Semantic Web Implementation Scheme
for National Vulnerability Database
(Common Platform Enumeration Data)

Vaibhav Khadilkar, Jyothsna Rachapalli,
Bhavani Thuraisingham
The University of Texas at Dallas
# Contents

Acknowledgements 1

1 Introduction 2

2 Project Objectives 4

3 Semantic Web Technologies 5
   3.1 Semantic Web Technologies - Examples .......................... 5
      3.1.1 Converter .................................................... 5
      3.1.2 RDF Parser and Serializer ................................. 5
      3.1.3 RDF store (or triple store) .............................. 6
      3.1.4 Reasoner ..................................................... 6
      3.1.5 SPARQL ....................................................... 6
      3.1.6 Application Interface ..................................... 6
   3.2 Semantic Web Framework - Jena .................................. 6

4 Semantic Web Application - Architecture 8

5 Semantic Web Application - Strategy 10
   5.1 Use Cases ......................................................... 10
   5.2 Ontology Creation and Validation ................................ 11
   5.3 Ontology Migration .............................................. 15
   5.4 Data Migration ................................................... 17
   5.5 Reasoning ......................................................... 18
   5.6 SPARQL Queries .................................................. 19
   5.7 Application ....................................................... 19
   5.8 Performance Metrics ............................................ 22
   5.9 Cyclic Process .................................................. 22

6 Semantic Web Application - Performance 24
   6.1 Experimental Setup ............................................... 25
   6.2 Data loading statistics using the utility ........................ 25
   6.3 Data loading (with inference) statistics using the utility ... 26
   6.4 Data querying statistics using the utility ........................ 26
   6.5 Data querying (with inference) statistics using the utility ... 27
List of Figures

3.1 Jena framework architecture ........................................ 7
4.1 Semantic Web application architecture ............................... 8
5.1 Class hierarchy diagram .............................................. 13
5.2 NVD CPE Ontology ..................................................... 16
5.3 Relational database to Semantic Web migration strategy .......... 23
C.1 Screenshot - Homepage ................................................. 44
C.2 Screenshot - First five products ...................................... 45
C.3 Screenshot - First five agents ......................................... 46
C.4 Screenshot - Features of a specific product .......................... 47
C.5 Screenshot - First five software products ............................ 48
C.6 Screenshot - Products using shared libraries ......................... 49
C.7 Screenshot - First five products that are applications .............. 50
C.8 Screenshot - First five products that are operating systems ..... 51
C.9 Screenshot - First five hardware products .......................... 52
C.10 Screenshot - First five products that are TCP/IP devices ........ 53
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Sample SPARQL and SQL queries - Q1 to Q5</td>
<td>20</td>
</tr>
<tr>
<td>5.2</td>
<td>Sample SPARQL and SQL queries - Q6 to Q10</td>
<td>21</td>
</tr>
<tr>
<td>5.3</td>
<td>Sample Inference SPARQL queries - Q11 to Q13</td>
<td>22</td>
</tr>
<tr>
<td>6.1</td>
<td>Data loading statistics - Utility</td>
<td>25</td>
</tr>
<tr>
<td>6.2</td>
<td>Data loading (with inference) statistics - Utility</td>
<td>26</td>
</tr>
<tr>
<td>6.3</td>
<td>Data querying statistics - Utility</td>
<td>27</td>
</tr>
<tr>
<td>6.4</td>
<td>Data querying (with inference) statistics - Utility</td>
<td>28</td>
</tr>
<tr>
<td>6.5</td>
<td>Data querying (with inference) statistics - Utility vs Application</td>
<td>28</td>
</tr>
</tbody>
</table>
Abstract

In software systems, information modeling is of prime significance. The attributes of real world objects that are captured and the way they are represented in the software system determines the operations a system can perform and the queries it can answer. Traditional relational database model and semantic model are two popular information modeling techniques and they differ in several ways. A relational model is based on tables and columns whereas a semantic model is based on classes and properties. Due to the inadequate expressivity of a relational data model, it imposes limitations on semantic interoperability apart from certain syntactic interoperability issues. Semantic web technologies are based on an information model that is designed to facilitate easy data sharing and interoperability (syntactic as well as semantic). This report addresses the importance of a semantic model and subsequently illustrates an efficient way to migrate data from a relational model to a semantic model using a real world application based on National Vulnerability Database.
Acknowledgements

We thank the National Institute of Standards and Technology (NIST) for collaborating with us on this project. In particular, we thank Paul Cichonski and Harold Booth for defining the research problem for us and for conducting the agile development plan with us. We are grateful for the opportunity to work on the National Vulnerability Database project which is a critical resource for the security community.

A version of this report was presented at the NIST Fifth Annual IT Security and Automation Conference in Baltimore, MD. on October 28, 2009.
Chapter 1

Introduction

National Vulnerability Database (NVD) is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP) [1]. This data enables automation of vulnerability management, security measurement, and compliance. NVD includes databases of security checklists, security related software flaws, misconfigurations, product names, and impact metrics. The Common Platform Enumeration (CPE) is a specification that describes a structured naming scheme for IT platforms [2]. The CPE specification includes a description of how to construct CPE names from product information. NVD contains a description of product information based on a comprehensive list of all CPE names. This data is based on a relational model.

However, this relational model poses certain limitations on the automation of vulnerability management. A relational model is suitable for applications that depend only on the asserted/explicit facts of the system, for e.g. records in a database table. Some facts although not present explicitly are present implicitly in the system. If an application depends on explicit as well as implicit facts of a system, it cannot rely entirely on a relational model. A semantic model facilitates computation of implicit facts from explicit facts of a system, i.e. inferences can be drawn from assertions in a systematic manner. Therefore a semantic model was chosen for the creation of this application.

In a semantic model information is represented as a set of assertions known as statements. A statement (also known as triple) is made up of three parts namely: subject, predicate and object. A subject of a statement is an entity that the statement describes. A predicate describes a relationship between a subject and an object. The object being the value that the subject takes for that particular predicate. Statements of this type naturally form a directed graph, with subjects and objects of each statement as nodes and predicates as edges. This is the data model used in the Semantic Web [3] and it is formalized in the language called Resource Description Framework (RDF) [4].

This report is organized into seven chapters. Chapter 2 briefly describes the project objectives. In chapter 3 we give an overview of Semantic Web tech-
nologies used in the creation of this Semantic Web application along with an overview of Jena [5, 6] (java framework for developing semantic web applications). Chapter 4 describes the architecture of the Semantic Web application and illustrates how various Semantic Web technologies are integrated. Chapter 5 delineates the strategy followed in the creation of this application while in chapter 6 we address issues concerning the performance of the newly created application. Finally, chapter 7 concludes the report along with scope for future work.
Chapter 2

Project Objectives

The main objective of this project involves creation of an ontology that models the product information contained in NVD. This ontology represents NVD’s viewpoint and is geared towards the use cases supporting statements of applicability between IT concepts and IT products. The consistency and validity of the ontology is checked by running a reasoner on the instances created within the ontology to make sure that the expected triples are inferred.

The next objective involves the identification of different use cases that form the initial requirements for creation of the ontology. These use cases must support statements of applicability between IT concepts and IT products. Some of the use cases are searching for products or vendors, equality among products, backwards compatibility with the CPE specification and so on.

The next objective involves migration of product data from NVD to a triple store. To achieve this objective a view needs to be created in NVD that contains the product data. Further, a utility needs to be constructed to read the data from the relational view and migrate it to a triple store using the ontology created previously. Performance of the utility is measured in terms of time required to perform the migration as well as the time required to perform different types of queries.

The final objective consists of creation of a Semantic Web application using the data migrated to the triple store. The main utility of the application is to search for products and agents using a search box provided on the home page. A different way to use the application is by navigating through the different types of products found in NVD by using the menu bar. A user can click on the different types shown to refine their search which then displays the products that belong to each category.
Chapter 3

Semantic Web Technologies

This chapter focuses on the semantic technologies used in this project as well as the architecture for the Jena framework which is one of the most prominently used Semantic Web frameworks.

3.1 Semantic Web Technologies - Examples

This section presents an overview of each technology followed by an adaptation of the technology for this project.

3.1.1 Converter

A converter is used to convert data from various sources such as relational database tables, spreadsheets and webpages into RDF [4]. D2RQ [7] was used in the project in first attempt to perform data migration from RDBMS to a triple store but this approach was replaced by a more intuitive approach. The next approach was based on creating a view in NVD and using the Jena API itself (instead of using 3rd party software like D2RQ) to perform this conversion process. Both these approaches are highlighted in section 5.4 and Appendix B.

3.1.2 RDF Parser and Serializer

An RDF parser or serializer facilitates reading or writing of RDF data into one of the several file formats such as N3\(^1\), N-TRIPLE\(^2\) or RDF/XML\(^3\). This project used the Jena API to read and write triples into any one of serialization formats.

---

\(^1\)http://www.w3.org/DesignIssues/Notation3
\(^2\)http://www.w3.org/2001/sw/RDFCore/ntriples/
\(^3\)http://www.w3.org/TR/REC-rdf-syntax/
3.1.3 RDF store (or triple store)

An RDF store (or triple store) is used to efficiently store and query RDF triples. Typically an RDF store is backed by a relational database such as MySQL\(^4\). In this project Jena’s relational database (RDB) model and SPARQL database (SDB) model, and AllegroGraph\(^5\) triple store were tested as triple stores for the migrated data. The RDB model uses a more traditional approach in storing triples. It is based on a triples table that stores triples as it is without any modification. The SDB model on the other hand stores hash values of triples in the triples table. AllegroGraph is an application designed to store triples in a byte format using files on disk.

3.1.4 Reasoner

A reasoner is a program that performs inference according to a given set of inference rules. In this project, Pellet\(^6\) reasoner is used to compute inferred triples from asserted triples (migrated from NVD RDBMS) and the newly created ontology.

3.1.5 SPARQL

SPARQL\(^7\) is a query language for querying RDF data and is a W3C recommendation. ARQ\(^8\) is Jena’s query language for RDF and is used in this project for testing the performance of the application.

3.1.6 Application Interface

The application interface uses the content of RDF triple stores in interaction with users. The Java J2EE framework\(^9\) with the Model-View-Controller (MVC\(^10\)) architecture is used to implement the application interface in this project.

3.2 Semantic Web Framework - Jena

This section is focused on the Jena framework that is used extensively in this project. Jena\(^11\) is an open source Java based framework that allows building Semantic Web applications by providing a programming environment for RDF, RDFS\(^12\), OWL\(^13\), SPARQL and inference using a rule-based engine. This sec-

\(^{4}\)http://www.mysql.com/
\(^{5}\)http://www.franz.com/agraph/allegrograph/
\(^{6}\)http://clarkparsia.com/pellet/
\(^{7}\)http://www.w3.org/TR/rdf-sparql-query/
\(^{8}\)http://jena.sourceforge.net/ARQ/
\(^{9}\)http://java.sun.com/j2ee/overview.html
\(^{10}\)http://en.wikipedia.org/wiki/Model-view-controller
\(^{11}\)http://jena.sourceforge.net/
\(^{12}\)http://www.w3.org/2004/02/sesw-rdf-schema/
\(^{13}\)http://www.w3.org/2004/02/sesw-owl2-overview/
tion describes the architecture and features of Jena since it is an integral part of every step of this project.

Figure 3.1: Jena framework architecture

Figure 3.1 shows the architecture of the Jena framework. Jena provides a RDF API for building and querying RDF graphs. Further, Jena also provides an OWL API to create ontology as well as instances. Jena also provides a serialization mechanism that allows reading and writing triples in several formats such as N3, N-TRIPLE and RDF/XML. Jena provides various options to store triples, in-memory and persistent storage in the form of RDB, SDB and TDB\footnote{http://openjena.org/wiki/TDB} models. Jena also provides a SPARQL query engine to query RDF graphs. Jena also provides the ability to perform inference using its built-in rule based inference engine. Finally, Jena provides extension points for plugging in third party functionality such as reasoners.
Chapter 4

Semantic Web Application - Architecture

This chapter describes how different Semantic Web technologies have been adapted for this project. Further, the architecture of the Semantic Web application is also described.

Figure 4.1: Semantic Web application architecture
Figure 4.1 shows the architecture of the Semantic Web application that has been adapted from Jena’s architecture presented in section 3.2. Each component is presented in more detail below:

- **Ontology, RDF, and Inference API’s** - The Ontology, RDF, and Inference API’s represent the ontology that models the product domain of NVD. The ontology and RDF API’s facilitate the manipulation of the instance data created using the ontology. The inference API allows inferring additional triples from the given instance data. For example, if we have two triples such as ‘cpe:/o:microsoft:windows ce rdf:type ont:OperatingSystem’ and ‘ont:OperatingSystem rdf:type ont:Software’, the inference engine adds the additional triple ‘cpe:/o:microsoft:windows ce rdf:type ont:Software’ into the triple store when it is run. The reader should note that ‘ont’ represents the prefix of the ontology.

- **Converter** - The conversion of tuples from NVD into triples was achieved by using the converter. The Jena API was used to load the data from a view created in NVD containing only relevant product information.

- **RDF parser and serializer** - The RDF parser and serializer were used to serialize the triples from a triple store into the N3 format so that they could be transported and loaded into other triple stores. The Semantic Web application was then built around this new triple store.

- **RDF stores** - Several triple store implementations such as Jena’s RDB and SDB models and the AllegroGraph triple store were tested for their performance. Based on an evaluation of these triple stores, Jena’s SDB model was selected as the underlying triple store for this project.

- **SPARQL** - The SPARQL component takes a query input by the user and converts it into a SPARQL query that can be answered using the triple store. As an example, if a user searches for a product ‘windows ce’, this query is translated into its corresponding SPARQL query, ‘?product ont:hasName ‘windows ce’ xsd:string’, where ‘ont’ represents the prefix for the ontology, that is then evaluated on the underlying SDB triple store.

- **Application interface** - Finally, the application layer of the architecture represents the user interface layer of the project. This layer presents an interface to the user that provides the ability to enter search queries as well as to navigate through the different types of products found in the triple store.
Chapter 5

Semantic Web Application - Strategy

This chapter presents the strategy used to build a Semantic Web application in this project. Each component of the strategy is presented in detail in the context of this project, leading to a cyclic process for the entire strategy.

5.1 Use Cases

This step is a critical and time consuming in the entire strategy. It defines the initial requirements for the ontology in a simple, conversational and informal English. This step requires the collaboration of ontology developers with domain experts to create use cases that need to be modeled under the current domain. For the NVD-CPE domain the following use cases were considered,

- Searching - This is the primary use case of any data set, the ontology must be able to answer such questions as, “What are all the products that have a Vendor of Microsoft and a product name of windows nt?” or “Show me all the instance data that has been created within the past 2 months, against a given triple store” or “What are all the products that have a network stack (TCP/IP implementation)” and so on...

- Equality / Identity - The ontology should be able to determine if two instances are equal. For this equality question a closed world assumption is used; the ontology needs to be able to determine if two instances are equal based on their given properties.

- Product Ranges - The user needs to be able to express ranges of products when making statements of applicability. This means the user must be able to say things such as “This vulnerability applies to all Cisco IOS products between version 12.3 and 14.2” or “This configuration issue applies to all products before Microsoft Windows NT 6.3”. In order to
support this the correct relationships need to be modeled in the T-Box so that A-Box individuals can be related (e.g. hasNext and hasPrevious, or hasParentVersion and hasChildVersion). The ontology also needs to be able to capture product ranges when making statements of applicability; which means the need to use some sort of ProductRange class.

- Backwards Compatibility - All new instance data must be backwards compatible with the CPE specification [2]. This means from every instance in our A-Box we must be able to generate the object value of a hasCpe-Name predicate. The value being in the format of a valid CPE name (i.e. cpe:/part:vendor:product:version:update:edition:language). Ideally, but not necessarily the ontology must be able to generate instance data out of a given CPE name.

- Temporal querying for products - This is querying based on an actual time, and is completely separate from the relationship between versions of products. This use case attempts to answer the question, “Show me all products released before 2008”. This could be accomplished with a predicate like “releasedOn”.

- Products with same codebase - For example, if there is a vulnerability in Windows Server 2003, and Windows XP is based on the same codebase the ontology should be able to answer questions such as “Show me all products based on the codebase from Windows Server 2003”. This would require a predicate like “isBasedOnCodeFrom”. The predicate isBasedOnCodeFrom can be populated with the previous version of the product using a rule.

- Shared libraries - There are many open source, and proprietary, shared libraries (e.g. DLLs, JARs) and if the ontology can capture what products use the shared libraries the ontology can then identify a vulnerability in the shared library and from that infer all the vulnerable products. The ontology could capture this relationship with a predicate such as “usesSharedLibrary”.

- Automated test for products - Ability to associate automated test that is then used to check for the given product on a system (e.g. OVAL Check). The predicate in this case could be something like “canBeTestedForBy”.

5.2 Ontology Creation and Validation

The next step in the strategy involves creation of an ontology based on the use cases identified in section 5.1, Protégé 4.0 is an open source ontology editor and knowledge-base framework. It facilitates creation and validation of ontologies and is used along with Pellet[8] reasoner as an external plugin. Reasoner/Inference engine enables one to validate correctness of a given ontology based on triples inferred by performing reasoning/inferencing on the asserted triples of
the ontology. Ontology creation includes the creation of classes and subclasses, properties including data and object properties, and individuals of classes to model and validate each of the use cases. Figure 5.1 represents the class hierarchy of the ontology based on each use case.

Following are NVD Product Ontology Goals. Firstly the ontology must support NVDs primary use case involving making statements of applicability between IT concepts (e.g. CVEs, Checklists) and IT products. Secondly the Ontology must support the ability to make statements of applicability at various levels of abstraction and across ranges of products (e.g. Microsoft Windows version 4.3 to 5.6). The next goal is that the Ontology must support the ability to capture granular product identification data which may vary on a per product basis. Finally the Ontology must support the Common Platform Enumeration which is the standardized method for naming IT products.

Statements of Applicability can be modeled as First Class Individuals. Applicability statements are a way of relating a grouping of products to a particular IT concept (e.g. CVE, CCE, Checklist). If modeled as actual classes in ontology applicability statements will provide the ability to create groupings of products at various levels of abstraction depending on the needs of a use case. For instance it would be possible to represent all products in a certain range and represent all products that use a certain shared library. Predicates can be defined to capture relationships between applicability statements. It would be possible to express that one applicability statement is a prerequisite for another statement and that one applicability statement subsumes another statement.

NVD Ontology models two separate concept structures as formal is-a hierarchies namely Category concept hierarchy and Identification concept hierarchy. Category concept hierarchy models various product categories and Identification concept hierarchy enables unique identification of products based on CPE names. NVD Ontology also includes other types of semantic relationships such as made up of or contains relationships between applications and codebases. Explicit differences between sets of products are modeled by defining disjoint sets such as hardware vs software products. Figure 5.1 depicts the various class and subclass using is-a relationships.

Product Category Hierarchy is modeled using the following predicates.

- hasIdentification : This property has domain as the “Product” class and range consists of the “IdentificationStrategy” class. This property is an owl:inverseFunctionalProperty property.
- hasReleaseDate: This property has domain as the “Product” class and range is a xsd:string.
- hasCpeName: Its domain is the “Product” class and range is a xsd:string.
- usesSharedLibrary: Its domain is “Application” class and range is SharedLibrary
- contains: Its domain as well as range is “Product” class. This is inverseOf the property “containedIn”.
Figure 5.1: Class hierarchy diagram
• hasOwner: The domain is “Product” and range is a “Foaf:Agent”. This property is inverseOf “ownedBy” property.

• Many other possibilities exist, very granular predicates can be defined further down tree.

Identification is modeled using the following predicates.

• hasName: Its domain is “IdentificationStrategy”
• hasModelNumber: Its domain is “PhysicalDeviceIdentificationStrategy”
• hasCiscoTrainIdentifier: Its domain is “CiscoIOS_Strategy”
• hasCiscoInterimBuildNumber: Its domain is “CiscoIOS_Strategy”
• hasMicrosoftMajorVersion: Its domain is “NTKernel_Strategy”
• hasVersion: Its domain is “GenericIdentificationStrategy”
• hasUpdate: Its domain is “GenericIdentificationStrategy”

Applicability Statements are modeled as first class Individuals using the following possible predicates. Predicates capturing information relating to products encompassed by a applicability statement.

• includesProduct : has domain “ApplicabilityStatement” and range “Product”. This property is inverseOf “memberOf”
• memberOf: has domain “Product” and range of “ApplicabilityStatement”. This property is inverseOf “includesProduct”
• minimumProduct: has domain “ProductRangeStatement” and range being “Product”
• maximumProduct: has domain “ProductRangeStatement” and range is “Product”
• sharedLibrary: has domain as well as range as “SharedLibrary”

Predicates capturing relationships between applicability statements and other applicability statements

• hasPrerequisite: with domain “ApplicabilityStatement” and range “ApplicabilityStatement”. This property is inverseOf “prerequisiteFor”
• prerequisiteFor: with domain “ApplicabilityStatement” and range also as “ApplicabilityStatement”. This property is inverseOf “hasPrerequisite”
• subsumes: with domain “ApplicabilityStatement” and range “ApplicabilityStatement”. It is inverseOf “subsumedBy”
Predicates capturing relationships between applicability statements and IT concepts (e.g. CVE, CCE, Checklist)

- hasApplicabilityStatement: with domain “some IT concept” and range “ApplicabilityStatement”. This property is inverseOf “appliesTo”
- appliesTo: has domain “ApplicabilityStatement” and range is “some IT concept”. This property is inverseOf “hasApplicabilityStatement”

Ontology provides the means to make more granular Statements of Applicability. Shared Library (e.g. DLL, JAR) instance data can be captured and related to typical product instance data. This can be done through predicates such as usesSharedLibrary. Analysts can then associate vulnerabilities with shared libraries and simple queries can be used to determine all products which use the shared library.

Classes can be added to the Ontology to capture codebases. Relationships can then be asserted on instance data to relate products to codebase from which they originate (e.g. isBasedOnCodeFrom). Analysts can assign vulnerability to a specific codebase, and the system can generate the list of all applicable products. These predicates could become the standard way for all product ontologies to declare these relationships. This would provide a shared understanding across a wide set of data.

The Ontology Provides the Capability for Modeling Ranges of Products. This is accomplished with four predicates: hasNextVersion, hasPreviousVersion hasLaterVersion (transitive), hasEarlierVersion (transitive). These four predicates are modeled using a predicate hierarchy such that the non-transitive predicates are related to the transitive predicates through rdfs:subPropertyOf. Ontology also provides the means to make inferencing for broad statements of applicability making it possible to define classes to identify all individuals which meet desired criteria. For example, a class could be defined to capture all CiscoIOS Products.

A Formalized model (ontology) will allow for a shared understanding of how to capture normalized product data. The Identification Strategy hierarchy provides a method to define vendors versioning strategies. The granularity of the model is up to the community. The model itself will show users the types of relationships that must be captured to identify a product. In the future it may even be possible to create a complementary ontology which tells users HOW to find the data for e.g. where to look, various commands and API calls. This will really allow us to put most of the logic in the ontology itself and provide a high level of confidence for users creating product instance data.

Figure 5.2 represents the class hierarchy of the ontology which we explain in detail based on each use case below.

5.3 Ontology Migration

After the creation and validation of the ontology in section 5.2, the next step is migration of this ontology to a Java class, that will be used for data migration.
Figure 5.2: NVD CPE Ontology
Jena’s schemagen\(^1\) was used to list the classes and properties from the NVD-CPE ontology in a “ProductOntology” java class as static constant variables. Jena’s schemagen is run as an Ant task as outlined in Appendix A.

5.4 Data Migration

The next step in the strategy is to migrate the product data from NVD to a triple store using the ontology created in section 5.2 and the corresponding Java class created in section 5.3. Two approaches were tested to perform this data migration, the first used a D2RQ [7] mapping file that allows the data in a non-RDF relational database to be viewed as RDF, and the second used a view created in NVD to perform data conversion. In this section we only describe the second approach since this was selected to perform the data migration. We refer the reader to Appendix A for further details on both approaches.

The approach taken to perform data migration is based on creating a view in NVD that is closely related to the ontology created in Sections 5.2 and 5.3. The view is constructed using data from different tables representing product information and is relevant to the ontology. The view serves as a schema as well as contains the data needed to perform the data migration. The SQL query used in NVD to create this view is,

```
SELECT distinct ncc.name as hasName, nct.hasLocale as hasLocale
    nct.title as hasTitle, ncl.cpe_name as hasCpeName,
    convert(varchar(10), ncl.create_date, 126) as hasCreationDate
INTO NVD_PRODUCT_ONTOLOGY
FROM NVD_CPE_LOOKUP as ncl
INNER JOIN NVD_CPE_COMPONENT as ncc
ON ncl.product_id = ncc.id
INNER JOIN NVD_CPE_TITLE as nct
ON ncl.title_group_id = nct.group_id
GROUP BY ncc.name, nct.locale, nct.title, ncl.cpe_name,
    convert(varchar(10), ncl.create_date, 126)
ORDER BY ncc.name;
```

The fields in the `SELECT` clause of the SQL query are needed to capture the product information in an ontological representation of the NVD-CPE data. The fields represent a product’s name, its locale information, a product’s title, a CPE name assigned to that product (this is the identifying attribute for any product in the triple store), and a product’s creation date. The view is then used to construct a triple as follows,

- CPE name → subject

\(^{1}\)http://jena.sourceforge.net/how-to/schemagen.html

17
• column name in view → predicate
• value of cell for that column → object

Using this approach the data from the view is converted into triples and then stored in a triple store to be used for further processing.

5.5 Reasoning

This section of the strategy focuses on the reasoning component used to infer additional triples from the data migrated from NVD to a triple store. Two different reasoning strategies are available for use, we briefly explain them below followed by the actual approach taken in this project.

Forward chaining [8] is a reasoning technique that begins with the available data and uses the set of inference rules to infer additional data until a goal is reached. An inference rule is made up of two parts, an antecedent and a consequent.

\[
\text{If } a, \text{ then } b \tag{5.1}
\]

where, \( a \) is the antecedent and \( b \) is the consequent.

An inference engine based on forward chaining uses the inference rules until the antecedent in a rule is true. If the inference engine can infer the consequent for the same rule then new knowledge is discovered and added to the knowledge base. Consider the following set of rules,

• If X croaks and eats flies - Then X is a frog
• If X chirps and sings - Then X is a canary
• If X is a frog - Then X is green
• If X is a canary - Then X is yellow

Consider that the goal is to find the color of a pet named Fritz who croaks and eats flies. With forward chaining, the first rule would be used to infer that Fritz is a frog, this data is added to the knowledge base. Again, all inference rules are checked, the third rule is satisfied, hence the data Fritz is green is added to the knowledge base. No further knowledge can be discovered and the original goal of finding the color of Fritz is reached.

Backward chaining [8] is a reasoning technique that begins with a set of goals and moves backwards from the consequent to the antecedent and checks if knowledge exists to support any consequent. An inference engine based on backward chaining would check the consequent of each inference rule to see if it is part of the set of goals. If the consequent of a rule matches one of the goals, its antecedent is checked, if it is not true then the antecedent is added to the set of goals. If on the other hand, the antecedent is true then the goal is true and the new inferred knowledge is added to the knowledge base. Consider the same
set of rules and goal as before. With backward chaining, the third and fourth rules are satisfied, since the goal is to find the color of Fritz. Their antecedents are added to the set of goals. The rule base is searched again, and the first two rules are selected. The antecedent of the first rule is true and hence the conclusion that Fritz is a frog is reached. Also, the goal of finding Fritz’s color is achieved. The reader should note that both examples are taken from [9].

In this project, the forward chaining approach was used to infer additional triples. The Pellet reasoner [10] is used as the inference engine during the data migration process, in conjunction with the triples from the view in NVD and the ontology to infer additional triples that are added to a triple store.

5.6 SPARQL Queries

The next step in the strategy is to write queries to test the data migration and reasoning done as part of Sections 5.4 and 5.5 based on use cases created in 5.1. SQL queries were written to test the view in NVD and SPARQL queries were written to test the data in a triple store. The similarity between a SPARQL query and its corresponding SQL query is shown by presenting both queries simultaneously for each use case in tables 5.1 and 5.2. On the other hand, table 5.3 shows only inference SPARQL queries that can be run on a triple store, there are no corresponding SQL queries since NVD cannot perform inference. Also note that the SPARQL queries use the following prefixes,

- PREFIX pO: <http://scap.nist.gov/productOntology#>
- PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
- PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

5.7 Application

This step in the strategy combines the data migrated from NVD into a triple store and the SPARQL queries into a web application. The web application is designed using the Java J2EE framework and it uses the model-view-controller (MVC) architecture to provide various functionalities. A user can search for products and agents using a search box that is provided on the home page. An additional navigation technique is to use the menu bar that provides different types of products that can be found in NVD. A user can simply click on each type of product in the bar which then further refines that category and also lists the different products that belong to the category. More details on the application and sample screenshots can be found in Appendix C.

---

2http://java.sun.com/j2ee/overview.html
3http://en.wikipedia.org/wiki/Model-view-controller
<table>
<thead>
<tr>
<th>Query</th>
<th>SPARQL Query</th>
<th>SQL Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find all Agents</td>
<td><code>SELECT ?a WHERE ?a rdf:type pO:Agent .</code></td>
<td><code>SELECT count(distinct ncl.vendor_id) FROM NVD_CPE_LOOKUP as ncl</code></td>
</tr>
<tr>
<td>Find all Products</td>
<td><code>SELECT DISTINCT ?p WHERE ?p rdf:type pO:Product .</code></td>
<td><code>SELECT count(distinct ncc.name) FROM NVD_CPE_LOOKUP as ncl, NVD_CPE_COMPONENT as ncc WHERE ncc.id=ncl.product_id.</code></td>
</tr>
<tr>
<td>Find products by <code>Microsoft</code></td>
<td><code>SELECT ?p WHERE ?p pO:hasOwner pO:microsoft .</code></td>
<td><code>SELECT count(distinct ncc2.name) FROM NVD_CPE_LOOKUP as ncl, NVD_CPE_COMPONENT as ncc, NVD_CPE_COMPONENT as ncc2 WHERE ncc.name='microsoft' AND ncc.id=ncl.product_id AND ncc2.id=ncl.product_id.</code></td>
</tr>
<tr>
<td>Find agent for a specific product</td>
<td><code>SELECT DISTINCT ?a WHERE ?p pO:hasName 'windows_ce'^xsd:string . ?p pO:hasOwner ?a .</code></td>
<td><code>SELECT distinct ncc2.name FROM NVD_CPE_COMPONENT as ncc, NVD_CPE_COMPONENT as ncc2, NVD_CPE_LOOKUP as ncl WHERE ncc.id=ncl.product_id AND ncc2.id=ncl.product_id AND ncc.name='windows_ce'</code></td>
</tr>
<tr>
<td>Find CPE names of all products</td>
<td><code>SELECT DISTINCT ?p WHERE ?p pO:hasCpeName ?cn .</code></td>
<td><code>SELECT count(CpeNames.countCpeNames) FROM ( SELECT count(distinct ncc2.name) as pNames, count(ncl.cpe_name) as countCpeNames FROM NVD_CPE_LOOKUP as ncl, NVD_CPE_COMPONENT as ncc, NVD_CPE_COMPONENT as ncc2 WHERE ncl.product_id = ncc2.id GROUP BY ncc2.name ) AS CpeNames</code></td>
</tr>
</tbody>
</table>

Table 5.1: SPARQL queries and the corresponding SQL queries used in our performance tests - Q1 to Q5
<table>
<thead>
<tr>
<th>Query</th>
<th>SPARQL Query</th>
<th>SQL Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find product with a CPE name</td>
<td><code>SELECT ?p</code> &lt;br&gt;<code>WHERE</code> &lt;br&gt;<code>?p pO:hasCpeName ?cn .</code> &lt;br&gt;<code>FILTER</code> &lt;br&gt;<code>(?cn = 'cpe:/a:mambo:mambelfish_component:1.1')</code></td>
<td><code>SELECT distinct ncc.name</code> &lt;br&gt;<code>FROM NVD_CPE_COMPONENT as ncc,</code> &lt;br&gt;<code>NVD_CPE_LOOKUP as ncl</code> &lt;br&gt;<code>WHERE ncc.id=ncl.product_id</code> &lt;br&gt;<code>AND ncl.cpe_name='cpe:/a:mambo:mambelfish_component:1.1'</code></td>
</tr>
<tr>
<td>Find products with dates</td>
<td><code>SELECT DISTINCT ?p</code> &lt;br&gt;<code>WHERE</code> &lt;br&gt;<code>?p pO:hasCreationDate ?d .</code></td>
<td><code>SELECT count((CreationDates.countCreateDates)</code> &lt;br&gt;<code>FROM ( SELECT count(distinct ncc2.name) as pNames,</code> &lt;br&gt;<code>count(ncl.cpe_name) as countCreateDates</code> &lt;br&gt;<code>FROM NVD_CPE_LOOKUP as ncl,</code> &lt;br&gt;<code>NVD_CPE_COMPONENT as ncc2</code> &lt;br&gt;<code>WHERE ncl.product_id = ncc2.id</code> &lt;br&gt;<code>GROUP BY ncc2.name ) AS CreationDates</code> &lt;br&gt;<code>SELECT count((distinct ncc.name)</code> &lt;br&gt;<code>FROM NVD_CPE_COMPONENT as ncc,</code> &lt;br&gt;<code>NVD_CPE_LOOKUP as ncl</code> &lt;br&gt;<code>WHERE ncc.id=ncl.product_id</code> &lt;br&gt;<code>AND convert(varchar(10), ncl.create_date, 126) = '2007-08-23'</code></td>
</tr>
<tr>
<td>Find products with creation date</td>
<td><code>SELECT ( count(?p) as ?productSameDate )</code> &lt;br&gt;<code>WHERE</code> &lt;br&gt;<code>?p pO:hasCreationDate ?cd .</code> &lt;br&gt;<code>FILTER(?cd = '2007-08-23'^^xsd:date)</code></td>
<td><code>SELECT count((distinct ncc.name)</code> &lt;br&gt;<code>FROM NVD_CPE_COMPONENT as ncc,</code> &lt;br&gt;<code>NVD_CPE_LOOKUP as ncl</code> &lt;br&gt;<code>WHERE ncc.id=ncl.product_id</code> &lt;br&gt;<code>AND convert(varchar(10), ncl.create_date, 126) = '2007-08-23'</code></td>
</tr>
<tr>
<td>Find all products with type = ‘a’</td>
<td><code>SELECT DISTINCT ?p</code> &lt;br&gt;<code>WHERE</code> &lt;br&gt;<code>?p pO:hasType ?t .</code> &lt;br&gt;<code>FILTER(?t = ‘a'^^xsd:token)</code></td>
<td><code>SELECT count((distinct ncc.name)</code> &lt;br&gt;<code>FROM NVD_CPE_COMPONENT as ncc,</code> &lt;br&gt;<code>NVD_CPE_LOOKUP as ncl</code> &lt;br&gt;<code>WHERE ncc.id=ncl.product_id AND ncc.type_lette='a'</code> &lt;br&gt;<code>SELECT ncc.type_lette, count((distinct ncc.name)</code> &lt;br&gt;<code>FROM NVD_CPE_COMPONENT as ncc,</code> &lt;br&gt;<code>NVD_CPE_LOOKUP as ncl</code> &lt;br&gt;<code>WHERE ncc.id=ncl.product_id GROUP BY ncc.type_lette</code></td>
</tr>
</tbody>
</table>

Table 5.2: SPARQL queries and the corresponding SQL queries used in our performance tests - Q6 to Q10
Table 5.3: Inference SPARQL queries used in performance tests - Q11 to Q13

<table>
<thead>
<tr>
<th>Query</th>
<th>SPARQL Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Products within range of release date</td>
<td>SELECT ?p WHERE ?p pO:hasReleaseDate ?d . WHERE FILTER(?d ≤ “ + d1 + ” &amp;&amp; ?d ≥ “ + d2 + ”)</td>
</tr>
</tbody>
</table>

5.8 Performance Metrics

The next logical step in the strategy is to test the application with performance metrics such as the time for data migration from NVD to a triple store as well as the time to query the triple store with SPARQL queries. The loading and querying times for several different triple stores were compared and the impact of running an inference engine to infer additional triples was also taken into consideration. This study is necessary to gain the maximum power of a triple store in the application, and is presented in Chapter 6.

5.9 Cyclic Process

Sections 5.1 through 5.8 outline the strategy used to transform the CPE information from NVD to a triple store, building a Semantic Web application around the resulting triple store and testing the application for various performance metrics. But a single pass through these steps may not be sufficient, there may be other use cases to be modeled or the ontology may need to be further refined. In these situations, additional passes through all the previous steps are made until all use cases are modeled and a satisfactory ontology is constructed. The steps given in sections 5.1 through sections 5.8 therefore represent a general strategy for converting any relational database into a triple store as a cyclic process.

Figure 5.3 represents a general methodology that can be used to convert a relational database to a triple store as a cyclic process. Each step in the figure
Figure 5.3: Relational database to Semantic Web migration strategy

represents a particular step in the methodology, beginning with use cases and terminating with performance evaluation.
Chapter 6
Semantic Web Application - Performance

This section begins by giving details about the experimental setup used to test the application followed by a description of the various performance metrics used to test the application. The experimental environment is composed of two parts, a utility to migrate the NVD-CPE data to a triple store, and an application built around the data in the triple store. The utility is tested with different triple stores such as Jena’s relational database model (RDB) and SPARQL database model (SDB), and also AllegroGraph triple store [11]. All these triple stores are indexed during each run of the experiment to optimize their performance. The utility is tested for time needed to convert the relational data into triples as well as time for sample query execution on the triple store. Again, both these tests were run for data that was converted with and without inferring additional triples using an inference engine. Only a subset of the total products from NVD were used for the inference case (5961) while for the non-inference case all the products from NVD (96216) were used. For the conversion from NVD to a triple store different approaches were compared using metrics such as the total space, the index space and the log space of the resulting triple store, as well as the actual time to perform the conversion. Based on the conclusions from the study using the utility with different triple stores, the application was built with Jena’s SDB triple store. Further, the data migrated to the SDB triple store contains inferred triples based on a subset of the actual data (5961 products). Again, some of the same sample queries that were used to test the utility were used to measure the performance of the application. For query execution time, sample queries such as the total products, total agents, products that are TCP/IP devices etc were used for testing various triple stores. These queries are presented in tables 5.1-5.3.
6.1 Experimental Setup

The experiments for measuring performance metrics across different triple stores were run using Microsoft Windows Server 2008 Standard\(^1\) (64 bit) on an Intel Xeon E5530 2.4GHz CPU with 9GB DDR2 667 MHz memory and a 1.5 TB SATA hard drive. The Java version used was JRE v1.6.0.18 and the relational database used was Microsoft SQL Server 2005\(^2\). 10 simple queries and 3 inference based queries were used to measure the performance metrics across the triple stores. These queries are quite diverse in nature and capture all use cases of the product domain(NVD).

The Semantic Web application was developed using Linux Ubuntu v8.04 Hardy Heron\(^3\), on an Intel Core2 Duo T5870 2GHz CPU with 2GB DDR2 667 MHz memory and a 250GB Hitachi HTS54322 SATA hard drive. The Java version used was JRE v1.6.0.10 and the relational database used was MySQL\(^4\). Again the same sample queries that were used to measure the performance metrics were also used to test the application.

6.2 Data loading statistics using the utility

<table>
<thead>
<tr>
<th>Metric</th>
<th>RDBMS</th>
<th>RDB</th>
<th>SDB</th>
<th>AllegroGraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>SQL Ser 05</td>
<td>Jena-2.5.6</td>
<td>SDB-1.1</td>
<td>AG-3.2</td>
</tr>
<tr>
<td>Size(Rows/Tr)</td>
<td>96485 (R)</td>
<td>982403 (T)</td>
<td>982403 (T)</td>
<td>982403 (T)</td>
</tr>
<tr>
<td>Total Space(MB)</td>
<td>13.08</td>
<td>1044.00</td>
<td>302.63</td>
<td>387.00</td>
</tr>
<tr>
<td>Index Space(MB)</td>
<td>0.008</td>
<td>674.22</td>
<td>75.55</td>
<td>316.06</td>
</tr>
<tr>
<td>Log Space(MB)</td>
<td>-</td>
<td>285.06</td>
<td>82.44</td>
<td>-</td>
</tr>
<tr>
<td>Load Time</td>
<td>-</td>
<td>231.6 s</td>
<td>284.6 s</td>
<td>164.8 s</td>
</tr>
</tbody>
</table>

Table 6.1: Data loading statistics for various triple stores with the utility

Table 6.1 shows the performance of different triple stores when they are used for data migration. For this particular set of experiments the NVD-CPE data was migrated without running an inference engine to infer additional triples, therefore all products were used. The view created in NVD needs the least amount of storage space but only simple queries can be run on this view. Amongst all triple stores, Jena’s SDB model needs the least amount of storage space. This is because of the denormalized schema of the SDB model and

\(^3\)http://releases.ubuntu.com/hardy/  
\(^4\)http://www.mysql.com/
the use of a hash value of the subject, predicate and object in the triples table rather than the actual values. The loading time for the AllegroGraph triple store [11] is the fastest.

6.3 Data loading (with inference) statistics using the utility

<table>
<thead>
<tr>
<th>Metric</th>
<th>RDB</th>
<th>SDB</th>
<th>AllegroGraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Jena-2.5.6</td>
<td>SDB-1.1</td>
<td>AG-3.2</td>
</tr>
<tr>
<td>Size(Rows/Triples)</td>
<td>97814 (T)</td>
<td>97814 (T)</td>
<td>97814 (T)</td>
</tr>
<tr>
<td>Total Space(MB)</td>
<td>118.31</td>
<td>61.38</td>
<td>38.00</td>
</tr>
<tr>
<td>Index Space(MB)</td>
<td>66.98</td>
<td>9.65</td>
<td>31.46</td>
</tr>
<tr>
<td>Log Space(MB)</td>
<td>13.31</td>
<td>38.38</td>
<td>-</td>
</tr>
<tr>
<td>Load Time</td>
<td>18.58 hrs</td>
<td>17.62 hrs</td>
<td>19.06 hrs</td>
</tr>
</tbody>
</table>

Table 6.2: Data loading (with inference) statistics for various triple stores using the utility

Table 6.2 shows the performance of different triple stores when they are used for data migration with the use of an inference engine to infer additional triples. Since the inference engine is run only a part of the actual product data from NVD is used (≈ 1/16 × data). As table 6.2 shows, the AllegroGraph triple store [11] uses the least amount of storage space while Jena’s SDB model loads the triples the fastest. A significant increase in time required for loading the triples is also seen in this case because the inference engine is run to infer additional triples. This significant time increase is acceptable because the inference engine is run only during the data migration process. The reader should also note that using more memory for the data migration process significantly improves loading time. (a speedup by a factor of ≈ 2 for an increase in memory by a factor of ≈ 8.75)

6.4 Data querying statistics using the utility

Table 6.3 shows a comparison of query execution times for simple non-inference queries with the NVD view in RDBMS as well as different triple stores. Clearly, the RDBMS is the fastest of all approaches, but the reader must note that queries to the view are in SQL [12] while queries to the triple stores are in SPARQL [13]. The SPARQL engines supported for the triple stores are slower than the SQL engine and hence this time difference between the SPARQL and
<table>
<thead>
<tr>
<th>Query (Triples)</th>
<th>RDBMS Time (ms)</th>
<th>RDB Time (ms)</th>
<th>SDB Time (ms)</th>
<th>AllegroGraph Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agents (9898)</td>
<td>53.2</td>
<td>737.4</td>
<td>711.2</td>
<td>945.6</td>
</tr>
<tr>
<td>Products (96216)</td>
<td>10.6</td>
<td>1013.2</td>
<td>723.4</td>
<td>5572.8</td>
</tr>
<tr>
<td>MS Products (2616)</td>
<td>12.0</td>
<td>26.4</td>
<td>30.0</td>
<td>141.4</td>
</tr>
<tr>
<td>'win_ce' Agent (1)</td>
<td>27.0</td>
<td>74.8</td>
<td>8.4</td>
<td>11.0</td>
</tr>
<tr>
<td>All CPE names (96216)</td>
<td>11.0</td>
<td>1235.0</td>
<td>1274.6</td>
<td>7321.2</td>
</tr>
<tr>
<td>Give CPE name (1)</td>
<td>1.0</td>
<td>838.6</td>
<td>472.2</td>
<td>5425.0</td>
</tr>
<tr>
<td>All Creation Dates (96216)</td>
<td>8.2</td>
<td>1183.8</td>
<td>1499.4</td>
<td>5464.4</td>
</tr>
<tr>
<td>Given Creation Date (56811)</td>
<td>70.6</td>
<td>937.4</td>
<td>1427.4</td>
<td>5519.0</td>
</tr>
<tr>
<td>Type ‘a’ (82981)</td>
<td>34.0</td>
<td>749.6</td>
<td>1120.6</td>
<td>5325.0</td>
</tr>
<tr>
<td>Group By Type</td>
<td>92.6</td>
<td>768.4</td>
<td>1243.8</td>
<td>5406.2</td>
</tr>
<tr>
<td>h(4941), o(8294), a(82891)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: Data querying statistics for various triple stores using the utility SQL queries. Further, even though RDBMS gives the fastest time, the queries are regular queries and do not involve any inference. To fully exploit the power of the Semantic Web, inference queries must be run with the application. This clearly cannot be done using the RDBMS. Further, Jena’s RDB and SDB models perform similarly for all queries with a much better query time than the AllegroGraph triple store. This is because the relational database underlying the RDB and SDB models implements query result caching which significantly improves query result times over the AllegroGraph triple store which stores triples in files on disk.

### 6.5 Data querying (with inference) statistics using the utility

Table 6.4 shows a comparison of the different triple stores for regular as well as inference queries. Regular queries are run to demonstrate that the triple stores can perform these as well as inference queries. This shows that a Semantic Web application can not only be used to perform the functions of traditional database applications but also has added power of inference. Again, Jena’s RDB and SDB models perform similarly for these queries, both being better than the AllegroGraph triple store. A similar argument as before applies for this speedup...
based on the query result caching ability of the underlying relational database.

The conclusions that can be drawn from tables 6.1-6.4 are that Jena’s RDB and SDB models perform similarly for query execution, both being better than the AllegroGraph triple store. Further, the data loading time for SDB is the best amongst all triple stores when an inference engine is used, and hence the SDB triple store was used to store and query the data in the application.

### 6.6 Data Querying (With Inference) Statistics using the Application

<table>
<thead>
<tr>
<th>Query (Triples)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Utility-SDB</td>
</tr>
<tr>
<td>Agents (923)</td>
<td>379.2</td>
</tr>
<tr>
<td>Products (5961)</td>
<td>325.6</td>
</tr>
<tr>
<td>TCP/IP Devices (92)</td>
<td>6.6</td>
</tr>
<tr>
<td>Products with shared library (2)</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 6.5: Data querying (with inference) statistics for utility versus application

From table 6.5 it is clear that the queries take a much longer time with the application than with the utility. This is because the system used to test the application is not as powerful as the system used to test the utility. Further, the utility directly runs the query whereas the application needs to first formulate the query based on the input of the user after which the query is run.
Chapter 7

Conclusion and Future Work

This project showed that the choice of a semantic model over a relational model enhances the automation of vulnerability and product management. Further, traditional relational models and semantic models differ in many ways. For example, a relational model is table/class focused while a semantic model is relationship focused; these relationships decide what inferences can be drawn. A modeler must take into account these differences when conducting a conversion from a relational model to a semantic model. An ontology cannot be built the same way as a relational database since many of the added benefits of using a semantic model will be lost.

The project also shows that creating a comprehensive list of use cases in the beginning of a project is challenging, even for domain experts. Certain use cases emerge during the project that were missed earlier; the cyclic process used in the project makes incorporation of these use cases flexible.

The project also shows that efforts must be made to optimize triple store performance with strategies such as indexing in order to gain the most power out of a triple store.

The project also shows that the implementation of a system must carefully choose a triple store/reasoner for their implementation. This implies a trade-off between speed and power. By adding speed you will normally lose power (i.e. will support less and less of OWL as speed goes up).

Future work for this project includes testing other implementations of triple stores and to evaluate different inference strategies to speed up the process of data migration. Further work could also be done on adding more features to the Semantic Web application developed as part of this project. Finally, other parts of NVD could also be migrated to a triple store to have complete vulnerability management using the Semantic Web technologies.
Bibliography


Appendix A

Ontology Migration

In the process of ontology migration the goal is to create a Java class, that can be used to add triples during data migration as well as to add new triples during instance creation, based on the ontology created to model CPE information from NVD. Jena Schemagen, a tool used to perform ontology migration, is outlined in the sections below.

A.1 Prerequisites for ontology migration

The ontology migration process was carried out using Jena Schemagen in a Linux Ubuntu v8.04 Hardy Heron environment with the following additional components properly installed and configured,

- Java version 1.6
- Jena-2.5.6
- Apache Ant 1.7

A.2 Ontology migration using Jena Schemagen

Jena Schemagen is used to convert an ontology into a Java class containing static constant terms that define classes and properties used by that ontology. Jena Schemagen is a part of the Jena distribution and can be run as a command line program or using an Apache Ant task, both methods being available with various configurable parameters. The following Ant tasks were used to perform ontology migration,

\[1\] http://jena.sourceforge.net/how-to/schemagen.html
\[2\] http://java.sun.com/javase/downloads/index.jsp
\[3\] http://jena.sourceforge.net/
\[4\] http://ant.apache.org/
An explanation for each Ant task and different parameters used above is also provided,

- **owl-vocab** - This Ant target creates a Java class based on an ontology that is input, it has several configurable parameters, the ones used in the utility are presented below,
  
  - `-i` = The input ontology
  - `-o` = The output Java class
  - `-noindividuals` = Flag that disables instances that are part of the ontology to be added to the Java class
  - `-ontology` = The generated Java class will use ontology terms instead of RDF terms

- **check.owl-vocab** - This Ant target is used by owl-vocab to ensure that a Java class is created only if an ontology has been updated after the Java class was created.
Appendix B

Data Migration

For the data conversion process from NVD to a triple store two different approaches were used. Details about the two approaches are presented in this section including instructions of how they can be setup to perform the conversion process.

B.1 Prerequisites for data conversion

This section provides the prerequisites needed to run the utility for data conversion. The environment used to perform the conversion process was Microsoft Windows Server 2008 Standard edition. The following components also need to be installed and configured:

- Microsoft SQL Server 2005
- Java version 1.6 (any release)
- Apache Ant 1.7
- AllegroGraph Server 3.2
- Jena-2.5.6
- Pellet-2.0.0-rc5 reasoner

B.2 D2RQ based conversion approach

Database to Relational Query (D2RQ) is a tool that allows access to the content of traditional non-RDF relational databases without the need of converting the

---

2http://www.franz.com/agraph/allegrograph/
3http://clarkparsia.com/pellet/
4http://www4.wiwiss.fu-berlin.de/bizer/d2rq/
database into RDF\textsuperscript{5}. D2RQ specifies mappings between a relational database schema and OWL\textsuperscript{6}/RDFS\textsuperscript{7} ontologies. This mapping file gives applications using D2RQ a RDF view of a non-RDF database.

The mapping file created using D2RQ gives a relationship between, tables and columns of the relational database schema, and the corresponding classes and properties in an OWL/RDFS ontology. The table and column names are viewed as classes and properties in the OWL/RDFS ontology respectively using the mapping file. The mapping file generated by D2RQ is used to migrate relational data into triples since these resulting triples will be manipulated by the application. From relational data a triple is constructed as follows,

- primary key value in each tuple \rightarrow subject
- column name other than primary key column \rightarrow predicate
- value of cell for that column \rightarrow object

For the D2RQ based conversion approach an Apache Ant based utility was created that provides command line programs to perform the conversion process. The conversion process is done in two steps. First step is to generate the D2RQ mapping file using NVD schema and second step is to migrate the data with help of the newly created mapping file. Data migration procedure with this utility was tested only with Jena’s RDB model for the following reason. A combined view of different tables from NVD is needed to represent the CPE data which cannot be obtained using D2RQ. Further, D2RQ does not allow database table updation which is critical in the application. Various parts of this utility are outlined below along with instructions on how the conversion process can be performed.

- **src** folder - This folder contains two source files that facilitate the conversion of a relational database to a triple store. The files are,
  - `d2rq.generate_mapping` - This class is copied from the D2RQ software, it is used to generate a mapping file that gives a RDF view of the relational database.
  - `edu.utdallas.nvd.RelationalToRDB` - This class reads the relational database using the generated mapping file and puts the data into a Jena RDB triple store.

- **lib** folder - contains the library files needed to run the utility.

- **build.properties** - This file provides the command line arguments needed by the utility to perform the migration process.

- **build.xml** - This file defines the programs used to perform data migration as Ant targets.

\textsuperscript{5}http://www.w3.org/RDF/
\textsuperscript{6}http://www.w3.org/TR/owl-features/
\textsuperscript{7}http://www.w3.org/TR/rdf-schema/
- `run-mapping` - This Ant target creates the mapping file for the relational database giving us a RDF view of the relational data. The parameters needed to run this script are given below as they appear in the build.properties file along with the meaning of that parameter,
  - `$\{relational.driver\}` = This is the database driver for the relational database.
  - `$\{relational.username\}` = This is the username of the owner of the relational database.
  - `$\{relational.password\}` = This is the password of the owner of the relational database.
  - `$\{d2rq.mappingfile\}` = This is the output mapping file name.
  - `$\{relational.db\}` = This is the actual database name.

- `run` - This Ant target converts the relational database to the Jena RDB triple store using the mapping file created above. The parameters are again given below,
  - `$\{rdb.driver\}` = This is the database driver for the triple store.
  - `$\{rdb.db\}` = This is the database name for the triple store.
  - `$\{rdb.username\}` = This is the database username of the owner of the triple store.
  - `$\{rdb.password\}` = This is the database password of the owner of the triple store.
  - `$\{rdb.type\}` = This is the database type for the triple store like MySQL.
  - `$\{rdb.storename\}` = This is the triple store name that Jena uses to reference the triple store.
  - `$\{d2rq.mappingfile\}` = This is the output mapping file name.

Since Microsoft SQL Server 2005 was used as the database, the D2RQ source code needed to be tweaked so that the `run-mapping` Ant target could generate the mapping file more efficiently. Following changes were made to the D2RQ source code to eliminate the system tables of Microsoft SQL Server 2005 being mapped to the mapping file,

- `de.fuberlin.wiwiss.d2rq.mapgen.MappingGenerator` - The following was added to the `run()` method to avoid mapping the system tables from Microsoft SQL Server 2005.

  ```java
  if( this.databaseType.equalsIgnoreCase( "Other" )
      && tableName.toString().length() > 18
      && tableName.toString().substring(0, 18).equalsIgnoreCase( "INFORMATION_SCHEMA" ) ) continue;

  if( this.databaseType.equalsIgnoreCase( "Other" )
      && tableName.toString().length() > 4
      && tableName.toString().substring(0, 4).equalsIgnoreCase( "sys." ) ) continue;
  ```
The following was added to the function `columnType(Attribute column)` to give the ability to convert the `nvarchar` and `ntext` datatypes to the triple store.

```java
    case Types.NVARCHAR: return TEXT_COLUMN;
    case Types.LONGNVARCHAR: return TEXT_COLUMN;
    // longnvarchar (-16) = ntext
```

### B.3 Relational View based conversion approach

For the relational view based conversion approach a view was first created in NVD with CPE data to be migrated. An Apache Ant based utility was then used to convert this data to a triple store using the ontology constructed in Section 5.2. This utility was tested with Jena’s RDB and SDB models as well as the AllegroGraph triple store for data conversion as well as query execution performance. The details of the utility are presented below.

- **src folder** - This folder contains source files that facilitate the conversion of a relational database to a triple store based on the constructed ontology. The files are,
  - `edu.utdallas.nvd.ProductOntology` - This class describes the ontology that is used during the data conversion process.
  - `edu.utdallas.nvd.RelationalToTripleStore` - This class reads the relational database and puts the data into the specified triple store.
  - `edu.utdallas.nvd.RelationalToRDBTripleStore` - This class gives a Jena RDB model for the data conversion process.
  - `edu.utdallas.nvd.RelationalToSDBTripleStore` - This class uses a Jena SDB model for the data conversion process.
  - `edu.utdallas.nvd.RelationalToAllegroGraphTripleStore` - The class provides an AllegroGraph triple store for the data conversion process.
  - `edu.utdallas.nvd.QueryTripleStore` - This class provides the base class used to perform query execution tests on the various triple stores.
  - `edu.utdallas.nvd.QueryRDBTripleStore` - This class gives a Jena RDB model with the migrated data for query execution.
  - `edu.utdallas.nvd.QuerySDBTripleStore` - This class uses a Jena SDB model with the migrated data for query execution.
  - `edu.utdallas.nvd.QueryAllegroGraphTripleStore` - This class provides an AllegroGraph triple store with the migrated data for query execution.

- **lib folder** - contains the library files needed to run the utility.
• **build.properties** - This file provides the command line arguments needed by the utility to perform the migration process.

• **build.xml** - This file defines the programs used to perform data migration and query execution as Ant targets.

  – All the data conversion Ant targets below use the following parameters from the build.properties file for establishing a connection with the relational database.
    
    » ${relational.driver} = This is the database driver for the relational database.
    » ${relational.db} = This is the actual database name.
    » ${relational.username} = This is the username of the owner of the relational database.
    » ${relational.password} = This is the password of the owner of the relational database.

  – **run-rdb** - This Ant target is used to perform the data conversion from the relational database to a Jena RDB model. The parameters needed to run this script are given below as they appear in the build.properties file along with the meaning of that parameter,

    » ${rdb.driver} = This is the database driver for the triple store.
    » ${rdb.db} = This is the database name for the triple store.
    » ${rdb.username} = This is the username of the owner of the triple store.
    » ${rdb.password} = This is the password of the owner of the triple store.
    » ${rdb.type} = This is the database type of the triple store such as MySQL.
    » ${rdb.storename} = This is the name of the triple store.

  – **run-sdb** - This Ant target is used to perform the data conversion from the relational database to a Jena SDB model. The parameters needed to run this script are given below as they appear in the build.properties file along with the meaning of that parameter,

    » ${sdb.layout} = This is the layout for the SDB triple store.
    » ${sdb.type} = This is the database type of the triple store such as MySQL.
    » ${sdb.driver} = This is the database driver for the triple store.
    » ${sdb.db} = This is the database name for the triple store.
    » ${sdb.username} = This is the username of the owner of the triple store.
    » ${sdb.password} = This is the password of the owner of the triple store.
- **run-aagraph** - This Ant target is used to perform the data conversion from the relational database to an AllegroGraph triple store. The parameters needed to run this script are given below as they appear in the build.properties file along with the meaning of that parameter,

  * `$\{agraph.storename\}` = This is the name of the AllegroGraph triple store.
  * `$\{agraph.storelocation\}` = This is the location on disk of the AllegroGraph triple store.

- **run-query-relational** - This Ant target is used for query execution on the relational database view. The parameters needed to run this script are given below as they appear in the build.properties file along with the meaning of that parameter,

  * `$\{relational.driver\}` = This is the database driver for the relational database.
  * `$\{relational.db\}` = This is the actual database name.
  * `$\{relational.username\}` = This is the username of the owner of the relational database.
  * `$\{relational.password\}` = This is the password of the owner of the relational database.

- **run-query-rdb** - This Ant target is used for query execution on Jena’s RDB triple store. The parameters needed to run this script are given below as they appear in the build.properties file along with the meaning of that parameter,

  * `$\{rdb.driver\}` = This is the database driver for the triple store.
  * `$\{rdb.db\}` = This is the database name for the triple store.
  * `$\{rdb.username\}` = This is the username of the owner of the triple store.
  * `$\{rdb.password\}` = This is the password of the owner of the triple store.
  * `$\{rdb.type\}` = This is the database type of the triple store such as MySQL.
  * `$\{rdb.storename\}` = This is the name of the triple store.

- **run-query-sdb** - This Ant target is used for query execution on Jena’s SDB triple store. The parameters needed to run this script are given below as they appear in the build.properties file along with the meaning of that parameter,

  * `$\{sdb.layout\}` = This is the layout for the SDB triple store.
  * `$\{sdb.type\}` = This is the database type of the triple store such as MySQL.
  * `$\{sdb.driver\}` = This is the database driver for the triple store.
  * `$\{sdb.db\}` = This is the database name for the triple store.
* `{sdb.username}` = This is the username of the owner of the triple store.
* `{sdb.password}` = This is the password of the owner of the triple store.

- `run-query-agraph` - This Ant target is used for query execution on the AllegroGraph triple store. The parameters needed to run this script are given below as they appear in the build.properties file along with the meaning of that parameter,
  * `{agraph.storename}` = This is the name of the AllegroGraph triple store.
  * `{agraph.storelocation}` = This is the location on disk of the AllegroGraph triple store.
Appendix C

Semantic Web Application

The Semantic Web application was built around a triple store constructed with data migrated from NVD. The details of the design of the application as well as instructions on how it can be set up are presented below. Some screenshots from the application are also presented.

C.1 Prerequisites for application

Linux Ubuntu v8.04 Hardy Heron was used as the operating system to develop and deploy the application. Based on experiments in Sections 6.2 through 6.5 Jena’s SDB model was used as the persistent triple store. Some other components that need to be installed and configured are,

- Java version 1.6 (any release)
- MySQL 5.0.67\(^1\)
- Apache Tomcat Web Server 6.0.18\(^2\)
- Jena-2.5.6
- Web browser such as Mozilla Firefox\(^3\) (any version)

C.2 Application development and deployment

The Semantic Web application construction using Jena’s SDB model backed triple store is presented below. The application is constructed based on the Java J2EE framework\(^4\) using the Model-View-Controller (MVC\(^5\)) architecture.

\(^1\)http://www.mysql.com/
\(^2\)http://tomcat.apache.org/
\(^3\)http://www.mozilla.com/en-US/
\(^4\)http://java.sun.com/j2ee/overview.html
\(^5\)http://en.wikipedia.org/wiki/Model-view-controller
A brief description of each of the components of the MVC architecture used in the Semantic Web application is also presented below.

- **Model** - For the Semantic Web application, the model layer constitutes the triple store that contains the data that was migrated from NVD. The model layer also provides functions to take the query from the controller layer and convert them to SPARQL queries to be passed to the triple store and to return the result of query execution back to the controller layer.

- **View** - The view layer is constructed using Java Server Pages (JSP\(^6\)). The application provides the ability to search for products and agents using the search box. The application also provides direct navigation capabilities using the menu bar that contains products, agents as well as other filtering capabilities.

- **Controller** - The controller layer uses traditional Java Servlets\(^7\) to perform the function of taking the query from the view layer and passing it to the model layer as well as passing the results back from the model layer to the view layer.

To run the application, the web server needs to be started (Apache Tomcat in this case) and then the application can be started using any web browser.

### C.3 Application Screenshots

This section provides screenshots of the Semantic Web application in action. The caption for each screenshot gives the details of the functionality being provided with that screenshot.

- **Screenshot 1** - Homepage for the Semantic Web application
- **Screenshot 2** - Listing of the first five products from the SDB triple store
- **Screenshot 3** - Listing of the first five agents from the SDB triple store
- **Screenshot 4** - Listing of the particular features of a product
- **Screenshot 5** - Listing of the first five software products from the SDB triple store
- **Screenshot 6** - Listing of the products that use shared libraries from the SDB triple store
- **Screenshot 7** - Listing of the first five products that are application from the SDB triple store

\(^6\)http://java.sun.com/products/jsp/
\(^7\)http://java.sun.com/products/servlet/
• Screenshot 8 - Listing of the first five products that are operating systems from the SDB triple store

• Screenshot 9 - Listing of the first five hardware products from the SDB triple store

• Screenshot 10 - Listing of the first products that are TCP/IP devices from the SDB triple store
Figure C.1: A screenshot of the homepage that gives options to list products and vendors as well as to search for them using the search box.
Figure C.2: A screenshot of the listing of the first five products from the SDB triple store
Figure C.3: A screenshot of the listing of the first five agents from the SDB triple store
Figure C.4: A screenshot of a particular product’s features from the SDB triple store
Figure C.5: A screenshot of the listing of the first five software products from the SDB triple store
Figure C.6: A screenshot of the listing of products using shared libraries from the SDB triple store
Figure C.7: A screenshot of the listing of the first five products that are applications from the SDB triple store
Figure C.8: A screenshot of the listing of the first five products that are operating systems from the SDB triple store
Figure C.9: A screenshot of the listing of the first five hardware products from the SDB triple store
Figure C.10: A screenshot of the listing of the first five products that are TCP/IP devices from the SDB triple store