DESIGN OF AN ONTOLOGY DRIVEN INFORMATION SYSTEM IN SEMANTIC WEB

THESIS

submitted in fulfillment of the requirement of the degree of

DOCTOR OF PHILOSOPHY

to

J.C. BOSE UNIVERSITY OF SCIENCE & TECHNOLOGY, YMCA

by

RANJNA JAIN
Registration No: YMCAUST/PH58/2011

Under the Supervision of

Dr. NEELAM DUHAN
Assistant Professor

(Late) Dr. A.K. SHARMA
Professor

Department of Computer Engineering
Faculty of Engineering and Technology
J.C. BOSE University of Science &Technology, YMCA
Sector-6, Mathura Road, Faridabad, Haryana, INDIA

July 2019

DEDICATED

to

My Husband Mr. Ravi Jain

and

My beloved Daughter Aadya Jain

CANDIDATE'S DECLARATION

I hereby declare that this thesis entitled "**DESIGN OF AN ONTOLOGY DRIVEN INFORMATION SYSTEM IN SEMANTIC WEB" by RANJNA JAIN,** being submitted in fulfillment of requirement for the award of Degree of Doctor of Philosophy in the Department of Computer Engineering under Faculty of Engineering and Technology of J.C. Bose University of Science and Technology, YMCA, Faridabad, during the academic year March 2012-December 2018, is a bonafide record of my original work carried out under the guidance and supervision of **Dr. NEELAM DUHAN, ASSISTANT PROFESSOR, DEPARTMENT OF COMPUTER ENGINEERING & (Late) Dr. A.K. SHARMA** & has not been presented elsewhere.

I further declare that the thesis does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university.

(RANJNA JAIN)

Registration No. YMCAUST/PH58/2011

CERTIFICATE

This is to certify that this thesis entitled **"DESIGN OF AN ONTOLOGY DRIVEN INFORMATION SYSTEM IN SEMANTIC WEB"** by **RANJNA JAIN** submitted in fulfillment of the requirement for the award of Degree of Doctor of Philosophy in **DEPARTMENT OF COMPUTER ENGINEERING**, under Faculty of Engineering and Technology of J.C. Bose University of Science and Technology, YMCA, Faridabad, during the academic year March 2012-December 2018, is a bonafide record of work carried out under my guidance and supervision.

I further declare that to the best of my knowledge; the thesis does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university.

Date:

Dr. NEELAM DUHAN
ASSISTANT PROFESSOR
Department of Computer Engineering
Faculty of Informatics & Computing
J.C. Bose University of Science & Technology, YMCA, Faridabad

ACKNOWLEDGEMENT

It gives me immense pleasure to acknowledge the people whose priceless contributions helped me reach here. Without their support, this piece of academic work, in the form of my doctoral thesis, would have just been reduced to mere ink on paper. First and foremost, my deepest and heartfelt thanks to (Late) Dr. A.K. Sharma who has been not just a guide par excellence but also an inspiring teacher, Sharma sir not only created a solid foundation in my mind, but also nurtured it over the years till his last days through his tireless efforts.

Sir, you have shown me the right path always, lit the lamp of clarity whenever I was in doubt, corrected me when I was wrong, encouraged me when I was right, criticized me when I became over-confident and motivated me whenever I would lose hope.

At this moment of accomplishment, I am greatly indebted to my research guide, Dr. Neelam Duhan, who accepted me as her Ph.D. student and offered me her mentorship, sisterly love and care. This work would not have been possible without her guidance and involvement, her support and encouragement on daily basis from the start of the journey till date. Under her guidance, I successfully overcame many difficulties and learnt a lot. Her own zeal for perfection, passion, unflinching courage and conviction has always inspired me to do more. She has taught me another aspect of life, that, “Believe in yourself” and this encouraged me to move toward my goals. For all these, I sincerely thank her from bottom of my heart and will be truly indebted to her throughout my life time.

I am thankful to B.S. Annapurna Institute of Technology & Management, Faridabad for giving me the permission to carry out Ph.D. I sincerely thank Mr. Divya Gupta, Dr. S.S. Tyagi and Dr. Pawan Bhadana for providing me time and space for the research work.

I would like to express my sincere gratitude to chairperson Dr. Komal Kumar Bhatia and Dr. Atul Mishra, Dean R&D for their valuable advice, support and information on different aspects.

It is my firm belief that no success in life can ever be achieved without the well wishes and support of one’s family. And when I look back from where I have reached today, my belief only becomes stronger. First and foremost, I show my very special gratitude towards my Parents, Sh. Uttam Chand Gupta and Smt. Manorama, for believing in me and giving me the best lesson of life. My parents inspired me to do better with every passing day. I am sure no one today would be as proud as them seeing me complete my doctoral thesis. I also heartily thank my dearest Father-in-law (Late) Sh. Netra Pal Singh, Mother-in-law Mrs. Kamlesh Jain, Uncle Sh. Satish Chander Goel, Aunt Smt. Chandrakanta Goel and my whole family for their unwavering support.

I owe thanks to a very special person, my husband, Mr. Ravi Jain for his continued and unfailing love, support and understanding during my pursuit of Ph.D. degree that made the completion of thesis possible. You were always around at times I thought that it is impossible to continue, you helped me to keep things in perspective. I greatly value his contribution and deeply appreciate his belief in me. I appreciate my beloved daughter Aadya Jain for abiding my ignorance and the patience she showed during my thesis writing. Words would never say how grateful I am to both of you. I consider myself the luckiest in the world to have such a lovely and caring family, standing beside me with their love and unconditional support.

It’s my fortune to gratefully acknowledge the support of my friends, Deepkiran Munjal, Bhawna Chauhan, Geetanjali Gandhi, Anju Gera and Girija Srikanth for their support and generous care throughout the research tenure. They were always beside me during the happy and hard moments to push me and motivate me. Big thanks to all my fellow research scholars and friends, particularly, Ms. Sumita Gupta and Ms. Mamta Kathuria for their co-operation and support.

And finally, thinking that one entity without whose grace, all the above wonderful people wouldn’t have come in to my life and without whose blessings, I wouldn’t have been where I am. Thank you almighty, the ever knowing, omnipresent and ever-forgiving God for being kind and watching over me all these years and I know that you will continue to do so forever.

(RANJNA JAIN)

Registration No. YMCAUST/PH58/2011

ABSTRACT

World Wide Web (WWW) is a huge repository of information and its success is due to its decentralized structure where anyone irrespective of its geographic location can publish its content. However, due to large amount of information; it is becoming difficult to access the relevant information. To deal with this, many keywords based search engines such as Google, Bing etc. are available which retrieves results with respect to user query. However, the main limitation of these search engines is that they produce results based on keyword matching due to unstructured format of data. This unable machines to understand the meaning of data and thus they are limited to use for presenting data only. To deal with these issues, Tim-Berner-Lee envisioned Semantic Web which gives more emphasis on data rather than documents.

In Semantic Web, data is represented in structured format using semantic web technologies such as RDF, OWL etc. Each data is uniquely identified as URI (Uniform Resource Identifier) which removes ambiguity from the data and thus makes data machine readable. Developers have started to represent data in structured format using these technologies and to provide more relevant results several Semantic Search Engines such as Swoogle, Falcon, and Semantic Web Search Engine (SWSE) etc. has been developed. But, it does not conclude that existing data available in the form of semi-structured and unstructured format is of no relevance. For instance, in the domain of Job, there exists several Jobboards which provide job posts to its users. But, these systems are keyword based, therefore retrieve results based on keyword matching only which results less relevant results. This information can also be utilized by transforming semi-structured/ unstructured data into structured format, thereby expanding the coverage area of information in semantic web. But, less efforts have been done in this area.

By knowing the advantages of Semantic Web and availability of abundant resources in the current web, the present thesis work contributes to the research efforts of designing and developing a framework of Ontology Driven Information System in Semantic Web in the domain of Job. The system has developed ontologies with respect to selected Jobboards. The "OntoJobextractor" framework extracts semi-structured data from several Jobboards and transforms into structured format using Jobboard specific ontologies. A framework for ontology alignment has been proposed which aligns the ontologies that will be useful during query processing. “OntoJob Query Processor” processes user’s given keyword based queries by converting into SPARQL format. The developed framework Jobology is integrated with Student domain to provide Job posts per his/her qualification and keyskills at one place.

The ontologies were developed in Protégé framework tool successfully and have been validated for consistency and certainty using Pellet reasoner with the set of SPARQL queries. The developed system is efficient in terms of covering and extracting the required relevant data from semi-structured formatted webpages. The developed system is efficient in terms of establishing alignment between ontologies for query processing. This system provides a friendly user interface to its users. The system has been compared with traditional mechanism of searching with the existing Jobboards using the evaluation metrics that shows an improvement over the existing systems. The developed system provides more relevant results at one place on a single click. The developed system supports scalability, robustness and generate more relevant results to fulfill the user requirement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
ARPANET	Advanced Research Projects Agency Network
DAML	DARPA Agent Markup Language
DAML+OIL	DARPA Agent Markup Language + Ontology Inference Layer
DBLP	DataBase systems and Logic Programming
EU	European Union
Flogic	Frame Logic
GUI	Graphical User Interface
IR	Information Retrieval
KIF	Knowledge Interchange Format
OCML	Options Configuration Modeling Language
OIL	Ontology Inference Layer
OILEd	OIL Editor
OML	Ontology Markup Language
OWL	Web Ontology Language
OWL-DL	Web Ontology Layer- Description Logic
QDex	Query Detection & Extraction
RDF	Resource Description Framework
RDFa	Resource Description framework with attributes
RDQL	RDF Query Language
RuleML	Rule Markup Language
SAOR	Scalable Authoritative OWL Reasoner
SeRQL	Sesame RDF Query Language
SHOE	Simple HTML Ontology Extension
SPARQL	SPARQL Protocol and RDF Query Language
SquishQL	SQL like Query Language
SWDB	Semantic Web DataBase
SWO	Semantic Web Object
TCP/IP	Transmission Control Protocol/Internet Protocol
URI	Uniform resource Identifier
URL	Uniform resource Locator
WWW	World Wide Web
XML	Extensible markup Language

Chapter I

INTRODUCTION

1.1 GENERAL

World Wide Web (WWW) [1] is regarded as the largest human construct in history which has narrowed the distance and enhanced the communication between individuals. Generally, user considers the term “Internet” [2, 3] and “The Web” interchangeably but in the actual, they are different but related. Internet is a massive network of networks, a networking infrastructure. It connects millions of computers together globally, forming a network in which any computer can communicate with any other computer. Web is a way of accessing information over the medium of the internet. It is an information sharing model that is built on the top of internet. It uses browsers to access web documents called webpages that are linked to each other via hyperlinks.

The development journey of WWW is broadly divided into three phases named as *Web1.0*, *Web2.0* and *Web3.0* [4, 5]. *Web1.0,* which is also named as web of documents, considers web as read only web. The main purpose of this phase was to provide content via static webpages. It was referred as first generation of WWW which was basically defined as:

Web 1.0 is an information space in which the items of internet referred to as resources are identified by global identifier called as Uniform Resource Identifiers (URIs).

Web 2.0, which is also known as current web, is the second generation of web. It is defined as:

Web 2.0 is the combination of concepts, trends and technologies that focus on user collaboration, sharing of user generated content and social networking.

The technologies of *Web 2.0* allow assembling and managing large global crowds with common interests in social interactions. In this era, the web user cannot only read the content but also write, modify and update the content online, it supports collaboration and help to gather collective intelligence. This feature brought revolution in WWW resulting in availability of billions of web pages and created the need of information retrieval tools as it was not possible for user to learn URLs of every webpage.

Web Mining [6, 7, 8] *or Web Information Retrieval (Web IR)* is the process of extracting useful information from among the petabytes of data that make up the WWW.

With such a large collection of information, search engines [9, 10] are emerging as an important information retrieval tool for searching the relevant information. A search engine provides a user an interface through which he/she enters his/ her query in natural language and in return, search engine provides a list of ranked web pages to user. For this purpose, search engine has many vital components such as crawler, indexer, query processor, ranker etc. Search engine takes query from user via user interface which is a short piece of text and because of unstructured behavior of data [11] in current web, query processor is not able to understand the semantic intend of the query and thus retrieves results only based on keyword matching i.e. lexical matching [12]; and sometimes results in generation of irrelevant results as engine is not able to understood the intend (context) of the user.

To deal with this issue, Tim- Berner Lee envisioned semantic web [13, 14] which emphasis on data rather than documents this is also known as Web 3.0.

Web 3.0 is an extension of the WWW where it can be expressed a natural language understandable and usable by software agents, thus permitting finding, sharing and integrating information easily.

In this, data is represented in structured format using defined languages and constructs. To make computers understand the meanings of things, each and every entity is uniquely identified as URI. This way, it deals with the ambiguity that comes with piece of text.

By knowing the advantages of semantic web and availability of abundant resources in the current web, there is a need to develop a system that represent semi-structured or unstructured data on the current web in structured format using the technologies provided by semantic web. This way, computer will also be able to understand intend of the user query and contribute in retrieving more relevant results. In this thesis, the domain of jobs has been taken into consideration and a system named as *"Jobology": An Ontology Driven Information System in Semantic Web* is proposed which transforms semi-structured data available on the current web in the domain of job into structured format using various components. These components are proposed and explained in the forthcoming chapters, which help the machine to understand the meaning of the content and correspondingly retrieve more relevant results with respect to user query.

1.2 SEMANTIC WEB TECHNOLOGIES

As Per W3C's official

"Semantic Web is defined as *a Common framework that allows data to be shared and reused across application, enterprise, and community boundaries"*

From a technical point of view, the Semantic Web consists primarily of three technical standards:

- **RDF (Resource Description Framework)** [15,16]

  RDF is data modeling language for the Semantic Web. All Semantic Web information is stored and represented in the RDF.

- **SPARQL (SPARQL Protocol and RDF Query Language)** [17,18]

  SPARQL is the query language of Semantic Web. It is specifically designed for querying data across various systems.

- **OWL (Web Ontology Language)** [19,20]

  OWL is the schema language, or knowledge representation (KR) language [21], of the Semantic Web.

1.3 MOTIVATION

Current web tools are keyword based and they produce a list of webpages by matching terms of query with the webpages content. In this web, Computers are used for data presentation only. Due to unstructured nature of data, computers are not able to understand the meaning of content; thus, making them helpless and relying only on keyword based search system. In contrast with this, Semantic Web is a development of the WWW in which data is represented in structured format using semantic web technologies standards such as RDF, OWL etc. In this, computers understand the meaning of data as they are represented in structured format. This factor is making semantic web quite popular and now developers have started representing their data using semantic web technologies to enhance the quality of search results by understanding the meaning of their query. The motivation that leads to get into this work is discussed as below:

a) **Representing semi-structured data in structured form**

A lot of research is going in semantic web as it is an emerging field; researchers have started representing their knowledge using these structured languages. But, current web is also carrying a huge amount of relevant data either in unstructured or semi-structured form [22]. So, there is a need to design a system which extracts relevant content from the current web and transform into structured format using semantic web technologies so that they can be compatible with existing data in semantic web.

b) **Centralized Information System**

There exist various websites in the current web related to one domain providing same services. For instance, there are many Jobboards such as naukri.com, monster.com, timesjob.com etc. which provides same kind of services to its users and these Jobboards have the same target audience. In the desire to get the best opportunity, Job seeker generally enroll with all possible Jobboards, but handling these profiles is very tedious for user. Therefore, there is a need to design an Information System that aligns data belonging to desperate data sources and provide results at one place. But, only few research efforts have been found where unstructured and semi-structured data has been tagged with the semantic web made metadata. Therefore, there is a requirement to create an information system that uses existing webpages as input, represent them in structured form using ontologies and then aligning [23] various websites of same domain and performing searching operation.

c) **Cross Domain Interoperability**

Interoperability [24] is the method in which two separate systems belonging to different domains takes the services of each other. Two systems become interoperable when they have common interest, understanding and some agreement. For instance, university domain has an interest in job domain because it is university one of the service to provide jobs to its students. Therefore, there is a need to design a system in which information systems are interoperable across domains to provide services of one domain to another domain's users.

1.4 PROBLEM DEFINITION

The overall goal of the proposed work is to build an Information System that first converts semi-structured data available on the current web into structured format belonging to the same domain and providing more relevant results to its users. Not only this, it also provides inter-operability between two domains so that complex queries could be handled.

1.5 OBJECTIVES OF THE PROPOSED WORK

The specific objectives of the present work are as follows:

a) **Creation of vocabulary for annotating data of Jobboards**

To convert semi-structured data into structured form, a way to how that knowledge is to be represented is to be decided.

Proposal: In this work, corresponding to selected website (Jobboard site) ontology is developed which define vocabulary that represent all the relevant entities of those websites in the conceptual form.

b) **To develop a mechanism to mine only relevant webpages from the Jobboards**

The entire URLs of the Jobboards may not be relevant for the information system, so there is a need to filter out the irrelevant webpages from mining.
Proposal: In this work, one technique is constructed which creates the URLs of the webpages which will provide desired data.

c) **Automatic identification of relevant section in webpages for data extraction**

The entire content of the information source may not be relevant for the system, so there is a need to filter out the irrelevant content and focus on relevant content on that webpage.
Proposal: In this work, a mechanism is constructed which extracts only the required data from the webpage.

d) **Conversion of semi-structured data into structured format**

A mechanism needs to be employed which will represent extracted semi-structured data into the structured format.

Proposal: Regarding this, a mechanism is built which transforms extracted semi-structured data from the current web into structured form which is well understood by computer.

e) **Globalization of heterogeneous data sources**

To develop an information system, data extracted from different Jobboards needs to be present at the same place to facilitate the user. Since every Jobboard has its own ontology, therefore, their local ontologies should be aligned to represent them at the global level.

Proposal: In this work, Global indexes are maintained that would be helpful in query planning.

f) **Efficient query processing system**

A query processing system needs to be build that will take keywords from user as query and plan query that can be run on structured content.

Proposal: In this work, a method is proposed that converts user keyword based query into SPARQL language query that is compatible with structured data.

g) **Collaborative search system**

There is a need of an information system in the domain of Jobboard that provides search results by collaborating with individual Jobboards and provides relevant results at one place.

Proposal: In this work, a user interface is provided to user which takes input from the user in the form of keywords and retrieves results from the selected Jobboards at one place.

h) **Cross-domain interoperability**

There is a need to design a platform where user can enter his profile such as educational information, personal information, job preferences etc. initially and then search system provides him the results accordingly.

Proposal: In this work, the job seeker ontology is built which takes required information from user in various domains and then this ontology is mapped with Search system which yields relevant results for the job seeker.

To achieve these objectives, a thorough knowledge of the domain must be collected, based on which ontology will be developed. Ontology development, data extraction, concept matching, alignment between ontologies and search system needs to be developed build the whole information system. To assure the practical implications of the objectives undertaken, the system should support scalability, extensibility, robustness and improvement over the evaluation metrics. An ontology driven information system "Jobology" has been developed that represent data in structured format and yields high performance.

1.6 ORGANIZATION OF THE THESIS

The present thesis is organized into nine chapters and is shown in Fig. 1.1. The content of each chapter is summarized as under:

Chapter II reviews the introduction of current web and semantic web. This chapter discusses the various information retrieval tools, evolution of semantic web and its architecture, its need, the concept of ontologies and research work carried out in semantic search engines.

Chapter III presents the process of ontology development with example and various ontology management techniques such as ontology merging and ontology alignment, ontology integration. Research carried out in the literature for ontology management has also been discussed in detail along with comparative study.

Chapter IV presents the architecture of novel ontology based information system for semantic web. The phases of development of proposed system have been introduced in brief. The methodology formulated for the research has been discussed in this chapter.

Chapter V discusses the ontology development in the domain of job with respect to selected Jobboards and in the domain of Student.

Chapter VI discusses an architecture which converts semi-structured data extracted from current web into structured format. The chapter discusses the architecture of

OntoJobextractor and its components in details with screenshots depicting the intermediate results at various phases.

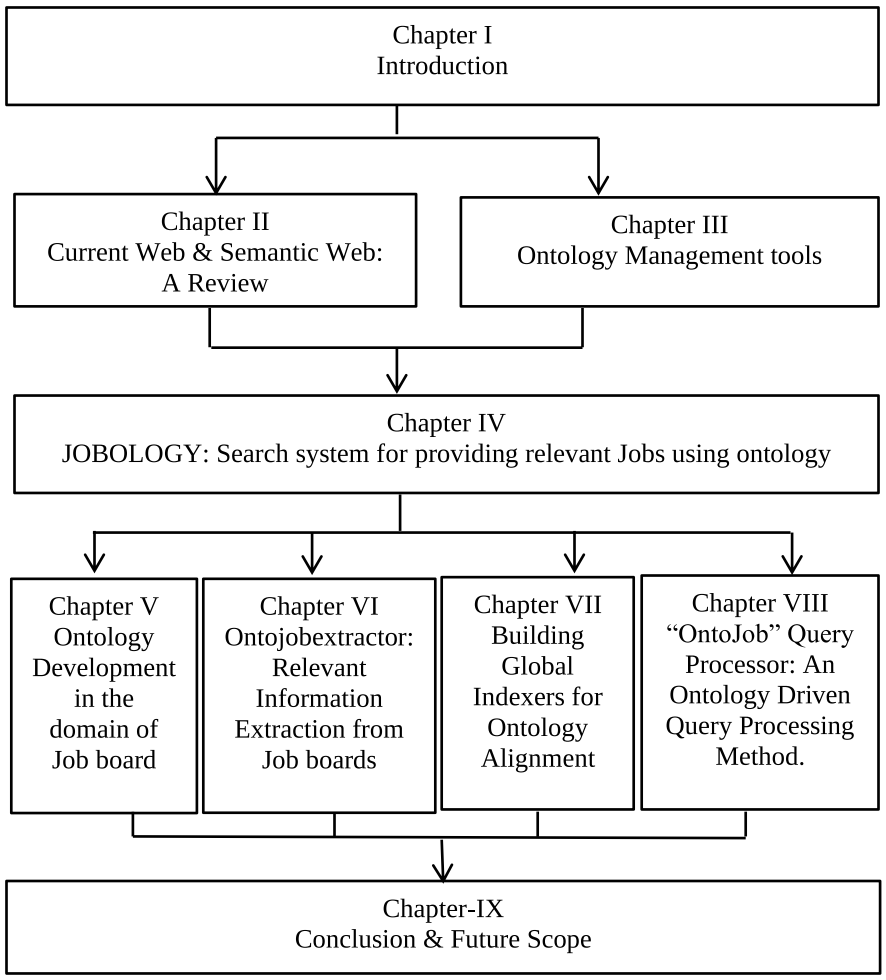

Chapter I
Introduction
Chapter II
Current Web & Semantic Web: A Review
Chapter III
Ontology Management tools
Chapter IV
JOBOLOGY: Search system for providing relevant Jobs using ontology
Chapter V
Ontology Development in the domain of Job board
Chapter VI
Ontojobextractor: Relevant Information Extraction from Job boards
Chapter VII
Building Global Indexers for Ontology Alignment
Chapter VIII
"OntoJob" Query Processor: An Ontology Driven Query Processing Method.
Chapter-IX
Conclusion & Future Scope

Fig. 1.1 Depiction of Flow of outline of the thesis

Chapter VII explains architecture of Ontology alignment between ontologies. Various data structures are defined which are required for building indexes.

Chapter VIII presents architecture of "Ontojob" query processor which builds SPARQL query which will be fired on structured data. It also covers the concept of cross domain interoperability which is proposed in this thesis along with the result analysis.

Chapter IX presents the contributions of the present research and suggestions for future research.

Chapter II

CURRENT WEB & SEMANTIC WEB: A REVIEW

2.1 INTRODUCTION

Internet [2] is the collection of large number of interconnected computers distributed across different geographical location over the world. The evolution of internet started by US department of Defense for the development of ARPANET [25] (Advanced Research Projects Agency Network) project. The initial purpose was to communicate with and share computer resources among mainly scientific users at the connected institutions. The development of TCP/IP [26] protocols in the 1970 made it possible to expand the size of the network.

In 1980, while working at the CERN, the European particle physics laboratory in Geneva, Tim-Berner-Lee wrote a program for storing information using random association. This formed the conceptual basis for the global hypertext project which was later proposed in 1989 as World Wide Web [1]. Tim- Berner-Lee envisioned WWW by proposing the linking of documents over the internet using hypertext. To make WWW executable, he developed the necessary tools such as HTTP [27], a web server, a language to display information which is also known as HTML [28] (Hyper Text Markup Language) and a web browser [29]. With all these tools Web became social. The Web is commonly understood to have had three overlapping phases of development. Under *Web 1.0*, the purpose of search engine such as World Wide Web Worm (WWWW) [30] was purely on determining the size of the web and content relevance was ignored. Because of the limited resources, their indexing and hence searching were limited to the titles and headings found in the web pages. During the phase of *Web 2.0* [31], with the exponential growth in the quality and complexity of information sources on the internet, IR systems evolved from a simple concern with the storage and distribution of artefacts to encompass a broader concern with the transfer of meaningful information. Over the last many years, many efforts are being put to deal with this complexity effectively and efficiently. Finding information from such a large information collection is unprecedently a very tough task. However, various IR tools [32] such as Search engines, Web directories, OPAC (Online Public Access Catalogue), online database, digital library and Web portals are available via the internet.

The upcoming sections present a meticulous study of current web. A portrayal of semantic web is provided in the subsequent sections.

2.2 INFORMATION RETRIEVAL TOOLS

The workings of IR tools are explained briefly as follows:

a) Search Engines

A Search Engine [10] is a program designed to search for information on the WWW. The search results presented in a list consist of web pages, images, information and other types of files. The architecture of a general search engine contains a front-end process and a back-end process as shown in Fig. 2.1.

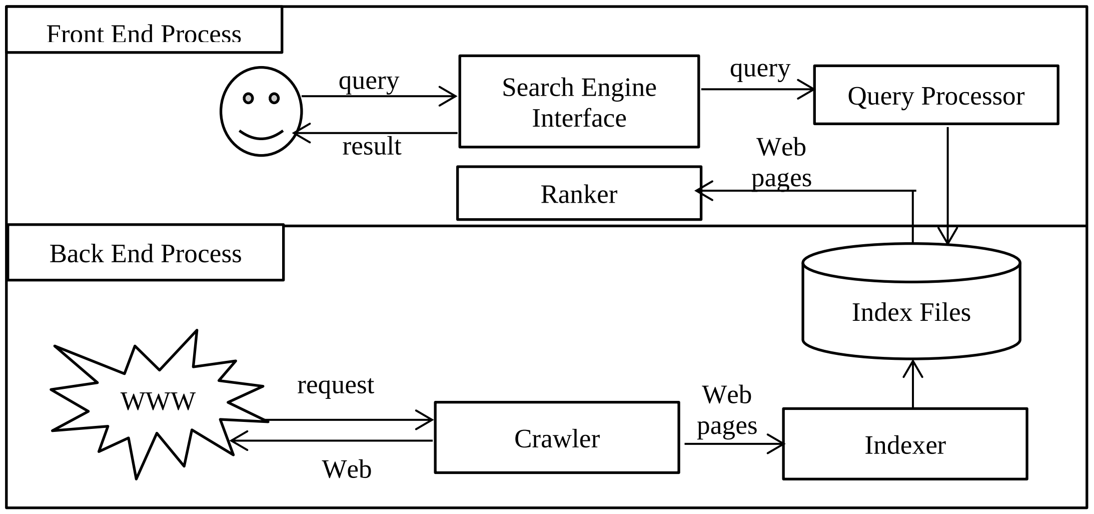


Fig. 2.1 Architecture of Search Engine

In the front-end side, user submits the search query to the search engine interface. The query processor then parses the search request into a form that the search engine can understand, and then it executes the search operation on the index files [10, 33]. After ranking [10, 34], the search results are returned to the user. In the back-end, the crawler [10, 35] module (spider or robot) fetches the web pages from the Web; the indexing subsystem parses those Web pages and stores them into the index files.

b) Web Directory

A Web Directory [35] organizes Web sites by subject, and is usually maintained by humans instead of software. The searcher looks at sites organized in a series of categories and menus. It does not display results in the form of web pages based on keywords rather results of directory are in the form of links that contains category and sub categories. The database size of directory is smaller as compared to engine's database; it is human-sited directory and not crawled by crawlers.

c) Digital Library

A digital library [36] also named as digital repository or digital collection is an online database of digital objects that can includes text, images, audio, video or digital media formats. It provides high quality resources that have been filtered by library professional and subject experts and added manually. For example; *American memory* is a digital library within the library of congress.

d) Online databases

These databases provide access to remote databases through a database vendor or service provider. For example; *Elsevier, IEEE, ACM* etc. are some examples of online databases.

e) OPAC (Online Public Access Catalogue)

It is a computerized catalogue [37] containing bibliographic records of items in a library. *Medline, ERIC, PsyCINFO* item are some catalogues that index journal articles and other research data

f) Meta-Search Engines

A Meta-Search engine [9, 38] performs a search by calling on more than one search engine to do the actual work. The general architecture of Meta-Search engine is shown in Fig. 2.2 where it sends user requests to several other search engines and/or databases and aggregates the results into a single list and displays them to their source. Meta-Search Engines enable users to enter search criteria once and access several search engines simultaneously. Meta-Search engines operate on the premise that the Web is too large for any one search-engine to index it all and that more comprehensive search results can be obtained by combining the results from several search engines.

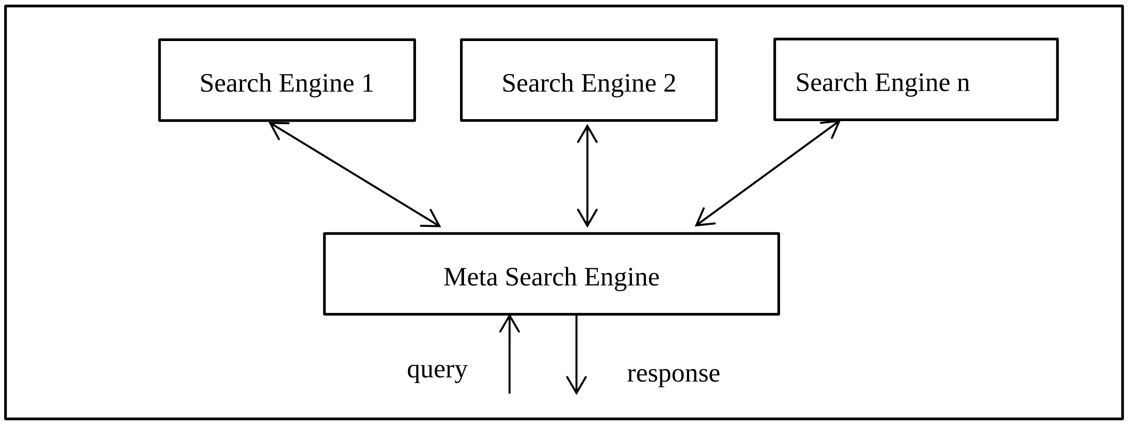


Fig. 2.2 Architecture of Meta-Search Engine

2.3 PROBLEM WITH CURRENT INFORMATION RETRIEVAL MODELS

Despite of the fact that WWW contains a lot of information and knowledge, search engines usually serve only to deliver and present the content of documents describing the knowledge. Apart from this, there exist other problems [39] that users are suffering from, which are discussed as follows:

- Current search engines are unable to provide direct answers to queries.
- Current search engines process queries based on keywords. Thus, they retrieve all web pages containing those keywords without considering the fact that an accurate answer is produced on the basis of user's context.
- Current search engines are unable to gather complex information.
- Current WWW contains a lot of information and knowledge, but current search engines are unable to retrieve complex information. For instance, user fires a query "find 10 engineering college for computer stream in India and the top computer companies in their proximity". Current search engines would not be able to yield desired results. For the results, user has to separately fire the query and manually merge the results.
- Current Search Engines are handicapped by being unable to figure out the context in which a word is being used.
- Although the search engines are very helpful in finding information on the Internet and are getting smarter with the passage of time, but they lack in finding the meanings

of the terms, expressions used in the Web pages and the relationships between them. The problem comes due to the existence of words which have many meanings also known as *polysemy* [40] and several words having same meaning also known as *synonymy* [40] in natura1 languages. Thus, when a user gives a search query like "Flip-Flop" to find the definition of "Flip-Flop" in Computer Science domain, the most accredited search engine, Google, is unable to find the right document (no document is relevant among the top ten results returned). This is because, Google does not know which Flip-Flop the user is talking about; a kind of female shoes, or a device for Electronics which is used for storing one bit memory storage. It was possible for Google to find the right document only if it knew the relationship between the two terms given to it; "Flip-Flop" and "Electronics".

To deal with such problem, Tim Berners-Lee, Hendler and Lassila presented a vision of a Web in which information is given well-defined meaning, better enabling computers to understand the meaning of content and help people to provide relevant information which is called Semantic Web [14]. The detail of Semantic web is presented in the next section.

2.4 INTRODUCTION TO SEMANTIC WEB

Tim-Berner-Lee, inventor of WWW and director of W3C envisioned about Semantic Web. The goal of semantic Web [14] is to represent data in structured format which would help machines to understand more information on the web which supports in richer discovery and data integration from different sources via linking hereby producing more exact results to the user as compared to current web search engines.

- **Architecture of Semantic Web**

  Semantic Web is the new generation Web that tries to represent information such that it can be used by machines, not just for display purposes, but for automation, integration, and reuse across applications. The architecture of semantic Web [41] (W3C) is shown in Fig. 2.3. The semantic Web technologies offer a new approach to managing information and processes, the fundamental principle of which is the creation and use of semantic metadata. All layers of semantic web are explained in detail as below:

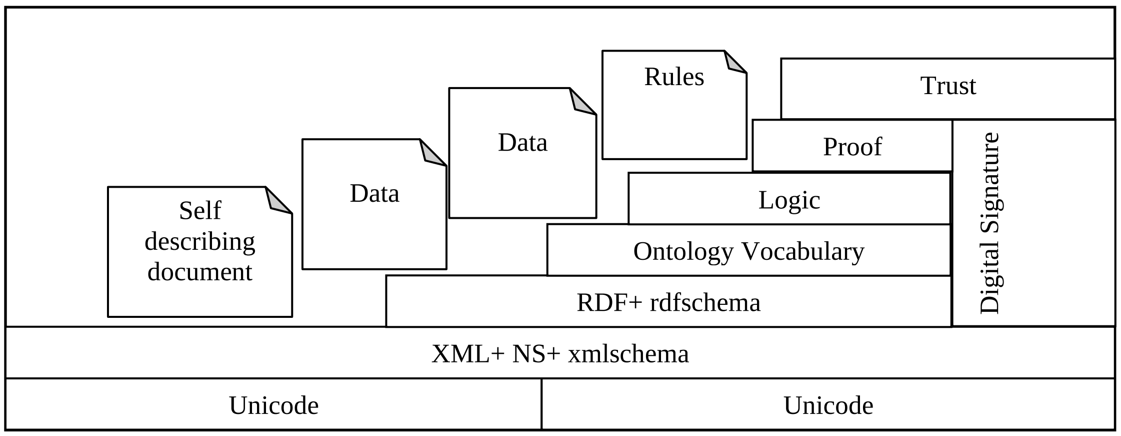


Fig. 2.3 Architecture of Semantic Web

a) URI

A Universal Resource Identifier (URI) [14] is a formatted string that serves as a means of identifying abstract or physical resource. A URI can be further classified as a locator, a name, or both. Uniform resource locator (URL) refers to the subset of URI that identifies resources via a representation of their primary access mechanism. A uniform resource name (URN) refers to the subset of URI that is required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

b) Unicode

Unicode provides a unique number for every character, independently of the underlying platform, program, or language.

c) XML and XML Namespace

XML (eXtensible Markup Language) [16] with XML namespace and XML schema definitions makes sure that there is a common syntax used in the semantic web. XML namespaces allow specifying different markup vocabularies in one XML document. XML schema serves for expressing schema definition of a XML document.

d) RDF and RDF Schema

On top of XML, is the Resource Description Framework (RDF) [42], for representing information about resources in a graph form. RDF is based on triples, resource-predicate-object. RDF Schema (RDFS) [43] defines the vocabulary of RDF model. It provides a mechanism to describe domain-specific properties and classes of resources to which those properties can be applied, using a set of basic modeling primitives (class, subclass-of, property, subproperty-of, domain, range, type). However, RDFS is rather simple and it still does not provide exact semantics of a domain.

e) **Ontology**

Ontology [19] comprises a set of knowledge terms, including the vocabulary, the semantic interconnections, simple rules of inference and logic for some topic. Ontologies applied to the Web are creating the semantic Web. Ontologies facilitate knowledge sharing [44] and provide reusable Web contents, Web services [45], and applications. Few of the ontology languages are DAML (DARPA Agent Markup Language) [46], OIL (Ontology Interference Layer) [47, 48] and OWL (Web Ontology Language) [19]. OWL is developed starting from description logic and DAML+OIL [46]. OWL is a set of XML elements and attributes, with well-defined meaning, that are used to define terms and their relationships (e.g. Class, equivalentProperty, intersectionOf, unionOf, etc.). OWL elements extend the set of RDF and RDFS elements, and the OWL namespace is used to denote OWL encoding.

f) **Logic, Proof, Trust and Digital Signature**

The logic layer [13, 49] is used to enhance the ontology language further and to allow the writing of application specific declarative knowledge. The proof layer involves the actual deductive process as well as the representation of proofs in Web languages and proof validation. Finally, the Trust layer will emerge using digital signatures [13] and other kinds of knowledge, based on recommendations by trusted agents or on rating and certification agencies and consumer bodies.

For the semantic Web to become more expressive enough to help in a wide range of situations, it will become necessary to construct a powerful logic language for making inferences. The next step in the architecture is ‘Trust’ and ‘Proof’. Trust and Proof is mainly concerned with two principles. First, the original source does make a statement (proof) and second, the source should be trustworthy (trust). Proof will be achieved on the Semantic Web by one or more different methods. Digital signatures are envisioned to play an important role in proof. In the next section ontologies and their role in the creation of the Semantic Web are discussed in detail.

2.5 ONTOLOGY

The word ontology is employed in the field of AI research, as it is useful to make the conceptualizations [50] of a domain explicit which enables their comparison and analyzes. Several definitions have been given by different researchers which are defined as below:

a) As per Gruber in 1993, Ontology is a formal, explicit specification of a shared conceptualization [51].

b) Fensel in 2001, defined Ontology as an abstract model of a phenomenon termed as "conceptualization", a precise mathematical description hints the word "formal", the precision of concepts and their relationships are expressed by the term "explicit"' and the existence of an agreement between ontology users is hinted by the term "shared" [52].

c) Russell & Norving in 1995 established that Ontology is a formal description of the concepts and relations which can exist in a community of agents [53].

d) Swartout et al. in 1996 defined Ontology as a hierarchically structured set of terms to describe a domain that can be used as a skeletal foundation for a knowledge base [54].

e) As per Noy & McGuinness in 2001, defined Ontology is a formal explicit representation of concepts in a domain, properties of each concept describe characteristics and attributes of the concept known as slots and constrains on these slots. Sometimes concepts are termed as classes, properties are also known as roles while facets are used rather than slots [55].

f) Fonseca et al. in 2002 defined Ontology as a theory which uses a specific vocabulary to describe entities, classes, properties and related functions with certain point of view [56].

g) As per Starlab in 2003; Ontology includes a specification of the terms used, ("terminology") and agreements to determine the meaning of these terms, along with the relationships between them [57].

From these definitions, some essential aspects of ontologies are identified such as:

- Ontologies are used to describe a specific domain. − The terms and relations are clearly defined in that domain.
- There is a mechanism to organize the terms, (commonly a hierarchical structure is used as well as IS−A or HAS−A relationships).
- There is an agreement between users of ontology in such a way the meaning of the terms is used consistently.

2.5.1 Main Functions of Ontologies

There are various functions of ontologies [58, 59] which are discussed as below:

a) Ontologies can be used to support a great variety of tasks in diverse research areas such as knowledge representation, natural language processing, information retrieval, databases, knowledge management, on line database integration, digital libraries, geographic information systems, visual information retrieval or multi agent systems.
b) Ontology provides meta-information which describes data semantics.
c) Ontologies enable shared knowledge and reuse where information resources can be communicated between human or software agents.
d) Semantically relationships in ontologies are machine readable, in such a way they enable making statements and asking queries about a subject domain due to the use of a conceptualization, which describes entities and their relationships. This conceptualization enables that software agent of a vocabulary to represent and to communicate knowledge. The usefulness of ontologies in agent based systems can be briefly summarized as they enable knowledge level interoperation.
e) In research areas, ontologies support shared understanding, interoperability between tools, systems engineering, reusability and declarative specification.
f) Ontologies are used to build knowledge bases.
g) Ontologies are able to operate as repositories to organize information for specific communities. They are used as a tool for knowledge acquisition, (teamwork can use ontologies as a common support to classify the knowledge of an organization).
h) Ontologies allow users to reuse knowledge in new systems. They can form a base to construct knowledge representation languages. Semantic integration of heterogeneous information sources such as digital libraries can benefit with the

incorporation of ontologies. Some applications use domain ontology to integrate information resources and others allow each resource to use its own ontology. Each user can also have his own ontology as per his/her interests, language or role in a determine domain.

i) Ontologies provide a source of precisely defined terms. In information retrieval applications, ontologies serve to disambiguate user queries, to elaborate taxonomies of terms or thesaurus to enhance the quality of retrieved results. Machine−learning techniques are also used to extend ontologies based on user's interactions.

2.5.2 Reasons for Developing Ontology

Ontology is the most important component of semantic web which is used to represent domain knowledge. According to Noy & McGuinnes; following reasons have been identified for the development of ontology [55]:

a) To share common understanding of the structure of information between people or software agents

Ontologies enable the concepts to be defined in a way that can be shared by people or agents. For example, if several websites contain information about a product and these websites shares the same ontology then agent must be able to aggregate the information about the product from the different sites and present it to the user or any required application.

b) To enable reuse of domain knowledge

To design ontology from scratch is a tedious and time consuming task. Hence, ontology defined for domain must be designed to cover the concepts so that it can be reused/ extended by some application rather than creating. This created ontology can be shared by keeping them in an ontology repository.

c) To make domain assumptions explicit

Explicit specification for domain knowledge makes it easy to change the assumption if the knowledge of that domain changes. It easily allows a new user to understand the domain terms easily.

d) To separate domain knowledge from operational knowledge

It is a better idea to separate operational knowledge from the domain knowledge from the knowledge management perspective because it leads to an inefficient system, such a design hinders knowledge engineer's ability to express deeper relationship among knowledge items [60].

e) To analyze domain knowledge

Ontologies are used to explain a domain completely with concepts, properties and relations that exists between them. Such a formal specification helps in analyzing a domain explicitly and allows knowledge reuse.

2.5.3 Kinds of Ontology

Ontologies are categorized into different kinds based on formality of the language or the level of dependence on a task or point of view.

a) Top level ontology [61]

It describes general concepts like space, time, matter, object, event or action, which do not depend on a problem or domain. However, the development of general enough top level ontology has not been accomplished yet.

b) Domain ontologies and task ontologies [62]

They describe the vocabulary for a generic domain (like biology or medicine), a task or activity (such as selling) by means of specialized terms.

c) Application ontologies [63]

They describe concepts which depend on a domain and task. The concepts respond to roles played by domain entities while performing certain task.

By knowing the kind of ontology according to a particular classification, it is useful to lead to the ontology building process.

2.6 LANGUAGES TO SUPPORT ONTOLOGY MANAGEMENT

There are various languages in which ontology can be specified. The language specifies the formal semantics of a language. The language adds the expressiveness to the representation of knowledge allowing the inferences and the reasoning support making the semantics of the language machine- accessible. The details of various ontological modeling languages are given as below:

a) KIF [64]

KIF short for *Knowledge Interchange Format*, is a language based on first order logic created in 1992 as an interchange format for diverse knowledge related systems. It was created by Michael Genesereth, Richard Fikes and others participating in the DARPA knowledge Sharing Effort. KIF has a declarative semantics. It is meant to describe facts about the world rather than processes or procedures. Knowledge can be described as objects, functions, relations, and rules. It is a formal language, i.e. it can express arbitrary statements in first order logic [47] and can support reasoners [65] that can prove the consistency of a set of KIF statements. KIF also supports non-monotonic reasoning.

b) Loom [66]

Loom is a knowledge representation language implemented by researchers in the AI research group at the University of Southern California’s Information Sciences Institute. Loom is not designed for implementing Ontologies, but for general KBs. It is developed based on DLs and production rules, and offers automatic classifications of concepts.

c) OCML [67]

OCML short for *Options Configuration Modeling Language*, was created in 1993 at the Open University KMI as a kind of ‘‘Operational Ontolingua’’. Indeed, most of the definitions that can express in OCML are analogous to the corresponding definitions in Ontolingua. OCML was constructed for developing executable Ontologies and models in problem solving methods.

d) FLogic [68]

FLogic short for *Frame Logic*, merges frames and first order logic, to allow concepts, Concept Taxonomies, Functions, Binary Relations, Instances, Axioms and Deductive rules representation. Ontobroker [69] can be used underlying FLogic based inference engine to check constraint and deduce new information.

e) SHOE [70]

SHOE was built in 1996 as a *Simple Html Ontology Extension* allowing web page authors the annotation of their web pages with machine-readable knowledge. SHOE makes the possibility for the agents to gather meaningful information about Web pages and Documents, which improves search mechanisms, and knowledge gathering. Moreover, SHOE combines Markup Languages, Knowledge Representation, Datalog and Ontologies features aiming to address the unique problems of the semantics on the Web.

f) OML [71]

OML Short for *Ontology Markup Language*, OML was initially developed at the University of Washington, and partially based on SHOE. It was initially considered an XML serialization of SHOE [70]. Additionally, OML forms a subset of CKML (Conceptual Markup Language) that allows rich knowledge representation capabilities.

g) XML [15, 16]

It is a W3C recommendation stands for *Extensible Markup Language*, was built in 1996 much like HTML and designed to describe data and not to display data. As an effect, XML has been used to modify SHOE syntax and subsequently, additional ontology languages were built on the XML syntax.

h) XOL [72]

XOL short for *Ontology Exchange Language* was developed by the AI center of SRI International, in 1999. It is designed by the US bioinformatics community and based on XML language. Any tool is allocated for the development of Ontologies using XOL. Although, based on syntax of XML, one can use an XML editor to author XOL files.

i) RDF [15, 16, 17]

RDF stands for *Resource Description Framework*, was developed by the W3C to describe Web resources. It is based on representing resource using *Subject-Predicate-Object* known as triple. The *subject* denotes the resource; *predicate* denotes traits or aspects of the resource and expresses the relationship between the subject and object. For example, to represent the notion "color of the apple is red" in RDF is as the triple: a subject denoting "apple", predicate denoting "color" and object denoting "red". The structure of any expression in RDF is a collection of triples, each consisting of a subject, predicate and an object. A set of such triples is called RDF graph. It is visualized as shown in Fig. 2.4.

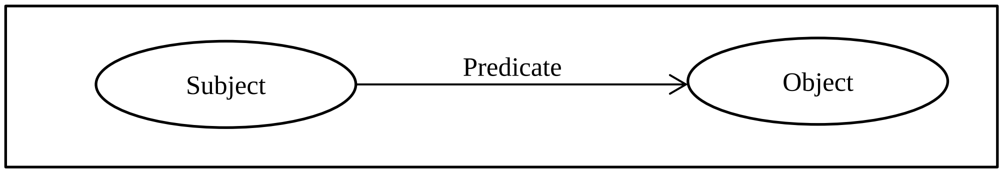


Fig. 2.4 RDF graph

The vocabulary defined by RDF specification is shown in Table 2.1.

Table 2.1 RDF Vocabulary

S.No.	RDF Vocabulary	Description
1	rdf: Literal	the class of XML literal specification
2	rdf: property	the class of properties
3	rdf: statement	the class of XML statements
4	rdf: Alt	container of alternatives
5	rdf: Bag	unordered container
6	rdf: seq	ordered container
7	rdf: list	the class of RDF lists
8	rdf: type	an instance of rdf: property used to state that a resource is an instance of a class.
9	rdf: first	he first item in the subject RDF list.
10	rdf: rest	the rest of the subject RDF list after rdf: first.
11	rdf:value	idiomatic property used for structured values.
12	rdf: subject	the subject of the subject RDF statement.
13	rdf: predicate	the predicate of the subject RDF statement.
14	rdf: object	the object of the subject RDF statement.

j) RDFS [73]

RDFS stands for *RDF Schema* and was built by the W3C as an extension to RDF with Frame- based Primitives. It is a semantic extension of RDF. It provides mechanism for describing groups of related resources and the relationships between the resources. These resources are used to determine characteristics of other resources such as domain and range of properties. The vocabulary defined by RDFS specification is defined in Table 2.2.

Table 2.2 RDFS Vocabulary

S.No.	Constructs	Description
1	rdfs:resource	the class of XML literal specification
2	rdf:class	the class of properties
3	rdf: literal	the class of XML statements
4	rdfs: datatype	container of alternatives
5	rdf: langstring	unordered container
6	rdf:HTML	ordered container
7	rdf: property	the class of RDF lists
8	rdfs: range	It is an instance of rdf: property.
9	rdfs: domain	It is an instance of rdf: property.
10	rdfs: subclassof	it is an instance of rdf: property that is used to state that all the instances of one class are instances of other.
11	rdf: subpropoertyof	it is an instance of rdf: property that is used to state that all resources related to one property are also related to other.
12	rdfs: label	it is an instance of rdf: property that may be used to provide a human readable version of a resource's name
13	rdfs: comment	it is an instance of rdf: propoerty that may be used to provide a human readable description of a resource.

k) RDFa [74]

It stands for *Resource Description Framework in Attributes*. It is a W3C Recommendation that adds a set of attributes-level extension to HTML, XHTML [75] and various XML based documents types for embedding rich metadata within web documents. The following example as shown in Fig. 2.5 Example of RDF shows the addition of Dublin Core metadata [76] to an XML element in an XHTML file.

```
<div xmlns: dc= “http:purl.org/dc/elements/1.1/”
About “http:www.example.com/books/test”>
<span property= “dc:title”> Data structures </span>
<span property= “dc:creator”> Dr. A.K. Sharma </span>
<span property= “dc:title”> 2010-01-01 </span>
```

Fig. 2.5 Example of RDFa

l) OIL [77]

OIL stands for *Ontology Interchange Language* and is based on RDF and RDFs which supports a well-defined semantic vocabulary and reasoning constructs for ontology development. OIL was developed as a research product of European Union Project (EU). It included following aspects:

- A more interactive choice of the modeling primitives and richer ways to define concepts and attributes.
- The definition of a formal semantic for OIL.
- The development of customized editors to inference engines to work with OIL.

It is frame based system which provides a context for modeling one aspect of a domain. In OIL, knowledge is represented via Description Logic which describes knowledge in terms of concepts and role restrictions that can automatically derive classification taxonomies. It is based on the web standards of W3C that has syntax of XML, RDF and RDFS. Knowing the fact that a single ontology language cannot fulfill all the needs of semantic web’s large range of applications, OIL has organized a series of ever increasing layers of sublanguages. Each additional layer adds the functionality and complexity of previous one. Consider a simple example of ontology defined in OIL language specification. The OIL expression shown in Fig. 2.6 defines *Herbivore* as a class, which is a subclass of animal and disjoint of all carnivores.

It encounters that herbivore is a subclass of animal and a subclass of a second class which it cannot understand properly. This seems to preserve complicated semantics for simple applications.

```
<rdfs:Class rdf:ID= “herbivore”>
<rdf:type            rdf:resource=            “http:www.ontoknowledge.org/oil/RDFS-
Schema/#DefinedClass”/>
<rdfs:subclassof rdf:resource=”#animal”/>
<rdfs:subclassof>
<OIL:NOT>
<OIL:hasoperand rdf:resource= “# carnivore/”>
</OIL:NOT>
</rdfs:subclassof>
</<rdfs:Class >
```

Fig. 2.6 Example in OIL Language Format

m) DAML [78]

DAML is short form for *DARPA Markup Language*. It is a semantic markup language that is specifically an extension to ML and the RDF. It is used for the U.S. Defense Advanced Research Project Agency (DARPA) and compared to the XML standard it offers a better capacity to express semantics which means a much higher level of interoperability between websites. Certain language constructs of DAML are defined in Table 2.3 and example in DAML language is shown in Fig. 2.7.

Table 2.3 DAML Language Constructs

S.No.	Construct	Description
1	Daml:restriction with daml:onproperty	specifies a slot being restricted on the property specified.
2	Daml:intersectioOf	disjunction of class expression
3	Daml:unionOf	conjunction of class expression
4	Daml:complementOf	negation of class expression
5	Daml:mincardinality	minimum cardinality constraint on a property
6	Daml:maxcardinality	maximum cardinality constraint on a property
7	Daml:transitiveproperty	specifying the transitive property
8	Daml:inverseOf	specifying the inverse property

The example shown in Fig. 2.7 describes ontology in DAML for a class *child* which is subclass of class *person*. It specifies that the *child* can have one *mother* by limiting the cardinality on property *#hasmother* to one. Property *#hasparent* has cardinality two, which is specified by <*daml:unionOf*> construct, specified with class mother and father.

```
<daml:Class rdf:ID= “child”>
<daml:subClassof rdf:resource= “#person”> <daml :restriction>
<daml:onProperty rdf:resource= “#hasMother”/>
<daml:cardinality> 1</ daml:cardinality >
</daml :restriction></daml:subClassof>
<daml:subClassof>
<daml :restriction maxcardinality= “2”>
<daml:onProperty rdf:resource= “#hasParents”/>
<daml:cardinality> 2</ daml:cardinality > <daml:Class>
<daml:unionOf rdf:parseType= “daml:collection”>
<daml:Class rdf:about= “Father”/>
<daml:Class rdf:about= “Mother”/>
</daml:Class </daml :restriction >  </daml:subClassof>
</daml:Class>
```

Fig. 2.7 Ontology in DAML

n) DAML+OIL [46]

DAML+OIL is the result of merging DAML-ONT (an early result of the DARPA Agent Markup Language (DAML) Program) and OIL (the Ontology Inference Layer), developed by a group of largely European researchers, several of whom were members of the European-funded On-To- knowledge consortium. As it is an ontology language, DAML+OIL is designed to describe the *structure* of a domain. DAML+OIL takes an object-oriented approach, with the structure of the domain being described in terms of *classes* and *properties*. DAML+ OIL languages allow Concepts, Taxonomies, Functions, Binary Relations and Instances representation. The tools that can author DAML+OIL Ontologies are OILEd [79], OntoEdit [80], Protégé2000 [81] and WebODE [82].

o) OWL [83]

OWL stands for Web Ontology Language, created in 2001 by a working group formed by W3C. It has emerged from DAML+OIL language on the recommendation of W3C. This language provides more vocabulary for describing properties and classes among others, relation between classes (e.g.; disjointness), cardinality (e.g.; “exactly one”), equality, richer typing of properties, characteristics of properties (e.g.; symmetry, transitive etc.) and enumerated classes.

OWL provides three increasable expressive sublanguages [19, 83]: OWL Lite, OWL DL, OWL Full.

- **OWL Lite:** It supports those users who are looking for classification hierarchy and simple constraints. For example; while it supports cardinality constraints, it only permits cardinality values of 0 and 1. It has a lower formal complexity than OWL DL.
- **OWL DL:** It is more expressive than OWL- Lite. It includes all OWL languages constructs, but they can be used only under restriction. It is named due to its correspondence with description logic, a field of research that has studied the logics that form the formal foundation of OWL.
- **OWL Full:** It has the maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. It allows ontology to augment the meaning of the pre-defined vocabulary.

In the example as shown in Fig. 2.8, OWL ontology with three plant classes are defined. The *flowering plants* class and *shrubs* class are both subclasses of the *planttype* class.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:plants="http://www.linkeddatatools.com/plants#">
<owl:Class rdf:about="http://www.linkeddatatools.com/plants#planttype">
<rdfs:label>The plant type</rdfs:label>
<rdfs:comment>The class of all plant types.</rdfs:comment> </owl:Class>
<owl:Class rdf:about="http://www.linkeddatatools.com/plants#flowers">
<rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#planttype"/>
<rdfs:label>Flowering plants</rdfs:label>
<rdfs:comment>Flowering plants, also known as angiosperms.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="http://www.linkeddatatools.com/plants#shrubs">
<rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#planttype"/>
<rdfs:label>Shrubbery</rdfs:label>
<rdfs:comment>Shrubs, a plant which branches from the base.</rdfs:comment>
</owl:Class>
<rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">
<rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>
</rdf:Description>
</rdf:RDF>
```

Fig. 2.8 Ontology in OWL Language

The quality and correctness of ontologies play a vital role in semantic representation and knowledge sharing. To ensure the quality of ontologies, there is a need for dealing with the inconsistency and uncertainty in the ontologies of real-world assumptions. To deal with this, in the next section semantic reasoners are discussed.

2.7 SEMANTIC REASONERS

A *semantic reasoner* [84], *reasoning engine, rules engine,* or simply a *reasoner*, is a piece of software to infer logical consequences from a set of asserted facts or axioms. Reasoner mainly deals with the inconsistency and uncertainty in the constructed ontology.

- Inconsistent ontology [85] means that an error or a conflict exist in ontology, because of which some concepts in the ontology cannot be interpreted correctly. The inconsistency results in false semantic understanding and knowledge representation.
- An uncertain ontology [86] means that the correctness of the ontology is probabilistic. Ontology reasoning reduces the redundancy of information in knowledge base and finds the conflicts in knowledge content. There are some examples of reasoners which are used widely. They are explained as below in Table 2.4.

Table 2.4 Semantic Reasoners

S. No.	**Reasoner**	**Free/ Licensed**	**Language built on**	**Supported Interface**	**Services**	**Supported Syntax**	**Supported ontologies**
1	**Fact++ (Univ. of Manchester) [87]**	Free	C++	Protégé, Command Line, OWL API	realisation, classification, satisfiability, entailment, consistency		OWL2DL
2	**Hermit (Univ. of Oxford) [88]**	Free	Java	Protégé, Command Line, OWL API	realisation, classification, satisfiability, entailment, consistency	All OWLAPI	OWL2DL
3	**DBOWL (Univ. of Malaga) [89]**	Licensed			classification, satisfiability, conjunctive query answering, consistency	RDF/XML	OWL
4	**Jfact (Univ. of Manchester) [90]**		Java	Protégé	realisation, classification, satisfiability, entailment, consistency		

5	**Ontop (Univ. of Bozen Bolzano) [91]**			Protégé, OWLAPI	conjunctive query answering, realization	All APIs	virtual RDF graph using SPARQL
6	**Pellet (Clark & Persia) [92, 93]**	Free	Java	Jena, Protégé, Command Line, OWLAPI	realisation, classification, satisfiability, conjunctive query answering, entailment, consistency, explanation	Turtle, RDF/XML, Krss2, OWL/XML , functional Manchester	OWL2 and SWRL
7	**Racer (Concordia Univ., Canada; Univ. of Lubeck, Germany) [94, 95]**			OWLLink, Protégé, Command Line, OWLAPI	realisation, classification, satisfiability, conjunctive query answering, entailment, consistency, explanation	RDF/XML, OWL/XML , functional, All OWL APIs	

In the next section, various ontology development tools are discussed which will be required for the construction of ontology.

2.8 ONTOLOGY DEVELOPMENT TOOLS

Several software tools related to Ontologies have been proposed by researchers in Semantic web. Especially, there exist significant attention accorded to Semantic web editors (responsible to the creation and manipulation of Ontologies). Some of these tools are explained as below:

a) OntoEdit

OntoEdit [80] is an *Ontology Editor* integrating various aspects of ontology engineering. OntoEdit is quite exceptional in its category since it is based on a modern method for ontology development and because it makes comprehensive use of inference.

b) Protégé

Protégé [81] is an ontology editor created at Stanford University and is very popular in the field of Semantic Web and the level of computer science research. Protégé is free, developed in Java and its source code is released under a free license (the Mozilla

Public License). Protégé can read and save ontologies in the ontologies formats: RDF, RDFS, OWL, etc. It is recognized for its ability to work on large Ontologies.

c) OILEd

OIL Editor [79] (OilEd) is a simple ontology editor that supports OIL-based Ontologies construction. The basic design has been deeply influenced by similar tools such as Protégé and OntoEdit, but OilEd has extended these approaches in several manners, especially using an extension of expressive power and a reasoner. OilEd supports the construction of OIL based Ontologies as an ontology editor.

d) Ontolingua

The Ontolingua [58] is an ontology tool created for Knowledge System Laboratory at Stanford University. Ontolingua is devoted for Ontologies development using a form-based Web interface. The ontology editor of Ontolingua is a tool supporting distributed, browsing, collaborative editing and Ontologies creation. Using Ontolingua, it is possible to export or import the following formats: KIF [24], DAML+ OIL [23], OKBC [96], LOOM [66], Ontolingua and CLIPS (C Language Integrated Production System) [97]. Additionally, it is also possible to only import Classic Ocelot and Protégé format, but not their export.

e) WebODE

WebODE [82], described in the Ontological Engineering Group webpage, was built as a Scalable, Extensible, Integrated workbench that covers and gave support to most of the activities involved in the ontology development process (conceptualization, reasoning, exchange, etc.) and supplied a comprehensive set of ontology related services that permit interoperation with other information systems. WebODE exports to WebODE's XML, RDF(S), Prolog, OIL, Java/Jess, DAML+OIL, and OWL, and imports from WebODE's XML, RDF(S), UML, X-CARIN and OWL.

f) WebOnto

WebOnto [98] is a tool which provides a web-based visualization, browsing and editing support to develop and maintain Ontologies and knowledge models specified in OCML [67]. An ontology can be viewed as a model of the conceptual structure of some domain and WebOnto provides the capability to represent this graphically.

g) SWOOP

SWOOP [99] short for *Semantic Web Ontology Editor*. It is a tool for creating, editing, and debugging OWL Ontologies. It was produced by the MIND lab at University of Maryland, College Park, but is now an open source project with contributors from all over the world.

h) TopBraid Composer

The Free Edition (FE) of Top-Braid Composer [100] is a professional tool for ontologies development. It uses the Eclipse platform [101] and the Jena API [102]. TopBraid Composer is a complete editor for RDF(S) and OWL models; additionally, it is a platform for other RDF-based components and services.

The comparative analysis of the above discussed development tools has been shown in Table 2.5. The comparison is done based on release date, base language, whether the tool is freely available or licensed, and whether they use any ontology library.

Table 2.5 Comparative study of Ontology Development Tools

S.No.	**Tool**	**Release Date**	**Base Language**	**Availability**	**Ontology Library**
1	**Ontoedit**	2004	F-Logic	Free	No
2	**OILEd**	2003	DAML+OIL	Free	Yes
3	**Protégé**	2004	OKBC+CLOS based meta-data	Free	Yes
4	**Ontolingua**	2001	Ontolingua	Free	Yes
5	**WebODE**	2002	HTML form & Java applet	Free	No
6	**WebOnto**	2001	OCML	Free	Yes
7	**SWOOP**	2007	OWL	Free	No
8	**Topbraid Composer**	2011	RDFS/OWL	License	Yes

Among these tools, protégé is the most widely used tool for ontology development because of the plug-ins and the features it supports. The next section discusses about the rule languages with inferential capabilities for ontologies.

2.9 ONTOLOGY RULE LANGUAGES

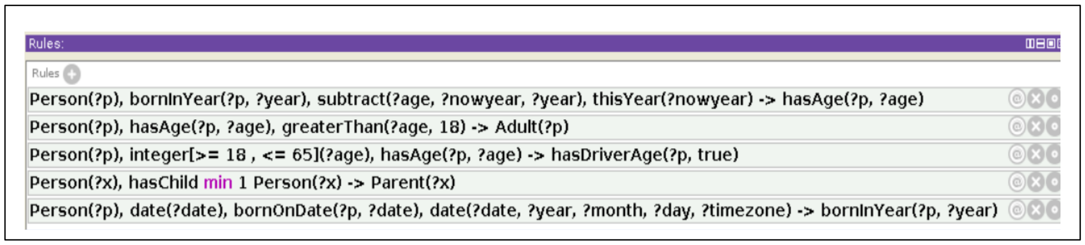
Ontologies are the mechanism for knowledge representation which can be specified by using different languages like RDF, RDF Schema, OWL etc. These languages offer a wide variety of expressiveness constructs to represent a domain. The classes, properties, property restrictions can be easily implemented using these languages. For inferential capability, that is to deduce new facts from the knowledge base. Various rule languages are used on these languages. The various rule languages [103] which are widely used for inference mechanism are:

a) SWRL (Semantic Web Rule Language)

The Semantic Web Rule Language (SWRL) [104, 105] is a language for the Semantic Web that can be used to express rules as well as logic. The specification was submitted in May 2004 to the W3C by the National Research Council of Canada, Network Inference (since acquired by web Methods), and Stanford University in association with the Joint US/EU ad hoc Agent Markup Language Committee. SWRL allows users to write Horn-like rules that can be expressed in terms of OWL concepts and that can reason about OWL individuals. SWRL rules are of the form antecedent-consequent pair where antecedent is referred to as body part of the rule and consequent refers to the head part of the rule. The head and body part of the rule may be conjunctions of one or more atoms. A SWRL rule is of the form:

$$A1,\ldots\ldots\ldots\ldots An \rightarrow B1,\ldots\ldots\ldots\ldots\ldots\ldots Bn$$

where *A1,…………An* refers to the head part of the rule and comma represent the conjunctions of one or more atoms and *B1,…………………Bn* refers to the body part of the rule. For example, consider family knowledge base whose SWRL rules for the same is defined as specified in Fig. 2.5.



```
Person(?p), bornInYear(?p, ?year), subtract(?age, ?nowyear, ?year), thisYear(?nowyear) -> hasAge(?p, ?age)
Person(?p), hasAge(?p, ?age), greaterThan(?age, 18) -> Adult(?p)
Person(?p), integer[>= 18 , <= 65](?age), hasAge(?p, ?age) -> hasDriverAge(?p, true)
Person(?x), hasChild min 1 Person(?x) -> Parent(?x)
Person(?p), date(?date), bornOnDate(?p, ?date), date(?date, ?year, ?month, ?day, ?timezone) -> bornInYear(?p, ?year)
```

Fig. 2.9 SWRL Rule

Fig. 2.9 shows SWRL rules implemented in Protégé. Consider one of the rules

Person(?p),bornInYear(?p,?year),subtract(?age,?nowyear,?year),thisyear(?nowyear)→hasAge(?p,?age)

This rule calculates the age of the person by subtracting born year from current year.

b) Rule Markup Language (RuleML)

RuleML [106] is a Rule Markup language for Semantic Web. RuleML has four categories of rules which are defined as below:

- **General reaction rules**

These rules are applied in forward direction for observing/ checking events/conditions and performing an action when all events/ conditions have been perceived/ fulfilled.

- **Integrity constraint rules**

These rules are also forward oriented, i.e. triggered by updates, mainly for efficiency reasons.

- **Derivation rules**

The category of these rules can be applied in the forward direction as well as in backward direction, the latter reducing the proof of a goal (conclusion) to proofs of all its sub goals.

```
<implies
<head>
<atom>
<rel>discount</rel>
<var>customer</var>
<Ind> 10% </Ind>
</atom></head>
<body>
<if>
<atom>
<rel>spend</rel>
<var>customer</var>
<Ind> 5000rs </Ind>
<Ind> bill </Ind>
</atom>
</if>
</body>
</implies>
```

Fig. 2.10 Example of RuleML

- **Facts rules**

  These classes of rules are used for an application direction.

  The rule "The customer is given 10% discount if he spends Rs.5000 for a bill" is represented in syntax of RuleML as shown in **Error! Reference source not found.**

  In the example, where t starts and end with <implies> </implies> syntax and is divided into <head><atom> and <body><atom>part. The relation predicate (discount, spend) is represented by <rel> tag. The variables (customer) are represented with <var> tag and constant values (10%, 5000rs) are represented by <ind> individual tag.

The next section discusses about semantic web query languages.

2.10 SEMANTIC WEB QUERY LANGUAGES

Several formalisms have been proposed for representing data and metadata on the Semantic Web. RDF and OWL allow one to describe relationships between data items, such as concept hierarchies and relations between the concepts. Now, in order access data, Semantic Web query languages [107] are required. A wide range of query languages for the Semantic Web exist which are discussed as below:

a) SquishQL (SQL like Query language)

Squish query [108] syntax is like SQL query language. It is a query language based on graph navigation. SQL query language for RDF provides consistent, human-understandable, access to repositories of semantic data, whether stored files or large databases, enabling application programmers to create semantic web applications quickly. For example, consider a query written in SquishQL language syntax as shown in Fig. 2.11. The figure shows a query in SquishQL which selects title from *http://example.com/xmleurope/presentations.rdf* document by selecting the document where the predicate <dc:title> and the document are of type FOAF documents. *Using* clause specifies abbreviation for long URIs by defining a string prefix; this example specifies URIs for Dublin Core (DC), FOAF (Friend-of-a-friend), RDF (Resource Description Framework.)

```
Select ?title
From http://example.com/xmleurope/presentations.rdf
Where (?doc,<dc:title>,?title)
(?doc,<rdf:type>,<foaff:Document>)
Using
dc for http://purl.org/dc/elements/1.1/,
foaf for <http://xmlns.com/foaf/0.1>,>,
rdf for http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

Fig. 2.11 Syntax of SqishQL

b) RDQL

RDQL [109] originated with the language SquishQL, which evolved into RDQL and then was later extended to the language SPARQL. These languages take RDF as triple data without schema or ontology information unless explicitly included in the RDF source. The syntax of RDQL is similar to SQL *select* clause but it does not include from clause. Consider the example of RDQL query shown below:

Select ?x where (?x, <rdfs:label>, "abc")

Above query lists all resources with "abc" in the variable x.

c) SeRQL (Sesame RDF Query Language)

SeRQL (pronounced "circle") [110] is considered as second generation RDF Query language. This language is based upon earlier query languages such as RDQL and N3. SeRQL uses a path expression syntax that is similar to the syntax used in RQL, and is based on the graph nature of RDF; the path is expressed as a collection of nodes and edges, where each node is denoted by surrounding curly brackets.

{node} edge {node} edge {node}

Consider an example to query, RDF graph for Book with Author name is Dr. AK Sharma, the path expression for this query would be specified by

{Book}<foo:hasAuthor> {Author}<rdf:type> {foo:Dr. AK Sharma>}

This query will list all Books whose author is Dr. AK Sharma.

d) SPARQL (SPARQL Protocol and RDF Query Language**)**

SPARQL [111, 112], pronounced 'sparkle', is the standard query language and protocol for Linked Open Data on the web or for semantic graph databases (also called RDF triple stores).It enables users to query information from databases or any data source that can be mapped to RDF. The SPARQL standard is designed and endorsed by the W3C and helps users and developers focus on what they would like to know instead of how a database is organized. Just like SQL allows users to retrieve and modify data in a relational database, SPARQL provides the same functionality for NoSQL graph databases like Ontotext's GraphDB.

In addition, a SPARQL query can also be executed on any database that can be viewed as RDF via a middleware. This feature makes SPARQL a powerful language for computation, filtering, aggregation and sub query functionality.

A SPARQL query consists of a set of triple patterns in which each element (the subject, predicate and object) can be a variable (wildcard). Solutions to the variables are then found by matching the patterns in the query to triples in the dataset.

SPARQL has four types of queries [111, 112], which can be used to:

- **ASK** whether there is at least one match of the query pattern in the RDF graph data;
- **SELECT** all or some of those matches in a tabular form (including aggregation, sampling and pagination through OFFSET and LIMIT);
- **CONSTRUCT** an RDF graph by substituting the variables in these matches in a set of triple templates;
- **DESCRIBE** the matches found by constructing a relevant RDF graph.

Like SQL, which is used for querying structured databases, SPARQL queries are used to query unstructured databases and have a SELECT-FROM-WHERE structure. There are other query languages which are considered as first generation query languages, which has a good expressive query constructs but they are not supported by all the tools for ontology development and lack interoperability feature, hence these query languages are not considered as the standard languages for querying. For example, consider a semantic data fragment of an FOAF ontology [113] which consist of name, designation and email-address and other information of a person. Query on such data to query name, designation with SPARQL can be used as shown in Fig. 2.12.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?designation WHERE
{ ?x foaf: name ?name.
  ?x foaf: designation ?designation.
}
```

Fig. 2.12 Example of SPARQL Query

The above query written in SPARQL retrieves name and designation of a person which is represented using "?x" variable. The main query constructs [114] used in this query are as follows:

- **PREFIX-** PREFIX keyword is used to declare a namespace for a URI.
- **SELECT**-This keyword is used to specify data items that will be included in the result set. In this for example variable name, designation is included in result set.
- **FROM-** This keyword specifies the data set on which the query will be executed.
- **WHERE-**This keyword specifies the triple/graph pattern which query will match against a RDF graph. Variable names have question mark in their beginning. This triple query will be evaluated against all the triple that exist in the semantic data.

In the proposed work, system takes keyword based query from the user and transforms into SPARQL query to fire on the respective ontologies to get the relevant results.

e) SQWRL (Semantic Query enhanced web rule Language) [115, 116]

It also has SQL-like operations to query knowledgebase of OWL. It is considered as an expressive language for performing queries on OWL ontologies. SQWRL takes a standard SWRL rule antecedent and effectively treats it as a pattern specification for a query. It replaces the rule consequent with a retrieval specification. The core SQWRL operator is sqwrl: select. Consider an example, Query: "Return all persons whose age is greater than 18". Its respective SPARQL query is shown as below.

Person(?p)^hasage(?p,?a)^swrlb: greaterthen(?a,18)-> sqwrl: select(?p, ?a)

Upon running the above query, it returns a list of person whose age is greater than 18.

2.11 PROGRAMMING THE SEMANTIC WEB

There are several ways to implement semantic web application using current and emerging standards and technologies namely the Jena framework, Protégé-OWL API and the WonderWeb OWL API, which are all available for Java language.

a) JENA

Jena (Jena 2002; Jena 2005) [117, 118] is a Java framework for building semantic Web applications developed by the HP Labs Semantic Web Program. It provides a programmatic environment for RDF, RDFS and OWL, including a rule-based inference engine and a query language for RDF called RDQL. This API supports several ontology description languages such as DAML, DAML+OIL and OWL. Jena OWL API supports all three OWL sublanguages, namely OWL Lite, OWL DL and OWL Full. Specifying an URI to an OWL ontology, Jena parses the ontology and creates a model for it. With this model, it is possible to manipulate the ontology, create new OWL classes, properties or individuals (instances). Jena includes an inference engine which gives reasoning capabilities. Jena provides three different reasoners that can be attached to an ontology model, each of them providing a different degree of reasoning capability.

b) Protégé API

The Protégé-OWL API [119, 120] is an open source Java library for OWL and RDF(S). This API provides classes and methods to load and store OWL files, to query and manipulate OWL data models, and to perform reasoning. This API, which is part of the Protégé-OWL plug-in, extends the Protégé Core System based on frames so that it can support OWL ontologies and allows users to develop OWL plug-ins for Protégé or even to create standalone applications. Protégé-OWL API uses Jena framework for the parsing and reasoning over OWL ontologies and provides additional support for programming graphical user interfaces based on Java Swing library.

c) OWLAPI

WonderWeb OWL API [121] (OWLAPI 2006) is another API providing programmatic services to manipulate OWL ontologies. *The OWL API* is a Java interface and implementation for the W3C Web Ontology Language OWL. The latest version of the API is focused towards OWL 2, which encompasses OWL-Lite, OWL-DL and some elements of OWL-Full. It can also infer new knowledge once a reasoner is attached to the ontology model. Pellet is one of the reasoners that are currently supported.

In the last few years, due to the popularity of semantic web, size of semantic web data such as ontologies, annotated structured data has increased very rapidly. To search semantic data, current search engines are not efficient and due to this, several semantic search engines have emerged recently. In the next section, architectures of some popular semantic search engines are explained followed by comparative analysis based on some parameters.

2.12 SEMANTIC SEARCH ENGINES

To access structured data, a number of semantic search engines has been introduced which understands the meaning of data and helps in displaying more exact result as compared to current search engines. Among them some of the existing semantic search engines has selected for discussion in this section with their architectures.

2.12.1 Swoogle

Swoogle [122, 123] is a crawler based indexing and retrieval system for semantic web documents written in RDF and OWL. SWDs are further categorized as SWO (Semantic Web Ontologies) and SWDB (Semantic Web Database). A document is considered as SWO when a significant proportion of the statements it makes define new terms or extends the definition of terms defined in other SWDs. A document is considered as SWDB when it does not define or extend a significant number of terms. Discovered documents are also indexed by an IR system which uses *URIrefs* as keywords to find relevant documents. The key goal in building Swoogle is to design a system that can handle millions and even tens of millions of documents. The architecture of Swoogle is shown in Fig. 2.13 as below.

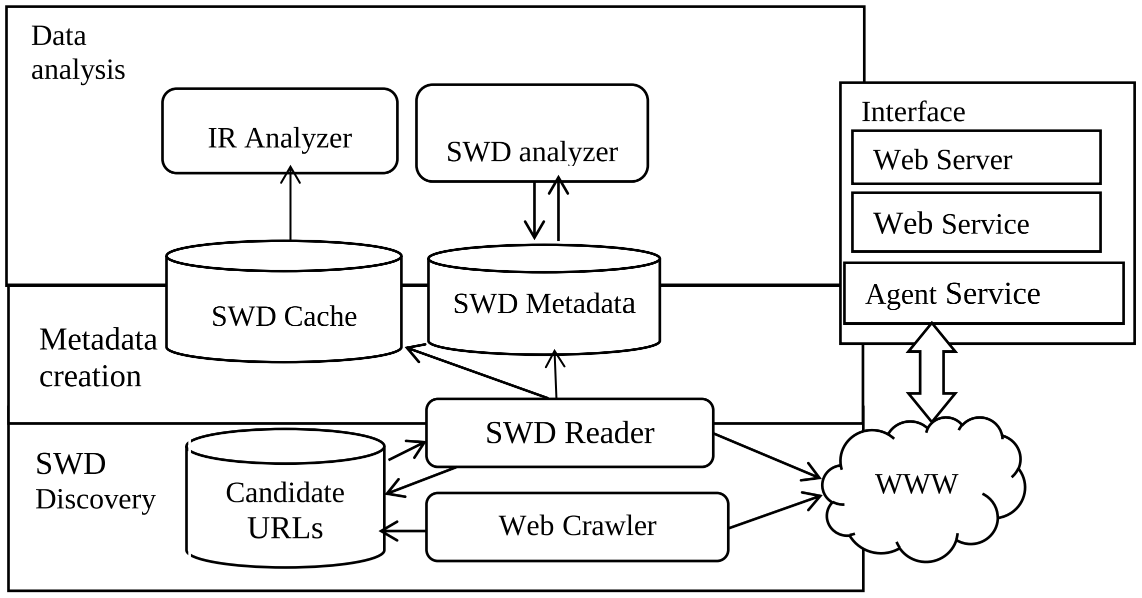


Fig. 2.13 Architecture of Swoogle Search Engine

Swoogle architecture can be broken into four major components: SWD discovery, metadata creation, data analysis and interface.

a) **SWD Discovery**

Swoogle adopts a hybrid approach to harvest the semantic web. It collects candidate URLs to find and cache SWDs using four mechanisms: submitted URLs of SEDs and sites, a web crawler that explores promising sites, a customized meta-crawler that discovers likely URLs using conventional search engines, swooglebot semantic web crawler which validates and analyses SWDs to produce new candidates.

b) **Indexing**

This component analyses the discovered SWDs and generates metadata about SWDs at both the syntax and semantic level. It captures features like encodings namely "RDF/XML", N-triple, language such as OWL, DAML, RDFS, RDF. It records ontology properties such as label, comment, version info. It also focuses on SWD level relations such as term reference relations between two SWDs, imports, extends etc. which are extracted from SWD by analyzing triples containing indicators such as owl:imports, daml:imports, rdfs:subclassof.

c) Analysis

This component uses the created metadata to derive analytical reports such as classification of SWOs and SWDB, ranking SWDs using rational surfer model.

d) Services

The interface component focuses on providing data services such as search services that search ontologies at the term level.

2.12.2 Falcon

It is a keyword based semantic search engine [124, 125] which generates all the ranked RDF documents that includes the terms in the fired query. For example, user wants to know people *peter mika*, then corresponding to this query, it tries to generates if those RDF documents that contains this kind of information and at the snippet that exact information is shown so that user does not need to crawl unnecessarily to other pages. It displays required information on the snippet itself, therefore user does not need to explore in that page also. The Architecture of Falcon is described in Fig. 2.14.

a) RDF Crawler

An RDF crawler is setup to crawl RDF documents. It creates queries by enumerating general keywords and sent them to Google and swoogle to generate RDF documents. The crawler is also customized to download RDF documents from DBpedia, Hannover, DBLP Bibliography.

b) Document level analysis

It contains Jena parser which parses the cache documents collected by RDF crawler. Everything in RDF document is represented by URIs, it may happen that new discovered URIs may further dereference to another SWD, therefore, they are queued in the seed to explore more RDF documents. As we know, traditional search engines index extracted terms from the crawled documents a map query terms resulting in displaying set of documents containing the terms present in the query. But here, in semantic web, semantic objects are identified by URIs, from which only limited useful terms (may be just local terms) can be extracted. So, widely used current semantic search engine use both local name and their associated literal of semantic web objects to form their textual description and then build the inverted index but this limits the indexing. Falcon expands the textual description of URI by including not only its local name and its associated literal values, but also description about its neighboring semantic web objects in RDF graph. For this, it creates a virtual document which contains for each object, its local name, associated literals and labels of its neighboring objects in RDF graph. This virtual document will be used in indexing.

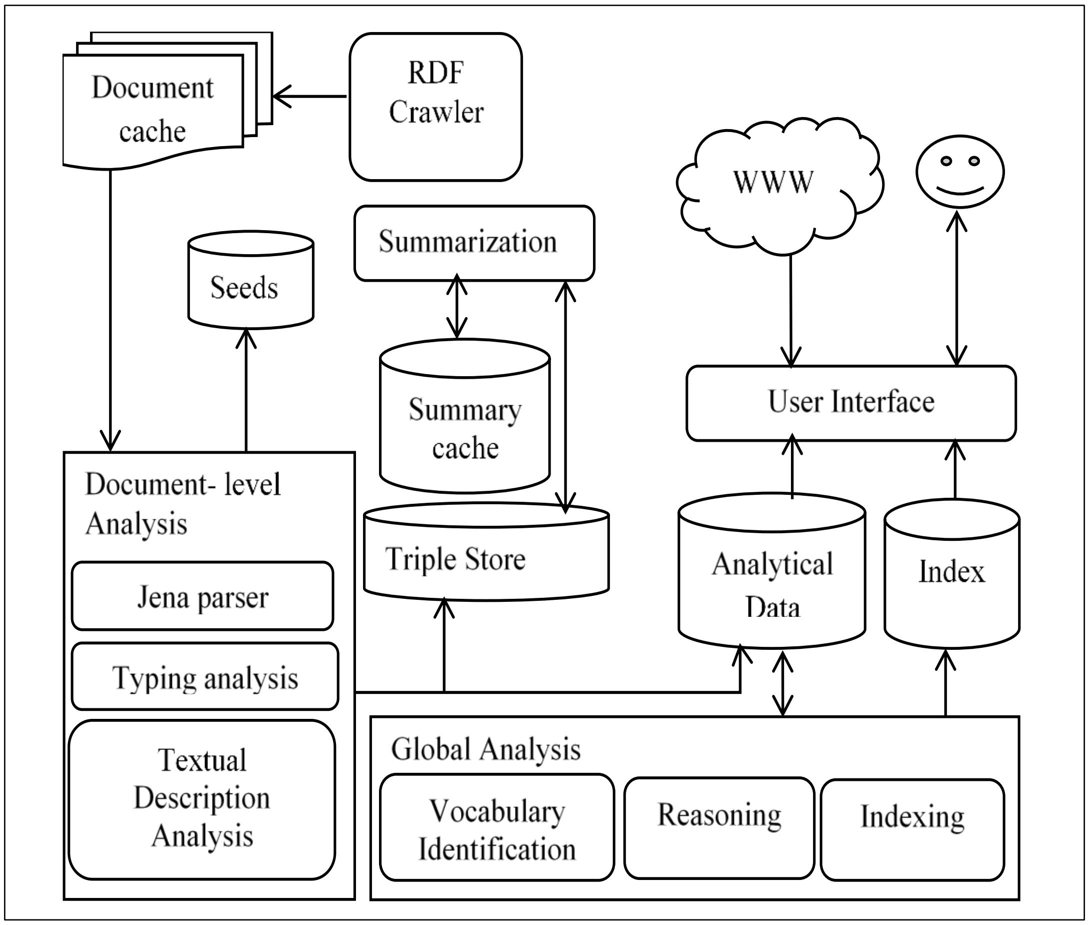


Fig. 2.14 Architecture of Falcon Search Engine

c) Global Analysis

Before indexing, vocabulary identification and then reasoning using class inclusion relation is preformed and then indexing is performed.

d) Summarization

A query dependent snippet of knowledge is provided to facilitate end user to gather its information from the snippet itself without exploring into that document and stored into summary cache with respect to that object.

e) User Interface

Just as Traditional Web search engines, which provide Web Pages that contain the keywords in a query at the user interface, this engine also does the same but it is not easy for the user to specify a dimension of knowledge about the subject, except for resubmitting queries with different combinations of keywords to try his/her luck in traditional search engines. To move out users from such kind of problems, falcon organizes its knowledge by utilizing typing information which is associated with its objects. Therefore, using this typing information, different dimensions of an object can be recognized and correspondingly its knowledge can be organized which helps its users to focus in one direction only.

2.12.3 Hakia [126]

Conventional search engines such as Google, yahoo, Bing etc. are keyword based search engines which retrieve a list of HTML documents in a rank order containing the terms present in the fired query. The processing of these search engines is syntactic in nature and does not understand content and query like how the human brain processes natural language. To deal with this, Hakia Corp. introduced a semantic search technology based search engine named as Hakia that bring relevant results based on concept match rather than keyword match and popularity ranking.

The Architecture of Hakia is described as shown in Fig. 2.15 and the description of various components is given below:

a) Crawler

Hakia crawls credible sites recommended by librarian, so that a collection of relevant documents can be formed. The result is the collection of quality pages in topics such as health, finance, environment, science and others. It also uses feeds from news, blogs, and databases to get the dynamic content.

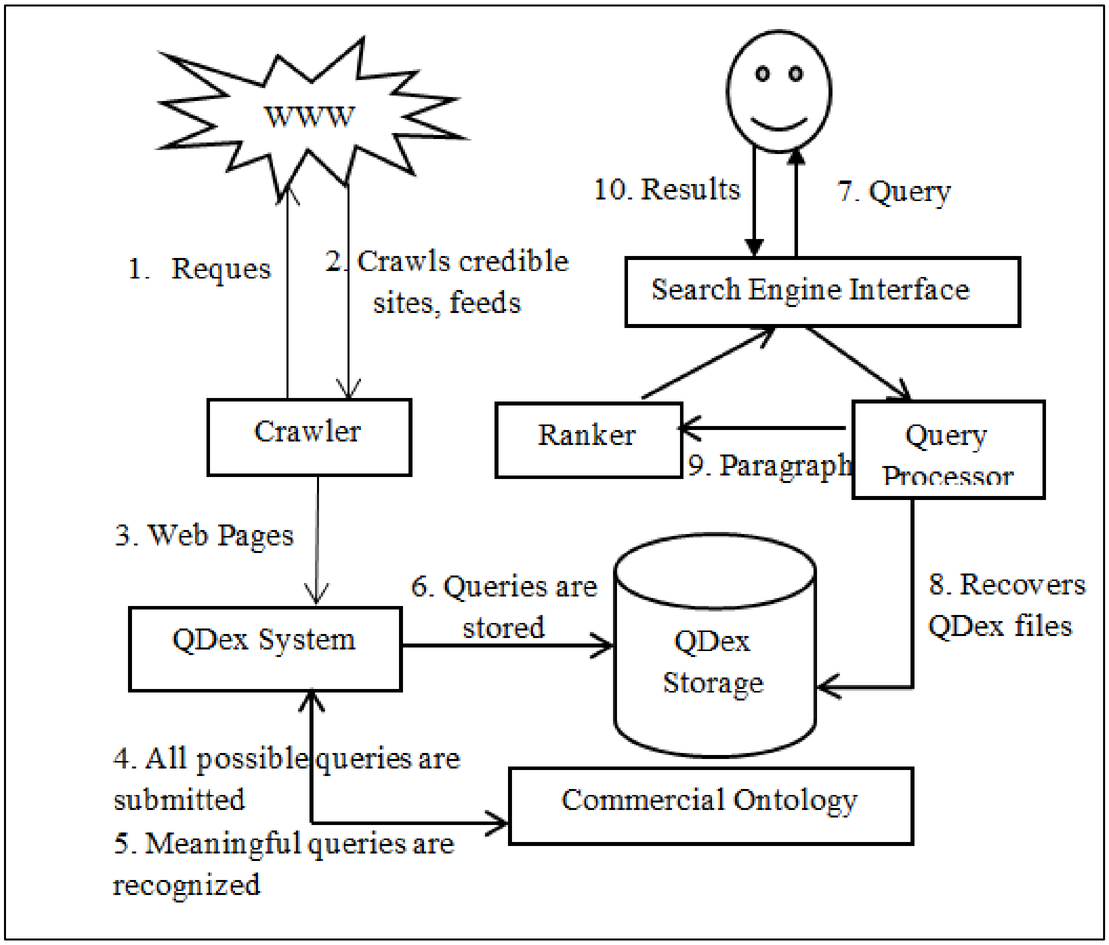


Fig. 2.15 Architecture of Hakia Search Engine

b) QDexing

After collecting data from different segments, QDex (stands for Query Detection & Extraction) analyzes each web page much more intensely. It extracts all the possible queries that can be asked to that page by decomposing sentences into sequences of words which generates the vast number of queries out of which only few dozen queries make sense. To deal with this challenge, Hakia uses one system which is known as *Commercial ontology* which helps in extracting senseful queries out from the exploratory space.

c) Commercial Ontology

Here, all the extracted queries are further analyzed such as morphological analysis, generalization, and characterization and thus queries are categorized into various senses they convey.

d) QDex Storage

Hakia QDexes every document and extracts queries for each document. Some queries repeat as they are extracted from different pages. Thus, for each query, a QDex file is created which contains information about the document, paragraph from which that query was extracted. If the query is new, a new file is created. After that, each Qdex file is placed in a known destination via hash-mode operation. All this work is performed offline.

e) Query Processor

The query is sent to the query analyzer which uses fall back algorithm to generate the sense and context of that query and with hash mode, their destination is known exactly and correspondingly all the requested QDex files are retrieved.

f) Ranking

From the above process, a pool of relevant paragraph comes from the Qdex system for given query term. Then, the final relevancy is determined by the semantic analysis ranking algorithm based on advanced sentence analysis and concept match between the query and the best sentence for each paragraph which will be highlighted in the snippet to attract the user.

2.12.4 Semantic Web Search Engine (SWSE) [127, 128]

The search engine as shown in Fig. 2.16 starts with a set of seed URIs, retrieves to content of URIs, parses and writes content to disk and recursively extracts new URIs for crawling. Currently, it crawls RDF/XML syntax documents which are most commonly used for publishing RDF on the web.

a) Consolidation

On semantic web, every object is identified by URIs and it has allowed publishers to create their own URIs for representing an object. This facility creates a problem in integrating knowledge about that object at one place because that object is named by different URIs. Consolidation is a step which provides a mean of identifying equivalent entities in RDF data e.g. OWL defines the owl:sameas property which relates two equivalent entities; entities representing the same real world individual but identified incongruously. This would enable the merging of information contributed on an entity given by heterogeneous source without the need for consistent URI naming of entities.

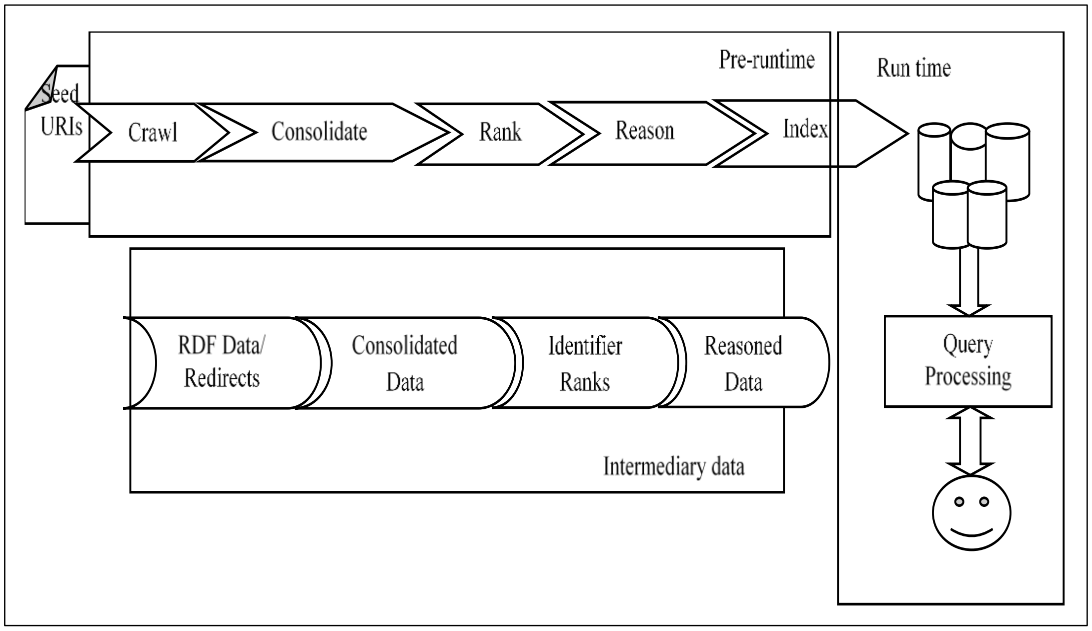


Fig. 2.16 Architecture of Semantic Web Search Engine

b) Ranking

Considering ranking as an important mechanism in the search process with the function of prioritizing data elements, it uses linked based analysis, proven for HTML web, for ranking linking data entities. Given that the notion of a hyperlink is missing in RDF web: linked data principles mandate implicit links to other data sources through re-use of dereferenceable URLs.

c) Reasoning

By appending instance data (i.e. assertion data) describing about the object, SWSE introduced scalable authoritative OWL reasoned (SAOR) system for performing large scale materialization using a rule based approach which helps to infer logical consequences from a set of facts or axioms described using classes and properties. The system does not produce inferences that would over-burden the indexing process and system should pre-compute inference to avoid the runtime expense otherwise it would impact upon response time.

d) Indexing component

It employs an inverted index for keyword lookups based on RDF literals (text), and a sparse index for lookups of structured data. With a pair of keys and pointers for every entity in the data file, every entity in this file is associated with a pointer to the block in the sorted data file. This block contains entity snippet containing a detailed description which is formed by aggregating from many sources, description also includes inferred data which is not necessarily been published but derived from the existing data through reasoning.

e) Query processing and User interface

It accepts user queries, retrieves top k hits and requests the snippet result data for each of the hits and displays as an output at interface.

2.12.5 DuckDuckGo [129]

It is a feature-rich semantic search engine which gives countless reasons to leave Google. If we search for a term that has more than one meaning, it will give you the chance to choose what you were originally looking for, with its disambiguation results. For example, searching for the term Apple will give you a long list of possible meanings including fruit, computer company, bank etc.

2.12.6 Sensebot [130]

Sensebot uses text mining to parse Web pages and identify their key semantic concepts. It then performs multi document summarization of content to produce a coherent summary. It gives a summarized accurate search results according to the query given. The summary gives a good idea of the topic of the query. The summary is readable and coherent. SenseBot saves time by providing an overview of the topic, and pointing to the right sources. The search engine itself tries to understand the concept of the query, what it contains and gives an appropriate result. The user need not go through many web pages to get the results.

2.12.7 Powerset [131]

The Microsoft-acquired search engine Powerset focuses on doing only one thing and doing it well by using natural language processing to understand the nature of the question and returns pages containing the answer. All search results on Powerset come from Wikipedia, making it the ultimate way to search Wikipedia, using semantics Search terms can be formulated as questions, which will be answered, or as simple terms, and results will be aggregated from all the relevant pages on Wikipedia. It helps to give comprehensive view of the thing that user searches for. It aggregates the information provided by the different resources. It provides a set of suggestions about the query given and the related queries.

2.12.8 Watson [132]

Watson is a gateway for the Semantic Web, which has been guided by the requirements of Semantic Web applications and by lessons learnt from previous systems. It uses Ontology crawling exploration technique. It provides explicit and implicit relations between ontology, providing rich, semantic access to data, focusing on semantic quality. It exploits the strengths of semantic technologies to provide fundamental functionalities for a more suitable access to online knowledge.

The comparison of the above discussed Search Engines is performed based on various measures like the approaches used, Output result format, Input format and Technique used. The detailed comparison study is outlined in Table 2.6.

Table 2.6 Comparative Study of Semantic Search Engines

Search Engine	Approaches used	Output Result Format	Input Format	Techniques Used
Hakia	It is based on producing relevant results based on concept match rather than Keyword match	HTML documents	Natural Language questions or Phrases, keywords.	QDEXing (Query Detection & Extraction)
DuckDuckGo	Results are compilation of over 400 sources such as Yahoo! Search BOSS; Wikipedia; Wolfram Alpha; Bing; its own Web crawler (the DuckDuckBot)	Classified results with their HTML web pages giving the possible meaning for the query entered.	Natural Language	Clustered approach and NLP techniques.
Cognition	It produces results based on ontology and wordnet [166, 167] vocabulary.	HTML link results	Natural Language phrases	It uses Linguistic, Boolean search, fuzzy search

				technologies to produce results.
SenseBot	Concept Search	Summarized results	Query using keywords	Using text mining algorithms that parse the web pages to produce results.
Powerset	Based on giving results searching the contents of Wikipedia.	HTML Web Pages	Query using keywords, natural language questions or phrases	Powerset semantic indexing is based on the XLE (Xerox Linguistic Environment), Natural Language Processing technology
Google	Keyword Matching	HTML Web Pages	Natural Language	Page Rank Algorithm
Swoogle	1. Search semantic web ontologies and documents 2. Searches SW terms i.e.; URIs 3. provides metadata of SWDs.	Finds appropriate ontologies and list them in ranked order.	Domain concepts	1. N Gram based indexing 2. Ontology rank based on PageRank
Watson	finds ontologies by integrating the search capabilities	Ontology listing	Domain concepts	1. Watson semantic gateway 2. NeOn Toolkit
Falcon	Concept Search	Produces ontology listing and generates query relevant structured snippets	Keywords	Popularity based approach for ranking of concepts and ontologies.
Semantic Web Search Engine (SWSE)	Keyword based search engine for object, operates over RDF data	Domain concepts	Keywords	1. Inverted indexing for literals, sparse indexing for structured data 2. Ranking through link based analysis

Above table shows the comparative analysis of various search engines done based on input format, out result format and the techniques used by these search engines. Generally, most of the search engines that are used in current web searches based on keyword matching like Google and semantic search engines which are used for finding ontology can be reused for a domain. The example of such search engines is Watson, Falcon etc.

2.13 SUMMARY

This chapter covers the complete literature required as pre-requisite before working on semantic web applications. The literature started with problem identification in current web then it moved towards semantic web as a solution of problems being faced in current web to various technologies, tools, implementation software's and semantic search engines. As an ending pointing of this chapter, a summary table as shown in Table 2.7 which gives a list of ontology tools which are used during various stages of ontology development.

Table 2.7 Summarization of various Ontology Tools

Tools	Examples
Ontology Editor tools	Protégé, SWOOP, NeOn toolkit, WeODE, OilEd, OntoEdit
Ontology Annotator tools	Annotea
Ontology reasoning tools	Pellet, racer, HermiT, Fact++, Kaon2
Ontology learning tools	Protégé withLT, ODEMapster
Ontology evaluation tools	Ontoanalyser, Ontoclean, radon
Ontology storage Frameworks	Redland, Sesame, Allegrograph, Virtuoso
Semantic Search Engines	Swoogle, Hakia, Cognition, Sensebot, Powerset, SWSE

In continuity with this, in the next chapter, process of developing ontology along with working example, various ontology management methods and tools with their comparative study are explained in detail.

Chapter III

ONTOLOGY MANAGEMENT TOOLS

3.1 INTRODUCTION

Ontology has been introduced in the semantic web with the intention of providing common vocabulary specific to a domain to the experts so that they can get interlink, combine and communicate knowledge. But, in actual it has been experienced that experts prefer to create their own ontologies rather than existing ontologies which results in existence of different conceptualization of the same domain. This practice has developed many challenges in various fields such as information integration, information services etc. In order to handle these challenges, there is a need to bridge the gaps between ontologies of same or different domain to form a communication. In this chapter, various ontology management methods such as ontology merging, ontology alignment and ontology integration has been discussed.

In the upcoming section, process of developing ontology is discussed with complete example.

3.2 PROCESS OF DEVELOPING ONTOLOGY

The process of ontology development [55] is not a linear process rather is an iterative process which requires the revision and refinement of concepts for the evolving ontology. To understand the ontology development process, a Human Family Tree ontology [133] was designed and developed during the course of work. Below is explained the development process of family tree which explains each step depicting the ontology lifecycle [134].

a) Determine the Domain and Scope of Ontology

The first step in the development of ontology involves determining the domain and scope of ontology. The various things to be kept in mind while designing ontology are as follows:

- Domain where the ontology design is to be applied.

- Application of the ontology.
- The characteristics of ontology.
- Type of application the ontology can be applied to.
- The type of question the ontology should be able to answer.
- User of ontology and maintenance of ontology.
- Languages to be used which will be appropriately mapped to the intended application.

For example, while designing ontology for Human Family Tree

(i) Domain is: "Human Family"

(ii) Scope covered is: Person family and its medical history. This ontology design describes the entities in relations and medical history of a person's family domain. Ontology scope refers to defining:

- Relation between persons that exist in a family
- Habits of person.
- Blood groups of the person.

b) Considering the reuse of Existing Ontology

Developing ontology from scratch is considered as a very difficult and time consuming process which requires a lot of domain knowledge, so it is always advisable to reuse the already existing ontology and extend it with own concepts to meet one's requirements.

The following things must be kept in the mind while considering the reuse of ontology

1. Application which can use the developed ontology for consideration of reuse.

2. Library from where ontology can also be reused rather and starting from the scratch e.g. DAML Library [46] and Ontolingua library [58] has a large collection of ontologies.

For current research, there was no existing ontology in the domain of Jobs that is meeting the specified requirement therefore the ontologies have been designed from scratch for the proposed research work.

c) Enumerate the important terms in ontology

All the important terms from the domain of interest are identified without worrying about which term would be used for what purpose. For example, the terms father, mother, brother, sister, habits, blood-group etc. are recognized.

d) Determine class hierarchy

At the basic level, 5 main super-classes are identified. They are explained as below in Table 3.1.

Table 3.1 Human Family Tree Ontology Super Class Description

S.No.	Class	Description
1.	Gender	This class tells the gender of the individual belonging to class Person.
2.	Person	This class consists of a hierarchy of sub-classes which describes the maximum possible relations that might exist in the biological family such as Parent, Sibling, Relative, Grandparent, Spouse etc.
3.	Blood group	This class consist the blood group type of the individual belonging to class Person.
4.	Lifestyle habits	This class holds the Lifestyle habits such as Alcohol, Smoke, Exercise, Food preference, diet habits as Subclass.
5.	Medical history	This class holds diseases as subclass which passes from one generation to next as hereditary disease. For example, Diabetes, Cancer, Heart attack etc.

e) Define the properties of the class

In this step, to describe the internal structure of the concepts, properties are defined which are used to link concepts. Two types of properties are defined:

- Object property
- Data property.

Object property links an individual to an individual. For example; *hasChild* data-property links an individual of person class to the other individual of person class.

In this ontology, 14 main data-properties are defined which have further sub-properties. Creating sub-properties have enhanced the flexibility. For example, with the help of *hasParent* property, all the parent individuals can be retrieved during query execution. A sample of object property which is used in this ontology is shown in Table 3.2

Table 3.2 Sample Object Property with its Sub-properties

Property	**SubProperty**	**Domain**	**Range**
HasSpouse	hasHusband	Person	Person
	hasWife	Person	Person

Here, *hasSpouse* is the object property which has further two subproperties named as *hasHusband* and *hasWife*. With this property, general spouse relations as well as specialized husband and wife relation can also be determined.

f) Creating Instances of the Classes

For each class, various individuals are declared which are discussed as follows:

- **Person Class**

For this class, consider Mr. U.C. Gupta Family tree as a set of individuals which are defined in Fig. 3.1 as follows:

Here, U.C.Gupta is an individual of class Person. This individual is related to Gender class with *hasGender* property whose instance is Male, which says in simple English that U.C. Gupta is a Male. With slash Manorama who is the wife of U.C. Gupta with a property *hasWife* is assigned. U.C. Gupta has 4 children which are shown at level 2. Madan is a son of U.C. Gupta and Ritu is his Wife. This way whole family tree is designed.

Initially some information using *hasGender, hasWife and hasFather* properties is provided to everyone of class *Person* which would be helpful inferring new knowledge.

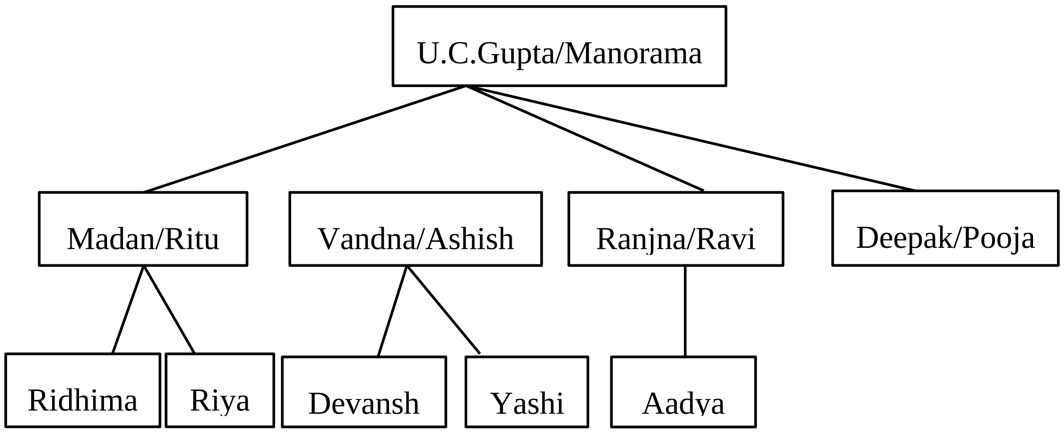


Fig. 3.1 Sample Family Tree as Instances for Class Person

- **BloodGroup Class**

Table 3.3 defines the relationship between blood groups on the basis of which it is decided who can donate blood to which other blood group person instance and from whom person's instance, other person instance can receive blood.

Table 3.3 BloodGroup Chart

Bloodgroup	DonateBloodto	ReceiveBloodfrom
A+	A+ , AB+	A+ , A- , O+ , O-
O+	O+ , A , B+ , AB+	O+ , O-
B+	B+ , AB+	B+ , B- , O+ , O-
AB+	AB+	Everyone
A-	A+ , AB+ , A- , AB-	A- , O-
O-	Everyone	O-
B-	B+ , B- , AB+ , AB-	B- , O-
AB-	AB+ , AB-	AB- , A- , B- , O-

Using this table, 8 instances of BloodGroup class are created and initially, to whom one can donate blood to and from whom one can receive blood from using *candonatebloodto* and *canrecbfrom* property is assigned to every *bloodgroup* instance. In the same way instances of *Medical_history, Gender* and *Lifestyle_habit* classes are created and defined.

Once the ontology is created, the next step to check the consistency of the ontology. For this, Pellet 1.5.2 (direct) [92, 93] which is embedded in Protégé itself is used. This completes the development of human family Tree ontology development.

3.3 BENEFITS OF ONTOLOGY

Ontology has many benefits [135,136] out of which some are discussed as below:

a) One of the main features of ontologies is that, by having the essential relationships between concepts built into them, they enable automated reasoning about data. Such reasoning is easy to implement in semantic graph databases that use ontologies as their semantic schemata.

b) Ontologies function like a 'brain'. They 'work and reason' with concepts and relationships in ways that are close to the way humans perceive interlinked concepts.

c) In addition to the reasoning feature, ontologies provide a more coherent and easy navigation as users move from one concept to another in the ontology structure.

d) Ontologies are easy to extend as relationships and concept matching are easy to add to existing ontologies. Thus, this model evolves with the growth of data without impacting dependent processes and systems if something goes wrong or needs to be changed.

e) Ontologies also provide the means to represent any data formats, including unstructured, semi-structured or structured data, enabling smoother data integration, easier concept and text mining, and data-driven analytics.

3.4 ISSUES IN DATA SHARING AND ONTOLOGY INTEGRATION

Despite of various benefits of ontology, some issues [137] are encountered while considering data sharing and data integration in a domain.

a) It is said to use existing ontology of a domain for representing data, but in actual; in place of reusing existing ontologies of required domain, domain experts create their own ontology leading in formation of multiple ontologies of the same domain containing incomplete concepts and relations. This causes ontology heterogeneity [138] and inconsistency problem.

b) Several challenges such as finding similarities and differences among ontologies in automatic and semi-automatic way, defining mapping between ontologies, composing mappings across different ontologies must be faced during managing these diverse ontologies.

Therefore, for better and precise results, managing these heterogeneous ontologies is necessary.

3.5 ARCHITECTURES OF ONTOLOGY MANAGEMENT

There are three main architectures that are implemented in ontology-based data integration applications, namely.

a) Single ontology approach

A single ontology approach [139] as shown in Fig. 3.2 is used as a global reference model in the system. This is the simplest approach as it can be simulated by other approaches.

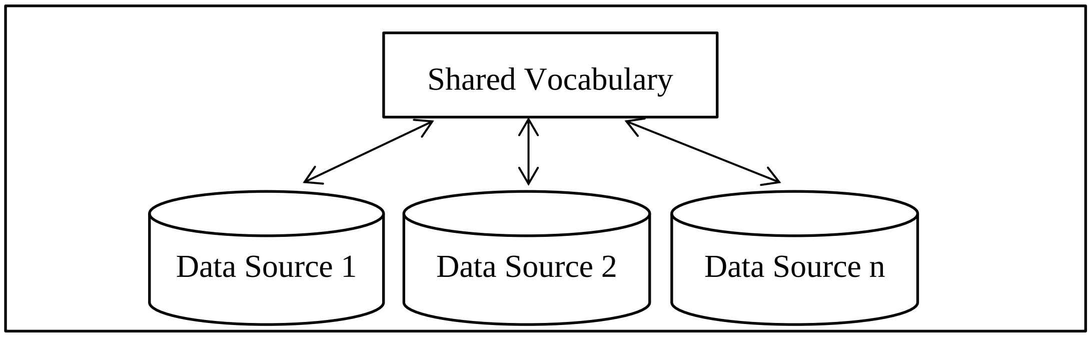


Fig. 3.2 Single Ontology Approach

b) Multiple ontologies

Multiple ontologies approach [139] as shown in Fig. 3.3, which models each data source as an individual and are used in combination for integration. Although, this approach is more flexible than single ontology approach, it requires creation of mappings between the multiple ontologies.

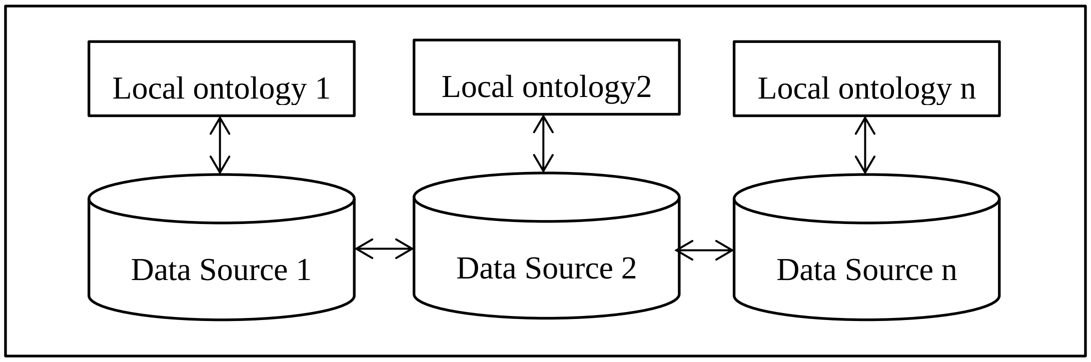


Fig. 3.3 Multiple Ontology Approach

c) Hybrid approaches

The hybrid approach [139] as shown in Fig. 3.4 involves the use of multiple ontologies that subscribe to a common, top-level vocabulary. The top-level vocabulary defines the basic terms of the domain. Thus, the hybrid approach makes it easier to use multiple ontologies for integration in presence of the common vocabulary.

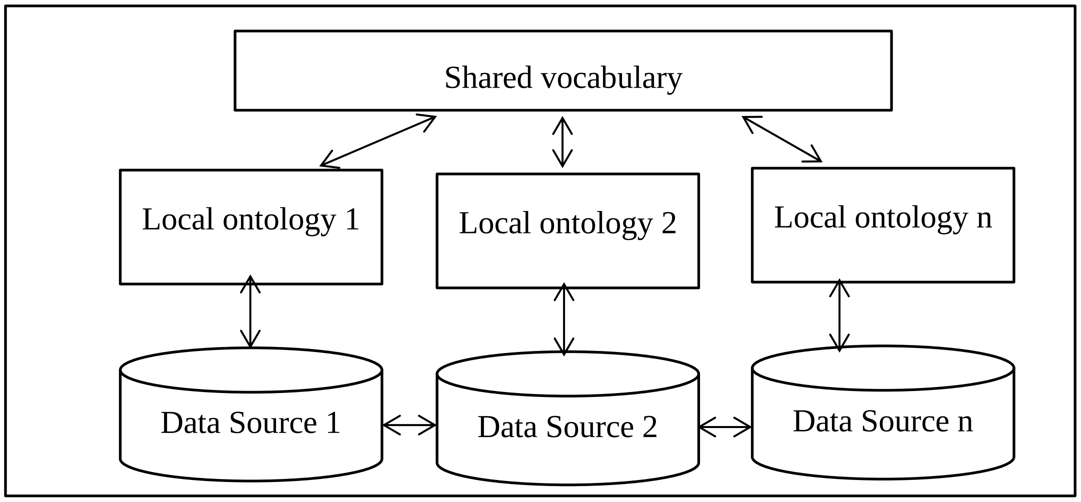


Fig. 3.4 Hybrid Ontology Approach

In the current work, multiple ontologies approach is used. Using this approach, a separate ontology is built with respect to every selected jobboard. A lot of work has been done in ontology management using above mentioned approaches. Among them, some of the existing ontology management techniques has selected for discussion in the next section.

3.6 ONTOLOGY MANAGEMENT METHODS

Ontology management [140] includes operations such as ontology integration [140], ontology merging [140] and ontology alignment [140]. *Ontology merging* is the process of generating a single coherent ontology from two or more existing and different ontologies related to the same subject. *Ontology alignment* is the task of creating links between two original ontologies. *Ontology integration* is the process of generating a single ontology in one subject from two or more existing and different ontologies in different subjects. The different subjects of different ontologies may be related.

3.6.1 Ontology Alignment Methods

Ontology alignment is the process of determining correspondences between concepts in ontologies. A set of correspondences is also called an alignment. There are three main dimensions for similarity- syntactic, semantic and structural, based on which it finds correspondence between two concepts or relations of two different ontologies. For example, one concept says ‘worker’ from one ontology *O1* and another concept ‘employee’ from other ontology *O2*. Syntactically, they are not similar but semantically, they are same as they are synonym to each other. So, with these similarity methods, alignment approach finds the correspondence between the concepts and relations of two different ontologies. Some of the prevalent ontology alignment methods have been discussed as below.

a) BLOOMS+ [141]

BLOOMS+ is an ontology alignment system based on bootstrapping information already present on LOD (Link on Data) cloud. It utilizes the Wikipedia category hierarchy for aligning ontologies. BLOOMS construct a forest (i.e. a set of trees) T_C (BLOOM forest for concept *C*) for each matching candidate class name C_i. It tokenizes the name of *C* and removes stop-words from the name and then it gives resulting terms as a search string to retrieve relevant Wikipedia pages using Wikipedia search web service. BLOOMS+ treats each page as a possible sense S_i of *C* and constructs a category hierarchy tree. It then compares each class *C’s* forest T_C in the source ontology with each class *D’s* in forest T_D in the target ontology to determine their similarity. Once the class similarity has been determined, it then computes contextual similarity. It uses superclass of *C* and *D* to determine if they are contextually same. Using class similarity and context similarity, BLOOMS+ finally determines whether *C* & *D* should be aligned.

b) ASMOV [142, 143]

It is short for *Automated Semantic Matching of Ontologies with Verification.* This method uses lexical and structural characteristics of two ontologies to iteratively calculate a similarity measure between them. It derives an alignment and then verifies to ensure that it does not contain semantic inconsistencies. It retrieves as input two ontologies to be matched. ASMOV process is an iterative process and is divided into two components: similarity calculation and similarity verification.

- The *similarity calculation* process computes a similarity value between all possible pairs of entities, one from each of the two ontologies using four similarity measures: lexical similarity, structural similarity, restriction similarity and extensional similarity. This process results in a similarity matrix containing the calculated similarity values for every pair of entities. From those similarity matrices, a pre-alignment is extracted by selecting the maximum similarity value for each entity.
- This pre-alignment is passed through a process of *semantic verification* which eliminates correspondences that cannot be verified by the assertions in the ontologies. Semantic verification process uses multiple entity correspondence, crisscross correspondence, disjointness subsumption, contradiction subsumption, equivalence incompleteness and domain range incompleteness for verification.

c) CIDER [144]

It is short for *Context and Inference baseD alignER (CIDER),* an ontology alignment system that extracts the ontological context of the compared terms by using synonyms, hyponyms, domains, etc. and then enriches such context by means of some lightweight inference rules. It performs similarity by first extracting the ontological context of each ontology term up to a certain depth (using synonym, hypernym, hyponym, textual description, properties, domains, roles, associated concepts etc.) using lightweight inference mechanism to add more semantic information that is not explicit in the asserted ontologies. Then, it uses linguistic (using Levenhstein method) and structural similarity (using vector space model) to find the similarity between each pair of terms. After this, the different similarities are combined within an

Artificial Neural Network (ANN) to provide a final similarity degree. ANNs constitute an adaptive type of systems composed of interconnected artificial neurons, which change the structure based on external or internal information that flows through the network during a learning phase. CIDER uses two different neural networks for computing similarities between classes and properties, respectively. Finally, a matrix *M* with all similarities is obtained. The final alignment *A* is then extracted from this matrix *M*, finding the highest rated one-to-one relationships among terms, and filtering out the ones that are below the given threshold.

d) RiMoM [145]

This multi-strategy ontology alignment framework aims at finding the optimal alignment by combining different strategies. It uses five strategies- *edit distance based strategy, statistical learning based strategy* for linguistic matching and three similarity propagation based strategies (including *concept to concept propagation strategy, property propagation strategy and concept to property propagation strategy*) for structural matching. If two ontologies have high structure similarity factors, then RiMoM employs an algorithm called similarity propagation to refine the discovered alignments.

e) COMA 3.0 [146]

COmmon MAtcher (COMA) is a schema and ontology matching tool. It has four modules where the three modules *storage, match execution* and *mapping processing* follow the input-processing-output pattern and the *user connection module* provides different ways to access the program. The *storage* consists of the importers that load schemas, ontologies, existing mappings and auxiliary information in the repository. From repository, these files can be directly used to carry out matching task. The *match execution* is the core of COMA. It gets two schema or ontologies as input, runs several matching algorithms on those ontologies and calculates the match result. In this module, the execution engine determines the relevant schema components for matching, applies multiple strategies and finally combines the partial results to the final match result. The obtained mappings are further used as input in the next iteration for further refinement. The match library is a large bundle of schema matching strategies that can be combined to extensive workflows. The *mapping* processing module allows automatically enriching mapping, merging module or transforming data. The *user connection module* consists of full-fledged GUI to provide convenient way to use COMA.

f) YAM++ [147]

YAM++ is a semi-automatic mapping tool which maps two ontologies at three levels. At the first level, which is known as *elementary level*, it uses machine learning based combination methods such as decision tree, SVM, Naive Bayes etc. For this, it takes training data either from the user or from knowledge base. After this, at the second level named as *structural level*, input ontologies are parsed and transformed into graph data structure. For this, YAM++ takes elementary level mapping results as input and runs a similarity flooding algorithm to run a similarity propagation process. Finally, at the third level it performs *semantic checking* where it uses global constraint optimization. The resultant mapping of the match process is displayed at the GUI and then user judges if the mapping is correct or not according to his/her knowledge.

g) SIMTSS [148]

This method forms alignment between ontologies written in different languages such as RDF, SKOS, turtle etc. including heterogeneous information. The result is new data stored as an XML file stored in inference phases (query answering and integrating data). The system is divided into five layers. The first layer called *Resource layer* contains a collection of ontologies written in different languages. The system integrates all the ontologies in the matching process by mapping only the entities (concept, instances, and properties). In the *pre-processing layer*, ontologies written in different languages are standardized to OWL and then are normalized (lemmatization, lower case conversion, stop words and delete links). After this process, these ontologies are moved to the *matching process layer*. It aims to find first the relationship between their entities and degree of similarity by calculating the similarity measure. It measures the similarity at three levels: terminological, structural and semantic. Different methods are used at each level for similarity measurement and correspondingly generate measures in matrix format. This matrix is given as the input to the *extracting alignment layer* where an algorithm, *Hungarian algorithm,* is applied which highlights the most correct matches and eliminates less relevant once. The obtained alignments are stored as an XML file containing the two entities matching similarity relationship and similarity values between them. At last, this file is passed to the *expert and configuration layer* where expert confirms and suggests another alignment; and finally configures the output by using available tools.

h) MAPSS [149]

It is an ontology alignment system that uses syntactic, structural and semantic metrics. This method has evaluated wide range of string similarity metrics along with string preprocessing strategies on different type of ontologies. It mainly concentrates on following points:

- which effective string similarity metric for ontology alignment to choose if the primary concern is precision, recall and f-measure,
- how to automatically select which string similarity metric and pre-processing strategies are best without any training data available,
- It has grouped string metrics along three major axes: Global versus local, set versus whole string and perfect sequence versus imperfect sequence. *Global versus local* refers to the amount of information the metric needs to classify a pair of strings as match or a non-match. Global metrics must compute some information over all the strings in one or both ontologies before it can match any strings whereas for local metrics it only requires only input string. *Perfect sequence metrics* require characters to occur in the same position in both strings in order to be considered a match. *Imperfect sequence metrics* equate matching characters if their positions in the string differ by less than some threshold. A *set based string metric* works by finding the degree of overlap between the words contained in two strings. *Word based set metrics* are generally perform well on long strings,
- For preprocessing, it has divided the categories in two major categories: syntactic and semantic. Syntactic pre-processing methods are based on the characters in the

strings such as tokenization, normalization, stemming, stop-word removal. Semantic methods relate to the meaning of the string.

i) SEM+ [150]

This is similarity based entity matching method, which implements a novel semantic computation model called the information entropy and weighted similarity model to suggest similarity measures between concepts from different ontologies and vocabularies. Based on the similarity measures, SEM+ creates “same as” links among those concepts. SEM+ also implements a new prefix based blocking algorithm, which groups possible matching pairs into one block. This blocking algorithm reduces the number of concepts pairs that are needed for similarity computation, which is useful when it is required to perform mapping between two large domain ontologies. The prefix blocking groups concepts that are likely to be similar to each other into one block and dissimilar concepts into different blocks based on literal description of the concepts such as rdfs:label, rdfs:comment. SEM+ builds an indexer of these literals and computes the concept frequency of words appears in the literal description and then compares only the prefix of concepts. Similar concepts come in one block and thus prefix of that block get associated with the block. With this approach, similar concepts come in one block which reduces the similarity computation between each concept. For concept matching, it uses information entropy and weighted similarity model.

j) MEDLEY [151]

MEDLEY is an ontology alignment system that uses lexical and structural methods to compute the alignment between classes, properties and instances. It also uses an external dictionary to tackle the problem of having concepts expressed in different natural languages. In the primary step, each entity in the first ontology is aligned with each entity in the second. In lexical metrics, it uses q-gram and levenshtein measure to calculate the similarity measure between nodes and then structural treatment is applied. For this, if an entity belonging to a given ontology has a neighbor that is always a part of alignment set then the node, that neighbor is aligned to, must be a neighbor of any prospective match for this entity.

k) RiMOM-IM [152]

The main idea behind the framework is to maximize the utilization of distinctive and available matching information to handle large scale instance matching tasks in an iterative way. It has proposed a new blocking method which uses predicate and their distinctive object features to select candidate instance pairs and unique instance set which effectively reduces the running time. For each candidate set, similarities over all aligned predicates with similarity over predicates and then through aggregation, final matching score of two instances is computed. For unique instance sets, it iteratively uses unique subject matching and one left object matching to generate aligned set until no new matching pairs are generated.

In the next section, the Ontology Merging methods proposed in the recent past have been reviewed.

3.6.2 Ontology Merging Methods

The process of creation of a new ontology from two or more existing ontologies belonging to same domain is known as *ontology merging*. For instance, say one ontology say *O1* contains the information of ‘cars’ in the context of ‘brand’ and another ontology say *O2* also explains information of car but in the context of ‘price’. By merging these two ontologies *O1* and *O2*, coverage area of car information can be extended and can be further used for annotation.

A number of ontology merging methods have been proposed by various researchers out of which some of the prevalent methods are discussed as below.

a) Chimaera [153]

Chimaera was developed at Knowledge Systems Laboratory at Stanford University to aid users for browsing, editing, merging and diagnosing of ontologies. It is built on top of the *Ontolingua* Distributed Collaborative Ontology Environment. The project started with keeping the goal is to develop a tool that can give substantial assistance for the task of merging knowledge bases produced by different users for different purposes with different assumptions and different vocabulary. Later, the goals of supporting testing and diagnosing ontologies arose as well. Chimaera merges two semantically identical terms from different ontologies so that they can be referred to by the same name in the resulting ontology. It identifies terms that are related via *is-a, disjointness or instance relationships* and provide support for introducing those relationships. Chimaera also supports the identification of the locations for editing and performing the edits. To assist the user, Chimaera generates name resolution lists that suggest terms that are candidates to be merged or to have taxonomic relationships not yet included in the merged ontology. It also generates a taxonomy resolution list where it suggests taxonomy areas that are candidates for reorganization. Based on these lists, user decides what should be done.

b) ATOM [154]

ATOM is an asymmetric merge approach that gives preference to the target taxonomy. In preliminary phases, it takes two taxonomies O_s and O_t and a mapping between them, provided by the set of concept correspondence and attribute correspondence. Its goal is the generation of an integrated concept graph. The main contribution of this work is new target-driven algorithm that automatically integrates taxonomies. The base algorithm takes as input two taxonomies and an equivalence matching between concepts. The algorithm generates taxonomies that preserve all instances of the input taxonomies as well as the structure of the target taxonomy. In contrast to previous work of ATOM, it does not necessarily preserve all source concepts but aim at limiting the semantic overlap in the merged taxonomy for improved understandability. This is achieved by utilizing the input mapping and giving preference to the target taxonomy when the same concepts are differently organized in source and target.

c) SAMBO [155]

This system is designed for Aligning and Merging Biomedical Ontologies. It is an alignment method for defining the relationship between terms in different ontologies and creating a new ontology containing the knowledge included in the source ontologies. The framework of SAMBO consists of two parts. The first part computes alignment suggestion. The second part interacts with the user to decide on the final alignments. The alignment algorithm receives as input two source ontologies.

Alignment suggestions are then determined by combining and filtering the results generated by one or more matchers. The suggestions are then presented to the user who accepts or rejects them. SAMBO contains five basic matchers: two terminological matchers, a structure-based matcher, a matcher based on domain knowledge and a learning matcher for terminological matching. It uses n-gram and edit distance and linguistic algorithm. Structural matchers are based on *is-a* and *part-of* hierarchies of ontologies. This algorithm checks if two concepts lies in the similar position with respect to is-a or part-of hierarchies relative to already aligned concepts in the two ontologies, then they are likely to be similar as well. SAMBO matcher uses UMLSK search that uses the meta-thesaurus in the Unified Medical Language System. The fifth matcher is learner matcher which It is based on the intuition that a similarity measure between concepts in different ontologies can be defined on the probability that documents about one concept are also about the other concept and vice-versa. SAMBO uses Naive Bayes classification algorithm.

d) HCONE [156]

It is short for Human-Centered Ontology Engineering. The goal of the approach is to validate the mapping and to find the minimum set of axioms for the new merged ontology. This approach is based on:

- capturing the intended informal interpretation of concepts by mapping them to wordnet [166, 167] senses using lexical semantic indexing and
- exploiting the formal semantics of concepts definition by means of description.

In this approach, ontology concepts are being mapped to WordNet senses. Using this mapping, HCONE merge constructs from the intermediate ontology that includes- a *vocabulary* with the lexicalization of the specific senses of WordNet synsets corresponding to the ontologies concepts and *axioms* that are translated axioms of the original ontologies. Having specified the mappings to the hidden intermediate ontology, the translated ontologies are merged following some merge actions such as rename, merge and classify.

e) PROMPT [157]

PROMPT is based on ontology-merging and ontology-alignment algorithm. It takes two ontologies as input and guides the user to generate a merged ontology as an output. It creates an initial list of matches based on class names and then the user triggers an operation by either selecting one of PROMPT's suggestions from the list or by using an ontology-editing environment to specify the desired operation directly.

f) Ontology Merging by Clustering & Inference Mechanism [158]

This method is based on the combination of statistical aspects represented by hierarchical clustering techniques and the inference mechanism. It generates global ontology automatically by four steps:

- It builds class of equivalent entities of different categories (concepts, properties, instance) by applying a hierarchical clustering algorithm.
- It makes an inference on detected classes to find new axioms and solves synonymy and homonymy conflicts. It also generates a set of concept pairs from ontology hierarchies.
- It merges different sets together and uses classes of synonyms and sets of concept pairs to solve semantic conflicts in the global set of concept pairs.
- Finally, it transforms this set to a new hierarchy which represents the global ontology.

In the next section, some of the popular Ontology Integration approaches have been discussed.

3.6.3 Ontology Integration Tools

The process of creation of new ontology by combining existing ontologies belonging to different domains is known as ontology integration. For example, combining ontology *A* of music domain and ontology *B* of singer domain and forming ontology *C* will hold the knowledge of songs along with their singer's information thereby expanding the coverage area by using existing knowledge available on the web. Below are presented some prevalent methods proposed by researchers in this area.

a) Merging of cross-domain lexical ontologies [159]

This method integrates multi-lingual thesaurus (AGROVOC, EUROVOC, GEMET, UNESCO, URBISOC thesaurus) in order to build a first draft of domain ontology in urbanism. The goal is to extract concepts and semantic relations from terms and linguistic relations. This method merges the knowledge from different domains to obtain a better definition of the urban domain. The process is composed of several steps:

- Initially, system takes as input a set of thesaurus of different knowledge area and transforms them in the same format to avoid from format related issues that may arise during the merging process.
- Once, thesauri get transformed in the common format, the next objective is to extract the concepts related to urbanism from the analyzed thesauri. For this, it uses linguistic similarity between the concepts for mapping. In the mapping process, every concept of every thesaurus is compared with every concept of the other treasures to find equivalence. Each set of mapped concepts is grouped into a cluster which is identified with the one of the URI of the original concepts.
- The clusters generated in the previous step describe the urban terminology used in different knowledge area. Now, the next task is to build a relation between these clusters to generate a network of urban concepts that can be seen as an urban ontology.
- For this, relations of the concepts contained in each cluster are used as a basis for the generation of the relations between clusters. Finally, to facilitate the visualization and reusability of the generated output, it is transformed into XML and OWL formats.

b) Integration of different web portals [160]

This technique combines domain ontologies and semantic web services to provide an integrated access to the information provided by different web portals. In order to provide this functionality, it provides a user interface that allows users to express their query using an ontology guided tool which assist users to express their goals. The domain ontology is loaded through the Protégé OWL API and its main concepts are used to form a simple menu where the user can choose the type of the objects they are looking for.

Through the query component, the system searches and selects the most appropriate web services by accessing their semantic description.

The comparison of various ontology management methods is performed on various parameters like operation, input, output, knowledge source and concept matching methods etc. The detailed comparison study is outlined in Table 3.4.

Table 3.4 Comparative Study on Various Ontology Management Tools

S. No.	**Ontology Mgmt. Methods**	**Operation**	**Input**	**Output**	**Knowledge source**	**Concept matching methods**	**Language**
1	**BLOOMS+**	Ontology Alignment	two ontologies	alignment between those ontologies	Wikipedia pages	Retrieves synset of each concept from Wikipedia pages and uses them as context of that concept.	OWL
2	**ASMOV**	Ontology Matching	two ontologies	alignment between those ontologies	set of input alignment containing a set of predetermin ed corresponde nce.	Lexical similarity, structural similarity, restriction similarity and extensional similarity	OWL
3	**CIDER**	Ontology Alignment	two ontologies	alignment between those ontologies	wordnet	Uses ANN for final similarity measure by combining semantic, lexical and structural similarity.	OWL
4	**RiMOM**	Ontology Alignment	two ontologies	alignment between those ontologies	None	Lexical and structural similarity.	OWL
5	**COMA 3.0**	Ontology Matching	two ontologies		None		OWL

				alignment between those ontologies			
6	**YAM++**	Ontology mapping	two ontologies	mapping between two ontologies	training data at elementary level	Machine learning based method, structural and at last semantic matching.	OWL
7	**SIMTSS**	Ontology Alignment	two ontologies	XML file	None	terminological, structural and semantic.	RDF, SKOS, turtle
8	**MAPSS**	Ontology Alignment				syntactic, structural & semantic metrics	
9	**SEM+**	Ontology Alignment	two ontologies	alignment b/w those ontologies	None	information entropy & weighted similarity model	OWL
10	**MEDLEY**	Ontology Alignment	two ontologies	alignment between those ontologies	external dictionary	lexical and structural methods	OWL
11	**RiMOM-IM**	instance matching	two ontologies	alignment between those ontologies	None	Finds similarity over aligned predicates for instance set, uses unique subject matching.	OWL
12	**Chimarea**	Ontology merging	initially two knowledge bases, later on two ontologies	merged ontology	None	Identifies similarity via is-a, disjointness or instance relationships between two terms.	initially knowledge bases, later on OWL
13	**SAMBO**	Ontology alignment and merging	two ontologies	merged ontology	None	A structure-based matcher, a matcher based on domain knowledge	OWL

						and a learning matcher for terminological matching	
14	**HCONE**	Ontology merging	two ontologies	merged ontology	Wordnet	Semantic matching	OWL
15	**PROMPT**	Ontology merging	two ontologies	merged ontology	None	Concept name string matching	OWL
16	**Ontology merging by clustering & inference mechanism**	Ontology merging	two ontologies	merged ontology	Wordnet	Terminological and structural matching	OWL
17	**Merging of cross domain lexical ontologies**	Ontology Integration	Thesaurus of different knowledge area	merged ontology	None	Linguistic matching	different knowledge bases of different formats

3.7 SUMMARY

After going through the literature review on current web and semantic web, it has been found that in the last few years, due to the popularity of semantic web, developers have started representing their knowledge in structured format using semantic web languages constructs. But, current web is also carrying a huge amount of relevant unstructured data. If this huge repository of unstructured data can be represented into structured form, then more relevant information can be provided to the user. For instance, in the domain of job, there are hundreds of Jobboards which facilitates jobs to its users. But, these Jobboards uses keyword based matching system for retrieving the outputs corresponding to user's query. If the semantic web technologies could be included with these existing knowledges available on the web, then coverage area of semantic web can be widened. But, only few research efforts have been found where unstructured and semi-structured data has been tagged with the semantic web made metadata. Therefore, there is a requirement to create an Information System that uses existing webpages as input, represent them in structured form using ontologies and then aligning various websites of same domain and upgrading the search systems.

In the next chapter, architecture of Ontology driven Information System for Semantic Web is presented which provides job information for its users at one place by extracting semi-structured data from different Jobboards. The descriptions of various components of proposed Information system are discussed in subsequent chapters.

Chapter IV

JOBOLOGY: SEARCH SYSTEM FOR PROVIDING RELEVANT JOBS USING ONTOLOGY

4.1 GENERAL

In the world of internet, Job portals / Jobboard sites are like the meeting point for the recruiters as well as the job seekers where both aims at meeting their individual requirements. Job seekers try to find a job opportunity from these Jobboard. But, there are a number of difficulties through which job seeker go through while accessing these Jobboard sites which are discussed as below:

a) Multi-Registration

The present market is too much crowded with different job portals. Therefore, to grab the best opportunities, job seeker registers himself with most of the sites which makes the process cumbersome for him.

b) High noise low output

Currently, almost every portal works based on keyword matching. Therefore, if a job seeker is looking for advanced java jobs which generally indicate 'advanced java' as a skill. It retrieves even those jobs also which require core java skills yielding irrelevant results to the user.

c) Irrelevant results

Sometimes, when a user searches a job with designation, for instance "project manager", the search will bring up results in a number of different sectors and possibly different locations.

Therefore, looking at the above issues, a semantic search based "Jobology" framework has been proposed which retrieves relevant jobs to jobseekers depending upon his information need by pulling up results from different Jobboard sites in one go.

4.2 JOBOLOGY SEARCH SYSTEM

The proposed Jobology search system functionality includes

- Deploying the strategy for crawling [161] the domain specific semi-structured web pages.
- Converting the extracted semi-structured data into structured format using ontology.
- Provision for alignment between ontologies belonging to different data sources.
- Applying semantic query on the integrated data sources.

The stepwise details of the design of the proposed system to achieve the proposed research objectives have been depicted in Fig. 4.1.

Step 1: Development of ontology in a particular domain

In the first step, ontology with respect to each selected Jobboard site was developed. For the present research, OWLAPI [120] is used with Java platform for designing the ontologies and protégé software tool which is an open source tool is used for visualizing the ontologies. Each Jobboard ontology deals with the information related to job profile such as job titles, job location, job keyskills, job salary package, job experience, job description, type of job etc.

Step 2: Development of query based URL builder

Query based URL builder builds URLs of the data sources/ webpages from where desired data should be extracted. This eliminates visiting of undesired webpages of Jobboard sites.

Step 3: Development of OntoJobExtractor module

The data extraction module extracts the desired data from the webpage with respect to each job post and stores it in a *repository*.

Step 4: Annotation of extracted data with Jobboard ontologies

This module annotates the extracted data using the ontologies created with respect to Jobboard sites.

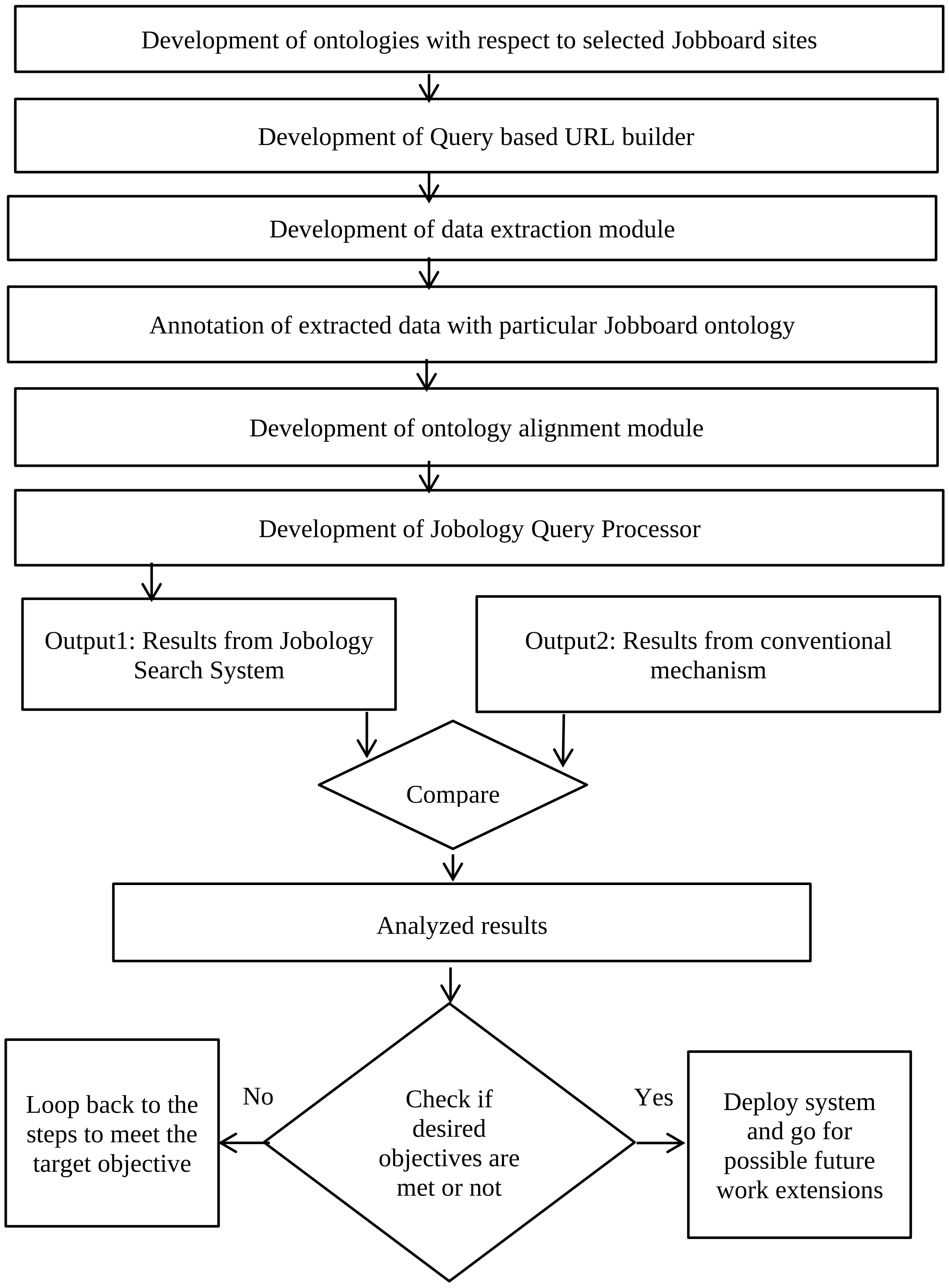

Development of ontologies with respect to selected Jobboard sites
Development of Query based URL builder
Development of data extraction module
Annotation of extracted data with particular Jobboard ontology
Development of ontology alignment module
Development of Jobology Query Processor
Output1: Results from Jobology Search System
Output2: Results from conventional mechanism
Compare
Analyzed results
Check if desired objectives are met or not
No
Yes
Loop back to the steps to meet the target objective
Deploy system and go for possible future work extensions

Fig. 4.1 Proposed Research Objective of The Proposed System

Step 5: Development of ontology alignment process

The ontology alignment module aligns the ontologies of same domain. It generates some global data structures which are used during query processing.

Step6: Development of OntoJob query processing module

This module translates user query into SPARQL queries with respect to each ontology which in turn are submitted on ontologies to retrieve matching jobs with respect to the query from all the ontologies at one place.

Step7: Comparison of outputs

The results obtained from Jobology search system thereof form the output set 1. The same query when submitted to Job boards retrieves the results forming the output set 2. These outputs are then compared and the results are analyzed for the input queries.

Step8: Check if the desired objectives are met

The results of the developed system are compared with conventional system. If the objectives are met, the system will be deployed and the possible future extensions of the work can be carried out otherwise the system needs to be modified with different perspectives.

4.3 FUNCTIONAL DIAGRAM OF THE PROPOSED SYSTEM

The macro architecture of the proposed system as two phase diagram is given in Fig. 4.2.

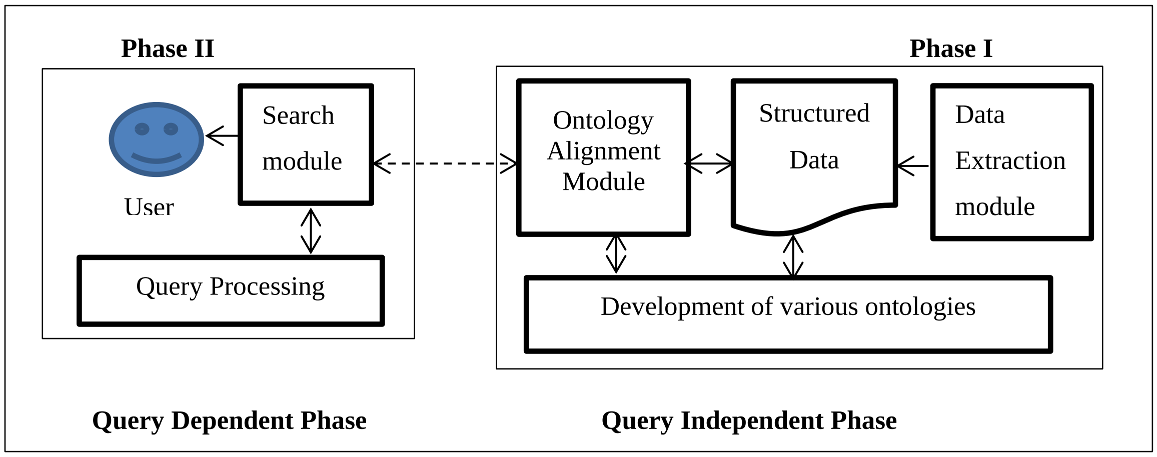


Fig. 4.2 Functional Diagram of the Proposed System "Jobology"

Fig. 4.2 depicts the two-phase development of the system where phase1 is query independent and phase II depicts query dependent phase. The query dependent phase includes mapping keywords of user query with the concepts and generates automatic

SPARQL queries [17, 18] with respect to every job board. The results are merged and presented to user in the sorted order at the same platform according to user's preferences. Query independent phase includes the development of ontology using OWLAPI [120] in Java platform and protégé development tool for visualization, extracting semi-structured relevant data from the webpages and converting them into structured data by annotating the semi-structured data with ontologies.

In this chapter, the proposed framework Jobology search system has been discussed. This chapter discusses architecture of searching semi-structured web pages of Jobboard domain which are annotated with the knowledge representative techniques called ontology. The ontologies are well represented with semantic web languages RDF [15, 16], OWL [118] etc. and can be created using various open source commercial tools like protégé [81], ALTOVA [162] semantic works. The annotation of semi-structured content with ontologies help machines to understand the semantic information represented and henceforth results into a more accurate retrieval of results for a query. The implementation of the proposed approach in the forthcoming chapters indicates that web information can be represented well with ontologies leading to better information retrieval. The research carried out envisions an approach of annotating semi-structured contents from Job boards only, which can be extended in future to include company's recruitment webpages and other resources also to widen the scope of more job opportunities to the job seekers.

4.4 COMPONENT DETAILS OF JOBOLOGY SEARCH SYSTEM

The proposed architecture consists of the following functional components.

- Ontology development module
- Data Extraction module
- Ontology Alignment module
- Query processor module.
- Search module

The detailed architecture of Jobology search system is shown in Fig. 4.3.

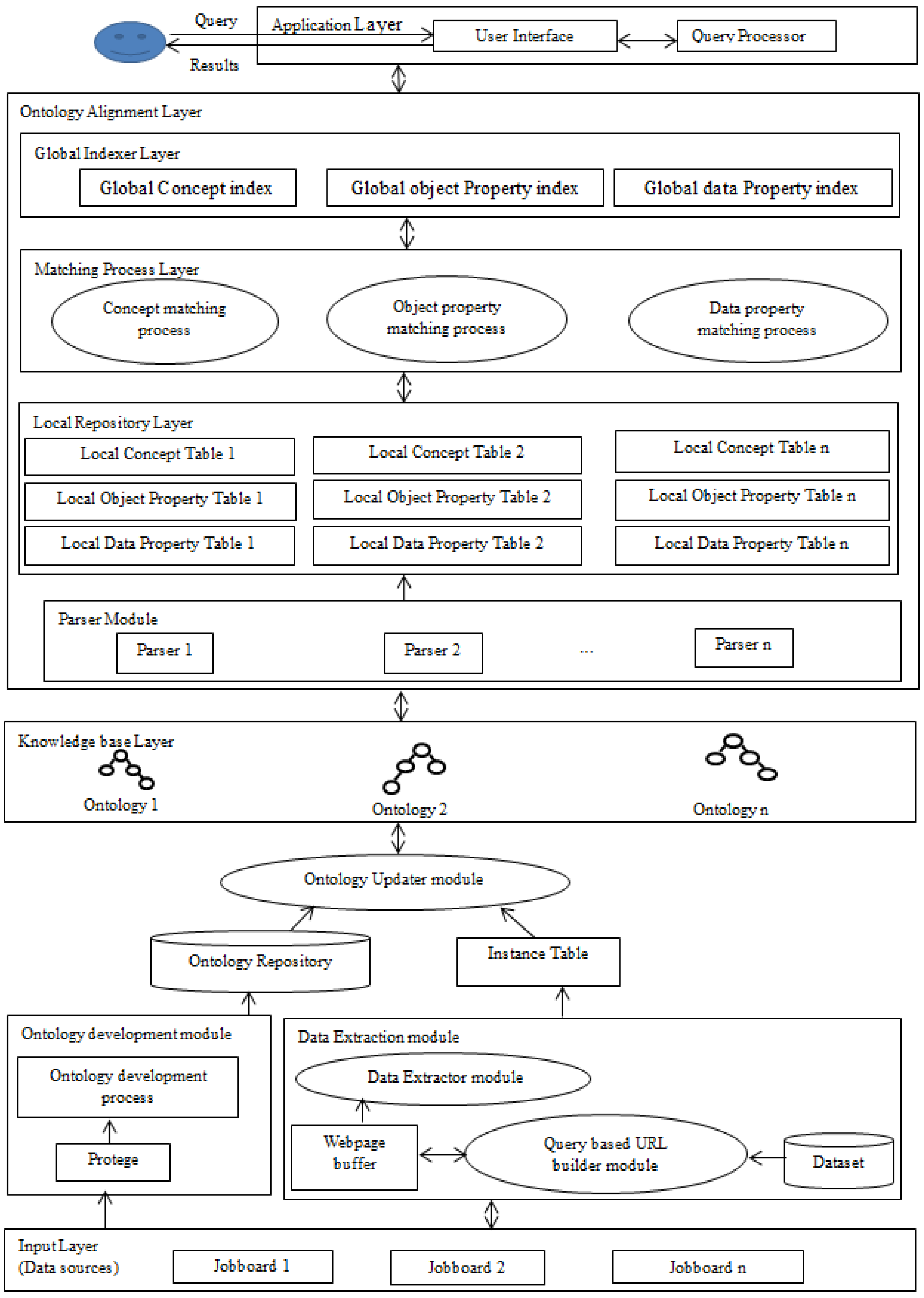


Fig. 4.3 Proposed Design of Jobology Search Engine

Each component of the proposed Jobology search system has been discussed in brief in this chapter and details of each component with implementation are discussed in the subsequent chapters.

4.4.1 Ontology Development Module

Large numbers of development frameworks are available for ontology engineering like protégé [81], Ontostudio [163], SWOOP [99], NeON toolkit [165], Altova semantic works [162] etc. In the current research, protégé development framework has been used for development of ontology in the Job domain. In this, with respect to every job boards, an independent ontology is developed consisting of specific concepts and properties using protégé tool. The consistency of the developed concepts can be checked by the different available reasoners. Different plugins reasoners available for protégé framework are pellet [92], fact++ [87], Hermit [88], RACER [94, 95] etc. Pellet reasoner is used to check the consistency of the concepts used. Query retrieval can be done in protégé framework using DL (Description Logic) Query [164], SPARQL (SPARQL Protocol and RDF Query Language) [17, 18]. SPARQL Query Language has been used to retrieve and manipulate data. This module has been discussed in detail in Chapter V.

4.4.2 Data Extraction Module

This is an important module that extracts relevant data from the desired webpages of Job boards and annotates them using ontology. The output of the data extraction module is given as input to the ontology alignment module. Fig. 4.4 shows the macro level algorithm for data extraction.

```
OntojobExtractor()
{
QueryURL Builder module();
Downloader module();
Selector module();
Data extractor module();
Ontology updater module();
}
```

Fig. 4.4 Pseudo Code for Ontojobextractor Module

Initially using QueryURL Builder process, first it creates URLs of the webpages which are to be visited and adds them in a queue. Then, Downloader process; visits and downloads the webpages. Selector process selects the webpage from the repository and forward to the data extractor module which extracts relevant content from the webpage and finally using ontology updater process, it transforms the semi-structured content into structured format. The detail of this module has been discussed in Data Extraction: A framework for populating ontology with instances of Jobboard sites in Chapter VI.

4.4.3 Ontology Alignment Module

This proposed ontology alignment module is responsible for developing alignment between various ontologies of same domain. It takes N number of data sources ontologies from the knowledge base side and develops global indexes which will be required during query processing. The source ontologies remain intact during this process. The algorithm for ontology alignment is depicted in Fig. 4.5 below which builds Global Concept Index, Global Object Property Index and Global Data Property Index. The detail of this module has been discussed in Chapter VII.

```
Ontology Alignment ()
{
While(empty(ontology reporsitory)) do
{
Ontology parsing module();
Build global concept index();
Build global object property index();
Build global data property index();
}
```

Fig. 4.5 Pseudo Code for Ontology Alignment Module

4.4.4 Search Module

The search module provides an interface through which user interacts with the Jobology search system. It provides a form where user enters its query. This query is then forwarded to query processing interface for execution.

4.4.5 Query Processing Module

This module processes the query given by user in the form of keywords. It converts the keyword based query into SPARQL format. The algorithm for query processing module is described in Fig. 4.6. The detail of this module has been discussed in Chapter VIII.

```
Query Processing Module()
{
Input(search term)
Tokenize the search terms.
Match in the datasets.
Find the concept to which they belong.
Design SPARQL query.
Apply SPARQL query in ontologies.
Sort the results based on date/ relevance.
Display to the user.
}
```

Fig. 4.6 Pseudo Code for Query Processing Module

4.5 SUMMARY

In this chapter, the proposed framework Jobology Search System has been discussed. This chapter discusses framework of searching semi-structured web pages of job board domain which are annotated with the knowledge representative techniques called ontology. The annotation of semi-structured content with ontologies help machines to understand the semantic information represented and henceforth results into a more accurate retrieval of results for a query. The implementation of the proposed approach in the forthcoming chapters indicates that web information can be represented well with ontologies leading to better information retrieval. The research carried out envisions an approach of annotating semi-structured contents from Jobboards only, which can be extended in future to include company’s recruitment webpages and other resources also to widen the scope of more job opportunities to the job seekers using Jobology Search System architecture discussed in this research.

Chapter V

ONTOLOGY DEVELOPMENT IN THE DOMAIN OF JOBBOARDS

5.1 GENERAL

There are large numbers of Job boards available on the web which provides job information to the students. Data available on these Jobboards are semi-structured [24] in nature. To convert these semi-structured data into structured format [15] using semantic web technologies, ontologies need to be constructed. In this chapter using multiple ontology approach, ontologies with respect to Jobboards and Student are defined. Jobseeker ontology will be used in cross domain integration.

5.2 ONTOLOGY DEVELOPMENT FOR JOB BOARDS

Ontology is the study or concern about what kind of things exist- what entities are in the universe. Keeping this concept in mind, in the proposed system three jobboard sites named as *www.Naukri.com*, *www.Timesjob.com* and *www.Shine.com* are selected for the execution of the system. An individual ontology with respect to each site is developed. Student ontology is also developed which will be used in annotating the student profile. For the development of ontology [55], an iterative ontology development process as discussed in Chapter III has been followed.

The ontologies developed for research purpose for above mentioned Jobboards and student domains are:

a) Timesjob ontology
b) Shine Ontology
c) Naukri Ontology
d) Student Ontology

These ontologies have been discussed in detail as follows:

a) Timesjob ontology

This ontology contains concepts related to job entity presented to user by Timesjob.com like title of the job entity; its location etc. and various other properties are also covered. The steps followed for developing Timesjob Ontology are as follows:

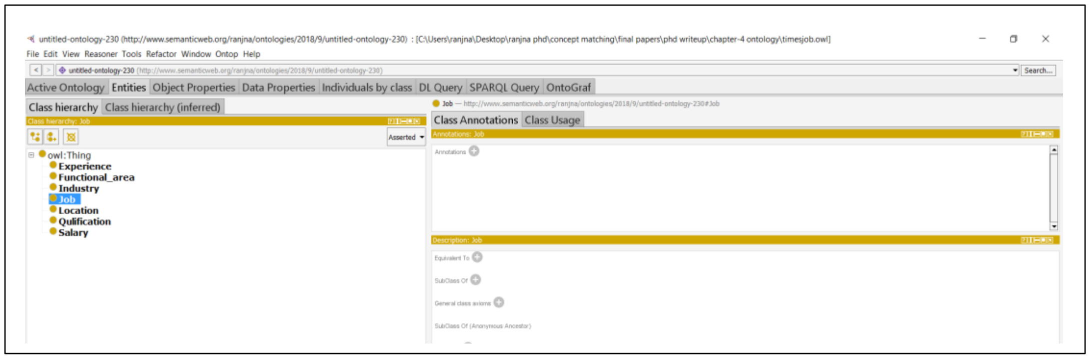
- **Identifying Concepts**

The different classes/ concepts for Timesjob ontology as retrieved from the corresponding Jobboard are depicted in Table 5.1.

Table 5.1 Classes for Timesjob Ontology

S. No.	Class
1	Job
2	Experience
3	Location
4	Functional Area
5	Qualification
6	Industry
7	Salary

The snapshot of Timesjob Ontology Class hierarchy depicting all these classes is shown in Fig. 5.1.


untitled-ontology-230 (http://www.semanticweb.org/ranjna/ontologies/2018/9/untitled-ontology-230) : [C:\Users\ranjna\Desktop\ranjna phd\concept matching\final papers\phd writeup\chapter-4 ontology\timesjob.owl]

owl:Thing
- Experience
- Functional_area
- Industry
- Job
- Location
- Qulification
- Salary

Job — http://www.semanticweb.org/ranjna/ontologies/2018/9/untitled-ontology-230#Job

Fig. 5.1 Snapshot of Class Hierarchy of Timesjob Ontology

- **Identifying Properties**

In this step, the properties that exist between different classes i.e. data properties which define the relation between a class and value of a class; and object properties that define the relation between two classes has been defined. The different properties for Timesjob ontology are depicted in Table 5.2

Table 5.2 Properties for Timesjob Ontology

Property	**Domain**	**Range**	**Type of Property**
hasCompany	Job	xsd:string	Data Property
hasTitle	Job	xsd:string	Data Property
hasDateofPost	Job	xsd:string	Data Property
hasSkillset	Job	xsd:string	Data Property
hasSpecialization	Job	xsd:string	Data Property
hasIndustry	Job	Industry	Object Property
hasQualification	Job	Qualification	Object Property
hasExp	Job	Experience	Object Property
hasLoc	Job	Location	Object Property
hasSal	Job	Salary	Object Property
hasIndustry	Job	Industry	Object Property
belongstojobfunc	Job	Functional_area	Object Property

The snapshot of Timesjob Ontology ObjectProperty and Data Property hierarchy is shown in Fig. 5.2 and Fig. 5.3.

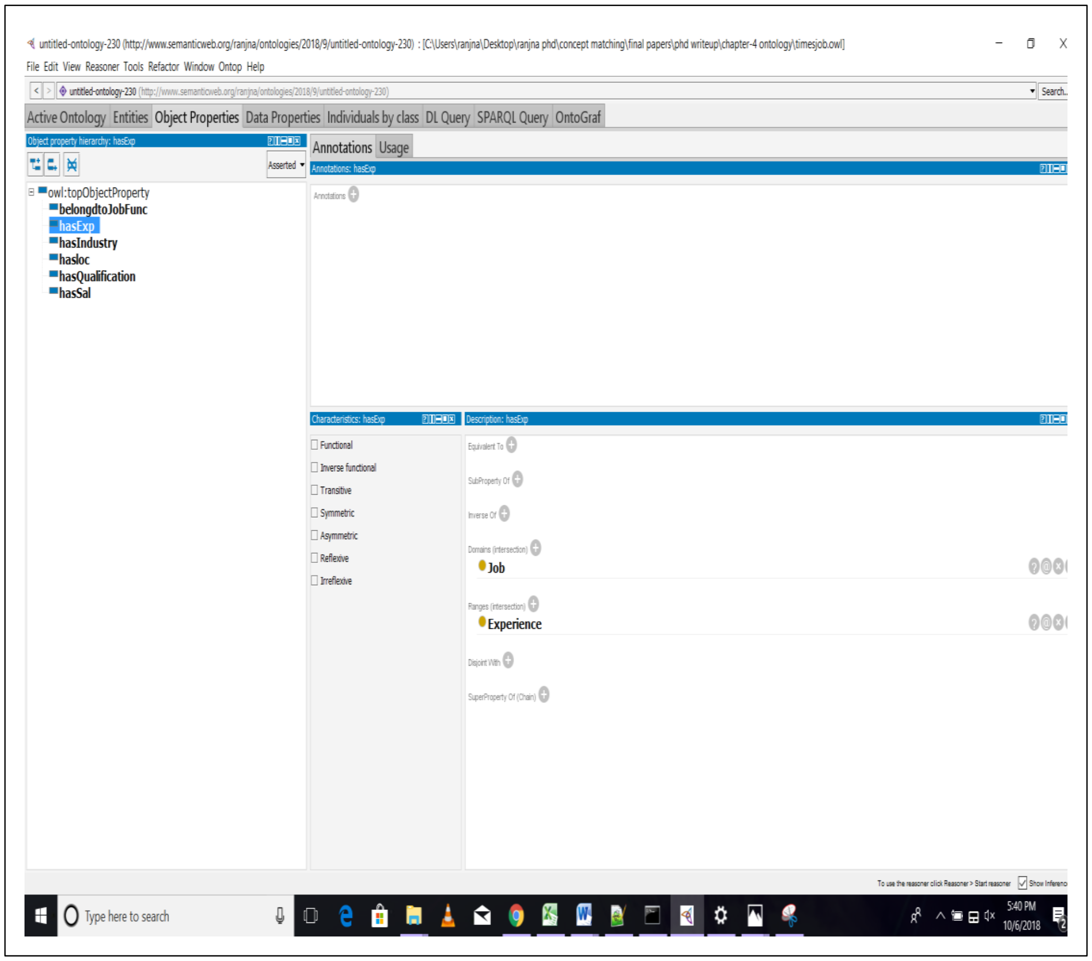


Fig. 5.2 Snapshot of Object Property Hierarchy of Timesjob Ontology

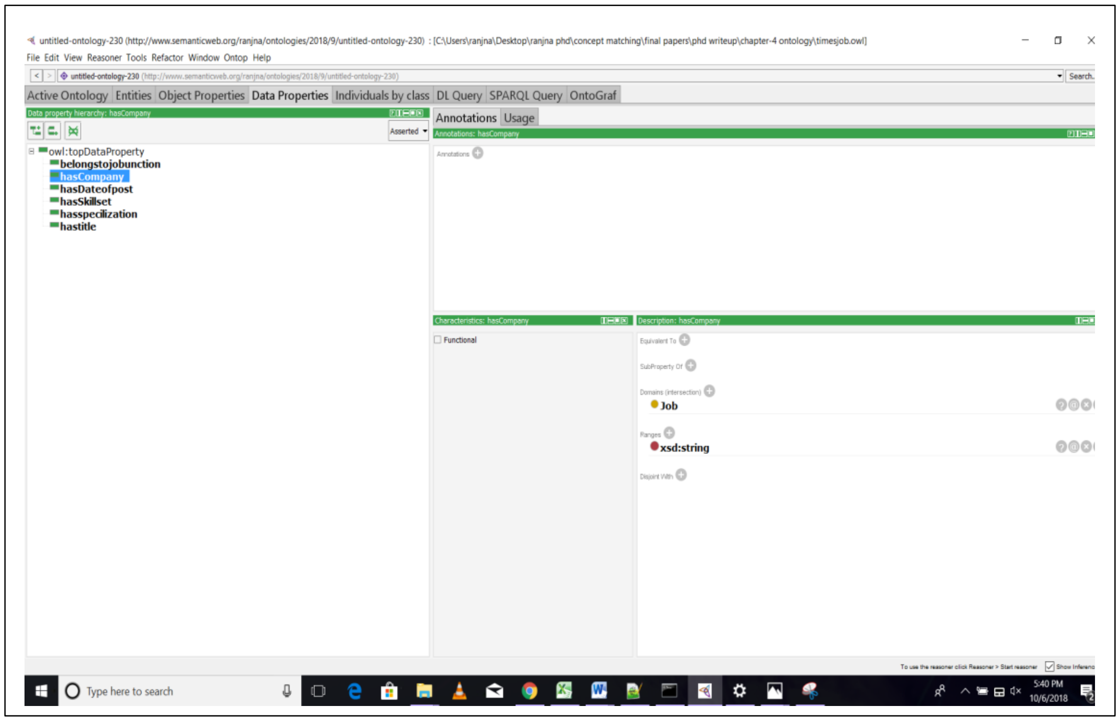

untitled-ontology-230 (http://www.semanticweb.org/ranjna/ontologies/2018/9/untitled-ontology-230) : [C:\Users\ranjna\Desktop\ranjna phd\concept matching\final papers\phd writeup\chapter-4 ontology\timesjob.owl]
File Edit View Reasoner Tools Refactor Window Ontop Help
Active Ontology Entities Object Properties Data Properties Individuals by class DL Query SPARQL Query OntoGraf
Data property hierarchy: hasCompany
Asserted
owl:topDataProperty
belongstojobunction
hasCompany
hasDateofpost
hasSkillset
hasspecilization
hastitle
Annotations Usage
Annotations: hasCompany
Characteristics: hasCompany
Functional
Description: hasCompany
Equivalent To
SubProperty Of
Domains (intersection)
Job
Ranges
xsd:string
Disjoint With
Type here to search
5:40 PM
10/6/2018

Fig. 5.3 Snapshot of Data Property Hierarchy of Timesjob Ontology

Fig. 5.4 shows the OntoGraph Visualizer of *Timesjob Ontology*. In this ontology, classes such as qualification, salary, location etc. along with various relationships that exist between classes and their values were developed which give the job information in Timesjob.com.

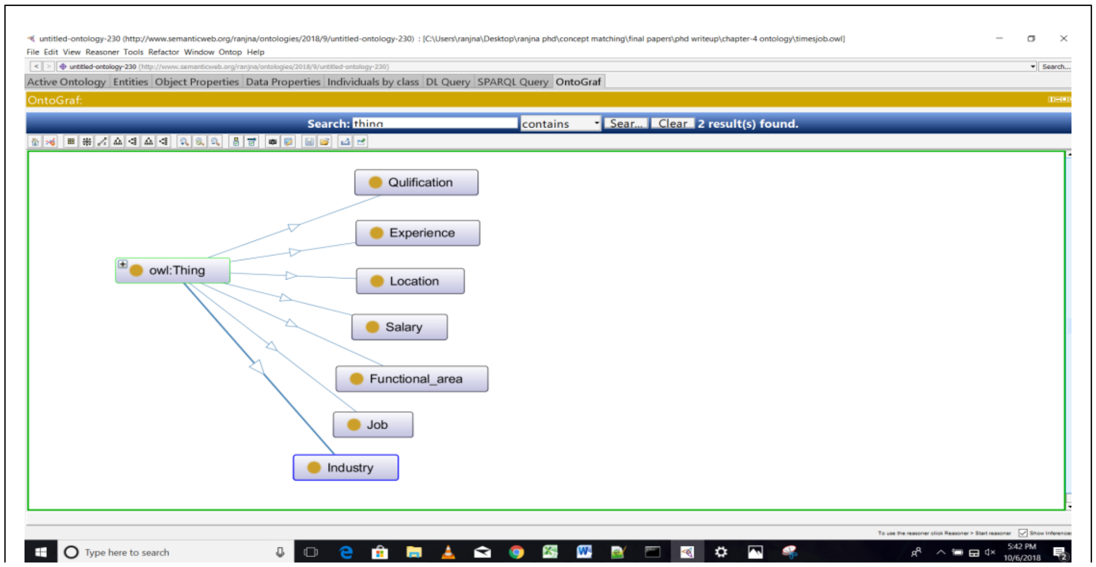

untitled-ontology-230 (http://www.semanticweb.org/ranjna/ontologies/2018/9/untitled-ontology-230) : [C:\Users\ranjna\Desktop\ranjna phd\concept matching\final papers\phd writeup\chapter-4 ontology\timesjob.owl]
File Edit View Reasoner Tools Refactor Window Ontop Help
Active Ontology Entities Object Properties Data Properties Individuals by class DL Query SPARQL Query OntoGraf
OntoGraf
Search: thing
contains
Sear...
Clear
2 result(s) found.
owl:Thing
Qulification
Experience
Location
Salary
Functional_area
Job
Industry
Type here to search
5:42 PM
10/6/2018

Fig. 5.4 Onto Visualizer Result of Timesjob Ontology

b) Naukri Ontology

This ontology creates a set of concepts representing the attributes of job post such as experience, title, designation etc. provided by the respective Jobboard.

- **Identifying Concepts**

The different classes for Naukri ontology are depicted in Table 5.3.

Table 5.3 Classes for Naukri Ontology

S.No.	Class	S. No.	Class
1	Education	6	Industry
2	Employment_type	7	Salary
3	Experience	8	Location
4	Functional_Area	9	Role
5	Job	10	Role Category

The snapshot of Naukri Ontology Class hierarchy containing the set concepts which will be used to annotate the information provided by Naukri.com job board is shown in Fig. 5.5.

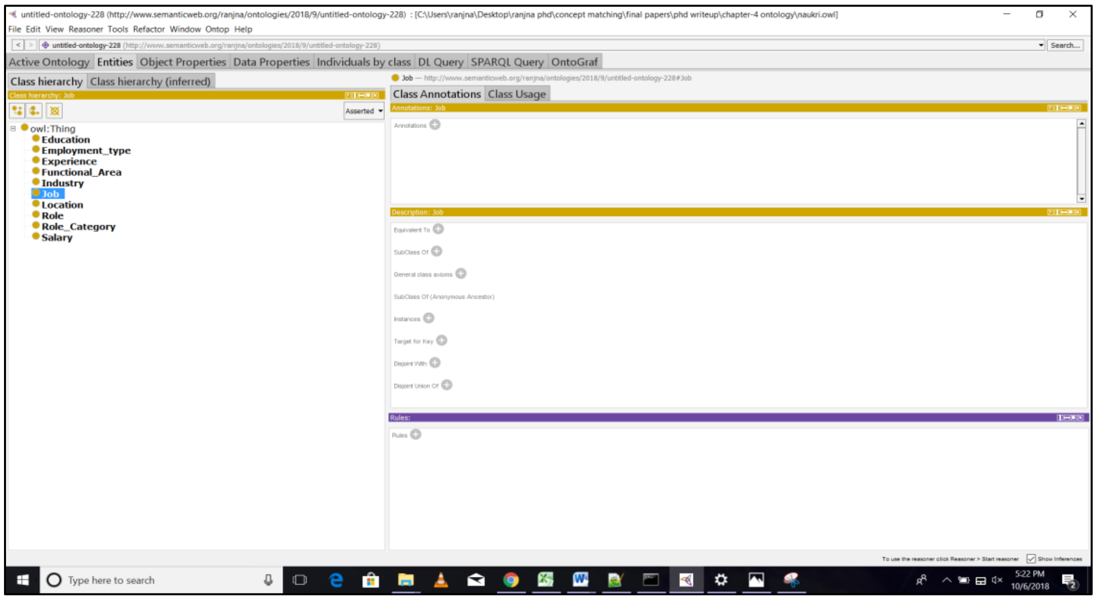


Fig. 5.5 Snapshot of Class Hierarchy of Naukri Ontology

- **Identifying Properties**

Once the conceptual model of the ontology has been defined, next step is to establish the relation between them. The different properties for Naukri ontology are depicted in Table 5.4.

Table 5.4 Properties for Naukri Ontology

Property	Domain	Range	Type of Property
hasEducation	Job	Education	Object Property
hasExperience	Job	Experience	Object Property
hasFunctional_area	Job	Functional_area	Object Property
hasLocation	Job	Location	Object Property
hasIndustry	Job	Industry	Object Property
hasRole	Job	Role	Object Property
hasRole_Category	Job	Role_Category	Object Property
hasSalary	Job	Salary	Object Property
hasDesignation	Job	xsd:string	Data Property
hasDate	Job	xsd:datetime	Data Property
hasDesc	Job	xsd:string	Data Property
hasKeyskill	Job	xsd:string	Data Property
hasOrganization	Job	xsd:string	Data Property
hasURL	Job	xsd:string	Data Property

The snapshot of Naukri Ontology ObjectProperty and Data Property hierarchy is shown in Fig. 5.6 and Fig. 5.7 . Fig. 5.6 shows the object properties that exist between two classes. For instance; hasSalary exist between class job and class salary. This property will exist between two class objects. For instance, post111 (an instance of job class) hasSalary 2 lac (an instance of salary class).

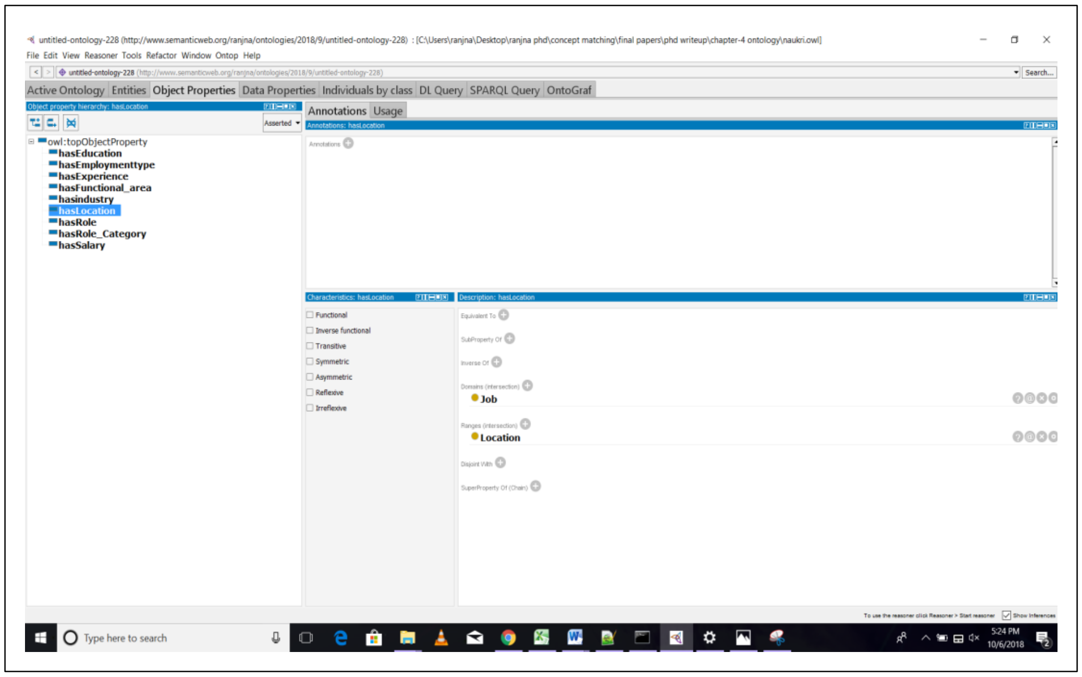


Fig. 5.6 Snapshot of Object Property Hierarchy of Naukri Ontology

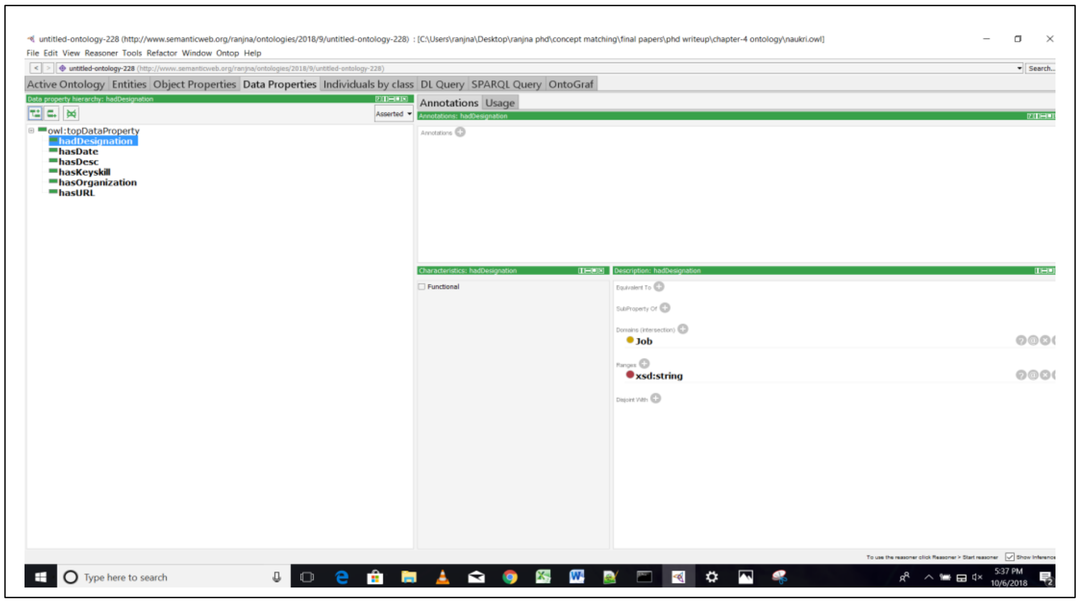


Fig. 5.7 Snapshot of Data Property Hierarchy of Naukri Ontology

Fig. 5.8 shows the OntoGraph Visualizer of *Naukri Ontology*.In this ontology, classes such as role, industry, employment type etc. were developed which give the job information in Naukri.com.

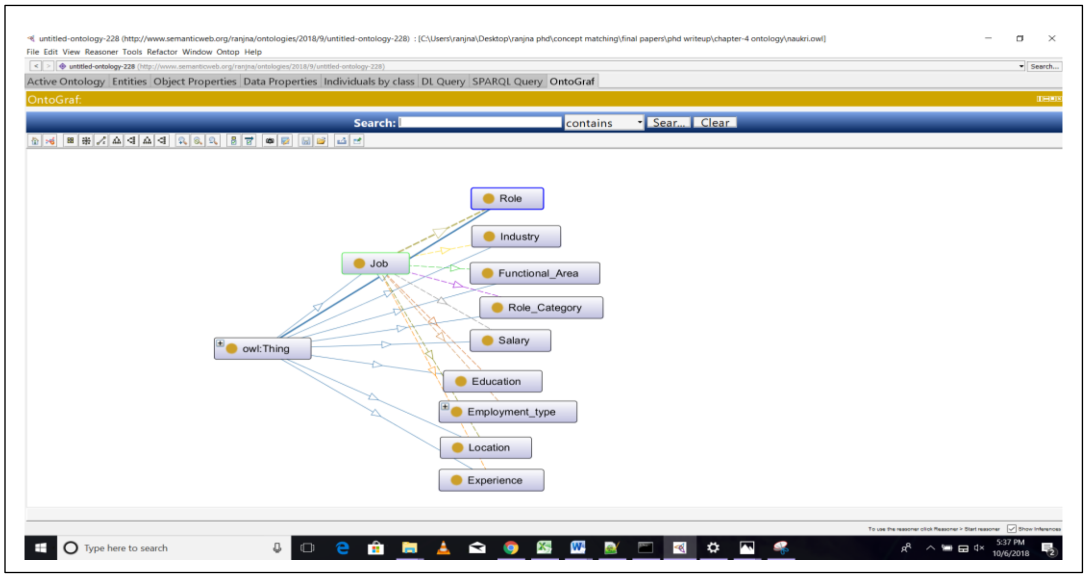


Fig. 5.8 OntoVisualizer Result of Naukri Ontology

c) Shine Ontology

This ontology contains concepts related to job entity presented to user by Shine.com. The steps followed for developing Shine Ontology are as follows:

- **Identifying Concepts**

The different classes for Shine ontology are depicted in Table 5.5.

Table 5.5 Classes for Shine Ontology

S.No.	Class
1	Department
2	Industry
3	Experience
4	Place
5	Job
6	Sal

The snapshot of Shine Ontology Class hierarchy is shown in Fig. 5.9. The classes shown below will be used to annotate the information extracted from Shine Job board to enrich it semantically.

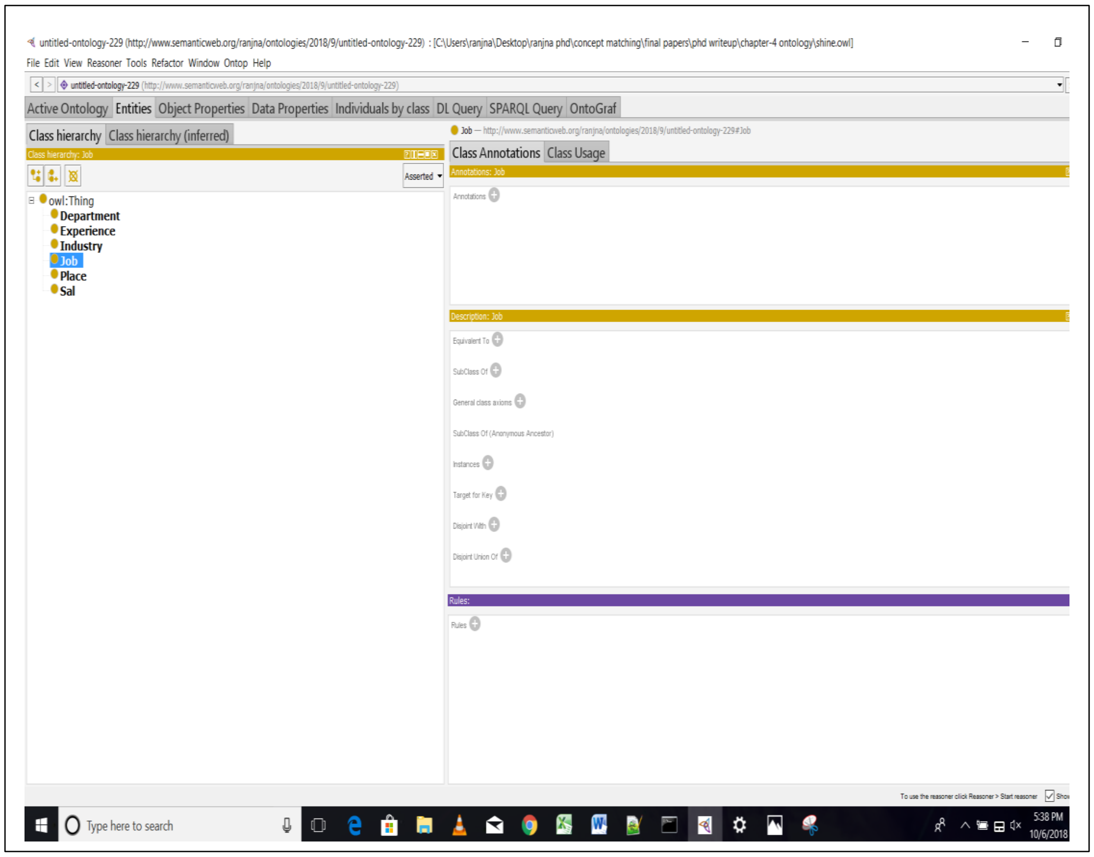

untitled-ontology-229 (http://www.semanticweb.org/ranjna/ontologies/2018/9/untitled-ontology-229) : [C:\Users\ranjna\Desktop\ranjna phd\concept matching\final papers\phd writeup\chapter-4 ontology\shine.owl]
File Edit View Reasoner Tools Refactor Window Ontop Help
Active Ontology Entities Object Properties Data Properties Individuals by class DL Query SPARQL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Class hierarchy: Job
Asserted
owl:Thing
Department
Experience
Industry
Job
Place
Sal
Job — http://www.semanticweb.org/ranjna/ontologies/2018/9/untitled-ontology-229#Job
Class Annotations Class Usage
Annotations: Job
Annotations
Description: Job
Equivalent To
SubClass Of
General class axioms
SubClass Of (Anonymous Ancestor)
Instances
Target for Key
Disjoint With
Disjoint Union Of
Rules:
Rules
To use the reasoner click Reasoner > Start reasoner
Type here to search
5:38 PM
10/6/2018

Fig. 5.9 Snapshot of Class Hierarchy of Shine Ontology

- **Identifying Properties**

In this step, to describe the internal structure of the chosen concepts, properties are identified. The different properties for Shine ontology are depicted in Table 5.6

Table 5.6 Classes for Shine Ontology

Property	**Domain**	**Range**	**Type of Property**
hasDepartment	Job	Department	Object Property
hasmaxExp	Job	Experience	Object Property
hasminExp	Job	Experience	Object Property
hasPlace	Job	Place	Object Property
hasSal	Job	Sal	Object Property
hasIndustry	Job	Industry	Object Property
hasJobtitle	Job	xsd:string	Data Property
hasDateoffpost	Job	xsd:datetime	Data Property
Hasotherskill	Job	xsd:string	Data Property
Hasskill	Job	xsd:string	Data Property
hasVenue	Job	xsd:string	Data Property

The snapshot of Shine Ontology ObjectProperty and Data Property hierarchy is shown in Fig. 5.10 and Fig 5.11.

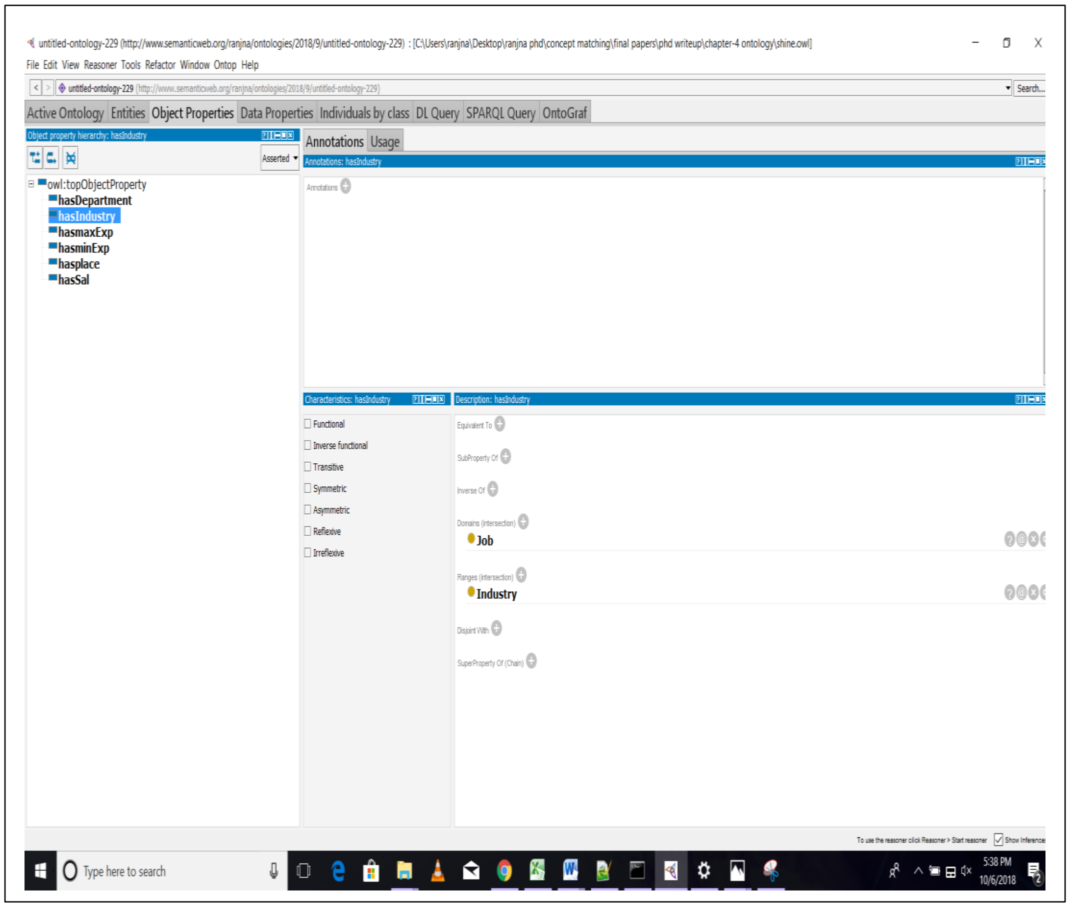

untitled-ontology-229 (http://www.semanticweb.org/ranjna/ontologies/2018/9/untitled-ontology-229) : [C:\Users\ranjna\Desktop\ranjna phd\concept matching\final papers\phd writeup\chapter-4 ontology\shine.owl]
File Edit View Reasoner Tools Refactor Window Ontop Help
Active Ontology Entities Object Properties Data Properties Individuals by class DL Query SPARQL Query OntoGraf
Object property hierarchy: hasIndustry
Asserted
owl:topObjectProperty
hasDepartment
hasIndustry
hasmaxExp
hasminExp
hasplace
hasSal
Annotations Usage
Annotations: hasIndustry
Characteristics: hasIndustry
Functional
Inverse functional
Transitive
Symmetric
Asymmetric
Reflexive
Irreflexive
Description: hasIndustry
Equivalent To
SubProperty Of
Inverse Of
Domains (intersection)
Job
Ranges (intersection)
Industry
Disjoint With
SuperProperty Of (Chain)
To use the reasoner click Reasoner > Start reasoner
Show Inferences
Type here to search
5:38 PM
10/6/2018

Fig. 5.10 Snapshot of Object Property Hierarchy of Shine Ontology

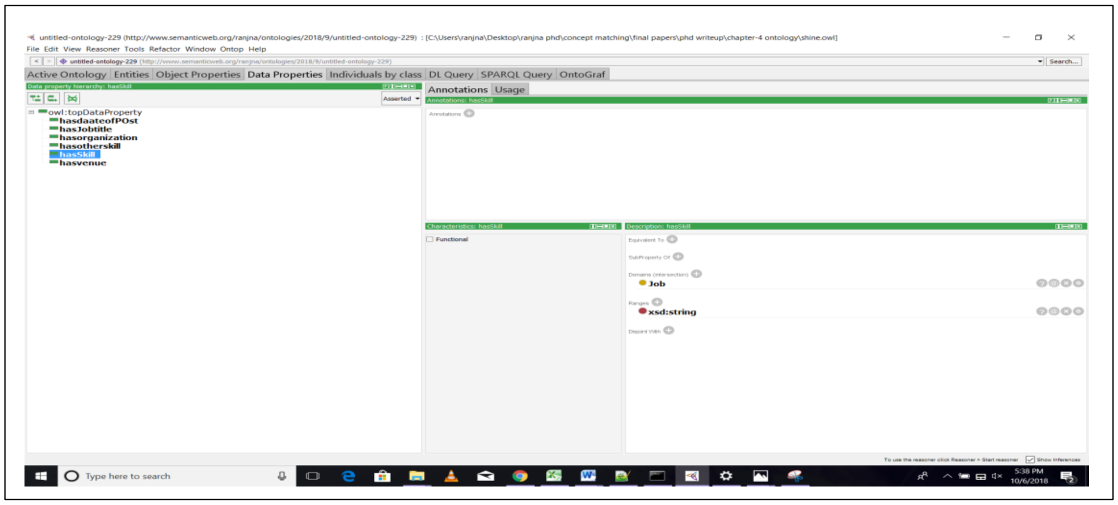


Fig. 5.11 Snapshot of Data Property Hierarchy of Shine Ontology

Fig. 5.12 shows the OntoGraph Visualizer of Shine Ontology. In this ontology, classes such as place, department, Sal etc. were developed which gives the job information in Shine.com.

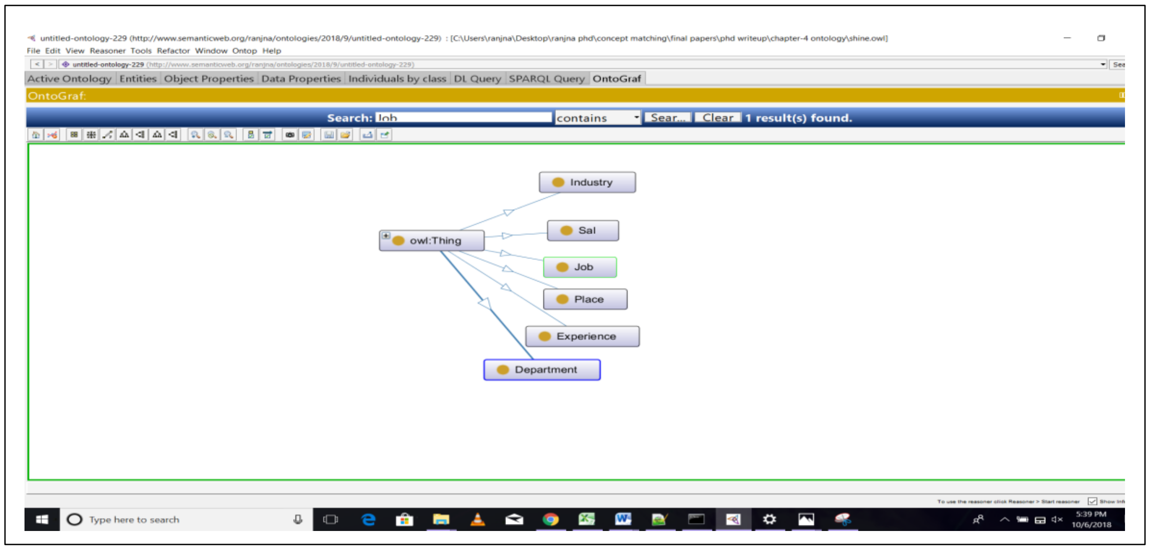


Fig. 5.12 OntoVisualizer Result of Shine Ontology

d) Student Ontology

This ontology represents knowledge related to student profile that is required during searching of suitable job. This ontology covers the educational, personal, career related information and career preferences of student.

- **Determine class hierarchy**

At the basic level, there are 14 main classes which are explained in Table 5.7

Table 5.7 Student Ontology Super Class Description

Class	Description
Student	This class contains the instances of student whose profile is maintaining using this ontology.
Skills	This class tells the skillset of instance that belongs to student class.
Highest qualification	This class contains the various qualifications as its subclasses such as Post-graduation, graduation, others.
University	This class contains the list of universities to which student can belongs to.
Branch	This class contains the branches in which student has taken his highest qualification.
Institute	This class contains the list of institutes as instances which can be selected by student to convey the information from where he has done his education.
Work experience	This class gives option weather student is fresher or experienced.
Work preferences	This class gives option to student if he wants full time or part time job.
State	This class contains the list of states from where student can choose and tell to which state he belongs to.
Job roles	This class contains a list of job roles that a student is looking for.
Academic Project	This class tells the types of academic projects that student instance has done.
Rating	This class contains the English communication rating to which one student belongs to.
Sublocation	This class contains a list of sub-location in the state where student resides.
Gender	This class tells the gender of instance that belongs to student class.

These main classes further have subclasses which are shown in Fig. 5.13.

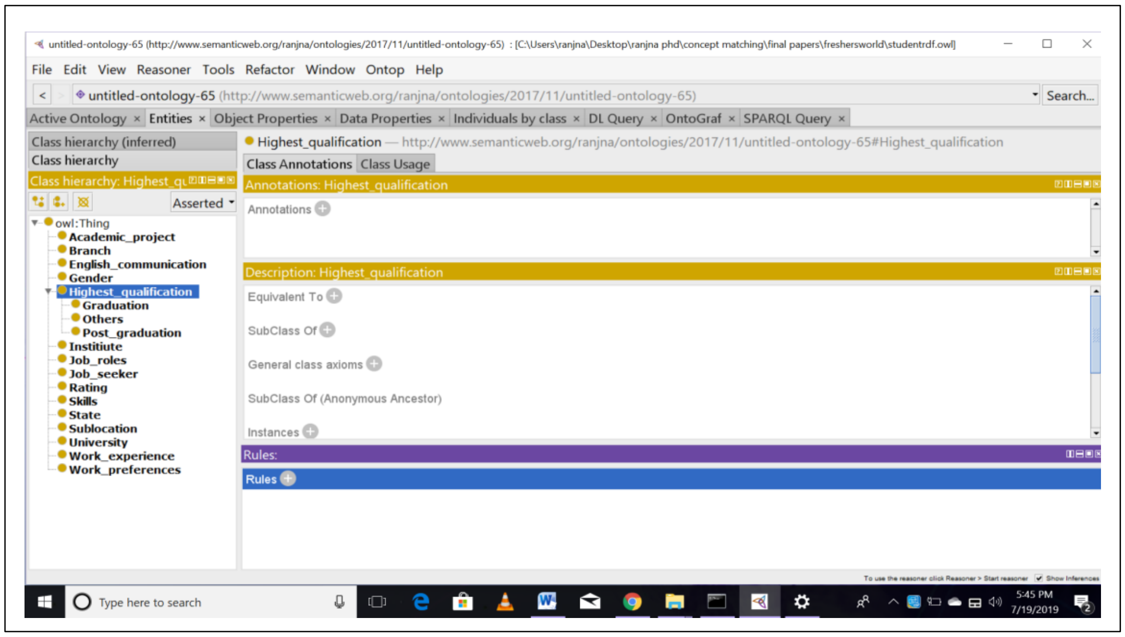

untitled-ontology-65 (http://www.semanticweb.org/ranjna/ontologies/2017/11/untitled-ontology-65) : [C:\Users\ranjna\Desktop\ranjna phd\concept matching\final papers\freshersworld\studentrdf.owl]
File Edit View Reasoner Tools Refactor Window Ontop Help
untitled-ontology-65 (http://www.semanticweb.org/ranjna/ontologies/2017/11/untitled-ontology-65)
Search...
Active Ontology
Entities
Object Properties
Data Properties
Individuals by class
DL Query
OntoGraf
SPARQL Query
Class hierarchy (inferred)
Class hierarchy
Class hierarchy: Highest_qu
Asserted
owl:Thing
Academic_project
Branch
English_communication
Gender
Highest_qualification
Graduation
Others
Post_graduation
Institute
Job_roles
Job_seeker
Rating
Skills
State
Sublocation
University
Work_experience
Work_preferences
Highest_qualification — http://www.semanticweb.org/ranjna/ontologies/2017/11/untitled-ontology-65#Highest_qualification
Class Annotations
Class Usage
Annotations: Highest_qualification
Annotations
Description: Highest_qualification
Equivalent To
SubClass Of
General class axioms
SubClass Of (Anonymous Ancestor)
Instances
Rules:
Rules
Type here to search
5:45 PM
7/19/2019

Fig. 5.13 Class Hierarchy of Student Ontology Using Protégé 5.2

Once the classes have been identified and created, next step is to establish a relation between these classes in order to infer new data. This is performed in next step.

- **Define the properties of the class**

In this ontology, in total 32 data and object properties are defined including sub-properties. Creating sub-properties has enhanced the flexibility. Each property is assigned with domain and range. Properties link individual from the domain to individuals from the range. For example, in our ontology, the property "belongstobranch" would probably link individuals of class student to individuals belonging to the class Branch. A sample of object property which is used in this ontology is shown in Table 5.8 and rest of the properties is shown in Fig. 5.14.

Table 5.8 Sample Object Properties of Student Ontology

Property	**Domain**	**Range**
belongstobranch	Student	Branch
hasjobpreferences	Student	Job preferences

where "belongstobranch" is the object property having student as domain and branch as range which indicates instance of student say ram belongs to branch say 'cse' where 'cse' is an instance of branch class. With this property relation between two instances belonging to different classes can be determined.

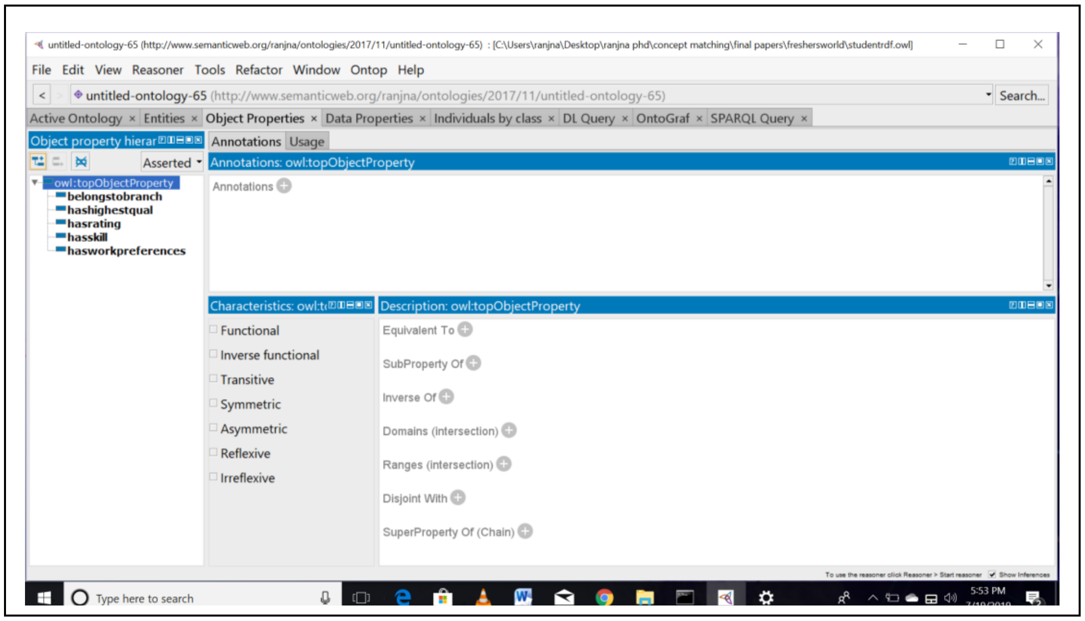

untitled-ontology-65 (http://www.semanticweb.org/ranjna/ontologies/2017/11/untitled-ontology-65) : [C:\Users\ranjna\Desktop\ranjna phd\concept matching\final papers\freshersworld\studentrdf.owl]
File Edit View Reasoner Tools Refactor Window Ontop Help
untitled-ontology-65 (http://www.semanticweb.org/ranjna/ontologies/2017/11/untitled-ontology-65)
Search...
Active Ontology Entities Object Properties Data Properties Individuals by class DL Query OntoGraf SPARQL Query
Object property hierar
Asserted
owl:topObjectProperty
belongstobranch
hashighestqual
hasrating
hasskill
hasworkpreferences
Annotations Usage
Annotations: owl:topObjectProperty
Annotations
Characteristics: owl:t
Functional
Inverse functional
Transitive
Symmetric
Asymmetric
Reflexive
Irreflexive
Description: owl:topObjectProperty
Equivalent To
SubProperty Of
Inverse Of
Domains (intersection)
Ranges (intersection)
Disjoint With
SuperProperty Of (Chain)
Type here to search
5:53 PM

Fig. 5.14 Object Property of Student Ontology using Protégé 5.2

Data Properties that is included in this ontology is shown in Fig. 5.15.

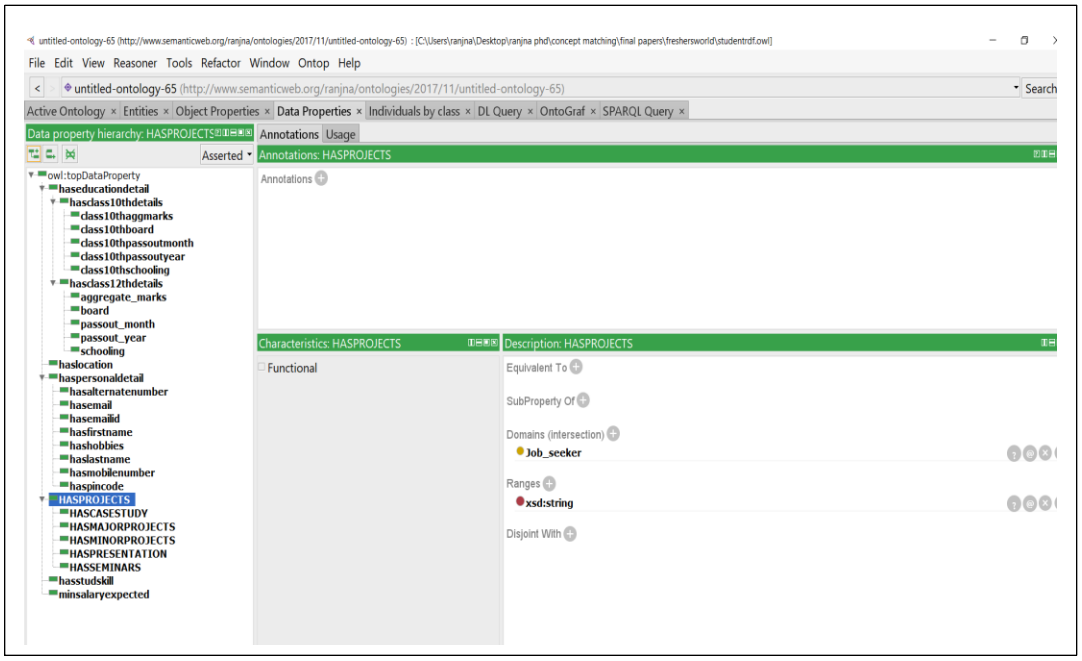


Fig. 5.15 Data Properties of Student Ontology Using Protégé 5.2

5.3 QUERY PROCESSING IN ONTOLOGY

To validate and verify the correctness of the developed ontology, various query languages and rule languages can be used, some of which are given below:

- DL Query
- SPARQL Query
- SWRL Rules
- RDQL

Among all these language, SPARQL Query Tab provides a powerful and easy to use feature for searching the classified ontology. It is a standard Protégé plug-in, available both as a tab and viewed as a view widget that can be positioned into any other tab.

To validate and verify the ontology regarding different competency questions, SPARQL was used in the proposed work. The set of queries and SPARQL Query format designed for the ontology developed in the domain of job has been indicated in

Table 5.9.

Table 5.9 Set of Queries to Be Executed in The Domain of Job

Query	SPARQL Query

Python, Delhi	"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p: <http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?job ?title ?skill ?location where" {
	"?job p:hasloc ?location."+ "?job p:hasskill ?skill."+
	 "FILTER(?skill="Python")." + "FILTER(?location= "Delhi")."
	}

Python, XML, Delhi	"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p: <http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?job ?title ?skill ?location where" {
	"?job p:hasloc ?location."+ "?job p:hasskill ?skill."+
	 "FILTER(?skill="Python")." + "FILTER(?skill= "XML")."+
	"FILTER(?location="Delhi")."
	}
	union
	{
	"?job p:hasloc ?location."+ "?job p:hasskill ?skill."+
	"FILTER(?skill="Python")." + "FILTER(?location= "Delhi")."
	}
	union
	{
	"?job p:hasloc ?location."+ "?job p:hasskill ?skill."+
	"FILTER(?skill="XML")." + "FILTER(?location= "Delhi")."
	}

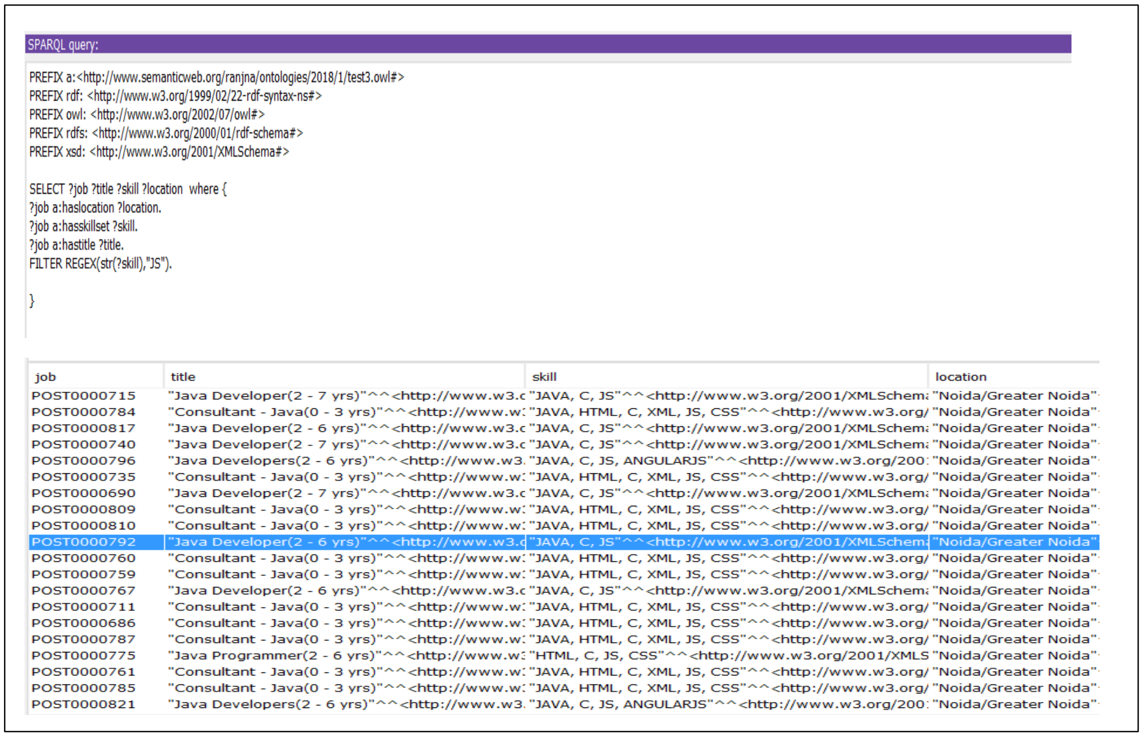
Java, 8yrs	"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p: <http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?job ?title ?skill ?location ?exp where" {
	"?job p:hasloc ?location."+ "?job p:hasexperience ?exp."+
	 "FILTER(?skill="Java")." + "FILTER(?minexp= "8 yrs")."
	}

PHP, 5yrs	"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p: <http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?job ?title ?skill ?location ?exp where" {
	"?job p:hasloc ?location."+ "?job p:hasexperience ?exp."+
	 "FILTER(?skill="PHP")." + "FILTER(?minexp= "5 yrs")."
	}

The queries mentioned in

Table 5.9 have been executed on defined ontologies in the Job domain. One of the results of SPARQL query corresponding to the query 'JS' executed on the developed ontology is depicted in Fig. 5.16.

For the query "JS" as skill depicts the list of job post which requires JS as skillset in the result. The correct result of SPARQL query indicates that the ontology has been well designed and returns the results for individuals and classes.


SPARQL query:

```
PREFIX a:<http://www.semanticweb.org/ranjna/ontologies/2018/1/test3.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?job ?title ?skill ?location  where {
?job a:haslocation ?location.
?job a:hasskillset ?skill.
?job a:hastitle ?title.
FILTER REGEX(str(?skill),"JS").

}
```

job	title	skill	location
POST0000715	"Java Developer(2 - 7 yrs)"^^<http://www.w3.c	"JAVA, C, JS"^^<http://www.w3.org/2001/XMLSchem:	"Noida/Greater Noida"
POST0000784	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000817	"Java Developer(2 - 6 yrs)"^^<http://www.w3.c	"JAVA, C, JS"^^<http://www.w3.org/2001/XMLSchem:	"Noida/Greater Noida"
POST0000740	"Java Developer(2 - 7 yrs)"^^<http://www.w3.c	"JAVA, C, JS"^^<http://www.w3.org/2001/XMLSchem:	"Noida/Greater Noida"
POST0000796	"Java Developers(2 - 6 yrs)"^^<http://www.w3.	"JAVA, C, JS, ANGULARJS"^^<http://www.w3.org/200:	"Noida/Greater Noida"
POST0000735	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000690	"Java Developer(2 - 7 yrs)"^^<http://www.w3.c	"JAVA, C, JS"^^<http://www.w3.org/2001/XMLSchem:	"Noida/Greater Noida"
POST0000809	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000810	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000792	"Java Developer(2 - 6 yrs)"^^<http://www.w3.c	"JAVA, C, JS"^^<http://www.w3.org/2001/XMLSchem:	"Noida/Greater Noida"
POST0000760	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000759	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000767	"Java Developer(2 - 6 yrs)"^^<http://www.w3.c	"JAVA, C, JS"^^<http://www.w3.org/2001/XMLSchem:	"Noida/Greater Noida"
POST0000711	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000686	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000787	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000775	"Java Programmer(2 - 6 yrs)"^^<http://www.w:	"HTML, C, JS, CSS"^^<http://www.w3.org/2001/XMLS	"Noida/Greater Noida"
POST0000761	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000785	"Consultant - Java(0 - 3 yrs)"^^<http://www.w:	"JAVA, HTML, C, XML, JS, CSS"^^<http://www.w3.org/	"Noida/Greater Noida"
POST0000821	"Java Developers(2 - 6 yrs)"^^<http://www.w3.	"JAVA, C, JS, ANGULARJS"^^<http://www.w3.org/200:	"Noida/Greater Noida"

Fig. 5.16 Execution of Query for Timesjob Ontology

5.4 SUMMARY

Building application specific ontologies is not a simple task as it requires a lot of effort and time to invest in domain conceptualization. Three ontologies with respect to each Jobboard site that are selected for implementation were designed in the domain of Jobboard sites. A set of queries were generated and implemented with the help of SPARQL query language. In the next chapter, it has been discussed that how the developed ontologies are used for annotating respective Jobboards sites data. The chapter discusses about proposed OntoJobextractor framework.

Chapter VI

ONTOJOBEXTRACTOR: RELEVANT INFORMATION EXTRACTION FROM JOB BOARDS

6.1 GENERAL

There are multitude of different job searching techniques and platforms available for job seekers such as networking events, social media, staffing firms, company career pages, online job boards, professional organization websites etc. Out of all these options, nowadays, online job boards remain one of the best ways to find out about available jobs and submit application for the same. But, the present job market in India is too much crowded with different job portals. Some of the problems that job seekers generally experience are as follows:

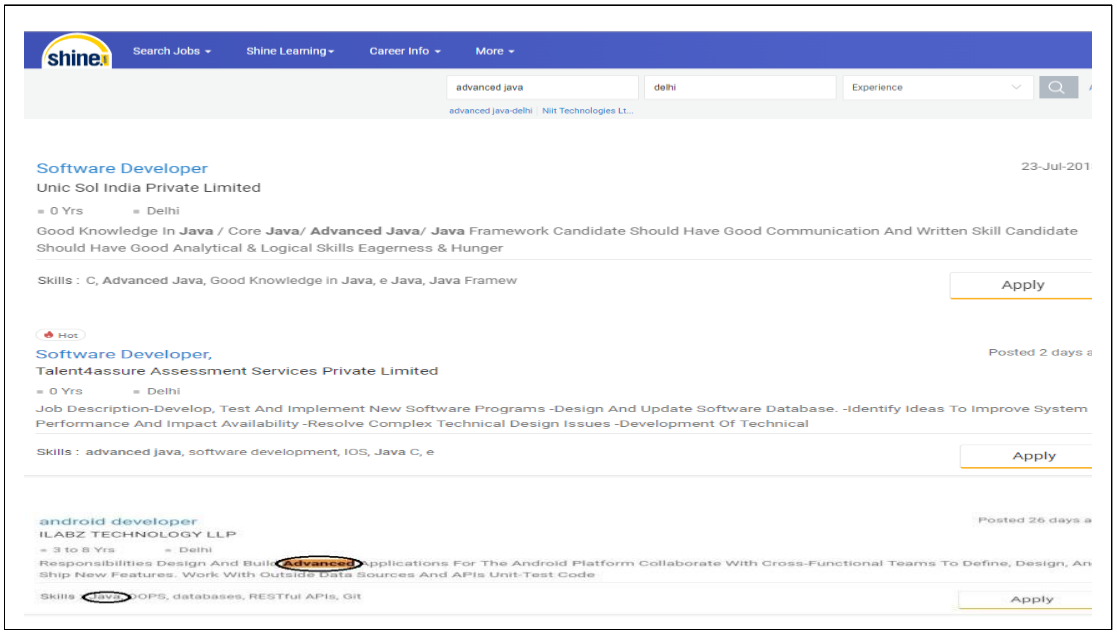
a) **High noise low output**

Almost every portal works on basic keyword matching which makes it easy for everyone to browse and apply for any job posting. But the problem is that this one-click way to "show internet" leads to many irrelevant search results that are further filtered by the job seeker manually.

For instance, upon firing the query *Advanced Java* as skill and *Delhi* as location on the www.Shine .com, it displayed 45 results. A snippet these of search results are shown in Fig. 6.1. Since this job board search system is keyword based, it displayed all the job post having either A*dvanced* or *Java* or A*dvanced Java* in its text part without knowing the fact whether it is a skill or some other attribute of the job post. After analyzing the results, it was identified that out of 45 search results, only 12 job posts were identified as relevant.

b) **Individual registration problem**

In order to get maximum job opportunities, job seeker makes individual profile on different career portals and job boards which ultimately makes job seeking process cumbersome for the user.



Software Developer
Unic Sol India Private Limited
0 Yrs Delhi
23-Jul-201
Good Knowledge In **Java** / Core **Java**/ **Advanced Java**/ **Java** Framework Candidate Should Have Good Communication And Written Skill Candidate Should Have Good Analytical & Logical Skills Eagerness & Hunger
Skills : C, **Advanced Java**, Good Knowledge in **Java**, e **Java**, **Java** Framew

Software Developer,
Talent4assure Assessment Services Private Limited
0 Yrs Delhi
Posted 2 days a
Job Description-Develop, Test And Implement New Software Programs -Design And Update Software Database. -Identify Ideas To Improve System Performance And Impact Availability -Resolve Complex Technical Design Issues -Development Of Technical
Skills : **advanced java**, software development, IOS, **Java** C, e

android developer
ILABZ TECHNOLOGY LLP
3 to 8 Yrs Delhi
Posted 26 days a
Responsibilities Design And Build **Advanced** Applications For The Android Platform Collaborate With Cross-Functional Teams To Define, Design, An
Ship New Features. Work With Outside Data Sources And APIs Unit-Test Code
Skills **Java**, OOPS, databases, RESTful APIs, Git

Fig. 6.1 Snapshot of Search Results From www.Shine.Com

c) Spam problem

Spamming is one of the often-cited problems with mainstream job boards like Naukri.com. When a job seeker creates a profile, it becomes a part of job board database, which is not only accessed by recruiters but also portal's partners.

In addition to job related information, job seeker starts getting many other mails viz staffing agency writes to pay money for training and assessment, offers to join remote learning program, convincing to join MBA from institutes which ultimately harasses the job seeker.

Looking at the above issues, an "Ontojobextractor" system is proposed which extracts only job post related relevant information from the job board pages and enriches them semantically using ontology. This process ensures that the job post repository contains quality Job posts.

6.2 PROPOSED APPROACH FOR EXTRACTING RELEVANT INFORMATION FROM JOBBOARD

In general, a Job board's web page consists of several job posts comprising of the contents giving details about it. Thus, the technique presented in the current section focuses on extracting those contents with respect to each job post. The proposed *OntoJobextractor* system visits web pages and extracts relevant content from it. This extracted information is in semi-structured/unstructured format which in turn enriched with the information semantically using ontology by the system and converted into structured data. The process of data extraction starts with building URLs which are to be crawled for extracting desired data. The architecture of building ontology from job board website is shown below in Fig. 6.2.

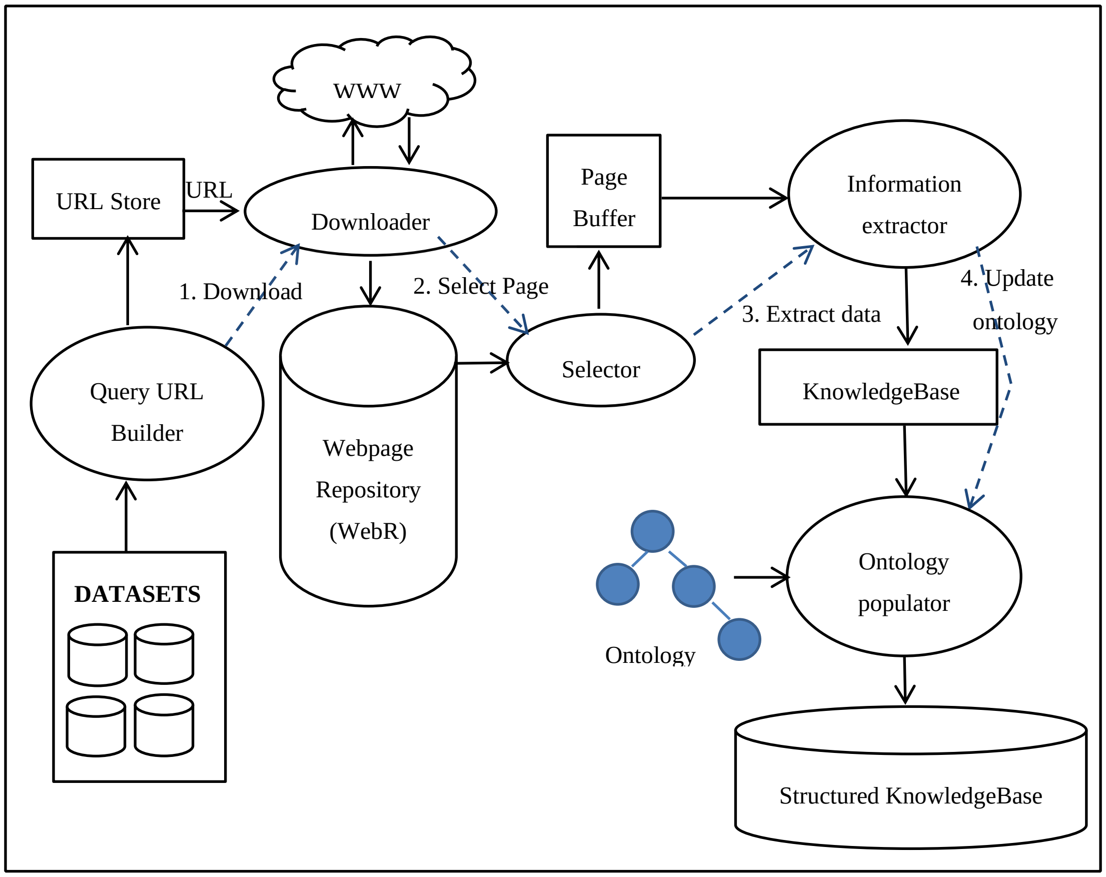


Fig. 6.2 Process of Populating Ontology from Job Board

The system maintains multiple datasets: A Skill dataset (a list of keyskills), Location dataset (a list of locations in terms of cities of India), Experience dataset (a list of experience a job is looking for, in terms of years) and Salary dataset (a list of salary which a job is offering in terms of salary per annum). Using these datasets, URLs are built. These URLS are then added in the URL Queue from where downloader picks the URL and downloads the respective webpages. The downloaded pages are temporarily stored in **Web**page **R**epository (WebR). Upon getting a signal from *downloader*; selector selects the webpage from WebR and store the webpage into page buffer. *Information Extractor* fetches webpage from the buffer and starts extracting data from that webpage once it gets signals from *selector* and stores the selected semi-structured extracted information in the knowledgebase. The O*ntology Populator* module takes this knowledgebase as an input and transforms semi-structured information into structured format by annotating data using ontology. The *Information Extractor* performs the same operation for other webpages also and thus new extracted data keeps on updating in the ontology using ontology updater process.

There are five main processes of the proposed system shown in Figure:

- QueryURL Builder
- Downloader
- Selector
- Information Extractor
- Ontology Populator

The detailed explanation of these modules is given in the following subsections:

6.2.1 Query URL Builder

Generally, Jobboard sites apart from job posts provide many other services such as question paper sets for job preparation, preparing resume for interview etc. The main motive behind this module is to create URLs of those webpages only whose information is to be extracted. It takes job board URL format, keywords from respective datasets as an input and generates URLs which are then added to the URL Queue for further processing. The format of URLs is specific to different Jobboard sites. The general steps for building the URLs are discussed in Fig. 6.3.

The process starts with fetching keywords from *skill dataset,* location dataset, salary dataset, experience dataset and a list of Jobboard sites whose webpages are to be extracted from the dataset. After fetching, *Fetcher* keeps the keyword related data in the Keyword buffer and job board list in the Jobboard Buffer. It then sends the signal to *Keyword Combination Generator*.

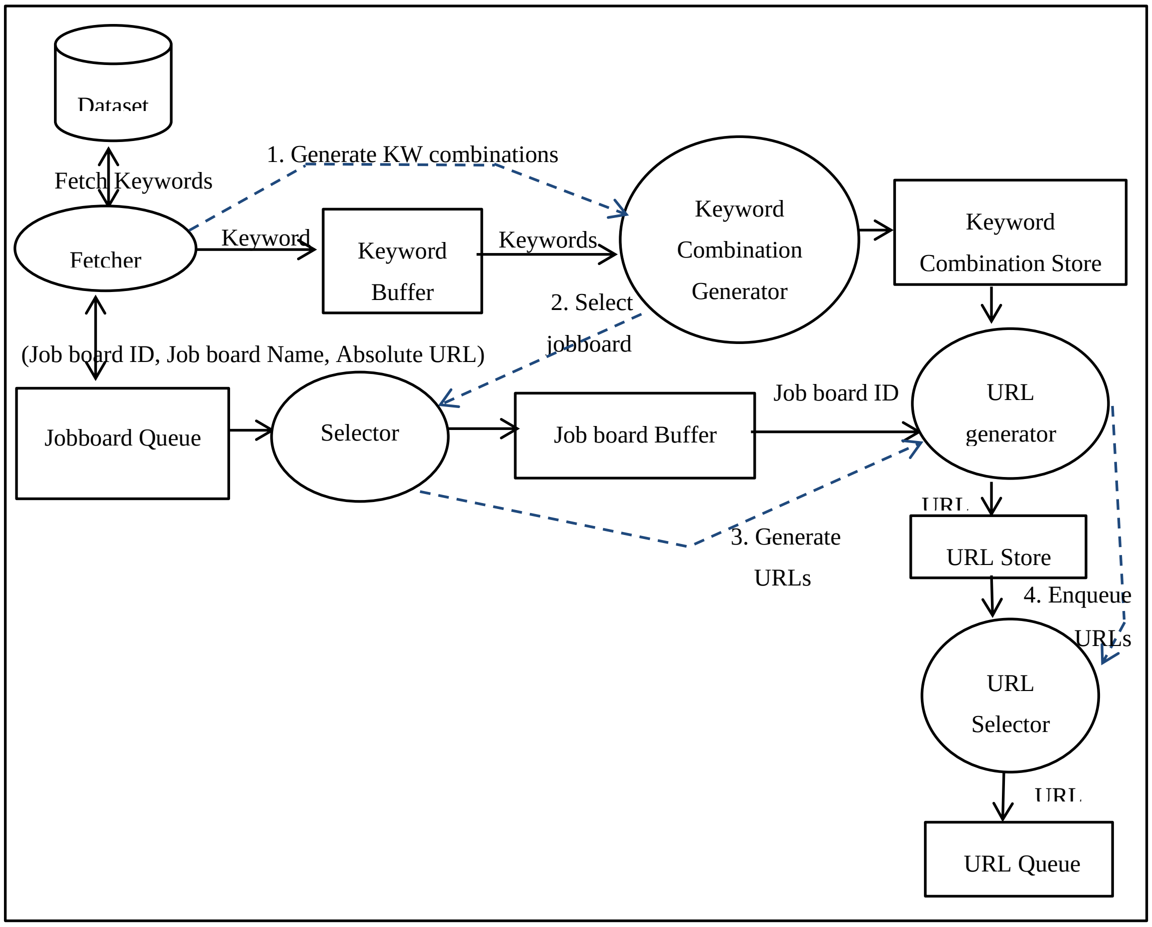


Fig. 6.3 Process of Building Urls

This generator takes keywords from the keyword buffer as inputs and generates all the possible combinations of keywords which are stored to keyword combination store. Once, keyword combinations have been generated, it sends signal to select the job board whose URLs are to be generated. *Selector* selects the job board from job board queue, adds it to the Jobboard Buffer and sends signals to *URL Generator*. *URL Generator* gets Jobboard ID as an input which helps the URL generator to select the algorithm to generate the URLs corresponding to Jobboard ID as every Jobboard has different format of URL representations.

Once the URLs have been generated, it stores them to the URL store and sends signal to URL selector to select the URLs from URL store and enqueue to the URL queue. The algorithm for building URLs is explained in Fig. 6.4.

```
Query URL Builder ()
{
[Input]: Skill dataset SD, Location dataset LD, Jobboard Table JT.
[Output]: URLs
Select the JobboardID from the jobboard queue and add it to jobboard buffer.
Fetch skill and location keywords from SD and LD and store in the keyword buffer.
Signal (generate keyword, KW combinations)
Generate keyword combinations i.e. KC.
Store KC in Keyword Combination Store (KCS).
Signal (select jobboard)
While(Jobboard buffer not empty)
        {
        dequeue jobboard buffer.
        Absolute address=Select jobboard absolute address.
        while(KCS is not empty)
        {
        Relative address=create relative address according to selected jobboard URL format.
        newURL= append(absolute address, relative address)
        insert (URLStore, newURL)
        }
Signal (Enqueue URLs)
Enqueue (URLQueue, URL Store)
}
```

Fig. 6.4 Algorithm of Query URL Builder

The working of Query URL builder is explained with the help of illustration shown in Fig. 6.5

The URL's lists in URL store in Fig. 6.5 are the URLs generated using URL_BUILDER. The relevant URLs generated by applying the proposed approach on skill and location datasets are given in Appendix-1. The output of Query URL Builder generated by proposed system is shown in section 6.3 (Refer Fig. 6.12).

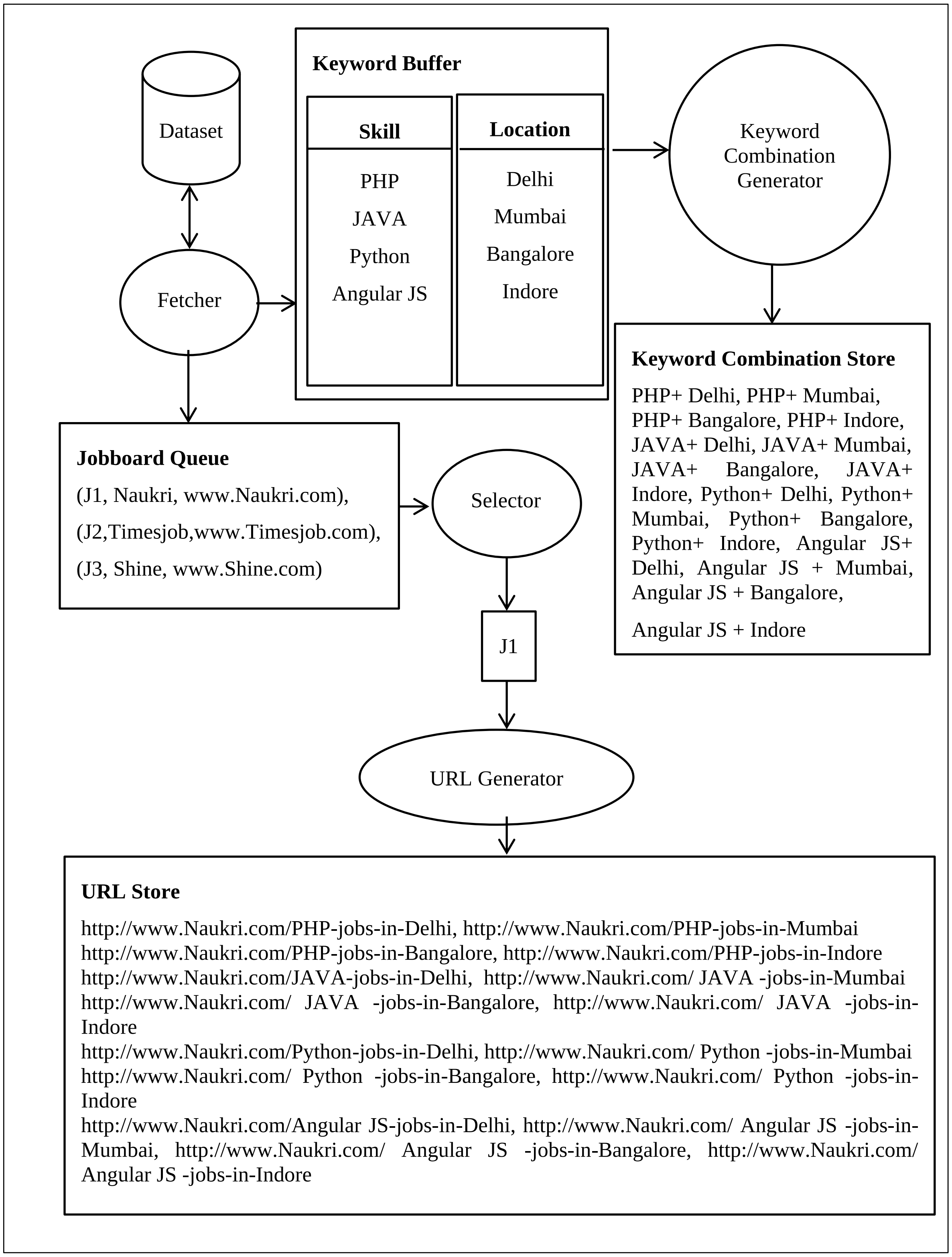

Dataset
Fetcher
Keyword Buffer
Skill
PHP
JAVA
Python
Angular JS
Location
Delhi
Mumbai
Bangalore
Indore
Keyword Combination Generator
Keyword Combination Store
PHP+ Delhi, PHP+ Mumbai, PHP+ Bangalore, PHP+ Indore, JAVA+ Delhi, JAVA+ Mumbai, JAVA+ Bangalore, JAVA+ Indore, Python+ Delhi, Python+ Mumbai, Python+ Bangalore, Python+ Indore, Angular JS+ Delhi, Angular JS + Mumbai, Angular JS + Bangalore,
Angular JS + Indore
Jobboard Queue
(J1, Naukri, www.Naukri.com),
(J2,Timesjob,www.Timesjob.com),
(J3, Shine, www.Shine.com)
Selector
J1
URL Generator
URL Store
http://www.Naukri.com/PHP-jobs-in-Delhi, http://www.Naukri.com/PHP-jobs-in-Mumbai http://www.Naukri.com/PHP-jobs-in-Bangalore, http://www.Naukri.com/PHP-jobs-in-Indore http://www.Naukri.com/JAVA-jobs-in-Delhi, http://www.Naukri.com/ JAVA -jobs-in-Mumbai http://www.Naukri.com/ JAVA -jobs-in-Bangalore, http://www.Naukri.com/ JAVA -jobs-in-Indore
http://www.Naukri.com/Python-jobs-in-Delhi, http://www.Naukri.com/ Python -jobs-in-Mumbai http://www.Naukri.com/ Python -jobs-in-Bangalore, http://www.Naukri.com/ Python -jobs-in-Indore
http://www.Naukri.com/Angular JS-jobs-in-Delhi, http://www.Naukri.com/ Angular JS -jobs-in-Mumbai, http://www.Naukri.com/ Angular JS -jobs-in-Bangalore, http://www.Naukri.com/ Angular JS -jobs-in-Indore

Fig. 6.5 Snippets of URLs Generated from URL_BUILDER

6.2.2 Downloader

The main purpose of downloader process is to download pages from the WWW. It waits for a signal called 'download' from the 'URL Query Builder'. Thereafter, it picks up the URL from the 'URL store' and downloads the pages corresponding to that URL. The algorithm to perform downloading process is explained as below in Fig. 6.6.

```
Downloader()
{
[Input]: URL
[Output]: Webpage w.
wait(download)
Extract URL from the queue.
Fetch corresponding webpage w from the web.
store w in the WebR.
signal(select page).
}
```

Fig. 6.6 Algorithm of Downloader Process

The downloaded webpages are stored in the WebR. Finally, it sends a signal called 'select page' to the selector.

6.2.3 Selector

This process selects the webpage from the WebR. It waits for a signal called 'select page' from downloader. Thereafter, it picks up the webpage from WebR and places it in the page buffer. The algorithm select page process is explained as below in Fig. 6.7

```
Selector ()
{
[Input]: webpage w.
wait(select page)
pick a webpage w from WebR.
put w in the page buffer.
signal(mine page).
}
```

Fig. 6.7 Algorithm of Selector Process

It finally sends a signal called 'mine page' to the Information Extractor.

6.2.4 Information Extractor

This module extracts information from the webpage. It waits for a signal called ‘mine page’ from the selector process. Thereafter, it collects the webpage over which data is to be extracted from the page buffer. The algorithm to perform data extraction process is explained as below in Fig. 6.8.

```
InformationExtractor()
{
[Input]: Webpage w.
[Output]:extracted data.
wait(mine w)
select w from page buffer.
identify the DIV container from the w.
extract data.
parse the data.
store data into Knowledgebase.
}
```

Fig. 6.8 Algorithm of Information Extraction Process

As an output, this process stores all the job posts in knowledgebase. The samples of extracted information related to jobs are shown in Fig. 6.9. These job lists are extracted from *www.Naukri.com*.

```
Web Source : www.Naukri.com
POST NO : 2
Title : Core Java Developer - Associate / Sr. Associate Roles @ Sapient
Org : Sapient Consulting Pvt. Ltd
Skills : core java, spring, hibernate, webservices, multithreading, javascript, java...
Location : Delhi NCR
Experience : 5-10 yrs
Salary :   Not disclosed
Datetime : 5 days ago
Description : -Providing technical expertise for every phase of the project lifecyclefrom concept development to ...
Link : https://www.Naukri.com/job-listings-Core-Java-Developer-Associate-Sr-Associate-Roles-Sapient-Sapient-Consulting-Pvt-Ltd-Delhi-NCR-5-to-10-years-151217005123?src=jobsearchDesk&sid=15172923637258&xp=2&px=1260118000883?src=jobsearchDesk&sid=15172923637258&xp=1&px=1
```

Fig. 6.9 Sample of Extracted Information from Jobboard

The information extracted from *www.Naukri.com*, *www.Timesjob.com* and *www.Shine.com* by applying the proposed approach has been given in Appendix-2.

6.2.5 Ontology Populator

In order to enrich the extracted data semantically, this process annotates data stored in knowledgebase using ontology.

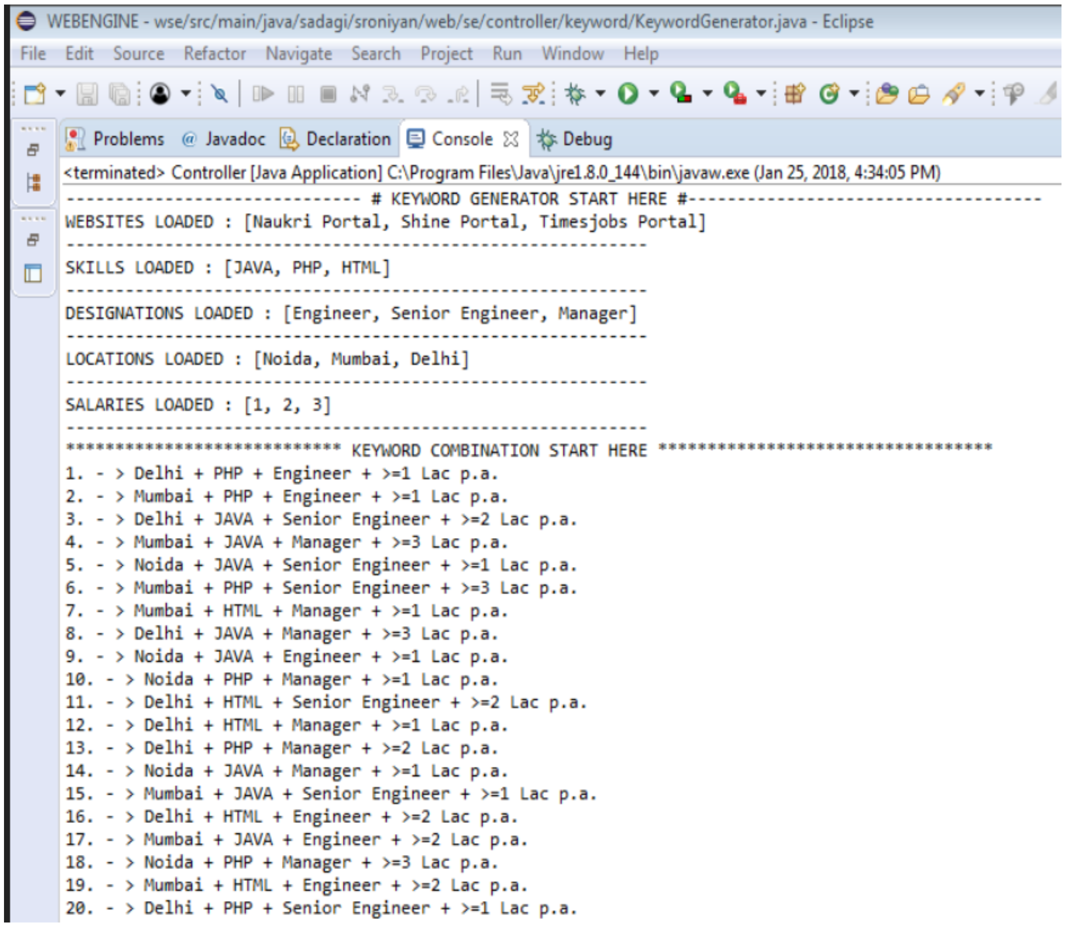
A sample of output generated by Ontology Populator with respect to data stored in knowledgebase is shown in Fig. 6.10.

```
<owl:NamedIndividual
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test3.owl#POST0000002">

<rdf:type rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test1.owl#job"/>

      <hascompany rdf:datatype="http://www.w3.org/2001/XMLSchema#string"> Sapient Consulting
Pvt. Ltd

</hascompany>

      <hasdescription rdf:datatype="http://www.w3.org/2001/XMLSchema#string"> Providing
technical expertise for every phase of the project lifecyclefrom concept development to…
</hasdescription>

<hasdt rdf:datatype="http://www.w3.org/2001/XMLSchema#string">5 days ago</hasdt>

<hasminexperience rdf:resource="5yrs"/>
<hasmaxexperience rdf:resource="10yrs"/>
<hasid rdf:resource=001"/>
<haslocation rdf:resource="Delhi"/>
<haslocation rdf:resource="NCR"/>
<hassalary rdf:resource="Not Disclosed"/>

<hasskillset rdf:datatype="http://www.w3.org/2001/XMLSchema#string"> core java, spring,
hibernate, webservices, multithreading, javascript, java...

</hasskillset>

<hastitle rdf:datatype="http://www.w3.org/2001/XMLSchema#string"> Core Java Developer -
Associate / Sr. Associate Roles @ Sapient </hastitle>

 <hasurl rdf:datatype=": https://www.Naukri.com/job-listings-Core-Java-Developer-Associate-Sr-
            Associate-Roles-Sapient-Sapient-Consulting-Pvt-Ltd-Delhi-NCR-5-to-10-years-
            151217005123?src=jobsearchDesk&sid=15172923637258&xp=2&px=1260118000883?src=
            jobsearchDesk&sid=15172923637258&xp=1&px=1”</hasurl>
   </owl:NamedIndividual>
```

Fig. 6.10 Structured Information Generated by Ontology Populator

6.3 IMPLEMENTATION OF THE PROPOSED SYSTEM

To analyze the proposed work, various experiments have been conducted. The proposed approach has been implemented in Java Eclipse. For the analysis of the proposed system, keywords were generated. The dataset of keywords was maintained in MYSQL. The snapshot of keyword generator module is shown in Fig. 6.11.

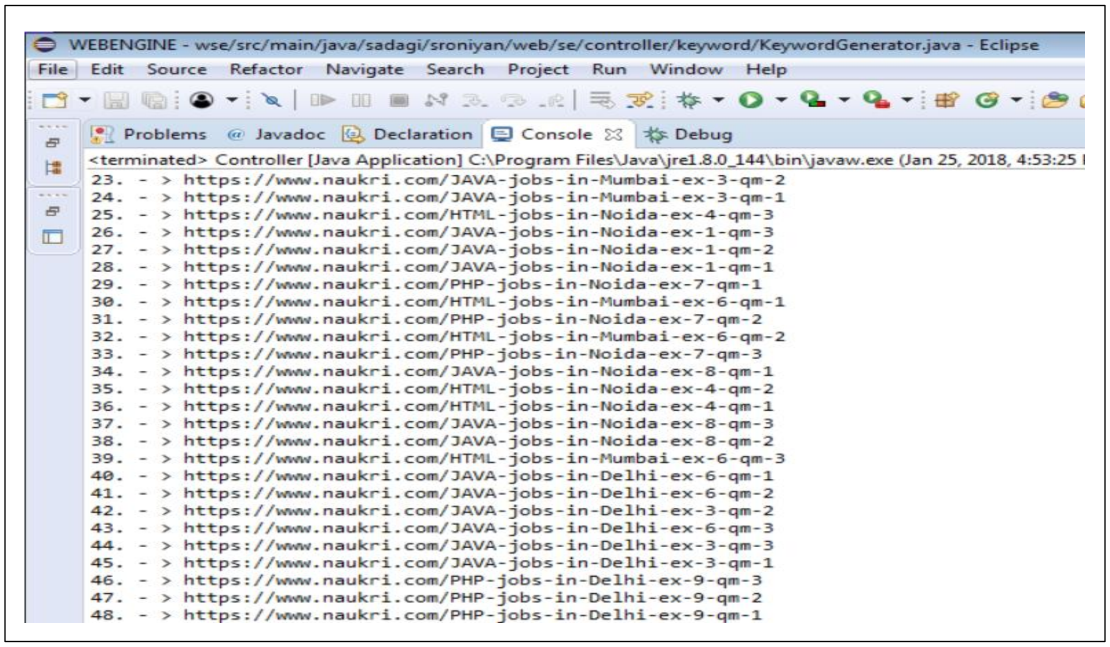

WEBENGINE - wse/src/main/java/sadagi/sroniyan/web/se/controller/keyword/KeywordGenerator.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems @ Javadoc Declaration Console Debug

```
<terminated> Controller [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (Jan 25, 2018, 4:34:05 PM)
----------------------------- # KEYWORD GENERATOR START HERE #-----------------------------------
WEBSITES LOADED : [Naukri Portal, Shine Portal, Timesjobs Portal]
----------------------------------------------------------
SKILLS LOADED : [JAVA, PHP, HTML]
----------------------------------------------------------
DESIGNATIONS LOADED : [Engineer, Senior Engineer, Manager]
----------------------------------------------------------
LOCATIONS LOADED : [Noida, Mumbai, Delhi]
----------------------------------------------------------
SALARIES LOADED : [1, 2, 3]
----------------------------------------------------------
*************************** KEYWORD COMBINATION START HERE *********************************
1. - > Delhi + PHP + Engineer + >=1 Lac p.a.
2. - > Mumbai + PHP + Engineer + >=1 Lac p.a.
3. - > Delhi + JAVA + Senior Engineer + >=2 Lac p.a.
4. - > Mumbai + JAVA + Manager + >=3 Lac p.a.
5. - > Noida + JAVA + Senior Engineer + >=1 Lac p.a.
6. - > Mumbai + PHP + Senior Engineer + >=3 Lac p.a.
7. - > Mumbai + HTML + Manager + >=1 Lac p.a.
8. - > Delhi + JAVA + Manager + >=3 Lac p.a.
9. - > Noida + JAVA + Engineer + >=1 Lac p.a.
10. - > Noida + PHP + Manager + >=1 Lac p.a.
11. - > Delhi + HTML + Senior Engineer + >=2 Lac p.a.
12. - > Delhi + HTML + Manager + >=1 Lac p.a.
13. - > Delhi + PHP + Manager + >=2 Lac p.a.
14. - > Noida + JAVA + Manager + >=1 Lac p.a.
15. - > Mumbai + JAVA + Senior Engineer + >=1 Lac p.a.
16. - > Delhi + HTML + Engineer + >=2 Lac p.a.
17. - > Mumbai + JAVA + Engineer + >=2 Lac p.a.
18. - > Noida + PHP + Manager + >=3 Lac p.a.
19. - > Mumbai + HTML + Engineer + >=2 Lac p.a.
20. - > Delhi + PHP + Senior Engineer + >=1 Lac p.a.
```

Fig. 6.11 Keyword Combination as An Output from Keyword Combination Generator

This module took input from Keyword Buffer which was maintained in SQL server and generated all possible combinations. These keyword combinations were then used by Query URL Builder that generated URLs of the webpages which are to be crawled. Then, URLs of three job boards were built using Query URL Builder module. The snapshot of generated URLs is shown in Fig. 6.12.

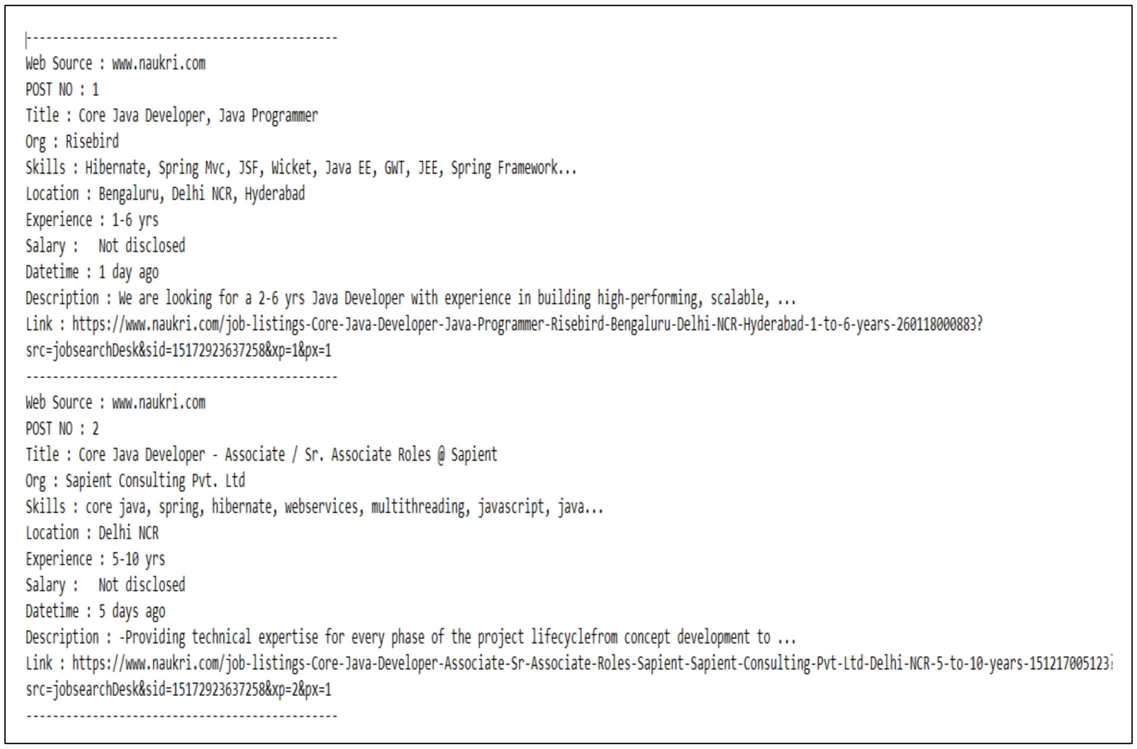
For instance, https://www.Naukri.com/JAVA-jobs-in-Mumbai-ex-3-qm-2 generated by appending absolute URL of Jobboard with keyword combination generated by keyword combination generator.


WEBENGINE - wse/src/main/java/sadagi/sroniyan/web/se/controller/keyword/KeywordGenerator.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Debug

```
<terminated> Controller [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (Jan 25, 2018, 4:53:25 
23. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-3-qm-2
24. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-3-qm-1
25. - > https://www.naukri.com/HTML-jobs-in-Noida-ex-4-qm-3
26. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-1-qm-3
27. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-1-qm-2
28. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-1-qm-1
29. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-7-qm-1
30. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-6-qm-1
31. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-7-qm-2
32. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-6-qm-2
33. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-7-qm-3
34. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-8-qm-1
35. - > https://www.naukri.com/HTML-jobs-in-Noida-ex-4-qm-2
36. - > https://www.naukri.com/HTML-jobs-in-Noida-ex-4-qm-1
37. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-8-qm-3
38. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-8-qm-2
39. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-6-qm-3
40. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-6-qm-1
41. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-6-qm-2
42. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-3-qm-2
43. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-6-qm-3
44. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-3-qm-3
45. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-3-qm-1
46. - > https://www.naukri.com/PHP-jobs-in-Delhi-ex-9-qm-3
47. - > https://www.naukri.com/PHP-jobs-in-Delhi-ex-9-qm-2
48. - > https://www.naukri.com/PHP-jobs-in-Delhi-ex-9-qm-1
```

Fig. 6.12 Generated URLs as an Output from Query URL Builder

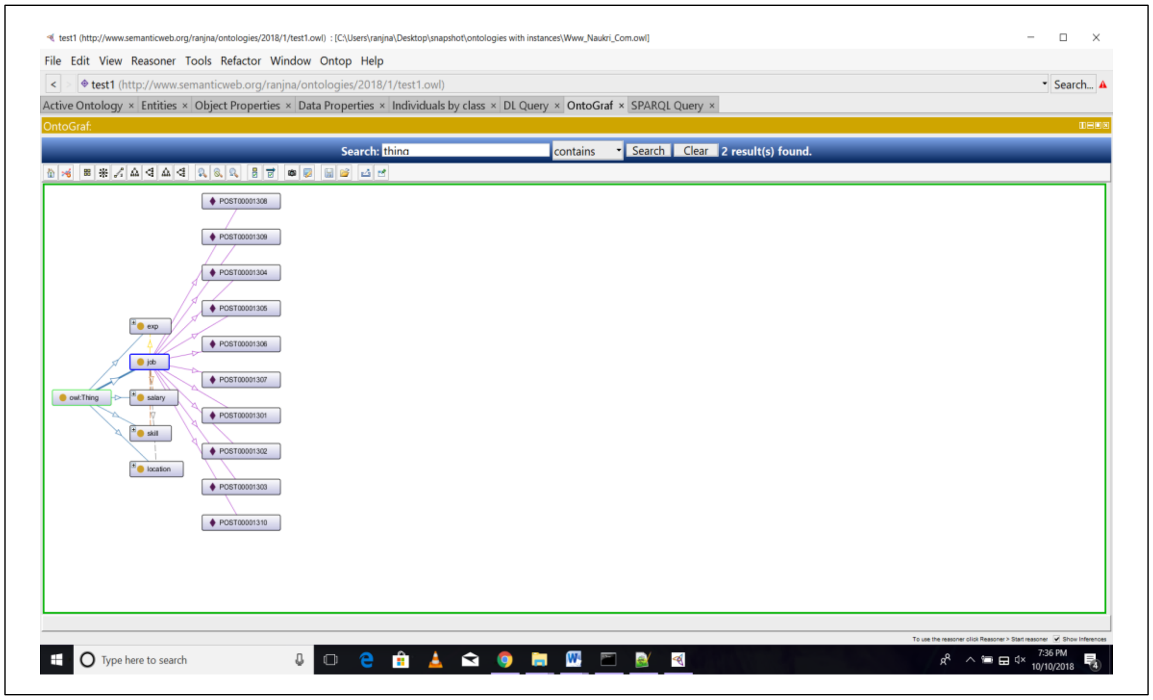
After this, relevant content from these webpages was extracted using Information Extractor module. The snapshot of extracted information is shown in Fig. 6.13.



```
|----------------------------------------------
Web Source : www.naukri.com
POST NO : 1
Title : Core Java Developer, Java Programmer
Org : Risebird
Skills : Hibernate, Spring Mvc, JSF, Wicket, Java EE, GWT, JEE, Spring Framework...
Location : Bengaluru, Delhi NCR, Hyderabad
Experience : 1-6 yrs
Salary :   Not disclosed
Datetime : 1 day ago
Description : We are looking for a 2-6 yrs Java Developer with experience in building high-performing, scalable, ...
Link : https://www.naukri.com/job-listings-Core-Java-Developer-Java-Programmer-Risebird-Bengaluru-Delhi-NCR-Hyderabad-1-to-6-years-260118000883?
src=jobsearchDesk&sid=15172923637258&xp=1&px=1
----------------------------------------------
Web Source : www.naukri.com
POST NO : 2
Title : Core Java Developer - Associate / Sr. Associate Roles @ Sapient
Org : Sapient Consulting Pvt. Ltd
Skills : core java, spring, hibernate, webservices, multithreading, javascript, java...
Location : Delhi NCR
Experience : 5-10 yrs
Salary :   Not disclosed
Datetime : 5 days ago
Description : -Providing technical expertise for every phase of the project lifecyclefrom concept development to ...
Link : https://www.naukri.com/job-listings-Core-Java-Developer-Associate-Sr-Associate-Roles-Sapient-Sapient-Consulting-Pvt-Ltd-Delhi-NCR-5-to-10-years-151217005123
src=jobsearchDesk&sid=15172923637258&xp=2&px=1
----------------------------------------------
```

Fig. 6.13 Snapshot of Extracted Data from The Jobboard

And at last; this unstructured extracted information was annotated using job board specific ontologies. The sample of extracted data annotated with ontology is given in Appendix-3. The snapshot of structured information is shown in Fig 6.14.



test1 (http://www.semanticweb.org/ranjna/ontologies/2018/1/test1.owl) : [C:\Users\ranjna\Desktop\snapshot\ontologies with instances\Www_Naukri_Com.owl]
File Edit View Reasoner Tools Refactor Window Ontop Help
test1 (http://www.semanticweb.org/ranjna/ontologies/2018/1/test1.owl)
Search...
Active Ontology Entities Object Properties Data Properties Individuals by class DL Query OntoGraf SPARQL Query
OntoGraf:
Search: thing
contains
Search
Clear
2 result(s) found.
owl:Thing
exp
job
salary
skill
location
POST00001308
POST00001309
POST00001304
POST00001305
POST00001306
POST00001307
POST00001301
POST00001302
POST00001303
POST00001310
Type here to search
7:36 PM
10/10/2018

Fig. 6.14 Ontovisualizer Results of Naukri Ontology with Instances

From above Fig. 6.14 it can be observed that the extracted data is annotated using Naukri ontology that was shown in Fig. 5.9 (Chapter V).

6.4 SUMMARY

In this chapter, the complete processing of information extraction from the Jobboard sites is presented. The extracted data is semi-structured in nature which is then enriched semantically. By using Jobboard specific ontology, the system is implemented using the most cutting edge technologies such as Ontology and SPARQL. In order to provide fresh data to the user, system recrawls pages periodically. An Application of this type becomes valuable when used across a large number of Jobboards to extract relevant information and annotate using ontology. In the next chapter, these ontologies will be aligned to bring all the relevant data at one place for user. Using this, with the help of single query, user will be able to get desired results from different Jobboard websites at the same platform.

Chapter VII

BUILDING GLOBAL INDEXES FOR ONTOLOGY ALIGNMENT

7.1 GENERAL

In the previous chapter, OntoJobExtractor module was explained for extracting relevant information from Jobboards and transforming it into structured format using ontology with semantically enriched content for better query processing. Moving ahead, next step is to align these ontologies and providing a platform to user where he can get results from various data sources. This requires ontology alignment between N number (where N>=2) of different ontologies of same domain.

In this work, the ontologies are aligned based on syntactic and semantic criterion as discussed, in the following section. A Global Concept Index, Global Data Property Index and Global Object Property Index is created during the process that will play a vital role in query processing.

7.2 PROPOSED SYSTEM FOR BUILDING GLOBAL INDEXES FOR ONTOLOGY ALIGNMENT

Ontology Alignment is the process of determining correspondence between concepts in ontologies. It is a promising solution to the semantic heterogeneity problem. It finds correspondence between semantically related entities of the ontologies. These correspondences can be used for various tasks such as ontology merging, query answering etc. The aim of the proposed work is to design a novel architecture for ontology alignment as it can be seen in Fig. 7.1. On the knowledge base side, ontology layer is there, where N number of source ontologies of the same domain retrieved from ontology database which are to be aligned (Here N>=2). A Global Concept Index (GCI), Global Object Property Index (GOBJPI) and Global Data Property Index (GDPI) are developed as an output which store the information related to alignment of the source ontologies. The source ontologies remain intact during the process i.e. no changes are made in source ontology.

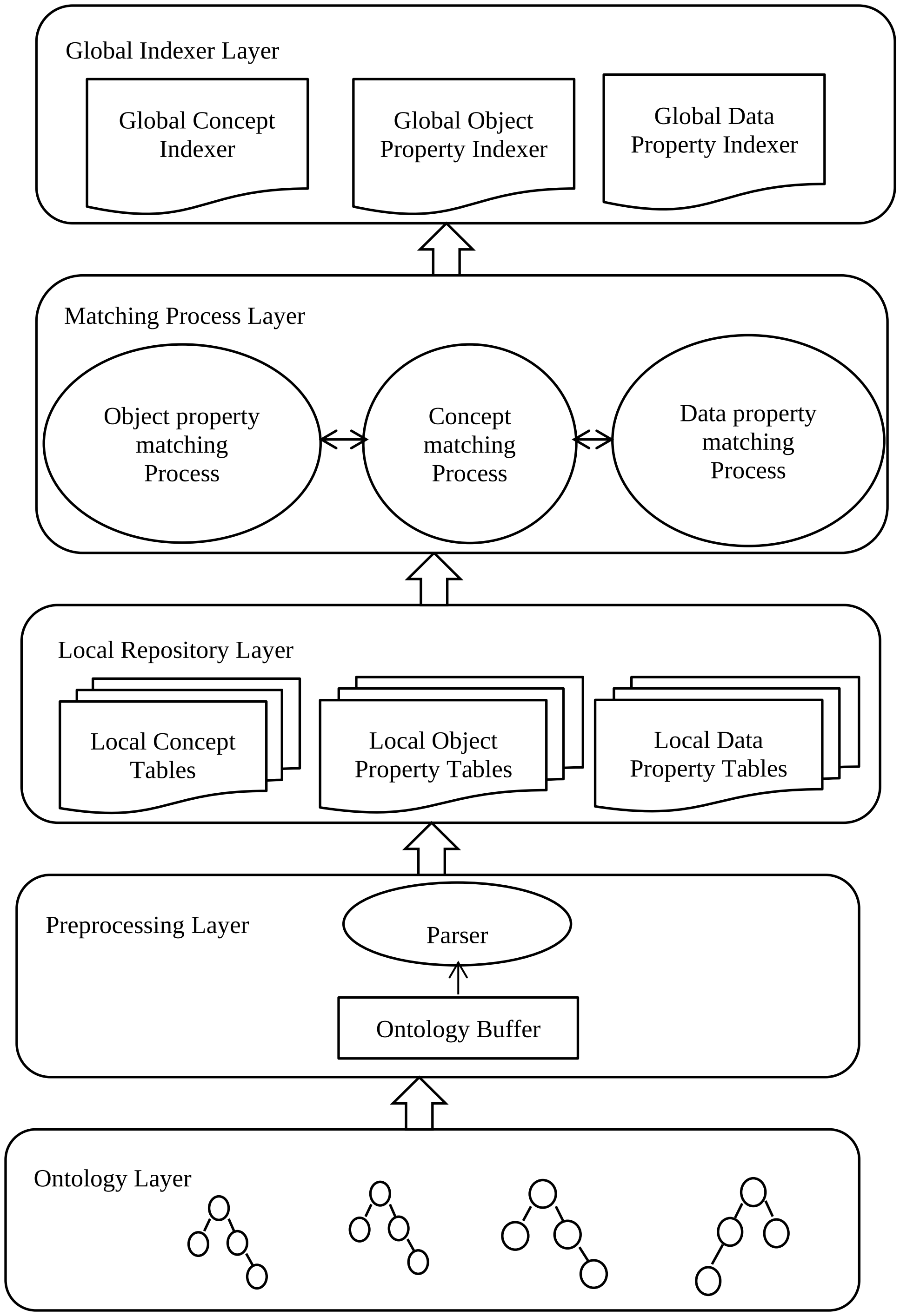


Fig. 7.1 Architecture of Building Global Indexes or Alignment

The proposed method uses matching algorithms to perform alignments. It maintains data structures necessary to keep track of concepts and their properties in the source ontologies. It automatically generates a list of unique concepts without user interference by taking reference from WordNet [166, 167] (for the synonyms) and builds Global Concept Index (GCI). For property matching, along with WordNet, it takes suggestion from the user and builds Global Object Property Index (GOBJPI) and Global Data Property Index (GDPI). These indexes hold the record of unique concepts and properties thereby filling gaps between conceptualization that were occurring before aligning the separate ontologies.

The proposed system consists of five layers: the bottom most layer i.e. *Ontology Layer*, contains a collection of ontologies which are to be aligned. At *Preprocessing Layer*, ontology buffer selects ontology which is to be parsed. Parser generate three tables with respect to each ontology named as Local Concept Table (LCT), Local Object Property Table(LOPT) and Local Data Property Table (LDPT). These tables are stored on *Local Repository Layer*. The next layer toward upward direction is *Matching Process Layer* which contains matching algorithms that will be used during alignment between ontologies. The topmost layer, *Global Index layer* maintains three global indexes named as *Global Concept Index, Global Object Property Index and Global Data Property index*. These indexes maintain aligned information of ontologies.

The detailed explanation of these layers is presented in following subsections:

7.2.1 Ontology Layer

This layer keeps a repository of source ontologies collected from the ontology database belonging to same domain which are to be aligned. The structure of the ontology contains concepts that represent the entities belonging to one domain and the relationships between those concepts which are maintained using properties.

7.2.2 Preprocessing Layer

At this layer, selected ontology from the ontology layer is kept in buffer and its parsing is done by parser. Parser parses the ontology and generates following three tables corresponding to the buffered ontology.

a) Local Concept Table
b) Local Object Property Table
c) Local Data Property Table

The schema of these data structures is shown in Fig. 7.2.

Local Concept Table

OID	LCID	GCID	LCN	URI

Local Object Property Table

OID	LOPID	GOPID	URI	LOPN	DOMAIN		RANGE	
					LCID	GCID	LCID	GCID

Local Data Property Table

OID	LDPID	GDPID	URI	LDPN	DOMAIN		RANGE
					LCID	GCID	

Fig. 7.2 Local Tables for Storing Ontology Specific Information

The three data structures are described in detail as follows.

a) Local Concept Table: Local concept table is a table that stores local information of each concept belonging to its respective ontology. For example, it stores concept name, its global id, its local id etc. The descriptions of various fields maintained in this table are described in Table 7.1. This information helps source ontology concepts to get interlinked with the Global Concept Index.

Table 7.1 Description of Local Concept Table

Field	**Description**
OID	Ontology ID.
LCID	Unique id of concept *c* in the local ontology.
GCID	Unique id of concept *c* belonging to the local ontology in the global concept index.
LCN	Local Concept Name in local ontology.
URI	Uniform Resource Identifier of concept *c* in the local ontology.

b) Local Object Property Table: Local object property table stores the local information of each object property belonging to its respective ontology. for example, it stores information about object property’s local id, its name, local concept id of domain and range etc. The descriptions of various fields are defined in Table 7.2. The information stored in this table helps source ontology object properties to get interlinked with the Global Object Property Index (GOBJPI).

Table 7.2 Description of Local Object Property Table

Field	Description
OID	Ontology id.
LOPID	Unique id of object property *obp* in the local ontology.
LOPN	Local Object Property name in local ontology.
DOMAIN	Domain of the *obp.*
RANGE	Range of the *obp.*
LCID	Unique local concept id of domain and range respectively.
GCID	Unique global concept id of domain and range respectively.
URI	Uniform Resource Identifier of *obp.*
GOPID	Unique id of *obp* belonging to the local ontology in the Global Object Property Index.

c) Local Data Property Table: Local data property table stores the local information of each data property belonging to its respective ontology. for example, it stores information about data property's local id, its name, local concept id of domain and range etc. The descriptions of various fields are defined in Table 7.3. The information stored in this table helps source ontology data properties to get interlinked with the Global Data Property Index (GDPI).

Table 7.3 Description of Local Data Property Table

Field	Description
OID	Ontology ID.
LDPID	Unique id of data property *dp* in the local ontology.
LDPN	Local data property name in local ontology.
DOMAIN	Domain of the *dp.*
RANGE	Datatype.
LCID	Unique local concept id of domain.
GCID	Unique global concept id of domain.
GDPID	Unique id of *dp* belonging to the local ontology in the Global Data Property Index.
URI	Uniform Resource Identifier of d*p.*

The above mentioned three local tables are generated corresponding to all the source ontologies after being processed by parser.

7.2.3 Local Repository Layer

This layer maintains a repository of Local Concept Tables (LCT), Local Object Property Tables (LOBJPT) and Local Data Property Tables (LDPT) which are generated by the preprocessing layer and store information of all the ontologies which are to be further aligned.

7.2.4 Matching Process Layer

It has been experienced that different ontologies use different names to represent the same entity. In this layer, by using Concept Matching (CM) and Property Matching (PM) algorithm, those concepts and properties are uniquely identified from different ontologies and thus can be addressed by the unique concepts and properties. The algorithms explained below will be used while developing Global Indexes.

a) Concept Matching algorithm

As explained above, to collect the maximum concepts from the domain, similarity between two concepts of different ontologies using syntactic and semantic matching is done. For syntactic matching, the algorithm uses longest common substring matching and prefix matching techniques. For semantic matching, it uses synonym matching (taken from WordNet) to match the concepts belonging to different ontologies. The following algorithms will be useful while constructing the GCI [169]. The description of longest common substring and prefix matching (syntactic based concept string matching) and semantic based synonym matching is described as follows:

Algorithm 1: Concept String Matching

It takes two strings (say *str1* and *str2*, which are labels of concepts) one given by GCI and other from the LCT of another ontology; and performs substring matching on both strings as explained in Fig. 7.3.

For instance, to find the substring matching between two strings named a keyskill and skillset, a table LC is maintained as shown in Fig. 7.4 that stores the length of longest common substrings. After applying the algorithm, the common substring comes out to be “skill” with length 5 as highlighted in Fig. 7.4 and then the similarity factor between these strings are performed by applying the above algorithm. The similarity measure i.e. sm, came out to be 62%. If sm> threshold value, then these two strings are considered as similar otherwise discarded.

```
Substring_Matching (str1, str2)
{
[Input:]  str1 and str2 are two strings which are labels of concepts (which are to be matched)
          given by GCI.
[Output:] true, if similarity_factor  >= threshold value
           false, otherwise.
LC[m+1][n+1]; // Create a matrix to store lengths of longest common substrings.
m = strlen(X);
n = strlen(Y);
result = 0; // To store length of the longest common substring
for (int i=0; i<=m; i++)          {
for (int j=0; j<=n; j++)          {
if (i == 0 || j == 0)
LC[i][j] = 0;
else if (X[i-1] == Y[j-1])
{
LC[i][j] = LC[i-1][j-1] + 1;
result = max(result, LC[i][j]);
}
else LC[i][j] = 0;
} }
Sm=result/max(m,n)
     if (sm >  th) // th is a threshold value
     return true
     else
     return false
     }
```

Fig. 7.3 Substring matching Algorithm

INPUT

str1/ str2		k	e	y	s	k	i	l	l
	0	0	0	0	0	0	0	0	0
s	0	0	0	0	1	0	0	0	0
k	0	1	0	0	0	2	0	0	0
i	0	0	0	0	0	0	3	0	0
l	0	0	0	0	0	0	0	4	1
l	0	0	0	0	0	0	0	1	5
s	0	0	0	0	1	0	0	0	0
e	0	0	1	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0

OUTPUT

str1/ str2		k	e	y	s	k	i	l	l
	0	0	0	0	0	0	0	0	0
s	0	0	0	0	1	0	0	0	0
k	0	1	0	0	0	2	0	0	0
i	0	0	0	0	0	0	3	0	0
l	0	0	0	0	0	0	0	4	1
l	0	0	0	0	0	0	0	1	5
s	0	0	0	0	1	0	0	0	0
e	0	0	1	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0

Fig. 7.4 Illustration of Finding Common Substring Between Two Strings

Algorithm 2: Prefix Matching

It is also experienced that developers use shorthand of the concepts while developing ontology. For instance, for term "location", developer usually uses "loc" which signifies the same meaning. Therefore, in order to perform matching between a term and its shorthand, prefix matching is performed. The algorithm to match a string with another string written in shorthand form is discussed in Fig. 7.5

```
Prefix_Matching (str1, str2)

[Input:]  str1 and str2 are two strings which are labels of concepts (which are to be matched)
          given by GCI.
[Output:] true, if result.count is greater than threshold value
      false, otherwise.
          result= “”
          str1len = str1.length();
           str2len = str2.length();
                  for ( i=0, j=0; i<=n1-1&&j<=n2-1; i++,j++)
                  {
                                  if (str1[i] != str2[j])
                                  break;
                                  result=result & str1[i];
                  }
          if (result.length > th)
          return true;
          else
          return false;
```

Fig. 7.5 Prefix Matching Algorithm

In this algorithm, result string contains the common outcome substring after comparing the two strings. If the length of result string is greater than specified threshold value then based on prefix matching, two strings are considered as similar otherwise discarded.

Algorithm 3: Synonym Matching

If concepts are syntactically not same, then it may also happen that synonymy occurs between them. For instance, if ontology Oi contains "company" and ontology Oj contains "organization" as a concept then these two concepts are representing the same entity. To deal this scenario, a Concept Synonym Table(CST) is maintained which contains the synonym of concepts. In this case, a list of synonyms corresponding to concept c_j from the *Concept Synonym table* (CST) using cj'GCID is retrieved and checked whether any one of synonym matches with c_i. or not If yes, then c_i and c_j are considered as same otherwise they are taken as different. The algorithm to perform synonym matching is shown as below in Fig. 7.6

```
Synonym_Match(ci,cj)
[Input:]Two concepts ci and cj, cj whose match is to be find out against second concept ci's
synonyms.
[Output:] true, if cj got matched with any synonym of ci.
     false, otherwise.
Step1:[check if any synonym of ci gets matched with cj]
match=false    // match is a Boolean which will store the final result if any match found or not.
                 Initially it is set to false which indicates that match is not found.
        for each c є si with cj        // The synonyms of ci are represented as si stored in CST
                                  corresponding   to ci's GCID.
                {
                match=string_matching(cj,c);
                if (match=true) then
                return true;
                }
                return false;
```

Fig. 7.6 Synonym Match Algorithm

All identified concept's synonyms are maintained in Concept Synonym Table (CST). The description of CST is explained as follows:

- ***Concept Synonym Table:***

This table stores a list of synonyms of concepts retrieved from wordNet. When a new concept is added in the GCI, its corresponding synonyms are retrieved from the wordnet and stored here so that a knowledge base of concept's synonym can be maintained. This would ultimately fasten the speed of matching process to match concepts with its synonyms offline otherwise they must be retrieved from wordNet every time. The schema of concept synonym table is shown in Fig. 7.7.

GCID	GCN	synonym

Fig. 7.7 Schema of Concept Synonym Table (CST)

The descriptions of various fields maintained by this table are defined in Table 7.4.

Table 7.4 Description of Fields of Concept Synonym Table

Field	Description
GCID	Unique id to each concept in the GCI.
GCN	Name of the Global Concept in GCI.
synonym	List of synonyms (nouns) from wordnet.

b) Property matching operation

In this operation, two properties *pi & pj* belonging to different ontologies *Oi & Oj* respectively are matched to build a global object/data property index. But, before working on matching process, stemming is applied on every object property and data property. Stemming is the process that reduces all forms of the words to a base or stemmed form. For example; if in one ontology O*i*, there exists an object property *father* whose domain and range is *person* and in another ontology O*j* same object property but with the name *hasfather* whose domain and range are *person* exist; in this case, string matching operation would return false despite of the fact that both properties are conveying the same meaning. To deal with this situation, stemming is applied which will reduce both the properties to a base term which is *father* in the present case. The algorithm to perform property matching is shown in Fig. 7.8.

The working of property matching algorithm is described as follows:

1) This algorithm is used as a function for building GOBJPI/GDPI.

2) It takes two properties Pi & Pj as parameter and before matching, it checks if their domains and ranges match or not. If they match then it allows the process to go further, otherwise at this step it returns false.

3) In case, if the above condition matches then the strings corresponding to these properties strings are submitted for string matching to the string matching function.
   a) If strings match, then these two properties are considered as same.
   b) Otherwise, it is checked if they are synonyms of each other and for this they are given to synonym_match function as shown in Fig. 7.6 as a parameter which returns true or false. If function returns true, these properties are considered same otherwise it goes to the next step.

4) If the above three steps do not identify if two properties are same, then it takes suggestion from user and accordingly returns the result either as true or false.

```
Property_Matching (obj1,obj2)
Step1: [Before matching two properties check if their domain and range same]
    If domain[objp1]==domain[objp2]then
                    If range[objp1]==range[objp2]then
          Goto step2
          Else
Return false
Step2:[Check if two properties objp1 and objp2 are lexically same]
Flag=substring_matching(objp1,objp2)
If flag== true
Return flag;
Else
Goto step3
Step3:[Check if two object properties objp1 and objp2 are synonym of each other]
  Flag=synonym_match(objp1,objp2)
  If (flag==true)
  Return flag;
  Else
  Goto step4
Step4:[User Suggestion]
Take feedback from user corresponding to the matching of object property and revert back to the
calling function accordingly.
                    If user answers yes
                    Return true;
                    Else
                    Return false;
Step 5: Exit
```

Fig. 7.8 Property Matching Algorithm

- **Property Synonym Table**

This table stores a list of synonyms of properties retrieved from wordnet. When a new property is added in the GOBJPI/ GDPI, its corresponding synonyms are retrieved from the wordnet and stored here so that a knowledge base of property synonyms can be maintained which would ultimately fasten the speed of matching properties with its synonyms otherwise they have to be retrieved from WordNet every time.

The schema of maintaining property synonym data structure is shown in Fig. 7.9.

GPID	GPN	Synonym

Fig. 7.9 Data Structure for Property Synonym Table

The descriptions of various fields are defined in Table 7.5.

Table 7.5 Property Synonym Table

Field	Description
GPID	Unique id to each property in the global object and data index.
GPN	Name of the global property in respective object and data property index.
synonym	List of synonyms (verb) from WordNet.

7.2.5 Alignment Layer

Once the concepts and properties that belong to different ontologies are matched, they are stored in their respective indexes. This layer contains GCI which contains a list of unique concepts which belong to that domain. Corresponding to those unique concepts, their respective object properties and data properties are stored in GOBJPI and GDPI [170]. To store such information, following data structures are used in the current work:

a) Global Concept Index
b) Global Object Property Index
c) Global Data Property Index

The schemas of these data structures are shown in Fig. 7.10.

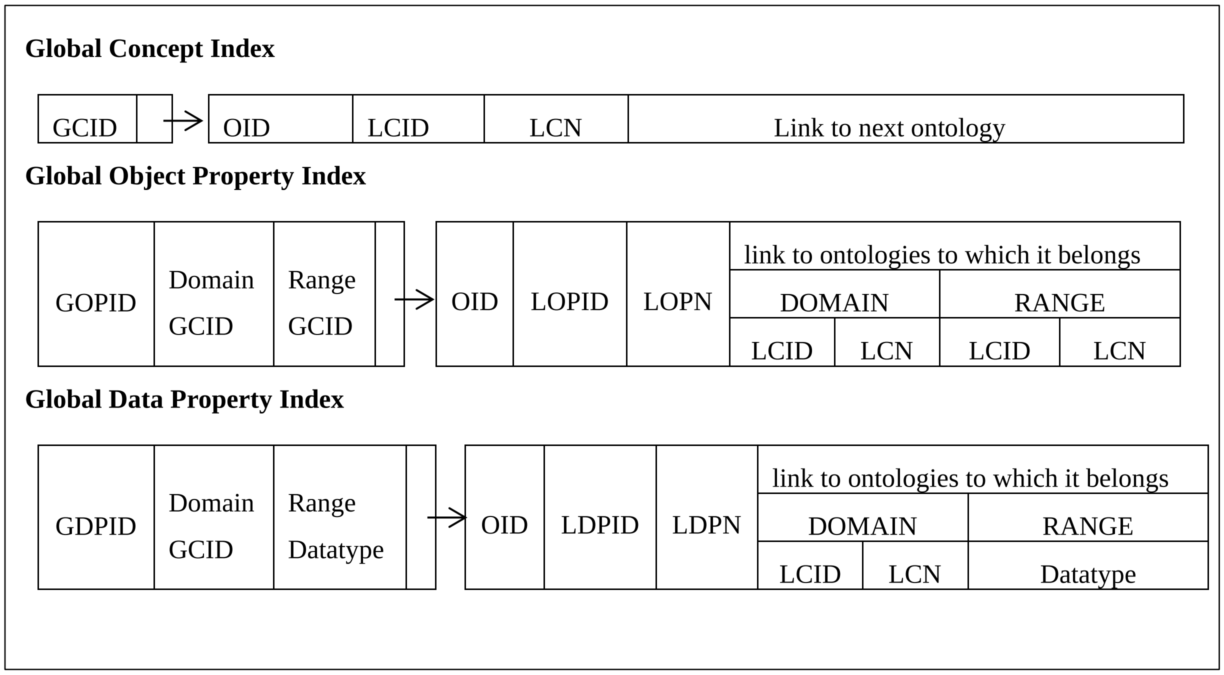


Fig. 7.10 Data Structures Used for Storing Information in Output Layer

a) Global Concept Index

It is well known that an index optimizes speed and performance in finding relevant documents for a search query. Therefore, keeping this in mind, an index is maintained that stores a list of global concepts identified after concept matching process along with the information that contains the list of source ontologies in which this concept was used. The description regarding various fields that are maintained in this table is defined in Table 7.6 which helps global concepts to interlink with their respective source ontologies

Table 7.6 Description of fields of Global Concept Index

Field	Description
GCID	Unique id to each concept in the global concept index
GCN	Name of the global concept in concept index.
OID	Ontology Id.
LCID	Local ID of the concept in its ontology.
Next pointer	Link to the next node.

b) Global Object Property Index

Analogous to Global Concept Index, Global Object Property Index is maintained to store the global object properties that are identified after object property matching process. The descriptions regarding various fields that are maintained in this table are defined as follows in Table 7.7 which helps global object properties to interlink with their respective source ontologies.

Table 7.7 Description of fields of Global Object Property Index

Field	Description
GOPID	Unique id of *obp* belonging to the local ontology in the global object property index.
Domain	Global object property domain concept.
Range	Global object property range concept.
OID	Ontology id.
LOPID	Local Object property id.
LOPN	Local Object Property name.
LCID	Local concept id of domain and range respectively.
LCN	Local concept name of domain and range respectively.

c) Global Data Property Index

Global Object Property Index is maintained to store the global data properties that are identified after object property matching process. The descriptions of various data fields that are maintained in this table are given in Table 7.8 which help global data properties to interlink with their respective source ontologies.

Table 7.8 Description of fields of Global Data Property Index

Field	Description
GDPID	Unique id of data property belonging to the local ontology in the global data property index.
Domain	Global object property domain concept.
Range	Global object property range concept.
OID	Ontology id.
LDPID	Local data property.
LDPN	Local Data Property name.
LCID	Local concept name of domain respectively.
LCN	Local concept name of domain respectively.

d) Building Global Concept Index

After performing the concept matching operation, their corresponding entries are made in the Global Concept Index. The algorithm for building concept index is explained in Fig. 7.11.

The process starts with assuming that GCI is initially empty and thus first ontology's (Oi's) concepts (i. e; ci) are given as seed to the GCI from its Local Concept Table (LCTi). Then, it starts taking other ontologies i.e; Oj's LCTj and starts growing itself by applying following steps.

A variable match is used corresponding to each concept cj which is to be matched against set of concepts ci available in GCI. Initially, match is set to false which indicates that there exists no match corresponding to the new concept.

In the next step, new concept cj is matched with every concept denoted as ci in the GCI starting from the first entry. Here, first it is checked whether ci and cj are syntactically same or not and for this, substring matching and prefix matching algorithm is called. If concepts are found to be syntactically same, then same process restarts for second concept. But, if they are not same, then synonyms are matched and corresponding steps are taken. If cj gets matched with any synonym of ci retrieved from CST, then respective information gets appended to the rear of ci links. Otherwise, a new entry of concept cj is made in GCI.

```
Building Global Concept Index(LCT)
 [Input]Local Concept Tables LCTj of ∀Oj ϵ ontology repository
[Output]Collection of unified concepts in concept index
// Start of algorithm
[Initialization]
When global concept index is empty initially, assign all the concepts ci of ontology Oi from its
local concept table LCTi to the index as seed concepts.

    For each concept cj of LCTj    //  This step adds all the concepts cj of  Oj stored in LCTj in
    GCI.
     {
    match=false  //match is a variable which  holds  result either as true or false. This indicates
    whether concept //cj got matched with any concept ci in GCI or not. Initially it is set to false
    indicating no match exists.

            for each concept ci of GCI        // this step performs matching operation where cj
                                              is matched with //every concept ci in GCI until a
                                              match is not found.
                {
                Switch(1)
                {
                        case 1:       match=substring_matching(cj,ci)
                                        if (match=true)
                                          break;
                case 2: match=prefix_matching(cj,ci)
                                        if (match=true)
                                          break;
                                case 3:match=synonym_match(cj,ci)
                                              If (match=true)
                                              Break;
                }
                }
     }
    If (match==true) // step to be taken if match found
    { create a new node corresponding to cj and link it with existing node ci and add ontology
    name and local concept id in that node. }
    Else // step to be taken if match is not found
    {add cj as the new entry to the rear of concepts in the GCI .
    }
    }
```

Fig. 7.11 Algorithm for Building Global Concept Index

The step by step illustration of proposed work is outlined in Fig. 7.12.

1)First, parser develops LCT with respect to given ontologies retrieved from the source ontology layer. The LCT stores LCID, LCN, URI and GCID which will be assigned to concept when this concept will be added in GCI. These LCT tables will be used as input for building the GCI.

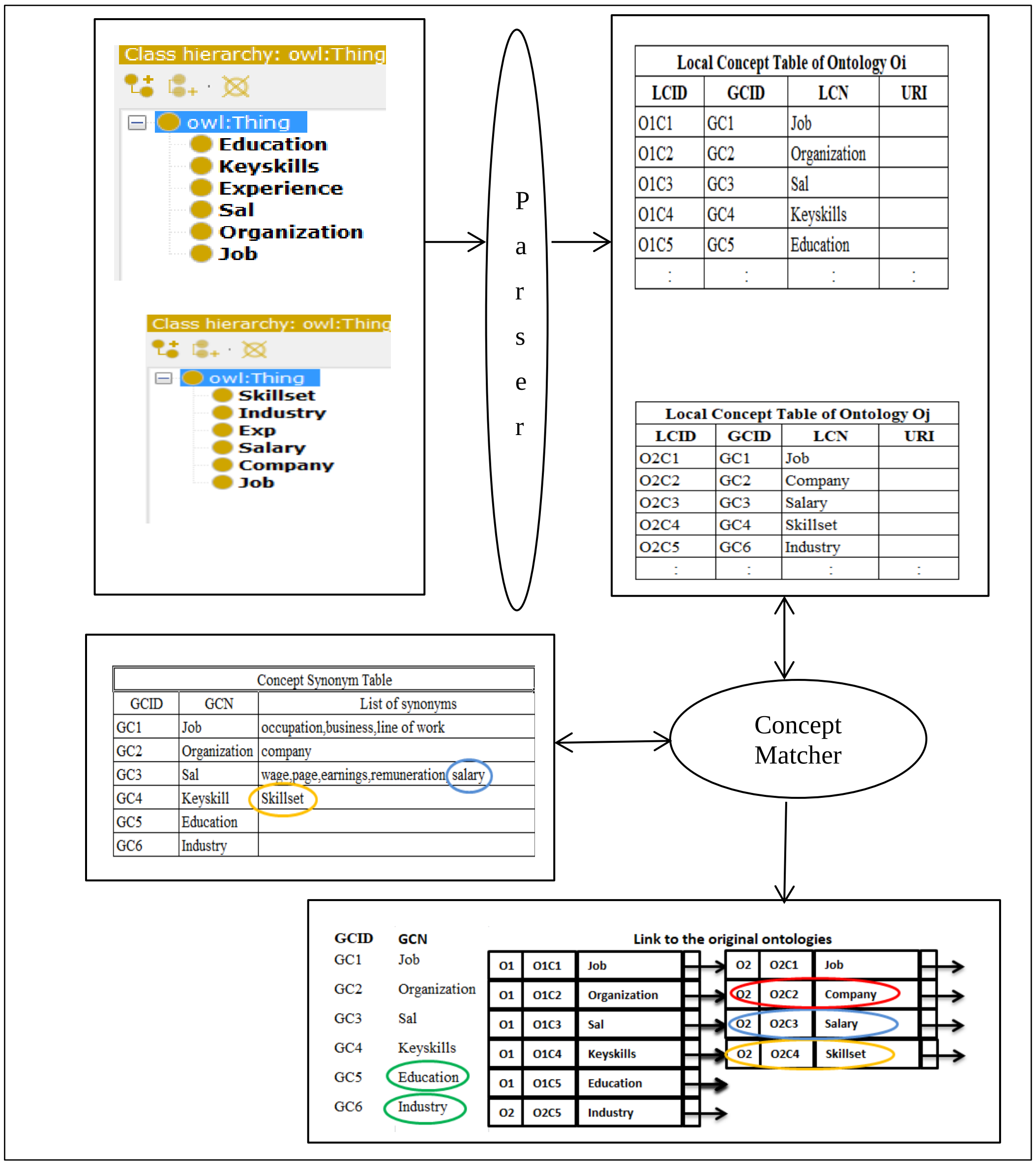
2) In this step, initially first ontology's concepts are given as seed inputs to the GCI. The GCI assigns unique GCID to these newly added concepts and contains the information such as its local concept name and the name of the ontology to which this concept belongs. Here for instance, concept "Job" belongs to ontology O1 and its LCN is *Job*. So, GCI creates a new node where it stores these two-information's corresponding to its GCID. Likewise, it adds the same information for other seed concepts in GCID.

3) A CST table containing a list of synonyms is maintained corresponding to all the concepts which exist in GCI. For example, corresponding to *Job* concept, its synonyms (occupation, business, line of work etc.) are maintained in CST. This table also contains a list of entities which are identified on the similar syntactic matching. For instance, *skillset* and *keyskill* are syntactically similar on the basis of substring matching. Therefore, with respect to *skillset*; *keyskill* is also considered as similar and thus listed in the CST for future reference.

4) From this step, onwards, main process of concept matching starts where it takes concepts from the next ontology's LCT and compares its concepts with the concepts in the GCI. It uses concept matcher algorithm to match two concepts. For instance, it starts with taking first concept from ontology O2's LCT say keyskill. Now, Concept matcher (CM) takes two parameters as an input; one concept "Keyskill" from ontology O2's LCT and one concept from GCI assuming "skillset" as concept chosen from the GCI. Now first it checks whether it is a synonym of *keyskill* from CST. If not, then then it performs substring matching followed by prefix matching in case substring matching fails. In this example, *keyskill* and *skillset* comes out to be similar thus, *skillset* get linked with *keyskill* in GCI. And along with this, CST is also updated by adding skillset as similar concept with respect to *keyskill* for the future purpose as shown using orange circle indicating that it got updated to CST after syntactic matching.in the same way, using prefix matching, *sal* and *salary* comes out to be

similar. Therefore, they are also considered as similar and thus added in the CST encircled using blue colour.

5)If no match is found then in that case, it added as a new concept in GCI as can be seen for concept Education and industry encircled with green colour and its corresponding synonyms are retrieved from the wordnet and added into CST for future purpose.



Local Concept Table of Ontology Oi

LCID	GCID	LCN	URI
O1C1	GC1	Job	
O1C2	GC2	Organization	
O1C3	GC3	Sal	
O1C4	GC4	Keyskills	
O1C5	GC5	Education	
:	:	:	:

Local Concept Table of Ontology Oj

LCID	GCID	LCN	URI
O2C1	GC1	Job	
O2C2	GC2	Company	
O2C3	GC3	Salary	
O2C4	GC4	Skillset	
O2C5	GC6	Industry	
:	:	:	:

Concept Synonym Table

GCID	GCN	List of synonyms
GC1	Job	occupation,business,line of work
GC2	Organization	company
GC3	Sal	wage,page,earnings,remuneration, salary
GC4	Keyskill	Skillset
GC5	Education	
GC6	Industry	

GCID	GCN	Link to the original ontologies					
GC1	Job	O1	O1C1	Job	O2	O2C1	Job
GC2	Organization	O1	O1C2	Organization	O2	O2C2	Company
GC3	Sal	O1	O1C3	Sal	O2	O2C3	Salary
GC4	Keyskills	O1	O1C4	Keyskills	O2	O2C4	Skillset
GC5	Education	O1	O1C5	Education			
GC6	Industry	O2	O2C5	Industry			

Fig. 7.12 Illustration of building Global Concept Index

e) Building Global Object Property Index

After performing the property matching operation, their corresponding entries are made in the Global Object Property Index. The algorithm for Building Global Object Property Index is explained in the Fig. 7.13.

```
Building Global Object Property Index(objP)
 [Initialization]
When global objectproperty index is initially empty, assign all the objectproperty objp of ontology
O to the index as seed objectproperty, objp.
[Input] Stemmed objp of Ontology Oi
[Output]Collection of unified objectproperties in objectproperty index
// Start of algorithm
    1.  Flag=false
    2.  Get the objectproperty objp of ontology Oi from its corresponding local objectproperty
        table LOPT.
    3.  Repeat until objectproperty objp  is compared with each global objp of global index
        3.1 calculate flag=property_Matching (ci,cj)
        3.2 If (flag== true)
        3.3 Create a new node and add the matched objectproperty to its corresponding matching
        global objectproperty
        Else
        3.4 Add a new global objectproperty in the global objectproperty  index.
```

Fig. 7.13 Algorithm for Building Global Object Property Index

The description of building global object property index algorithm is explained step by step as follows:

1) Initially, it is assumed that Global Object Property Index is empty and thus first ontology's stemmed object properties are given as seed to the index from its Local Object Property Table. The main process starts from next step.

2) A variable *flag* is used corresponding to each object property which is to be matched against set of object properties available in the Global Object Property Index. Initially, *flag* is set to false which indicates that there exists no match corresponding to the new object property.

3) In the third step, new object property obj_i is matched with every object property denoted as obj_j in the Global Object Property Index starting from the first entry in the

Global Object Property Index. These two variables are given as parameter to property matching algorithm which returns true or false.

4) If the flag is true, then it represents that it has found its matching object property in Global Object Property Index and thus creates a new node that stores information as explained in data structures discussed above corresponding to the object property obj_j and append this node after the matched concept.

5) Otherwise, it creates a new entry in the Global Object Property Index and adds a new node corresponding to it.

f) Building Global Data Property Index: Analogous to global object property index, after performing the property matching operation, their corresponding entries are made in the Global Data Property Index. The algorithm for Building Global Data Property Index is explained in the Fig. 7.14.

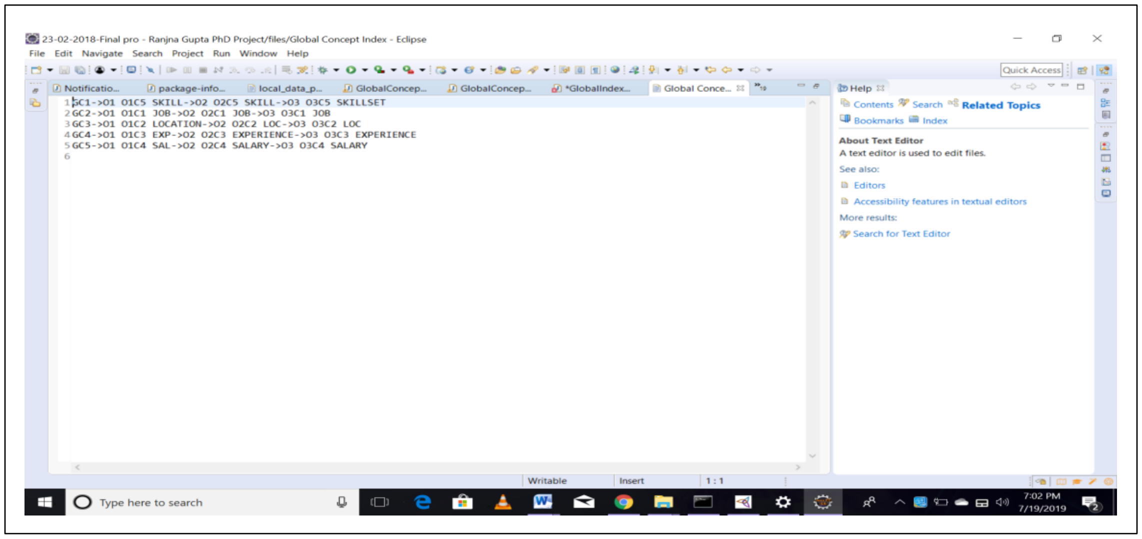
```
Building Global Data Property Index(dp)
 [Initialization]
When global dataproperty index is empty initially, assign all the dataproperty dp of ontology O to
the index as seed dataproperty, dp.
[Input] Stemmed dp of Ontology Oi
[Output]Collection of unified dataproperties in dataproperty index
// Start of algorithm
    1. Flag=false
    2. Get the dataproperty  dp of ontology Oi from its corresponding local dataproperty table
       DPTI.
    3. Repeat until dataproperty objp  is compared with each global dp of global index
       3.1 calculate flag= property_Matching (dpi,dpj)
       3.2 If (flag== true)
       3.3 Create a new node and add the matched dataproperty to its corresponding matching
       global dataproperty
       Else
       3.4 Add a new global dataproperty in the global dataproperty index.
```

Fig. 7.14 Algorithm for Building Global Dataproperty Index

The description of building global data property index algorithm is similar to global object property index as explained above.

7.3 IMPLEMENTATION OF THE PROPOSED WORK

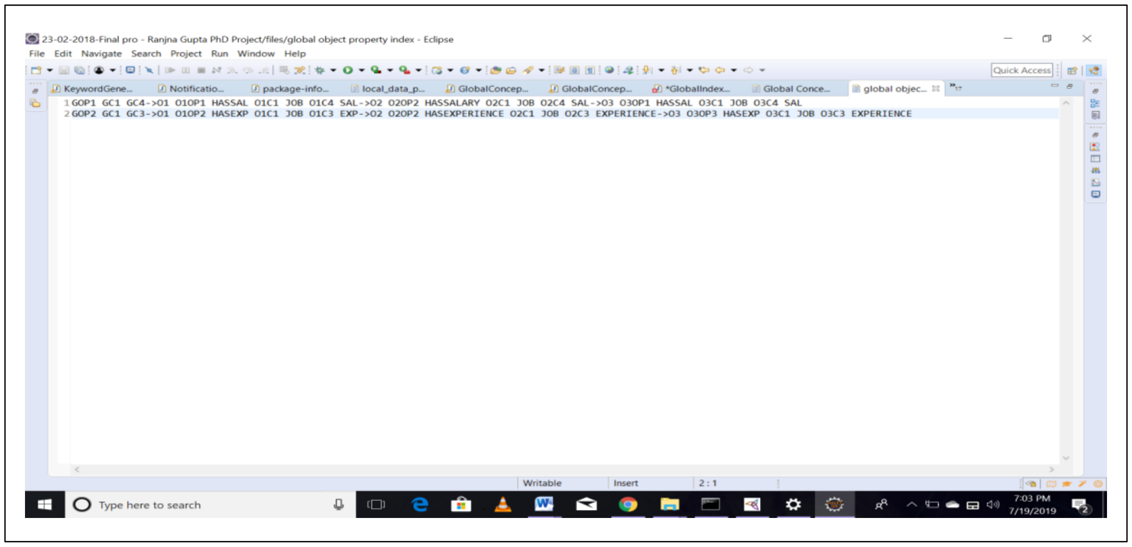
To analyze the proposed work, it has been implemented in Java Eclipse. The proposed system, parsed local concept table, local data property table and object property tables were stored in MYSQL database and then using respective matching algorithms final Global Concept Index, Global Object Property Index and Global Data Property Index are generated. The snapshots of various Global Concept Index, Global Object Property index and Global Data Property Index are shown in Fig. 7.15, Fig. 7.16 and Fig. 7.17 respectively.



```
GC1->01 01C5 SKILL->02 02C5 SKILL->03 03C5 SKILLSET
GC2->01 01C1 JOB->02 02C1 JOB->03 03C1 JOB
GC3->01 01C2 LOCATION->02 02C2 LOC->03 03C2 LOC
GC4->01 01C3 EXP->02 02C3 EXPERIENCE->03 03C3 EXPERIENCE
GC5->01 01C4 SAL->02 02C4 SALARY->03 03C4 SALARY
```

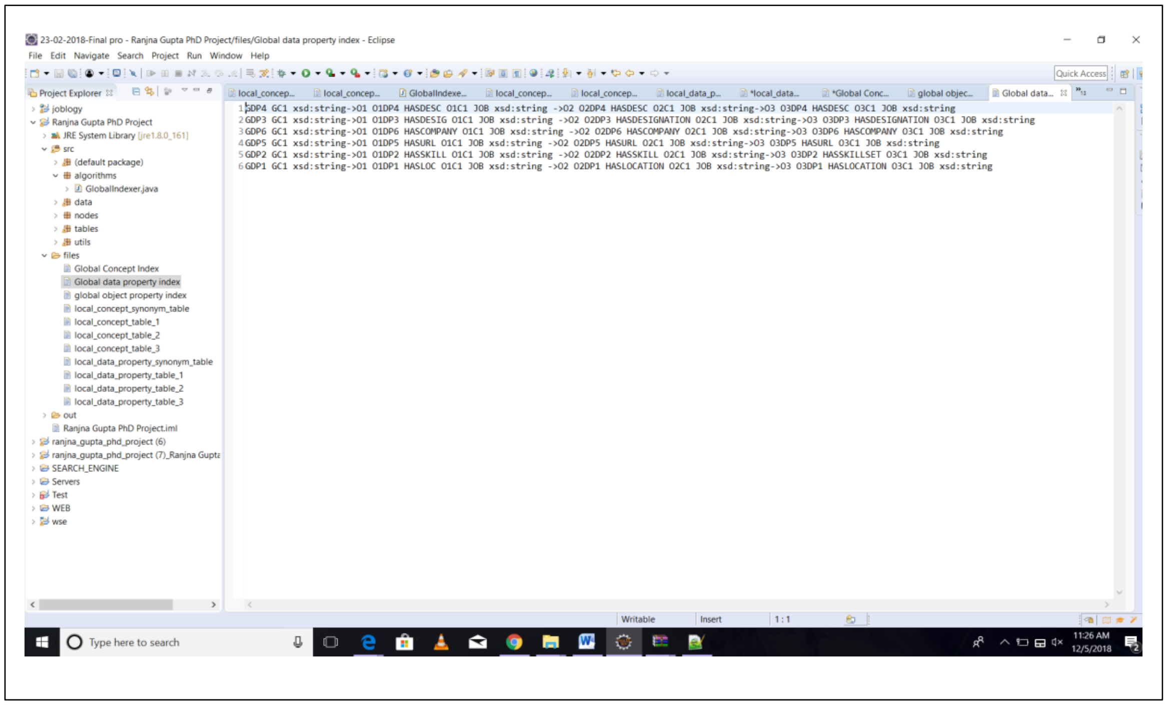
About Text Editor
A text editor is used to edit files.

Fig. 7.15 Snapshot of Global Concept Index



```
GOP1 GC1 GC4->01 01OP1 HASSAL 01C1 JOB 01C4 SAL->02 02OP2 HASSALARY 02C1 JOB 02C4 SAL->03 03OP1 HASSAL 03C1 JOB 03C4 SAL
GOP2 GC1 GC3->01 01OP2 HASEXP 01C1 JOB 01C3 EXP->02 02OP2 HASEXPERIENCE 02C1 JOB 02C3 EXPERIENCE->03 03OP3 HASEXP 03C1 JOB 03C3 EXPERIENCE
```

Fig. 7.16 Snapshot of Global Object Property Index



```
GDP4 GC1 xsd:string->O1 O1DP4 HASDESC O1C1 JOB xsd:string ->O2 O2DP4 HASDESC O2C1 JOB xsd:string->O3 O3DP4 HASDESC O3C1 JOB xsd:string
GDP3 GC1 xsd:string->O1 O1DP3 HASDESIG O1C1 JOB xsd:string ->O2 O2DP3 HASDESIGNATION O2C1 JOB xsd:string->O3 O3DP3 HASDESIGNATION O3C1 JOB xsd:string
GDP6 GC1 xsd:string->O1 O1DP6 HASCOMPANY O1C1 JOB xsd:string ->O2 O2DP6 HASCOMPANY O2C1 JOB xsd:string->O3 O3DP6 HASCOMPANY O3C1 JOB xsd:string
GDP5 GC1 xsd:string->O1 O1DP5 HASURL O1C1 JOB xsd:string ->O2 O2DP5 HASURL O2C1 JOB xsd:string->O3 O3DP5 HASURL O3C1 JOB xsd:string
GDP2 GC1 xsd:string->O1 O1DP2 HASSKILL O1C1 JOB xsd:string ->O2 O2DP2 HASSKILL O2C1 JOB xsd:string->O3 O3DP2 HASSKILLSET O3C1 JOB xsd:string
GDP1 GC1 xsd:string->O1 O1DP1 HASLOC O1C1 JOB xsd:string ->O2 O2DP1 HASLOCATION O2C1 JOB xsd:string->O3 O3DP1 HASLOCATION O3C1 JOB xsd:string
```

Fig. 7.17 Snapshot of Global Data Property Index

7.4 SUMMARY

In this chapter, a novel method for ontology alignment is proposed which supports

a) taking n number of ontologies as an input which are to be aligned concurrently,
b) performing semantic matching on concepts, data properties and object properties,
c) developing knowledge base of synonym of concepts and properties to fasten the matching process,
d) building Global Concept Index, Global Data Property Index, Global Object Property Index which store all information of the merged ontologies and maps them with their local ontologies to which they actually belong, thereby supporting backward engineering. These indexes will be very helpful in making query processing easier and faster.

In the next chapter, emphasis will be on query processing module to make querying and retrieval from the system systematic and fast.

Chapter VIII

ONTOJOB QUERY PROCESSOR: AN ONTOLOGY DRIVEN QUERY PROCESSING METHOD

8.1 GENERAL

In the previous chapter, Ontology alignment module was explained in which alignment between the heterogeneous data sources was done using ontology. Moving ahead, next step is to handle user queries and retrieve relevant results from multiple of data sources and presenting to the user at one place.

In this work, "OntoJob" query processing design is being proposed that transforms keyword based user query into SPARQL query with respect to each data source. These queries are then run on their respective ontologies individually and then the results are presented at one place by merging the result generated from different data sources. By processing the query in such a way, navigation time of the user can be decreased while increasing the precision of the results obtained.

8.2 PROPOSED SYSTEM FOR ONTOJOB QUERY PROCESSING

The aim of the proposed work is to present a novel architecture for query processing on aligned ontologies. Here, a repository of different datasets is maintained as listed in Table 8.1 which plays a very important role during query processing because ontologies are normally defined at conceptual level.

Table 8.1 List of Datasets

S.No.	Dataset	Description
1	Skill Dataset	List of skillsets.
2.	Indlocation Dataset	List of locations.
3.	Salary Dataset	List of salary packages from minimum to maximum range.
4.	Experience Dataset	List of experiences in terms of years a Job can ask for.
5.	Designation dataset	List of Job titles.

These datasets maintain a list of their respective data. These datasets are built by extracting relevant data from various Job boards. With the recognition of new data, the information gets updated in its corresponding dataset. The architecture of Query processor is shown in Fig. 8.1.

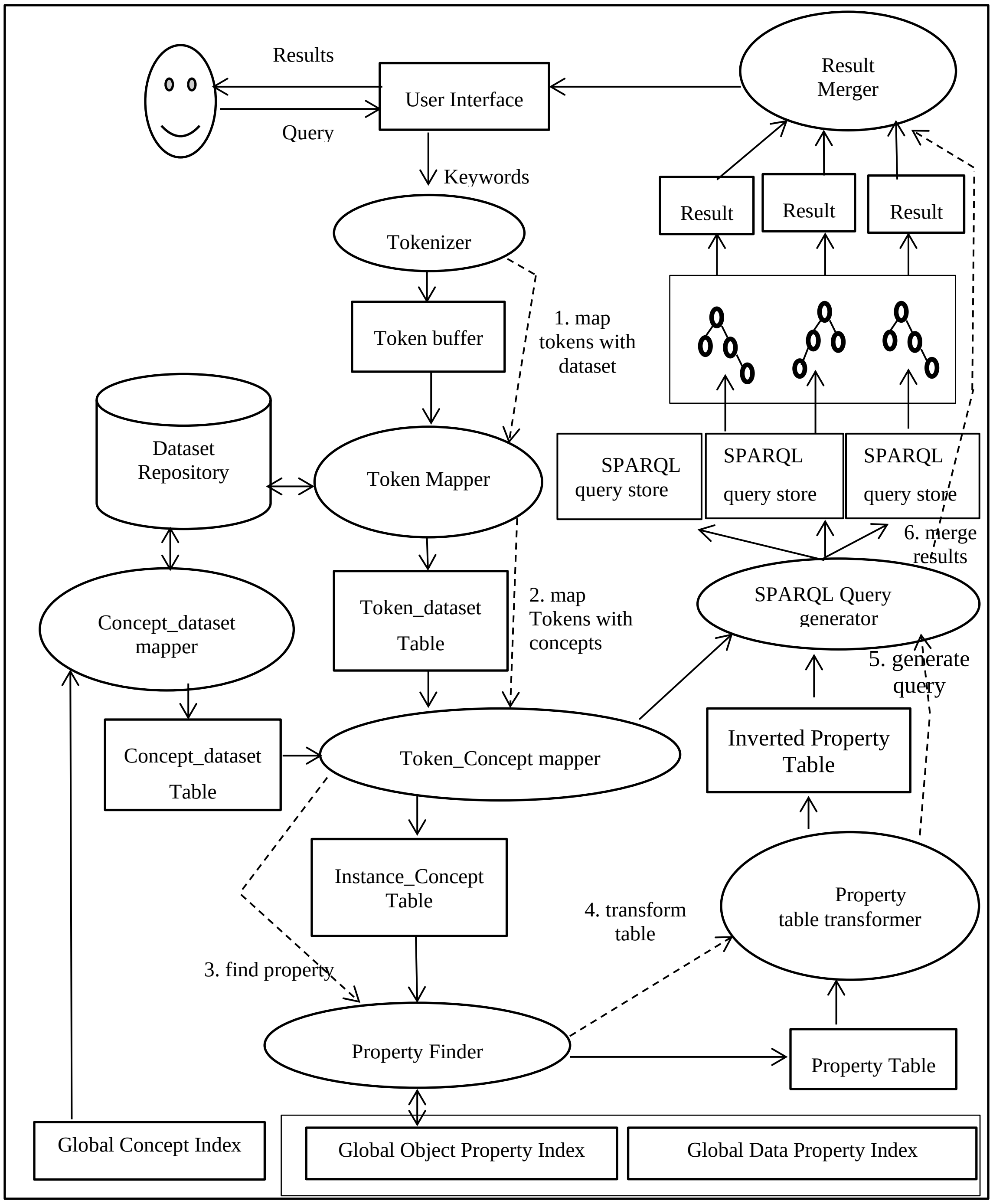


Fig. 8.1 Architecture of Query Processing Process

At an abstract level, the process starts with tokenizing the query given by user which resides at the *Token Buffer*. Query is entered by the user in keyword form and keywords are separated by the delimiter by the user itself. Once the tokenization is done, *tokenizer* sends signal to *Token Mapper* to find if token belongs to any dataset and respective information gets stored in *Token_Dataset table*. At the back end, *Dataset_Concept mapping table* is maintained which contains a list designating which concept is mapped with which dataset. For instance, skill dataset contains a list of keyskills which can be the instance of skill concept. Once this is done, *token mapper* sends the signal to T*oken_Concept mapper* to map the tokens with their respective concepts. For this, *Token_Concept mapper* refers *Token_Dataset table* and C*oncept_Dataset table* and generates *Instance_Concept table*. Once, it is decided that token belongs to which concept; next task is to find the relation between the classes and for this, *Token_Concept mapper* sends the signal to *property finder* to find the relation and the ontologies in which those concepts and property exist.

The generated information gets stored in the *Object Property table and Data Property Table*. Once this is done, *property finder* sends signal to *Property Transformer* which creates an *Inverse Property Table* by placing all the properties at one place corresponding to each ontology which would be helpful in planning a query with respect to selected ontologies. Once this is done, *Inverse Query Transformer* sends signal to *Query Generator* process to take input from *Inverse Property Table* and generate individual SPARQL queries for selected ontologies. These queries are then fired to the respective ontologies which in turn generates results. At last, *Result Merger* merges all the results and displays it to the user at one place.

8.2.1 Various Data Structures used for Query Processing

The schemas of various data structures used during query processing are shown in Fig. 8.2 are explained as under.

1) **Token Buffer**

   This buffer contains the tokens, *t* generated by *tokenizer*. The descriptions of various fields maintained in this table are described in Table 8.2.

.

Table 8.2 Description of Token Buffer

Field	Description
TID	Unique id of token *t* in the query *q*.
TN	Token Name in the query *q*.

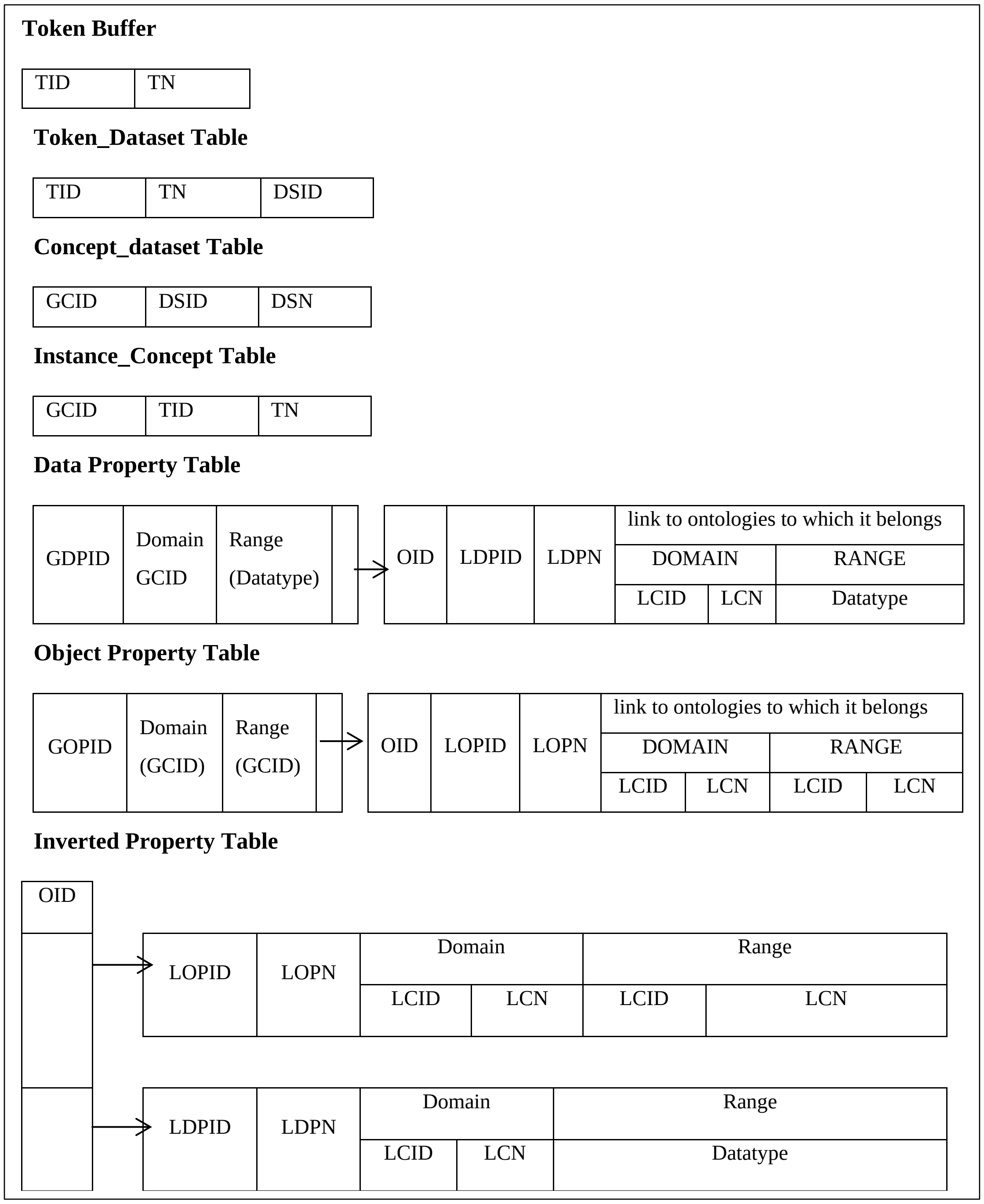


Fig. 8.2 Schema of Various Data Structures Used in Query Processor

2) Dataset Table

This table contains the list of datasets with their unique ids and names stored in dataset repository. The description of various fields of *Dataset Table, DS* are described in Table 8.3.

Table 8.3 Description of Dataset Table

Field	Description
DSID	Unique id of dataset *ds* in dataset repository.
DSN	Name of Dataset *ds*.

3) Token_Dataset Table

Token mapper process finds whether a token belongs to any dataset maintained in dataset repository and its corresponding information gets stored in *Token_Dataset Table, TDT*. The descriptions of various fields of this table are described in Table 8.4.

Table 8.4 Description of Token_Dataset Table

Field	Description
TID	Unique id of token *t* in the query *q*.
TN	Token Name in the query *q*.
DSID	Unique Dataset id of *ds*.

4) Concept_Dataset Table

This table stores the mapping information between concept and dataset. For instance, skill dataset is mapped with skill concept. The descriptions of various fields maintained in *Concept_Dataset Table, CDT* table are described in Table 8.5.

Table 8.5 Description of Concept_Dataset Table

Field	Description
GCID	Global Concept id of the concept *c* maintained in GCI.
DSID	Unique Dataset id of *ds*.
DSN	Name of the dataset *ds*.

5) Token_Concept Table

This table stores the mapping information between token and concept. For instance, if token ‘PHP’ belongs to skill dataset and skill dataset is mapped with skill concept then PHP is considered as instance of skill concept. The descriptions of various fields maintained in *Token_Concept Table, TCT* are described in Table 8.6.

Table 8.6 Description of Instance_Concept Table

Field	Description
GCID	Global Concept id of the concept *c* maintained in GCI.
TID	Unique id of token *t* in the query *q*.
TN	Token Name in the query *q*.

6) Property Table

Once the tokens get mapped with their respective concepts, next step is to find the relationship that exists between the concepts. For this, all the relationships which are maintained between Job concept and identified concept *c* are collected from *GOPI* and *GDPI* which in turn are maintained in respective *Object Property Table (OPT)* and *Data Property Table (DPT)*. The descriptions of various fields maintained in *Property Table, PT* are described in Table 8.7 and

Table 8.8 respectively.

Table 8.7 Description of Object Property Table

Field	Description
GOPID	Unique id of *op* belonging to the local ontology *O* in the GOPI.
OID	Ontology id of ontology *O*.
Domain	GCID as domain of *op*.
Range	GCID as range of *op*.
LOPID	Local Object Property id of object property *op*.
LOPN	Local Object Property name of *op*.
LCID	Local concept id of domain and range of *op* respectively.
LCN	Local concept name of domain and range of *op* respectively.

Table 8.8 Description of Data Property Table

Field	Description
GDPID	Unique id of data property *dp* belonging to the local ontology *O* in the GDPI.
Domain	GCID as domain of d*p*.
Range	GCID as range of *dp*.
OID	Ontology id of Ontology *O*.
LDPID	Local Data Property id of data property *dp*.
LDPN	Local Data Property name of d*p*.
LCID	Local concept id of domain and range of *dp* respectively.
LCN	Local concept name of domain and range of *dp* respectively.

7) **Inverted Property Table**

This table is an inverted version of *Property Table*. It contains the list of properties with respect to individual ontology which will help in constructing SPARQL query. The descriptions of various fields maintained in *Inverse Property Table, IPT* are described in Table 8.9.

Table 8.9 Description of Property Table

Field	**Description**
OID	Ontology ID of ontology *O*.
LOPID	Local Object Property id of object property *op*.
LOPN	Local Object Property name of *op*.
LCID	Local concept id of domain and range respectively.
LDPID	Local Data property id of data property *dp*.
LDPN	Local Data Property name of *dp*.

The details of the various modules along with their working are outlined as below:

8.2.2 Component Modules of Query processor

The six main components that are used during query processing are given as:

a) Tokenizer
b) Token Mapper
c) Dataset_Concept Mapper
d) Token_Concept Mapper
e) Property finder
f) Property table transformer

A brief description of the above parameters is given below:

a) Tokenizer

It takes user query keywords as an input and split it into tokens. The generated tokens are stored *in token buffer*. Once the tokens have been generated, it then sends signal to *token mapper* for further process.

The algorithm for tokenizing the user query is shown Fig. 8.3.

```
Tokenizer ( str, l, e, s,d1)
{  [Input:] keyword str given by the user via keyword interface.
            Location l given by the user via location interface.
            Experience e given by the user via experience interface.
            Salary s given by the user via salary interface.
            Delimiter d1 between the keywords.
   [Output:] token_list stored in token_buffer [10][10].
  Initialize word=""
Initialize num=0
Str=str+d1
l=str.size
set i=0
            while (i< l-1)
            {
                        if(str[i]!=d1)
                                    word=word+str[i]
                        elseif(word.size!=0)
                                    {
                                                token_list[num]=word       // storing token in token buffer
                                                num=num+1
                                    }
                        word=""
            }
return num;
Signal (map token)
}
```

Fig. 8.3 Tokenizer Algorithm

b) Token Mapper

Upon getting the signals from *Tokenizer*, it finds to which dataset the token may belong to. For instance, if a *java* as token is received from *token buffer*, and if that is found in the skill dataset, then at the generalized level, it would be considered as skill. The process of mapping a token with respective dataset is defined in Fig. 8.4.

```
Token_mapper(t)
{
[Input:]token t from token buffer.
[Output]:token_concept table.
wait(map token)
for each token t ϵ token_buffer {
      For each dataset ds ϵ dataset_repository {
            if (t ϵ ds)
                        {
                                    store(t,dsname) to token_dataset table.
                                    signal(map token with concept)
                        }
                                                            }
                                                }
}
```

Fig. 8.4 Token mapper Algorithm

c) Dataset_Concept Mapper

This process maps datasets present in dataset repository with the concepts indexed in GCI. This is a single time process in which concepts are already mapped with the datasets. For instance, skill concept is mapped with skill dataset; location concept is mapped with location dataset and so on. The process of mapping a concept with respective dataset is defined in Fig. 8.5.

```
Dataset_concept mapper (ds)
{
[Input:]Dataset ds є dataset repository.
Dataset repository={skillds, locds, expds, salds}
Global Concept Index, GCI.
[Output:]dataset_concept table.
Map skillds with skill concept.
Map locds with location concept.
Map expds with experience concept
Map salds with salary concept.
Generate dataset_concept table.
}
```

Fig. 8.5 Dataset_Concept Mapper Algorithm

d) Token_Concept Mapper

This process maps tokens with their respective concepts by referring *token_dataset table* which contains a list of tokens along with the dataset (to which they belong) and *dataset_concept table* which holds a list of concepts mapped with datasets. By joining these two tables, the resultant table *Token_concept table* gets generated which contains a list of instances with their respective concepts. The process of mapping a concept with respective instances is given in Fig. 8.6.

```
Token_Concept Mapper (tdt,cdt)
{
[Input:]Token_Dataset table.
          Dataset_concept table.
 [Output:]token_Concept table.
token_Concept table=Apply join between Token_Dataset table and Instance_Concept Table.
}
```

Fig. 8.6 Token_Concept mapper algorithm

e) Property finder

Once the concept has been identified with respect to query keywords, *Token_concept mapper* sends a signal to *property finder* to find the relation that exists between the concept ‘Job’ and the identified concept. *Property finder* refers to *GOPI* and *GDPI* and retrieves the property that exists between the two concepts and stores it into the Object *Property Table (OPT) and Data Property Table (DPT)*. Along with this, it retrieves other information such as ontologies in which this property exists; and concepts in domain and range which would be required during query building. Once it is done, it sends transform property signal to *Property Transformer process*. The process of property finding from the respective indexers is defined in Fig. 8.7.

```
Property Finder (ci, cj)
{
[Input:] ci is a Job concept that will be domain for every property p.
            cj is a concept/datatype that will be range for the property p.
[Output:] OPT and DPT.
Wait (find property)
if p ϵ GOPI  then store p in OPT.
if p ϵ GDPI then store p in DPT.
signal (transform table)
}
```

Fig. 8.7 Property Finder Algorithm

f) Property Table Transformer

The data generated from *Property Finder* process gets collected in its respective OPT and DPT. In this table, the head of every row is the property followed by the nodes containing information about the ontology and local property. This defines the ontologies in which the property exists. *Property Table Transformer* represents the same information but in inverted form. This process upon getting the signals from *property finder* process creates an *Inverted property table* in which each row is headed with ontology name followed by the nodes containing the properties that are identified from *GOPI* and *GDPI* with respect to the query. After this, a signal is sent to a

SPARQL_query_generator process. With this step, writing SPARQL query becomes a simple process. The process of *Property Table Transformer* is defined in Fig. 8.8.

```
Property_table_transformer(OPT, DPT)
{
[Input:]OPT and DPT.
[Output]: Inverted Property Table, IPT.
Wait(transform table)
Collect the ontologies to be indexed.
IPT=Index the properties by creating an inverted index.
Signal(generate query)
}
```

Fig. 8.8 Property Table Transformer Algorithm

Once the instances get mapped with their respective concepts and properties have been identified, next step is to generate SPARQL queries. In the next section, the process of generation of SPARQL queries with respect to ontologies is presented.

8.3 GENERATION OF SPARQL QUERIES

This phase generates SPARQL queries with respect to selected ontologies using *Token_Concept Table*. The process of SPARQL Query Generator is shown in Fig. 8.9.

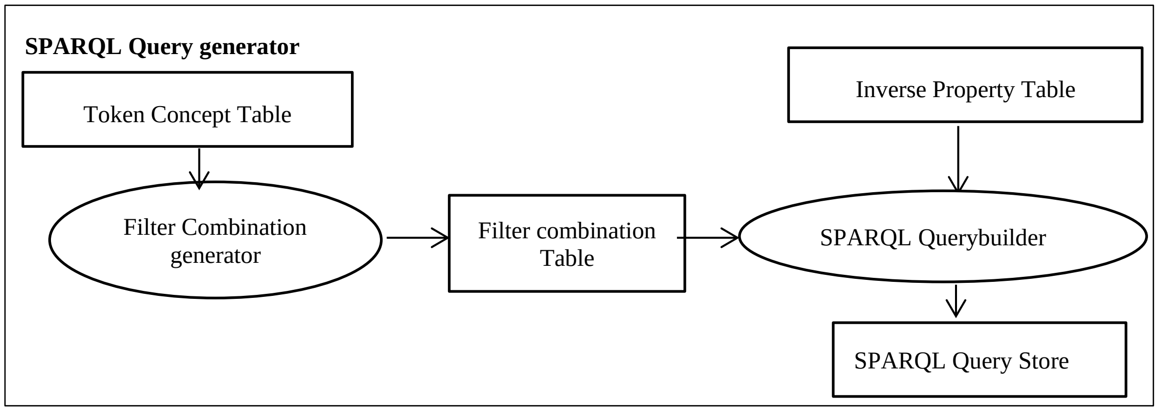


Fig. 8.9 SPARQL Query Generator

It starts working once it gets signal from the *Inverse Property Transformer*. It plans separate queries corresponding to each ontology listed in the property table. It collects ontology name, property name, domain and range, and constructs separate SPARQL queries with respect to the ontologies. It performs two tasks: It first, builds SPARQL Queries using SPARQL Query builder process and second, generates filters that will be appended to SPARQL queries using Filter combination generator. The process of generating filters using Filter Combination generator is shown in Fig. 8.10.

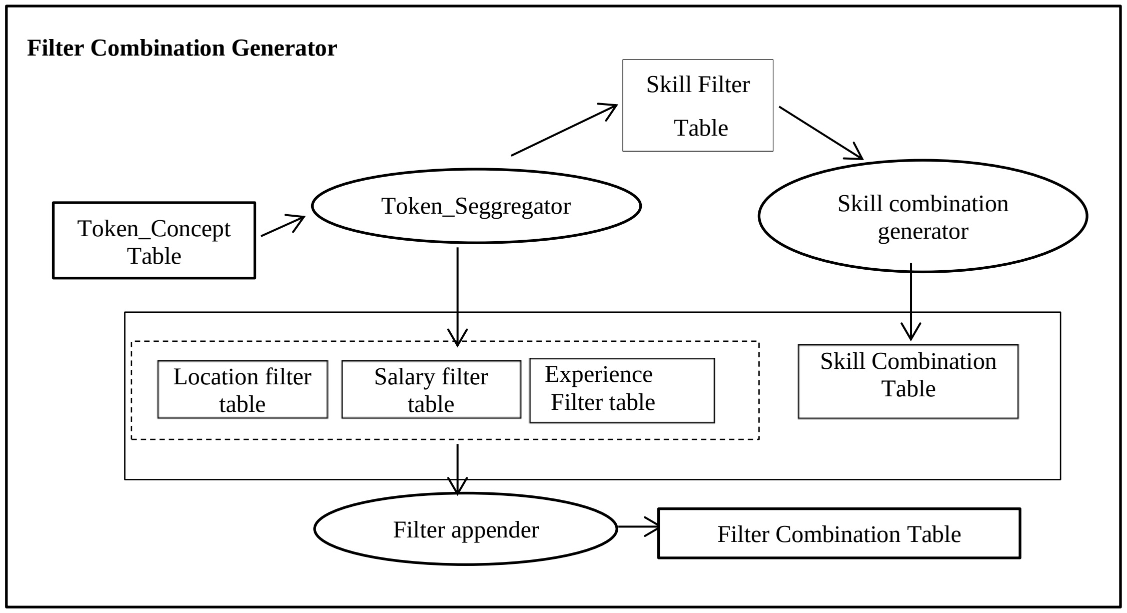


Fig. 8.10 Filter Combination Generator

The process of SPARQL query generation is defined in Fig. 8.11.

At one side, *filter combination generator* generates all the combinations of the tokens and stores them in the filter combination table and on the other side, *SPARQL querybuilder* builds SPARQL queries using *inverse property table*. it then appends filters in the query by taking them from filter combination table and builds final SPARQL query.

Once the SPARQL queries have been generated, these are applied on their respective ontologies to retrieve results which in turn are stored in their respective result tables. Along with this, each row of the result table contains the count of filters taken from filter combination table. For instance, if the filter_count of the retrieved Jobposts is 4, then this designates that this Job post contains 4 keywords given by user. This will be required during merging operation. Once this is done, it sends signal to result merger to merge the results and present to the user.

```
SPARQL_Query_generator (PT, TCT)
{
[Input:]Property Table, PT; Token_Concept Table.
[Output:] SPARQL queries.
Wait (generate query)
Transform property table into ontology based property table.
        For each ontology Oi
        {
                SPARQL query q=Build SPARQL query().
                Filter_combination_table=Filter_combination_generator(TCT);
                   While(!empty(filter combination table))
                        {
                        fi=Get a filter from filter combination table.
                        q=append(q,fi).
                        Store q in SPARQL Query Store
                        run q.
                        Store the results in Result table RTi.
                        }
                Find and remove duplicate Jobpost from the result table.
                Signal (merge results).
        }
}

Build SPARQL query()
{
Create a SPARQL query format where prop1, prop2 etc. are properties from inverted property table ;
?var1(domain) denotes to Job concept and ?var2,?var3(range) denotes to concept stored in
concept_instance table.
"PREFIX rdf:<>"
"PREFIX p:<>"
"select ?var1 ?var2… ?var k where {?var1 p:prop1 ?var2.
  ?var1 p:prop1 ?var3.  …  … ?var1 p:prop1 ?var k.
}"
Return query;
}

Instance_segregator()
{
Create a skill_instance schema with two fields: skill_num and skill_name;
Store the skill instances from instance_concept table into the skill_instance table;
Create a location_instance schema with two fields: loc_num and loc_name;
Store the location instances from instance_concept table into the location_instance table;
Create an exp_instance schema with three fields: exp_num, min_exp and max_exp;
Store the experience instances from instance_concept table into the exp_instance table;
Create a sal_instance table with two fields: sal_num and sal;
Store the salary instances from instance_concept table into the sal_instance table;
}
```

Fig. 8.11 Process of SPARQL Query Generator

The step by step process of Filter Combination Generator is shown in Fig. 8.12.

```
Filter_Combination_Generator(TCT)
{
// create all the possible combination of skill instances.
//no_of_skills is the total number of skills given by user in query.
n=1// no. of combination.
for(r=1;r<=no_of_skills;r++)
{
 skill_comb= Combination (skill_instance[],n,r RTr)
}
Filter_combination table= Concatenator(skill_comb,location_instance table,exp_instance table,
sal_instance table)//Concatenate skill_comb with location_instance table, exp_instance table and
salary_instance table if exist and store the results in filter_combination table.
Add filter_count column in Filter_combination table.
Count no. of  filters in each row of filter_combination table and update the filter_count.
Return
}


Skill_Combination_generator (skill_instance,n,r)
{
// create a temporary table data that will store all the combination of skill.
Skill_comb = Combinationutil(skill_instance,data,0,n-1,r) // Store the possible generated skill
combinations in Skill_comb list.
Return skill_comb;
}
Combinationutil (skill_instance, data,start,end,index,r)
{
// skill instance is a list that stores all skill instances.
// data is a temporary table to store current combination.
// start and end are starting and ending indexes in skill_instance list.
// r is the size of a combination.
If(index== r)
{
    For(j=0;j<r;j++)
    Store skill_instance[skill_name] in data table.
return
}
For(i=start;i<=end && end- i+1>r-index;i++)
    {
    Data[index]=skill_instance[i].skill_num
    Combinationutil(skill_instance, data,i+1,end,index+1,r)
    }
}
Concatenator(skill_comb,location_instance table,exp_instance table, sal_instance table)
{
Concatenate skill_comb with location_instance table, exp_instance table and salary_instance table, if
exists and store the results in filter_combination table.
}
```

Fig. 8.12 Process of Filter Combination Generator

Once the SPARQL queries have been generated, these are applied on their respective ontologies to retrieve results which in turn are stored in their respective result tables. Along with this, each row of the result table contains the count of filters taken from filter combination table. For instance, if the filter_count of the retrieved Jobposts is 4, then this designates that this Job post contains 4 keywords given by user. This will be required during merging operation. Once this is done, it sends signal to result merger to merge the results and present to the user.

8.4 RESULT MERGER

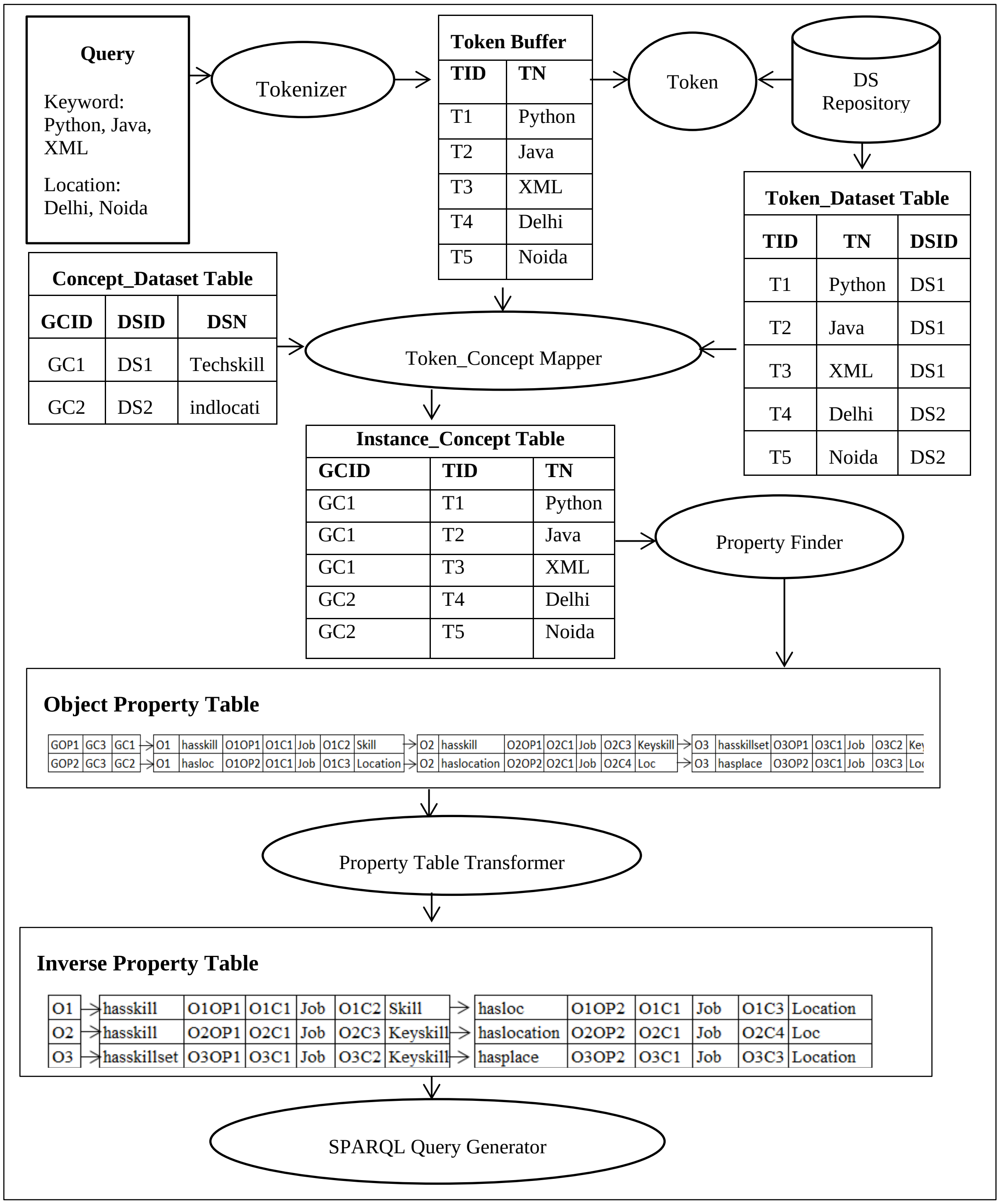
It merges the results upon getting the signal from query generator. Now the motive is to provide those Jobposts on the top from various Job boards which contain maximum keywords given by user at user interface. Therefore, looking at the filter_count in each result table, merging operation is performed. For instance, Jobposts containing all the keywords will be displayed on the top with Jobpost having less matching keywords downwards. Further user can sort the results based on date and time of Job post uploaded. The process of result merging is defined in Fig. 8.13.

```
Result_merger (RT1, RT2,.., RTn)
{
[Input:] result tables RTi
[Output:] results.
Wait (merge results)
merge (RT1,RT2,..,RTn)
Display results via user interface to user.
}
```

Fig. 8.13 Process of Result Merger

8.5 EXAMPLE ILLUSTRATION & IMPLEMENTATION

The illustration as shown in Fig. 8.14 shows the formation of Inverse Property table which will be used for SPARQL query construction. Considering a query: **Python, Java, XML. Location: Delhi, Noida**



Token Buffer

TID	TN
T1	Python
T2	Java
T3	XML
T4	Delhi
T5	Noida

Token_Dataset Table

TID	TN	DSID
T1	Python	DS1
T2	Java	DS1
T3	XML	DS1
T4	Delhi	DS2
T5	Noida	DS2

Concept_Dataset Table

GCID	DSID	DSN
GC1	DS1	Techskill
GC2	DS2	indlocati

Instance_Concept Table

GCID	TID	TN
GC1	T1	Python
GC1	T2	Java
GC1	T3	XML
GC2	T4	Delhi
GC2	T5	Noida

Object Property Table

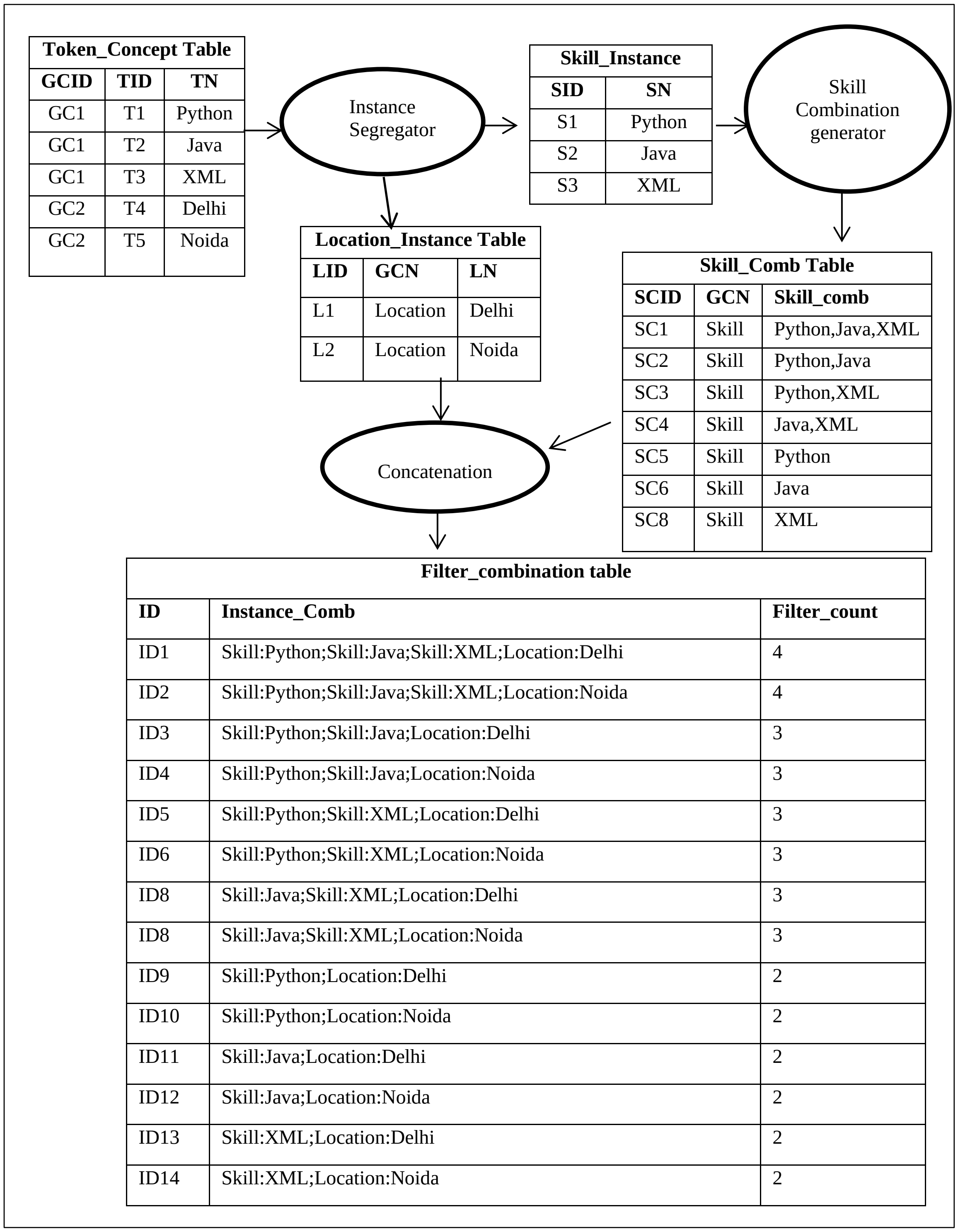
GOP1	GC3	GC1	O1	hasskill	O1OP1	O1C1	Job	O1C2	Skill	O2	hasskill	O2OP1	O2C1	Job	O2C3	Keyskill	O3	hasskillset	O3OP1	O3C1	Job	O3C2	Key
GOP2	GC3	GC2	O1	hasloc	O1OP2	O1C1	Job	O1C3	Location	O2	haslocation	O2OP2	O2C1	Job	O2C4	Loc	O3	hasplace	O3OP2	O3C1	Job	O3C3	Loc

Inverse Property Table

O1	hasskill	O1OP1	O1C1	Job	O1C2	Skill	hasloc	O1OP2	O1C1	Job	O1C3	Location
O2	hasskill	O2OP1	O2C1	Job	O2C3	Keyskill	haslocation	O2OP2	O2C1	Job	O2C4	Loc
O3	hasskillset	O3OP1	O3C1	Job	O3C2	Keyskill	hasplace	O3OP2	O3C1	Job	O3C3	Location

Fig. 8.14 Formation of Inverse Property Table

Once the Inverse Property table is generated, next step is to build filters. The illustration of building filters is shown in Fig. 8.15.



Token_Concept Table

GCID	TID	TN
GC1	T1	Python
GC1	T2	Java
GC1	T3	XML
GC2	T4	Delhi
GC2	T5	Noida

Skill_Instance

SID	SN
S1	Python
S2	Java
S3	XML

Location_Instance Table

LID	GCN	LN
L1	Location	Delhi
L2	Location	Noida

Skill_Comb Table

SCID	GCN	Skill_comb
SC1	Skill	Python,Java,XML
SC2	Skill	Python,Java
SC3	Skill	Python,XML
SC4	Skill	Java,XML
SC5	Skill	Python
SC6	Skill	Java
SC8	Skill	XML

Filter_combination table

ID	Instance_Comb	Filter_count
ID1	Skill:Python;Skill:Java;Skill:XML;Location:Delhi	4
ID2	Skill:Python;Skill:Java;Skill:XML;Location:Noida	4
ID3	Skill:Python;Skill:Java;Location:Delhi	3
ID4	Skill:Python;Skill:Java;Location:Noida	3
ID5	Skill:Python;Skill:XML;Location:Delhi	3
ID6	Skill:Python;Skill:XML;Location:Noida	3
ID8	Skill:Java;Skill:XML;Location:Delhi	3
ID8	Skill:Java;Skill:XML;Location:Noida	3
ID9	Skill:Python;Location:Delhi	2
ID10	Skill:Python;Location:Noida	2
ID11	Skill:Java;Location:Delhi	2
ID12	Skill:Java;Location:Noida	2
ID13	Skill:XML;Location:Delhi	2
ID14	Skill:XML;Location:Noida	2

Fig. 8.15 Illustration of Creating Filters

The output of Filter Combination Generator i.e.; Filter Combination Table and Inverse property table is given to SPARQL Query builder which yields various SPARQL queries. The sample of newly generated SPARQL queries from the above process is shown in Fig. 8.16

QID	SPARQL Query	Filter_count

O1Q1	"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p:
	<http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?Job ?title ?skill ?location where" {
	"?Job p:hasloc ?location."+ "?Job p:hasskill ?skill."+
	“FILTER(?skill=”Python”)." + “FILTER(?skill=”Java”)."+
	“FILTER(?skill=”XML”)."+ “FILTER(?location=”Delhi”)."
	}
		4

O1Q2	"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p:
	http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?Job ?title ?skill ?location where" {
	"?Job p:hasloc ?location."+ "?Job p:hasskill ?skill."+
	“FILTER(?skill=”Python”)." + “FILTER(?skill=”Java”)."+
	“FILTER(?skill=”XML”)."+ “FILTER(?location=”Noida”)."
	}
		4

O1Q3	"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p:
	<http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?Job ?title ?skill ?location where" {
	{
	"?Job p:hasloc ?location."+ "?Job p:hasskill ?skill."+
	“FILTER(?skill=”Python”)." + “FILTER(?skill=”Java”)."+
	“FILTER(?location=”Delhi”)."
	}
		3

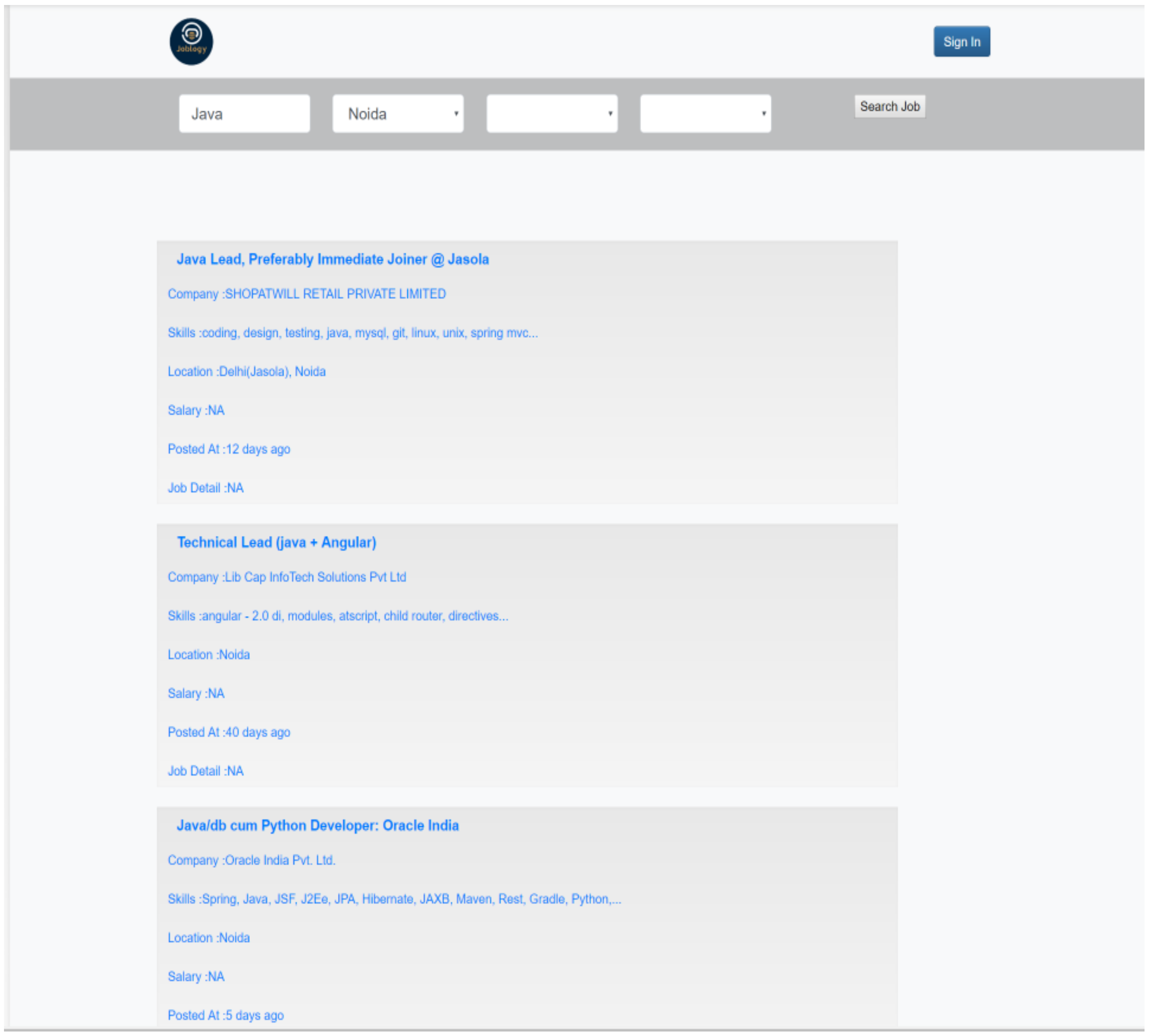
Fig. 8.16 SPARQL Queries

The illustrations as explained above in Fig. 8.14 , Fig. 8.15 and Fig. 8.16 shows the step by step process of converting keyword based queries into SPARQL queries. These queries are then fired over respective ontologies independently. The generated results are then merged by result merger and finally get available to the user.

8.6 IMPLEMENTATION OF THE PROPOSED WORK

To analyze the proposed work, various experiments have been conducted. The proposed approach has been implemented in Java Eclipse. For the implementation of the proposed system, keyword based query is taken from user interface which retrieves results from ‘n’ number of ontologies and displays the results at the same platform. The snapshot shown in

Fig 8.17 illustrates the results related to the user query "Java" as keyword and "Noida" as location.



Java Lead, Preferably Immediate Joiner @ Jasola
Company :SHOPATWILL RETAIL PRIVATE LIMITED
Skills :coding, design, testing, java, mysql, git, linux, unix, spring mvc...
Location :Delhi(Jasola), Noida
Salary :NA
Posted At :12 days ago
Job Detail :NA

Technical Lead (java + Angular)
Company :Lib Cap InfoTech Solutions Pvt Ltd
Skills :angular - 2.0 di, modules, atscript, child router, directives...
Location :Noida
Salary :NA
Posted At :40 days ago
Job Detail :NA

Java/db cum Python Developer: Oracle India
Company :Oracle India Pvt. Ltd.
Skills :Spring, Java, JSF, J2Ee, JPA, Hibernate, JAXB, Maven, Rest, Gradle, Python,...
Location :Noida
Salary :NA
Posted At :5 days ago

Fig. 8.17 Jobology Result Output

This approach is advantageous in comparison to existing Job boards in terms of

- Semantically enriched information
- Availability of Job posts belonging to different data sources at the same platform.

8.7 INTEGRATING JOBOLOGY SEARCH SYSTEM WITH STUDENT DOMAIN

Placement is a crucial interface between the stages of completion of academic program of the students and their entry into the suitable employment. The main Job responsibility of placement department in any institute is to arrange campus recruitment. The main challenge through which placement cell goes across is to arrange recruitment or providing Job information with respect to every student qualification and skillset.

This module works to meet the employment needs of current college student and recent graduates. The motive behind this module is to provide beginner's level Job opportunity to a student from a reservoir of Jobposts maintained by online Job boards without sifting through volumes of posts that require more experience, at one place. For this, student ontology is developed that has been explained in detail in chapter 4 section. The student ontology is integrated with JOBOLOGY by mapping concepts such as qualification, skillset etc. this cross-domain integration leads to many benefits which are discussed as follows:

1) From the placement cell perspective,
   - They will be able to provide more Job opportunities to their students.
   - They will be able to provide Job opportunity according to individual student skillset, qualification and other preferences such as location etc.
2) From the student perspective,
   - Student will be able to get Job opportunity from the various reservoirs of Jobposts maintained by their respective Jobboards at one place.
   - He/she will be able to get the list of Jobpost according to their preferences. For instance, if a student prefers only Delhi/NCR region Jobs then Jobology will provide only those Jobposts which satisfy their preferences.

8.8 PERFORMANCE EVALUATION OF THE PROPOSED SYSTEM

To evaluate the Proposed Jobology Search System, the architecture has been implemented in JDK 1.8 Eclipse framework.

The system has been evaluated in two phases:

Phase 1: Considering individual Jobboards

In this phase, existing Jobboards and their respective proposed ontologies are compared individually. For instance, a set of queries is given to a user and is asked to apply those queries on the naukri.com and give his feedback. Then, same set of query is applied on the proposed naukri ontology developed by the proposed system. The results are analyzed and with respect to that performance metric is calculated.

Phase 2: Considering the integrated system

In this phase, the performance of the overall system is calculated and compared with Jobboards and proposed individual ontologies. For this phase, users have been provided with 3 query sets and they were asked to apply those queries on the entire platform. Based on user feedback, a comparative analysis is done.

8.8.1 Evaluation for individual Jobboards

For analysis, a set of 18 queries belonging to three query sets, each of 6 queries was taken as a sample. For each query set, a group of 20 different users were selected for participation. Users were asked to apply these queries on three Jobboards i.e. *www.naukri.com, www.timesjob.com* and *www.shine.com (existing systems)* and proposed ontologies i.e. *Naukri ontology, Timesjob ontology* and *Shine ontology* respectively. Top 50 posts were considered as retrieved post and out of those, top 10 posts were used for making decision.

The performance of Jobology Search System has been measured on the basis of Precision metric, P which is defined as the fraction of retrieved Jobposts that are relevant to the query. Mathematically, the precision is calculated as:

$$Precision = \frac{|\{relevant\ posts\} \cap \{retrieved\ posts\}|}{\{retrieved\ posts\}} \tag{8.1}$$

For example, for a text search on a set of documents, precision is the number of correct results divided by the number of all returned results.
At first, a set of 6 queries were prepared for query set 1 and it was handed over to 20 users. The first user fired each given query on Naukri.com (existing system) and Naukri ontology (proposed system). On each query, the feedback of the user was taken in terms of “relevant” or “irrelevant” with respect to top 10 Jobbposts. In the same way, same user performed the same procedure on remaining two Jobboards and their respective proposed ontologies (i.e. *www.timejob.com, timesjob ontology and www.shine.com and shine ontology*). In this manner, the same set of queries was given to the second user following the same process

and so on. The same procedure was carried for the set of queries belonging to other query sets by choosing different twenty users.

A detailed discussion on the performance analysis of results given by Jobology for each set of queries is given in the following sections.

a) QUERY SET-1

The set consisted of 6 queries is shown in Table 8.10.

Table 8.10 Query Set 1

S.No	**Query**
1	Java,Delhi
2	PHP,Bangalore
3	Python,Chennai
4	CSS,Delhi
5	.net,Indore
6	AngularJS, Ahemdabad

The queries were analyzed using the performance metric, Precision (P), which is calculated using (8.1) as mentioned above by each of the twenty users and the response obtained thereof is given in Table 8.11 in terms of precision result.

The comparative analysis of Precision P and P' for the queries belonging to query set 1 with respect to Jobboards and their proposed ontologies is shown in Table 8.11.

Table 8.11 Comparative Analysis Between Existing and Proposed Systems

QS1	**Naukri.com**		**Timesjob.com**		**Shine.com**	
	P	**P'**	**P**	**P'**	**P**	**P'**
q1	0.60	0.85	0.54	0.81	0.41	0.82
q2	0.60	0.77	0.54	0.82	0.46	0.69
q3	0.65	0.79	0.51	0.67	0.39	0.71
q4	0.50	0.79	0.49	0.72	0.42	0.77
q5	0.52	0.70	0.43	0.84	0.41	0.69
q6	0.55	0.75	0.58	0.86	0.46	0.75
Average	**0.57**	**0.78**	**0.51**	**0.79**	**0.43**	**0.74**

Comparing Naukri.com with proposed Naukri Ontology, it can be observed that the precision *P* of query *q1* on the basis of user data came out to be 0.60 whereas precision *P'* has been found to be 0.85. Fig. 8.18 shows the comparative analysis of queries with respect to Jobboards and proposed ontologies.

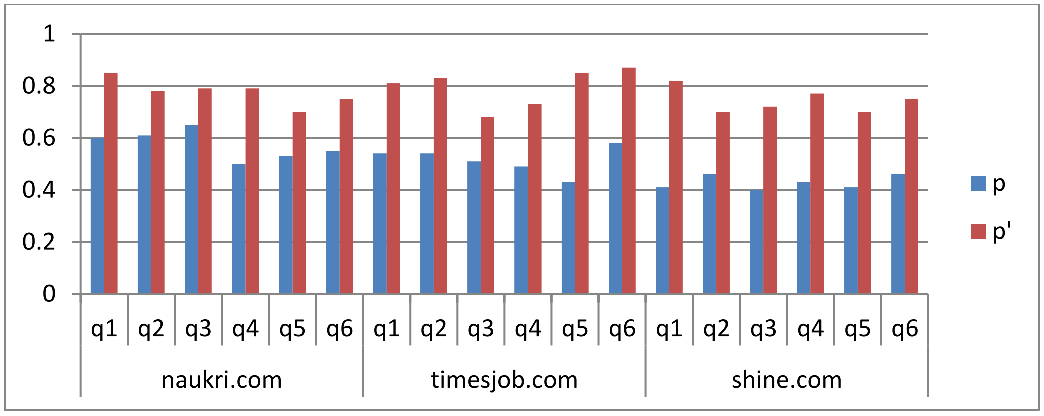


Fig. 8.18 Precision Analysis of Queries for Query Set1

Fig. 8.19 shows the average comparative analysis of queries of query set QS1 with respect to Jobboards and proposed ontologies.

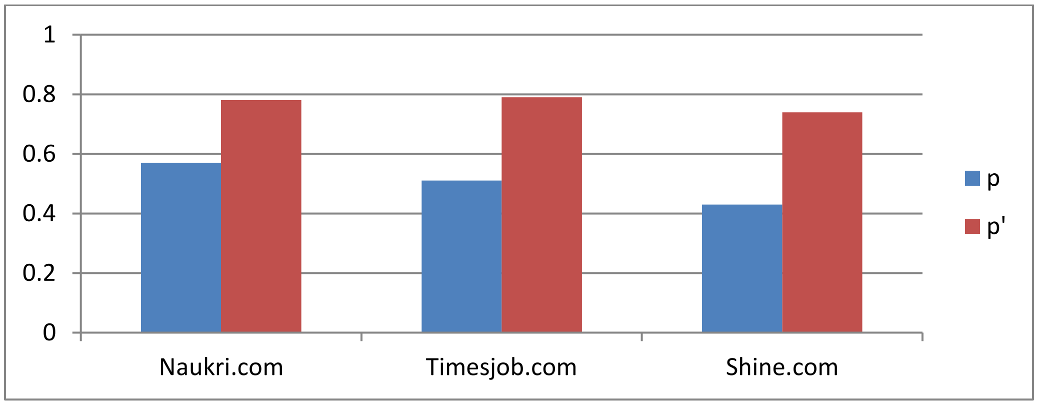


Fig. 8.19 Average Precision of Queries for Query Set1

The same process was performed for queries of Query Set 2 which is explained in the next subsection.

b) QUERY SET-2

The set consisting of 6 queries is shown in Table 8.12 .

Table 8.12 Query Set 2

S.No	Query
1	Oracle,Delhi
2	SAP,Gurugram
3	ADO,Dehradun
4	Core Java,Ahmednagar
5	Java,Struts, Bhopal
6	Java,Hibernate,Bhubeneshwar

In the same way, for this set of queries, response by each of the twenty users of different group was taken which is given shown in Table 8.13 in terms of Precision and its average.

The comparative analysis of precision for the queries belonging to query set 2 with respect to selected jobboards i.e. naukri.com, Timesjob.com and Shine.com and their corresponding proposed ontologies i.e. naukri ontology, timesjob ontology and shine ontology is shown in Table 8.13.

Table 8.13 Comparative Analysis Between Existing and Proposed Systems

QS2	**Naukri.com**		**Timesjob.com**		**Shine.com**	
	P	**P'**	**P**	**P'**	**P**	**P'**
q1	0.54	0.80	0.49	0.78	0.49	0.78
q2	0.53	0.76	0.51	0.75	0.51	0.82
q3	0.56	0.73	0.60	0.74	0.56	0.79
q4	0.62	0.84	0.67	0.74	0.46	0.71
q5	0.55	0.68	0.68	0.74	0.48	0.80
q6	0.45	0.80	0.60	0.78	0.47	0.78
Average	**0.54**	**0.77**	**0.59**	**0.75**	**0.49**	**0.78**

Comparing Naukri.com with Naukri Ontology, it can be observed that upon applying q1 on naukri.com, precision came out to be 0.54 whereas precision of Naukri ontology for q1 has been found to be 0.80. Fig. 8.20 shows the comparative analysis of queries with respect to Jobboards and proposed ontologies.

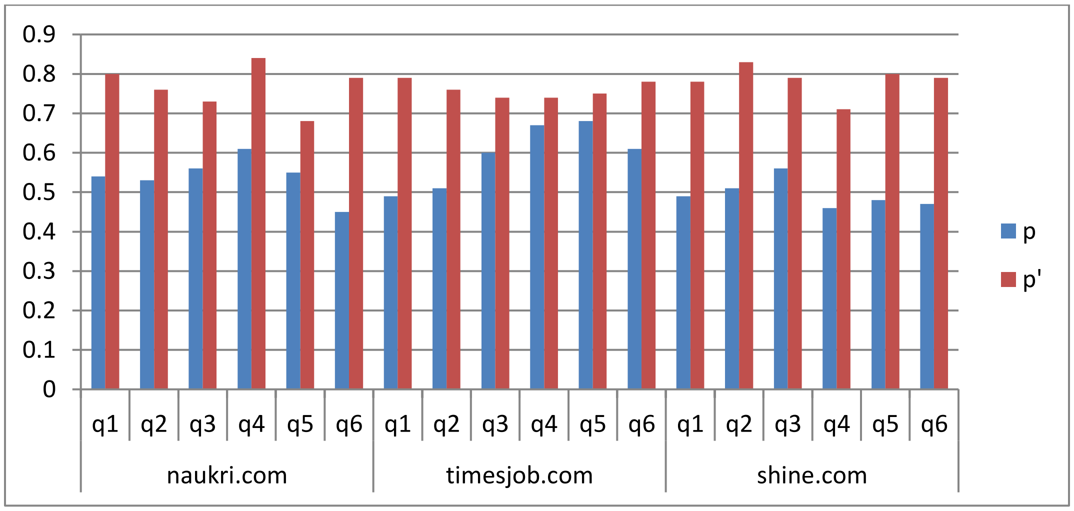


Fig. 8.20 Precision Analysis of Queries for Query Set2

Fig. 8.21 shows the average comparative analysis of queries of query set QS2 with respect to Jobboards and proposed ontologies.

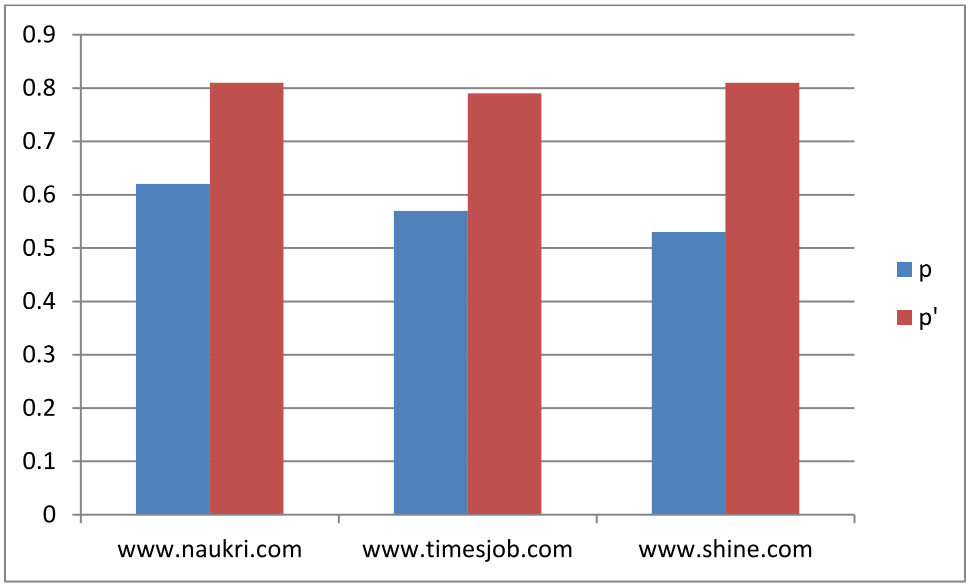


Fig. 8.21 Average Precision of Queries for Query Set2

c) QUERY SET 3

The set consisting of 6 queries is shown in Table 8.14

Table 8.14 Query Set 3

S.No	Query
1	Advanced Java, Mumbai
2	.net, Pune
3	HTML,Javascript,Delhi
4	NodeJS,Javs,Ahemdabad
5	Abndroid, Bangalore
6	Java,Spring, Kolkata

Similarly, for this set of queries of QS3, response by each of the twenty users of different group was taken which is given shown in Table 8.15 in terms of Precision and its average.

The comparative analysis of precision for the queries belonging to query set 3 with respect to selected jobboards i.e. naukri.com, Timesjob.com and Shine.com and their corresponding proposed ontologies i.e. naukri ontology, timesjob ontology and shine ontology is shown in Table 8.15.

Table 8.15 Comparative Analysis Between Existing and Proposed Systems

QS3	Naukri.com		Timesjob.com		Shine.com	
	P	**P'**	**P**	**P'**	**P**	**P'**
q1	0.60	0.85	0.61	0.78	0.44	0.77
q2	0.62	0.77	0.62	0.80	0.47	0.84
q3	0.65	0.80	0.67	0.82	0.52	0.78
q4	0.56	0.79	0.49	0.84	0.55	0.82
q5	0.63	0.83	0.55	0.77	0.62	0.84
q6	0.62	0.82	0.48	0.73	0.54	0.77

Comparing Naukri.com with Naukri Ontology, it can be observed that the precision of Naukri.com (Existing system) for query q1 on the basis of user data, it came out to be 0.6 whereas precision of Naukri ontology has been found to be 0.85. Fig. 8.22 shows the comparative analysis of queries with respect to Jobboards and proposed ontologies.

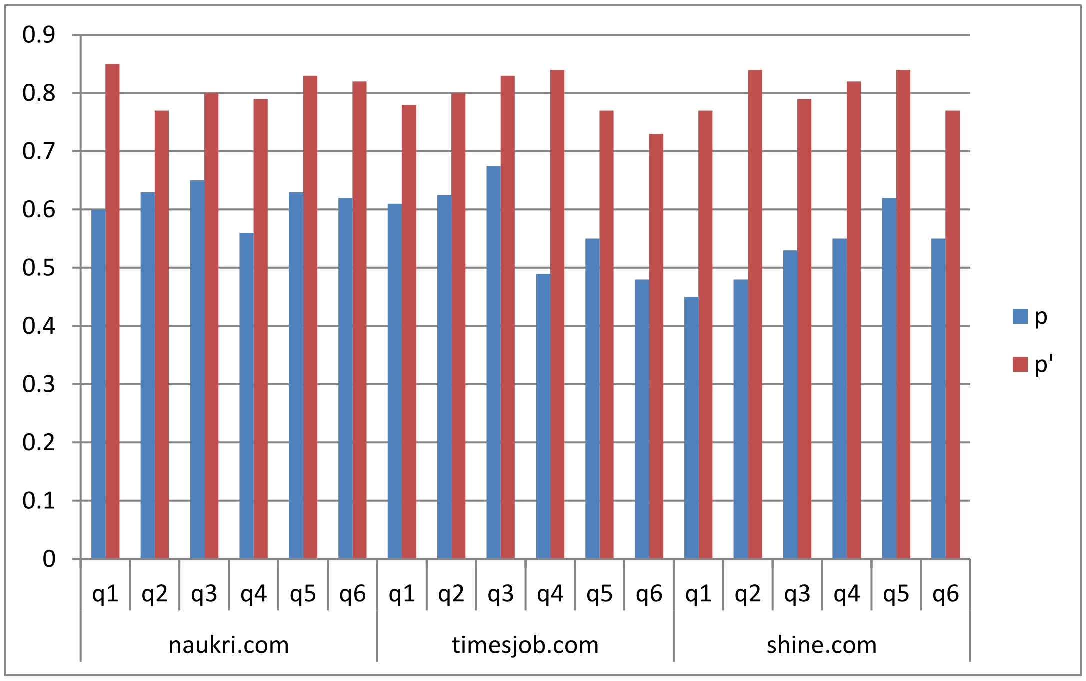


Fig. 8.22 Precision Analysis of Queries for Query Set 3

Fig. 8.23 shows the average comparative analysis of queries of query set QS3 with respect to Jobboards and proposed ontologies.

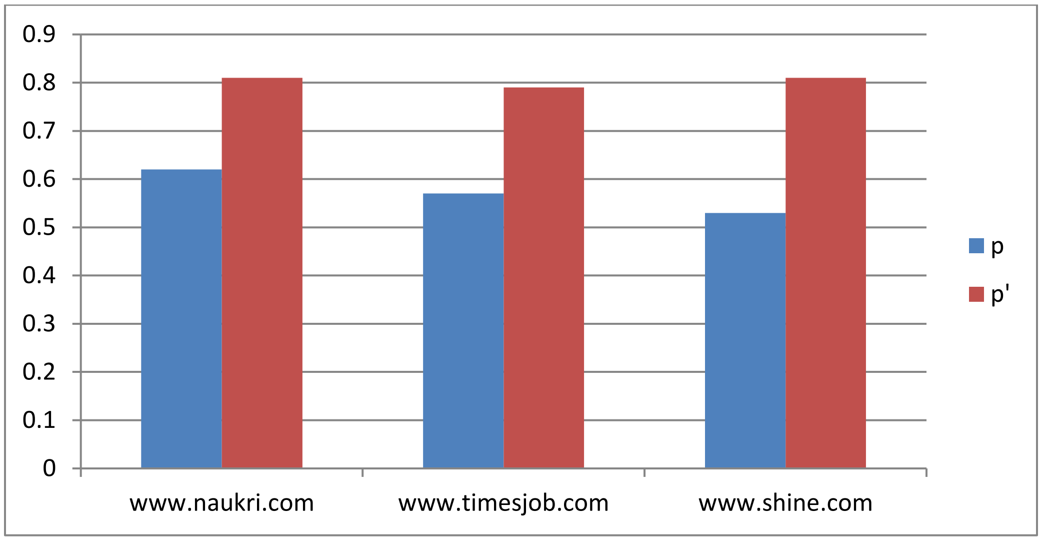


Fig. 8.23 Average Precision of Queries for Query Set3

It can be observed from the graphs that the plotted values of precision are higher for proposed system as compared to existing Joboards.

8.8.2 Evaluation at System Level

In this phase, comparison analysis between Jobboards, proposed ontologies and integrated system was performed. For analysis, same set of query sets were given to the 20 users. Top 50 posts were considered as retrieved post and out of those, top 10 posts were used for making decision. The analysis of precision P, P' and P'' for the queries belonging to query set QS1 is shown in Table 8.16, where p depicts the average precision with respect to query q1 from all the jobboard, p' depicts the average precision with respect to q1 from all the individual proposed ontologies and P'' depicts the average precision from the proposed integrated system i.e. Jobology search system.

Table 8.16 Average Precision with Respect to Queries of Query Set 1

QS1	P	P'	P''
Q1	0.52	0.83	0.79
Q2	0.54	0.77	0.81
Q3	0.52	0.73	0.78
Q4	0.47	0.76	0.84
Q5	0.46	0.75	0.82
Q6	0.53	0.79	0.82
Average	**0.51**	**0.77**	**0.81**

The graph shown in Fig. 8.24 shows the query accuracy for each query of query set QS1.

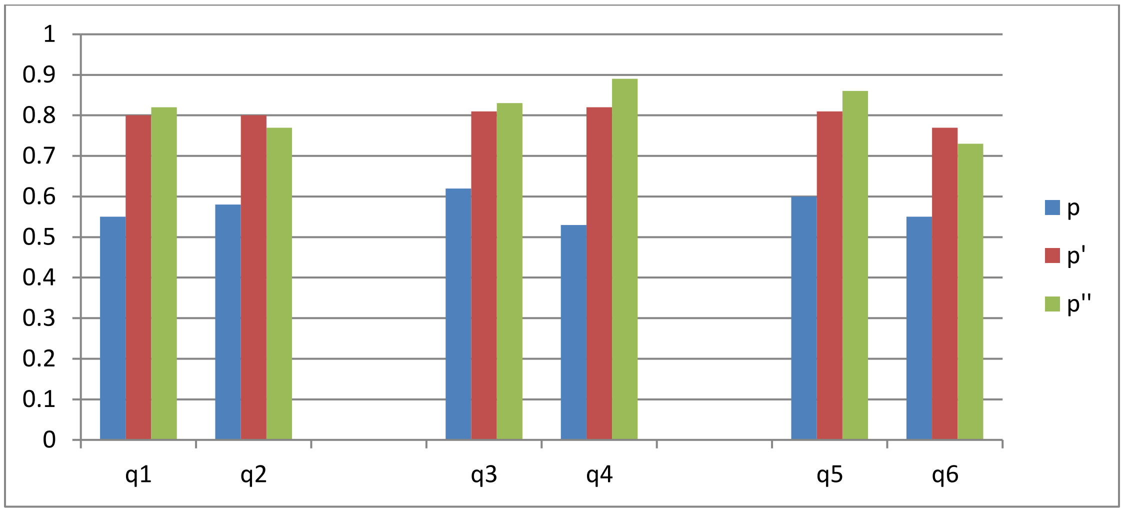


Fig. 8.24 Plotted Values of Precision of Query Set 1

In the same way, analysis was performed for Query Set QS2 which is shown in Table 8.17.

Table 8.17 Average Precision with Respect to Queries of Query Set 2

QS2	P	P'	P''
Q1	0.51	0.79	0.81
Q2	0.52	0.78	0.77
Q3	0.57	0.75	0.93
Q4	0.58	0.76	0.82
Q5	0.57	0.74	0.82
Q6	0.51	0.79	0.74
Average	**0.54**	**0.76**	**0.81**

The graph shown in Fig. 8.25 shows the query accuracy for each query of query set QS2.

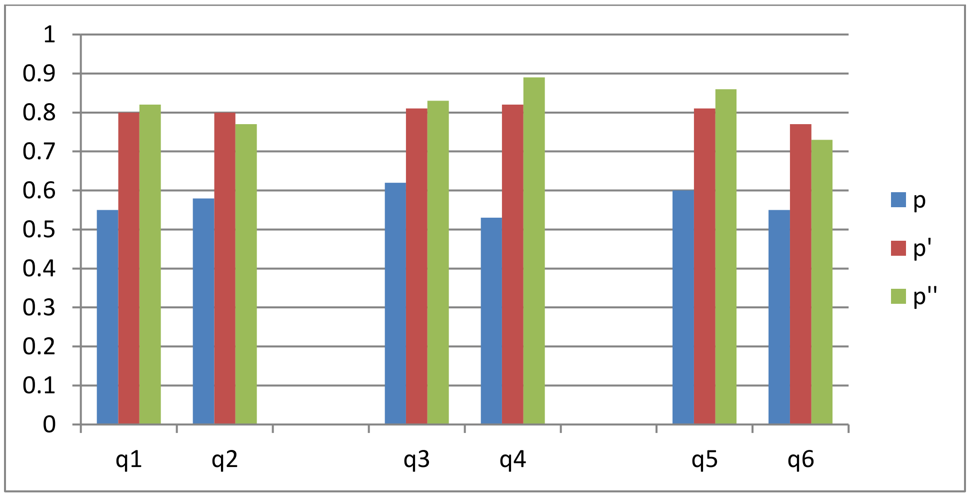


Fig. 8.25 Plotted Values of Precision of Query Set 2

Similarly, the analysis of precision P, P' and P'' for the queries belonging to query set QS3 is shown in Table 8.18.

Table 8.18 Average Precision with Respect to Queries of Query Set 3

QS3	P	P'	P''
Q1	0.51	0.79	0.81
Q2	0.52	0.78	0.77
Q3	0.57	0.75	0.93
Q4	0.58	0.76	0.82
Q5	0.57	0.74	0.82
Q6	0.51	0.79	0.74
Average	**0.54**	**0.76**	**0.81**

The graph shown in Fig. 8.26 shows the query accuracy for each query of query set QS3.

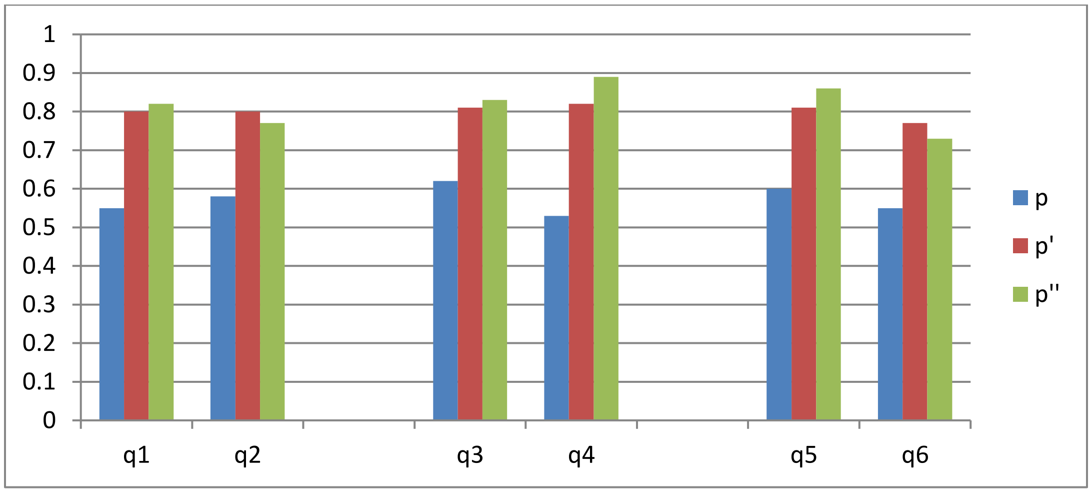


Fig. 8.26 Plotted values of Precision of Query Set 3

The average precision graph at system level is shown in Fig. 8.27

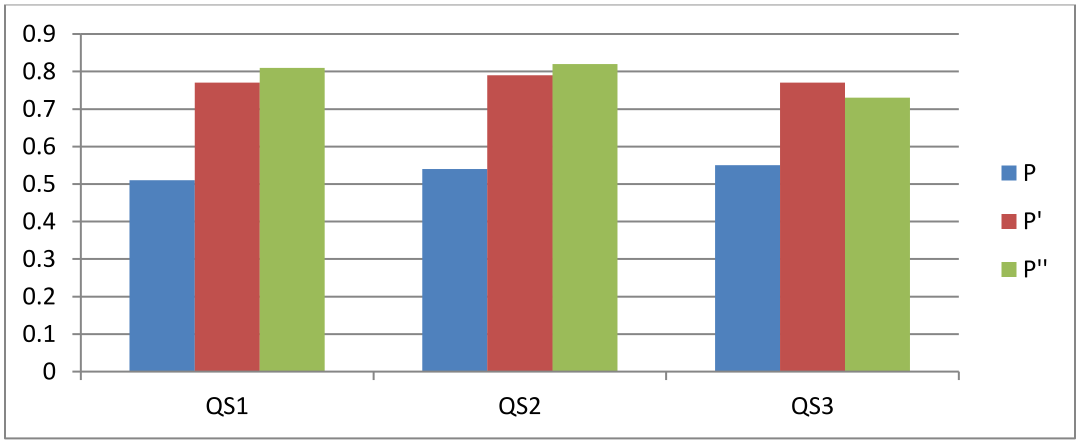


Fig. 8.27 Plotted Values of Average Precision of Query Sets

It can be observed from Fig. 8.27 that the proposed system gives more relevant results as it exhibits high precision in comparison with jobboards and individual jobboard's ontologies. For QS3, the plotted values are comparatively low.

8.9 SUMMARY

In this chapter, a novel method for query processing is proposed which supports querying on aligned ontologies by

a) Transforming keyword based query into SPARQL query
b) Generating separate SPARQL queries with respect to each ontology
c) Retrieving relevant results from the ontology annotated data sources.
d) Merging the results retrieved from multiple data sources and presenting to the user in such a way that Job posts containing maximum keywords given by user in query are presented at the top.
e) With this, the benefits of integration of JOBOLOGY with student domain are also discussed.

The next chapter is devoted to conclusion and future work.

Chapter IX

CONCLUSION AND FUTURE WORK

9.1 CONCLUSION

Semantic Web [13, 14, 15, 16, 17] has a set of technologies that not only provides the data but also the meaning associated with those data, thus making computers to understand the meaning of that data. It describes the relationship that exists between data. This allows the development of new applications such as search engines that can answer more complex answers. But, it does not mean that semi-structured data [24] or unstructured data that is available on the current web has lost its importance. However, looking at the benefits of semantic web technologies [15], there is a need to develop a system which can transform semi-structured/ unstructured data from current web into structured format and make compatible with semantic web tools.

With this regard, Jobology Search System has been developed which presents search results with respect to user query based on the concepts represented in them by ontologies. Ontologies have been developed with respect to Jobboards and Student domain. The architecture of OntoJobextractor has been developed which extracts semi-structured data from Jobboards and enriches extracted data semantically by annotating using ontologies. Architecture for ontology alignment has been proposed to index the concepts, object properties and data properties of ontologies. A framework "ONTOJOB" query processor has been proposed which processes user's keyword based query by converting it into SPARQL format query. It creates SPARQL query with respect to ontologies, merges the retrieved results and presents the results at one place. The proposed system is also integrated with student domain to provide the recruitment services to college students.

Following are the milestones that are achieved in this thesis:

1) **Development of Ontology**

   Ontologies were developed using protégé development framework by following the complete steps of ontology development lifecycle. Two main categories of ontologies were developed: one with respect to selected Jobboards and other in the domain of student.

2) **Framework for Data Extraction system**

   A framework for extracting semi-structured data from Jobboards has been proposed to enrich them semantically by annotating them using ontology and represent them in structured format.

3) **Framework for Ontology Alignment System**

   A framework for ontology alignment has been proposed that aligns ontologies by maintaining Global Concept Index, Global Object data property Index and Global Data Property Index. These indexing play a vital role in building SPARQL query.

4) **Framework for Query Processing System**

   A framework for query processing mechanism i.e. ONTOJOB query processor has been proposed to build SPARQL query from keyword based query.

5) **Framework for Search System**

   A framework Jobology has been proposed to present the results from various Jobboards at one place using ontology.

6) **Cross Domain Integration**

   A mechanism has been proposed that integrates Jobology with student domain to provide the recruitment services to college students.

Domain specific and cross domain integration approach has been followed to develop the system. Ontology based data representation, alignment and query processing has been done for developing the system. The system has been implemented in Java using Eclipse framework for project development. OWLAPI was used for building the ontologies in Java framework. For storing indexes and various datasets, MYSQL server was used.

To ensure the practical implications, the developed system, Jobology supports the following features:

1) **Scalability**

   The feature of addition of more concepts in ontologies to enhance the vocabulary of Jobboards supports the feature of scalability. More Jobboards can be added without affecting the existing system.

2) **Relevancy**

   The results are more relevant to fulfill the user requirement.

3) **Robustness**

   Ontologies are so designed that ontology revision will not change the foundedness of the resources that commit to an earlier version of the ontology.

4) **Improved Evaluation**

   It has been observed that the performance of the proposed system is fairly high as compared to existing systems due to addition of semantics using ontology.

The next section discusses the future scope of the proposed work.

9.2 FUTURE SCOPE

In this thesis, a search engine "Jobology" has been designed and implemented that includes data extraction, alignment and query processing on ontology annotated data. Some of the possible extensions that can be done in the future in this area are as follows:

1) **Automatic Development of Ontologies**

   To add any Jobboard in the Jobology system, first its ontology is created. This ontology has been created manually after analyzing the structure of the Jobboard. The research work can be extended to generate the ontology automatically.

2) **Extensibility**

   The proposed system can be applied to other domains such as travel domain, hotel domain with little modification with respect to the domains i.e. generalization of the system can be carried out.

3) Working with unstructured data

The proposed system is focusing on converting semi-structured data available on the current web into structured data. The data coverage of proposed system can be expanded by taking unstructured text data into consideration and converting it into structured data using various available tools such as GATE [168] tool.

4) Data coverage

The proposed system is availing data from Jobboard sites. Many companies and other organizations also publish jobs at their own websites. Therefore, to expand the coverage area, company's website job post webpages should be extracted to provide more job post to the users.

REFERENCES

[1] Berners-Lee, Tim, and Mark Fischetti. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. DIANE Publishing Company, 2001.

[2] Leiner, Barry M., Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. "A brief history of the Internet." *ACM SIGCOMM Computer Communication Review* 39, Vol. 5, 2009: 22-31.

[3] Dorogovtsev, Sergei N., and José FF Mendes. *Evolution of networks: From biological nets to the Internet and WWW*. OUP Oxford, 2013.

[4] Fuchs, Christian, Wolfgang Hofkirchner, Matthias Schafranek, Celina Raffl, Marisol Sandoval, and Robert Bichler. "Theoretical foundations of the web: cognition, communication, and co-operation. Towards an understanding of Web 1.0, 2.0, 3.0." *Future Internet* 2, Vol. 1, 2010: 41-59.

[5] Cormode, Graham, and Balachander Krishnamurthy. "Key differences between Web 1.0 and Web 2.0." *First Monday* 13, Vol. 6, 2008.

[6] Kosala, Raymond, and Hendrik Blockeel. "Web mining research: A survey." *ACM Sigkdd Explorations Newsletter* 2, Vol. 1, 2000: 1-15.

[7] Tan, Pang-Ning. *Introduction to data mining*. Pearson Education India, 2007.

[8] Cooley, Robert, Bamshad Mobasher, and Jaideep Srivastava. "Web mining: Information and pattern discovery on the world wide web." In *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*, 1997: 558-567.

[9] Levene, Mark. *An introduction to search engines and web navigation*. John Wiley & Sons, 2011.

[10] Brin, Sergey, and Lawrence Page. "The anatomy of a large-scale hypertextual web search engine." *Computer networks and ISDN systems* 30, Vol. 1-7, 1998: 107-117.

[11] Boulton, David, and Martyn Hammersley. "Analysis of unstructured data." *Data collection and analysis*,1996:282-297.

[12] Metzler, Donald, Susan Dumais, and Christopher Meek. "Similarity measures for short segments of text." In *European conference on information retrieval*, Springer, Berlin, Heidelberg, 2007: 16-27.

[13] Berners-Lee, Tim. "Semantic web road map." 1998.

[14] Berners-Lee, Tim, James Hendler, and Ora Lassila. "The semantic web." *Scientific american*, Vol. 5, 2001: 34-43.

[15] Wang, Xiaoshu, Robert Gorlitsky, and Jonas S. Almeida. "From XML to RDF: how semantic web technologies will change the design of'omic'standards." *Nature biotechnology*, Vol. 9, 2005: 1099.

[16] Decker, Stefan, Sergey Melnik, Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks. "The semantic web: The roles of XML and RDF." *IEEE Internet computing* 4, Vol. 5 ,2000: 63-73.

[17] Quilitz, Bastian, and Ulf Leser. "Querying distributed RDF data sources with SPARQL." In *European Semantic Web Conference*: 524-538. Springer, Berlin, Heidelberg, 2008.

[18] Hartig, Olaf, Christian Bizer, and Johann-Christoph Freytag. "Executing SPARQL queries over the web of linked data." In *International Semantic Web Conference*: 293-309. Springer, Berlin, Heidelberg, 2009.

[19] McGuinness, Deborah L., and Frank Van Harmelen. "OWL web ontology language overview." *W3C recommendation* 10, Vol. 10 ,2004: 2004.

[20] Knublauch, Holger, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. "The Protégé OWL plugin: An open development environment for semantic web applications." In *International Semantic Web Conference*: 229-243. Springer, Berlin, Heidelberg, 2004.

[21] Van Harmelen, Frank, Vladimir Lifschitz, and Bruce Porter, eds. *Handbook of knowledge representation*. Vol. 1. Elsevier, 2008.

[22] Chung, Christina Yip, and Neelakantan Sundaresan. "Majority schema in semi-structured data." U.S. Patent 6,604,099, issued August 5, 2003.

[23] Davies, John, Dieter Fensel, and Frank Van Harmelen, eds. *Towards the semantic web: ontology-driven knowledge management*. John Wiley & Sons, 2003.

[24] Fonseca, Frederico T., Max J. Egenhofer, Peggy Agouris, and Gilberto Câmara. "Using ontologies for integrated geographic information systems." *Transactions in GIS* 6, Vol. 3 ,2002: 231-257.

[25] Salus, Peter H., and G. Vinton. *Casting the Net: From ARPANET to Internet and Beyond...* Addison-Wesley Longman Publishing Co., Inc., 1995.

[26] Forouzan, Behrouz A., and Sophia Chung Fegan. *TCP/IP protocol suite*. McGraw-Hill Higher Education, 2002.

[27] Abdelnur, Alejandro, Abhay Gupta, and Brent Callaghan. "Resources sharing on the internet via the HTTP." U.S. Patent 6,212,640, issued April 3, 2001.

[28] Powell, Thomas A. *HTML: the complete reference*. McGraw-Hill Professional, 2002.

[29] Judson, David H. "Web browser with dynamic display of information objects during linking." U.S. Patent 5,572,643, issued November 5, 1996.

[30] McBryan, Oliver A. "GENVL and WWWW: Tools for taming the web." In *Proceedings of the first international world wide web conference*, vol. 341. 1994.

[31] Alexander, Bryzan. "Web 2.0." *A New Wave of Innovation for Teachning and learning* ,2006: 32-44.

[32] Liaw, Shu-Sheng, and Hsiu-Mei Huang. "An investigation of user attitudes toward search engines as an information retrieval tool." *Computers in human behavior* 19, Vol. 6 ,2003: 751-765.

[33] Curtis, John Andrew, and Gordon Frank Scherer. "Search engine using indexing method for storing and retrieving data." U.S. Patent 6,278,992, issued August 21, 2001.

[34] Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd. *The PageRank citation ranking: Bringing order to the web*. Stanford InfoLab, 1999.

[35] Kobayashi, Mei, and Koichi Takeda. "Information retrieval on the web." *ACM Computing Surveys , (CSUR* 32), Vol. 2 ,2000: 144-173.

[36] Baeza-Yates, Ricardo, and Berthier de Araújo Neto Ribeiro. *Modern information retrieval*. New York: ACM Press; Harlow, England: Addison-Wesley,, 2011.

[37] Mitev, Nathalie N., Gillian M. Venner, and Stephen Walker. *Designing an online public access catalogue: Okapi, a catalogue on a local area network*. The British Library, 1985.

[38] Glover, Eric J., Steve Lawrence, William P. Birmingham, and C. Lee Giles. "Architecture of a metasearch engine that supports user information needs." In *Proceedings of the eighth international conference on Information and knowledge management*: 210-216. ACM, 1999.

[39] Jain, Ranjna & Duhan, Neelam & Sharma, Ashok. ,2015. Comparative Study on Semantic Search Engines. International Journal of Computer Applications. 131. 4-11. 10.5120/ijca2015907370.

[40] Miller, George A., Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. "Introduction to WordNet: An on-line lexical database." *International journal of lexicography* 3, Vol. 4 ,1990: 235-244.

[41] Gerber, Aurona, Alta Van der Merwe, and Andries Barnard. "A functional semantic web architecture." In *European Semantic Web Conference*: 273-287. Springer, Berlin, Heidelberg, 2008.

[42] Berners-Lee, Tim. "Semantic Web: Why RDF is more than XML." ,1998.

[43] Brickley, Dan. "RDF vocabulary description language 1.0: RDF schema." *http://www. w3. org/TR/rdf-schema/* ,2004.

[44] Gruber, Thomas R. "Toward principles for the design of ontologies used for knowledge sharing?." *International journal of human-computer studies* 43, Vol. 5-6 ,1995: 907-928.

[45] McIlraith, Sheila A., Tran Cao Son, and Honglei Zeng. "Semantic web services." *IEEE intelligent systems* 16, Vol. 2 ,2001: 46-53.

[46] Horrocks, Ian. "DAML+OIL, A Description Logic for the Semantic Web." *IEEE Data Eng. Bull.* 25, Vol. 1 ,2002: 4-9.

[47] Gómez-Pérez, Asunción, and Oscar Corcho. "Ontology languages for the semantic web." *IEEE Intelligent systems* 17, Vol. 1 ,2002: 54-60.

[48] Fensel, Dieter, Frank Van Harmelen, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. "OIL: An ontology infrastructure for the semantic web." *IEEE intelligent systems*16, Vol. 2 ,2001: 38-45.

[49] Gerber, Aurona J., Andries Barnard, and Alta J. Van der Merwe. "Towards a semantic web layered architecture." ,2007.

[50] Guarino, Nicola, Daniel Oberle, and Steffen Staab. "What is an ontology?." In *Handbook on ontologies*: 1-17. Springer, Berlin, Heidelberg, 2009.

[51] Gruber, Tom. "What is an Ontology." *WWW Site http://www-ksl. stanford. edu/kst/whatis-an-ontology. html* ,1993.

[52] Fensel, Dieter. "Ontologies." In *Ontologies*: 11-18. Springer, Berlin, Heidelberg, 2001.

[53] Russell, Stuart J., and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[54] Swartout, Bill, Ramesh Patil, Kevin Knight, and Tom Russ. "Toward distributed use of large-scale ontologies." In *Proc. of the Tenth Workshop on Knowledge Acquisition for Knowledge-Based Systems*: 138-148. 1996.

[55] Noy, Natalya F., and Deborah L. McGuinness. "Ontology development 101: A guide to creating your first ontology." ,2001.

[56] Fonseca, Frederico T., Max J. Egenhofer, Peggy Agouris, and Gilberto Câmara. "Using ontologies for integrated geographic information systems." *Transactions in GIS* 6, Vol. 3 ,2002: 231-257.

[57] Starlab 2003. Systems Technology and Applications Research Laboratory home page. Faculty of Sciences, Department of Computer Science, Vrije Universiteit Brussel.

[58] Farquhar, Adam, Richard Fikes, and James Rice. "The ontolingua server: A tool for collaborative ontology construction." *International journal of human-computer studies*46, Vol. 6 ,1997: 707-727.

[59] Fensel, Dieter, Ora Lassila, Frank Van Harmelen, Ian Horrocks, James Hendler, and Deborah L. McGuinness. "The semantic web and its languages." *IEEE Intelligent Systems and their Applications* 15 ,2000: 67-73.

[60] Eardley, Alan, ed. *Innovative Knowledge Management: Concepts for Organizational Creativity and Collaborative Design: Concepts for Organizational Creativity and Collaborative Design*. IGI Global, 2010.

[61] Guarino, Nicola. "Some organizing principles for a unified top-level ontology." In *AAAI Spring Symposium on Ontological Engineering*: 57-63. AAAI Press Menlo Park, 1997.

[62] Gómez-Pérez, Asunción, Mariano Fernández, and A. de Vicente. "Towards a method to conceptualize domain ontologies." ,1996.

[63] Shaw, Marianne, Landon T. Detwiler, James F. Brinkley, and Dan Suciu. "Generating application ontologies from reference ontologies." In *AMIA Annual Symposium Proceedings*, vol. 2008, p. 672. American Medical Informatics Association, 2008.

[64] Genesereth, Michael R., and Richard E. Fikes. "Knowledge interchange format-version 3.0: reference manual." ,1992.

[65] Abburu, Sunitha. "A survey on ontology reasoners and comparison." *International Journal of Computer Applications*57, Vol. 17 ,2012.

[66] MacGregor, Robert, and Raymond Bates. *The Loom Knowledge Representation Language*. Vol. ISI/RS-87-188. UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST, 1987.

[67] Motta, Enrico. "An overview of the OCML modelling language." In *the 8th Workshop on Methods and Languages*. 1998.

[68] Kifer, Michael, and Georg Lausen. "F-logic: a higher-order language for reasoning about objects, inheritance, and scheme." In *ACM SIGMOD Record*, vol. 18, Vol. 2: 134-146. ACM, 1989.

[69] Decker, Stefan, Michael Erdmann, Dieter Fensel, and Rudi Studer. "Ontobroker: Ontology based access to distributed and semi-structured information." In *Database Semantics*: 351-369. Springer, Boston, MA, 1999.

[70] Heflin, Jeff, James Hendler, and Sean Luke. *SHOE: A knowledge representation language for internet applications*. 1999.

[71] Firesmith, Donald, Brian Henderson-Sellers, and Ian Graham. *OPEN modeling language ,OML reference manual*. (CUP Archive), 1998.

[72] Corcho, Oscar, and Asunción Gómez-Pérez. "A roadmap to ontology specification languages." In *International Conference on Knowledge Engineering and Knowledge Management*: 80-96. Springer, Berlin, Heidelberg, 2000.

[73] Allemang, Dean, and James Hendler. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011.

[74] Adida, Ben, Mark Birbeck, Shane McCarron, and Steven Pemberton. "RDFa in XHTML: Syntax and processing." *Recommendation, W3C* 7 ,2008.

[75] Musciano, Chuck, and Bill Kennedy. *HTML & XHTML: The Definitive Guide: The Definitive Guide*. " O'Reilly Media, Inc.", 2002.

[76] Weibel, Stuart, John Kunze, Carl Lagoze, and Misha Wolf. *Dublin core metadata for resource discovery*. Vol. RFC 2413. 1998.

[77] Fensel, Dieter, Frank van Harmelen, and Ian Horrocks. "OIL: A standard proposal for the Semantic Web." *On-To-Knowledge deliverable D-0, Vrije Universiteit Amsterdam*,1999.

[78] Ankolekar, Anupriya, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David Martin, Drew McDermott, Sheila A. McIlraith et al. "DAML-S: Web service description for the semantic web." In *International Semantic Web Conference*: 348-363. Springer, Berlin, Heidelberg, 2002.

[79] Bechhofer, Sean, Ian Horrocks, Carole Goble, and Robert Stevens. "OilEd: a reason-able ontology editor for the semantic web." In *Annual Conference on Artificial Intelligence*: 396-408. Springer, Berlin, Heidelberg, 2001.

[80] Sure, York, Michael Erdmann, Jürgen Angele, Steffen Staab, Rudi Studer, and Dirk Wenke. "OntoEdit: Collaborative ontology development for the semantic web." In *International Semantic Web Conference*: 221-235. Springer, Berlin, Heidelberg, 2002.

[81] Noy, Natalya Fridman, Ray W. Fergerson, and Mark A. Musen. "The knowledge model of Protege-2000: Combining interoperability and flexibility." In *International Conference on Knowledge Engineering and Knowledge Management*: 17-32. Springer, Berlin, Heidelberg, 2000.

[82] Arpírez, Julio C., Oscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. "WebODE: a scalable workbench for ontological engineering." In *Proceedings of the 1st international conference on Knowledge capture*: 6-13. ACM, 2001.

[83] Bechhofer, Sean, Frank Van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. "OWL web ontology language reference." *W3C recommendation* 10, Vol. 02 ,2004.

[84] Abburu, Sunitha. "A survey on ontology reasoners and comparison." *International Journal of Computer Applications*57, Vol. 17 ,2012.

[85] Haase, Peter, Frank Van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure. "A framework for handling inconsistency in changing ontologies." In *International semantic web conference*: 353-367. Springer, Berlin, Heidelberg, 2005.

[86] Ming, DENG Zhihong TANG Shiwei ZHANG, and YANG Dongqing CHEN Jie. "Overview of Ontology [J]." *Acta Scicentiarum Naturalum Universitis Pekinesis* 5 ,2002: 027.

[87] Tsarkov, Dmitry, and Ian Horrocks. "FaCT++ description logic reasoner: System description." In *International Joint Conference on Automated Reasoning*: 292-297. Springer, Berlin, Heidelberg, 2006.

[88] Shearer, Rob, Boris Motik, and Ian Horrocks. "HermiT: A Highly-Efficient OWL Reasoner." In *OWLED*, vol. 432, p. 91. 2008.

[89] del Mar Roldan-Garcia, Maria, and Jose F. Aldana-Montes. "DBOWL: Towards a Scalable and Persistent OWL reasoner." In *Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on*: 174-179. IEEE, 2008.

[90] I. PalmisaVol. JFact repository, 2015.

[91] Calvanese, Diego, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. "Ontop: Answering SPARQL queries over relational databases." *Semantic Web* 8, Vol. 3 ,2017: 471-487.

[92] Sirin, Evren, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. "Pellet: A practical owl-dl reasoner." *Web Semantics: science, services and agents on the World Wide Web* 5, Vol. 2 ,2007: 51-53.

[93] Parsia, Bijan, and Evren Sirin. "Pellet: An owl dl reasoner." In *Third international semantic web conference-poster*, vol. 18, p. 2. Publishing, 2004.

[94] Haarslev, Volker, and Ralf Möller. "RACER system description." In *International Joint Conference on Automated Reasoning*: 701-705. Springer, Berlin, Heidelberg, 2001.

[95] Haarslev, Volker, and Ralf Möller. "Description of the RACER System and its Applications." *Description Logics* 49 ,2001.

[96] Chaudhri, Vinay K., Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. "OKBC: A programmatic foundation for knowledge base interoperability." In *AAAI/IAAI*: 600-607. 1998.

[97] Giarratano, C. "CLIPS: C language integrated production system." *CLIPS users guide-version* 6 ,1993.

[98] Domingue, John, Enrico Motta, and O. Corcho Garcia. "Knowledge modelling in webonto and ocml: A user guide." ,1999.

[99] Kalyanpur, Aditya, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. "Swoop: A web ontology editing browser." *Web Semantics: Science, Services and Agents on the World Wide Web* 4, Vol. 2 ,2006: 144-153.

[100] COMPOSER, TOPBRAID. "TopBraid Composer 2007 features and getting started guide version 1.0, created by TopQuadrant." *US. US* ,2007.

[101] Vogel, Lars. "Eclipse IDE tutorial." ,2014.

[102] McBride, Brian. "Jena: Implementing the rdf model and syntax specification." In *Proceedings of the Second International Conference on Semantic Web-Volume 40*: 23-28. CEUR-WS. org, 2001.

[103] Golbreich, Christine. "Combining rule and ontology reasoners for the semantic web." In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*: 6-22. Springer, Berlin, Heidelberg, 2004.

[104] Horrocks, Ian, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. "SWRL: A semantic web rule language combining OWL and RuleML." *W3C Member submission* 21 ,2004: 79.

[105] O'connor, Martin, Holger Knublauch, Samson Tu, Benjamin Grosof, Mike Dean, William Grosso, and Mark Musen. "Supporting rule system interoperability on the semantic web with SWRL." In *International Semantic Web Conference*: 974-986. Springer, Berlin, Heidelberg, 2005.

[106] Boley, Harold, Said Tabet, and Gerd Wagner. "Design rationale of RuleML: A markup language for semantic web rules." In *Proceedings of the First International Conference on Semantic Web Working*: 381-401. CEUR-WS. org, 2001.

[107] Bailey, James, François Bry, Tim Furche, and Sebastian Schaffert. "Web and semantic web query languages: A survey." In *Proceedings of the First international conference on Reasoning Web*: 35-133. Springer-Verlag, 2005.

[108] Miller, Libby, Andy Seaborne, and Alberto Reggiori. "Three implementations of SquishQL, a simple RDF query language." In *International Semantic Web Conference*: 423-435. Springer, Berlin, Heidelberg, 2002.

[109] Karvounarakis, Gregory, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. "RQL: a declarative query language for RDF." In *Proceedings of the 11th international conference on World Wide Web*: 592-603. ACM, 2002.

[110] Broekstra, Jeen, and Arjohn Kampman. "SeRQL: a second generation RDF query language." In *Proc. SWAD-Europe Workshop on Semantic Web Storage and Retrieval*: 13-14, 2003.

[111] Quilitz, Bastian, and Ulf Leser. "Querying distributed RDF data sources with SPARQL." In *European Semantic Web Conference*: 524-538. Springer, Berlin, Heidelberg, 2008.

[112] Harris, Steve, Andy Seaborne, and Eric Prud'hommeaux. "SPARQL 1.1 query language." *W3C recommendation* 21, Vol. 10 ,2013.

[113] Al-Mukhtar, Mumtaz M. Ali, and Ahmed T. Abbass Al-Assafy. "The Implementation of FOAF Ontology for an Academic Social Network." *International Journal of Computer Science Engineering & Technology*, Vol. 4 ,2014.

[114] Sirin, Evren, and Bijan Parsia. "SPARQL-DL: SPARQL Query for OWL-DL." In *OWLED*, Vol. 258. 2007.

[115] O'Connor, Martin J., and Amar K. Das. "SQWRL: A Query Language for OWL." In *OWLED*, Vol. 529. 2009.

[116] O'Connor, Martin J., and Amar K. Das. "A method for representing and querying temporal information in OWL." In *International Joint Conference on Biomedical*

Engineering Systems and Technologies: 97-110. Springer, Berlin, Heidelberg, 2010.

[117] McBride, Brian. "Jena: A semantic web toolkit." *IEEE Internet computing* 6, Vol. 6 ,2002: 55-59.

[118] Carroll, Jeremy J., Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. "Jena: implementing the semantic web recommendations." In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*: 74-83. ACM, 2004.

[119] Knublauch, Holger, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. "The Protégé OWL plugin: An open development environment for semantic web applications." In *International Semantic Web Conference*: 229-243. Springer, Berlin, Heidelberg, 2004.

[120] Gennari, John H., Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu. "The evolution of Protégé: an environment for knowledge-based systems development." *International Journal of Human-computer studies* 58, Vol. 1 ,2003: 89-123.

[121] Horridge, Matthew, and Sean Bechhofer. "The owl api: A java api for owl ontologies." *Semantic Web* 2, Vol. 1 ,2011: 11-21.

[122] Ding, Li, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. "Swoogle: a search and metadata engine for the semantic web." In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*: 652-659. ACM, 2004.

[123] Finin, Tim, Li Ding, Rong Pan, Anupam Joshi, Pranam Kolari, Akshay Java, and Yun Peng. "Swoogle: Searching for knowledge on the Semantic Web." *AAAI 05 , intelligent systems demo* ,2005.

[124] Cheng, Gong, Weiyi Ge, and Yuzhong Qu. "Falcons: searching and browsing entities on the semantic web." In *Proceedings of the 17th international conference on World Wide Web*: 1101-1102. ACM, 2008.

[125] Cheng, Gong, and Yuzhong Qu. "Searching linked objects with falcons: Approach, implementation and evaluation." *International Journal on Semantic Web and Information Systems , (IJSWIS*), Vol. 3 ,2009: 49-70.

[126] Team, Hakia. "hakia Semantic Search Technology making sense of the worlds information." *White paper Jan* ,2010.

[127] Käfer, Tobias, Jürgen Umbrich, Aidan Hogan, and Axel Polleres. "Towards a dynamic linked data observatory." *LDOW at WWW* ,2012.

[128] Hogan, Aidan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. "Searching and browsing linked data with swse, The semantic web search engine." *Web semantics: science, services and agents on the world-wide web* 9, Vol. 4 ,2011: 365-401.

[129] Krieger, M. "Search Engine" Duck Duck Go" Experiences Traffic Surge in Wake of NSA Scandal." *Liberty Blitzkrieg*,2013.

[130] Mukhopadhyay, Debajyoti, Manoj Sharma, Gajanan Joshi, Trupti Pagare, and Adarsha Palwe. "Experience of Developing a Meta-Semantic Search Engine." *International Conference on Cloud & Ubiquitous Computing & Emerging Technologies:* 167-171 ,2013.

[131] Wang, Hai-Feng, Kai-Fu Lee, and Qiang Yang. "Search engine with natural language-based robust parsing for user query and relevance feedback learning." U.S. Patent 6,766,320, issued July 20, 2004.

[132] d'Aquin, Mathieu, and Enrico Motta. "Watson, more than a semantic web search engine." *Semantic Web* 2, Vol. 1 ,2011: 55-63.

[133] R. Jain, N. Duhan and A. K. Sharma, "Developing human family tree using SWRL rules," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, 2016: 3374-3379.

[134] Haase, Peter, Holger Lewen, Rudi Studer, Duc Thanh Tran, Michael Erdmann, Mathieu d'Aquin, and Enrico Motta. "The neon ontology engineering toolkit." *WWW* ,2008.

[135] Menzies, Tim. "Cost benefits of ontologies." *intelligence* 10, Vol. 3 ,1999: 26-32.

[136] Milton, Simon K., Chris D. Keen, and Sherah Kurnia. "Understanding the benefits of ontology use for australian industry: a conceptual study." In *21st Australasian Conference on Information Systems*: 1-3. 2010.

[137] Cruz, Isabel F., and Huiyong Xiao. "The role of ontologies in data integration." *Engineering intelligent systems for electrical engineering and communications* 13, Vol. 4 ,2005: 245.

[138] Visser, Pepijn RS, Dean M. Jones, Trevor JM Bench-Capon, and M. J. R. Shave. "An analysis of ontology mismatches; heterogeneity versus interoperability." In *AAAI 1997 Spring Symposium on Ontological Engineering, Stanford CA., USA*: 164-72. 1997.

[139] Wache, Holger, Thomas Voegele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Hübner. "Ontology-based integration of information-a survey of existing approaches." In *IJCAI-01 workshop: ontologies and information sharing*, vol. 2001: 108-117. 2001.

[140] Ranjna Jain, Neelam Duhan, A.K.Sharma, "Comparative Study on Ontology Management Approaches in Semantic Web", International Journal of Computer Sciences and Engineering, Vol.6, Issue.1:132-140, 2018.

[141] Pesquita, Catia, Cosmin Stroe, Isabel F. Cruz, and Francisco M. Couto. "BLOOMS on AgreementMaker: Results for OAEI 2010." *Ontology Matching* ,2010: 134.

[142] Jean-Mary, Yves R., E. Patrick Shironoshita, and Mansur R. Kabuka. "Asmov: Results for oaei 2010." *Ontology Matching*126 ,2010: 2010.

[143] Jean-Mary, Yves, and Mansur Kabuka. "Asmov: Ontology alignment with semantic validation." In *Joint SWDB-ODBIS Workshop*: 15-20. 2007.

[144] Gracia, Jorge, Jordi Bernad, and Eduardo Mena. "Ontology matching with CIDER: evaluation report for OAEI 2011." *Ontology Matching* ,2011: 126.

[145] Li, Juanzi, Jie Tang, Yi Li, and Qiong Luo. "RiMOM: A dynamic multistrategy ontology alignment framework." *IEEE Transactions on Knowledge and data Engineering* 21, Vol. 8 ,2009: 1218-1232.

[146] Massmann, Sabine, Salvatore Raunich, David Aumüller, Patrick Arnold, and Erhard Rahm. "Evolution of the COMA match system." In *Proceedings of the 6th*

International Conference on Ontology Matching-Volume 814: 49-60. CEUR-WS. org, 2011.

[147] Ngo, DuyHoa, and Zohra Bellahsene. "YAM++: a multi-strategy based approach for ontology matching task." In *International Conference on Knowledge Engineering and Knowledge Management*: 421-425. Springer, Berlin, Heidelberg, 2012.

[148] Essayeh, Aroua, and Mourad Abed. "Towards ontology matching based system through terminological, structural and semantic level." *Procedia computer science* 60 ,2015: 403-412.

[149] Cheatham, Michelle, and Pascal Hitzler. "String similarity metrics for ontology alignment." In *International Semantic Web Conference*: 294-309. Springer, Berlin, Heidelberg, 2013.

[150] Zheng, Jin Guang, Linyun Fu, Xiaogang Ma, and Peter Fox. "SEM+: tool for discovering concept mapping in Earth science related domain." *Earth Science Informatics* 8, Vol. 1 ,2015: 95-102.

[151] Hassen, Walid. "Medley results for OAEI 2012." In *Proceedings of the 7th International Conference on Ontology Matching-Volume 946*: 168-172. CEUR-WS. org, 2012.

[152] Shao, Chao, Lin-Mei Hu, Juan-Zi Li, Zhi-Chun Wang, Tonglee Chung, and Jun-Bo Xia. "RiMOM-IM: a novel iterative framework for instance matching." *Journal of computer science and technology* 31, Vol. 1 ,2016: 185-197.

[153] McGuinness, Deborah L., Richard Fikes, James Rice, and Steve Wilder. "The chimaera ontology environment." *AAAI/IAAI* 2000 ,2000: 1123-1124.

[154] Raunich, Salvatore, and Erhard Rahm. "Target-driven merging of taxonomies with Atom." *Information Systems* 42 ,2014: 1-14.

[155] Lambrix, Patrick, and He Tan. "SAMBO—a system for aligning and merging biomedical ontologies." *Web Semantics: Science, Services and Agents on the World Wide Web* 4, Vol. 3 ,2006: 196-206.

[156] Kotis, Konstantinos, George A. Vouros, and Konstantinos Stergiou. "Towards automatic merging of domain ontologies: The HCONE-merge approach." *Web*

semantics: Science, services and agents on the world wide web 4, Vol. 1 ,2006: 60-79.

[157] Noy, Natalya F., and Mark A. Musen. "The PROMPT suite: interactive tools for ontology merging and mapping." *International Journal of Human-Computer Studies* 59, Vol. 6, 2003, pp: 983-1024.

[158] Nora Maiz, Muhammad Fahad ,Omar Boussaid, Fadlia Bentayeb, “Automatic Ontology Merging by Hierarchical Clustering and Inference Mechanisms” , proceeding of IKNOW, 2010:81-93

[159] Lacasta, J., J. Nogueras-Isso, P. Zarazaga-Soria, and R. Muro-MedraVol. "Generating an urban domain ontology through the merging of cross-domain lexical ontologies." *Conceptual Models for Urban Practitioners*, 2008: 69-84.

[160] De Mello, Marília T., Mara Abel, and Francisco García-sánchez. "Using semantic web services to integrate data and processes from different web portals.", 2007.

[161] Pant, Gautam, Padmini Srinivasan, and Filippo Menczer. "Crawling the web." In *Web Dynamics*: 153-177. Springer, Berlin, Heidelberg, 2004.

[162] Islam, Noman, Muhammad Shahab Siddiqui, and Zubair A. Shaikh. "TODE: A dot net based tool for ontology development and editing." In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, vol. 6: V6-229. IEEE, 2010.

[163] Weiten, Moritz. "Ontostudio as a ontology engineering environment." In *Semantic knowledge management*: 51-60. Springer, Berlin, Heidelberg, 2009.

[164] Calvanese, Diego, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. "Tractable reasoning and efficient query answering in description logics: The DL-Lite family." *Journal of Automated reasoning* 39, Vol. 3, 2007: 385-429.

[165] Haase, Peter, Holger Lewen, Rudi Studer, Duc Thanh Tran, Michael Erdmann, Mathieu d’Aquin, and Enrico Motta. "The neon ontology engineering toolkit." *WWW* , 2008.

[166] Miller, George A. "WordNet: a lexical database for English." *Communications of the ACM* 38, Vol. 11, 1995: 39-41.

[167] Miller, George A., Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. "Introduction to WordNet: An on-line lexical database." *International journal of lexicography* 3, Vol. 4, 1990: 235-244.

[168] Popov, Borislav, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, Angel Kirilov, and Miroslav Goranov. "Towards semantic web information extraction." In *Human Language Technologies Workshop at the 2nd International Semantic Web Conference (ISWC2003)*, vol. 20. 2003.

[169] Jain, Ranjna, Neelam Duhan, A. K. Sharma. "Design of Building Automatic Global Concept Indexer for Ontology Alignment". *International Journal of Engineering and Technology (IJET),* Vol 9, No 3, 2017: 1532-1541.

[170] Jain, Ranjna, Neelam Duhan, A. K. Sharma. "A Novel Method for Building Indexer for Aligning Ontologies". *International Journal of Information Retrieval and Research, IGI Global*, Vol 8, Issue 4, 2017:67-86.

APPENDIX-1

The following is the list of sample URLs generated as an output from Query URL Builder Process.

1. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-10-qm-2
2. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-10-qm-1
3. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-10-qm-3
4. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-2-qm-2
5. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-2-qm-1
6. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-9-qm-3
7. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-9-qm-1
8. - > https://www.naukri.com/HTML-jobs-in-Noida-ex-6-qm-1
9. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-9-qm-2
10. - > https://www.naukri.com/HTML-jobs-in-Noida-ex-6-qm-2
11. - > https://www.naukri.com/HTML-jobs-in-Noida-ex-6-qm-3
12. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-2-qm-3
13. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-10-qm-1
14. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-4-qm-3
15. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-4-qm-2
16. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-6-qm-1
17. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-10-qm-2
18. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-10-qm-3
19. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-4-qm-1
20. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-6-qm-2
21. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-6-qm-3
22. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-3-qm-3
23. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-3-qm-2
24. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-3-qm-1
25. - > https://www.naukri.com/HTML-jobs-in-Noida-ex-4-qm-3
26. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-1-qm-3
27. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-1-qm-2
28. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-1-qm-1

29. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-7-qm-1
30. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-6-qm-1
31. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-7-qm-2
32. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-6-qm-2
33. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-7-qm-3
34. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-8-qm-1
35. - > https://www.naukri.com/HTML-jobs-in-Noida-ex-4-qm-2
36. - > https://www.naukri.com/HTML-jobs-in-Noida-ex-4-qm-1
37. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-8-qm-3
38. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-8-qm-2
39. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-6-qm-3
40. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-6-qm-1
41. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-6-qm-2
42. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-3-qm-2
43. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-6-qm-3
44. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-3-qm-3
45. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-3-qm-1
46. - > https://www.naukri.com/PHP-jobs-in-Delhi-ex-9-qm-3
47. - > https://www.naukri.com/PHP-jobs-in-Delhi-ex-9-qm-2
48. - > https://www.naukri.com/PHP-jobs-in-Delhi-ex-9-qm-1
49. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-8-qm-3
50. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-2-qm-1
51. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-8-qm-2
52. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-10-qm-3
53. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-8-qm-1
54. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-10-qm-2
55. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-2-qm-3
56. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-10-qm-1
57. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-2-qm-2
58. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-2-qm-1
59. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-8-qm-2
60. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-8-qm-3
61. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-2-qm-2

62. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-8-qm-1
63. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-2-qm-3
64. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-4-qm-1
65. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-4-qm-3
66. - > https://www.naukri.com/JAVA-jobs-in-Noida-ex-4-qm-2
67. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-8-qm-1
68. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-8-qm-2
69. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-6-qm-2
70. - > https://www.naukri.com/HTML-jobs-in-Mumbai-ex-8-qm-3
71. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-6-qm-3
72. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-4-qm-2
73. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-4-qm-3
74. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-4-qm-1
75. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-3-qm-2
76. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-3-qm-1
77. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-6-qm-1
78. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-3-qm-3
79. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-7-qm-3
80. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-7-qm-2
81. - > https://www.naukri.com/PHP-jobs-in-Mumbai-ex-7-qm-1
82. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-3-qm-3
83. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-2-qm-1
84. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-2-qm-2
85. - > https://www.naukri.com/PHP-jobs-in-Noida-ex-2-qm-3
86. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-3-qm-2
87. - > https://www.naukri.com/HTML-jobs-in-Delhi-ex-3-qm-1
88. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-8-qm-3
89. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-8-qm-2
90. - > https://www.naukri.com/JAVA-jobs-in-Mumbai-ex-8-qm-1
91. - > https://www.naukri.com/JAVA-jobs-in-Delhi-ex-7-qm-1

APPENDIX-2

The sample of information extracted from *www.naukri.com*, by applying the proposed OntoJobExtractor framework is shown as below:

```
----------------------------------------------
Web Source : www.naukri.com
POST NO : 1
Title : Core Java Developer, Java Programmer
Org : Risebird
Skills : Hibernate, Spring Mvc, JSF, Wicket, Java EE, GWT, JEE, Spring Framework...
Location : Bengaluru, Delhi NCR, Hyderabad
Experience : 1-6 yrs
Salary :   Not disclosed
Datetime : 1 day ago
Description : We are looking for a 2-6 yrs Java Developer with experience in building high-
performing, scalable, ...
Link    :    https://www.naukri.com/job-listings-Core-Java-Developer-Java-Programmer-
Risebird-Bengaluru-Delhi-NCR-Hyderabad-1-to-6-years-
260118000883?src=jobsearchDesk&sid=15172923637258&xp=1&px=1
----------------------------------------------
Web Source : www.naukri.com
POST NO : 2
Title : Core Java Developer - Associate / Sr. Associate Roles @ Sapient
Org : Sapient Consulting Pvt. Ltd
Skills : core java, spring, hibernate, webservices, multithreading, javascript, java...
Location : Delhi NCR
Experience : 5-10 yrs
Salary :   Not disclosed
Datetime : 5 days ago
Description : -Providing technical expertise for every phase of the project lifecyclefrom
concept development to ...
Link : https://www.naukri.com/job-listings-Core-Java-Developer-Associate-Sr-Associate-
Roles-Sapient-Sapient-Consulting-Pvt-Ltd-Delhi-NCR-5-to-10-years-
151217005123?src=jobsearchDesk&sid=15172923637258&xp=2&px=1
----------------------------------------------
Web Source : www.naukri.com
POST NO : 3
Title : Core Java Developer - Multithreading
Org : Premium
Skills : Maven, Ant, Core Java, Jenkins, Eclipse, Design Patterns, Multithreading...
Location : Delhi NCR, Gurgaon
Experience : 4-9 yrs
Salary :   Not disclosed
Datetime : 1 day ago
Description : Must have Skills:  Exp : 5+ Yrs  - Experience: 4 to 8 years  - Core Java ...
```

Link : https://www.naukri.com/job-listings-Core-Java-Developer-Multithreading-Delhi-NCR-Gurgaon-4-to-9-years-290118901408?src=jobsearchDesk&sid=15172923637258&xp=3&px=1

Web Source : www.naukri.com
POST NO : 4
Title : Senior Associate - Core Java
Org : Sapient Consulting Pvt. Ltd
Skills : hibernate, spring, core java, design patterns, java, j2ee, multithreading...
Location : Noida, Gurgaon, Bengaluru
Experience : 5-9 yrs
Salary : Not disclosed
Datetime : 12 days ago
Description : NA
Link : https://www.naukri.com/job-listings-Senior-Associate-Core-Java-Sapient-Consulting-Pvt-Ltd-Noida-Gurgaon-Bengaluru-5-to-9-years-180118006818?src=jobsearchDesk&sid=15172923637258&xp=4&px=1

Web Source : www.naukri.com
POST NO : 5
Title : Senior Core Java Developer
Org : NIIT Ltd.
Skills : Javascript, JQuery, Angularjs, Html5, CSS, JDBC, Swing, JUnit, GIT...
Location : Gurgaon
Experience : 1-3 yrs
Salary : Not disclosed
Datetime : 1 day ago
Description : Job Title : Senior Software Engineer (Core Java) Required Technical Skills: ...
Link : https://www.naukri.com/job-listings-Senior-Core-Java-Developer-NIIT-Ltd-Gurgaon-1-to-3-years-090118003727?src=jobsearchDesk&sid=15172923637258&xp=5&px=1

Web Source : www.naukri.com
POST NO : 6
Title : Sr. Software Engineer/ Technical Lead- Core Java, Multithreading
Org : SafeNet Infotech. Pvt. Ltd.
Skills : JSP Servlets, Core Java, Maven, Multithreading, MySQL, Algorithms, Software...
Location : Noida
Experience : 4-9 yrs
Salary : Not disclosed
Datetime : 15 days ago
Description : As a Senior Software Engineer/ Technical Lead&nbsp;in Gemalto,&nbsp; you will design ...

Link : https://www.naukri.com/job-listings-Sr-Software-Engineer-Technical-Lead-Core-Java-Multithreading-SafeNet-Infotech-Pvt-Ltd-Noida-4-to-9-years-020417001007?src=jobsearchDesk&sid=15172923637258&xp=6&px=1

APPENDIX-3

The following is the list of jobs in structured format.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#"
     xml:base="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl"/>
     <!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#haslocation -->
    <owl:ObjectProperty
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#haslocation"
>
        <rdfs:domain
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#job"/>
        <rdfs:range
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#location"
/>
    </owl:ObjectProperty>
    <!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#hasskill -->
    <owl:ObjectProperty
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#hasskill">
        <rdfs:domain
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#job"/>
        <rdfs:range
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#skill"/>
    </owl:ObjectProperty>
```

```
   <!-- ///////////////////////////////////////////////Data properties/////////////////////////////////////////      -
->
   <!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#hascompany -->
   <owl:DatatypeProperty
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#hascompany
">
       <rdfs:domain
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#job"/>
       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>
  <!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#hastitle -->
   <owl:DatatypeProperty
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#hastitle">
       <rdfs:domain
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#job"/>
       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>
     <!-- ///////////////////////////////////////////////Classes/////////////////////////////////////////      -->
   <!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#job -->
   <owl:Class
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#job"/>
  <!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#location -->
   <owl:Class
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#location"/>
       <!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#skill -->
   <owl:Class
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#skill"/>
   <!-- ///////////////////////////////////////////////Individuals/////////////////////////////////////////      -->
   <!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#ggn -->
   <owl:NamedIndividual
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#ggn">
```

```
        <rdf:type
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#location"
/>
    </owl:NamedIndividual>
     <!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#java -->
    <owl:NamedIndividual
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#java">
        <rdf:type
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#skill"/>
    </owl:NamedIndividual>
<!-- http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#job111 -->
    <owl:NamedIndividual
rdf:about="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#job111">
        <rdf:type
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#job"/>
        <hascompany
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">TCS</hascompany>
        <hastitle          rdf:datatype="http://www.w3.org/2001/XMLSchema#string">project
mananger</hastitle>
        <haslocation
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#ggn"/>
        <hasskill
rdf:resource="http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#php"/>
    </owl:NamedIndividual>
```

BRIEF PROFILE OF THE RESEARCH SCHOLOR

Name:	Ms. Ranjna Jain
Designation:	Assistant Professor, Department of Information Technology, BSAITM, Faridabad
Qualification:	Ph.D. (2011- to current) M.Tech (2010), First Class with Distinction B.E (2005), First Class with Distinction
Research Interests:	Semantic Web, Information retrieval, ontology, Web Mining.

Work Experiences:

- Assistant Professor, Department of Information Technology, BSAITM, Faridabad (2015-to current)
- Assistant professor, Department of Computer Engineering, Rawal Institute of Engineering and Technology. (2011-2012).
- Sr. Lecturer, Department of Computer Science and Engineering, BSAITM, Faridabad (2008-2010)
- Lecturer, Department of Computer Science and Engineering, BSAITM, Faridabad (2005-2008)

LIST OF PUBLICATIONS

List of Published Papers

S.No.	Title of Paper	Name of Journal/ Conference where published	Volume & Issue	Year	Pages
1	**Developing Human Family Tree using SWRL Rules**	IEEE Conference ID: 37465 2016 3rd International Conference on "Computing for Sustainable Global Development"		March, 2016	pp. 3374-3379
2	**Comparative Study on Semantic Search Engines**	International Journal of Computer Applications	Vol. 131, Issue 14	Dec., 2015	pp. 4-11
3	**Design of Building Automatic Global Concept Indexer for Ontology Alignment** (Paper indexed in Scopus)	International Journal of Engineering and Technology (IJET)	Vol. 9, Issue 3	May, 2017	pp. 1532-1541
4	**A Novel Method for Building Indexer for Aligning Ontologies** (Paper indexed in ACM library)	International Journal of Information Retrieval Research (IJIRR)	Vol. 8, Issue 4	Oct., 2018	pp. 67-86
5	**Comparative Study on Ontology Management Approaches in Semantic Web**	International Journal of Computer Sciences and Engineering	Vol.6, Issue.1,	Jan., 2018	pp.132-140

List of Communicated Papers

S. No.	Title of the paper	Name of Journal	Present Status	Year
1	**Ontojobextractor: Relevant Information Extraction from Job Boards**	International Journal of Web Engineering and Technology(IJWET) Journal Indexed in Scopus	Communicated	2018
2	**Ontojob Query Processor: An Ontology Driven Query Processing Method**	Journal of Computer Science Indexed in Scopus	Communicated	2018
3	**Ontology Based Cross Domain Interoperability Search System**	International Journal of Computer Application in Technology(IJCAT) Journal Indexed in Scopus	Communicated	2018