \( A \) which, when combined with some theory \( T \) achieves some set of goals \( G \) without causing some contradiction [25]. That is:

\[
EQ_1 : V \cup A \vdash G
\]

\[
EQ_2 : T \cup A \not\vdash \bot
\]

### Example 7.2.3

Consider the use case in Figure 5.3 and the rule:

*Whenever med:HeartSurgery_1 is controlled by med:Surgeon_1 then the patient has a critical disease.* If we know that a patient Bob has a critical disease, we try to infer that med:Surgeon_1 performed the process med:HeartSurgery_1 as the explanation for Bob’s disease. In other words, the explanation is that a surgeon who is identified as med:Surgeon_1 performed the surgery on Bob.

It could be that Bob’s disease is not related to heart surgery. The set of conditions leading to critical conditions varies and is not limited to one rule, or there may not be any one (best) explanation for an observation. Abductive reasoning starts when an inquirer considers a set
of seemingly unrelated facts, armed with an intuition that they are somehow connected:

\[ D \text{ is a collection of data} \]

Hypothesis \( H \) explains \( D \)

No other hypothesis explains \( D \) as well as \( H \) does

Therefore, \( H \) is probably correct

### 7.2.1.4 Inference by Analogical Reasoning

In reasoning by analogy, statements, such as “\( X \) is like \( Y \)” , are used to infer properties of \( X \) when given the properties of \( Y \).

**Example 7.2.4** If the properties of an entity \( A \) are a secret and the properties of an entity \( B \) are prior knowledge of a user. Further, if the statement “\( A \) is like \( B \)” is also prior knowledge, then a user could infer the properties of \( A \).

**Example 7.2.5** Patient 2 had a check up and went into surgery immediately after the follow up visit. Patient 1 also had a check up and a follow up visit. The user also knows that Patient 1 and Patient 2 are the same age, same height and same ethnicity. The user may reason that Patient 1 also had the same disease as Patient 2 and therefore underwent the same surgery procedure.

An analogy can be seen as reasoning or explaining from parallel cases. In other words, an analogy is a comparison between two different things in order to highlight some point of similarity. An argument from analogy could an argument that has the form:

\[ \text{All P are like Q} \]

\[ Q \text{ has such-and-such characteristic.} \]

Thus \( P \) has such-and-such characteristic.
7.2.1.5 Inference by Heuristic Reasoning

Heuristics are criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal [133, 108]. In general, a heuristic is not well defined and may be a rule of thumb that is used to guide one’s actions. Experts often use heuristics in order to solve a problem. Inference by heuristic reasoning is the process of deducing new information using various heuristics.

Example 7.2.6 Given some information $\alpha$, heuristic rules and past experience are used to infer some information $\beta$.

The web contains many sources of information about patients, hospital procedures and physicians. There may be many formal procedures, tests and treatments for a disease. Given some information $\alpha$, which describes Bob, the user may wish to determine Bob’s disease. However, the user may not be able to acquire all relevant information to derive the correct disease for Bob: It may not be possible to use our question-answering interface to get all the information about Bob and it may not be possible to research all possible explanations for Bob’s condition given the sheer size of the web (and prior knowledge). In this situation, the user may use shortcuts (or heuristic rules) and past experiences to infer Bob’s disease.

7.2.1.6 Inference by Semantic Association

In this strategy, association between entities is inferred from the knowledge of the entities themselves [133]. Various types of semantic associations have been identified. They include context-based associations, aggregation-based associations, and dependency-based associations.

Example 7.2.7 Consider the use case in Figure 5.3.

Assume there is a semantic relation between med:Surgeon$_1$ the process med:HeartSurgery$_1$. Then, by revealing either the surgeon identified as med:Surgeon$_1$ or the the process identified as med:HeartSurgery$_1$ would reveal the identities of the surgeon or the process. Further,
these two entities, med:Surgeon_1 and med:HeartSurgery_1 taken together would reveal a critical condition about Patient 1.

7.2.1.7 Inferred Existence

Using the strategy of inferred existence, one can infer the existence of an entity Y from certain information on Y [134].

Example 7.2.8 From the triple

med:HeartSurgery_1 opm:wasControlledBy med:Surgeon_1

we can infer that there is a surgeon.

7.2.2 Ontologies

Tools exist that support RDF and its semantic extensions (e.g. RDF, RDFS, OWL). These tools support reasoners. An example is Jena [32]. This is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS, OWL and SPARQL and includes a rule-based inference engine.

Knowledge representation languages like RDF, RDFS and OWL are used for creating ontologies. An ontology is a specification that represents knowledge as a set of individuals and a set of property assertions, which relate these individuals to each other. It consists of a set of axioms which place constraints on sets of individuals (called classes) and the types of relationships (or properties) permitted between them. Data items are connected to each other and to literal data via properties. A statement in an ontology is of the form of a triple (s p o).

An ontology for provenance would model the data items and processes as instances of a class and the links between them as properties. In this ontology, we could have statements like

med:CheckUp_1 opm:wasControlledBy med:Physician_1
An ontology for the open provenance model [96, 144] contains three basic entities: Artifact, Process and Agent. The instances of these objects are the nodes in a provenance graph. This model represents an edge as a causal dependency, between its source, denoting the effect, and its destination, denoting the cause [97]. The semantics of RDF and OWL allow us to define terms so that a reasoner can make logical inferences about the interconnections among the data items and processes in the provenance graph. Given a provenance graph, all of the direct and indirect connections it has to other data items and processes can be inferred and then retrieved by a SPARQL query [56].

### 7.2.3 Rules

For the inferences that are not captured by modeling the data as an ontology, we add rules to the knowledge base. That is, we add rules to step three for the controller module described in section 7.1.2.1. Rules are of two types: those used to express the privacy policies and those used to improve the inferencing power of the controller. So far we assume a user combines his/her background with the information in the answers to infer new data, which was not explicit in $D_V \cup P_V$. We formalize this as an inference rule:

$$\bigwedge_i^n B_i \land \bigwedge_{j=1}^m \text{Ans}(Q_j) \land \text{Prov}(\text{Ans}(Q_j)) \land \text{Ans}(Q_{m+1})$$

$$\land \text{Prov}(\text{Ans}(Q_{m+1})) \rightarrow \varphi.$$

This rule represents basic implication and is a method for deducing information from existing ones. We also need a way to assign labels to triples in a knowledge base. For example, if there is a person *Jack* in our knowledge base, and some information about *Jack* is private, then if the fact that *Jack* has the flu is private we must label the triple, (*Jack* has *Disease* flu) as private. In addition, if part of the triple is private, we need to make the entire triple private. For instance, if *Jack* is private, then revealing any information in triples with *Jack* as the subject or object would reveal that there is someone named *Jack* in the knowledge base system.

**Definition 7.2.1** (*Private Triple*). Let $t$ be a triple of subparts $s$, $p$ and $o$ and $L_u$ the privacy label for a user. For any subpart $v$ of $t$, if $\ell(v) = L$, and $L > L_u$, then $\ell(t) = L$. 
The classification of an inferred triple $\varphi$ using the basic implication rule could be used to collect statistics about a user. By keeping track of how many times $\varphi$ is labeled private from Definition 7.2.1 since the last $n$ queries, as well as the total number of times this happens, we can formulate a profile of the background knowledge of the user. This could be used to build new rules or adjust existing ones to enforced the policies under the new estimates of the user’s background patterns.

There are different ways we can build our rules. In an association rule, $\alpha \land \beta \rightarrow \varphi$, if $(\ell(\alpha) \leq L_u) \land (\ell(\beta) \leq L_u)$ and we already release $\alpha$, we may want to adjust the label for $\beta$ to $(\ell(\beta) > L_u)$. This follows from the fact that both $\alpha$ and $\beta$ when taken together reveal private information.

### 7.2.4 Query Logs

Posing queries to a dataset to get answers with high utility is not new. In information retrieval (IR), users pose similar queries against sets of documents, and various literature highlights that these queries have context [10, 122]. The context could determine the user’s domain of interest, knowledge, preferences, relevance, judgments, etc. These could be used to construct a user profile to reflect the user’s domains of interest and background. In IR, the use of previous queries and clickthrough history information has been suggested to improve accuracy of the answers. This is worth considering for our initial model. We could also perform a fine-grained recording of an agent’s previous activities in order to examine in more detail the potential threat posed by each query. This would allow us to capture a subset of the background associated with the user’s queries. Under our RDF data model, SPARQL is the standard query language. A simple syntax of a SELECT query in SPARQL is

\[
\text{SELECT } \vec{V} \text{ FROM } u \text{ WHERE } \{ \ T \ },
\]

where $u$ is the URL of an RDF graph $G$, $T$ is a SPARQL graph pattern and $\vec{V}$ is a tuple of variables appearing in $T$. 
For brevity, $T$ is matched against $G$ and bindings are returned for each solution for the variables in $T$. The SELECT is a projection over the solutions from the bindings. An answer may provide the user with feedback about the matching of $T$. If the query was matched, and we return an answer, the user knows that $T$ exists in the knowledge base (i.e. $T \subseteq KB$). If there is no answer, then the user still knows something about the pattern. However, under the open world assumption of RDF, there is a probability that $T \subseteq KB$, either it is part of the private section $\mathcal{D}_H \cup \mathcal{P}_H$ or it is not part of the system (i.e. $KB \cap T = \emptyset$). Therefore, keeping query logs could provide useful information about the possible inferencing capabilities and patterns of the user’s background.
In this chapter, we integrate the various parts of the system into an automatic framework for provenance. We preserve the features discussed in Chapter 4, such as scalability, efficiency, and interoperability. This framework can be used to execute various policies, including access control policies (which was the subject of Chapter 5), redaction policies (which was the subject of Chapter 6) and inference strategies (which was the subject of Chapter 7). Our framework can also be used as a testbed for evaluating different policy sets over a provenance graph and their outcomes graphically. Our recent work in [28] proposes new mechanisms for extending our framework. One of them includes comparing the words described by regular expression queries to determine equivalence and subsumption of policies. Hence, we will be able to compare and write more compact policies as well as eliminate redundancies and inefficiencies.

8.1 Design of Our Framework

The final architecture for our provenance manager is extended to include an inference controller. This enables us to add the risk-based mechanism discussed in Chapter 7 into our framework. Our architecture takes a user’s input query and returns a response which has been pruned using a set of user-defined policy constraints. We assume that a user could interact with our system to obtain both traditional data and provenance. In our design we will assume that the available information is divided into two parts: the actual data and provenance. Both, the data and provenance are represented as RDF graphs, but they are not limited to any data format since tools can map existing formats to RDF [15].

The architecture is built using a modular approach, therefore it is very flexible in that most of the modules can be extended or replaced by another application module. For
example, an application user may substitute a policy parser module that handles the parsing of high-level business policies to low-level policy objects; or replace or extend one policy layer without changing the inner workings of the other policy layer modules. This substitution or replacement of modules would allow the application user to continue using high-level business policies independent of our software implementation.

A user application can submit a query for access to the data and its associated provenance or vice versa. Figure 8.1 shows our design and modules in our prototype implementation. We now present a description of these modules in Figure 8.1.

**User Interface Manager**

The User Interface Manager is responsible for processing the user’s requests, authenticating the user and providing suitable responses back to the user. The interface manager also provides an abstraction layer that allows a user to interact with the system. A user can therefore pose either a data query or a provenance query to this layer. The user interface
manager also determines whether the query should be evaluated against the traditional data or provenance.

**Policy Manager**

The Policy Manager is responsible for ensuring that the querying user is authorized to use the system. It evaluates the policies against a user’s query and associated query results to ensure that no confidential information is released to unauthorized users. The policy manager may enforce the policies against the traditional data or against the provenance data. Each data type may have its own policy manager, for example the traditional data may be stored in a different format from the provenance data. Hence, we may require different implementations for each policy manager.

**Inference Engine**

The Inference Engine is the heart of the inference controller described in Chapter 7. The engine is equipt to use a variety of inference strategies that are supported by a particular reasoner. Since there are many implementations of reasoners available, our inference controller offers an added feature of flexibility, whereby we can select from among any reasoning tool for each reasoning task. We can improve the efficiency of the inference controller since each inference strategy (or a combination of strategies) could be executed on a separate processor. An inference engine typically uses software programs that have the capability of reasoning over some data representation, for example a relational data model or a RDF graph model representation. The inference controller is basically an implementation of the function $I : K \times B \rightarrow 2^O$, which was introduced in Section 7.1. Basically, the inference controller is used to address the inference problem.

The inference problem is an open problem and a lot of research has been pivoted around its implementations based on traditional databases [134, 140, 91, 67, 33, 43, 91, 140]. However, since provenance has a logical graph structure, it can also be represented and stored in a graph data model, therefore it is not limited to any particular data format. Although our focus in this thesis is on building an inference controller over the directed graph rep-
presentation of provenance, our inference controller could be used to protect the case when provenance is represented and stored in a traditional relational database model. Also, the use of a RDF data model does not overburden our implementation with restrictions, since other data formats are well served by a RDF data model. Furthermore, tools such as [15] converts relational data to RDF data and [71] provides a set of tools to convert various data formats to RDF.

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 Policy manager (and/or the inference engine layer) if a policy allows the requesting user access to the data item. This layer then executes the request over the stored data and returns results back to the policy layer (and/or the inference engine layer) where it is re-evaluated based on a set of 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. In the case when we select a graph representation of provenance, 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 graph query and evaluates it over the provenance information. This query evaluation returns a subgraph back to the inference controller layer where it is re-examined using a set of policies.
8.2 Inference Controller

In Chapter 7, we presented a model which can be used to determine the expected risk of releasing provenance. We further extend this model in Section 7.1.3 in order to determine whether the addition of provenance improves the user’s utility when interacting with our system. The extended model formally describes what we hope to accomplish by incorporating inference tools into our architecture. In other words, we release provenance data to a user whenever $E[L(O)] + U(O) \geq E[L(\emptyset)]$. By carefully adjusting $E[L(\emptyset)]$, we can adjust how much provenance we release based on the identity of a querying user. This is important, for example, in research whereby some provenance is released for the advancement of science. Our inference controller is aware of these special cases and has features for supporting multi-level users. Recall that our model supports different labels; therefore we could label the different users, for example, $l = \{\text{provider}, \text{researcher}, \text{pharmacist}, \text{intern}, \text{physician}\}$, $P_l < P_{l+1}$, $\forall l$.

We can also use our inference controller to provide feedback so that a high-level domain user can reconfigure the business rules (or high-level policy set).

8.3 Inference Tools

Newly published data, when combined with existing public knowledge, allows for complex and sometimes unintended inferences. Therefore, we need semi-automated tools for detecting these inferences prior to releasing provenance information. These tools should give data owners a fuller understanding of the implications of releasing the provenance information, as well as helping them adjust the amount of information they release in order to avoid unwanted inferences [129].

The inference controller is a tool that implements some of the inference strategies that a user may use to infer confidential information that is encoded into a provenance graph. Our inference controller leverages from existing software tools that perform inferencing, for example pellet [127], Fact++ [136], Racer [63], hermit [121], and CWM [12]. Therefore, we can add more expressive power by replacing the default base engine of our inference con-
controller with a more powerful reasoner. Furthermore, since there is a trade-off of expressivity and decidability, an application user has more flexibility in selecting the most appropriate reasoner for his/her application domain.

For our default reasoner, we employ the services of Pellet [127]. Pellet has support for OWL-DL (SHOIN(D) and is also extended to support OWL 2 specification (SROIQ(D)). The OWL 2 specification adds the following language constructs:

- qualified cardinality restrictions
- complex subproperty axioms (between a property chain and a property)
- local reflexivity restrictions
- reflexive, irreflexive, symmetric, and anti-symmetric properties
- disjoint properties
- negative property assertions
- vocabulary sharing (punning) between individuals, classes, and properties
- user-defined dataranges

In addition Pellet provides all the standard inference services that are traditionally provided by DL reasoners. These are:

- **Consistency checking**
  This ensures that an ontology does not contain any contradictory facts. The OWL 2 Direct Semantics provides the formal definition of ontology consistency used by Pellet.

- **Concept satisfiability**
  This determines whether its possible for a class to have any instances. If a class is unsatisfiable, then defining an instance of that class will cause the whole ontology to be inconsistent.
• **Classification**
  
  This computes the subclass relations between every named class to create the complete class hierarchy. The class hierarchy can be used to answer queries such as getting all or only the direct subclasses of a class [127].

• **Realization**

  This finds the most specific classes that an individual belongs to; i.e., realization computes the direct types for each of the individuals. Realization can only be performed after classification since direct types are defined with respect to a class hierarchy [127]. Using the classification hierarchy, it is also possible to get all the types for each individual.

### 8.4 Interface Module

The User Interface Module provides a layer of abstraction that allows a user to interact with the system. The user interacts with the system via a **User Interface Layer**. This layer accepts a user’s credentials and authenticates the user. Our interface module hides the actual internal representation of our system 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. This layer also returns results after they have been examined against a set of policies. Figure 8.4 is a representation of our interface module, which allows a user to interact with the underlying provenance store.

**The query processing module** in Figure 8.4 is responsible for accepting a user’s query, parsing it and submitting it to the provenance knowledge base. After the query results are evaluated against a set of policies, it is returned to the user via the User Interface Layer. The query processing module can accept any standard provenance query as well as any query written in the SPARQL format. A standard query can be any of the ones discussed
in Section 5.8.1. The querying user is allowed to view the errors that are due to the syntax of a query, as well as the responses constructed by the underlying processes of the inference controller.

### 8.5 Policy Module

The Policy Module is responsible for enforcing any high-level policy defined by an high-level application user or administrator. The policies are not restricted to any particular security policy definition, model or mechanism. In fact, we can support different access control policies, for example role-based access control (RBAC), access control based on context such as time (TRBAC), location (LBAC), etc. Besides the traditional and well-established security models built on top of access control mechanisms, we also support redaction policies that are based on sharing data for the ongoing mutual relationships among businesses and stakeholders. The policy layer also interacts with any reasoners in the inference layer, which offer further protection against inference attacks. The inference layer enforces policies that are in the form of DL constraints, OWL restrictions or SWRL rules. We also observe that
some of the access control policies can be expressed as inference rules or queries via query rewrite or views. Our policy module therefore has many layers equipped with security features, thus ensuring we are enforcing the maximal protection over the underlying provenance store.

Figure 8.3 shows the policy modules making up the policy manager. The user interacts with the policy modules via the query processing module. Each query passed to the policy manager from the query processing module is evaluated against a set of policies. These policies can be encoded as access control rules via any access control mechanism or suitable policy language (for example, the language in [29]). The policies can also be expressed as rules that operate directly over a directed graph or they can be encoded as DL constraints.
or SWRL rules. The default reasoner employed at the inference layer is Pellet [127], which has support for the following.

- SWRL Rules Support
- SPARQL-DL Conjunctive Query Answering
- Datatype Reasoning
- Multi-Ontology Reasoning using E-Connections
- Incremental Reasoning
- Ontology Analysis and Repair
- Ontology Debugging

8.6 Parsing Process

A high-level policy has to be translated to a suitable format and representation in order to be processed by a provenance inference controller. This often involves the parsing of a high-level policy to a low-level representation. Our design makes use of an extensible language for expressing policies. This language has been used successfully to write access control policies [100, 102]. Our policies are written as XML documents [24], which resides on disk until they are requested. XML is also equipt with features for writing rules [60, 58, 70] and representing RDF and OWL in a XML syntax [11, 69]. Our choice of a XML language allows us to take as input any high-level policy specification and an associated parser that maps it to a low-level policy format. The high-level application user also benefits from our use of a XML language, since XML is an open standard that is widely used and many data exchange formats are based on XML. For the rest of this thesis, we will refer to the policies as though they are in their XML standard form.

Figure 8.4 provides us with an overview of a policy parsing process. When a XML policy file is loaded, each policy in the policy file is parsed using a compatible parser. The parser is
responsible for ensuring that the policies are well-formed. The default policies (i.e., access control, redaction, inference rules) are written in a XML file and the parser evaluates the XML file against a XML schema file. The policies in a successfully parsed XML file are then translated to a low-level representation.

8.7 High-Level Policy Translation

In this section, we will discuss how a correctly parsed high-level policy is translated to an internal low-level policy. We will first discuss two inference assemblers, the SWRL Rule assembler and the Description Logics Rule Assembler. Then we will discuss two policy assemblers, which translate the access control and redaction high-level policies respectively.

8.7.1 SWRL Rule Assembler

This module maps a high level XML file onto a set of SWRL rules. A SWRL rule has a head and a body. The body is used to encode a condition that must be satisfied before the
information encoded in the head is applied to the provenance knowledge base.

**A SWRL policy translation**

The following is a policy which states that if a doctor has (or is attending to) a patient, then that doctor can also read the patient’s record.

```xml
<policies>
  <policy ID="1">
    <description>...some description....</description>
    <body>
      <atom>?x rdf:type provac:Doctor</atom>
      <atom>?y rdf:type provac:Patient</atom>
      <atom>?y provac:patientHasDoctor ?x</atom>
      <atom>?y provac:hasRecord ?r</atom>
    </body>
    <head>
      <atom>?x provac:canReadRecord ?r</atom>
    </head>
  </policy>
</policies>
```

This policy could be represented internally as

\[
\text{Doctor}(?x) \land \text{Patient}(?y) \land \text{patientHasDoctor}(?y, ?x) \land \text{hasRecord}(?y, ?r) \\
\rightarrow \text{canReadRecord}(?x, ?r).
\]

**8.7.2 Description Logics Rule Assembler**

This module maps a high level XML file onto a set of OWL restrictions. The OWL properties are used to create restrictions, which are used to restrict the individuals that belong to a class. These restrictions can be placed into three main categories:

1. Quantifier Restrictions
2. Cardinality Restrictions
3. hasValue Restrictions
**Quantifier Restrictions.** Quantifier restrictions consist of three parts:

1. A quantifier, which is either the existential quantifier (∃), or the universal quantifier (∀).
2. A property, along which the restriction acts.
3. A filler that is a class description.

For a given individual, the quantifier effectively puts constraints on the relationships that the individual participates in. It does this by either specifying that at least one kind of relationship must exist, or by specifying the only kinds of relationships that can exist (if they exist). An example of an existential quantifier can be used to define a physician as someone with a medical degree:

\[ \text{Physician} \sqsubseteq \exists \text{has.MedicalDegree}. \]

Universal restriction states that if a relationship exists for the property then it must be to individuals that are members of a specific class. An example of a universal quantifier can be used to define a happy physician as someone, all of whose patients have insurance:

\[ \text{HappyPhysician} \sqsubseteq \forall \text{hasPatients.(∃hasCoverage.Insurer)}. \]

**Cardinality Restrictions.** OWL Cardinality Restrictions describe the class of individuals that have at least (≥), at most (≤) or exactly a specified number of relationships with other individuals or datatype values. Let \( P \) be a property, then

1. A Minimum Cardinality Restriction specifies the minimum number of \( P \) relationships that an individual must participate in.
2. A Maximum Cardinality Restriction specifies the maximum number of \( P \) relationships that an individual can participate in.
3. A Cardinality Restriction specifies the exact number of \( P \) relationships that an individual must participate in.
**hasValue restriction.** A hasValue restriction describes the set of individuals that have at least one relationship along a specified property to a specific individual. The hasValue restriction is denoted by the symbol $\in$. An example of a hasValue restriction is $\text{hasCountryOfOrigin} \in \text{Italy}$ (where Italy is an individual). This describes the set of individuals (the anonymous class of individuals) that have at least one relationship along the hasCountryOfOrigin property to the specific individual Italy.

**Supporting Restrictions**

We currently supported the following OWL restrictions:

1. **SomeValuesFromRestriction**

   SomeValuesFrom restrictions are existential restrictions which describe the set of individuals that have at least one specific kind of relationship to individuals that are members of a specific class.

2. **AllValuesFromRestriction**

   AllValuesFromRestriction are Universal restrictions which constrain the filler for a given property to a specific class.

3. **MinCardinalityRestriction**

   MinCardinalityRestriction are cardinality restrictions which specify the minimum number of relationships that an individual must participate in for a given property. The symbol for a minimum cardinality restriction is the 'greater than or equal to' symbol ($\geq$).

4. **MaxCardinalityRestriction**

   MaxCardinalityRestriction are cardinality restrictions which specify the maximum number of relationships that an individual can participate in for a given property. The symbol for maximum cardinality restrictions is the 'less than or equal to' symbol ($\leq$).

5. **DataRange**

   This is a built-in property that links a property (or some instance of the class rdf:Property)
to either a class description or a data range. An rdfs:range axiom asserts that the values of this property must belong to the class extension of the class description or to data values in the specified data range.

6. **Domain**

This is a built-in property that links a property (or some instance of the class rdf:Property) to a class description. An rdfs:domain axiom asserts that the subjects of such property statements must belong to the class extension of the indicated class description.

### A DL policy translation

The following is a policy which states that any process that is controlled by a surgeon is a sensitive process.

```xml
<policies>
  <policy ID="1">
    <description>...some description....</description>
    <rule>
      <restriction>AllValuesFromRestriction</restriction>
      <property>opm:WasControlledBy</property>
      <class>provac:Surgeon</class>
      <label>provac:SensitiveProcess</label>
    </rule>
  </policy>
</policies>
```

This policy is converted internally as

\[ \forall \text{WascontrolledBy.Surgeon} \sqsubseteq \text{SensitiveProcess}. \]

### 8.7.3 Access Control Policy Assembler

This module maps a high-level access control XML policy file to a low-level access control policy.
An Access control policy translation

The following is a policy which states that any user has permission to access Doc_2 if it was generated by a process that was controlled by a surgeon.

```
<policies>
  <policy ID="1">
    <description>description</description>
    <target>
      <subject>anyuser</subject>
      <record>provac:Doc_2</record>
      <restriction>Doc.WasGeneratedBy == opm:Process</restriction>
      <restriction>process.WasControlledBy == provac:Surgeon</restriction>
    </target>
    <effect>NecessaryPermit</effect>
  </policy>
</policies>
```

This policy could be translated to a query that retrieves the part of a provenance graph that this policy is allowing a user to view. A corresponding SPARQL query would then be

```
Select ?x
{
  med:Doc1_2 gleen:OnPath("([opm:WasGeneratedBy]/
                 [opm:WasControlledBy])" ?x
}
```

8.7.4 Redaction Policy Assembler

This module maps a high-level XML redaction policy file to a low-level redaction policy.

A Redaction policy translation

The following is a policy which states that if there is a path which starts at Doc_4 and Doc_4 was derived from an artifact which was generated by a process that was controlled by a physician, then we should redact this path from the provenance subgraph containing the path.
This policy would evaluate over a provenance graph replacing any path that starts with a node labeled Doc_4 and connected to a process via a WasGeneratedBy link followed by a WasControlledBy link which has an end node labeled as physician (or is of type physician). Each such path would be replaced by a blank label _:A1 and _:A1 would be joined to the original provenance graph to some node labeled provac:HearthSurgery_1 using a link with the label opm:Used.

8.8 Explanation Service Layer

A cool feature to have is one where the reasoner derives new knowledge and then explains how it derived that new knowledge. For this we could rely on a reasoner which provides some explanation service.

The default base reasoner, pellet [127] has a service that can explain its inferences by providing the minimal set of facts or other knowledge necessary to justify the inference. For any
inference that Pellet computes, we exploit Pellet inference service, which will explain why that inference holds. The explanation itself is a set of OWL axioms which, taken together, justify or support the inference in question. There may be many (even infinitely many) explanations for an inference; Pellet heuristically attempts to provide a good explanation.

Our provenance inference controller can then provide information about the classification of the knowledge base. For example, we may be interested in why a set of RDF triples were classified as sensitive, or why a concept is considered sensitive. The answers to these questions are left to the Explanation Service layer. This layer is built on top of Pellet explanation service and displays the set of axioms used to derive the concepts that are subsumed by another class.

The Explanation Service layer uses the services of a reasoner to provide justifications (also warrants) for each piece of the provenance that is classified as sensitive. The explanation service layer is also useful for providing feedback to the application designer or policy administrator.

The explanations displayed back to a policy administrator may be in terms of the low-level descriptions. Furthermore, the explanations may reveal low-level details of the particular software implementation. This is an optional feature, which can be turned on for policy designers familiar with Description logics or OWL. This service provides a desired feature, whereby the application designer can view how the policies are interpreted by the low-level inference services. For example, since a high-level description logic rule may be applied differently from what the author intended, the policy designer now has an opportunity to tweet the high-level policies for the desired outcome.

8.9 Provenance in a Healthcare Domain

The healthcare domain sees provenance as a critical component of its operations. The provenance can be used to facilitate the communication and coordination between organizations and among members of a medical team. It can be used to provide an integrated view of the execution of treatment processes, to analyze the performance of distributed healthcare
services, and to carry out audits of a system to assess that, for a given patient, the proper decisions were made and the proper procedures were followed [77].

We describe a medical domain with respect to sources available online such as http://www.webmd.com/. Our medical domain is made up of patients, physicians, nurses, technicians, equipments, medical procedures, etc. We focus on one example of the medical domain, therefore a fictitious hospital. This is a toy example of a hospital that carries out procedures described at credible websites such as http://www.nlm.nih.gov/ and http://www.mghp.com/services/procedure/. These procedures include heart surgery procedures, hip replacement procedures, etc. Since the procedures are described by actual documents on the Web, our generated workflow structures typically follow a set of guidelines that are also known to the user. However, the workflows generated by our system may not reflect exactly what goes on in a real hospital. We take into consideration that real hospitals follow guidelines related to a patient’s privacy, therefore our fictitious hospital generates workflows so that the entities in the provenance graph are known only internally. This ensures that the content of a record (i.e., an artifact), the agent who generated a version of a record, the time when the record was updated and the workflow processes are only revealed to a user via queries. Furthermore, the laws governing the release of the provenance (i.e., the contents of the generated workflow) are enforced by policies, which are implemented by translating them into a suitable format for use internally by our system.

8.10 Populating the Provenance Knowledge Base

The provenance knowledge base is updated using a set of generators. There are background generators, which are responsible for extracting background information which is normally available online. There is also a workflow generator that produces the actual provenance. The workflow generator produces synthetic provenance data that is not available online. It is this provenance data that has subsets which we must protect. We populate the provenance store by extracting information related to a healthcare domain. The healthcare domain is suitable in two ways. First, this domain actually records provenance [77]; and second, data about this domain is partially available online.
Generating and Populating the Knowledge Base

We create a set of seeds which consist of a first name, a last name, a state and city. Each seed is used to create a query which is issued against the http://www.yellowpages.com/ or the http://www.whitepages.com/ website. These websites are useful for locating businesses and individuals via search terms. In order to extract information from these websites, we employ the services of a web crawler. A Web crawler is a computer program that browses the World Wide Web in a methodical, automated manner or in an orderly fashion. Web crawlers are sometimes referred to as automatic indexers, bots, ants, harvesters, Web spiders, Web robots, or Web scutters. These crawlers are computer programs that follow the link structure of the World Wide Web and perform some tasks on the web pages they visit.

After the pages matching our initial query seed are crawled, we store the results in an appropriate format in a text file. Because this process is expensive, we build all our web crawl routines off-line, and load the text file contents into memory during the test cycles of our experimental phase. The first crawl gathers our assumed lists of patients. We use the zip codes of the patients to create queries for hospitals, doctors and their specialties. The results of these searches are also stored in text files, which have predetermined formats. These predetermined formatted text files allow us to build object classes with properties for persons, hospitals, doctors, nurses, etc.

8.11 Generators

We now discuss the generators we used to extract background and workflow information. We make use of three set of generators in order to systematically extract data from web pages. There is a set of background generators, which are responsible for extracting background information which is normally available online. This background information is public and is available to all users of our system. These is also a set of miscellaneous generators which build synthetic data about diseases, medication, tests and treatments. The miscel-
laneous generator uses online sources to guide it in associating diseases with the related
tests, treatment and diseases, thus there is additional background information produced by
these miscellaneous generators. Finally, there is a workflow generator that produces the
actual provenance. The workflow generator produces synthetic provenance data that is not
available online. It is this provenance data that has subsets that we must protect.

8.11.1 Selecting Background Information

We use real information that actually exists on current web pages so that we can demonstrate
the effectiveness of the inference controller with respect to a set of prior knowledge of the
querying agent. We identify a City and State in the USA. For this city, we target a set of zip
codes. The information is downloaded from freely available websites such as yellow pages and
white pages. We crawl these websites and extract the name, address, telephone numbers,
age, sex and relatives of various individuals, by setting a seed for a list of popular first
and last names. This would allow us to capture similar attribute values for each individual
patient in our toy hospital. For the hospitals, we select only those hospitals within the zip
codes for the patients. Each hospital has a name, address and telephone numbers. Because
many hospitals do not release the names of their staff, we perform searches for doctors and
their specialty within the same zip codes. This is normal, since most specialists are affiliated
with a particular hospital close to their practice. Some insurance companies do provide a
list of the doctors and their affiliation on their websites, but many of these websites require
a login id, or different verification code each time it is accessed. Due to these obstacles, we
satisfy with a less accurate picture of the actual hospital. Also, since our system is user
driven, automation and efficiency became a greater priority. This does not preclude a client
from populating the knowledge base with their own data.

Generating data this way makes the system more realistic than if we had used complete
synthetic data. A querying user can combine the responses from the system with accessible
background information to draw inferences. The querying user could then issue new queries
to verify their guesses about the data in the knowledge base.
8.11.2 Background Generator Module

Figure 8.5 is a diagram of the different background generators. Each generator is built to target specific websites (or pages), which contain some information of interest. For example www.ratemds.com provides structured information about doctors at a specific zipcode. This generator consists of three sub-generators: one for the patients, one for the physicians and one for the hospital.

8.11.2.1 Patient Generator

The Patient Generator extracts the attributes of a person from a set of web pages. Algorithm 3 details the job of the patient generator.

The base uri (line 1 of Algorithm 3) is extended to include three parameters: the fname, lname and zipcode. We extend the base uri www.yellowpages.com to the following:

http://www.yellowpages.com/findaperson
?fap_terms%5Bfirst%5D=" fname "
&fap_terms%5Blast%5D=" Lname "
&fap_terms%5Bstate%5D=" State "
&page=1"
Algorithm 3 \textit{findPersons()}

1: baseUri $\leftarrow$ yellowpages.com; 
2: uri $\leftarrow$ baseUri + name + zip; 
3: Link[] $\leftarrow$ Spider(uri); 
4: \textbf{for all} $r \in RS$ \textbf{do} 
5: \hspace{1em} Contents $\leftarrow$ Extract(Link[i]); 
6: \hspace{1em} Person $\leftarrow$ Parse(Contents); 
7: \hspace{1em} AddToDatabase(Person); 
8: \textbf{end for}

Figure 8.6 shows the result when \textit{fname} = John, \textit{lname} = Smith and \textit{State} = CA. We then extract the address and telephone number from the result page.

![Figure 8.6. Partial Result page (obtained from yellowpage.com)](image_url)

Figure 8.7 shows the result of executing Algorithm 3 when the base uri is www.whitepages.com and the parameters are \textit{fname} = John, \textit{lname} = Smith and \textit{State} = CA. We then extract the address and age from the result page.

Figure 8.8 is a list of attributes we collect for each patient in our provenance knowledge base.
Figure 8.7. Partial Result page (obtained from whitepage.com)

Figure 8.8. Patient Attributes
8.11.2.2 Physician Generator

The Physician Generator extracts information as attribute values for a doctor. We modify the line 1 of Algorithm 3 by replacing the value of the base Uri to base rateMd.com. Figures 8.9- 8.11 display the user interface for searching a doctor and the results respectively.

![Search page](image)

Figure 8.9. Search page (obtained from ratemd.com)

Figure 8.12 is a list of attributes we extracted for each physician in our provenance knowledge base.

8.11.2.3 Hospital Generator

We also generate hospital information from yellowpages.com website. Figure 8.13 shows a result returned from searching for a hospital.
Figure 8.10. Single Result page (obtained from ratemd.com)

Figure 8.11. Multi-Result page (obtained from ratemd.com)
Figure 8.12. Physician Attributes

Figure 8.13. Hospital Result page (obtained from yellowpages.com)
8.11.3 Miscellaneous Generators

This module uses http://www.webMD.com to determine the relationships between a disease, a medication, a test and a treatment. Therefore, this allows us to add semantic associations among these entities to our knowledge base. Since the relationships are background information that are also available to any user, we build rules that take these semantic relationships into consideration when disclosing provenance information.

8.11.4 Workflow Generator

We build a set of standard workflows, which are taken from the procedures described at http://www.mghp.com/services/procedure/. Since these procedures are freely available, we build rules to protect some sensitive components in the generated workflows. Furthermore, the relationships among the entities in these workflows can be explicit or implicit. Therefore, our system applies a mixture of policies and inference rules to protect the information in these workflows.

Algorithm 4 \texttt{generateworkflow()}

1: \texttt{patient $\leftarrow$ getPatient(); }
2: \texttt{graph $\leftarrow$ generateOpmGraph(); }
3: \texttt{annotateGraph(graph); }

Annotating the workflow

We annotate our workflow using the data produced by the background generator. Therefore, the association between the attributes of a patient are the ones gathered from http://www.yellowpages.com and http://www.whitepages.com. Similarly, the hospital and physician attributes are in fact the ones gathered from http://www.yellowpages.com and http://www.ratemd.com.
8.11.4.1 Generating Workflows

For each patient in our toy hospital example, we initiate workflows that update the record for the patient. The recorded provenance is the only confidential data we assumed in our system. The intent is to give the querying user an opportunity to guess the patient’s disease, medications or tests associated with the record. Provenance poses more challenges than that of traditional data. The controller not only anticipates inferences involving a user’s prior knowledge, but also considers the inferences associated with the causal relationships among the data items as well as the provenance entities.

Properties of the Workflow

We observe a few properties of the workflows we generated:

- Our provenance workflow are generated using the OPM toolbox*. This toolbox captures the skeleton of a workflow generated by using the predicates in $V_G^P$, where $V_G^P = \{ WasControlledBy, Used, WasDerivedFrom, WasGeneratedBy, WasTriggeredBy \}$.

That is, the initial workflows we generate are typically not annotated with RDF triples that are related to the entities in our workflow; for example, triples which make assertion about an agent name, address or age. Therefore, we avoid clutter, which makes it easier to visualize the medical procedures.

- Each entity in our workflow graph, $G$, can be annotated by RDF triples, which makes assertions about the entities. Our workflow is typically stored in its full form. That is, we add annotations to each workflow by transforming it into one that has relevant background information corresponding to the entities in the workflow.

- Each entity in $G$ has attributes which were derived from either the yellow pages(\url{http://www.yellowpages.com}) or the white pages (\url{http://www.whitepages.com/}) website. These attributes are the ones that are a part of the user background knowledge. We also add other fictitious attributes to the entities in our provenance graph. These

*Available at \url{http://openprovenance.org/}
fictitious attributes allow us to scale the size and complexity of a provenance graph so that we will have some experimental data about the scalability our system.

- The workflow graph \( G \) contains the private information, i.e., the provenance of the activities performed on a patient’s record. This is the information our inference controller is protecting. The primary methods we employ for protecting this information are provided by the reasoning services available in Semantic Web reasoners and policies that operate over \( G \).
In this thesis, we presented a security framework for provenance. We identified provenance as important as the data it describes; therefore, provenance requires the same attention that is given to protecting traditional data items. Provenance, however, has a directed structure, which makes it different from traditional data. Furthermore, since security policies like access control and redaction were mainly develop to protect single data items, they do not extend to protecting provenance. We explored many areas where we could improve current security mechanisms to support provenance. Some of these areas were addressed by this thesis:

• The first part of our framework demonstrated the implementation of a scalable role-based access control application for provenance, with particular reference to the healthcare domain. This implementation addressed the problems related to achieving scalable and efficient access control mechanisms for protecting both traditional data and the provenance that describes it. We demonstrated the success of this implementation by building a scalable prototype on the Semantic Web. We also demonstrated that our prototype can overcome the limitations of performing reasoning with very large ABoxes. For this, we provided a solution that partitions the ABoxes. We limit the scope to a healthcare domain, which has hospital systems that are large, distributed and heterogeneous. Most importantly, such an hospital generates provenance that describes the activities on a patient record. We advocated that by retrieving only the relevant assertions, our reasoning task can be reduced from being global to being local and still correctly and efficiently determine the policy-decisions with respect to a patient.

• We then explored the possibility of defining an access control policy language that
extends existing access control policy to support provenance. Our solution uses 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 then incorporate an implementation of this extended policy language into our framework. We again used Semantic Web technologies (RDF, SPARQL) in order to evaluate the effectiveness of this policy language in the cases where are protecting both the single data items and their relationships in a provenance graph.

- Next, we explored other policies for protecting provenance. These policies were developed to share provenance information rather than specify restrictions on the provenance graph. To support these policies, we proposed a graph rewriting approach for redacting a provenance graph. We also used a simple utility-based strategy to preserve as much of the provenance information as possible when the policies are applied to a provenance graph. This strategy ensures a high quality in the information shared. We also extend our framework with an implementation of this graph rewriting approach, which was also successfully evaluated in a Semantic Web environment.

- We then take both the fact that we may want to protect provenance using policies and also support the ongoing mutual relationships among entities that rely on sharing quality information. We discussed challenges in providing high quality data to a user, while protecting the private information in a provenance knowledge base. The challenges we identified were primarily due to background information available to the user and inferencing techniques employed by the user. For this solution, we provided a simple risk analysis model to this problem. Most importantly, we incorporate provenance as part of the answer set released to a user, so that the user can have confidence in the information stored in our knowledge base. This model uses state of the art Semantic Web reasoners to draw inferences in order to determine what can be learnt from releasing both provenance and traditional data in response to a user’s query.
Finally, we integrate all the individual solutions into our unified framework. The framework was designed using a modular approach, whereby any of the above solutions can be implemented separately or altogether. This approach allows a domain user a choice of policies for both protecting and sharing provenance information. For example, the user may be comfortable with using only one set of policies: The user may want to hide provenance using access control mechanisms or hide sensitive information using redaction policies or hide provenance after taking known inferences over provenance into account.

In summary, our work extends previous policy definitions to support provenance. We demonstrated the success of the different parts of our framework by leveraging over a closely knit set of open technologies (RDF, SPARQL, OPM). We plan to pursue this avenue of research further with the emphasis on optimization and policy conflict resolution in the presence of large provenance graphs and large policy sets. We continue to explore many directions for future research. The applications we have been considering are in the areas of healthcare. We plan to apply our framework to other applications, especially in the area of e-science, and intelligence. In addition, we are currently improving our unified framework with new functionalities to address some of the open issues with our graph rewriting system. One of the new functions takes as input a valid OPM graph, a production rule and a set of embedding instructions and return a valid OPM graph. A new direction we are investigating is the optimization of our framework, which uses regular expressions for the queries that enforces our policies. Our goal is to use the notion that if two automata accept the same language, then one of the languages may be redundant in our framework. This is based on an algorithmic translation of a finite regular expression over the RDF representation of a provenance graph to a finite state machine. Therefore, this direction will allow us to derive an optimized and compact set of policies. We can also compare policies for overlaps as well as identify conflicts and suitable resolutions.
REFERENCES


[31] D. Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and

international World Wide Web conference on Alternate track papers & posters, pages

[33] L. Chang and I. Moskowitz. A bayesian network schema for lessening database in-
ference. In Proceedings of CIMCA-01, International Conference on Computational

[34] H. Chen, Z. Wu, H. Wang, and Y. Mao. RDF/RDFS-based relational database integ-
ration. 2006.

of Security Properties on the Quality of Information in Tactical Military Net-
works.

[36] L. Cirio, I. Cruz, and R. Tamassia. A role and attribute based access control system
using semantic web technologies. In Proceedings of the 2007 OTM Confederated in-
ternational conference on On the move to meaningful internet systems-Volume Part

[37] B. Corcoran, N. Swamy, and M. Hicks. Combining provenance and security policies in
a web-based document management system. In On-line Proceedings of the Workshop


276, 2008.

XACML for access control in distributed systems. In Proceedings of the 2003 ACM

[91] D. Marks. Inference in mls database systems. Knowledge and Data Engineering, IEEE

[92] A. Masood, A. Ghafoor, and A. Mathur. Scalable and effective test generation for
access control systems that employ RBAC policies. Purdue University.


S. Miles, P. Missier, J. Myers, et al. The open provenance model core specification


VITA

Tyrone Cadenhead received his Master of Arts degree (Computer Science) from Queens College in 2006. He has also received his Master of Science degree (Management Information Systems) in 1999 and Bachelor of Science degree (Computer Science) in 1995 from Univ. of West Indies, Jamaica. His research interests include Web information processing, Semantic Web inferencing and data mining. He then joined The University of Texas at Dallas, where he completed his PhD. Tyrone also worked in the industry between the years of completing his Bachelor and starting his PhD.