Role-Based Access Control in Collaborative Research Environments

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Chowdary Davuluri

Graduate Program in Computer Science and Engineering

The Ohio State University

2010

Master's Examination Committee:

Dr. Rajiv Ramnath, Advisor

Dr. Michael Freitas

Dr. Jay Ramanathan
Abstract

Providing information security and privacy assurance through Role-Based Access Control is not a trivial job in today’s increasingly connected systems. Dynamic environments make access control policies difficult to implement and evolve. This thesis focuses on the security and privacy provisions in a restricted organizational environment through access control mechanism. To address some of the current limitations of traditional access control models, we propose a knowledge-based architecture for Role-Based Access Control for project oriented environments based on Semantic Web Technologies. The solution includes representation of the semantics of the organization and its access control mechanism as an ontology exploiting the Web Ontology Language. Access to the resources is controlled by defining differential access rights formulated based on the roles of the individuals, and the departments, projects and groups they belong to. The expressivity of the knowledge base is further enhanced by executing semantic web rules using Pellet OWL-DL reasoner over the designed ontology. The ontology is designed to cope with organization restructuring with minimal efforts. To validate the proposed solution, the component architecture and functionality of its simple yet powerful implementation, MetaDB is described.
Dedication

To my family
I would like to express my sincere gratitude to my advisor Dr. Rajiv Ramnath for all the guidance and support. His immense knowledge and critical thinking was vital in guiding my research, especially when I ran into technical challenges. I would like to thank Dr. Michael Freitas for having me on this interesting project that was the key inspiration for my work. I would also like to thank Dr. Jay Ramanathan for helping me shape up my research and providing me the motivation I needed throughout the process.
Vita

June 2007 ........................................... Bachelor of Technology,

IIIT Allahabad, India

July 2007 – August 2008 ......................... Software Engineer, Aricent, India

September 2009 to present ..................... Graduate Research Assistant,

Comprehensive Cancer Center,

The Ohio State University

Fields of Study

Major Field: Computer Science and Engineering
Table of Contents

Abstract ................................................................................................................................. ii

Dedication ............................................................................................................................. iii

Acknowledgements .............................................................................................................. iv

Vita ........................................................................................................................................ v

Table of Contents ................................................................................................................ vi

List of Tables ........................................................................................................................ ix

List of Figures ........................................................................................................................ x

Chapter 1: Introduction ........................................................................................................ 1

1.1 Role-Based Access Control ............................................................................................ 1

1.2 Why Semantic Web Technologies for Access Control? ............................................... 2

1.3 Access Control in Dynamic Environments ................................................................. 2

1.4 Outline of Thesis .......................................................................................................... 3

Chapter 2: Problem Statement ............................................................................................ 4

2.1 Typical Project Hierarchy ............................................................................................. 4

2.2 Access Scenarios in Project Oriented Environments .................................................. 5
Chapter 3: Solution Approach

3.1 Semantic Web Technologies

3.1.1 Resource Description Framework

3.1.2 Using Ontology

3.1.3 Introduction to the Web Ontology Language

3.1.4 Description Logics for OWL

Chapter 4: MetaDB Ontology

4.1 Defining Concepts through Classes

4.2 Realizing Concepts through Instances

4.3 Defining Relations through Properties

Chapter 5: Access Control by Enhancing the Expressivity of OWL

5.1 Introduction to the Semantic Web Rule Language

5.2 Pellet OWL-DL Reasoner

5.3 Enhancing the Knowledge Base

Chapter 6: Implementation: MetaDB

6.1 Component Diagram

6.2 Technology Architecture

6.2.1 OWL API

6.2.2 OWLDB
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.3 Object Model</td>
<td>35</td>
</tr>
<tr>
<td>6.2.4 Database Schema</td>
<td>36</td>
</tr>
<tr>
<td>6.3 Data Management and Access Control Using MetaDB – A Real Example</td>
<td>38</td>
</tr>
<tr>
<td>Chapter 7: Evaluation</td>
<td>49</td>
</tr>
<tr>
<td>7.1 Flexibility</td>
<td>49</td>
</tr>
<tr>
<td>7.2 Performance</td>
<td>51</td>
</tr>
<tr>
<td>7.3 User Study</td>
<td>52</td>
</tr>
<tr>
<td>Chapter 8: Related Work</td>
<td>56</td>
</tr>
<tr>
<td>Chapter 9: Conclusion and Future Work</td>
<td>59</td>
</tr>
<tr>
<td>9.1 Ontology Related Tasks</td>
<td>60</td>
</tr>
<tr>
<td>9.1.1 Incremental Reasoning</td>
<td>60</td>
</tr>
<tr>
<td>9.1.2 Storing Ontology in Relational Databases</td>
<td>61</td>
</tr>
<tr>
<td>9.2 User Study</td>
<td>61</td>
</tr>
<tr>
<td>9.3 Extending the framework</td>
<td>61</td>
</tr>
<tr>
<td>References</td>
<td>63</td>
</tr>
<tr>
<td>Appendix A: MetaDB’s Access Control Rules</td>
<td>69</td>
</tr>
<tr>
<td>Appendix B: MetaDB’s Ontology Manager Code</td>
<td>71</td>
</tr>
</tbody>
</table>
List of Tables

Table 1: Roles and privileges to access corresponding resources........................................7

Table 2: User Questionnaire - Statements ...........................................................................53
List of Figures

Figure 1: Typical Project Hierarchy ................................................................. 5
Figure 2: The Semantic Web Stack ................................................................. 9
Figure 3: MetaDB Ontology ................................................................. 15
Figure 4: MetaDB – Component Diagram .......................................................... 26
Figure 5: MetaDB – Implemented Components .................................................. 27
Figure 6: Dynamic Architecture Flow ............................................................. 29
Figure 7: MetaDB – Technology Architecture .................................................. 31
Figure 8: Sequence Diagram – Add Role ......................................................... 33
Figure 9: MetaDB – Object Model ................................................................. 36
Figure 10: MetaDB – Database Schema .......................................................... 37
Figure 11: User Login/Registration ................................................................. 39
Figure 12: Creating/Deleting Departments, Adding/Deleting User Roles ............... 40
Figure 13: Adding/Deleting Rules ................................................................. 42
Figure 14: Data Hierarchy and Create Tag ....................................................... 43
Figure 15: Multiple Views ................................................................. 44
Figure 16: Roles Panel ................................................................. 45
Figure 17: Shared Files ................................................................. 47
Figure 18: Permissions Dialog .................................................................48
Figure 19: Modified Project Hierarchy......................................................50
Information security and privacy assurance is not a trivial job in today’s increasingly connected but dynamically evolving organizations. This thesis focuses on the security and privacy provisions in research organizations which work in project oriented environments. Before describing the details of a knowledge-based architecture for Role-based Access Control (RBAC) [1] for project oriented environments, it is necessary to motivate the discussion with an introduction to RBAC in general. This introduction contains a definition of RBAC, the challenges involved in the design and maintenance of access control constraints for dynamic project oriented environments and describes how the limitations of traditional RBAC models can be overcome by using semantic web technologies [2].

1.1 Role-Based Access Control

Role-based Access Control (RBAC) is a means for controlling access to resources based on the roles that individual users have within an organization. RBAC models have generated great interest in the security community as a powerful and generalized approach to security management [3]. In RBAC, users are assigned to roles and roles are associated with permissions. The permission determines what operations a user assigned to a role can perform on information resources. Instead of specifying all the accesses (read, write, etc.) each individual user is allowed, access authorizations on objects are
1.2 Why Semantic Web Technologies for Access Control?

Traditional RBAC models lack granularity in the levels of restrictions and require effort to modify [4]. These limitations can be overcome by using Semantic Web Technologies [5]. The Semantic Web is an extension of the current web in which information is given well-defined meaning, enabling computers and people to work better in cooperation. By using ontologies to model the organizational structure and its access control mechanisms, semantics promises information consistency and information processing. Adding semantics can facilitate machine interpretability and automated processing of these access control information and mechanisms. The W3C is designing the Web Ontology Language (OWL) [6], which is a semantic markup language that provides formalized knowledge expression and shares a great deal of common semantics about expressing access control constraints.

1.3 Access Control in Dynamic Environments

Design and maintenance of access control constraints in organizations are challenging problems as the organizational structure, roles, user pools, security requirements are always changing. Conceptual organizational semantics and its access control mechanisms are formally represented in an ontology using Web Ontology Language (OWL). The system controls access to the resources of the organization by providing differential access privileges. In this ontology, these privileges are formulated
based on the roles of the individuals. In addition, it considers which departments, projects or groups they belong to. The proposed solution is designed to cope with the dynamic nature of the organization.

1.4 Outline of Thesis

The rest of this thesis is organized as follows. The next chapter discusses the problem statement and a motivating example. Chapter Three briefly describes the solution approach using Semantic Web Technologies. The MetaDB ontology for representing organizational semantics is described in Chapter Four. Chapter Five illustrates the access control mechanism, the processes and results based on the proposed ontology. Chapter Six gives a detailed description of the system architecture and a real example of how the system could be used for data management and access control. In chapter Seven, the proposed solution is evaluated in the context of organizational restructuring. Chapter eight contains overview of the related work and the thesis concludes briefly stating the future work.
Chapter 2: Problem Statement

Nowadays people in research organizations increasingly work in project-oriented environments. Project-oriented environments can be characterized as those in which project members devote most of their time to projects that may not be part of their functional department. The members are less specialized and have broader tasks, skills and responsibilities. Some of the projects deal with sensitive information which should not be leaked to unauthorized members. The project members come from different departments. They don’t enjoy the same rights or privileges within a project environment. This situation is more prevalent while accessing resources owned by the departments of projects. Furthermore, there are situations where a person holds multiple roles and privileges.

2.1 Typical Project Hierarchy

Figure 1 illustrates a specific organizational environment. It deals with the roles Department Head (DeptHead), Principal Investigator (PI), Group Leader (GL) and Technician (Tech). CCC (Department) contains two projects - Proteomics and Genomics. Proteomics (Project) in turn contains groups MetaDB and Pegasus. A department will be headed by a Department Head, a project by a Principal Investigator and a group by a Group Leader respectively. A group may contain multiple technicians. A researcher’s resource contains the set of files owned by him. A person may have multiple roles like
David who is not only a PI (Project – Genomics) but also a GL (Group – MetaDB). He should have access to the corresponding project and group resources in which he is involved. So, access rights depend on one’s role in the respective departments, projects and groups.

Figure 1: Typical Project Hierarchy

2.2 Access Scenarios in Project Oriented Environments

Following are some of the examples of restricted access scenarios,
1. DeptHead is the head of a Department. He can read the files owned by PIs of all the projects owned by his department. He can add/delete projects to his department. He can also add/delete PIs to the projects his department owns.

2. PI is the head of a project. He can read the files of GLs of all the groups owned by his project. He can add/delete groups to his project. He can also add/delete GLs to the groups his project owns.

3. GL is the head of a group. He can read and write the files of all the Technicians of his group. He can add/delete Technicians to his group.

4. A Technician can read the files of all the others Technicians in his group.

Therefore, the architecture should manage access to resources not only based on the roles but also based on the involvement in organizational divisions (departments, projects and groups). Access scenarios of the example project hierarchy are described in Table 1.
<table>
<thead>
<tr>
<th>Researcher</th>
<th>Role</th>
<th>WorkUnit</th>
<th>Privilege</th>
<th>Access to Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>PI</td>
<td>Proteomics</td>
<td>Read Write</td>
<td>David/ Andrew/ Eric/ Josef</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Add Delete</td>
<td>Groups to Proteomics/ GLs to MetaDB &amp; Pegasus</td>
</tr>
<tr>
<td>David</td>
<td>PI</td>
<td>Genomics</td>
<td>Add Delete</td>
<td>Groups to Genomics</td>
</tr>
<tr>
<td>GL</td>
<td>MetaDB</td>
<td></td>
<td>Read Write</td>
<td>Andrew/ Josef</td>
</tr>
<tr>
<td>Add Delete</td>
<td></td>
<td></td>
<td>Add Delete</td>
<td>Techs to MetaDB</td>
</tr>
<tr>
<td>Andrew</td>
<td>GL</td>
<td>Pegasus</td>
<td>Read Write</td>
<td>Eric</td>
</tr>
<tr>
<td>Add Delete</td>
<td></td>
<td></td>
<td>Add Delete</td>
<td>Techs to Pegasus</td>
</tr>
<tr>
<td>Josef</td>
<td>Tech</td>
<td>MetaDB</td>
<td>Read</td>
<td>Josef</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Andrew</td>
</tr>
</tbody>
</table>

Table 1: Roles and privileges to access corresponding resources
Chapter 3: Solution Approach

We advocate that access control solutions should adopt semantic technologies as the key building blocks for supporting expressive representation of organizational semantics [7].

Our access control mechanism adopts an Ontological Approach - the semantics of the organization and its access control mechanism are represented as an ontology exploiting the Web Ontology Language. Access to the resources is controlled by defining differential access rights based on the roles of the individuals and the department, projects and groups they belong to. Additional facts which aren’t explicitly defined are inferred by executing semantic rules over the designed ontology.

3.1 Semantic Web Technologies

Semantic Web is a term coined by the World Wide Web Consortium (W3C) director Tim Berners-Lee. It describes methods and technologies to allow machines to understand the meaning – or ‘semantics’ – of information on the World Wide Web [23].

While the term “Semantic Web” is not formally defined, it is mainly used to describe the model and technologies proposed by the W3C. These technologies include the Resource Description Framework (RDF) [26], a variety of data interchange formats (e.g. RDF/XML, N3, Turtle, N-Triples [28]), and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal
description of concepts, terms, and relationships within a given knowledge domain.

Figure 2 shows how these standards and tools are organized in the Semantic Web Stack [2].
- XML provides an elemental syntax for content structure within documents, yet associates no semantics with the meaning of the content contained within [29].
- XML Schema is a language for providing and restricting the structure and content of elements contained within XML documents.
- RDF is a simple language for expressing data models, which refer to objects ("resources") and their relationships. An RDF-based model can be expressed in XML.
- RDF Schema extends RDF and is a vocabulary for describing properties and classes of RDF-based resources, with semantics for generalized hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes [17].
- SPARQL is a protocol and query language for semantic web data sources [24].

Current ongoing standardizations include:
- Rule Interchange Format (RIF) as the Rule Layer of the Semantic Web Stack

Not yet fully realized layers include:
- Unifying Logic and Proof layers are undergoing active research.
3.1.1 Resource Description Framework

The Resource Description Framework (RDF) is a family of W3C specifications originally designed as a metadata description model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources using a variety of syntax formats.

The RDF data model is similar to classic conceptual modeling approaches such as Entity-Relationship or Class diagrams, as is based upon the idea of making statements about the resources (in particular Web resources) in the form of subject-predicate-object expressions. These expressions are known as triples in RDF terminology. The subject denotes the resource, and the predicate denotes the traits of aspects of the resources and expresses a relationship between the subject and the object. For example, one way to represent the notion of “The sky has the color blue” in RDF is as the triple: a subject denoting “the sky”, a predicate denoting “has the color”, and an object denoting “blue”.

RDF is an abstract model with several serialization formats (i.e., file formats), and so the particular way in which a resource or triple is encoded varies from format to format. RDF’s simple data model and ability to model disparate, abstract concepts has also led to its increasing use in knowledge management applications unrelated to Semantic Web activity.

A collection of RDF statements intrinsically represents a labeled, directed multigraph. As such, an RDF-based data model is more naturally suited to certain kinds of knowledge representation than the relational model and other ontological models. However, in practice, RDF data is often persisted in relational database or native
representations also called Triple stores, or Quad stores if context (i.e. the named graph) is also persisted for each RDF triple. Additional ontology languages can also be built upon RDF.

3.1.2 Using Ontology

Ontologies [8] are the cornerstone technology of the Semantic Web, providing structured vocabularies that describe a formal specification of a shared conceptualization. It is used to capture the knowledge about a domain of interest in the form of concepts and their relationships. It permits the description of organizational structures, roles, privileges and resources at different levels of abstraction and supports reasoning about both the structure and the properties of the elements that constitute the system. We believe that designing a consistent ontology based on a sound conceptual foundation is worthwhile because it can be reused in various organizations to control access to their resources.

3.1.3 Introduction to the Web Ontology Language

Among the different ontology languages, we are focusing on the Web Ontology Language (OWL) [17] suggested by the W3C. It is a markup language that builds on RDF and RDF Schema. OWL allows us to describe a domain in terms of:

- Individuals – Particular objects in our domain
- Classes – Collections of objects (usually sharing some common characteristics)
- Properties – Binary relationships between individuals, cardinality, equality, richer typing of properties, etc
• A collection of axioms describing how these classes, individuals, properties etc. should be interpreted

OWL is chosen because it provides more vocabularies for describing concepts and properties than that supported by XML, RDF and RDFS.

The W3C-endorsed OWL specification includes the definitions of three different variants of OWL, with different levels of expressiveness:

• OWL Full is the union of OWL syntax and RDF
• OWL DL is restricted to FOL fragment (~DAML+OIL)
• OWL Lite is “simpler” subset of OWL DL

Each of these sublanguages is a syntactic extension of its simpler predecessor.

3.1.4 Description Logics for OWL

Of the three different sub-languages OWL offers, we decided to use OWL DL. It is based on Description Logics (hence the suffix DL). These logics are the decidable part of First Order Logic and are therefore amenable to automated reasoning. Though OWL DL lacks in expressivity compared with OWL Full, it maintains decidability and regains computational efficiency. The computational efficiency is an important feature since the chosen language is expected to support scores of relations. Since the access control mechanism is supposed to evaluate and grant permissions to access resources, it is necessary to add reasoning support to it. In order to achieve this, we use Semantic Web Rule Language (Chapter 5) which is designed as an extension of OWL DL.
Chapter 4: MetaDB Ontology

MetaDB ontology models the organizational structures described in Chapter two.

4.1 Defining Concepts through Classes

Figure 3 illustrates the proposed MetaDB ontology in conceptual level. The design of the ontology has greatly been influenced by the work of Mohammad M.R. Chowdhury, Javier Chamizo, Josef Noll and Juan Miguel in [7]. OWL classes are the concrete representations of concepts. In the proposed ontology, Researcher class defines researchers of the organization. We specify Department, Project and Group as subclasses of WorkUnit in order to avoid defining explicit relationships between department/project/group and roles. Role is the positional hierarchy of researchers in the organization. Through this, individuals are restricted in access to the correct resources. The class hierarchy is a critical issue in inheritance of properties. The classes Department Head, Principal Investigator, Group Leader and Technician are a subset of the Role class. The File class defines the files owned by a researcher. The Tag class defines user folksonomies.
4.2 Realizing Concepts through Instances

OWL classes are interpreted as sets that contain instances or individuals. Figure 3 illustrates the instances of classes in ellipses. Instances of the researchers are defined here simply as their names. Instances to the subclasses of Role are added in accordance with the individual roles to realize multiple roles of a person. As for example, David_PI instance of Principal Investigator corresponds to the principal investigator instance of David. Similarly, David_GL corresponds to the group leader role of David.

![Figure 3: MetaDB Ontology](image-url)
4.3 Defining Relations through Properties

Properties are the binary relations between the two things, more specifically between the instances of classes. A property relates instances from the *domain* with the instances from the *range*. Syntactically, a *domain* links a property to a class and *range* links a property to either a class or a data range. Due to the class hierarchy and *domain* and *range* specifications, subclasses inherit the relationships between the respective classes. Figure 3 also provides the properties and their relationships with classes.

*rolePlaysIn* property specifies the fact that in which specific *Work Unit* (department/project/group) a *Role* instance plays its role. The MetaDB ontology is supposed to answer what kinds of permissions a researcher has on a resource and *hasReadPermission* and *hasWritePermission* properties define this situation. The relationships of *hasReadPermission* and *hasWritePermission* are not defined explicitly. These relationships of the properties are filled in through the inferencing process.
Chapter 5: Access Control by Enhancing the Expressivity of OWL

Access control is achieved by enhancing the expressivity of OWL through the inference process. This chapter describes the access control logic and expressivity needs, reviews the language used and its application to the goals of the system.

5.1 Introduction to the Semantic Web Rule Language

The expressivity provided by the OWL is limited to tree like structures [9]. This means that knowledge cannot be inferred from indirect relations between the entities. However, the solution spends most part of its power in inferring indirect relationships that will determine whether a subject has access to a resource or not and which are its privileges over it. The hierarchical structures defined in Chapter Four and the inherent relationships between working units and the hierarchies of resources are a perfect field where inference can extract these knowledge. We do inferencing using rules described over the ontology, using Semantic Web Rule Language (SWRL) [20] as a complementary feature of OWL. The rules can be used to infer new knowledge from existing OWL knowledge bases.

SWRL [27] is based on a combination of the OWL DL and OWL Lite sublanguages of the OWL and the Unary/Binary Dialog sublanguages of the Rule Markup Language [30]. SWRL allows users to write Horn-like rules expressed in terms of OWL concepts to reason about OWL individuals.
The SWRL specification does not impose restrictions on how reasoning should be performed with SWRL rules. Thus, investigators are free to use a variety of rule engines to reason with the SWRL rules stored in an OWL knowledge base. They are also free to implement their own editing facilities to create SWRL rules.

In common with many other rule languages, SWRL rules are written as antecedent-consequent pairs. In SWRL terminology, the antecedent is referred to as the rule body and the consequent is referred to as the head. The head and body each consist of a conjunction of one or more atoms. At present, SWRL does not support more complex logical combinations of atoms like having both conjunctions and disjunctions.

SWRL rules reason about OWL individuals, primarily in terms of OWL classes and properties. For example, a SWRL rule expressing that a person with a male sibling has a brother would require capturing the concepts of ‘person’, ‘male’, ‘sibling’ and ‘brother’ in OWL. Intuitively, the concept of person and male can be captured using an OWL class called Person with a subclass Man; the sibling and brother relationships can be expressed using OWL properties hasSibling and hasBrother, which are attached to a Person. The rule in SWRL would then be:

\[
\text{Person} (\text{?x1}) \land \text{hasSibling} (\text{?x1}, \text{?x2}) \land \text{Man} (\text{?x2}) \Rightarrow \text{hasBrother} (\text{?x1}, \text{?x2})
\]

Executing this rule would have the effect of setting the hasBrother property to x2 in the individual that satisfies the rule, named x1.

SWRL rules can also refer explicitly to OWL individuals. For example, the following example is a variant of the above rule, inferring that a particular individual Fred has a brother:
Person (Fred) ^ hasSibling (Fred, ?x2) ^ Man (?x2) → hasBrother (Fred, ?x2)

In this case Fred is the name of an OWL individual.

SWRL also supports data literals. For example, assuming an individual has a hasAge property, it is possible to ask if Fred had a 40 year-old brother:

Person (Fred) ^ hasSibling (Fred, ?x2) ^ Man (?x2) ^ hasAge (?x2, 40) → has40YearOldBrother (Fred, ?x2)

String literals – which are enclosed in single quotes – are also supported.

SWRL also supports the common same-as and different-from concepts. For example, the SWRL sameAs atom can determine if two OWL individuals Fred and Frederick are the same individual:

sameAs (Fred, Frederick)

Similarly, the differentFrom atom can be used to express that two OWL individuals are not the same.

SWRL also has an atom to determine if an individual, property, or variable is of a particular type. For example, the following example determines if variable x is of type unsigned integer:

Xsd:unsignedInt (?x)

These atoms – which are called data range atoms in SWRL – must be preceded by the ‘xsd:’ namespace qualifier. The type specified must be an XML Schema data type.

A second form of a data range atom can be used to express one-of relationships in SWRL. For example, the following SWRL atom indicates that variable x must be one of 3, 4 or 5:

Xsd:integer (?x)

Xsd:or (3, 4, 5) (?x)
SWRL also supports a range of built-in predicates, which greatly expand its expressive power. SWRL built-ins are predicates that accept several arguments. They are described in detail in the SWRL Built-in Specification. The simplest built-ins are comparison operations. For example, the greaterThan built-in determines if an individual has an older brother:

$$hasBrother(?x1, ?x2) \land hasAge(?x1, ?age1) \land hasAge(?x2, ?age2) \land swrlb:greaterThan(?age2, ?age1) \rightarrow hasOlderBrother(?x1, ?x2)$$

All built-ins in SWRL must be preceded by the namespace qualifier ‘swrlb:’.

Finally, SWRL supports more complex mathematics built-ins. For example, following rule determines if an individual has a brother who is exactly 10 years older:

$$hasBrother(?x1, ?x2) \land hasAge(?x1, ?age1) \land hasAge(?x2, ?age2) \land swrlb:subtract(10, ?age2, ?age1) \rightarrow hasDecadeOlderBrother(?x1, ?x2)$$

The SWRL Built-in Ontology describes the range of built-ins supported by SWRL. In addition to mathematical built-ins, there are built-ins for strings, dates, and lists. Additions may be made to this namespace in the future so the range of built-ins supported by SWRL can grow.

5.2 Pellet OWL-DL Reasoner

We used Pellet reasoner to run the rules. Pellet [19] is a complete OWL-DL reasoner with acceptable to very good performance, extensive middleware, and a number of unique features. It is the first sound and complete OWL-DL reasoner with extensive support for reasoning with individuals (including nominal support and conjunctive
query), user-defined data types, and debugging support for ontologies. It implements several extensions to OWL-DL including a combination formalism for OWL-DL ontologies, a non-monotonic operator, and preliminary support for OWL/Rule hybrid reasoning. Pellet is written in Java and is open source. The following are Pellet’s primary features and capabilities [32]:

- **Conjunctive ABox Query**

  Pellet includes a query engine that can efficiently answer conjunctive ABox queries expressed in SPARQL or RDQL [31]. In the presence of non-distinguished variables in the query, the “rolling-up” technique is used to answer tree-shaped queries. Otherwise, every query atom can be answered in isolation and arbitrary shaped queries can be handled. For such queries, the two factors affecting the query answering time are the number of atoms in the query and the order these atoms are evaluated. Pellet has two optimization techniques to deal with these cases: Query Simplification, finding redundant atoms in the query by using domain/range axioms, and Query Reordering, sorting the query atoms by utilizing a randomized sampling technique as adopted in relational databases.

- **Data type Reasoning**

  Pellet uses the type system approach to support reasoning with datatypes. In particular, Pellet has a datatype oracle that can reason with XML Schema based datatypes. The datatype oracle can check the consistency of conjunctions of (built-in or derived) XML Schema datatypes. Pellet supports user derived types
based on numeric or date/time types so, for example, numeric or date/time intervals can be defined and used as new datatypes.

- **Axiom Pinpointing and Debugging**
  
  Axiom pinpointing is a nonstandard DL inference service that provides a justification for any arbitrary entailment derived by a reasoner from an OWL-DL knowledge base. Given an ontology and any of its logical consequences, the axiom pinpointing service determines the premises in the Knowledge Base (KB) that are sufficient for the entailment to hold. The justification is useful for understanding the output of the reasoner, which is key for many tasks, such as ontology debugging, design and evolution.

- **Integration with Rules Formalism**
- **Multi-Ontology Reasoning using E-Connections**
- **Non-monotonic Reasoning**

5.3 Enhancing the Knowledge Base

Objects of the properties `hasReadPermission` and `hasWritePermission` are filled in through the inferred knowledge from executing the rules. In order to do this, MetaDB ontology and SWRL rules are transferred to Pellet. Running the reasoner then initiates the inference process, generates knowledge. This inferred knowledge can be used by the external interface or can optionally be passed back to the ontology to enrich it. Rules are formulated using the SWRL as follows,

- Rule1: A group leader has write permission over the files owned by all the technicians in his group
File(?f), Group(?G),

Researcher(?p), Researcher(?t),

GroupLeader(?p_GL), Technician(?t_Technician),

hasRole(?p, ?p_GL), hasRole(?t, ?t_Technician),

isFileOwnedBy(?f, ?t),

rolePlaysIn(?p_GL, ?G), rolePlaysIn(?t_Technician, ?G) ->

hasWritePermission(?p, ?f)

➢ Rule2: A technician has read permission over the files owned by all the other technicians in his group

File(?f), Group(?G),

Researcher(?p), Researcher(?t),

Technician(?t_Technician), Technician(?p_Technician),

hasRole(?p, ?p_Technician), hasRole(?t, ?t_Technician),

isFileOwnedBy(?f, ?t),

rolePlaysIn(?p_Technician, ?G), rolePlaysIn(?t_Technician, ?G) ->

hasReadPermission(?p, ?f)

All the relationships in the ontology have not been explicitly defined. As for example, hasWritePermission relationship of group leader David (David_GL) has not been explicitly defined. Executing Rule 1 infers those relationships, which answer to which resources David_GL has write access. From Figure 1, executing Rule 1 infers that researcher David, as group leader of MetaDB, has write permission to the resources
owned by researchers Andrew and Josef. The inferred results are exported back to the ontology to fill these empty relationships. Execution of Rule 2 shows the similar result. Researcher Andrew, as a technician in group MetaDB, would have gained read access to the resources owned by researcher Josef.
Chapter 6: Implementation: MetaDB

This chapter discusses the underlying architecture of MetaDB and then presents a real example of how the software is used for data management and access control.

6.1 Component Diagram

Figure 4 shows the complete component diagram for MetaDB. The components that are functional at this stage are depicted in Figure 5. Figure 4 included all the services and components that MetaDB would use to fulfill the need of a data management and access control solution. Let’s look at its components one by one.

A simple and powerful API is at the heart of MetaDB. We call it the MetaDB Engine. This core functionality unit provides APIs for managing meta-data, fetching consolidated logical data sets, updating information objects etc. This component is domain independent, very generic and can be used to manage almost any kind of data, even your songs collection. The core component would interact with different data sources using standard access methods like web services or a ReSTful interface.
The whole framework including the core component is built on extensible Java EE framework and provides hooks for adding on domain specific functionality to it in the form of plug-ins. For example, we could have a proteomics plug-in built to handle proteomics metadata. The domain specific plug-ins would typically also have another logically related part for the UI. The primary purpose of plug-ins is to create a user specific logical view of data. A songs management plug-in might interpret “artist” metadata differently at the back-end and could also present it a specific format of the UI.
Access Control ontology with access rules and Domain-specific ontological mappings with integration rules are specified in owl files and are interpreted by the Ontology Extraction Component, which is a part of the middle-tier. This component, would

- Periodically extract information from different information objects present in separate data sources, consolidate that information based on the domain mappings and store it in the RDBMS as meta-data for frequent, faster centralized access (This functionality hasn’t yet been implemented in the system).
Access and update the access control ontology which is later used by the Business Rules Engine and Access Control Components to provide Role-Based Access Control in the system.

The Business Rules Engine Component constitutes an OWL Reasoner and/or a Rule Engine to execute

- Integration rules over the Domain-Specific ontological mappings to consolidate the information extracted from multiple data sources, and
- Access rules over the Access Control Ontology to compute permissions based on the roles played by the users.

Pellet, an Open Source OWL 2 Reasoner, is currently being used to execute rules which are written using Semantic Web Rule Language (SWRL), over the Access Control Ontology.

The Access Control component constitutes:

- A component independent from all other business logic components which would let any client that wants to use MetaDB services including its own UI have access to it first by having to authenticate and authorize themselves.
- A dependent component which together with the Business Rules Engine and Ontology Extraction Components would provide Role-Based Access Control in the system exploiting Semantic Web Technologies.
The Query Service provides an advanced query mechanism to the user for executing complex queries. Query interface would have a Structured Query Language (SQL) like syntax with the ability to use domain concepts in forming the queries. Figure 6 shows the dynamic architectural diagram depicting how a typical query would be executed.

Figure 6: Dynamic Architecture Flow

Workflow Service or Business Rules Engine Service would be used to interface to an external workflow engine or a business rules engine. Caching service would enable object and information caching at the local level to enable faster access to data.
The user-interface is a web-based rich-client UI that is very intuitive. Simplicity has been preferred over providing every possible feature on the main window. All related features have been placed on their respective windows.

6.2 Technology Architecture

Figure 7 shows the Technology Architecture for MetaDB. Java EE has been chosen over other technologies to enable platform independence and extensibility. The server-side component architecture is built upon the EJB 3.0 specification with JBoss as the application server to provide a component based extensible architecture. The client application uses Swing, a platform-independent Model-View-Controller GUI framework for Java to provide a rich client web based user interface. The thick Swing client and the server-side Java objects communicate using Java RMI, which uses Java Naming and Directory Interface (JNDI), a Java API for a directory service that allows Java Software clients to discover and lookup data and objects in a distributed computing environment via a name. MySQL, an open source relational database management system (RDBMS) is used as the database.
The MetaDB ontology shown in Figure 3 is a conceptual representation of collaborative project-oriented research environments in the scientific domain. The ontology is built using Protégé-OWL [21], a free, open-source editor to build ontologies for the Semantic Web, in particular in the W3C’s Web Ontology Language (OWL). An Ontology Developer who also acts as the administrator will be responsible for building and maintaining the ontology. The MetaDB ontology is responsible for carrying out a wide variety of tasks like

- Providing role-based access control
- Restricting the users to a tag hierarchy etc.

Figure 7: MetaDB – Technology Architecture
Objects of the properties, \textit{hasReadPermission} and \textit{hasWritePermission} in the MetaDB ontology are filled in through the inferred knowledge from executing the rules. We used Pellet reasoner, an open-source OWL DL Reasoner to run the rules. First, MetaDB ontology, which also contains the access rules defined in SWRL, is transferred to Pellet. Running the reasoner then initiates the inference process, generates inferred knowledge which is passed back to the ontology to enrich it.

The Ontology Manager class on the server-side is responsible for carrying out the tasks of manipulating and executing the rules over the MetaDB ontology. OWL API, a high level Application Programming Interface (API) for working with the OWL 2 ontologies is used to access/update the ontology and Pellet OWL Reasoner API is used to do the inferencing. The ontology is currently being efficiently represented in-memory using the data structures provided by the OWL API reference implementation. The OWL API has also been designed so that it is possible to provide other storage mechanisms for ontologies. An exemplar solution called OWLDB, stores ontologies in a relational database, using a “native” mapping of OWL constructs (as opposed to a mapping to triples) to a custom database schema. Figure 8 shows the sequence diagram depicting how the various components would interact when a new role is added by any user through the client.
The OWL API [18] is an open source Java API and reference implementation for creating, manipulating and serializing OWL ontologies. The latest version of the API is focused towards OWL 2. It includes the following components:

- An API for OWL 2 and an efficient in-memory reference implementation
- RDF/XML parser and writer
- OWL/XML parser and writer
- OWL Functional Syntax parser and writer
- Turtle parser and writer
• KRSS parser
• OBO Flat file format parser
• Reasoner interfaces for working with reasoners such as FaCT++, HermiT, Pellet and Racer

The OWL API is primarily targeted at OWL-DL, which does not mean that it cannot handle OWL-Full ontologies, but a number of design decisions reflect this assumption.

The current implementation has the following features:

• Modeling – Provide data structures that represent OWL ontologies/documents.
• Parsing – Taking some syntactic presentation, e.g. OWL-RDF and converting it to some [useful] internal data structure.
• Serializing – Producing a syntactic presentation, e.g. OWL-XML from a local data structure.
• Manipulation/Change – Being able to manipulate the underlying objects.
• Inference – Providing a representation that implements/understands the formal semantics of the language.

6.2.2 OWLDB

OWLDB (alias Mnemosyne) [22] is a storage system based on object-relational mappings utilizing the OWL-API for the W3C Web Ontology Language OWL. The following are its primary features:

• Native OWL persistence
• Object relational mapping
Based on the OWL-API

DB independent

6.2.3 Object Model

Figure 9 shows the object model for MetaDB. The Tag object is used to represent user folksonomies. The Permission object is used to store the permission a User has on a particular InformationObject. The Department, Project and Group objects inherit the WorkUnit object and are used to store the work units in an organization. The UserRole object contains the multiple roles users play.
6.2.4 Database Schema

Following is the database schema of MetaDB. The colored entities represent the tables that are involved in providing role-based access control in the system. 

ObjectTagRelationship models the many-to-many relationship between InformationObject and Tag, whereas UserTagRelationship models the many-to-many relationship between User and Tag.
A close look at the schema identifies its resemblance to the MetaDB ontology. The MetaDB ontology only stores contents of the database which lets it uniquely identify each and every entity defined in the system to carry out the tasks which will require inferencing using rules. This replication of information between the ontology and the schema as opposed to having the MetaDB ontology store all the information in the system is due to the following reasons:
Querying ontologies is expensive and hence frequent querying would drastically reduce the performance of the application when working with huge data.

The current state of the art in DL reasoners does not try to handle updates incrementally [19], and hence efficient incremental classification and incremental consistency checking cannot be achieved in changing knowledge bases.

6.3 Data Management and Access Control Using MetaDB – A Real Example

Let us now see how we can use MetaDB to manage and share data in a collaborative research environment.

Figure 1 illustrates the use case scenario which describes a specific organizational environment. It deals with the roles like Department Head, Principal Investigator, Group Leader and Technician and work units like Department, Project and Group. A department contains multiple projects and a project in turn may contain multiple groups. Departments, projects and groups will be headed by department heads, principal investigators and group leaders respectively. A group contains multiple technicians. A person has multiple roles like David is not only a PI (Project – Genomics) but also a GL (Group – MetaDB). He should have access to corresponding department and project resources where he involves in. The architecture manages access to resources not only based on the roles but also based on the involvement in organizational divisions (departments, projects and groups).

Let us now look at how the following restricted access scenario is implemented.

“A Department Head has read permission to the files of Principal Investigators of all the projects under the department he heads”.

38
The above access scenario lets user Mike, Department Head of CCC, gain read access to the files of user Tom, Principal Investigator of Proteomics project.

All the users of MetaDB are expected to register with the application by clicking the Register button on the Login Panel shown below.

![User Login/Registration](image)

Figure 11: User Login/Registration

An administrator account with username, password credentials as “admin” and “admin” respectively is created at application start up. An administrator, who might also be the Ontology Developer, is responsible for creating Departments, assigning user roles and adding access rules to the ontology. Figure 12 shows the panel an administrator uses
to create/delete Departments and add/delete user roles. The bottom left component displays the existing departments in the organization. Clicking a department displays its corresponding hierarchy on the adjacent right component. Figure 12 displays the hierarchy of department CCC. As you can see, user Mike plays the role of Department Head in CCC.

Figure 12: Creating/Deleting Departments, Adding/Deleting User Roles

Access rules can be added dynamically by the administrator through the Rules Panel shown in Figure 13. Inferencing over the existing knowledge base is performed when we add/delete

- User Roles
Rules

Information Objects (Files)

The following access rule (in SWRL) has to be added to let Department Heads gain read permission to files of Principal Investigators of projects part of their respective departments.

“Department(?D) , Project(?P) , File(?f) ,
isProjectOf(?P, ?D) ,
Researcher(?p) , Researcher(?t) ,
DepartmentHead(?p_DeptHead) , PrincipalInvestigator(?t_PI) ,
hasRole(?p, ?p_DeptHead) , hasRole(?t, ?t_PI) ,
isFileOwnedBy(?f, ?t) ,
rolePlaysIn(?p_DeptHead, ?D) , rolePlaysIn(?t_PI, ?P)
-> hasReadPermission(?p, ?f)”
Figure 13: Adding/Deleting Rules

Assuming user Tom is currently logged in; let us now see how MetaDB can be used to manage data from Tom’s perspective.
Data files are embedded in many levels of directories. Different types of files e.g. raw data files, analysis result files and result summary files are scattered all over the place and deep inside the folders. To solve this problem with MetaDB, start by simply dragging and dropping the top-level project folder on MetaDB. MetaDB copies all these files to the data source, stores the parent and child relationships and present a tree-like structure in the UI. But this tree structure is very different from what we have on the hard disk. We can now rearrange the tree structure, add tags to it, without worrying about the actual physical file structure. As shown in the picture above, a tag is a simple text which describes the tagged object. A user can either use a tag created earlier in the system or can create a new tag. All the tags that are created are shown in the left panel as well as in
the Apply Tag combo box in the tool bar at the top. Now, to view all the files tagged with “ExpData”, all we need to do is click on ExpData under tags on the left panel. Files are displayed linearly in a tabular format irrespective of how deeply embedded they are in the package hierarchy.

Figure 15: Multiple Views

We can also select multiple tags on the left panel in order to display all the files that have been tagged with either of the selected tags. There are no limits on how to tag files or folders. We can also tag the objects on the basis of the file type (e.g. raw data, analysis results etc), the project structure (e.g. module 1, sub module a etc), the status of
the project or file (e.g. pending, done, urgent etc) or based on any other combination of
the above. In this way, users can easily switch the perspective of the data based on what
they are doing or what they want to see. This dynamicity is the real power of this
approach.

Let’s now look at how Role-Based Access Control is achieved in MetaDB.
Clicking “Roles” on the left panel displays the Roles panel as shown below for the user
currently logged in.

![Figure 16: Roles Panel](image-url)

45
The bottom left component displays all the roles the user logged in are currently playing. The user has to select the Role and the Work Unit and click the Refresh button on the “Select Role” component to access permissions acquired by playing that particular role. The “Add/Remove WorkUnits” and “Add/Remove Roles” components will be populated with appropriate values as according to the role selected. The client-side business logic is intelligent enough not to let a user playing the role of a Technician add Groups or add Technicians. Let’s assume that User Mike created a project called Genomics under CCC department and then assigned user Tom the role of Principal Investigator to it using the interface shown above.

All the shared files a user can currently access can be viewed by clicking “Shared Files” on the left panel.
Figure 17: Shared Files

The Shared Files Panel displays all the files to which the user currently logged in is identified to have access as a result of executing the rules over the MetaDB ontology. It also displays the files to which the user has manually been assigned access by right-clicking them as shown below.
Figure 18: Permissions Dialog

Figure 17 displays the shared files from user Mike’s perspective. Clicking a file displays the actual owner of the file and the permissions the user has on the right panel. In Figure 17, the value for “Owner” displays Tom as user Mike has gained read permission to all of Tom’s files as computed by the access rule added before.
Chapter 7: Evaluation

The proposed solution is more maintainable since there exists a general schema for any organization that can easily be adapted, thanks to its expressivity capacity nearer to human understanding. Expressivity and inference capabilities avoid the inclusion of redundant information in the ontology.

7.1 Flexibility

The proposed MetaDB ontology can reflect the organization changes of a research environment with minimum efforts. Now, we are going to describe an example of this. A new group, Grp-A has been created in the project hierarchy shown in Figure 1. The group is added to the Genomics project as shown in Figure 19. Researchers Adam and Josef have joined as the group leader and technician of the group respectively.

It is expected that a group leader (Adam) has read permission over the files owned by the technicians (Josef) in his group. As a principal investigator, researcher David gains read and write permissions to the files owned by research Adam, group leader of group Grp-A, under project Genomics. The corresponding actions to reflect these changes into the ontology are described in the following points. Through this, we are going to evaluate the strength of the proposed MetaDB ontology.

- Add new researcher instance: Adam
- Add new group instance: *Grp-A*
- Add new instance of *Group Leader* role: *Adam_GL*
- Fill up all the corresponding relationships.
- If we execute Rule 1 described in section 5.3, new relationships are inferred with *
  hasWritePermission*
  property. These are exported back to the ontology to fill in the empty relationships. As expected, researcher David gains read access to the files owned by researcher Adam.

![Modified Project Hierarchy](image)

Figure 19: Modified Project Hierarchy

Only a few changes are required to the existing ontology. All the additions are in the instances, which is quite apparent. Conceptually, the proposed MetaDB ontology can
follow organizational changes. One of the ways of simplifying access rights management is to store the access control matrix using access control list (ACL). Though ACLs are simple to implement, management steps are quite tedious especially when required to revoke somebody’s privilege. The proposed ontology provides a simple mechanism to revoke somebody’s role or privilege. One can simply delete the relationship between the role instance and the corresponding permission instance to withdraw the privilege and afterward can delete the corresponding role instance to repeal the subject’s role entirely.

Among the few disadvantages, this approach is computationally expensive and decidability is not guaranteed by SWRL. In addition, XML processing is slower than that of database. But ontologies can be stored in the relational databases or even be mapped from them.

7.2 Performance

Though ontologies can be stored in relational databases, querying can still be computationally expensive and frequent querying would drastically reduce the performance of the system when working with huge data. Also, the current state of the art in DL reasoners does not try to handle updates to the ontology incrementally at run-time.

The proposed architecture overcomes these limitations by letting the MetaDB ontology store contents of the database which lets it uniquely identify each and every entity defined in the system to carry out the tasks that will require inferencing using rules. This replication of information between the MetaDB ontology and the relational database as opposed to having the ontology store all the information in the system results in very minimal redundancy and helps improve the performance by many folds.
The architecture has also been designed to be partly ontology driven for the above stated performance reasons.

7.3 User Study

To evaluate the proposed MetaDB architecture and its role as a data management and access control solution, we need to consider multiple perspectives. First, we are interested in the usefulness of the software so that we can identify any changes required in the existing functionality and further extend the system. Secondly, we are interested in how easy is the software to use. To assess the MetaDB system based on these factors, we deployed it in a research laboratory setting at the Biomedical Research Center and let a sample of prototypical end users use it.

A total of five users, part of a research project, have been asked to use the system after a brief introduction to it. The software is delivered as an executable jar file and the users have been provided with a User Guide which outlines the software’s important features and the steps required to use them. To understand our users better, they have been asked to fill out a questionnaire which contains statements on the various aspects of usefulness and ease of use of the software. The users were asked to select the level of agreement that applies (where 1 means strongly disagree, 3 means neither agree nor disagree, and 5 means strongly agree) to each of the statements. The following table shows the results of the questionnaire.
<table>
<thead>
<tr>
<th>Statement</th>
<th>Average Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>It helps me be more effective</td>
<td>3.25</td>
</tr>
<tr>
<td>It is useful</td>
<td>3.25</td>
</tr>
<tr>
<td>I can use it successfully every time</td>
<td>4.25</td>
</tr>
<tr>
<td>It saves me time when I use it</td>
<td>3</td>
</tr>
<tr>
<td>It meets my needs</td>
<td>3</td>
</tr>
<tr>
<td>It does everything I would expect it to do</td>
<td>4</td>
</tr>
<tr>
<td>It is user-friendly and simple to use</td>
<td>4</td>
</tr>
<tr>
<td>I can use it without written instructions</td>
<td>4.5</td>
</tr>
<tr>
<td>I don’t notice any inconsistencies as I use it</td>
<td>3.5</td>
</tr>
<tr>
<td>I am satisfied with it</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 2: User Questionnaire - Statements

Statements 1, 2, 4, 5 and 6 are meant to assess the usefulness of the software, whereas statements 3, 7, 8 and 9 are meant to assess the ease of use. At the end of the questionnaire, the users were asked to share their interactions using the software. The intention is to try and get a sense of what users believe the most important issues in the system are and also to correlate the questionnaire responses with the corresponding statements made by them.

Most of the users felt that the software would be more useful to their respective supervisors than they do. They supported their statement by saying that they now know
where all their data is and how it is organized. This perspective of the users has also been the reason behind a low rating for all the statements meant to assess the usefulness of the software. However, we strongly disagree with their reports for the following reasons. Firstly, files of individual users brought into the application are not currently being transferred over to the server. So, users were asked to just use their data on the shared drive to test the software. Secondly, all the users participated in the study were part of multiple projects with no clearly defined roles. So, we believe that they could not actually realize the strength of the access control solution used beneath. If the study had been conducted on at least 10-15 users coming from multiple projects/groups with clearly defined roles, and if individual user files are being transferred over to the server, we believe that we would have received much positive feedback.

One user reported that the executable file did not work for him. A slight drop in the ratings of statements 3, 8 and 9 reflects this. The reason for this behavior of the executable is that the initial java heap size settings on that particular user’s machine were configured to slightly lower values than that required by the software. Executing from the command line setting the required heap size worked though. Hence, in the future, it is better to wrap the jar file as a windows executable with all these settings configured.

One of the users reported that he is happy with his file organization system and could gain nothing out of using this application. We believe that the lack of user’s ability to look beyond MetaDB as just a data organizing solution is one of the reasons for this statement. Also, individual files not being transferred over to the server, and having spent a limited time working with the tagging/un-tagging features of the application might also
be the other reasons behind his statement. The same user also reported that using the application consumed him more time rather than saving. He suggested that the entire user panel should have been receptive to dropping of files rather than just the “Root” node which occupies a very small space of the user files panel. However, all the other users reported that they are happy with how the current User Interface (UI) handles this issue.

Though the statements meant to assess the usefulness of the software have been rated low, for the above stated reasons; the statements meant to assess the ease of use of the software have received much positive feedback. Almost all the users reported that the UI is very intuitive and they can use it without any written instructions.

To summarize, we felt that the user questionnaire and recording the user interactions has helped a great deal in assessing the usefulness and ease of use of the MetaDB system. It has also helped us identify the changes existing functionality might require and our immediate tasks. We believe the statements meant to assess the usefulness of the software would have received much positive feedback if the individual user files are being transferred over to the server and if the study had been conducted on atleast 10-15 users coming from multiple projects/groups with clearly defined roles.
Chapter 8: Related Work

Information security and privacy assurance are more evident in a business environment. These issues are handled in this paper through access control mechanisms in the context of project oriented collaborative research environments.

Among the access control technologies, Access Control List (ACL) is widely used. The semantics of ACLs have been proven to be insecure in many situations. Instead of maintaining a centralized ACL, a trust based group has been proposed to delegate access rights [10], [11] where FOAF (friend of a friend) based social networks acted as a mean for delegation. A private key based signature scheme was proposed to ensure the privacy of networks. But such a scheme requires secure distribution and maintenance of keys. A similar concept of trust and reputation has also been used by [12] to create and access communities. A distributed trust management approach is also considered as one of the main components to secure the Semantic Web [13]. They intended to provide access to community resources and privacy solutions only by means of trust/reputation management. But access to sensitive business resources based on a trust mechanism is simply not suited to business contexts, since in business contexts, things are more deterministic. In addition, trust is affected by various factors and therefore difficult to quantify.
FOAF [25] uses RDF technology. In this thesis, we use OWL which facilitates greater machine interoperability of the Web content than that supported by XML, RDF and RDF Schema (RDF-S) by providing an additional vocabulary along with a formal semantics. OWL also maintains decidability and computational efficiency.

We considered the concept of Role Based Access Control (RBAC) as part of our access control mechanism. Sandhu introduced RBAC in [1] with early multi-user computer systems, where users’ access permissions are associated with roles, and users are made members of appropriate roles. A major advantage of RBAC is its ability to constraint access based on the concept of separation of duties which significantly simplifies the management of permissions. In the proposed solution, access control also depends on how the project hierarchy is structured.

There are two types of RBAC constraints: dynamic and static. Authors in [14] described an approach for RBAC with dynamic constraints using automated reasoning techniques. Though we focused on static constraints on roles, rules were included within the ontology to infer new knowledge which can be passed back to the ontology. Through this, verification of access control constraints defined in the ontology are also achieved. The proposed solution can also adapt to changing organizational structure with less effort.

In [15], authors presented an approach to reduce the inefficiencies of the management (coordination, verification and validation, and enforcement) of many role-based access control policies and mechanisms using OWL. They focused on the representation of XACML (eXtensible Access Control Markup Language) policies in
DL. In [16], the authors also suggested expressing access control policies based on OWL and SWRL. The solution was limited to the definition of OWL ontology and declaration of SWRL rules. They predicted the use of an engine to deduce more information by adding rules. The proposed solution actually uses an OWL reasoner called Pellet to execute rules and deduce more information. Finini in his paper [13] also proposed using OWL for constructing ontologies which defines policies/privileges.
Chapter 9: Conclusion and Future Work

In this thesis, we addressed the security and privacy challenges of project-based environments through access control mechanism exploiting Semantic Web technologies. In this regard, we developed the MetaDB ontology to represent the conceptual structure of a project oriented collaborative research environment and the roles of individuals. In the ontology, all the relationships between the entities have not been defined explicitly. Semantic rules facilitated expressing this additional knowledge. Pellet OWL-DL reasoner executed these rules and new facts were transferred back to the ontology to enrich it and check its validity.

To validate the solution, we described the component architecture and functionality of its simple yet powerful implementation, MetaDB. Apart from these, we evaluated the inference capabilities of the proposed solution by restructuring the project hierarchy and described in detail the performance issues.

After presenting our solution and building the initial implementation, we can conclude that the use of semantic web technologies is a powerful approach for providing RBAC in project oriented environments. In the near future, we are planning to extend the MetaDB framework in several different directions.
9.1 Ontology Related Tasks

The proposed MetaDB ontology is currently being used just for providing access control. But it could be leveraged to perform a variety of tasks like limiting the researchers to a tag hierarchy, specifying static and dynamic RBAC constraints, querying for permissible actions, driving the user interface (UI) etc. Also, the MetaDB ontology together with the Domain-specific ontological mappings are to be interpreted periodically by the Ontology Extraction Component to extract information from different information objects present in separate data sources, consolidate that information based on the integration rules and store it in the RDBMS as meta-data for frequent and faster centralized access.

9.1.1 Incremental Reasoning

The MetaDB system involves repeated changes in its OWL ontology knowledge bases in a relatively short period of time. Hence, it is critical for the reasoner to support incremental reasoning, i.e. recompute as little as possible after each update. The current state of the art in DL reasoners does not try to handle updates incrementally. Though Pellet, the OWL-DL reasoner being used in MetaDB, contains support for incremental classification and incremental consistency checking, it still doesn’t support incremental realization. Integrating this feature with MetaDB, whenever available, will increase the performance of the system by many folds, particularly when reasoning over large knowledge bases.
9.1.2 Storing Ontology in Relational Databases

Though we have successfully tested a solution called OWLDB, which stores ontologies in a relational database, using a “native” mapping of OWL constructs to a custom database schema, we couldn’t actually integrate it with the MetaDB system. The MetaDB ontology is currently being represented in-memory using the data structures provided by the OWL API. For many purposes, this is sufficient. However, storing the ontology in a relational database or a triple store will be efficient for huge knowledge bases as information loss can be prevented when the system goes down.

9.2 User Study

Use of semantic web technologies is increasing for web and application domain and its full potential is yet to be explored. This thesis is just a small but significant step in this direction. The initial user study has helped us identify the changes existing functionality might require and our immediate tasks. As a future work, we need to validate the system as well as the framework in an iterative manner as we add in new features. We would also like to expand the user study to assess various other factors that might help us to measure the quality of the system.

9.3 Extending the framework

We also need to implement various other components such as caching and query service, data integration service etc. We should also need to extend and configure the system for various IT infrastructures and platforms. We strongly feel that the right way to do this is to do the core functionalities very generic and extensible and then extending the
features by developing appropriate plug-ins for a given environment and needs. Also, development should be done in an iterative way and user feedback should be taken into account after each iteration.

We plan to make the project source code open once the project gains a certain amount of maturity. That would enable the scientific communities to make their own extensions to the software by adding plug-ins.
References


http://www.w3.org/TR/owl-features/


http://owlapi.sourceforge.net/


http://clarkparsia.com/pellet/


http://www.w3.org/Submission/SWRL/


http://protege.stanford.edu/

[22] OWLDB.

http://owldb.sourceforge.net/

[23] World Wide Web

http://www.w3.org/WWW/
[24] SPARQL Query Language for RDF
http://www.w3.org/TR/rdf-sparql-query/

[25] The Friend of a Friend (FOAF) project
http://www.foaf-project.org/

[26] Resource Description Framework (RDF)
http://www.w3.org/RDF/

[27] Martin O’Connor, Holger Knublauch, Samson Tu, Benjamin Grosof, Mike Dean, William Grosso, Mark Musen, “Supporting Rule System Interoperability on the Semantic Web with SWRL”.

[28] N-Triples
http://www.w3.org/2001/sw/RDFCore/ntriples/

[29] Extensible Markup Language
http://www.w3.org/XML/

[30] Rule Markup Language (RuleML)
http://ruleml.org/
[31] RDQL – A Query Language for RDF

http://www.w3.org/Submission/RDQL/

Appendix A: MetaDB’s Access Control Rules

This appendix contains the SWRL access control rules currently defined in MetaDB.

- **Rule1**: A principal investigator has read permission over the files owned by the technicians of all the groups in his project.

  \[
  \text{File}(?f), \text{Group}(?G), \text{Project}(?P), \text{Researcher}(?p), \text{Researcher}(?t), \\
  \text{PrincipalInvestigator}(?p\_PI), \text{Technician}(?t\_Technician), \\
  \text{hasRole}(?p, ?p\_PI), \text{hasRole}(?t, ?t\_Technician), \\
  \text{isFileOwnedBy}(?f, ?t), \\
  \text{isGroupOf}(?G, ?P), \\
  \text{rolePlaysIn}(?p\_PI, ?P), \text{rolePlaysIn}(?t\_Technician, ?G) \\
  \rightarrow \text{hasReadPermission}(?p, ?f) 
  \]

- **Rule2**: A principal investigator has write permission over the files owned by the technicians of all the groups in his project.

  \[
  \text{File}(?f), \text{Group}(?G), \text{Project}(?P), \text{Researcher}(?p), \text{Researcher}(?t), \\
  \text{PrincipalInvestigator}(?p\_PI), \text{Technician}(?t\_Technician), \\
  \text{hasRole}(?p, ?p\_PI), \text{hasRole}(?t, ?t\_Technician), \\
  \text{isFileOwnedBy}(?f, ?t), \\
  \text{isGroupOf}(?G, ?P), \\
  \text{rolePlaysIn}(?p\_PI, ?P), \text{rolePlaysIn}(?t\_Technician, ?G) \\
  \rightarrow \text{hasWritePermission}(?p, ?f) 
  \]

- **Rule3**: A technician has read permission over the files owned by all the other technicians in his group.

  \[
  \text{File}(?f), \text{Group}(?G), \text{Researcher}(?p), \text{Researcher}(?t), \\
  \rightarrow \text{hasReadPermission}(?p, ?f) 
  \]
Rule 4: A department head has read permission over the files owned by the principal investigators of all the projects in his department.

Rule 5: A department head has read permission over the files owned by the group leaders of all the groups in his department.

Rule 6: A group leader has read permission over the files owned by all the technicians under his group.
Appendix B: MetaDB’s Ontology Manager Code

This appendix contains the code for all the MetaDB’s ontology related tasks namely,

- Setting up the Ontology Manager
- Manipulating the ontology using OWL API
- Executing SWRL rules over the ontology using Pellet Reasoner API

```java
public class OntologyDBManager {
    private OWLOntologyManager manager;
    private OWLOntology ontology;
    private OWLDataFactory factory;
    private Reasoner reasoner;
    private OWLEntityRemover remover;
    private URL url;
    private URI uri;

    private static OntologyDBManager ontologyDBManager = null;

    protected OntologyDBManager() {
    }

    public static OntologyDBManager getInstance() {
        if (ontologyDBManager == null) {
            ontologyDBManager = new OntologyDBManager();
        }
        return ontologyDBManager;
    }

    /**
     * Method to set up the ontology manager. Called at system start up.
     *
     */
    public void setupDBManager() {
        try {
            manager = OWLManager.createOWLOntologyManager();
            url = getClass().getResource(OntologyConstants.ONTOLOGY_URI);
            uri = new URI(url.toString().replace(" ", ";%20"));

            // Other setup code...
        } catch (Exception e) {
            // Handle exception...
        }
    }
}
```
ontology=manager.loadOntologyFromPhysicalURI(uri);
    factory=manager.getOWLDataFactory();

    reasoner=new Reasoner(manager);
    reasoner.loadDataFactory(factory);
    reasoner.loadOntology(ontology);
    reasoner.classify();
    System.out.println("ONTOLOGY CONSISTENCY:-
    "+reasoner.isConsistent());

    //displayObjectProperties();
    }catch(OWLOntologyCreationException e){
        System.out.println("Error loading the ontology");
        e.printStackTrace();
    }catch(URISyntaxException e){
        System.out.println("Error converting the URL to an URI");
        e.printStackTrace();
    }

    }

    public void displayIndividualsUsingReasoner(){
        System.out.println("--------PRINTING OUT THE INDIVIDUALS USING A REASONER--------");
        Set<OWLClass> classes=reasoner.getClasses();
        for(OWLClass concept:classes){
            System.out.println("---CLASS:---"+concept);
            Set<OWLIndividual> individuals=reasoner.getIndividuals(concept, true);
            for(OWLIndividual ind:individuals){
                System.out.println(ind.getURI());
            }
        }
    }

    public void displayIndividualsUsingOWLAPI(){
        System.out.println("--------PRINTING OUT THE INDIVIDUALS USING OWL API--------");
        Set<OWLClass> classes=ontology.getReferencedClasses();
        for(OWLClass concept:classes){
            System.out.println("---CLASS:---"+concept);
            Set<OWLIndividual> individuals=concept.getIndividuals(ontology);
            for(OWLIndividual ind:individuals){
                System.out.println(ind);
            }
        }
    }

    /**
     * Method to add an instance to the ontology
     * @param individualName Name of the instance to be added
     * @param className Name of the concept to which the instance has to be added
     */
public void addIndividual(String individualName, String className) {
    System.out.println("Adding INDIVIDUAL - " + INDIVIDUAL:" + individualName +" CLASS:" + className);
    OWLClass concept;
    OWLIndividual individual;
    OWLAxiom classAxiom;
    AddAxiom addClassAxiomChange;
    URI indURI;
    URI classURI;

    try{
        indURI=URI.create(OntologyConstants.BASE+individualName.replace(" ","%20");
        // System.out.println("----INDIVIDUAL URI:-" +indURI);
        individual=factory.getOWLIndividual(indURI);
        classURI=URI.create(OntologyConstants.BASE+className);
        concept=factory.getOWLClass(classURI);
        classAxiom=factory.getOWLClassAssertionAxiom(individual, concept);
        addClassAxiomChange=new AddAxiom(ontology, classAxiom);
        manager.applyChange(addClassAxiomChange);
        System.out.println("ADDED INDIVIDUAL:" + individualName + " TO:" + className);
    }catch(OWLOntologyChangeException e){
        System.out.println("Exception while adding individual of : "+className);
    }
}

/**
 * Method to delete an instance from the ontology
 * @param individualName Name of the instance to be added
 * @param className Name of the concept from which the instance has to be removed
 */
public void removeIndividual(String individualName, String className) {
    System.out.println("Removing INDIVIDUAL - " + INDIVIDUAL:" + individualName +" CLASS:" + className);
    OWLIndividual individual;
    URI indURI;
    try{
        remover=new OWLEntityRemover(manager, Collections.singleton(ontology));

        remov
indURI=URI.create(OntologyConstants.BASE+individualName.replace(" ","%20"));
    individual=factory.getOWLIndividual(indURI);
    individual.accept(remover);
    manager.applyChanges(remover.getChanges());
    System.out.println("REMOVED INDIVIDUAL:"+individualName+" FROM:"+className);
}catch(OWLOntologyChangeException e){
    System.out.println("Exception while removing individual of :"+className);
}

/**
 * Method to add an object property to the ontology. An object property is a binary relationship which maps individuals of two concepts.
 * @param subjectName Name of the subject instance in the property
 * @param objectProperty Name of the object property
 * @param objectName Name of the object instance in the property
 */
public void addRelationShip(String subjectName,OntologyObjectPropertyEnum objectProperty,String objectName){
    System.out.println("Adding RELATIONSHIP - SUBJECT:"+subjectName+" PROPERTY:"+objectProperty+" OBJECT:"+objectName);
    OWLIndividual subject;
    OWLObjectProperty property;
    OWLIndividual object;
    OWLObjectPropertyAssertionAxiom assertion;
    AddAxiom addAxiomChange;
    URI subURI;
    URI propURI;
    URI objURI;
    try{
        subURI=URI.create(OntologyConstants.BASE+subjectName.replace(" ","%20"));
        System.out.println("-----SUBJECT URI:-"+subURI);
        subject=factory.getOWLIndividual(subURI);

        propURI=URI.create(OntologyConstants.BASE+objectProperty);
        property=factory.getOWLObjectProperty(propURI);

        objURI=URI.create(OntologyConstants.BASE+objectName.replace(" ","%20"));
        object=factory.getOWLIndividual(objURI);
assertion=factory.getOWLObjectPropertyAssertionAxiom(subject, property, object);
    addAxiomChange=new AddAxiom(ontology,assertion);
    manager.applyChange(addAxiomChange);
    System.out.println("ADDED RELATIONSHIP:"+subjectName+"--"+objectProperty+"--"+objectName);
})catch(OWLOntologyChangeException e){
    System.out.println("Exception while adding a relationship : addRelationShip()");
}

/**
 * Method to remove an object property from the ontology. An object property is a binary relationship which maps individuals of two concepts.
 * @param subjectName Name of the subject instance in the property
 * @param objectProperty Name of the object property
 * @param objectName Name of the object instance in the property
 */
public void removeRelationShip(String subjectName,OntologyObjectPropertyEnum objectProperty,String objectName){
    System.out.println("Removing RELATIONSHIP - "+"SUBJECT:"+subjectName+" PROPERTY:"+objectProperty+" OBJECT:"+objectName);
    OWLIndividual subject;
    OWLObjectProperty property;
    OWLIndividual object;
    OWLObjectPropertyAssertionAxiom assertion;
    RemoveAxiom removeAxiomChange;
    URI subURI;
    URI propURI;
    URI objURI;
    try{
        subURI=URI.create(OntologyConstants.BASE+subjectName.replace(" ","%20"));
        subject=factory.getOWLIndividual(subURI);

        propURI=URI.create(OntologyConstants.BASE+objectProperty);
        property=factory.getOWLObjectProperty(propURI);

        objURI=URI.create(OntologyConstants.BASE+objectName.replace(" ","%20"));
        object=factory.getOWLIndividual(objURI);

    }catch(OWLOntologyChangeException e){
        System.out.println("Exception while removing a relationship : removeRelationShip()");
    }
}
assertion=factory.getOWLObjectPropertyAssertionAxiom(subject, property, object);
removeAxiomChange=new RemoveAxiom(ontology, assertion);
manager.applyChange(removeAxiomChange);
System.out.println("ADDED RELATIONSHIP:"+subjectName+"--"+objectProperty+"--"+objectName);
}

catch(OWLOntologyChangeException e){
System.out.println("Exception while adding a relationship : addRelationShip()");
}
/**
 * Method to execute the SWRL rules using Pellet Reasoner API.
 * @return A HashMap containing the researchers and the permissions they have on each individual file.
 */
public Map<String,Map<Integer,Permission>> doClassification() {
    Map<String,Map<Integer,Permission>> permissionMap=new HashMap<String,Map<Integer,Permission>>();
    System.out.println("RE-CLASSIFICATION");
    reasoner.refresh();

    OWLClass researcher=factory.getOWLClass(URI.create(OntologyConstants.BASE+
OntologyClassEnum.RESEARCHER.getOntologyClassName()));
    OWLObjectProperty hasReadProperty=factory.getOWLObjectProperty(URI.create(OntologyConstants.BASE+
OntologyObjectPropertyEnum.HAS_READ_PERMISSION.getOntologyPropertyName()));
    OWLObjectProperty hasWriteProperty=factory.getOWLObjectProperty(URI.create(OntologyConstants.BASE+
OntologyObjectPropertyEnum.HAS_WRITE_PERMISSION.getOntologyPropertyName()));
    Set<OWLIndividual> researchers=reasoner.getIndividuals(researcher,true);
    System.out.println("RESEARCHERS SIZE:"+researchers.size());
    for(OWLIndividual ind:researchers){
        Map<Integer,Permission> objectPermissionMap=new HashMap<Integer, Permission>();
        String researcherURI=ind.getURI().toString();
        String researcherName=researcherURI.substring(researcherURI.lastIndexOf('#')+1).replace("%20", " ");
System.out.println("RESEARCHER:"+researcherName);

Set<OWLIndividual> canReadFiles=reasoner.getRelatedIndividuals(ind,hasReadProperty);
System.out.println("CAN READ-----");
for(OWLIndividual file:canReadFiles)
{
    String fileURI=file.getURI().toString();
    Integer fileId=Integer.parseInt(fileURI.substring(fileURI.lastIndexOf('_')+1));
    objectPermissionMap.put(fileId,
    Permission.getPermissionType(true, false));
}

Set<OWLIndividual> canWriteFiles=reasoner.getRelatedIndividuals(ind,hasWriteProperty);
System.out.println("CAN WRITE-----");
for(OWLIndividual file:canWriteFiles)
{
    String fileURI=file.getURI().toString();
    Integer fileId=Integer.parseInt(fileURI.substring(fileURI.lastIndexOf('_')+1));
    if(objectPermissionMap.containsKey(fileId)){
        objectPermissionMap.put(fileId,
        Permission.getPermissionType(true, true));
    }
    objectPermissionMap.put(fileId,
    Permission.getPermissionType(false, true));
}

// might be not a good approach if there are many
users and files...just break this up
permissionMap.put(researcherName,
objectPermissionMap);
}
return permissionMap;
}