Comparative Analysis of RDF Query Languages

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By:
Amrita Bhandari
(800932003)

Under the supervision of:
Mrs. Shalini Batra
Assistant Professor (CSED)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2011

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Comparative Analysis of RDF Query Languages*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Mrs. Shalini Batra and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(Amrita Bhandari)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mrs. Shalini Batra)
Assistant Professor,
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

[i]

ACKNOWLEDGEMENT

The completion of this thesis work is an end, of what is actually a new start to a learning experience, the value of which cannot be measured quantitatively. This thesis report is a result of continual work for 5 months and drew intellectual support from varied sources. Submission of this thesis report, gives me an opportunity to convey my gratitude to all those who have helped me to reach at a stage where I feel immense confidence within myself to step in the vast world of research. The efforts put in by me would not have been fruitful if the people around me, who encouraged me at all times, would not have been there. It is therefore almost impossible to express adequately the debts to many persons who have been instrumental in imparting this work a successful status.

First of all, I would like to express my gratitude towards **Thapar University,** for providing me a platform to do my thesis work at such an esteemed institute.

I owe this moment of satisfaction with deep sense of gratitude to my guide, **Mrs. Shalini Batra,** Assistant Professor**,** CSED, Thapar University, Patiala, for her deep concern and kind support. Her continual guidance and diligent efforts led me to work in the right direction from the very beginning. She paved a way for me to explore the vast topic in an organized manner and kept my enthusiasm high with her immense moral support and encouragement at every step.

I also offer my sincere thanks to **Dr. Maninder Singh,** Head, CSED, Thapar University, for providing the required infrastructure for my thesis work in the department.

I would also like to thank all the staff members of CSED, who were always there at the need of the hour and provided me with their help in completion of my thesis work.

I would rather fail in my duty if I do not acknowledge my friends for keeping my spirits high in bad times.

Finally, I thank my parents and my brother for their constant encouragement. Their unobtrusive support and suggestions bolstered my confidence as usual. Their inspiring words will always be a guiding force in all my endeavors to attain greater heights.

AMRITA BHANDARI

ABSTRACT

The global rapid development and maturing of Internet prompts the exploding increase of information in each field and provides people with abundant sources for seeking and obtaining useful information. The concept of information retrieval using search methodologies thus becomes the most active part of network informatics. Semantic Search serves as a key to efficient and accurate information retrieval to the users as in today's human-readable Web the information cannot be easily processed by machine and even highly sophisticated, efficient keyword based Syntactic Search techniques have not been able to provide accurate retrieval.

The Semantic processing for data retrieval is accomplished by the use of a similarity measures known as Resource Description Framework (RDF), Ontology Web Language (OWL), etc. that have been layered on the top of Extensible Markup Language (XML). The annotations increased the efficiency of Semantic Search by enabling the metadata to be associated with the resources. The RDF data stored along with the annotations requires some methodologies to query the repository and thus the query languages like SPARQL, SeRQL, SquishQL etc. came into existence.

In this thesis, the intent is to generate RDF data model for semantic retrieval of Homonyms by adding annotations and its validation using W3C Validation service. The six prominent query languages, SPARQL, SeRQL, SquishQL, LAQRS, XsRQL and RDQL, have been studied on the basis of some features which are necessary and required for being a robust query language and a comparative analysis has been done using the results obtained for the retrieval of Homonyms.

TABLE OF CONTENTS

Certificate…….. **i**

Acknowledgement…… **ii**

Abstract…….. **iii**

Table of Contents…….. **iv**

List of Figures…… **vi**

List of Tables…… **vii**

Chapter 1 Introduction…….. **1**

1.1 A comparison of Syntactic and Semantic search……..… 1

1.1.1 Problems encountered in Syntactic search……..... 2

1.1.2 Role of Semantics……. 3

1.2 Resource Description Framework (RDF)…….. 5

1.2.1 The Motivation…….. 5

1.2.2 RDF Design Goals…… 6

1.2.3 The RDF Model…… 6

1.2.4 Advantages of RDF Model……... 11

1.2.5 RDF Application areas…….. 11

1.3 Role of Annotations in Semantic search……..... 12

1.4 Query Languages…… 13

1.4.1 SQL like Query Language (SquishQL)……. 14

1.4.2 RDF Query Language (RDQL)……..... 14

1.4.3 SPARQL……... 15

1.4.4 Sesame RDF Query Language (SeRQL)…….. 15

1.4.5 LAQRS……..... 16

1.4.6 XsRQL…….. 16

1.5 Structure of the Thesis…… 17

Chapter 2 Literature Review…… **18**

Chapter 3 Problem Statement……..... **21**

3.1 Problem Definition……. 21

3.2 The Proposed Solution……... 21

3.3 The Methodology used to solve the problem……. 22

Chapter 4 Evaluating RDF Query Languages……..... 23
4.1 Query Language Features……... 23
4.2 Study of Prominent Query Languages……... 25
4.2.1 SQL like Query Language (SquishQL)……. 25
4.2.1.1 The Framework required……... 26
4.2.2 RDF Query Language (RDQL)…… 28
4.2.2.1 The Framework required……... 29
4.2.3 SPARQL……... 30
4.2.3.1 The Framework required……... 31
4.2.4 LAQRS…… 31
4.2.4.1 The Framework required……... 32
4.2.5 Sesame Query Language (SeRQL)……... 33
4.2.5.1 The Framework required……... 33
4.2.6 XsRQL…….. 35
Chapter 5 Implementation and Results…… 36
5.1 RDF Repository Creation…….. 36
5.2 Running the Twinkle-SPARQL tool and querying RDF…….. 37
5.3 Installation of SESAME and running query on it……. 43
5.4 Querying the RDQL and SquishQL using JENA……. 47
5.5 Comparison of the Query Languages…… 49
Chapter 6 Conclusions and Future Scope…… 50
6.1 Conclusions……... 50
6.2 Future Scope……. 51
References…….. 52
List of Publications…… 55
Appendix - A……..... 56

LIST OF FIGURES

Figure 1.1	An Example RDF Graph	8
Figure 1.2	The RDF Model of annotation	13
Figure 4.1	JENA Architecture	28
Figure 4.2	SESAME Architecture	34
Figure 5.1	Creating RDF document using Notepad	36
Figure 5.2	Validation of RDF using W3C RDF Validation Service	37
Figure 5.3	GUI of The Twinkle – SPARQL tool	37
Figure 5.4	Query and output for Literal ‘Movie’	38
Figure 5.5	Query and output for ‘eclipse’ of type ‘Movie’ using FILTER constraint	38
Figure 5.6	Query and output for the type, detail and info; constraining the result for stars which belong to any homonym of ‘eclipse’	39
Figure 5.7	Query and output for type, stars and info, constraining the result for stars, info	39
Figure 5.8	Query and output for type of word ‘eclipse’ by matching the alternatives	40
Figure 5.9	Query and output for type, stars, info and sorting type using ORDER BY	40
Figure 5.10	Query and output for projecting results of all the types of word ‘eclipse’	41
Figure 5.11	Query and output for information, belonging to all the types and are distinct	41
Figure 5.12	Query and output for the info after leaving the top 3 results	42
Figure 5.13	Query and output for the info for only the top 3 results	42
Figure 5.14	Tomcat Server startup window	44
Figure 5.15	Apache tomcat on local host	44
Figure 5.16	Create new repository in OpenRDF workbench	45
Figure 5.17	List of created repositories	45
Figure 5.18	The example for Querying repository using SeRQL	46
Figure 5.19	Query for selecting the path defined for all the ‘eclipse’ types	46
Figure 5.20	Query result for showing the paths of all types of word ‘eclipse’	47

LIST OF TABLES

Table 5.1 Scorecard for the various RDF Query Languages 49

CHAPTER 1
INTRODUCTION

With the exponential rapid development of Internet and it's advent as an indispensable medium for electronic sharing of data, it has emerged as a huge repository of unstructured data. This has resulted in making the data search and knowledge extraction, a very cumbersome task. Information retrieval provides user with the means to analyze, classify and characterize the text documents based on their content. The retrieval of required information in minimum possible time and with maximum efficiency has thereby turned up as a major challenge. In addition to this, the searching of applicable data and verification of its origin that is the source from where the data is generated has become very difficult. Furthermore, most of the information on the Web is presented as natural language text with occasional pictures and graphics, which is convenient for human users to read and view but difficult for computers to understand. This limits the indexing capabilities of state of the art search engines, since they do not have capabilities to infer meaning that whether a page is referring to software called Eclipse or to the Movie with the same name, as the difference is not evident to them. Thus users of internet share a significant burden in terms of constructing the search query intelligently. To overcome these upcoming problems and to accomplish the need for efficient data retrieval, semantic methods for retrieval of data are being studied. The Semantic search techniques offer an approach in which knowledge can be published by computers and shared among them using symbols with a well defined, machine-interpretable semantics [1].

1.1 A comparison of Syntactic and Semantic search

The word '**Semantics**' refers to the grammatical rules for assigning meaning to a sentence whereas '**Syntax**' refers to the grammatical rules for specifying correct word order and inflectional structure in a sentence.

Word order is the basic principal of syntax i.e. while trying to understand what is written in a sentence, the syntactic cues of word order are used to help in giving the sentence a proper structure and meaning. Word order gives a sentence the correct intonation especially with the use of function words such as, a, who, due to which its meaning becomes much more clear. If the words of a sentence were not in the correct

order there would be no syntactic cues which can help in understanding the meaning of that sentence. Semantics are an individual's own interpretation of the meaning of a "sentence" based on their prior knowledge. Therefore a sentence that seemingly makes no syntactic sense can have meaning when using semantic cues. For example, the sentence "Baby milk drinks" does not have a syntactic meaning, but through semantics most people would interpret it as meaning, as our prior knowledge tells us that a baby drinks milk, and therefore we can find a meaning from the key words. The difference between syntax and semantics is that syntax is only concerned with what is linguistically and grammatically correct, semantics requires prior knowledge which goes far beyond anything which is just language specific.

The **Syntactic search** uses words or multi-word phrases as atomic elements in document and query representations. The search procedure is essentially based on the syntactic matching of document and query representations [2].

1.1.1 Problems encountered in Syntactic search

The traditional retrieval techniques which involve searching of data with the help of index, content, keywords, etc. are not efficient in terms of optimized search and the search engines that are based on simple keyword matching process return the unexpected results for searched data. The Syntactic search has given rise to varied problems like:

1. Retrieval of true expression
2. No exact disclosure of essential information content
3. Use of morphology match other than acceptation match to retrieve arithmetic
4. Blindly pursuing for the perfectibility has lead to too much amount of retrieval results that users don't have enough time and energy to deal with it [3].

Another problem with syntactic search has been observed in the case of retrieval of results for homonyms. **Homonyms** - the words with similar meanings, when searched syntactically, show irrelevant results and sometimes millions of outputs appear, which are not desired by the user. For example, if user gives query for keyword 'Eclipse' in syntactic search engine like Google, it will return the results corresponding to:

1. Eclipse – a Java Software
2. Eclipse – Solar / Lunar eclipse
3. Eclipse – a Movie

4. Eclipse – a Mitsubishi company's car
5. Eclipse – a Novel

Similarly if we query for the word 'Ring', it will show numerous results and some of them will be related to:

1. Ring – an Ornament
2. Ring – a Hollywood horror movie
3. Ring – a Data structure used for programming
4. Ring – a Piston ring available in Guns
5. Ring – an arrangement for nodes in networking

But these results that users get while they search for the keyword 'Eclipse' and 'Ring' may not comply with his needs because if the user is interested only in word 'Eclipse' related to computer's technology development field, the search results for the word 'Eclipse' also include extra results corresponding to eclipse car, eclipse movie, etc. which are of no value to him.

Therefore, retrieval techniques based on keywords are not able to meet the users' need at semantic and knowledge level and thus identification of new methods is becoming the focus of present research [4]. To present more precise result to each user with different occupations, hobbies, and some other background profiles; there is a need to modify search methodology and involve some factors of user's background information. The idea is to make computers present formatted documents to the user and enable them to deal with the contents and therefore, documents should be supplemented with additional mark-ups containing Meta information. In order to get accurate retrieval, the data is stored along with the annotations [5], which are also referred to as metadata i.e. the data about data. This is possible with the help of semantic data storage along with annotations and consequently semantic data retrieval using the query languages.

1.1.2 Role of Semantics

The study of meaning of any data is referred to as Semantics, which focuses on the relation between signifiers, such as words, phrases, signs and symbols and what they stand for i.e. their denotation. Denotation semantics refers to the addition as well as representation of hidden meanings in form of encoding and formalization of the meaning of coding and query languages by constructing mathematical symbols. With

the help of Semantics, meaning of the data can be added which has to be discovered by computers. It is a vision of a new architecture for the World Wide Web, characterized by the association of machine-accessible formal semantics with more traditional web content. The core idea is to create metadata describing the data, which will enable computers to process the meaning of things.

Semantic search is based on fetching document and query representations through a semantic analysis of their contents using natural language processing techniques and, later, on retrieving documents by matching these semantic representations [2]. The key idea is that semantic search exploits the meaning of words, thus avoiding many of the well known problems of syntactic search e.g., the problems of homonymy and synonymy. The semantic search techniques therefore account for many advantages against the traditional syntactic search techniques.

The Semantic processing for data retrieval is accomplished by the use of a similarity measures known as Resource Description Framework (RDF), Ontology Web Language (OWL), etc. that have been layered on the top of Extensible Markup Language (XML).

Extensible Markup Language (XML)

XML is an industry-standard, system-independent way of representing data. Like HyperText Markup Language (HTML), XML encloses data in tags, but there are significant differences between the two markup languages. While XML tags relate to the meaning of the enclosed text, HTML tags specify how to display the enclosed text. Unlike HTML, the tag vocabulary of XML is not restricted and it enables users to create arbitrary tags that have arbitrary meanings. Because XML tags indicate the content and structure of the data they enclose, they make it possible to do things like archiving and searching. Another major difference between XML and HTML is that XML tags are extensible and allow writing own XML tags to describe the content. With HTML, only limited tags can be used that have been predefined in the HTML specification. With the extensibility that XML provides, the required tags can be created for a particular type of document. Tags are defined using an XML schema language, which describes the structure of a set of XML documents and can be used to constrain the contents of the XML documents. XML allows any type of data to be encoded, as there are no restrictions on XML tags. The only constraint is that a well-

formed and valid XML document conforms to a schema i.e. it requires the XML file adhere to a pre-defined grammar. The XML schema provides a syntactic description of data and the does not provide semantics to the data. If web users all over the world will use the same terms to describe entities, then parsing a XML file will yield both syntactic and semantic information. XML Schemas use a restricted tag set to describe the structure of an XML document and are syntactically constrained to form a balanced tree (i.e., every starting tag must have a corresponding closing tag). RDF overcomes the semantics limitations to a great extent and can represent semantics more efficiently than XML [6].

1.2 Resource Description Framework (RDF)

The Resource Description Framework (RDF) [7] is a W3C standard for describing Web resources, such as the title, author, modification date, content, and copyright information of a Web page. It is a flexible, extensible architecture to represent metadata information about World Wide Web resources [8]. RDF provides a common framework for expressing web resource metadata so that it can be automatically exchanged between computers without loss of meaning [9]. It was originally created in 1999 as a standard on top of XML for encoding metadata. The most exciting uses of RDF are not in encoding of information about web resources, but information about things and relations between them in the real world like people, places, concepts, etc. This mechanism for describing resources is a major component that has been proposed in the W3C's Semantic Web activity. This activity is an evolutionary stage of the World Wide Web in which automated software can store, exchange, and use machine-readable information distributed throughout the Web, in turn enabling users to deal with the information with greater efficiency and certainty. RDF contains a model and XML syntax for representing information in a way that allows programs to understand the intended meaning. RDF's simple data model and ability to model disparate and abstract concepts has also led to its increasing use in knowledge management applications unrelated to Semantic Web activity.

1.2.1 The Motivation

The World Wide Web was originally built for human consumption, and although everything that is accessible through it is machine-readable, this data is not machine-

understandable. It is very hard to automate anything on the Web and because of the volume of data the Web contains; it is not possible to manage it manually. This gave rise to a need for such a standard that can handle the enormous data efficiently.

The development of RDF has been motivated by following scenarios as shown in [7]:

1. There is an exponential growth in the need for Web metadata; which provides information about Web resources and the systems that use them e.g. content rating, capability descriptions, privacy preferences, etc.
2. Due to the development of applications that require open rather than constrained information models e.g. scheduling activities, describing organizational processes, annotation of Web resources, etc.
3. To allow the data to be processed outside the particular environment in which it was created, in a fashion that can work at Internet scale.
4. To allow interoperability among applications, so that by combining data from several applications, new information can be retrieved.
5. To enable automated processing of Web information by software agents as the Web is moving from having just human-readable information to being a world-wide network of cooperating processes. RDF thus provides a world-wide lingua franca for these processes.

All these needs motivated the development of RDF, which has led a foundation for processing the metadata that has emphasized on facilities to enable automated processing of Web resources and has provided interoperability between applications that exchange machine-understandable information on the Web.

1.2.2 RDF Design Goals

The design of RDF is intended to meet the following goals:

1. To have a simple data model that is easy for applications to process and manipulate.
2. To have formal semantics and provable inference that provides a dependable basis to query about meaning of an RDF expression.
3. To use and extensible URI based vocabulary.
4. To use an XML based syntax which can be used to encode the data model for exchange of information among applications.

5. Support for the use of XML schema data types thus assisting the exchange of information between RDF and other XML applications.
6. To allow all the users to make statements about any resource in order to facilitate the operations at Internet scale.

1.2.3 The RDF Model

RDF consists of a model, which is a graphical representation of data. RDF, which has been built on the concept of a statement, provides a general, flexible method to decompose any knowledge into small pieces, called triples, with some rules about the semantics of those pieces [10].

The abstract model of RDF can be described by four simple rules:

1. A fact is expressed as a Subject-Predicate-Object triple, also known as a statement. It is just like a little English sentence.
2. Subjects, predicates, and objects are given as names for entities, also called as resources or nodes (from graph terminology). Entities represent something, a person, website, or something more abstract like states and relations.
3. Names are URIs, which are global in scope, always referring to the same entity in any RDF document in which they appear.
4. Objects can also be given as text values, called literal values, which may or may not be typed using XML Schema data types.

The basic data model consists of three object types:

1. **Resources**

   All things being described by RDF expressions are called resources. A resource may be an entire Web page; such as the HTML document "http://www.w3.org/Overview.html" or it may be a part of a Web page e.g. a specific HTML or XML element within the document source. It may also be a complete collection of pages; e.g. an entire Web site or an object that is not directly accessible via the Web; e.g. a printed book [11]. Resources are always named by URIs plus optional anchor ids. The extensibility of URIs allows the introduction of identifiers for any entity possible.

2. **Property**

   A property is a specific aspect, characteristic, attribute, or relation used to describe a resource. Each property has a specific meaning, defines its

permitted values, the types of resources it can describe and its relationship with other properties. This object type does not address how the characteristics of properties are expressed.

3. **Statement**

   A specific resource together with a named property plus the value of that property for that resource is an RDF statement. These three individual parts of a statement are called, the subject, the predicate, and the object respectively. It is built on the concept of a statement; a triple of the form {predicate, subject, object}. Interpretation of this RDF triple is that <subject> has a property called <predicate> whose value is <object>. In RDF, a <subject> is always a resource named by a URI with an optional anchor id, <predicate> is a property of the resource, and the <object> is the value of the property [11].

Example

RDF is based on identifying web resources (Subjects), utilizing Uniform Resource Identifiers (URI's) and describing Subjects with their respective Properties and Property values. Resources are identified by a resource identifier. A resource identifier is a URI plus an optional anchor id. Consider as a simple example the sentence: "there is a person identified by web resource http://www.rh.edu/~amritab whose full name is Amrita Bhandari" can be represented by the following RDF graph in figure 1.1:

Figure 1.1: An Example RDF Graph

In the graph, resources are represented as circles, properties are represented by directed arcs and property values are represented by a box. These values are called graph end nodes and these values can also become resources if they are described by further properties, i.e., if a value forms a resource in another triple. They are then represented by a circle.

The graph in Figure 1.1 can be also represented in an XML syntax which is the main syntax for exchanging RDF data.

1 <?xmlversion="1.0"?>

2 <rdf:RDF xmlns: rdf="http://www.w3.org/syntax-ns# "

3 kxmlns:exterms=http://www.rh.edu/vocab>

4 <rdf:Description rdf :about = http://www.rh.edu/~amritab>

5 <exterms:fullname>Amrita Bhandari</exterms:fullname>

6 </rdf:Description>

7 </rdf:RDF>

Line 1 is an XML declaration which indicates that the following content is XML. Line 2 indicates that the following XML content represents RDF. The <rdf: RDF> is the root element of an RDF document. It defines the XML document to be an RDF document. It also contains a reference to the RDF namespace.

Line 3 declares another namespace which indicates that URI refs using the exterms prefix that are associated with the vocabulary defined by the organization at http://www.rh.edu [9].

Line 4 starts with the Description element which is a container element for the RDF element being described. The <RDF: Description> element identifies a resource with the 'about' attribute that describes the subject of the statement. It gives the detailed description of what the particular RDF is all about.

Line 5 describes the property and the property value (Object).

Lines 6 and 7 are the closing tags for their respective elements [9].

Representing URI's:

In work with RDF, URI's are abbreviated in several ways, using: namespace, PREFIX and ENTITY definitions, depending on the context:

xmlns: lib = "http://some.host.edu/directory"

or, PREFIX <lib:http://some.host.edu/directory>

or, !ENTITY lib"http://some.host.edu/directory"

If the namespace abbreviation for "eclipse" is substituted for each occurrence of "eclipse:" in the data encoding using XML entities above, then,

<eclipse: type>Car</eclipse: type>, is actually being represented as:

< http://www.flickr.com/eclipse#type>

Car

</ http://www.flickr.com/eclipse #type>

Representing Properties:

Representing properties when they are encoded as XML entities:

<?xml version="1.0" encoding="UTF-8"?>

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

xmlns: eclipse="http://www.flickr.com/eclipse#">

<rdf: Description rdf:

about="http://en.wikipedia.org/wiki/Mitsubishi_Eclipse">

<eclipse: type>Car</eclipse: type>

<eclipse: brand>Mistubishi</eclipse: brand>

<eclipse: meaning>Racehorse</eclipse: meaning>

<eclipse: info>car</eclipse: info>

</rdf: Description>

Representing properties when they are encoded as XML attributes:

<?xml version="1.0" encoding="UTF-8"?>

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

xmlns: eclipse="http://www.flickr.com/eclipse#">

<rdf: Description rdf:

about="http://en.wikipedia.org/wiki/Mitsubishi_Eclipse">

eclipse: type= “Car”

eclipse: brand= “Mistubishi”

eclipse: meaning= “Racehorse”

eclipse: info= “car” </rdf: Description>

RDF Model has following important features [12]:

- **Reification**: An important feature of RDF is its representation of a collection of resources. In addition to making assertions about resources, RDF can be used to make assertions about other RDF statements. This means that an RDF statement can be used in place of an object or property value in an RDF triple, which is known as reification.
- **Collection of Resources:** For representing a list of resources, RDF uses the type “Container” and reified statements are represented using the type “Statement”. A Container object is used to represent a collection of resources in RDF.

The three different types of container objects defined by the RDF specification are Bag, Sequence and Alternative.

- **Bag:** It is an unordered list of resources or literals. Bags are used to declare that a property has multiple values and when there is no significance to the order in which the values are given.

- **Sequence:** It is an ordered list of resources or literals. Sequence is used to declare that a property has multiple values and when is the order of values significant. Sequence might be used, for example, to preserve an alphabetical ordering of values. It permits the occurrence of duplicate values.
- **Alternative:** It is a list of resources or literals that represent alternatives for the (single) value of a property. Alternative might be used to provide alternative language translations for the title of a work, or to provide a list of Internet mirror sites at which a resource might be found. An application using a property, whose value is an alternative collection, is aware of the fact that it can choose any one of the items in the list as appropriate.

1.2.4 Advantages of RDF Model

1. The RDF model is made up of triples: as such, it can be efficiently implemented and stored; other models using variable-length fields would require a more cumbersome implementation.
2. The RDF model is essentially the canonicalization of a (directed) graph, and thus has all the advantages of structuring information using graphs.
3. The basic RDF model can be processed even in absence of more detailed information (an "RDF schema") on the semantics; it already allows basic inferences to take place, since it can be logically seen as a fact basis.
4. The RDF model has the important property of being modular, the union of knowledge (directed graphs) can be mapped into the union of the corresponding RDF structures; this means that the information processing can be fully parallelized and in presence of partial information the output is still a consistent RDF model which can be successfully processed.

1.2.5 RDF Application areas

RDF can be applied in a variety of areas, for example:

- In resource discovery to provide better search engine capabilities.
- In cataloguing for describing the content and content relationships available at a particular Website, page, or digital library.
- By intelligent software agents to facilitate knowledge sharing.

- In content rating and in describing collections of pages that represent a single logical document.
- For describing intellectual property rights of Web pages.
- To express privacy preferences of a user and privacy policies of a Web site.
- RDF with digital signatures is a key in building the "Web of Trust" for electronic commerce, collaboration and other applications.

1.3 Role of Annotations in Semantic search

Annotation is about appending the names, attributes, comments, descriptions, etc. to a document or to a selected part in a text which provides additional information about an existing piece of data. Meaningful use of any data requires knowledge about its organization and content. Contextual information that establishes relationships between the data and its real world aspects is called metadata. In other words, metadata is data that describes information about a piece of data, thereby creating a context in terms of the content and functionality of that data. Broadly speaking, there are two kinds of metadata -structural and syntactic metadata [12].

Structural metadata provides information about the organization and structure of some data, e.g. format of the document. Semantic metadata on the other hand, provides information 'about' the data, for example, what the data is about and the available semantic relationships from a domain model in which the data is defined. The key aspect behind the realization of the Semantic search is the provision of metadata and the association of metadata with web resources. The process of associating metadata with resources like audio, video, structured text, unstructured text, web pages, images, etc, is called annotation. Semantic annotation is therefore defined as a process of annotating resources with semantic metadata. Semantic annotations can be coarsely classified as being formal or informal.

Formal semantic annotations, unlike informal semantic annotations follow representation mechanisms, drawing on conceptual models represented using well-defined knowledge representation languages. Such annotations on web resources can result in vastly improved and automated search capabilities, unambiguous resource discoveries, information analytics etc. The annotation of Web based resources like text files or digital content is very different from the annotation of Web services [12].

The RDF Model for representing annotations is shown in figure 1.2.

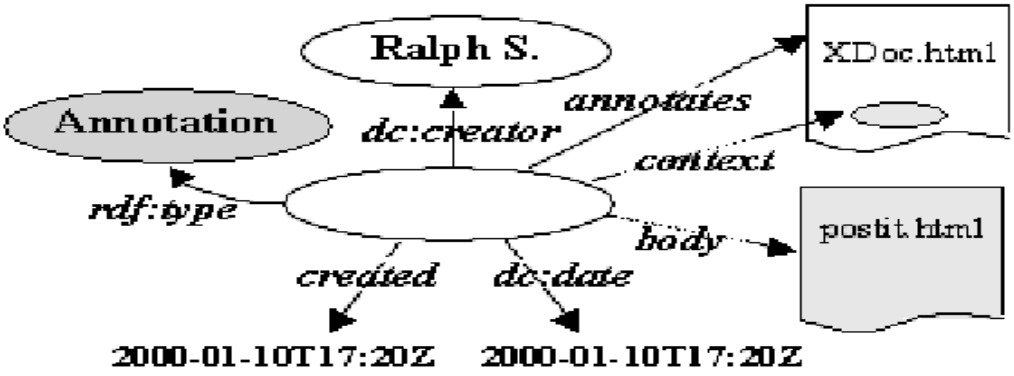

Figure 1.2: The RDF Model of annotation [13]

Annotation speeds up searching and helps in finding relevant and precise information in comparison to Tagging; which is a sequence of characters used to provide some basic information to a document. Semantic Annotation goes one level deeper:

- It enriches the unstructured or semi-structured data with a context that is further linked to the structured knowledge of a domain.
- It allows results that are not explicitly related to the original search.

So, if tagging is about promptly finding the most relevant result, semantic annotation adds diversity and richness to the process [14]. An annotation must have a type which is used to create classes of similar annotations, usually linked together by their semantics. Semantic Annotation helps to bridge the ambiguity of the natural language when expressing notions and their computational representation in a formal language. By telling a computer how data items are related and how these relations can be evaluated automatically, it becomes possible to process complex filter and search operations.

1.4 Query Languages

As the semantic information can be annotated to the documents using the RDF graph data model, some corresponding methodologies to query for the information in the RDF graphs are needed [15]. A working group was formed by W3C with the goal of producing a candidate recommendation for RDF based query language. Prior to the W3C forming a working group, there have been many RDF-based query languages proposed and implemented over the last several years [16]. Since there is a wide

variety of existing proposed RDF-query languages, each with its proponents, a W3C standard for RDF query is viewed to be an important milestone in the development of RDF-query languages. Before presenting the query languages, it is important to firstly mention some of the key motivations behind a robust RDF based query language:

1. A declarative query language will provide easy access to RDF repositories which will enable programmers to deploy applications quickly and efficiently.
2. A robust RDF-based query language will be able to work with many different schema languages since requiring each resource author to use the same schema language may not be possible [17].
3. A good query language will leverage the semantics inherent in the RDF schemas and provide more meaningful results to the user.
4. An RDF-based query language is also needed in order to achieve the desired logical and physical data independence which allows the higher level language to specify the resources required and thereby letting the database engine determine how to store and access the resources [16].

Based on these key motivations, several query languages have been proposed by different working groups out of which the prominent ones have been listed in following subsections.

1.4.1 SQL like Query Language (SquishQL)

An SQL-ish query language for RDF provides consistent, human-understandable, access to repositories of semantic data, whether stored files or large databases, enabling application programmers to create semantic web applications quickly. SquishQL syntax and model is designed to reflect RDF's graph syntax, and uses SQL-like constructs so that application developers can pick it up as quickly as possible. It is implemented with the help of Jena which is an API in the Java programming language, for the creation and manipulation of RDF graphs.

1.4.2 RDF Query Language (RDQL)

RDQL was developed by Hewlett Packard and submitted to the W3C in January 2004. It has been implemented in a several RDF systems including Jena, RDFStore, Sesame, PHP XML Classes, 3 Store and RAP-RDF API for PHP [18]. RDQL was derived mainly from SQUISH, an earlier language. The syntax of RDQL is similar to

a SQL-like select pattern where the select clause allows the projection of the variables [4]. Based on the limited amount of functionality offered in RDQL, it would appear that RDQL was designed to be a simple language. The RDQL approach "suggests a strategy for possible standardization: an RDQL-like language could be developed and deployed without detailed treatment of rule or inference facilities, yet subsequently we used to query 'smarter' RDF services which make use of inferences licensed by OWL or RDF-rule semantics" [16].

1.4.3 SPARQL

SPARQL (pronounced "Sparkle") is a query language for RDF based on graph pattern matching, which is defined in [19]. The SPARQL algebra of Perez and his colleagues [20] defines

- The unary operators sigma (filtering) and ∏ (projection) that correspond to the SPARQL constructs FILTER and SELECT, respectively, and
- The binary operators, ∪, ⋈, ⟕ for the SPARQL constructs UNION, AND, and OPTIONAL respectively.

The basic building block from which more complex SPARQL query patterns are constructed is a basic graph pattern (BGP). A BGP is a set of triple patterns which are RDF triples that may contain query variables at the subject, predicate, and object position. A SPARQL query is evaluated against a dataset consisting of a set of named graphs (declared using FROM NAMED clauses) and a default graph, which is the union of one or more named graphs (declared using FROM clauses).

1.4.4 Sesame RDF Query Language (SeRQL)

SeRQL (pronounced as "circle") is loosely based on several earlier languages such as RQL, RDQL and N3 [21] and as such represents a second generation language. It aims to reconcile ideas from existing proposals into a proposal that satisfies a lot of the key requirements [21]. The overall design goal of SeRQL includes compiling and implementing many of the best features from earlier languages and delivering a light-weight yet powerful language.

SeRQL represents collaboration between industry and the open source community and seemingly offers a viable alternative to RDF-query standardization over the W3C

process. SeRQL supports many of the basic RDF-query features such as path expressions, Boolean constraints and advanced features such as optional matching. SeRQL has become the default query language for Sesame which is an open source RDF database that offers support for many query languages.

1.4.5 LAQRS

LAQRS (pronounced as "LAKERS"), is an experimental set of extensions for SPARQL. It adds to the querying capabilities of SPARQL for querying the RDF.

Major additions that have been done are:

1. Inserting Triples
2. Deleting Triples
3. Explain Query
4. Select Expressions
5. Grouping the Query results and
6. Aggregate functions

1.4.6 XsRQL

XsRQL is an RDF query language that derives most of syntax and style from X-Query while leveraging many of the useful features developed by the W3C XML Query Working Group and omitting parts of the X-Query specification that are specific to XML and hence not required in an RDF environment [22].

The interaction between query evaluation and the data model in XsRQL is similar to the X-Query: as expressions are evaluated within XsRQL, they populate instances into the result set from the target graph. As the query processor evaluates higher expressions on the query tree, new items populate the result set or existing items disappear and whatever remains at the top of the query tree are the results of the query. Because of the relative maturity of XQuery, XsRQL really shines with its support of built in functions and currently supports string functions such as chr, empty, exists, string-length as well as aggregate functions such as count [22]. XsRQL does not follow the familiar SQL syntax; rather it follows slightly abbreviated syntax of XQuery.

1.5 Structure of the Thesis

The remainder of the thesis is organized as follows:

Chapter 2: In this chapter, a review of all the work done in the area of Semantic Search and the information retrieval using RDF Query Languages is presented.

Chapter 3: After going through literature review, the problem statement has been identified and defined in this chapter.

Chapter 4: A detailed view is presented for the RDF Query languages, the features to become a robust query language, the ways of semantic homonyms retrieval through RDF and the query processing frameworks.

Chapter 5: The experiments performed are explained and the results achieved are evaluated in this chapter.

Chapter 6: Conclusion of the work done and the scope for future work is proposed in this chapter.

CHAPTER 2
LITERATURE REVIEW

Since its emergence in 1990's, the *World Wide Web* (WWW) has rapidly evolved into a huge mine of global information and it is growing in size every day. Content is expanding more rapidly than indexers i.e. robot, humans or hybrid systems can index. With the evolution of Internet there came the need of searching and accessing information from the WWW. Gerard Salton came with the development of *SMART information retrieval* in 1960's that moved further the concept of hypertext which was coined by Vannevar Bush in 1945 and gave rise to the term *Search Engine* [23].

A *search engine* [24] is a document retrieval system designed to help find information stored in a computer system, such as on the World Wide Web, inside a corporate or proprietary network, or in a personal computer. But the presence of huge amount of resources on the Web poses a serious problem of accurate search. This is mainly because today's Web is a human-readable Web where information cannot be easily processed by machine. Highly sophisticated, efficient keyword based search engines that have evolved today have not been able to bridge this gap [24]. Requirement was to reconstruct the Web by adding some information into the documents stored on Internet so that computers can process this extra information and understand what a given document is really about.

One of the promising solutions to this problem is *Semantic Web*, which is envisioned by Tim Berners-Lee [25]. He defines it as "The Web of data, with meaning in the sense that a computer program can learn enough about what the data means [in order] to process it". Berners-Lee's vision includes an extended Web that incorporates machine interpretable information, enabling machines to process the volumes of available information, acting on behalf of their human counterparts.

Presently, the *Hypertext Transport Protocol* (HTTP) is used to request documents which are received by programs and a Web server uses a URL provided in the request to determine which file to deliver. Despite of its popularity, HTML suffered from two problems. First, whenever it is felt to users that HTML is insufficient they would simply add additional tags to their documents, resulting in a number of non-standard variants. Second, because HTML was mostly designed for presentation to humans, it

is difficult for machines to extract content and perform automated processing on the documents [24].

The question was how to access this vast amount of information and further making this information available in a computer readable form.

To solve these problems, the World Wide Web Consortium (W3C) developed the *Extensible Markup Language* (XML) [6]. XML was designed to transport and store data, with focus on what data is whereas, HTML was designed to display data, with focus on how data looks. In other words, it can be said that HTML is about displaying information; while XML is about carrying information. But, XML too could not provide machine understandable construct as the XML schema provides a syntactic description of data and it does not provide semantics to the data. Although, XML provides many different possibilities to encode domain of discourse, but this leads to difficulties when understanding of the foreign documents is required.

Thus *Resource Description Framework* (RDF) [7] has been developed by the W3C to encode metadata. Brickley & Guha in [10] explained that RDF can be used to describe any kind of resource. These resources can be accessed and uniquely identified through a URI. This allows for the distribution of a resource's description over different nodes. RDF can be represented in many ways, e.g., with a graph, with triples, or in an XML mark-up called RDF/XML. The *RDF Vocabulary Description Language, RDF Schema* (RDFS), adds semantics to an RDF data model by defining its vocabulary. To enhance the semantic interpretability of data, semantic *annotation* can be added for describing the basic characteristics of Homonyms in RDF as explained in [5].

Data reuse and integration is the basic idea behind the Semantic search effort but extremely large and highly distributed setting of the Web makes reuse and integration a difficult task. To solve this, Schenk and Staab in [26] proposed *Networked Graphs*, which allow users to define RDF graphs both, by extensionally listing content and also by using views on other graphs.

As the semantic information can be annotated to the documents using the RDF graph data model, some corresponding methodologies to query for the information in the RDF graphs are needed [15]. Thus many RDF based Query Languages were proposed and implemented after a working group was formed by W3C with the goal of

producing a candidate recommendation for RDF based query language as mentioned in [16].

A SQL-ish query language, *SquishQL*, which uses SQL like constructs for RDF and provides consistent, human-understandable, access to repositories of semantic data, and enables application programmers to create semantic web applications quickly, was developed.

A language with major mainstream industry support (*RDQL*), derived mainly from SquishQL, which gives insight into the level of current commitment within mainstream technology companies such as Hewlett Packard was submitted to W3C in 2004 [16].

Another language *SPARQL* that has strong W3C support and gives insight into the future direction of standardization efforts within the W3C, was proposed and it offers almost all the basic query features along with advanced Construct query features. *LAQRS* came into light by covering many shortcomings of SPARQL.

To overcome the low semantic functionality offered by all the existing languages, SeRQL was developed which is loosely based on several earlier languages such as RDQL and N3. The overall design goal of SeRQL includes compiling and implementing many of the best features from earlier languages and delivering a light-weight yet powerful language as explained by Broekstra and Kampmen in [21].

After going through various proposals presented for semantic search of data, it has been realized that precision of results for retrieval of Homonyms need to be worked on, by adding annotation to the words available on WWW. RDF which serves as a basic language for realizing the dream of Semantic Web can be used as a means to describe all the resources based on some predefined rules. This makes the searching process more accurate because search engine directly approaches the RDF documents for getting the reliable information related to search and returns more relevant results.

Here a semantic annotation method for retrieval of Homonyms using RDF has been proposed and a comparison of search results has been done with the use of query engines for SPARQL, SeRQL, RDQL, SquishQL, LAQRS, XsRQL which take RDF as input.

CHAPTER 3
PROBLEM STATEMENT

3.1 Problem definition

During the literature review, it was analyzed that Semantic Search aims in making the process of data retrieval through WWW more efficient than Syntactic Search and it focuses on satisfying the needs of user by making the search operations based on annotations rather than by using the traditional keyword matching techniques only. Many algorithms have been developed so far which correspond to searching of text by keyword matching, but when applied to Homonyms, they fail as they give the irrelevant results. So, it has been realized that precision of results for retrieval of Homonyms need to be worked on, by adding annotation to the words available on WWW so that more accurate results can be obtained.

New concepts like XML, RDF, OWL, etc. have been introduced to facilitate the semantic search on the Web out of which, RDF makes the searching process accurate as the search engine directly approaches the RDF documents for getting reliable information related to search. As a result, Syntactic Search needs to be replaced by the Semantic Search which is based on the RDF documents of resources (like Homonyms).

3.2 The Proposed Solution

The solution to the problem is resolved by using RDF to create resource repository of Homonyms related to some real world examples like Eclipse, Swift, Ring, etc, which is done to annotate the words which have different meaning in different scenarios and this RDF document serves as a data model. The data model developed will be validated for its integrity and correctness by using online W3C RDF validation service and finally by querying the RDF model using query engines like Twinkle Sparql for SPARQL, Sesame for SeRQL, Jena for RDQL and SquishQL, Rasqal for LAQRS, XsRQL and further the results obtained will be compared so as to find out the best suitable language for retrieval of Homonyms.

3.3 The Methodology used to solve the problem

The step-by-step methodology to be followed in enabling Semantic Search on the resource repository of Homonyms is given below:

1. Study of the semantic Web technologies like RDF, OWL etc.
2. Review of all syntactic search engine methods used for text and for homonyms.
3. Generation of a RDF document for homonyms “Eclipse” and all its different meanings and then annotation of the document.
4. Validation of RDF repository against the W3C RDF validation service.
5. Study of the basic features required for a robust query language.
6. Installation of Twinkle Sparql tool and querying the RDF repository using the SPARQL and its constructs.
7. Installation of Java and Apache Tomcat server, then configuring the server and deploying Sesame on it, then querying the RDF repository using SeRQL queries.
8. Installation of Jena and creating the same RDF repository through java programming. After that to query it using RDQL and SquishQL constructs.
9. Querying the RDF repository running the XsRQL queries through Sesame.
10. Comparison of the results obtained and analysis of them against the features listed as the basic requirements for a robust query language.

CHAPTER 4

EVALUATING RDF QUERY LANGUAGES

The Resource Description Framework (RDF) provides a flexible, extensible architecture to represent metadata information about World Wide Web resources [17] enabling the web users to utilize the metadata present in RDF-based web resources to enable more efficient information retrieval. For retrieval of information from the RDF repository, some methodologies to query the data from the created repository are required and thus RDF querying becomes an integral part of current Knowledge engineering. Handling of the Repository of Homonyms, which contains the metadata in the form of annotations, requires specific properties of both query languages and storage systems. As the semantic information can be annotated into the documents using the RDF graph data model, corresponding languages to query information from RDF graphs are required [15]. Such languages known as **RDF query languages** are computer language that retrieve and manipulate data stored in RDF format. A W3C standard for RDF query is an important milestone in the development of RDF query languages, as it will likely foster the necessary industry investment and facilitate widespread adoption [16]. There exists a wide variety of proposed RDF query languages each with its proponents. Even though many languages have been implemented with widely varying syntax, there exist many commonalities in the languages that have already been implemented with respect to the overall functionality. For the purposes of comparison and to get an insight into which language(s) are the most mature to date, a baseline set of query language features needs to be established to enable comparison between languages on the basis of most desired attributes of an RDF query language.

4.1 Query Language Features

Generic Path Expressions

Path expression syntax is an important feature that matches a user-input graph pattern, which is in the form of triple {subject-property-object}, against the underlying RDF graph model. These triple patterns match the user-supplied triple patterns against the target graph and return a sub graph wherever the triple pattern finds a match [8].

The Pattern matching features include:

- Support for substituting variables in place of a node or property.
- Support for recursive querying component that can traverse the graph and find all matches reachable from a certain node.
- Support for constraining values using Boolean expressions, True/ False or 0/1.
- Support for searching for a specific input pattern within the graph.

Most of RDF query languages are syntactical heirs of SQL, but are adapted to be better suitable for matching triples. Therefore, by combining several triple matches, it should be possible to query the entire sub graph structure [27].

The Closure Property

Closure property in terms of RDF states that the results of any query operation should also be graphs and not any other data structure i.e. the results of one query can be used as input to another query and this is useful when one wants to break a larger query into smaller queries [21]. With the nodeID attribute provided by RDF, Query language should be able to reference nodes identified in previous queries, so that the sub tree identification does not have to be reiterated [27].

Data Type Support

A data type is defined as a set of character strings known as the lexical space of d, a non-empty set called the value space of d and a mapping from the lexical space of d to the value space of d. This mapping is known as the lexical-to-value mapping of d [21]. For instance, let d be a Boolean data type where the lexical space = {1, 0} (i.e. the list of legal values to represent the data type) and space = {True, False}. In this case, the lexical-to-value mapping maps the string 1 to True and the string 0 to False. In order to provide adequate support for value comparison, it is necessary for the query language to exploit any data types inherent in the RDF model.

Value Support

Property values often contain literals which can be simple strings such as “Thapar University” or typed literals such as #123-131. An RDF-typed literal consists of a string literal and its URI reference (data type URI’s) which are external to RDF [16]. The Query Language should have support for these property values.

Semantic Capabilities

RDF data model provides built-in support for semantic inference via RDF Schema, which stands as an important property of it. Semantic support of a query language might include:

- Support to query the subClass / subProperties of the result set
- Support to query the subClass or SubProperties of the schema
- Support for multiple schemas as it is not possible force content providers to use just one schema [9]

Optional Values

Support for specifying optional sections of the graph similar to an outer join in the world of SQL is an important requirement of RDF data retrieval. This functionality assists in returning the results based on partial matches of pattern data.

Aggregate Functions

SQL supports Aggregate functions such as MIN, MAX and COUNT and its addition to query languages could provide benefit to the RDF community.

Mathematical Set Operations

The operations selection, projection, Cartesian product, set difference and set union form the basis of relational completeness. The three basic operations selection, projection and product are supported by all of the languages however, some languages also support set union and set intersection [21].

4.2 Study of Prominent Query Languages

Several query languages have been proposed by different working groups till date out of which the prominent ones have been explained in following subsections:

4.2.1 SQL like Query Language (SquishQL)

SquishQL is a simple graph-navigation query language for RDF and is termed as SQL-ish, as the query syntax used is designed to resemble the basic structure of SQL in which some database is asked for possible values corresponding to a selection of

variables given some constraining expression. In SquishQL, this constraining expression can be thought of as a list of RDF statements where some parts of each statement have missing values (this is indicated by '?' variables in place of URIs or string literals). This query language is based on a single query mechanism which is the Subgraph matching and also introduces filter functions over the variables so to restrict their values and not to change the expressive power of the graph pattern.
The pattern language is formed from:

- **Triple patterns**, which describe one edge of the graph, allowing either a variable or an explicit value for each of subject, predicate and object.
- **Graph patterns**, which describe the graph shape, expressed as a collection of triple patterns.

This results in quite a weak pattern language but it does ensure that in a result all variables are bounded i.e. there is no disjunction. Further, this language does not express transitivity or other forms of unknown length paths in the graph [28].

Syntax example:

SELECT ?type
FROM http://example.com/xmleurope/presentations.rdf
WHERE (?doc, <dc:type>, ?type) , (?doc, <rdf:type>, <eclipse:Document>)
USING
dc FOR <http://purl.org/dc/elements/1.1/>,
eclipse FOR <http http://www.flickr.com/eclipse>,
rdf FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

4.2.1.1 The Framework required

JENA:

Jena is a leading Semantic Web programmers' toolkit [29], which is an open-source project, implemented in Java, and offers a simple abstraction of the RDF graph as its central internal interface. This is used uniformly for graph implementations, including in-memory, database-backed, and inferred graphs [30].
It implements the interpretation of the RDF specifications and was developed to satisfy two goals:

- To provide an API that was easier for the programmer to use than alternative implementations

- To be conformant to the RDF specifications

Since Jena allows users to manipulate and query RDF graphs, it can be used to write a program that uses Jena classes to:

- Retrieve and parse an RDF file containing a graph or a collection of graphs,
- Store it in memory,
- Examine each triple in turn, examine one component (say, the subject) of each triple in turn, or examine only triples that meet specified criteria, and,
- Write a serialized version of a graph to a file.

JENA Architecture

The heart of the Jena2 architecture is the RDF graph, a set of triples of nodes. As seen in figure 4.1, graph layer following the RDF abstract syntax, is minimal by design wherever possible functionality is done in other layers. This permits a range of implementations of this layer such as in memory or persistence triple stores. The EnhGraph layer is the extension point on which to build APIs. Within Jena2 the functionality offered by the EnhGraph layer is used to implement the Jena1 Model API and the new Ontology functionality for OWL and RDFS, upgrading the Jena1 DAML API [6]. Input Output is done in the Model layer. The Jena2 architecture supports fast path query that goes all the way through the layers from RDQL at the top right through to an SQL database at the bottom, allowing user queries to be optimized by the SQL query optimizer.

The two key architectural goals of Jena are:

- Multiple, flexible presentations of RDF graphs to the application programmer. This allows easy access to, and manipulation of, data in graphs enabling the application programmer to navigate the triple structure.
- A simple minimalist view of the RDF graph to the system programmer wishing to expose data as triples.

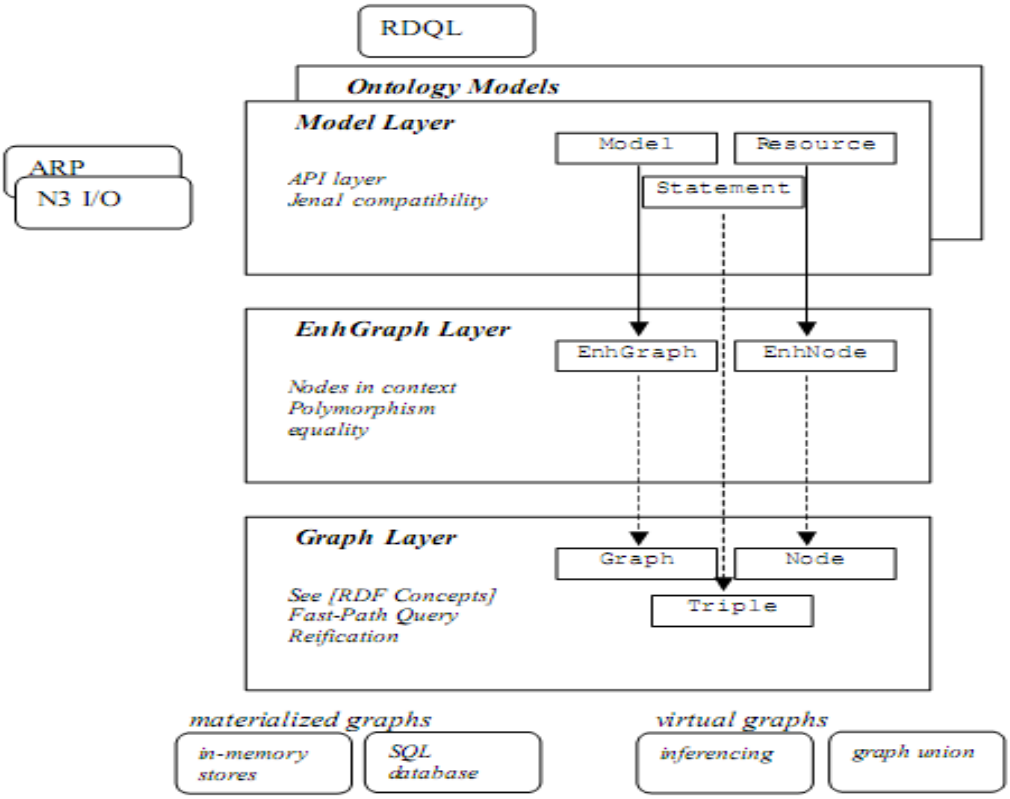

Figure 4.1: JENA Architecture [31]

The general structure of Jena

Jena is a Java class library, and is composed mainly of API (Application Programming Interface) and SPI (System Programming Interface). In general, the users only use API in programming and SPI mainly provides the core data structure for the Jena system and most of the functions are realized in API. Like all the Java codes, the Jena class library is maintained by the Package mechanism. The Jena API has been defined in terms of interfaces so that application code can work with different implementations without change.

4.2.2 RDF Query Language (RDQL)

RDQL is a query language for RDF in Jena models, derived from SquishQL and provides a data-oriented query model which is more declarative approach to complement the fine-grained, procedural Jena API. It is "data-oriented" language as it only queries the information held in the models and there is no actual inference being done. RDQL system does not do anything other than to take the description of what the application wants, in the form of a query, and returns that information in the form

of a set of bindings. Based on the limited amount of functionality offered in RDQL, it would appear that RDQL was designed to be a simple language.

Syntax example:

A simple path expression:

SELECT ?x

WHERE (?x, http://www.w3.org/1999/02/22-rdf-syntax,ns#type,http://any.com/a >)

This select statement matches all statements in the graph that have a property of http://www.w3.org/1999/02/22-rdf-syntax-ns#type and object of http://any.com/a.

The variable ?x is bound to the subject resource [17]. Variables are introduced with a leading '?' and URIs are quoted with <>; unquoted URIs can be used where there is no ambiguity.

SELECT Clause identifies the variables to be returned to the application. If all the variables are not needed by the application, then specifying the required results can reduce the amount of memory needed for the results set as well as providing information to a query optimizer.

FROM Clause specifies the model by URI.

WHERE Clause specifies that the graph pattern used is a list of triple patterns.

AND Clause specifies the Boolean expressions.

USING Clause is a way to shorten the length of URIs. As RDQL is likely to be written by people, this mechanism helps make for an easier to understand syntax

4.2.2.1 The Framework required

JENA

For RDQL in JENA, a query is created by passing a string to the Query class. If the source is not specified within the query, then the source must then be given, usually by passing a model to the query object, or specifying the URL of a model as shown below,

```
String queryString = "SELECT statement";
Query query = new Query(queryString) ;
// Need to set the source if the query does not.
query.setSource(model);
```

```
QueryExecution qe = new QueryEngine(query) ;
QueryResults results = qe.exec() ;
for ( Iterator iter = results ; iter.hasNext() ; )
{  ResultBinding res = (ResultBinding)iter.next() ;
... process result here ...}
results.close() ;
```

Results are returned as an iterator, with each call to QueryResults.next returning one set of variables bindings. The text formatter called by jena.rdfquery prints out each ResultBinding as a single line.

4.2.3 SPARQL

The SPARQL query language for RDF is designed to meet the use cases and requirements identified by the RDF Data Access Working Group in RDF Data Access Use Cases and Requirements. SPARQL offers many basic features desired in an RDF-based query language. It provides a subset of operations on plain literals, XSD integers and XSD floats defined in XQuery and XPath functions and operators such as comparison of numeric values, functions on string values casting, comparison of duration, time and date values [8]. One interesting feature of SPARQL is the RDF keyword CONSTRUCT, a CONSTRUCT query matches a graph pattern against one or more input graphs. The resulting variable bindings are embedded into a graph template in order to generate new RDF data. To further restrict the bindings produced by pattern matching, filter expressions can be used, for example to check for (in-) equality of variables, data types of literals etc. [26].

Syntax example:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX eclipse: <http://www.flickr.com/eclipse#>
SELECT ?type ?detail ?stars ?info
WHERE
{ ?x eclipse:type ?type FILTER regex(?type, "Movie")
?x eclipse:detail ?detail FILTER regex(?detail, "^")
?x eclipse:stars ?stars FILTER regex(?stars, "^")
?x eclipse:info ?info FILTER regex(?info, "^") }
```

4.2.3.1 The Framework required

Twinkle: Sparql Tool

Twinkle is a simple GUI interface that wraps the ARQ SPARQL query engine. The tool is useful both for people wanting to learn the SPARQL query language, as well as for those doing Semantic Web development. Some features of Twinkle: sparql tool are:

- Ability to load, edit and save SPARQL queries.
- Insert PREFIX statements into queries.
- Configure custom namespaces so they can be quickly inserted into queries.
- Cancel long running queries.
- Save the results to file.
- Query local files and remote RDF documents.
- Query RDF data held in relational databases.
- Query online SPARQL endpoints, such as DBpedia, reyvu.com and GovTrack.
- Query using standard SPARQL, or ARQ extended syntax that supports COUNT.
- Use ARQ extension functions and property functions.
- Apply inference (e.g. RDF Schema, OWL ontology) when running queries.
- Configure commonly used data sources for quick access.

4.2.4 LAQRS

LAQRS is an experimental set of syntax extensions for SPARQL whose syntax and features may change at any time. It provides parsing for alternate update syntaxes, the EXPLAIN keyword along with a few experimental extension functions: NOW ( ) / CURRENT_DATETIME ( ), FROM_UNIXTIME ( ) and TO_UNIXTIME ( ).

This language adds to the SPARQL functionality by providing additional features for:

1. Inserting Triples
2. Deleting Triples
3. Explain Query
4. Select Expressions

5. Grouping the Query results and
6. Aggregate functions

Syntax example:

```
PREFIX ex: <http://example.org/ex#>
EXPLAIN SELECT $number $name (3 + $number) AS foo COUNT (*) AS bar
FROM <data.rdf>
WHERE
{$r a ex:Class ;
ex:numPred $number ;
ex:stringPred $name
}
GROUP BY $name ORDER BY $number
```

4.2.4.1 The Framework required

RASQAL

It is an Open Source software containing C library that handles Resource Description Framework (RDF) query language syntaxes, query construction and execution of queries returning results as bindings, Boolean, RDF graphs/triples or syntaxes. The supported query languages are SPARQL 1.0, Draft SPARQL Query 1.1, Update 1.1 Syntax and Experimental SPARQL extensions i.e. LAQRS.

RASQAL provides:

- RDF query construction, query result binding API and access API.
- Query language support for SPARQL 1.0 and parts of draft SPARQL 1.1 Query.
- Query language support for RDQL.
- A query execution engine including grouping, aggregate expression and filtering evaluation.
- Triple store querying APIs to support running over external RDF graphs.
- No memory leaks.

It does not provide an RDF API or triple store, but relies on external libraries implementing the triple store API providing matched RDF data originally from a specified content URI.

4.2.5 Sesame RDF Query Language (SeRQL)

SeRQL, pronounced as “circle”, was developed as a second generation language [32] which aims to reconcile ideas from existing proposals into a proposal that satisfies a lot of the key requirements [21]. It uses a query result mechanism that can return not only variable bindings, but also sets of RDF statements. This construction makes use of an alternative clause, called CONSTRUCT that can be used as an alternative to the SELECT clause. SeRQL supports RDF Schema semantics in the language specification and although there is no need to be concerned about the implementation of Schema awareness feature of the language in Sesame, the specification of the language explicitly defines the constructions as being schema-aware. Therefore, any implementation that supports SeRQL will have to implement Schema inference in order to be fully compatible [32].

Syntax example:

SELECT *
FROM {info} eclipse:type {type}
USING NAMESPACE
rdf = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>,
eclipse = http://www.flickr.com/eclipse#

4.2.5.1 The Framework required

SESAME

Sesame is an open source Java framework for storage and querying of RDF data which is fully extensible and configurable with respect to storage mechanisms, inferences, RDF file formats, query result formats and query languages. Sesame offers a JBDC-like user API, streamlined system APIs and HTTP interface supporting the SPARQL Protocol for RDF, as well as SeRQL. It also supports most popular RDF file formats and query result formats. Originally, Sesame was developed by Aduna as a research prototype for the hugely successful EU research project On-To-Knowledge. Sesame provides the necessary tools to parse, interpret, query and store all this information, embedded in the application if you want, or, if you prefer, in a separate database or even on a remote server [33].

SESAME Library

The Sesame library consists of a set of java archives:

- Sesame.jar. The Sesame core classes.
- rio.jar. Rio (RDF I/O) is a set of parsers and writers for different RDF serialization formats (RDF/XML, Turtle, N-Triples).
- openRDF-model.jar. Shared interfaces and classes for the RDF model.
- openRDF-util.jar. Shared utility classes.

SESAME Features

- Sesame is completely targeted at Java 5. All APIs use Java 5 features such as typed collections and iterates.
- It contains a Revised Repository API that is much more targeted at the use of Sesame as a library.
- Support for context/provenance, allowing you to keep track of individual RDF data units (like files, for instance).
- Proper transaction / rollback support.
- Support for the SPARQL and SeRQL Query Language.
- Similar to Jena as it supports triple storage, reasoning and Web services.

SESAME Architecture

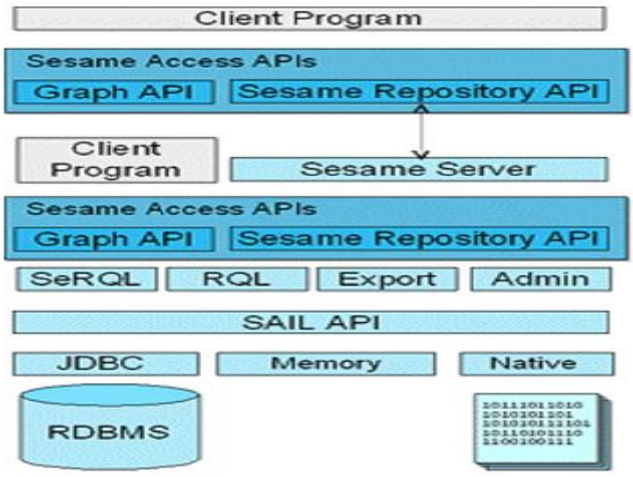

Figure 4.2: Sesame Architecture [33]

In figure 4.2, showing Sesame architecture, the Storage and Inference Layer, or SAIL API, is an internal Sesame API that abstracts from the storage format used and

[34]

provides reasoning support. SAIL implementations can also be stacked on top of each other, to provide functionality such as caching or concurrent access handling. Each Sesame repository has its own SAIL object to represent it and on top of the SAIL, there are Sesame's functional modules, such as the SeRQL, RQL and RDQL query engines, the admin module, and RDF export. Access to these functional modules is available through Sesame's Access APIs, consisting of two separate parts: the Repository API and the Graph API. The Repository API provides high-level access to Sesame repositories, such as querying, storing of RDF files, extracting RDF, etc. The Graph API provides more fine-grained support for RDF manipulation, such as adding and removing individual statements, and creation of small RDF models directly from code. The Access APIs provide direct access to Sesame's functional modules, either to a client program or to the next component of Sesame's architecture, the Sesame server.

4.2.6 XsRQL

XsRQL is a functional language based on a formal data model which understands RDF Items and XML Schema atomic types; has a procedural feel that will be readily familiar to many programmers and its result lists are free-form heterogeneous sequences of data model items. It is a typed language that adapts an XPath-style navigational metaphor to meet the needs of RDF graphs and borrows the XQuery's style of built-in functions [22].

The main objectives of XsRQL are:

- To keep the language as simple and as elegant as possible, provide the end user an opportunity of choosing a query style that sits on a continuum between concise-but-readable and a more verbose style that is more self-documenting.
- To allow the user a similar choice on the emit side of the equation between result sequence concision and a full-blown, ad hoc report-generating capability.

Syntax example:

SELECT ?x, ?fname

WHERE (?x <http://www.w3.org/2001/vcard-rdf/3.0#fn> ?fname

In the abbreviated world of XsRQL, the query would look like: @< http://www.w3.org/2001/vcard-rdf/3.0#fn>/* [21

CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1 RDF Repository Creation

RDF Editor or Notepad can be used to make the RDF document for Homonym 'Eclipse' repository. Self defined tags are used to annotate the words and RDF files are saved with the extension (.rdf). Figure 5.1 shows RDF document created using Notepad.

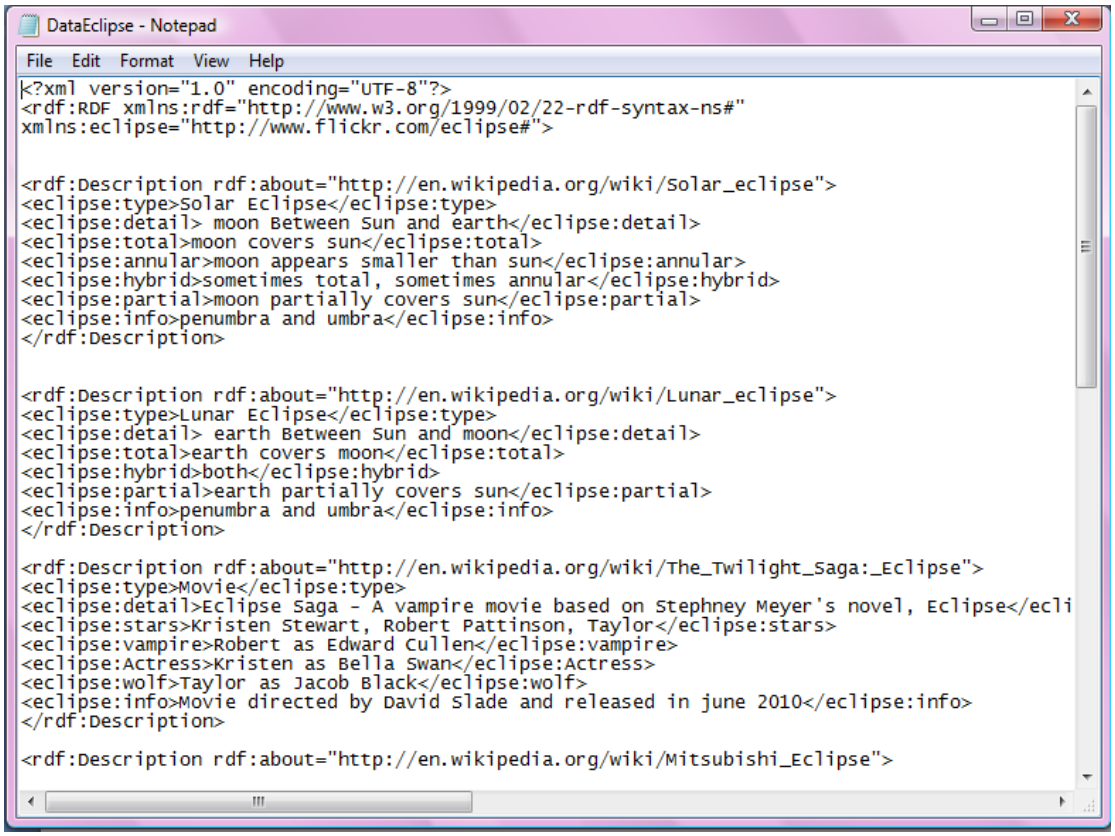

Figure 5.1: Creating RDF document using Notepad

The RDF document has been validated through online W3C RDF validation Service. This validation process generates the triples (subject, predicate, and object) for the RDF document (Data Model). The validation process is shown in figure 5.2:

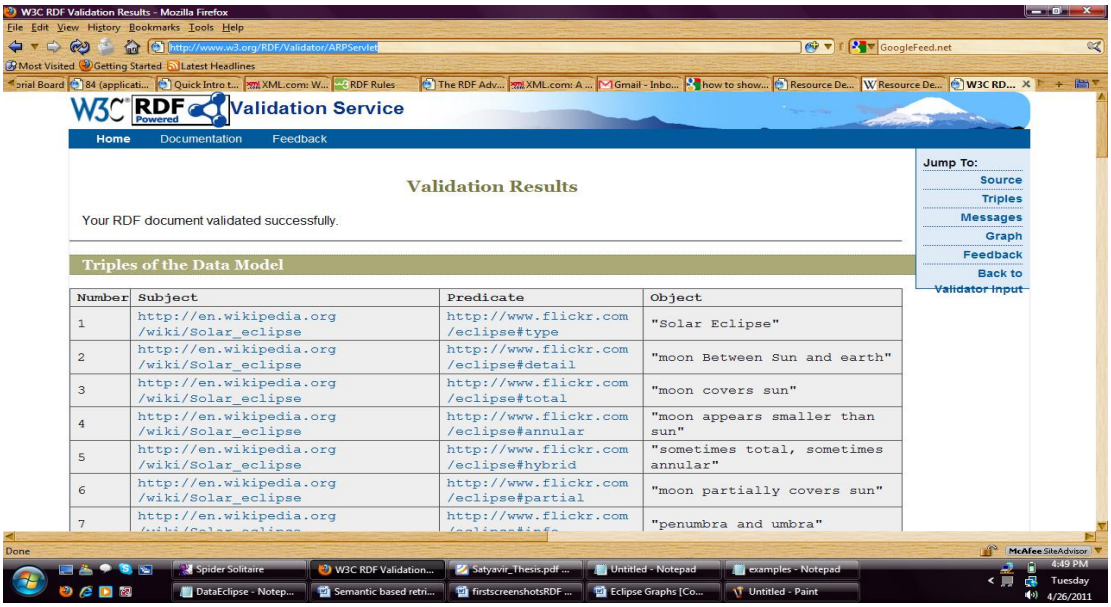

Figure 5.2: Validation of RDF using W3C RDF Validation Service

5.2 Running the Twinkle – SPARQL tool and querying RDF

SPARQL tool is used to query the RDF document from different perspectives. To run Twinkle - SPARQL tool, it has to be imported inside the java platform ECLIPSE and after executing the file, Twinkle.java, a GUI as shown in figure 5.3 is obtained:

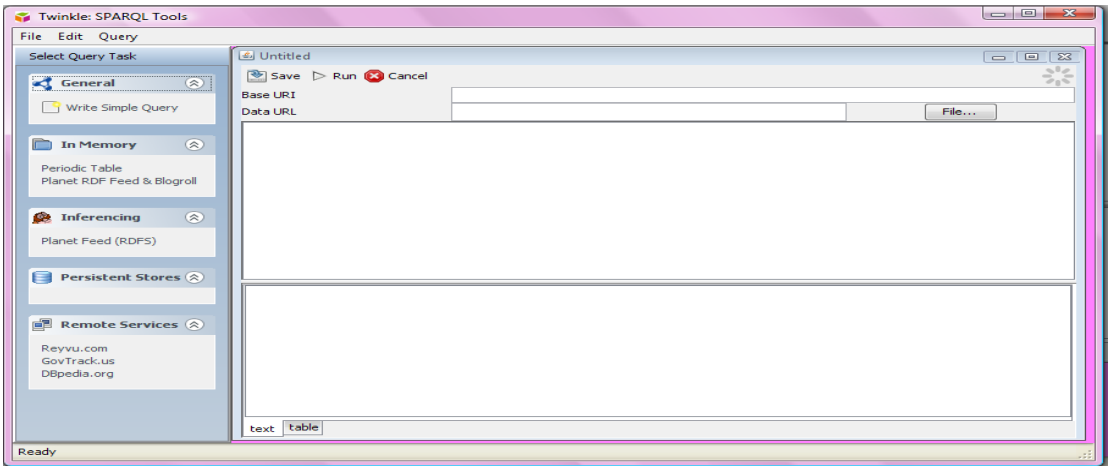

Figure 5.3: GUI of The Twinkle – SPARQL tool

Querying RDF through SPARQL

The SPARQL, a RDF Query Language, has been used to write queries. The queries corresponding to the required features have been run and their results have been shown.

Query 1 - Matching RDF Literals

Query 1(as shown in figure 5.4) retrieves the path for the Eclipse word whose type is a literal 'Movie' from the RDF document generated in figure 5.1.

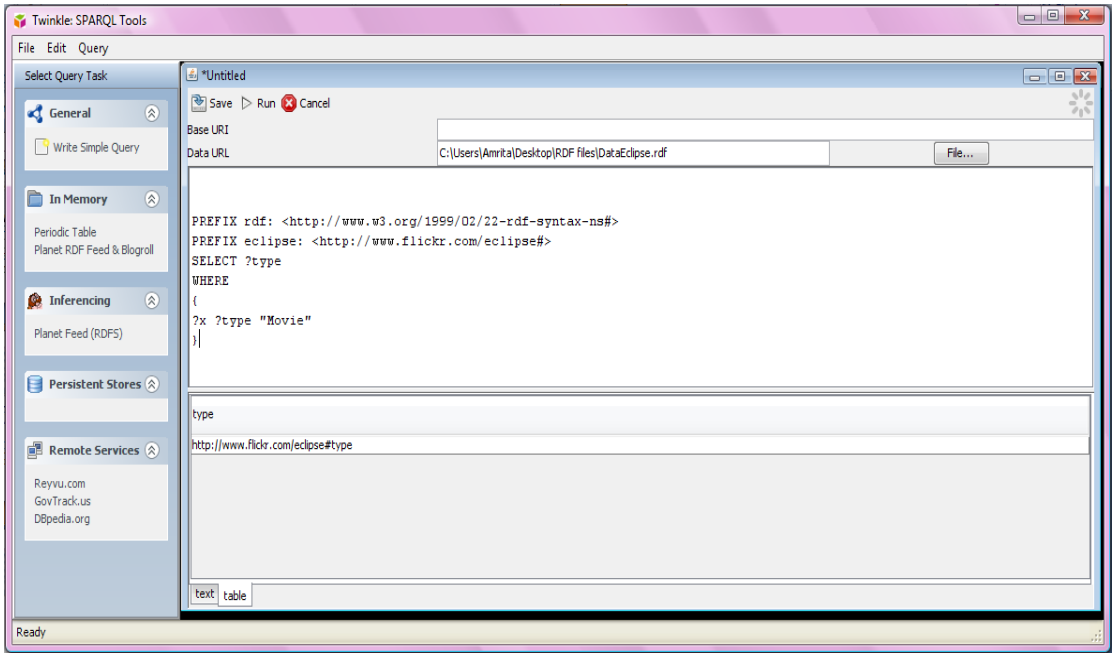

Figure 5.4: Query and output for Literal 'Movie'

Query 2 - RDF Term constraints: Restricting the values of strings

Figure 5.5 shows output for word 'eclipse' of type movie irrespective of other meanings.

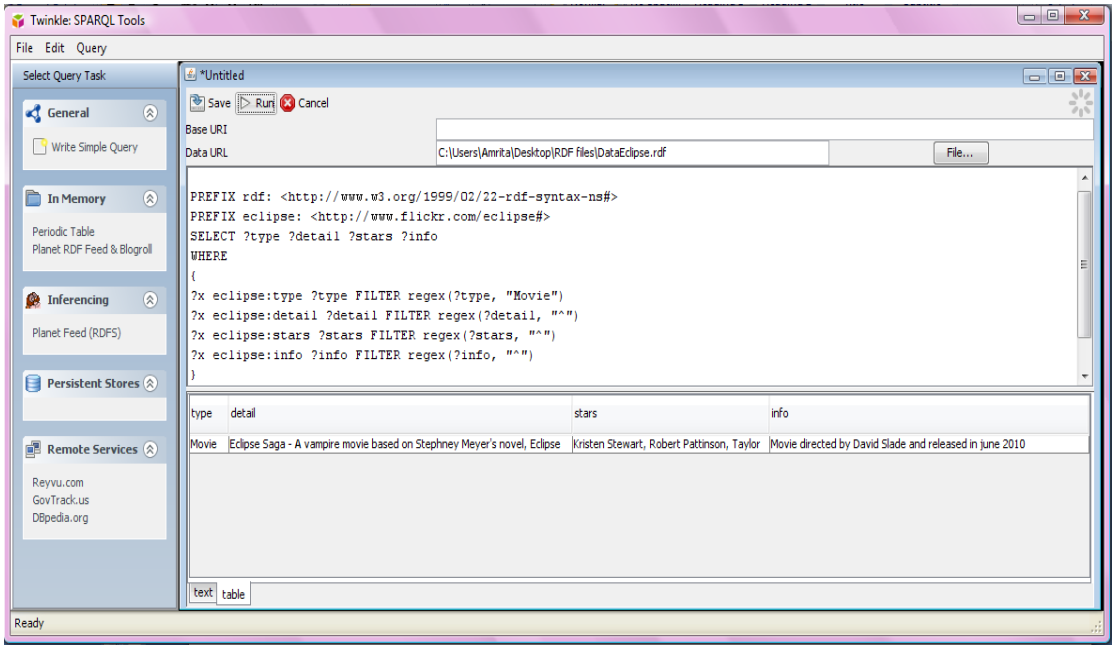

Figure 5.5: Query and output for 'eclipse' of type 'Movie' using FILTER constraint

Query 3 - Optional Pattern Matching

In an optional match, either the optional graph pattern matches a graph, thereby defining and adding bindings to one or more solutions, or it leaves a solution unchanged without adding any additional bindings as shown in Figure 5.6.

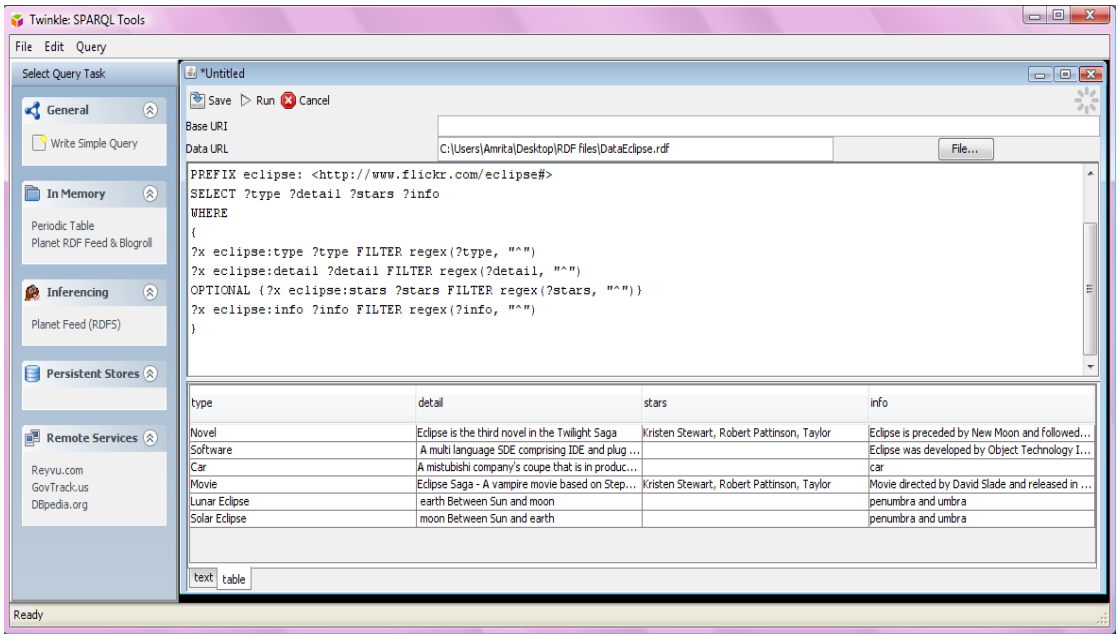

Figure 5.6: Query and output for the type, detail and info, constraining the result for stars which belong to any homonym of ‘eclipse’

Query 4 - Multiple Optional Graph Patterns

Figure 5.7 shows the results for retrieving multiple optional graph patterns.

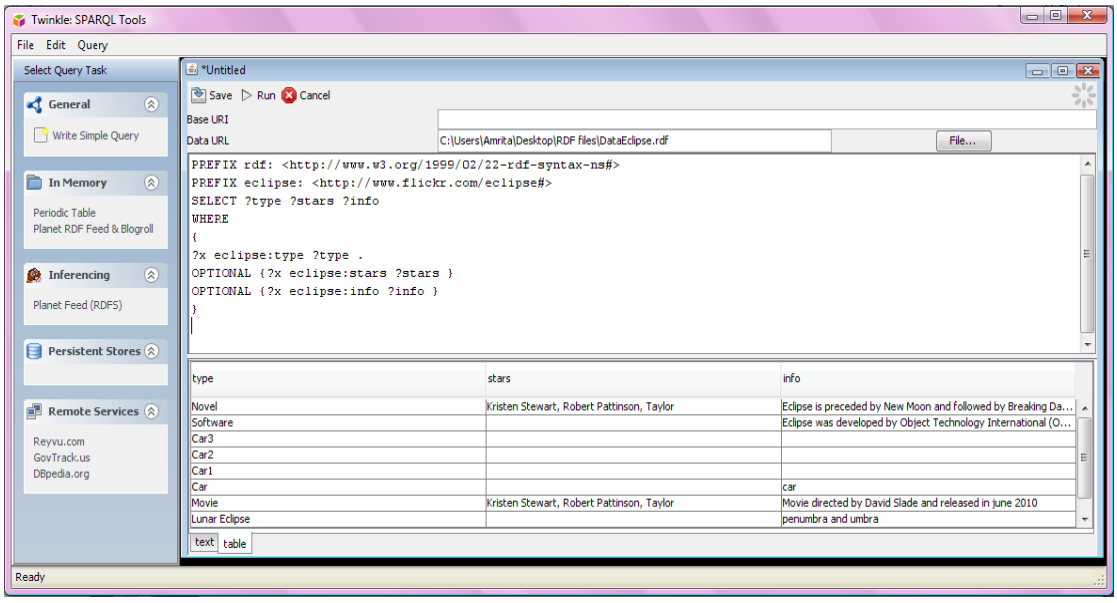

Figure 5.7: Query and output for type, stars and info, constraining the result for stars, info

Query 5 - Matching Alternatives

Figure 5.8 shows the results for matching alternatives for all types of words for 'eclipse' which have either the information for stars or the info.

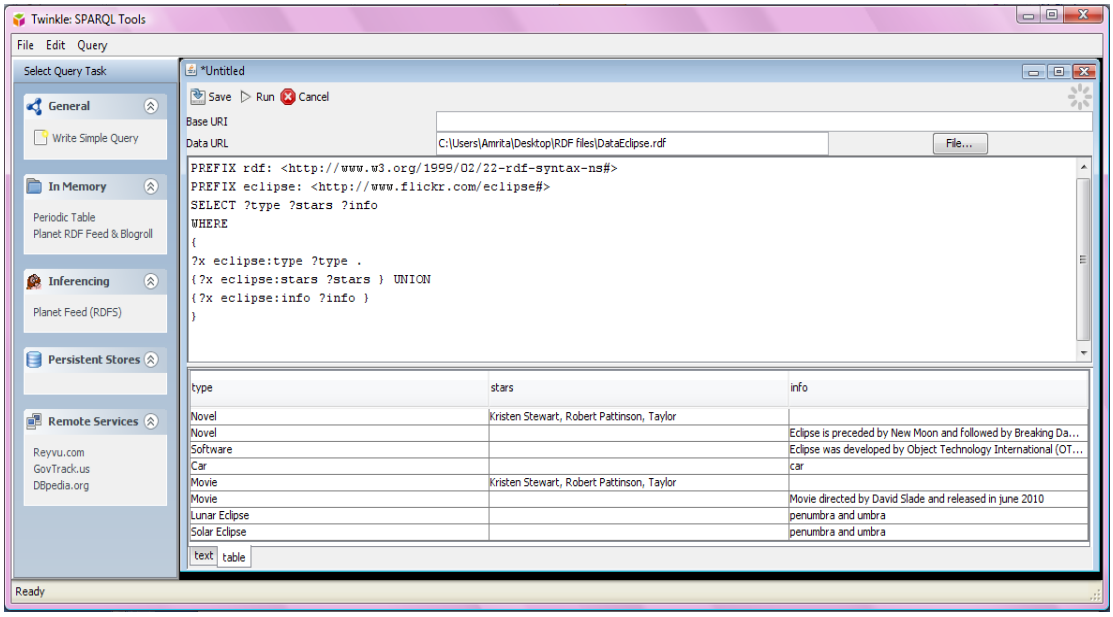

Figure 5.8: Query and output for type of word 'eclipse' by matching the alternatives

Query 6 – Solution Sequences and modifiers

1. ORDER BY :

Figure 5.9 shows results for querying type, stars and info and sorting results by type.

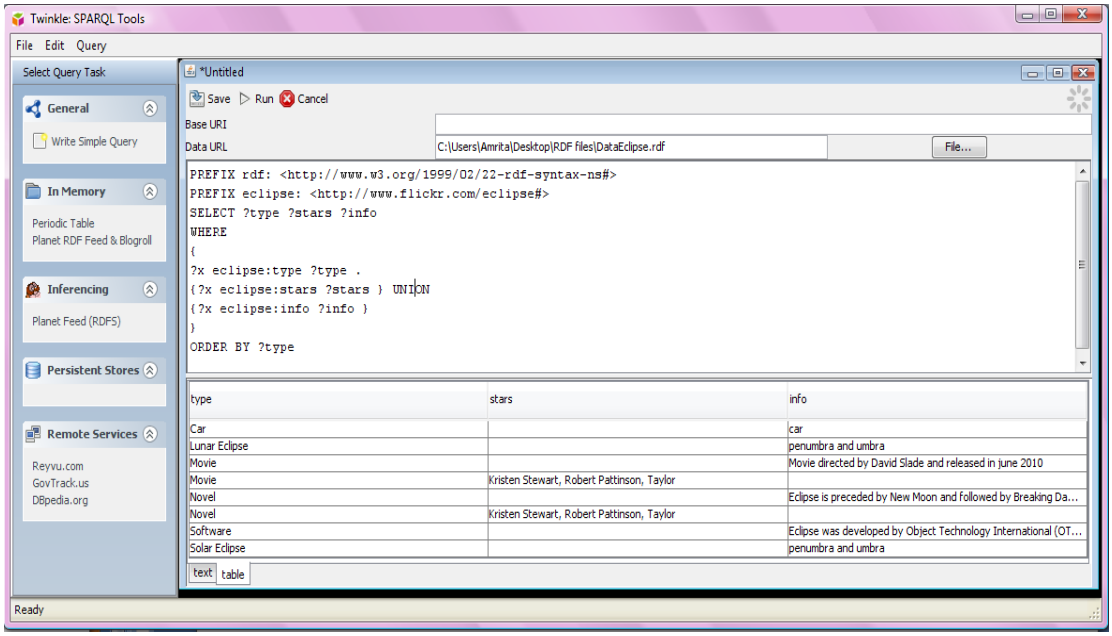

Figure 5.9: Query and output for type, stars, info and sorting type by using ORDER BY

2. PROJECTION:

Figure 5.10 shows the result for projecting the output for all the types of word eclipse.

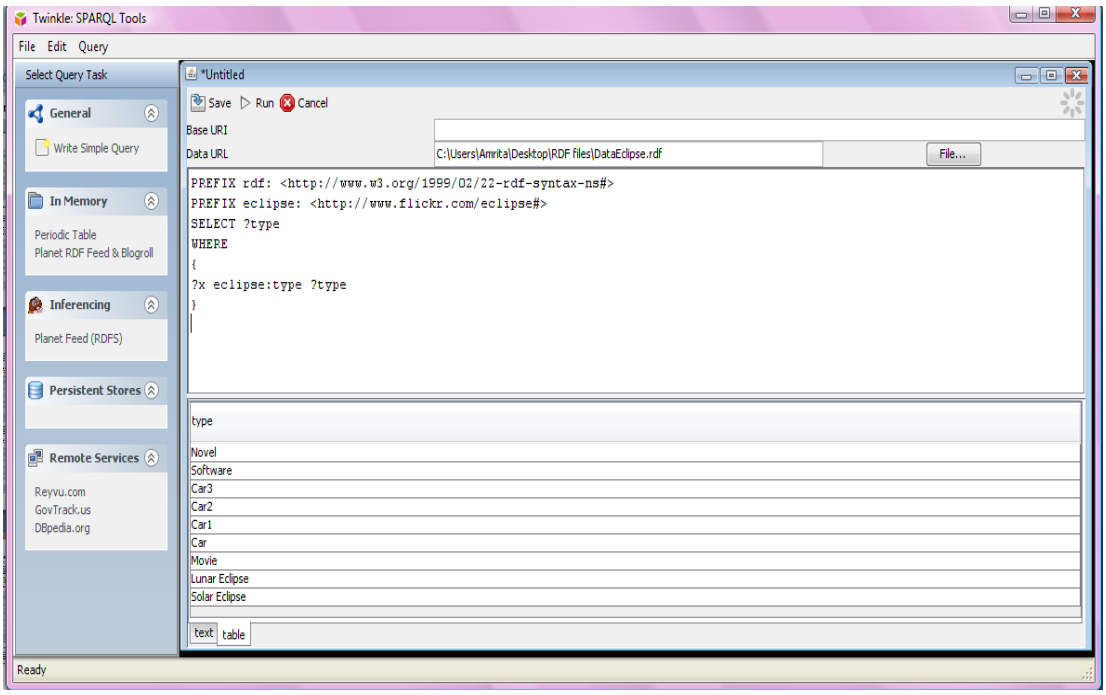

Figure 5.10: Query and output for projecting results of all the types of word ‘eclipse’

3. Duplicate solutions, DISTINCT / REDUCED

Figure 5.11 shows the results for the information that are distinct.

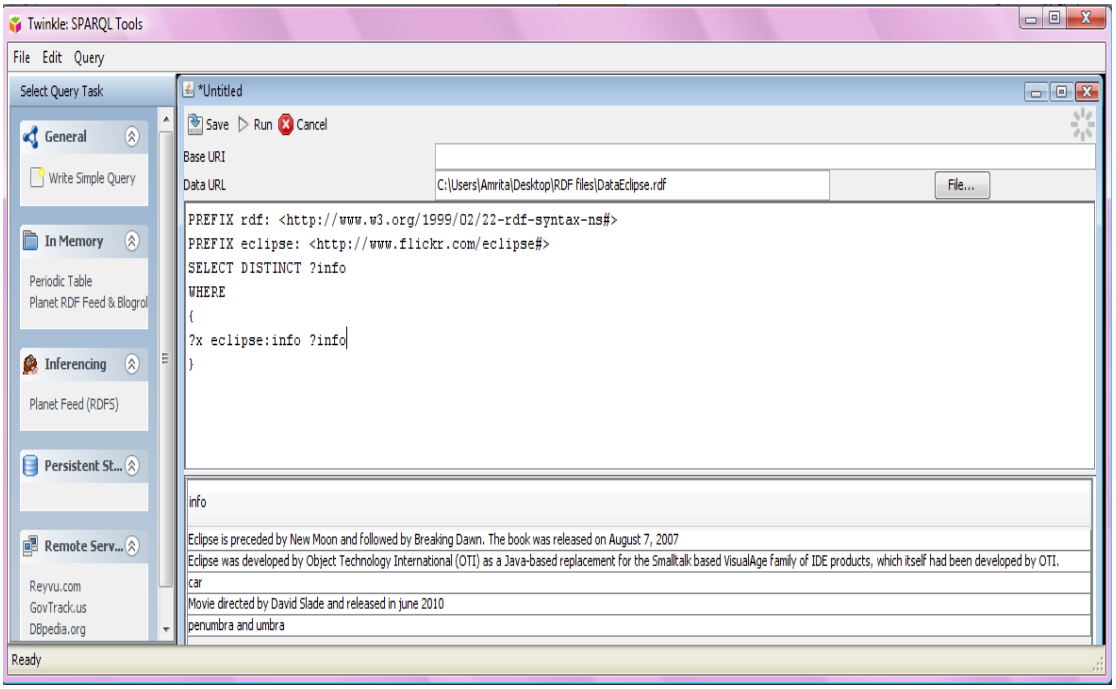

Figure 5.11: Query and output for information, belonging to all the types and are distinct

4. OFFSET

OFFSET causes the solutions generated to start after the specified number of solutions as shown in Figure 5.12. An OFFSET of zero has no effect.

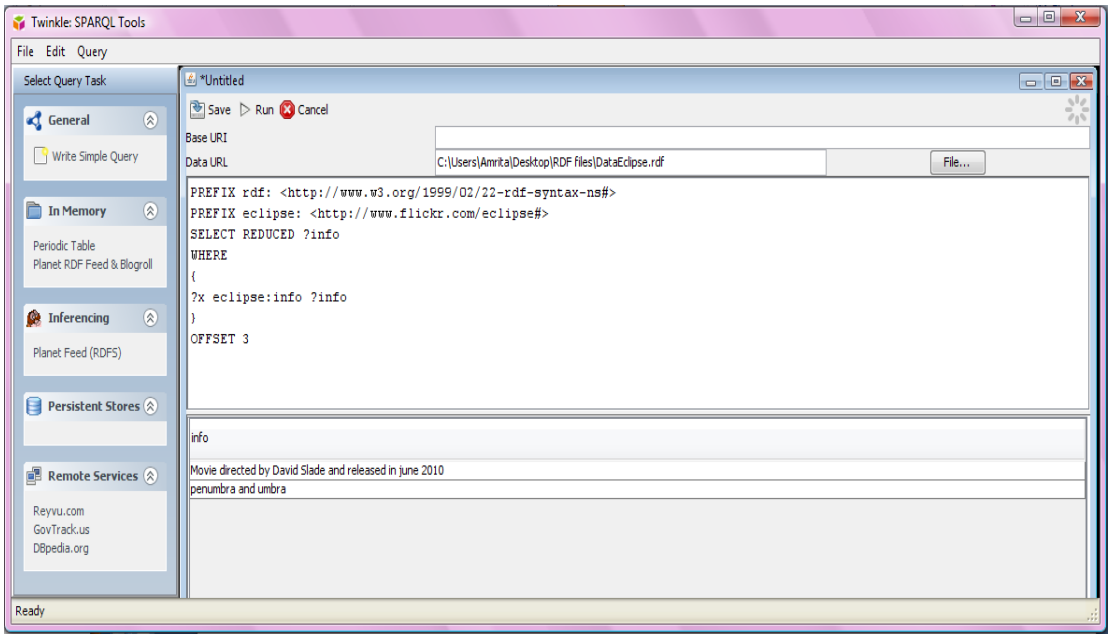

Figure 5.12: Query and output for the info after leaving the top 3 results

5. LIMIT

The LIMIT clause puts an upper bound on the number of solutions returned. If the number of actual solutions is greater than the limit, then at most the limit number of solutions will be returned as shown in Figure 5.13.

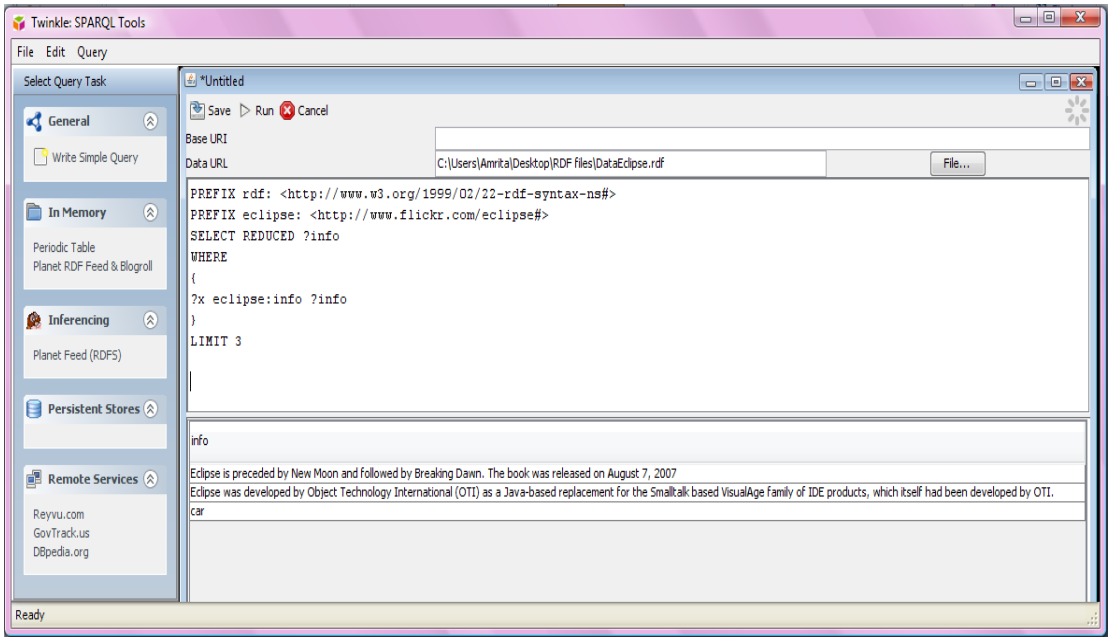

Figure 5.13: Query and output for the info for only the top 3 results

5.3 Installation of SESAME and running query on it

The Sesame 2.0 server software comes in the form of two Java Web Applications:

- Sesame (HTTP) server
- OpenRDF Workbench.

Sesame Server provides HTTP access to Sesame repositories and is meant to be accessed by other applications. Apart from some functionality to view the server's log messages, it doesn't provide any user oriented functionality; which is provided by the OpenRDF workbench. OpenRDF Workbench provides a web interface for querying, updating and exploring the repositories of a Sesame Server.

Firstly, Sesame 2.0 SDK has to be downloaded. The Sesame Server and OpenRDF Workbench can be found in the war directory of the SDK. The war-files in this directory need to be deployed in a Java Servlet Container i.e. apache tomcat and this deployment process is container specific.

The installation process starts with installing java 5 or newer with its proper path then the environment variable has to be set to JAVA_HOME and its path must be the location of the java where it is installed.

Next step is start Apache Tomcat server. This can be done by following the steps;

- Start button
- Run
- Cmd
- Go to the location directory where Tomcat is installed
- Run startup.

After following these steps Tomcat server will start as shown in figure 5.14, and new command prompt window will open with name Tomcat in the title bar and it will as show the message “INFO: Server startup in 18442 ms”. This message shows that Server has started successfully and the number of milliseconds shown in message may not be the same at all the time.

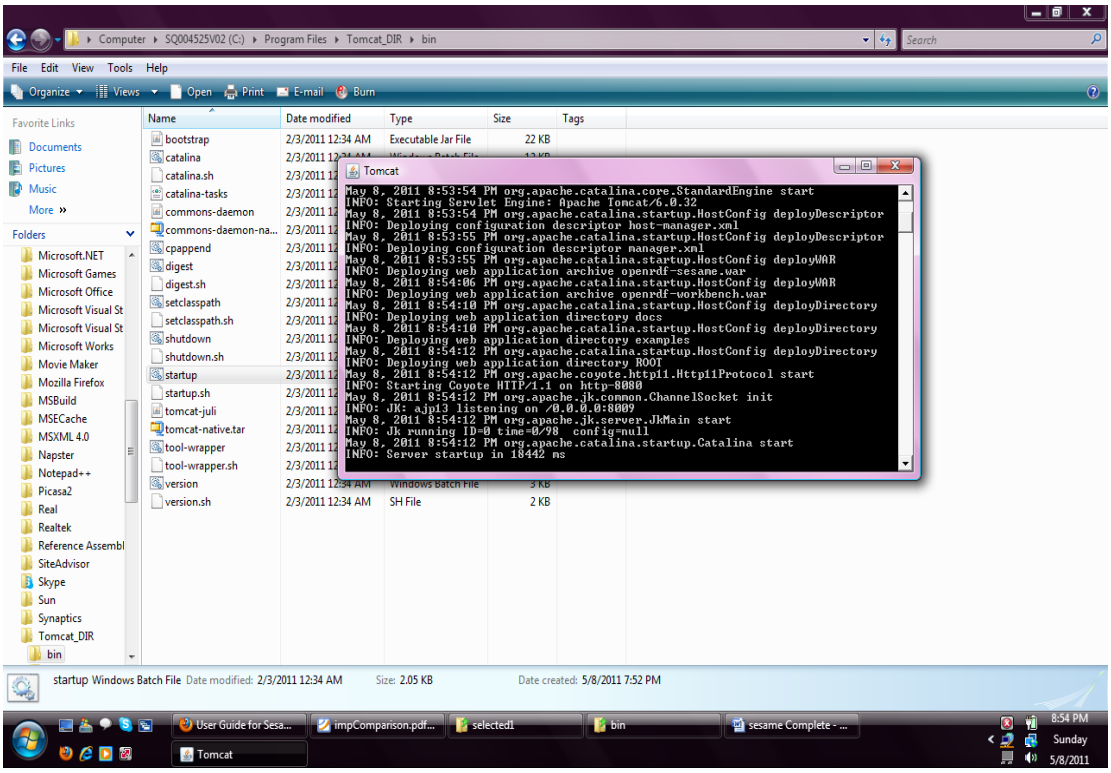

Figure 5.14: Tomcat Server startup window

Next step is to start any browser and write http://localhost:8080/ in the URL link. By clicking on this URL Apache server will run on local machine as shown in figure 5.15.

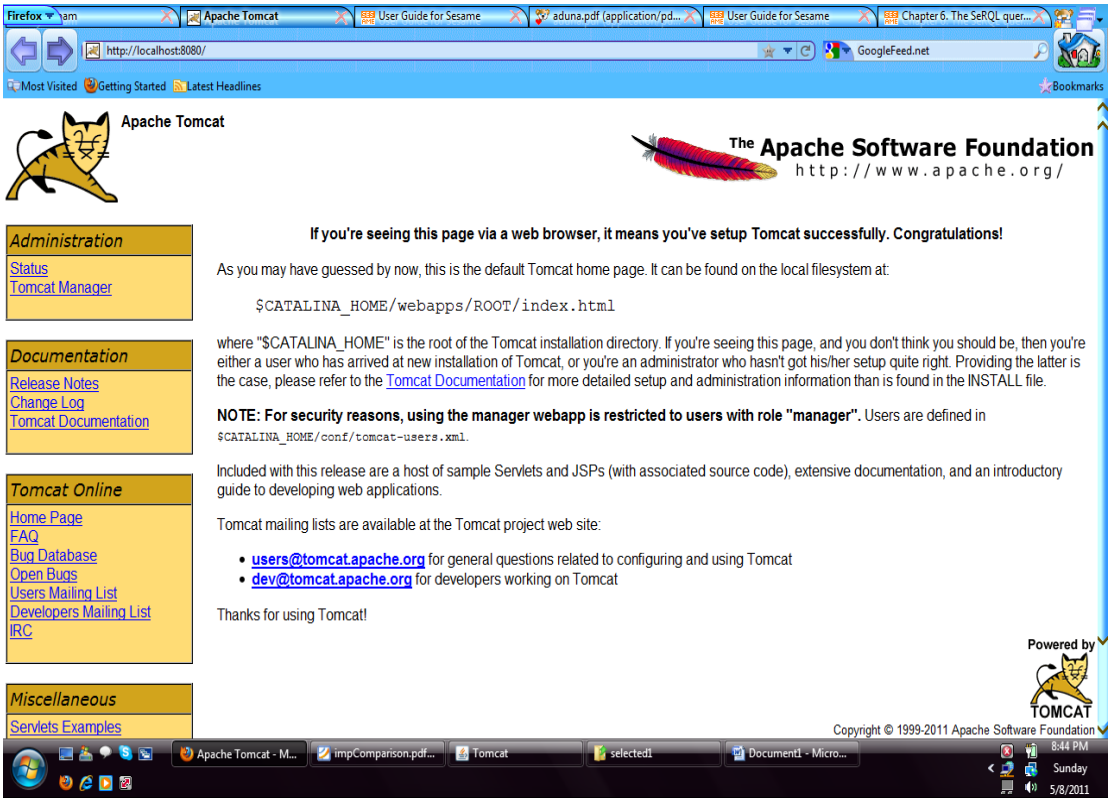

Figure 5.15: Apache tomcat on localhost

The Sesame Server and OpenRDF Workbench war-files have to be deployed here. After deploying the Sesame Server webapp it can be accessed at path /OpenRDF-sesame. Similarly, after deployment, the OpenRDF Workbench should be available at path /OpenRDF-workbench. After deploying Open RDF Workbench, create New Repository as shown in figure 5.16.

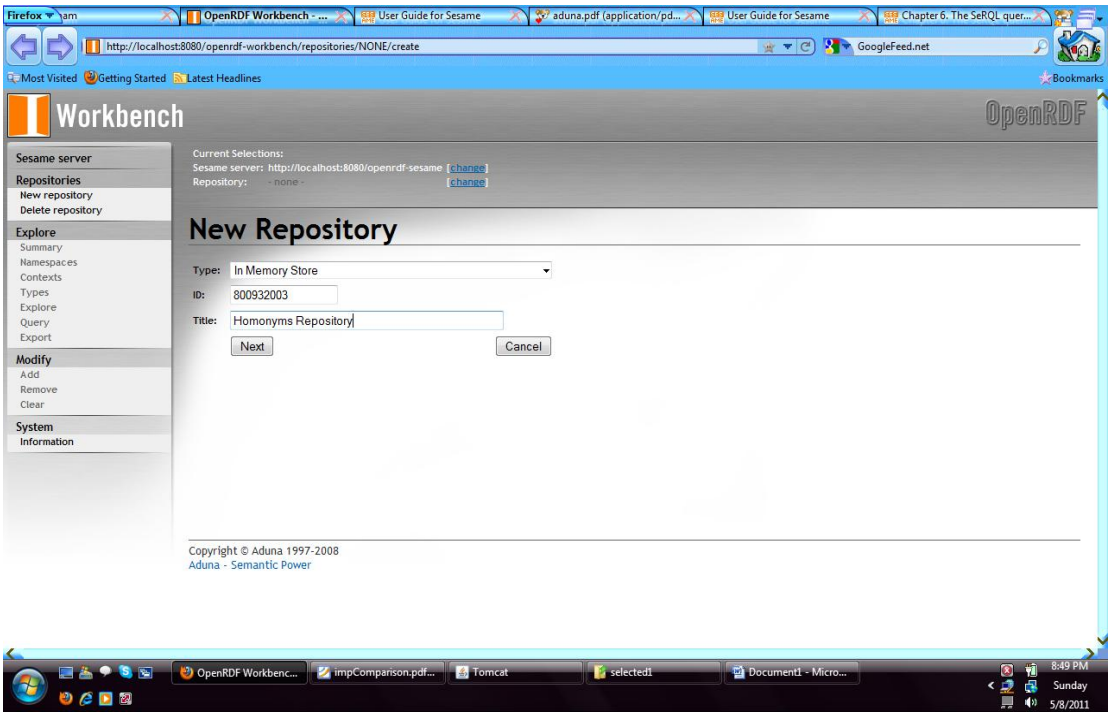

Figure 5.16: Create new repository in OpenRDF workbench

The list of repositories along with the default System repository can be viewed as shown in figure 5.17.

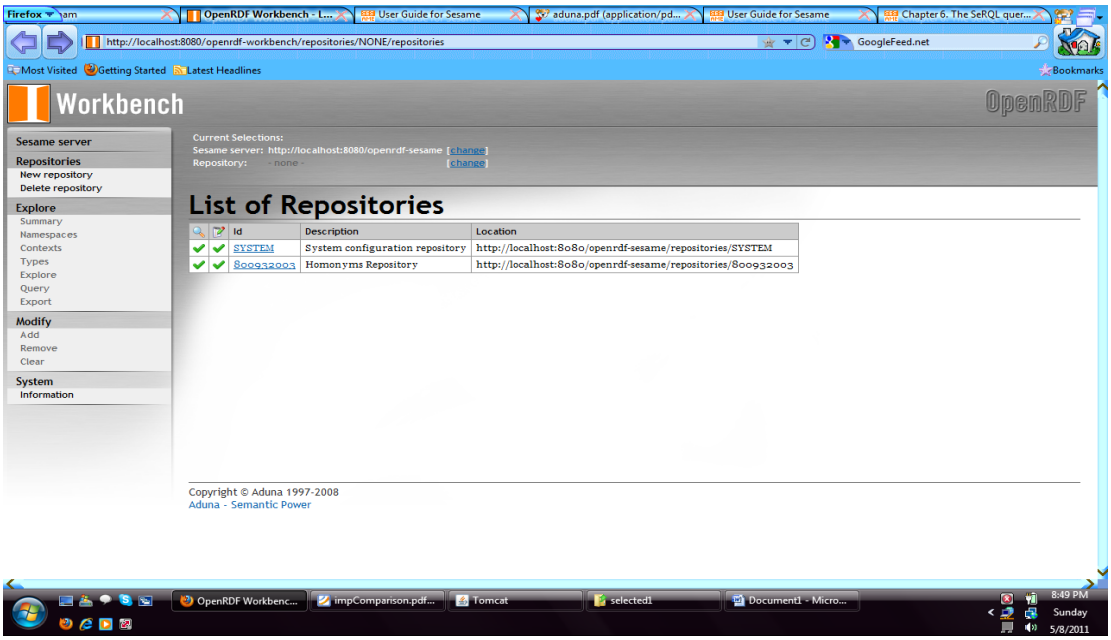

Figure 5.17: List of created repositories

Next step is to query the repositories for all the examples as shown in section 5.3 and an example query using SeRQLis shown in figure 5.18.

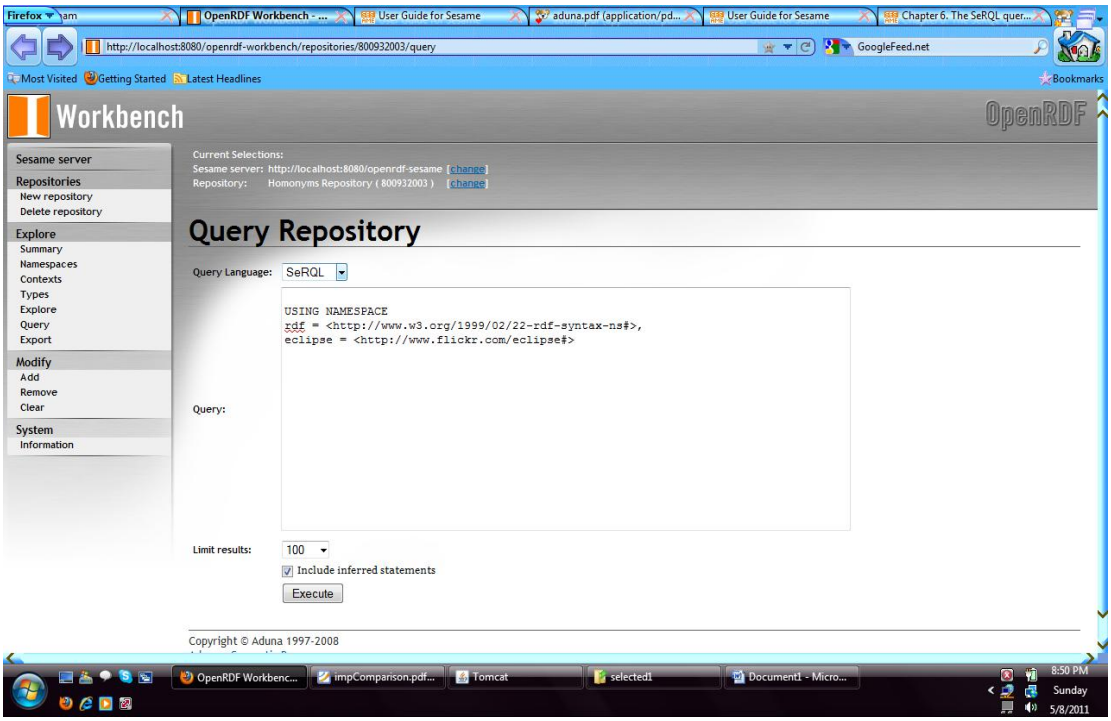

Figure 5.18: The example for Querying repository using SeRQL

To select all the path and the value corresponding to them, a query as shown in figure 5.19 is run.

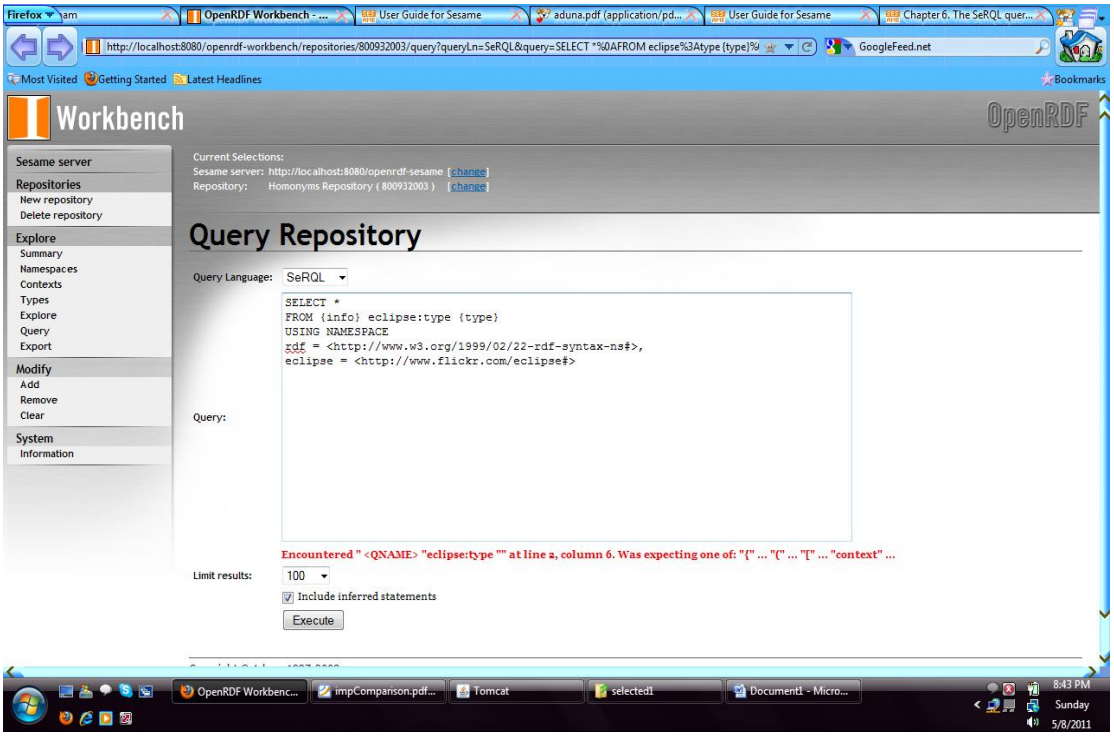

Figure 5.19: Query for selecting the path defined for all the ‘eclipse’ types

The result of this query is shown in figure 5.20.

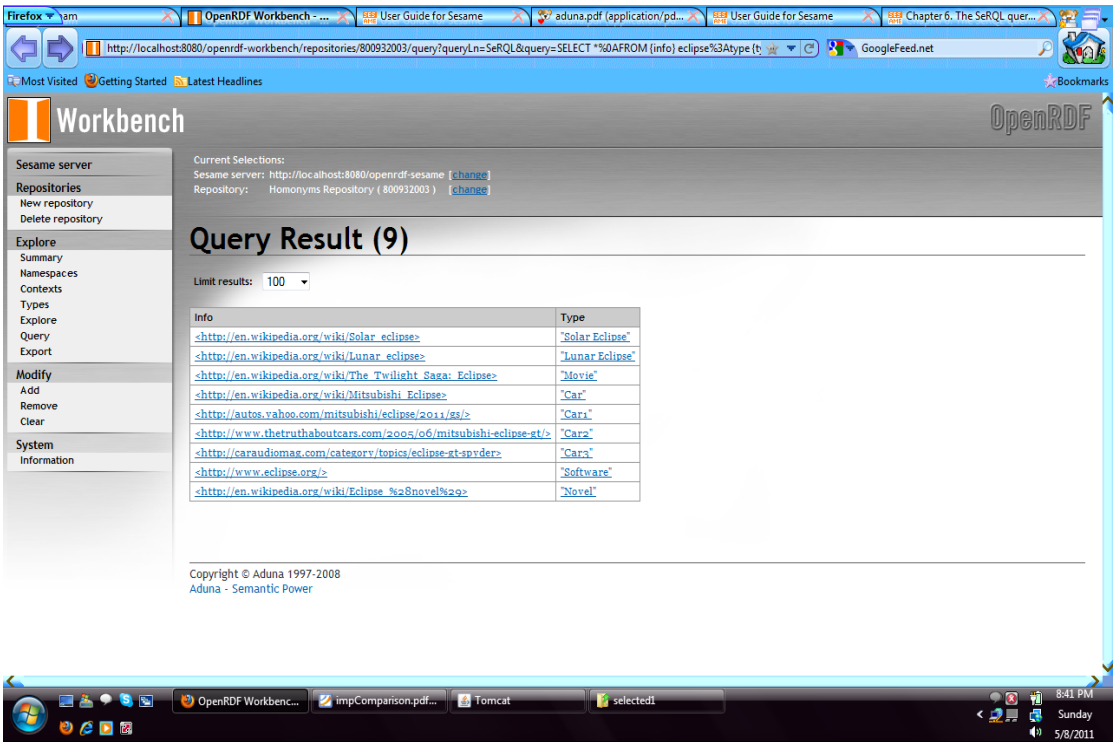

Figure 5.20: Query result for showing the paths of all types of word ‘eclipse’

5.4 Querying the RDQL and SquishQL using JENA

Jena rich API is used for manipulating RDF graphs. Jena is a Java class library, and is composed mainly of API (Application Programming Interface) and SPI (System Programming Interface. This API provides an interface for the Semantic Web application developer that makes it an ideal programming toolkit to process the RDF file. In the RDF API, the key RDF package for the application developer is com.hp.hpl.jena.rdf.model. This package contains interfaces for representing models, resources, properties, literals, statements and all the other key concepts of RDF, and a ModelFactory for creating models.

Code to read a RDF file is:

```
package tutorial;
import java.io.FileInputStream;
import java.io.InputStream;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
public class HelloRDFWorld extends Object {
public static void main (String args[]) {
```

```
String  inputFile="C:/DATA/StudyMaterial/MtechData/Thesis/RDF_Thesis/RDF
files/DataEclipse.rdf";
Model m = ModelFactory.createDefaultModel();
try{
    InputStream in =new FileInputStream(inputFile);
    if (in == null)
        {
            System.out.println("File not found");
        }
        m.read(in," ");
        m.write(System.out);
    }catch(Exception e){}
}
```

Code to examine each triple in a model is:

```
    public static void main (String args[]) {
    String inputFile="C:/Users/Amrita/Desktop/RDF files/DataEclipse.rdf";
    Model m = ModelFactory.createDefaultModel();
    try{
            InputStream in =new  FileInputStream(inputFile);
            if (in == null) {
                        System.out.println("File not found");
                        }
            m.read(in," ");
            }catch(Exception e){}
    StmtIterator iterator = m.listStatements();   // Statements composed of triples
            while( iterator.hasNext() )
            {
                    Statement statement = iterator.nextStatement();
                    Resource subject = statement.getSubject();
                    Property predicate = statement.getPredicate();
                    RDFNode object = statement.getObject();
                    System.out.println( subject.toString() );
                    System.out.println( " " + predicate.toString() );
                    if( object instanceof Resource ){       // it's a resource.
```

```
                System.out.println( " " + object.toString() );      }
            else
            {// it's a literal that will be printed with surrounding quotes.
                System.out.println( "\"" + object.toString() + "\"" );
            }
        }
    }
```

5.5 Comparison of the Query Languages

On the basis of the results obtained by querying the RDF repository of homonyms for all the features listed to be important and necessary for being a robust query language, the result is shown in the table 5.1 below.

Table 5.1: Scorecard for the various RDF Query Languages

	Generic Path Expression	**The Closure Property**	**Data Type Support**	**The Value Support**	**The Semantic Capability**	**The Optional Values**	**The Aggregate functions**	**Mathematical Set Operation**
SquishQL	Yes	No	Yes	No	No	No	No	No
RDQL	Yes	No	Yes	Yes	No	No	No	No
SPARQL	Yes	Yes	Yes	Yes	No	Yes	No	No
LAQRS	Yes	Yes	Yes	Yes	Yes	Yes	No	No
SeRQL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
XsRQL	Yes	No	Yes	Yes	No	No	Yes	No

CHAPTER 6

CONCLUSIONS AND FUTURE SCOPE

6.1 Conclusions

This thesis work concludes that the Semantic search gives more efficient and accurate results as compared to Syntactic search and RDF is a very powerful tool in Semantic search to annotate homonyms and create data models for the repositories of different types of resources. The predefined tags used in RDF capture all the targeted information about the resources (homonyms in our case) and put ahead a way to make all the searches over the defined repository more relevant and accurate.

On the basis of Table 5.1, showing scorecard for prominent RDF query languages, the query language that provides most of the desired language features for the retrieval of the required results from the RDF repository is Sesame Query language (SeRQL) and the features provided by this Query Language are:

- The Generic Path Expressions
- The Closure Property
- Data Type Support
- Value Support
- Semantic Capabilities
- Optional Values
- Aggregate Functions

SeRQL with strong support from the open source community and SPARQL with strong support from the W3C are leading the charge towards standardization. Standardization of RDF Query is an important step in evolution of the Semantic Web and for efficient Semantic retrieval. Both, the RDF and the SeRQL combined together can serve as a platform to fulfill the vision of Semantic Web, as proposed by Tin Berners Lee.

6.2 Future Scope

In the presented work, only a sample RDF repository is created which can be modified by adding more annotations to it for making retrieval at a more fine level of granularity that can bring additional information to data model of Homonym – Eclipse.

A generic repository for all the homonyms can be created thereby enhancing the users' capability to search more meaningful contents or making a way to enhanced semantic search over the Internet. The extension of the Semantic retrieval can be done through the use of OWL and intelligent fuzzy retrieval can be achieved by using Fuzzy Logic.

REFERENCES

[1] J. Hendler, T. Berners-Lee, and E. Miller. "Integrating Applications on the Semantic Web", Journal of the Institute of Electrical Engineers of Japan, Vol. 122, pp. 10, October 2002.

[2] F. Giunchiglia, U. Kharkevich, and I. zaihrayeu, "Concept search: Semantics enabled information retrieval", Technical Report # DISI-10-004, DISI, Trento (Italy), January 2010, [Online document] Available: http://www.disi.unitn.it.

[3] J. Zhai and K. Zhou, "Semantic Retrieval for Sports Information Based on Ontology and SPARQL", International Conference of Information Science and Management Engineering, 2010.

[4] S. Gauch, J. Chaffee, and A. Pretschner, "Ontology-Based Personalized Search and Browsing", Web Intelligence and Agent Systems, Vol. 1, No. 3-4, pp. 219-234, 2003.

[5] Jason C. Hung, "The Semantic Annotated Documents - From HTML to the Semantic Web", Proceedings of the 200 WSEAS International Conference on Computer Engineering and Applications, Gold Coast, Australia, January 2007, [Online document] Available: http://member.mine.tku.edu.tw/www/jhung.

[6] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks, "The Semantic Web: The Roles of XML and RDF", IEEE Internet Computing, Vol. 4(5), pp. 63-74, Sept/Oct 2000.

[7] Resource Description Framework (RDF) [Online document] Available: http://www.w3.org/RDF/.

[8] E. Prud'hommeaux, A. Seaborne, "SPARQL Query Language for RDF", W3C Working Draft 12 October 2, [Online document] Available: http://www.w3.org/tr/rdf-sparql-query/.

[9] F. Manola, B. McBride, and E. Miller, "RDF Primer", A W3C Recommendation,
February 2004, [Online document] Available: http://www.w3.org/TR/rdf-primer/.

[10] D. Brickley, R. Guha, "Resource Description Framework (RDF) Schema Specification", W3C Candidate Recommendation, March, 2000, [Online document] Available: http://www.w3.org/TR/2000/CR-rdf-schema- 20000327.

[11] O. Lassila and R. Swick, “Resource Description Framework (RDF) Model and Syntax Specification”, World Wide Web Consortium Recommendation, Feb 1999.

[12] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff, “Semantic annotation, indexing, and retrieval", Journal of Web Semantics. Vol 2, No 1, pp 49-79, 2004.

[13] K. Jose, K. Marja-Ritta, Eric Prud H, and Ralph R., “Annotea: An Open RDF Infrastructure for Shared Web Annotations”, In Proceedings of WWW 10th International Conference, Hong Kong, May 2001.

[14] “Semantic Annotation”, [Online document] Available: http://www.ontotext.com/kim/semanticannotation.html.

[15] S. Elbassuoni, M. Ramanath, R. Schenkel, and G. Weikum, “Searching RDF Graphs with SPARQL and Keywords”, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2010.

[16] K. Hutt, “A Comparison of RDF Query Languages”, In 21st Computer Science Seminar, SE1-T4-1, 2009.

[17] W. Nejdl, W. Siberski and M. Sintek, “Design Issues and Challenges for RDF and Schema–Based Peer-to-Peer Systems”, SIGMOD Record, Vol 32, No. 3, pp. 41- 46, September 2003.

[18] A. Seaborne, “RDQL, A Query Language for RDF”, W3C Member Submission, 2004, [Online document], Available: http://www.w3.org/submission/2004/subm-rdql-20040109.

[19] C. Kiefer, “Imprecise SPARQL: Towards a Unified Framework for Similarity – Based Semantic Web Tasks”, Knowledge Web PhD Symposium 2007, 2007.

[20] Pérez, M. Arenas, and C. Gutierrez, “Semantics and Complexity of SPARQL,” ACM Trans. Database Systems, Vol. 34, No. 3, 2009, [Online document] Available: http://doi.acm.org/10.1145/1567274.1567278.

[21] J. Broekstra, A. Kampman, “Se-RQL: An RDF Query and Transformation Language, Draft”, [Online document] Available: http://www.cs.vu.nl/~jbroeks/papers/SeRQL.pdf.

[22] H. Katz, “XsRQL: An X-Query-style Query Language for RDF, A Submission to the RDF Data Access Working Group”, June 2004, [Online document], Available: http://www.fatdog.com/xsrql.html.

[23] History of Search Engines: From 1945 to Google Today, [Online document] Available: http://www.searchenginehistory.com/.

[24] D. Mukhopadhya, A. Banik, S. Mukherjee, and J. Bhattacharya, "A Domain Specific Ontology Based Semantic Web Search Engine", Arxiv preprint, arXiv, 2011, Available: arxiv.org

[25] T. Berners-Lee, "Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor", New York: Harper San Francisco, 1999.

[26] S. Schenk and S. Staab, "Networked Graphs: A Declarative Mechanism for SPARQL Rules, SPARQL Views and RDF Data Integration on the Web", WWW 2008, Referred Track: Semantic / Data Web - Semantic Web II April 21-25, 2008.

[27] C. Schonberg, B. Freitag, "Evaluating RDF Querying Frameworks for Document Metadata", Technical report, MIP – 0903, University of Passau, February 2009.

[28] L. Miller, A. Seaborne and A. Reggiori, "Three Implementations of SquishQL, a Simple RDF Query Language", First International Semantic Web Conference (ISWC 2002), Sardinia, ILRT, Bristol University, Bristol, UK, June, 2002.

[29] "JENA - A Semantic Framework for Java", [Online document] Available: http://jena.sourceforge.net/.

[30] K. Wilkinson, C. Sayers, H. Kuno and D. Reynolds, "Efficient RDF Storage and Retrieval in Jena2", [Online document] Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.5486.pdf/.

[31] J. Jeremy Carroll, I. Dickinson and C. Dollin, "Jena: Implementing the Semantic Web Recommendations", ACM 1-58113-912-8/04/0005, WWW 2004, New York, USA, May, 2004.

[32] A. Kampman and J. Broekstra, "SeRQL: A Second Generation RDF Query Language", W3C recommendation, Nov 2003, [Online document] Available: http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/aduna.pdf.

[33] "Sesame Architecture for Storing and Querying RDF Data", [Online Document] Available: http://www.openrdf.org/doc/sesame/users/ch01.html

LIST OF PUBLICATIONS

Amrita Bhandari and Shalini Batra, "Semantic Retrieval for Homonyms using RDF and SPARQL", Journal of Global Research in Computer Science, Volume 2, No. 4, April 2011. Available at: http://jgrcs.info/index.php/jgrcs/article/viewFile/183/103.

APPENDIX - A

The original RDF/XML document developed for various meanings of Homonym – 'Eclipse'.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:eclipse="http://www.flickr.com/eclipse#">
<rdf:Description rdf:about="http://en.wikipedia.org/wiki/Solar_eclipse">
        <eclipse:type>Solar Eclipse</eclipse:type>
        <eclipse:detail> moon Between Sun and earth</eclipse:detail>
        <eclipse:total>moon covers sun</eclipse:total>
        <eclipse:annular>moon appears smaller than sun</eclipse:annular>
        <eclipse:hybrid>sometimes total, sometimes annular</eclipse:hybrid>
        <eclipse:partial>moon partially covers sun</eclipse:partial>
        <eclipse:info>penumbra and umbra</eclipse:info>
</rdf:Description>
<rdf:Description rdf:about="http://en.wikipedia.org/wiki/Lunar_eclipse">
        <eclipse:type>Lunar Eclipse</eclipse:type>
        <eclipse:detail> earth Between Sun and moon</eclipse:detail>
        <eclipse:total>earth covers moon</eclipse:total>
        <eclipse:hybrid>both</eclipse:hybrid>
        <eclipse:partial>earth partially covers sun</eclipse:partial>
        <eclipse:info>penumbra and umbra</eclipse:info>
</rdf:Description>
<rdf:Description
rdf:about="http://en.wikipedia.org/wiki/The_Twilight_Saga:_Eclipse">
        <eclipse:type>Movie</eclipse:type>
        <eclipse:detail>Eclipse Saga - A vampire movie based on Stephney Meyer's
        novel, Eclipse</eclipse:detail>
        <eclipse:stars>Kristen Stewart, Robert Pattinson, Taylor</eclipse:stars>
        <eclipse:vampire>Robert as Edward Cullen</eclipse:vampire>
        <eclipse:Actress>Kristen as Bella Swan</eclipse:Actress>
        <eclipse:wolf>Taylor as Jacob Black</eclipse:wolf>
```

```
        <eclipse:info>Movie directed by David Slade and released in june
        2010</eclipse:info>
</rdf:Description>
<rdf:Description rdf:about="http://en.wikipedia.org/wiki/Mitsubishi_Eclipse">
        <eclipse:type>Car</eclipse:type>
        <eclipse:detail>A mistubishi company's coupe that is in production since 1989
        for right hand driven markets</eclipse:detail>
        <eclipse:brand>Mistubishi</eclipse:brand>
        <eclipse:meaning>Racehorse</eclipse:meaning>
        <eclipse:info>car</eclipse:info>
</rdf:Description>
<rdf:Description rdf:about="http://autos.yahoo.com/mitsubishi/eclipse/2011/gs/">
        <eclipse:type>Car1</eclipse:type>
        <eclipse:name> Eclipse GS </eclipse:name>
        <eclipse:color> red </eclipse:color>
</rdf:Description>
<rdf:Description rdf:about="http://www.thetruthaboutcars.com/2005/06/mitsubishi-
eclipse-gt/">
        <eclipse:type>Car2</eclipse:type>
        <eclipse:name> Eclipse GT </eclipse:name>
        <eclipse:color> white </eclipse:color>
</rdf:Description>
<rdf:Description rdf:about="http://caraudiomag.com/category/topics/eclipse-gt-
spyder">
        <eclipse:type>Car3</eclipse:type>
        <eclipse:name> Eclipse GT-Spyder</eclipse:name>
        <eclipse:color> black </eclipse:color>
</rdf:Description>
<rdf:Description rdf:about="http://www.eclipse.org/">
        <eclipse:type>Software</eclipse:type>
        <eclipse:detail> A multi language SDE comprising IDE and plug in
        system</eclipse:detail>
        <eclipse:Use>Java</eclipse:Use>
```

<eclipse:developer>free and open source software community</eclipse:developer>
<eclipse:platform>Equinox</eclipse:platform>
<eclipse:info>Eclipse was developed by Object Technology International (OTI) as a Java-based replacement for the Smalltalk based VisualAge family of IDE products, which itself had been developed by OTI.</eclipse:info>
</rdf:Description>
<rdf:Description
rdf:about="http://en.wikipedia.org/wiki/Eclipse_%28novel%29">
<eclipse:type>Novel</eclipse:type>
<eclipse:detail>Eclipse is the third novel in the Twilight Saga</eclipse:detail>
<eclipse:stars>Kristen Stewart, Robert Pattinson, Taylor</eclipse:stars>
<eclipse:author>Stephney Mayor</eclipse:author>
<eclipse:vampire>Robert as Edward Cullen</eclipse:vampire>
<eclipse:Actress>Kristen as Bella Swan</eclipse:Actress>
<eclipse:wolf>Taylor as Jacob Black</eclipse:wolf>
<eclipse:info>Eclipse is preceded by New Moon and followed by Breaking Dawn. The book was released on August 7, 2007</eclipse:info>
</rdf:Description>
</rdf:RDF>