VIEW MAINTENANCE OF RDF STRUCTURES

Syed Muhammad Shehram Shah

201280660

This dissertation was submitted in part fulfilment of requirements for the degree of MSc

Advanced Computer Science

DEPT. OF COMPUTER AND INFORMATION SCIENCES

UNIVERSITY OF STRATHCLYDE

GLASGOW

SEPTEMBER 2013

DECLARATION

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc. Advanced Computer Science of the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a copy of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available
archive.

(please tick) Yes [ ] No [ ]

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices is. 18,993.

I confirm that I wish this to be assessed as a Type 1 2 3 4 5
Dissertation (please circle)

Signature:

Date

ABSTRACT

The Semantic Web is an extension of current Web technologies that aims to transform the way the Web functions. The ultimate goal of the Semantic Web is data integration and sharing of information on the Internet. Most of the present Web content is meant for human consumption, computers are unable to understand the meaning of the content. Semantic Web provides a complete framework for creating a new generation of Web content that is also intelligible by machines. At the heart of the Semantic Web is the RDF (Resource Description Framework) which has been recommended by W3C (World Wide Web Consortium) as the new universal data format for information interchange on the internet. RDF is a structured data format and solves the problems of existing data representation formats. The current practice of updating RDF data involves regenerating the underlying structure after modifications, which is expensive in terms of time and computational resources. As more and more Web content is created using RDF, there is a need for simple and efficient mechanisms to manage voluminous RDF structures.

This project aims to assess the performance of update operations on RDF structures through views. Views are a powerful way to manipulate a section of data from the underlying database. A view is subset of the underlying database, when changes are made to the underlying database; the views also need to be updated. RDF statements are made up of three components i.e. subject, predicate and object. In relational terms, RDF structures are three dimensional databases. The goal of this project is to assess performance of view maintenance of RDF structures i.e. to update RDF structures through views and measure performance of insert and delete operations with respect to the size of the RDF structures. A prototype application has been developed to measure the performance through experiments. The results show that as size increases, the time of insert operations increases proportionally where as for delete operations time remains stable.

Acknowledgements

I would like to acknowledge the cooperation and guidance of the following people who supported me during the course of this project:

I would first like to thank my supervisor, Dr. John Wilson, for his untiring support, guidance, personal attention and encouragement throughout the project. His patience and faith in me was a source of encouragement and enabled me to successfully overcome the most challenging phases of this project.

Dr. John Levine, course director of the MSc. Advanced Computer Science program, for his advice and support throughout my time at Strathclyde. His advice has been instrumental in steering me through difficult times.

I would like to thank the Commonwealth Scholarship Commission and University of Strathclyde for providing me the opportunity to undertake studies at this prestigious university in this great country.

I would also like to thank the faculty and staff of Department of Computer and Information Sciences, University of Strathclyde for their cooperation throughout my time at the university.

Finally, I would like to thank my parents for their continuous encouragement and love. I express gratitude to them for their care, understanding, and appreciation for my hard work which has always been a source of strength for me.

Muhammad S. S. Syed

September 2013

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction and Rationale

The Web is full of data. The magnitude of the data spread across the Internet is enormous. Human beings are not fully capable of processing this information manually without using any assistive technologies. Interestingly, the massive volume of data created over the web is also not understandable by machines. Computers can only interpret and process a fraction of Web content (rendering information) limited only for end user presentation. The growth of the Internet has led to creation of enormous number of documents and web pages. Some estimates suggest that currently the Internet consists of at least 3.5 billion Web pages (World Wide Web Size, 2013). The large size has come along with its set of problems. The data on the Internet is difficult to manage and is often redundant due to its polymorphic nature with no effective mechanisms to manage, share and organize data on the Internet; and hence the Semantic Web was introduced.

The Semantic Web was proposed as a revolutionary new way for the Web to function. Most of the data on the Internet today is meant for human consumption. A set of technologies have been proposed to create a new generation of data that can also be understood (at varying levels) by computers. It is achieved through RDF (Resource Description Framework) and its associated technologies which create data that is more structured than existing versions. Associated metadata is also generated alongside the main data, containing information about the intended meaning and use of the subjected data, thereby making it more meaningful to computers. The whole idea of the Semantic Web was to make the Web more intelligent by adding meaning (logic) to it. This dissertation is based on Semantic Web technologies and looks closely at updating RDF structures through views. The project is based on three main components. The literature review gives an assessment of progress of the Semantic Web and highlights core contributions. The software development component aims to develop an application to update RDF structures (described in detail in section 1.2). The last component of the dissertation involves evaluating performance of update operations on RDF structures of various sizes.

1.1 Research

We start by describing the principles of the Semantic Web and explaining the way it intends to change the Internet. This project involves a review of scientific publications to highlight key contributions towards the development of the Semantic Web. We look into some of the key research themes and application areas. We also shed light on some of the factors which act as a bottle neck in widespread uptake of the technology. The foundation of the Semantic Web is the RDF, proposed to be the new universal data representation format for the Internet, replacing current technologies like XML and HTML. We discuss RDF data representation format in detail and its advantages over traditional formats. Unlike existing technologies, which have been widely used over the Internet, Semantic Web technologies are relatively new and there is a dearth of skilled professionals to develop and maintain Semantic Web data and applications. The main advantage of RDF and RDFS (RDF Schema) is uniform representation format for creating structured data. This data can then be integrated with other sets of related data created by different sources. One of the main principles of the Semantic Web is universal data integration which is difficult to achieve with current technologies. The problem with current Semantic Web applications is the maintenance of the underlying data. The fact that the size of these collections runs into billions of triples and is shared amongst several applications over the Internet presents a challenge in the form of defining simple and efficient mechanisms for keeping this data up to date.

1.2 Application Development

RDF data is represented in the form of triples. In Relational terms, RDF structures can be referred to as three dimensional tables. RDF structures are typically extracted from a relational database into the RDF graph. This is effectively a view (abstract) of the relational database that has been materialized as a graph. This project involves development of a prototype application that aims to maintain RDF views directly without the need to regenerate underlying RDF structures. In the above context, the main research question is

- How View maintenance over an RDF collection perform and in particular, do inserts and delete operations perform in the same way?

In order to find out the above, we develop an application that is designed to measure the performance of update operations through views over RDF data and to analyse the impact of size of data on update operations. The software development process was conducted in a professional manner, focusing on all the four main stages (Analysis, Design, Code and Test) of the development cycle. As the main objectives of the project were based on the correctness of the application developed, it was important to ensure that the entire process has been completed properly.

1.3 Experiments

Once the application was developed and validated, experiments were conducted in the context of the main research question. The basic idea was to study view maintenance of RDF files. View maintenance involves updating the View in line with changes made to the database. Tests were run to evaluate the effectiveness of this technique and to study the impact of RDF file size on performance of update operations. The report discusses the experiments and the outcomes and reflects on them in the context of research.

1.4 Structure of Dissertation

The rest of the dissertation is presented as follows:

- Chapter 2 provides the background of the work done so far in the area of the Semantic Web since the introduction of the concept. It pulls together significant developments in the area and identifies major research themes. It also points out to some of the bottle necks and discusses the proposed solutions.
- Chapter 3 provides an account of the development process of the application. It contains sections about the general architecture, system development and data used in the experiments. It explains the detailed design of the application and appreciates the decisions made during the development activity based on the merits and demerits of the options available.
- Chapter 4 focuses on two main areas; it starts by describes the testing of the application and details the software verification process. The second section describes the experiments and their outcomes. It also provides analysis about the observations made from the experiments.

- Chapter 5 concludes the report with a set of observations made on the outcome of experiments. It includes a brief account on the approach taken for implementing the project. The recommendations also include directions for future work as an extension of this project.

Chapter 2
Literature Review

This chapter introduces the concept of the Semantic Web as proposed by Tim Berners-Lee in 2001 and gives the motivation behind Semantic Web. It also explains how it intends to change the Web. The word Semantics refers to the study of meaning, the term Semantic Web can be rightly described as a web (Internet) that tries to understand the meaning of its contents by focusing on the relationships with words and their attributed meaning. In other words the Semantic Web aims to make sense of the data through metadata (data about data) generated alongside the primary data. It is proposed that computers would be able to process (partially, at varying levels) a new generation of Web content to understand the meaning of data and its intended use. We pull together noteworthy research during the past decade to identify major advancements and possible bottlenecks for widespread uptake of this technology.

2.1 The Semantic Web

The Semantic Web was proposed as a revolutionary new way for the Web to function (Berners-Lee, et al, 2001). The whole idea was to make the Web more intelligent by adding meaning to it, essentially creating a very large integrated information system for the whole world. The idea of the Semantic Web has two main objectives; the first one is to enable data integration. The second one is to make data more intelligible for machines to enhance the Web experience of users. The goals, to be performed through means of Semantic Web applications, are interlinked in a manner that we cannot integrate data if machines do not understand what it means. At the core of the Semantic Web is the Resource Description Framework (RDF) which provides a mechanism to structure and represent data in a uniform way. In order to be able to integrate data we need to have an agreed upon, easy to use format for defining data. Additional metadata is generated alongside main data, containing information about its intended meaning and use, thereby defining Semantics to govern its use. The present generation of data on the Internet has been created by different entities using various representation formats, understood primarily by humans. Applications are only able to process a marginable part of it. The idea is to standardize content creation to remove ambiguity by introducing a uniform data representation format and vocabulary to create a new generation of Web Content and to also generate associated metadata along with the data to define its Semantics. This metadata contains information about the Semantics of data and can be used by applications to understand the data itself. Computers will be then able to process the data content to perform tasks more efficiently. This way it was envisioned the World Wide Web would become meaningful to machines.

As of now a majority of the content on the Internet is readable and intelligible by humans only as this information is designed mostly for human consumption. Web 2.0 allows users to share, create, process and manage information from different sources over the Web (Buffaa, et al, 2011). Machines do not really understand content on the Web and cannot process them completely. Currently, the Internet is a rich collection of disparate but similar or even identical information. Often the same thing is represented in different forms; this similarity is only recognizable by humans. Computer systems are unable to make the correlation between the two apparently different objects. The most important characteristic of the Web is its universality, which is a result of the decentralized way the Internet works. Typically users work their way through the Web by jumping to and from Web pages using hypertext links. One approach to solve this problem was the use of tags (folksonomies) to relate concepts together. It was due to tagging that similar Web pages were able to relate to each other which provide some level of interconnection of the content on the Web. Tagging has been for long used as a means of linking information (from different sources) together by identifying keywords or trigger words (Peters, 2009). Search engines are based on complex algorithms about page rank and other characteristics but are fundamentally based on tags. However this approach does provide a way of describing connections between Web pages but isn't a coherent mechanism to manage the Web and for grouping Web content. The tags provide a sort of informal connection to other similar document and content on the Web. Tagging has gained considerable success in online discussion forms or blogs, both technical and non technical, where people can post questions and answers and use keywords or tags to classify their work according to different themes. Tags are a form of Web content produced that can be processed by machines. Search engines widely use these tags and other similar features to return a collection of similar items to the user.

Efforts have been underway to further integrate the content on the Web using Semantic Web technologies (Shadbolt, Hall & Berners-Lee, 2006). However there is one fundamental obstacle towards this whole idea i.e. the way the Web has been brought into existence. The content on the Web is largely diverse and inconsistent in nature. Apart from being available in a range of different languages and formats, these documents are constructed and organized in different forms. Web Content is available from highly structured to barely structured format. Web systems have to deal with an asymmetric collection of data (in terms of representation formats) and provide as much information as possible to the user. Let us take the example of a search engine. A search engine is designed to return to the user a set of results (a collection of documents and other form of Web content) that it assumes to be relevant to the user (most likely to meet the user's requirements). Search engines work in a complex mechanism of recall and precision and return documents on the basis of page rankings. The pages are indexed and then matched against user queries. The more appropriate the pages are indexed the more likely they are to be returned to a user query. This is an example of how computers can process information and make the overall net experience easier for us. The Semantic Web aims to extend this concept and apply it over the entire World Wide Web to achieve an Internet that is more intelligent and that aims to support the user directly in performing specific tasks rather than just aiming to assist the user in achieving its purpose.

2.2 Semantic Web Technologies

In order to make the World Wide Web more coherent we need to have certain mechanisms in place to enable computers to create information that can be read (partially) by computers. The Semantic Web framework provides a complete set of technologies to transform the Internet as shown in Figure 1.

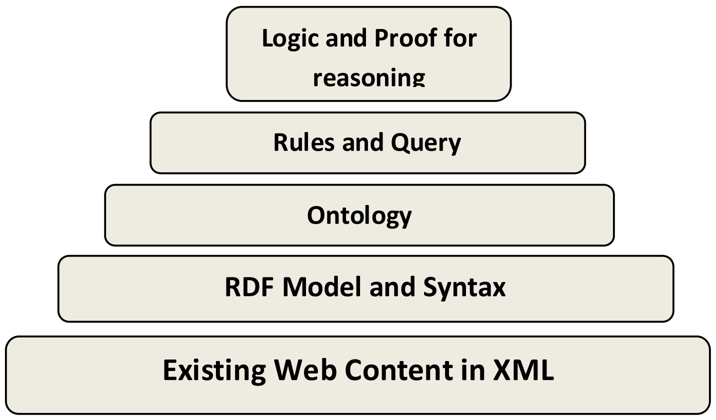

Figure1: Semantic Web Stack

The Semantic Web is an extension of current Web technologies. The above framework provides a mechanism for data integration and sharing and is built on top of existing technologies. The first step is knowledge representation i.e. to create knowledge based on a common format and to give some structure to it. It involves defining common vocabulary to describe concepts about a particular domain unambiguously. The second step is to create a mechanism for information to be shared over the Internet (Semantic Web applications and Ontologies) and the final step is to create inference rules on top of this data so computers can process and reason with the information to discover data that has not been stated explicitly. Knowledge Representation has its roots in Artificial Intelligence (Brooks, 1991). It is based on common definition and understanding of concepts by everyone creating information on the Web. This shared approach towards defining information on the Web would automatically give it a structure. Having the information representation sorted out, we can then create knowledge from data based on some facts and Figures using inference rules on top. In traditional knowledge representation systems the ever growing size of information makes it difficult to manage centrally. Such systems are limited in scope and are able to cater to a certain problem domain (Berners-Lee, 2001). These systems (however robust) are designed to answer a finite number of questions. There are also concerns about the reliability of the system to answer certain type of questions. Such systems are highly complex, i.e. the complexity increases with the effectiveness of the system. However every system regardless how complex, is capable of answering a finite number of questions based on the inference rules that govern the knowledge contained within the system. There always will remain some questions that the system is not able to answer reliably or even incapable of answering at all. Scientists name such kind of questions as paradoxes. Semantic Web acknowledges the concept of paradoxes while promoting the idea of representing information as expressively as possible. The basic motivation for that is the specification of information in a uniform format will allow a clear understanding allows computers to process this information and reason with it. The main challenge is to define a language that allows for clear, unambiguous representation of data and associated metadata, so it can be processed by machines.

There are a number of technologies that can be used to express content in a structured machine readable format, for example Extensible Mark-up Language (XML) and RDF (Resource Distribution Framework). XML allows users to create custom tags for representing data. These tags can be processed using browsers and scripts. The underlying structure of all XML documents is a tree. Although this structure does have a meaning i.e. XML but does not give users the provision to describe the meaning of their data or structure. The tree structure of XML documents is unintuitive and it is not suitable for the Semantic Web. Although ID/IDREF provide the capability of extending it to provide graph like structures but it does not solve ambiguity issues.

RDF is a semi structured data model to define data and is at the core of Semantic Web. It has been proposed as the new universal data format for information interchange on the Internet (Lassila & Swick, 1999), (W3C Recommendation, 2004). Data in RDF is represented in the form of triples i.e. subject, predicate and object. A RDF triple contains information about entities (subjects and objects) and the relationship between them (predicate). Every component of a triple can be specified using XML tags, literals or URIs (Universal Resource Identifier) which identifies uniquely a resource on the Internet. We can describe subject-property-object using RDF in the following terms, the subject and object point to a resource on the Web and properties can be described as attributes of resources. Every subject has an object and has a relationship between them. In other words we can say that RDF describes properties between two entities. By specifying information in the form of triples, we can identify the stated information based on the URI or URL, hence making the description of the triples to be available for access on the Web by everyone (Berners-Lee, 2001). This allows us to define information in a form that can be easily used across the Web and does not suffer from the ambiguity involved with XML. One of the main problems with XML was the lack of standard convention for describing an object, leaving the possibility of similar data to be described in multiple ways. Machines, unlike humans were not able to understand the similarity between them. RDF solves two main problems of XML; it provides a uniform format for creating data and also provides a more suitable architectural structure in the form of a graph. Figure 2 shows a RDF triple represented in the form of a graph. The subject is represented by a URI while the predicate is denoted as an arc and the object is represented by a literal.

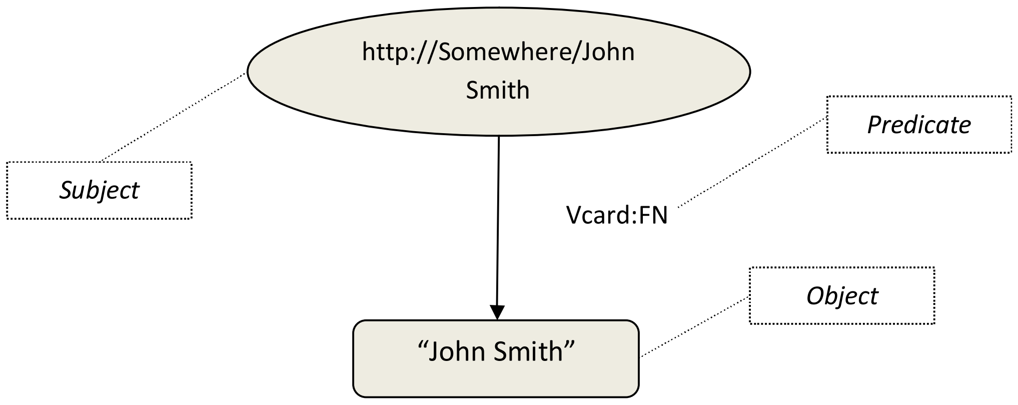

Figure 2: RDF Graph (Apache Software Foundation, 2011)

After having a uniform data representation format, we need a more advanced technology that could provide a schema to the data we have created. RDF does provide us with a structure to represent data in a form of triples but it does not allow us to define hierarchical relationships between resources and linkages between a resource and another. It only allows us to create a RDF triple. It however does not provide mechanisms to describe these properties or the relationships between properties and other resources. Considering the diverse, on the fly nature of RDF, use of this data is a challenge in the absence of a schema.

RDFS (Resource Description Framework Schema) complements RDF triples to tell us about the meaning of information. This is the metadata that defines its Semantics. It borrows the concept of inheritance from Object Oriented Paradigm (OOP). Inheritance is one of the three main principles of OOP and is the concept of defining the hierarchical relationship between two entities. It describes the relationship between classes used in RDF triples. Using RDFS we can group related resources and also identify the relationship between them. We create classes and properties similar to object oriented paradigm but the main difference is that these relationships apply to the classes rather than their instances. Each property has a range and domain whose value is a resource. For e.g. the property eg:isfather has domain identified by a resource eg:person1 and domain by a resource eg:person2. RDFS is also referred to as RDF vocabulary description language (W3C, 2004). It provides a set of standard vocabulary that is well represented and well understood by everyone thereby allowing shared understanding of concepts in a domain thus removing any ambiguities. Using RDFS we can create several layers of information by identifying relations between resources and properties. The relationship between resources is defined through properties between them and their values. However RDFS too has certain limitations. We cannot define cardinality and the scope of properties between the resources nor can we restrict the range of properties.

RDF and RDFS allow us to structure data and define hierarchical relationships regarding elements of that data. It may work for data created by one source. However given the universality of the World Wide Web, the same information can be specified quite differently over the Internet i.e. the same concept has been defined using different identifiers. This problem is solved by Ontologies. Ontologies have been defined as a formal specification of a shared conceptualization (Gruber, 1994, Gruber 2009) and enable us to associate similar pieces of information with each other. They are a means of specifying that two particular terms point to the same thing or are the same thing. Ontologies provide a common understanding of concepts. Information is described using classes, sub classes, properties and relations in Ontologies. They also contain a set of inference rules, which govern the use of information, to be used by machines to discover new information (not stated explicitly) and make deductions about data based on these rules. Ontologies are created using the OWL (Web Ontology Language) language, like XML and RDF it can be processed by a computer program which manipulates information to deduce information that has not been specified explicitly, effectively adding logic to the Web (McGuinness & Van Harmelen, 2004). Users can create Ontologies of their information and use relations such as type-of, same-as, subclassof to link to other Ontologies. Using Semantic Web technologies such as XML, RDF, RDFS we can create Ontologies to define data and more importantly generate the metadata that supplements the structured data for applications to process it while understanding the

Semantics. These technologies add logic to the Web and make it much more useful for the user. After specifying a mechanism to structure information by describing its meaning, specifying the rules that govern it and also allowing relations to be identified between information across the Web, The next logical step is to create computer programs that can process and understand the structure of the information, understand what the structure means, indentify the relationship between entities and be able to make conclusions based on some rules using Ontologies. Such types of computer program are known as agents. Agents collect information from various sources over the Web, process it and exchange information amongst other agents over the Web, therefore truly utilizing the capabilities of Semantic Web (Berners-Lee, 2001). The Semantic Web can be utilized to answer questions that require tremendous effort by humans who use bits and pieces of information available over the Internet to create new knowledge. Adding meaning to the Web will make the Internet experience more enriching for humans by allowing them easy access to a whole new wealth of information (Shadbolt Et al, 2006).

This was the original vision of the Semantic Web however the current practice is somewhat limited in terms of its application. There has not been widespread uptake of the technology due to various reasons. Current implementations have been limited to specialized application areas. Given the current state of things, the proposed integration of data on the World Wide Web has not been achieved, atleast not on a universal level as originally expected. There are examples of the use of Ontologies in the area of biology, medicine and genomics. Efforts are underway to apply this approach to manage information in areas such as social networking, search engines, geo-physical, metrological applications. Some progress has been made in this direction by governments.

- An example of a Semantic Web search engine is Corese which uses Semantic Web technologies to search through structured data (Corby et al, 2004).
- Several experimental Semantic Web projects have been the creation of Semantic Wikis. A number of new Semantic Wikis have been developed by professionals (Schaffert, 2006, Buffa, 2011) gives an indication of a new trend of research.
- The UK government has developed an Integrated Public Sector Vocabulary for common use by stakeholders and also constituted the Office of Public Sector

Information which allows a treasure of information, gathered and created by the government, be used by the public (Shadbolt et al, 2006).

- Another significant Semantic Web project is DBpedia (www.wiki.dbpedia.org) which is part of the larger Wikipedia project. DBpedia aims to provide structured information of all the knowledge created and collected for Wikipedia. Wikipedia has proved to be a wildly popular platform for sharing of information between humans. The DBpedia project aims to enable computers to process and understand knowledge by representing the data in a more structured form. Users can access this structured information repository using SPARQL (SPARQL Protocol and RDF Query Language) and an endpoint.
- A major milestone in the path towards a widespread use of the Semantic Web was the formation of Schema.org in 2011 to improve search engine performance. It is a joint venture of major engines such as Google, Yahoo, Bing and Yandex to form a common collection of tags to optimize the precision and recall search engine performance. Google has been harnessing structure data to display intelligent snippets (Starr, 2012). Other major players have been following suit including commercial players such as Walmart and Amazon.
- Given the current state of information available on the Internet, traditional search engines have complex algorithms to find useful Web pages due to the sheer number of pages. In order to deal with this data in a more structured manner, some work has been done to create Semantic Search Engines (SSE) that aim to link the Semantics of the query with the Semantics of the indexed documents, allowing computers to understand exactly what is required by the user (Renteria et al, 2010).

Despite these efforts and several others, the Semantic Web still falls short of expectations. Progress towards the transformation of the World Wide Web into the Semantic Web has been slow. International standardization organisations such as Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) have worked to specify mechanisms and standardize technologies such as RDF, RDFS, OWL, SPARQL, Rule Interchange Format among others that allow meaning and concepts to be shared (Shadbolt et al, 2006). Ontologies are at the top level of all Semantic Web technologies and allow information to be created and distributed across systems. Keeping in line with the evolving computing scenario which involves ubiquitous and pervasive computing, the Semantic Web is being explored to enhance the mobile computing environment i.e. enabling machines to understand the context and assist in completing tasks (Sheshagiri et al, 2004).

The Semantic Web aims to maximise the sharing and reuse of information. It focuses on combining and integrating related knowledge created all over the world. Proper implementation and utilization of Semantic Web will enable humans to add Semantics to existing data (Berners-Lee et al, 2001). By combining the knowledge pool of the world we would be able to save a lot of resources currently utilized to create information that is already created by someone else but is unknown to the user or is in another form. Exploiting its true potential depends on how widely this technology is taken. Current implementations of this technology have allowed un-paralleled sharing of information amongst the users in numerous areas as discussed above (Shadbolt et al, 2006). These projects have proved to be an important demonstration of the power of the Semantic Web.

2.3 Size of RDF Collections

In line with the objectives of our project, it was important to research the size of RDF collections and how they are managed. A single RDF collection can range from several hundred to billions of triples. A list of RDF collections and their size complied from RDF data dumps available at the W3C website has been shown in Table 1. Such mushrooming growth has raised concerns about performance issues of applications that rely on RDF content. Large data collections based on RDF are difficult to maintain and update as current practices mostly involve regenerating the entire data collection after modification operations. Some of the initial applications that allowed retrieval and maintenance of RDF structures have become increasingly troublesome with the increasing size of triples (Alexaki et al, 2001). Languages such as SPARQL (SPARQL Protocol and RDF Query Language), RDFQL (RDF Query Language) and RQL (RDF Query Language) among others have gained prominence as the language of choice by researchers for querying RDF data. These languages are declarative and are similar to SQL.

Table 1: Size of RDF collections (W3C RDF Dumps, 2013)

Project Name	**Approximate Size of Data Set**
Data-gov Wiki	5+ billion triples
Billion triples Challenge Dataset 2010	3.2 billion triples
Bio2RDF	2.7 billion triples
Linked Sensor Data	1.7 billion triples
Billion triples Challenge Dataset 2009	1.14 billion triples
Billion triples Challenge Dataset 2008	1 billion triples
U.S. Census data	1 billion triples
UniProt	300+ million triples
DBpedia	247 million triples
World Bank Linked Data	160 million triples
Wikipedia	47 million triples
RKB Explorer Data	60 million triples
GovTrack.us	13 million triples
LinkedCT	9.8 million triples
TaxonConcept Knowledge Base	8.2 million triples
LinkedMDB	6.1 million triples
GeoSpecies Knowledge Base	1.9 million triples
U.S. SEC data	1.8 million triples
OpenCyc	1.6 million triples
Jamendo from DBtune.org	1.1 million triples
British Geological Survey (BGS)	840000 triples
Airport Data	754585 triples
BBC John Peel sessions from DBtune.org	277,000 triples
OSM Semantic Network	130,000 triples

Researchers have been exploring ways to improve performance of RDF queries. Some of them have proposed entirely new languages to retrieve data from RDF structures, others have proposed using parallel processing by partitioning extremely large files over several machines, efficient query algorithms, join processing, indexing and to apply hashing techniques for faster access of data as ways to enhance performance of queries (Chong, 2005,) (Neumann & Weikum, 2009). Similarly, efforts have been underway to benchmark query performance in RDF structures (Neumann & Weikum, 2008). Performance considerations have also been a major factor as more and more content on the Web in proliferated by the RDF. The increasing size of RDF data also raises challenges for content management and maintenance. One of the main reasons for this is the transformation of more and more Web content into RDF which is expected to grow exponentially in the coming years. Several search engines have designed mechanisms to process Semantic Web (RDF) content as part of their normal searches. RDF data management systems have also been introduced to allow for a proper way to edit, share and manage RDF data(Neumann & Weikum, 2008).

2.4 SPARQL/SPARUL

SPARQL is a W3C recommendation for querying and manipulating RDF structures. It is a declarative language (need to describe control and function step by step) with a structure similar to SQL. Most Semantic Web applications use RDF query languages such as SPARQL to retrieve data from triples. SPARQL provides a very robust set of features that allow us to include options based on conjunctions, disjunctions and patterns in a query. It has four different modes of operations that provide a variety of outputs. These are the SELECT form used to extract data, another form is CONSTRUCT which can be used to construct a RDF graph, the ASK statement is the most simplest of them all which returns a Boolean value based on the query. The last one is DESCRIBE which is used to generate an RDF graph. The most common of these is the SELECT statement which is used to query data. There have been several experiments which evaluate the performance of SPARQL queries and several options have been suggested to improve SPARQL performance (Chong et al., 2005), (Neumann & Weikum, 2008), (Neumann & Weikum, 2009).

SPARQL can be used to perform a variety of functions on RDF data, its update language SPARUL (SPARQL Update) can be used to update an RDF structure. We can use SPARUL to describe, communicate and store changes to a remote RDF store (Seaborne et al, 2008). It is an extension of SPARQL and provides the ability to modify an RDF document. Modification is performed through add and delete operations. There is no mechanism to 'alter' a triple by 'editing' its components. W3C recommendation indentifies the main capabilities of SPARUL as follows, insertion of triples to an existing RDF structure, deletion of triples from an existing RDF structure, performing several (group) modification operations as a single task, creating a new RDF structure and deleting an existing RDF structure from the database (W3C SPARUL, 2008).

2.5 SPARQL Implementations

SPARQL has been used to develop applications to provide a framework for easy use and maintenance of RDF structures. These are similar to the concept of RDMBS (Relational Database Management Systems) and often referred to as RDF engines. (Neumann & Weikum, 2008) developed an RDF engine called RDF: 3X (RDF Triple Express) to query and manage RDF collections. It is proposed to be a fast and effective RDF data management system. RDF: 3X is based on the RISC (Reduced Instruction Set Computing) paradigm, designed to be light weight in terms of the application itself including the algorithms for querying, manipulating and processing RDF content. It defines four major factors that affect the development of efficient and effective RDF Engines. The first one is the flexibility that the RDF does not restrict us with any schema constraints as schema is provided by RDFS. We cannot use automated mechanisms for the physical data design of the application. The second one is about the diversified nature of RDF content which also is a limitation in this regard. Complex queries traversing multiple predicates in large RDF collections take a higher toll on the performance as connections are made at the fine grain level i.e. the RDF triples rather than the file. The nature of join condition varies from query to query. This poses a challenge for the choice of query processing algorithms that may be used at the application level as performance considerations attract more importance with the increasing size of RDF collections. The third consideration for the development of any application is the possibility of evaluating performance.

The performance of RDF: 3X is based on three key design parameters. As a first step towards improving performance during execution of queries, the application builds indexes of the collection on the basis of all possible permutations of RDF triple collection. Indexes are also created by count aggregating existing indexes. To put simply, indexes in multiple configurations are created and allow for faster access of the data. This removes the need of an auto tuning wizard to handle the physical design tuning required to manipulate raw RDF collections. These indexes can be compressed to keep the size to a minimum. The second factor is an efficient query processor (RDF engine), as the exhaustive indexes created in the first stage only need to be manipulated by using join conditions. Due to the robustness of the indexes, the actual RDF triple collection does not need to be accessed for a majority of queries. The third factor is the operation of the query operator, which is designed to optimize queries and formulate execution plans (Neumann & Weikum, 2008).

Virtuoso is a very popular system that provides data management capabilities for data based on a variety of formats such as relational, XML, object oriented databases, object relational databases and even RDF databases. It provides data integration services as it can work with external databases also. This approach fits well in terms of the Semantic Web as data can be used from external sources (Erling & Mikhailov , 2007).

2.6 RDF Views

Views are a logical subset of data and have been extensively used on existing data management systems based on XML, object oriented and relational format (Ceri & Widom , 1991). It is a matured technology and is focused on the use of data rather than its storage and management. It provides an extra layer of functionality on top of the database and there has been extensive work regarding the creation and maintenance of views on traditional data management systems. Researchers have been experimenting with implementing views on RDF data and have proposed algorithms and techniques to optimize data manipulation operations on RDF through views. However the main difference in creating views on RDF data is that RDF is a schema free data format (schema is provided by RDFS) where as XML, object oriented and relational databases adhere to a fixed schema. In these databases the views are essentially stored queries and the output of these views is in the form of a multi dimensional array where as this is not true for RDF.

W3C (World Wide Web Consortium) has recommended RDF as a language to be used for creating Web content in terms of its goal of Web integration. In order to realize the vision of the Semantic Web it is very important to be able to manipulate data to suit the requirements of all the applications as different applications require data in different format all varying in terms of detail, scope etc. Views provide an excellent way to utilize RDF data to enable it to be used by applications. Views provide the flexibility and power to applications to extract information from the Web and manipulate it according to their own requirements. Security has always been a major concern when data comes from external sources or when external parties require access to the data. The true potential of a database can only be exploited if it provides the capability of creating views. This has been proved by its widespread use in RDBMS (Relational Data Base Management Systems) and OOBDMS (Object Oriented Database Management Systems). Views are an established technology and serve multiple purposes such as presenting data in different formats without manipulating the original data. Data is retrieved from views instead of the tables thereby restricting access to the base tables thus providing another layer of security. Views also provide the capability to control the level of access to the database, instead of providing access to the complete table or database we can control the level of access by horizontally partitioning the database. Views are a perfect way to present data in a customized form without modifying any of the sources, they also allow for integrating data from multiple sources. They are essential to utilizing the full potential of any database as they have significant implications in terms of security and performance. The performance considerations regarding large RDF databases have been discussed earlier in this report.

(Volz et al, 2002) introduced the concept of creating views on RDF data. Their work was based on RQL (RDF Query Language) which is a declarative language for querying RDF graphs. There have been several different applications such as Virtuoso (Erling & Mikhailov, 2007) and Triple Store that support the storage and maintenance of RDF databases. Data can be retrieved from these data bases using SPARQL which is a powerful data retrieval language designed to work with triples. It functions differently than standard RDBMS query languages as its mode of operation is through graph traversal and based on predicates. SPARQL has been specifically designed to work with RDF data.

View maintenance refers to updating of queries that make up the View after the underlying data source has been modified. Maintaining RDF views is different from maintaining XML or relational views, primarily because of the fundamental difference of the structure of the databases. XML data is stored or represented in the form of a tree and has nodes, while relational data is stored in the form of tables. However RDF data is represented in the form of a graph. RDF data is completely different from XML and relational data hence RDF Views need to be managed quite differently than existing practices. (Erling & Mikhailov, 2007) has come up with a comprehensive approach towards creating, maintaining and updating (insertion, deletion and modification operations on) RDF views. He has proposed a set of algorithms for insertion maintenance, deletion maintenance including algorithms for modifying triples and resources. These experiments were based on RDQL (RDF Data Query Language). RDQL like SPARQL is a query language developed by Hewlett Packard (Seaborne, 2003) as part of efforts by industry and academia to develop a sustainable and robust Semantic Web. This language has been used in a number of projects for manipulating RDF databases. The syntax of this language is quite similar to SQL by following the SELECT-FROM-WHERE structure. The mode of operation is graph traversal of the triples through pattern matching.

2.7 Relational Approach

In Relational terms, RDF structures can be viewed as a large three dimensional table. Another approach towards working with RDF databases is to transform the triples into a relational database. Applications such as Virtuoso support storing RDF into tabular form. This technique has been used widely during the initial stages of RDF structures (Chong et al., 2005). It provides users with a way of working with RDF data due to the popularity of RDMBS and relative expertise and skills of professionals. This approach too has its roots from the transformation of XML data into the relational structure. An RDF triple has three main components i.e. Subject-Predicate-Object also referred to as Resource-Property-Value. The idea is to create views from the now created relational table and perform insertion, deletion or modification as required by using existing mechanisms with respect to relational views. In other words, the data can be updated through views. The addition of a new record into the database through the view would effectively mean adding a new tuple into the table and similarly other data manipulation operations. If the underlying database has been modified then the view based on the database has to be modified as well. This process is called view maintenance and is shown in Figure 3.

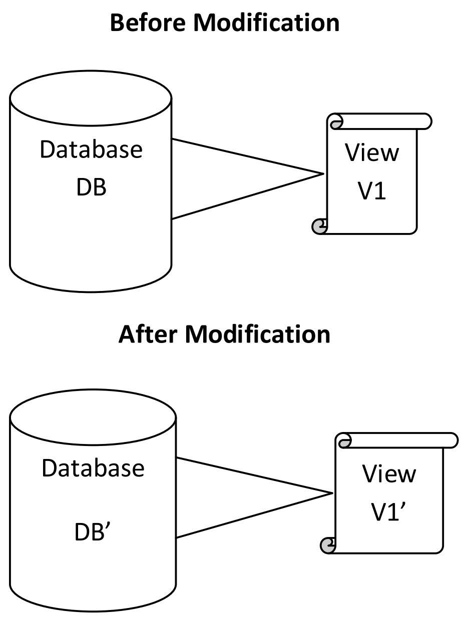

Figure 3: View Maintenance

2.8 Summary of Chapter

The aim of the Semantic Web is to add Semantics to the Internet. The Semantic Web framework provides a complete suite of technologies to interlink and integrate data. At the lowest level this has been achieved by RDF. It has been recommended by W3C as a universal data interchange format and is at the core of the Semantic Web. It overcomes all the problems of existing data representation formats. The size of some of the RDF collections is already in the number of billions of triples and it is expected to grow further. The need for an efficient mechanism for management of large RDF structures has given rise to a number of approaches towards maintaining and generating RDF data. RDF structures are typically generated from a triple store (RDF engines). As data changes over time, these structures need to be updated. One approach is to regenerate the entire file after modifications; however this is inefficient and expensive (computationally) for large collections. We explore the possibility of maintaining RDF structures directly through views without the need of regenerating the complete structure.

Chapter 3

Methodology

This chapter presents the Methodology in three sections describing the overall problem, the System Design including the software development process and the Detailed Design of the application.

3.1 Problem Description

The Internet was created to allow sharing of information between people and entities. The idea of the Internet was based on Internetworking of entities on the World Wide Web. Even though the Internet has seen unprecedented growth during the last two and a half decades, it still could not really provide a framework to integrate data as a whole. This is due to the lack of a uniform mechanism in current Web technologies to standardise the interchange of information. Ever since the advent of the Internet, technology has evolved to provide more efficient ways to exchange information over the Web, the most noteworthy being the HTML and XML. The most widely used data interchange format on the Internet today is XML, which provides a standard for entities to exchange information between each other which can be then used for further processing by applications on the respective host machines.

Java technologies such as JavaServer Pages (JSP) have also played a very important role in making the Internet a very robust and resourceful platform for global commerce and information interchange. The Internet has revolutionized the way people live and do things; it has rendered physical distances and boundaries meaningless primarily due to the versatility of Java technologies locally and on the Internet. However the mushrooming growth of the Internet and the number of Web pages available has come along with its problems. The enormous size of the Internet makes it difficult to manage the information within it. The second main problem is that a considerable amount of Web content is redundant due to the ambiguity in the representation mechanisms.

3.1.1 Modelling Problems with XML

XML is the most widely used data interchange format on the Web. It provides a standard language for representing information and exchanging it between entities on the World Wide Web, however its main drawback is its ambiguity. XML provides us with HTML like tags to represent information in the form of a tree. The tree contains data as well as metadata about the tree. However the metadata, represented in DTD (Document Type Definition) and XML Schema, governs the items permitted in the tree but it does not contain any information about its meaning. Applications can then manipulate the tree structure to perform user specified tasks. Even though XML is a standard format, it does not describe a uniform representation of similar entities, there is no standard way to define a piece of information. It lacks mechanisms to define the meaning of its contents. XML tags are represented arbitrarily in the tree with no meaning. Applications process the tree in their different manner which also effects how its meaning is interpreted by the application. In order to further explain the ambiguity associated with XML, let us take the example of a piece of information about a horse race named 'Grand National' which was won by a horse named 'Thunder Bolt'. Based on as much information is available, the representation of the above information in XML would be something of the sort shown in Figure 4.

```
<race name="Grand National">
        <winner> Thunder Bolt </winner>
</race>
```

Figure 4: XML Representation # 1

However, someone else may represent the same information as shown in Figure 5.

```
<winner name=" Thunder Bolt ">
<race> Grand National</race>
</winner>
```

Figure 5: XML Representation # 2

The same information can be represented through two completely different XML formats. There is no way to associate the intended meaning of the tags with XML. The metadata only contains information for rendering (information about font size, colour and style) and other necessary information for presentation or processing by the application. The lack of a uniform structure for representing information on the Web has resulted in a lot of redundant information being created.

The second disadvantage of XML is its tree structure just like its predecessor HTML. The tree structure is not adequate for representing complex data. As data becomes more complicated, the resulting tree becomes complex and unintuitive. The various components of information about an entity are spread all over the tree which is a somewhat un-integrated way of representing information. It is not possible to link together equivalent nodes in separate trees using XML hence that is not optimal data integration. XML is fundamentally a tree, although ID/IREF provides the capability of extending it to graph structures but the underlying architecture remains same, by contrast the intention of RDF is that it is a graph. RDF data is represented as a directed graph. The nodes in the graph are resources identifiable through a URI. The arcs in the graph represent properties of resources. At the other end of the arc are literal values or any other resource representing the value of that property. The three components form the RDF triple in the form of subject-predicate-object; each can be represented using a URI.

The idea of the Semantic Web envisages global data integration by allowing entities on the Web to reuse and share information between one another. This is not possible within XML due to the problems discussed above. A new data representation format was required to address these shortcomings. RDF provides us with mechanisms to express information without ambiguity and allows the intended meaning of information to be expressed alongside (using RDF/S). It is designed specifically for creating Semantic Web content and provides a uniform way of representing data and its metadata (using RDF/S) in a structured manner that can be used by applications to understand the intended meaning of a piece of information. Data is easy to create using RDF and there are no schema considerations at this level. This paradigm is known as pay as you go (Nueman, 2008). Additional metadata can be easily appended with the original payload. It allows linking up of information, enabling it to evolve incrementally as new pieces of data are created or appended. The world is slowly but steadily realizing the power and the creation of a significant amount of RDF data. Researchers in the field of biology, medicine and physics have taken keen interest into the use of RDF for building knowledge bases to allow sharing and exchange of information. Other name worthy projects based on RDF are DBpedia and Freebase (Nueman, 2008). The entire knowledge base of Wikipdea is also based on RDF content. RDF along with its associated technologies like RDF/S and OWL (Web Ontology Language) provide a complete framework for creating unambiguous Web content in a uniform manner. It has been designed to be a universal machine readable Internet exchange format. Structured information is encoded in the form of a graph using RDF. RDF has been discussed earlier in detail in section 2.1. The Semantic Web functions over RDF structures and involves extensive sharing and reusing of information between entities over the Web using Ontologies. Ontologies are RDF/S and OWL structures that are formal representations of concepts in a particular domain; they are potentially complete knowledge bases about a domain and the relationship between its concepts. We can model relationships between various domains using Ontologies.

The Internet in undergoing a transformation towards the Semantic Web and Web content is being increasingly created in RDF format. The Semantic Web is designed to function in a tightly integrated manner involving significant sharing and reuse of information between Web applications. The newer generation of Web content is created in RDF/OWL. Semantic Web technologies provide a complete framework for seamless interaction between applications for information interchange. The size of RDF data created in some major projects is enormous; The Data-gov Wiki project aims to convert all public U.S. government data into Semantic Web format. At present the size of its data collections is approximately 7.3 Billion triples (Data-gov Wiki, 2010). Other similar projects too have the number of triples in billions as shown earlier in Table 1.

In terms of relational databases, RDF collections can be described as three dimensional tables. They are views of relational data. As information changes over time and needs to be updated, it is necessary that all the applications that use that data have access to the updated information in a timely manner. Currently these views are maintained by regenerating the data after modification, this practice is inefficient in terms of time and resources with significant implications on performance. It is important to explore the possibility of defining a mechanism that allows for direct update of RDF structures through views. In order words, to update RDF structures through a similar mechanism like view maintenance. The primary aim of this project is to assess the possibility of maintaining RDF structures through views and to analyse the effects of increasing file size of RDF data on update operations. We also explore the various ways of handling RDF files in Jena and also the options within the Jena toolkit to store and manipulate these files.

3.2 System Design

This section describes the overall software development process, the technologies used and the design of the application. It explains the approach taken during subsequent stages of the software development process and discusses the various options available to implement features. Some insight has been provided about the decisions made during the design, implementation and testing of the application based on their merits and de-merits.

3.2.1 Requirements

The application was developed in a professional manner with due diligence to Software Engineering principles. Every effort was taken to apply best practices throughout the development process. Initial requirements were elicited and refined before initiation of the design process. The proposed application is of experimental nature and has been designed in a manner to allow for changes to be incorporated easily.

The success of any software development project depends on the effort applied for obtaining proper requirements. A project initiated on sound requirements is completed smoothly within the estimated time and resources. As with any software development exercise, the requirements engineering process for this project was completed and requirements were reviewed and refined. The final set of requirements for the proposed application is as follows

- The system should read in three RDF files and store them into in-memory models.
- All models should be separate from each other.
- Insert operations (100 statements) should be performed on each model.
- Delete operations (100 statements) should be performed on each model.
- Time should be calculated for the above mentioned operations and recorded separately for each task for individual models.

- A separate time file should be created containing the time measurements including measurements for time taken to insert/delete one statement into the models.
- The modified models should be written into three separate RDF files.
- Information regarding execution stage of the code and time measurements should be displayed on the console.

3.2.2 Development Approach:

Various software development approaches could be applied for the development of the prototype application. Two software development approaches were shortlisted based on the nature of the project. The first option was to use the Waterfall model and the second option was to use the (Evolutionary) Iterative Prototyping model. The Waterfall model (shown in Figure 6) envisions conclusively progressing through each stage of the SDLC (Software Development Life Cycle) process before moving on to the next phase in the model. There is no mechanism for going back to modify the design once it has been finalized. The only way to incorporate change in the original specification is to redo the whole process. Since the nature of this project was experimental, and required frequent modification in design and code, the Waterfall model was deemed unsuitable for this project.

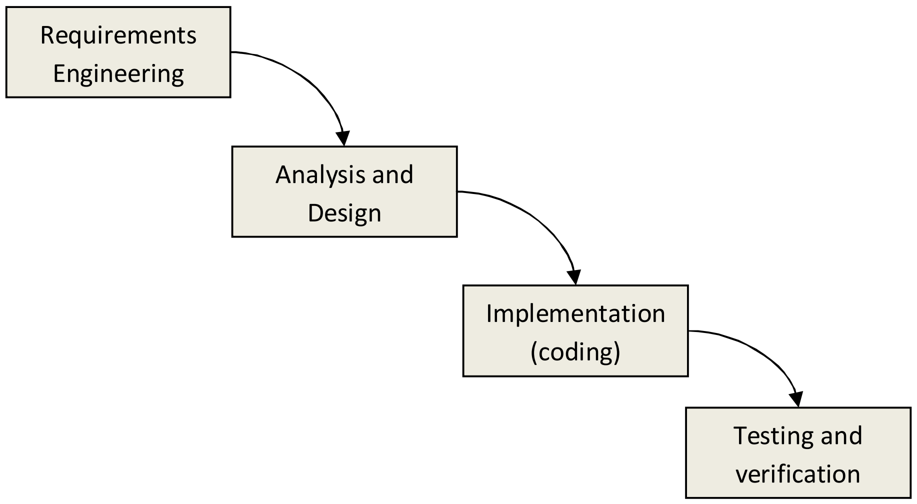

Figure 6: The basic Waterfall Model

The Incremental or the Evolutionary Prototyping model (shown in Figure 7) is based on the iterative development of the software. It works in cycles until the desired level of functionality in the application is achieved. Each cycle refines the existing code and adds new features to it. The iterative approach towards software development allows us to go back to the initial stages of the SDLC and make changes as required which fits perfectly with the requirements of application we are aiming to develop.

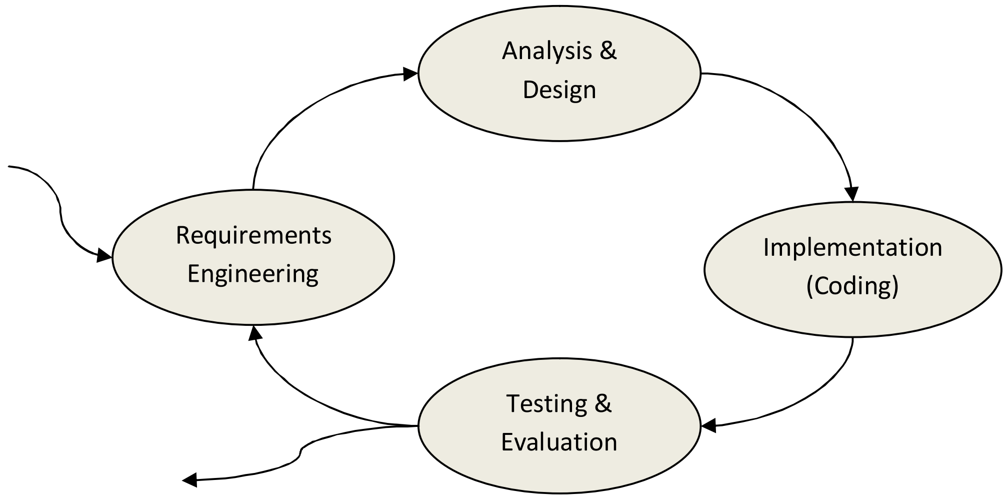

Figure 7: Incremental Model

After due consideration, Incremental or Evolutionary Prototyping methodology was chosen for the development of this application. Since the project involved developing skills in Jena, it was decided to proceed in a structured manner by step by step inclusion of functionality to the code in each cycle, after testing and validation of new and existing code the feature was expanded in line with the requirements. The end result of every cycle was a prototype, a complete subsystem of the application. The next cycle would involve addition of a new subsystem of the application to the code and so on.

There are 5 main subsystems of the application as follows,

1. Reading RDF files into Model
2. Insert Operations
3. Delete Operations
4. Generating Time.txt
5. Writing modified files externally

In terms of input-process-output the block diagram of the application is shown in Figure 8.

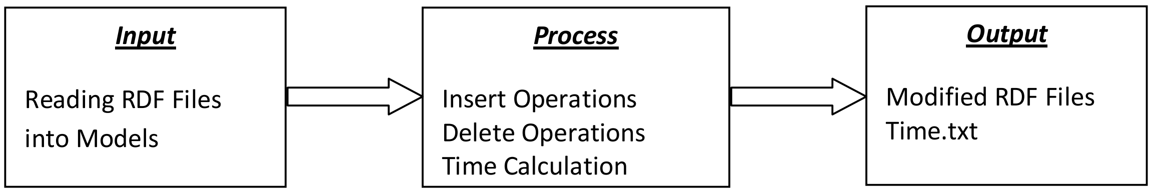

Figure 8: Block diagram of application

3.2.3 Architecture

The application reads-in RDF files of three different sizes and stores each file into separate in-memory models. The RDF files are then altered through insert and deletion operations applied through Jena methods. For this experiment we have chosen three RDF files with sizes of 1 MB, 5 MB and 10 MB respectively. The experiment proceeds with performing various update operations on these models and calculates the time taken to complete a task. The time is then recorded and analysed. The experiment and the analysis are discussed in detail in section 4.2. The general architecture of the application is shown in Figure 9.

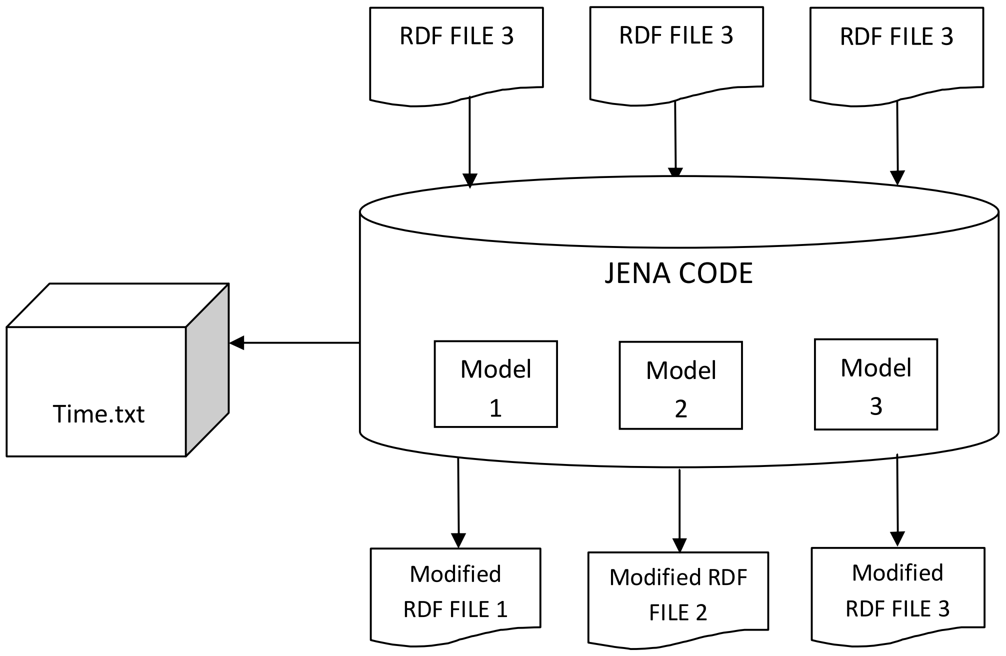

Figure 9: General Architecture of application

The application is based on a single class and makes use of the Jena libraries (discussed in detail in the next section). Jena libraries are not part of the standard Java package and have to be imported into the JVM. Eclipse Juno IDE has been used for developing the application.

The code has been kept as simple as possible to allow for uncomplicated modification as required. The system architecture in terms of the MVC (Model – View – Controller) paradigm is represented in Figure 10. The model is the data i.e. RDF files loaded into the in-memory models. Controller is Jena code that performs update operations on the RDF structures and records time. The View is the modified RDF files and the Time file created externally. The MVC approach for designing the application allows us to separate data, code and the output (as the model, view and control are decoupled), and it will allow us to modify the code (control) as per the needs of the tests to be performed during the experiments.

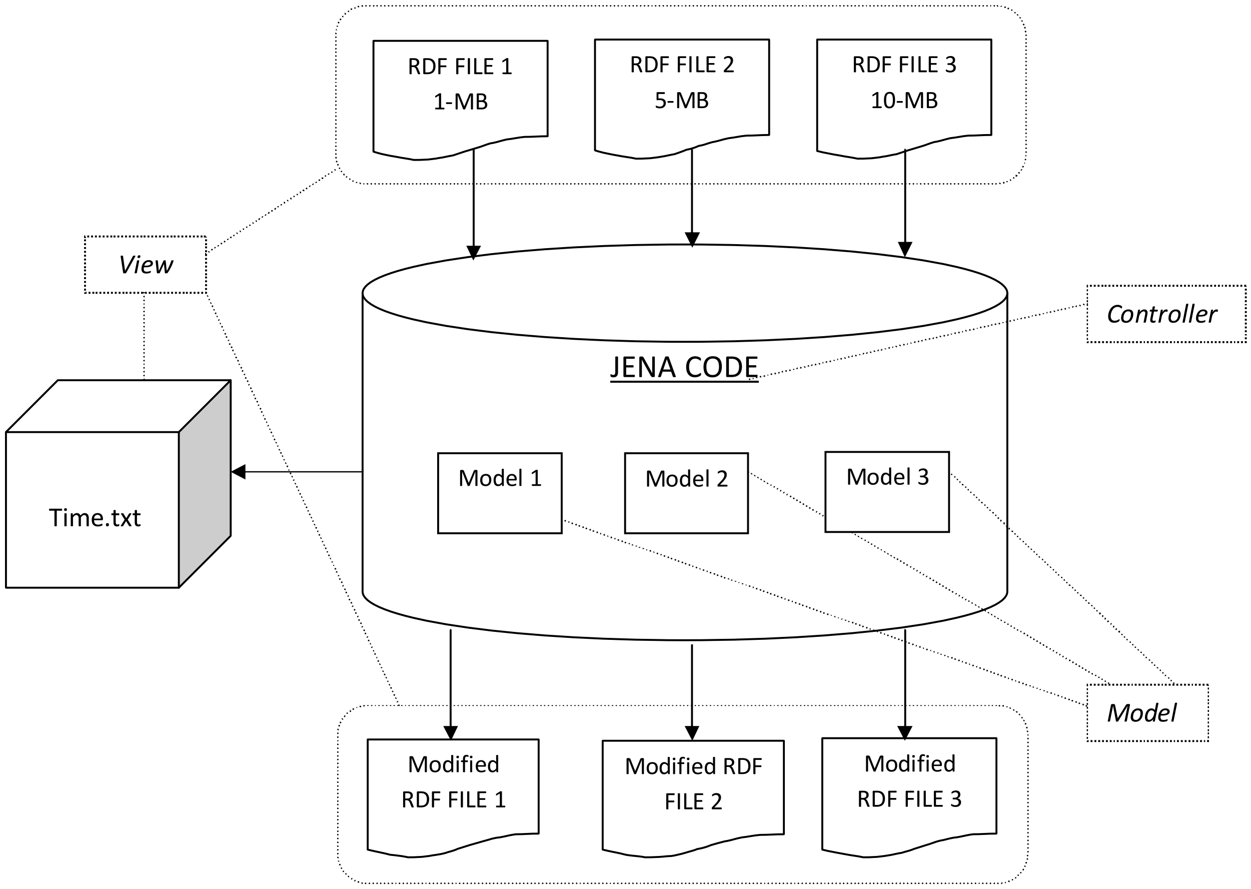

Figure 10: System Architecture in MVC paradigm

3.2.4 Jena

Introduced in 2000 by HP labs in Bristol UK, Jena is an open source framework for developing Semantic Web applications. In 2009 it was taken over by the Apache Software Foundation which supports development of open source software projects (Apache Jena, 2011). It is an extension of the Java language comprising of a set of libraries that allow programmers to create applications based on the Semantic Web model. These applications have the ability to interact with other Semantic Web applications and manipulate Semantic Web content. It also provides an extensive mechanism to handle RDF data through various data structures capable of storing and manipulating RDF structures. Application developers have the flexibility to use built-in methods or SPARQL/SPARUL queries to manipulate data. Jena provides various storage strategies (Graph, Model and OntModel) to store RDF structures, based on in-memory constructs or external files. The Jena framework consists of the following main components (Apache Jena, 2011).

- An RDF API that enables us to read, process and write RDF data in several formats (Turtle, N-triples and XML).
- An ontology API that handles OWL and RDFS Ontologies.
- A SPARQL API to support SPARQL/SPARUL queries.
- A rule-based inference engine for reasoning with RDF and OWL data sources.
- Stores to allow large numbers of RDF triples to be handled efficiently.

Figure 11 shows Jena's architecture. At the core of Jena is the RDF API which contains classes, methods and other constructs to handle RDF Triples. Each component of the triple can be represented using URLs, and each triple can be called a statement. The entire RDF can be represented using the Graph, Model or OntModel constructs (explained in next section). At the abstract level all RDF structures are stored as graphs. This API supports basic insert or remove (triples) operations on the graph and identifies triples complying with a particular pattern and even merges multiple graphs together. We can read in RDF files through a URL or an external file. RDF formats of N3, Turtle, XML/RDF are supported for both read and write operations. The Model interface consists of a range of methods that can be used to manipulate the structure without using SPARUL queries. It stores the RDF data into an in-memory model. We can use inbuilt methods of Model interface to perform a query or update operation on the RDF. The Graph interface is much simpler and does not provide additional features like the Model interface. The SPARQL API has been included in the framework to allow for executing queries of RDF structures within the application or using a front end for external sources. We can execute SPARQL and SPARUL queries directly on the graph. Both interfaces provide different storage strategies and flexibility to manipulate the RDF structure. The choice for using the Graph or the Model depends on the programmer as the same function can be performed on the data using either one. Additionally Jena provides the capability to store RDF files in a RDBMS, RDF database or in memory. The Inference API supports inference capabilities i.e. the ability to discover information, within the graph, that has not been stated explicitly. For example, if class C is a sub-class of class B, and B a sub-class of A, then by implication C is a sub-class of A (Apache Jena, 2011). We can use the Jena Inference Engine to discover this information and display it in the form of triples. It consists of various built in rule sets to infer information from the graph. However we can also use external reasoners based on Description Logic.

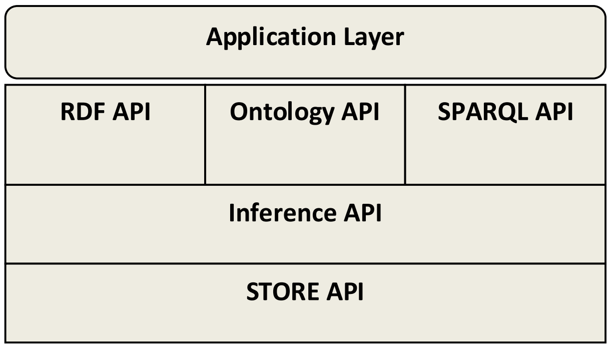

Figure 11: Jena Architecture showing all APIs (Apache Jena, 2011)

3.2.5 Model v/s Graph

Model and Graph are two interfaces in Jena that can be used to manipulate RDF files. They both allow similar kinds of operations but the mechanisms are different as they both operate at different levels. Virtually, Model and Graph form two layers of processing RDF files in Jena. At the abstract level all RDF files are stored as graphs. A model can simply be defined as a stateless wrapper around the graph with a set of convenience methods for added functionality. It adds an additional layer of functionality over an RDF graph through a range of methods and other constructs to simplify tasks. From an implementation point of view Graph contains basic functions for adding and removing triples. We can use embedded SPARUL queries on graphs. With Models we can use various methods to work with RDF Data including modification operations. Jena has three layers of functions and each layer complements the one below it, these are OntModel layer, Model Layer and Graph layer in the order of top to bottom as shown in Figure 12.

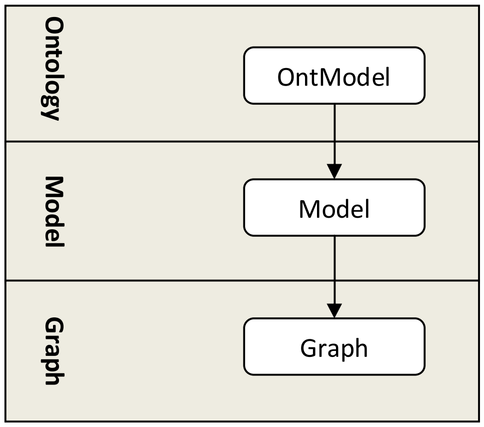

Figure 12: Three layers of operations in Jena (Apache Jena, 2011)

The Graph is at the lowest level with respect to implementation of RDF specification, it limits number of operations that can be performed on the RDF data using the Graph interface. It is at the granular level and allows limited flexibility to manipulate RDF data. However programmers can exploit the Graph to perform advanced operations using SPARQL and SPARUL queries. It does provide us with a powerful way of working with graphs at the lowest level. The Graph layer is also referred to as SPI (Service Provider Interface), its classes, methods and interfaces are designed by Jena development team to be used for enhancing Jena capabilities or to integrate Jena with other system components (Carroll et al, 2004).

The Model layer extends the core functionality of the Graph layer and provides a set of methods to provide additional functions. The Model layer was introduced in the second major upgrade to Jena, also referred to as Jena 2 (Carroll et al, 2004) in which the RDF specification implementation was decoupled from application level functions through the Model. Apart from providing additional methods, the Model layer provides the capability for Reification which is representing an existing triple into four triples to convey i.e. to represent the same information represented in one triple by a set of four triples known as reification quad (Carroll et al, 2004). This allows us to uncover information that has not been explicitly represented and express it in the form of triples. The third layer of operations is the OntModel layer that works with the inference engine for reasoning which is beyond the scope of this project.

To further explain the level of operation at the Model and Graph levels, an example is shown which retrieves the *VCARD.FN* property of resource *johnSmithURI*. The same operation is performed at the Graph and Model layers using their respective methods.

At the Graph level, a query would have to be created and stored in a query string. The query string would be executed via the ARQ query engine and finally the results would be displayed using the *ResultSet* class as shown in Figure 13.

```
String s2 = "SELECT ?a
             WHERE {
                    johnSmithURI VCARD.FN ?a.
                   } ";

Query query = QueryFactory.create(s2);
        QueryExecution qExe = QueryExecutionFactory.sparqlService(query);
        ResultSet results = qExe.execSelect();
        ResultSetFormatter.out(System.out, results, query) ;
```

Figure 13: Retrieving data from RDF structure through Graph Interface

At the Model layer, we can perform the same task by using methods provided by the Model interface as shown in Figure 14.

```
Resource vcard =
model.getResource(johnSmithURI);

Resource name = vcard.getProperty(VCARD.FN)
                     .getResource();
```

Figure 14: Retrieving data from RDF structure through Model Interface

Hence, the Model interface is much simpler to use and it provides programmers the option to create Semantic Web applications without needing to develop exceptional skills in SPARQL/SPARUL to manipulate RDF data. Due to the simplicity of the Model interface and the convenience provided by its methods to perform tasks easily, it was decided to use Model interface in our application.

3.2.6 Data

The test data has been based around the vCard Ontology, VCARD specification (RFC6350) has been mapped into RDF. It is a W3C working draft by the Semantic Web Interest Group (W3C SWIG, 2006) for describing people and organisations (W3C VCARD-RDF, 2013). The goal of this document was to promote the use of vCard ontology in Semantic Web applications to have a uniform way of representing people and organisations in line with existing representation mechanisms of non Semantic Web applications. The original vCard specification was developed my IETF (Internet Engineering Task Force). There have been subsequent upgrades to the specification, and the most recent one was version 4 (RFC6350). Apart from RDF, this specification has also been represented in text based descriptions and XML. A sample of the test data set is given in Figure 15. The vCard ontology has a comprehensive set of classes, subclasses, properties and modifiers that provide elaborate vocabulary to describe persons and organisations. This ontology can store complete bio-data including work and home information.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:vcard="http://www.w3.org/2006/vcard/ns#">

  <vcard:Individual rdf:about="http://example.com/me/corky" >
    <vcard:formattedName>Corky Crystal</vcard:formattedName>
    <vcard:nickName>Corks</vcard:nickName>
    <vcard:hasTelephone rdf:parseType="Resource">
        <vcard:telephone>tel:61755555555</vcard:telephone>
        <rdf:type
rdf:resource="http://www.w3.org/2013/vcard/ns#Home"/>
        <rdf:type
rdf:resource="http://www.w3.org/2013/vcard/ns#Voice"/>
    </vcard:hasTelephone>
    <vcard:email rdf:resource="mailto:corky@example.com"/>
    <vcard:hasAddress rdf:parseType="Resource">
        <vcard:streetAddress>111 Lake Drive</vcard:streetAddress>
        <vcard:locality>WonderCity</vcard:locality>
        <vcard:postalCode>5555</vcard:postalCode>
        <vcard:country>Australia</vcard:country>
        <rdf:type
rdf:resource="http://www.w3.org/2013/vcard/ns#Home"/>
    </vcard:hasAddress>
  </vcard:Individual>
</rdf:RDF>
```

Figure 15: Sample data from vCard Ontology

The vCard structure was used as the data source in the experiments since it provides a simple and easily readable structure to input. Other datasets downloaded and analysed for use in the experiments could not be read into the Jena models due to issues such as incompatible encoding formats. For the experiments, the ontology was expanded by creating new resources and property values. Three separate test data sets were created with sizes of 1 MB, 5 MB and 10 MB. As the experiment is designed to study the time taken to update RDF structures of various sizes, the content of these data sets are not the primary focus of analysis.

3.3 Detailed Design:

The design of the application is highly modular and strictly adheres to principles of Object Oriented Programming. Main aspects of functionality have been logically separated in separate methods. A resource folder containing test data has been included in the class path. The application does not have a graphical user interface; however information is displayed on the console regarding various execution stages of the code. This section explains the implementation of the features of the application in a sequential manner.

The application reads in three RDF files and stores each into instances of the model. The *readFiles* method reads-in RDF files and stores it into in-memory models. Figure 16 shows the *readFiles* method.

```
void readFiles (String inputfilename, Model modelname) throws IOException
        {
         InputStream in = FileManager.get().open(inputfilename);
           if (in == null) {
                            throw new IllegalArgumentException ( "File: " +
inputfilename + " not found");
           }
           modelname.read(new InputStreamReader(in), "");
           in.close();
             }
```

Figure 16: readFiles Method

All three test files are passed into the three separate models. An example of reading in the first test file is shown in Figure 17 where the first file is being read into Model 1.

```
UpdateRDF UpdateRDF1= new UpdateRDF();
Model model1 = ModelFactory.createDefaultModel();
UpdateRDF1.readFiles(inputFileName1,model1);
System.out.println( "== File 1 Read ==" );
```

Figure 17: Reading file 1 into Model 1

After the files have been read into the models, insert and delete operations are performed on the models and elapsed time is recorded for all the models. The same is repeated for delete operations on all three models. For calculating time there were several options available. The first one was to decide between elapsed time and CPU (Central Processing Unit) time. CPU time records the time taken by a computers processor to complete a set of instructions. It does not take into account time taken by any additional task such as an I/O operation. As CPU time for every machine is subject to the threading mechanisms utilised by different processors, it was decided to opt for elapsed time. Elapsed time is a simple measurement of time taken for a process to complete.

Elapsed time can be calculated in numerous ways in a Java program. There are several classes, such as *System* and *Calender,* which provide multiple methods for recording time. It was decided to use the *System* class through which we can query system parameters of the JVM (Java Virtual Machine) running on the host machine. The *getCurrentTimeMillis()* method returns the time calculated in milliseconds. However after running some initial tests using this method it was observed that the time readings were not precise enough due to which *nanoTime()* method of *System* class has been used to calculate time. This method returns highly precise time values in nanoseconds. Time is recorded before the loop starts and after the iterations complete. It includes both the time for creating the Statement and time to add/delete the statement into the model. Time for insert and delete operations is recorded and displayed on the console and is also stored separately in the *Time.txt* file which is generated externally. An example of insert operations along with the time calculation is shown in Figure 18.

```
long startAddTime1 = System.nanoTime();
   for(int i=0; i<=100; i++)
       {
             Statement S1 =
ResourceFactory.createStatement(model1.createResource("http://somewhere
/ShehramShah/" + i ),VCARD.FN,model1.createLiteral( "Shehram Shah" ));
       model1.add(S1);
       }
       long endAddTime1 = System.nanoTime();
       long TimeTakenAdd1=endAddTime1-startAddTime1;
```

Figure 18: Recording time for insert operations

Statements are added to the models using the *model.add()*method. A statement is created using the *createStatement* method of *ResourceFactory* class and is passed as the argument of this method. There are three arguments of this method, resource, property and value. As we are inserting a completely new statement into the model, we are also creating a new resource named '*http://somewhere/ShehramShah/*' using the *createResource* method. The statement to be added uses an existing property of the vCard ontology, *VCARD.FN,* which represents the full name. The last argument of the method is the value of the property which can either be another resource or literal. For this case we are specifying it as a literal by the string *'Shehram Shah*'. As discussed earlier, the contents of the RDF files or contents of the statements added into or removed from RDF files are not of interest with respect to the nature of this project. Hence the insertion and removal operations have been simplified by using a *for* loop. The *createResource* method creates only the subject of an RDF triple where as the *createStatement* creates a complete RDF triple, referred to as a statement in Jena when operating on the Model layer.

The delete operations are performed through the same mechanism as applied for the insert operations. The *model.remove()* performs the inverse of the *model.add()* method. An example of removing statements is shown in Figure 19.

```
long startDelTime1 = System.nanoTime();
    for(int i=0; i<=100; i++)
       {
model1.remove(model1.createResource( "http://somewhere/ShehramShah/" +
i),VCARD.FN,model1.createLiteral( "Shehram Shah"));
        }
        long endDelTime1 = System.nanoTime();
        long TimeTakenDel1=endDelTime1-startDelTime1;
```

Figure 19: Recording time for delete operations

The insert and delete operations could have been decoupled from the main code body by creating separate methods but it was decided to keep this section of code in the main method as it will be modified frequently for running tests on the data and any object or method dependencies created as a result of modularising this function would have made it complicated to change the code frequently.

The most important deliverable of the application is the time file, containing the time measurements discussed earlier in details, which will be used for further analysis to assess the possibility of modifying RDF structure through views. It was important that the file produced is readable so it can be used for performing analysis. The file is written using the *FileWriter* class. It could have also been done using the *FileOutputStream* class but as the contents of the target file are characters rather than raw bytes, the *FileWriter* class has been used. The object of *FIleWriter* has been casted in *BufferedWritter* to allow the use of *newline()* method which is not provided by *FileWriter*. The output of the file can be fed into macros in MS Excel for further manipulation and analysis. An extract of the code that creates the Time file is shown in Figure 20.

```
 File fileName4 = new File("C:/Users/DELL/Desktop/Time.txt");
          if (!fileName4.exists()) {
                   fileName4.createNewFile();
            }
 BufferedWriter out4= new BufferedWriter(new FileWriter(
fileName4 ));
          try {
             out4.write("Time for Insert Operations");
             out4.newLine();
             out4.newLine();
             out4.write(String.valueOf(TimeTakenAdd1));
             out4.newLine();
             out4.write(String.valueOf(TimeTakenAdd2));
             out4.newLine();
             out4.write(String.valueOf(TimeTakenAdd3));
             out4.newLine();

          finally {
             try {
                 out4.close();
             }
             catch (IOException closeException) {
                 // ignore
             }
          }
```

Figure 20: Generating Time.txt

Finally the modified RDF structures still contained in the models are then written into external files using the *writeFiles* method. Jena allows RDF to be displayed in several formats; this method displays it in RDF/XML format. A separate file is created for all the three models. For technical considerations of preserving the original test files so as to be able to use them for further tests, the modified RDF structures are not overwritten on the original files. The *writeFiles* method is shown below in Figure 21.

```
void writeFiles (File fileName, Model model) throws IOException
             {
              if (!fileName.exists()) {
                          fileName.createNewFile();
                    }
              FileWriter out = new FileWriter( fileName );
                 try {
                     model.write( out, "RDF/XML-ABBREV" );
                 }
                 finally {
                    try {
                        out.close();
                    }
                    catch (IOException closeException) {
                        // ignore
                    }
                 }
             }
```

Figure 21: Regenerating modified RDF files

3.4 Summary of Chapter

Having presented the overall development methodology used and discussion on detailed design of the application, the next chapter goes on to describe the testing process to validate the application and also the experiments conducted through this application to assess performance of updating RDF structures through views.

Chapter 4

Analysis

The main aim of the work represented in this dissertation is to measure the process of view maintenance is materialized RDF views. This is to be accomplished through the application as discussed in previous chapter. Before using the application, it is important to verify that the application works as intended. This chapter describes the application verification and evaluation process separately. The first section, Testing Strategy, describes the tests performed to validate the application in the testing phase of the software development process. The second section, Evaluation, describes in detail the experiments conducted using the application and results obtained from these tests. The discussion provides analysis and insights based on research and the outcome of the experiments.

4.1. Test Strategy

Testing is an integral part of the software development process. It is one of the most important phases of the SDLC in which the correctness of the system is assessed. The testing process involves auditing the application to ensure that it satisfies system requirements and also to identify potential bugs. As with any software project, testing was conducted with proper planning. The overall testing strategy was based on dynamic testing involving a combination of white and black box tests at the unit, system and subsystem level. Unit tests were applied to test the newly created piece of code. At the subsystem level, integration tests were performed at the end of each cycle to ensure that new features work properly in conjunction with existing code and fulfil collective objectives as intended. Functional tests were carried out at the end of each cycle to ensure that the implemented functionality is achieved as intended from the sections of code that account for the function. After finalising the code, system tests were performed to verify that all modules of code work collectively to fulfil overall system requirements. In line with the main objectives of the application, testing was focused on three main aspects. The first one being the ability of the application to read in RDF files and store it as in-memory models, the second area of interest was to test its ability to modify these models and verify the correctness of mechanisms to calculate and record time. The third important area of interest was to test the ability of the application to regenerate the modified models as RDF files as these files would then be used by Semantic Web applications. A template was standardized for creating test cases. Each test case contained the details of the unit of code being tested, the feature it is being tested for, the condition before execution of that code, the post execution condition, the expected result, the actual result and finally the result of the test.

The unit tests were performed in a step by step manner by testing through the sequential order of the flow of code. Micro level unit tests were performed for basic verification of code, gradually increasing from low to medium level functional testing. A variety of triggers (input and arguments for methods) were used during the testing. Initially the correct trigger was passed to the code and results were recorded after which an incorrect trigger was passed on the code. It allowed to verify that the code behaves as intended and to confirm the implemented exception handling mechanism functions as designed. As this project is of experimental nature and all the experiments were to be conducted under controlled environment, extensive testing was not performed on the application to check for un-expected inputs. However the blocks of code responsible for major features of the application were thoroughly tested. Some of the tests uncovered bugs in the code which were remedied and retested. Functional testing was applied on the finished code. Some of the tests carried for validating the application are discussed below.

The first step towards testing the software was to ensure that the RDF files are properly read into and stored into the in-memory model. Sample data was used during the low level tests where as the actual experimental files were used for functional and system level tests. This test case is named UNIT01 and is shown in Table 2. It tests whether the method *readFiles* reads in RDF files into models.

Table 2: Test case UNIT 01

Test ID	UNIT 01
Unit Under Test	Method: readFiles
Objective	To test if file is read in and stored in model
Pre-Conditions	Model does not exist
Post-Conditions	Model is created
Expected Results	Model is created and RDF file is read-in
Actual Results	Success

The second unit test, UNIT 02, was designed to check the behaviour of the code incase the file to be loaded into the models did not exist in the class path of the project. The test case is shown in Table 3.

Table 3: Test case UNIT 02

Test ID	UNIT 02
Unit Under Test	Method: readFiles
Objective	To test exception handling if file doesn't exist
Pre-Conditions	No error message on console
Post-Conditions	Error message displayed on console
Expected Results	Error message displayed on console
Actual Results	Success

The second phase involved testing that the RDF statement is created using the *createStatement* method. An RDF statement consists of three main parts, the subject, predicate and the object. The process of inserting a statement involves creating a new resource, using an existing property and creating a new object for that property. The *createStatement* method has *createResource* and *createLiteral* methods nested within it and it was important to ensure that the construct works smoothly and a correct RDF statement has been created. For testing purposes, incorrect arguments were passed on to the method along with valid ones. Table 4 represents the test case UNIT03 as discussed above.

Table 4: Test case UNIT 03

Test ID	UNIT 03
Unit Under Test	Method: createStatement
Objective	To test whether a new statement is created
Pre-Conditions	Statement does not exist
Post-Conditions	Statement is created
Expected Results	Statement is created
Actual Results	Success

The next logical step was to test whether the statement is added to model through the *model.add* method. The purpose of this method is to add a new statement to an existing model. The statement can either be appended to an existing resource or could involve creation of a new resource with corresponding predicates and objects. The statement created earlier is inserted into the model which is then verified by checking the contents of the model as there is no return type of this method. This unit test, named UNIT 04 is shown in Table 5.

Table 5: Test case UNIT 04

Test ID	UNIT 04
Unit Under Test	Method: add
Objective	To test whether a new statement is inserted in model
Pre-Conditions	Statement does not exist in model
Post-Conditions	Statement exists in model
Expected Results	Targeted Statement is added
Actual Results	Success

After verifying the statement insertion operation, we moved to the statement removal process which is performed by the *model.remove()* method. The argument passed on to this method is again a statement of the form discussed earlier. As with the *model.add()*, proper execution was verified by checking the model's contents. The test case UNIT 05 is shown in Table 6.

Table 6: Test case UNIT 05

Test ID	UNIT 05
Unit Under Test	Method: remove
Objective	To test whether a statement is removed from model
Pre-Conditions	Statement exists in model
Post-Conditions	Statement does not exist in model
Expected Results	Targeted Statement is removed
Actual Results	Success

Time was calculated for statement insertion and removal operations separately for all three models. It was ensured that the logic applied for measuring time was correct and accurate. Debugging was also used to test the code by commenting out various lines of code and using pointers and multiple variables, recording time at the same instance and then comparing the results obtained using the two ways. Special attention was given to the accuracy of the methods used as well as the use of the appropriate data type to contain the value of recorded time. The above test case, named UNIT 06 shown in Table 7 verifies the time calculation mechanism and its display on console.

Table 7: Test case UNIT 06

Test ID	UNIT 06
Unit Under Test	Class: UpdateRDF
Objective	To test time is calculated properly
Pre-Conditions	Time is not displayed on console
Post-Conditions	Time is displayed on console
Expected Results	Accurate time is displayed on console
Actual Results	Success

Table 8 shows test UNIT 07 which tests the creation of external file *Time.txt* and its contents with proper formatting so they can be easily read in by a tool for analysis. It is shown in Table 8.

Table 8: Test case UNIT 07

Test ID	UNIT 07
Unit Under Test	Class: UpdateRDF
Objective	To test Time.txt is created properly
Pre-Conditions	Time.txt is does not exist
Post-Conditions	Time.txt exists with proper values
Expected Results	Time.txt is created with correct values
Actual Results	Success

The last unit test UNIT 08, shown in Table 9, checks that the method *writeFiles* regenerates updated RDF files from the modified models correctly. This method specifies XML/RDF as the representation style of the RDF structure.

Table 9: Test case UNIT 08

Test ID	UNIT 08
Unit Under Test	Method: writeFiles
Objective	To test RDF files are created properly with correct representation style
Pre-Conditions	File does not exist
Post-Conditions	File exists with proper format
Expected Results	File is created with correct formatting
Actual Results	Success

Most of the unit tests discussed above were performed on one model and have been extended to include all three models as only the corresponding object name of the concerned RDF file was changed to incorporate them into the experiment. After unit testing, some level of black box testing was performed to ensure that the section of code being tested is performing as intended.

The first functional test was to check whether the code reads in RDF files into models and generates external RDF files without modifying the contents of the files; all three RDF files selected for use in the experiments were used for this test. For the test, part of the code performing the modification and responsible for calculating time was commented out. Test case BB 01 shown in Table 10 describes the outcome of the above test scenario.

Table 10: Test case BB 01

Test ID	BB 01
Unit Under Test	Class: UpdateRDF
Objective	To test RDF files are read into the model and regenerated externally
Pre-Conditions	Models do not exist
Post-Conditions	Models and File exist
Expected Results	Files are created properly
Actual Results	Success

The second black box test involved testing the complete functionality of the application by loading the same RDF file (same size) into three separate models and performing modification operations on them to ensure if similar results are obtained. This test case is shown in Table 11.

Table 11: Test case BB 02

Test ID	BB 02
Unit Under Test	Class: UpdateRDF
Objective	To compare results of modification operations on three models of same size
Pre-Conditions	Models do not exist
Post-Conditions	Models and File exist
Expected Results	Files should be created and time measurements should be similar
Actual Results	Success

System tests were performed at the end of the testing phases to verify the overall behaviour of the system and ensuring that requirements are fulfilled. RDF files to be used for experiments were used for conducting this test and time was measured for inserting one statement and removing one statement from the models. The output on the console was reviewed and time measurements displayed were compared to those in the external time file. The outputs, both on the console and external files were verified for their correctness and proper formatting. This test completes the testing phase of the software development process. The test case is shown in Table 12.

Table 12: Test case SYSTEM 01

Test ID	SYSTEM 01
Unit Under Test	Class: UpdateRDF
Objective	To test overall system function
Pre-Conditions	Models do not exist, no external files
Post-Conditions	Models and files exit
Expected Results	The software executes as intended with delivery of proper outputs
Actual Results	Success

As the developed application is of an experimental structure, and will be used under a controlled environment, the applied testing strategy provides reasonable guarantees that the application fulfils its overall objectives. The implementation mechanisms that perform the functions have been verified for their correctness. The classes, methods and data structures have been validated for their appropriateness in line with the technical requirements of the project. The non functional attributes such as performance, reliability, responsiveness of the application have been verified. From a project point of view, the testing process has ensured that software engineering and OOP principles have been adhered to throughout the software development cycle.

4.2. Evaluation

After validating the application, experiments were conducted in line with project objectives. The primary goal was to assess view maintenance of materialized RDF data. Other objectives included exploring different options available within Jena to manipulate RDF structures and analyse the performance of update operations with respect to the size of RDF structures. Three RDF files of sizes 1 MB, 5 MB and 10 MB were used in the experiments. The RDF equivalent of a (relational) record is a statement. Unlike traditional data structures, in which records can be modified directly by using the primary key, we cannot modify a RDF statement directly. The only way to modify an RDF statement is to overwrite it. For example, an RDF statement (*ShehramShahURI, VCARD:FN, Shehram*) has a property *VCARD:FN* whose value needs to be changed from *Shehram* to *Shehram Shah.* We perform the required modification through an insertion operation by simply adding a new statement to the model with the same resource, same property but with the new value of the property. It will replace the current triple corresponding to the same subject, predicate and object to (*Shehram Shah, VCARD:FN, Shehram Shah*).

A series of experiments were conducted to modify the experimental RDF structures using insert and delete statements. The following sections describe the experiments and their outcomes.

4.2.1 Experiment 1

The first experiment involved inserting and deleting one statement separately into the models. Time was recorded for the insert and delete operations separately for all the three models. Table 13 shows time values for creating and inserting one statement in three models.

Table 13: Results of Experiment 1

Data Volume (MB)	Insert (ns)	Insert (sec)	Delete (ns)	Delete (sec)
1	46514	0.046514	18045	0.018045
5	17804	0.017804	17112	0.017112
10	19325	0.019325	17661	0.017661

Figure 22 shows the scatter chart plotted from the time recorded after the experiment. The blue line represents time taken for insert operations while the red line represents time for delete operations. Even though the first model is of the smallest size, it takes longer to insert or delete statements from it.

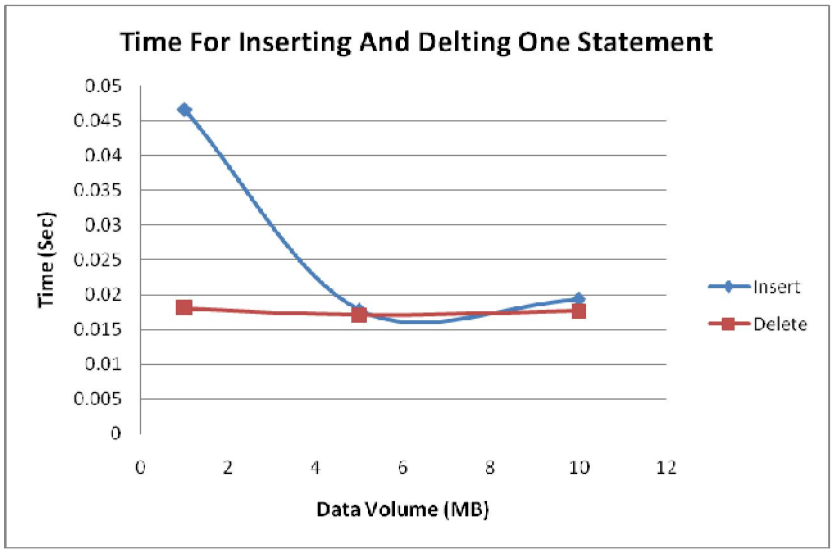

Figure 22: Scatter chart showing time for inserting and deleting one statement

4.2.2 Experiment 2

The second experiment was designed to measure time for bulk update operations. Time was recorded for inserting and deleting 100 statements in all three models. The time measurements for Experiment 2 are given in Table 14.

Table 14: Results of Experiment 2

Data Volume (MB)	Insert (ns)	Insert (sec)	Delete (ns)	Delete (sec)
1	4359071	4.359071	1804523	1.804523
5	1655010	1.65501	1711263	1.711263
10	1750946	1.750946	1766148	1.766148

The results obtained from this experiment are similar to those of the first one. It always takes longer to perform insert or delete operations on the first model and the size of file does not have a significant impact on performance. The scatter plot shown in Figure 23 shows that time taken for deleting statements from the first model having size 1 MB is slightly higher than that for second model having size 5 MB. It then increases slightly for the third model whose size is double (10 MB) then that of model 2.

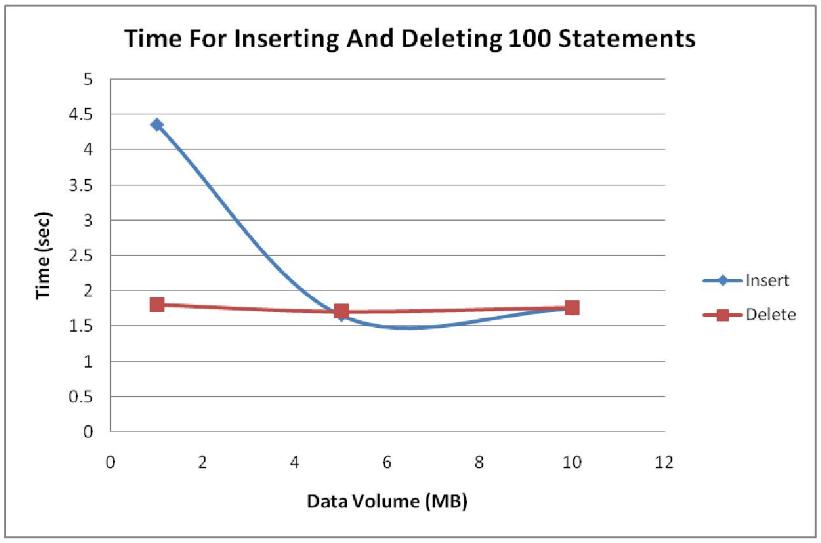

Figure 23: Scatter plot showing time for inserting and deleting 100 statements

4.2.3 Experiment 3

This experiment was conducted to confirm the observation made in the first two experiments i.e. it always takes longer to modify the first model regardless of size. Therefore all models were loaded with the same RDF file (size) and 100 statements were inserted and deleted. The time recorded is displayed in Table 15.

Table 15: Results of Experiment 3

Model	Insert (ns)	Insert (sec)	Delete (ns)	Delete (sec)
1	95470743	95.47074	2189575	2.189575
2	1967536	1.967536	1920508	1.920508
3	1983791	1.983791	1979854	1.979854

The results confirm the previous observations. The size of the RDF does not have a major impact on the operations. The results follow the same pattern as in previous experiments. It is graphically represented in Figure 24.

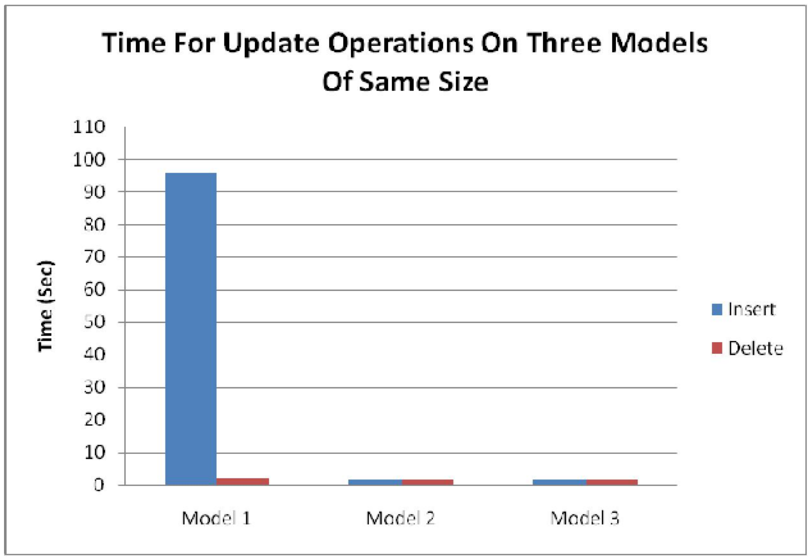

Figure 24: Graph showing time for update operations on three Models of same size

4.2.4 Experiment 4

In this experiment we switched the size of the first and third model. Model 1 is now of 10 MB and Model 3 is now of 1 MB. Insert operations were performed in the same sequence on all models (Model 1 to Model 3). Time recorded for inserting and deleting 100 statements into each model is shown in Table 16.

Table 16: Results of Experiment 4

Data Volume (MB)	Insert (ns)	Insert (sec)	Delete (ns)	Delete (sec)
10	6349188	6.349188	1207011	1.207011
5	2557256	2.557256	1190054	1.190054
1	2476045	2.476045	973641	0.973641

The results of this experiment confirm the conclusions of pervious experiments i.e. the first model irrespective of size always takes the longer time as shown in Figure 25.

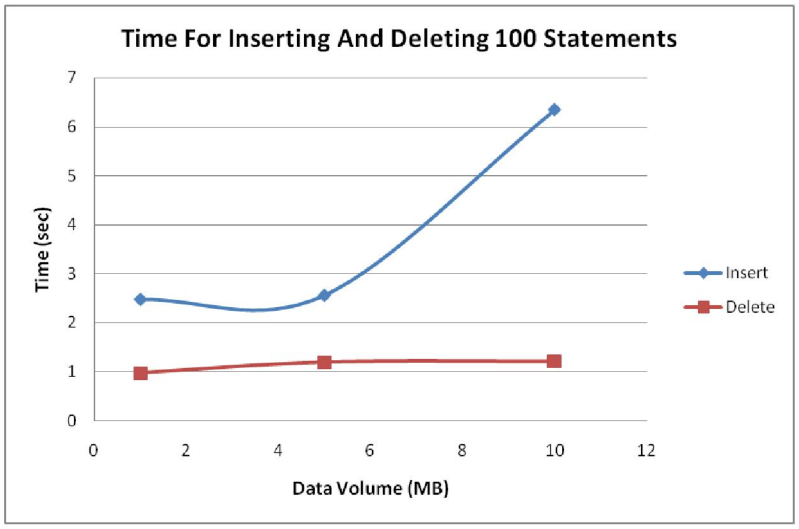

Figure 25: Scatter plot showing time for inserting and deleting 100 statements
(Note: As in Table 16, the 10 MB model was used before smaller models)

There was a clear pattern in results obtained from experiments 1 to 4 i.e. the time taken to insert a statement or a bulk of statements in the first model is always higher than the time taken to insert the same number of statements into the other two models. The size of the model does not have a significant effect on performance. There was consistency in the results of update and delete operations in all four experiments. There does not appear to be a clear relationship between performance of update operations and file size. The first model always takes longest to modify irrespective of size. The results were analysed from several angles to find out a proper explanation for the longer time taken to update the first model. After research to understand the underlying factors that can affect performance of a Java program, it was learnt that the initial run of a piece of code takes longer because the classes and other static blocks from the library have to be loaded into JVM. This process is often referred to as warming up of the JVM, and is performed whenever a new class or method of the Java API is executed. The second execution of the same code is always faster than the first one. After executing the same code 10,000 times, JVM compiles the native code to machine code which again significantly improves performance. Frequently executed blocks of code are also placed in cache memory by JVM for faster access, which too is a factor for varying performance of frequently executed sections of code. This observation was also supported by analyzing the results gathered for the deletion operations in both models in which the first model always takes longer as compared to others larger models however the difference in values for the first model and the other two models were not so disparate.

4.2.5 Experiment 5

Further experiments were aimed at studying the factors which affect the efficiency of insert operations as they take relatively longer then delete operations. A new RDF statement is created using the *createStatement* method. As discussed earlier in section 3.3, the create statement also has the *createResource* and *createLiteral* methods nested in it to create the subject and object of the RDF statement as shown below in Figure 26.

```
Statement S1 =
ResourceFactory.createStatement(model1.createResource("http://somewhere/S
hehramShah/" + i ),VCARD.FN,model1.createLiteral( "Shehram Shah" ));
```

Figure 26: Create Statement method

It was important to isolate the time taken for creating the statement from the total time (statement creation and insertion) to find out the time utilized to only insert the statements into the model. This was done by subtracting the values obtained in this experiment with those obtained earlier. This experiment is designed to find out the time taken for only inserting the statements into the models. The *model.add()* method in the code was commented out and time was calculated for only creating the statements. The results are shown in Table 17.

Table 17: Results of Experiment 5

Data Volume (MB)	Total Time (sec)	Time To Create Statement (sec)	Time To Insert Without Creating (sec)
1	4.651422	3.631358	1.020064
5	1.780427	1.140098	0.640329
10	1.932589	1.278874	0.653715

Figure 27 shows the relationship between time taken to insert values into the models minus the time taken to create the statements. The chart plots time taken to insert a statement into the model.

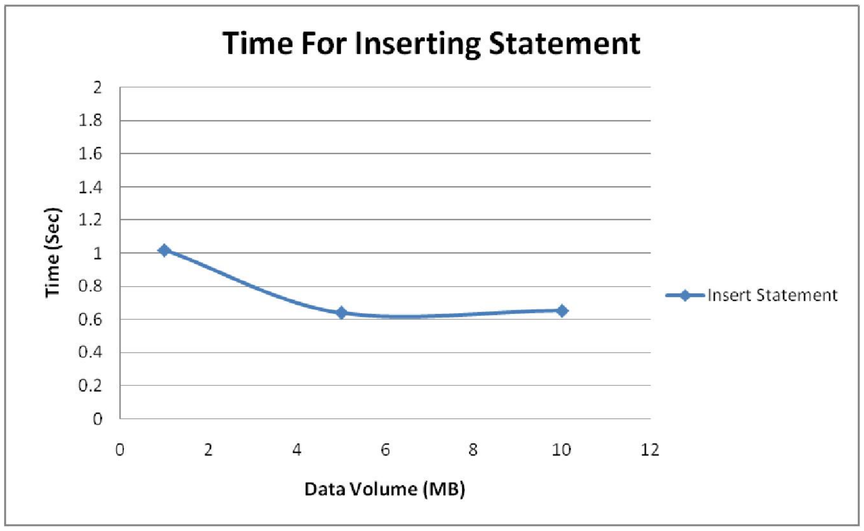

Figure 27: Scatter plot showing time for inserting one statement

The next test finds out the time taken for creating statements that are to be inserted into the model. This test is continuation of the previous experiment; the difference is that it focuses on time for creating the statement and not inserting them. The results are shown in Figure 28.

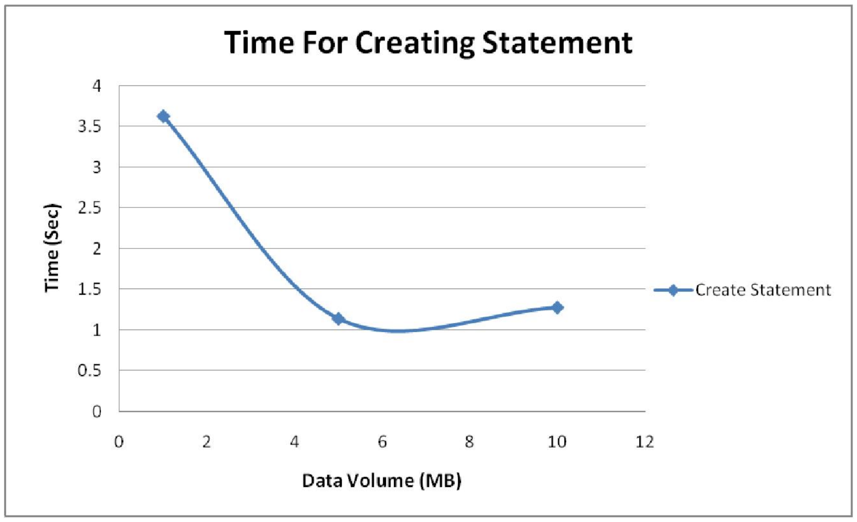

Figure 28: Scatter plot showing time for creating one statement

This experiment shows that it takes longer to create a statement then to insert it as creating a statement is a complex task involving multiple nested methods. Hence the longer duration of the overall insert operation can be attributed to statement creation.

4.2.6 Experiment 6

In order to confirm the observations made from the previous experiments, it was important to isolate the optimizing factor of JVM in order to get a correct measure of performance of update operations and to assess the effect of increasing file size on update operations. For this experiment, four models were used. Insert and delete operations were first performed on a test model which allowed the JVM to be 'warmed up' after which time was recorded for inserting and deleting 100 statements from three models of increasing file size similar to a scenario as in experiment 1. The results of this experiment are shown in Table 18.

Table 18: Results of Experiment 6

Data Volume (MB)	Insert (ns)	Insert (sec)	Delete (ns)	Delete (sec)
1	1802277	1.802277	1997721	1.997721
5	1835297	1.835297	2061084	2.061084
10	1832620	1.83262	2299811	2.299811

Figure 29 confirms the two observations made earlier i.e. the initial run of insert operations takes longer. The effect of this phenomenon has been negated in this experiment by performing the insert and delete operations on a test Model. It also confirms that the size of the RDF does not have a significant impact on update operations. This can be said true with respect to the size of RDF files used in this experiment.

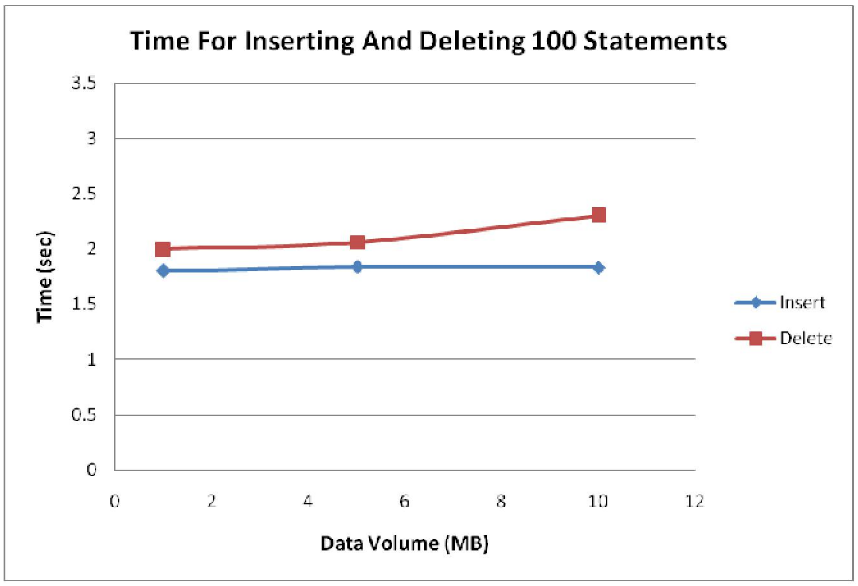

Figure 29: Scatter plot showing time for inserting and deleting 100 statements in 4 Models

4.3 Summary of Results:

From the above mentioned experiments we can summarize that every time a new code is executed for the first time, it takes longer as the corresponding classes, methods and constructors are loaded into the JVM. All subsequent executions of the same code are always faster than the first one. It is much quicker to delete statements from the model then to add them as inserting statements may also involve the creation of a completely new subject, predicate and object or any combination of new values and existing components of the triple in the model. The size of the RDF does not have a significant impact on the performance of update operations.

Chapter 5
Conclusion and Recommendations

This chapter concludes the dissertation by providing a summary of the project and the application developed. It also provides insights from analysis of the outcome of the experiments. The discussion reflects on the motivation for carrying out this exercise in the context of the Semantic Web and to explore various factors that come into play towards the transformation of the World Wide Web into the Semantic Web. It also contains a brief discussion on the limitations that were encountered while following the experimental approach and finally some directions for future work in continuation of this contribution.

5.1 Conclusion:

The purpose of this project has been to discuss the idea of the Semantic Web and understand the technologies and processes involved in realizing the vision of Tim Berners-Lee. The research investigates the myths and facts about the Semantic Web by reviewing scientific publications. It also identifies key contributions. The ultimate goal of the Semantic Web is global data integration on the Internet. It is to be achieved by creating a new generation of Web content, using RDF, which can be processed by machines to enable them to understand the semantics of the data. It aims to utilize superior computational power of machines to provide a richer and more useful Web experience to users. The Semantic Web proposes to integrate data by streamlining representation mechanisms. A vast majority of existing web content is only intelligible by humans. New mechanisms for creating data have been centered on the idea of dual consumption, targeted for both man and machines. Traditional technologies like XML, currently used to create information for the Internet are ambiguous and its architectural structure (tree) is not suitable for the new requirements. The same information is represented in numerous forms, resulting in redundancy. Other factors such as the lack of a uniform system to represent information hamper data integration.

The Semantic Web introduces with it a complete suite of technologies (Figure 1) to support the transformation of the Internet. We discuss current technologies for exchanging information over the Internet and their shortcomings. We also discuss Semantic Web technologies such as RDF, RDF/S, OWL and other associated technologies like SPARQL and SPARUL in the context of their role in the Web of the future. We study some specialized applications of the Semantic Web and discuss their advantages over similar applications based on traditional technologies. The foundations of the Semantic Web are based on RDF, which has been recommended as the universal data interchange format for the Internet by W3C. RDF data is represented in the form triples i.e. subject, predicate and object. Each part of the triple can be identified using a URI which points to a resource on the web. This removes the possibility of ambiguity and redundancy as all representations point to the same URI. Applications based on the concept of the Semantic Web have shown markable success and have clear advantages over traditional applications. RDF and its associated technologies (RDFS, OWL) solve some of the fundamental problems for data integration by providing a uniform vocabulary and representation structure thereby allowing data to be shared and reused across the internet. Applications have the ability to combine and manipulate multiple pieces of related data (defined in RDFS vocabulary) that have been created by different users be treated as a single entity. The data integration process is complex and as like any data it needs to be modified as time passes. Some large applications have RDF data sets containing billions of triples. As industry and academia realize the true potential of RDF, more and more applications are transitioning from legacy to RDF based data structures; we need to have clear mechanisms that enable us to update the actual RDF files that are fed into the applications. Researchers have proposed several RDF engines to use and manage RDF data in an efficient manner using SPARQL and SPARUL. SPARUL is an update to the SPARQL query language, this allows us to insert or delete triples in an existing RDF structure. Our project is based on Jena, it is a Semantic Web tool kit based on Java for developing Semantic Web applications and can manipulate Ontologies and RDF based data structures.

From a relational point of view, RDF data structures can be seen as a three dimensional table. They can be considered as materialized views of the relational table. As with RDBMS, we also explore the possibility of updating RDF structures through views. In this project we have focused exploiting Jena to modify RDF structures. A prototype application has been developed to update RDF structures through views. The idea is based on the concept of view maintenance for the advantages of views in terms of security, convenience in the use and management of data. We experimented with taking sections of RDF structures of different data sizes and performed modification operations on them. The experimental data was based on the vCard ontology. Three datasets of sizes 1 MB, 5 MB and 10 MB were used for measuring the performance of update operations. The three RDF files were read and stored into three models respectively. Models are in-memory structures for storing RDF contents in Jena. There were several alternatives to models with in Jena that can be used at different levels of RDF data depending on the intended use. However models were deemed as the most appropriate due to technical factors discussed earlier in detail. The application was designed to read in RDF files and store each into separate models. Performance was then measured for a range of modification operations applied on the models individually. For measuring performance, elapsed time was decided over CPU time as CPU time varies from machine to machine since CPU resources are distributed by the operating system depending on the multi threading mechanism implemented. The JVM also performs some degree of scheduling which also affects the value of CPU time. Whereas the elapsed time is a simple value of time taken to complete one task based on clock time. Time duration was recorded with nanosecond precision for all of the experiments. The basic aim of the experiments was to analyse the impact of the size of RDF structures on update operations. Initially experiments were conducted by inserting one statement into each model and time taken to complete the task was recorded. The same was repeated for deleting one statement from the models. In the second experiment we performed a bulk of insert and delete operations (100 statements for both) into the models and recorded time for them. Initial results showed that regardless of the size of RDF, the first model always took longer to insert or delete statements. We also attempted to update three models of the same size in Experiment 3. In this experiment the results were similar to the ones recorded earlier; it always takes longer to update the first model irrespective of size. This trend was observed for both, inserting one statement and inserting a bulk of statements. This was further experimented by switching the file size i.e. the smaller file into the first model and the largest file into the first model (Experiment 4) but the outcome was same. It was important to investigate the reason for this observation. Further research uncovered that whenever a new method is called, its classes, constructors and other static blocks are loaded into the memory which takes some more time. Once loaded into the JVM, all subsequent executions of the code will always be quicker than the first one. The above set of experiments showed similar results and the actions of the optimiser dominate the time pattern.

Further experiments (Experiment 5), conducted on the basis of the outcome of these experiments, focused on the closer analysis of the overall time taken for insert operations by isolating the time taken to create a statement and the time taken to insert the statement. The process of inserting a statement into the model involves creation of a new statement. An RDF statement is made up three components, subject, predicate and object, which are also created separately and combined to form a single statement. The statement is then added it into the model. This process involves nesting of several methods within each other which takes some time to execute. Hence time for insert operations was calculated by subtracting time for creating a statement from time taken to perform the complete operation (creation and insertion). These results were used to analyse the relationship between the size of RDF and time of update operations.
A final set of experiments (Experiment 6) was carried out by 'warming up' the JVM and then measuring performance. This showed that the time for insert operations increases with model size over 1-10 MB whereas the time for delete operations remains stable (Figure 28).
Based on the outcome of the experiments and research, the observations can be summarized as follows.

- The first time execution of a piece of code always takes longer as compared to subsequent executions. This is attributed to the phenomenon known as JVM warm up.
- It takes longer to insert statements into an RDF file then to delete one. This is because the insertion operation involves first creating a statement and then inserting it.
- The size of RDF structure does not have a significant impact on performance of update operations.
- RDF structures can be easily updated using the view maintenance approach for RDMS databases.

5.2 Limitations

This section discusses the limitations of the experimentation approach applied for the project and describes some unexpected encounters during the development process.

- As Jena is not part of the standard Java package, its libraries need to be added separately. Initially they were imported into Eclipse Indigo IDE using the recommended method however it appeared that they were not compatible with this version of the IDE as a basic Jena program did not execute and no error message was displayed. After trouble shooting a different version of Eclipse IDE, Eclipse Juno was used for the project.
- The test data was carefully chosen and prepared for the tests. While selecting the data set, it was decided that RDF structures with simple data (uncomplicated statements) and size according to the scope of the experiments will be chosen. During the process of analyzing potential datasets, some datasets downloaded from W3C repository could not be read into Jena. This was unexpected, the possible reasons could be an incompatible encoding scheme or corrupted file. However the vCard ontology was easily processed by Jena. It is uncomplicated and easy to read and understand. Hence it was decided to further expand it for our project. The size of the test files was expanded according to scope of the project.
- Another interesting point was regarding the behaviour of Jena code (inserting statement, deleting statements). There is no way to find out if a piece of code has been executed. The only way to confirm whether a statement has been inserted or deleted is to manually check the contents of the file. No error was displayed if the statement targeted to be deleted from the model did not exist.

Throughout the project, decisions were made according to the situation and in line with the overall objectives.

5.3 Recommendations and Future Work:

The prototype application and the findings of this project can be used to further investigate the possibility of maintaining RDF structures through views and measure performance with respect to the size of RDF. This work can be used as a stepping stone for applications that aim to modify RDF structures without using SPARUL (SPARQL Update). Our primary research objective was to measure performance of view maintenance on RDF structures in general and comparisons of insert and delete operations in particular. Experiments performed in the above research context provide a detailed analysis of factors which impact performance. The results of this project and other similar work done so far by academia can be used to explore other options available for maintaining RDF structures. Based on our literature review and outcomes of this project, following recommendations are made for future work.

- The use of Model class in Jena provides a promising direction to bridge the knowledge gap between software engineers and domain experts which is one of the reasons for slow uptake of Semantic Web technologies. It is much easier to modify an RDF structure using methods of Model class as compared to designing SPARUL (SPARQL Update) queries. We do not recommend the use of Jena's capability over SPARUL (SPARQL Update) based on performance criteria. However from an implementation perspective, it is much easier for a programmer to use programming constructs rather than to develop skills in a new query language for accomplishing the task.
- More detailed experiments can be conducted by use of performance analysis tools (profilers) to provide a deeper understanding of time taken to maintain RDF structures. These tools can be first used to compare the performance of updating RDF structures through models and also through SPARUL (SPARQL Update) to identify the most efficient way of updating RDF. Apart from time, the computational power utilized by either technique would also be a useful criterion.
- Diverse test data can be made part of the experiment by including larger RDF file sizes (GBs) to conduct tests in greater detail.
- All the tests conducted in this project were performed on RDF files that were stored locally on the machine, an attempt can be made at the possibility of developing an application that can access and modify an RDF file remotely through the Internet.

References

Alexaki, S. et. Al. , (2001). On Storing Voluminous RDF Descriptions: The Case of Web Portal Catalogs. *WebDB*, pp. 43-48

Apache Jena, (2011). *What is Jena?* . [online] Available at: <http://jena.apache.org/about_jena/about.html> [Accessed 30 August 2013].

Apache Jena 2011. *An Introduction to RDF and the Jena RDF API.* [online] Available at: <http://jena.apache.org/tutorials/rdf_api.html> [Accessed 30 August 2013].

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). *The Semantic Web*. Scientific American, 284(5), pp. 28-37.

Brooks, R. A. (1991). Intelligence without representation. *Artificial intelligence*, *47*(1), pp. 139-159.

Buffa, M.et. al. (2008). SweetWiki: A semantic wiki. *Web Semantics: Science, Services and Agents on the World Wide Web*, *6*(1), pp. 84-97.

Carroll, J. J. et. al. (2004). Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters.* New York, Monday May 17th to Thursday May 20th 2004, New York: ACM pp. 74-83.

Ceri, S., & Widom, J. (1991). Deriving production rules for incremental view maintenance. In: VLDB(Very Large DataBase) Endowment, *In Proceedings of the Seventeenth International Conference on Very Large Data Bases*. Barcelona, Tuesday September 3rd to Friday September 6th 1991, Barcelona:Citeseer. pp. 577-589

Chong, E. I. et. al. (2005). An efficient SQL-based RDF querying scheme. In: VLDB(Very Large DataBase) Endowment, *Proceedings of the 31st international conference on Very large Data Bases*. Trondheim,Norway August 30th -September 2nd 2005, New York:ACM.

Corby, O., Dieng-Kuntz, R. & Faron-Zucker, C. (2004). Querying the semantic web with corese search engine. In *ECAI* (European Conference on Artificial Intelligence).Valencia, Sunday August 22nd to Thursday August 27th 2004. Valencia: IOS Press. p. 705

Data-gov Wiki, (2010) [online] Available at: <http://data-gov.tw.rpi.edu/wiki> [Accessed 30 August 2013].

Erling, O., & Mikhailov, I. (2009). *RDF Support in the Virtuoso DBMS. Networked* Knowledge-Networked Media ,221, pp. 7-24.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2), 199-220.

Gruber, T. (2009). Ontology. 1963 – 1965.

Lassila, O., Swick, R. (1999). *Resource Description Framework (RDF) Model and Syntax Specification*. Available at: < http://www.w3.org/TR/REC-rdf-syntax/> [Accessed 30 August 2013].

McGuinness, D. L., & Van Harmelen, F. (2004). OWL web ontology language overview. *W3C recommendation*, *10*(2004-03), p. 10.

Neumann, T., & Weikum, G. (2009). Scalable join processing on very large RDF graphs. In: SIGMOD (Special Interest Group on Management of Data) , *In Proceedings of the 35th SIGMOD international conference on Management of data* . Providence, RI Monday June 29th to Thursday July 2nd 2009, Providence, RI: ACM. pp. 627-640

Neumann, T., & Weikum, G. (2008). RDF-3X: a RISC-style engine for RDF. In: VLDB(Very Large DataBase) Endowment, *In Proceedings of the Thirty fourth International Conference on Very Large Data Bases*. Auckland, New Zealand Saturday August 23rd to Thursday August 28th 2008, Auckland: ACM. pp. 647-659

Peters, I. (2009). Folksonomies: Indexing and Retrieval in the Web 2.0. *Walter de Gruyter* .

Renteria, W. et. a.l, (2010).. Exploring the Advances in Semantic Search Engines. Distributed Computing and Artificial Intelligence,79, pp.647-85.

Schaffert, S. (2006). IkeWiki: A semantic wiki for collaborative knowledge management. In: *WETICE Enabling Technologies: 15th IEEE International Workshop on Infrastructure for Collaborative Enterprises.* Manchester, UK 26-28 June 2006. New York: IEEE.

Seaborne, A., et al. (2008). SPARQL/Update: A language for updating RDF graphs. *W3C member submission*.

Seaborne, A., (2004). *RDQL - A Query Language for RDF.* [online] Available at: <http://www.w3.org/Submission/RDQL/> [Accessed 30 August 2013].

Shadbolt, N., Hall, W., & Berners-Lee, T. (2006). The Semantic Web revisited. *Intelligent Systems, IEEE*, 21(3), pp. 96-101.

Sheshagiri, M., Sadeh, N., & Gandon, F. (2004). Using Semantic Web services for context-aware mobile applications. In: *MobiSys 2004 Workshop on Context Awareness.* Boston, USA 06-09 June 2004.

Starr, B. (2012). *How Search & Social Engines Are Using Semantic Search.* [online] Available at: <http://searchengineland.com/semantic-search-what-is-it-how-are-major-search-and-social-engines-use-it-part-1-133160> [Accessed 30 August 2013].

Volz, R., Oberle, D., & Studer, R. (2002). Towards views in the semantic web. In: *DBFUSION, 2nd Int'l Workshop on Databases, Documents and Information Fusion.* Karlsruhe, Germany July 2002.

World Wide Web Consortium (W3C) Semantic Web interest group, (2006). *Semantic Web Interest Group* . [online] Available at: <http://www.w3.org/2001/sw/interest/> [Accessed 30 August 2013].

World Wide Web Consortium (W3C) vCard Ontology, (2013). *RDF vCard RFC6350 .* [online] Available at: < http://www.w3.org/TR/vcard-rdf/#RFC6350> [Accessed 30 August 2013].

World Wide Web Consortium (W3C)., (2004). *RDF Vocabulary Description Language 1.0: RDF Schema.* [online] Available at: <http://www.w3.org/TR/rdf-schema/> [Accessed 30 August 2013].

World Wide Web Consortium (W3C)., (2013). *DataSetRDFDumps.* [online] Available at: <http://www.w3.org/wiki/DataSetRDFDumps> [Accessed 30 August 2013].

World Wide Web Consortium (W3C).,(2008). *SPARQL Update.* [online] Available at: < http://www.w3.org/Submission/SPARQL-Update/> [Accessed 30 August 2013].

World Wide Web Size.,(2013). *The size of the World Wide Web (The Internet).* [online] Available at: < http://www.worldwidewebsize.com/> [Accessed 30 August 2013].

Appendix A – Code

The program listing of the application is as follows

```
package Project;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.Calendar;
import java.util.Date;

import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.ResultSetFormatter;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.ResourceFactory;
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.vocabulary.VCARD;

public class UpdateRDF {

        void readFiles(String inputfilename, Model modelname) throws IOException
                {
           InputStream in = FileManager.get().open(inputfilename);
           if (in == null) {
              throw new IllegalArgumentException ( "File: " + inputfilename + " not found");
           }
           modelname.read(new InputStreamReader(in), "");
           in.close();
                }

        void writeFiles (File fileName, Model model) throws IOException
                {
                 if (!fileName.exists()) {
                                fileName.createNewFile();
                        }
                FileWriter out = new FileWriter( fileName );
                  try {
                     model.write( out, "RDF/XML-ABBREV" );
```

```
                    }
                    finally {
                      try {
                         out.close();
                      }
                      catch (IOException closeException) {
                         // ignore
                      }
                    }
                 }

                 /**
       * @param args
       * @throws IOException
       */

      public static void main(String[] args) throws IOException {
                 // TODO Auto-generated method stub

                 final String  inputFileName1 = "data1024.rdf";
                 final String  inputFileName2 = "data5120.rdf";
                 final String  inputFileName3 = "data10240.rdf";

                 final File fileName1 = new File("C:/Users/DELL/Desktop/File1.txt");
                 final File fileName2 = new File("C:/Users/DELL/Desktop/File2.txt");
                 final File fileName3 = new File("C:/Users/DELL/Desktop/File3.txt");

                 //reading files
                 System.out.println("Reading Files...");

                 //1 MB
                 UpdateRDF UpdateRDF1= new UpdateRDF();
                 Model model1 = ModelFactory.createDefaultModel();
                 UpdateRDF1.readFiles(inputFileName1,model1);
System.out.println( "== File 1 Read ==" );

        //5 MB
UpdateRDF UpdateRDF2= new UpdateRDF();
                 Model model2 = ModelFactory.createDefaultModel();
                 UpdateRDF2.readFiles(inputFileName2,model2);
System.out.println( "== File 2 Read ==" );

                 //10 MB
                 UpdateRDF UpdateRDF3= new UpdateRDF();
                 Model model3 = ModelFactory.createDefaultModel();
                 UpdateRDF3.readFiles(inputFileName3,model3);
System.out.println( "== File 3 Read ==" );

//Inserting 100 Statements in model 1
System.out.println("\nInitiating Insert Operations on Model 1...");
```

```
    long startAddTime1 = System.nanoTime();
    for(int i=0; i<=100; i++)
        {
                        Statement S1 =
ResourceFactory.createStatement(model1.createResource("http://somewhere/Model A/" + i
),VCARD.FN,model1.createLiteral( "Model A" ));
                model1.add(S1);
        }
    long endAddTime1 = System.nanoTime();
    long TimeTakenAdd1=endAddTime1-startAddTime1;

    System.out.println("== Insert Operations on Model 1 Completed ==\n");

    //Inserting 100 Statements in model 2
    System.out.println("Initiating Insert Operations on Model 2...");

    long startAddTime2 = System.nanoTime();
    for(int i=0; i<=100; i++)
        {
                Statement S2 =
ResourceFactory.createStatement(model2.createResource("http://somewhere/Model B/" + i
),VCARD.FN,model2.createLiteral( "Model B" ));
                model2.add(S2);
        }
    long endAddTime2 = System.nanoTime();
    long TimeTakenAdd2=endAddTime2-startAddTime2;

    System.out.println("== Insert Operations on Model 2 Completed ==\n");

    //Inserting 100 Statements in model 3
    System.out.println("Initiating Insert Operations on Model 3...");

    long startAddTime3 = System.nanoTime();
    for(int i=0; i<=100; i++)
        {
                Statement S3 =
ResourceFactory.createStatement(model3.createResource("http://somewhere/Model C/" + i
),VCARD.FN,model3.createLiteral( "Model C" ));
                model3.add(S3);
        }
    long endAddTime3 = System.nanoTime();
    long TimeTakenAdd3=endAddTime3-startAddTime3;

    System.out.println("== Insert Operations on Model 3 Completed ==\n");

    //Deleting 100 statements from model1
    System.out.println("Initiating Delete Operations on Model 1...");

    long startDelTime1 = System.nanoTime();
    for(int i=0; i<=100; i++)
        {
```

```
                    model1.remove(model1.createResource( "http://somewhere/Model A/" +
i),VCARD.FN,model1.createLiteral( "Model A"));
            }
        long endDelTime1 = System.nanoTime();
        long TimeTakenDel1=endDelTime1-startDelTime1;

        System.out.println("== Delete Operations on Model 1 Completed ==\n");

        //Deleting 100 statements from model2
        System.out.println("== Initiating Delete Operations on Model 2 ==");

        long startDelTime2 = System.nanoTime();
        for(int i=0; i<=100; i++)
            {
                    model2.remove(model2.createResource( "http://somewhere/Model B/" +
i),VCARD.FN,model2.createLiteral( "Model B"));
            }
        long endDelTime2 = System.nanoTime();
        long TimeTakenDel2=endDelTime2-startDelTime2;

        System.out.println("== Delete Operations on Model 2 Completed ==\n");

        //Deleting 100 statements from model3
        System.out.println("Initiating Delete Operations on Model 3...");

        long startDelTime3 = System.nanoTime();
        for(int i=0; i<=100; i++)
            {
                    model3.remove(model3.createResource( "http://somewhere/Model C/" +
i),VCARD.FN,model3.createLiteral( "Model C"));
            }
        long endDelTime3 = System.nanoTime();
        long TimeTakenDel3=endDelTime3-startDelTime3;

        System.out.println("== Delete Operations on Model 3 Completed ==");

        //Display Insertion times for statements
        System.out.println("\nInsertion Opertation Elapsed Time\n");
        System.out.println("File1(1MB): "+ TimeTakenAdd1 + " Nanoseconds");
        System.out.println("File2(5MB): "+ TimeTakenAdd2 + " Nanoseconds");
        System.out.println("File3(10MB): "+ TimeTakenAdd3 + " Nanoseconds");

        //Display Deletion times for statements
        System.out.println("\nDeletion Operation Elapsed Time\n");
        System.out.println("File1(1MB): "+ TimeTakenDel1 + " Nanoseconds");
        System.out.println("File2(5MB): "+ TimeTakenDel2 + " Nanoseconds");
        System.out.println("File3(10MB): "+ TimeTakenDel3 + " Nanoseconds");

        //Display Average Insertion times for 1 statement
        System.out.println("\nAverage Insertion time for 1 Statement\n");
        System.out.println("File1(1MB): "+ TimeTakenAdd1/100 + " Nanoseconds");
        System.out.println("File2(5MB): "+ TimeTakenAdd2/100 + " Nanoseconds");
        System.out.println("File3(10MB): "+ TimeTakenAdd3/100 + " Nanoseconds");

        //Display Average Deletion time1 for 1 statement
        System.out.println("\nAverage Deletion time for 1 Statement\n");
        System.out.println("File1(1MB): "+ TimeTakenDel1/100 + " Nanoseconds");
        System.out.println("File2(5MB): "+ TimeTakenDel2/100 + " Nanoseconds");
        System.out.println("File3(10MB): "+ TimeTakenDel3/100 + " Nanoseconds");

        //Write time to external file
        File fileName4 = new File("C:/Users/DELL/Desktop/Time.txt");
                if (!fileName4.exists()) {
                                fileName4.createNewFile();
                    }

                BufferedWriter out4= new BufferedWriter(new FileWriter( fileName4 ));
                try {
                    out4.write("Time for Insert Operations");
                    out4.newLine();
                    out4.newLine();
                out4.write(String.valueOf(TimeTakenAdd1));
                out4.newLine();
                out4.write(String.valueOf(TimeTakenAdd2));
                out4.newLine();
                out4.write(String.valueOf(TimeTakenAdd3));

                out4.newLine();
                out4.write("=========================================");
                out4.newLine();
                out4.newLine();

                    out4.write("Time for Remove Operations");
                    out4.newLine();
                    out4.newLine();
                    out4.write(String.valueOf(TimeTakenDel1));
                    out4.newLine();
                    out4.write(String.valueOf(TimeTakenDel2));
                    out4.newLine();
                out4.write(String.valueOf(TimeTakenDel3));

                out4.newLine();
                out4.write("=========================================");
                out4.newLine();
                out4.newLine();

        out4.write("Average Time for Inserting One Statement");
        out4.newLine();
        out4.newLine();
                out4.write(String.valueOf(TimeTakenAdd1/100));
                out4.newLine();
                out4.write(String.valueOf(TimeTakenAdd2/100));
                out4.newLine();
                out4.write(String.valueOf(TimeTakenAdd3/100));

                out4.newLine();
                out4.write("======================================");
                out4.newLine();
                out4.newLine();

        out4.write("Average Time for Deleting One Statement");

        out4.newLine();
        out4.newLine();
                out4.write(String.valueOf(TimeTakenDel1/100));
                out4.newLine();
                out4.write(String.valueOf(TimeTakenDel2/100));
                out4.newLine();
                out4.write(String.valueOf(TimeTakenDel3/100));
                }
                finally {
                try {
                  out4.close();
                }
                catch (IOException closeException) {
                  // ignore
                }
                }

                System.out.println("\n== Time File Created ==");

                    UpdateRDF1.writeFiles(fileName1,model1);
                    System.out.println("\n== File1 Created ==");
                    UpdateRDF2.writeFiles(fileName2,model2);
                    System.out.println("== File2 Created ==");
                    UpdateRDF3.writeFiles(fileName3,model3);
                    System.out.println("== File3 Created ==");
          }

    }
```

Appendix B – User & System Guide

1. Import Jena libraries into preferred IDE for Java.
2. Import test data sets into the project's class path.
3. Modify code according to requirement of experiment.
4. Results can be viewed on console, a separate file 'Time.txt' is also created on desktop which can be used for graphical analysis of data.

Sample output:

```
Time for Insert Operations

4401014
1642070
1802707
========================================

Time for Remove Operations

1749607
1573353
1634038
========================================

Average Time for Inserting One Statement

44010
16420
18027
========================================

Average Time for Deleting One Statement

17496
15733
16340
```