ANALYSIS AND APPLICATION OF SEMANTIC WEB MECHANISM FOR STORING AND QUERYING ONTOLOGIES

by

Raoul Kwuimi

207010251

Dissertation submitted in fulfilment of the requirement for the degree of

Magister Technologiae: Information Technology

in the

Department of Information and Communications Technology, Faculty of Applied and Computer Sciences, Vaal University of Technology

Supervisor: Dr J.V. Fonou-Dombeu

February, 2017

DECLARATION

I hereby declare that this dissertation, which I submit for the qualification of

Magister Technologiae: Information Technology

To the Vaal University of Technology, Department of Information and Communications Technology, Faculty of Applied and Computer Sciences, apart from the recognized assistance of my supervisor and provided citations, is my own work and has not previously been submitted to any other institution for any degree.

_____ on this _____ day of _____
Candidate

_____ on this _____ day of _____
Supervisor

DEDICATION

I dedicate this work to the Almighty, my mother Blanche, my father Mr Tapja, my two children Anthony and Elia, my grandmother Marie, my lovely wife Christelle and lastly the whole Tapja family.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Dr J.V. Fonou-Dombeu for the useful comments, remarks and engagement through the learning process of this Master thesis. Furthermore, I would like to thank my colleagues for the support on the way. I would like to thank my loved ones, who have supported me throughout entire process, both by keeping me harmonious and helping me putting pieces together. I will be grateful forever for your love.

PUBLICATIONS

The following publications have resulted from this work:

Kwuimi, R. & Fonou-Dombeu, J.V. (2015) Storing and Querying Ontologies in Relational Databases: An Empirical Evaluation of Performance of Database-Based Ontology Stores, *In Proceedings of the 9th International Conference on Advances in Semantic Processing (SEMAPRO 2015)*, Nice, France, ISBN: 978-1-61208-420-6, 19-24 July, 2015, pp. 6-12.

Kwuimi, R., Fonou-Dombeu, J.V. & Zuva, T. (2015) An Empirical Analysis of Semantic Web Mechanisms for Storage and Query of Ontologies in Relational Databases, *In Proceedings of the 3rd IEEE International Conference on Advances in Computing & Communication Engineering 2016 (ICACCE 2016),* Durban, South Africa, ISBN: 987-1-5090-2576-6, 28-29 November, pp. 142-146.

.

ABSTRACT

Since the introduction of Semantic Web in the early 2000, storing and querying ontologies have been a subject of greater research. Thus, several types of storage media and mechanisms have been developed to increase storage and query speed and efficiency of ontologies in Semantic Web applications. Existing Semantic Web mechanisms for storing and querying ontologies are implemented on several storage media and support different languages. However, there is a shortage of studies that provide an empirical analysis and application of these ontology storage and query mechanisms in the Semantic Web domain.

This study conducted an analysis and application of the Semantic Web mechanisms for storing and querying ontologies. A thorough literature review was carried out to identify relevant publications pertaining to existing Semantic Web mechanisms for storing and querying ontologies as well as the platforms and storage media for implementing these mechanisms. Thereafter, the Design research method was used consisting of a set of predefined steps, namely, awareness, suggestion, development, evaluation, and conclusion. The awareness stage identified the need for an architecture to test several ontology storage media and mechanisms. In the suggestion stage a framework was proposed to empirically analyse and evaluate existing ontology storage and query mechanisms. The required Semantic Web platforms were identified to implement the framework in the development stage. The evaluation stage used a set of metrics to evaluate the framework including: the loading times of ontologies, the disc space used to store the ontology repositories and the mean and variance of query response times. Further, the evaluation stage analysed and discussed the storage mechanisms implemented in Semantic Web platforms. Finally, the outcome of the performance of the framework is presented in the conclusion stage.

The framework was practically tested with six ontologies of different formats and sizes on two popular Semantic Web platforms, namely, Sesame and Jena API and the ontology storage and query mechanisms were analysed and compared. Although the underlying structures of repositories in the in-memory and native files in Jena and Sesame could not be accessed, it was possible to access and analyse the data in the repositories in the relational database storage in both Sesame and Jena. The results showed that Sesame relational uses a combination of mechanisms such as normalized triples store in combination with vertical partitioning. That combination allows

Sesame to store ontologies based on their contents; in other words, each ontology has a different database schema in Sesame. Jena on the other hand, uses only a normalized triple store mechanism, also known as generic schema mechanism to store ontologies; thus, all ontologies in Jena have the same database schema.

The study would be useful to the Semantic Web and Computer Science communities as it does not only provide theoretical knowledge but also the empirical findings that may serve as a base for further development of ontology storage media and mechanisms.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

Chapter 1: INTRODUCTION

1.1 Background

Currently, the contents of the Internet is only interpreted and understood by human beings. Consequently, a large volume of information available on the current Web is inaccessible and unused. To address this issue, a new type of Web has emerged recently and is called Semantic Web. Semantic Web was introduced in 2001 by Tim Berners-Lee, the inventor of the World Wide Web (WWW). In his vision, software agents could interact and negotiate with each other to schedule appointments, find documents and locate services for users over the Internet (Berners-Lee, Hendler & Lassila, 2001). The goal is to enable computers to process information on the Web with less human intervention. In recent years, Semantic Web has attracted many leading companies. For example, Facebook, Google and Twitter have adopted Semantic Web technologies; they produce and consume a large volume of Semantic Web data in Resource Description Framework (RDF) format on a daily basis (Splendiani et al., 2011). The Semantic Web has also attracted industries and governments because it enables the building of Web-based systems that have the potential to perform complex and intelligent transactions over the Internet. This has led to Semantic Web applications being developed in areas such as e-commerce, e-business, e-government, Multimedia and Audio-Visual, and many more (Domingue, Fensel & Hendler, 2011).

Developing a Semantic Web application requires the building of a knowledge base model, namely, ontology. The ontology mainly represents the relevant data to be stored and manipulated by the Semantic Web applications. Over the past few years, different mechanisms have been developed to store ontology in traditional Relational Database Management Systems (RDBMS), in computer memory, or in disk files (Ramanujam et al., 2009, LiLi, Lee & Kim, 2010); however, they are largely unknown in the broader software engineering community.

In the e-government domain for instance, several domain ontologies have been developed to enable knowledge sharing and integration amongst government departments (Goudos et al., 2007; Stadlhofer, Salhofer & Tretter, 2009; Fonou-Dombeu & Huisman, 2011). These e-government

domain ontologies need to be persistently stored to enable the daily running of government businesses. Furthermore, there is a large volume of government data that has been developed over the years and stored in legacy RDBMS. The advent of Semantic Web requires that these data be transferred into ontologies for their consumption by the Semantic Web applications. They also require an efficient storage and query mechanism of the resulting ontologies to satisfy the expected functionalities.

It is estimated that billions of video are uploaded on YouTube and over billions of photos on Flick per year (Domingue et al., 2011). Furthermore, Multimedia data are found on almost every type of communication device. This widespread consumption of Multimedia data on the Internet has resulted in the development of several Multimedia ontologies (Chebotko et al., 2005; Verborgh & Van de Walle, 2011) which are stored and kept in various RDBMS over the Internet. However, the storage and query mechanisms of these Multimedia domain ontologies are largely unknown from the broader software engineering community due to the young nature of the field of Semantic Web.

Several ontologies have been developed to specify and describe the biomedical domain including UMLS (Unified Medical Language System) ontology (Lindberg, Humphreys & McCray, 1993), Gene ontology (Sidhu, Dillon & Chang, 2008), TAMBIS (Transparent Access to Multiple Bioinformatics Systems) ontology (Sidhu et al., 2008), Protein ontology that provide a unified and structured vocabulary for protein synthesis and concepts (Sidhu et al., 2006), and many more. An efficient technique is required to store and share these ontologies across software systems in the biomedical domain. At present, relational databases technologies are appropriate solutions. However, the underlying appropriate mechanisms and how to implement these solutions are not known.

1.2 Problem Statement

The widespread development and storage of large volumes of domain ontologies within various application domains of the Semantic Web such as Biomedical, e-government, e-commerce, e-business, e-Science, Multimedia and Audio Visual, etc. (Stadlhofer et al., 2009; VijayaLakshmi et al., 2011; Verborgh et al., 2011), there is need for a detailed investigation, analysis and application of existing storage and query mechanisms of ontologies in traditional relational database (RDB),

memory , and native file storage. There is also a need to provide a clear discussion of these mechanisms to the broader software engineering community that is still learning how to develop Semantic Web applications based on ontologies (Hesse, 2005).

1.3 Research Objectives

The main objective of this study is to analyze and apply existing Semantic Web mechanisms for storing and querying ontologies. Secondary objectives encompass:

1. To investigate existing storage and query mechanisms of ontologies.
2. To implement existing ontologies storage and query mechanisms using selected ontologies.
3. To analyze and discuss the performance of existing ontologies storage and query mechanisms.

1.4 Methodology

1.4.1 Data Collection

A literature search is used as the data collection technique in this study. Journals articles, conference papers and books related to the topics of semantic storage and query of ontologies are targeted.

1.4.2 Research Method

The Design research method is used in this study. It consists of following a set of predefined steps in order to solve a problem or create new knowledge (Hevner & Chatterjee, 2010). The steps consist of five activities namely: awareness, suggestion, development, evaluation, and conclusion. In this study the awareness stage identifies the need to have an architecture to test several ontology storage types and mechanisms. The suggestion stage proposed a framework for the empirical analysis and evaluation of existing ontology storage and query mechanisms. In the development stage, the required Semantic Web tools are identified to implement the framework. The evaluation stage uses a set of metrics to test the framework. These metrics include the loading times of ontologies, the disc space used to store the ontology repositories and the query response times. The evaluation stage also analyses and discusses the storage mechanisms implemented on Semantic Web platforms. Finally, the conclusion stage presents the outcome of the performance of the framework.

1.4.3 Implementation

In the development stage of the design method, Semantic Web platforms including Java Jena Ontology API, Sesame and Protégé (Ramanujam et al., 2009; Fan, Zhang, & Zhao, 2010; Zhou, 2010) are used to create and load selected RDF and OWL ontologies into different ontology storage types. The ontologies are stored into relational databases using MySQL RDBMS (Zhou, 2010). The stored RDF and OWL ontologies are queried using SPARQL ontologies query language (Magkanaraki et al., 2002; Ma et al., 2007).

1.5 Dissertation Outline

The rest of the dissertation is structured as follows:

Chapter 2 provides a literature review on what has been done in the field of ontology storage and query. Furthermore, the concepts of Semantic Web and ontology are defined and existing ontology storage platforms and mechanisms are presented.

Chapter 3 reviews existing research methodologies in the field of computer science and emphasizes the Design Research as the methodology used in this study.

Chapter 4 presents the framework developed using the methodology presented in Chapter 3. It describes the layers of the framework and compares it to related frameworks.

Chapter 5 presents and discusses the experimental results of the study.
Chapter 6 concludes the study and provides recommendations and future directions of the research.

1.6 Original Contributions

The main contributions of this research are as follows:

- A comprehensive review and discussion of Semantic web mechanisms for storing and querying ontologies (Chapter 2, Section 2.3). This theoretical work is relevant in computer science as it does not only provide theoretical information but it could also serve as the base for further development of ontology storage media.
- The design of a framework for the empirical application of Semantic Web mechanisms for storing and querying ontologies (Chapter 4). Unlike related studies, the proposed

framework enables the storage of several ontologies of different formats into ontology repositories and the analyses of performances of the mechanisms and platforms employed.

- The evaluation and comparison of ontologies storage and query under two popular semantic platforms, namely, Sesame and Jena. This work was published in the semantic web community in (Kwuimi & Fonou-Dombeu, 2015).
- The empirical analysis of the mechanisms used by Sesame and Jena to store ontologies in relational database. This work was accepted for presentation and publication in the proceedings of the IEEE ICACCE 2016.

1.7 Publications

The following publications resulted from this study:

Kwuimi, R. and Fonou-Dombeu, J.V. (2015) Storing and Querying Ontologies in Relational Databases: An Empirical Evaluation of Performance of Database-Based Ontology Stores, *In Proceedings of the 9th International Conference on Advances in Semantic Processing (SEMAPRO 2015)*, Nice, France, ISBN: 978-1-61208-420-6, 19-24 July, 2015, pp. 6-12.

Kwuimi, R., Fonou-Dombeu, J.V. and T. Zuva (2015) An Empirical Analysis of Semantic Web Mechanisms for Storage and Query of Ontologies in Relational Databases, *In Proceedings of the 3rd IEEE International Conference on Advances in Computing & Communication Engineering 2016 (ICACCE 2016)*, Durban, South Africa, ISBN: 987-1-5090-2576-6, 28-29 November, pp. 142-146.

Chapter 2: LITERATURE REVIEW

2.1 Introduction

This chapter introduces the concept of semantic web and ontologies. Semantic web is an improvement of the current internet which enables computers to communicate with each other autonomously. Ontologies are components used by semantic web applications to intelligently share data on the World Wide Web. Ontologies storage and query platforms are introduced; the most popular are: Kowari, Virtuoso, AllegroGraph, Minerva, Open Anzo, Oracle semantic, Sesame and Jena. These platforms implement one or more of the storage types, namely, in-memory, native and database. The main advantages and disadvantages of those storage types are presented. The storage mechanisms identified in the literature are discussed thoroughly and the literature on similar work is presented.

2.2 Semantic Web

Semantic Web, introduced by Sir Bernee-Lee around 2001 (Berners-Lee et al., 2001) is intended to enable computers to interact with each other with less human interaction or intervention. Semantic web is very popular in the ICT world, and many ICT companies such as Google, Facebook, Yahoo and many others have started using it (Splendiani et al., 2011). Semantic web provides search engines with the capabilities to perform meaningful search and provides relevant result (Chowdhury, Kaysar, & Deb, 2012).

Developing semantic web applications requires the building of a knowledge base model, namely, ontology. Different platforms and technologies have been developed to store and query semantic data represented as ontology. Further, the W3C consortium has adopted several standard languages for representing ontology such as RDF, Web Ontologies Language (OWL), and so forth. The ontology mainly represents the relevant data to be stored and manipulated by semantic web applications. They are the knowledge base model used by the applications, web services or any other computer software to store any knowledge about a particular domain. It provides common vocabulary to researcher who needs to share information on a specific domain (Zhang & Hobart, 2008).

2.3 Ontology Concepts

The word ontology was first used in the philosophy field (Gruber, 2013); then it was widely adopted in Artificial Intelligence (AI) more specifically in natural language processing, intelligent information integration and in Semantic Web (Corcho, Fernandez-Lopez & Gomez-Perez 2003). Due to its multipurpose usage in different fields, different definitions were adopted (Corcho et al., 2003). Gruber (2013) defines ontology as an explicit formal specification of the terms in the domain and relation among them. Ontology provides common vocabulary that can be used to facilitate information sharing in a domain (Imandi & Rizvi 2012). The reasons why ontology should be used differ as per field of expertise. But they can be summarized as follows (Noy & McGuinness, 2001):

- It allows common understanding of information structure among people or software agents,
- It enables the reuse of domain knowledge,
- It makes domain assumption explicit,
- It provides the separation of domain knowledge from operational knowledge and
- It allows domain knowledge to be analysed.

Ontology consists of several components. Let $\mathcal{O}$ represent an ontology, it contents six components which are $\mathcal{C}$ for concepts, $\mathcal{P}$ for properties, $\mathcal{I}$ for instances, $\mathcal{R}$ for restrictions, $\mathcal{A}$ for axioms, and $\mathcal{F}$ for facts such that $\mathcal{O} = (\mathcal{C}, \mathcal{P}, \mathcal{I}, \mathcal{R}, \mathcal{A}, \mathcal{F})$ (Alamri, 2012). $\mathcal{C}$ represents a set of concepts where each concept is represented as a class (Alamri, 2012) or URI (Xu, Lee, & Kim,2010). A class regroups all instances which share similar description or characteristics. Instances ($\mathcal{I}$) also called individuals are objects in the domain that are of interest to users (Alamri, 2012). Individuals are instances or occurrences of a class. Properties ($\mathcal{P}$), also called slots, represents binary relationships. Ontologies presents two types of properties namely objects and datatypes properties (Alalwan, Zedan, & Siewe, 2009). Objects properties are relationships between individuals of different classes while datatypes properties are relationships between individuals and data values (Alalwan et al., 2009). Object properties are divided into subcategories, namely, functional, inverse functional, transitive, symmetric, antisymmetric, reflexive, and irreflexive properties. Restriction ($\mathcal{R}$) is defined on properties. They set constraints and ranges on properties. Axioms ($\mathcal{A}$) are statements that define data in the ontology; they can be class or property axioms. Class axioms are statement about classes; they describe relation between classes; property axioms are statements about properties. Facts ($\mathcal{F}$) are statements or axioms about instances; facts define which individuals are related to each other through

relationships and restrictions.

2.4 Platforms for Storing and Querying Ontology

Several ontologies storage platforms have been developed in the past years. These platforms have some characteristics in common which are: storage type, inference support, update support, scalability and network distribution (Faye, Cure, & Blin, 2012). Storage type describes the system used to store the ontology; it could be memory based, file based or database based (Hertel, Broestra, & Stuckenschmidt, 2009). Inference support determines if the platform can create new knowledge from the stored ontology triples. Update support allows the modification of the content of the ontology stored in the repository. The scalability allows semantic web applications to manipulate ontology with larger triples (order of millions). Finally, network distribution allows ontologies storage platforms to handle bandwidth, network status and system usage (Faye et al., 2012). As mentioned earlier, several platforms have been developed to store ontologies. In the next paragraph, the most popular are presented including:

- Kowari (Wood, Gearon & Adams, 2005),
- Virtuoso (Erling & Mikhailov, 2009),
- Hexastore (Weiss, Karras, & Bernstein, 2008),
- Sesame (Broekstra, Kampman & Van Harmelen, 2002),
- AllegroGraph (AllegroGraph notes, 2015),
- Jena (Stegmaier et al., 2009),
- Open Anzo (Open Anzo notes, 2015),
- SOR or Minerva (Zhou et al., 2006; Lu et al., 2007), and
- Oracle Semantic (Stegmaier et al., 2009).

2.4.1 Kowari

Kowari initially was developed by Tucana Technologies between 2001 and 2004 and it became an open source project. Kowari is a storage platform that provides several APIs for storing and accessing

ontologies. Data are stored in Kowari using a native storage i.e. Kowari does not support memory and database storage. Data are stored in Kowari as quad statements; quad statements are statements that consist of four elements or four columns instead of three as in the normal RDF statement. The first three elements consist of elements of an RDF triple statement (subject, predicate, and object) and the fourth element represents the model or the graph from which the statement originated (Wood et al., 2005). Every statement is unique, therefore, if a statement appears twice in two different models, there will be two identical statements in the storage but with different model name (Wood et al., 2005).

2.4.2 Virtuoso

Virtuoso is another ontology storage platform that provides several backend database storages through Open Database Connectivity (OBDC) or Java Database Connectivity (JDBC) connectors. In addition, it supports XML and RDF triple store databases. It has a built-in web server that provides SOAP repository end points (Erling et al., 2009). Initially, virtuoso storage techniques were simple, i.e. it used a single table to store quads statement just as the Kowari platforms; this required three columns to store each element of a triple (subject, predicate, object) and the fourth column to store the graph to which the triple belongs. Virtuoso supports the SPAQL query language as part of the SQL language provided by the platform. Internally, all SPARQL queries are converted into SQL before being processed (Erling et al., 2009).

2.4.3 AllegroGraph

Another ontology platform, namely, AllegroGraph stores ontology as a graph (Stegmaier et al., 2009). It is a full application that does not require any particular configuration to run. Unless a virtual machine software is used, AllegroGraph cannot be installed on windows platform. AllegroGraph stores ontology in files also known as native files. It does not require any external databases. It is installed as a server application and requires client applications such as Java, C#, Python, Ruby, Perl or Lips to access it. Sesame and Jena can also be used as client applications for AllegroGraph server repository. AllegroGraph support the SPARQL query language and provides API for direct access to Subject, Predicates and Objects of ontology triples/statements. Five columns are used to store triples in AllegroGraph; three columns are for subject, object and predicate. The fourth column stores the graph to which the statement belongs and the fifth column a unique ID generated to identify each

statement (Stegmaier et al., 2009; AllegroGraph notes, 2015).

2.4.4 Minerva

Minerva also known as SOR (System for Ontology Storage Reasoning and Search) (Lu et al., 2007) is a component of the Integrated Ontology Development Toolkit (IODT). It is used as a plugin in Eclipse IDE. It stores OWL ontologies and support the SPARQL query language. Minerva uses four groups of tables to store ontology in databases; these include the class constructor tables, atomic tables, ABox tables and TBox tables (Zhou et al., 2006).

The class constructor tables are tables which are created to store OWL restriction quantifiers such as *someValuesFrom*, *allValuesFrom* and complex class description such as Intersection and Union classes. Atomic tables store all primitive classes in two columns in which the ID column identifies the class and the URI column represents the full URI of the class. These classes include Literal, Individual, *PrimitiveClass*, Datatypes and Property (Zhou et al., 2006). ABox axiom tables store the relationships in the ontology (Zhou et al., 2006) while the TBox tables stores ontology elements such as InversePropertyOf, DisjointClass, SubPropertyOf, Range, and Domain. Minerva supports SPARQL as query language and IBM DB2 and Derby as backend databases. At the time of writing, Minerva was unavailable on the IBM website.

2.4.5 Open Anzo

Open Anzo is a semantic web platform that can be used in three different modes: embedded in an application, installed as a server application and accessed remotely by client applications and run locally (Stegmaier et al., 2009). It stores data as quad store and uses databases as back-end stores. Further, it supports persistent storage through its Storage Service Layer which interacts with relational databases. In order to interact with Open Anzo, the client stack layer uses three different languages, namely, Java, Java Script and .NET (Stegmaier et al., 2009). Open Anzo supports the SPARQL query language and the typed full text search.

2.4.6 Oracle Semantic

The Oracle Semantic (Cheng, Kotoulas, & Ward, 2012) is a Jena API that works with Oracle databases (Stegmaier et al., 2009). It is a plug-in that implements Jena Graph and Jena Model interfaces. It also supports the SPARQL query language.

Oracle semantic not only stores ontology in database stores, but also has relational storage capability. Duplicated triples are not stored in oracle semantic; for each new triple, oracle checks if it is an existing triple with identical subject, property and object. Oracle semantic also checks if the triples are canonically equivalent i.e. if the values of the subject, property and object are identical.

2.4.7 Sesame

Sesame is a Software Development Kit (SDK) that was developed in the European IST project On-to-Knowledge (Askar, Siril, & Dougherty, 2005). It enables ontologies to be queried and exported. Two languages are used for ontology query in Sesame, namely, SPARQL and SeRQL. SeRQL is a proprietary repository for Sesame. The Sesame architecture has one component called the SAIL API which translates an ontology file into its RDB representation as well as enables Sesame to interface with DBMS, namely, MySQL and PostgreSQL. The SAIL API has three client modules:

- RQL query engine which is the engine used to query the repository through the SAIL API,
- the RDF admin module which is the module that allows data to be inserted into the repository during the loading of the ontology from the file to the repository, and
- the RDF export module which allows data to be exported from the repository into a file (Ye, Ouyang, Dong, 2010). Sesame can be used in a Graphical User Interface (GUI) with a browser or a Command Line Interface (CLI) mode.

2.4.8 Jena API

Like Sesame, Jena is an API that enables ontologies to be stored in three storage models: in-memory, native or RDB. Jena API is integrated into Eclipse IDE as a library and uses a variety of DBMS such as Oracle, PostgreSQL and MySQL (Astrova, Korda, & Kalja, 2007). The query languages supported by Jena API are SPARQL and RDQL. Jena can be used as an application by using the Jena Fuseki or as an API configured in Eclipse Integrated Development Environment (IDE).

Table 2.1 shows a summary of the platforms discussed. In the second column named license, all the platforms which are free and the source codes used to access them are identified. This feature will allow developers to improve it for better performance after testing and understanding how the mechanism works. A software which is available for free usually has all the features available. A trial version of a software usually does not make available all it features or reduces the capabilities

of some features. The third column named operating system, shows operating systems supported by platforms. The storage type column presents the different types of storages which could be memory based, file based of database based. SPARQL column shows if the platform support SPARQL query language. A table which contains a comprehensive list of RDF storage platforms identified by (Stegmaier et al., 2009) can be found in appendix B.

Table 2.1: Ontology storage tools summary.

Platform	License	Operating system	Storage type	SPARQL
AllegroGraph	Commercial/Free	Linux	Native	Yes
Jena	Free/Open Source	Windows/Linux	Memory, Native, RDB	Yes
Sesame	Free/ Open Source	Windows/Linux	Memory, Native, RDB	Yes
Open Anzo	Free/ Open Source	Windows Linux	RDB	Yes
Oracle semantic	Commercial/Free	Windows/Linux	RDB	Yes
Minerva	Free	Windows/Linux	RDB	Yes
Kowari	Free	Windows/Linux	Native	Yes
virtuoso	Trial/Commercial	Windows/Linux	RDB	Yes

Sesame and Jena will be used in this study due to the following reasons: (1) their storage mechanisms support all the three types of ontology storages (memory, native and RDB), (2) they both support the W3C standard SPARQL query language, and (3) they can be installed and used on the Windows and Linux platforms.

2.5 Types of Ontology Storage platforms

Ontology storage is based on 3 main types (Figure 2.1). These include: (1) In-memory storage, (2) Native file or File-based storage, and (3) Databases-based storage (Dieter et al., 2000; Hui**j**un, Wen**g**uo & Jian, 2011). In-memory or memory based storage uses the central memory of the computer to store ontologies. It is very efficient and fast with small scale ontologies. The drawback of this technique is that as the ontology become larger, it becomes more difficult to manipulate. In fact, ontologies stored using the in-memory storage technique need to be loaded in the memory every time a user wants to run an application that is using it. The native or file storage technique uses files to store ontologies. Ontologies statements are stored in triple store in the form of (S, P, O) where S

is the Subject, P the Predicate and O the Object. The advantages of native storage are that data loading and data query are fast (Heymans et al., 2008). In order to retrieve data easily and quickly with fewer errors, index algorithms such as the B-tree or B+ (Heymans et al., 2008 ; Hertel et al., 2009) are used. For structuring and editing of ontologies, they are very efficient as well (Zhou & Yongkang, 2013). The main drawback of native storage is that large scale ontologies are difficult to process. Furthermore, native storage needs to implement functionality such as data recovering, query optimization, controlled access and transaction processing in order to improve its data processing and management (Heymans et al., 2008).

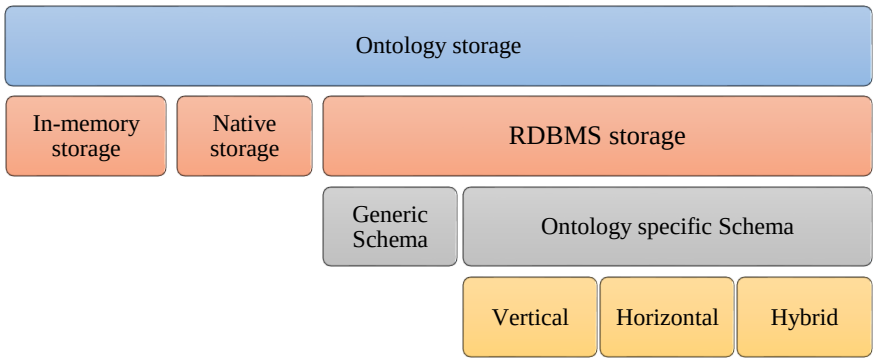


Figure 2.1. Ontology storage types

In the database-based storage, the ontology is stored in a Relational Database (RDB). Ontology storage in RDB needs to provide at least three of the following technologies: store and scalability, support for reasoning, and SPARQL query facilities (Lu et al., 2007, Yuang, Li & Wang, 2013, Zhou et al., 2013). Database-based storage is usually grouped into 2 main types (Zhou et al., 2013), namely, generic and ontology specific (Figure 2.1). The generic schema (Dieter et al., 2000) uses one table to store all triples or statements in the ontology. The table contains 3 columns, each representing an element of the ontology statement including Subject, Predicate and Object. Every row in the table is an ABox fact (Dieter et al., 2000). ABox are statements that describe the relationship between instances of the ontology (Zhou et al., 2006). TBox facts are ontology statements that describe relationship between classes and properties (Zhou et al., 2006). Many tables are required to store axioms or TBox facts of the ontology. Several architectures for DBMS have been developed to store Ontology data; they include triple table, vertical partitioning approaches and property table (Faye et al., 2012). The choice of which approach to use mostly depends on the

tools used for query performance or for Adding/Updating knowledge in the ontology (Faye et al., 2012). They are described in next section.

2.6 Ontology storage and query mechanisms

Ontology storage stores also called triple store systems (TSS) (Cheng et al., 2012), store data by following the RDF data model. Some of the TSS provides inferring and reasoning (Cheng et al., 2012). TSS have different architectures but follow the same principles which are (1) offer a triple store capability because RDF statement mostly consist of triple elements, and (2) provides a structure for inference engine to retrieve explicit and implicit answers from data sources (Alamri, 2012).

2.6.1 Triple Table mechanism

In triple table also identified as Generic Schemas in Hertel, et al. (2009), vertical table (Dehainsala, Pierra, & Bellatreche, 2005), schema oblivious (Theoharis, Chrisotphides, & Karvounarakis, 2005), or vertical representation approach by Luo et al. (2012), data are stored as RDF statement; a table contains three columns which are subject, property and object, respectively. Figure 2.2 shows a sample of triples from Levandoski & Mokbel (2009) ontology and Figure 2.3 shows how instances are stored in a triple table approach.

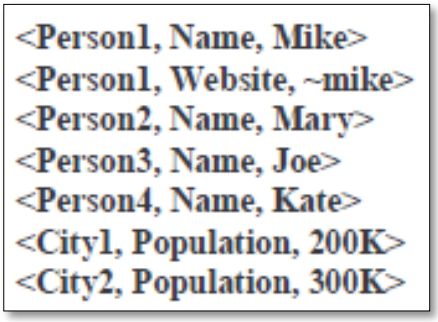


Figure 2.2: RDF triples (Levandoski et al., 2009)

Figure 2.3 shows the representation of triples from Figure 2.2 in a triple table approach. The subject column stores all the subjects of the ontology, while the property column collects all properties available and finally the object column stores all objects of the ontology. We observe that every instance in the table effectively represents an RDF statement.

TS		
Subj	Prop	Obj
Person1	Name	Mike
Person1	Website	~mike
Person2	Name	Mary
Person3	Name	Joe
Person4	Name	Kate
City1	Pop.	200K
City2	Pop.	300K

Figure 2.3: Triple table approach (Levandoski et al., 2009)

An improved version of the triple store also called a normalized triple store has two more tables added for resources and literals for the purpose of making join query less expensive (Hertel et al., 2009; Luo, et al., 2012). Instead of using the full name of the resources or the literals, indexes are used (Dehainsala et al., 2007; Luo et al., 2012). Those indexes are stored in tables under the column named ID (Identifier). Those IDs are used instead of the full name of the URI or the literals. The IDs are computed either as hash-based or as counter-based (Luo et al., 2012). The counter-based index sorts the resources lexicographically then assigns an identifier till all resources have their own ID; in such a case, a dictionary table is used for reference by other tables. Counter-based index has some variants when literals are too small to serve as an identifier; they are used without any further processing (Luo et al., 2012). When the ontology is very large such as DBPedia which is over one Gigabytes in size (Lehmann et al., 2012), the main memory might be out of size while processing complex queries that requires many unions and joins (Faye, et al., 2012). Simple statements are very fast as they do not require many self-joins. Triple store technique is used by Oracle (Al-Jadir, Parent, & Spaccapietra 2010, Faye et al., 2012), 3store, RDFStore (Faye et al., 2012). Figure 2.4 represents an improved version of the triple store approach in which two more tables are created for resources and literals and in the main table, IDs are used to represent the resources and literals.

Triples:

Subject	Predicate	IsLiteral	Object
r1	*r2*	*False*	*r3*
r1	*r4*	*True*	*l1*
…	…	…	…

Resources:

ID	URI
r1	*…#1*
r2	*…#2*
…	…

Literals:

ID	Value
l1	*Value1*
…	…
…	…

Figure 2.4: Improved triple store approach (Hertel et al., 2009)

2.6.2 Two tables mechanism

The two tables approach creates two tables, one for classes and one for properties (Al-Jadir et al., 2010). This approach, unlike the triple table mechanism, groups class instances in one table and properties instances in another. The class instance table contains two columns, namely, individual and class which contain instances' names and class names, respectively. The property table contains three columns, namely, domain, property and range (Al-Jadir et al., 2010). The domain column contains the individuals stored in the class tables, the property column represents the property in which the individual is linked and finally the range column adds some semantic to the property. The range value could be the individual from the class table.

2.6.3 Horizontal Representation approach

In horizontal representation, one table is used as in the vertical representation but it has more columns than in the vertical representation (Luo et al., 2012). In fact, for all properties in the ontology, a column is created in the table and a row for each subject in the ontology. If an ontology has ***n*** properties and ***m*** subjects, there will be a ***m*** x ***n*** matrix converted into a table of ***m*** rows with ***n*** columns. For triple (S, P, O), the object O is placed in the cell resulting from the S row with the P column. Then if a subject has a relation with only one specific column P, the remaining columns will have no values (Luo et al., 2012). In addition, if a subject has more than one object for a specific predicate there will be more multivalued attributes in a row.

a) Property table mechanism

The property table mechanism was introduced to minimize, and if possible to eliminate the empty cells in the initial horizontal approach. The technique allows multiple triple patterns referencing the same subjects to be retrieved without expensive join (Wang, Wang, Du, & Feng, 2010; Faye et al., 2012). Every Table contains a subject and several properties. Properties that share the same subjects are clustered (Wang et al., 2010, Faye et al., 2012). The first column as shown in Figure 2.5 stores subjects, the second and n^{th} columns contains the properties. An updated version of the property table called the property-class table makes use of the *rdf:type* property to cluster a set of similar subjects in the same table, and therefore self-joins are avoided when queries are issued (Faye et al., 2012). The technique is used by tools like Sesame, Jena, RDFSuite, 3store (Hertel et al., 2009; Wang et al.,

2010; Faye et al., 2012). Figure 2.5 shows how the property tables approach creates tables based on the properties that share the same subject. One can observe that the properties that share the same subjects are grouped together; thus reducing null values generated in the horizontal approach (Faye et al., 2012). More tables could be created based on the attributes or predicates of the ontologies.

NameWebsite		
Subj	Name	Website
Person1	Mike	~mike
Person2	Mary	NULL
Person3	Joe	NULL
Person4	Kate	NULL

Population	
Subj	Pop.
City1	200K
City2	300K

Figure 2.5: Sample Representation of Triple Statement in Property Table Approach (Levandoski et al., 2009)

b) Vertical partitioning mechanism

Vertical partitioning, also called decomposed storage schema in Levandoski et al. (2009), creates *n* binary tables where *n* represent the number of unique properties in the ontology (Wang et al., 2010; Faye et al., 2012). The first column in the table stores subjects, whereas, the second stores objects. This approach eliminates null values completely because every table that represent a property stores only one subject that uses them as part of the statement (Al-Jadir et al., 2010). As shown in Figure 2.6, properties are broken into tables and each instance of the table represent a triple of the ontology. This approach is similar to the multiple table presented in Al-Jadir et al., (2010). The main difference between the two approaches is that the multiple tables approach creates tables for every class and every property in the ontology.

Name	
Subj	Obj
Person1	Mike
Person2	Mary
Person3	Joe
Person4	Kate

Website	
Subj	Obj
Person1	~mike

Population	
Subj	Obj
City1	200K
City2	300K

Figure 2.6: Sample Representation of Triple Statement in using Vertical Partitioning Approach (Levandoski et al., 2009)

Figure 2.7 shows that ontologies storage in RDBMS uses two main mechanisms, namely, generic schemas and ontology based schemas (Hertel et al., 2009). The generic schema stores ontologies in one table and in more than one tables when indexes are used. The ontology schema defines the tables used based on the ontology; it consists of the two tables approach mechanism and the horizontal approach mechanism. The horizontal approach taken alone generates null values; the property and vertical partitioning mechanisms were introduced to reduce and eliminates null values.

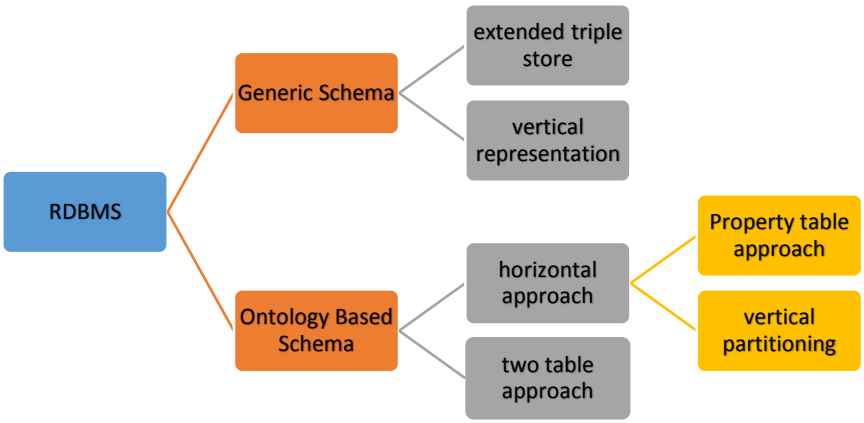


Figure 2.7: Ontology Storage Mechanisms in Relational Database

2.7 Related Work

Jalali, Zhou & Wu (2011) presented a study on RDF data management techniques. The techniques focus on storing RDF data in databases using three storage mechanisms, namely, Triple Store, Vertical Partitioning and Property Table. Jalali et al. (2011) show that storage efficiency using the Vertical Partitioning method is highly affected by the number of unique properties in the ontology, thus increasing the storage overhead of the schema data. On the other hand, the storage efficiency using the Properties Table method is not influenced by the number of individual properties. Instead, other factors such as the number of classes, and the average number of predicates per class have a higher impact on it (Jalali et al., 2011). The storage efficiency using Triple Store approach is independent of the number of individual predicates, thus making the storage overhead less time consuming.

Another study by Yuanbo, Zhengxiang, & Jeff (2005) proposed a benchmark of OWL knowledge base systems. The platforms used include: Sesame-Memory and OWLJessKB, Sesame-DB and

DBLB-OWL. A list of performance metrics including, Data loading, Query response time, Query completeness soundness, and Combined Metric Values are used. The result of the benchmark shows that loading time of OWL ontologies is proportional to their sizes, independently of the platform. Sesame-Memory demonstrated to be the fastest ontology platform in terms of loading time. But Sesame-Memory could not load large ontologies. Further, the experiments showed that loading large ontologies could only be done on DLDB-OWL.
The research by Yuanbo et al., (2005) and Jalali et al., (2011) have similar a drawback in that a specific dataset is used. Yuanbo et al., (2005) uses OWL ontologies and not the other ontologies format such as RDF, RDFS and TTL.

Faye et al., (2012) provided an overview of existing ontologies platforms. The authors compared the platforms against a list of features which are: data storage schema, storage, inference and update supports, scalability, query language, and network distribution. Another study in (Hertel et al., 2009), discussed storage models such as generic schema and ontology specific schema. The shortcoming of the study in Hertel et al. (2009) and Faye et al. (2012) is that they do not discuss which platform is efficient under a specific storage schema.

A survey on ontology indexing, more specifically on RDF indexing is presented in Luo et al., (2012). In this research, it is reported that indexing in relational database store uses unique identifiers. Those identifiers are computed in two methods, namely, hash-based and counter-based. Further, in order to improve scalability two indexing capabilities are used, namely, unclustered indexing and BTree clustered indexes.

In (Alamri, 2012), it is noticed that most OWL stores use one table to store ontology. Thus, the author proposed a new model in which tables are grouped in four layers whereby each layer represent one component of the OWL ontology, namely, classes, properties, instances, property restrictions and OWL axioms.

In Cheng et al. (2012), triple stores, namely, Jena, Sesame and RDF-3X are compared in order to provide insight on how to design future distributed systems and to optimize RDF data processing architecture. The work mainly focuses on the querying processes of triple stores. The experiments run three queries on the LUBM ontology. A data-centric approach is used to store RDF data in Levandoski et al. (2009). The method requires a tailored schema of the tables for specific ontologies

to minimize the use of join-query and increase the percentage of data values that are non-null. The technique consists of two main phases, namely, clustering and partitioning. Clustering minimizes the joins, whereas, partitioning increases non-null data values.

In Fonou-Dombeu, Phiri & Huisman (2014) examples of storing e-Government ontologies in relational database using Jena and MySQL are presented. The results reveal that storing e-government ontologies with Jena is proportional to the size of the ontologies because the bigger the ontology the greater the time needed to load the ontology. Furthermore, it is reported that the SPARQL queries execution times are proportional to the size of the ontology independently of the platform.

2.8 Conclusion

In this chapter, the concept of semantic web and ontologies were explained. Thereafter, the main components which make up an ontology were described. The ontology storage and query types which are in-memory, native and database were discussed. Popular ontology and query platforms were presented, and their main advantages and disadvantages were discussed. Jena and Sesame were the best ontologies platforms as they support all three storage and query mechanisms, they are open source and are multiplatform. The ontology storage and query mechanisms in database were discussed. Finally, related works in the field of ontology storage and query were presented.

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

This chapter presents research methodologies in the field of computer science such as design research, experimental research, creative research, descriptive research and many more. The characteristics of each methodology is discussed. The methodology used in this study, namely, design research is described in more detail. Firstly, design research is introduced; then all stages involved in design research which are awareness, suggestion, development, evaluation and conclusion, are described. Finally, the application of design research in the study is presented and discussed.

3.2 Research methodologies

Performing good research is based on using an appropriate and efficient methodology. Using a methodology increases the accuracy of results, and make these results more credible (Oates, 2005). Several research methodologies exist and are identified by Goddard & Melville (2004) and Oates (2005):

- **Experimental research**: in this type of research, a variable known as the independent variable or cause has its value changed; a second variable known as the dependent variable or effect is analysed every time the cause changes. Other variables which are neither independent nor dependent might still be considered in the experiment.
- **Creative research**: this research deals with the discovery or creation of new models, theorems, algorithms etc. It is less structured, and may not always be planned. Trial and error are the main methods used in this type of research.
- **Descriptive research**: also called "case-study" research, is a research method in which a specific case or situation is studied to see if it fits any existing general theories or if a general theory can be identified from the specific case.
- ***Ex post facto* research**: unlike experimental research, which moves from cause to effects, *ex post facto* moves from effects to causes. It analyses the effects and try to deduce the relevant and related causes.
- **Action research**: this research consists of identifying a problem; gather comprehensive data; some agreements are entered into by the researcher and the stakeholders who need the solution to

be implemented; the remedial actions are implemented, and after a period of time, an evaluation is performed to see if the problem has been resolved.

- **Historical research**: this method involves studying past events to identify effects-causes patterns. Current situations are examined using past effect-cause patterns, in order to predict future events.
- **Expository Research**: this research methodology is based entirely on existing information. Researchers widely read, compare, contrast, analyse and synthetize all points of view on a specific subject; new important insights can be developed as a result of that deep analysis and comparison (Goddard et al. 2004).
- **Design Research**: consists of following a set of predefined steps in order to solve a problem or create new knowledge. The steps consist of seven activities namely: awareness, suggestion, development, evaluation, and conclusion.

The next section describes the methodology used in this research which is *design research.*

3.3 Design research process

According to Friedman (2003), design is the process followed to solve a problem, create something new or improve something that already exists. Design is used in several domains such as engineering or architecture (Hevner et al., 2009). Research, as defined by Oates (2005), is done to create new knowledge or to contribute to the body of knowledge. Oates (2005) describe six important elements of research which must be considered: they are: purpose, product, process, paradigm, participants and presentation. The purpose indicates the reason why the research is undertaken. The product represents the outcome of the research; it represents the contribution to the body of knowledge. The process represents the sequence of tasks or activities undertaken for the research. Participants include people directly or indirectly involved in the research. They could be people that you interviewed, people editing the research report and more. Involving people must be done ethically and legally. Paradigm represents a pattern or shared model of thinking. The paradigms which can be used by a researcher are positivism, interpretivism, and critical-research (Friedman, 2003). The positivism paradigm defines the reality as anything that can be perceived by human senses. With interpretivism, the researchers interpret elements of the study or the society; the reality exists only in their mind. It is experienced through social interaction with several actors. In critical-research, the reality is created by people who tend to manipulate others and drive them to perceive things according to their belief.

Presentation is the means by which the research is disseminated to the public or stakeholder. It may be presented as a written paper, thesis and computer-based product.

Design research can be summarized as a process followed to develop or create new knowledge. Knowledge consists of artefacts that solves a problem or improves a situation. Simon (1996), defined artefacts as things that do not occur naturally but instead, are produced by humans. Hevner et al (2010) provides some guideline to follow when performing design research, which are:

- The output of the research must be an artefact used in the field of computer science. It does not have to be a completed product or artefact; enough knowledge can be gained in the analysis and the design, thus, the artefact does need to be completed for knowledge to be gained.
- The goal of the research must solve an identified and relevant problem.
- The resulting product or artefact must be evaluated using appropriate methods.
- The research process and creation of the artefact must add to the body of knowledge.
- A proper research process must be followed.
- A proper search and investigation into appropriate solutions must be done, the environment in which the artefact is applied must be taken into account during the search.
- A report of the research outcome must be provided to all involved parties.

The abovementioned guidelines by Hevner et al. (2010) aims to increase the value of design research. The design research process consists of five stages namely: Awareness, Suggestion, Development, Evaluation and Conclusion (Vaishnavi & Kuechler, 2004). Figure 3.1 shows the five stages involved in design research. The output of one process is used as the input in the next process as shown in Figure 3.1.

Figure 3.1: Design Research diagram (Vaishnavi et al., 2004)

The description of each layer with its role is described in the next section.

3.3.1 First Stage: Awareness

The main purpose of the awareness stage is to identify, recognize and state that a problem exists. Several methods exist on how to become aware of a problem to be solved, they are:

- Problems are identified in literature and future works by authors.
- Users have experienced a problem which need to be solved.
- New findings are made and,
- Technological developments are made.

The awareness stage produces outputs that are used in the second stage, namely, the suggestion stage. The output are statements describing problems needed to be addressed. Nevertheless, no solutions need to be found. But those statements will be of great help in proposing possible solutions and benefits. The goal of the research will be identified by those statements. The statements will be of great help during the development and the evaluation stages. In the development phase, they will be used as guidelines to follow while in the evaluation phase they will be used as criteria to evaluate the output.

3.3.2 Second Stage: Suggestion

Also known as the proposition stage, possible solutions to the statements identified in the awareness stage are formulated. Those solutions emerge from looking at currently available knowledge bases, and in related works. Some suggestions may be found in the literature done by other researchers. In

the case where no solutions have been identified, instigating new ideas as possible solutions is welcomed. As the output of this stage, a proposal of possible solutions is formulated.

3.3.3 Third stage: Development

In the development stage, one of the proposed solutions is implemented. The implementation is either partial or full. Partial implementation of a solution is considered when there are limitations such as time or resources. The output of the development stage, as identified by Oates et al. (2005) must be an artefact; thus traditional software life cycle methodologies should be followed. Software life cycle typically consist of five phases namely: requirements, design, implementation, testing and maintenance (Van Vliet, 1993; Kumar & Bhatia, 2014).

The main purpose of the requirement phase is to get a complete description of the problem that needs to be solved; also, it identifies all the conditions that should be met by the solution. The design phase proposes a model which represents the solution to be implemented. The implementation phase implements the model using a specific technology or programming language; the solution is physically implemented. In the testing phase, the solution is tested. The goal of the testing is to ensure that problems identified in the requirements phase are addressed. The maintenance phase helps fix errors which have gone undetected in the testing phase and other phases.

3.3.4 Fourth stage: Evaluation

This stage evaluates the proposed solution. The evaluation stage finds out if the questions identified in the awareness stages have been answered. Hevner et al. (2010) have identified three reasons why the evaluation phase should take place. The first reason is that the solution is applicable to the problem and that it adds value; the second reason is that the evaluation gives credibility to the research and, thirdly, it helps determine the practical value of the development or the experiment. Methods chosen to evaluate should be aligned with the research objectives (Oates, 2005). Evaluating the proposed solution can result in suggestions to modify the solution. A final conclusion must be taken.

3.3.5 Fifth stage: Conclusion

This last stage presents the output of the research. This stage presents the research and its contribution

to the body of knowledge. The contribution must be listed in such a way that all questions identified in the awareness stage are identified. The result of the research must be published and knowledge gained must be identified. The result could lead to further investigations that could be done or further research.

The output of design research is to provide an artefact. Oates (2005) identified and listed several artefact types which could be identified in the computer science field. They are:

- Constructs: these represent conceptual vocabulary of a specific domain which are constructed in the initial stage of design.
- Models: these are representation of relationships between constructs.
- Methods: a set of steps followed in order to perform some activities.
- Instantiations: which are the practical implementation of constructs, models and methods in a predefined environment.
- Theory developments: this is the development of new theories and improvement of existing theories in a knowledge domain.

3.4 Design Research applied to this study

As explained by Vaishnavi et al, (2004), research projects are unique and different from each other. Thus, general guidelines of the research process are adapted based on the research topics. The previous section presented design research and the different processes involved. The following section presents design research applied to this study.

3.4.1 Awareness

Since the inception of semantic web, several studies have been done to elaborate how semantic data also known as ontologies can be stored and queried. Those ontologies are stored in several types of storages such as in- memory, native file or relational database. Several platforms have been developed such as Minerva, AllegroGraph, Oracle Semantic, Jena API, Sesame SDK, and many more to integrate those storages. As more and more ontologies are being developed, the need to store them increases as well. Also the need to store data and retrieve results quickly and efficiently became evident. Thus using the right mechanism to store and query ontology is a goal to achieve. Several ontology storage and query mechanisms have been presented and discussed (Astrova et al., 2007; Lu

et al., 2007; Hertel et al., 2009; Alamri, 2012; Cheng et al., 2012). But a comparison and analysis of these mechanisms is missing in related works. Such an analysis would provide a comprehensive view of the existing ontology storage mechanisms. This study provides a framework for the empirical analysis and evaluation of existing ontology storage and query mechanisms.

3.4.2 Suggestion

Several studies and experiments have been performed in order to analyse and understand how ontology stores operate, perform, and process ontologies. As shown in Chapter 4, section 4.3, most studies focus on describing how a specific ontology type is stored on a specific platform using a specific mechanism (Gherabi, Addakiri & Bahaj, 2012), how several ontologies are stored on a specific platform using one mechanism (Xu et al., 2010; Fonou-Dombeu et al., 2014), and how one specific ontology namely LUBM, performs on different platforms using different mechanisms (Yuanbo et al., 2005; Cheng et al., 2012). In this study, several ontologies are used to evaluate ontologies stores. Using several ontologies of different formats (RDF, OWL, TTL) will provide a more global understanding of storage platform mechanisms; thus providing ontology independent results. To answer all statements or questions identified in the awareness stage, several storage mechanisms are tested on two popular ontologies storage platforms, namely, Sesame and Jena. Performance metrics identified by Agrawal, Somani & Xu (2001), Yuanbo et al. (2004), Zhou et al. (2006), Fonou-Dombeu et al. (2014), are used in order to perform the different comparisons and analyses.

3.4.3 Development stage

The development process can be different, based on the artefact being developed. The suggested solution is the elaboration of a framework. The framework provides simple models to test storage platforms and evaluate their mechanisms. In addition, it aims at providing different performance of ontology storage platforms and storage mechanisms. The design of the framework is fully covered in Chapter 4. The remaining part of this section presents the development activities within the framework.

a) Hardware and Software environment

The experiments are done on a Dell Optiplex 755 computer, Pentuim dual core 2 with 150G hard

drive and 4G of RAM. The operating system is a windows 7 ultimate, 32-bit operating system. The Java runtime environment (JRE) is the first application installed. It is used by Jena API and Sesame. The backend database used is MySQL. Tomcat server is used to run the Sesame as a server. Sesame can be used in different modes such as: as API or as a Server application that can be accessed by client machine remotely or locally. Jena API is used with Eclipse; thus a library which contain Jena is created in Eclipse. Eclipse is an Integrated Development Environment (IDE), used to write and run the Java code.

b) Setup and Implementation

The first application installed is Protégé. It does not require any specific configuration; Figure 3.2 shows an overview of protégé with OntoDPM.

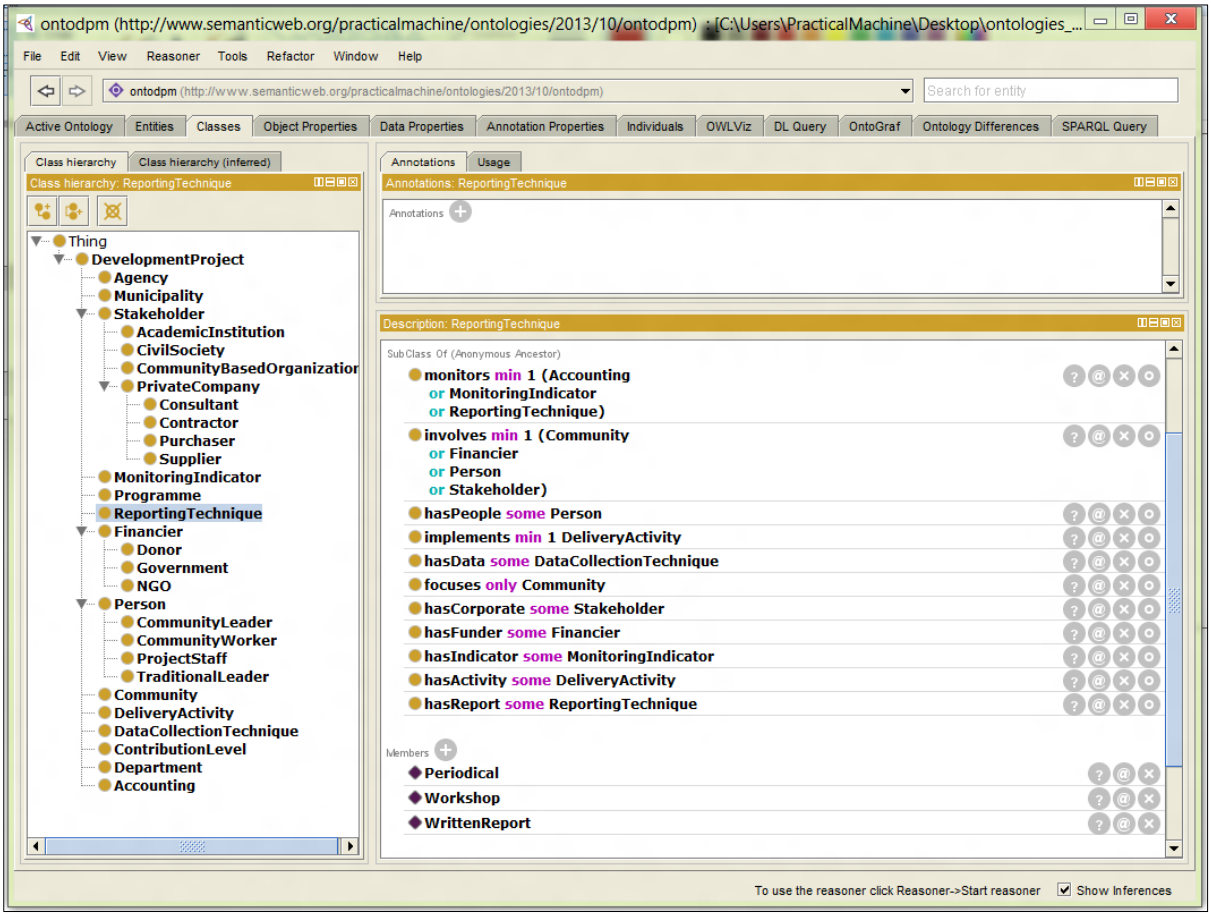


Figure 3.2: Overview of protégé GUI

After installing Protégé, the next application is the tomcat server which is required to run the Sesame SDK as a server application. Tomcat installation is straightforward and no particular configuration

is needed. After Tomcat has been installed, Sesame SDK is downloaded for free from the openrdf web site. To install Sesame SDK, two Java web application files are needed: *openrdf-sesame.war* which installs Sesame server and provides access to repositories and *openrdf-workbench.war* which provides the GUI to create, query and delete repositories in the Sesame server; these two files are located in the war folder of the unzipped Sesame SDK as shown in Figure 3.3.

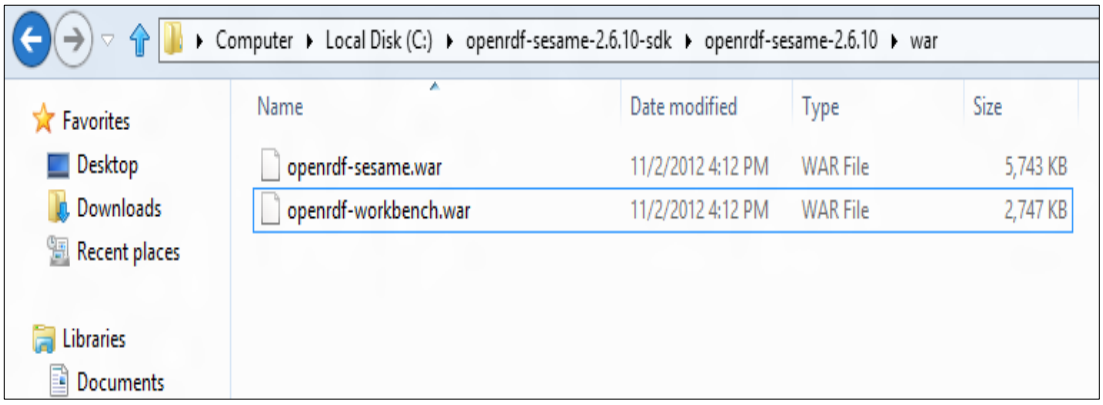


Figure 3.3: Sesame's .war files

The two files are copied and pasted in the tomcat *webapps* folder. When Tomcat starts, it automatically unpacks the two files and executes them. In order for the repositories to access relational database, an API named *com.mysql.jbdc_x.x.x.jar* where *x.x.x* is the version number is downloaded from MySQL web site and copied into the *lib* folder located in Tomcat folder as shown in Figure 3.4.

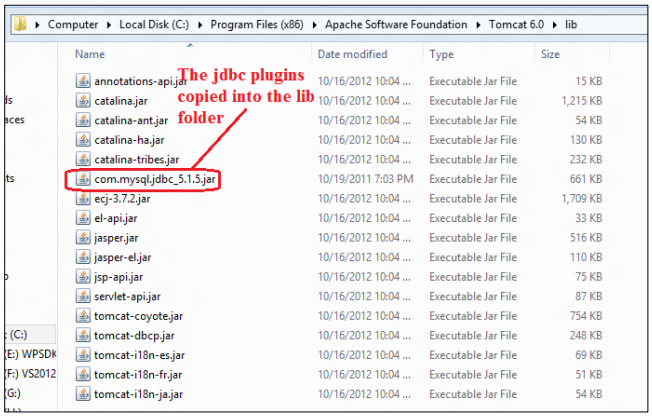


Figure 3.4: The JDBC file copied in lib folder

To start the Sesame server, "openrdf-sesame" is entered in the address bar as shown in Figure 3.5.

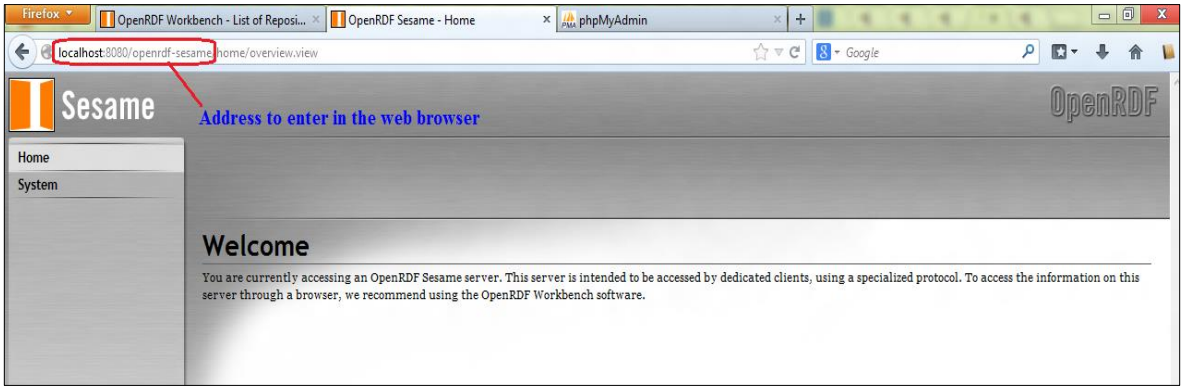


Figure 3.5: Sesame server main screen

In order to create repositories in Sesame, the workbench must be executed from the bowser, as shown in Figure 3.6.

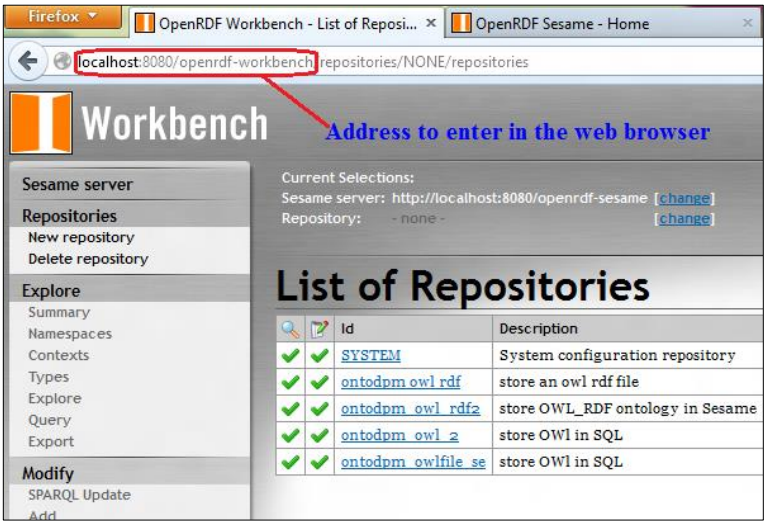


Figure 3.6: Sesame workbench

After installing Sesame and Jena, MySQL is installed. MySQL is used as the backend media in Sesame and Jena. Wamp server is a web server application which contains Apache, PHP, and MySQL. Installing the Wamp server does not require any particular or specific configuration. Figure 3.7 shows an overview of Wamp server with PHP, MySQL and Apache server.

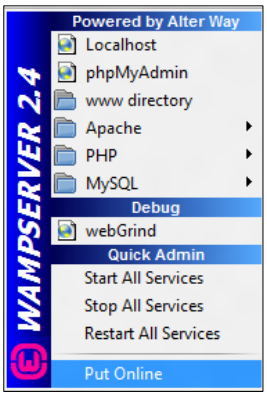


Figure 3.7: Wamp server menu after installation.

The query of the ontology in Sesame is done through the GUI provided by Sesame as shown in Figure 3.8.

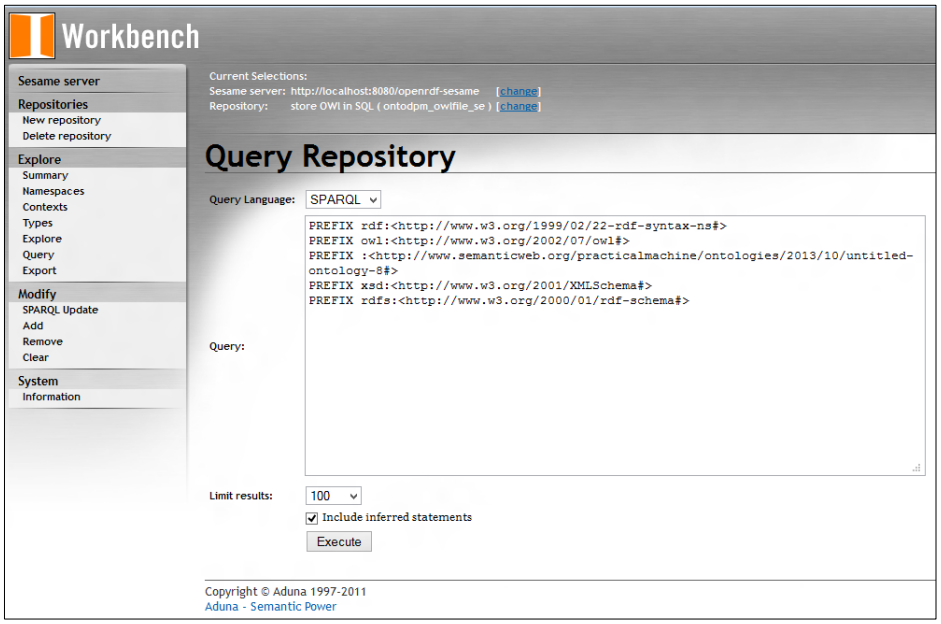


Figure 3.8: Sesame queries' screen

Jena API is used entirely in Eclipse and thus codes need to be written. Figure 3.9 shows the code used to create in-memory repository.

Jena_Memory.java

```
1  package com.jenaapi.fresh;
2
3  import java.io.InputStream;
16
17 public class Jena_Memory {
18     public static void main (String args[]) {
19
20         // Create an empty in-memory ontology model
21          System.out.println("===============Starting With Repository Creation============================");
22          long startTime = System.currentTimeMillis();
23         OntDocumentManager mgr = new OntDocumentManager();
24         OntModelSpec s = new OntModelSpec( OntModelSpec.RDFS_MEM );
25         s.setDocumentManager( mgr );
26         OntModel m = ModelFactory.createOntologyModel( s, null );
27          System.out.println("--------Ontology memory created --------------------");
28
29         // open file ontology to be loaded
30         String inputFileName="c://ontologies/gene.owl";
31         InputStream in = FileManager.get().open(inputFileName);
32         if (in == null) {
33         throw new IllegalArgumentException( "File: " + inputFileName + " not found"); }
34         System.out.println("----File successfuly loaded-----------");
35
36         // load the ontology into the memory repository
37         m.read(in,"");
38         System.out.println("===Ontology successfuly loaded in Memory ============================");
39         long estimatedTime = System.currentTimeMillis() - startTime;
40         System.out.println("===============End time Time for loading:"+ estimatedTime +"=================");
41
42         /*************************************************************************************************/
```

Figure 3.9: Jena Code to Create in-Memory Ontology Stores

The full codes used to create, delete, and query repositories is provided in appendix A.

3.4.4. Evaluation

In the evaluation phase, the testing of the framework is undertaken. The evaluation stage ascertains whether the framework does provide answers to concerns identified in the awareness stage. The evaluation consists of loading ontologies in the storage platforms using all backend media provided by the platform; the metric called loading time is used to identify which platform and which technique takes less time to load ontologies in repositories. A second metric called Query Response Time (QRT) is used to evaluate which platform and technique processes queries much faster. A third metric, namely, repositories size identifies which platform and technique uses more physical disk space. The fourth metric, *viz*, efficiency is used to compare the size of the ontology file against the size of the repository in which it is loaded. Finally, the storage technique of the database backend is empirically analysed to identify how ontologies are transformed into relational data. The full discussion of the evaluation of the framework is covered in Chapter 5.

3.4.5 Research outcome and contribution

The final phase of the design research methodology is the conclusion. Its goal or purpose is to elaborate on the findings of the research and share its outcomes (Vaishnavi et al. 2004). As stated by Vaishnavi et al. (2004), design research should create knew knowledge. Vaishnavi et al. (2004) categorise knowledge into two classes, namely "solid knowledge" and "loose knowledge". Loose knowledge represents knowledge that is not solved yet; knowledge with outstanding explanation; this type of knowledge lays the foundation for further research. While solid knowledge, is knowledge that can be applied by any party. The outcome of the development stage is a framework. The framework can be easily implemented. The tools needed are available and easily accessible; thus this research produce solid knowledge.

The main contributions of the research design are as follows:

- A comprehensive review and discussion of Semantic web mechanisms for storing and querying ontologies (Chapter 2 section 2.3). This theoretical work is relevant in computer science as it does not only provide theoretical information but it could also serve as a base for further development of ontology storage medium.
- The design of a framework for the empirical application of semantic web mechanisms for storing and querying ontologies (Chapter 4). The proposed framework is provided in Chapter 4 and applied in Chapter 5.
- The evaluation and comparison of ontologies storage and query under two popular semantic platforms, namely, Sesame and Jena. This work was published in the semantic web community in (Kwuimi & Fonou-Dombeu, 2015).
- The empirical analysis of the mechanisms used by Sesame and Jena to store ontologies in relational database presented in Chapter 5.

3.5 Conclusion

This chapter introduces methodologies applied to research in computer sciences. Design research which is the method used in this study is described and all stages involved are presented. Design research is applied to this study and the result of each stage is discussed. The awareness stage identified the need to have an architecture to test several storage types and mechanisms. The suggestion stage proposed a framework for the empirical analysis and evaluation of existing ontology storage and query mechanisms. In the development stage, the tools needed to implement the

framework are identified and presented. The evaluation stage presents the guidelines to test the framework. These guidelines include the loading of ontologies, the usage of metrics, the test of storage mechanisms and types on Sesame and Jena. Finally, the conclusion stage identified the outcome of the performance of the framework.

CHAPTER 4: FRAMEWORK OF APPLICATION OF SEMANTIC WEB MECHANISMS FOR STORING AND QUERYING ONTOLOGIES

4.1 Introduction

This chapter presents a framework of application of the semantic web mechanisms for storing and querying ontologies reviewed in Chapter 2. The first layer of the framework is the Ontology acquisition layer. The purpose of this layer is to acquire ontologies which will be used by other layers. The second layer is the Ontology Application Programming Interfaces (API) layer, used to create, edit, browse and delete ontology repositories. It is also used to load ontology into repositories. The third layer which is the backend media, is used by the repositories to physically store the ontology in the computer memory. Performance evaluation is the last layer. It uses metrics such as the loading time, query response time, disk space, and the storage efficiency, to test and compare the ontology storage, and query mechanisms implemented.

4.2 Presentation of the Framework

The framework consists of four layers as shown in Figure 4.1. The layers are: Ontology acquisition, Ontology API, Backend media, and Performance evaluation.

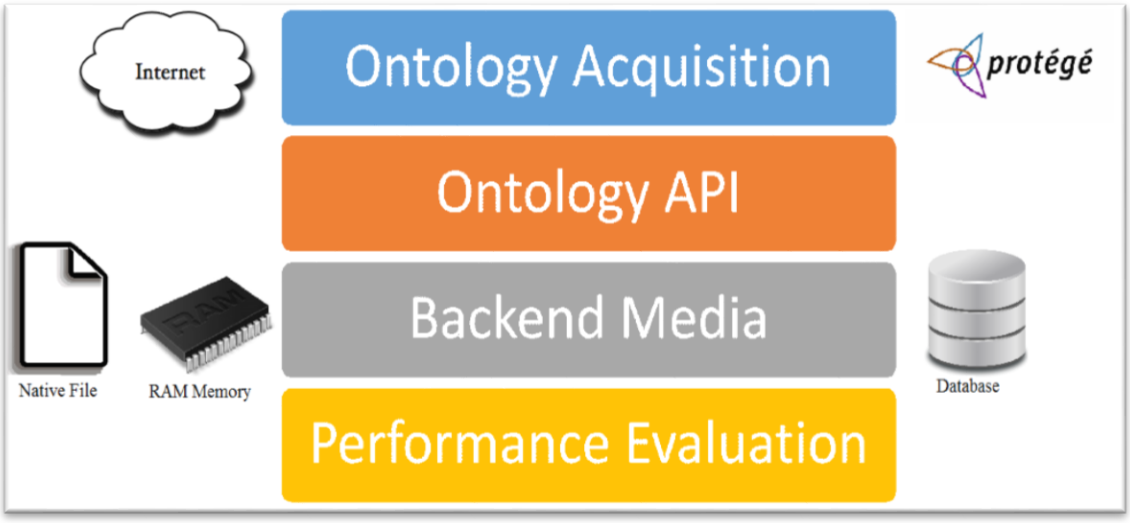


Figure 4.1: Framework for ontology storage, query and evaluation (Source: own research)

4.2.1 Ontology Acquisition

The otology acquisition layer focuses on acquiring ontologies that will be used. Ontology acquisition does not include the development of any new ontology. This layer solely uses ontology already developed and made available publicly. In order to get the ontology, the user must use the internet to download existing ontologies in different formats such as RDF, RDFS, OWL and N3. Once the ontologies have been downloaded, they can be used directly in the Ontology API layer. In case the user decides to create an ontology, he/she must first acquire a conceptual ontology. A conceptual ontology, is an ontology written in first order predicate logic, Cycl or Knowledge Interchange Format (KIF) (Kalibatiene & Vasilecas, 2011) which represents a specific domain ontology. The user must use an ontology editor to convert the conceptual ontology into a formal language such as OWL, RDF, DAM-OIL, N3, TTL or RDFS. Several ontology editors are available such as: Protégé, TopBraid composer, SWOOP, and OntoEdit (Sure et al., 2002; Kalyanpur et al., 2006; Alatrish, 2013). Ontology editors can be used as a desktop application or as an online web application; editors working as an online web application, require internet connection. One of the most popular ontology editor is Protégé. Protégé is freely available on the internet. It was developed by the University of Stanford at the Center of Biomedical Informatics Research in partnership with the University of Manchester.

4.2.2 Ontologies API

Ontologies API is the second layer of the framework as shown in Figure 4.1. It provides the interface to the user to create, edit, browse and delete the repository. It also provides the interface to load and to query ontologies in the repository using SPARQL query language.

An API is a set of controls, routine and protocols used by an application. They allow application development to be fast and efficient. Different types of APIs are available including Operating system APIs, database APIs, and web APIs (Dudhe & Sherekar, 2014). APIs can also be used to add new features onto an application and these types of APIs are called plug-ins (Dudhe et al., 2014).
Ontology APIs are APIs that manipulate ontologies. Most available ontologies API processes only one ontology format, thus, processing several ontology formats, requires one ontology API for each format (Horridge & Sean, 2011). In this study, to develop the repositories, multi-format APIs are used. These APIs allow the manipulation of several ontologies format such as RDF, OWL, and TTL. The APIs provide the main communication channels between the user and the ontology repositories. Several ontology APIs exist such as Sesame API, Jena API, and RDFox. Jena and Sesame are the only two APIs that cover all ontology storage technics such as in-memory, native file and Database, while RDFox only supports in-memory storage. Jena and Sesame are both open-source and are the APIs used in this study.

4.2.3 Backend Media

Ontology Backend media as shown in Figure 4.1, is the third layer of the framework. It implements ontology storage and query mechanisms identified in the literature review in section 2.5 and section 2.6. Its main purpose is to allow ontologies to be stored and queried physically in and from the computer memory, respectively. As presented by Dieter et al. (2000) and Theoharis et al. (2005), ontology storages are of three types: in-memory, native file, and database (Figure 4.1).
In-memory storage uses the computers main memory, meaning the RAM, as the storage media. Once the computer is off, the ontology stored is discarded and needs to be loaded again when needed again. Native file storage uses the computer physical disk drive to store ontologies as a single file. In database storage, a Database Management System (DBMS) must be provided to the repository. Relational databases have been used for years. Popular relational databases that exist are MySQL, SQL Server, Oracle Databases, DB2 and many more. In this study, MySQL is used because it is free and available on all operating system platforms. In order for the repository to use a relational database, it must use a database API which will link the repository to the database. The database used

as the backend by a repository must be created in the DBMS prior to the creation of the repository itself. Software packages used to create and manipulate repositories allow users to select or specify the repository backend as an option when creating a repository.

4.2.4 Performance Evaluation

Performance evaluation is the last layer of the framework. It provides metrics used to evaluate the ontology storage platform.

For ontologies storage platforms to be analysed and compared, several tools and methods have been set up. Metrics such as the repository size, loading time (Yuanbo et al., 2004), the Query Response Time (QRT) (Agrawal et al., 2001), the loading time (Yuanbo et al., 2005; Zhou et al., 2006; Fonou-Dombeu et al., 2014) and the storage platform efficiency (Jalali et al., 2011) are used to analyse and compare ontologies storage platforms. The loading time is the most commonly used performance metric in information systems (Auer & Ives 2007; Hertel et al., 2009; Kollia, Glimm & Horrocks 2011). It is used to represent the total time taken by platforms to process, parse and load the ontology in the repository. The QRT (Weiss & Bernstein 2009) represents the time taken by the platform to process a SPARQL query and display the results to users. To obtain the QRT, the ontology loaded in the repository is queried ten times. The values of the ten results are computed to get the average which represents the QRT. The repository size represents the total physical disk space used by the repository (Hertel et al., 2009) on the computer. The repository size is evaluated only on two of the three types of backend Media, because it is almost impossible to evaluate how much RAM is reserved for the data being used by a specific application. The final metric, which is the platform efficiency, identifies the efficiency of a storage platform. The efficiency is the ratio between the disk space occupied by the file of the ontology and the physical disk space used by the repository when the ontology is loaded in the repository. The platform with a ratio value closer to zero, is the most efficient platform; while the platform with a ratio value further from zero, is the less efficient platform.

4.3 Related work

Several ontologies storage platforms have been compared in order to determine which ones are efficient under specific conditions. Xu et al. (2010) presented a framework that they developed to store ontologies in relational databases. One of the main drawbacks of this study is that the

framework did not support any other storage type such as in-memory and native file storage. Only the database-based storage is supported. In this study, three storage type including in-memory, native file and database-based are used in the framework. A similarly study to those of Xu et al., (2010) and, Gherabi et al. (2012) developed an architecture to store OWL ontology. No provision is made to support other ontology formats such as RDF, RDFS, N3 or turtle. They mapped relational data into predetermined OWL structure file. Thus they limit ontology storage to OWL file only. This study uses RDF, OWL and TTL files to allow the developed ontology storage and query framework to be more accessible and increase its usefulness.

Stegmaier et al. (2009) presented a set of ontology storage platforms. They provide three categories of criteria that platforms need to have, thus providing a comparison framework of ontology storage platform based on those criteria. The first category provides general information about the platform such as: the software developer of the platform, the associated licenses that exist for the platform, the documentation of the platform, and the availability of the online support. The next category checks if the architecture of the platform can be obtained; if the programming language used is known, and if the framework covers the RDF format which is the most used and popular format. The last component identifies the available query languages supported, including SPARQL; it identifies how expressive the SPARQL queries results are and how the platform performs. The platforms which are used includes, AllegroGraph, Jena, Open Anzo, Oracle Semantic, and Sesame. The main drawback of the study is the fact that the performance test of the framework was entirely based on evaluating query performance, while in this study, other aspects such as loading time and storage efficiency were considered.

Stegmaier et al. (2009) argued that AllegroGraph does not respond to all their criteria, especially with regard to extensibility. While Jena and Sesame were extensible. Fonou-Dombeu et al. (2014) compared e-government ontologies in relational databases. They demonstrated that the Jena platform is scalable and can easily parse and store e-government ontologies in relational databases; and that the QRT was proportional to the size of the ontology.

Cheng et al. (2012) presented the benchmarking of three ontologies storage platforms. They focused solely on how the querying process is done. The LUBM ontology is used under three different storage platforms which are RDF-3X, Jena and Sesame. The research showed that on all systems, the

execution time of queries is proportional to the ontology file size. Further, the research showed that Jena takes less time in feeding back the query results to the user. The drawback of this work is that only one ontology was used in the experiment while in this study four ontologies are used.

4.4 Conclusion

In this chapter, the mechanisms used to store and query ontologies were identified. The framework with all the required components needed to store and query ontologies was presented. The framework consists of Ontology acquisition, Ontology API, Backend media and performance evaluation layers. Furthermore, the related works done in the field of ontologies storage and query were discussed. Four main differences between this study and related works were noted; they are: the study uses identified platforms (Sesame and Jena) to test all backend media types such as in-memory, native file and relational database. It also, through the different backend media, tests several types of storage mechanisms. It uses three different ontologies format (RFD, OWL, and TTL) instead of one as in other studies. Four metrics are used to increase the comparison criteria. The study also provides a detailed explanation on how ontologies are stored in relational database using Sesame and Jena.

Chapter 5: EXPERIMENTS

5.1 Introduction

This chapter presents the experiments conducted in this study. Firstly, the dataset used is presented. Secondly, the experimental results are presented and discussed. Finally, the mechanisms used by the platforms to store ontologies in database are identified and discussed.

The experiments consist of loading six ontologies in different types of repositories including in-memory, native file and database. The loading time of ontologies on each platform is also recorded in order to identify which storage mechanism enhanced its loading time. The ontologies are queried several times and the response times are recorded as well as the mean of successive query response times. Furthermore, the variance of each query result is calculated to determine how the individual query result is spread around the mean. The physical disk space used by each repository is also recorded to determine the efficiency of the mechanism used in the platform. Finally, the data stored are analysed to identify the mechanism or the group of mechanisms used on both platforms.

5.2 Experiment Requirements

This section presents the dataset, hardware and software used in this study.

5.2.1 Dataset

Six ontologies of different domains constitute the dataset in this study. They are OntoDPM and CGOV, which are ontologies used in e-government; DBLP which was developed for computer sciences; Gene Ontology and BioTop used in bio-science; and WordNet used in human and social sciences. The OntoDPM was created using Protégé and the other five were downloaded from the Internet.

a) OntoDPM

The OntoDPM ontology is a knowledge-based model for e-government monitoring of

development projects in developing countries (Fonou-Dombeu & Huisman, 2011). In fact, governments in developing countries are involved in developing infrastructures and providing better services to the citizens. Therefore they undertake projects that focus on building hospitals, improving healthcare, providing education, water, electricity and so forth (Fonou-Dombeu et al., 2011) to the population. OntoDPM may be used in e-government applications that interface those projects for better monitoring, transparency and the efficiency of projects. Figure 5.1 shows OntoDPM in protégé.

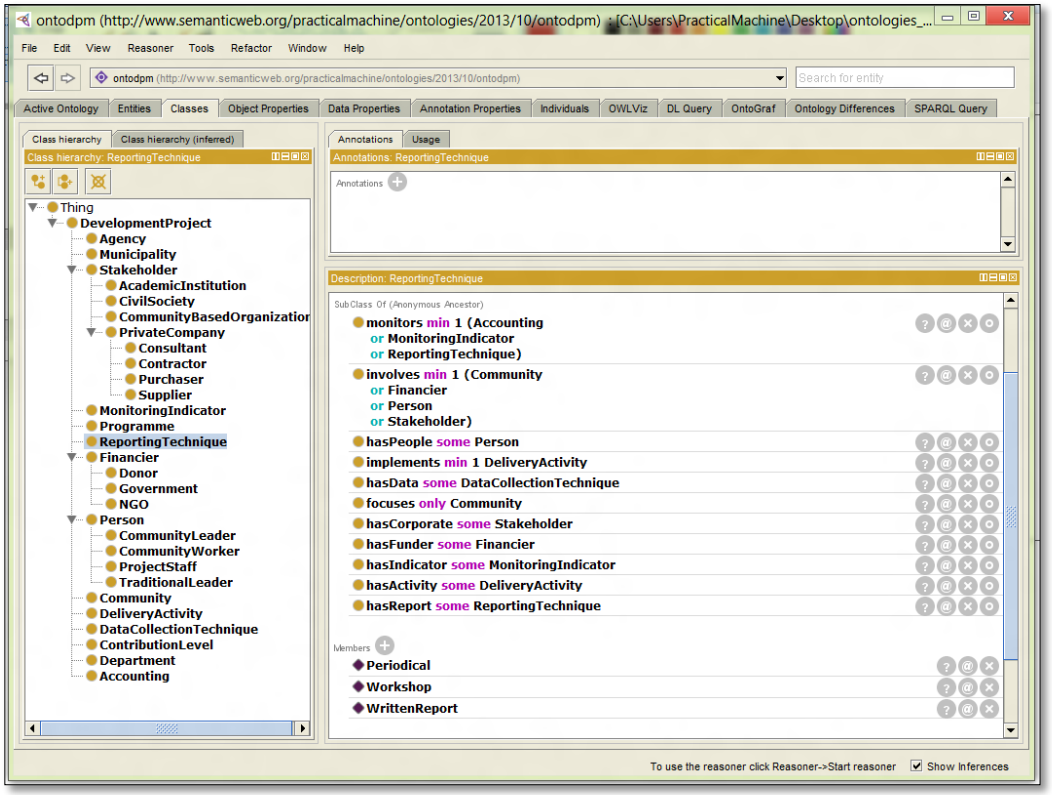


Figure 5.1 OntoDPM in Protégé

OntoDPM has 30 classes, 19 object properties and 18 individuals.

b) DBLP

Digital Bibliography and Library Project (DBLP) is an ontology of computer science research, created at the University of Trier in Germany. In 2014, it contained over 2.3 million articles (Ley, 2009) from conferences and journals in the computer science fields. The DBLP is very useful to researchers when they wish to trace work of colleagues and retrieve appropriate bibliographic data for referencing. The DBLP RDF file is written in XML format in which every parent tag represents

an article, and the child tags represent the attributes of the article such as author, title, pages and so forth. The DBLP ontology used in this study has 57 674 239 statements and weight 512 MB (Deng, King, & Lyu, 2008). Protégé could not load DBLP.

c) Gene Ontology

The gene ontology (GO) was created in the GO project which is a combination of three subprojects, namely, FlyBase, Mouse Genome Informatics and Saccharomyces Genome Database (GO Consortium, 2004). The GO ontology is a Directed Acyclic Graph in which the nodes are the concepts/terms connected to each other with "is-a" and "part-of" relations (Taha, 2013). The GO ontology (GO Consortium, 2004; Xu et al., 2007) covers three main biology domains: including molecular function, cellular components and biological process. The GO ontology contains the vocabulary/terms used in the biology field and the relationship between those terms (GO Consortium, 2004). Figure 5.2 shows a partial view of Gene Ontology in Protégé.

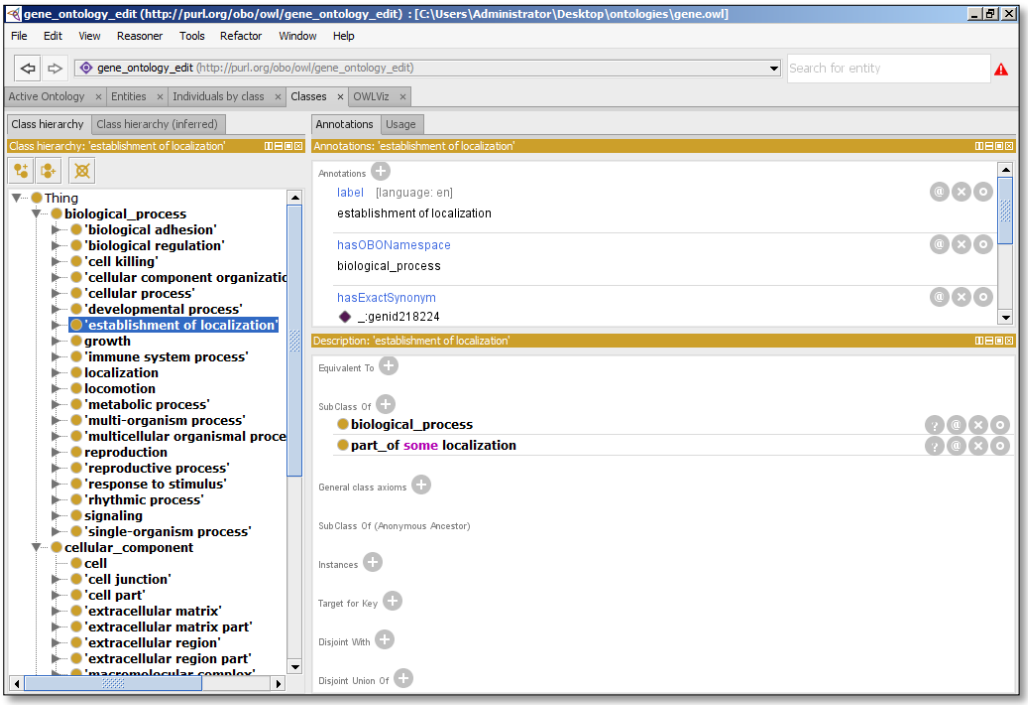


Figure 5.2: GO in Protégé

The GO ontology consists of 40 675 classes, 5 object properties, and 17 individuals.

d) WordNet

The WordNet ontology is an electronic lexical database for the English language. It contains verbs, nouns, adverbs and adjectives. Written in a machine readable format, online dictionaries access it for public usage (Miller et al., 1990). WordNet has three main concepts: the word, the word sense and the synonym sets (synset). The word is the basic lexical unit of WordNet; it represents the smallest piece of data inside WordNet. The sense is the context in which the word is used, specifically it represent the sense in which a specific word is used in a relation. The synset is the set of word with the same meaning; in other words, a synset groups synonyms together (Van Assem, Gangemi, & Schreiber, 2006). A synset is divided into 4 disjoint classes namely: adverbs, nouns, verbs and adjectives. WordNet has a total of 17 relationships where 10 are relationships between synsets, 5 are relationships between wordsenses, and the remaining two, gloss and frame, participates in relation between a synset and a sentence as well as between a synset and a verb construction patterns, respectively (Van Assem et al., 2006).

WordNet is a graph (Figure 5.3) where lexical units and sets of lexical units are represented by vertices, lexico-semantic relations by arcs. The WordNet used in this study weighted 100MB and contains word senses and words only.

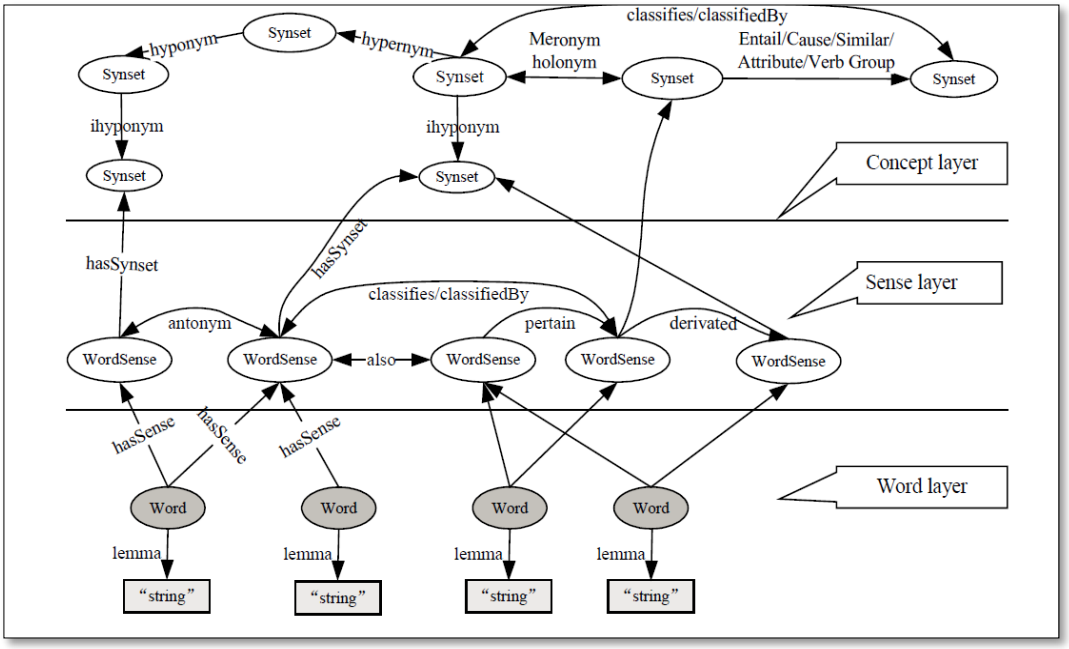


Figure 5.3. Overview of WordNet in OWL (GO Consortium, 2004)

Figure 5.3 shows a partial view of WordNet in protégé.It can be noticed that , WordNet has 7 classes

and 349 359 individuals

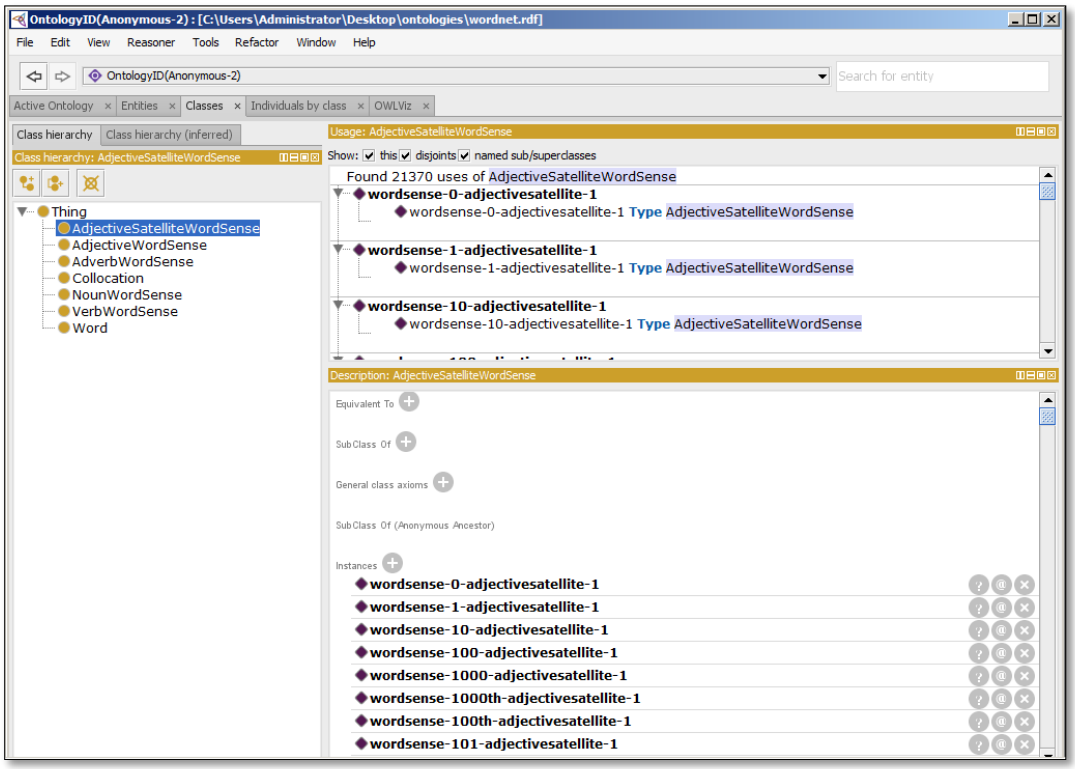


Figure 5.4: GO in protégé

e) CGOV

Central Government (CGOV) is an ontology of the United Kingdom government central government. The Friend of a Friend (FOAF) ontology is part of CGOV. FOAF describes social relationships amongst people. It also depicts people and their activities. CGOV adds the professional relationship on top of FOAF and thus enables CGOV to describe the social and professional relationship amongst government officials. CGOV, therefore, is used to model people, relationships with other people in the United Kingdom central government (Xu et al., 2007). Figure 5.5 shows CGOV in protégé.

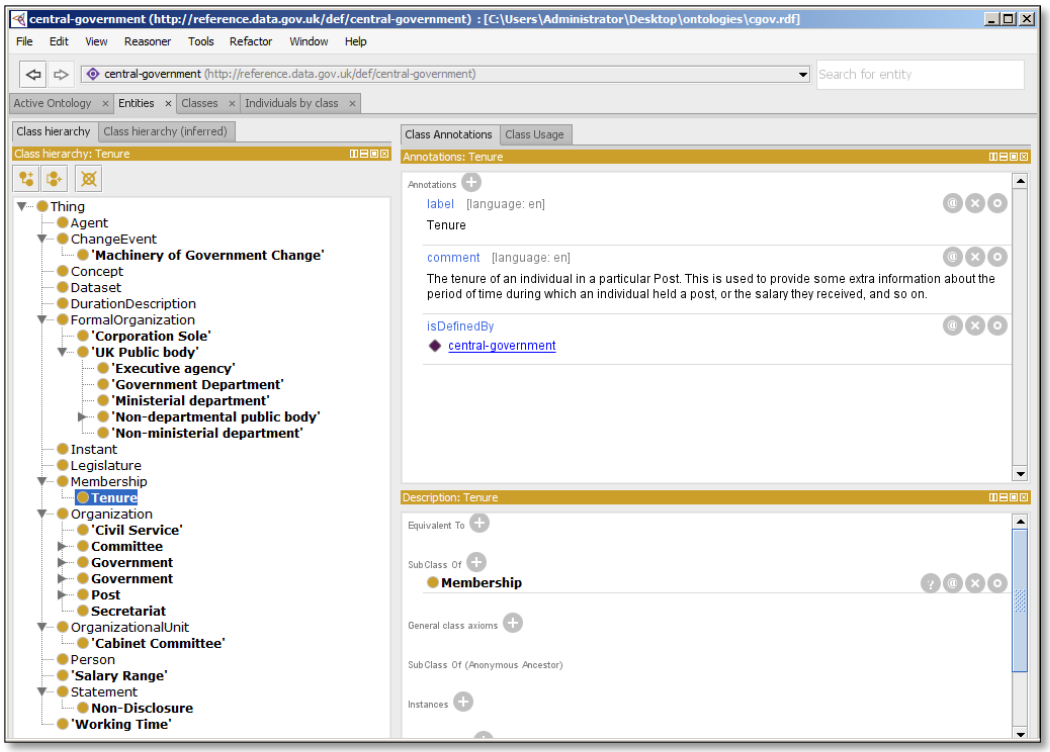


Figure 5.5: CGOV in Protégé

CGOV has a total of 59 classes, 53 object properties, 4 data properties, and 1 individual.

f) BioTop

The Biological Top Level (BioTop) is an ontology of the life sciences domain which focuses on molecular biology (Elena et al., 2008). It is used as a top level ontology to link the Open Biomedical Ontologies (OBO). Most of the OBO ontologies were independent from each other and lacked deeper conceptual and linkage integration (Schulz et al., 2006). BioTop offers the representation of implicit relation between the different life sciences ontologies found in OBO (Schulz et al., 2006). Figure 5.6 shows a partial view of BioTop in protégé.

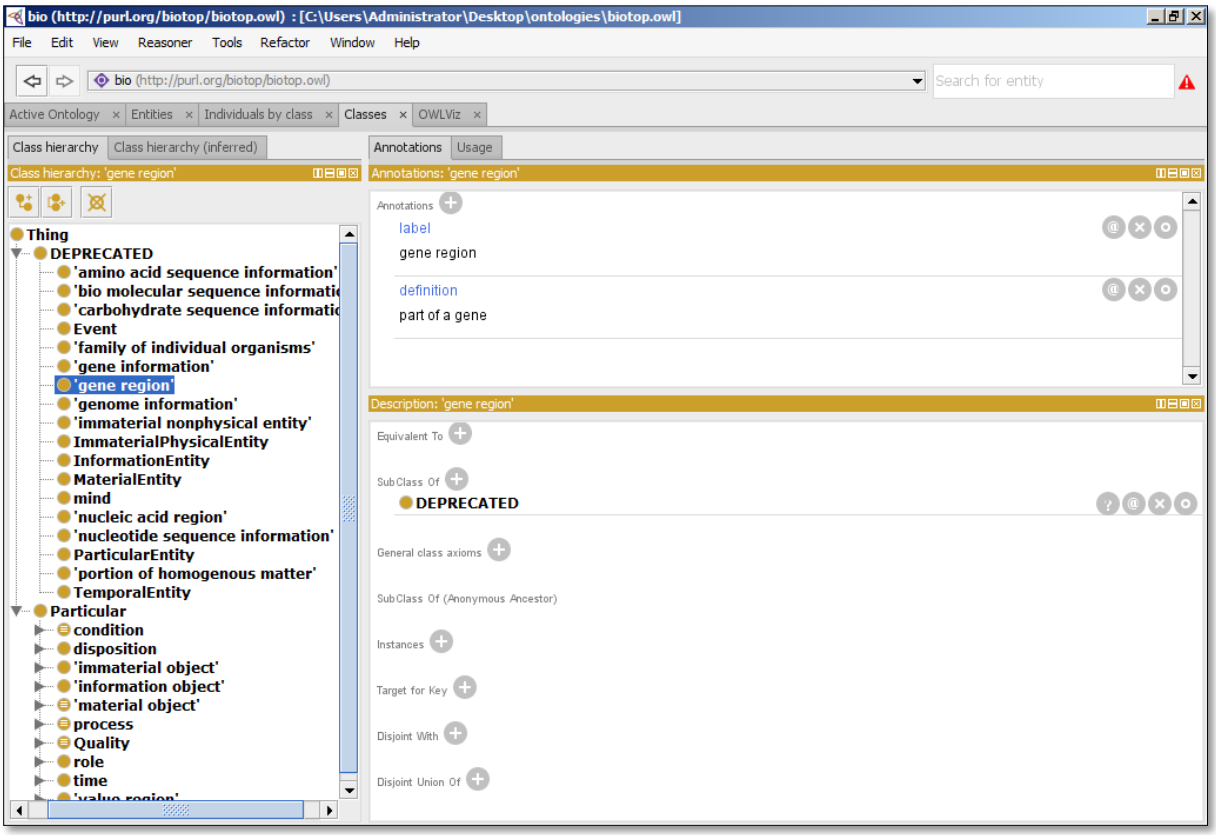


Figure 5.6: BioTop in Protégé

BioTop contains 390 classes and 83 properties. The characteristics of ontologies in the dataset are summarized in Table 5.1.

Table 5.1: Characteristics of Ontologies in the Dataset

Ontologies	Format	Size (bytes)	classes	properties	Individuals
OntoDPM	OWL	38 578	30	19	18
CGOV	RDF	68 551	59	57	1
BioTop	OWL	429 989	390	83	0
WordNet	RDF	100 428 111	7	0	349 359
GO	OWL	106 912 638	40 675	5	17
DBLP	RDF	-	-	-	-

5.2.2 Computer and Software Environments

The experiments were performed on a computer running Windows 8 64 bits operating system, with a Genuine Intel 2160 processor, a four gigabytes RAM and a 160 Gigabytes Hard drive. To create ontologies, Protégé version 4.3 was installed and used to create the OntoDPM ontology. To run and deploy Sesame SDK, the Apache tomcat server was installed. The MySQL server from the Wamp server was also installed to provide backend for database repository. To Use Jena API, Eclipse IDE version 4.2 was installed. It was used to write the Java code to load and query ontologies using Jena API.

5.3 Experimental Results

This section presents and discusses the experiments conducted in this study. The experiment consisted of loading six ontologies on all ontology storage types available including in-memory, native and relational database. Several metrics were used to record the different results and perform the comparison on both Sesame and Jena API platforms.

5.3.1 Analysis of Results of Ontology Storage in Computer memory

The in-memory storage type does not permanently store ontologies. Once the computer is off, ontologies are discarded. Two metrics are used in this storage type i.e. loading time and query response time. The variance is used to emphasize the QRT results.

a) Loading Time

The loading in Sesame and Jena memory was performed on all ontologies except on DBLP. The loading time of each ontology is presented in Table 5.2. The second column, shows the size of the ontology file; the third and fourth columns shows the loading time of each ontology in milliseconds (*ms*) in Sesame and Jena API, respectively. The last row of the table shows that DBLP could not load due to insufficient memory. The second column shows that loading time in Sesame memory is proportional to the ontologies' sizes.

Table 5.2: Loading Time of Ontologies in Sesame and Jena

Ontologies	File size (bytes)	Loading Time in Sesame (*ms*)	Loading Time Jena (*ms*)
OntoDPM	40 960	413	5 464
CGOV	69 632	204	1 878
BioTop	475 136	992	2 264
WordNet	100 429 824	120 505	76 263
GO	106 913 792	263 668	67 478
DBLP	591 462 400	Fail to load	Fail to load

Figure 5.7 shows that Sesame takes more time than Jena to store bigger ontologies. For smaller ontologies, Sesame took less time than Jena. Figure 5.7 shows that WordNet and GO took more time to be loaded on Sesame memory and took less time in Jena while the other three ontologies took more time in Jena than in Sesame. The figure shows that even though the loading time is not linear, bigger ontologies takes more time than smaller ontologies.

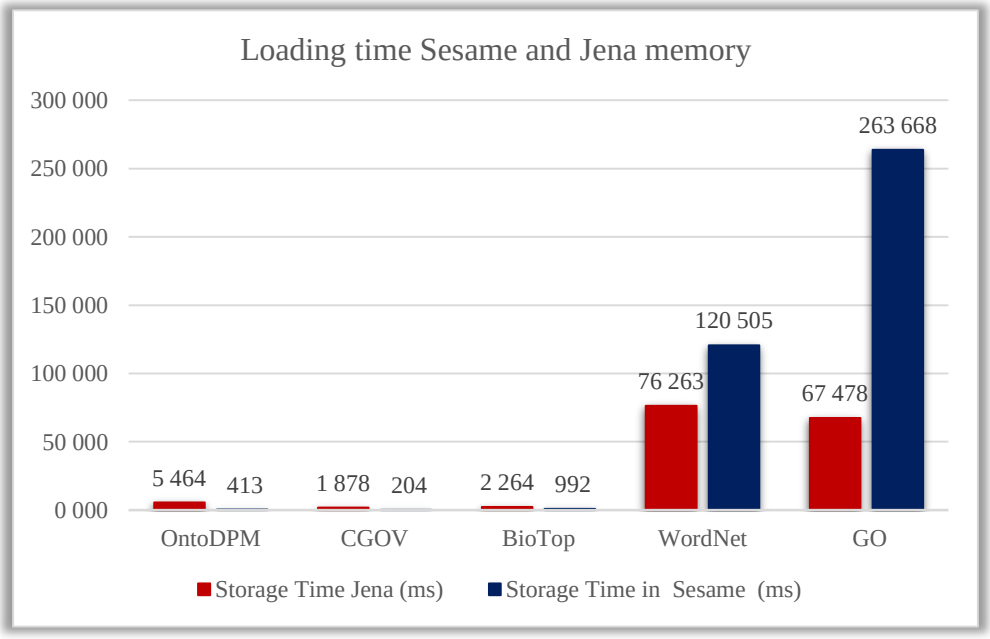


Figure 5.7: Chart of the Loading Times of Ontologies in Sesame and Jena API

b) Mean and Variance of Query Response Time

To perform the **Query Response Time (**QRT), a simple SPARQL query was used to query each ontology ten times, and the time in milliseconds taken by each query was recorded as shown in Table 5.3; the mean (μ) of those response time was computed using the formula in Equation 5.1.

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \qquad (5.1)$$

where n =10 and x_i the different recorded values of the response time.

The variance (s^2) of the queries was performed in order to compare the variation of the QRT Sesame and Jena API platforms. The variance is shown in the last column of Table 5.3. The formula used is presented in Equation 5.2.

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2 \qquad (5.2)$$

where N =10, represents the number of time the query was performed, and x_i the different recorded values of the query result.

Table 5.3: Mean and Variance of QRT of Ontologies in Sesame for in-Memory Storage Type

Ontologies	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	μ	s^2
OntoDPM	26	44	37	20	22	19	25	20	18	25	26	8.4748
CGOV	59	50	55	55	52	57	54	49	51	53	54	3.13581
BioTop	57	58	46	53	51	54	51	56	48	52	53	3.83551
WordNet	61	54	54	68	50	50	51	47	47	49	53	6.67416
GO	298	36	44	42	28	32	39	37	40	58	65	82.1181

Table 5.3 shows that the mean in Sesame memory is proportional to the size of the ontology; small ontologies take less time to calculate a result while bigger ontologies take more time to provide results. Table 5.4 shows the QRT, the mean and the variance obtained from Jena memory. The different values of the QRT for each ontology is shown columns Q1 to Q10, while the last two

columns show the mean and variance, respectively.

Table 5.4: QRT, mean and variance in Jena

Ontologies	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	μ	s^2
OntoDPM	1,435	1	31	1	16	1	1	15	1	16	152	203 388.4
CGOV	562	15	16	1	16	1	62	1	16	31	72	29 962.32
BioTop	530	1	1	16	15	1	16	1	1	16	60	27 348.62
WordNet	921	16	15	16	1	1	16	1	1	1	99	83 491.88
GO	687	15	1	16	1	1	1	16	1	1	74	46 439.11

The QRT mean graph in Figure 5.8, shows that Jena takes more time to return query results than Sesame. As shown in Table 5.4, the first time the ontology is queried; it takes more time to return the result, therefore, affecting the mean. While on Sesame, the QRT is linear. Figure 5.8 shows that queries in Jena memory is not always proportional to the ontology size; OntoDPM which is the smallest ontology has the highest mean. The highest mean shows that the query took very long to provide the results in Jena memory. GO which is the largest ontology took less time than OntoDPM, and WordNet.

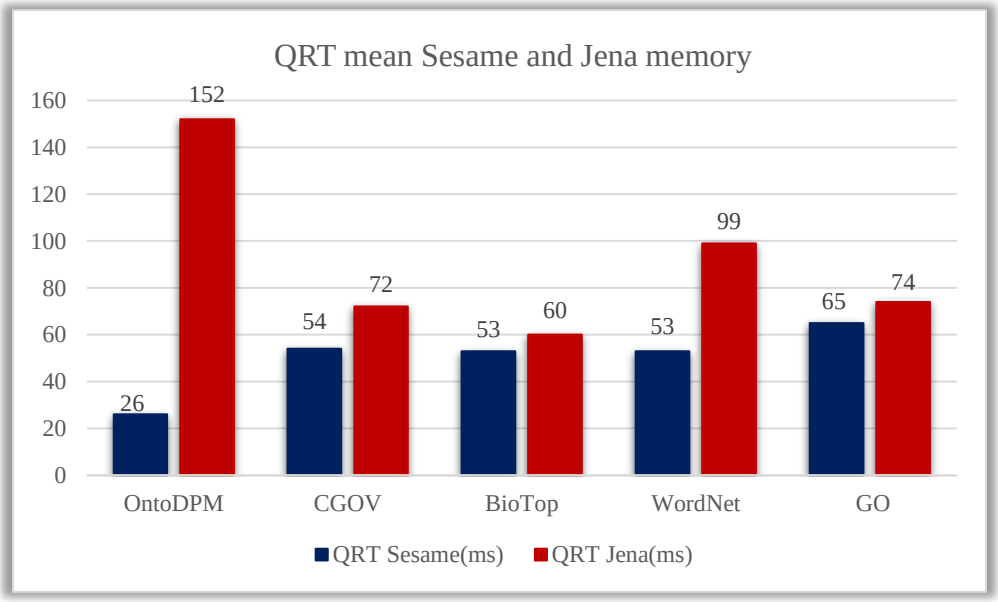


Figure 5.8: Mean of QRT in Sesame and Jena

Figure 5.9 shows the variance of Sesame and Jena memory. The variance graph shows that query on Sesame is more efficient than in Jena. QRT in Sesame are close to the mean, therefore, making their variance smaller as shown in the last column of Table 5.3. Table 5.4 shows that QRT in Jena are spread around the mean, therefore, making their variance higher than those of Sesame.

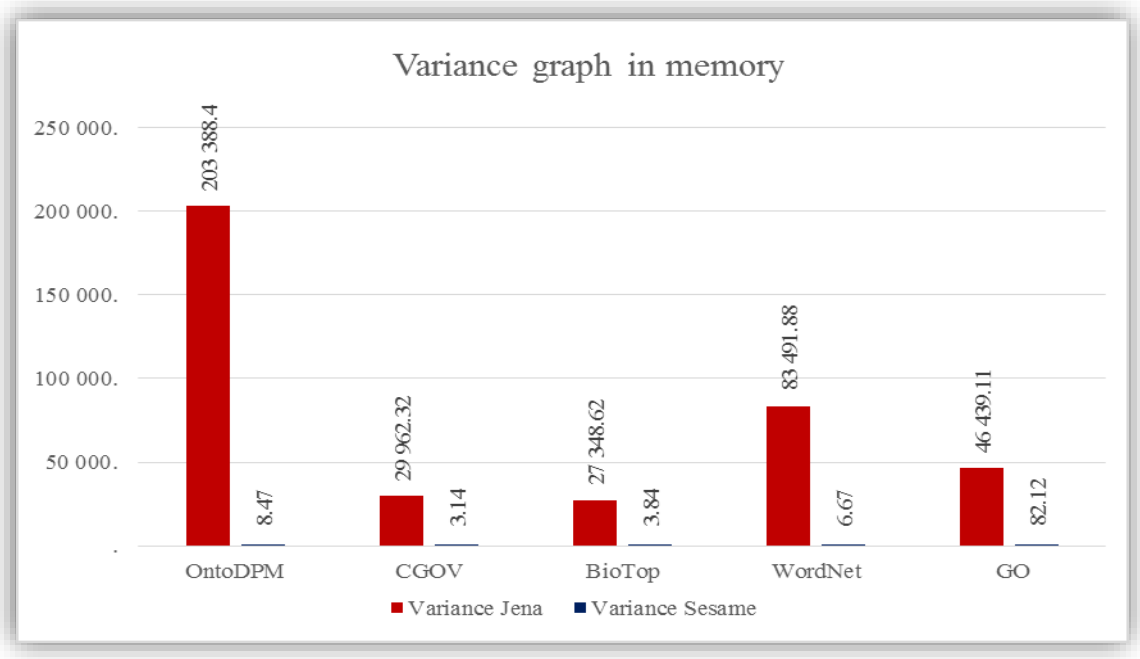


Figure 5.9: Chart of the Variance of QRT on Sesame and Jena API

5.3.2 Analysis of Results on Ontology Storage in Native File

This subsection discusses the native storage in Sesame and Jena. It uses four metrics which are loading time, QRT, efficiency, and storage space also known as repository size.

a) Loading Time in Native Storage

Table 5.5 shows the time taken by Sesame and Jena to load ontologies in native memories. The table shows that loading time is proportional to the ontology size in Sesame and Jena. Sesame native could not load DBLP ontology due to memory constraints but Jena native managed to load all ontologies including DBLP. It took almost a day and half to load DBLP which is just 500Mb in size.

Table 5.5: Loading Time in the Native Storage in Sesame and Jena

Ontologies	File size (bytes)	Loading time Sesame (*ms*)	Loading time Jena (*ms*)
OntoDPM	40 960	711	3 112
CGOV	69 632	444	3 183
BioTop	475 136	2 417	4 899
WordNet	100 429 824	449 948	515 225
GO	106 913 792	1 045 907	551 166
DBLP	591 462 400	Fail to load	100 245 820

The data in Table 5.5 shows that storage in Sesame native is faster than that in Jena. Figure 5.10 shows the graph with DBLP included; due to the amount of time taken by Jena to store DBLP the others ontologies bars are very short.

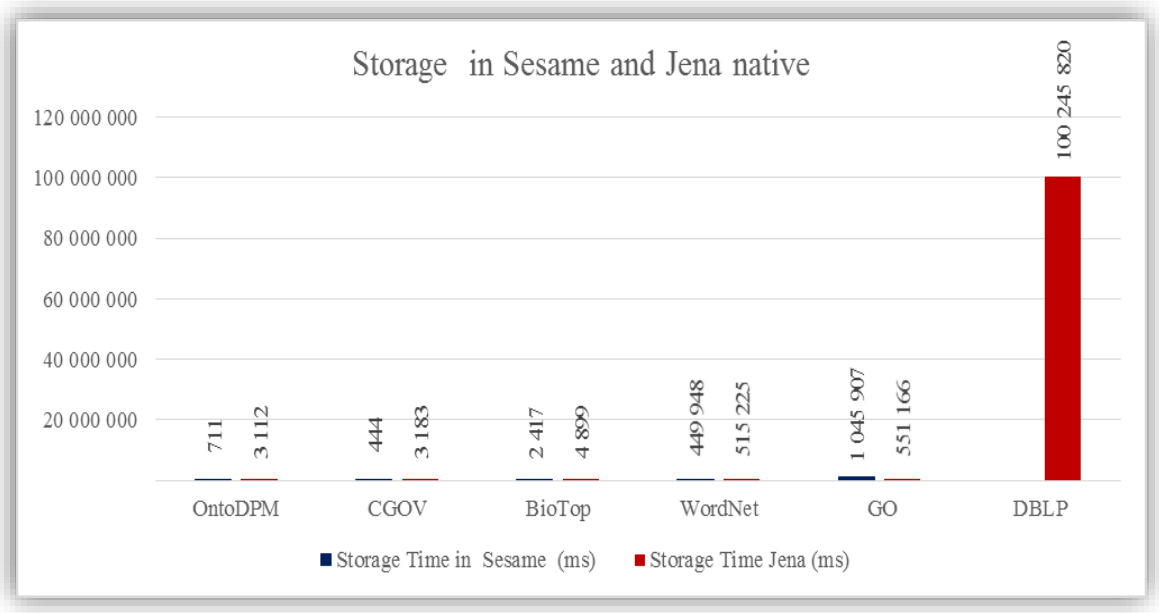


Figure 5.10 Sesame and Jena loading time

b) Mean and Variance of QRT in Native Storage

The QRT in Sesame native is not proportional to the ontology file size as shown in Table 5.9. The last two columns shows the mean and variance of QRT, respectively.

Table 5.6: Mean and Variance of QRT in Native Storage in Sesame

Ontologies	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	μ	s^2
OntoDPM	584	28	69	46	34	56	57	67	42	62	105	25 715.25
CGOV	417	38	50	55	47	48	49	45	54	51	85	12 238.24
BioTop	181	44	42	44	40	42	38	50	43	40	56	1 735.44
WordNet	534	40	42	36	39	40	41	46	38	30	89	22 057.84
GO	664	42	42	43	40	41	45	44	40	37	104	34 873.96

Table 5.7 shows the time taken by each query on each ontology on Jena native. The last two columns shows the mean and variance of QRT, respectively.

Table 5.7: Mean and Variance of QRT in Native Storage in Jena

Ontologies	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	μ	s^2
OntoDPM	687	1	1	31	1	16	1	15	1	16	77	46 042.44
CGOV	515	1	1	1	31	1	16	1	1	16	58	25 844.27
BioTop	454	15	1	16	1	1	15	1	16	1	52	19 993.21
WordNet	3 433	16	1	1	1	31	1	1	15	1	350	1 173 470.77
GO	5 991	16	1	1	1	31	1	16	1	1	606	3 580 133.33
DBLP	17 723	1	32	1	1	15	1	1	31	1	1 781	31377 557.79

A comparison of mean of QRT time in both Sesame and Jena API is provided in Figure 5.11. The figure shows that the mean of QRT and loading time in Sesame native, is not proportional to the ontology file size. The chart in Figure 5.11 shows that in Jena native, small ontologies takes more time to respond; but as the ontology grows in size, the mean of QRT is proportional to the ontology size; the bigger the ontology the bigger the mean.

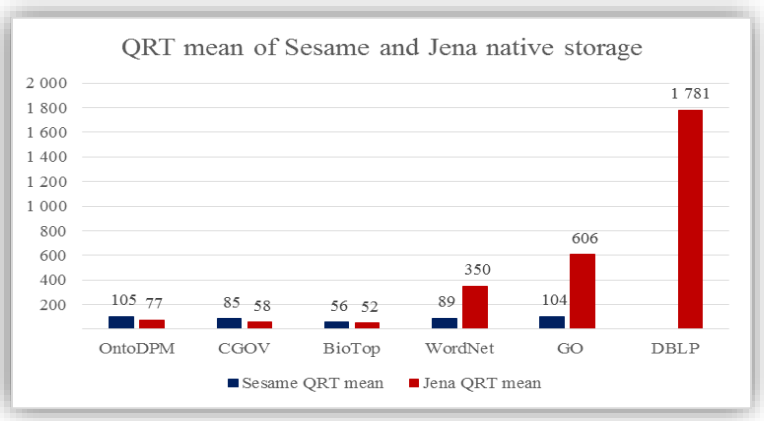


Figure 5.11: Chart of Comparison of the Mean of QRT in Sesame and Jena

Similarly, Figure 5.12 compares the variance of QRT in Sesame and Jena API. The Chart of variance of QRT (Figure 5.120 shows that Sesame performs better than Jena. The graph shows that the variances of QRT on Sesame are all smaller than those in Jena.

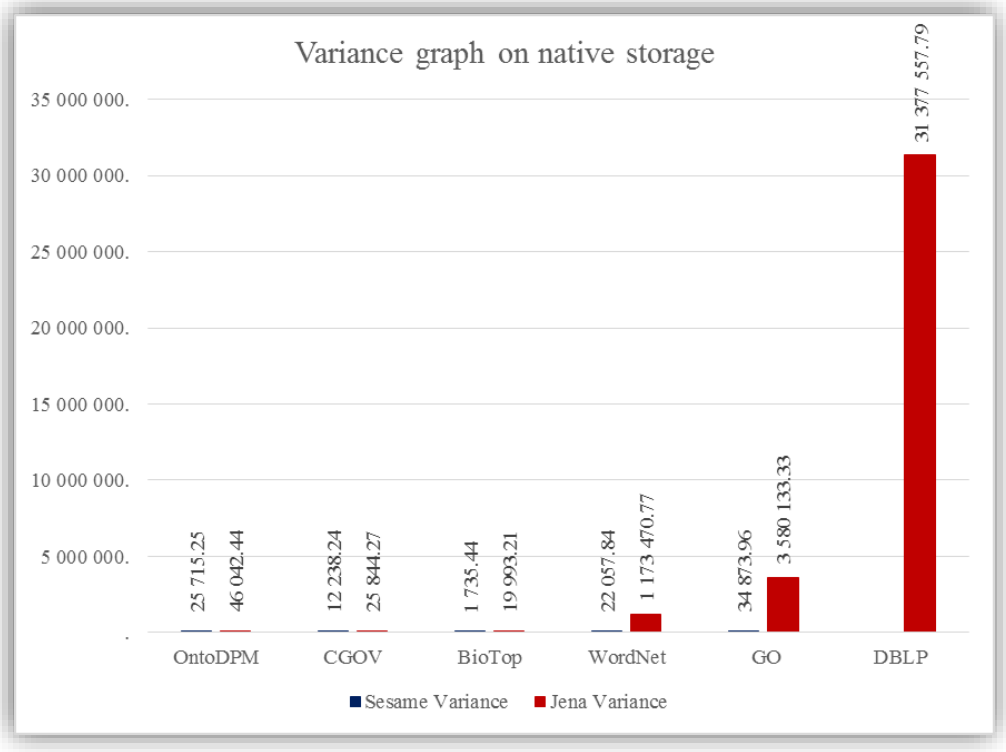


Figure 5.12: Chart of Comparison of Variance of QRT in Sesame and Jena

c) Disk Space and Efficiency in Native Storage

The disk space in Sesame and Jena native is presented in Table 5.8. The efficiency columns show that storing ontologies in Sesame native required between 1.5 to 2.5 more space of the actual file ontology on the disk to store its repositories. To store in Jena native, at least 4 to 6 times the size of the original file will be required. Thus, storing in Sesame native is more efficient than storing in Jena native. The storage space and efficiency on Sesame for DBLP are not available because Sesame was not able to load DBLP.

Table 5.8: Disk Space used to Store Ontologies in Sesame and Jena in Native Storage

Ontologies	File Size (bytes) $S(D)$	Disk Space $S_M(D)$		Efficiency $SE_M^D = \frac{S_M(D)}{S(D)}$	
		Sesame	Jena	Sesame	Jena
OntoDPM	40 960	102 400	258 048	2.5	6.3
CGOV	69 632	163 840	389 120	2.35	5.59
BioTop	475 136	966 656	2 162 688	2.03	4.55
WordNet	100 429 824	154 259 456	546 533 376	1.54	5.44
GO	106 913 792	230 617 088	553 869 312	2.16	5.18
DBLP	591 462 400		3 349 827 584	2.5	5.66

Figure 5.13 shows that Jena native takes more space than Sesame native. For all ontologies, Jena took more space than Sesame to store the ontologies repositories.

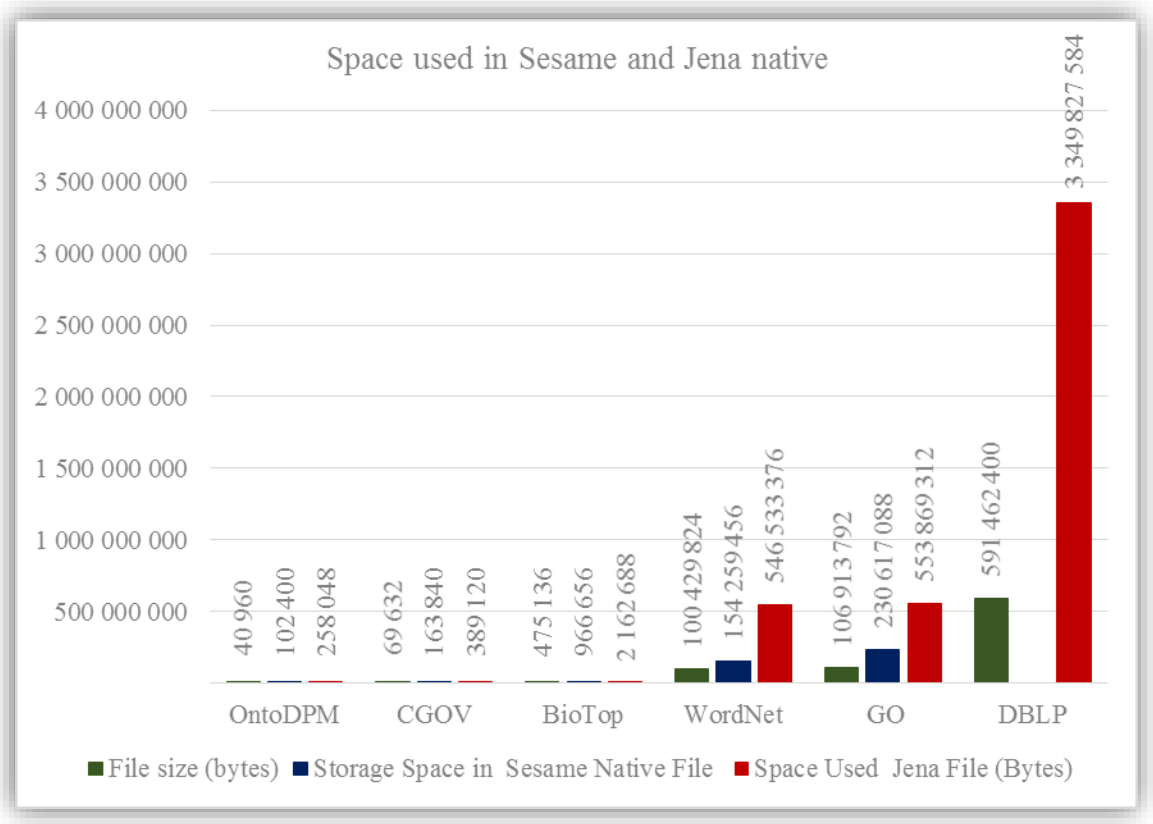


Figure 5.13: Chart of the Comparison of Disk Space in Jena and Sesame

5.3.2 Analysis of Results of Ontology Storage in Relational Database

This subsection presents and discusses the experiment on the relational database (RDB) storage type. The discussion covers the metrics, namely, loading time, disk space, and mean and variance of QRT.

a) Loading Time in Database Storage

Storage in RDB was not possible with DBLP because Sesame had insufficient memory space; while Jena RDB was able to load DBLP after almost 3 days. Table 5.9 shows the loading times of the ontologies in the dataset in Sesame and Jena into RDB.

Table 5.9: Loading Times of Ontologies into RDB in Sesame and Jena

Ontologies	File size (bytes)	Loading Time (*ms*)	
		Sesame	Jena
OntoDPM	40 960	57 629	15 570
CGOV	69 632	109 690	24 103
BioTop	475 136	296 213	138 389
WordNet	100 429 824	232 907 489	49 191 675
GO	106 913 792	444 858 955	33 498 626
DBLP	591 462 400	Fail to load	259 720 259

The chart in Figure 5.14 shows that loading time in Sesame database is proportional to the ontology file. Big ontologies take more time than smaller ontologies. Jena on the other hands took less time than Sesame to load the ontologies in the database including the DBLP. Figure 5.14 shows that the database loading time in Sesame and Jena increases exponentially as the ontologies size increases.

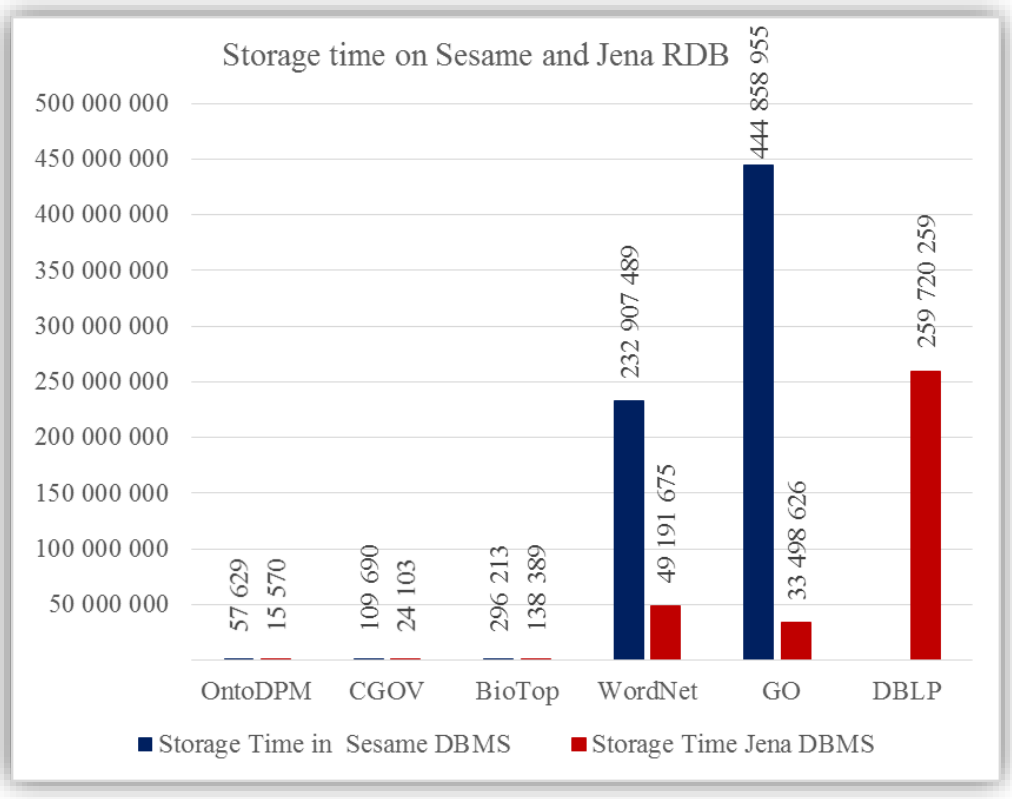


Figure 5.14: Chart of the Database Loading Time in Sesame and Jena

b) Mean and Variance of QRT in Database Storage

Table 5.10 shows the queries response for all ontologies except for DBLP. The last two columns show the mean and variance of QRT, respectively.

Table 5.10: Mean and Variance of QRT in Database Storage in Sesame

Ontologies	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	μ	s^2
OntoDPM	3 747	162	128	123	234	206	188	179	245	258	547	1 266 315.78
CGOV	2 549	167	120	90	104	84	94	82	620	85	400	597 427.17
BioTop	532	118	84	110	87	81	77	92	93	98	137	19 404.62
WordNet	588	104	80	531	78	99	242	236	88	231	228	35 299.79
GO	700	200	102	260	300	80	76	224	600	245	279	44 849.34

The mean and variance of QRT in Jena RDB are shown in the last two columns in Table 5.11.

Table 5.11: Mean and Variance of QRT in Database Storage in Jena

Ontologies	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	μ	s^2
OntoDPM	843	812	797	781	843	796	781	796	797	796	804	494.84
CGOV	750	749	749	750	765	749	765	750	859	781	767	1 167.34
BioTop	859	885	1046	947	867	890	922	876	890	884	907	3 066.71
WordNet	11219	11342	11342	11111	11342	11014	10983	10897	10687	10406	11,034	95 700.9
GO	8379	8160	8082	8238	8144	8144	7848	7848	7489	7598	7,993	82 502.67
DBLP	71,172	71,662	70,126	67,352	67,641	68,414	67,389	67,449	67,125	67,267	68,560	3 062 120.01

Figure 5.15 shows the mean of QRT in Sesame and Jena RDB. The mean QRT in Jena is far higher than that in Sesame thus making the mechanism used by Sesame faster than the mechanism implemented by Jena RDB. The variances of QRT in Sesame are higher than those in Jena. This means that the QRT of Jena tends to be close to the mean as (Table 5.11), while in Sesame, the QRT are spread further around the mean of QRT (Table 5.10).

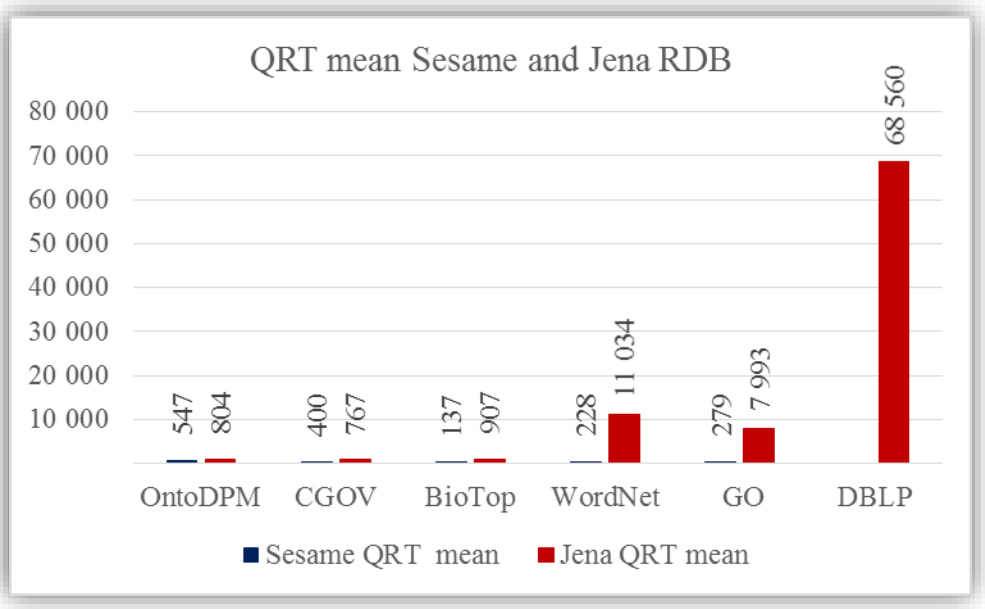


Figure 5.15: Chart of Comparison of the Mean of QRT in Sesame and Jena

c) Disk Space and Efficiency in Database Storage

Sesame and Jena behaved similarly; smaller ontology files took more space than bigger files in terms of efficiency in RDB storage. Jena took almost 4.5 Go of space on the disk to load DBLP which is around 500MB in size. Thus storing big ontologies required more space under Jena.

Table 5.12: Disk Space used to Store Ontologies in Sesame and Jena in RDB

Ontologies	File Size (bytes)	Disk Space		Efficiency	
		Sesame	Jena	Sesame	Jena
OntoDPM	40 960	3 104 768	1 089 536	75.8	26.6
CGOV	69 632	5 181 440	1 286 144	74.41	18.47
BioTop	475 136	7 368 704	12 410 880	15.51	26.12
WordNet	100 429 824	284 446 720	823 058 432	2.83	8.2
GO	106 913 792	322 490 368	701 128 704	3.02	6.56
DBLP			4 510 490 624		7.63

In Table 5.12, it can be seen that Sesame storage in database is proportional to the ontology file; the

bigger the file the bigger the space needed. Figure 5.16 shows that storing in Sesame RDB requires less space than in Jena RDB.

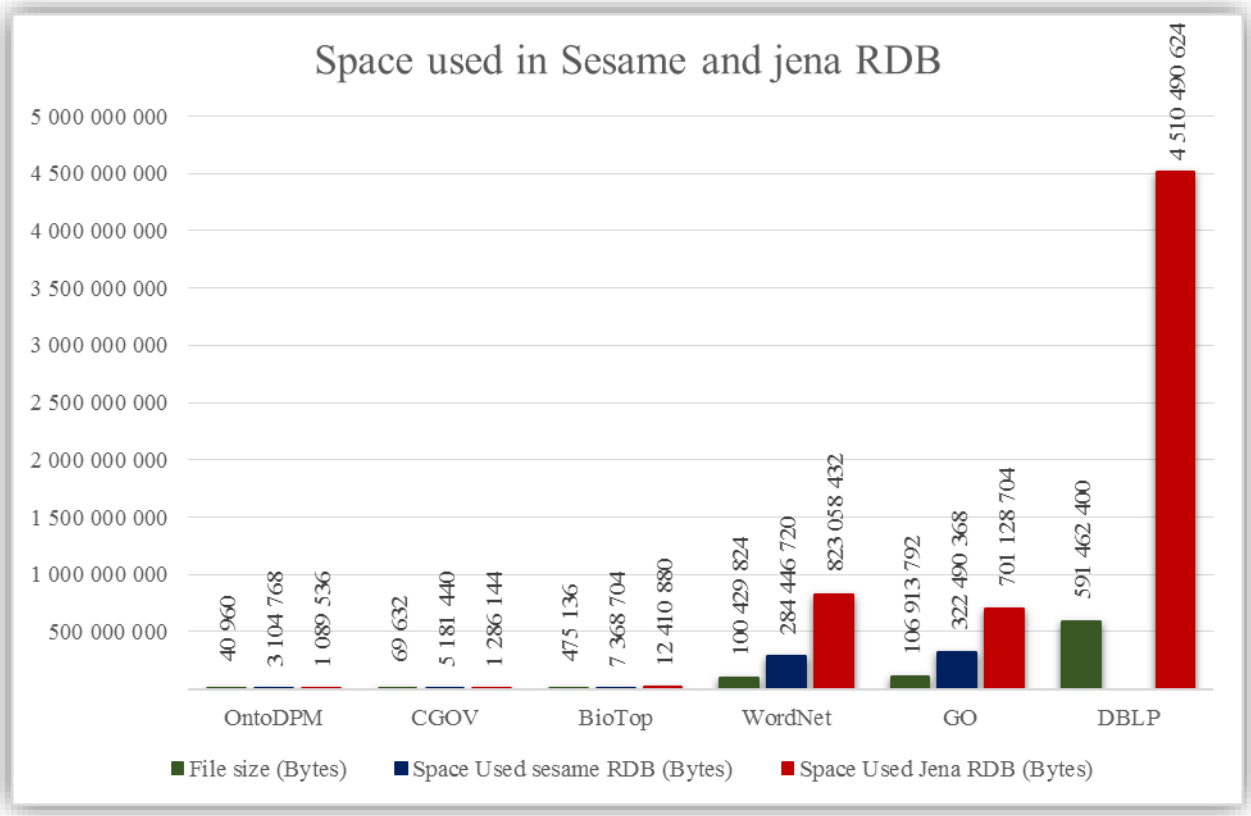


Figure 5.16: Chart of Comparison of Disk Space in RDB Storage in Sesame and Jena

5.4 Analysis of Storage Mechanisms in Sesame and Jena API

The section presents the storage mechanisms in Sesame RDB and Jena RDB. The data stored in the repository are analysed and the mechanisms used are identified.

5.4.1 Storage Mechanisms in Sesame

The following subsections discuss the creation or repository on Sesame, the tables and the mechanisms used by Sesame to store ontology in RDB.

a) Creation of the Sesame repository in RDB

When a repository is created in Sesame, 12 tables are created as shown in Figure 5.17. All tables

except the locked table, have 2 attributes. The locked table is created every time the database is used to store a repository in Sesame.

Figure 5.17: Tables created by Sesame in RDB.

The locked table contains one attribute named process and it contain the process *ID* of the application currently using the databases; when the application ends, the locked table is deleted and the database is released to be used by other applications. Figure 5.18 shows the locked table with the process *ID* that is using it; which in this case is the tomcat server which runs Sesame.

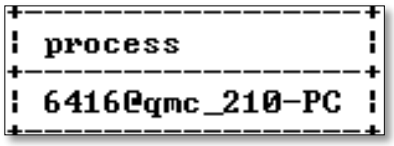


Figure 5.18: Locked table

Figure 5.19 shows the tomcat server and the process ID which is used in the locked table to lock the database from being accessed by any application during its connection time with Sesame.

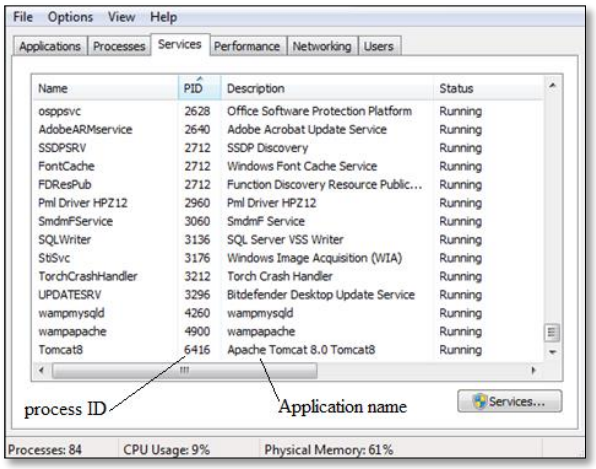


Figure 5.19: Tomcat process ID in the taskbar

b) Loading of the Ontology in the Repository and the RDB

Sesame SDK itself does not store any repository in the computer memory or into file; instead, it hands over the storage of repositories it creates, to the RDBMS. Based on the ontology being loaded, Sesame creates additional tables in addition to the default 12 tables presented earlier in Figure 5.17. Figure 5.20 (a) and (b) shows the OntoDPM repository with 24 tables and the CGOV repository with 36 tables in MySQL, respectively.

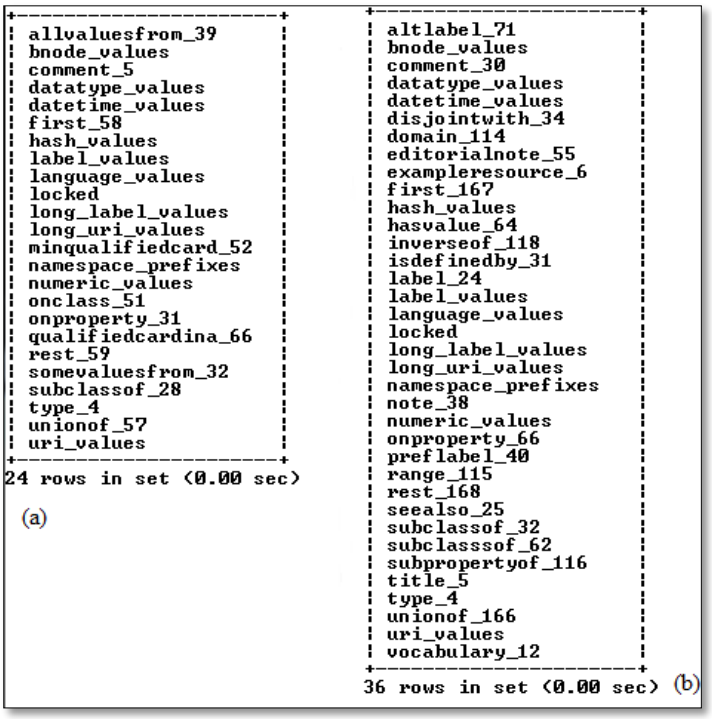


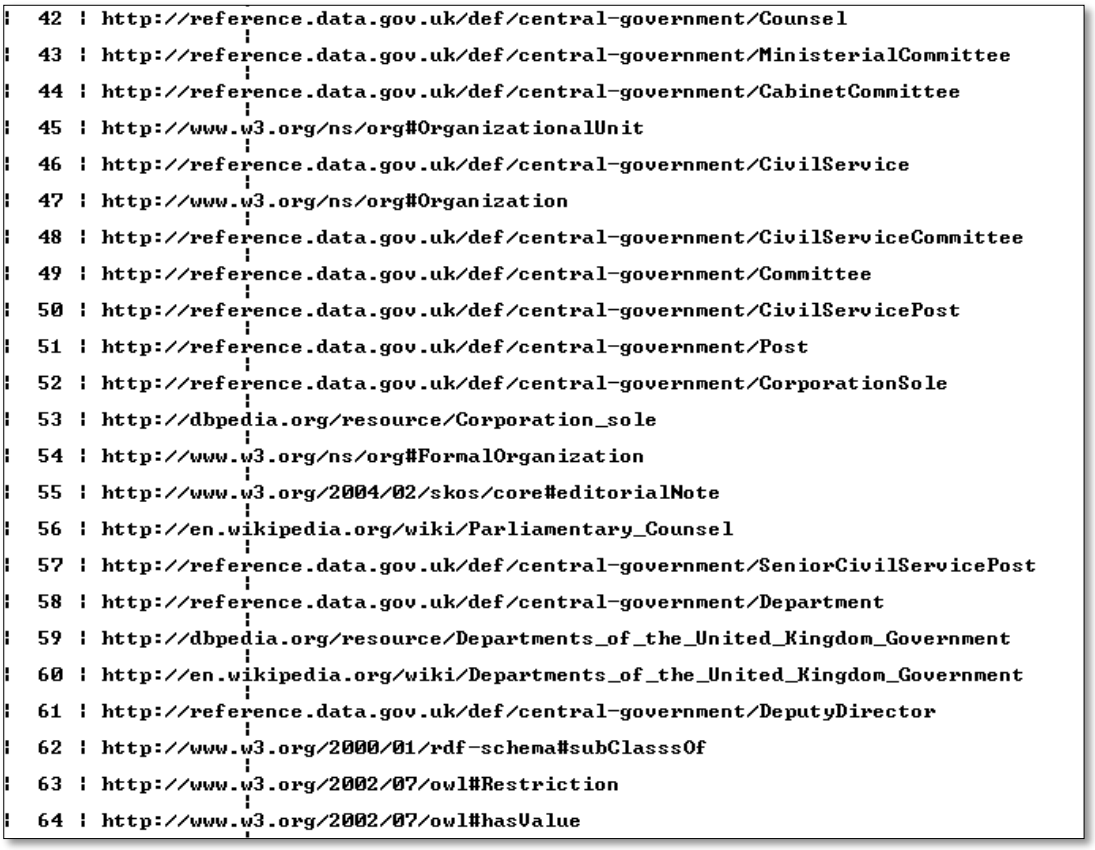
Figure 5.20: Tables Created by Sesame to Store Ontologies Repositories: (a) OntoDPM, (b) CGOV.

The additional tables created by Sesame are used to accommodate all ontologies components. But the 12 default tables (Figure 5.17) store the same types of information independently of the ontology.

c) Description of the Mechanism used by Sesame.

As mentioned earlier, the repositories created in Sesame RDB, has the same number of tables. One could conclude that the mechanism used in Sesame to store ontology in RDB is the generic schema discussed in Chapter 2, at Section 2.2; the generic schema representation is an improved version of a triple store as demonstrated in Chapter 2, Subsection 2.3.1, where all ontologies use the same relational data schema; databases have more than one tables and all databases have the same number of tables. After ontologies have been loaded into the repositories created by Sesame, the structure of databases changes. In fact, the number of table increases based on the content of the ontologies as shown in Figure 5.20. Further, Sesame stores all literals and resources in a table called *uri_values.* The table plays a similar role to the two tables described in Subsection 2.3.1 of Chapter 2 in which Hertel et al. (2009) present a normalized triple tables with two extras tables used to store resources

and literals in order to make query less expensive. But in Sesame, the two tables are combined in the *uri_values* (Figure 5.21).



42	http://reference.data.gov.uk/def/central-government/Counsel
43	http://reference.data.gov.uk/def/central-government/MinisterialCommittee
44	http://reference.data.gov.uk/def/central-government/CabinetCommittee
45	http://www.w3.org/ns/org#OrganizationalUnit
46	http://reference.data.gov.uk/def/central-government/CivilService
47	http://www.w3.org/ns/org#Organization
48	http://reference.data.gov.uk/def/central-government/CivilServiceCommittee
49	http://reference.data.gov.uk/def/central-government/Committee
50	http://reference.data.gov.uk/def/central-government/CivilServicePost
51	http://reference.data.gov.uk/def/central-government/Post
52	http://reference.data.gov.uk/def/central-government/CorporationSole
53	http://dbpedia.org/resource/Corporation_sole
54	http://www.w3.org/ns/org#FormalOrganization
55	http://www.w3.org/2004/02/skos/core#editorialNote
56	http://en.wikipedia.org/wiki/Parliamentary_Counsel
57	http://reference.data.gov.uk/def/central-government/SeniorCivilServicePost
58	http://reference.data.gov.uk/def/central-government/Department
59	http://dbpedia.org/resource/Departments_of_the_United_Kingdom_Government
60	http://en.wikipedia.org/wiki/Departments_of_the_United_Kingdom_Government
61	http://reference.data.gov.uk/def/central-government/DeputyDirector
62	http://www.w3.org/2000/01/rdf-schema#subClasssOf
63	http://www.w3.org/2002/07/owl#Restriction
64	http://www.w3.org/2002/07/owl#hasValue

Figure 5.21: Partial View of *uri_values* Table

As shown on the left column of Figure 5.21, indexes are used for resources and literals to make queries efficient. Index used by Sesame is the counter-based index. Counter-based indexing sorts the resources and literals in alphabetical order (Luo et al., 2012) and gives them a unique number. That unique number is the *id* of the resource of literal as shown in the left column of the *uri_values* table in Figure 5.21.

Beside the literals and resources, Sesame creates other tables that represent relationships in the ontology. The Vertical partitioning mechanism presented in Chapter 2, Subsection 2.3.3-b, created a table for every property in the ontology (Faye et al., 2012). The same mechanism is used by Sesame to create a table for every (1) restriction (quantifier restriction, cardinality restriction, and universal restriction), (2) object property (functional, inverse functional, transitive, symmetric, asymmetric,

reflexive and irreflexive) and data property which describe metadata of the ontology. Therefore, if an ontology includes many restrictions, object and data properties, Sesame will create more tables in the database.

d) Description of Tables Created by Sesame to Store Ontology into RDB

i. Default Tables

The default tables are created along with the repository; they do not contain any ontology data. They are independent of the format of the ontology. Table 5.13 shows the 12 default tables created by Sesame and short descriptions of their functionalities.

Table 5.13: Default Tables Created in Sesame to Store Ontologies in RDB

Table name	**Description**
Uri_values	stores resources and literals
Long_uri_values	stores resources and literals longer than 255 characters
Namespace_prefixes	stores all namespaces found in the ontology
Datetime_values	stores all dates and time used as values
Numeric_values	stores numeric value found in the ontology.
Label_values	store labels found in the ontology
Long_label_values	store labels found in the ontology, longer than 255 characters
Language_values	stores the languages found in the ontology
Datatype_values	stores the different datatypes found in the ontology
Hash_values	stores hash values generated for the uri_values data
Bnode_values	stores blank nodes
locked	used to prevent the concurrent access of the database

The ontology specific tables are discussed next.

ii. Ontology Specific Tables

These tables are created based on the content of the ontology. They are grouped into: restriction, object property, complex class description, and triple tables. The restriction and object property tables

contain four columns namely: *ctx, subj, obj and expl* (Fig. 5.22) to store ontology metadata and ontology property relationships, respectively.

```
mysql> select * from unionof_57;
+-----+-----------+-----------+------+
| ctx | subj      | obj       | expl |
+-----+-----------+-----------+------+
|   0 | 536870931 | 536870932 |    1 |
|   0 | 536870936 | 536870937 |    1 |
|   0 | 536870951 | 536870952 |    1 |
+-----+-----------+-----------+------+
```

Figure 5.22: Columns of the *unionof* _57 Table.

The object properties tables which could exist in the ontology are the functional, the inverse functional, transitive symmetric, asymmetric, reflexive and irreflexive properties. Object properties are used to identify relationships between properties of ontologies. Figure 5.22 shows a sample representation of the restriction "union of" table under the name as *unionof_57*. The complex class description tables store classes which are created through union or intersection operations. Those complex class descriptions are identified by the name of the table such as unionof_57 in Figure 5.22. Triples tables are established through the use of several tables such as *somevaluesfrom*, *onproperty*, and *subclassof* tables. Figure 5.23 shows sample records of the onproperty_31 table.

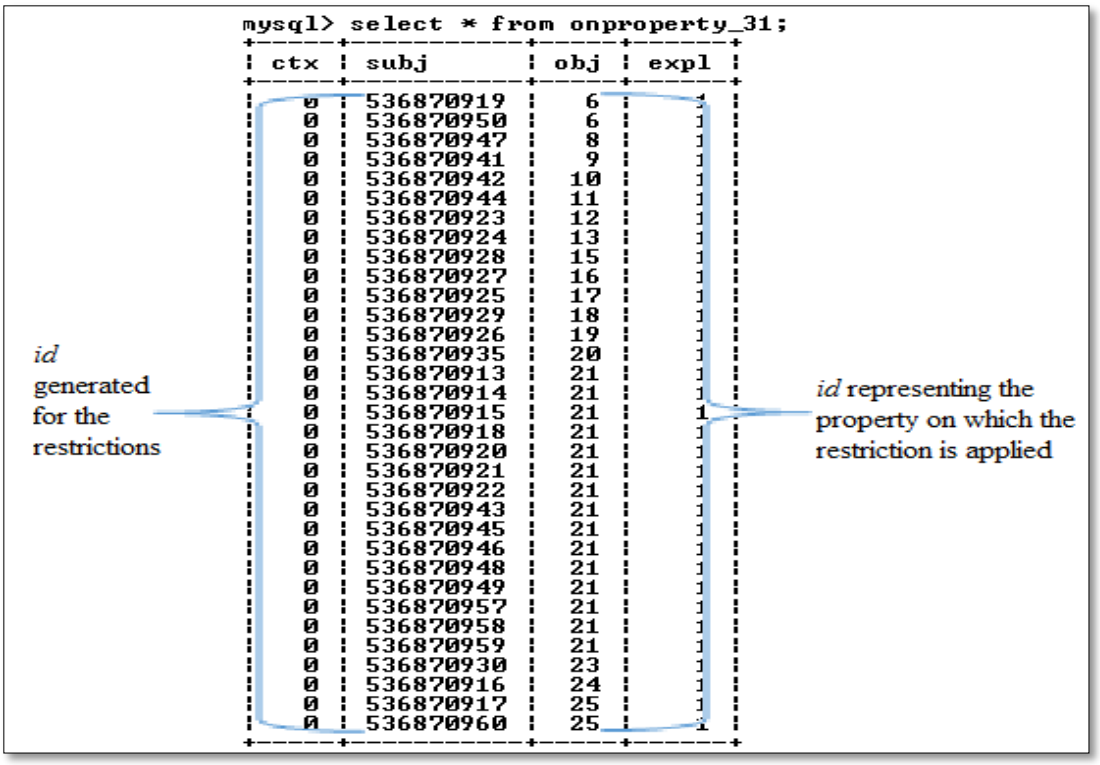


Figure 5.23: Sample onproperty_31 Triples Statement Table

After analysing all the ontologies stored into RDB, it was noticed that they have a different number of tables (Table 5.13). Therefore, it can be concluded that Sesame uses ontology specific schema in which the tables created depends on the content of the ontologies as elaborated in Chapter 2, Section 2.2 (Dieter et al., 2000; Lu et al., 2007; Zhou et al., 2013).

Table 5.14: Number of Tables Created in Sesame to Store Ontologies in RDB

Ontologies	Number of Tables
OntoDPM	22
CGOV	35
BioTop	45
WordNet	20
GO	36

5.4.2 Storage Mechanism in Jena

Jena repositories are created in MySQL with seven tables. Unlike Sesame, the numbers of tables do not increase after ontologies have been loaded in Jena repositories. Out of the seven tables created, only one table named *jena_g1t1_stmt* stores all the RDF statements of the ontology; this is similar to the analysis done by Dieter et al. (2000) and Hertel et al. (2009) where one table is used to store all triples of the ontology in triple store approach. The subject, property and object are stored in the *Subj*, *Prop* and *Obj* columns, respectively. The seven tables created to store ontologies in Jena are labelled with a Jena prefixes as shown in Fig. 5.24.

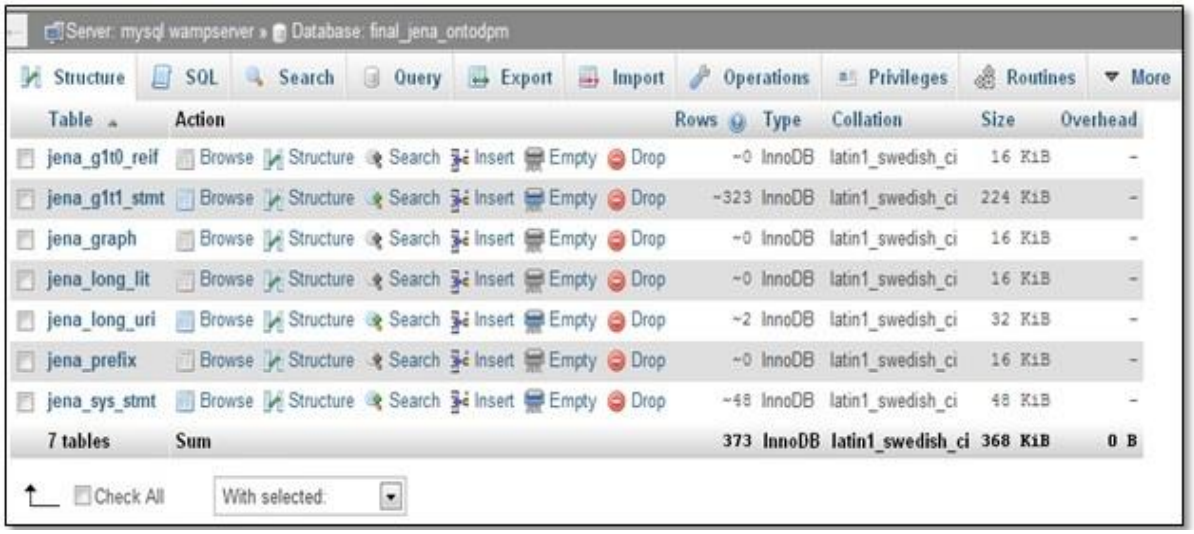


Figure 5.24: Tables Created in Jena to Store Ontology

The description of each of the seven tables created by Jena (Figure 5.24) is provided in the second column of Table 5.15.

Table 5.15: Tables Created in Jena API to Store Ontologies into RDB

Table name	**Description**
Jena_g1t0_reif	stores reified data
Jena_long_uri	stores all long URI in order to simplify their used in other tables
Jena_long_lit	stores literals longer than 255 characters
Jena_prefix	stores all prefixes found in the ontologies
Jena_graph	stores data about all ontologies loaded in the same repository.
Jena_sys_stmt	stores all information related to the repository and the relational database
Jena_g1t1_stmt	Stores all RDF statements of the ontology

After analysing all the created ontology repositories, it was observed that all ontologies stored in Jena have the same number of tables as shown in Table 5.16. Therefore, it can be concluded that unlike Sesame which uses ontology specific schema to store ontologies in RDB, Jena uses generic schema presented in Chapter 2, Section 2.2 in which all ontologies stored in the database have the same number of tables.

Table 5.16: Number of tables in Jena repositories

Ontologies	Number of tables
OntoDPM	7
CGOV	7
BioTop	7
WordNet	7
GO	7
DBLP	7

5.6 Conclusion

In this Chapter, the dataset used in the experiments was presented. Five metrics were used in the experiments to record the values of the different activities performed. The employed metrics include: the loading time, the repository size, the mean and variance of query response time. The results show

that it is difficult to access the underlying structures of repositories in the in-memory and native files in Jena and Sesame. Nevertheless, in relational database, both in Sesame and Jena, it was possible to access and analyse the data in the repositories. Sesame relational uses a combination of mechanisms such as normalized triples store in combination with vertical partitioning. This combination allows Sesame to store ontologies differently, based on the content of the ontologies; in other words, each ontology has a different database schema in Sesame. Jena on the other hand, uses only a normalized triple store mechanism, also known as generic schema mechanism to store ontologies; thus, all ontologies in Jena have the same database schema.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Summary of the study

In this study, existing Semantic Web mechanisms for storage and query of ontology were investigated, analysed and applied. The study began with a review of related work on ontology storage and query mechanisms as well as the platforms that are used to implement those mechanisms. The shortcomings of the related works were identified. These shortcomings include the lack of (1) methodology and requirements to access and use ontologies storage platforms, (2) identification and discussion of ontology storage mechanisms used by ontology storage and query platforms and (3) performance evaluation of ontology storage mechanisms.

Following the design research method, a framework was developed to address the abovementioned challenges. The proposed framework serves as a practical guideline for storing and querying ontologies in Semantic Web repositories. It also provides guidelines to evaluate and compare the performances of ontology storage and query mechanisms.

The framework was practically tested with six ontologies of different formats and sizes on two popular Semantic Web platforms, namely, Sesame and Jena API. Five metrics were used to perform the analysis and comparison of the ontology storage and query mechanisms and platforms.
Although the underlying structures of repositories in the in-memory and native files in Jena and Sesame could not be accessed, in the relational database storage on both Sesame and Jena, it was possible to access and analyse the data in the repositories. The results showed that Sesame relational uses a combination of mechanisms such as normalized triples store in combination with vertical partitioning. That combination allows Sesame to store ontologies differently, based on the content of the ontologies; in other words, each ontology has a different database schema in Sesame. Jena on the other hand, uses only a normalized triple store mechanism, also known as generic schema mechanism to store ontologies; thus, all ontologies in Jena have the same database schema.

6.2 Limitations and Recommendations

Throughout the study, some challenges and limitations were encountered.

- None of the mechanisms used by Sesame was able to load ontologies bigger than 500MB in size.
- All the mechanisms in both Sesame and Jena API could only load ontologies of small sizes (less than 500MB). Therefore, research need to be undertaken to improve loading capacities of Semantic Web platforms to enable the storage and query of large ontologies.
- The study used only two Semantic Web platforms, namely, Sesame and Jena API in the experiments. Further research could focus on repeating the experiments on other existing Semantic Web platforms for storing and querying ontologies and evaluate their performances.

6.3 Conclusion

In this study, the empirical analysis and application of Semantic Web mechanisms for storing and querying ontologies were carried out. Several ontologies of different formats were loaded and stored in various types of ontology storages using two popular Semantic Web platforms, namely, Sesame and Jena API. Furthermore, various metrics were used to analyse and discuss the performances of the two platforms as well as their underlying ontology storage mechanisms. The results showed that both platforms, indeed, implement existing ontology storage mechanisms reported in the literature. The study provides a comprehensive review, application and discussion of Semantic Web mechanisms for storing and querying ontologies. This work would be relevant in Semantic Web (Kwuimi & Fonou-Dombeu, 2015) and Computer Science (Kwuimi, Fonou-Dombeu & Zuva, 2016) as it does not only provide theoretical knowledge but also the empirical findings that may serve as a base for further development of ontology storage media.

BIBLIOGRAPHY

AGRAWAL, R. SOMANI, A. & XU, Y. (2001) "Storage and Querying of Ecommerce Data", In Proceedings of the 27th International Conference on Very Large Data Bases, Rome, Italy, 11 -14 September, pp.149-158

AL-JADIR, L. PARENT, C. & SPACCAPIETRA, S. (2010) "Reasoning with large ontologies stored in relational database: The OntoMid approach", *Journal of Data and Knowledge*, Vol. 69, pp. 1158-1180.

ALAMRI, A. (2012) "The Relational Database Layout to Store Ontology Knowledge Base", In Proceedings of the International Conference on Information Retrieval & Knowledge Management (CAMP), Kuala Lumpur, Malaysia, 13-15 March, pp. 74–81.

ALALWAN, N. ZEDAN, H. & SIEWE, F. (2009) "Generating Owl Ontology for Database Integration", In Proceedings of the 3rd International Conference on Advances in Semantic Processing, Sliema, Malta, 11-16 October, pp. 22-31.

ALATRISH, E. (2013) "Comparison Some of Ontology Editors", *Journal of Management Information Systems,* Vol. 8, No. 2, pp. 018-024

AllegroGraph notes (2015) "AllegroGraph 64-bit RDFStore Processing Billions of RDF Triples Efficiently", available at http://franz.com/agraph/cresources/white_papers/ [accessed 11-November -2015]

ASTROVA, I. KORDA, N. & KALJA, A. (2007) "Storing OWL Ontologies in SQL Relational Databases", *International Journal of Electrical, Computer, and Systems Engineering*, Vol. 1, No 4. pp. 242-247.

ASKAR, K. SIRIL, Y. & DOUGHERTY, M. (2005) "Passerelle -a plug-in that connects Protégé to Sesame", In Proceedings of the 8th International Protégé Conference, Madrid, Spain, July 18-21.

AUER, S. & IVES, Z. (2007) "Integrating Ontologies and Relational Data", University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-07-24.

BERNERS-LEE, T, HENDLER, J. & LASSILA, O. (2001) "The Semantic Web", *Scientific American*, May, pp.29-37.

BROEKSTRA, J., KAMPMAN, A. & VAN HARMELEN, F. (2002), "Sesame: A generic architecture for storing and querying rdf and rdf schema", In Proceedings of the 1st International Semantic Web Conference, Sardinia, Italy, 9-12 June, pp. 54-68.

BÖNSTRÖM, V. HINZE, A. & SCHWEPPE, H. (2003) "Storing RDF as a Graph", In Proceedings of the 1st Latin American Web Congress, Santiago, Chile, 10-12 November, pp. 27–36.

CHENG, L., KOTOULAS, S. & WARD, T. (2012) "Runtime Characterisation of Triple Store: An Initial Investigation", In Proceedings of the 23rd IET Irish Signals and Systems Conference, Maynooth, Ireland, 28-29 June, pp.1–6.

CORCHO, O, A., FERNÁNDEZ-LÓPEZ, M, &. GOMEZ-PEREZ (2006), "Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web", Springer Science & Business Media.

DEHAINSALA, H., PIERRA, G. & BELLATRECHE, L. (2007) "OntoDB: An Ontology-Based Database for Data Intensive Applications", In Proceedings of the 12th International Conference on Database Systems for Advanced Applications, Bangkok, Thailand, April 9-12, pp. 497-508.

DENG, H. KING, I. & LYU, M.R. (2008) "Formal models for expert finding on dblp bibliography data." In Proceedings of the 8th IEEE International Conference on InData Mining, ICDM'08, Pisa, Italy, 15 December, pp. 163-172.

DIETER, F., FRANK, V.H., MICHEL, K. & HANS, A. (2000) "On-To- Knowledge: Ontology-based Tools for Knowledge Management", In Proceedings of the eBusiness and eWork Conference, Madrid, Spain, 18-20 October, pp. 1-7.

DOMINGUE, J., FENSEL, D. and HENDLER, J. A. (2011) "Handbook of Semantic Web Technologies", Springer.

DUDHE A. & SHEREKAR, S.S. (2014), "Performance Analysis of SOAP and RESTful Mobile Web Services in Cloud Environment", In Proceedings of the Second National Conference on Recent Trends in Information Security, GHRCE, Nagpur, India, Jan-2014

CHEBOTKO, A., DENG, Y., LU, S., FOTOUHI, F. & ARISTAR, A. (2005). An Ontology-Based Multimedia Annotator for the Semantic Web of Language Engineering. International Journal on Semantic Web and Information Systems (IJSWIS), 1(1), 50-67.

CHOWDHURY, S., KAYSAR, M. & DEB, K. (2012). "Designing a semantic web ontology of agricultural domain.", In Proceedings of the 7th International Forum on Strategic Technology, Tomsk, Russia ,18 Sep - 21 Sep 2012, pp. 1- 4.

ELENA, B. STEFAN, S. HOLGER, S. & HAHN, U. (2008) "BioTop: An upper domain ontology for the life sciences A description of its current structure, contents and interfaces to OBO ontologies", *Journal of Applied Ontology - Towards a Metaontology for the Biomedical Domain*, Volume 3 Issue 4, December, Pages 205-212

ERLING, O. & MIKHAILOV, I. (2009) "RDF Support in the Virtuoso DBMS. In Networked Knowledge-Networked Media", Springer Berlin Heidelberg, pp. 7-24.

FAN, X., ZHANG, P. & ZHAO, J. (2010) "Transformation of Relational Database Schema to Semantics Web Model", In proceedings of the Second International Conference on Communication Systems, Networks and Applications, Hong Kong, China, 29 June - 1 July, 379-

384.

FAYE, D.C. CURE, O. & BLIN, G. (2012) "A Survey of RDF storage approaches", *ARIMA Journal*, Vol. 15, pp. 11-35.

FEELBAUM, C. (1998) "WordNet: An Electronic Lexical Database." Cambridge, MA: MIT Press

FONOU-DOMBEU, J. V. & HUISMAN, M. (2011) "Semantic-Driven e-Government: Application of Uschold and King Ontology Building Methodology for Semantic Ontology Models Development", *International Journal of Web & Semantic Technology*, Vol. 2, Issue 4, October, pp. 1-20.

FONOU-DOMBEU, J.V., PHIRI, N. M. & HUISMAN, M. (2014) "Persistent Storage and Query of E-government Ontologies in Relational Databases", In Proceedings of the 3rd International Conference, EGOVIS, Munich, Germany, 1-3 September, pp.118-132.

FRIEDMAN, K. (2003), "Theory construction in design research: criteria: approaches, and methods", In: Design Studies 24, pp. 507–522.

GENE ONTOLOGY CONSORTIUM, (2004) "The Gene Ontology (GO) database and informatics resource" *Journal of Nucleic Acids Research*, Vol. 32, Issue 1, pp. 258-261

GHERABI, N. ADDAKIRI, K. & BAHAJ, M. (2012) "Mapping relational database into OWL Structure with data semantic preservation", *International Journal of Computer Science and Information Security*, Vol. 10, No 1, January.

GODDARD, W. & MELVILLE, S. (2004), "Research Methodology: An Introduction", Juta Academic.

GOUDOS, S.K., PERISTARAS, V. & TARABANIS, K. (2007). Public Administration Domain Ontology for a Semantic Web Services E-government Framework. In Proceedings of the IEEE International Conference on Services Computing (SCC 2008), Salt Lake City, Utah, USA, 9-13 July, 270 -277.

GRUBER, T. R. (1995) "Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*,Vol 43, No 5, pp.907-928.

HERTEL, A., BROESTRA, J. & STUCKENSCHMIDT, H. (2009) "RDF storage and Retrieval Systems", In Handbook on ontologies, pp. 489-508. Springer Berlin Heidelberg.

HESSE, W. (2005). Ontologies in the Software Engineering Process. In Proceedings of the 2nd Workshop on Enterprise Application Integration (EAI'05), Marburg, Germany, 30 June -1st July, 11-25.

HEVNER, A. R. & CHATTERJEE, S. (2010). "Evaluation". In: Design Research in Information Systems. Vol. 22. Integrated Series in Information Systems. Springer US, pp. 109–120

HEYMANS, S., MA, L. ANICIC, D., MA, Z. , STEINMETZ, N., PAN, Y., MEI, J., FOKOUE, A., KALYANPUR, A., KERSHENBAUM, A., SCHONBERG, E., SRINIVAS, K., FEIER, C.,

HENCH, G., WETZSTEIN, B. & KELLER, U. (2008) "Ontology Reasoning with Large Repositories Semantic Web, Semantic Web Services, and Business Applications", Springer US, pp. 89-128.

HORRIDGE, M. & BECHHOFER, S. (2011) "The OWL API: A Java API for OWL Ontologies Semantic" Web, Vol 2, No 1, pp.11-21

HUIJUN, D., WENGUO, W. & JIAN, Y. (2011) "A B-Tree Algorithm for Partitioned Index based on CIDR List" In Proceedings of the International Conference of Information Technology, Computer Engineering and Management Sciences, Nanjing, Jiangsu, 24- 25 September, pp. 124-128.

JALALI, V., ZHOU, M. & WU, Y. (2011) "A Study of RDB-based RDF Data Management Techniques", In Proceedings of the Proceedings of the 12th international conference on Web-age information management, Wuhan, China, 14-16 September, pp. 366-378.

KALIBATIENE, D. & VASILECAS, O. (2011) "Survey on Ontology Languages", In Proceedings of the 10th International Conference Perspectives in Business Informatics Research, Riga, Latvia, October 6-8, pp. 124-141.

KALYANPUR, A., PARSIA, B., SIRIN, E., GRAU, B.C. & HENDLER, J. (2006), "Swoop: A web ontology editing browser", *Journal of the Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 4, Issue 2, pp.144-153.

KOLLIA, I., GLIMM, B. & HORROCKS, I. (2011) "Answering Queries over OWL Ontologies with SPARQL", In Proceedings of the 8th Extended Semantic Web Conference, ESWC 2011 Part I, Heraklion, Greece, 29-June 2, pp. 382-396.

KUMAR, G. & BHATIA, P.K. (2014)." Comparative analysis of software engineering Models from traditional to modern methodologies", In Proceedings of the Fourth International Conference in Advanced Computing & Communication Technologies (ACCT), IEEE, pp189-197

KWUIMI, R. & FONOU-DOMBEU, J.V. (2015) "Storing and Querying Ontologies in Relational Databases: An Empirical Evaluation of Performance of Database-Based Ontology Stores", *In Proceedings of the 9th International Conference on Advances in Semantic Processing (SEMAPRO 2015)*, Nice, France, ISBN: 978-1-61208-420-6, 19-24 July, 2015, pp. 6-12.

KWUIMI, R., FONOU-DOMBEU, J.V. & T. ZUVA (2015) "An Empirical Analysis of Semantic Web Mechanisms for Storage and Query of Ontologies in Relational Databases", Accepted for Presentation and Publication in the *Proceedings of the 3rd IEEE International Conference on Advances in Computing & Communication Engineering 2016 (ICACCE 2016).*

LEHMANN, J., ISELE, R., JAKOB, M., JENTZSCH, A., KONTOKOSTAS, D., MENDES, P.N., HELLMAN, S., MORSEY, M., VAN KLEEF, P., AUER, S. & BIZER, C. (2012) "DBpedia A large

scale, Multilingual Knowledge Base Extraction from Wikipedia", *Semantic Web 1*, IOS press.

LEVANDOSKI, J. J., & MOKBEL, M. F. (2009) "RDF Data-Centric Storage", In Proceedings of the IEEE International Conference on Web Services, Los Angeles, CA, USA, 6-10 July, pp. 911-918.

LEY, M. (2009) "DBLP: some lessons learned", *Journal of the VLDB Endowment*, Aug 1, Vol. 2, Issue 2, pp.1493-500.

LILI, X., LEE, S. & KIM, S. 2010. E-R model based RDF data storage in RDB. In Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), Chengdu, China, 9-11 July, 258-262.

LINDBERG, D.A., HUMPHREYS, B.L. & MCGRAY, A.T. (1993). The Unified Medical Language System", Methods of Information in Medicine, 32(4), 281-291.

LU, J. LI, M. LEI, Z. JEAN-SEBASTIEN, B. CHEN, W. YUE, P. & YONG, Y. (2007) "SOR: A practical System for Ontology Storage, Reasoning and Search", In Proceedings of the 33rd International conference on very large databases, Vienna, Austria, September 23-28, pp.1402-1405.

LUO, Y., PICALAUSA, F., FLETCHER, G.H. L., HIDDERS, J. & VANSUMMEREN, S. (2012) "Storing and Indexing Massive RDF Datasets", Semantic Search over the Web, Springer Berlin Heidelberg, pp.31-60.

MA, L., MEI, J., PAN, Y., KULKAMI, K., FOKOUE, A. & RANGANATHAN, A. (2007) "Semantic Web Technologies and Data Management", In proceedings of W3C Workshop on RDF Access to Relational Databases, Cambridge, MA, USA, 25-26 October, 1-5.

MAGKANARAKI, A., KARVOUNARAKIS, G., TUAN ANH, T., CHRISTOPHIDES, V. & PLEXOUSAKIS, D. (2002), "Ontology Storage and Querying", Technical Report No. 308, Hellas Institute of Computer Science, Information Systems Laboratory. Available at http://www.ics.forth.gr/techreports/2002/2002.TR308.Ontology_Storage_and_Querying.pdf.gz (Last Access 04/02/2013)

MILLER, G.A., BECKWITH, R., FELLBAUM, C., GROSS, D. & MILLER, K.J. (1990) "Introduction to WordNet: An on-line lexical database", *International journal of lexicography*, Vol. 3, Issue 4, pp.235-244.

NOY, N. F., & MCGUINNESS, D. L. (2001) "Ontology development 101: A guide to creating your first ontology"

OATES, B.J. (2005). *Researching information systems and computing*. Sage.
Open Anzo notes (2015), http://www.openanzo.org/projects/openanzo/wiki/DBLayout [accessed 11-November -2015]

RAMANUJAM, S., GUPTA, A., KHAN, L., SEIDA, S. & THURAISINGHAM, B. 2009. R2D: A Bridge between the Semantic Web and Relational Visualization Tools. In Proceedings of the 3rd IEEE International Conference on Semantic Computing (ICSC), Berkeley, CA, USA, 14-16 September, 303-311.

SCHULZ, S., BEISSWANGER, E., WERMTER, J. & HAHN, U. (2006), "Towards an upper-level ontology for molecular biology", In Proceedings of the AMIA Annual Symposium, 11-15 November, Washington, USA.

SIDHU, A.S., DILLON, T.S. & CHANG, E. (2008). Current Status of Biomedical Ontologies: Developments in 2007. In Proceedings of the Second IEEE International Conference on Digital Ecosystems and Technologies, Phitsanuloke, Thailand, 26-29 February, 538 – 542.

SIMON, H. A. (1996), The Sciences of the Artificial, 3rd edition, The MIT Press.

SPLENDIANI, A., BURGER, A., PASCHKE, A., ROMANO, P., & MARSHALL, M. S. (2011). "Biomedical semantics in the Semantic Web", *Journal of Biomedical Semantics,* Vol. 2, Issue 1.

SURE, Y., ERDMANN, M., ANGELE, J., STAAB, S., STUDER, R. & WENKE, D. (2002). "OntoEdit: Collaborative Ontology Development for the Semantic Web", The Semantic Web — ISWC 2002 Volume 2342 of the series Lecture Notes in Computer Science pp. 221-235

STEGMAIER, F., GRÖBNER, U., DÖLLER, M., KOSCH, H. & BAESE, G. (2009), "Evaluation of current RDF database solutions", In Proceedings of the 10th International Workshop on Semantic Multimedia Database Technologies (SeMuDaTe), 4th International Conference on Semantics and Digital Media Technologies (SAMT), Graz, Austria, 2-4 December, pp. 39-55.

TAHA, K. (2013) "GOSeek: A Gene Ontology Search Engine using Enhanced Keywords" In Proceedings of the 35th Annual International Conference of Engineering in Medicine and Biology Society (EMBC), Osaka, Japan, 3-7 July, pp.1502 – 1505.

THEOHARIS, Y. CHRISOTPHIDES, V. & KARVOUNARAKIS, G. (2005) "Benchmarking database representations of RDF/S stores", In Proceedings of the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, 6-10 November, pp. 685-701.

VAISHNAVI, V. & KUECHLER, W. (2004), "Design Research in Information Systems", URL: http: / / desrist. org / design - research - in - information - systems/ (visited on 03/28/2012)

VAN ASSEM, M., GANGEMI, A. & SCHREIBER, G. (2006) "Conversion of WordNet to a standard RDF/OWL representation", In Proceedings of the 5th International Conference on

Language Resources and Evaluation, Genoa, Italy, pp. 237-242.

VAN VLIET, H., VAN VLIET, H. & VAN VLIET, J.C (1993) "Software Engineering Principles and Practice" Vol 3, Wiley

VERBORGH, R. & VAN DE WALLE, R. (2011). Application of Semantic Web Technologies for Multimedia Interpretation. In Proceedings of the 20th International World Wide Web Conference, Hyderabad, India, 427-431.

VIJAVALAKSHMI, B., GAUTHAMILATHA, A., SRINIVAS, Y., & RAJESH, K. 2011. Perspectives of Semantic Web in E-Commerce. International Journal of Computer Applications, 5(10), 55-56.

WANG, X., WANG, S., DU, P. & FENG, Z. (2010) "Storing and indexing RDF data in a Colum–Oriented DBMS", In Proceedings of the 2nd International Workshop on Database Technology and Applications (DBTA), Wuhan, China, 27-28 November, pp. 1- 4.

WEISS, C. & BERNSTEIN, A. (2009) "On-Disk Storage techniques for Semantic Web data –Are BTree always the Optimal solution?", In Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems, Chantilly, USA, 26 October, pp. 49-64.

WEISS, C. KARRAS, P. & BERNSTEIN, A. (2008) "Hexastore: Sextuple Indexing for Semantic Web Data Management, Hexastore: sextuple indexing for semantic web data management", In Proceedings of the VLDB Endowment, Vol.1, No. 1, pp. 1008-1019.

WOOD, D., GEARON, P. & ADAMS, T. (2005), "Kowari: A platform for semantic web storage and analysis", In Proceedings of the XTech 2005 Conference, Amsterdam, Netherlands ,16-19 May, pp. 05-12.

XU, L., LEE, S.W. & KIM, S. (2010) "E-R Model based RDF data storage in RDB" In Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT) Chengdu, China 9-11 July, pp. 258 – 262.

YE, Y. OUYANG, D. & DONG, X. (2010) "Persistent Storage and Query of Expressive Ontology", In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, Phuket, Thailand, 9-10 January, pp. 462–465.

YUANBO, G. ZHENGXIANG, P. & JEFF, H. (2005) "LUBM: A Benchmark for OWL Knowledge Base Systems", *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 3, Issue 2-3, October, pp. 158-182.

YUANG, Y., LI, G. & WANG, X. (2013) "Semantic Web Rough Ontology: Definition, Model and Storage Method", In Proceedings of the 6th International Conference on Information Management, Innovation Management and Industrial Engineering (ICIII), Xi'an, China, 23-24 November, pp.104-107.

ZHANG, J., & HOBART, T. A. S. (2008), "A review of ontology and web service composition." *Project Report.*

ZHIHONG, Z. & MINGTIAN, Z. (2003) "Web Ontology Language OWL and its Description Logic Foundation", In Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, Chengdu, China 27-29 August, pp. 157–160.

ZHOU, S. (2010) "Relational Databases Access based on RDF View", In proceedings of the 1st International Conference on E-Business and E-Government, Guangzhou , China, 7-9 May, 5486- 5489.

ZHOU, J., MA, L., LIU, O., ZHANG, L., YU, Y. & PEN, Y. (2006) "Minerva: a scalable owl ontology storage and inference system", In Proceedings of the 1st Asian conference on The Semantic Web, Beijing, China, 3-7 September, pp.426-443.

ZHOU, Z. & YONGKANG, X. (2013) "A study on ontology storage based on relational database", *InConference Anthology*, *IEEE*, pp.1-5.

APPENDIX A: JAVA CODE USED FOR JENA STORAGE AND QUERY

Jena_Memory.java

```java
package com.jenaapi.fresh;

import java.io.InputStream;

public class Jena_Memory {
    public static void main (String args[]) {

        // Create an empty in-memory ontology model
         System.out.println("==============Starting With Repository Creation============================");
         long startTime = System.currentTimeMillis();
        OntDocumentManager mgr = new OntDocumentManager();
        OntModelSpec s = new OntModelSpec( OntModelSpec.RDFS_MEM );
        s.setDocumentManager( mgr );
        OntModel m = ModelFactory.createOntologyModel( s, null );
         System.out.println("--------Ontology memory created --------------------");

        // open file ontology to be loaded
        String inputFileName="c://ontologies/gene.owl";
        InputStream in = FileManager.get().open(inputFileName);
        if (in == null) {
        throw new IllegalArgumentException( "File: " + inputFileName + " not found"); }
        System.out.println("----File successfuly loaded-----------");

        // load the ontology into the memory repository
        m.read(in,"");
        System.out.println("===Ontology successfuly loaded in Memory ============================");
        long estimatedTime = System.currentTimeMillis() - startTime;
        System.out.println("==============End time Time for loading:"+ estimatedTime +"=================");

        /*****************************************************************************************************/

        for (int i=1; i <= 10;i++)
        {
            // Create a new query
            String queryString =
            "PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
            "SELECT ?subject ?object " +
            "WHERE {" +
            " ?subject rdf:type ?object  " +
            " }" +
            " LIMIT 10" ;

            System.out.println("==============Starting With query: "+i +"============================");
             long startTime2 = System.currentTimeMillis();

            Query query = QueryFactory.create(queryString);
            // Execute the query and obtain results
            QueryExecution qe = QueryExecutionFactory.create(query, m);
            ResultSet results = qe.execSelect();
            // Output query results
            ResultSetFormatter.out(System.out, results, query);
            // Important -free up resources used running the query
            qe.close();
            long estimatedTime2 = System.currentTimeMillis() - startTime2;
            System.out.println("==============End time Time for querying: "+i +"  Ellapsed time is:"+ estimatedTime2 +"=================");
        }
        }
}

```

Figure A.1: Code to load and query ontologies in Jena Memory

Jena_TDB.java

```java
package com.jenaapi.fresh;

import java.io.InputStream;

public class Jena_TDB {

    public static void main(String[] args) {

        // Make a TDB-backed dataset
        System.out.println("==============Starting With Repository Creation==========================");
        long startTime = System.currentTimeMillis();
        String directory = "c://ontologies/TDB_Dataset/YAGODataset";
        Dataset dataset = TDBFactory.createDataset(directory);
        System.out.println("--------TDB Repository created --------------------");
        // get the file to load
        String inputFileName = "c://ontologies/yago.ttl";
        InputStream in = FileManager.get().open(inputFileName);
        if (in == null) { throw new IllegalArgumentException("File: " + inputFileName + " not found");}
        // load data in the dataset
        Model model = dataset.getNamedModel("http://yago");
        TDBLoader.loadModel(model, inputFileName);
        dataset.end();
        System.out.println("===Ontology successfuly loaded in TDB ============================");
        long estimatedTime = System.currentTimeMillis() - startTime;
        System.out.println("===============End time Time for loading:"+ estimatedTime + " milliseconds =================");
        System.out.println("Dataset Repository closed");

        /*****************************************************************************************************/
        for (int i=1; i <= 10;i++)
        {
            // Create a new query
            String queryString =
            "PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
            "SELECT ?subject ?object " +
            "WHERE {" +
            " ?subject rdf:type ?object  " +
            " }" +
            " LIMIT 10" ;

            System.out.println("==============Starting With query: "+i +"==========================");
             long startTime2 = System.currentTimeMillis();

            Query query = QueryFactory.create(queryString);
            // Execute the query and obtain results
            QueryExecution qe = QueryExecutionFactory.create(query, model);
            ResultSet results = qe.execSelect();
            // Output query results
            ResultSetFormatter.out(System.out, results, query);
            // Important -free up resources used running the query
            qe.close();
            long estimatedTime2 = System.currentTimeMillis() - startTime2;
            System.out.println("===============End time Time for querying: "+i +"  Ellapsed time is:"+ estimatedTime2 +"=================");
        }


    }

}
```

Figure A.2: Code to load and query Jena in Native files

Owl_mysql2.java

```java
public class Owl_mysql2 {

    public static final String MYSQL_DB_CLASSNAME = "com.mysql.jdbc.Driver";
    public static final String MYSQL_DB_URL ="jdbc:mysql://localhost:3306/final_jena_gene2?useUnicode=true&characterEncoding=UTF8";
    public static final String MYSQL_DB_USER = "root";
    public static final String MYSQL_DB_PASSWD = "";
    public static final String DB = "MySQL";
    public static void main(String[] args) {
         System.out.println("==============Starting With Repository Creation============================");
         long startTime = System.currentTimeMillis();

           try {
               Class.forName(MYSQL_DB_CLASSNAME).newInstance();
           } catch (ClassNotFoundException e) { e.printStackTrace();
           } catch (InstantiationException e) {e.printStackTrace();
         } catch (IllegalAccessException e) {e.printStackTrace();}

           //Create database connection
           IDBConnection conn = new DBConnection(MYSQL_DB_URL, MYSQL_DB_USER, MYSQL_DB_PASSWD, DB);
           System.out.println("--------Connection to database found --------------------");
           //Use the connection to create a model maker
           ModelMaker maker = ModelFactory.createModelRDBMaker(conn);
           Model base = maker.createModel("GENE");
           FileInputStream inputStreamfile = null;
           File file = new File("c://ontologies/gene.owl");
           try {
               inputStreamfile = new FileInputStream(file);
           } catch (FileNotFoundException e1) { e1.printStackTrace();}

           System.out.println("---------------------------------");
           InputStreamReader in = null;
           try {
               in =new InputStreamReader(inputStreamfile, "UTF-8");
           } catch (UnsupportedEncodingException e1) { e1.printStackTrace();}
           System.out.println("----File successfuly found and loaded------------");
           base.read(in, null);
           try {
               in.close();
           } catch (IOException e1) {
               e1.printStackTrace();
           }
           base.commit();
           System.out.println("===Ontology successfuly loaded in Database ==============================");
           //Close the database connection
           try {
               conn.close();
           } catch (SQLException e) {
               e.printStackTrace();

           }
           System.out.println("==============database connection successfully closed==============");
           long estimatedTime = System.currentTimeMillis() - startTime;
            System.out.println("===============End time Time for loading:"+ estimatedTime +"==================");
       }

   }
```

Figure A.3: Code to load ontology in Jena Relational database

Jena_Query.java

```
package OWL;

import java.sql.SQLException;

public class Jena_Query {
    public static final String MYSQL_DB_CLASSNAME = "com.mysql.jdbc.Driver";
    public static final String MYSQL_DB_URL ="jdbc:mysql://localhost:3306/final_jena_wordnet2?useUnicode=true&characterEncoding=UTF8";
    public static final String MYSQL_DB_USER = "root";
    public static final String MYSQL_DB_PASSWD = "";
    public static final String DB = "MySQL";


    public static void main(String[] args) {

         System.out.println("============== Opening  Repository ========================");
            long startTime = System.currentTimeMillis();

              try {
                  Class.forName(MYSQL_DB_CLASSNAME).newInstance();
              } catch (ClassNotFoundException e) {e.printStackTrace();
              } catch (InstantiationException e) {e.printStackTrace();
           } catch (IllegalAccessException e) {e.printStackTrace();}
              //Create database connection
              IDBConnection conn = new DBConnection(MYSQL_DB_URL, MYSQL_DB_USER, MYSQL_DB_PASSWD, DB);
              System.out.println("--------Connection to database found --------------------");
              //Use the connection to create a model maker
              ModelMaker maker = ModelFactory.createModelRDBMaker(conn);
              Model base = maker.openModel("WORDNET");
        /*************************************************************************************************************/
        // Create a new query for Gene Only
                String queryString =
                "PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#> " +
                "SELECT ?subject ?object " +
                "WHERE {" +
                " ?subject rdfs:label ?object " +
                " }" +
                " LIMIT 10" ;
                    System.out.println("==============Starting With query: ============================");
             long startTime2 = System.currentTimeMillis();

            Query query = QueryFactory.create(queryString);
            QueryExecution qe = QueryExecutionFactory.create(query, base);
            com.hp.hpl.jena.query.ResultSet results = qe.execSelect();
            ResultSetFormatter.out(System.out, results, query);
            qe.close();
            try {
                conn.close();
            } catch (SQLException e) {e.printStackTrace();}
            long estimatedTime2 = System.currentTimeMillis() - startTime2;
            System.out.println("==============End time Time for querying:  Ellapsed time is:"+ estimatedTime2 +"=================");
        }
    }
```

Figure A.4: Code to query ontology in Jena Relational database

APPENDIX B: A COMPREHENSIVE LIST OF EXISTING ONTOLOGY STORES

Name	State	Programming language	Supported query language	Supported storage	Part of eval.	License
3Store	o.	C	SPARQL, RDQL	MySQL, Berkley DB	no	GPL
AllegroGraph	o.	Lisp	SPARQL	– (native disk storage)	yes	commercial
ARC	o.	PHP	SPARQL	MySQL	no	open source
BigOWLIM	o.	Java	SPARQL	– (plug-in of Sesame)	no	commercial
Bigdata	o.	Java	SPARQL	distributed databases	no	GPL
Boca	disc.	Java	SPARQL	relational databases	no	Eclipse Public License
Inkling	disc.	Java	SquishQL	relational databases	no	GPL
Jena	o.	Java	SPARQL, RDQL	in–memory, native disk storage, relational backends	yes	open source
Heart	e.d.s.	u.	u.	u.	no	u.
Kowari metastore	disc.	Java	SPARQL, RDQL, iTQL	native disk storage	no	Mozilla Public License
Mulgara	o.	Java	SPARQL, TQL & Jena bindings	integrated database	no	Open Software License v3.0
Open Anzo	o.	Java	SPARQL	relational database	yes	Eclipse Public License
Oracle's Semantic Technologies	o.	Java	SPARQL	relational database	yes	BSD-style license
RAP	o.	PHP	SPARQL, RDQL	in–memory, relational database	no	LGPL
rdfDB	o.	Perl	SQLish query language	Sleepycat Berkeley DB	no	open source
RDFStore	o.	Perl	SPARQL, RDQL	relational database	no	open source
Redland	o.	C	SPARQL, RDQL	relational databases	no	LGPL 2.1, GPL 2 or Apache 2
Semantics.Server 1.0	o.	.NET	SPARQL	MySQL	no	commercial
SemWeb – DotNet	o.	.NET	SPARQL	in–memory, relational database	no	GPL
Sesame	o.	Java	SPARQL, SeRQL	in–memory, native disk storage, relational database	yes	BSD-style license
Virtuoso	o.	Java	SPARQL	relational database	no	open source & commercial & open source
YARS	o.	Java	subset of N3	Berkeley DB	no	BSD-style license

Figure B.1: Overview of available RDF Triple Stores (abbreviations: o. = ongoing, disc.= discontinued, e.d.s. = early developing stage, u. = unknown) Stegmaier et al., 2009.