Is a SPARQL Endpoint a good way to Manage Nursing Documentation

by

Muhammad Aslam

Supervisor: Associate Professor Jan Pettersen Nytun

Project report for Master Thesis in Information & Communication Technology
IKT 590 in Spring 2014

University of Agder
Faculty of Engineering and Science

Grimstad, 2 June 2014

Status: Final

Keywords: Semantic Web, Ontology, Jena, SPARQL, Triple Store, Relational Database

Abstract.

In Semantic Web there are different technologies available, among these technologies ontologies are considered a basic technology to promote semantic management and activities. An ontology is capable to exhibits a common, shareable and reusable view of a specific application domain, and they give meaning to information structures that are exchanged by information systems [63]. In this project our main goal is to develop an application that helps to store and manage the patient related clinical data. For this reason first we made an ontology, in ontology we add some patient related records. After that we made a Java application in which we read this ontology by the help of Jena. Then we checked this application with some other database solutions such as Triple Store (Jena TDB) and Relational database (Jena SDB). After that we performed SPARQL Queries to get results that reads from databases we have used, on the basis of results that we received after performing SPARQL Queries, we made an analysis on the performance and efficiency of databases. In these results we found that Triple Stores (Jena TDB) have capabilities to response very fast among other databases. In this report we also try to present an idea about [62] load times of other native triple stores and discuss the inferencing capabilities of Sesame native, Mulgara and Virtuoso backed with Jena [62]. In this [62] also discuss the suitability and performance of the triple store when used as a backend for Bioportal.

Preface

This report is the final result of a 30 credits Master Thesis, IKT590, completed at the Faculty of Engineering and Science, University of Agder (UiA) in Grimstad, Norway. The work on this project started from 10th January 2014 and ended on 30th May 2014. The main goal of our project is, “Is a SPARQL endpoint a good way to manage nursing documentation”.

I am really very obliged to thank our university supervisor, Associate Professor Jan Pettersen Nytun for his constructive support and supervision throughout my thesis without which it would have been really difficult to achieve my goals. His timely feedbacks helped me correcting my mistakes and improving the thesis. I am again very thankful to him for his assistance during meetings at University of Agder.

I am also grateful to my family and friends back in Pakistan, specially my parents to support me to study at the University of Agder.

Muhammad Aslam
University of Agder,
Grimstad, Norway
June 2nd , 2014

Contents

List of figures

List of Tables.

Abbreviations.

RDF	Resource Description Framework.
XML	Extensible Markup Language.
OWL	Web Ontology Language.
KRL	Knowledge Representation Languages.
EHR	Electronic Health Record.
ICNP	International Classification for Nursing Practice.
POJO	Plain Old Java Objects.
HTTP	Hyper Text Transfer Protocol.
RIF	Rule Interchange Format
SPARQL	Simple Protocol and RDF Query Language.

1 Introduction.

The Semantic Web is a liaison of the information that is labeled in a manner that allows information processing systems and systems to access and utilize it effectively. The basic aim of the semantic web is to provide a framework that permits information to be applied again and again across multiple applications.

The Semantic Web is a collaborative movement led by the international standards body, the World Wide Web Consortium (W3C) [1]. The standard promotes common data formats on the World Wide Web [1].

Moreover, The Semantic Web is an evolving extension of the World Wide Web in which the semantics of information and services on the web is explained, making it possible to interpret and meet the requests of people and machines to use it well.
At its heart, the semantic web consists of a lot of design principles, collaborative working groups, and a sort of enabling technologies. Some components of the semantic web are yet to be carried out or taken in. Other components of the semantic web are defined in formal specifications. Some of these include Resource Description Framework (RDF), a form of data exchange formats (e.g. RDF/XML, N3, Turtle, N-Triples), Knowledge representation languages (KRLs) or notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are meant to offer a schematic description of concepts, terms, and relationships within a given knowledge area. You will find a brief explanation about above listed concepts in chapter 2 (Theoretical Background) of this report.
A different interpretation of the semantic web is that it would be a good deal more time-consuming to produce and put out content because there would need to be two formats for one piece of information: one for human viewing and one for machines [1]. Nevertheless, many web applications in development are addressing this event by creating a machine-readable format upon the publishing of information or the request of a machine for such data. To sustain this idea we try to build a semantic web application that keep the record of nursing patients in the form of machine readable data.

1.1 Background.

The “semantic web” is an elongation of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [16]. The semantic web is much more useful than the present day web where we use HTML. The semantic web comprises a lot of design principles, collaborative working groups, and a sort of enabling technologies [17]. The restrictions of the current web can be solved by the semantic web. The Semantic Web standards and tools include XML, XML Schema, RDF, RDF Schema and OWL that are organized in the stack. This provides service in e-commercialism and commercial enterprise to business applications.

The accelerated adoption of nursing terminologies supports new opportunities to create semantically interoperable healthcare applications and solutions for evidence-based medical specialty. One essential for meaningful usage of miscellaneous Electronic Health Record (EHR) data is a common incorporated clinical model to ensure explicit data representation, rendering, and substitution within and across heterogeneous sources and applications. To have a semantic web application that can query the patients data over HTTP, here we tried to make a semantic web application that store and manage the nursing patient records efficiently and medical staff (nurses) can use that data when they need. For example, from the use of this application they are able to query the data about patients and their records, but to understand whole scenario of semantic web application and how they work there is a need to have some general understanding which we have presented in next parts of this report.

1.1.1 Purpose of Semantic Web.

Most of today’s Web content is suitable for human “ingestion”, even Web content that is generated automatically from databases is normally shown without the original structural information found in databases. Most of the people uses web today for seeking and making use of information, searching for and getting in touch with other people, reviewing catalogues of online shops and ordering products by filling out

forms. The current Web activities are not particularly well supported by software tools Except for keyword-based search engines (e.g. Google, AltaVista, Yahoo).
The Key Problem of Today's Web is that the meaning of Web content is not machine-accessible, which means lack of semantics. The Semantic Web Approach is to define Web content in a strain that is more easily machine-process able, apply intelligent techniques to take vantage of these representations.
Straight off a day's it is very common that man are capable of using the Web to conduct out their jobs such as getting a Dutch word for "train", reserving an airplane ticket, and searching for a low price for an iPhone. Nonetheless, just only a computer is not being able to deliver the goods the same tasks without human directions because web pages are designed to be read by people, not machines.

The primary concept of a 'semantic web' certainly coming from some marking code other than bare HTML, is built along the acceptance that it is not achievable for a machine to suitably interpret code based on goose egg but the order relationships of letters and lyric. If this is not truthful, and so it may be possible to make a 'semantic web' an HTML alone, pulling in a specially built 'semantic web' coding system unnecessary. In that location are hidden dynamic network models that can, under certain conditions, be 'trained' to appropriately 'learn' which means on the basis of data of the order in the "learning" process for relations (a variety of elementary grammar work).

The Semantic Web Impact actually means the Knowledge Management, Knowledge management concerns itself with acquiring, accessing, and maintaining knowledge within an establishment. Semantic web enabled knowledge management it means knowledge will be organized in conceptual spaces according to its meaning for that because there are automated tools for maintenance and knowledge discovery. Moreover, Semantic Web does not build just on text based manipulation, but rather on machine-processable metadata.

1.1.2 Limitations on Today's Web.

Numerous files on a conventional computer can be generally divided into documents and records. Documents like educational notes, mail messages, reports, and patient care clinical records are read by humans. Data, like calendars, address records, play lists, and spreadsheets are presented using an application program which lets them be viewed, searched and combined in many ways. At present, the World Wide Web is built mainly on documents written in Hypertext Markup Language (HTML), a markup convention that is used for coding a body of text scatter with multimedia objects such as images and related courses.

We can take here an example Metadata tags;

<meta name="keywords" content="computing, computer studies, computer">

<meta name="description" content="5478 for sale">

<meta name="author" content="AAAA"> [18]

This example can provide a method by which computers can identify the content of web pages.

With HTML and a tool to contribute it (possible web browser, possibly with user agent), one can develop and present a page that lists items for sale. The HTML of this catalog page can get simple, document-level assertions such as "this document's title is 'Taj Superstore'". But there is no competence within the HTML itself to declare unambiguously that, for example, item number 5478 is an AAAA with a retail price of Rs.200, or that it is a consumer product. To a certain degree, HTML can only say that the span of text "5478" is something that should be positioned near "AAAA" and "Rs.200", etc. There is no way to say "this is a catalog" or even to establish that "AAAA" is a kind of title or that "Rs.200" is a price. There is also no way to express that these pieces of information are bound together in describing a discrete item, distinct from other items perhaps listed on the page.

Semantic HTML [3] refers to the traditional HTML practice of markup following intention, rather than specifying layout details at once. For example, the use of

denoting "emphasis" rather than <i>, which specifies italics. Layout details are left up to the browser, in combination with Cascading Style Sheets [3]. But this practice falls short of specifying the semantics of objects such as items for sale or prices. Microformats[4] represent unofficial attempts to extend the HTML syntax to create machine-readable semantic markup about objects such as retail stores and items for sale. For this kind or problems Semantic Web is a solution.

1.1.3 Solution by Semantic Web.

The Semantic Web takes the solution further. It involves publishing in languages specifically designed for data: Resource Description Framework (RDF) [5], Web Ontology Language (OWL) [6], and Extensible Markup Language (XML) [7]. HTML describes documents and the links between them. RDF, OWL, and XML, by difference, can describe superficial things such as people, meetings, or automobile parts. These technologies are together in order to provide descriptions that continuation or replace the content of Web documents. Thus, content may obviously as descriptive data stored in Web-accessible databases, or as markup within documents. The machine-readable descriptions enable content managers to add meaning to the content, i.e. to describe the structure of the knowledge we have about that content. In this way, a machine can process knowledge itself, rather of text, using processes similar to the human deductive reasoning and inference, thereby obtaining more meaningful results and expedite the automated information gathering and research by computers.

Here is an example of a tag that would be used in a non-semantic web page:

<item>monkey</item>

Encoding similar information in a semantic web page might look like this:

<item rdf:about="http://dbpedia.org/resource/Monkey">Monkey</item>

1.1.4 Example Application about Semantic Web.

Let’s suppose a semantic web system was created to conduct or to carry out the selling and buying of used cars over the internet.

The system could contain two main applications. One for people who wanted to buy a car and the other for people who wanted to put up a car for sale.

In a “real life” this application may ask to identify ourselves the first time we used it. Our ID would be stored in an RDF file. ID would identify as a person with name, address, email and a Personal number. When we submitted the query, the application would return a list of cars for sale, and the list could be trained down and categorized by year, price, location, and availability. This information would be returned by searching the web for RDF files continuously.

People who want to sell a car can use the ISA application. When we submit the form, the application would ask for more information and store ID and the information in RDF file made available to the web. The RDF file would contain information like:

ID: Name, address, email, Personal Number, Phone Number.

Selling item: type, model, picture, price, picture, specifications.

Behind the scene the "ISA" application creates an RDF file with a lot of RDF pointers. It creates an RDF pointer to a file with information about your person, an RDF pointer to information about BMW and BMW models, an RDF pointer to BMW dealers and resellers, about parts, about prices, and much more. An RDF pointer is a pointer (actually an URL, i.e. <http://www.w3.org/1999/02/22-rdf-syntax-ns#>) to information about things (like a knowledge database). The beauty about this is that you don't have to describe yourself, or the car model. The RDF application will sort it out for you with the help of SPARQL and Jena Programming tools.

1.2 Problem Statement.

The Semantic Web aims to establish a common framework that allows data to be shared and reused across applications, enterprises, and community boundaries. In current situations nursing reports are generally written by hands or in a text editor. There is an on-going project at UiA that concerns correct and professional use of nursing terminology in nursing documentation. The nursing terminology in question

is described in The International Classification for Nursing Practice (ICPN®), which is an ontology expressed in Web Ontology Language (OWL). An application for learning nursing terminology is being developed; the application involves testing of student's terminology skills.

Our project is a subproject of this concerning how to effectively store such data (where one requirement is use of SPARQL). RDF is the representational language being used and SPARQL is the preferred RDF query language. A triple store is a database used for storing and querying RDF data. We have to build a semantic web application that is capable to demonstrate how we can set up our own SPARQL endpoint (services that accept SPARQL queries and return results).

1.3 Literature Review.

1.3.1 Correct and professional use of nursing terminology in nursing Documentation.

Any written or printed record of activities is termed as a document. In health care domain the patient's medical record is a document, which is legal and encompasses all the activities involving patient care. It primarily includes the patient's bio data, administration of tests, past and present medical history, treatments, procedures, results and response of patient to treatments, changes in patient's condition, response to intervention, evaluation of expected events and complaints from patients and household.

The methods of reporting and recoding the relevant data about the patient care have been produced as a response to the standards of practice, policies, society's norms and for legal and regulatory measures. This transcription and accounting system are counted as the major ways of communication between health care providers [31,32].

1.3.2 Purposes of Documentation.

There are two main functions of documentation, that is, professional responsibility and the accountability. The documentation provides the evidence of accountability of health care professionals and their responsibilities, to the patient, institutions,

profession and the society. Apart from this the documentation is also a source of communication, and for a purpose of research and auditing [31,32].

Therefore, nursing documentation is of the most important clinical documentation. There must be the practice of correct and professional use of terminology in nursing documentation. A thorough nursing documentation is believed as a precondition for secure patient care and the efficient cooperation and communication within the team of health care professionals [33,34]. The systematic methodology for nursing practice generally in International Classification for Nursing Practice (ICNP) consists of three phases, there are other process models with different number of phases, that is an assessment of the relevant information, diagnosis and definition of patients problem & resources, derivation of nursing aims, planning tasks, implementation & documentation of performed tasks, evaluation of nursing care and redefinition of possible care plan [35]. In [38] they state:

- **Assessment of relevant data:** The assessment of the information related to an actual and potential health care need is summarized. During review, any new findings and changes in patient's condition are highlighted.

- **Definition of patient's problem and resources:** with the help of International Classification of Nursing Practice (ICNP) international terminology, patient's problems and needs are identified.

- **Planning and outcome detection:** The probable outcomes and goals of patient/client care should be documented on the care plan or on the critical pathway instead of on progress notes.

- **Implementation:** After performing intervention, observations, findings, treatments, teaching, and relevant clinical judgments and patient/client's response should be documented on the progress notes and on flow sheet.

- **Evaluation of nursing care:** The effectiveness of interventions is evaluated and documented in terms of expected outcomes. Advance towards the goal, response of patient to tests and treatments, the client and his/her family response to teaching, questions, statements, complains and nursing interventions should all be documented.

- **Redefinition of possible care plan:** The revision of planned care and reasons along with supporting evidence for it and clients agreement are also documented. [36]

Following the nursing process the nursing documentation must be logical, focused, related and relevant to care, the outcomes representing the each phase of the nursing process.

1.3.3 Systems of documentation:

Systems of documentation of data relevant to patient/client care have been developed primarily in response to the demands and requirements, that health care professionals be held to societal norms, necessity of professional standards of practice and of legal & regulatory standards. It is also the requirement of institutional policies and standards [31,32]. The paper based system was usually used for nursing documentation, and that takes high documentation efforts, low quality and the limited acceptance of the nursing process are reported [34, 39-41]. Thus, to support documentation, system documentation and computerized nursing process has been innovated to dilute the high efforts, documentation, improve the quality and to allow the reuse of data for management of the patient cure process and inquiry. [42-47].

Subsequently, several terminologies have been developed in order to serve as a response sets for nursing diagnosis, outcomes, and interventions. It is presently the preview of the American Nurses Association [ANA] Committee on Nursing Practice Information Infrastructure, to develop the recognition criteria that formally recognize the terminologies meeting the established criteria [48]. During the year, early 1990s,

the Committee on Nursing Practice Information Infrastructure, has recognized more than one terminology [i.e. Response set] for each of the data elements such as diagnosis, intervention and outcome. More examples of the recent recognition of entities are, Systematized Nomenclature of Medicine Clinical Terms [SNOMED CT] and ABC Codes. These two entities encompass the content from the originally recognized nursing terminologies such as, NANDA, NOC, NIC, Omaha System, Peri-operative Nursing Data Set, the Clinical Care Classification and International Classification of Nursing Practice (ICNP). But the need is to use these recognized entities in a correct way to achieve our professional goal of generating the comparable nursing practice data [49].

It has been considered in previous researches that medical support is represented usually as a free text, which almost makes it unusable for all other analytical functions. Therefore, it would be of great benefit of having data structured in a computerized readable form. Such type of web of data enables web applications to access ample sources of information and provides the intelligence overhaul. The main idea or focus is to complement the natural language text in the web with the unambiguous and explicit semantics based on the formal knowledge representation. Therefore, this can be accessed automatically by the computer to interpret the information represented in the form of natural language. The RDF, which is Resource Description Framework, is used usually as the formal language to represent such data. Hence, semantic web technologies consist of such technologies that help in a buildup of semantic based representation of data and processing of web information. [50]

1.4 Problem Solution.

The effectiveness of discovering SPARQL endpoints for a given URI has been expressed as a desired property of Linked Open Data [55]. In our case, we will try to present different strategies for that URI-to-endpoint resolution, based on a sample of a Triple Dataset. The main objective of this application is to look into how to best store and manage such data, and what are the profiling of these characters of data

when it adds up to efficiency and performance. First, we will set the working Ontology to insert data, after having the data in ontology we will make a web application for setup a SPARQL Endpoint application, then we try to build Triple Store and Relational Databases for the same Ontology, through which we will be able to access data on http server by applying SPARQL Queries. You will find a complete scheme of setup SPARQL Endpoint and how we query our patient records from different type of Databases in Chapter 3 (Solution).

1.5 Report Outline.

In this report I try to survey the state of the art current enabling technologies for Semantic Web Services for making a semantic web application that will provide assistance to manage clinical data. Further, we analyze and contrast three approaches of Databases for semantic web services according to the proposed dimensions. The rest of the report is structured as follows: in Chapter 2 (Theoretical Background) we provide a general overview of Semantic Web services, tools and technologies and how we are going to use these technologies for our problem solution; Chapter 3 (Solution) presents the whole setup and implementation of ontology data and code through which we have solved our problem, this includes; Requirements (which tools we need to solve the problem are) its implementation, design specification and finally its validation and Testing. In chapters 4-5 we discuss and conclude the principal differences among the approaches used to store and handle the patient information and their adequacy, efficiency, productivity and effectiveness that are submitted.

2 Theoretical Background.

The desire to expand the capabilities of the Web to publishing of structured data is not new, and can be drawn back to the earliest proposal for the World Wide Web [Endnote: http://www.w3.org/History/1989/proposal.html]. According to Berners-Lee [60]"The first step is putting data on the Web in a form that machines can naturally understand, or converting it to that form. This creates what I call a Semantic Web – a web of data that can be processed directly or indirectly by machines". While the Semantic Web, or Web of Data, is the goal or the end result of this process, Linked Data offers the way to achieve that end. Nevertheless, in recent years the Web has evolved from a global information space of connected documents to one where both documents and information are related. At the core of this development is a set of best practices for publishing and connecting structured Web known related data.
Medical documentation is represented usually as a free text, which almost makes it unusable for all analytical purposes. Therefore, it would be of great benefit of having data structured in a computerized readable form. Web search is a key technology of the Web, since it is the primary way to access content in the ocean of Web data. Current Web search technologies are essentially based on a combination of textual keyword search with an importance ranking of documents via the link structure of the Web [23]. For this reason, however, current standard Web search does not allow for a semantic processing of Web search queries, which analyzes both Web search queries and Web pages with respect to their meaning, and returns exactly the semantically relevant pages for a query. For the same reason, current standard Web search also does not allow for evaluating complex Web search queries that involve reasoning over the Web [24].

2.1 Technologies in Semantic Web.

The semantic web consists of the standards and tools of XML, XML Schema, RDF, RDF Schema and OWL that are organized in the Semantic Web Stack [9]. The OWL Web Ontology Language Overview [8] describes the function and relationship of each of these components of the semantic web.

The RDF, which is Resource Description Framework is used usually as the formal language to represent such data. Hence, semantic web technologies consist of such technologies that help in a buildup of semantic based representation of data and processing of wed information [25].

RDF [5] is a simple language for expressing data models, which refer to objects [11] and their relationships. An RDF-based model can be represented in XML syntax. RDF Schema [10] is a vocabulary for describing properties and classes of RDF-based resources, with semantics for generalized-hierarchies of such properties and classes. Moreover, RDF provides the common framework that helps in representing data or information in the semantic web. Previous researches have shown that this framework has provided the means by encoding the data in the form of 'set of triples'. The set of triples is in the form of subject-predicate-object. The subject and the Predicate must be the URI and the Object either can be a URI or a literal and the collection of triples forms the RDF graph. The statement in RDF triple states the relationship, which is indicated by the predicate that exists between the subject and the object of the triple. Therefore the meaning of the RDF graph is the conjunction of the statements that corresponds to all of the triples it contains [26].

XML [7] provides an elemental syntax for content, structure within documents, yet associates no semantics with the meaning of the content contained within. XML Schema [10] is a language for support and restricting the structure and content of elements contained within XML documents.

OWL [8] adds more vocabulary for describing properties and classes: among others, relations between classes, cardinality equality, richer typing of properties, and characteristics of properties (e.g. Symmetry), and enumerated classes.

Current ongoing Semantic Web standardizations include:

Rule Interchange Format (RIF) [12] as the Rule Layer of the Semantic Web Stack. The intent is to improve the usability and value of the Web and its interconnected resources [13] through:

Servers which expose existing data systems using the RDF and SPARQL standards. Many converts to RDF exist for different applications. Relational databases [14] are an important source. The semantic web server attaches to the existing system without affecting its operation.

Documents "marked up" with semantic information (an extension of the HTML <meta> tags that used in today's Web pages to supply information for Web search engine that using the web crawler) [1]. This could be machine-understandable information about the human-understandable content of the document (such as the developer, label, description, etc., of the document) or it could be purely metadata representing a set of facts (such as resources and services elsewhere in the site). (Note that anything that can be identified with a Uniform Resource Identifier (URI) can be described, so the semantic web can reason about animals, people, places, ideas, etc.) Semantic markup is often generated automatically, rather than manually. Ordinary metadata terminologies (ontologies) and maps between terminologies that allow document developers to know how to mark up their documents so that agents can use the information in the supplied metadata (so that Author in the sense of 'the Author of the page' won't be confused with Author in the sense of a book that is the subject of a book review) [1].

Programmed agents to perform tasks for users of the semantic web using this data; Web-based services (often with agents of their own) to supply information specifically to agents (for example, a Trust service that an agent could ask if some online store has a history of poor service or Spamming) [1].

2.2 Terminologies Used.

In this section we present the general idea about the tools and terminologies which we are going to use to solve our problem of this project to understand the basic concepts about ontology like; how classes, objects, individual and their objects and data properties works. Here we present the general idea the implementation and design work you will find in Chapter 3 (Solution).

In Computer science modeling techniques, traditional knowledge includes semantic web, expert systems, conceptual schema and the entity-relationship diagram. These traditional techniques work better in tight areas and are not good for reuse, web sharing or scalability. The main difficulty is that knowledge sharing modelers differ in the interpretation of relationships, properties and objects in a domain.

The semantic web is a reinforced plan in which agents communicate calculated to accomplish the agenda web. Ontology-based models are used increasingly to provide precise definitions and logical deductions necessary to automate processes in a shareable web environment.

2.2.1 What is an ontology?

Ontologies are considered as one of the pillars of the Semantic Web, even if they do not have a universally accepted definition [19]. A (Semantic Web) vocabulary can be regarded as a special form of (usually mild) ontology, or sometimes simply as a collection of URI with a (usually informal) descriptive sense [19].

In Computer science ontology is a model of wider knowledge with a reasoning mechanism that facilitates the sharing of knowledge on the semantic web. In ontology, knowledge about a domain is modeled using a knowledge representation language with a reasoning mechanism. The knowledge representation language is used to create a set of conditions and assumptions (axioms) on the meanings of words and to specify the classes, properties and relationships between classes and objects in the domain. Reasoning engines are available to reason based on the semantics of properties and relationships between classes and individuals referred. There are reasoning algorithms to check the consistency of a model and build taxonomic structures.

Construction of ontologies is a research topic very relevant in terms of extracting information from the web [20]. Ontologies are constructed using an ontology language, such as RDF, OWL, etc. and connected to each other in a decentralized manner to clearly express the semantic content and organize semantic boundaries to extract specific information [20].

The role of Ontologies on the web is to provide a shared understanding of domain like; semantic interoperability, overcome differences in terminology and mapping between ontologies. Ontologies are useful for the organizations and navigation of web sites. Ontologies are also useful in web search and it improves the accuracy of web search [51]. For example, search engines can look for the pages that refer to a precise concept in an ontology. By the use of ontology web search accomplish generalization or specialization of an information. For example, if any query fails to find a relevant document the search engine can suggest the user a more general query and there is a need of too many answers to retrieve, the search engine can suggest the user a specialization [52].

Ontology helps in providing solutions for the document identification, authentication of end to end services, authorization, data integrity, confidentiality, organization and sharing of isolated pieces of information, it also faces some limitations, In [20] that are stated as;

- Quite impossible to define the boundaries of ontologies of a particular domain's abstract model.
- Automatic ontology creations, the automatic emergence of ontologies to create new ontologies and identification of possible existing relationships between classes to draw the taxonomy hierarchy automatically is required.
- Ontology validators are restricted and not capable of validating all kinds of ontologies e.g. based on complex inheritance relationship.
- Domain specific ontologies are highly dependent on the domain of the application and because of this dependency it is not possible to find out the general purpose ontologies from them.

Because of these limitations in the Semantic Web ontology is not currently able to achieve the real objectives of the structuring of any information on the web in the process of the machine format.

The idea to define here ontology is to give some information, because to solve our problem area we have to make an ontology to have data and through this we will be able to access and manage that data by using different databases. The whole structure of our ontology you will find in chapter three (3). In next parts of this chapter

you find information about the tools and technologies we used to make our semantic web application.

2.2.2 What is Protégé.

In this section I will introduce the basic idea of ontology modelling using the Protégé development tool.

An ontology for semantic web is usually built in a notation (language) as Resource Description Framework Schema (RDFS) and Web Ontology Language (OWL). These assessments are derived from Extended Markup Language (XML). Tools are available to help in the construction of ontologies. Current editing tools support, visualization, query and inference as well as more advanced tasks such as aligning and merging ontologies. Tools may include a knowledge representation language or can support a range of languages and Knowledge Representation can promote a particular development methodology.

Protégé is a system used to generate ontologies. It offers the power to define the logical relationships between classes and individuals, and for the generation and debugging of ontologies and translation into several basic notations.

The Protégé platform supports two main ways of modeling ontologies via the Protégé-Frames and Protégé-OWL editors. Protégé ontologies can be exported into a variety of formats including RDF, RDFS, OWL, and XML Schema. Protégé is based on Java, is extensible, and provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development. Examples are a visual editor for OWL (called OWLViz), storage back-ends to Jena and Sesame, as well as an OWL-S plugin, which provides some specialized capabilities for editing OWL-S descriptions of Web services [21].

2.2.3 Ontology Terminologies.

In the literature on ontologies, you will determine that the terms are employed reciprocally in books, tutorials and documents. For example, the term object is used interchangeably with the term individual. This utilization is due in part because the subject of study of ontologies overlaps with the domains of databases, and object-

oriented programming mathematical logic. Terminology is suitable for these is used in dissimilar ways. In this master thesis, I have tried to use the terminology as described in the following paragraphs.

2.2.3.1 What is a Class.

The term "class" shall mean a collection or group of individuals. For example, the class of cars will be equipped with individuals representing cars that may exist, such as your Audi and his Corolla. In this section we use proper names in Courier font to designate classes. For example, the class containing all persons is denoted by `Person`.

2.2.3.2 What is an Individual (Object).

An individual is a specific entity that occurs and belongs to a class. The term individual is used interchangeably with the term object. For this we use capitalized names with numeric suffixes to denote specific individuals such as,

`C1, C2, Person1, Person2`

2.2.3.3 Subclasses and Superclasses.

A subclass is a sub collection of objects in the way that the class of Corolla Cars forms a subcollection of the class containing all (types of) Cars. In the same way the class of all (types of) Cars is a superclass of the class of Corolla Cars.

2.2.3.4 Object Property.

An object property of an individual is a relation of that individual to a second individual. For example, an individual person may have a friend. We like having a friend as a property of an individual. In other words, we will say 'hasFriend' as the name of a property that connects an individual to another individual who is the friend. So, here we use paired notation to denote the property or relation indicating that Person1 has Person2 as a friend such as,

`Person1 hasFriend Person2`

2.2.3.5 Datatype.

The word datatype is used to indicate the nature of a datum. For example, the datatype int shows the type of an integer number and the datatype string shows the type of a string of characters.

2.2.3.6 Datatype Property.

A property of an individual type of data is a relation to that person on one type of data. For example, the age of the individual may be 18. We say that the age of an individual is a property relate to the individual to the particular integer that represents that age of the individual. So, here we will use again paired notation to indicate the datatype property or relation indicating that Person1 is 18 years old such as;

```
Person1 hasAge 18
```

2.2.3.7 Difference of a Relation and a Property.

The term 'relation' is used interchangeably with the term 'property' to indicate a linking of an individual to a second individual or to a datatype. Moreover, a relation may be an association between two classes, for example a link between a document and an organization that published that document. A property is a characteristic of a class in a specific dimension such as legal name of an organization.

2.2.3.8 Description Logic.

A description logic, is a logical notation for describing or forming classes. For example, if Cycle and CycleSteps are classes then

```
Cycle or CycleSteps
```

is a description logic expression describing or creating a further class that contains all individuals from either class Person1 or class Person2. The class `Person1 or Person2` is a superclass of both class Person1 and class Person2.

As a second example, if Teacher is the name of a class and hasFriend the name of an object property the expression

```
hasFriend some Teacher
```

describes the class of all individuals who have as a friend an individual in the class Teacher, i.e., have a teacher as a friend.

2.3 SPARQL.

Basically the term stands for **S**imple **P**rotocol **a**nd **R**DF **Q**uery **L**anguage. SPARQL is the standard query language for the RDF data, which is developed by W3C SPARQL Working Group [27]. It has syntax similarities to SQL and queries the RDF graph by pattern matching. SPARQL consists of series of clauses that defines the desired information and runs against a data source, which is specified by URI that returns the result set [28].

SPARQL is the standard query language for RDF, as SQL is the standard query language for relational databases. In [22] they have define that If you are familiar with SQL, you will see some similarities because it shares several keywords such as `SELECT, WHERE,` etc. SPARQL contain some new keywords that you have never seen if you come from a SQL world such as `OPTIONAL, FILTER` and much more [22]. Main idea of SPARQL is pattern matching that describes the sub graphs of the queried RDF graph, sub graph that match your description yield a result.

A SPARQL query is executed on a RDF dataset, which can be a native RDF database, or on a Relational Database to RDF (RDB2RDF) system, such as Ultrawrap. These databases have SPARQL endpoints which accept queries and return results via HTTP[22].

2.3.1 SPARQL Query.

SPARQL is the standard language to query graph data represented as RDF triples. SPARQL is one of the three core standards of Semantic Web, alongwith RDF, and Owl.

2.3.2 Types of SPARQL Queries.

There are several kind of SPARQL queries which are defined below,

- **SELECT** Return a table of all X, Y, etc. satisfying the defined conditions.

- **CONSTRUCT** Find all X, Y, etc. satisfying the conditions, and substitute them into the defined template in order to generate (possibly new) RDF statements, creating a new graph.
- **DESCRIBE** Find all statements in the dataset that provide information about the resource(s), (identified by name or description).
- **ASK** Are there any X, Y, etc, that satisfying the conditions.

2.3.3 Structure of SPARQL Query.

The structure of a SPARQL query is define below,

PREFIX owl: <http://www.w3.org/2002/07/owl#> (Definition of all prefixes define like that)

SELECT ?name (this is the type of query that what to search for, i.e., Variable)

WHERE

{ ?x rov:legalName ?name } (this is RDF triple pattern, i.e., the conditions that to be met).

2.4 Jena.

Jena is a framework for Java. It provides an API to extract data from and write to graphs. The RDF graphs are represented as an abstract "model". A model can be sourced with data from files, databases, URLs or a combination of these. A Model can also be queried through SPARQL and updated through SPARUL [30]. Jena implements APIs for dealing with Semantic Web building blocks such as RDF and OWL. Jena's fundamental class for users is the Model, an API for dealing with a set of RDF triples. A Model can be created from the file system or from a remote file. Using JDBC, it can also be tied to an existing RDBMS such as MySQL [29]. In our scenario I am using Jena because it will help me to access data from our OWL file, Triple Store and Relational Database, the whole scheme of accessing data from these databases have decribed in chapter 3 (Application Design).

2.5 Related Works

In the past, a couple of attempts were cleared in a couple of institutions to avail with the documentation of data and metadata. Some projects similar to this very project have been attained in different countries Universities and Colleges but still there is an indigence of a program specially designed for writing of nurse reports of patients is there. Some of the related works done I have mentioned as follows:

- RDF, Jena, SPARQL and the “Semantic Web” [53].

  In this paper [53] they show how RDF-XML is used to serialize the information represented using graphs, how RDF graphs can be read and written by using the Jena software package, and how distributed graphs can be queried using the SPARQL query language. It includes examples showing how SPARQL can be used to query the data (such as the Gene Ontology) that is structured in hierarchies, and how SPARQL queries can be submitted through sparkle “endpoints” [53].

- Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints [54].

  In this paper [54] they introduce the Berlin SPARQL Benchmark (BSBM) for comparing the performance of these systems across architectures. The benchmark query mix illustrates the search and navigation pattern of a consumer looking for a product. After affording an overview about the invention of the benchmark, the paper demonstrates the outcomes of an experiment comparing the performance of D2R Server, a relational database to RDF wrapper, with the performance of Sesame, Virtuoso, and Jena SDB, three popular RDF stores [54].

- Discoverability of SPARQL Endpoints in Linked Open Data [55].

  This paper [55] presents a quantitative analysis along the automatic discoverability of SPARQL endpoints using different mechanisms. They explore the success rates of those strategies using a large representative

sample of URIs in Linked Open Data, and hash out the consequences. Furthermore, they exhibit a simple, multi-strategy resolution service which delivers SPARQL endpoints for URIs [55].

- Efficiently Querying RDF Data in Triple Stores [56].

  In this report [56], they have introduced a novel scheme to store, index, and query RDF data in triple stores. A graph feature of RDF information is taken into considerations which might help trim down the join costs on the vertical database structure. They partitioned RDF triples into overlapped groups, store them in a triple table with one more column of group identity, and make up a signature tree to index them [56].

- Usage-Centric Benchmarking of RDF Triple Stores [57].

  In this report [57], they purposed a generic benchmark creation procedure for SPARQL, which they give to the DBpedia knowledge base. In contrast to previous approaches, their benchmark is based on queries that were actually issued by humans and applications against existing RDF data not resembling a relational schema. In add-on, their approach does not simply contain the query string, but also the features of the queries into consideration during the benchmark generation process [57].

- SP2Bench: A SPARQL Performance Benchmark [58].

  In this paper [58], they have developed SP2Bench, a publicly available, language-specific SPARQL performance benchmark. SP2Bench is settled in the DBLP scenario and comprises both a data generator for creating arbitrarily large DBLP-like documents and a set of carefully designed benchmark queries. The generated documents mirror key characteristics and social-world distributions encountered in the original DBLP data set, while the queries implement meaningful requests on top of this data, insuring a variety of SPARQL operator constellations and RDF access patterns [58].

- Deep integration of spatial query processing into native RDF triple stores [59].

  In this paper [59], they have discussed how spatial data processing can be natively integrated into RDF modeling and querying. Their solution models, spatial features as typed complex literals, and defines spatial predicates as filter functions in SPARQL. Moreover, they discuss the rich integration of these concepts into RDF triple stores, and present an implementation of a triple store with spatial functionality [59].

3 Application Design.

This section of report divided into different section. As I know that for my solution I need to setup a SPARQL Endpoint, for that first we look at the tools and technologies we need. After that I will start development and implementation of an application.

3.1 Requirements for SPARQL Endpoint Setup.

First, we need to fix up our development environment. This environment makes the basis for most of the examples in the report. Here I keep it simple and straightforward. Here is an overview of the component which I used to build my SPARQL Endpoint environment:

- Compiling and execution tools: Java Software Development Kit 7 (SDK)
- Code-editing tools: Eclipse Kepler Integrated Development Environment (IDE)
- Ontology editing tool: Protege Ontology Editor 4.3
- Semantic Web Programming Framework: Apache Jena (SDB and TDB) Semantic Web Framework 2.5.6.
- MySQL (for Relational Database Connectivity).
- REST Easy Web services.
- JQUERY.
- Tomcat Apache Web Server 7.0.

Within the Semantic Web community, most of the tools that have been developed to date use the Java programming language, and for our problem solution we are using them as well. Therefore, our examples require a Java Software Development Kit (Java SDK).I assume that readers are familiar with Java. The SDK provides you with compiling tools and a runtime virtual machine to run Java programs. I am using here the latest release of Java, The examples also work with Java 1.6. In addition to the SDK, you will need an editor. You can use any Java editor you like, but all of the examples in my application make use of the Eclipse Kepler JEE Integrated Development Environment (IDE). While my examples are oriented toward Eclipse, none of them depend on Eclipse, so can be run with other Java editors also e.g. NetBeans IDE.

We need to create, edit, and combine ontologies using an ontology editor. These files can also be hand-coded in a standard text editor, but an ontology editor offers many conveniences and features specific to ontologies. I used version 4.3 of the freely available Protégé ontology editor. In order to operate an ontology programmatically, we also need a Semantic Web programming framework. This contains the libraries to allow programs to interact with Semantic Web data, such as ontologies and instance data, and to allow application to take advantage of reasoners and query languages. For this purpose, I use the Apache Jena Semantic Web framework version 2.11.1. To help to understand the examples of Apache Jena I already have given an overview about Jena framework in chapter 2. The standard Jena download also includes extensive documentation. The documentation resides in the doc directory underneath the directory where you unzipped Jena. The Jena framework download includes several reasoners. So we don't need to worry about downloading and installing these reasoners separately.

Rest Easy is java JBOSS framework for applications for Restfull web service which is used to transfer state of resources to end user (browser) using JSON or XML in my case I am using JSON.

Tomcat Apache is a web server for java web applications, In our case our rest full java application runs inside tomcat server, when browser request url(http://localhost:8080/endpoint/message/patient) tomcat handover request to our web application. Our web application uses Jena and execute query and return result (java object) to our Rest Easy, Rest Easy converts that result into JSON format which Jquery parses and renders on browser.

3.2 Design Specification for SPARQL Endpoint.

For design specification of our SPARQL Endpoint solution, first we need to have some data which we can access over HTTP and to check its efficiency in term of use of different database approaches. For that reason we need a working ontology in which we insert some clinical related data. To build an ontology we need a Protégé Software tool (as we have mentioned earlier in our previous section) through which

can construct an ontology, as our main focus is to maintain the clinical patient record for nursing students, so will try to make an ontology that fits this goal.

After conducting several meetings with supervisor we have decided to take first a UML Model (provided from supervisor) for our ontology, then we will make a working ontology on the basis of that UML Model and insert some random data in it. The model for ontology is shown in figure 1 below.

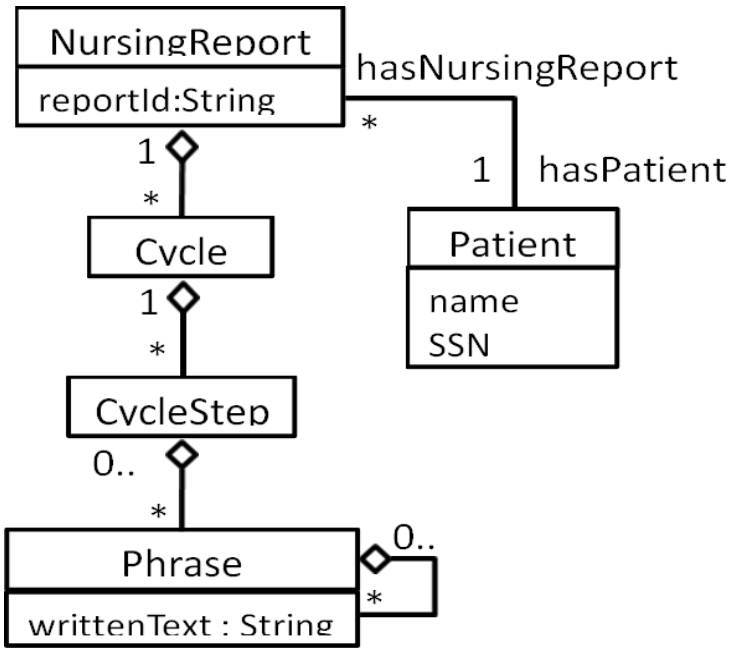


Fig. 01. UML Model for Ontology

This model has been taken to build an ontology over it, as it clearly stated in figure 1 that model have five classes, 'NursingReport', 'Cycle', 'CycleStep', 'Phrase' and 'Patient'. There are two main classes in this model. Class 'NursingReport' and class 'Patient'. The system on this model will be like, a 'Patient' has a 'NurseReport', and 'NurseReport' have one or more 'Cycle' for a patient, which has some 'CycleStep' and 'CycleStep' contains 'Phrase' which has some written text about the specific disease related to a patient. If we see generalization between classes 'NursingReport' to 'Patient' it means a 'NursingReport' contain at least one 'Patient' like written in model 'hasPatient'. While class 'Patient' have attributes 'Name' and 'SSN' that contain the information about a patient. In next sections of this report you will find the complete development mechanism of Ontology on above said Model.

3.2.1 Making Classes in Protégé.

In this section of the report we will define the complete Ontology Modeling. An Ontology consists of classes, individuals and properties of the individuals in a domain. Logical relationships between classes and individuals are specified using a description logic (DL). Reasoners are programs that interpret the description logic of the ontology and are able to check the consistency and structure of the ontology.

In this introduction we give definitions and examples of classes, objects and properties, from which we have design our problem solution.

There are several naming conventions for classes and individuals. We will use the convention that a class name starts with an uppercase letter, such as Patient. For individuals we often use a name, such as Ola.

We begin by using Protégé to construct an ontology with all classes which we have mentioned in our Figure 1 (UML Model).

Step 1. Open Protégé and set the URI to the URL where the ontology will be published, as in our case (e.g., http://www.semanticweb.org/muhammadaslam/ontologies/2014/1/untitled-ontology-32) and set the path to the location where the ontology is to be stored (e.g., C:Ontology/NurseReport.owl).

Step2. Choose the Entities tab and select the Thing class in the asserted class hierarchy window. Using the 'add subclass' button successively add classes as needed for one ontology. In our case we have added “Cycle”, “CycleStep”, “NursingReport”, “Person”, “patient” a subclass of class “Person” and a class “Phrase”, as subclasses of the universal class Thing. The asserted class hierarchy should now appear as shown in Figure 2.

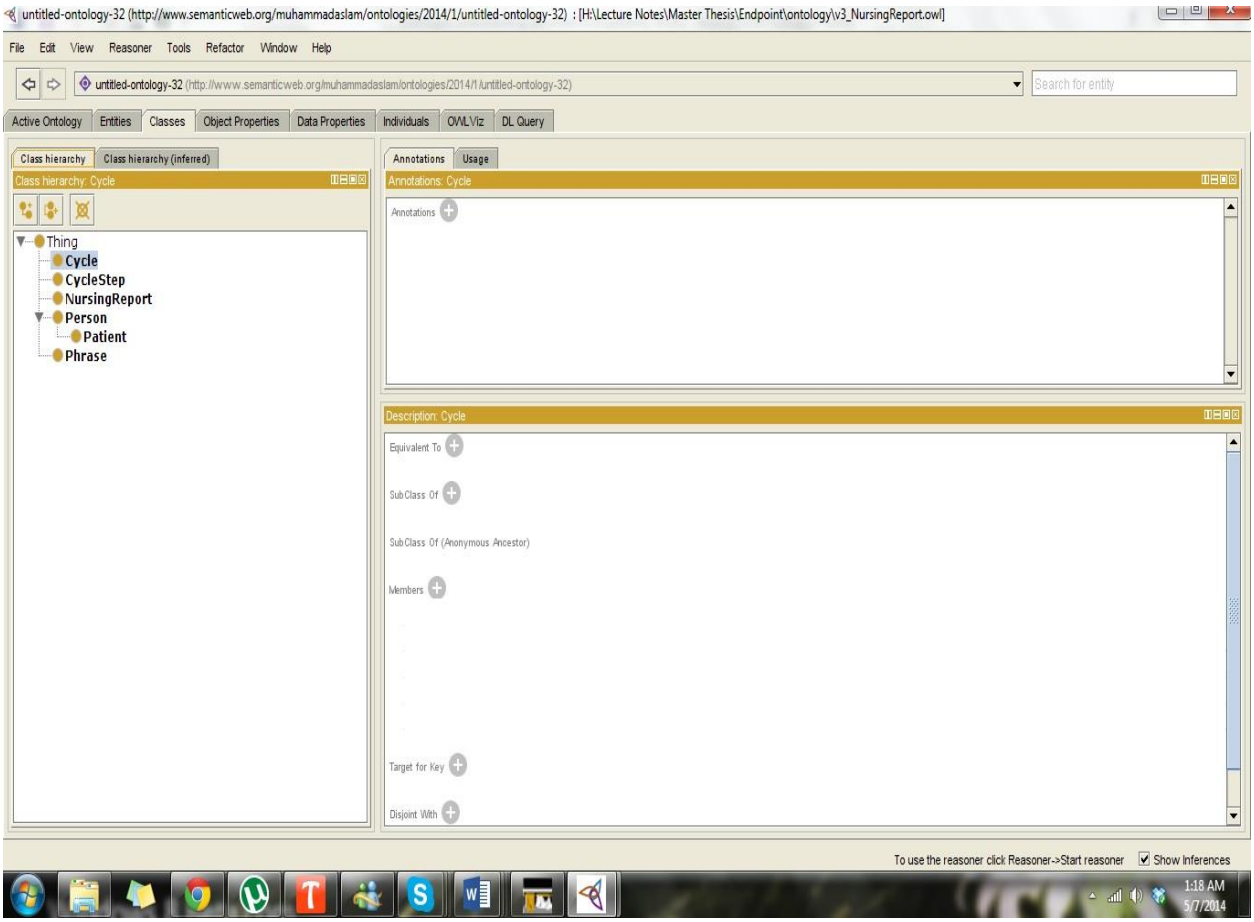


Fig. 02 Class Hierarchy of Ontology.

Representing knowledge in a domain ontology usually requires a hierarchy of classes and subclasses. A class hierarchy is also called a taxonomy. In an Owl hierarchy all classes are subclasses of the universal class called Thing, so the root of every hierarchy is the class called Thing as you can see in Figure 2 above.

The OWLViz tab in protégé provides options for visualizing the classes in the ontology as shown in Figure3. The figure shows that all classes are subclasses of the universal class that is called Thing.

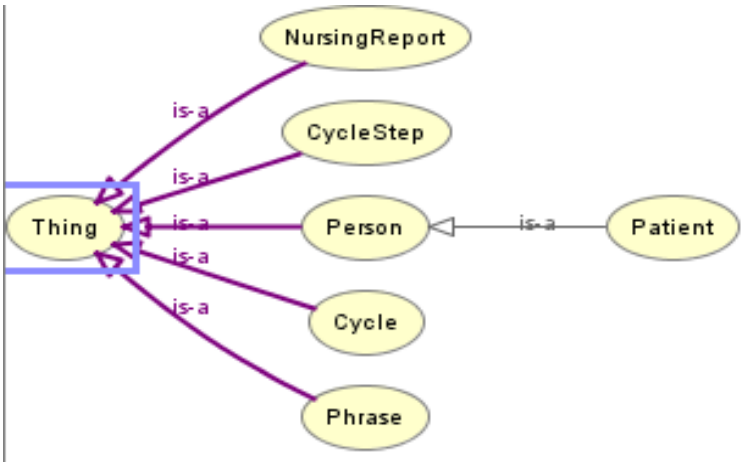


Fig. 03 OWL Viz Graph of Ontology.

3.2.2 Object Properties of Ontology.

There are two types of properties supported by Protégé (OWL): the first type of property is called an object property and the second type is called a datatype property. Knowledge base designers have found that certain categories of properties occur so frequently that it is convenient to include these categories in the basic mechanisms of Knowledge Representation Logics (KRLs). An object property may have a domain and a ranges, super- and sub-properties, inverse properties, equivalent properties and property chains. In this section we explain object properties which we have inserted for our ontology. We adopted a naming convention for properties in which properties are given names that link related objects such as 'hasPatient'. An object property specifies a relationship or link between two individuals (objects). For example in our ontology to representing relationships there is a class called Patient, Each individual patient is linked to a NurseReport. This link could be called the hasPatient link or property.

To create Object properties in Protégé Under the 'Object Properties' tab click the 'add property' button and enter the name object property as need to be create to the textfield. In the same way we have added object properties called 'hasCycle' ‘hasCycleStep’ ‘hasNurseReport’ ‘hasPatient’ ‘hasPhrase’ ‘partOfCycle’ ‘partOfCycleStep’ ‘partOfNurseReport’ in connection with the classes mentioned above. For example, for property ‘hasCycle’ we have insert class ‘NursingReport’ as domain and class “Cycle” as Range, same we have inserted different classes as

domain and range for other object properties set into the ontology . The resulting property list shown in the figure 4 below.

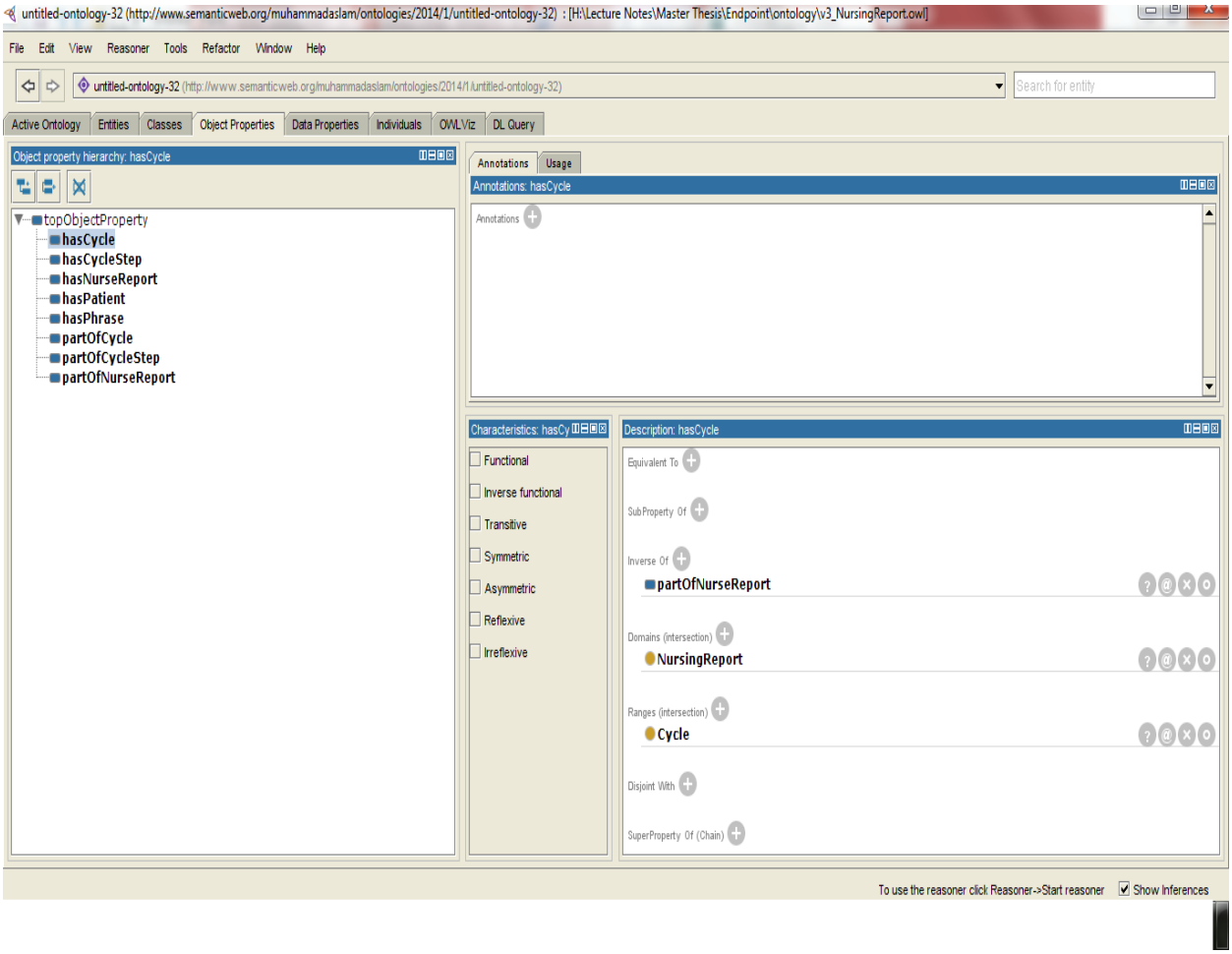


Fig 04. List of Object Properties.

If we look at figure 4 in “Description hasCycle” section we will find there is an object property ‘partOfNurseReport’ have inserted as inverse, it is because the ontology of clinical record would need to contain inverse relationship. We could call such a relation 'partOfNurseReport', that shows there is inverse relationship between classes ‘NurisngReport’ and ‘Cycle’. Which means there is always a ‘Cycle’ for a ‘NursingReport’ or a ‘Cycle’ is always being a part of ‘NursingReport’. This scenario could be more easily understand by figure 5 shown below.

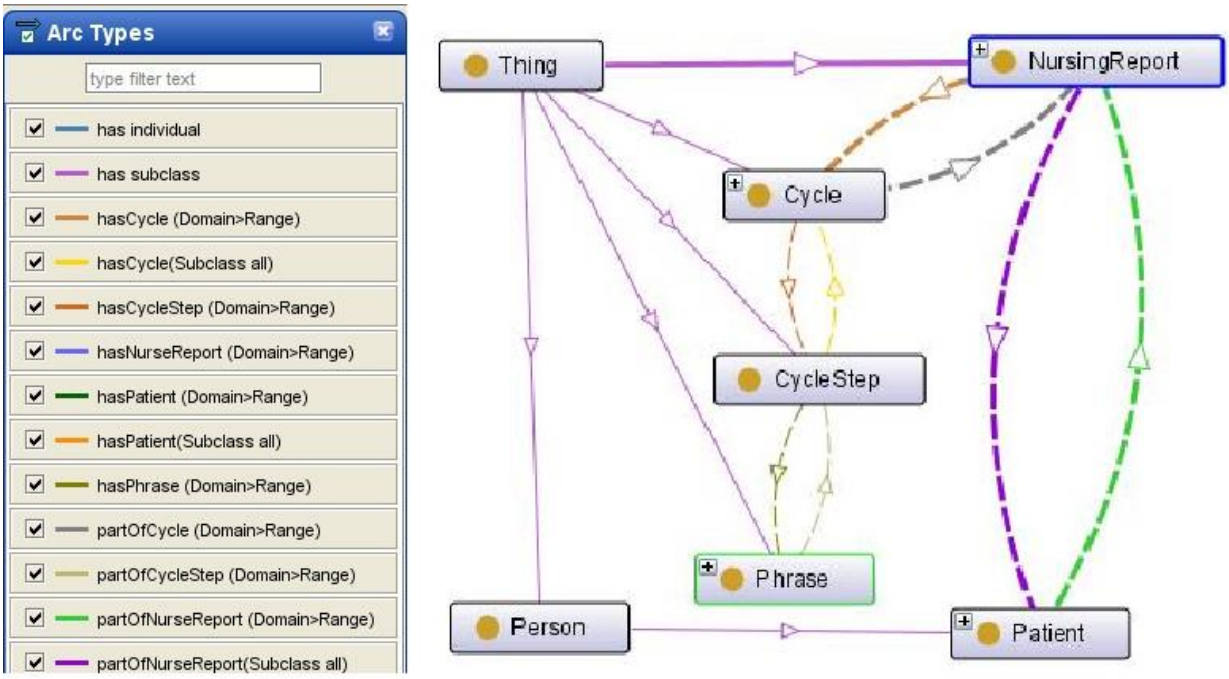


Fig. 05. Inverse Relationship of Classes.

It is clearly seen from the figure above that there is inverse relationship among other classes of ontology as well. The lines with arrow indicating that which classes have inverse relations and the type of relation among them can be seen from arc types as given left side of the figure. For that reason we have inserted inverse object properties also for these classes. If we see figure 4, in this figure we will find object property 'partOfCycle' which shows inverse relation between classes 'Cycle' and CycleStep', same as object property 'partOfCycleStep' shows inverse relation among classes 'CycleStep' and 'Phrase' and property 'hasPatient' shows inverse relation among 'Patient' and 'NursingReport'.

3.2.3 Data Properties of Ontology.

Datatype properties relate individuals to predefined datatype values or literals. For example a person individual may be linked by a datatype property to a string value representing the name of the person or to an integer representing the age of the person. Predefined datatypes include types as strings, integers and Booleans. Object properties relate objects to other objects and may be functional, inverse functional, symmetric, antisymmetric, reflexive, irreflexive and transitive. Datatype properties relate objects to datatypes and so cannot be symmetric, antisymmetric, reflexive,

irreflexive or transitive. Which means Datatype properties may also not be inverse functional.

To create a datatype property in protégé select the Data Property button and enter ‘hasName’ into the popup window. From the description menu we can select a class as a domain and a range for the defined datatype. For example, we have selected class ‘Patient’ as domain and given range ‘String’ to this ‘hasName’ datatype. The result of this operation is shown in figure 6 below.

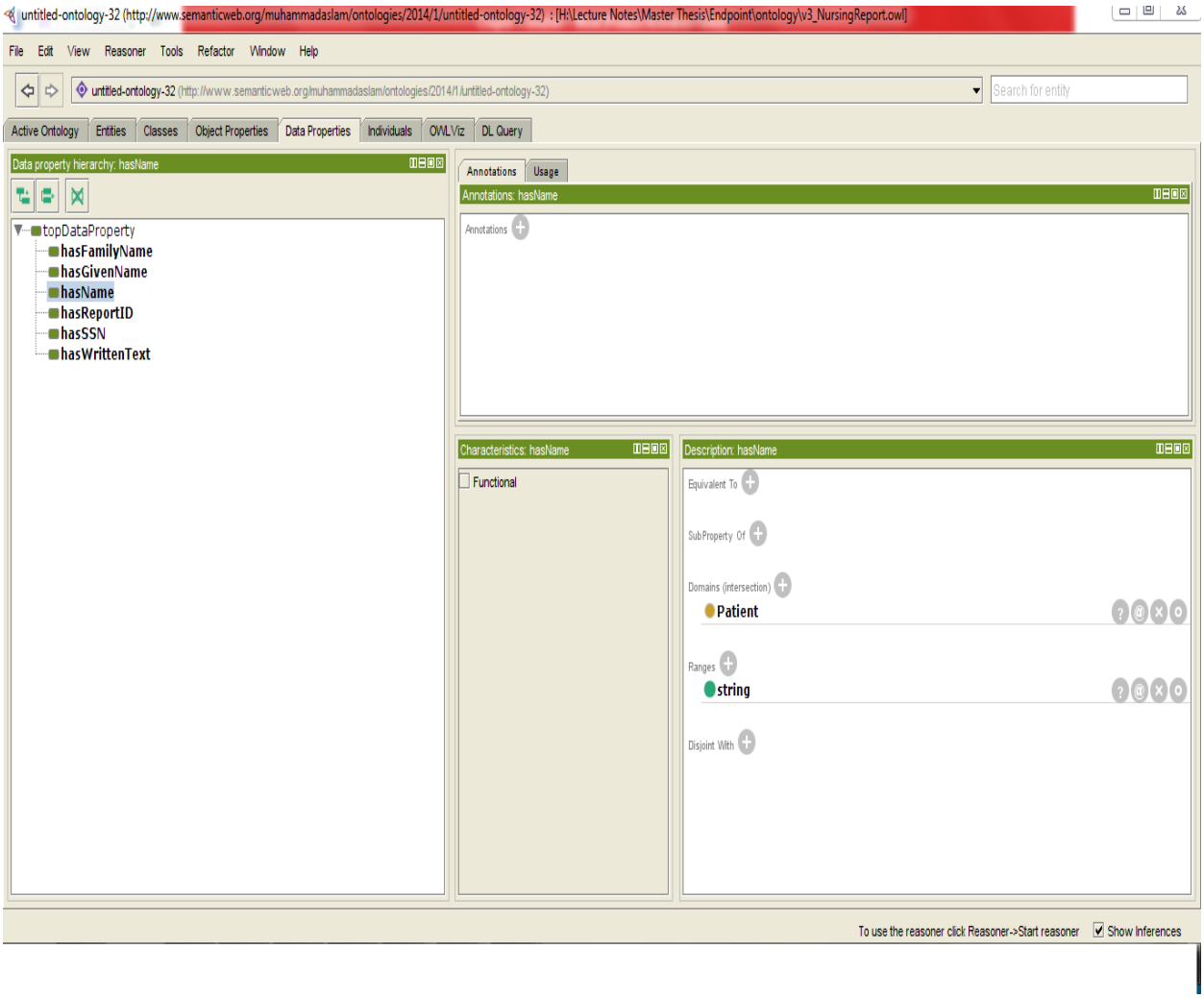


Fig. 06 Datatype Properties

From the figure 6 above it is clearly seen that we have created other datatypes also with different class assigning as domain. For datatypes ‘hasFamilyName’, hasGivenName’ and ‘hasSSN’ have asigned class ‘Person’ as domain and have selected ‘String’ as range. While datatype ’hasReportId’ has class ‘NursingReport’ as

domain and type ‘String’ as range and datatype ‘hasWrittenText’ has class ‘Phrase’ as domain and type ‘String’ as range.

These datatypes we have defined in ontology for individuals as said earlier. The details of individuals and development scheme of individuals, for example relation of individuals to classes and their object and data type properties will find next section of this report.

3.2.4 Instances of Patient as Individuals.

We can say that individuals, if they exist, belong to classes. An individual may belong to any number of classes. Individuals are said to have types. The type of an individual is the class to which the individual belongs. A class may have zero or more individuals or objects belonging to it.

In an ontology representing knowledge about a nursing record (related to our scenario) we could represent knowledge about individual patients and their individual cycles and individual NurseReports. In this case each particular patient could be an individual belonging to the type called Patient.

Creating an individual is a two-step process. First the individual is created and secondly the class (type) to which the individual belongs is specified. In this Example we use Protégé to create individual patient. As the medical record contains thousands of patient but we have added few of the existing patients for testing purpose but by the prototype the data can be entered into ontology for further use. The restrictions and the properties will be same to define for each and every patient of the record. In this section some individuals will be discussed to get knowledge about the individuals in this project. The total number of patient added to this ontology is given below.

Fig. 07. Instances of Patient

This instance of Patients can be more clearly understand in term of Ontology meaning by the Figure 8 below, this graph has been taken from “OntoGraf” function which is provided in Protégé.

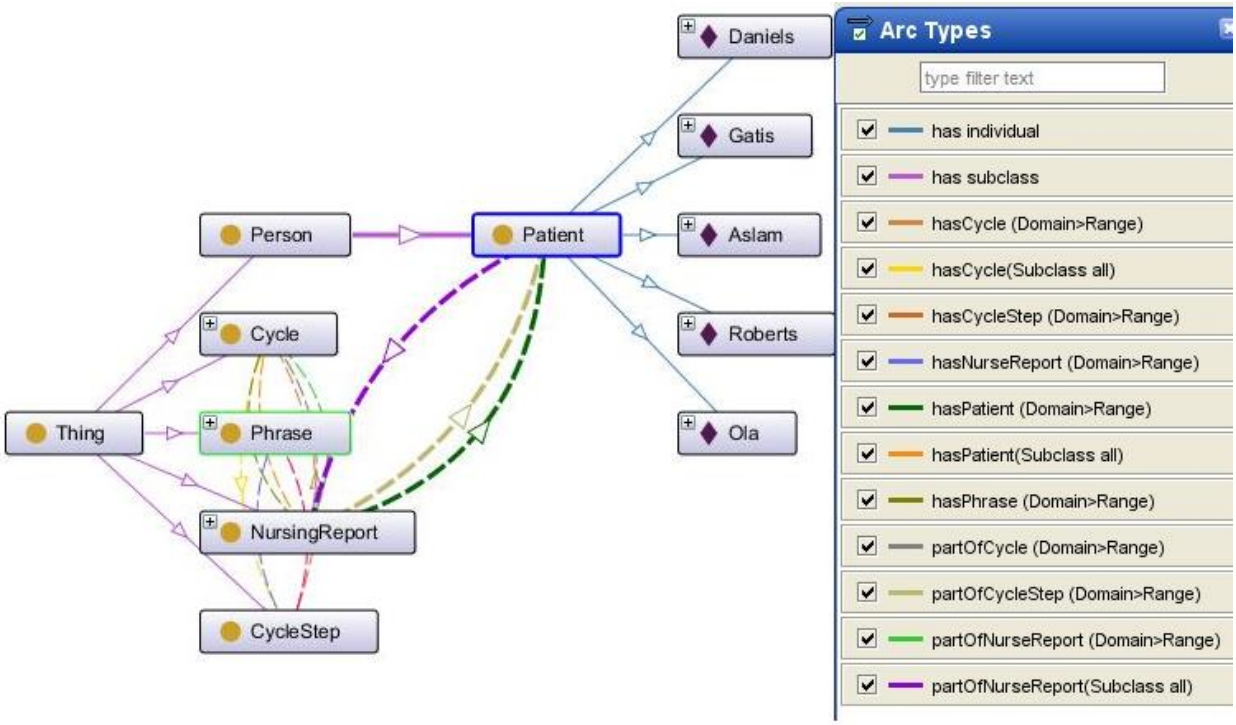


Fig. 08. Graph of Instance of Patients

To explain the graph above in term of Instance of patients and their object and data properties we can take one individual ‘Aslam’. This is an instance of class ‘Patient’. There are object properties and the data properties which define this instance. It has one type of object property asserted. First of all it has type patient class. According to our ontology as we have mentioned earlier section of this report that our ontology classes have some inverse functions. If we see above graph it clearly shows that there are one or more cycles for each and every patient, each cycle contains some cycle steps and these cycle steps contains some phrase. Phrase that has some written text about the patient disease. In this case for the moment ‘Aslam’ has object property asserted that is ‘hasNurseReport’.

There are three type of Data type properties asserted to this individual. The data type property added to this individual is ‘hasFamilyName’ which is the name of patient. The data type of this property is ‘string’ for this the value is ‘Aslam’. Another data type

property asserted to this individual is ‘hasGivenName’ which has the data type of ‘string’ the main task of this restriction is to add information of the patient in this case the value is ‘Muhammad’. The ‘hasFamilyName’ and ‘hasGivenName’ datatype properties add the information of the full name of patient. Another data type property asserted to this individual is ‘hasSSN’ which has the datatype of ‘string’ here the value of SSN is ‘4614’ which is a number for patient for identity and this ID is unique for every patient. This all instance of patient ‘Aslam’ can be more understand from the figure 9 below.

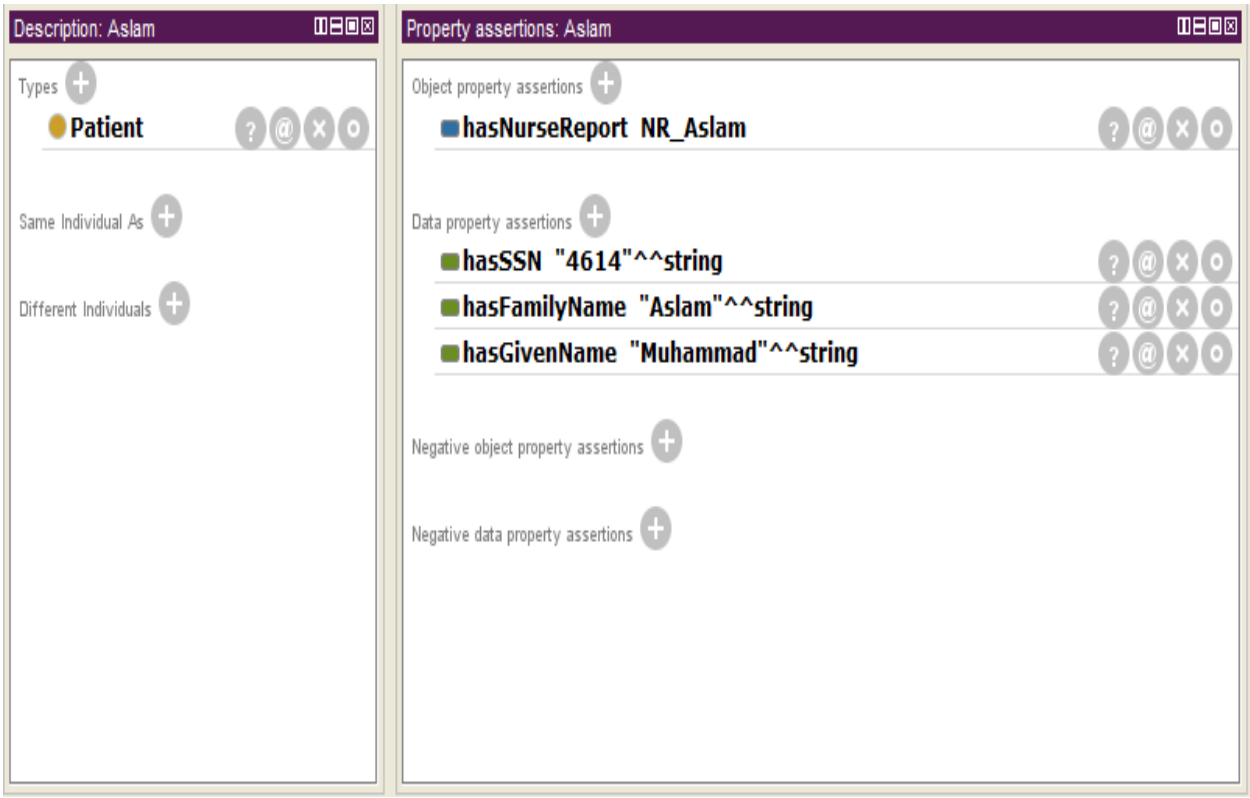


Fig. 09. Object and Data Property assertion for Instance ‘Aslam’.

It is clearly seen from the figure above that the patient ‘Aslam’ has a nurse report, so we have asserted one more individual ‘NR_Aslam’ to make a nursing report for patient ‘Aslam’ that contain all the information of that patient, as shown in figure 10 below.

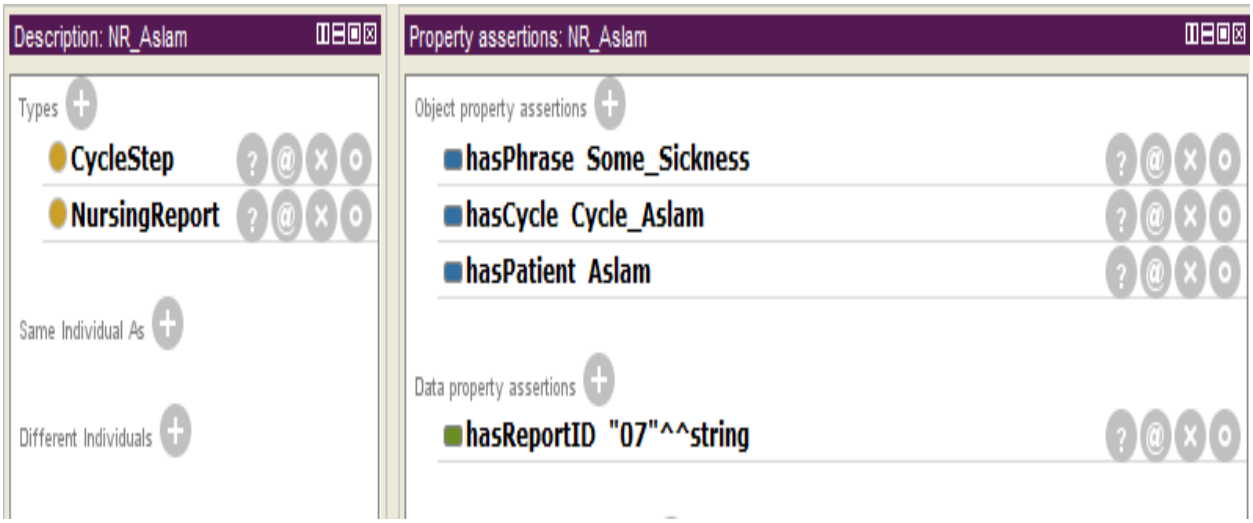


Fig. 10. Object and Data Property assertion for Instance 'NR_Aslam'.

This instance of 'NR_Aslam' have been created to define that patient 'Aslam' has a 'NursingReport' 'NR_Aslam'. This instance contain three type of object properties, 'hasPatient' which defines that 'NR_Aslam' contain patient 'Aslam', object property 'hasCycle' defines that it has a cycle and object property 'hasPhrase' defines that it also relates to the class 'Phrase' that contain a text related to the disease about that patient.

There is only one data type property asserted to this instance which is 'hasReportID" which has the datatype of 'string' here the value of report ID is '07' which is a number for nurse report and this ID is unique for each nurse report for every patient.

In connection to complete one patient process according to our ontology there must be a 'Cycle' for each patient. Therefore, we have created instance 'Cycle_Aslam'. To simplify the ontology we have made here only one 'Cycle' for each patient, but according to the ontology model there could be more than one 'Cycle' for a patient, because for every visit of patient to clinic it added a new Cycle for patient. This instance have class type 'Cycle' which shows that it is cycle of patient 'Aslam'. It has two object properties asserted 'partOfNursingReport' with instance 'NR_Aslam', which shows it is continuation with 'NursingReport' and 'hasCycleStep' with instance 'CS_C_NR_Aslam' that indicates it has some 'CycleStep' as shown in figure 11 below.

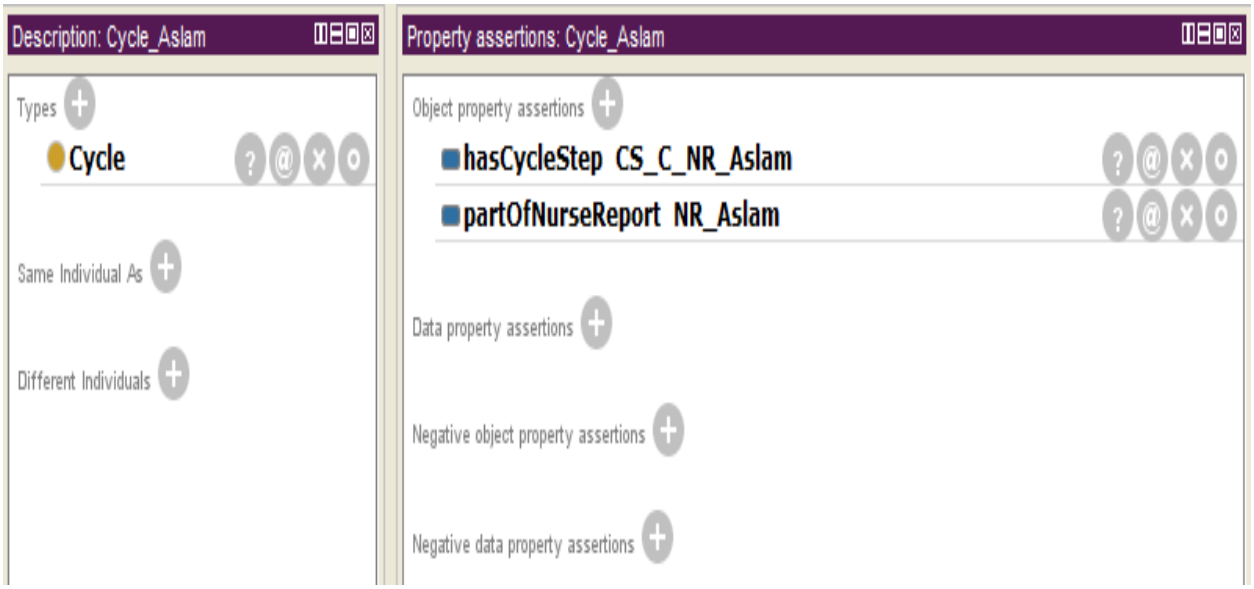


Fig. 11. Object Property Assertion for Instance Cycle_Aslam.

From the figure above it is clearly seen that there is one more instance asserted here, which is 'CS_C_NR_Aslam'. This instance has types 'CycleStep' and 'Phrase' because it is the part of a cycle step that has a phrase. There are three object properties asserted to this instance, 'partOfCycle', because it is a part of cycle of patient 'Aslam'. The other two properties are 'partOfCycleStep' and 'hasPhrase' added to this as shown in figure 12 below.

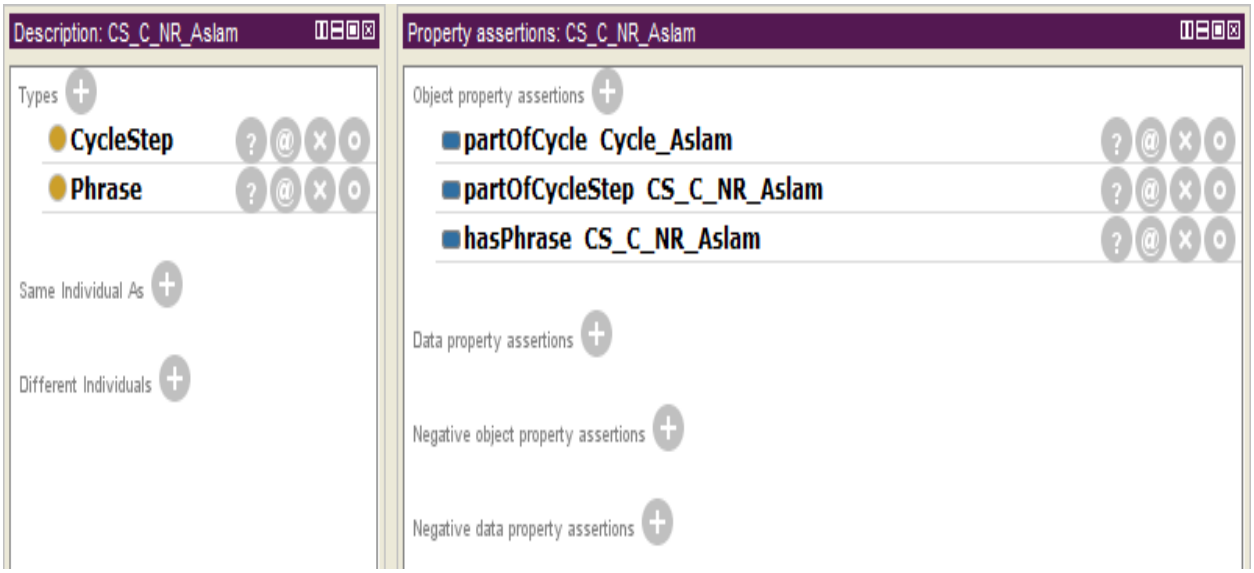


Fig. 12. Object Properties for the Instance CS_C_NR_Aslam.

One more instance ‘Some_Sickness’ have asserted here to complete the one nurse report for a patient that contain a phrase about the disease. It has one object property asserted ‘partOfCycleStep’ with instance ‘NR_Aslam’. It has only one data type property asserted ‘hasWrittenText’ with the datatype ‘string and value ‘Not Really Sick’ as shown in figure 13 below.

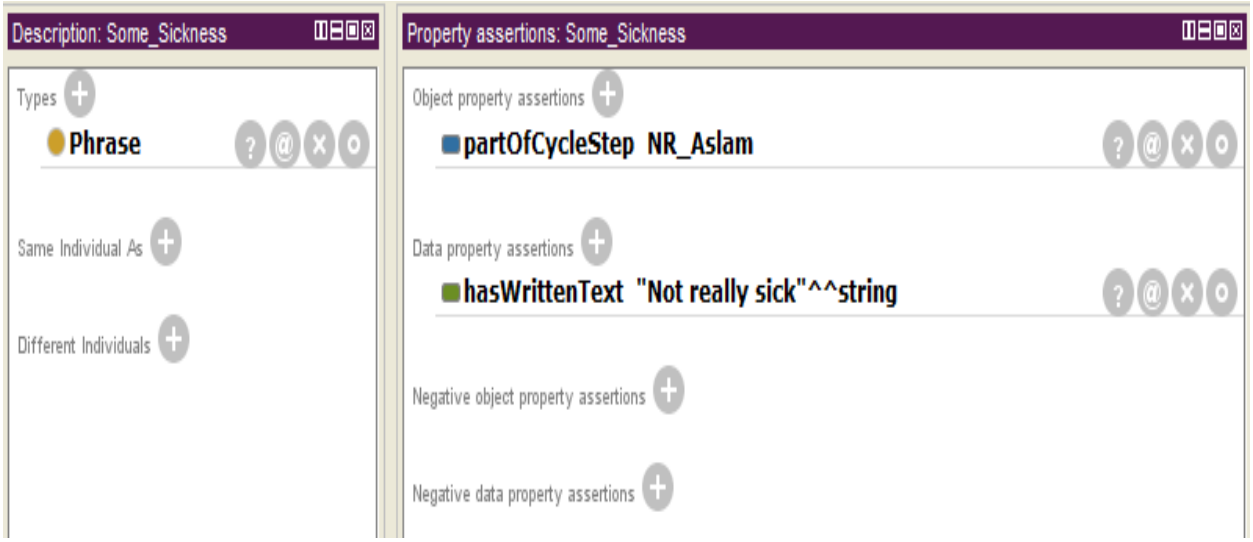


Fig. 13. Object and data type properties for Instance of Some_Sickness.

These all patient instances which we have added into our ontology have the same process like patient ‘Aslam’ its nursing report ‘NR_Aslam’, that contains cycle ‘Cycle_Aslam’ and cycle have some cycle steps as ‘CS_C_NR_Aslam’. To simplify our ontology we just have connected the cycle step directly to the phrase (but there are some cycle steps e.g. Diagnosis, Intervention and outcome, through which a ‘CycleStep’ completed and make a report for patient) that has ‘Some_Sickness’ which have some written text about the patient disease, as shown is figure 13 above. The above said scenario can be more understandable from the RDF statement graph as shown in figure 14 below.

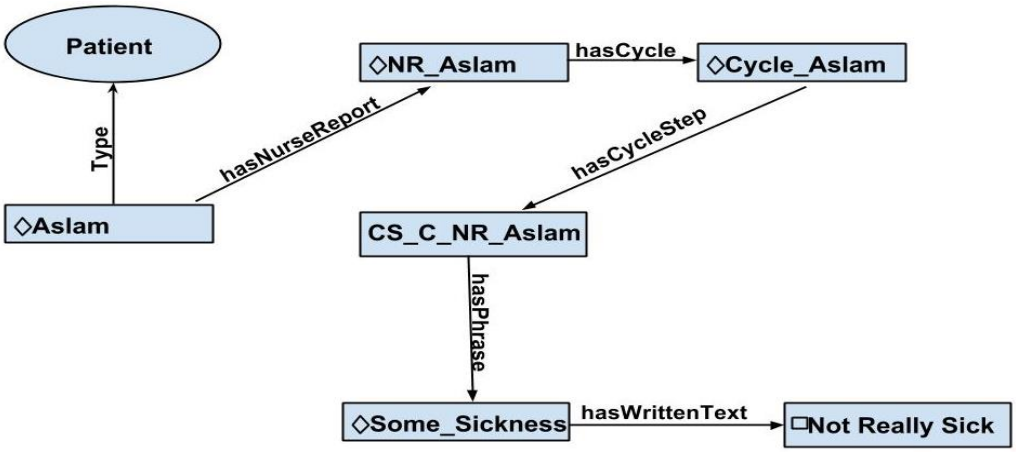


Fig. 14. RDF graph statement for a patient Instance.

The above graph present the instances of only one ‘Patient’, but if you see figure 8 above you will find that we have added total five ‘Patient’ instances that have their own ‘NursingReport’, Cycle’, ‘CycleSteps’ and ‘Phrase’ about specific diseases. All added data into ontology can be seen from figure 15 below.

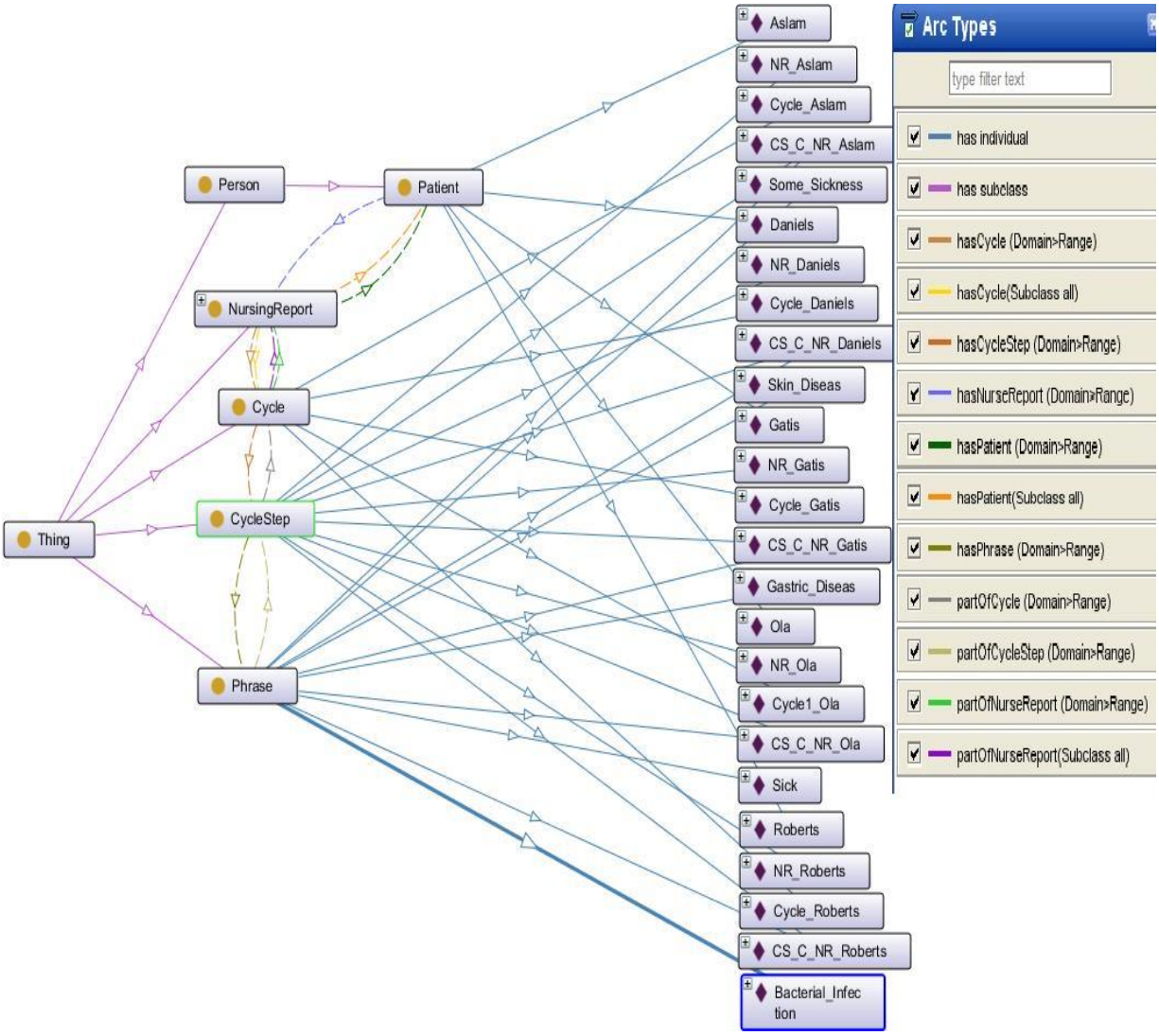


Fig. 15. Graph of complete Ontology.

The above figure shows the complete scenario of Ontology. At the moment we have added five patients their nurse reports, cycles, cycle steps and phrases, but on the prototype prospective more data could be add in this ontology, due to the master thesis we have a limited time so we decided to strict here and the next task we have to make a java application that setup a SPARQL Endpoint to access this all Ontology data over HTTP. The development of java application you will find in next section.

3.3 Implementation of SPARQL Endpoint.

In this implementation section I will try to explain how I have built my java application by using jena to access the data from the above said ontology and then we try to show the results on web browser.

First of all I have created two packages in java in which I have define java classes that holds all the application mechanism as per need for our SPARQL Endpoint solution. The first package is 'com.mkyong.app' (I have given name here mkyong, but it could be any other name as user want to set for their packages) it contains application's core functionality Java classes. The second package I have created is com.mkyong.rest, which contains the web services Java classes, this package is handling all the patient related phrases which we will be queried from the browser. The packages details is shown in figure 16 below.

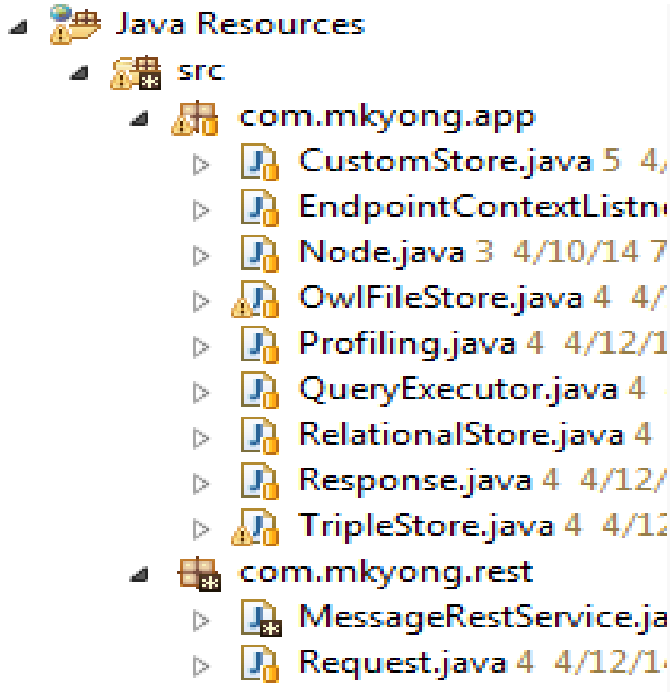


Fig. 16. Java packages with their Classes.

3.3.1 Package Classes for Web Services.

As I have explained before that in package 'com.mkyong.rest' we have setup the web services code logic. If you see in figure 16 above, you will find two classes, 'MessageRestService.java' and 'Request.java'. A restfull web service name as

"MessageRestService" has a method "getPatient" . The implementation on getPatient method is that; take an input (Request) ; execute on databases (i.e. Relational database , Owl file and Triple store) and return the result in Response object. It means that WebServce is responsible to execute given query on Relational database, OWL file and Triple Store, also note the query execution time on different databases and return these execution time as response. As can be seen from the code below.

```
public Response getPatient(Request requestObj) {}
QueryExecutor executor;
            Response response = new Response();
            List<Profiling> profiles = new ArrayList<Profiling>();
            if (requestObj.isCustomQuery()) {
                  executor = new QueryExecutor(requestObj.getQuery());
            } else {
                  executor = new QueryExecutor();
            }

            try {

                  List<Node> map = new ArrayList<Node>();
                  Node rDB = new Node("Relational Database", "");
                  Node oFile = new Node("Owl File","");
                  Node tStore = new Node("Triple Store",  "");

                  long start = System.currentTimeMillis();
                  List<Node> map1 = executor.getFromRelationalStore();
                  long end = System.currentTimeMillis();
                  Node my = new Node("Time (ms)", end - start + "");

                  Profiling time = new Profiling("Relational Database Time", end
- start);
                  profiles.add(time);

                  start = System.currentTimeMillis();
                  List<Node> map2 = executor.getFromOwlFileStore();
                  end = System.currentTimeMillis();
                  Node owl = new Node("Time (ms) ", end - start + "");

                  time = new Profiling("Owl File", end - start);
                  profiles.add(time);

                  start = System.currentTimeMillis();
                  List<Node> map3 = executor.getFromTripleStore();
                  end = System.currentTimeMillis();
                  Node tps = new Node("Time  (ms)", end - start +"" );
                  time = new Profiling("Triple Store", end - start);
                  profiles.add(time);

                  map.add(rDB);
```

```
                map.addAll(map1);
                map.add(my);

                map.add(oFile);
                map.addAll(map2);
                map.add(owl);

                map.add(tStore);
                map.addAll(map3);
                map.add(tps);

                response.setNodes(map);
            } catch (QueryException e) {
                response.setResponseCode("QUERY_ERROR");
            }
            response.setProfiles(profiles);
            return response;
        }
        }
```

'Request.java' class is use to save user given query. The object of this class is pass to Web Service as parameter. This class is POJO (Plain Old Java Object) and have two members 'customQuery' and 'query'. 'customQuery' is a boolean type member, its use for default query or user enterd query flag. 'Query' is a string type member, it is being used for saving the user input query. As it can be seen from code below.

```
public class Request {

    public Request() {

    }

    private boolean customQuery;


    public boolean isCustomQuery() {
        return customQuery;
    }

    public void setCustomQuery(boolean customQuery) {
        this.customQuery = customQuery;
    }

    private String query;

    public String getQuery() {
        return query;
    }

    public void setQuery(String query) {
        this.query = query;
    }

}
```

If we look into the package 'com.mkyong.app' in figure 16 above, in that I have define a class 'Response.java'. This class is used to return response of query. It contain three properties 'responseCode', 'nodes' and 'profiles'.

The ‘responseCode’ property is used to give query execution status, successful or error. The property ‘profiles’ function is to store the query executaion time and its type. Anthoer object property is ‘nodes’ which is used to store key value of query result from databse. As in our case I am using three type of databases owl, triple store and relational database. Resposnse code is use to retrun status of query execution on these databases. Either query run successfully or have some error. On the ***behafe*** of responese code we display messge to end user. The scheme of this class can be seen below.

```
public Response() {

}

private String responseCode;

private List<Node> nodes;
private List<Profiling> profiles;

public List<Profiling> getProfiles() {
    return profiles;
}

public void setProfiles(List<Profiling> profiles) {
    this.profiles = profiles;
}

public List<Node> getNodes() {
    return nodes;
}

public void setNodes(List<Node> nodes) {
    this.nodes = nodes;
}

public String getResponseCode() {
    return responseCode;
}

public void setResponseCode(String responseCode) {
    this.responseCode = responseCode;
}
```

3.3.2 Package Classes for Application.

There is another class in package ‘com.mkyong.app’ is ‘EndpointContextListner’ class. The purpose of this class is to initialize the Triple Store and database values. Also it have init parameters of relational database connection, owl file location and

triple store location. These all parameters are used for application initialization. As it can be seen from code below.

```
public class EndpointContextListner implements ServletContextListener {
    public static final String DB_LOCATION = "H:\\Lecture Notes\\Master Thesis\\Endpoint\\ontology\\nurse";
    public static final String OWL_LOCATION = "H:\\Lecture Notes\\Master Thesis\\Endpoint\\ontology\\v3_NursingReport.owl";
    public static final String MYSQL_DRIVER = "jdbc:mysql://localhost/owl";
    public static final String DB_USER = "root";
    public static final String DB_PSW = "";

    @Override
    public void contextDestroyed(ServletContextEvent event) {

    }

    @Override
    public void contextInitialized(ServletContextEvent event) {
        try {
            new TripleStore().setupStore(OWL_LOCATION, DB_LOCATION);

            new RelationalStore().setupStore();

        } catch (Exception e) {
            e.printStackTrace();

        }
```

From the code above you can see there is class type of ‘ServletContextListener’ that contains the basic information. When server starts it read this type of class for initialization. Which means that it setups Triple store and Relational Databases on start-up if it is not exist there before.

The class ‘CustomStore’ is used for execute Query on different models (i.e. on relational database, owl file and triple store models). This is an abstract class and have abstract method “public abstract void setupStore() throws Exception;” in new line.

The child classes of custom store are ‘TripleStore’, ‘RelationalStore’ and ‘OwlFileStore’. These all classes override abstract method to setup the store.

Customer store also have another method ‘getDefaultModel’ method signature, this method also override by child classes to return models. The complete code scheme of this class is shown below.

```
public abstract class CustomStore {

    public List<Node> executeQuery(String queryString, Model model) {
        QueryExecution qe = QueryExecutionFactory.create(queryString, model);

        List<Node> nodes = new ArrayList<Node>();

        try {
            com.hp.hpl.jena.query.ResultSet results = qe.execSelect();
            List<String> vars = results.getResultVars();
            while (results.hasNext()) {
                QuerySolution binding = results.nextSolution();
                for (String var : vars) {

                    String node = binding.getLiteral(var).getString();
                    nodes.add(new Node(var, node));

                }

            }
        } catch (QueryException e) {
            throw new QueryException();
        } finally {
            qe.close();
        }

        return nodes;
    }

    public Model getDefaultModel() {

        return null;
    }

    public abstract void setupStore() throws Exception;
}
```

As have said above that there are some child classes of ‘CustomStore’ class, which are ‘OwlFileStore’, ‘TripleStore’ and ‘RelationalStore’. First I will describe here the ‘OwlFileStore’ class. This class use to setup Owl file model as shown in code below.

```
public class OwlFileStore extends CustomStore {

    public Model getDefaultModel() {
        Model model = ModelFactory.createDefaultModel();
        try {
            model.read(
                    new FileInputStream(EndpointContextListner.OWL_LOCATION),
                    null, "TTL");
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return model;
    }

    @Override
    public void setupStore() throws Exception {

    }
}
```

To understand this 'OwlFileStore' scenario, that how it is providing response upon the browser request we have created a sequence diagram as shown in figure 17 below.

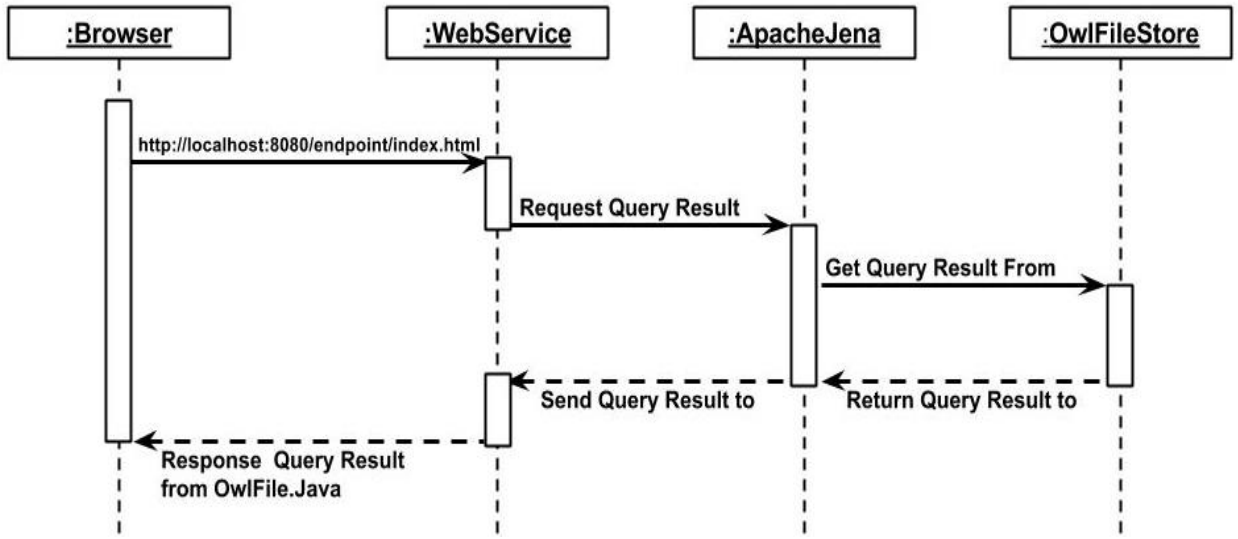


Fig. 17. Sequence Diagram for Response from 'OwlFileStore' class.

The class 'TripleStore' is also extends from the class 'CustomStore'. This class used to set triple store model. TripleStore class read OWL file and convert OWL file into Triple Database and write on disk. This class also provide Model of Triple Database. We use model to run query on it. Model is a set of Statements. Methods are provided for creating resources, properties and literals and the Statements which link them, for adding statements to and removing them from a model, for querying a model and set operations for combining models [61]. The set model scheme can be seen in code below.

```
public class TripleStore extends CustomStore {

    public void setupStore(String input, String output)
            throws FileNotFoundException {
        Dataset dataset = TDBFactory.createDataset(output);
        dataset.begin(ReadWrite.WRITE);
        Model tdb = dataset.getDefaultModel();

        tdb.read(new FileInputStream(input), null, "TTL");

        tdb.close();
        dataset.commit();
        dataset.end();
    }

    public Model getDefaultModel() {

        Dataset dataset = TDBFactory
                .createDataset(EndpointContextListner.DB_LOCATION);
        dataset.begin(ReadWrite.READ);
        return dataset.getDefaultModel();
    }

    @Override
    public void setupStore() throws Exception {

    }
}
```

To define the above 'TripleStore' class code, I have used here 'SetupStore' method for convert OWL file to Triplet Store. It will take two parameter one is OWL file location and second for Triplet Store output location. Dataset 'dataset = TDBFactory.*createDataset*(output);' output is Triple Store location. 'TDBFactory' class use here for creating objects datasets backed by storage. 'CreateDataSet' method used for create or connect dataset. 'DataSet' a collection of named graphs and a background graph. dataset.begin(ReadWrite.***WRITE***); initialize to write mode. For model we used 'tdb.read(new FileInputStream(input), null, "TTL");' reading form OWL file, this parameter take 3 input, 1st is OWL file location, 2nd is the base uri to be used, for our case I set here 'null'. This is because when converting relative URI's to absolute URI's. (Resolving relative URIs and fragment IDs is done by prepending the base URI to the relative URI/fragment.) If there are no relative URIs in the source, this argument may safely be `null`. If the base is the empty string, then relative URIs *will be retained in the model* [61], and the 3rd thing is done by this is add RDF statements represented in language to the model in our case it is 'TTL'. Predefined values for `language` are "RDF/XML", "N-TRIPLE", "TURTLE" (or "TTL") and "N3". `Null` represents the default language, "RDF/XML". "RDF/XML-ABBREV" is a synonym for "RDF/XML"[61]. The response to the web services from the 'TripleStore' class can be understand from the following figure.

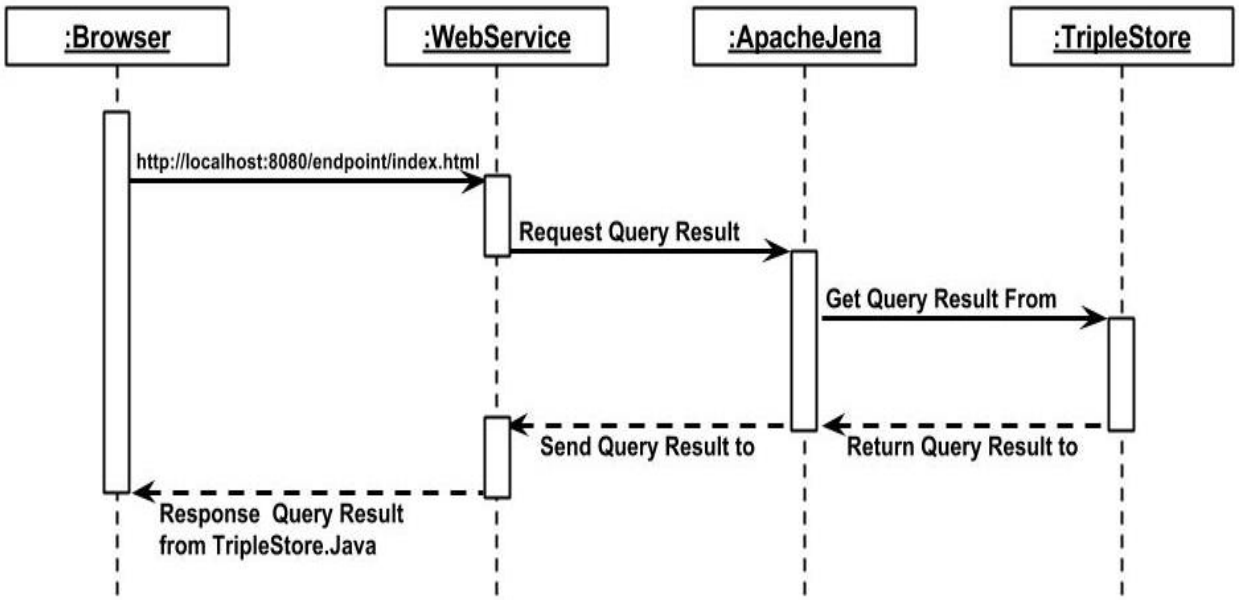
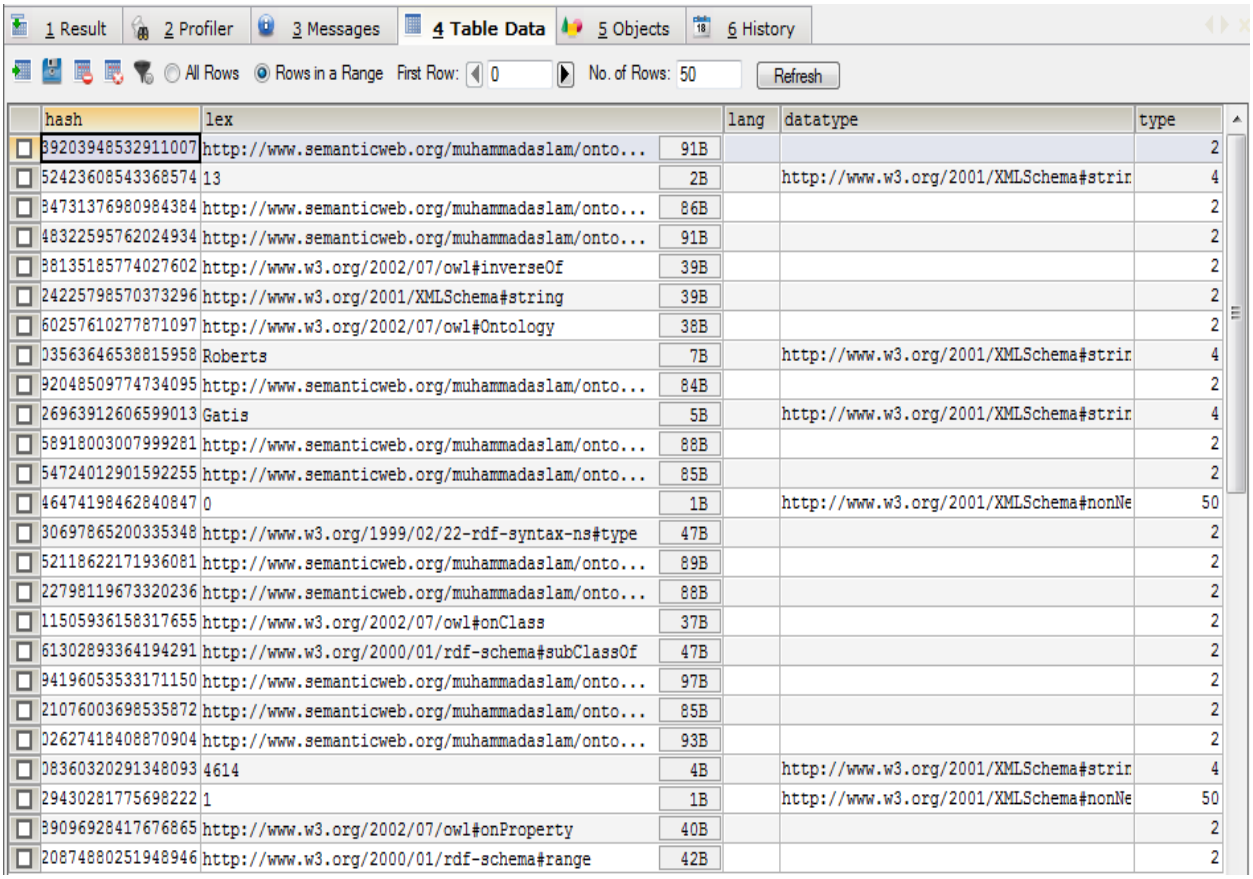


Fig. 18. Sequence Diagram for class 'TripleStore' response.

Another class ‘RelationalStore’ also extends from the class ‘CustomStore’. This class use to setup relational database connectivity and model initialization. For setup database connection see the code below.

```
public class RelationalStore extends CustomStore {

    private static StoreDesc storeDesc = new StoreDesc(
            LayoutType.LayoutTripleNodesHash, DatabaseType.MySQL);;
    private static SDBConnection conn;
    private static Store store;
    static {
        JDBC.loadDriverMySQL();
        conn = new SDBConnection(EndpointContextListner.MYSQL_DRIVER,
                EndpointContextListner.DB_USER, EndpointContextListner.DB_PSW);
        store = SDBFactory.connectStore(conn, storeDesc);
    }

    public Model getDefaultModel() {

        Model model = SDBFactory.connectDefaultModel(store);

        return model;

    }

    @Override
    public void setupStore() throws Exception {

        store.getTableFormatter().create();
        Model model = SDBFactory.connectDefaultModel(store);

        model.read(new FileInputStream(EndpointContextListner.OWL_LOCATION),
                null, "TTL");

    }
```

To define the code here we used ‘StoreDesc’ store description identifies which storage layout is being used. ‘SDBConnection’: this class is being used to connect Relational Database. ‘Store**’:** Jena loads and queries data based on the unit of a Store. The Store object has all the information for formatting, loading and accessing database. ‘SDBFactory’: it connects to Relational Database and creates the necessary tables, Such as prefixes, nodes and triples as shown below.

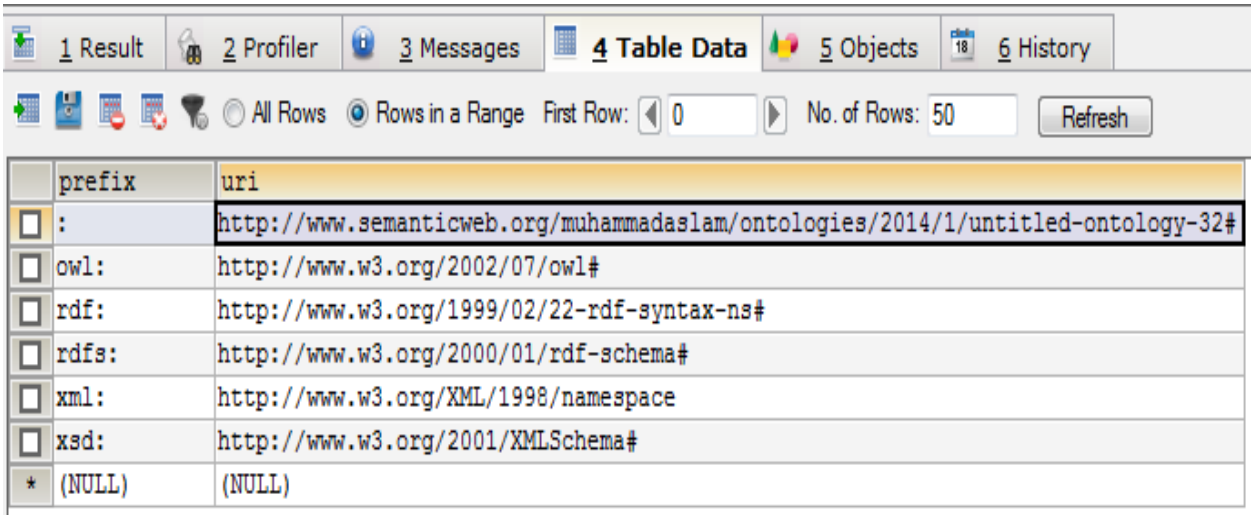


1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Objects | 6 History

All Rows | Rows in a Range | First Row: 0 | No. of Rows: 50 | Refresh

hash	lex		lang	datatype	type
39203948532911007	http://www.semanticweb.org/muhammadaslam/onto...	91B			2
52423608543368574	13	2B		http://www.w3.org/2001/XMLSchema#strir	4
34731376980984384	http://www.semanticweb.org/muhammadaslam/onto...	86B			2
48322595762024934	http://www.semanticweb.org/muhammadaslam/onto...	91B			2
38135185774027602	http://www.w3.org/2002/07/owl#inverseOf	39B			2
24225798570373296	http://www.w3.org/2001/XMLSchema#string	39B			2
60257610277871097	http://www.w3.org/2002/07/owl#Ontology	38B			2
03563646538815958	Roberts	7B		http://www.w3.org/2001/XMLSchema#strir	4
92048509774734095	http://www.semanticweb.org/muhammadaslam/onto...	84B			2
26963912606599013	Gatis	5B		http://www.w3.org/2001/XMLSchema#strir	4
58918003007999281	http://www.semanticweb.org/muhammadaslam/onto...	88B			2
54724012901592255	http://www.semanticweb.org/muhammadaslam/onto...	85B			2
46474198462840847	0	1B		http://www.w3.org/2001/XMLSchema#nonNe	50
30697865200335348	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	47B			2
52118622171936081	http://www.semanticweb.org/muhammadaslam/onto...	89B			2
22798119673320236	http://www.semanticweb.org/muhammadaslam/onto...	88B			2
11505936158317655	http://www.w3.org/2002/07/owl#onClass	37B			2
61302893364194291	http://www.w3.org/2000/01/rdf-schema#subClassOf	47B			2
94196053533171150	http://www.semanticweb.org/muhammadaslam/onto...	97B			2
21076003698535872	http://www.semanticweb.org/muhammadaslam/onto...	85B			2
02627418408870904	http://www.semanticweb.org/muhammadaslam/onto...	93B			2
08360320291348093	4614	4B		http://www.w3.org/2001/XMLSchema#strir	4
29430281775698222	1	1B		http://www.w3.org/2001/XMLSchema#nonNe	50
39096928417676865	http://www.w3.org/2002/07/owl#onProperty	40B			2
20874880251948946	http://www.w3.org/2000/01/rdf-schema#range	42B			2

Fig. 19. Relational database Table for Nodes.

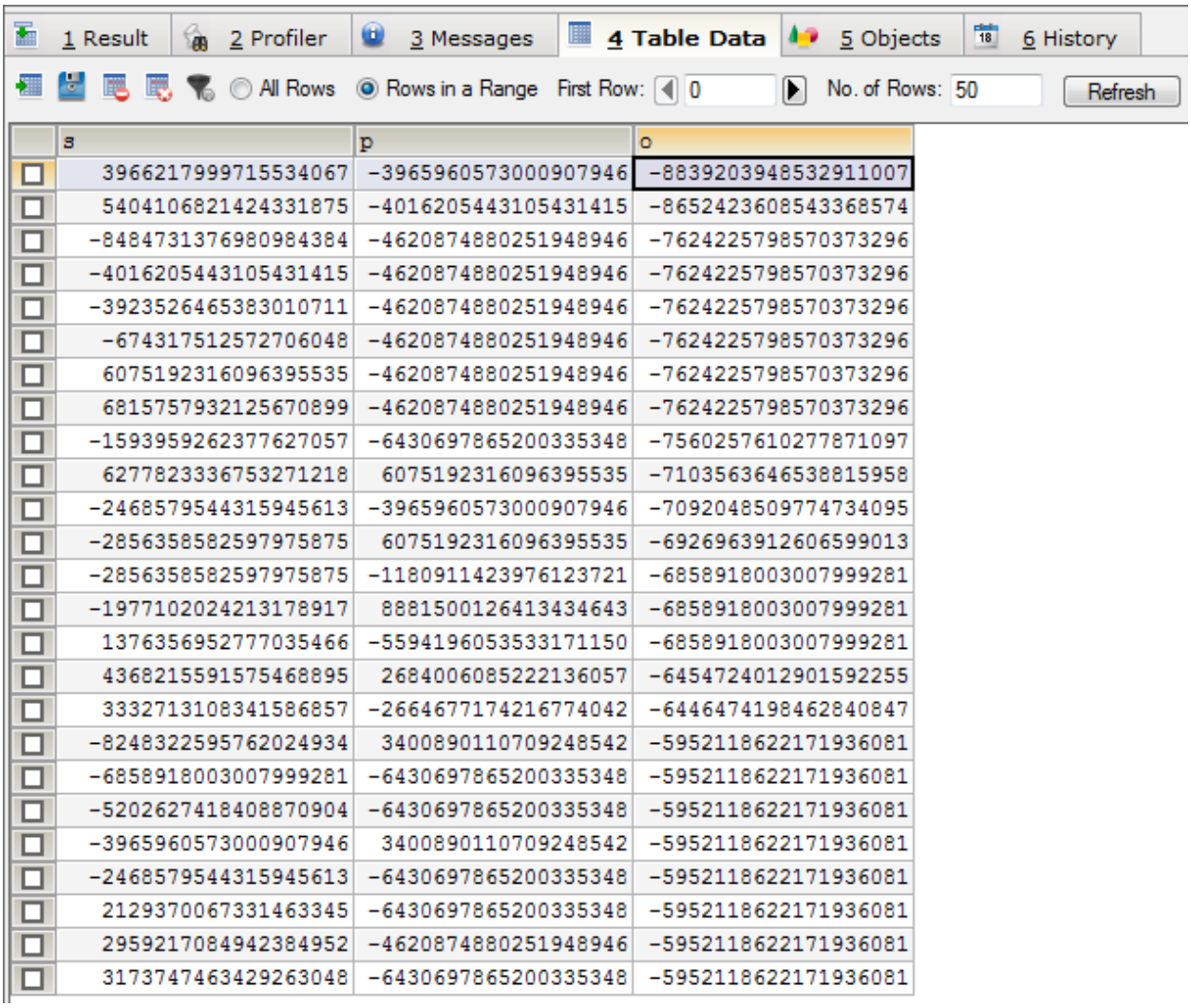


1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Objects | 6 History

All Rows | Rows in a Range | First Row: 0 | No. of Rows: 50 | Refresh

prefix	uri
:	http://www.semanticweb.org/muhammadaslam/ontologies/2014/1/untitled-ontology-32#
owl:	http://www.w3.org/2002/07/owl#
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
xml:	http://www.w3.org/XML/1998/namespace
xsd:	http://www.w3.org/2001/XMLSchema#
* (NULL)	(NULL)

Fig. 20. Relational database Table for Prefixes.



s	p	o
3966217999715534067	-3965960573000907946	-8839203948532911007
5404106821424331875	-4016205443105431415	-8652423608543368574
-8484731376980984384	-4620874880251948946	-7624225798570373296
-4016205443105431415	-4620874880251948946	-7624225798570373296
-3923526465383010711	-4620874880251948946	-7624225798570373296
-674317512572706048	-4620874880251948946	-7624225798570373296
6075192316096395535	-4620874880251948946	-7624225798570373296
6815757932125670899	-4620874880251948946	-7624225798570373296
-1593959262377627057	-6430697865200335348	-7560257610277871097
6277823336753271218	6075192316096395535	-7103563646538815958
-2468579544315945613	-3965960573000907946	-7092048509774734095
-2856358582597975875	6075192316096395535	-6926963912606599013
-2856358582597975875	-1180911423976123721	-6858918003007999281
-1977102024213178917	8881500126413434643	-6858918003007999281
1376356952777035466	-5594196053533171150	-6858918003007999281
4368215591575468895	2684006085222136057	-6454724012901592255
3332713108341586857	-2664677174216774042	-6446474198462840847
-8248322595762024934	3400890110709248542	-5952118622171936081
-6858918003007999281	-6430697865200335348	-5952118622171936081
-5202627418408870904	-6430697865200335348	-5952118622171936081
-3965960573000907946	3400890110709248542	-5952118622171936081
-2468579544315945613	-6430697865200335348	-5952118622171936081
2129370067331463345	-6430697865200335348	-5952118622171936081
2959217084942384952	-4620874880251948946	-5952118622171936081
3173747463429263048	-6430697865200335348	-5952118622171936081

Fig. 21. Relational database Table for Triples.

For model initialization is the same as define in ‘TripleStore’ class.

To explain this ‘RelationalStore’ class scenario that how it respond upon the query request from the web browser I have made a sequence diagram as shown in figure 22 below.

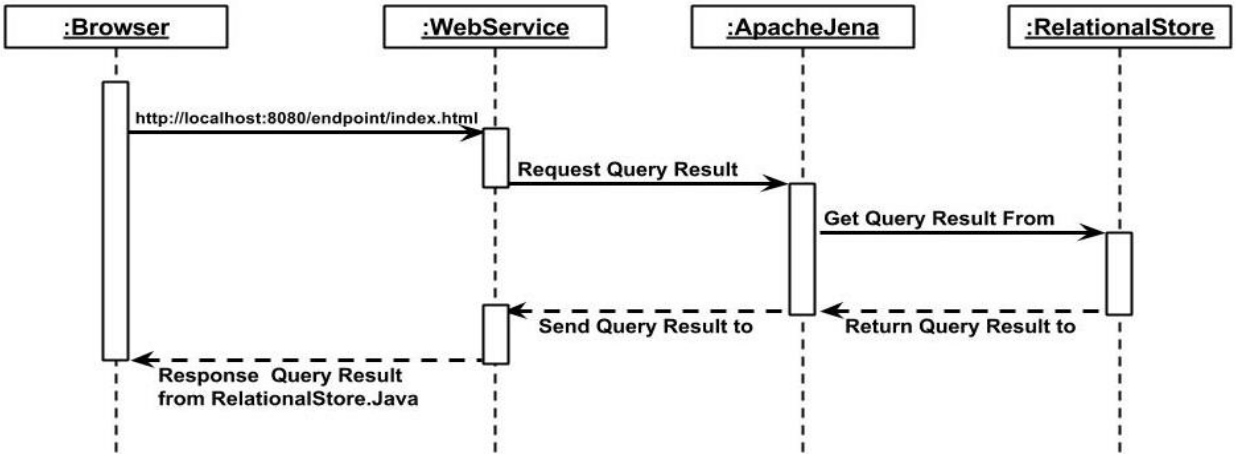


Fig. 22. Sequence Diagram for 'RelationalStore' response.

This whole application of web service, jena, browser and the results from the databases can be more easily understand by the figure 23 below.

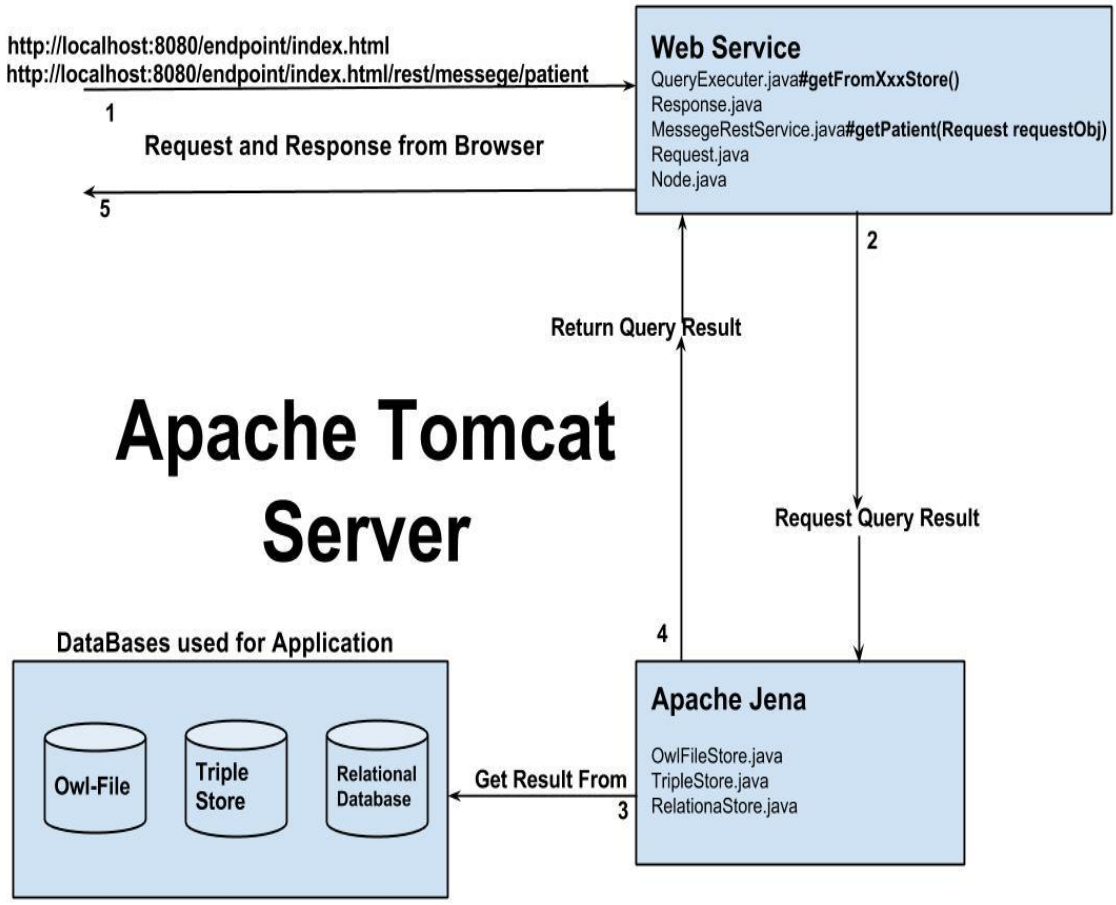


Fig. 23. Flow Chart of SPARQL Endpoint Application.

It is clearly seen from the flow chart above that we are using the apache tomcat web server to run that SPARQL Endpoint application. When there is a request for a query from the web browser, it goes to the web services, so the web services sends request for the query result to apache jena through some java objects (java objects can be seen in web services section). Apache jena get the requested query results from databases (i.e. from OwlFile, TripleStore, and RelationalStore) and return the query results to the web services. Here when web services received the query results it is in the form of Java objects and web browser cannot understand the Java objects. So that here Web Service (Rest Easy) come to the rescue it converts that Java object into JSON. So the web browser will display the requested query results. The results will be discuss in next section of this report.

3.4 Results on SPARQL Endpoint.

To represent the results here we going to test our SPARQL Endpoint application by doing some SPARQL queries regarding our ontology.

First of all to run the application we need to start our web server, in our case we are using Apache Tomcat Web Server, which we have installed with Java IDE, to run this we just need to go in server tab in Java IDE and click on the green button place on the right side of the bar, and we will have our server ready to run the application.

Once web server get started we just open the web browser and enter link, as we have set link for our application is 'http://localhost:8080/endpoint/index.html'. Once we enter that link in web browser (any web browser can be used here) a page will be appeared as shown in figure below.

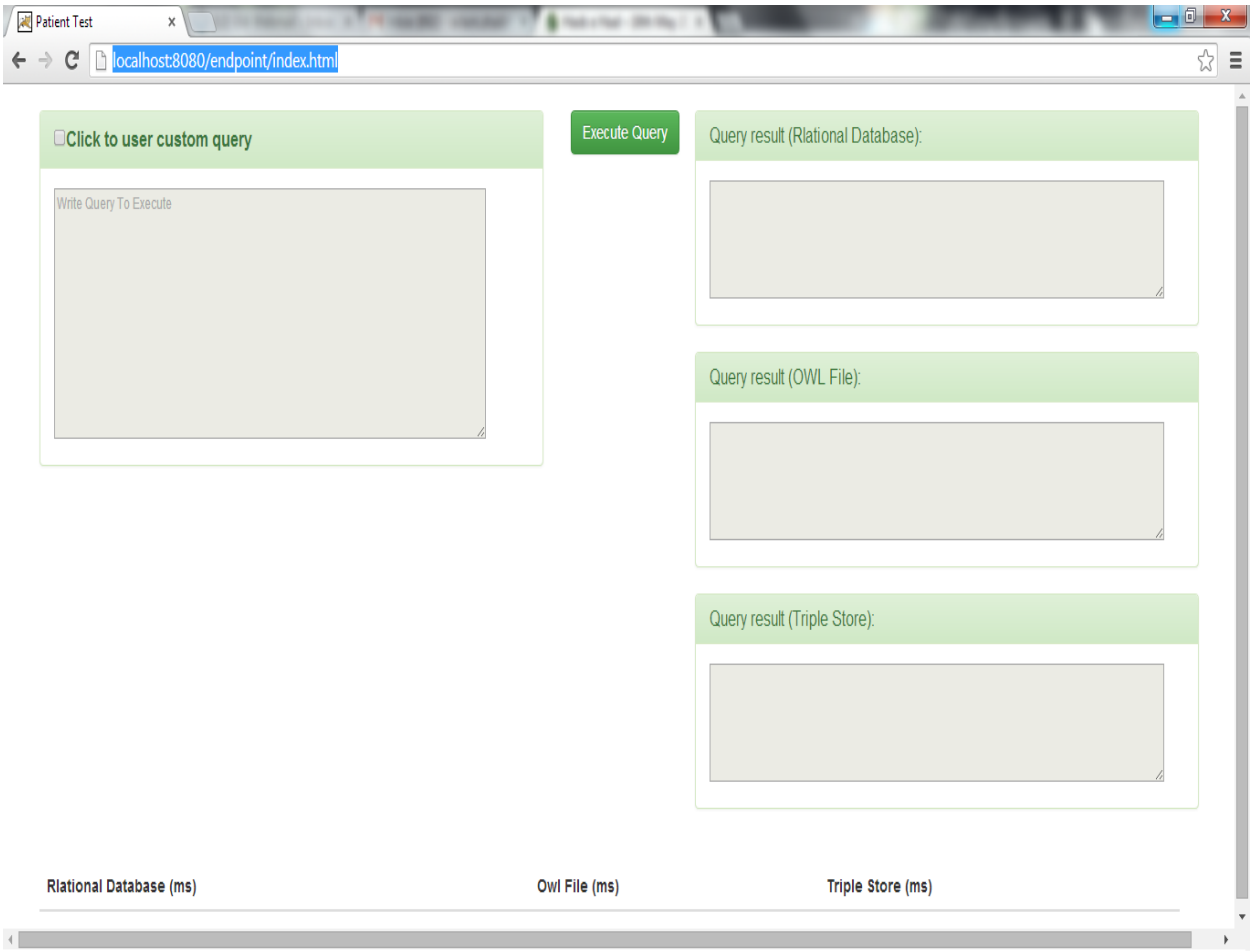


Fig. 24. Graphical User Interface (GUI) for SPARQL Endpoint Application.

As we can see from the figure above that there is a panel on right side of the page to take any query from the user, which should be related to the patient phrase, we have

added here a button ‘Execute Query’, when we hit this button it will call for the query results from the database. As we are using here three types of databases so we have added three panels as shown on the right side of the figure above to present database results separately (i.e. Query Result from Relational Store, Owl File and Triple Store). Here we have added query execution time from all the databases to check their efficiency and performance as shown in bottom of the page in figure 21 above.
Here we try to run some SPARQL queries related to our ontology in term of patient phrases on our application and will see the results.

3.4.1 SPARQL Queries and Their Results.

In this section we will try to execute some SPARQL Queries which we have made according to our ontology data and examine the queries results on the basis of performance and efficiency of our databases. Below you will find the queries and their results.

Query to print all patients names.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX :<http://www.semanticweb.org/muhammadaslam/ontologies/2014/1/untitled-ontology-32#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?patientName
WHERE { ?Patient :hasFamilyName ?patientName }
```

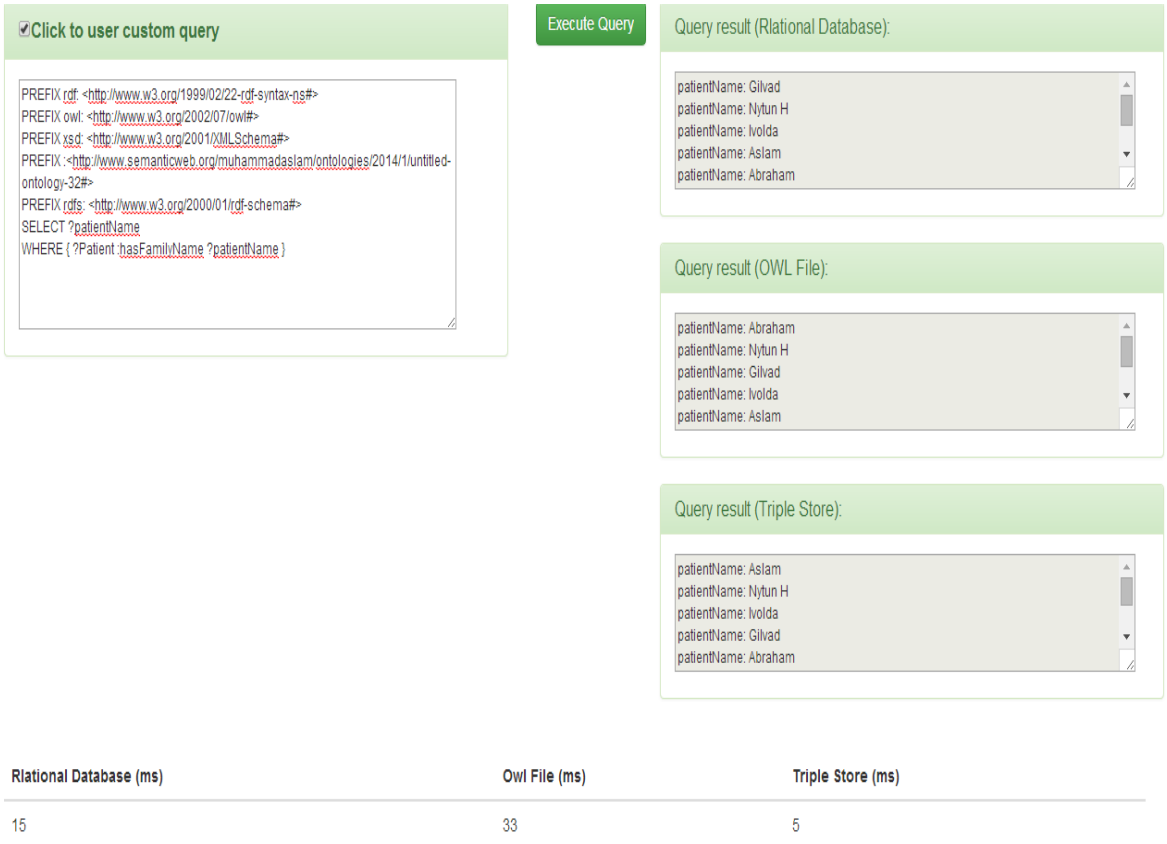


Fig. 25. SPARQL Query Results for print All Patient Names.

Query to Print all Patients Given Name.

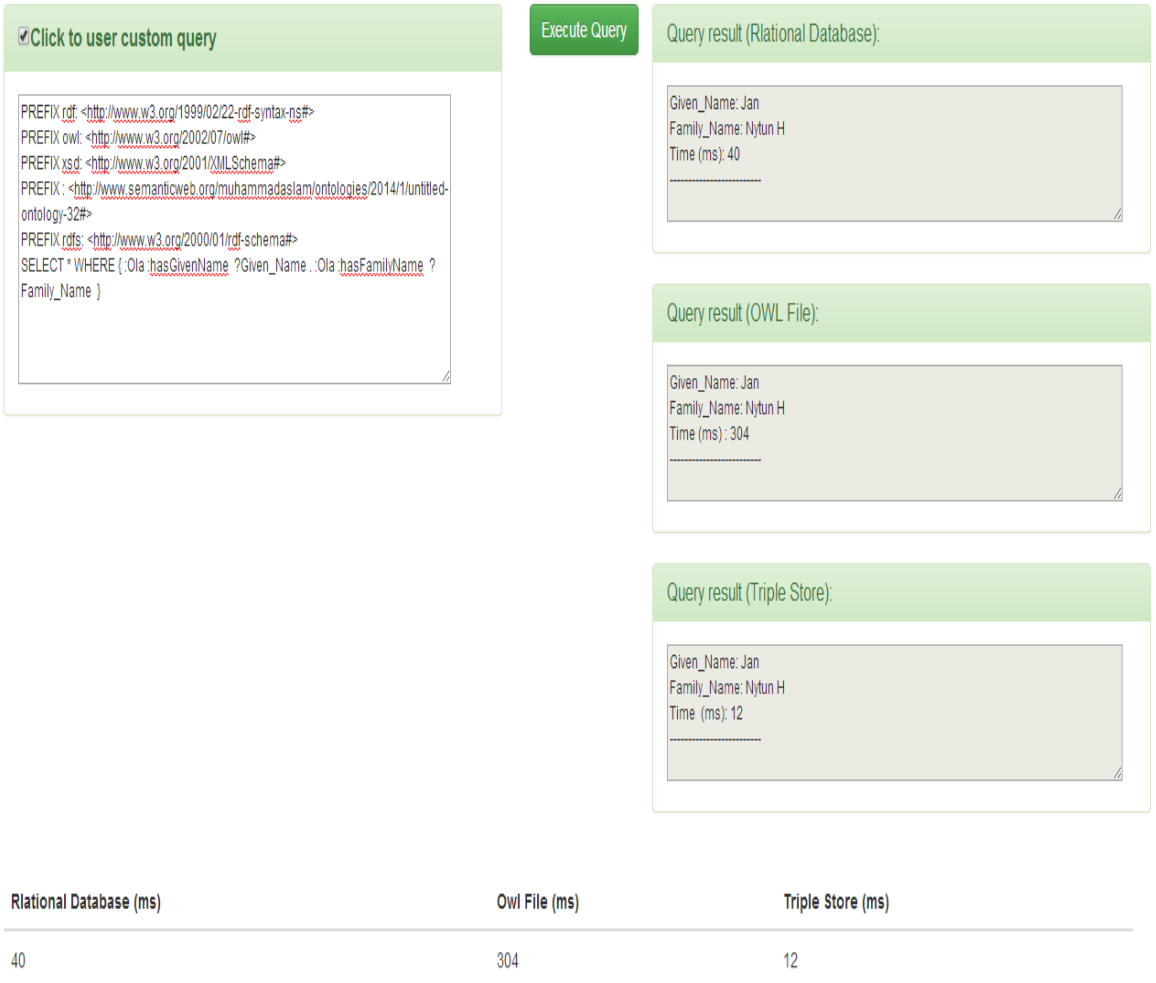
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX
:<http://www.semanticweb.org/muhammadaslam/ontologies/2014/1/untitled-ontology-32#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?givenName
WHERE { ?Patient :hasGivenName ?givenName }
```

Rlational Database (ms)	Owl File (ms)	Triple Store (ms)
27	100	25

Fig. 26. SPARQL Query Results for Print all Given Names of Patients.

SPARQL Query to print One Patient Given Name and Family Name whose name is Ola.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://www.semanticweb.org/muhammadaslam/ontologies/2014/1/untitled-ontology-32#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE { :Ola :hasGivenName  ?Given_Name . :Ola :hasFamilyName ?Family_Name  }
```

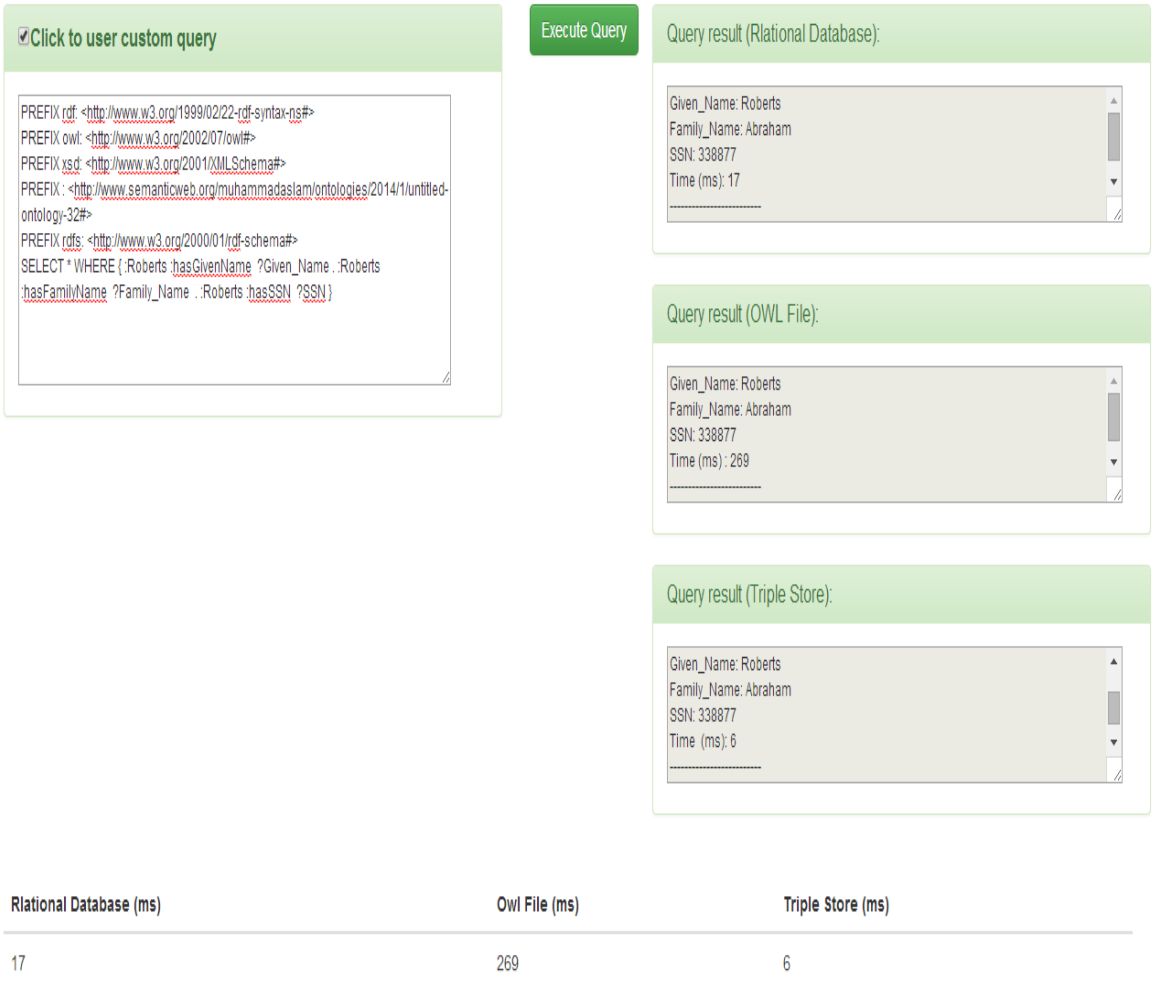


Rlational Database (ms)	Owl File (ms)	Triple Store (ms)
40	304	12

Fig. 27. SPARQL Query to Print Patient Given Name and Family Name whose name is Ola.

SPARQL Query to print one Patient Personal Information.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://www.semanticweb.org/muhammadaslam/ontologies/2014/1/untitled-ontology-32#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE { :Roberts :hasGivenName ?Given_Name . :Roberts :hasFamilyName ?Family_Name . :Roberts :hasSSN ?SSN }
```

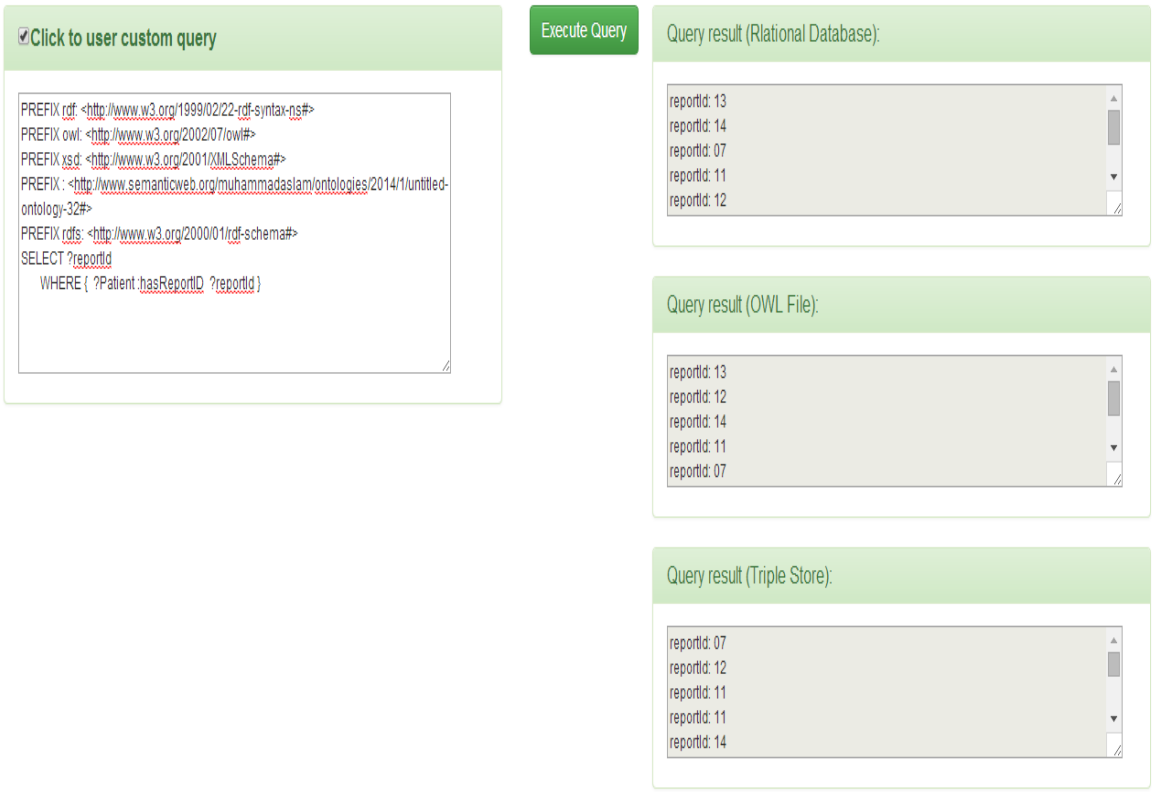


Rlational Database (ms)	Owl File (ms)	Triple Store (ms)
17	269	6

Fig. 28. SPARQL Query Result for Print one Patient Complete Information.

SPARQL Query to Print all Patients Report IDs.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX                                                                    :
<http://www.semanticweb.org/muhammadaslam/ontologies/2014/1/untitled-
ontology-32#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?reportId
      WHERE {  ?Patient :hasReportID  ?reportId }
```



Rlational Database (ms)	Owl File (ms)	Triple Store (ms)
52	69	26

Fig. 29. SPARQL Query Results for print all Patients Report IDs.

SPARQL Query to print SSN Numbers of all Patients.

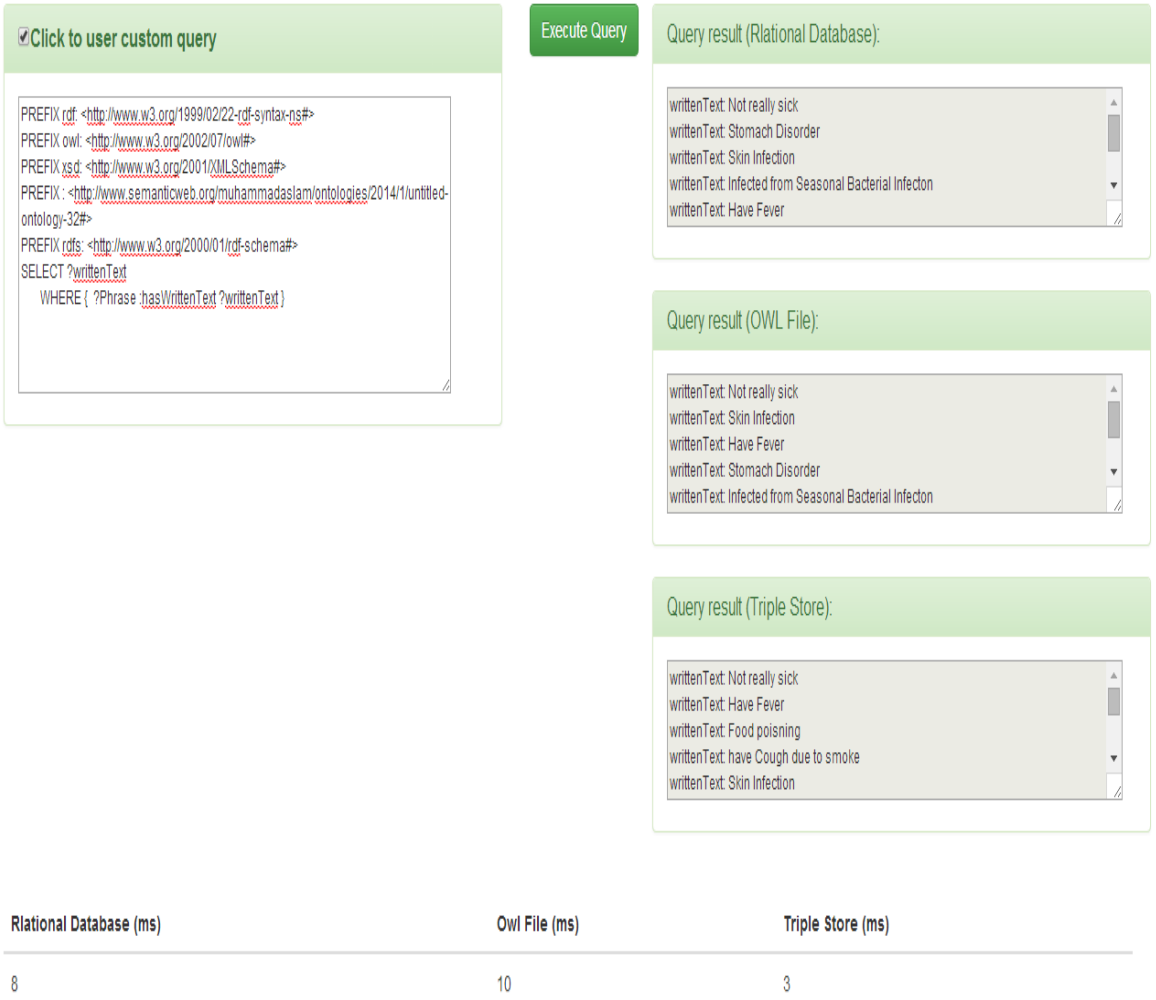
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX                                                                    :
<http://www.semanticweb.org/muhammadaslam/ontologies/2014/1/untitled-
ontology-32#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?patientSSN
      WHERE { ?Patient :hasSSN ?patientSSN }
```

Rlational Database (ms)	Owl File (ms)	Triple Store (ms)
16	121	4

Fig. 30. SPARQL Query Results to print all SSN Numbers of Patients.

SPARQL Query to print Patients Disease Phrases.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX                                                                    :
<http://www.semanticweb.org/muhammadaslam/ontologies/2014/1/untitled-
ontology-32#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?writtenText
      WHERE {  ?Phrase :hasWrittenText ?writtenText }
```



Rlational Database (ms)	Owl File (ms)	Triple Store (ms)
8	10	3

Fig. 31. SPARQL Query Results for print all Patients Diseases Phrase.

As discuss earlier that we have added some data in our ontology for checking the SPARQL Endpoint results, but on prototype bases there can be add more data in future. On the basis of that data we have made some SPARQL Queries and run them on our application as shown in results figure above. After running these queries and having results of each query separately we analyses that the Triple Store is more efficient among the other data bases used here to provide results. The performance of databases on performed SPARQL Queries in Milliseconds (ms) can be seen from the table given below.

SPARQL Query No.	Relational Database (Jena SDB) (ms)	Owl File (ms)	Triple Store(Jena TDB) (ms)
01	15	33	05
02	27	100	25
03	40	302	12
04	17	269	06
05	52	69	26
06	16	121	04
07	08	10	03

Table 1. Comparison of Databases in term of performance and efficiency.

The query 01 is most similar to query 02 but if we see the results we can easily see that query 02 takes longer time than the 01 query from all databases response. From the table above it is evident that each query we have performed above Triple Store have given the quickest response as compared to other database approaches. Therefore we can say that the Triple Store is best solution for using database for a semantic web application in term of providing results.

3.5 Comparison of other Native Triple Stores.

In previous section we have presented the performance of triple stores and relational databases with our developed application. In this section we will provide a brief overview of other native triple stores being evaluated [62] before similar to our project. Triple stores can be divided into 3 broad categories – in-memory, native, non-memory non-native - based on the architecture of their execution. In-memory, triple stores, store the RDF graph in main memory. Storing everything in-memory cannot be a serious method for storing extremely large masses of data [62]. However, they can act as useful benchmark and can be utilized for executing certain operations like caching data from remote sites or for performing reasoning. Most of the in-memory stores have efficient reasoners available and can help figure out the problem of performing reasoning in persistent RDF stores, which otherwise can be very difficult to do. A second, now a dominant category of triple stores is the native triple stores which provide persistent storage with their own implementation of the databases, e.g., Virtuoso, Mulgara, AllegroGraph, Garlik JXT. The third category of triple stores, the non-native non memory triples stores are set up to run on third party databases, i.e., Jena SDB which can be coupled with almost all relational databases like MySQL, PostsgreSQL, Oracle. Recently native triple stores due to their superior load times and ability to be optimized for RDF have gained popularity [62].

In the following part we give a brief overview of the triple stores being evaluated. The following sections describe the dataset being used, the results and the conclusion on the results.

3.5.1 Data Sets

To facilitate the evaluation of the triple stores we [62] use 2 datasets. UNIPROT dataset is a publicly available dataset which contains information on proteins. It is a central repository of protein sequence and function created by joining UniProt Knowledgebase (UniProtKB), the UniProt Reference Clusters (UniRef), and the UniProt Archive (UniParc). Since our primary use case is to evaluate feasibility of a triple store as backend for Bioportal, our second dataset comprises ontologies from Bioportal [62].

The two datasets serve different purposes and help ensure that the triple stores are compared on a number of different axis. We use two datasets from UNIPROT, the first has 1 Million triples and the second 10 Million. Even though 10 million is a small number in the world of datasets, it helps quantify scalability of the RDF triple stores [62].

3.5.2 Ontologies from Bioportal

The ontologies from bioportal help evaluate feasibility of using a triple store as a backend for Bioportal. In addition to this, most of the use-cases in bioportal require some level of inferencing over the ontologies. Thus using this data helps measure the support for inferencing in the RDF triple stores [62].

If a triple store is used as backend of the bioportal the queries need to be performed as they are by the current implementation of bioportal. Common name of the term is used in the current queries. However, when used with triple store as the backend, the namespace to which the term belongs needs to be made available for execution of queries. Since currently ontologies in the bioportal do not have a common namespace, the namespaces of each of the ontology needs to be extracted and stored as the ontology is being loaded into the triple store. This preprocessing step helps measure the effectiveness of the API with the triple store [62].

Since the user only supplies the common name of the term to query, the SPARQL query with the namespace of the term needs to be created. The namespaces extracted in the above preprocessing are used in this step. Since the namespace to which the term originally belongs is not known, combination with all the namespaces extracted is performed. This causes a single query to fan-out by a factor of ~100 and puts strain on the triple store and helps measure the ability of the triple store to perform a large number of queries in a short duration. More information about this is available in the methodology section [62].

3.5.3 Evaluation Methodology

All the triple stores were accessed through their suggested APIs i.e. Jena SDB through the Jena API, Sesame Native and Mulgara through Sesame and Virtuoso

through its connection API. Even though the UNIPROT data can be loaded into the triple store without the utilization of an API, we [62] preferred testing, loading using an API since it was a necessity for the ontologies of bioportal. As already noted, the ontologies of bioportal don't have a common namespace. The namespaces associated with the ontologies are extracted before loading them. This enables us to transform a user query into 'common name' into a SPARQL query with complete namespaces. Figure 30 illustrates the process of loading the data into a triple store [62].

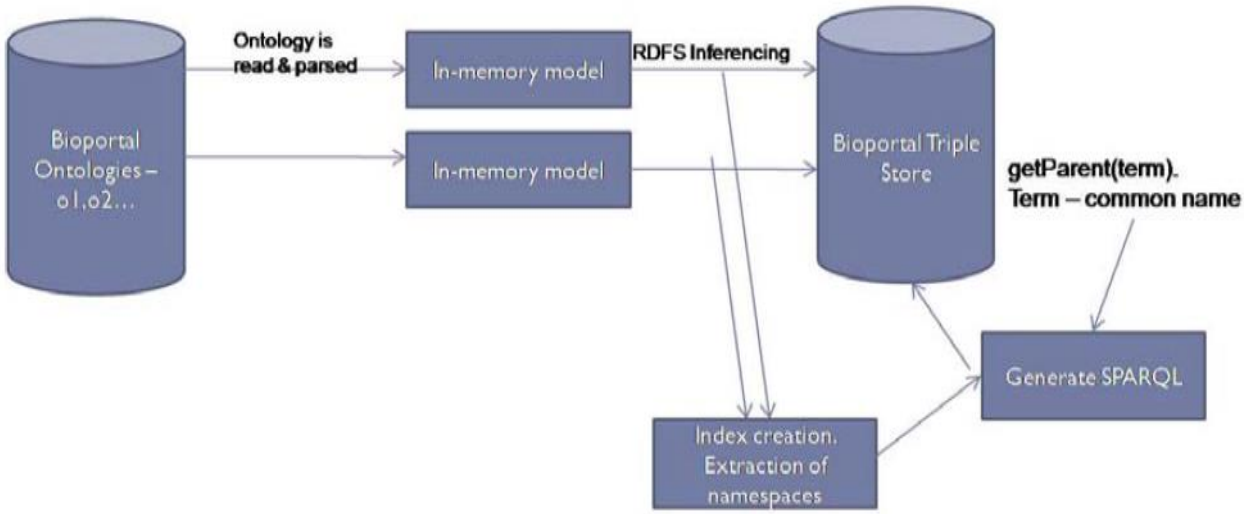


Fig. 32. Process of loading Data into Triple Stores [62].

3.5.4 Inferencing Approach.

A bioportal installation based on triple stores requires inferencing at least to the level of RDFS i.e. In [62] they need an inferencing mechanism that performs transitive closure on the sub-property and sub-class hierarchy. All the off the shelf reasoners available expect the data to be cached in-memory to perform the reasoning. However, in our case due to size of the data it is stored in persistent storage [62].

This inferencing limitation further precipitates the need for the triple store [62] to have an available API. In our current implementation, we perform inferencing as the ontologies are being loaded. The ontologies are cached into an in-memory model,

their namespaces extracted and RDFS inferencing performed on them before pushing them into the triple store [62].

3.5.5 Query Results.

The queries on bioportal mainly extract the parent and child of a given term. Since these are first order simple queries, not much difference is seen in the performance. More complex queries that recursively calculate the path to root of a term should be used to get a better indication of the querying capabilities [62].

3.5.6 Child queries.

The child queries are used to extract the children of a given term. As already mentioned the user only supplies the 'term', without the associated namespace for which he needs the children. To create the SPARQL query we plug in the term with all the known namespaces extracted. This created query is then executed against the triple store. For eg. if the user wants to find the children of the term **'Size',** a concept mentioned in the amino-acid ontology, the following query is created:

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x FROM <bioportal:amino-acid>
WHERE { ?x rdfs:subClassOf
*<http://www.co-ode.org/ontologies/amino-acid/2006/05/18/amino-acid.owl#**Size**>*

This query is in addition to the other queries generated using different namespaces like *http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#.* Using this namespace another query(shown below) is generated. Similarly queries for other namespaces are generated [62].

Thus, for each term we need to generate as many queries as the number of namespaces present in Bioportal. This results in an amount of the order of ~300.

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x FROM <bioportal:Thesaurus >
WHERE { ?x rdfs:subClassOf
< http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#Size>

3.5.7 Parent queries.

A similar procedure of generating multiple queries for extracting the parents [62] of a term is used. Multiple SPARQL queries with different namespaces are created and executed against the triple store. The only difference from the child queries, is that the variable is at object position instead of the subject. For eg. to extract the parents of the term '**Size',** one of the generated queries is:

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x FROM <bioportal:amino-acid>
WHERE {
*<http://www.co-ode.org/ontologies/amino-acid/2006/05/18/amino-acid.owl#**Size***
rdfs:subClassOf ?x>

Similar to the case of children queries, the fanout for parent queries is of the order ~300 or the number of namespaces in Bioportal.

However, the Parent/Children queries provide us no information on the completeness of the support of SPARQL by a triple store. Both these queries only use the 'where' clause and don't test whether the triple store supports more complicated clauses like 'filter' and 'optional'.

Due to the simplicity of the queries and the fact that they use only one SPARQL construct – 'where' not much difference is seen in the performance of the triple stores. The difference is of the order of a few seconds(<2s) and even that gets reduced when the cache gets warm [62].

3.5.8 Results of Native Triple Stores.

Table 1, 2 and 3 present the time taken to load the three datasets. ‘Couldn’t load’ in the time field denotes that an out-of-memory error started appearing. These results are discussed in the next section.

Triple Stores	Time (min)
Jena SDB	164
Sesame Native	7
Mulgara	12
Virtuoso	Couldn’t load

Table 2. Time taken to Load Bioportal Ontology [62]

Triple Stores	Time (min)
Jena SDB	8.19
Sesame Native	4.46
Mulgara	Couldn’t load
Virtuoso	Couldn’t load

Table. 3. Time taken to load the UNIPROT 1M[62].

Triple Stores	Time (min)
Jena SDB	175
Sesame Native	123
Mulgara	Couldn’t load
Virtuoso	Couldn’t load

Table. 4. Time taken to load the UNIPROT 10M[62].

The above results indicate a superior performance of native stores like Sesame native, Mulgara and Virtuoso. This is in coherence with the current emphasis on development of native stores since their performance can be optimized for RDF. However, these native triple stores especially Mulgara and Virtuoso are constrained by the absence of an API. For certain use cases in Bioportal, presence of an API is necessary. Further, Table 2 and Table 3 point out that the inability of Mulgara and Virtuoso to load large datasets (>1M triples). It is important to note that the problem is not of lack of scalability with the two mentioned triple stores, but the inability of the third party APIs to handle large datasets. These APIs work perfectly fine when used

to access and query existing stores but throw up memory errors when used to load the data through them. Both Mulgara and Virtuoso are able to load the same datasets without any problems when used through their loaders (scripts or command line utilities like load). The documentation of both Mulgara and Virtuoso notes that the preferred way to load large datasets is through the supplied loaders instead of the third party APIs [62].

Tables 1,2 and 3 indicate that Sesame native gives the best performance of the 4 triple stores being compared. In addition to this, Sesame native comes with an inherent API and thus doesn't suffer from the same problem as the other 2 native stores Mulgara and Virtuoso do. However, Sesame native was only tested with only 10 Million triples. This size currently suffices the need of Bioportal. However [62] mentions that the largest dataset known to be loaded into Sesame native is only 50Million triples, both Mulgara and Virtuoso are said to be lot more scalable than this. This can be a serious limitation when the dataset becomes in-order of a few billions [62].

Jena SDB is able to load all the three datasets but is the slowest. This was expected since SDB is simply a loader which relies on third party relational databases which are formatted by it before loading the data. It is unable to be optimized for a particular database[62].

4 Discussion.

In this project I have presented an application that reads the data from three different databases and presents the results. Results can be acquired by applying some SPARQL Queries. If we closely take a look into the table 01 as shown in results part we can see that for each query performed here Triple Store (Jena TDB) have provided the fastest results. The second best we found here is the Relational Database (Jena SDB). Therefore, on the basis of results we can say that triple store performance and efficiency have found best among the other databases, and have recommend the best solution for using in semantic web applications.

Due to the time limit and to present the idea of Jena SDB we have tried Jena SDB with MySQL. There are some other triple stores like, Sesame, Mulgara and Virtuoso backed with Jena SDB I would like to try these native triple stores in future.

The Sesame is an open source framework for storage, inferencing and querying of RDF data. Sesame matches the features of Jena with the availability of a connection API, inferencing support, availability of a web server and SPARQL endpoint [62].

Mulgara is a native RDF triple store written in Java. It provides a Connection API that can be used to connect to the Mulgara store. Being a native triple store it has a 'load' script which can be used to load RDF data into the triple store [62].

Virtuoso, is a native triple store available in both open source and commercial licenses. It provides command line loaders, a connection API, supports for SPARQL and web server to perform SPARQL queries and uploading of data over HTTP [62].

From the project I have learnt many things about semantic web technology, especially about triple stores. The triple stores like Jena SDB are based on relational databases and simply provide an RDF or SPARQL interface. If we say like what features that triple stores contains and Relational Database do not have, from my point of view simple answer is that the triple store support for SPARQL, RDF and OWL (i.e the Semantic Web Technology stack).

In future I would like to enhance the data in ontology and try to read the data with above said native triple stores with my developed application to check the query execution performance and efficiency.

5 Conclusion.

During this project, I have introduced a novel approach to Semantic Web search, which allows for a semantic processing of Web search queries relative to an underlying ontology. I have presented a theoretical study about several technologies used in semantic web development to provide reliable and robust communication over the HTTP. After a careful survey of existing technologies used in semantic web, I have identified to setup a SPARQL Endpoint application that helps to investigate how to best store and manage the clinical data.

For the SPARQL Endpoint Application I have made an ontology, which is accessible with a Java application in which we used web services and semantic web tool Jena. From the help of Jena we able to read ontology data within this application. We applied several type of database schemes such as Relational Database and Triple Stores, that are able to read the ontology data and present results with the help of SPARQL Queries over the HTTP.

Results presents the remarkable performance of triple stores like Jena TDB and Relational Database backed with Jena SDB.

In conclusion, the overall goal of our project has been achieved. Since, I can now connect all type of data base in application which I have made for getting the functionalities of SPARQL Endpoint, so the whole application exhibits as “plug and play”.

For future work I would like to add more data into ontology. Make more complex SPARQL Queries and check the application with other native triple stores such as Virtuoso and Sesame.

References:

[1].http://en.wikipedia.org/wiki/Semantic_Web; Access date: 15.02.2014.

[2].http://www.icn.ch/images/stories/documents/programs/icnp/icnp_catalogue_development.pdf Access date: 10.02.2014.
[3]. http://en.wikipedia.org/wiki/HTML#Semantic_HTML access date: 20.02.2014.

[4]. http://en.wikipedia.org/wiki/Microformat access date: 20.02.2014.

[5]. http://en.wikipedia.org/wiki/Resource_Description_Framework access date: 20.02.2014.

[6]. http://en.wikipedia.org/wiki/Web_Ontology_Language access date: 20.02.2014.

[7]. http://en.wikipedia.org/wiki/XML access date: 20.02.2014.

[8]. http://www.w3.org/TR/owl-features/ access date: 21.02.2014.

[9]. http://en.wikipedia.org/wiki/Semantic_Web_Stack access date: 20.02.2014.

[10]. http://en.wikipedia.org/wiki/W3C_XML_Schema access date: 20.02.2014.

[11]. http://en.wikipedia.org/wiki/Resource_(Web) access date: 21.02.2014.

[12]. http://en.wikipedia.org/wiki/Rule_Interchange_Format access date: 25.02.2014.

[13]. http://en.wikipedia.org/wiki/Resource_(computer_science). Access date: 25.02.2014.

[14]. http://en.wikipedia.org/wiki/Relational_database. Access date: 25.02.2014.

[15]. James, A., *Qualibria Constraint Definition Language (CDL), Language Guide*. 2010.

[16]. http://www.w3.org/RDF/Metalog/docs/sw-easy. Access date: 10. 01. 2014.

[17]. http://psychology.wikia.com/wiki/Semantic_web. Access date: 10.01.2014.

[18]. http://en.wikipedia.org/wiki/Semantic_Web. Access Date: 10.01.2014.

[19]. http://semanticweb.org/wiki/Ontology Access date: 22.02.2014.

[20]. Zeeshan Ahmed and Detlef Gerhard, 2010, “Role of Ontology in Semantic Web Development”, Mechanical Engineering Information and Virtual Product Development (MIVP), Vienna University of Technology, Vienna, Austria.

[21]. http://www.w3.org/2001/sw/wiki/Protege Access date: 10.01.2014.

[22]. http://semanticweb.com/introduction-to-sparql_b22498 Access date: 22.02.2014.

[23]. S. Brin, L. Page, “The anatomy of a large-scale hypertextual Web search engine”,

Comput. Netw. 30 (1–7) (1998) 107–117.

[24]. B. Fazzinga et al. /"Web Semantics: Science, Services and Agents on the World Wide Web" 9 (2011) 453–473.

[25]. Berners-Lee, T., Hendler, J., & Lassila, O. (2001, May). "The Semantic Web. Scientific American", pp. 29---37.

[26]. Klyne, G., Caroll, J. J., & McBride, B. (2004, February). "Resource Description Framework (RDF):Concepts& Abstract Syntax. Retrieved from W3C Recommendation": http://www.w3.org/TR/rdf-concepts/.

[27]. Prudhommeaux, E., & Seaborne, A. (2008, January). "SPARQL Query Language for RDF". Retrieved from W3C Recommendation: http://www.w3.org/TR/rdf-sparql-query/.

[28]. Heiko Paulheim1 and Sven Hertling. (2006). "Discoverability of SPARQL Endpoints in Linked Open Data". University of Mannheim, Germany. Research Group Data and Web Science.

[29].http://www.bioontology.org/wiki/images/6/6a/Triple_Stores.pdf.Accessdate:10.03.2014.

[30]. http://en.wikipedia.org/wiki/Jena_(framework). Access date: 13.03.2014.

[31]. "Nursing Documentation", Dr. Ali D. Abbas/ Instructor, Fundamentals of Nursing Department, College of Nursing, University of Baghdad, ali_dukhan@yahoo.com.

[32]. White, L.; Duncan, G.; and Baumle, W.: Foundation of Nursing, 3rd ed., 2011, Australia: CENGAGE, P.P.173-190

[33]. B. Davis, J. Billings, R. Ryland, "Evaluation of nursing process documentation", J. Adv. Nurs. 19(5) (1994) 960–968.

[34]. S. Sahlstedt, H. Adolfsson, M. Ehnfors, B. Kallstrom, "Nursing process documentation—effects on workload and quality when using a computer program and a key word model for nursing documentation", in: U. Gerdin, M. Tallberg, P. Wainwright (Eds.), Nursing Informatics—The Impact of Nursing Knowledge on Health Care Informatics, IOS Press, Amsterdam, 1997, pp. 330–336.

[35]. V. Fiechter, M. Meier, Pflegeplanung-EineAnleitung fur die Praxis ("Nursing care planning—a practical introduction"), Recom, Basel, 1993.

[36]. C. Varcoe, "Disparagement of the nursing process: the new dogma?", J. Adv. Nurs. 23 (1) (1996) 120–125.

[37]. "Nursing Terminologies to Support Nursing Practice", Judith J. Warren, PhD, RN, BC, FAAN, FACMI Christine A. Hartley Centennial Professor University of Kansas School of Nursing

[38]. Thede, L., Schwiran, P., (February 25, 2011) "Informatics: The Standardized Nursing Terminologies: A National Survey of Nurses' Experiences and Attitudes - Survey I*" *OJIN: The Online Journal of Issues in Nursing* Vol. 16 No. 2. **DOI:** 10. /OJIN.Vol16No02InfoCol01

[39]. P. Hanisch, S. Honan, R. Torkelson, "Quality improvement approach to nursing care planning: implementing practical computerized standards", J.Healthc. Qual. 15 (5) (1993) 6–12.

[40]. M. Peterson, "Time and the nursing process", Holist. Nurs. Pract. 1 (3) (1987) 72–80.

[41]. C. Varcoe, "Disparagement of the nursing process: the new dogma?", J. Adv. Nurs. 23 (1) (1996) 120–125.

[42]. J. Allan, J. Englebright, "Patient-centered documentation—an effective and efficient use of clinical information systems", JONA 30 (2) (2000) 90–95.

[43]. A. Bu¨ ssing, B. Herbig, "Recent developments of care information systems in Germany", Comput. Nurs. 16 (6) (1998) 307–310.

[44]. W. Goossen, P. Epping, T. Dassen, A. Hasman, W. van den Heuvel, "Can we solve current problems with nursing information systems"?,Computer. Methods Programs Biomed. 54 (1–2) (1997) 85–91.

[45]. W. Goossen, P. Epping, T. Dassen, "Criteria for nursing information systems as a component of the electronic patient record": an International Delphi study, Comput. Nurs. 15 (6) (1997) 307–315.

[46]. B. Harris, Becoming de-professionalized: "one aspect of the staff nurse's perspective on computer mediated nursing care plans", Adv. Nurs. Sci. 13 (2) (1990) 63–74.

[47]. D. Hinson, S. Huether, J. Blaufuss, M. Neiswanger, A. Tinker, K. Meyer, R. Jensen, "Measuring the impact of a clinical nursing information system on one nursing unit", in: Proceedings of the AMIA Annual Fall Symposium, McGraw-Hill, New York, 1994, pp. 203–210.

[48]. Coenen A, McNeil B, Bakken S, et al. "Toward comparable nursing data: criteria for data sets, classification systems, and nomenclature". ComputNurs. 2001;19(6):240–6. [PubMed].

[49]. American Nurses Association. Nursing practice information infrastructure. 2006. [Accessed October 24, 2006]. Available at: http://www.nursingworld .org/npii/terminololgies.htm.

[50]. Berners-Lee, T., Hendler, J., &Lassila, O. (2001, May). "The Semantic Web. Scientific American", pp. 29-37.

[51]. http://webdam.inria.fr/Jorge/html/wdmch8.html. Access date: 27.03.2014.

[52]. https://cs.uwaterloo.ca/~gweddell/cs848/semanticweb.pdf. Access Date: 07.03.2014.

[53]. Grobe, Michael. "Rdf, jena, sparql and the'semantic web'." In *Proceedings of the 37th annual ACM SIGUCCS fall conference*, pp. 131-138. ACM, 2009.

[54]. Bizer, Christian, and Andreas Schultz. "Benchmarking the performance of storage systems that expose SPARQL endpoints." *World Wide Web Internet And Web Information Systems* (2008).

[55]. Paulheim, Heiko, and Sven Hertling. "Discoverability of SPARQL Endpoints in Linked Open Data." In *Proceedings of the ISWC*. 2013.

[56]. Yan, Ying, Chen Wang, Aoying Zhou, Weining Qian, Li Ma, and Yue Pan. "Efficiently querying rdf data in triple stores." In *Proceedings of the 17th international conference on World Wide Web*, pp. 1053-1054. ACM, 2008.

[57]. Morsey, Mohamed, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. "Usage-Centric Benchmarking of RDF Triple Stores." In *AAAI*. 2012.

[58]. Schmidt, Michael, Thomas Hornung, Georg Lausen, and Christoph Pinkel. "SP^ 2Bench: a SPARQL performance benchmark." In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pp. 222-233. IEEE, 2009.

[59]. Brodt, Andreas, Daniela Nicklas, and Bernhard Mitschang. "Deep integration of spatial query processing into native RDF triple stores." Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2010.

[60]. Berners-Lee, T. (2000): “Weaving the Web: The Past, Present and Future of the World Wide Web by its Inventor”. London, Texere.

[61].http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Model.html. Access date: 25.04.2014.

[62]http://www.bioontology.org/wiki/images/6/6a/Triple_Stores.pdf.AccessDate:27.04.2014.

[63]. Fernandez-Breis, Jesualdo Tomas, Marcos Menarguez-Tortosa, Catalina Martinez-Costa, Eneko Fernandez-Breis, Jose Herrero-Sempere, David Moner, Jesus Sanchez, Rafael Valencia-Garcia, and Montserrat Robles. "A Semantic Web-based System for Managing Clinical Archetypes." In *Conference proceedings:... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, vol. 2008, pp. 1482-1485. 2007.