ARCHITECTURAL DESIGN DECISION KNOWLEDGE MANAGEMENT SYSTEM

Shashi Lakshan Chandrasinghe

(179311P)

M.Sc. in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

May 2019

ARCHITECTURAL DESIGN DECISION KNOWLEDGE MANAGEMENT SYSTEM

Shashi Lakshan Chandrasinghe

(179311P)

This dissertation submitted in partial fulfillment of the requirements for the Degree of MSc in Computer Science specializing in Software Architecture

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

May 2019

DECLARATION

I declare that this is my own work and this MSc Thesis Project Report does not incorporate without acknowledgement of any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works.

....

Shashi Lakshan Chandrasinghe   Date

I certify that the declaration above by the candidate is true to the best of my knowledge and that this project report is acceptable for evaluation for the MSc Thesis (CS6997).

....

Dr. Indika Perera   Date

ACKNOWLEDGEMENT

My sincere appreciation is dedicated to my research supervisor Dr. Indika Perera for his valuable support and guidance to complete the MSc Thesis Report successfully and also I would like to express my gratitude to my friends and colleagues in MSc' 17 who supported to complete the task.

Finally, I would like to thank the academic and non-academic staff of Department of Computer Science and Engineering of the University of Moratuwa.

ABSTRACT

The software systems typically fail and deviate from its prescriptive architecture due to various reasons such as incorrect architectural design, lack of experience and lack of domain knowledge. After precise software requirements are gathered from customers, those requirements should be converted into an appropriate design. Suppose if any inappropriate design is constructed from these requirements, it may cause to reconstruct the system implementation. So, a set of good architectural design decisions form a good system architecture and those architectural design decisions should be documented or stored as knowledge bases to use further.

Various methodologies exist to store architectural design decisions and trace them. Most of them have some drawbacks such as lack of time to gather and store and additional cost to maintain such knowledge bases. As the key objective, this report proposes an ontological knowledge management system to solve above mentioned problems in software engineering industry for avoiding the extra costs to redevelop or refine the software system implementation.

Though the implemented solution is ontology-based knowledge management system, it seems to be a simple web application to the end user. User-friendly web interfaces are implemented to store and retrieve the architectural design decisions, based on completed or already initiated software projects. Those design decisions would be useful for the professionals who design the effective software architecture designs.

Finally, empirical and Likert questionnaires were conducted to prove that the implemented solution works perfectly as a solution for the stated problems and this report ends mentioning some limitations and future work with relevant to ontological knowledge management systems and its technologies.

Keywords: knowledge bases, ontology, architectural design decisions

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
SDLC	Software Development Life Cycle
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
OWL	Web Ontology Language
JESS	Java Expert System Shell
RDQL	RDF Data Query Language
OWL-QL	OWL Query Language
W3C	World Web Consortium
SPARQL	Simple Protocol and RDF Query Language
OOP	Object Oriented Programming
RAM	Random Access Memory

CHAPTER 1

INTRODUCTION

This chapter covers the background of the research work and illustrates the problem statement with proposed solution and latter part contains the objectives and the overview of the report.

1.1 Background

Software systems are constructed to satisfy the customer requirements and to attain the business goals of organization at present. Therefore, the software architecture is considered as the bridge between those business goals among stakeholders and acts as the blueprint of a constructed software system [1]. Software architecture consists of the earlier design decisions which are engaged to development and maintenance of the Software Development Life Cycle (SDLC).

Software architect is a professional who is able to design the software or system architecture by considering the other requirements such as low cost, time constraint, performance, allocation of resources, and end-user satisfaction, etc., and has clear understanding about the certain domain and having experience about handling networking and computer systems such as hardware, software, internet, web portals and security etc.

There are two different software architecture views those are prescriptive architecture and the descriptive architecture. Prescriptive architecture of a system captures the architectural design decisions which are made prior to the system construction and the descriptive architecture describes how the system has been built.

Software are developed based on these concepts and used for long-time. But within the lifespan of utilization of software, there will be a deviation between the prescriptive architecture and the descriptive architecture which is called as architectural degradation. Architectural degradation can be divided into two concepts. Those are architectural drift and architectural erosion. Architectural erosion is the most critical factor because it violates the prescriptive architecture while architectural drift does not violate any prescriptive architectural design decisions. Both architectural drift and architectural erosion have chances to exist by many reasons such as developers may

not have an idea about prescriptive architecture of the system, but they only focus on the descriptive architecture due to the time constraints for software development.
An effective architect should consider on the architectural influences when design the system architecture. The relevant influences that impact on system architecture are stakeholders, development organization, background and experience of the architect and the technical environment. Initial architecture or the prescriptive architecture should be consistent for long time because a changing requirement to the prescriptive architecture can consume high cost. Therefore, the architectural design decisions should be very precise and consistent.

Obtaining the right architectural design decisions is the most challenging process in the software industry. Lack of experience, lack of domain knowledge, and ineffective communication among stakeholders can be the main reasons to fail the initial architectural design decisions. During previous years, researchers have explored many experiments to solve these issues. Most of them provided the solutions with knowledge-based systems with different technologies.

1.2 Research Question

While developing complex software systems, architectural design decision are considered as the first class entities since it affects all the concerns of each stakeholder [2]. How the system architects find the architectural design decisions are properly evaluated to the particular business domain.

When a new software project is initiated to implement, senior developers or architects have to face many challenges such as developing the right design, and choosing the appropriate technologies and resources etc. Inappropriate design decisions may provide the results of an excessive budget for a software system with long-time period of development.

There are few reasons for why initial architectural designs fail,

- Lack of domain knowledge
- Lack of experience in relevant field
- Not conducting enough evaluation techniques for selecting the correct design decisions

When developers start a new project or a new feature for an existing software system, it is hard to find similar implementations for the similar kinds of requirements.

1.3 Motivations to Solve the Problem

A knowledge-based system for architectural design decisions will be preferable solution for software professionals to design a better software architecture with good architectural design decisions. Therefore, implementing the knowledge-based system is the major motivation and solution for above stated problems.

Earlier, researchers tried to solve problems by implementing such knowledge-based systems using several technologies. Kruchten implemented a document-based ontology, but it contained several issues [2]. Documents should be maintained in and readers had to spend more time to retrieve the relevant information because it was a manual task. It required more tools for some other requirements to deal with the knowledge bases. So, the reader had to learn more for overcoming such stuff. Furthermore, readers need to learn more about tools like protégé and to analyse [3]. However, this system led to create an ontology

Main motivation of this research is to focus on those issues, to mitigate them and to develop with a user-friendly knowledge-based system with the following features,

- End-user can enter the data using a web interface
- End-user can analyse/retrieve information by using the same system

1.4 Research Scope

Identifying the most important factors which are relevant to the architectural design decisions and identifying the most suitable technology stack to implement the proposed solution which is the ontology-based knowledge management system. The boundaries and the constraints are listed below.

- Ontology management system only supports for OWL version 1.
- Apache Jena will be used as the OWL generating tool.

1.5 Research Objectives

The main objective of this research is to find a solution for the above stated problems. To achieve this main objective, there are several sub objectives established to be accomplished for this research. The relevant sub objectives are;

- Identify and analyze the past experience about the system design from system architects.
- Identify and analyze the literature of research in the similar context and the current state of the research problems and the solutions.
- Define the research methodology to resolve the research problem with suitable technologies.
- Conduct both theoretical and empirical evolutions to prove that the implemented solution will help to solve the stated problems.
- Implement the ontology-based knowledge management system as the proposed solution.

1.6 Overview of the Document

This thesis consists of six chapters. This initial chapter included background knowledge on software architecture and the importance of the architectural design decisions which describes that how they affect to the system existence. It also described the research problem and motivation factors to solve the research problem. Finally, it contained the research objectives.

The second chapter illustrates the findings on the literature survey of similar context. It describes the important aspects of the ontology-based knowledge management system for architectural design decisions. Literature survey also includes an introduction to ontology and its technologies with the relevant technical tools.

The third chapter includes the research methodology which describes the solution by using system diagrams and the technology stack.

The forth chapter narrates the solution architecture and compromises that how the solution is implemented based on the methodology.

The fifth chapter describes that how the implemented solution is evaluated in both theoretically and empirically. Finally, this chapter concludes with some information on how this solution performs by comparing the existing solutions.

The final chapter is the conclusion of this thesis which describes the chapter by illustrating the limitations and future work.

CHAPTER 2

LITERATURE REVIEW

This chapter covers the background studies of importance of architectural design decisions and the existing knowledge-based systems to store and retrieve the architectural design decisions.

2.1 Importance of Architectural Design Decisions

Architectural design decisions are considered as the first-class entities in the process of software development. The necessity of a software system emerges from business requirements from various stakeholders who engaged with the software system. Those requirements can be functional or non-functional. Also, architectural decisions should satisfy those requirements in high level manner [4]. In complex software designing process, a group of architects or senior developers who are expertise in software development process and procedures, assemble in a place or via online, discuss about the design options and choose the best one which should fulfil the business requirements. So, this process proposes that the architecture designing is a collaborative process [5].

Philippe Kruchten defines three main categories of architectural design decisions [2]:

1. Existence decisions.
    - These kinds of decisions are visible in the system implementation.
        - Ex: Java messaging services should be used to send and receive messages.
2. Property decisions.
    - These decisions can help to formulate rules or guide lines.
        - Ex: implementation should not use propriety libraries.
3. Executive decisions.
    - These decisions do not connect to the design directly. Most probably, these decisions are made by business environment and created the impact on the development processes and the people in the organization.
        - Ex: development process decisions such as team should follow the agile methodology.

Throughout this chapter, the literature review clearly says that a good set of architectural design decisions make a good software architecture. A good software architecture should fulfil the whole business needs of every stakeholder.

2.2 Challenges in Architectural Design Decisions

After identifying the business requirements for a complex software system, the initial step or the software architecture design should be developed to initiate the system development. But, obtaining architectural design decisions is a difficult task in software architecture designing process because a set of architectural design decisions formulate a software architecture which consider as the first-class entity to the software system [6]. Some of the challenges in architectural design decisions are listed below.

- Violating the design rules and constraints
    - If the architecture design contains violated rules and constraints, it may lead to architectural drift when the architecture design is in the software development phase. Therefore, it may require additional costs to recover the architectural drift.

- Less experience and less technical knowledge
    - Sometimes hiring well-experienced and technically skilled people requires high costs for an organization. Therefore, business organisations motivate the people who work within the organization, for developing the architectural designing process. But it may provide an output with wrong architectural design. In the middle of software development phase, it may be realized that the architectural decisions they made are wrong. So, it again requires the additional costs to recover.

- Quality attributes trade-off
    - Architectural decisions are constructed to fulfil the business requirements which may be functional or non-functional. Non-functional requirements are the quality attributes for the software system. Those quality attributes can be named as performance, security, availability, and fault tolerance etc. When the software architecture satisfies the performance, it may lead to the system with less security features. So, an architect should balance those quality attributes when designing the software architecture.

- Globally distributed knowledge
    - Today, many software organizations are operating in different locations all over the world. Sometime, development and architecture tasks can be divided to different locations. Therefore, it is difficult to assemble and collaborate those people for effectively utilizing their knowledge.

- Insufficient knowledge bases of previous software architecture [7]
    - The software architects typically make architectural design decisions to form software architecture, but they fail to create documents in written format on their ideas and rationales behind the design decisions. When he/she leaves the organization, his/her cognitive is also lost to the organization. So, knowledge of a person should be stored for future architectural design decisions. Important aspects of such knowledge bases are [8];
        - Used for education/training purposes.
        - Used for system analysis.
        - Used as a communication medium among stakeholders.

This research provides a solution to the problem which is mentioned above. This research illustrates an ontology-based knowledge management system for architectural knowledge.

2.3 Some Knowledge Bases for Architectural Knowledge

From early ages of software development, people used to store the architectural knowledge for the purpose of reusing and tracing. Researchers suggested a several mechanisms to create knowledge bases such as document based, monolithic system based, web based, etc. Currently researchers are finding solutions to manage the knowledge in the concept of ontology. Some of the mechanism used to create knowledge bases are listed below.

- A web-based framework for managing architecture knowledge.
    - Abdullah et.al [5] proposed a 3-tier web-based solution to manage the architectural knowledge. The solution mainly focused on the scenarios which indicated that users can create and evaluate scenarios through the system. In the system, the data will be stored as the architectural knowledge.
    - Ali et.al [9] also suggested a web-based tool called PAKME to manage architectural knowledge. This solution contained three types of search functionality for the users.
        - Key-word based search functionality
        - Advanced search functionality
        - Navigation based search functionality

      PAKME provides several interfaces or the templates to facilitate the architectural knowledge before storing them. It also stores the scenario-based knowledge.

- Automated systems for managing architectural knowledge.
    - Ali et.al [4] recommended a template which is made by mining architectural patterns that are applied to scenarios, for storing architectural knowledge. Following table 2-1 shows how they store the architectural knowledge though a template.

Table 1-1 Sample Data of Architectural Knowledge

Generic quality attribute	Flexibility/Scalability (ASR entity)
Abstract scenario	Application shall instantly notify changes to the interested clients (Scenario entity).
Abstract scenario	Application shall be able to handle simultaneous notification requests from increased number of client (Scenario entity)
Architecture Decision	Event notification (Architecture Decision entity)
Design option 1	Publish scribe (Alternative entity)
Design option 2	Java RMI (Alternative entity)
Design Pattern	Publish on demand (Pattern entity)

    - Tyree et.al [10] also proposed a template based automated system for architectural knowledge and mainly focused on the design rationale.

- Documentation framework for managing architectural knowledge**.**

    - Heesh et.al [11] introduced a document based framework which represents the architectural knowledge with 4 different viewpoints such as a decision detail view point, a decision relationship view point, a decision chronology view point and a decision stakeholder involvement view point.

- Semantic web techniques for managing architectural knowledge.
    - Present day, research engineers acquire the usage of semantic web technologies for representing knowledge bases for the several domains which have similar experiences [12]. Both human and computer can understand and work in a collaborate environment through semantic web. Later a rich conceptual schema called ontologies emerged to play a key role of knowledge bases with managing and reasoning [13].

2.4 Why Ontology Driven Software Engineering is Needed.

In computer science, ontology become a conceptual trend to have knowledge bases which focus on specific domains. Over the past two decades, many of software systems failed due to lack of domain knowledge, lack of experience of software engineers and the deviation from the initial design architecture. When the consideration of the life cycle of a software system was emerged, it is constructed by several phases including requirement gathering phase, design phase, implementation phase, testing phase and maintenance phase. In each phase of SDLC has a specific domain knowledge. A research [14] was conducted by Bhastia et.al and categorized ontologies in SDLC phases. The following figure 2-1 shows that how they did the categorization.

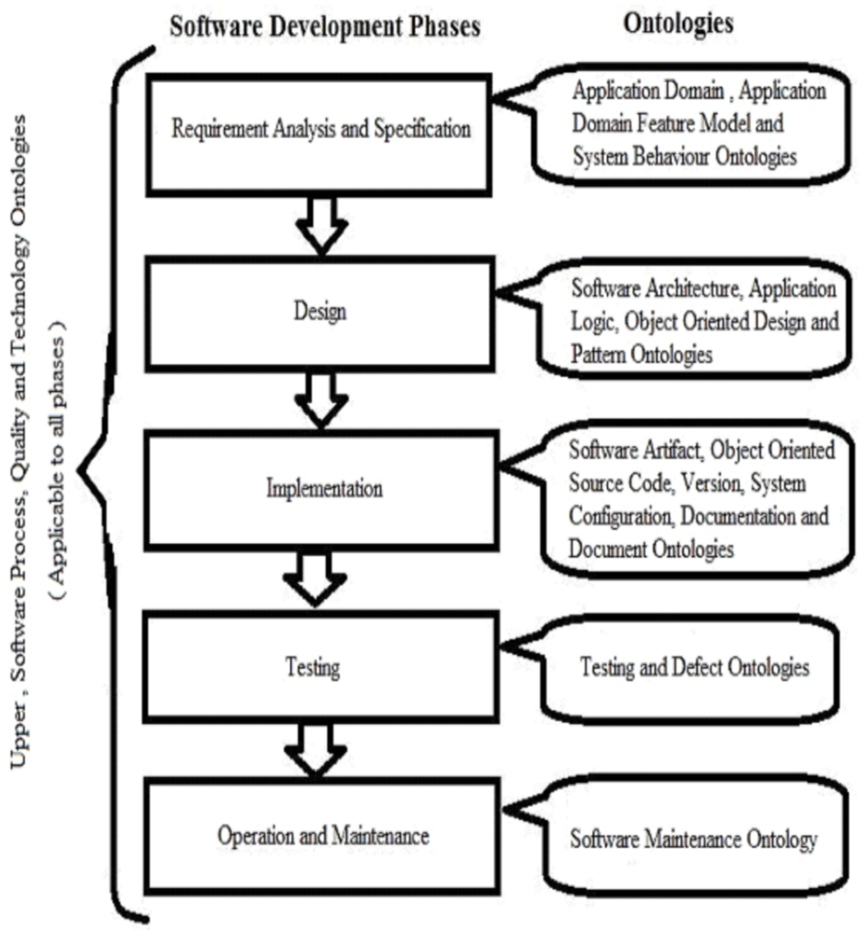

Software Development Phases
Ontologies
Requirement Analysis and Specification
Application Domain , Application Domain Feature Model and System Behaviour Ontologies
Design
Software Architecture, Application Logic, Object Oriented Design and Pattern Ontologies
Implementation
Software Artifact, Object Oriented Source Code, Version, System Configuration, Documentation and Document Ontologies
Testing
Testing and Defect Ontologies
Operation and Maintenance
Software Maintenance Ontology
Upper , Software Process, Quality and Technology Ontologies
(Applicable to all phases)

Figure 2-1 Ontology Categorization in SDLC Phases

Dillon et.al [15] pointed out the necessity of software engineering knowledge management system which provided better communication of software engineering domain knowledge among human and computers.

Requirement engineering considered as the initial phase of SDLC. Some of the problems in this phase are [16];

- Mostly, the software engineers are not the domain experts.
- Incomplete and ambiguous requirements may tend to do reworks after the system implementation.
- Need to contact customers time to time to grab the domain knowledge.

Above stated problems can be solved by an ontology-based knowledge management approach.

To test a software system successfully, test engineers should have the strong domain knowledge within the context. One of researcher has implemented a software testing knowledge management portal to share and retrieve the testing knowledge among the software testers [17]. Mainly three categories of information were stored in the knowledge management system. They are general information (project details), test cases and test results. Figure 2-2 below, illustrates the architecture of above-mentioned knowledge management system with five layers.

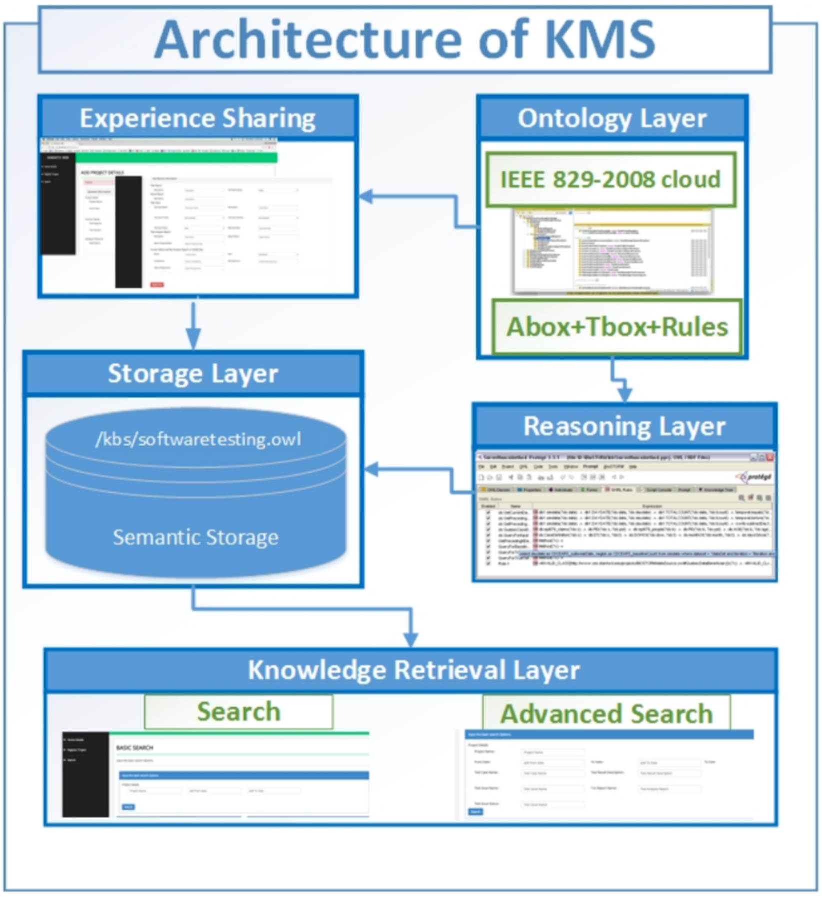


Figure 2-2 Design Architecture of Knowledge Management Portal

2.5 Strengths and Weaknesses of Having Ontology Driven Software Engineering

Bhaia et.al [18] analyzed the SWOT (Strengths, Weakness, Opportunities and Threats) analysis of ontology driven software engineering.

- Strengths using ontology to drive software engineering
    - Ability to share and reuse software engineering knowledge.
    - Software engineering knowledge is available in the format which is understandable by both human and machines.
    - Effective communication media.
- Weakness of using ontology to drive software engineering
    - No standardization to generate ontologies for software engineering.
    - Considerable time to develop an ontology which gain an extra cost.

2.6 Designing Ontology-based Knowledge Management System.

At the beginning, ontology life cycle and the ontology development process were introduced in 1997 [19]. Typically, designing an ontology can be divided into two main sections that are conceptualization and specification [20]. Conceptualization describes about organizing the knowledge. Specification describes about to grab the informal knowledge from a specific domain.

Basic ontology design steps are [20]:

- Identifying the ontology goal and scope.
- Focusing the domain description.
- Identifying motivation factors and competence questions to build the ontology.
- Identifying the relations among terms in the focused domain.
- Identifying the classes, attributes and relations among the classes.
- Implementing the ontology with supporting tools like protégé [21].

2.7 Problems in Ontology Development

- All ontology development approaches do not cover all the processes in ontology life cycle but most of approaches are focused on the ontology implementation activities. There is a lack of attention to other aspects such as ontology management, learning and ontology evaluation.
- Mostly, the ontologies are not designed for general usage because ontologies are typically developed for a single domain.
- Available tools do not cover all the necessary activities in ontology development.

2.8 Ontology Implementation

To implement ontology-based knowledge management system, there is a tool to support or code implementation approach with using apache Jena [22]. A domain specific research was implemented by using protégé tool because it has a plug-and-play environment [20]. Other thing that ontologies which are implemented by using protégé, that can be exported into different formats such as RDFS and OWL.

If the ontology developer is capable of java programming, Apache Jena library can be used to implement the ontology-based system. Jena API supports to create ontology classes and properties which formulate a ontological model [23]. Basic building blocks are represented by using ontology classes in Jena API. The main operations which are union, intersection and difference, provided by Jena API. Those operations are used to create new ontology models. The following figure 2-4 shows the development steps for the ontology-based knowledge management system.

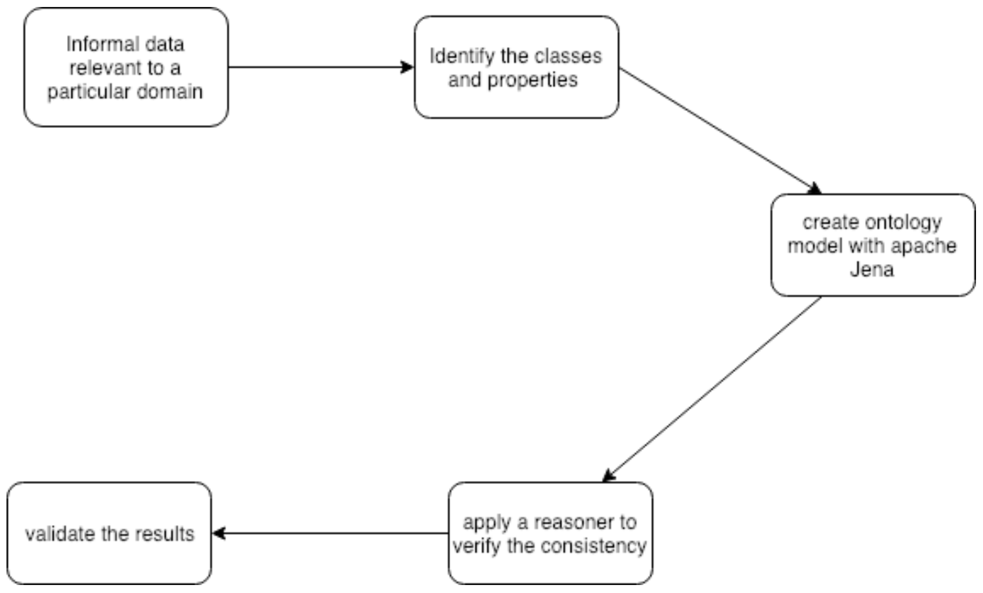

Informal data relevant to a particular domain
Identify the classes and properties
create ontology model with apache Jena
apply a reasoner to verify the consistency
validate the results

Figure 2-3 Ontology Development Process Using Apache Jena API

2.9 Ontology Validation

After implementing the ontology system, the system should be validated. RDF Data Query Language (RDQL) [20], OWL Query Language (OWL-QL) can be used to query the ontology knowledge management system. Otherwise the FaCT++ [17] and HermiT which are the inbuilt reasoners in the protégé, can be used to evaluate the ontology system. As a conclusion of ontology evaluation tools, there are six dimensions with respect to the quality of ontology system [24].

1. Human understandable
2. Logical consistency
3. Modelling issues
4. Ontology language specification (whether the syntax is correct)
5. Real world representation
6. Semantic applications (how to map the ontology to a software system)

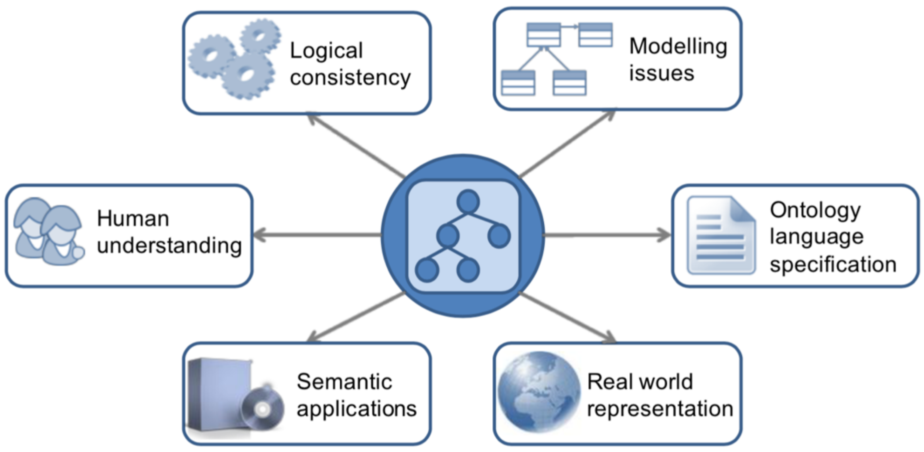


Figure 2-4 Ontology Evaluation Dimensions [24]

2.10 Basic Introduction to Ontology and Its Languages

Ontology is a data model of knowledge with a set of concepts within a certain domain and the relationships among those concepts. In philosophy, ontology is the study of what exists in general [25]. Basically, the word ontology come up with ‘onto’ and ‘logia’. ‘onto’ means existence or being real and ‘logia’ means science or study.

If particular entity such as a person in human domain want to be considered, particular person might have many relationships among other individuals; attributes and properties would describe those relationships. The languages which supported for the ontologies, are RDF (Resource Description Framework), RDFS (Resource Description Framework Schema) and OWL (Web Ontology Language).

There are four main types of ontologies.

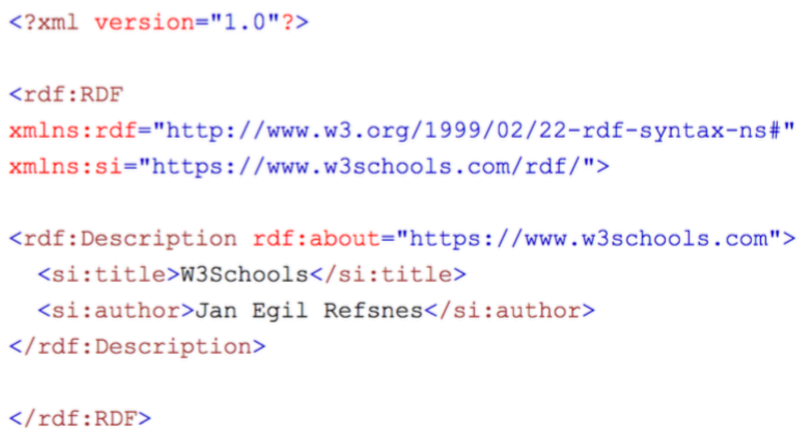
- Top-level ontology or Upper ontology
    - Represents very general concepts

- Domain ontology
    - Fundamental concepts according to a generic domain
- Task ontology
    - Fundamental concepts according to a general activity or task
- Application ontology
    - Specialized ontology focused on a specific task and domain

To contain a universal understanding of the web resources between software and people, to share the domain knowledge, and to analyse the domain knowledge are the main reasons behind the implementation of ontology. Ontology can be applied for different domains such as software process domain, healthcare domain, education domain, aviation domain, etc.

2.10.1 RDF

RDF [26] is considered as the first ontology language and was developed my W3C (World Wide Web Consortium). RDF can describe the information in a general manner. So, computer applications easily understand the RDF content which is modelled with a form of subject-predicate-object. Subject is considered as the 'thing' which is the class in Object Oriented Programming, and predicate description about the relationship between Subject and Object. Object is the value which is assigned to the Predicate. This is called 'triple' terminology in RDF.

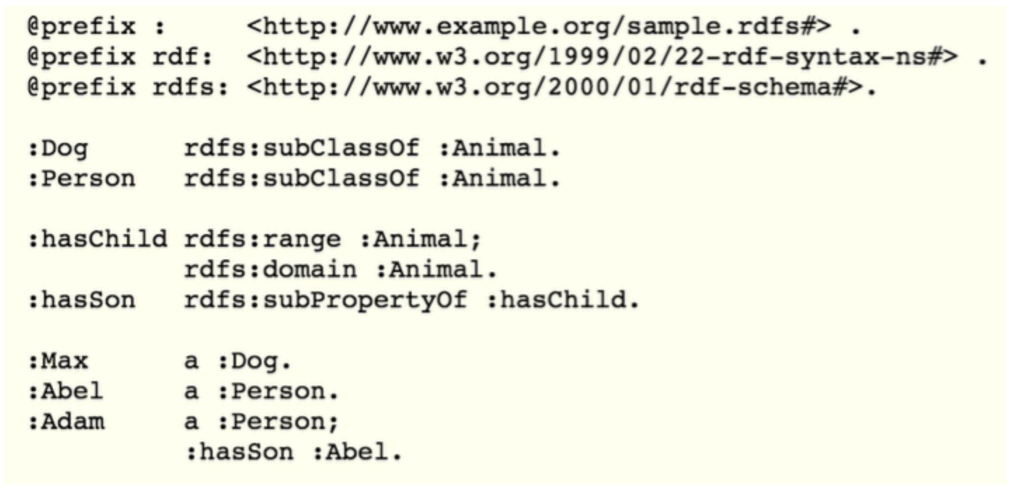


```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:si="https://www.w3schools.com/rdf/">

<rdf:Description rdf:about="https://www.w3schools.com">
  <si:title>W3Schools</si:title>
  <si:author>Jan Egil Refsnes</si:author>
</rdf:Description>

</rdf:RDF>
```

Figure 2-5 Sample RDF Format

2.10.2 RDFS

RDFS [27] is much richer than the RDF semantically. RDFS could describe the resources in a concept of classes, properties and values. Resources are like instances and properties are like attributes. The best thing is to create statements on resources and type of the relationship. When considering for adding new properties to the classes of ‘thing’, (‘thing’ is the root class of any other class) there might be some limitation on RDFS while OWL would solve those problems.



```
@prefix :     <http://www.example.org/sample.rdfs#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

:Dog      rdfs:subClassOf :Animal.
:Person   rdfs:subClassOf :Animal.

:hasChild rdfs:range :Animal;
          rdfs:domain :Animal.
:hasSon   rdfs:subPropertyOf :hasChild.

:Max      a :Dog.
:Abel     a :Person.
:Adam     a :Person;
          :hasSon :Abel.
```

Figure 2-6 Sample RDFS Format

2.10.3 OWL

OWL [28] is the current trending ontology language. It is more expressive compared to the RDF and RDFS. It is considered as the next generation of the web (i.e. web 3.0) and has three sub languages which are OWL Full, OWL DL and OWL Lite. OWL2 is the latest updated version, extended from OWL1 and has new features compare to OWL1. OWL basics are classes, individuals, properties and special classes. Every OWL class would be a sub class of ‘OWL: Thing’ which is the root class in OWL. Instances of a class are called individuals. Below example shows how to define a class using OWL.

```
<owl:class rdf:ID= “Man”>

<rdfs:subClassOf rdf:resource:“#Person” />

</owl:class>
```

Name of the class would be defined by ‘rdf:ID’. The hierarchy of the classes can be defined by ‘rdfs:subClassOf’ element. Man is the sub class of the Person class. Very complex classes can be expressed by adding Boolean operators such as union, complement and intersection.

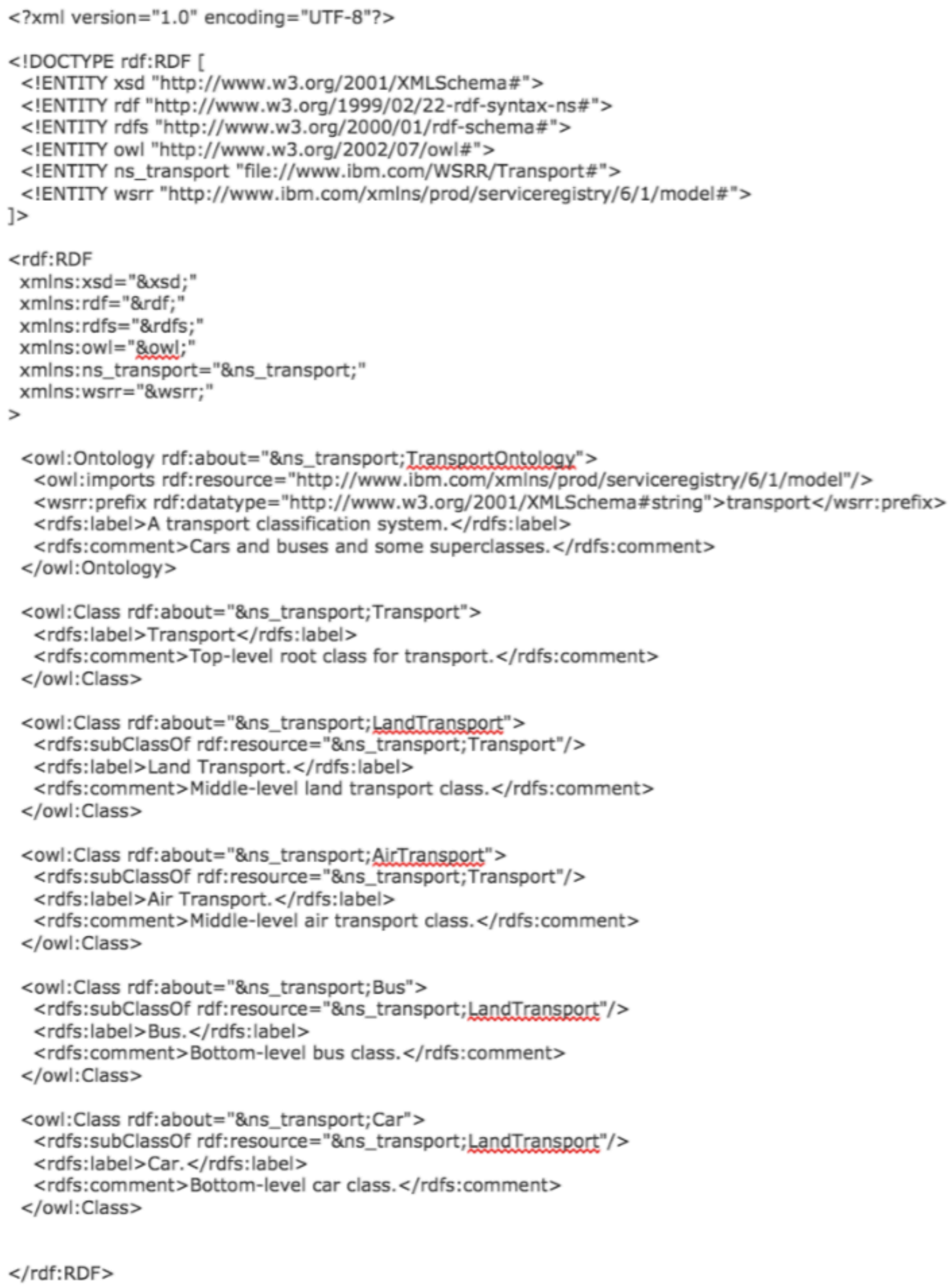
Individuals are the instances of a class and are constant.

```
<Man rdf:ID = “Sam” />
```

OWL properties show the relationship between instances of two classes. Following example shows how instances are related to each other.

```
<owl:ObjectProperty idf:ID = “isSister”>
        <owl:domain rdf:resource = “#Woman” />
        <owl:range rdf:resource = “#Person” />
</owl:ObjectProperty>
```

‘ObjectProperty’ relates to class. ‘Woman’ is a sister of ‘Person’ class instance.



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

<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY ns_transport "file://www.ibm.com/WSRR/Transport#">
  <!ENTITY wsrr "http://www.ibm.com/xmlns/prod/serviceregistry/6/1/model#">
]>

<rdf:RDF
  xmlns:xsd="&xsd;"
  xmlns:rdf="&rdf;"
  xmlns:rdfs="&rdfs;"
  xmlns:owl="&owl;"
  xmlns:ns_transport="&ns_transport;"
  xmlns:wsrr="&wsrr;"
>

  <owl:Ontology rdf:about="&ns_transport;TransportOntology">
    <owl:imports rdf:resource="http://www.ibm.com/xmlns/prod/serviceregistry/6/1/model"/>
    <wsrr:prefix rdf:datatype="http://www.w3.org/2001/XMLSchema#string">transport</wsrr:prefix>
    <rdfs:label>A transport classification system.</rdfs:label>
    <rdfs:comment>Cars and buses and some superclasses.</rdfs:comment>
  </owl:Ontology>

  <owl:Class rdf:about="&ns_transport;Transport">
    <rdfs:label>Transport</rdfs:label>
    <rdfs:comment>Top-level root class for transport.</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="&ns_transport;LandTransport">
    <rdfs:subClassOf rdf:resource="&ns_transport;Transport"/>
    <rdfs:label>Land Transport.</rdfs:label>
    <rdfs:comment>Middle-level land transport class.</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="&ns_transport;AirTransport">
    <rdfs:subClassOf rdf:resource="&ns_transport;Transport"/>
    <rdfs:label>Air Transport.</rdfs:label>
    <rdfs:comment>Middle-level air transport class.</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="&ns_transport;Bus">
    <rdfs:subClassOf rdf:resource="&ns_transport;LandTransport"/>
    <rdfs:label>Bus.</rdfs:label>
    <rdfs:comment>Bottom-level bus class.</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="&ns_transport;Car">
    <rdfs:subClassOf rdf:resource="&ns_transport;LandTransport"/>
    <rdfs:label>Car.</rdfs:label>
    <rdfs:comment>Bottom-level car class.</rdfs:comment>
  </owl:Class>


</rdf:RDF>
```

Figure 2-7 Sample OWL Content

2.11 Ontology Generation

The process of ontology generation is shown in Figure 2-6.

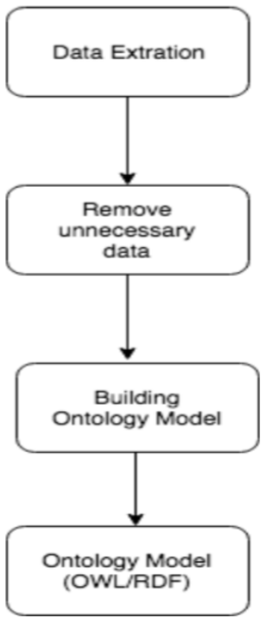


Figure 2-8 Ontology Generation Process

The data extraction is relevant to specific domain such as software process domain, and healthcare domain, etc. Several techniques are used to extract the data such as a python script, and apache Jena API which is open source and semantic framework. Apache Jena is limited with Java language and provides data extraction as well as generating RDF and OWL files using extracted data. The limitation of Apache Jena narrates that it can unable to generate OWL2 which is the latest ontology web language. Data extraction process can be automated using Apache Jena. A sample scenario is described in figure 2-6. Data in extraction process can be XML, JSON, or text. The sources of data may include lot of unwanted stuff which is needed to be filtered out by the programmer.

Sometimes the fields to be extracted from sources would be changed based on the requirements in time to time. So, the programmer needs to change the program logic to extract the data. Data format of source files might be changed completely to a different format then the programmer spends a huge effort to change the program logic to extract data.

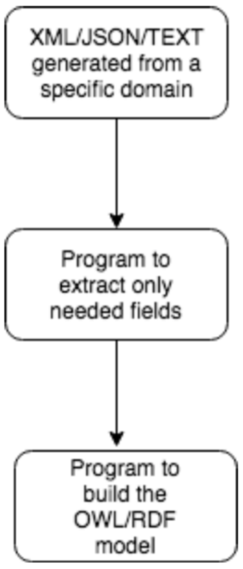

XML/JSON/TEXT generated from a specific domain
Program to extract only needed fields
Program to build the OWL/RDF model

Figure 2-9 Data Extraction Process Automation Using Apache Jena

The output of the OWL/RDF model which is implemented using Apache Jena, is the real OWL or RDF file. Apache Jena also can be used to read an OWL or RDF file to query for further analysis. Other than Apache Jena, protégé tool can be used to generate OWL files.

The following diagram describes a sample ontology modelling scenario of family ontology [29]. A family may contain the roles such as mother, father, son, daughter, grandfather, uncle and aunt. The main super class of any entity is the Person class which has two sub classes such as Man and Woman. How human entities are related to each other is shown by figure 2-8.

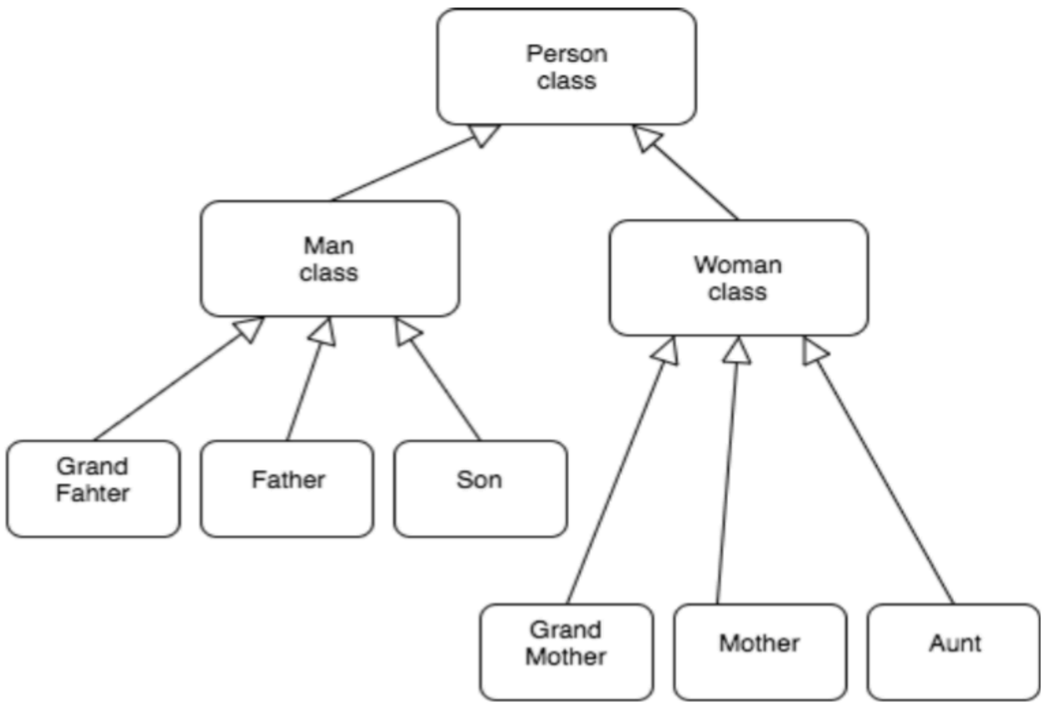


Figure 2-10 Class Level Hierarchy in Family Ontology

After setting all needed classes in the family ontology, properties (data types and objects) need to be defined with their domain and range values. The user defined classes (Grandfather, Father, Uncle, Son) are the sub classes of Man class. So, their domain must be Man class and their range must be Person class. Special case for Nephew is a subclass from both Man class and Woman class. So, its domain and range classes should be Person class. The domain for data types properties must be Person class and the range must be literal values such as integer, and string. Sample object properties for family ontology is given below.

isHuband – the domain is Man and the range would be Woman

isSon – the domain is Man and the range would be Person

Data type properties somewhat like;

hasName – the domain is Man and the range would be String

hasAge – the domain is Man and the range would be integer

The next step would be the process of adding the individuals for the classes in family ontology which indicates adding real values for each property. Protégé tool can be used to create above family ontology then ontology visualization can be illustrated from 'Jambalaya' tab [30].

2.12 Ontology Supporting Tools

Protégé is the most popular free and open source ontology editor and consists of user-friendly interfaces to interact with users easily. Protégé is developed by Stanford university in 2013. Both industry and the academic utilize protégé tool to build ontology and analyse the existence of ontologies. Complex and simple ontologies can be illustrated using protégé tool. Other than above functionalities, third party libraries (jar files) can be added to the protégé tool such as JESS (Java Expert System Shell) jar for rule-based reasoning.

Several tables such as 'Jambalaya' which is an inbuilt plugin and 'OntoGraf', can be used to visualize the generated ontology. Figure 2-9 shows sample ontology visualization using 'Jambalaya' tab in protégé.

Other than protégé tool, some other tools are available as ontology editors such as Neon Toolkit [31] which is specially designed for heavy-weight project, SWOOP [32] which is for small domain projects and OWLGrEd [33] which is a graphical ontology editor for OWL.

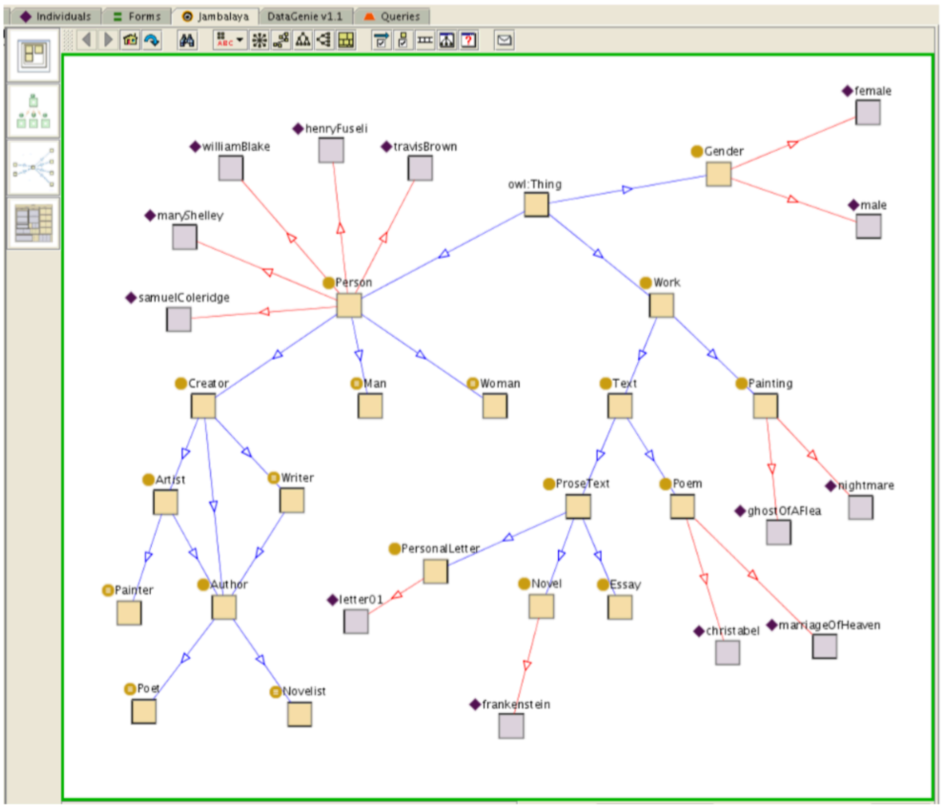


Figure 2-11 Sample Visualization for People Ontology Using 'Jambalaya' Tab in Protégé

After conducting the literature, it is obvious to say that ontologies are the most suitable knowledge bases for the stated research problem.

CHAPTER 3

METHODOLOGY

This chapter describes how ontology can be used to retrieve the architectural design decisions from an ontology-based knowledge management system. To decide which design architecture is best for the system implementation, a broad software design experience and knowledge are the essential facts. So, this research is intending to implement the ontology-based knowledge management system to fulfil the experience gap of software designers.

3.1 Qualities of a Software Architecture

The non-functional requirements often create the quality requirements for a software system. Some of the main quality requirements for a software system are [34];

- Functionality
    - Software system should meet the stated functionalities.
- Reliability
    - Ability of a software system functions under stated conditions.
- Usability
    - How end users learn and understand the software system.
- Efficiency
    - About the performance of the software system.
- Maintainability
    - Ability to modify the software system with less effort.
- Portability
    - Ability to transfer the software system to another environment.

A good software architecture design should balance these quality attributes.

3.2 Proposed Knowledge Management System

The proposed system is the ontology-based knowledge management system for architectural design decisions. The architectural design decisions are impacted by organizational changes, development methodologies, cost, time to complete and the stakeholders. Some decisions may relate to other architectural decisions and those relations can be categorized based on how they tightly coupled with each decision.

At the first stage, the system need to store the information of each architectural decision for a software system [2]. That information are the properties of the OWL class which is represented an architectural design decision. Stored information can be illustrated as below.

- Architectural decision identifier
- Architectural decision title
- The rationale behind the decision
- Scope
- Cost
- Risk

Other than above information, there will be more points if other facts are important to get the architectural design decision.

3.3 Automate the Proposed Knowledge Management System

At the first stage, the experience or the knowledge should be gathered from experienced developers or system architects. In order to gather such information, a web user interface needs to be implemented. Therefore, the end user can insert his/her knowledge through the web interface easily. Then collected information will be stored in a database to retrieve for further processing.

After that, an OWL file needs to be generated using apache Jena API with stored information, then OWL file helps to analysis or process the data retrieval activity. Another web interface needs to be implemented to retrieve the data upon querying the ontology using SPARQL. That web interface may involve simple and advanced search functionalities to retrieve the relevant information.

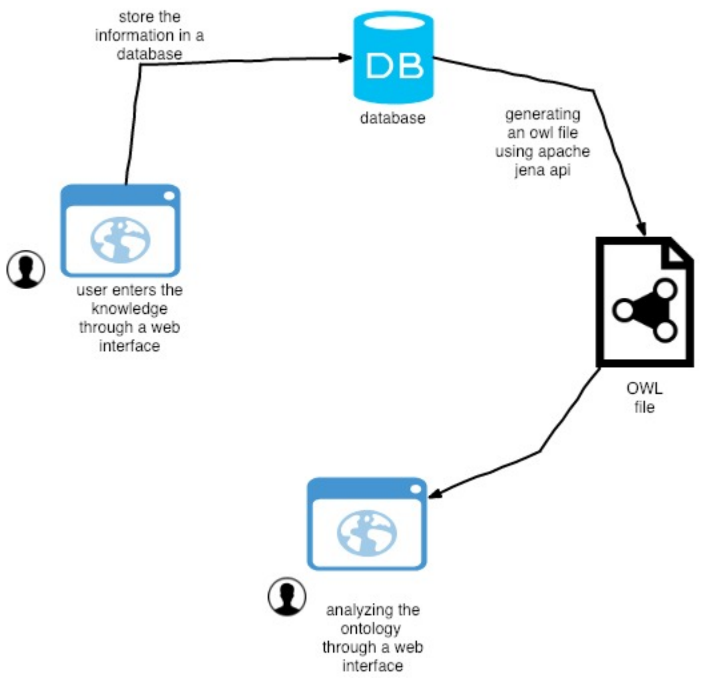

store the information in a database
database
generating an owl file using apache jena api
user enters the knowledge through a web interface
OWL file
analyzing the ontology through a web interface

Figure 3-1 : flow of developing the ontology-based knowledge management system.

Finally, iterative process will be selected to develop the ontology-based knowledge management system for the proposed solution.

CHAPTER 4

SOLUTION ARCHITECTURE AND IMPLEMENTATION

Ontology development is a creative process as well as there is no proper and correct methodology to develop an ontology system [35]. There are some similarities as well as differences between ontology development and object-oriented development. In both cases, classes and relations need to be defined but reasons of selecting a class in Object Oriented Programming (OOP) are different than ontology development. OOP designs are based on the operational properties while Ontology designs are based on structural properties. In general, concepts are considered as classes in ontology designing. After reading several literatures, they illustrated that iterative process is the most applicable methodology to develop an ontology.

Practically, below abstract steps need be followed to develop an ontology system.

- Identifying and defining the classes in ontology.
- Define the class taxonomic. (class hierarchy)
- Define class properties with their allowed values.
- Define the instances for relevant classes.

When designing the application, it should be more extendable, intuitive and maintainable. This research has followed some steps to design the model of ontology which lines with above general steps.

4.1 Steps to Model the Ontology

Mainly 5 steps were followed to implement the overall ontology system. First two steps are the most time-consuming steps since if there is a mistake happened in those steps, this study need to revise of those two steps.

4.1.1 Step 1 - Determine the Domain and the Scope

This is the initial and important step and affects the overall ontology system. The domain of this research is only about software architectural knowledge. Again, a software architectural knowledge is a vast area. Therefore, this research has chosen a little part of it which is only aided to the architectural design decision from several stakeholders. When determining the domain and scope, there are some questions to answer.

1. For what purpose, we are going to use the ontology.

This ontology system is used to gain the appropriate architectural solutions for similar architectural design issues.

2. Who are the users of ontology system.

The people who design the architecture of a software solution and can be software architect, senior developers, or any person who has the knowledge to design the software architecture.

3. Some questions of end users.

There are some additional questions like competency questions or similar questions from end users.

Ex: what characteristics should I want to consider when choosing a software architecture.

4.1.2 Step 2 - Define the Classes and Taxonomy

This step is a time consuming while this research needs to think of several aspects when defining the classes. Here mainly the concepts of classes generated from the real world. Also, this research might struggle to differentiate classes and properties. Six main classes were identified in out ontology system.

1. Stakeholder
2. Requirement
3. Decision
4. Project
5. Alternative
6. Decision history

When defining the taxonomy of classes, this research can use three main approaches such as top-down, bottom-up and combination of both.

- Top-down
    - It starts with most general or the abstract classes in the ontology domain and then composes into sub classes.
- Bottom-up
    - It starts with specialized classes and then go up.
- Combination
    - It is mixture of both top-down and bottom-up approaches. This is most practical approach to define the taxonomy.

According to requirements of this research, this research do not need the class taxonomy because this research mainly focus on architectural decisions itself.

4.1.3 Step 3 - Define the Properties of Classes

Class properties are the internal structure of class and can be either object properties or data properties. Object properties are the instances of a class. Also, those instances are properties are in another class. Data properties are somewhat like name, colour, age, etc. which may consider as numeric, string, values and Boolean etc. Below illustrated figure 4-1 shows the properties, this research has selected for 'Decision' class of this ontology system.

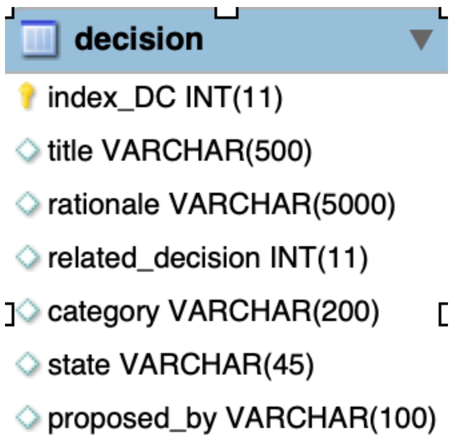

decision
index_DC INT(11)
title VARCHAR(500)
rationale VARCHAR(5000)
related_decision INT(11)
category VARCHAR(200)
state VARCHAR(45)
proposed_by VARCHAR(100)

Figure 4-1 Properties of Decision Class

Each ontology class comprises with unique index to identify each separately. Furthermore, below points describes above properties of 'Decision class'.

- Title
  - Tells the architectural design decisions.
    - Ex: Choosing a database system for a telecommunication system.
- Rationale
  - It is a brief description about why selecting the relevant architectural decision.
- Related decision
  - This points to a similar kind of architectural decision.

- Category
    - According to Kruchten [2], there are three major categories of design decisions such as;
        - Existence decisions
        - Property decisions
        - Executive decisions

      Above categories were described in Chapter 2.
- State
    - Values for 'State' property depend on the organizational wise and this research have selected some values for State property such as;
        - Initial
        - Approved
        - Rejected
        - Obsolesced
            - This is similar to 'Rejected' but cannot be fully rejected. So, it can be considerable.
- Proposed By
    - The stakeholder who propose the architectural decision.

Other properties of other ontology classes are simple, and we can understand them by observing at those property names.

4.1.4 Step 4 - Define the Relationships Between Classes

These relationships are important aspects when we consider implementing the ontology system. Following figure 4-2 shows the classes, attributes and relationships between those classes.

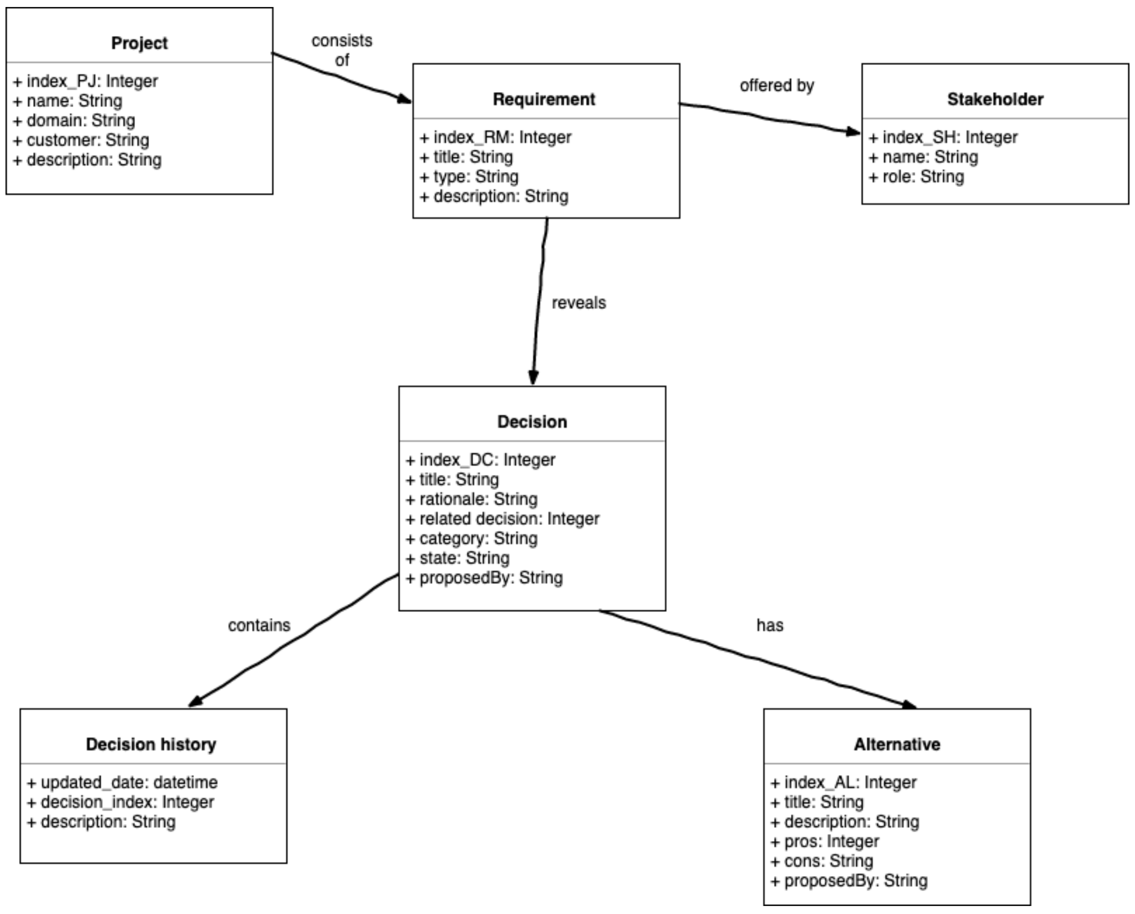


Figure 4-2 Class Diagram of Ontology System

Above provided figure illustrates that a project includes business requirements which is obtained from several stakeholders such as business analysts, project clients, etc. The requirements can be either functional or non-functional. To fulfil those requirements, a proper and precise architecture is needed. Moreover, to design such architecture, a good set of architectural design decisions are needed. One of those architectural decision can be selected from other similar decisions which are called as alternatives. Furthermore, decision history will be maintained separately.

After completing the forth step, implementation of ontology system can be started. To implement such system, several approaches can be used such as tool support like protégé, and program support which is used by a programming language such as Java, PHP.

4.2 Implementing the Ontology System.

For implementing an ontology system, there is no proper standard system architecture. Below figure 4-3 shows how the system architecture is composed for this ontology system.

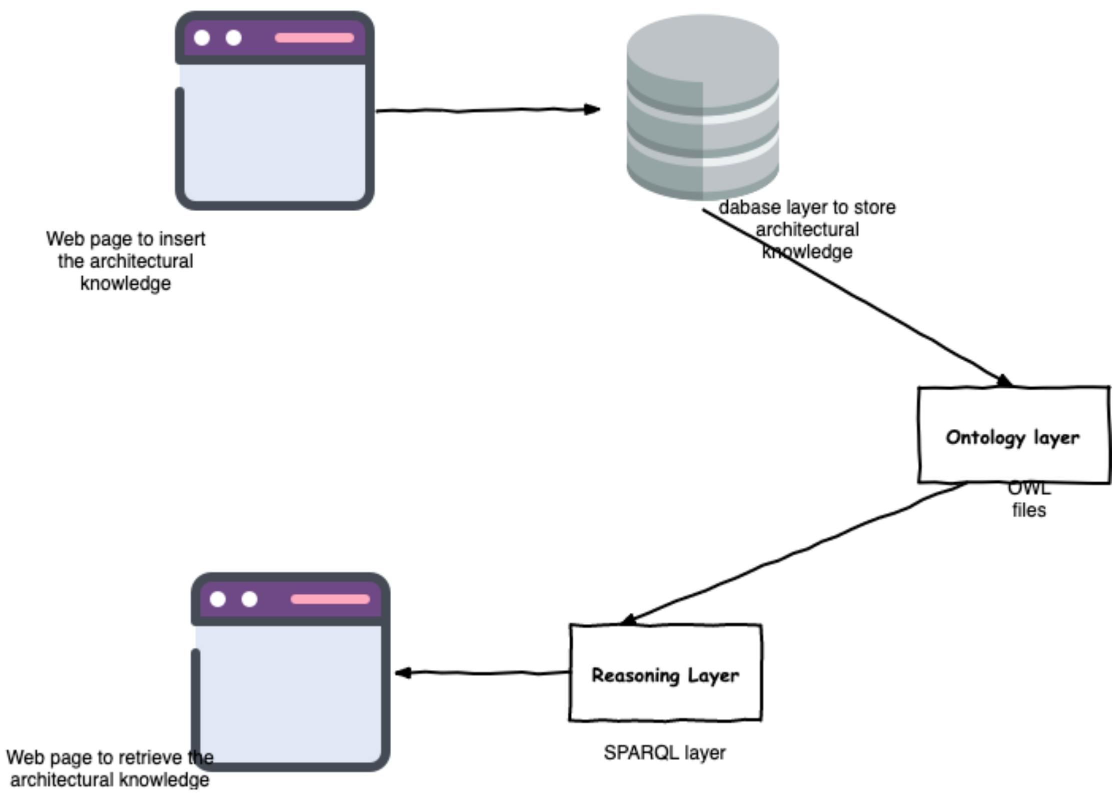
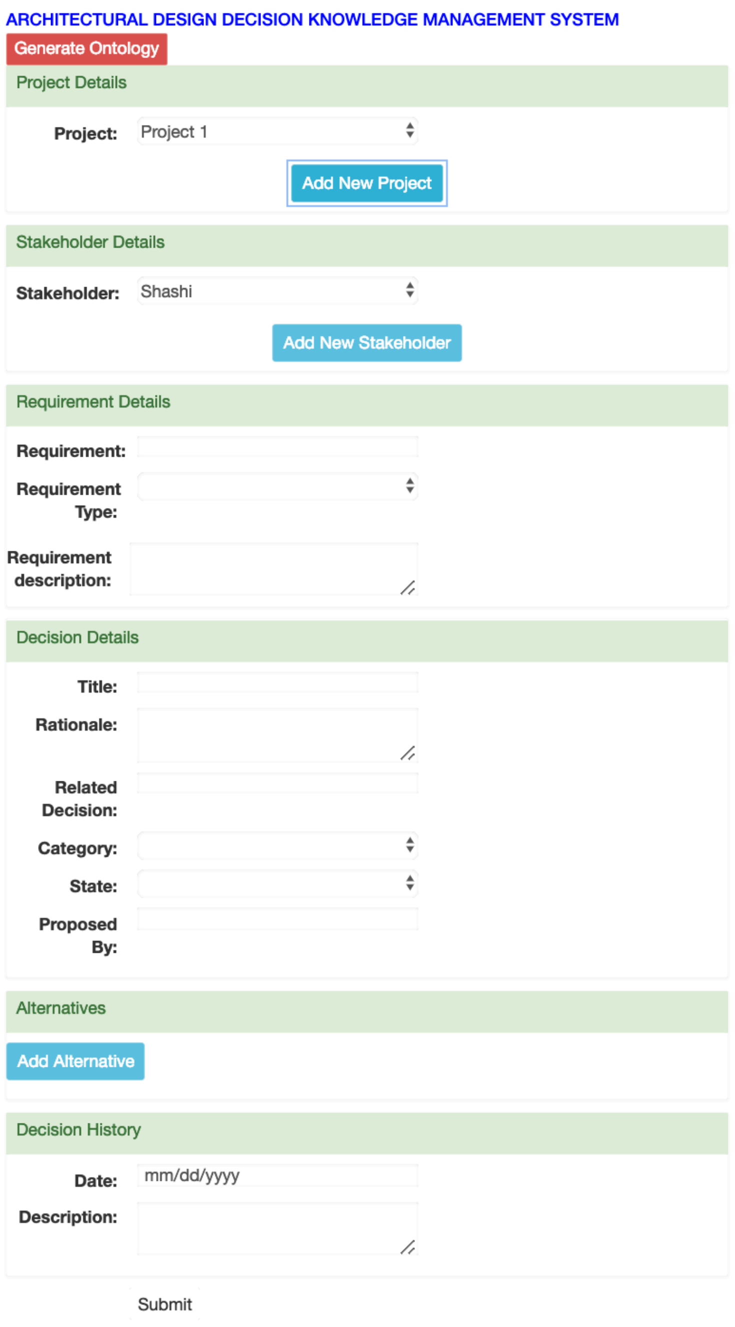

Web page to insert the architectural knowledge
dabase layer to store architectural knowledge
Ontology layer
OWL files
Reasoning Layer
SPARQL layer
Web page to retrieve the architectural knowledge

Figure 4-3 System Architecture for Ontology System

End users of the system can insert their architectural knowledge through the web interface. Those inserted data will be stored in a database. Below illustrated figure 4-4 shows specific web interface to insert data.


ARCHITECTURAL DESIGN DECISION KNOWLEDGE MANAGEMENT SYSTEM

Generate Ontology

Project Details

Project:	Project 1

Add New Project

Stakeholder Details

Stakeholder:	Shashi

Add New Stakeholder

Requirement Details

Requirement:	
Requirement Type:	
Requirement description:	

Decision Details

Title:	
Rationale:	
Related Decision:	
Category:	
State:	
Proposed By:	

Alternatives

Add Alternative

Decision History

Date:	mm/dd/yyyy
Description:	

Submit

Figure 4-4 Web Interface to Insert Architectural Design Decisions

According to the above web interface, this research firstly needs to select a project from the drop down filed. Otherwise, web user can insert a new project via another web interface which is shown in figure 4-5.

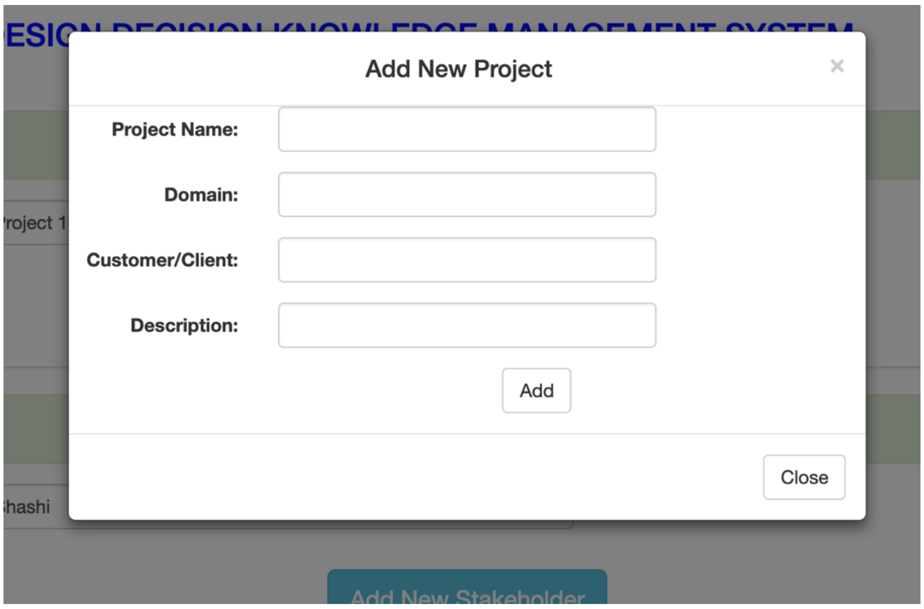


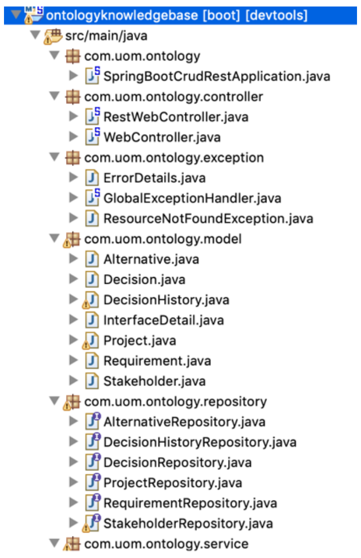
Figure 4-5 Web Interface to Insert New Project Details

After selecting the project, stakeholder needs to be selected or else a new stakeholder can be inserted into the system like inserting a new project previously. Then requirement details for the architectural decision needs to be inserted with the decision details. Finally, alternatives and decision history details are to be inserted for generating the ontology.

4.2.1 Technology Stack Used to Implement the Ontology System.

As previously mentioned, ontology systems can be implemented by using either tool support or program support. If tool support is selected to implement the ontology, then the ontology developer needs to learn the tool also. Therefore, for implementing this ontology system, we choose the Apache Jena [22] because it is convenient for any programmer. To develop the web application which includes web interfaces to insert and retrieve knowledge, Spring Boot [36] framework is used because it is less configurable, embedded with apache tomcat [37] web server and easy to combine with other systems such as database systems. To build the project and to manage third party libraries, apache maven [38] is used. But it is not just a build tool. It supports for entire life cycle for project development. AngularJS version 1 [39] is used to implement front-end the web application which has main two interfaces to insert and retrieve the knowledge. Single Page Application development is the main reason is to choose the AngularJS framework and easy for validation and integration with the backend services.

System code base is packaged according the layers such as model, service, repository and service layers and easy to maintain and keep clean code. Below mentioned figure 4-6 shows how the source code is wrapped into packages.



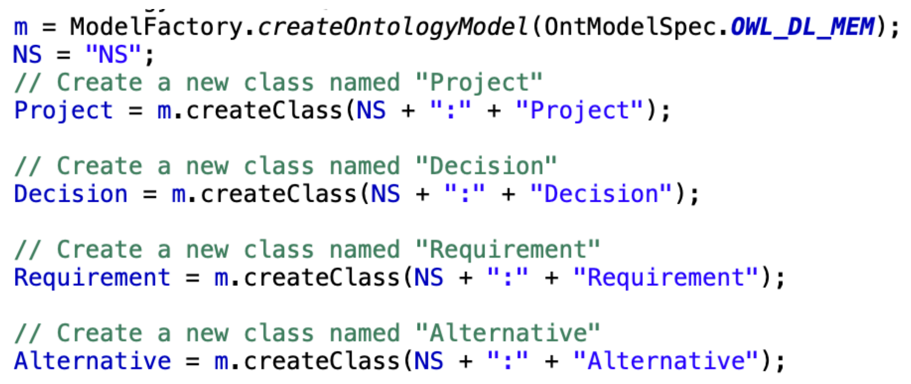
```
ontologyknowledgebase [boot] [devtools]
  src/main/java
    com.uom.ontology
      SpringBootCrudRestApplication.java
    com.uom.ontology.controller
      RestWebController.java
      WebController.java
    com.uom.ontology.exception
      ErrorDetails.java
      GlobalExceptionHandler.java
      ResourceNotFoundException.java
    com.uom.ontology.model
      Alternative.java
      Decision.java
      DecisionHistory.java
      InterfaceDetail.java
      Project.java
      Requirement.java
      Stakeholder.java
    com.uom.ontology.repository
      AlternativeRepository.java
      DecisionHistoryRepository.java
      DecisionRepository.java
      ProjectRepository.java
      RequirementRepository.java
      StakeholderRepository.java
    com.uom.ontology.service
```

Figure 4-6 Package Structure of Source Code

Inserted data will be stored in MySQL [40] database with suitable relationships among entities, which are mapped from ontology classes.

4.2.1 OWL File Generation

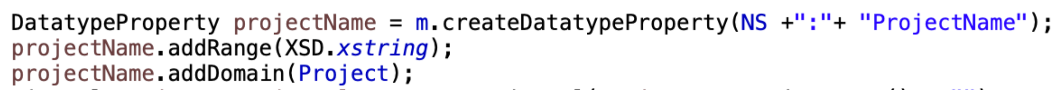
Apache Jena API provides simple methods to implement ontology classes, their properties and the relationships. First, we need to implement ontology classes. Below figure 4-7 indicates the java source code for ontology classes.



```
m = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM);
NS = "NS";
// Create a new class named "Project"
Project = m.createClass(NS + ":" + "Project");

// Create a new class named "Decision"
Decision = m.createClass(NS + ":" + "Decision");

// Create a new class named "Requirement"
Requirement = m.createClass(NS + ":" + "Requirement");

// Create a new class named "Alternative"
Alternative = m.createClass(NS + ":" + "Alternative");
```

Figure 4-7 Defining Ontology Classes Through Jena API

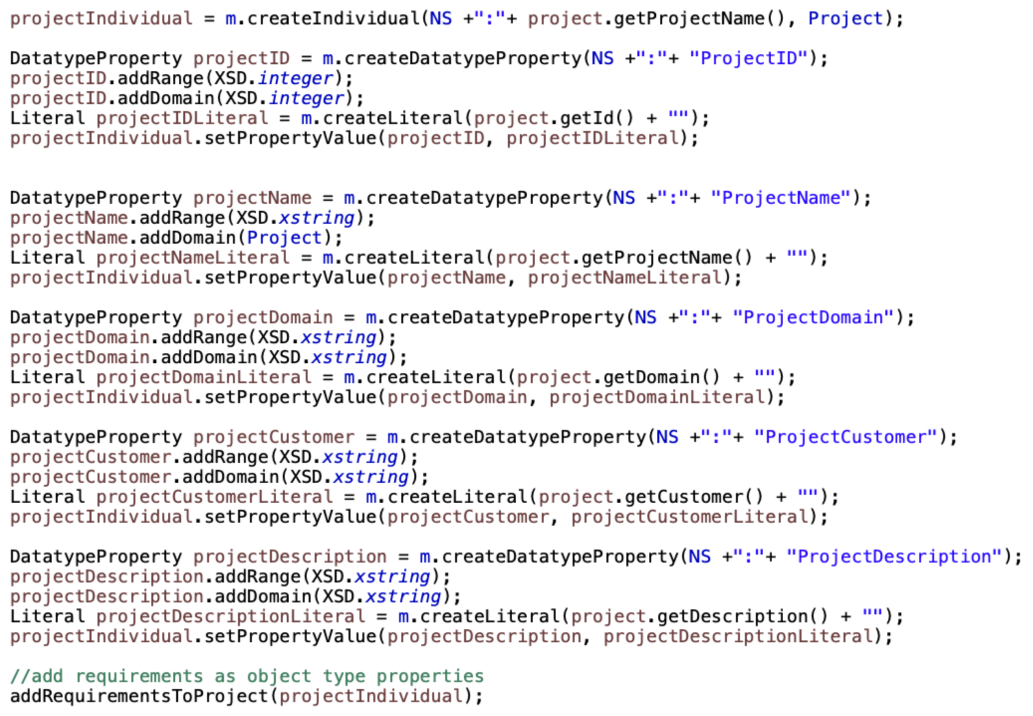
After defining the ontology classes, data properties and objects properties need to be defined with their domain and range values. Below figure 4-8 shows that how to define a data property for ‘Project’ class.



```
DatatypeProperty projectName = m.createDatatypeProperty(NS +":"+ "ProjectName");
projectName.addRange(XSD.xstring);
projectName.addDomain(Project);
```

Figure 4-8 Defining the Data Property for Project class

After defining the ontology classes using Jena API, individuals need to be created. Here individuals are somewhat similar to the objects in OOP programming. Below figure 4-9 shows that how to initiate an individual for ‘Project class’ using Jena API. In this system, the data which creates individuals, are stored in the database. Moreover, after creating individuals, an OWL file is created which represents the software architectural design decision ontology.



```java
projectIndividual = m.createIndividual(NS +":"+ project.getProjectName(), Project);

DatatypeProperty projectID = m.createDatatypeProperty(NS +":"+ "ProjectID");
projectID.addRange(XSD.integer);
projectID.addDomain(XSD.integer);
Literal projectIDLiteral = m.createLiteral(project.getId() + "");
projectIndividual.setPropertyValue(projectID, projectIDLiteral);


DatatypeProperty projectName = m.createDatatypeProperty(NS +":"+ "ProjectName");
projectName.addRange(XSD.xstring);
projectName.addDomain(Project);
Literal projectNameLiteral = m.createLiteral(project.getProjectName() + "");
projectIndividual.setPropertyValue(projectName, projectNameLiteral);

DatatypeProperty projectDomain = m.createDatatypeProperty(NS +":"+ "ProjectDomain");
projectDomain.addRange(XSD.xstring);
projectDomain.addDomain(XSD.xstring);
Literal projectDomainLiteral = m.createLiteral(project.getDomain() + "");
projectIndividual.setPropertyValue(projectDomain, projectDomainLiteral);

DatatypeProperty projectCustomer = m.createDatatypeProperty(NS +":"+ "ProjectCustomer");
projectCustomer.addRange(XSD.xstring);
projectCustomer.addDomain(XSD.xstring);
Literal projectCustomerLiteral = m.createLiteral(project.getCustomer() + "");
projectIndividual.setPropertyValue(projectCustomer, projectCustomerLiteral);

DatatypeProperty projectDescription = m.createDatatypeProperty(NS +":"+ "ProjectDescription");
projectDescription.addRange(XSD.xstring);
projectDescription.addDomain(XSD.xstring);
Literal projectDescriptionLiteral = m.createLiteral(project.getDescription() + "");
projectIndividual.setPropertyValue(projectDescription, projectDescriptionLiteral);

//add requirements as object type properties
addRequirementsToProject(projectIndividual);
```

Figure 4-9 Source Code for 'Project' individual

4.2.1 Implementation of Data Retrieval Functionality

To retrieve the information from ontology system, SPARQL (Protocol and RDF Query Language) is used. SPARQL is mainly a query language for semantic web. It is not much difficult to learn SPARQL and is similar to SQL queries. Triple patterns are somewhat similar to RDF graph triple. Those patterns are the basic component for SPARQL. Basic structure for the ‘SELECT’ statement is;

```
SELECT <variables>

WHERE {

        <graph pattern>

}
```

Similar to SQL, SPARQL 'SELECT' queries can be embedded with 'WHERE', 'ORDER BY', 'LIMIT' and mathematical operations. The main purpose and functionality of this ontology system is the sharing of knowledge among the relevant people such as software design architects, and senior developers etc. To accomplish this purpose, two main web interfaces including Basic Search and Advanced Search, are developed. Below figures 4-10 and 4-11 illustrate the user interfaces with certain search functionalities.

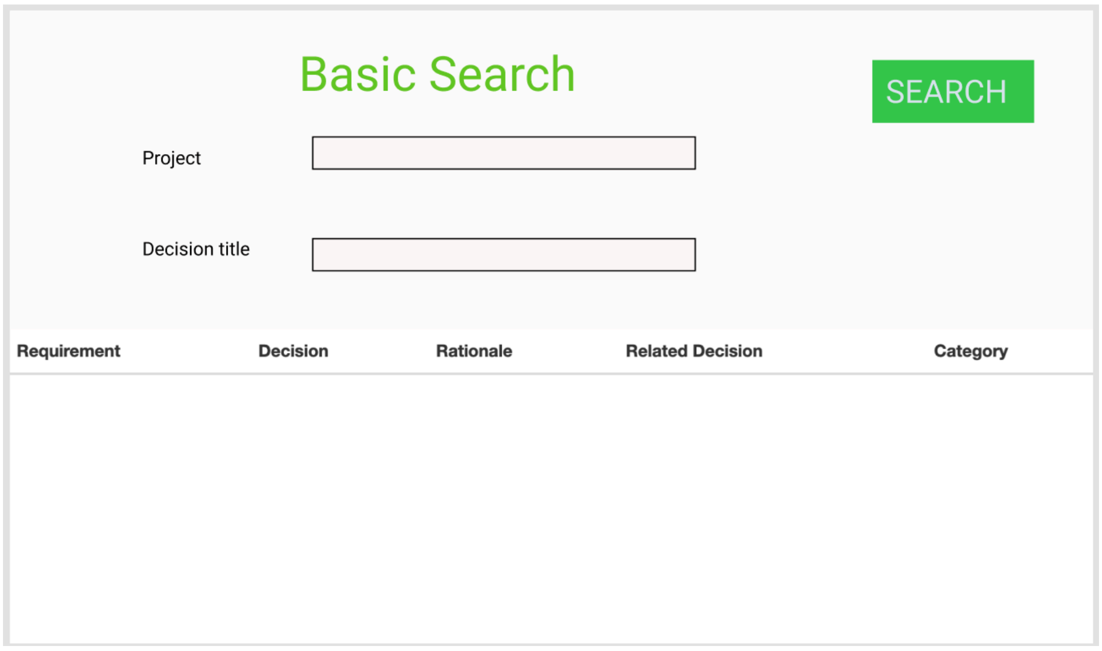


Figure 4-10 User Interface for Basic Search

There are two text boxes which has two labels that are ‘Project’ and ‘Decision title’. Those two text boxes are capable of auto filtering when the user starts typing. After clicking the SEARCH button, it will filter design decisions from this ontology system. The system is capable for searching only by project wise or decision wise or both values.

Advanced Search functionality is another a web interface and similar to the Basic Search. But it consists of three more additional fields that are ‘Stakeholder’, ‘Requirement’, and ‘Proposed By’. Through the ‘Advanced Search’, the system operates more filtering process for results from Architectural Knowledge Management System of this research’s system. Figure 4-11 shows that how ‘Advanced Search’ User Interface is designed.

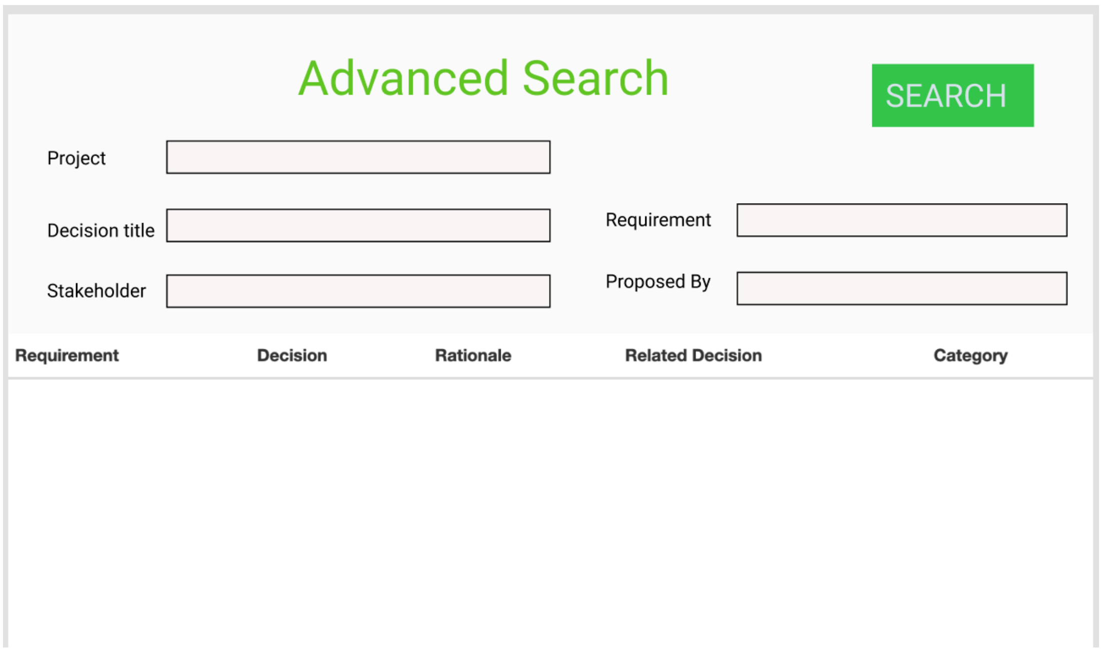


Figure 4-11User Interface for Advanced Search

Instead of using a programming language, protégé tool can be used to retrieve the architectural knowledge from an ontology system. But there is a requirement for the knowledge of using the protégé tool. After implementing this ontology system, end-user can only interact with web interfaces which is more user-friendly for them. Few of java classes for implemented system are included in Appendix C.

CHAPTER 5

EVALUATION

The implemented system and the ontology are evaluated in this chapter. When considering the ontology evaluation, there are two important aspects such as quality and correctness [41]. Quality of ontology combines with ontology verification and ontology validation. Ontology verification indicates building an ontology system correctly. Likewise, ontology validation indicates building the correct ontology system. Evaluation mainly depends on metrics. Those metrics provide the comparable results which help to get the decisions on the evaluated system.

There are three main ontology types [41]:

- Structural metrics
  - Syntax and semantics are concerned.
- Functional metrics
  - Intended usage and components of ontology are concerned.
- Usability profiling
  - Communication aspects of ontology are concerned.

When considering the structural metrics, Chapter 4 describes that how syntax and semantics are selected on the specific domain in this research.

5.1 Structural Evaluation

Implemented ontology (i.e. OWL file) will be evaluated through protégé tool to check whether it contains correct semantics and syntaxes.

5.1.1 Ontology Classes

After uploading the .owl file to the protégé tool, it shows the ontology classes which are implemented programmatically. Below figure 5-1 shows that how they are illustrated in class tab in protégé.

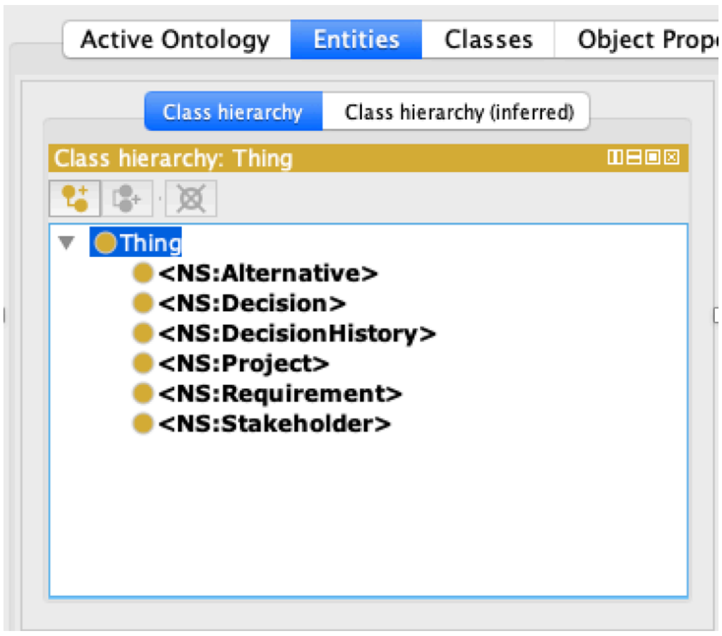


Figure 5-1 Ontology Class in Protégé

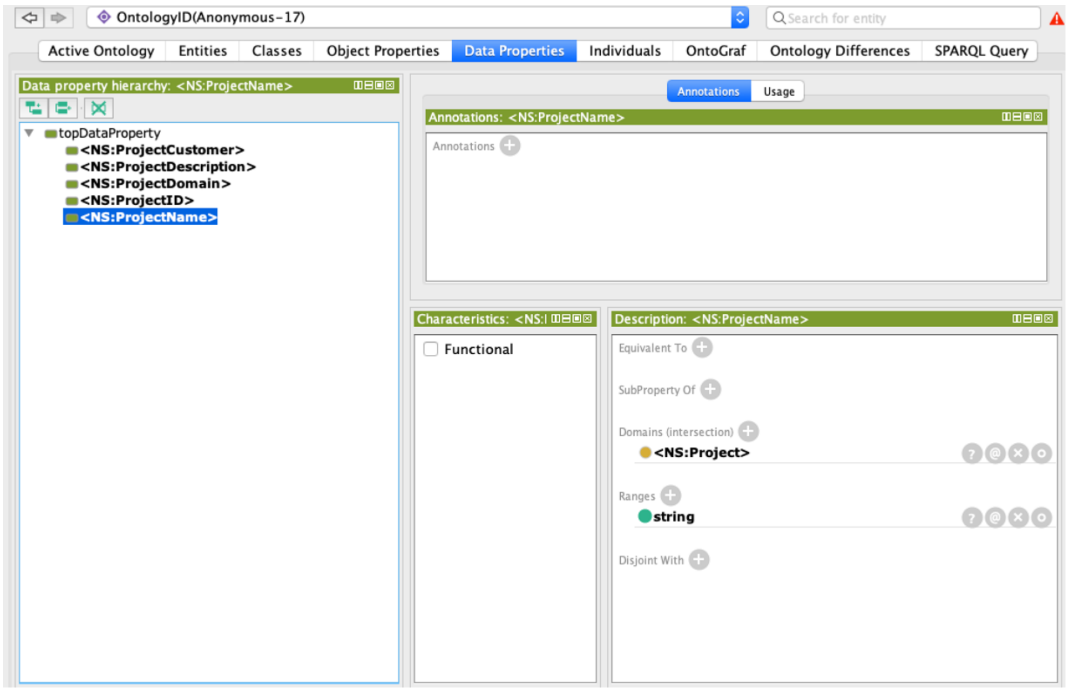
Implemented solution encompasses with six main ontology classes such as:

- Alternative
- Decision
- Decision History
- Project
- Requirement
- Stakeholder

And protégé tool also is illustrated same ontology classes. Therefore, the ontology class implementation is valid according to the above evidence.

5.1.2 Ontology Data Properties

For the 'Project' ontology class authors have defined some data properties such as ProjectID, and ProjectName, etc. Data property tab in the protégé tool shows the properties with the domain and range values in ‘Project’ ontology class and indicates that the ontology implementation is valid with the data properties. Such kind of sample is illustrated in below figure 5-2.



Data property hierarchy: <NS:ProjectName>

- topDataProperty
  - <NS:ProjectCustomer>
  - <NS:ProjectDescription>
  - <NS:ProjectDomain>
  - <NS:ProjectID>
  - <NS:ProjectName>

Annotations: <NS:ProjectName>

Annotations

Characteristics: <NS:I

Functional

Description: <NS:ProjectName>

Equivalent To

SubProperty Of

Domains (intersection)
<NS:Project>

Ranges
string

Disjoint With

Figure 5-2 Data Properties with Domains and Ranges

5.1.3 Individuals in Ontology

Ontology individuals are the real data values of the ontology classes and similar to the Objects and Classes in Object Oriented Programming. Protégé tool has a tab which is called ‘Individual’ and shows the read data values relevant to the ontology class and properties. Following figures (5-3 and 5-4 figures) show a sample of an individual of the uploaded .owl file.

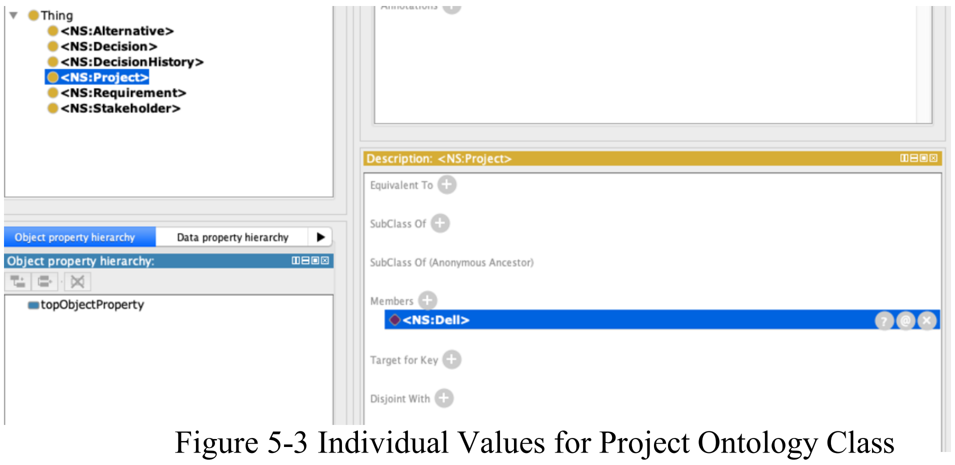


Figure 5-3 Individual Values for Project Ontology Class

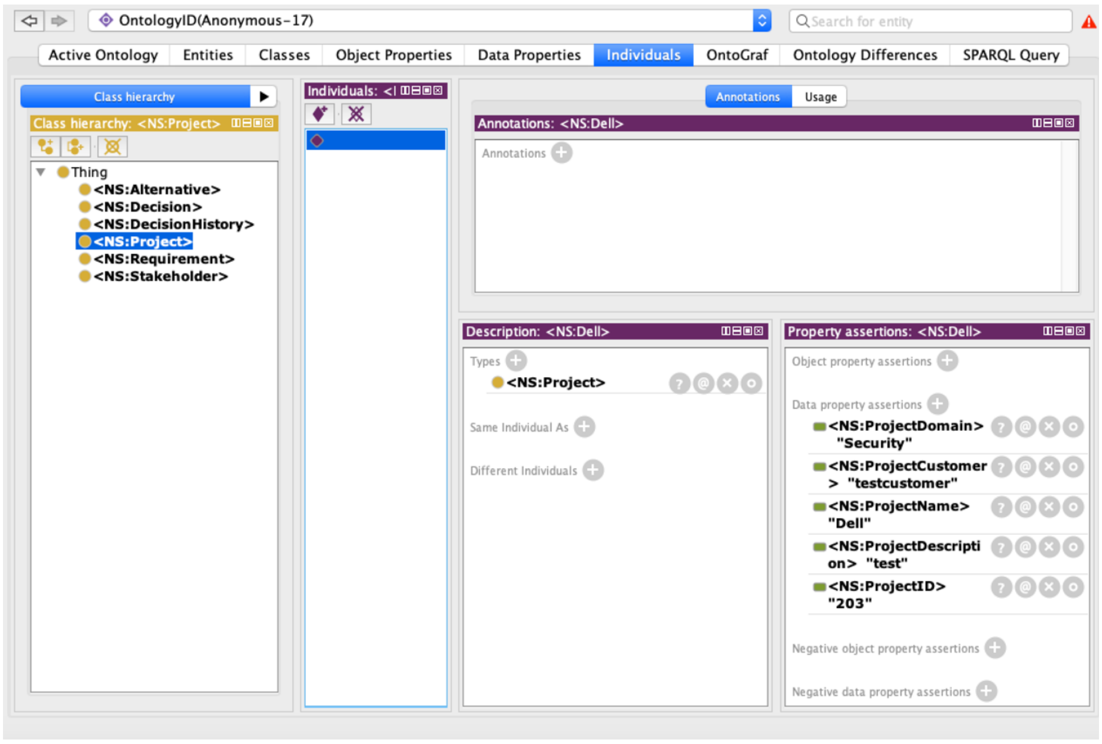


Figure 5-4 Individual Property Values for Project Class

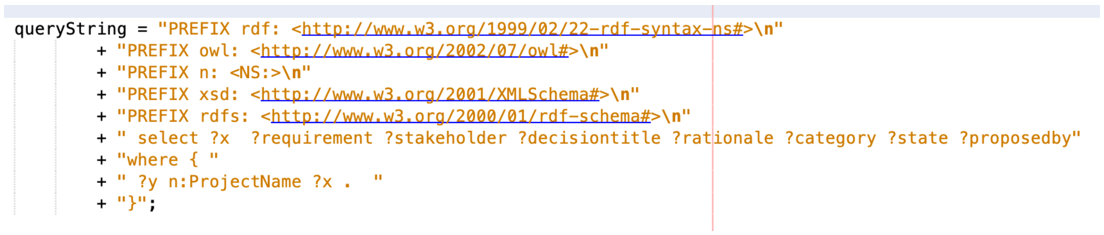
After analysing the .owl file with the protégé tool, it is confirmed that the implemented ontology is semantically and syntactically correct.

5.2 Evaluate the system performance

Main objective of this research is to provide a knowledge base for architectural design decisions which is the intended usage of the solution that is provided by the authors. After feeding the data to the system, it creates an owl file which is the architectural design decision ontology. Then, the owl file is evaluated by a search query to visualize the architectural knowledge which is the intended usage of ontology. Different sizes of owl files are evaluated by their average executed times. Hardware specification of the computer system is listed below and has been used to evaluate the ontological knowledge base.

- Processor: 2.7 GHz dual-core Intel Core i5 processor
- RAM: 8 GB
- Operating System: MacOS Mojave

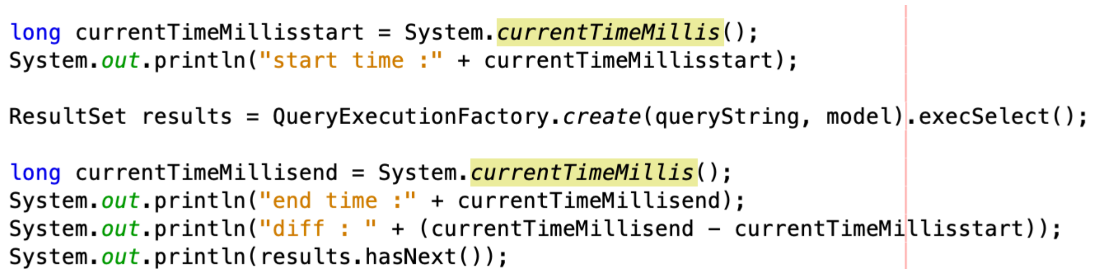
A SPARQL is used to get the relevant architectural knowledge. Following figure 5-5 illustrates it.



```
queryString = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n"
        + "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n"
        + "PREFIX n: <NS:>\n"
        + "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n"
        + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n"
        + " select ?x  ?requirement ?stakeholder ?decisiontitle ?rationale ?category ?state ?proposedby"
        + "where { "
        + " ?y n:ProjectName ?x .  "
        + "}";
```

Figure 5-5 SPARQL Search Query

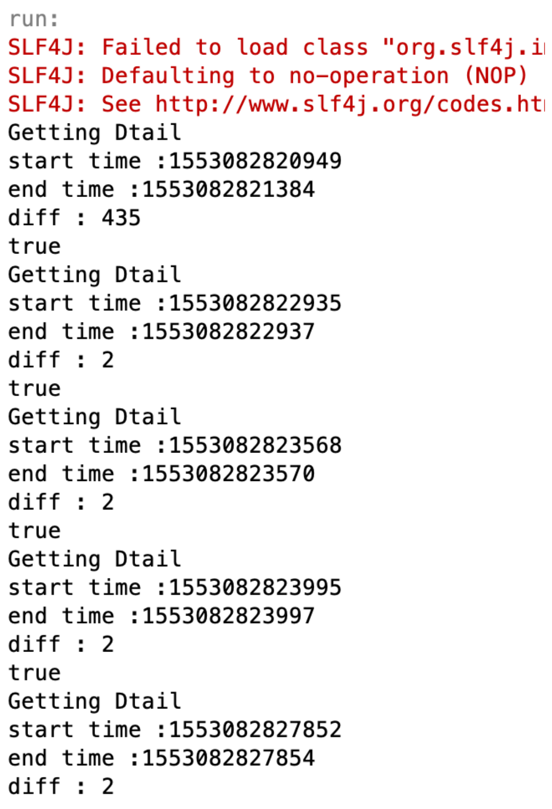
Some lines of java added to the program file and gets the execution time in milliseconds. Following figure 5-6 shows that how authors calculate the execution time through programmatically.



```
long currentTimeMillisstart = System.currentTimeMillis();
System.out.println("start time :" + currentTimeMillisstart);

ResultSet results = QueryExecutionFactory.create(queryString, model).execSelect();

long currentTimeMillisend = System.currentTimeMillis();
System.out.println("end time :" + currentTimeMillisend);
System.out.println("diff : " + (currentTimeMillisend - currentTimeMillisstart));
System.out.println(results.hasNext());
```

Figure 5-6 Calculate Execution Time Programmatically

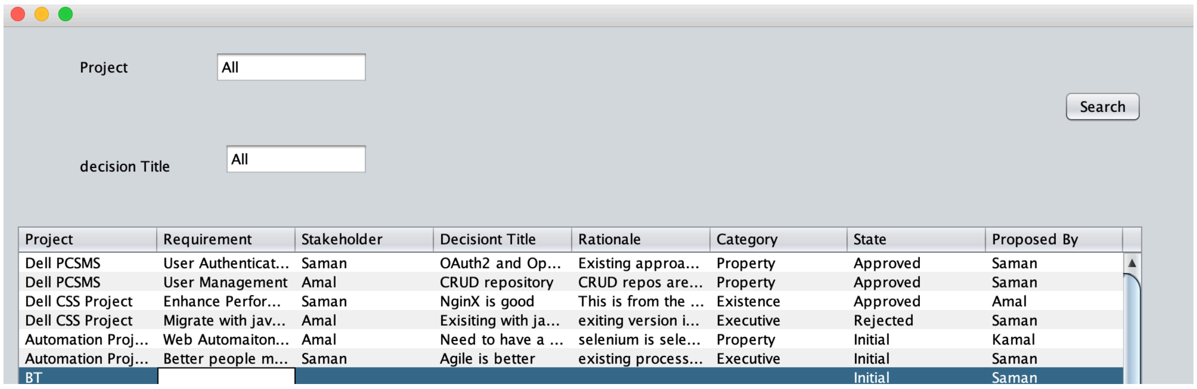
Three consecutive execution times are added, then obtain the average time as the execution time to below chart and figure 5-3 indicates some consecutive execution time. The initial time massively distinguish compared to the other execution times although it holds the ontology loading time to the system as well.



```
run:
SLF4J: Failed to load class "org.slf4j.i
SLF4J: Defaulting to no-operation (NOP)
SLF4J: See http://www.slf4j.org/codes.ht
Getting Dtail
start time :1553082820949
end time :1553082821384
diff : 435
true
Getting Dtail
start time :1553082822935
end time :1553082822937
diff : 2
true
Getting Dtail
start time :1553082823568
end time :1553082823570
diff : 2
true
Getting Dtail
start time :1553082823995
end time :1553082823997
diff : 2
true
Getting Dtail
start time :1553082827852
end time :1553082827854
diff : 2
```

Figure 5-7 Consecutive Execution Times on Search Functionality

After executed the search query, the results are located into a table which contains the important details of an architectural design decisions. Figure 5-4 shows sample results of a SPARQL query.



Project	Requirement	Stakeholder	Decisiont Title	Rationale	Category	State	Proposed By
Dell PCSMS	User Authenticat...	Saman	OAuth2 and Op...	Existing approa...	Property	Approved	Saman
Dell PCSMS	User Management	Amal	CRUD repository	CRUD repos are...	Property	Approved	Saman
Dell CSS Project	Enhance Perfor...	Saman	NginX is good	This is from the ...	Existence	Approved	Amal
Dell CSS Project	Migrate with jav...	Amal	Exisiting with ja...	exiting version i...	Executive	Rejected	Saman
Automation Proj...	Web Automaiton...	Amal	Need to have a ...	selenium is sele...	Property	Initial	Kamal
Automation Proj...	Better people m...	Saman	Agile is better	existing process...	Executive	Initial	Saman
BT						Initial	Saman

Figure 5-8 A Sample Results of a SPARQL Query

Five different sizes of owl files have been selected to evaluate the ontology, and those owl details are listed below table.

Table 5-1 OWL File Sizes Over Execution Times

OWL file size (KB)	Execution time (milliseconds)	Number of decisions
16	3	1
25	2.6	25
36	2.8	50
52	3.6	75
80	3.6	100

The results shown in table 5-1 are converted into a bar chart which have x axis labelled as OWL file size and y axis labelled as execution times in milliseconds. Following chart (figure 5-9) illustrates relevance to such details. There are no ample differences in execution time with increasing the size owl file. Therefore, it indicates that the performance of the implemented ontology based on architectural knowledge management system is better than even the bulk data of knowledge. It might be an impact if the owl file includes minimum 20 megabytes to execute the SPARQL query, but it may require more than one million records to reach certain size.

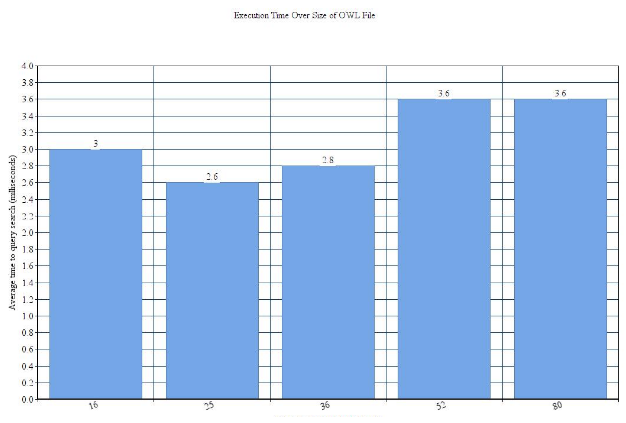


Figure 5-9 Execution Times Over Size of OWL Files

5.3 System Evaluation

System evaluation is divided into two main strategies that are functionality evaluation and user interface evaluation. Both strategies are evaluated through questionnaires which is given to the users.

5.3.1 Functionality Evaluation

Ten people were selected who has a knowledge about ontology concepts and the tools like protégé and Apache Jena. They were allowed to go through the implemented system and give their feedbacks to the questionnaire (Appendix A). Questionnaire is constructed with several questions. Five of them are selected as scenarios. Those scenarios are evaluated and create an impact to functionalities of the system. Each question is made with scale which is called as Likert scale and contains 1 to 5 scales for this research. The scales are described the following:

1. Strongly agreed
2. Agreed
3. Slightly agreed
4. Slightly disagreed
5. Disagreed

Following table 5-2 illustrates the scenario number and the point value which are obtained by the user who involved in this research.

Table 5-2 Appendix A - User Feedbacks

User	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
1	1	2	1	4	2
2	1	2	1	4	2
3	2	2	1	5	2
4	2	2	1	4	1
5	3	2	1	5	2
6	1	3	1	4	2
7	2	2	2	4	2
8	2	1	1	4	2
9	3	2	1	4	2
10	2	2	1	4	2
Average Value	1.9	2	1.1	4.1	1.9

Following chart (figure 5-10) illustrates the average point values which captured from each scenario. Each scenario is explained after the figure 5-10.

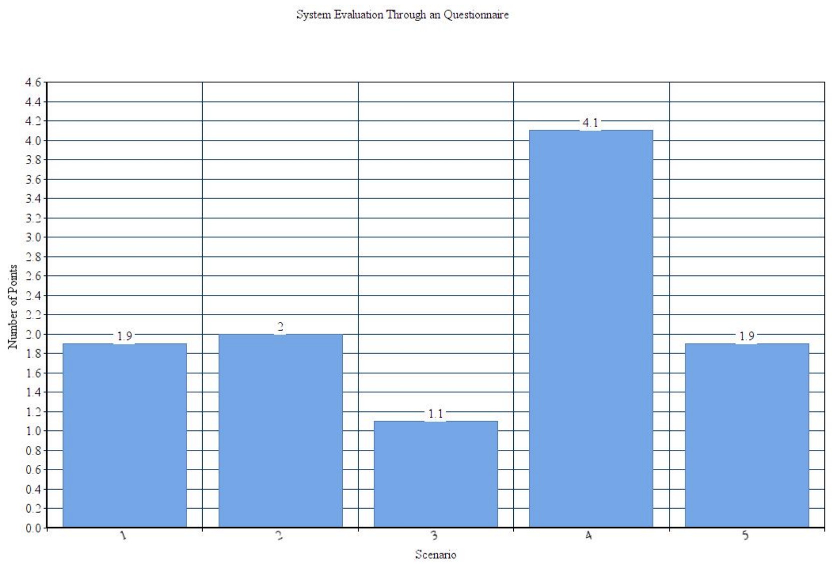


Figure 5-10 Functionality Evaluation Based on Questionnaire

- Scenario 1:
    - Ease of ontology creation through the system but without knowing any ontology concepts.
- Scenario 2:
    - Ontology creation is consistent through the system.

- Scenario 3:
    - Manage the ontology through system without protégé tool. (i.e. Querying the ontology.)
- Scenario 4:
    - faced some difficulties when using the system. (i.e. it asks about whether the system can be considered as a traditional system which get the results from database)

- Scenario 5:
    - Overall satisfaction on system functionalities.

According to the above chart, many users were able to manage the ontology system without facing any difficulties. The users attained their accomplishment to move the intended usage to the ontology. Mostly, the users provided the point two for scenario 2 because they face some difficulties to use with consistency of the ontology system. Furthermore, it requires reasoning and inferencing techniques to validate and check whether the ontology has a proper consistent.

5.3.2 User Interface Evaluation.

The solution is completely web based and may require strong web interfaces to convey the ideas to the end user. For this kind of evolution, evaluators might have the knowledge about ontology concepts, but it is not much important to evaluate the user interfaces of the system. Later, ten users were selected and allowed them to go through the system. They included some additional architectural decisions to the system, queried on that information and finally provided some feedbacks to given questionnaire (Appendix B). This questionnaire is constructed based on some scenarios like previous questionnaire. Five questions of new questionnaire were selected to evaluate the user interfaces and provided scales which are same as previous questionnaire, and which is used for functionality evaluation. The relevant feedbacks information is listed in below table 5-3.

Table 5-3 Appendix C - User Feedbacks

User	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
1	1	2	2	4	1
2	2	1	2	4	1
3	1	2	1	4	2
4	1	1	2	5	1
5	2	1	2	4	1
6	1	2	2	4	2
7	1	2	2	4	1
8	1	2	2	5	1
9	1	2	2	4	1
10	1	2	2	5	1
Average	1.2	1.7	1.9	4.3	1.2

Then the feedback data is illustrated with their scenarios in following figure 5-11.

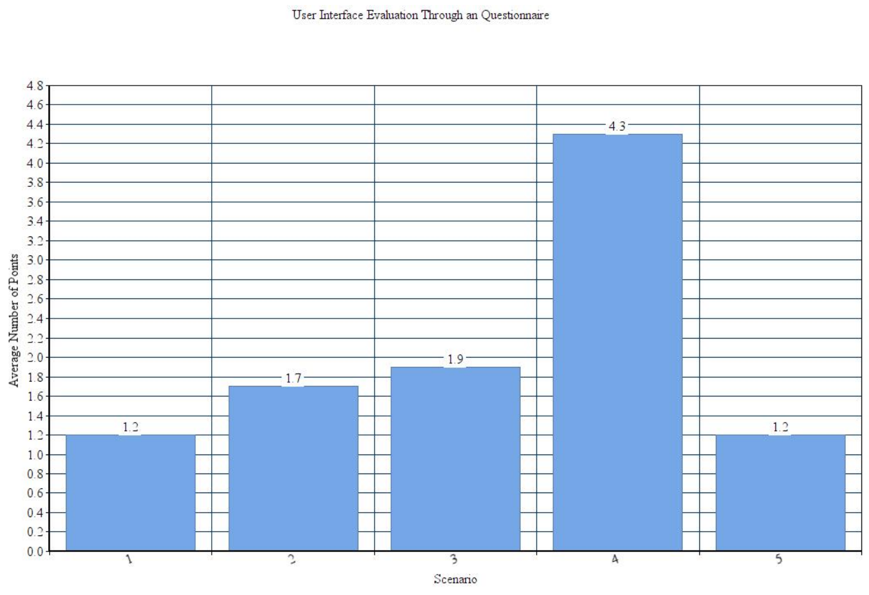


Figure 5-11 User Interface Evaluation Results

- Scenario 1:
    - User can manage the system without any instructions.
- Scenario 2:
    - User can learn system functionalities easily and quickly.
- Scenario 3:
    - Web interfaces are made of well-organized elements.
- Scenario 4:
    - Web interfaces may require additional enhancements or improvements.
- Scenario 5:
    - User can recover from mistakes easily.

Evaluation of Feedback on user interfaces indicates that the system has strong interfaces, and users are satisfied with the system. According to the above illustrated chart, scenario one and five have the highest score which conveys that the user can handle the system without any instructions and can easily recover their mistakes.

Above evaluated methodologies are covered with two main evaluation metrics which is described in the beginning of this chapter and those two metrics are:

- Structural metrics
    - Authors evaluated through the protégé tool.
- Functional metrics
    - System performance evaluation and functionality evaluation are wrapped with this metric.

Throughout the evaluation process, authors have successfully achieved the project objectives with some limitation which will be described in chapter six.

CHAPTER 6

CONCLUSTION

Typically, the software projects fail due to wrong architectural design decisions. Inappropriate architectural design decisions may lead to construct inappropriate architectural design which is the footprint or the initial step of any kind projects which may be smaller or larger. So, knowledge bases are emerged as a key solution for such kind of problems. Nowadays, researchers are typically finding the knowledge bases through ontological aspects because ontologies are the future trend of knowledge which is made of semantics. Semantics are the relationships among the data elements and helps to find the strong relationships among data.

6.1 Research Contribution

The main research objective is to construct an ontology-based knowledge management system for architectural design decisions. Authors have identified the important aspects on architectural design decisions and developed a knowledge base with step by step. Initially authors identified the domain and defined the ontology classes and the taxonomy with data properties and object properties. With the help of Apache Jena authors implemented a system with containing ontology creation (OWL file) and ontology querying programmatically.

Architects or any senior software engineers who has strong experience on architectural designing can input the data into the system through web interfaces of the system. Furthermore, any user who needs the knowledge on architectural designing can make query/queries into the system by providing inputs to the search queries through a web interface. After that, the previously inputted data or the past knowledge which is stored on architectural design decisions, will be displayed when it is needed.

In chapter five, authors performed evaluations for the implemented solution based on structural metrics and functional metrics. Those evaluations have proved that the implemented solution possesses the quality and the correctness which is considered as the important aspects of ontology evaluations. So, the industry or the organizations can get the real benefit of architectural designing from the implemented solution with some enhancements of the system such as distributed knowledge base.

6.2 Research Limitation

Even though many tools are available for ontology creation and analysis, only protégé tool is the popular one and has vast functioning domain for ontologies written in RDF, RDFS, OWL and OWL2. So, ontology specialists or expertise people have to learn those kinds of tools thoroughly. In this research, authors have attempted to find a solution without any tool-based solutions but implemented a web-based ontology for users who may have or may not have knowledge of tools like protégé. Although Apache Jena is the library, authors have used it to implement the ontology, but it supports only Java programming language currently. But Apache Jena still does not support for OWL2 which the latest ontology language is.

Unable to edit and save the existing ontology file is another limitation of Apache Jena library. But it can be resolved if the user opens the owl file by using the protégé tool. Then the user has to completely learn the protégé tool. Rather than Apache Jena, some ontology developers implemented some other libraries such as OWLAPI [42] to create and manipulate the ontology files, but the above mentioned limitations are still exist on those libraries.

6.2 Future Work

Implemented solution is a web-based system which has three tier architecture and was implemented as a proof of concept for the research objective. Existing solution does not directly create the ontology when the end-user clicks the 'create ontology' button since the fed data are stored in the database and then those data are mapped into an ontology. So, it requires a mechanism to edit the existing ontology file programmatically.

Many Organizations maintain the architectural design documentations and other important documentation. So, feeding those data into another system may be an extra burden to the important people in those organizations. Therefore, the implemented solution can be enhanced to grab the architectural knowledge from those documentations by using some annotations. Finally, to establish the implemented solution from this research to any organization, there will not be extra work and the system would help organizations to keep their projects success.

REFERENCES

[1] B. Len, P. Clements, and R. Kazman, *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional; 2 edition (2003-04-19) (1656), 2003.

[2] P. Kruchten, "An ontology of architectural design decisions in software intensive systems," in *2nd Groningen workshop on software variability*, 2004, pp. 54–61.

[3] V. Jain and M. Singh, "Ontology development and query retrieval using protégé tool," *Int. J. Intell. Syst. Appl.*, vol. 9, pp. 67–75, 2013.

[4] M. A. Babar, I. Gorton, and B. Kitchenham, "A framework for supporting architecture knowledge and rationale management," in *Rationale Management in Software Engineering*, Springer, 2006, pp. 237–254.

[5] R. Abdullah, Z. M. Shah, and A. M. Talib, "A Framework of Tools for Managing Software Architecture Knowledge," *Comput. Inf. Sci.*, vol. 4, no. 2, p. 2, 2011.

[6] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, 2005, pp. 109–120.

[7] N. Choobdaran, S. Mehran Sharfi, and M. R. Khayyambashi, "An Ontology-Based Approach For Software Architectural Knowledge Management," *J. Math. Comput. Sci.*, vol. 11, pp. 93–104, 2014.

[8] K. A. De Graaf, A. Tang, P. Liang, and H. Van Vliet, "Ontology-based software architecture documentation," in *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, 2012, pp. 121–130.

[9] M. A. Babar and I. Gorton, "A tool for managing software architecture knowledge," in *Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent, 2007. SHARK/ADI'07: ICSE Workshops 2007. Second Workshop on*, 2007, p. 11.

[10] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *IEEE Softw.*, vol. 22, no. 2, pp. 19–27, 2005.

[11] U. Van Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions," *J. Syst. Softw.*, vol. 85, no. 4, pp. 795–820, 2012.

[12] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Sci. Am.*, vol. 284, no. 5, pp. 34–43, 2001.

[13] I. Horrocks, "Ontologies and the semantic web," *Commun. ACM*, vol. 51, no. 12, pp. 58–67, 2008.

[14] M. P. S. Bhatia, A. Kumar, and R. Beniwal, "Ontologies for software engineering: Past, present and future," *Indian J. Sci. Technol.*, vol. 9, no. 9, 2016.

[15] T. S. Dillon, E. Chang, and P. Wongthongtham, "Ontology-based software engineering-software engineering 2.0," in *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, 2008, pp. 13–23.

[16] H.-J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *Proc. of Workshop on Sematic Web Enabled Software Engineering"(SWESE) on the ISWC*, 2006, pp. 5–9.

[17] S. Vasanthapriyan, J. Tian, D. Zhao, S. Xiong, and J. Xiang, "An ontology-based knowledge management system for software testing," in *The Twenty-Ninth International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2017, pp. 522–525.

[18] M. P. S. Bhatia, A. Kumar, and R. Beniwal, "SWOT Analysis of Ontology Driven Software Engineering," *Indian J. Sci. Technol.*, vol. 9, no. 38, 2016.

[19] O. Corcho, M. Fernandez-Lopez, and A. Gomez-Perez, "Ontological engineering: what are ontologies and how can we build them?," 2007.

[20] G. Brusa, M. L. Caliusco, and O. Chiotti, “A process for building a domain ontology: an experience in developing a government budgetary ontology,” in *Proceedings of the second Australasian workshop on Advances in ontologies-Volume 72*, 2006, pp. 7–15.

[21] A. LeClair and R. Khedri, “Conto: a protégé plugin for configuring ontologies,” *Procedia Comput. Sci.*, vol. 83, pp. 179–186, 2016.

[22] “Apache Jena,” 2018. [Online]. Available: https://jena.apache.org/. [Accessed: 21-Mar-2019].

[23] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, “Jena: implementing the semantic web recommendations,” in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, 2004, pp. 74–83.

[24] M. Poveda-Villalón, M. Suárez-Figueroa, and A. Gómez-Pérez, “Validating ontologies with oops!,” *Knowl. Eng. Knowl. Manag.*, pp. 267–281, 2012.

[25] H. Sack, “Ontology as Central Concept in Philosophy.” [Online]. Available: https://www.youtube.com/watch?v=mXdswAsFxO0. [Accessed: 21-Mar-2019].

[26] “Resource Description Framework.” [Online]. Available: https://en.wikipedia.org/wiki/Resource_Description_Framework.

[27] “RDFS,” 2010. [Online]. Available: https://www.w3.org/2001/sw/wiki/RDFS. [Accessed: 22-Mar-2019].

[28] “OWL,” 2012. [Online]. Available: https://www.w3.org/OWL/. [Accessed: 21-Mar-2019].

[29] “Ontology-based modelling and querying.” [Online]. Available: http://www-inf.it-sudparis.eu/~gaaloulw/KM/Labs/Lab2/Ontology-based-modeling.htm.

[30] M.-A. Storey, M. Musen, J. Silva, C. Best, R. Fergerson, and N. Ernst, "Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé," *Proc. 7th Int. Conf. Intell. user interfaces (IUI '02)*, p. 239, 2002.

[31] "This is a static HTML export of the Neon-Toolkit Wiki.," 2014. [Online]. Available: http://neon-toolkit.org/wiki/Main_Page.html. [Accessed: 21-Mar-2019].

[32] ""Swoop," 2007. [Online]. Available: http://semanticweb.org/wiki/Swoop.html. [Accessed: 21-Mar-2019].

[33] J. Bārzdiņš, G. Bārzdiņš, K. Čerāns, R. Liepiņš, and A. Spro, "OWLGrED : a UML Style Graphical Editor for OWL Interoperation with Protégé," *Computer (Long. Beach. Calif).*

[34] F. Losavio, L. Chirinos, N. Lévy, and A. Ramdane-Cherif, "Quality characteristics for software architecture," *J. Object Technol.*, vol. 2, no. 2, pp. 133–150, 2003.

[35] N. F. Noy, D. L. McGuinness, and others, "Ontology development 101: A guide to creating your first ontology." Stanford knowledge systems laboratory technical report KSL-01-05 and~…, 2001.

[36] "Spring Boot." [Online]. Available: https://spring.io/projects/spring-boot. [Accessed: 20-Mar-2019].

[37] "Apache Tomcat." [Online]. Available: http://tomcat.apache.org/. [Accessed: 15-Mar-2019].

[38] "Apache Maven Project." [Online]. Available: https://maven.apache.org/. [Accessed: 15-Mar-2019].

[39] "AngularJS Tutorial." [Online]. Available: https://www.w3schools.com/angular/. [Accessed: 02-Mar-2019].

[40] "MySQL." [Online]. Available: https://www.mysql.com/. [Accessed: 02-Mar-2019].

[41] and D. S. Hlomani Hlomani ∗, “Full-Text,” vol. 1, pp. 1–5, 2014.

[42] “OWLAPI.” [Online]. Available: https://github.com/owlcs/owlapi. [Accessed: 02-Mar-2019].

Appendix A How Ontology Concepts Affect to the User

Do you know what an ontology or knowledge base is?	
1. Yes	
2. No	
Do you feel or understand the ontology concept when you use the system?	
1. Yes	
2. No	
Have you experience with protégé?	
1. Yes	
2. No	
Have you ever tried out to create an ontology through protégé?	
1. Yes	
2. No	
If you have experience with protégé, do you think ontology creation using the implemented system is very easy?	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	
5. Disagreed	
Do you think that the ontology creation through the system is consistent?	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	

5. Disagreed	
Without knowing protégé tool, have you abled to manage the ontology through the system.	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	
5. Disagreed	
Without knowing ontology concepts and ontology tools like protégé, do you feel any difficulties with the system.	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	
5. Disagreed	
Are you completely satisfy with the system?	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	
5. Disagreed	

Appendix B User Interface Evaluation

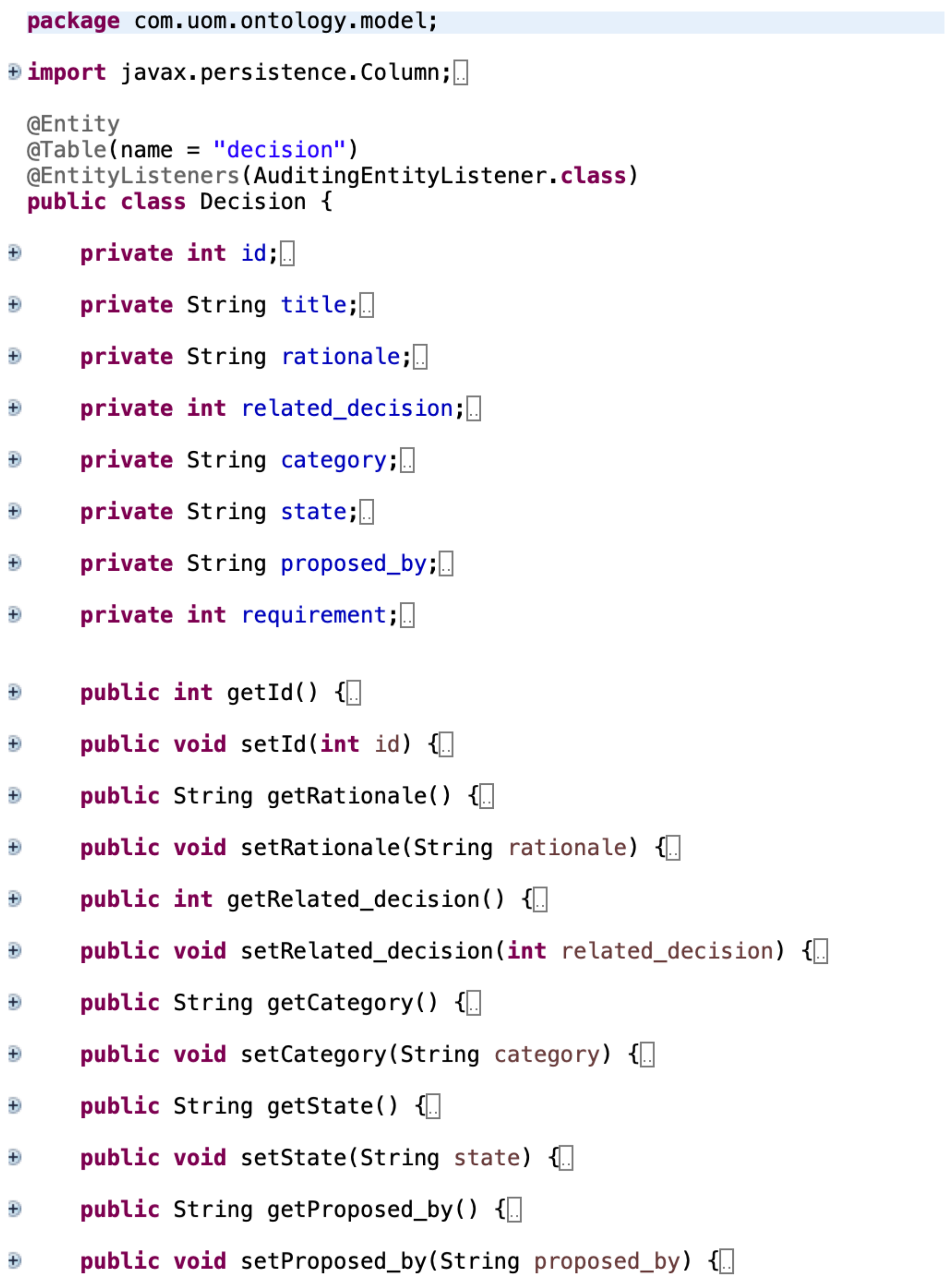
I can use the system without any user instructions	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	
5. Disagreed	
I can learn the system (functionalities) very quick and easily	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	
5. Disagreed	
Web interfaces consist well organized elements (text inputs and labels)	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	
5. Disagreed	
It may need more enhancements/improvements	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	
5. Disagreed	

I can recover from mistakes easily	
1. Strongly agreed	
2. Agreed	
3. Slightly agreed	
4. Slightly disagreed	
5. Disagreed	

Appendix C Sample Java Classes for Proof of Implementation

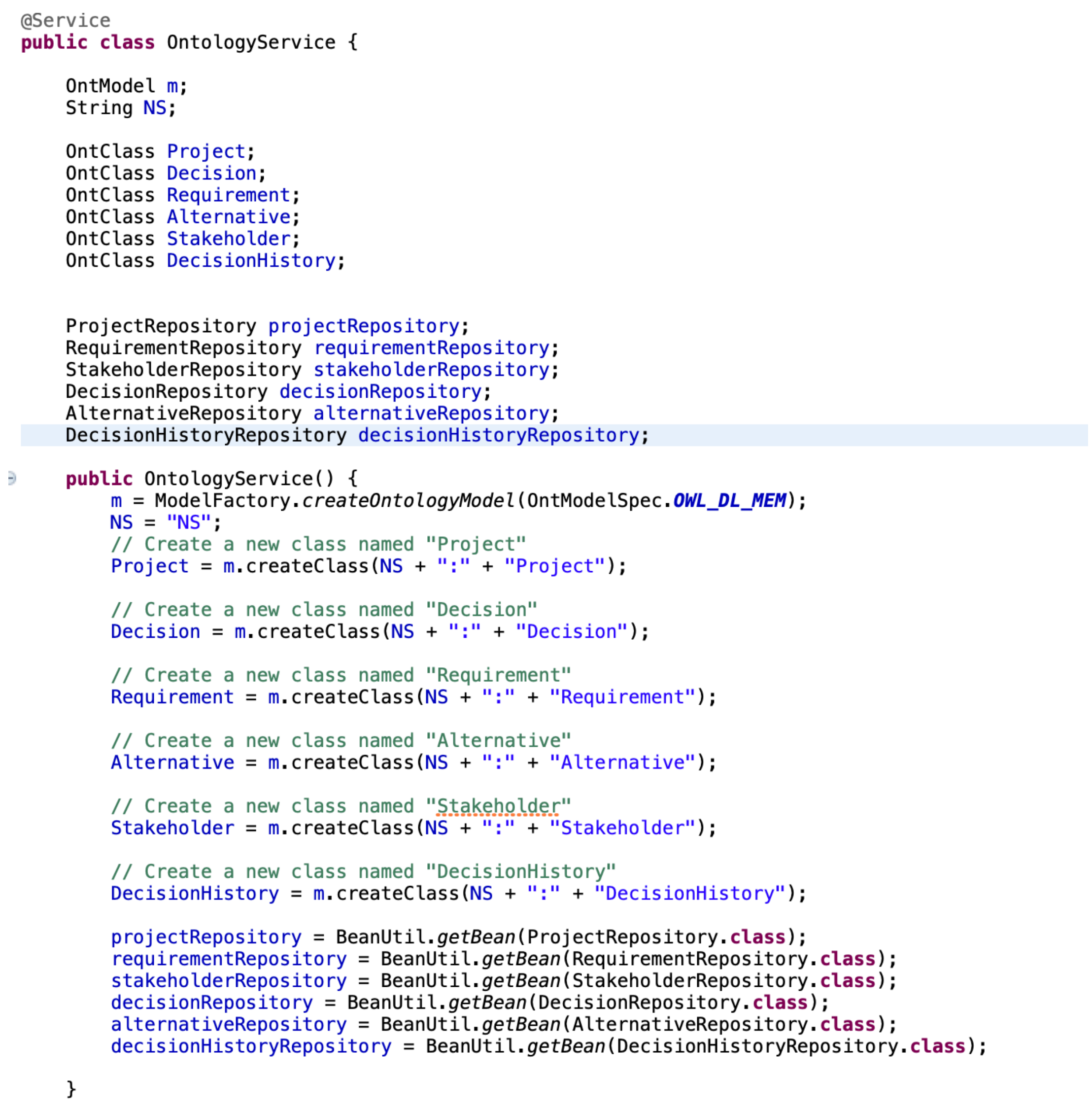
Some important java classes are illustrated below for the proof of implementation.

- Decision.java



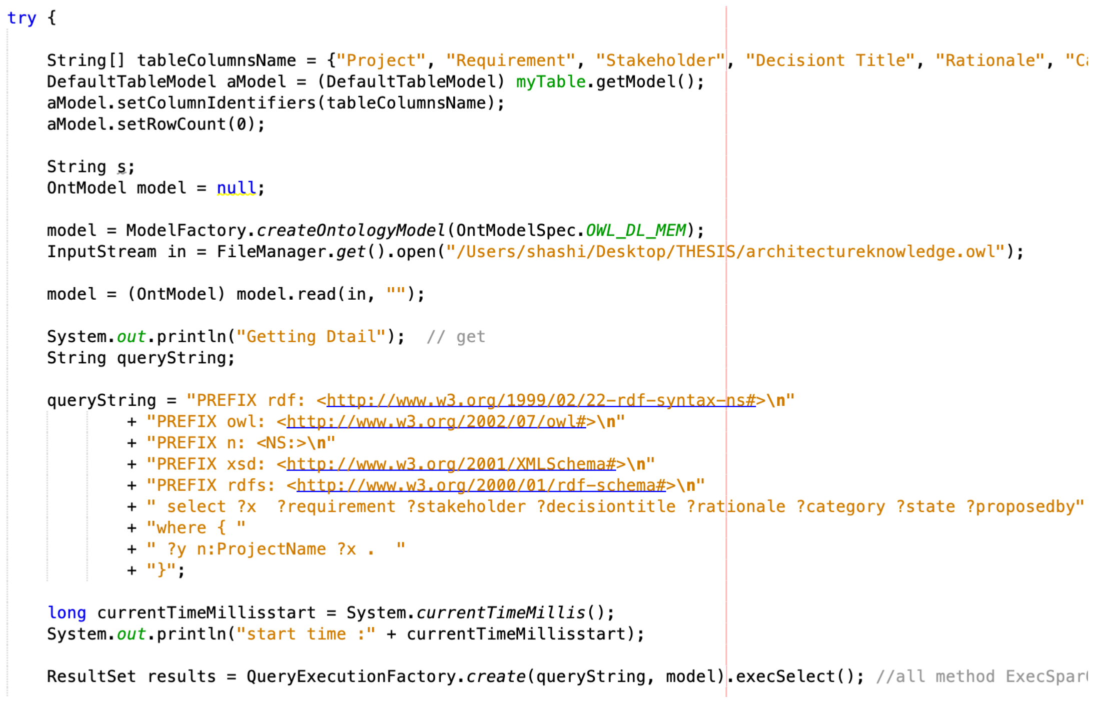
```
package com.uom.ontology.model;

import javax.persistence.Column;

@Entity
@Table(name = "decision")
@EntityListeners(AuditingEntityListener.class)
public class Decision {

    private int id;

    private String title;

    private String rationale;

    private int related_decision;

    private String category;

    private String state;

    private String proposed_by;

    private int requirement;


    public int getId() {

    public void setId(int id) {

    public String getRationale() {

    public void setRationale(String rationale) {

    public int getRelated_decision() {

    public void setRelated_decision(int related_decision) {

    public String getCategory() {

    public void setCategory(String category) {

    public String getState() {

    public void setState(String state) {

    public String getProposed_by() {

    public void setProposed_by(String proposed_by) {
```

- OntologyService.java



```java
@Service
public class OntologyService {

    OntModel m;
    String NS;

    OntClass Project;
    OntClass Decision;
    OntClass Requirement;
    OntClass Alternative;
    OntClass Stakeholder;
    OntClass DecisionHistory;


    ProjectRepository projectRepository;
    RequirementRepository requirementRepository;
    StakeholderRepository stakeholderRepository;
    DecisionRepository decisionRepository;
    AlternativeRepository alternativeRepository;
    DecisionHistoryRepository decisionHistoryRepository;

    public OntologyService() {
        m = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM);
        NS = "NS";
        // Create a new class named "Project"
        Project = m.createClass(NS + ":" + "Project");

        // Create a new class named "Decision"
        Decision = m.createClass(NS + ":" + "Decision");

        // Create a new class named "Requirement"
        Requirement = m.createClass(NS + ":" + "Requirement");

        // Create a new class named "Alternative"
        Alternative = m.createClass(NS + ":" + "Alternative");

        // Create a new class named "Stakeholder"
        Stakeholder = m.createClass(NS + ":" + "Stakeholder");

        // Create a new class named "DecisionHistory"
        DecisionHistory = m.createClass(NS + ":" + "DecisionHistory");

        projectRepository = BeanUtil.getBean(ProjectRepository.class);
        requirementRepository = BeanUtil.getBean(RequirementRepository.class);
        stakeholderRepository = BeanUtil.getBean(StakeholderRepository.class);
        decisionRepository = BeanUtil.getBean(DecisionRepository.class);
        alternativeRepository = BeanUtil.getBean(AlternativeRepository.class);
        decisionHistoryRepository = BeanUtil.getBean(DecisionHistoryRepository.class);

    }
```

- QuerySearch.java



```
try {

    String[] tableColumnsName = {"Project", "Requirement", "Stakeholder", "Decisiont Title", "Rationale", "C
    DefaultTableModel aModel = (DefaultTableModel) myTable.getModel();
    aModel.setColumnIdentifiers(tableColumnsName);
    aModel.setRowCount(0);

    String s;
    OntModel model = null;

    model = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM);
    InputStream in = FileManager.get().open("/Users/shashi/Desktop/THESIS/architectureknowledge.owl");

    model = (OntModel) model.read(in, "");

    System.out.println("Getting Dtail");  // get
    String queryString;

    queryString = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n"
            + "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n"
            + "PREFIX n: <NS:>\n"
            + "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n"
            + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n"
            + " select ?x  ?requirement ?stakeholder ?decisiontitle ?rationale ?category ?state ?proposedby"
            + "where { "
            + " ?y n:ProjectName ?x .  "
            + "}";

    long currentTimeMillisstart = System.currentTimeMillis();
    System.out.println("start time :" + currentTimeMillisstart);

    ResultSet results = QueryExecutionFactory.create(queryString, model).execSelect(); //all method ExecSpar
```