Implementation of Sesame Tool for Semantic Web Search Engine Swoogle

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

Master of Engineering

In

Computer Science and Engineering

Thapar University, Patiala

Submitted By:

Kotalwar Balaji Rameshrao

(Roll No. 80732011)

Under the supervision of:
Mr. Karun Verma
Lecturer

JUNE 2009

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

Certificate

I hereby certify that the work which is being presented in the thesis report entitled, **"Implementation of Sesame tool for Semantic Web Search Engine Swoogle."**, 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 a authentic record of my own work carried out under the supervision of Mr. Karun Verma.

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

Kotalwar Balaji Rameshrao

Roll No. 80732011.

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

Mr. Karun Verma
Lecturer, CSED
Thapar University
Patiala-147004, Punjab.

Countersigned by:

Head
Computer Science & Engineering Department,
Thapar University,
Patiala.

26/06/2009

(Dr. R.K.Sharma)
(Dean Academic Affairs)
Thapar University,
Patiala.

Acknowledgement

It is a great pleasure for me to acknowledge the guidance, assistance and help I have received from Mr. Karun Verma, Lecturer, Computer Science and Engineering Department. He has prepared me well for thesis work and has worked hard to promote my work. I am thankful for his continual support, encouragement, and invaluable suggestions throughout the thesis work. He always offered words of encouragement when I was feeling down. He not only provided me help whenever needed, but also provided me the resources required to complete this thesis report on time.

I would like to say special thanks to Dr. Rajesh Kumar Bhatia, Head, Computer Science and Engineering Department for giving me an opportunity to do this work.

I am also thankful to Dr. Mrs. Seema Bawa Professor, Computer science and Engineering Department As well as Dr. Mrs. Inderveer Channa PG coordinator for providing me required help for the completion of this work.

I am deeply grateful to my parents and brother for the inspiration and ever encouraging moral support, which enabled me to pursue my studies.

I would like to thanks for everyday support to my dearest Classmates. I want to express my appreciation to every person who contributed with either inspirational or actual work to this report.

Kotalwar Balaji

Abstract

"Change is the Rule of the World."

Everything in this world is keep go on changing because change is the rule of the world. Everyone and Everything is going so fast and so accurate, cause world need accuracy as well. So to run with the world we need to remove our drawbacks and have to adopt new technology to overcome the disadvantages of existing system. The present way of the World Wide Web (WWW) has taken the availability of information to an unprecedented level. The rapid growth of the web poses new problems. The next generation of Web is expected to be the Semantic Web. The vision of the Semantic Web is to give data on the web a well defined meaning by representing it in languages like RDF and OWL and linking it to commonly accepted ontology.

The Semantic Web provides a way to encode information and knowledge on web pages in a form that is easier for computers to understand and process. This Thesis discusses the issues underlying the discovery, indexing and search over web documents that contain semantic web markup. Unlike conventional Web search engines, which use information retrieval techniques designed for documents of unstructured text, Semantic Web search engines must handle documents comprised of semi-structured data. Moreover, the meaning of data is defined by associated ontologies that are also encoded as semantic web documents whose processing may require significant amount of reasoning.

This thesis work describe Swoogle, semantic web search engine that discovers, analyzes, and indexes knowledge encoded in semantic web documents throughout the Web, and its use to help human users and software agents find relevant knowledge. As Swoogle Search Engine extracts SWD which are store in the repositories so there is need of repositories. Sesame can be used for creating arbitrary repositories, ranging from traditional Data Base Management Systems, to dedicated RDF triple stores. Sesame also implements a query engine for RQL, the most powerful RDF/RDF Schema query language to date.

Table of Contents

List of Figures and Tables

CHAPTER 1
INTRODUCTION

As the scale and the impact of the World Wide Web have grown, search engines have assumed a central role in the Web's infrastructure. In the earliest days of the Web, people found pages of interest by navigating (quickly dubbed surfing) from pages whose locations they remembered or bookmarked. Rapid growth in the number of pages gave rise to web directories like Yahoo that manually organized web pages into taxonomy of topics. As growth continued, these were augmented by search engines such as Lycos, HotBot and AltaVista, which automatically discovered new and modified web pages, added them to databases and indexed them by their keywords and features. Today, the search engines such as Google and Yahoo dominate the Web's infrastructure and largely define our Web experience.

Most knowledge on the Web is presented as natural language text with occasional pictures and graphics. This 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 can't infer meaning that a page is referring to a bird called Parrot or the sports team with the same name is not evident to them. Thus users share a significant burden in terms of constructing the search query intelligently. Even with increased use of XML encoded information, computers still need to process the tags and literal symbols using application dependent semantics. The Semantic Web offers an approach in which knowledge can be published by and shared among computers using symbols with a well defined, machine-interpretable semantics [10].

Search on the Semantic Web differs from conventional web search for several reasons: First, Semantic Web content is intended to be published by machines for machines, e.g., tools, web services, software agents, information systems, etc. Semantic Web annotations and markup may well be used to help people find human readable documents, but there will likely be a layer of agents between human users and Semantic Web search engines [22].

Second, knowledge encoded in semantic web languages such as RDF differs from both the largely unstructured free text found on most web pages and the highly structured information found in databases. Such semi-structured information requires

a combination of techniques for effective indexing and retrieval. RDF and OWL introduce aspects beyond those for ordinary XML, allowing one to define terms (i.e., classes and properties), express relationships among them, and assert constraints and axioms that hold for well formed data.

Third, Semantic Web documents can be a mixture of concrete facts, classes and property definitions, logic constraints and metadata, even within a single document. Fully understanding the document can require substantial reasoning, so search engines will have to face the design issue of how much reasoning to do and when to do it. This reasoning produces additional facts, constraints and metadata which may also need to be indexed, potentially along with the supporting justifications. Conventional search engines do not try to understand document content because the task is just too difficult and requires more research on text understanding.

Finally, the graph structure formed by a collection of Semantic Web documents differs in significant ways from the structure that emerges from the collection of HTML documents. This will influence effective strategies to automatically discover Semantic Web documents as well as appropriate metrics for ranking their importance.

Hence the next generation of search engine would be: Semantic Web search system Swoogle. Swoogle discovers, analyzes and indexes semantic web documents on the web. Just as modern search engines and their services have given people access to much of the world's knowledge and data [13]. Swoogle aims to support semantic web developers, software agents and programmers in finding RDF data and schema level knowledge on the web. In Today's condition number of available SWD at Swoogle are very less to increase the SWD need to create and import SWDs repositories.

Sesame is an open source Java framework for creating, storing and querying of repositories. The framework is fully extensible and configurable with respect to storage mechanisms, inferences, RDF file formats, query result formats and query languages. Sesame supports SPARQL and SeRQL querying, a memory-based and a disk-based RDF store and RDF Schema inferences. It also supports most popular RDF file formats and query result formats. We will demonstrate the numerous features of the Sesame framework in several scenarios.

1.1 Semantic web

1.1.1 Motivation

An ample amount of information is now days available via the World Wide Web which has led to enhancement, effectiveness and selectivity of various search engines. The World Wide Web is possible only because of a set of broadly established standards that guarantee interoperability at various levels. Till now, the Web has been intended for direct human processing, but the next-generation Web called the "Semantic Web", aims at machine-processable information. The word "Semantic" refers to "meaning" a semantic web is actually a web of meaning or it can be said that it is a web that knows what the entities on the web mean and can make use of that knowledge efficiently [1]. It can only become possible if further levels of interoperability have been established. Standards have to be defined not only for the syntactic form of documents, but also for their semantic content. Semantic Web will enable intelligent services, which offer greater functionality and interoperability than current stand-alone services and for the semantic web to function properly and efficiently, computers must have an access to structured collections of information and sets of information and sets of inference rules that they can use to conduct automated reasoning of reference rules which can be used use to carry out automated reasoning. The aim of the semantic web is to make the WWW agents friendlier than their present scenario. The World Wide Web has become the largest database in the world. But finding relevant information is more and more difficult.

The Semantic Web is a project that intends to create a universal medium for information exchange by giving meaning (semantics) to the content of documents on the Web, in a manner understandable by machines. The idea is to make computers present formatted documents to the user and enable them to deal with the contents. Therefore, documents should be supplemented with additional mark-ups containing Meta information. For the realization of this idea, vocabularies for the Meta information, like RDFS (Resource Description Framework Schema) or ontologies, would be required.

1.1.2 Structure of Semantic Web

Currently, the World Wide Web consists of documents written in HTML. This makes the Web readable for humans, but since HTML has limited ability to classify the blocks of text apart from the roles they play, the Web in its current form is very hard to understand. The purpose of the Semantic Web is to add a layer of descriptive technologies to web pages so they become readable [4]. The Semantic Web is implemented in the layers of Web technologies and standards.

The layers are presented in Figure 1.1 [20] and described as follows:

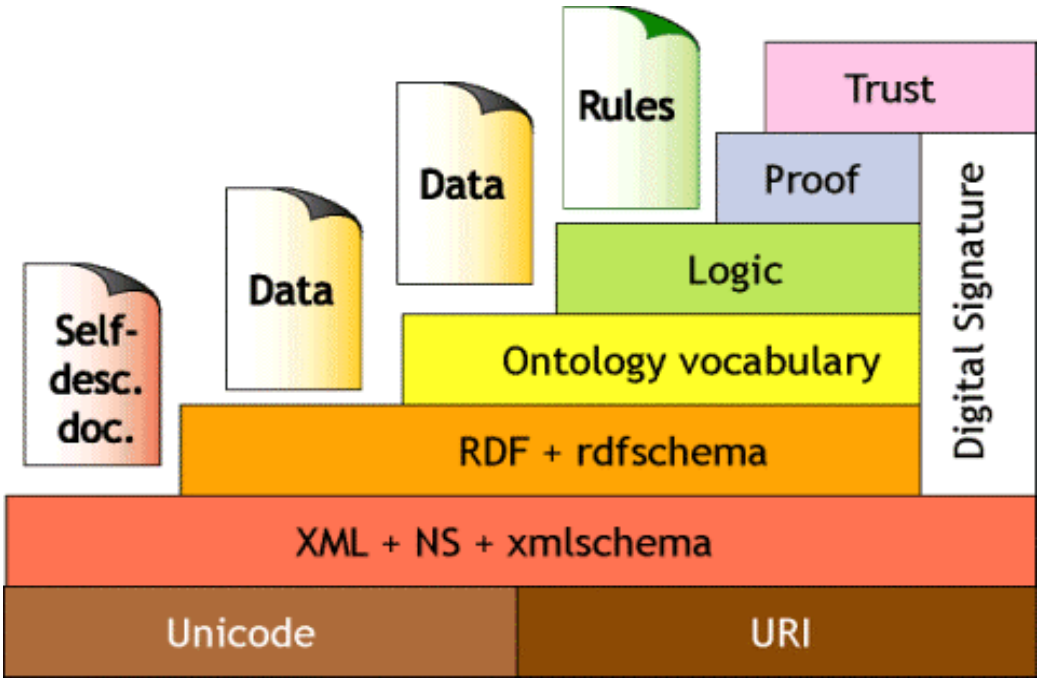

Figure 1.1: Berners-Lee's Semantic Web 'Birthday Cake'

- The **Unicode and Uniform Resource Identifier** (URI) layers make sure that international characters sets are used and provide means for identifying the objects in the Semantic Web. The most popular URI.s on the World Wide Web is Uniform Resource Locaters (urls).
- The **XML** layer with namespace and schema definitions make sure the Semantic Web definitions can integrate with the other XML based standards. XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents. XML Schema is a language for restricting the structure of XML documents [2].

- **RDF Schema** is a simple type modeling language for describing classes of resources and properties between them in the basic RDF model. It provides a simple reasoning framework for inferring types of resources. .
- The **ontology layer** supports the evolution of vocabularies as it can define relations between the different concepts [9]. In this layer knowledge is expressed as descriptive statements, stating some relationship exists between one thing and another. .
- A **digital signature** is an electronic signature that can be used to authenticate the identity of the sender of a message or the signer of a document. The Digital Signature layer ensures that the original content of the message or document is unaltered. .
- The top layers **Logic, Proof and Trust** are currently being researched and simple application demonstrations are being constructed. The Logic layer enables the writing of rules while the Proof layer executes the rules and evaluates together with the Trust layer mechanism for applications whether to trust the given proof or not. The Semantic Web comprises the standards and tools of XML, XML Schema, RDF, RDF Schema and OWL.

1.1.3 Technologies of current Semantic Web

Two important technologies for developing the Semantic Web are already in place: Extensible Markup Language (XML) and the Resource Description Framework (RDF). XML allows everyone to create their own tags. Hidden labels such as or those annotate Web pages or sections of text on a page. Scripts, or programs, can make use of these tags in sophisticated ways, but the scriptwriter has to know what the page writer uses each tag for. In short, XML allows users to add arbitrary structure to their documents but says nothing about what the structures mean [3].

Meaning is expressed through RDF, which encodes it in sets of triples; each triple is composed of a subject, predicate, and object that form elementary sentence. These triples can be written using XML tags. This structure turns out to be a natural way to describe the vast majority of the data processed by machines. Subject and object are each identified by a Universal Resource Identifier (URI), just as used in a link on a Web page. (URLs, Uniform Resource Locators, are the most common type of URI.)

The verbs are also identified by URIs, which enables anyone to define a new concept, a new verb, just by defining a URI for it somewhere on the Web.

Another approach to making Web applications a bit smarter is to write program code in a general-purpose language (e.g., C, Perl, Java, Lisp, Python, or XSLT) that keeps data from different places up to date [14]. Code for this purpose is often organized in a relational database application in the form of stored procedures; in XML applications, it can be affected using a transformational language like XSLT. These solutions are more cumbersome to implement, since they require special purpose code to be written for each linkage of data, but they have the advantage over a centralized database that they do not require all the publishers of the data to agree on and share a single data source. Furthermore, such approaches could provide a solution to the conference mapping problem by transforming data from one source to another. Just as in the query/presentation solution, this solution does not make the data any smarter; it just puts an informed infrastructure around the data, whose job it is to keep the various data sources consistent.

1.2 XML (Extensible Markup Language)

1.2.1 Overview of XML

XML is a general-purpose specification for creating custom markup languages. It is classified as an extensible language, because it allows the user to define the mark-up elements. XML's purpose is to aid information systems in sharing structured data, especially via the internet, to encode documents, and to serialize data. XML's set of tools helps developers in creating web pages but its usefulness goes well beyond that. XML, in combination with other standards, makes it possible to define the content of a document separately from its formatting, making it easy to reuse that content in other applications or for other presentation environments [21]. Most importantly, XML provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations without needing to pass through many layers of conversion.

- XML stands for Extensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data

- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

The Difference between XML and HTML:

- HTML is presentation language where as XML is not either a programming language or a presentation language. It is used to transfer data between applications and databases.
- XML is case-sensitive where as HTML is not case-sensitive.
- In XML we can define our own tags as it is not possible in HTML.
- In XML it is mandatory to close each and every tag where as in HTML it is not required.
- XML describes the data where as HTML only defines the data.
- XML was designed to transport and store data, with focus on what data is.HTML was designed to display data, with focus on how data looks.

1.2.2 XML Syntax and Semantics

XML documents have syntactic and semantic structures. The syntax (think spelling and punctuation) is made up of a minimum of rules:

- **XML documents always have one and only one root element**
  The structure of an XML document is a tree structure where there is one trunk and optionally many branches [19]. The single trunk represents the root element of the XML document.
- **Element names are case-sensitive**
  Element names, the basic vocabulary of XML documents, are case-sensitive. There are five elements: html, head, title, body, and p. since each element's name is case-sensitive, the element html does not equal HTML, nor does it equal HTmL or Html. The same is true for the other elements.
- **Elements are always closed**
  Each element is denoted by opening and closing brackets, the less than sign (<) and greater than sign (>), respectively. XML elements are rarely empty; they are usually used to provide some sort of meaning or context to some data, and consequently, XML elements usually surround data. For example, the title

of the document is denoted with the <title> and </title> elements and the only paragraph of the document is denoted with and elements. An opened element does not contain the initial forward slash but closing elements do.

- **Elements must be correctly nested**

  Consecutive XML elements may not be opened and then closed without closing the elements that were opened last first. Doing so is called improper nesting.

- **Elements' attributes must always be quoted**

  XML elements are often qualified using attributes. For example, an integer might be marked up as a length and the length element might be qualified to denote feet as the unit of measure. For example: <length unit='feet'>5</length>.The attribute is named unit, and its value is always quoted. It does not matter whether or not it is quoted with an apostrophe (') or a double quote (").

- **There are only three entities defined by default**

  Certain characters in XML documents have special significance, specifically, the less than (<), greater than (>), and ampersand (&) characters. The first two characters are used to delimit the existence of element names. The ampersand is used to delimit the display of special characters commonly known as entities.

- **Namespaces**

  The concept of a "namespace" is used to avoid clashes in XML vocabularies. Remember, the X in XML stands for extensible. This means you are allowed to create your own XML vocabulary. There is no centralized authority to dictate what all the valid vocabularies are and how they are used but with so many XML vocabularies there are bound to be similarities between them [2]. For example, it is quite likely that different vocabularies will want some sort of date element. Others will want a name or description element. In each of these vocabularies the values expected to be stored in date, name, or description elements may be different. How to tell the difference? Namespaces. Namespaces have a one-to-one relationship with a URI (Universal Resource Identifier), and namespace attributes defined with URIs can be inserted into XML elements to denote how an element is to be used.

Namespace attributes always begin with "xmlns". Namespace attributes are always end with some sort of identifier call the "local part". These two things are always delimited by a colon and finally equated with a URI.

For example, a conventional namespace defining the Dublin Core namespace is written as: xmlns:dc="http://purl.org/dc/elements/1.1/"

Where: A gentle introduction to XML markup

- xmlns denotes a namespace
- dc denotes the name of the namespace, the local part
- http://purl.org/dc/elements/1.1/ is a unique identifier (URI)

This whole namespace will become very useful when an XML document uses two or more XML vocabularies. For example, it is entirely possible (if not necessary) to have more than one vocabulary in RDF streams. There is one vocabulary used to describe the RDF structure, and there is another vocabulary used to describe the metadata. The example below is a case in point:

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="http://www.AcronymFinder.com/">
<dc:title>Acronym Finder</dc:title>
<dc:description>The Acronym Finder is a world wide web (WWW) searchable database of more than 169,000 abbreviations and acronyms about computers, technology, telecommunications, and military acronyms and abbreviations.</dc:description> <dc:subject>
<rdf:Bag>
<rdf:li>Astronomy</rdf:li>
<rdf:li>Literature</rdf:li>
<rdf:li>Mathematics</rdf:li>
<rdf:li>Music</rdf:li>
<rdf:li>Philosophy</rdf:li>
</rdf:Bag>
</dc:subject>
</rdf:Description>
</rdf:RDF>

Here you have two vocabularies going on. One is defined as rdf and associated with the URI http://www.w3.org/1999/02/22-rdf-syntax-ns#. The second one is defined as dc and associated with http://purl.org/dc/elements/1.1/. The namespace local parts are then associated with various elements a needed. URI are simply unique identifiers. They often take the shape of URLs, but need not point to anything accessible over the Web. They are just strings of text.

XML Semantics

The semantics of an XML document (think grammar) is an articulation of what XML elements can exist in a file, their relationship(s) to each other, and their meaning. Ironically, this is the really hard part about XML and has manifested itself as a multitude of XML "languages" such as: RSS, RDF, TEILite, XMLMARC, EAD, XSL, etc. In the following, valid, XML file there are a number of XML elements. It is these elements that give the data value and meaning:

<catalog>
<work type='Inspiring Book' date='2002'>
<title>You can win</title>
<author>Shiv Khera</author>
</work>
<work type='poem' date='1900'>
<title>Death</title>
<author>Rabindranath Tagore</author>
</work>
<work type='Story' date='2009'>
<title>Life at Thapar</title>
<author>Balaji Kotalwar</author>
</work>
</catalog>

1.2.3 Weakness of XML

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 DTD or schema that is it requires the XML file adhere to a pre-defined grammar. The XML schema provides a syntactic description of data and it 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). Only advantage of using XML is reusing the parser and document validation. XML provides many different possibilities to encode a domain of discourse, but this leads to difficulties when understanding of foreign documents is required [21]. RDF overcomes the semantics limitations to a great extent and can represent semantics more efficiently than XML.

1.3 RDF (Resource Description Framework)

1.3.1 Introduction to RDF

The World Wide Web was originally built for human consumption, and although everything on 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 information the Web contains; it is not possible to manage it manually. The solution is to use metadata to describe the data contained on the Web. Metadata is "data about data" (for example, a library catalog is metadata, since it describes publications) or specifically in the context of this specification "data describing Web resources" [5]. The distinction between "data" and "metadata" is not an absolute one; it is a distinction created primarily by a particular application, and many times the same resource will be interpreted in both ways simultaneously.

RDF and its schema language Resource Description Framework Schema (RDFS) were proposed by the World Wide Web Consortium to overcome the semantic limitations of XML. Resource Description Framework (RDF) is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. RDF emphasizes facilities to enable automated processing of Web resources.

RDF can be used in a variety of application 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 and exchange,
- In content rating, in describing collections of pages that represent a single logical "document",
- For describing intellectual property rights of Web pages,
- For expressing the privacy preferences of a user as well as the privacy policies of a Web site.
- RDF with digital signatures will be key to build the "Web of Trust" for electronic commerce, collaboration, and other applications.

One of the goals of RDF is to make it possible to specify semantics for data based on XML in a standardized, interoperable manner. RDF is a model of metadata and only addresses by reference many of the encoding issues that transportation and file storage require (such as internationalization, character sets, etc.). For these issues, RDF relies on the support of XML. It is also important to understand that this XML syntax is one possible syntax for RDF and alternate ways to represent the same RDF data model may emerge. The broad goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines (a priori) the semantics of any application domain.

RDF has a class system much like many object-oriented programming and modeling systems. A collection of classes is called a schema. Classes are organized in a hierarchy, and offer extensibility through subclass refinement. This way, in order to create a schema slightly different from an existing one, it is not necessary to "reinvent the wheel" but one can just provide incremental modifications to the base schema. Through the share ability of schemas, RDF will support the reusability of metadata definitions [7]. Due to RDF's incremental extensibility, agents processing metadata will be able to trace the origins of schemata they are unfamiliar with back to known schemata and perform meaningful actions on metadata they weren't originally designed to process. The share ability and extensibility of RDF also allows metadata authors to use multiple inheritance to "mix" definitions, to provide multiple views to their data, leveraging work done by others. In addition, it is possible to create RDF instance data based on multiple schemata from multiple sources. Schemas may themselves be written in RDF; RDF Schema describes one set of properties and classes for describing RDF schemas.

As a result of many communities coming together and agreeing on basic principles of metadata representation and transport, RDF has drawn influence from several different sources. The main influences have come from the Web standardization community itself in the form of HTML metadata and PICS, the library community, the structured document community in the form of SGML and more importantly XML, and also the knowledge representation (KR) community. There are also other areas of technology that contributed to the RDF design; these include object oriented programming and modeling languages, as well as databases. While RDF draws from the KR community, that RDF does not specify a mechanism for reasoning.

1.3.2 Basic RDF Model

RDF is model and XML syntax for representing information in a way that allows programs to understand the intended meaning. RDF represents information about resources on the Web, i.e., metadata about resources. A resource in RDF is mapped to a Uniform Resource Identifier (URI) and is described in terms of its properties. The purpose of RDF is to make statements about resources, which can be viewed as labeled edge (property) between two nodes (object and property value). The foundation of RDF is a model for representing named properties and property values. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs [6]. RDF properties also represent relationships between resources and an RDF model can therefore resemble an entity relationship diagram. In object-oriented design terminology, resources correspond to objects and properties correspond to instance variables. The RDF data model is a syntax-neutral way of representing RDF expressions. Two RDF expressions are equivalent if and only if their data model representations are the same. The basic data model consists of three object types:

- **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" for example. A resource may be a part of a Web page; e.g. a specific HTML or XML element within the document source. A resource may also be a whole collection of pages; e.g. an entire Web site. A resource may also be an object that is not directly accessible via the Web; e.g. a printed book. Resources are always named by

URIs plus optional anchor ids. Anything can have a URI. The extensibility of URIs allows the introduction of identifiers for any entity imaginable.

- **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
- **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's built on the concept of a statement, a triple of the form {predicate, subject, object}. The interpretation of a triple is that <subject> has a property <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 for the resource. The object of a statement (i.e., the property value) can be another resource or it can be a literal; i.e., a resource (specified by a URI) or a simple string or other primitive data type defined by XML. In RDF terms, a *literal* may have content that is XML markup but is not further evaluated by the RDF processor.

Examples

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: Balaji Kotalwar is the creator of the resource: http://www.youtube.com/watch?v=ppigyg7jva0.

{Creator, http://www.youtube.com/watch?v=ppigyg7jva0, Balaji Kotalwar}

This sentence has the following parts:

Subject (Resource)	http://www.youtube.com/watch?v=ppigyg7jva0.
Predicate (property)	Creator
Object (literal)	"Balaji Kotalwar"

Table 1.1: RDF Format

The diagram of this RDF statement pictorially using directed labeled graphs (also called "nodes and arcs diagrams") is shown in Figure 1.2. In these diagrams, the nodes (drawn as ovals) represent resources and arcs represent named properties.

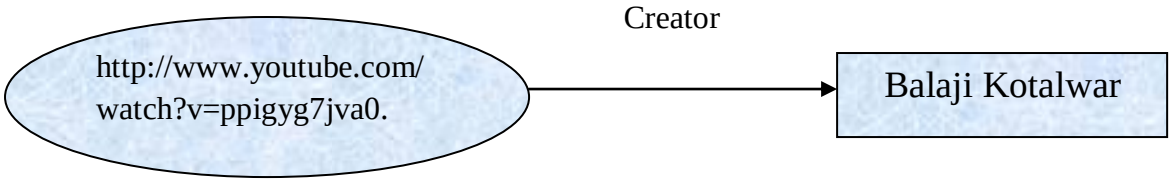

Figure 1.2: Simple node and arc diagram

Nodes that represent string literals will be drawn as rectangles. In prose, such a sentence would be:

The individual whose name is Balaji Kotalwar, email balaji.kotalwar08@gmail.com is the creator of http://www.youtube.com/watch?v=ppigyg7jva0.

The intention of this sentence is to make the value of the Creator property a structured entity. In RDF such an entity is represented as another resource. The sentence above does not give a name to that resource. It is anonymous, so in the diagram below we represent it with an empty oval:

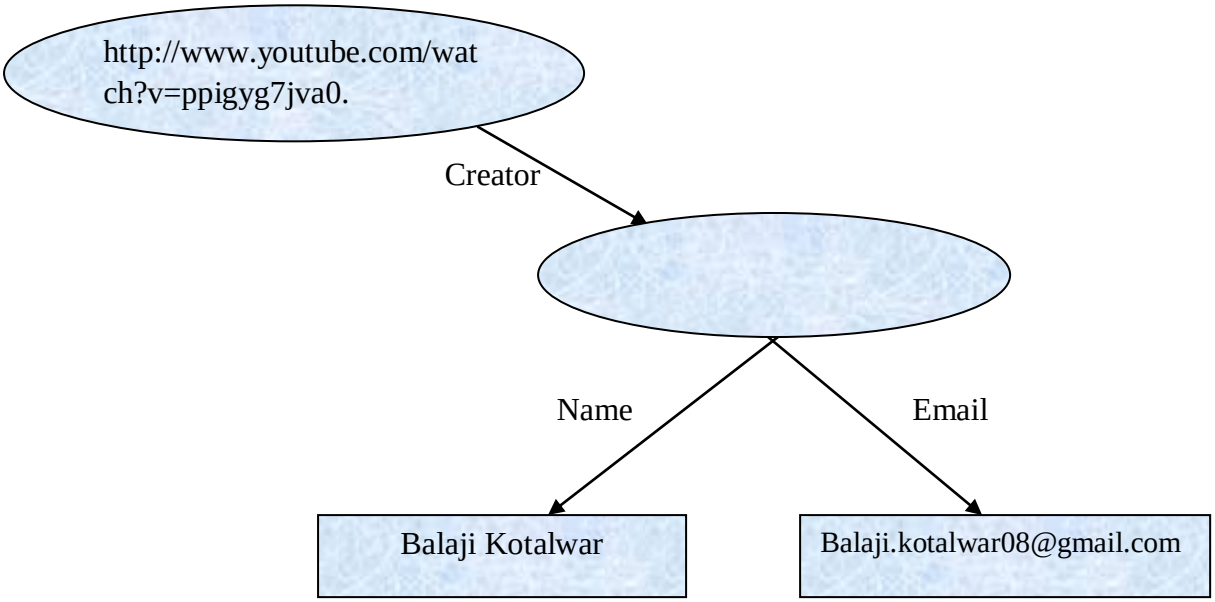

Figure 1.3: Property with structured value

1.3.3 RDF General Format

<? xml version="1.0"?>

<*class* **rdf:ID**="*Resource*"

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

xmlns="*uri*">

<*property*>*value*</*property*>

<*property*>*value*</*property*>

...

</*Class*>

Element description

- rdf:ID: RDF provides an ID attribute for identifying the resources being described. The value of rdf:ID is a "relative URI". The "complete URI" is obtained by concatenating the URL of the XML document with "#" and then the value of rdf:ID.
- Resource: Identifies the resource being described.
- xmlns:rdf: The ID attribute is in the RDF namespace.
- Class: Identifies the type (class) of the resource being described.
- Property: These are properties, or attributes, of the type (class).
- Value: Values of the properties.

Some other elements

rdf: about: Instead of identifying a resource with a relative URI (which then requires a base URI to be attached before relative URI), we can give the complete identity of a resource. However, we use rdf: about, rather than rdf:ID.

rdf:ID versus rdf: about **:**

- To introduce a resource, and provide an initial set of information about a resource use rdf:ID.
- To extend the information about a resource use rdf: about .

Data Model-XML Syntax

- Well-formed XML
- Validation through RDF schemas

Here is an example of an XML document that specifies data about China's Yangtze river:

```
<?xml version="1.0"?>
<River id="Yangtze" xmlns="http://www.geodesy.org/river">
<length>6300 kilometers</length>
<startingLocation>western China’s Qinghai-Tibet Plateau</startingLocation>
<endingLocation>East China Sea</endingLocation>
</River>
```

Data about the Yangtze River. It has a length of 6300 kilometers. Its startingLocation is western China’s Qinghai-Tibet Plateau. Its endingLocation is the East China Sea. Modify the previous XML document so that it is also a valid RDF document:

```
<?xml version="1.0"?>
<River rdf:ID="Yangtze"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://www.geodesy.org/river#">
<length>6300 kilometers</length>
<startingLocation>western China’s Qinghai-Tibet Plateau</startingLocation>
<endingLocation>East China Sea</endingLocation>
</River>
```

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. *Bag*s 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 the order of the values is significant. *Sequence* might be used, for example, to preserve an alphabetical ordering of values. Duplicate values are permitted.
- **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 that it can choose any one of the items in the list as appropriate.

1.3.4 Advantages of using RDF

- The RDF format, if widely used, will help to make XML more interoperable:
  - Tools can instantly characterize the structure, "this element is a type (class), and here are its properties.
  - RDF promotes the use of standardized vocabularies, standardized types (classes) and standardized properties.
- The RDF format gives a structured approach to design XML documents. The RDF format is a regular, recurring pattern.
- It enables too quick identification of weaknesses and inconsistencies of non-RDF-compliant XML designs. It helps to better understand the data.
- RDFS gives the benefits of both worlds:
  - Standard XML editors and validators to create, edit, and validate XML file.
  - Use the RDF tools to apply inferencing to the data.
- It positions the data for the Semantic Web.

1.3.5 RDFS (RDF Schema)

RDF asserts statements about resources in terms of properties, the description of these properties is provided by the RDF schema specification, i.e. RDF schemas add semantics to RDF. They provide a type system for resources and define relationships between resources and can be encoded using RDF resources, such as Class, SubclassOf, Property, etc. which are defined in the namespace .rdfs.. Unlike XML schemas, they do not impose syntactic constraints on the RDF document but provide a mechanism to describe metadata about resources. Every entity in RDF is a resource and a class represents a group of resources [6]. Every resource in RDF is an instance of at least one class. RDF schemas use a core set of classes and properties to describe metadata about web resources, *i.e.*, there exists a subset of RDF resources using that all other RDF schema constructs can be defined. The core classes in RDF schema are rdfs:class, rdfs:resource, rdfs:property and rdfs:constraint resource. rdfs:class represents the set of entities used to describe concepts in a schema, rdfs:resource represents the set of all resources, rdfs:property represents the set of all RDF properties and rdfs:constraintresource represents the set of all constraints applicable to RDF resources. Core properties include properties of the class rdfs:property namely,

rdf:type, rdfs:subclassof, rdfs:range and rdfs:domain. The RDF model assigns a dual role to these core classes and properties, which is used to define other schema constructs and can be used in the specification of an RDF schema itself that causes self-referentiality in the RDF model [7].

Some important feature of RDFS:

- RDF describes resources with classes**,** properties, and values. In addition, RDF also needs a way to define application-specific classes and properties.
- Application-specific classes and properties must be defined using extensions to RDF. One such extension is RDF Schema RDFS.
- RDF Schema does not provide actual application-specific classes and properties. Instead RDF Schema provides the framework to describe application specific classes and properties.
- Classes in RDF Schema are much like classes in object oriented programming languages. This allows resources to be defined as instances of classes, and subclasses of classes.

1.4 Ontology

1.4.1 Defining Ontology

"Ontology is a formal, explicit specification of a shared conceptualization". The idea is to devise new ways of constructing knowledge-based systems, such that these systems need not to be constructed from scratch, but the changes can be done only by assembling reusable component. Ontology consists of entities, attributes, and interrelationships between entities, domain vocabulary and factual knowledge all connected in a semantic manner. Ontology is often viewed as the key means through which the Semantic Web vision can be realized. Ontology agrees upon unambiguous vocabulary to be used in reusable and sharable form. Originally introduced by Aristotle, ontology is a formal model about how we perceive a domain of interest and provide a precise, logical account of the intended meaning of terms, data structures and other elements modeling the real world. [11] Ontology provides a common understanding of a particular domain. Ontology provides interoperability among software systems and improves the design and the quality of software systems.

When we talk about ontology there are two or more agents communicating together for establishing a common terminology. Agents are nothing but applications which

help in establishing friendly messages between two different types of terminologies. An efficient communication between two or more agents, in order to exchange knowledge, needs a “common understanding” of the concepts [9]. This means that agents does not need to agree about the terms used for the concepts described, but need to have an agreement about the meaning of these terms and the relationships among those terms. One major problem that might emerge is that two agents use the same term but with a different semantics. The two agents would believe they talk about the same thing, but they do not. Similar problems might emerge when two agents use different terms that have the same meaning. The agents would not recognize that they actually talk about the same thing. The aim of ontologies is to improve the efficiency and consistency in communication between agents. They do this by defining hierarchical organized vocabularies, which describe the concepts of the domain of interest, the relationships among these concepts and their properties.

The concepts are sometimes also called classes or categories used to model the world [12]. These categories and their relationships are clearly defined in the domain. Properties or slots, describe the “various features and attributes of the concept”. Typical relationships between concepts are is a part of relationships. The first one can be compared with generalisation and inheritance from object-oriented programming, the later is sometimes referred to as the part whole relationship. Instantiation is another important relationship between ontologies [11] which describe the relation between a concept and the instances of that concept. Instances are “entities of the world”.

The Need for Ontologies on the Web

Ontology is a description of a domain in terms of entities and relationships that exist between those entities. Ontologies provide a precise understanding of the domain, i.e. they provide an unambiguous and consistent interpretation of the domain in all scenarios. They enable web to share knowledge and provide reasoning support, verification of formalisms and rules. Checking for constraint violations in knowledge shared through the ontology can be automated.

1.4.2 Advantages of Developing Ontology

- **Sharing common understanding of the structure of information among people or software agents**: It is one of the most common goals in developing Ontologies. For example several different web sites contain medical information or provide e-commerce services. These web sites share and publish the same underlying ontology of the terms they use, and then computer agents can extract and aggregate information from different sites[9]. The agents can use this aggregated information to answer user queries or as input data to other applications.
- **Enabling reuse of domain knowledge:** It is one of the driving forces behind recent surge in ontology research. For example, models for many different areas need to represent the opinion of time. This representation includes the planning of time intervals, points in time, relative measures of time, etc. If one group of researchers develops such ontology in detail, others can simply reuse it for their area. To build a large ontology need to integrate several existing Ontologies describing portions of the large domain [9]. General ontology can be reused and extend it to describe desired ontology.
- **Separating the domain knowledge from the operational knowledge:** It describes a task of configuring a product from its components according to a required specification and implements.
- **Analyzing domain knowledge:** Analyzing is possible once a declarative specification of the terms is available. Formal analysis of terms is valuable when attempting to reuse existing Ontologies and extending them for solving problem purposes.

1.4.3 Beneficial Applications of Ontology

It is stated that Ontologies are used in e-commerce to enable machine based communication between buyers and sellers, vertical integration of markets and description reuse between different marketplaces. Search engines also use Ontologies to find pages with words that are syntactically different but semantically similar. In particular, the following area will benefit from the use of Ontologies: Semantic Web.

- **Semantic Web**

  The Semantic Web aims at tackling the growing problems of traversing the expanding web space, where currently most web resources can only be found by syntactical matches. The Semantic Web relies heavily on formal Ontologies that structure underlying data for the purpose of comprehensive and transportable machine understanding. They properly define the meaning of data and metadata. In general, one may consider the Semantic Web more as a vision than a concrete application.

- **Knowledge Management**

  Knowledge management deals with acquiring, maintaining and accessing knowledge of an organization. The technologies of the Semantic Web build the foundation to move from a document oriented view of knowledge management to a knowledge pieces oriented view where knowledge pieces are connected in a flexible way [12]. Intelligent push service, the integration of knowledge management and business process as well as concepts and methods for supporting the vision of ubiquitous knowledge are urgently needed. Ontologies are the key means to achieve this functionality. They are used to annotate unstructured information with semantic information, to integrate information and to generate user specific views that make knowledge access easier.

- **Interoperability**

  An important application area for ontology is the integration of existing systems. In order to enable machines to understand each other we need to explicate the context of each system in a formal way. Ontologies are then used as inter-lingua for providing interoperability since they serve as a common format for data interchange [16].

- **Information Retrieval**

  Common information retrieval techniques either rely on a specific encoding of available information or simple full-text analysis. Both approaches suffer from problems like the query entered by the user may not be completely consistent with the vocabulary of the documents and the recall of a query will be reduced since related information with slightly different encoding is not matched. Using ontology to explicate the vocabulary may overcome some of the

problems. When used for the description of available documents as well as for query formulation, ontology serves as a common basis for matching queries against potential results on a semantic level. In some cases, the ontology can also be directly used as a user interface to navigate through available document. Information retrieval benefits from the use of ontologies, because ontologies help to decouple description and query vocabulary and increase retrieval performance [12].

- **Service Retrieval**

  The ability to rapidly locate useful online service (e.g. software applications, software components, process models, or service organizations), as opposed to simple useful documents, is becoming increasingly critical in many domains.

CHAPTER 2
LITERATURE REVIEW

2.1 Swoogle

Swoogle is a crawler-based indexing and retrieval system for the Semantic Web. It extracts metadata for each discovered document, and computes relations between documents. Discovered documents are also indexed by an information retrieval system which can use either character N-Gram or URI refs as keywords to find relevant documents and to compute the similarity among a set of documents. One of the interesting properties computed is ontology rank [8],a measure of the importance of a Semantic Web document.

The Swoogle Semantic Web search system discovers, analyzes and indexes semantic web documents on the web. Just as modern search engines and their services have given people access to much of the world's knowledge and data [13]. Swoogle aims to support semantic web developers, software agents and programmers in finding RDF data and schema level knowledge on the web. Swoogle act as component in number of information integration task and are working on several others.

2.1.1 Semantic web documents (SWD)

Semantic web languages based on RDF (e.g. RDFS, DAML, and OWL) allow one to make statements that define general terms (classes and properties), extend the definition of terms, create individuals, and to make assertions about terms and individuals already defined or created. Semantic Web Document (SWD) is a document in a semantic web language that is online and accessible to web users and software agents. Similar to a document in IR, a SWD is an atomic information exchange object in the Semantic Web [8].

Current practice favors the use of two kinds of documents which will refer to as semantic web ontologies (SWOs) and semantic web databases (SWDBs). These correspond to what are called T-Boxes and A-Boxes in the description logic literature [23]. Since a document may consist of both T-Boxes and A-Boxes, they adopt a threshold based measure. Consider a document to be a SWO when a significant proportion of the statements it makes define new terms (e.g., new classes and

properties) or extend the definitions of terms defined in other SWDs by adding new properties or constraints. A document is considered as a SWDB when it does not define or extend a significant number of terms. A SWDB can introduce individuals and make assertions about them or make assertions about individuals defined in other SWDs. For example, the SWD http://xmlns.com/foaf/0.1/index.rdf is considered a SWO since its 466 statements (i.e. triples) define 12 classes and 51 properties but introduce no individuals. The SWD http://umbc.edu/finin/foaf.rdf is considered to be a SWDB since it defines or extends no terms but defines three individuals and makes statements about them. Between these two extremes, some SWDs are intended to be both an ontology that defines a set of terms to be used by others, as well as a useful database of information about a set of individuals. Even a document that is intended as ontology might define individuals as part of the ontology. Similarly, a document that is intended as defining a set of individuals might introduce some new terms in order to make it easier to describe the individuals,
(e.g. http://www.daml.ri.cmu.edu/ont/USCity.daml,
http://reliant.teknowledge.com/DAML/Government.owl).

2.1.2 SWD Metadata

SWD metadata is collected to make SWD search more efficient and effective. It is derived from the content of SWDs as well as the relation among SWDs. Swoogle identifies three categories of metadata [15]:

- Basic metadata, which considers the syntactic and semantic features of a SWD,
- Relations, which consider the explicit semantics between individual SWDs,
- Analytical results such as SWO/SWDB classification, and SWD ranking.

The first two categories are objective information about SWDs, the third category is subjective.
The basic metadata about a SWD falls into three categories**: language feature, RDF statistics and ontology annotation.**

- **Language feature** refers to the properties describing the syntactic or semantic features of a SWD. Swoogle capture the features Encoding which shows the syntactic encoding of a SWD, Language which shows the Semantic Web

language used by a SWD and OWL Species that shows the language species of a SWD written in OWL.

- **RDF statistics** refers to the properties summarizing node distribution of the RDF graph of a SWD. Which focus on how SWDs define new classes, properties and individuals? In an RDF graph, a node is recognized as a class if & only if it is not an anonymous node and it is an instance of rdfs:Class. Similarly, a node is a property if & if it is not an anonymous node and it is an instance of rdf:Property. An individual is a node which is an instance of any user defined class.
- **Ontology annotation** refers to the properties that describe a SWD as ontology. In practice, when a SWD has an instance of OWL:Ontology, Swoogle records its properties as: label, comment, and versionInfo.

2.1.3 Relations among SWDs

Looking at the entire semantic web, it is hard to capture and analyze relations at the RDF node level. Therefore, Swoogle focuses on SWD level relations which generalize RDF node level relations [8]. Swoogle captures the following SWD level relations:

- **TM**: captures term reference relations between two SWDs; SWD is using terms defined by some other SWDs. By retrieving and processing the reference SWD, the type of term (class, property or individual) can be determined.
- **IM** shows that ontology imports ontology. The URLs of referenced ontologies are collected by recording the objects in triples whose predicate is owl: imports.
- **EX** shows that ontology extends another. Such a relation may be produced by many properties. For example, if ontology A defines class AC which has the “rdfs:subClassOf" relation with class BC defined in ontology B, Swoogle will record the EX relation from A to B.
- **PV** shows that ontology is a prior version of another.
- **CPV** shows that ontology is a prior version of and is compatible with another.
- **IPV** shows that ontology is a prior version of but is incompatible with another.

2.1.4. Ranking SWDS

PageRank, introduced by Google [24], evaluates the relative importance of web documents. Given a document A, A's PageRank is computed by equation (1):

$PR(A) = PR_{direct}(A) + PR_{link}(A)$

$PR_{direct}(A) = (1 - d)$ (1)

$PR_{link}(A) = d\left(\frac{PR(T1)}{C(T1)} + \cdots + \frac{PR(Tn)}{C(Tn)}\right)$

where $T_1 \ldots T_n$ are web documents that link to A. $C(T_i)$ is the total outlinks of T_i and d is a damping factor, which is typically set to 0.85. The intuition of PageRank is to measure the probability that a random surfer will visit a page. Equation captures the probability that a user will arrive at a given page either by directly addressing it (via $PR_{direct}(A)$), or by following one of the links pointing to it (via $PR_{link}(A)$).

Unfortunately, this random surfing model is not appropriate for the Semantic Web. The semantics of links lead to a non-uniform probability of following a particular outgoing link. Therefore, Swoogle uses a rational random surfing model which accounts for the various types of links that can exist between SWDs. Given SWDs A and B, Swoogle classifies inter-SWD links into four categories: (i) imports (A,B), A imports all content of B (ii) uses-term(A,B), A uses some of terms defined by B without importing B (iii) extends(A,B), A extends the definitions of terms defined by B and (iv) asserts(A,B), A makes assertions about the individuals defined by B.

These relations should be treated differently. For instance, when a surfer observes imports (A,B) while visiting A, it is natural for it to follow this link because B is semantically part of A. Similarly, the surfer may follow the extends (A,B) relation because it can understand the defined term completely only when it browses both A and B. Therefore, assign different weights to the four categories of inter-SWD relations. Since the generalized RDF node level relations to SWD level relations, you have to also count the number of references. The more terms in B referenced by A, the more likely a surfer will follow the link from A to B. Based on the above considerations, given SWD *a*, Swoogle computes its raw rank using equation (2). [8]

$rawPR(a) = (1 - d) + d \sum_{x \epsilon L(a)} rawPR(x) \frac{f(x,a)}{f(x)}$ (2)

$f(x, a) = = \sum_{l \epsilon Links\ (x,a)} weight(l)$

$f(x) = \sum_{a \epsilon T(x)} f(x, a)$

Where L (a) is the set of SWDs that link to *a*, and T(x) is the set of SWDs that *x* links to. Then Swoogle computes the rank for SWDBs and SWOs using equation (3) and (4) respectively.

PR_{SWDB} (a) = rawPR (a) (3)

PR_{SWO} (a) = $\sum_{x \epsilon TC(a)}$ rawPR(x) (4)

Where TC (a) is the transitive closure of SWOs imported by *a*.

2.1.5 Architecture of Swoogle

Figure 2.1: Swoogle architecture.

Figure shows Swoogle architecture. [8] There are four major components such as Discovery, Digest, Analysis, and Service. Set of integrated web crawlers are used in the discovery component. This **discovery component** automatically discovers and revisits SWDs. The **digest** component computes metadata for SWDs and semantic web terms (SWTs) as well as identifies relations among them, e.g. "an SWD instantiates an SWT class", and "an SWT class is the domain of an SWT property". The **analysis** component its name itself indicates that it is used to analysis the SWDs.

This component uses cached SWDs and their metadata to derive analytical reports, such as classifying ontologies among SWDs and ranking SWDs by their importance [14]. The **service** component supports both human and software agents through conventional web interfaces and SOAP-based web service APIs. Three key services are:

- Swoogle search service that searches for SWDs by constraints on their URLs, the sites which host them, and the classes/properties used or defined by them.
- An ontology dictionary service that searches for SWTs and their relationships with other SWTs and SWDs.
- Swoogle statists are used for the ranking and revisiting rate of the SWDs.

2.1.6 Searching the Semantic Web in Swoogle

Search engines for both the conventional Web and the Semantic Web involve the same set of high level tasks: discovering or revisiting online documents, processing user's queries, and ordering search results. This section will describe how system, Swoogle, has addressed the tasks in particular. When considering these tasks, two facts should be kept in mind.[22] First, The processing Semantic Web documents which are distinct from regular HTML documents and there are far fewer of them. Second, on the Semantic Web, search clients are more likely to be software agents than people.

Discovering and revisiting documents

Conventional search engines scan all possible IP addresses and/or employ crawlers to discover new web documents. A typical crawler starts from a set of seed URLs, visits documents, and traverses the Web by following the hyperlinks found in visited documents. The fact that the Web forms a well connected graph and the ability for people to manually submit new URLs make this an effective process.

A Semantic Web crawler must deal with several problems. SWDs are needles in the haystack of the Web, so an exhaustive crawl of the Web is not an efficient approach. Moreover, the graph of SWDs is not (yet) as dense and well-connected as the graph formed by conventional web pages. Finally, many of the URLs found in a SWD point to documents which are not SWDs. Following these can be computationally expensive, so heuristics to limit and prune candidate links are beneficial.

A Semantic Web crawler can also use conventional search engines to discover initial seed SWDs from which to crawl. Swoogle, for example, uses Google to find likely initial candidate documents based on their file names, e.g. searching for documents whose file names end with .rdf or .owl. It also actively prunes links that are unlikely to contain semantic markups.

For the most part, how often to revisit documents to monitor for changes is the same for both the conventional Web and the Semantic Web. However, modifying a SWD can have far reaching and non-local effects if any class or property definitions used by other documents are changed. The Semantic Web can be viewed at different levels of granularity, from the universal graph comprising all RDF data on the Web to individuals' triples and their constituent resources and literals. The nature and amount of reasoning that is done when documents are analyzed and indexed, updating a SWD can trigger significant work for a Semantic Web search engine.

Query Processing

The core task of a search engine is processing queries against the data it has indexed. This can be broken down into three issues: what should be returned as query results, over what data the queries should be run, and what constraints can be used in a query. Semantic Web data can be aggregated at several levels of granularity, ranging from the universal graph of all RDF data on the Web to a single RDF triple and the term URIs it comprises. Since search engines usually return the references (or location) of search results, there are three types of output at different levels.

- **The term URI**. At the lowest level is a URI representing a single RDF term a class, property or instance. For Semantic Web content, these terms are analogous to words in natural language. Knowing the appropriate terms used to describe a domain is an essential requirement for constructing Semantic Web queries.
- **An RDF Graph**. In order to access knowledge in the Semantic Web, users need to fetch an arbitrary sub-graph from a target RDF graph. The sub-graph might correspond to a named graph, a collection of triples with a common subject, or an RDF molecule [25].
- **The URL of a Semantic Web Document**. This corresponds to the result returned by a conventional Web search engine a reference to the physical

document that serializes an RDF graph. This level of granularity helps improve efficiency in filtering out huge amount of irrelevant knowledge. Some documents, such as those representing consensus ontologies, are indented to be shared and reused. Discovering them is essential to the workings of the Semantic Web community.

In order to search the RDF graph, all triples need to be stored. This is essentially the basic features of an RDF database (i.e. triple store). The first and third output requirements are similar to dictionary lookup and web search respectively; and the prohibitively space cost for storing all triples can be avoid by utilizing compact metadata model.

As for a term, the following metadata needs to be considered: the namespace and local name extracted from the term's URI, the literal description of the term, the type of a term, in/out degree of the corresponding RDF node, and the binary relation among terms, namespace and SWDs [15].

For a semantic web document, metadata about itself (such as document URL and last-modified time) and its content (such as terms being defined or populated, and ontology documents being imported) should be considered. One interesting case is searching the provenance of RDF graph that searches for SWDs that imply the given RDF graph in whole or in part. It stores every triple in all indexed SWDs and has the same scalability issue as RDF database.

The structured metadata provides greater freedom in defining matching constraints: users can specify 2D constraints in (property, value) format. Note that the possible values of `property' are predetermined by the schema of metadata, but the possible values of `value' are undetermined since metadata is accumulated continuously.

The following figure shows the current window of swoogle search engine. It is titled with Swoogle followed by Semantic web search 2007,and its URL link is http://swoogle.umbc.edu/. It allows to search ontology,documents,terms over 10,000 ontologies.

Figure 2.2: Swoogle Search Engine Window

2.1.7 Uses of Swoogle

- **Supporting Semantic Web developers:** The first and most common use of Swoogle is to support developers and researchers. Swoogle helps developers to finding ontologies and individual RDF terms (i.e. classes and properties) for study and reuse, in finding data to illustrate how these ontologies have been used and finding RDF data of interest.

- **Helping scientists for publishing and finding data:** Sharing data is extremely important in natural science and experimental engineering disciplines [15]. The semantic web offers new way for scientists and engineers to publish and find data both and associated ontologies.

- **Discovering ontology mappings:** Swoogle can also be used to support ontology mapping. it can be used to assemble partial ontology mapping from multiple sources by collecting assertions specifying mapping expressed using OWL primitives or terms from special ontology Mapping ontologies. It can be used to compile instance data for terms in different ontologies that can be used to induce mapping relationships.

- **Learning trust relationships**: Swoogle can be used to provide evidence for trust relationship based on who is using and what ontologies and what data. When integrated with other metadata and semantic web data interesting relationship can perhaps derived.

- **Discovering facts**: Swoogle can be used to collect data matching certain patterns e.g. find all RDF triples asserting facts about foaf:person instance with foaf:lastName equal to "Balaji". Now we will take look on the structure of semantic web engine Swoogle.

2.2 SESAME

Sesame is an open source Java framework for storage and querying of RDF data. The framework 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 [18]. Aduna continued the development in cooperation with NLnet Foundation, developers from Ontotext, and a number of volunteer developers who contribute ideas, bug reports and fixes.

- Sesame is completely targeted at Java 5. All APIs use Java 5 features such as typed collections and iterates.
- 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.
- Sesame is similar to Jena
    - Supports triple storage
    - Supports reasoning
    - Supports Web services.

2.2.1 Important Features of Sesame

- Portability -It is written completely in Java.
- Repository independence-Provides RAL.
- Extensibility -Other functional modules can be created and be plugged in it.

- Flexible communication by using protocol handlers. The architecture separates the communication details from the actual functionality through the use of protocol handlers.

2.2.2 Architecture of Sesame

An overview of Sesame's architecture is shown in following [17] Figure.

Fig 2.3: Sesame Architecture

REPOSITORY:

To store RDF data persistently, Sesame needs a scalable repository. Any kind of repositories that can store RDF is used, such as:

- DBMSs: Any kind of database can be used.
- Existing RDF stores: Sesame can use an RDF store if a RAL that can communicate with it is written.

- RDF files: These can also be used; it becomes more practical when combined with a RAL that caches all the data in memory.
- RDF network services: When performance is not an issue, any network service that offers basic functionality for storing, retrieving and deleting RDF data can be used.

REPOSITRY ABSTRACTION LEVEL (RAL):

This is an interface that offers RDF-specific methods to its clients and translates these methods to calls to its specific DBMS. It contains all DBMS specific code, so as to keep Sesame DBMS independent, making it possible to implement Sesame on top of a wide variety of repositories without changing any other component of Sesame.

Stacking Abstraction Layers

It is possible to put one RAL on top of the other, however, Sesame's functional modules are only able to communicate with the RAL at the top of the stack Calls to the RAL from the modules are propagated down the stack, until one of the RALs finally handles the actual request and the result is then propagated back up again [17].

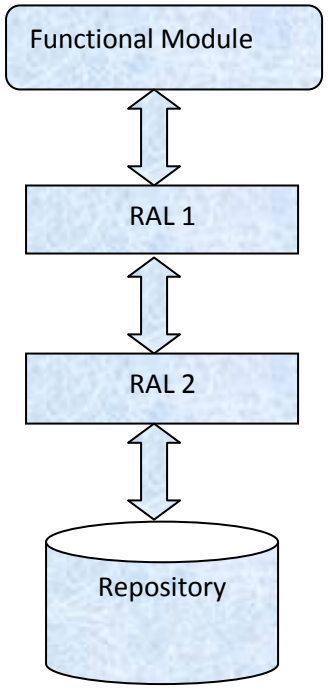

Fig 2.4: Stacking Abstraction Layers

FUNCTIONAL MODULES:

They are the clients of the RAL and currently, there are three such modules:

- **Admin Module**

  This module allows inserting or deleting RDF data in repository & retrieves its information from an RDF(S) source, and parses it using an RDF parser. Checks each (S, P, O) statement for consistency and infers implied information if necessary for instance. Example If P equals ‘type’, it infers that O must be a class. If P equals ‘subClassOf’, it infers that S and O must be classes. If P equals ‘subPropertyOf’, then it infers that both S and O must be properties .If P equals domain or range then it infers that S must be a property and O must be a class.

- **Query Module**

  This module evaluates RQL queries posed by the user. It is independent of the underlying repository [18]. It cannot use optimizations and query evaluations offered by specific DBMSs. RQL queries are translated into a set of calls to the RAL. Example when a query contains a join operation over two subqueries, each of the subqueries is evaluated, and the join operation is then executed by the query engine on the results.

- **Export Module**

  This module allows for the extraction of the complete schema and/or data from a model in RDF format. It supplies the basis for using Sesame with other RDF tools.

CHAPTER 3

PROBLEM STATEMENT

3.1 Problem Definition

During the literature review it was found that, The Swoogle Semantic Web search system discovers, analyzes and indexes semantic web documents on the web. Just as modern search engines and their services have given people access to much of the world's knowledge and data. Swoogle aims to support semantic web developers, software agents and programmers in finding RDF data and schema-level knowledge on the web. But the number of SWDs that are available are very less, So to increase the number of SWD and its ontologies we required considerable number of repositories which can be made by users. There are numerous amount of information available on net and to use it in a user friendly manner a common terminology is needed amongst all the applications. To solve this purpose repositories are to be developed. It has been widely accepted in the computer research community that repositories will serve the basis for the fulfillment of vision of Semantic Web search engine Swoogle. The name Swoogle is composed of Semantic Web and the Google so to bring Swoogle to the level of Google need to increase repositories. Moreover more number of SWDs will bring scalability and the performance of the Swoogle upto the mark.

3.2 Methodology used to solve the problem

To solve the above problem, tool is required with the following features:

- Tool which is capable of creating Different types of the repositories.
- Tool has to accept the data in RDF, N triples, Trutle, N3, TriX so many other.
- Tool should have capability of converting different formats among each other (e.g. RDF,N triples,Trultle,N3,TriX)
- Repositories are to be made easily.
- Able to integrate more no of RDF files in the single repository.
- Provide the facility of querying on the RDF files in the repository.
- Provide the convenient way to import repositories.
- Able to create the remote repository.

The Sesame server is the better option because its features satisfy all above requirements. The Sesame Server is an open source Java framework for storage and querying of RDF data. The framework 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 such as N triples,Trutle,N3,TriX,TriG and query result formats.

CHAPTER 4
IMPLEMENTATION

4.1 Starting Sesame server

The Sesame 2.0 server software comes in the form of two Java Web Applications: Sesame (HTTP) server and 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. Instead, the user oriented functionality is part of OpenRDF Workbench. OpenRDF Workbench provides a web interface for querying, updating and exploring the repositories of a Sesame Server.

First need to be download the Sesame 2.0 SDK. Both 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 that is apache tomcat. The deployment process is container specific.

Start with installing java 5 or newer with its proper path then set environment variable JAVA_HOME (must be in capital) 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 step **Start button→Run→Cmd→go to the location directory where Tomcat is install→startup.**

After following these steps Tomcat server will be start and new command prompt window will open with named Tomcat in the title bar. As well as shows the message "INFO: Server startup in 9852 ms". This message shows that Server started successfully. The number of millisecond shown in message may not be the same at all the time. The following figure shows the Tomcat server startup window.

.

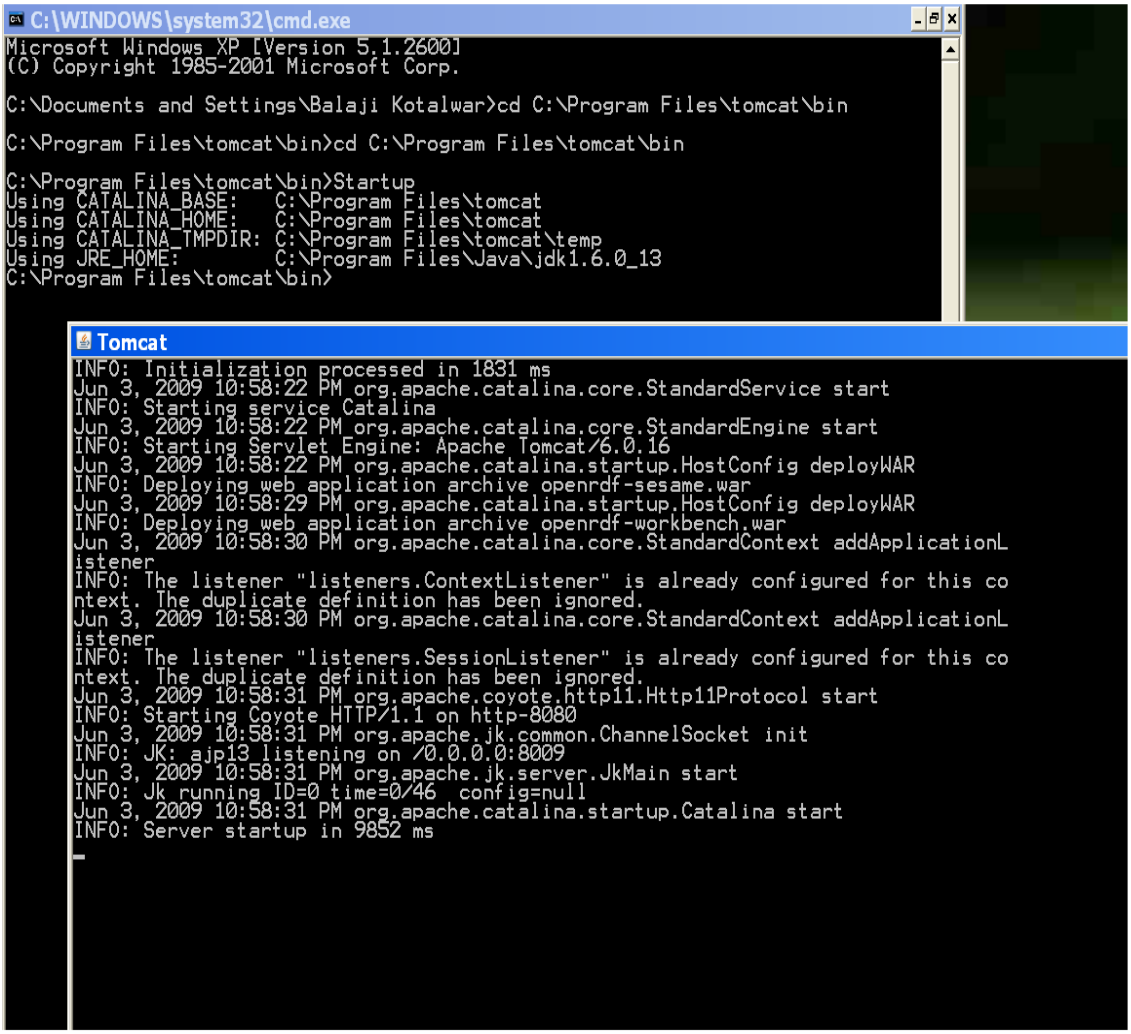

Figure 4.1: Tomcat Server startup window

Next step is to start any browser and write http://localhost:8080/manager/html in the URL link. By clicking on this URL Apache server will run on local machine Then Both Sesame Server and OpenRDF Workbench war-files need to be deployed in apache Tomcat sever. After deployed the Sesame Server webapp, as shown in the figure you should be able to access it, by default, at path OpenRDF sesame. By pointing browser to that location to verify that the deployment succeeded. Browser will show the Sesame welcome screen as well as some options to view the server logs, among other things. Similarly, after deployment, the OpenRDF Workbench should be available at path /OpenRDF-workbench. The following figure will come out on computer.

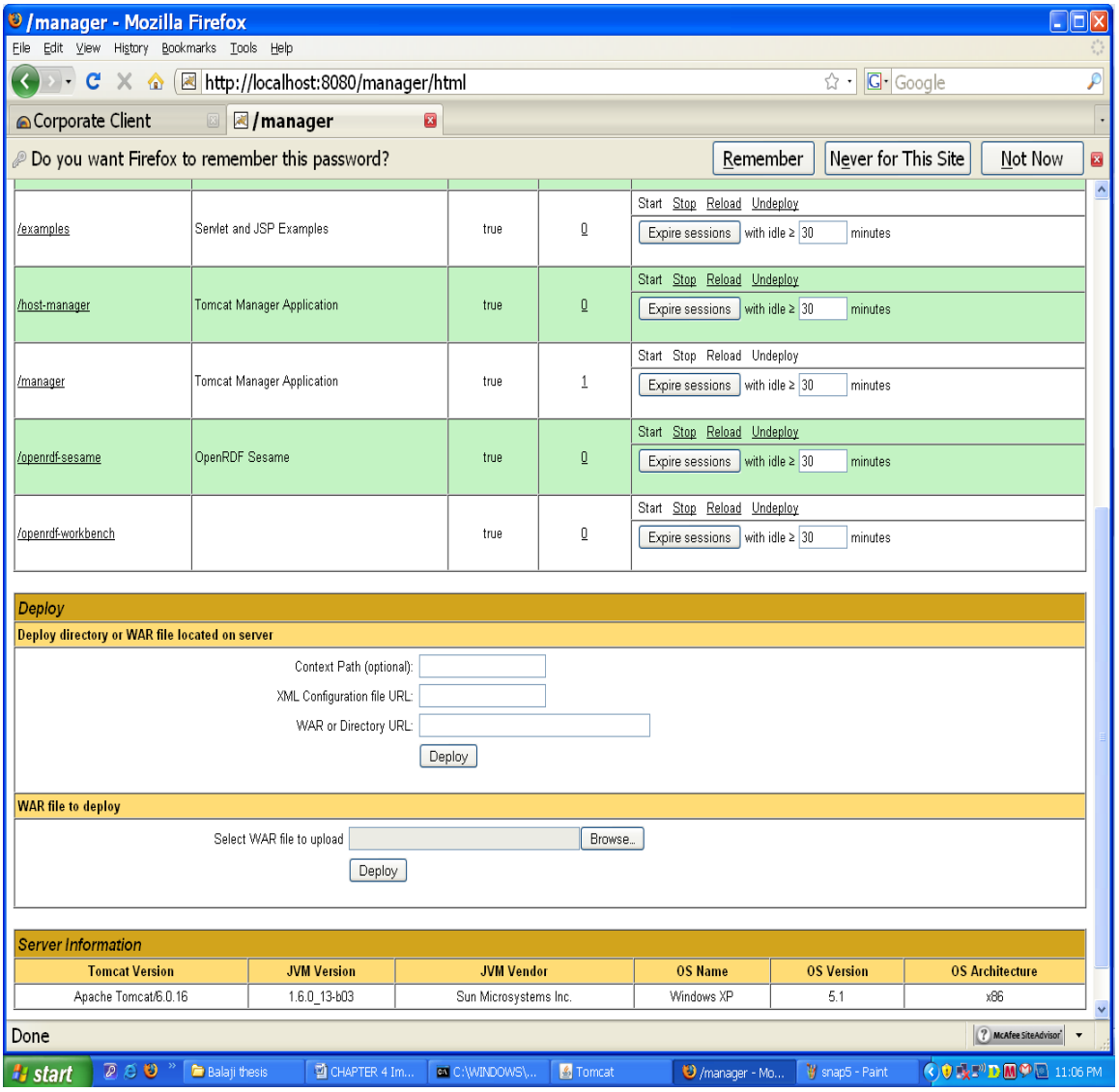

Figure 4.2: Apache Sever Deploy Window

There are only really two things likely to go wrong during the stand-alone Tomcat installation:

(1) the most common problem is when another web server (or any process for that matter) has laid claim to port 8080. This is the default HTTP port that Tomcat attempts to bind to at startup. To change this, open the file:

$CATALINA_HOME/conf/server.xml and search for '8080'. Change it to a port that is not in use, and is greater than 1024, as ports less than or equal to 1024 require super user access to bind under UNIX. Restart Tomcat then only it is ready to work. Replace the "8080"in the URL with new port no which you're using to access Tomcat. For example, if port no 8080is replaced by the port to 1977, and then must request the URL http://localhost:1977/ in the browser.

(2) The 'localhost' machine isn't found. This could happen if server runs behind a proxy. If that's the case, make sure the proxy configuration for browser knows that one shouldn't be going through the proxy to access the "localhost". Change it.

In Internet Explorer, Tools -> Internet Options -> Connections -> LAN Settings.

Just click on the: /OpenRDF-Sesame in the apache server window. The following Window will come. That window is 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. Instead, the user oriented functionality is part of OpenRDF Workbench

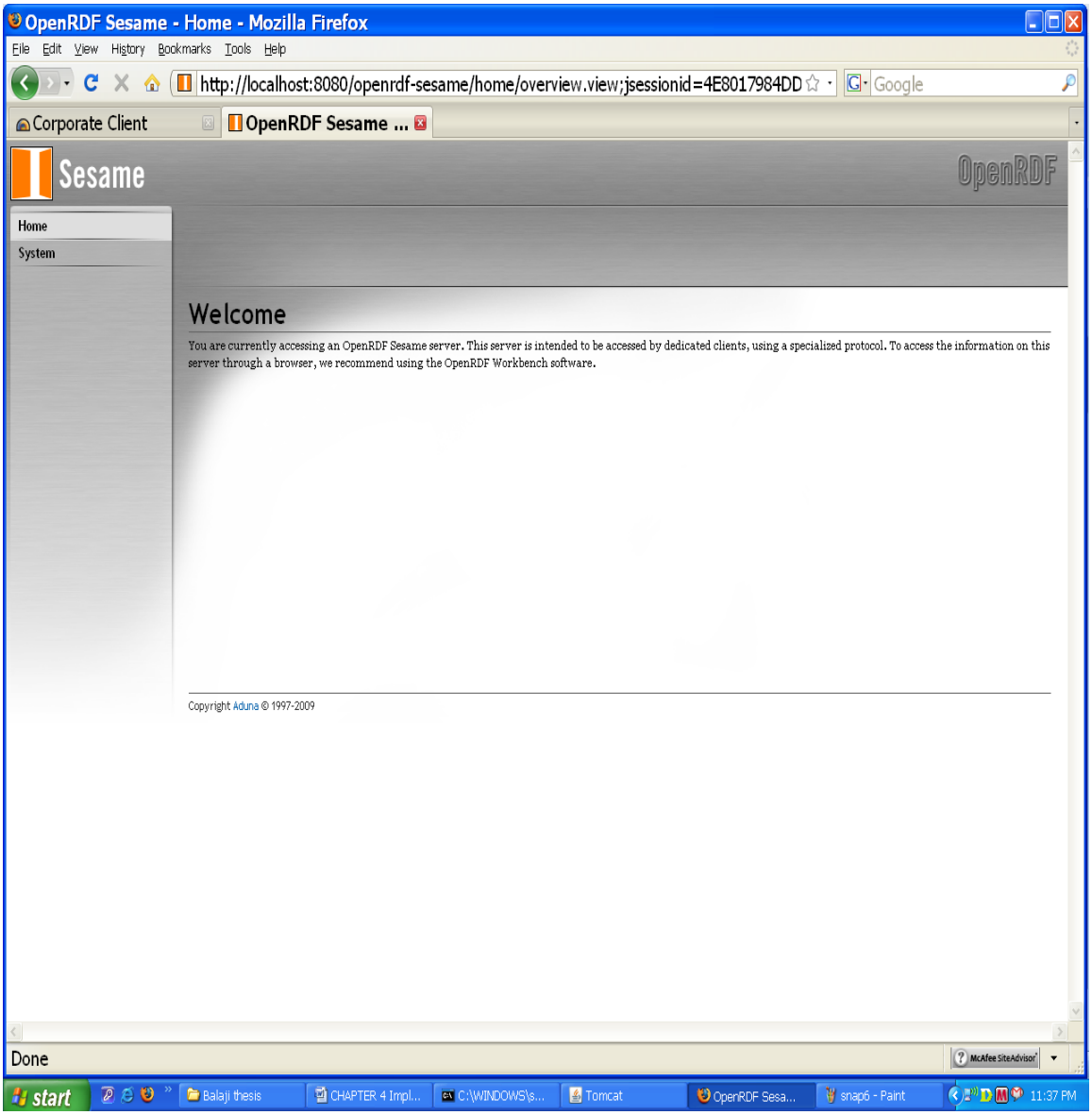

Figure 4.3: Sesame Welcome Window

Sesame Workbench is provides all the applications of sesame to the user. It is very user friendly and provides all the operations. The following window shows the sesame workbench window. Sesame workbench is use to create and delete repositories as well as provide number of options such as summary, namespace, contexts, explore, query, export, etc for operation on repository.

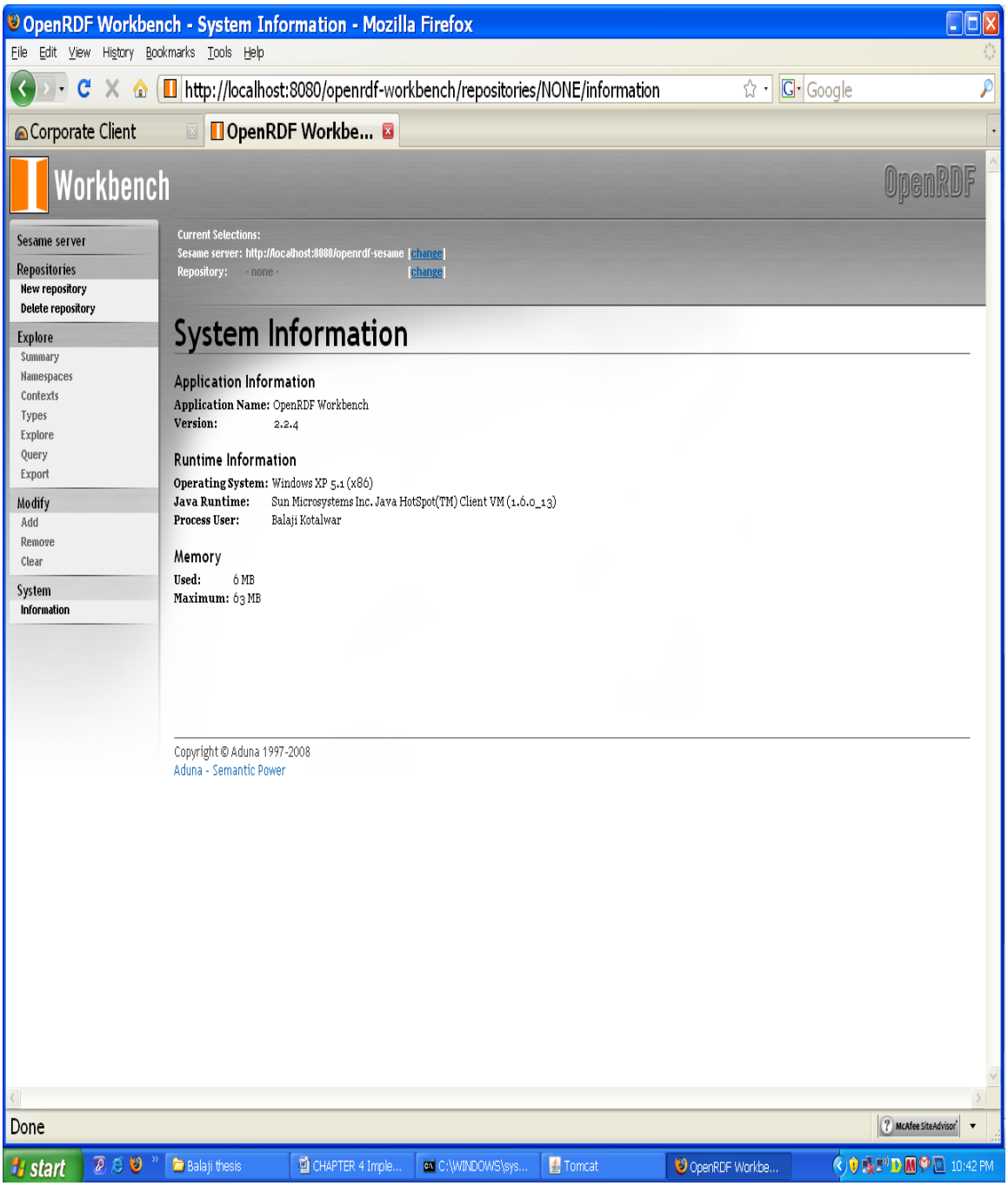

Figure 4.4: Sesame Workbench Window

4.2The Repository API

The Repository API is the central access point for Sesame repositories. Its purpose is to give a developer-friendly access point to RDF repositories, offering various methods for querying and updating the data, while hiding a lot of the unwanted details of the underlying machinery from user.

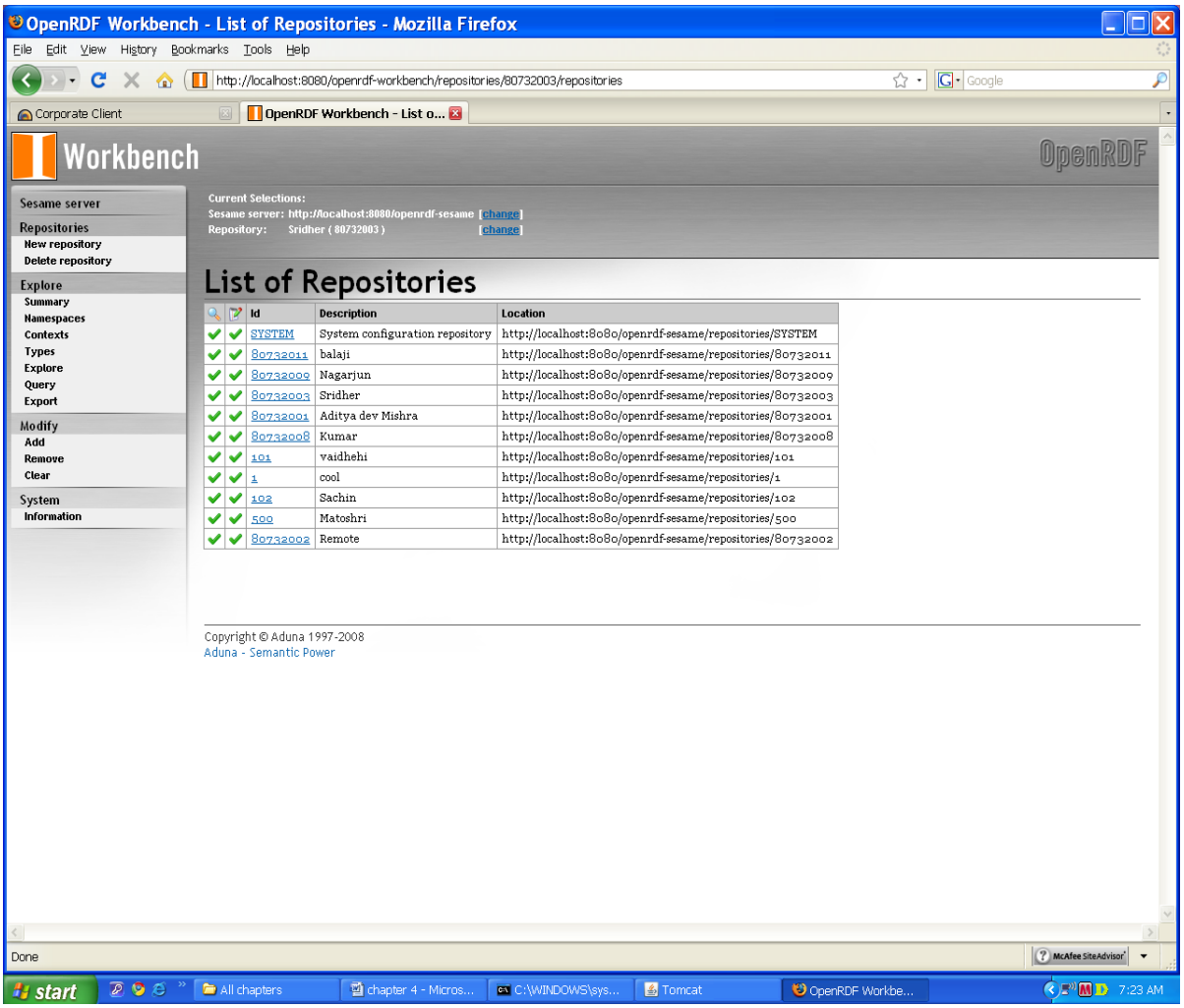

Figure 4.5: Repository API

Click on the Button New repository following window will come in which need to specify the type of repository, repository ID and Title of the repository. By clicking Next and then Create buttons a repository with desired type will be created.

There are many different ways to create and access Repositories:

1. Creating a Main Memory RDF Repository

One of the simplest configurations is a repository that just stores RDF data in main

memory without applying any inferencing or whatever. This is most common way to create and the fastest type of repository that can be used. The following window shows how to create and initializes a non-inferencing main-memory repository:

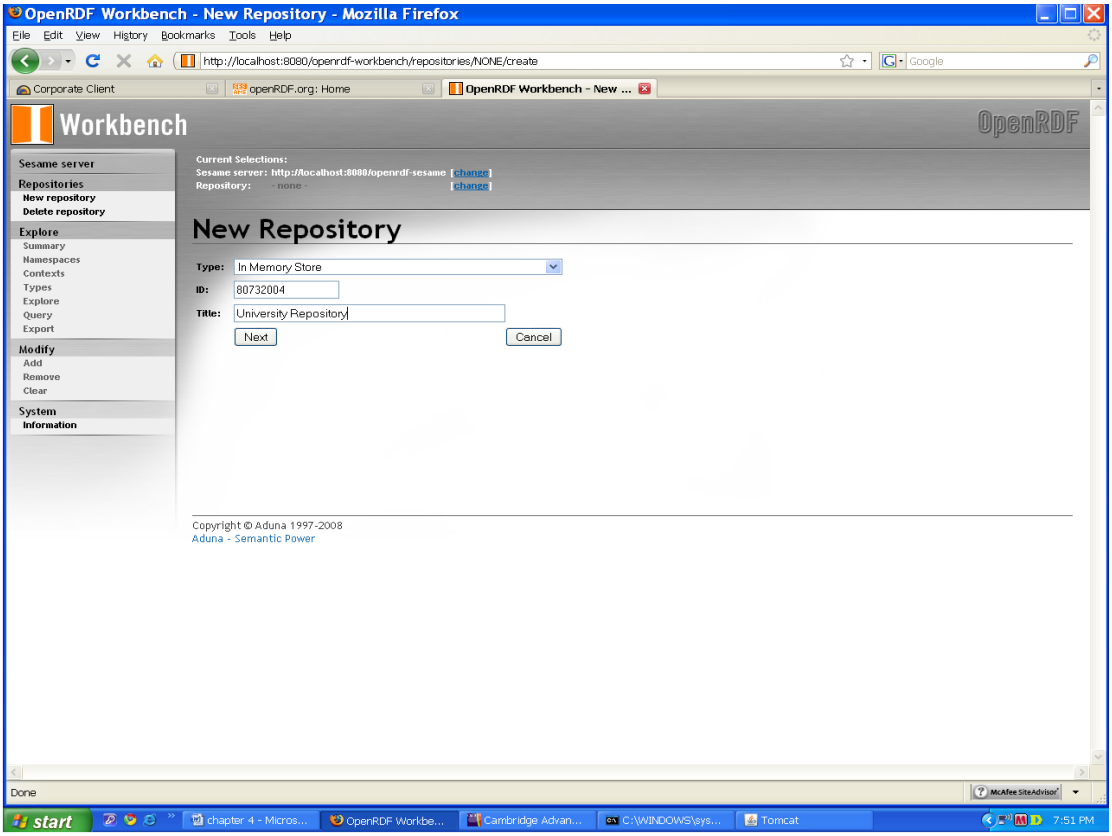

Figure 4.6: Main Memory RDF repository window

The repository that is created by the above way is *volatile*: its contents are lost when the object is garbage collected or when the program is shut down. This is fine for cases where, for example, the repository is used as a means for manipulating an RDF model in memory.

2 Creating a Native RDF Repository

A Native RDF repository does not keep its data in main memory, but instead stores it directly to disk (in a binary format optimized for compact storage and fast retrieval). It is an efficient, scalable and fast solution for RDF storage of datasets that are too large to keep entirely in memory.

3 Accessing a Remote Repository

Working with remote repositories is just as easy as working with local ones. here simply using a different `Repository` object, the `HTTP Repository`, instead of the `SailRepository` class. The SAIL API is a set of Java interfaces that has been specifically designed for storage and retrieval of RDFS-based information. The main design principles of SAIL are that the API should:

- Define a basic interface for storing RDF and RDFS, and retrieving and deleting RDF and RDFS from (persistent) repositories.
- Abstract from the actual storage mechanism; it should be applicable to RDBMSs, file systems, or in memory storage.
- Be usable on low end hardware like PDAs, but also offer enough freedom for optimizations to handle huge amounts of data efficiently on e.g. enterprise level database clusters.
- Be extendable to other RDF-based languages like DAML+OIL.

To create remote repository requirement is of course that there is a Sesame server running on some remote system, which is accessible over HTTP. This type of repository can be made by following way Choose the option “Remote RDF Store” in the Type button at the time of creating repository by clicking next button sesame will ask to specify Sesame sever Location and Remote repository ID by filling these field click on Create button it will create desired remote repository.

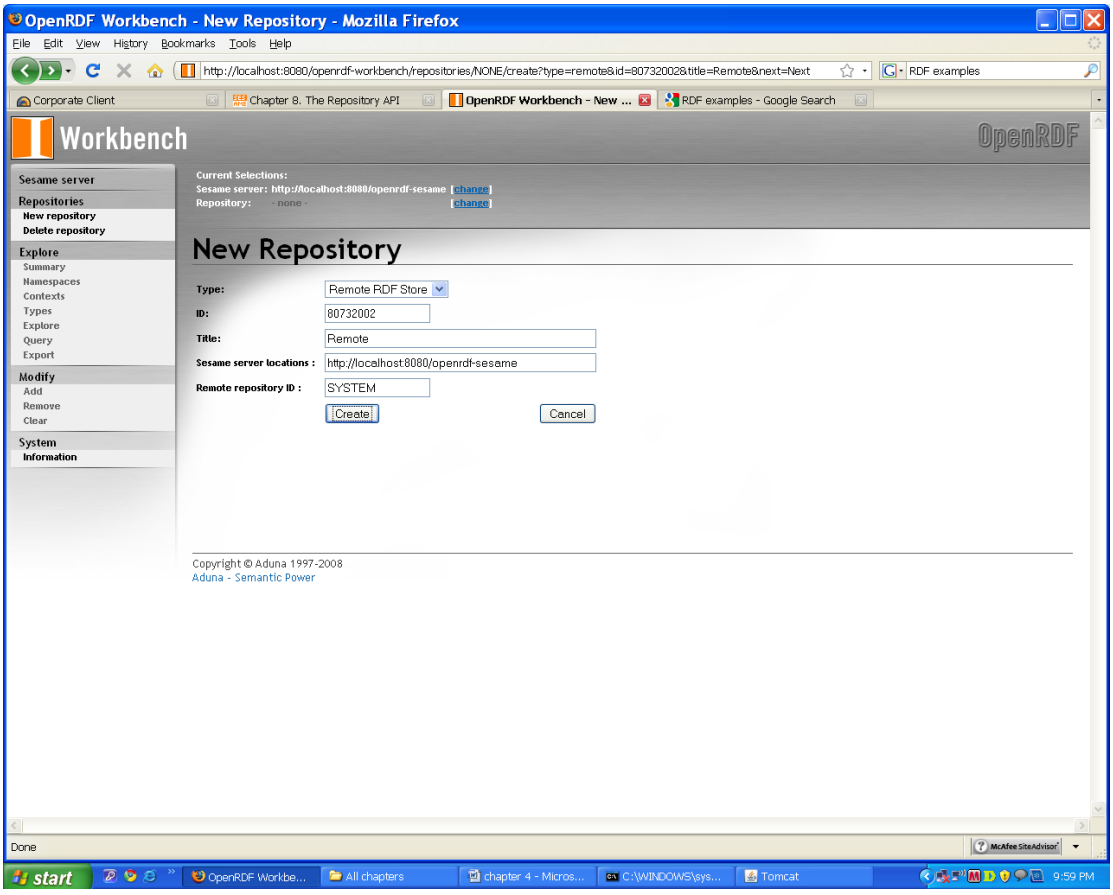

Figure 4.7: Remote RDF repository window

4.3 Adding RDF to a repository

The Repository API offers various methods for adding data to a repository. Data can be added by specifying the location of a file that contains RDF data, and statements can be added individually or in collections. More specifically, one way simply browse the RDF file stored on computer or another way is add the RDF file from the internet by specifying the location to the “RDF data URL” field. Third way to add RDF file in repository is directly write the RDF code to the “RDF content” field.

Base URI field show the location of RDF file. If someone browsing file he need not have to specify this filed it will show automatically the location of file. Sesame supports the notion of context, which is a way to group sets of statements together through a single group identifier (this identifier can be a blank node or a URI). A very typical way to use context is tracking provenance of the statements in a repository, that is, from which file these statements originate. For example, consider an

application where RDF data added from different files to a repository, and then one of those files is updated. One has to replace the data from that single file in the repository, and to be able to do this there is need a way to figure out which statements are to be removed. The context mechanism gives a way to do that.

Data Format field plays important role. Sesame accepts data in not only RDF format but also TriG, TriX, N-Triples, N3, Turtle format. Choose the data format before uploading file to the repository. Then clicking the button "Upload" file will successful upload to the repository.

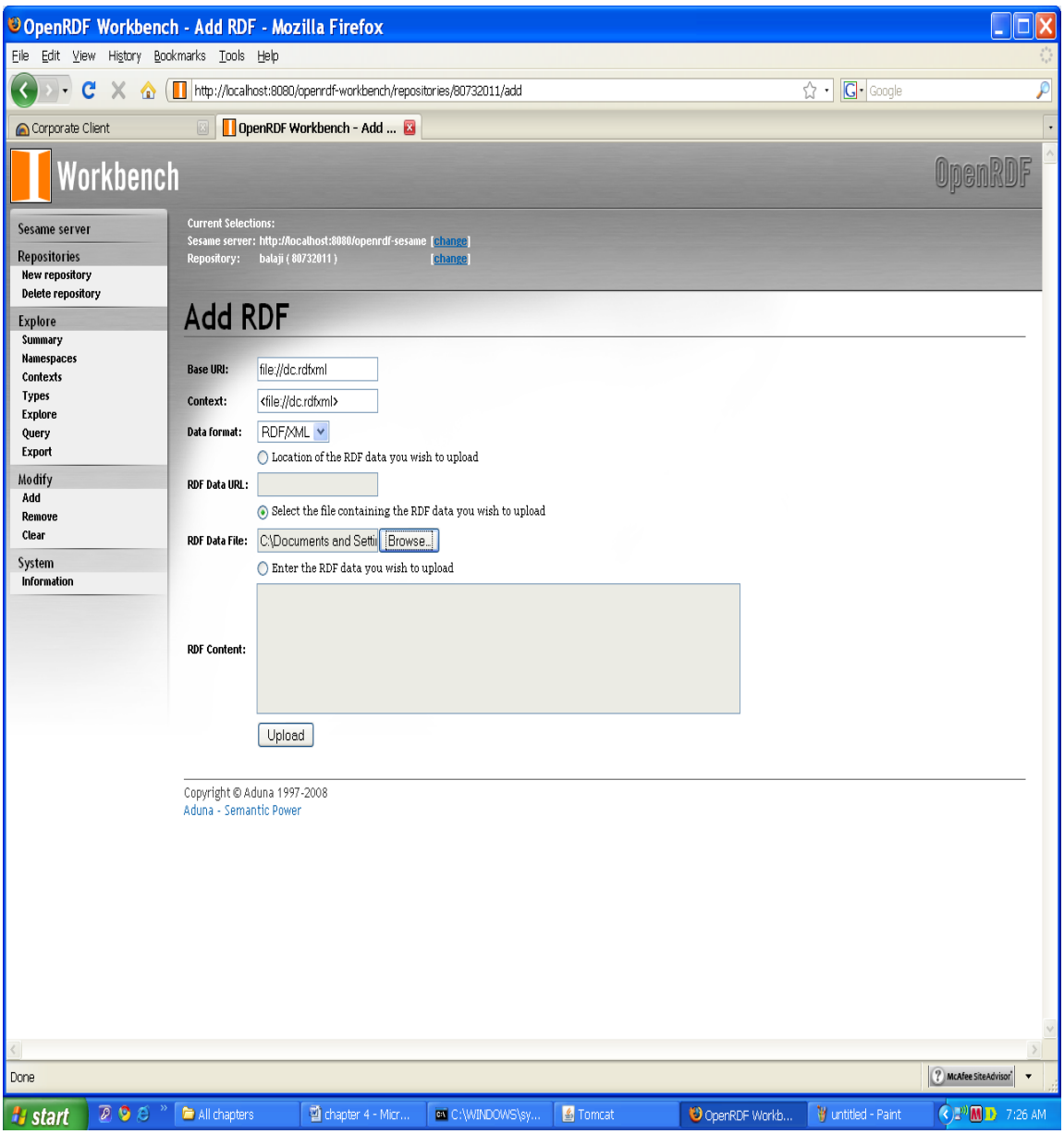

Figure 4.8: Add RDF Window

4.4 Exploring RDF contexts in the repository

Sesame supports the notion of context, which is a way to group sets of statements together through a single group identifier. Other way we can say that these contexts are the subsections in the repository. By clicking on the explore button a window will open which shows the available contexts in that repository. Choose the context whose data want to explore by clicking it the explored data in the form of subject, predicate, object format will be available. Following figure shows result.

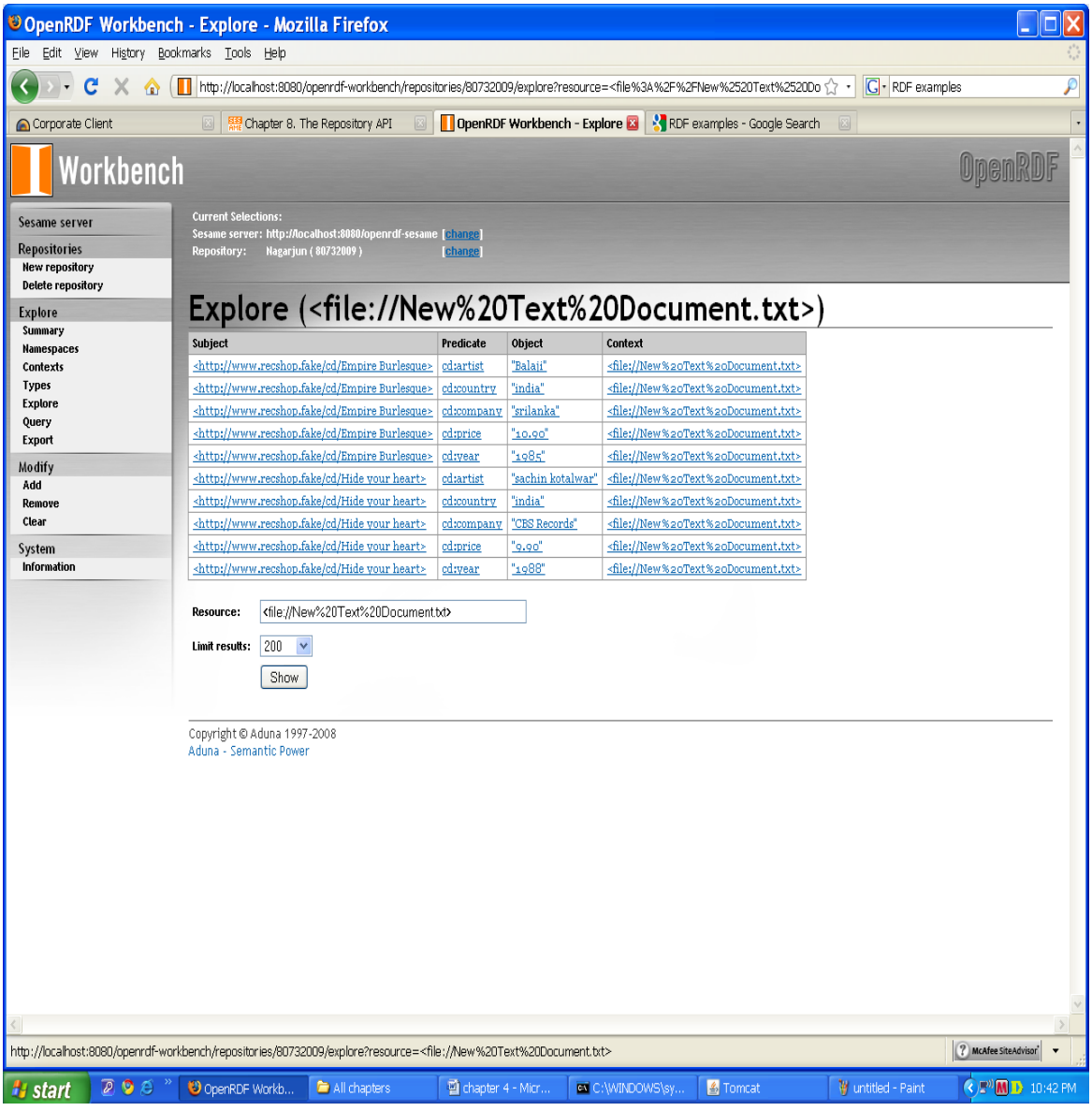

Figure 4.9: Explore Window

4.5 Exporting repository

Sesame is also providing the facility to export the repositories which are available on the server. This can be done by clicking the export button. The following window will come on computer screen. Choose the format which is required. Sesame provides the six different formats to download repositories. Clicking “download” button repository with given format will be download.

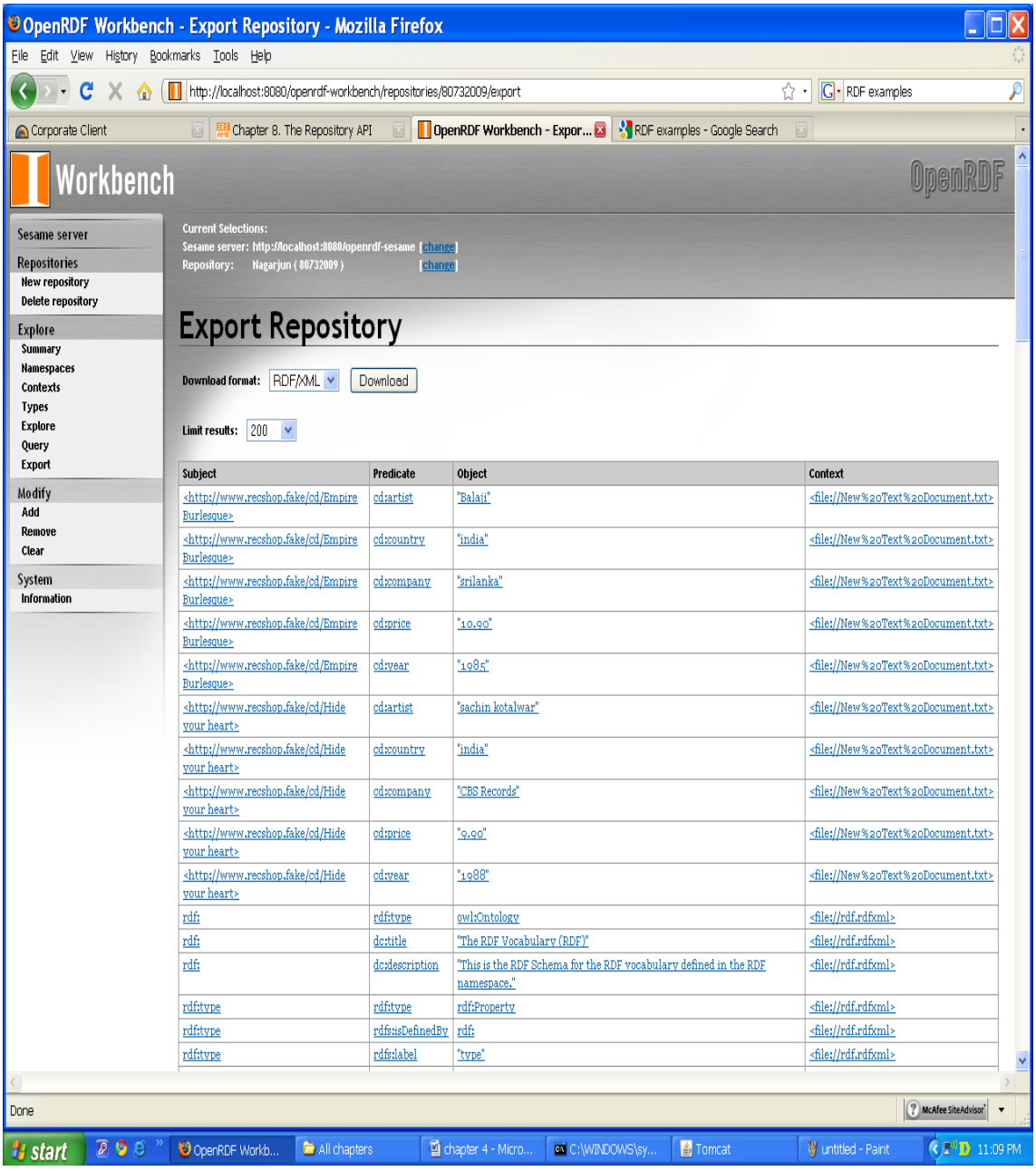

Figure 4.10: Export Repository Window

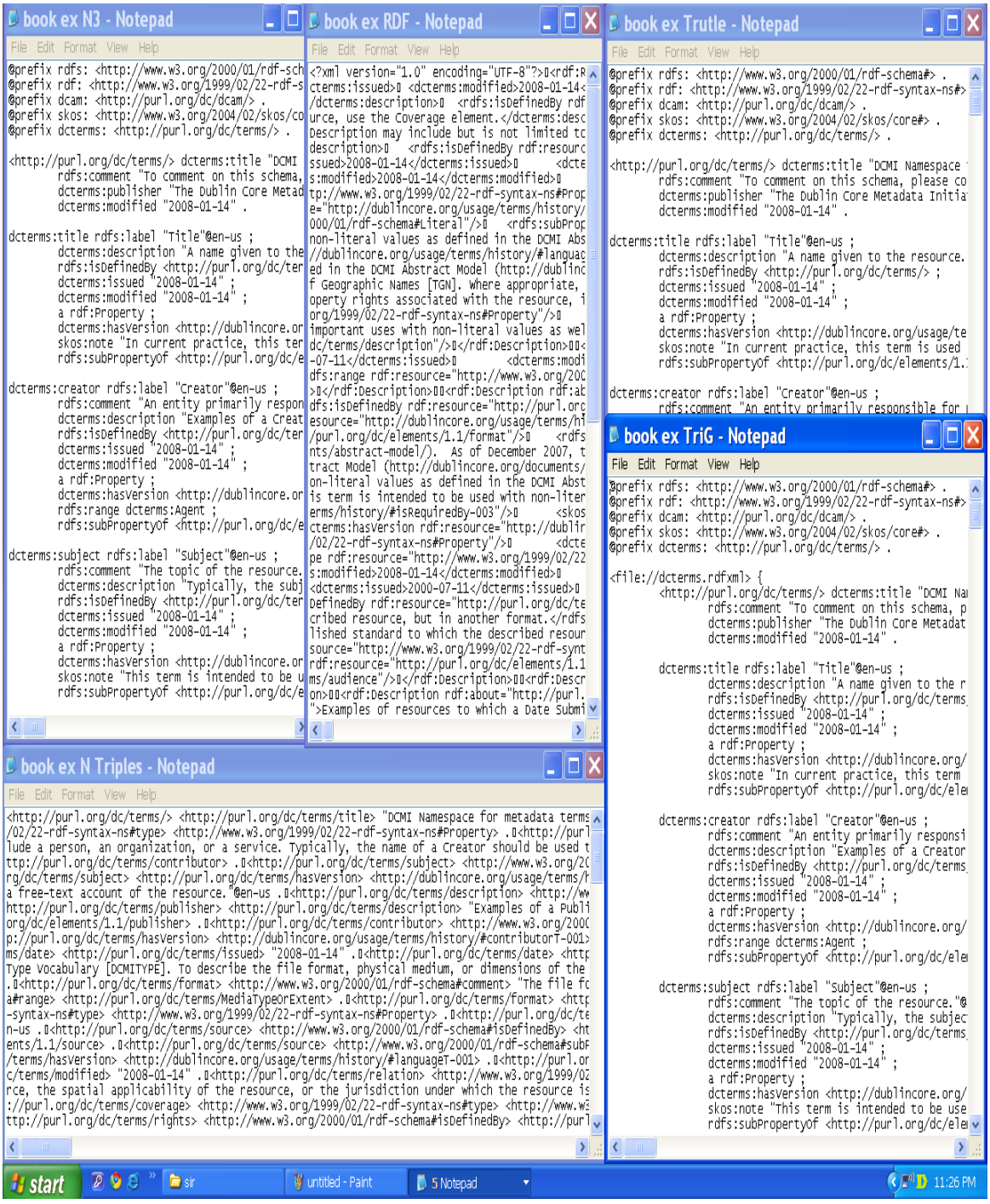

Figure 4.11: Exported Repository in all Formats

4.6 Querying a repository

The Repository API has a number of methods for creating and evaluating queries. Three types of queries are distinguished: tuple queries, graph queries and Boolean

queries. The query types differ in the type of results that they produce. The result of a tuple query is a set of tuples (or variable bindings), where each tuple represents a solution of a query. This type of query is commonly used to get specific values (URIs, blank nodes, and literals) from the stored RDF data.

The result of graph queries is an RDF graph (or set of statements). This type of query is very useful for extracting sub-graphs from the stored RDF data, which can then be queried further, serialized to an RDF document, etc.

The result of Boolean queries is a simple Boolean value, i.e. true or false. This type of query can be used to check if a repository contains specific information.

Sesame supports two query languages: SeRQL and SPARQL. The following Figure will shows the Query Repository Screen

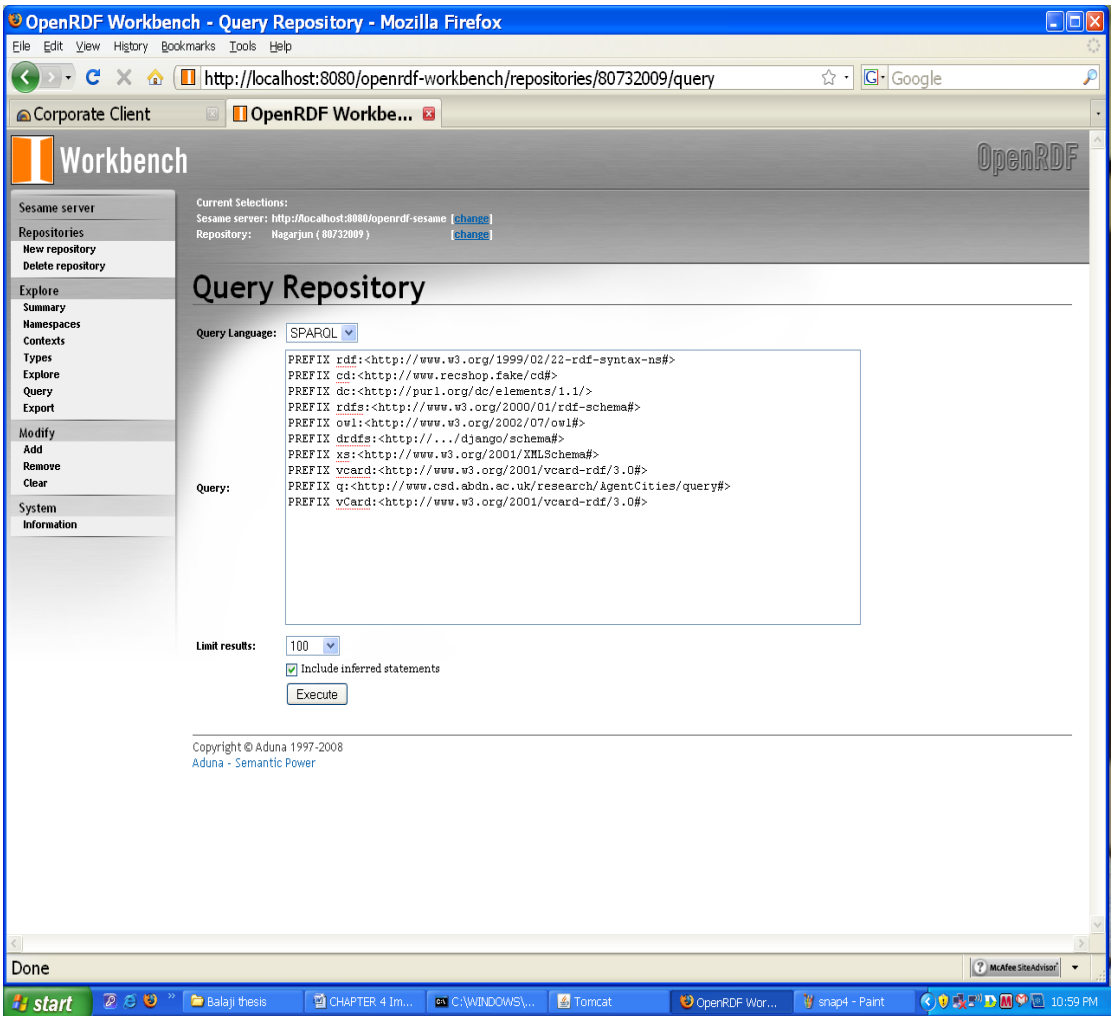

Fig 4.12: Query Repository window

4.7 Comparison with others

Database Compatibility:

	My SQL	**PostgreSQL**	**Oracle**	**SQL Server**	**Derby**
Sesame	Yes	Yes	Yes	Yes	No
Jena	Yes	Yes	Yes	Yes	Yes
AllegroGraph	NA	NA	NA(backup)	NA	NA
Mulgara	NA	NA	NA	NA	NA

Table 4.1: Database Compatibility Comparison Table

Tool Support:

	Database Management	**Query Execution**
Sesame	GUI and Web Interface	Web Interface
Jena	Command-line tool & Java API	Joseki & Twinkle
AllegroGraph	Java, HTTP, Lisp	Java, HTTP, Lisp
Mulgara	Java, Perl, iTQL	Java, Perl, iTQL

Table 4.2: Tool Support Comparison Table

Reasoning level:

	Reasoning
Sesame	RDFS
Jena	Owl-DL
Mulgara	OWL-Lite
AllegroGraph	OWL

Table 4.3: Reasoning Comparison Table

API Compatibility:

	Sesame API	**Jena API**
Sesame	Yes	Yes (Via Jena Sesame Model)
Jena	Yes (Via Sesame-Jena Adapter)	Yes
AllegroGraph	Yes (Via AllegroGraph interfaces)	Yes (AllegroGraph interfaces)
Mulgara	No	Yes (exposes its own JRDF API)

Table 4.4: API Compatibility Comparison Table

Query Language Support:

	Native RDF Query Lang	**SPARQL Support**
Sesame	RQL	Yes
Jena	RDQL	Yes
AllegroGraph	SPARQL	Yes
Mulgara	iTQL	No

Table 4.5: Query Language Support Comparison Table

CHAPTER 5

CONCLUSION AND FUTURE SCOPE OF WORK

5.1 Conclusion

This work concludes that, The Swoogle Web search system discovers analyzes and indexes semantic web documents on the web. Just as modern search engines and their services have given people access to much of the world's knowledge and data. Swoogle aims to support semantic web developers, software agents and programmers in finding RDF data and schema-level knowledge on the web. Swoogle act as component in number of information integration task and are working on several others.

In this thesis work we successfully worked on Sesame. Sesame is a generic architecture for storing and querying both RDF and RDFS information. Sesame is an important step beyond the currently available storage and query devices for RDF, since it is the first publicly available implementation of a query language that is aware of the RDFS semantics.

An important feature of the Sesame architecture is its abstraction from the details of any particular repository used for the actual storage. This makes it possible to port Sesame to a large variety of different repositories, including relational databases, RDF triple stores, and even remote storage services on the Web. Hence Sesame can be used for creating SWD repositories for Swoogle. Sesame also implements a query engine for RQL, the most powerful RDF/RDF Schema query language to date. And Therefore Sesame would be the good for Swoogle as an query engine also.

5.2 Future Scope of Work

Sesame itself is a server-based application, and can therefore be used as a remote service for storing and querying data on the Semantic Web. As with the storage layer, Sesame abstracts from any particular communication protocol, so that Sesame can easily be connected to different clients by writing different protocol handlers.

Important next steps to expand Sesame towards a full edged storage and querying service for the Semantic Web include implementing transaction rollback support, versioning, extension from RDFS to DAML+OIL and implementations for different repositories. This last feature especially will be greatly facilitated by the fact that the current SAIL implementation is a generic SQL92 implementation, rather than specific for a particular DBMS.

REFERENCES

[1] T. Berners-Lee, .Weaving the Web. Harper, San Francisco, 1999.

[2] 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.

[3] James Hendler, Tim Berners-Lee, .Eric Miller. Integrating Applications on the Semantic Web, Journal of the Institute of Electrical Engineers of Japan, Vol 122(10), October 2002.

[4] I. Horrocks and J. Hendler, The Semantic Web, in proceedings of First International Semantic.

[5] F. Manola, E. Miller, .RDF Primer: W3C Proposed Recommendation., December, 2003. http://www.w3.org/TR/2003/PRrdfprimer-20031215.

[6] Jun Yuan, David H. Jones, Enabling Semantic Access to Enterprise RDB Data. Position Paper for W3C workshop on RDF Access to Relational Databases.

[7] Resource Description Framework (RDF) Model and Syntax Specification http://www.w3.org/TR/1999/REC-rdfsyntax- 19990222.

[8] Li Ding Tim Finin Anupam Joshi Rong Pan R. Scott Cost Yun Peng Pavan Reddivari Vishal Doshi Joel Sachs *Swoogle*: A Search and Metadata Engine for the Semantic Web.

[9] Natalya F. Noy and Deborah L. McGuinness "Ontology Development: A Guide to Creating Your First Ontology", http://protege.stanford.edu/publications/ontology/development/ontology101Noy-mcguinness.html.

[10] T. Berners-Lee. The semantic web. Scientific American, 284(5):35–43, 2001.

[11] N. Guarino. Formal ontology in information systems. In Proceedings of FOIS'98 (Formal Ontology in Information Systems), pages 3–15. IOS Press, 1998.

[12] C. M. Keet. Aspects of ontology integration. Technical report, School of Computing, Napier University, 2004.

[13] Swoogle: searching for knowledge on the semantic web by Tim finin,Li Ding,Rong Pan,Anupam joshi.

[14] The semantic web in one day by York sure,Pascal Hitzler and Eberhart, university of Karlsruhe.

[15] Information Integration and Semantic Web. Tim finin and Anupam Joshi,Li Ding University of Maryland,Baltimore MD.USA.

[16] OntoSearch: Retrieval and Reuse of Ontologies. Edward Thomas, Yi Zhang, Derek sleeman, Joe wright, University of Aberdeen, UK.

[17] Sesame: Architecture for Storing and Querying RDF Data and Schema Information.Jeen Broekstra, Arjohn Kampman,Aidministrator Nederland b.v. And Frank van Harmelen Faculty of Sciences, Vrije University, Amsterdam.

[18] Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen Aidministrator Nederland b.v., Amersfoort, The Netherlands Faculty of Sciences, Vrije Universiteit, Amsterdam, The Netherlands.

[19] J.Davidson, Java API for XML Parsing, Version 1.0 Final Release. http://java.sun.com/aboutJava/communityprocess/final/jsr005/index.html.

[20] From Data Federation Pyramid to the Semantic Web “Birthday Cake”, http://www.mkbergman.com.

[21] Erik Wilde, Katrina Rhoads Lindholm, .XML Foundations. September, 2006, http://dret.net/netdret/docs/wilde-elpub2006-xml.pdf.

[22] Search on the Semantic Web Li Ding, Tim Finin, Anupam Joshi, Yun Peng, Rong Pan, Pavan Reddivari University of Maryland, Baltimore County,Baltimore MD 21250 USA.

[23] I. Horrocks. DAML+OIL: description logic for the semantic web. IEEE Data Engineering Bulletin, 25(1):4{9, 2002.

[24] T. Haveliwala. Efficient computation of pageRank. Technical Report 1999-31, 1999.

[25] Li Ding, Tim Finin, Yun Peng, Paulo Pinheiro da Silva, and Deborah L.McGuinness. Tracking rdf graph provenance using rdf molecules. Technical Report TR-CS-05-06, UMBC, April 2005.

LIST OF PUBLICATIONS

[1] Kotalwar Balaji and Karun Verma, “Semantic Web +Google= Swoogle.” at national Conference on advances in computer networks and information technology NCACNIT-09, held on 24-25 march at Hisar, Haryana [Published]. Page no. 253-256. (ID_139).