ONTOLOGY BASED SOFTWARE DESIGN DOUMENTATION FOR DESIGN REASONING

K.A.K.D.D.B. Jayasuriya

(179325L)

M.Sc. in Computer Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

May 2019

ONTOLOGY BASED SOFTWARE DESIGN DOUMENTATION FOR DESIGN REASONING

K.A.K.D.D.B. Jayasuriya

(179325L)

This thesis 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 PG Diploma Project Report does not incorporate without acknowledgement 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.

....

Dhanushka Bhavanthi Jayasuriya 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 CS5999 PG Diploma Project.

....

Dr. Indika Perera Date

ABSTRACT

Designing a quality software product adhering to all the functional requirements and non-functional requirements is a difficult task in software architecture designing. This needs much practice and experience regarding the designing knowledge. Selecting the best designs to apply in the project includes design reasoning. The discussion on the selections are important, but it dies when the discussion ends. As reasoning is important in the decision-making process, documenting the reasoning that was applied throughout the process is important for maintenance purpose and to overcome architectural evolution at different stages of the project. There are tools and standards that have been proposed on how to carry out the reasoning process and documenting it by other researchers. The use of ontology for the software architecture processes has been a topic of interest among researches at present. Creating a tool to generate design reasoning based on an ontology approach and evaluating its usability has not been successfully conducted. Hence for this research, an ontology-based approach has been chosen as a method to conduct the software architecture reasoning documentation. As software designing is a vast area of design decisions the research was narrowed down to the RESTful web service domain. An ontology was created comprising the architectural elements and the design decisions applied in the domain. Based on the ontology design reasoning is generated for a given software project. The document text would be first extracted and then processed based on the ontology values. Three techniques were used in deriving the key words and architectural elements on the document. The techniques included were key word matching, deriving architectural elements based on Part of Speech tagging and using ontology reasoning to derive the architectural elements. For the Part of Speech tagging a training data set was used to derive the elements and for the ontology reasoning a reasoning tool was used. Using these techniques, the architectural elements were extracted, and the design reasoning was generated using the ontology. The captured data was then documented in a user-friendly manner. A prototype of this approach was developed and evaluated to prove its usability and accuracy. An overall precision of 0.58 was calculated with the use of the prototype application developed.

ACKNOWLEDGEMENTS

My profound gratitude goes to Dr. Indika Perera, my supervisor for the knowledge, supervision, advice and guidance provided with his expertise, throughout in making the thesis a success.

My appreciation goes to my family for the motivation and support provided throughout my life. I also would like to thank my colleagues in the MSc batch and at my work place for the help and support provided in managing my research work.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
OWL	Web Ontology Language
RDF	Resource Description Framework
IE	Information Extraction
REST	Representational State Transfer
API	Application Programming Interface
URI	Uniform Resource Identifier
HATEOAS	Hypermedia As The Engine Of Application State
CRUD	Create, Read, Update and Delete
NLP	Natural Language Processing
OBIE	Ontology Based Information Extraction

Chapter 1

INTRODUCTION

The software development methodology and standards have evolved rapidly during the last couple of years. Some methods proved to be successful and continued to be practiced but others haven’t lasted long. This is mainly since those methods haven’t provide expected results hence, they died very quickly. The level of abstraction depicted through the code for modeling and implementation has increased. Starting from assembly programming, next structured programming, then high level languages and finally object-oriented programming. The main concern of the abstraction has been to reduce the complexity that lies within the software that is developed.

Software design is a process in the software development life cycle which would lay the foundation of the software to be developed. It would construct the blueprint of the software based on the requirements that have been gathered in the previous development cycle. During this process the structuring, defining the functionalities that are covered through the objects and the methods of implementation are noted. The decisions taken during this phase should align with the user requirements that have been specified and should be designed in a way that could easily accommodate any changes that would be proposed in the future.

With the evolution, software designing was abstracted to a level of architectural software design. Software architecture defines the structural level design of the system. It provides a visualization of the system to be developed and would help in aligning the functionalities that are developed with the requirements. Through the designing process both the technical aspects and the non-technical aspects of the system would be considered. The decisions should be taken regarding the functional, social and behavioral requirements, within the context of the system being developed [1]. It focuses on optimizing the common quality attributes such as security, performance and manageability.

A consensus on the approach followed when constructing the software architecture have not been established yet. The architectural design of the project can change as the project moves on for the following reasons. There could be conflicting goals when analyzing the requirements and since development is progressed to the next phase the conflicts have to be resolved. An approach to resolve the conflicting goals is to make continuous changes to the architecture as the project progresses which would ease the development since more information would be available. But since the project needs to progress with the development the architectural knowledge at the early stages is required.

Another approach adapted for the design is heavily depending on the client's architectural perception on the project rather than deriving the design based on the architectural solutions of the project. The perception of the architectural design of the client and the design based on the architectural design has a significant gap. The clients would sometimes miss the clarity of the goals which causes wrong assumptions of the design. This would lead to inappropriate scoping which leads to the fundamental abstractions being missed and redundant abstractions being applied.

Third approach occurs when the requirement sources, the client has provided are not rich with the semantics needed when modeling the architecture and for composing the architectural abstraction. This would result in an architecture where the grouping of the components would be semantically poor.

The final approach is using solution domain analysis to derive the architectural abstracts needed and gain the component semantics needed for modelling. Managing this solution domain would be difficult and the depth of detail within the domain model would not be enough in deriving the architectural abstractions.

1.1 Importance of software architecture

The success factor of a project depends on how well the architecture of the project is constructed to meet the goals of the users and the IT infrastructure available for the project development. A well-defined software architecture would have positive influences for the development cycle of the project. These positive influences can be mentioned as reuse, understanding, evolution, management and analysis.

Reuse: This is mainly focused on the reuse of the components and frameworks. Usually the reuse of libraries is popular and are used in many projects. The usage of components and frameworks for projects in the same domain can be applied by referring to the architectural documentations of those projects.

Understanding: A high level abstraction of the system to be implemented can be represented through the architecture. A complex system is easier to understand through this representation and rationale behind the architectural decisions. The high level constrains of the system design is included in the architectural description.

Evolution: An understanding on which aspects would evolve in the system could be gathered through the software architecture. It also gives a better understanding of the costs that would have to be incurred due to this change. The separation of concerns on the functionality of the components and how each component is connected, are expressed in the architecture. This knowledge helps in adding or modifying these connections with the evolution [2].

Analysis: New opportunities for analysis is provided through the architectural descriptions. This involves the high-level system consistency checking, conformance on the quality attributes and the analysis on the architectural style applied.

Management: Creating a viable software architecture is a key milestone in the software development process. Aligning with the initial software requirements and achieving its operational capabilities required, the anticipated dimensions of grow, the rationale on the design decisions taken would provide direction on the growth of the system.

Without satisfying these conditions the system would be unable to accommodate the changes [3].

1.2 Design Reasoning

Researchers have identified that there are two cognitive systems that run as the basis for the design reasoning. One approach being applying logical standards and proposing the design and the second approach being identifying the input modules and contextual background of the system in making the design [4]. These approaches together formulate the explanation for the theory, "rational thinking failure". The roots in design failure is due to relying heavily on the intuition and prior understandings rather than following the rationale of the design [4, 5]. This heavily impacts the quality of the decision and is based on the expertise and experience of the designer rather than a logical reasoning for the design. The combination of both rationale and the natural intuition is the approach that is followed at present in the industry [6].

Architects sometimes without knowingly, apply cognitive reasoning when designing the architecture. This application of reasoning leads in to producing a better outcome, but also the designers experience and personal preference would influence the design. This can have either a positive outcome or a negative outcome. For a design to be successful the people involved in designing the architecture should be experienced and having good analytical skills. Then only, the design not contain any over engineering, requirement misses, underestimated effort and design quality flaws. To ensure that the designing has a considerable level of consistency and quality to be guaranteed, the reasoning abilities of the designers must improve.

A key part of the reasoning process is the design rationale. For this the understanding on what design rationale constitutes of should be clarified first. Rationale is a process followed in reasoning which involves deliberation and negotiation of the methods that are proposed. The designing process is known as a "wicked problem", where there is no strict right or wrong answer defining the potential solutions [4]. The process would have a fruitful outcome if the people involved in it knows how to weigh the alternatives and propose the design.

When designing the architecture, the goals and the requirements of the system are considered as the inputs which are the basis of the deign decisions. The requirements can also be stated as the design concerns, which provides direction to the design decision. Using these inputs, the design decisions are made, and the outcome is the designing rules that are presented. The design decisions depict why the design outcomes were produced. The relationship between the inputs, decisions and the output, the justification in to why the decisions were made is included in the design reasoning. Changes that would occur to the product and process, due to the requirements changes can be tracked with its impact when design reasoning is present.

1.3 Problem Statement

The main research problem that is addressed through this research is to develop a tool that would document the design reasoning that would be applied during the software architecture design phase. The design reasoning would be an output based on the design rationale that would be applied when making the decisions. In the industry, designing the ideal software architecture for the project would determine how successful the project would be in satisfying the goals and the requirements. Therefore, it is important in documenting the design decisions that would be taken during this process. Although the importance is known mostly in the industry level, the effort needed to document the reasoning is high and hence the majority avoids the documentation. Not having a systematic way in documenting the applied design reasoning is also one reason for not documenting it. Through this research a tool that would have a systematic way in documenting the design reasoning is proposed to address this issue.

1.4 Motivation

As mentioned in section 1.3 there is no systematic way practiced in the industry at present to document the reasoning logics behind the proposed software architectural design. The lack of reasoning documentation causes issues when the reasoning behind the decisions needs to be recalled for various circumstances. This could be simply needed for design verification with the clients or for getting more clarity on the design.

Deciding if this architecture could be applied for a software problem in the same domain is not feasible. The reasoning documentation is also important for the maintenance of the domain but, if the reasoning in applying certain design decisions are unclear the reuse system and for evolution of the software with the new requirement changes that would be proposed in the future.

Considering all these factors the documentation of the design reasoning is understood as an important factor to develop and maintain a software successfully.

1.5 Objectives

- Gather knowledge on the design decisions taken during the software designing and design concerns that led to the decisions.
- Construct an ontology using the knowledge collected regarding the design decisions.
- Capture the design reasoning in the ontology to be presented in documentation.
- Use natural language processing techniques to extract words from the documents presented.
- Create a trained data set to derive architectural elements from the documents presented.
- Develop an ontology based tool that would generate a design documentation capturing the design reasoning that was used when taking design decision.

1.6 Scope

At the completion of the research an ontology would be created for the RESTful architecture domain that was selected for the research. The ontology would comprise of the architectural elements, the key terms and design decisions that would be applicable to the RESTful architecture domain.

A trained data set to extract the architectural elements would also be created to derive the architectural elements that are available in the software documents. This trained data set would differ from one domain to another hence this was created to align with the RESTful architecture.

An ontology reasoner would be selected to generate an inferred ontology using the property values included in the document. The wordings with the properties would be identified as individuals. These individuals would be mapped with the ontology elements and be derived as architectural elements.

Finally, using these mentioned techniques a tool that would extract the architectural elements for a given software document would be created generate the reasoning of a software document.

1.7 Outline

The first chapter discussed on the research problem that would be addressed through the research. It also provided the motivation and the objectives that needs to be achieved during the completion of the research. The second chapter presents the other research that has been done under the architecture rationale, software design reasoning and ontology-based documentation. These three areas were selected as current knowledge regarding the exiting tools needed to be identified.

The methodology, which is chapter three presents the approach that was followed during designing the tool. Several steps have been defined into creating the tool for design reasoning. The next chapter discusses on ontology creation steps followed and how the architecture of the prototype project should be structured.

The final two chapters provide the evaluation criteria to evaluate the prototype tool that was created. The chapter also contained the survey results of the questionnaire provided to evaluate the current design documentation status and its importance. The conclusion chapter discusses research contribution, limitations of the research and the future work that can done under this research.

Chapter 2

LITREATURE REVIEW

Research on the software architecture design reasoning was analyzed to understand the research backgrounds and their solutions proposed under this domain. Design rationale which captures the knowledge and the reasoning behind the decisions taken was a relatively similar topic and the approaches followed in those researches gave direction to the approach that is proposed through this research. Since the research is done based on ontologies, techniques used for information extractions from ontologies and the research areas conducted for software architecture design reasoning were also analyzed.

2.1 Research carried under software architectural rationale

Design rationale provides the reasoning and the knowledge used in the resulting design. This is inclusive of how the design satisfies the functional and non-functional requirements as well as why the design choices were made over the alternatives. It also provides the understanding on how the system would behave under different environments. Argument structure is a commonly advocated method in capturing the design rationale [7]. Issue Based Information Systems (IBIS), Design Rationale Language (DRL) [8], Questions, Options and Criteria (QOC) [9] are formal methods that can be used for design rationale. QOC is preferred by dome designers as it makes it a necessity to define the design criteria explicitly [10].

The major objective of the rationale is to understand the designer's perception on selecting a rationale and the important elements of it. Rationale is important when performing maintenance tasks and modifications which is more effective with the guidance of the rationale.

A simple approach to maintain design rationale is by the designer writing a short explanation on why the lower level aggregation was needed and the purpose of the component at each aggregation level of the system. This would help in maintaining a strong association with the actual code [11]. This can be practiced by using a code entry tool to prompt the aggregation levels which would also result in providing the documentation. This would maintain the strong association as well as it would update the documentation when a code level change has been done. Another approach is the Architectural Rationalization Method (ARM), which uses quantitative and qualitative rationale for the reasoning.

A generic list of design rationales can be mentioned as:

- Design constraints which are the limitations in the design [12].
- Design assumptions which are the factors that would affect the design [13].
- The design would benefit the technical and functional requirements. [14].
- Weakness would be the unachievable technical or functional aspects of the design [14].
- Cost of the design would explain the implicit and explicit costs that are incurred to the system [14].
- Complexity of the design is a relative measure of the design complexity, with reference to the implementation and maintenance [14].

Tools that are used in the industry level to document the rationale are UML tools, organizational templates and standards for documenting using Word, PowerPoint, excel and Visio, IBM GS methodology, requirement traceability matrix and architecture tool CORE. Architectural decisions made through the rationale have a huge impact on the quality of the resulting system. Some weakness in the design rationale are mentioned as substantial amount of training needed for the effective use of it. The weak association that would arise between the design rationale and the actual code is another weakness.

2.2 Software architecture design reasoning

The design reasoning process must consider the architectural design requirements, the design issues and the tradeoffs among the available design options [16]. A rationale-based model constructed through reasoning guides in designing the software architecture and provide support for the maintenance activities [16].

Software designers tend to make decisions based on their previous experiences and intuition rather than on the logical reasoning process [4]. This is a common behavior that has been observed among designers which influences the reasoning ability [16]. The designers want to give the design thought before they act which would avoid the mistakes that would arise through spontaneity and ignorance [17]. There are two possible failure types that can occur from the design. One is that the design doesn't accomplish the desired outcome and the second type is after the execution there are side effects that were unintended and unforeseen. Design reasoning provides a framework which can guide, inexperienced designers to support and creating a mental image of the current design. But for the experienced designers this assistance is not much needed as they are confident on how to proceed [18].

Architects rely on experience and their intuition when designing architectures. Experienced architects have better architecture designs than the inexperience designers. So, to overcome this ad hoc designing practices and to improve the quality of the software a methodical approach needs to be followed. The proposed architectural design is based on the designer's past experiences and personal preferences [19].

When investigating the design reasoning first the reasoning model needs to be established. Figure 2.1 represents the reasoning process followed in Architectural Rationale and Elements Linkage (AREL). For this the input, decisions and the output need to be considered. The inputs can be defined as the requirements an-d the goals of the system which captures the conceptual information. The decisions are the reasoning and the justifications for taking the decision during the designing process. The output is the outcome of the designing process. For the modeling of design reasoning, the relationship between these three needs to be understood.

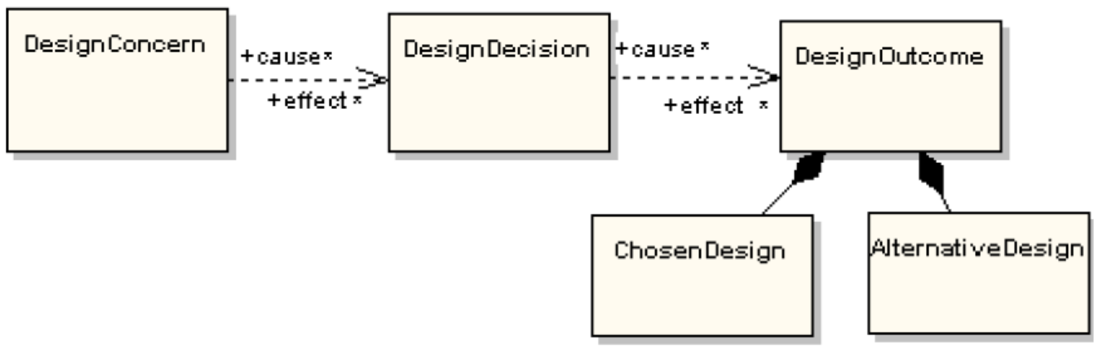


Figure 2.1: AREL – A Design Reasoning Model

Source: [19]

In the AREL, a five-step process is followed. This process is done repeatedly to decompose the architectural design and create design at each reasoning cycle. The five-steps are defined as specifying design concerns, associating design concerns, identifying design options, evaluating design options and backtracking to compromise design concerns.

The first step is specifying design concerns of the project. The architectural designs are reflected through the goals and purposes, functional requirements, non-functional requirements, business environment, information system environment, technology environment and design. The business goals are defined by the purpose and goals. The functionality of the system is defined by the functional requirements. The non-functional requirements define the quality constraints of the system such as the usability and performance. The business and organizational factors that defines the strategic and long-term business goals are the business environment. The environmental factors such as budget, expertise and schedule are the environmental factors. The organizational policies and technology define the technological factors that would influence the technology environment. If there are existing designs that have been selected prior in the system, they would influence the design that would be selected [15]. These should be available at the project start for the designing process to begin. These architectural decisions influence the architectural design of the project.

Next step is associating design concerns, where architects make decisions for a situation that arises based on more than one design concern. Here the relevant decisions must be selected by the architect based on the knowledge and domain of the system. Based on the conjunction design decisions, the architectural design is proposed. Design issues occur at times there are conflicting and competing influences of the design concerns. Design rationale assists in the design reasoning process and it would provide the justification in selecting a design. Design rationale is divided into two broader topics as qualitative and the quantitative design rationale [20].

Qualitative rationale factors can be listed as:

- Design issue, which is the problem that needs to be addressed.
- Design assumptions, which are made by the designer.
- Design constraints, are the design constraints which are either technical or contextual
- Strengths and weaknesses, are the strengths and weaknesses of the design options.
- Tradeoffs, are after weighing and prioritizing the design options selecting the appropriate option.
- Risks and no risks, are the uncertainties and the certainties of the design options.

The quantitative rationale factors are mentioned as:

- Cost, that are incurred during the development phase, platform support, maintenance and other legal liability costs.
- Benefit, to what extend the proposed design meets the requirements and the quality attributes.
- Implementation risk is the risk that the implementation might not be successful as planned due to the lack of experience or capabilities of the developers.
- Outcome certainty risk, where the design might not satisfy the requirements since they are not well defined or are unachievable.

Based on the quantitative decisions the architects would be able to weigh the alternatives and select the design that would be least risky.

The level of reasoning applied for the designing assists for a detailed design and development of the system. The reasoning can be controlled through risk mitigation strategy. If the risk is low the design is complete and ready for development. If the risk is high more detailed designing is needed for the current design.

Identifying design options is the next step where design reasoning is applied in identifying the appropriate design options. Enough knowledge on the problem domain and designing is required for this step. Experienced architects usually tend to be biased for their initial design and would modify this design considering the alternatives that are available.

In evaluating design options step, the best suited design which would satisfy the design concerns has to be selected. If there are no design available for the design concerns, then the constraints need to be relaxed. If there are multiple designs available depending on the pros and cons, a qualitative approach can be followed to select the best design. If exactly one design is proposed the architects could go ahead with that design.

The final step is backtracking decisions for revising design concerns which occurs when there are interrelated software design concerns which lead to chain of design decisions. When interrelated design decisions are made the initial solutions may not be viable and these decisions must be revised, and new decisions must be considered.

The usage of decision maps is also used in design structuring and sequencing. The maps depict design concerns and the design decisions in a sequence of decisions that would relate to the same topic. The visualization eases in providing guidance in taking design decisions [21]. During the reasoning process different structures can be followed by the participants of the reasoning process.

One approach is the semi-structured approach where the designers would start with the design discussion that would focus on a queue of design concerns and progress along as push and pop operations while exploring the solutions to structure the design. Another approach is the compositional structuring approach. Here first the major components of the system are identified, and some time is spent on investigating them. Starting from the basic components, the complex design structure is developed which helps in tackling the simple designs rather than moving to the complex design initially. Another reasoning approach is the high-low structuring approach. Here the major design concerns are addressed first and when the decisions for those are provided, low level design concern are addressed.

There are design reasoning techniques that can be followed such as inductive and deductive reasoning. Inductive reasoning generalizes the specific observations or facts to be based when creating a theory to be applied for other scenarios. Deductive reasoning is based on commonly known facts and situations to derive the logical conclusions.

2.3 Ontology based design documentation

Ontology is used as a system of concepts and for a vocabulary in modeling a domain or building information systems [22]. It provides the support in modelling the domain with the usage of labeled concepts, relationships, attributes and either specialized or generalized hierarchies. The aim of creating an ontology is to use it as a common form of vocabulary among the designers which would maintain the standards and a method of sharing knowledge among others [23]. Ontologies in the problem-solving context are divided in to two types as task ontology and domain ontology. Task ontology focuses on the system design process and the domain ontology is a concrete representation of a conceptualized domain [24].

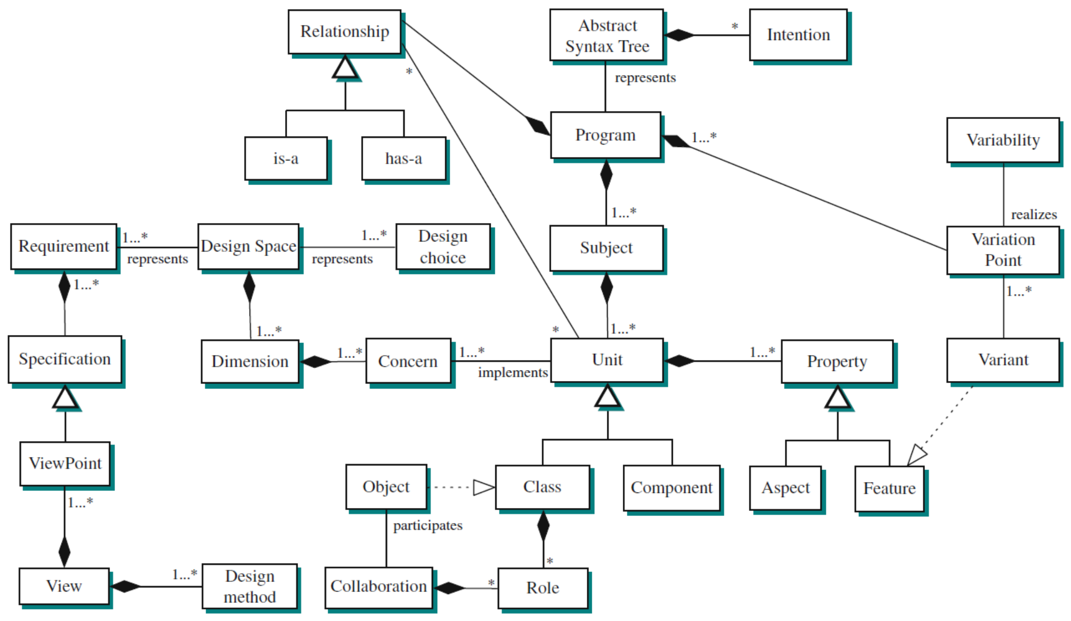


Figure 2.2: Ontology of Software design domain concept

Source: [23]

The design options and the design requirements comprise the design domain. The implementations of the software components and the classes are address the design concerns [23].

Design decisions can be modeled as first-class entities when constructing an ontology model. The design decisions identified were broadly classified in to three sections [25, 26] as:

- Existence decisions
- Property decisions
- Executive decisions

2.3.1 Existence Decisions

These decisions results in the artifacts or the elements that would be prevail in the systems design implementation. These decisions comprise of structural and behavioral decisions.

Under this the non-existence decisions are also modeled, and they are the subclasses of existential decision.

2.3.2 Property Decisions

These decisions would maintain the quality of the system by laying the rules, guidelines and the constrains the system should abide to. Properties are hard to be traced to an element as they can affect more than one element and can often raise cross cutting concerns. These would result in the architectural rationale.

2.3.3 Executive Decisions

Decisions that doesn’t relate directly to the design but are driven by the business environment and have effect on the development process, the people, the choice of tools for the development and the organization.

Table 1.1: Attributes of decision

Name	Type
Epitome	Text
Rational	Text or Pointer
Scope	Text

State	Enumeration
History	List of (time stamp, author, change)
Cost	Value
Risk	Exposure level

Source: [25]

Table 1.1 presents the different decision attributes that are in the ontologies. The types of relationships that would exist between decisions would be constraints, forbids, enables, subsumes, conflicts with, overrides, comprises, is bound to, is an alternative to, is related to, traces to, does not comply with.

The attributes and the relationships are be mapped in the ontology using OWL and RDF is used to describe the ontologies.

2.4 Knowledge extraction from ontologies

Gathering the knowledge existing in the web is a time-consuming task if it must be done manually. This is almost be impossible as well. For this task a promising approach is IE which extracts the required knowledge and reduce them to tabular structures. These extractions are presented as answers when queried for knowledge from the Knowledge Base. This requires manually annotating the texts and creating templates that stipulate the information to be extracted which consumes high time and effort [27]. IE systems mainly rely on pattern-based extraction rules and predefined templates. They also use machine learning techniques to identify certain keywords in texts. Vocabularies, composition styles and structures are used for web documents to approximately identify the similar contents.

The idea of IE is to automatically gather certain information from the natural language. It processes the natural language and retrieves the occurrences of objects or events and their relationships among them. OBIE, emerged as a subfield under IE [28]. Ontologies are needed for knowledge extraction as it understands the relevant information needed to be extracted. The extracted information is used to populate and enhance the existing ontologies. Hence, they become interdependent tasks where the ontology is needed for the IE and the extracted IE are integrated with the ontology. The extracted information is represented through the ontology.

Ontology is defined as a shared conceptualization based of a formal and explicit specialization [29]. The general idea of an ontology is specialized for a domain. Since IE is also concerned with extracting information for a domain, depicting the information extracted formally and explicitly through an ontology is convenient. First the ontology construction with its specified concepts and relations must be defined based on the domain information. This process also needs information extraction and is formally specified as the open information extraction.

After constructing the ontology, the ontology population task is performed which extracts the property values and instances based on the classes and the properties defined in the ontology. The populated ontology then represents a knowledge base for the domain specified. These ontologies are used as a key source of automatically processing domain specific information contained in the natural language.

The Text-To-Onto also known as onto learning purpose there are different algorithms. These algorithms have been proven successful by the researchers. Some of these algorithms are listed as clustering, pattern matching, classification, inductive logic programming, association rules and data correlation for relational schemas.

Lexical entry and concept extraction are a base line method that is used for acquiring lexical entries. The multiword terms extracted using n-grams is a statistic-based technique. The lexical entries are proposed on the weighted statistic frequencies obtained using the n-gram text preprocessing technique. The new lexical entries does not have an association concept at first. This task is assigned to the ontology engineer to assign the entry to an existing concept or introduce a new concept to the ontology and associate the entry with it [30].

The acquired lexical entries and concepts must go under a taxonomic concept classification. A generally applicable classification is a hierarchical based clustering which analyzes item similarities and proposes categories based on a hierarchy. Association rule learning algorithm can also be used to identify the relations between the concepts. The most relevant binary rules for modeling the relations in the ontology are proposed by this algorithm. On the ontology constructed the IE is performed on documents that are relevant to the domain. The knowledge that has been gathered would be enhanced and needed information can be extracted according to the tasks needed.

The Figure 2.3 represents an Ontology Based Information Extraction System [30]. The text input would first go through the preprocessor which converts text to a format identified by the information extraction module. Using one of the techniques discussed above the information is extracted with the guidance of the ontology. The ontology can be created using an editor such as protégé or might be a predefined ontology. The information extracted can be stored in an ontology structure or can be saved as a knowledge base.

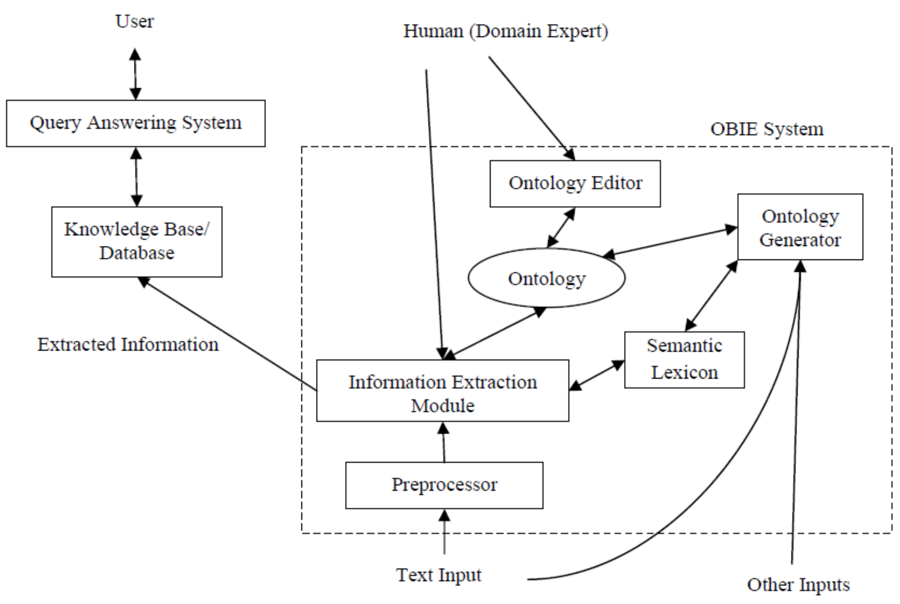


Figure 2.3: General Architecture of an OBIE System

Source: [30]

2.4 Summary

This chapter discussed on the research that have been done under design rationale, design reasoning and ontologies. Based on the design reasoning and rationale the IBIS was stated to not map the inter-related issues hence it was difficult into modeling the deliberations. QOC is mainly focused on the design options and provides the justification in making that selection. DRL explains the rationale by describing how the artefact satisfies the required functionality. ADDT, focuses on representation model and the deliberation of the design process. It doesn't provide a guidance on the reasoning process.

The ontology-based design documentation discussed mainly three sections under this chapter. Since an ontological approach was not directly done for design reasoning the mentioned sections were taken into consideration when creating the ontology for the research.

Chapter 3

METHODOLOGY

The methodology followed in the research for software design reasoning of the software architecture would consist of an ontology-based approach. This approach would first need an ontology defined for the chosen domain and the structure should be finalized. Using the created ontology, the Architectural elements and the Architectural rationale should be extracted. Based on the extracted rationale the reasoning for applying the architectural decisions can be specified for the software design.

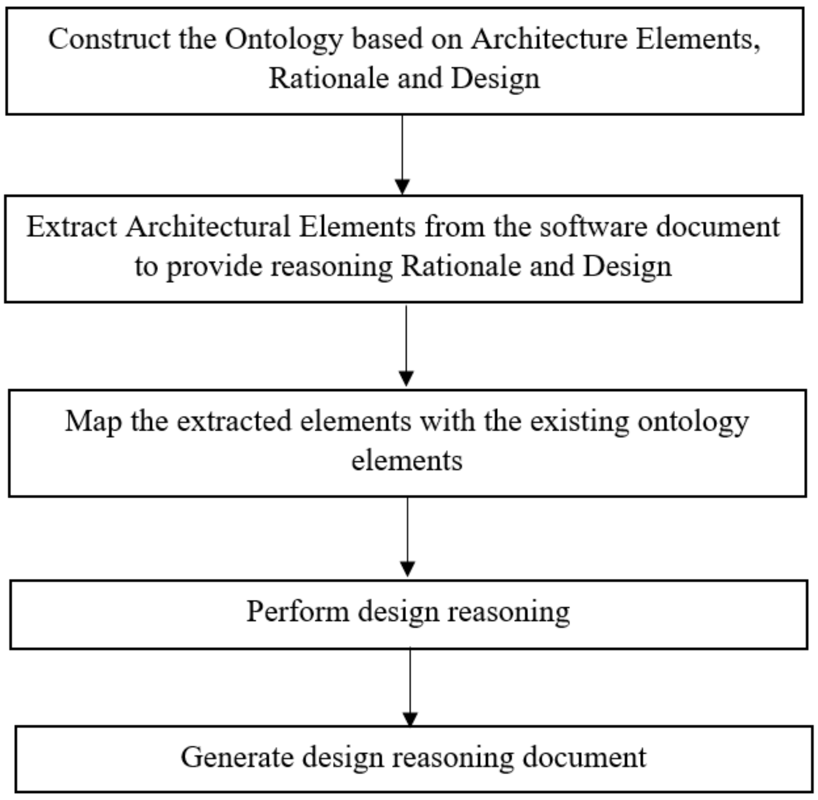

Construct the Ontology based on Architecture Elements, Rationale and Design
Extract Architectural Elements from the software document to provide reasoning Rationale and Design
Map the extracted elements with the existing ontology elements
Perform design reasoning
Generate design reasoning document

Figure 3.1: Flow chart of the methodology for the design reasoning

Represented in Figure 3.1 is the process that would be followed in generating the design reasoning document for a given software product of a domain. This approach was selected with the understanding gained through the literature review done and the development experience in the software industry. Following AREL tool first the knowledge base that would be used for the reasoning should be created. Based on the ontology created, to apply the reasoning the document provided for reasoning should be read using the tool. The text extraction process and using the text matching the key words and deriving the elements had to be done in the next steps. This followed with retrieving the reasoning information from the ontology which is needed for the documentation. The reasoning would have to be restricted to a domain since the architectural design would differ from domain to domain. As well as the knowledge that would be needed to store would be vast if all the domains had to be covered. Hence this research is conducted on the REST service-oriented architecture.

In the process mentioned above the following step are covered.

1. Ontology construction based on Architecture Elements, Rationale and Design.
2. Architectural Elements extraction from the software document to provide reasoning.
3. Map the extracted elements with the existing ontology elements.
4. Perform design reasoning.
5. Generate design reasoning document.

3.1 Ontology construction based on Architecture Elements, Rationale and Design

Ontology is a form of representing knowledge for a given domain. In the computer science domain, the term ontology has emerged as a separate engineering field as Ontology Engineering.

It is used as a formal model to unambiguously describe the aspects of a domain. The model would comprise of terminology that would be accepted as domain knowledge. Ontologies covers the aspects of interoperability and inter-agent communication. It has been interpreted in different ways and several dimensions such as authoritativeness, quality and degree of formality [31].

According to the specificity dimension specified by Oberle, generic, core and domain ontologies can be identified. Generic and core ontologies provide a higher level of generality while the domain specifies a specific area. The ontologies formal expressiveness distinguishes if it is a lightweight or a heavyweight ontology.

With the semantic web emergence, vision ontology has been a topic that has attracted much interest. Along with this trend other technologies have been developed for the representation of ontology, ontology sharing and machine representation. Ontologies have also been used in different forms. Humans use it as a way of communication among peers and in information technology it facilitates the integration and the content-based access through maintaining precision of the data. This leads to the importance of the ontology being clearly structured, well designed and understandable to the relevant individuals.

The use of ontology in the software engineering aspects have increased and frameworks have been implemented in categorizing the use of ontologies in this field. Designing an ontology for a domain is a complex procedure and the designers are provided with different design methodologies, editors and reasoning tools. To apply reasoning for software architecture the ontology had to be structured in an explicitly defining manner so that the reasoning can be applied.

To structure the ontology of software design decisions the computational complexity of OWL full was needed. This allowed the users to get maximum expressiveness of the ontology and provide syntactic freedom of RDF [32]. Since the research was narrowed to the REST service-oriented architecture, the domain related knowledge was extracted. For this the domain specific material on REST API service-related rules and models followed were collected into forming the ontology.

As a standard the Richardson Maturity Model was followed as the basis for REST API services which was developed by Leonard Richardson [33, 34]. The model defines the REST API in to three levels all together. Apart from these levels there is level 0 which defines that the service is not belonging to a REST API. The model provides a guideline into how to structure the REST service and it’s not vital that the developer must stick to this model, but it would be according to a standard and have better practice.

Level 0 has one URI and the operations that needs to be performed is be contained in the message request with the action needed to be performed. So, all the actions are directed to one URI containing the action in XML format in body.

Introducing the resource URI concept, the REST API would be formed starting at level one. The request would still contain the operation that needs to be performed as either a create, delete or requesting information, but would differ according to the resources.

Introducing different HTTP methods for the operations would advance the request into level two. They use standard HTTP methods like GET, POST, DELETE and PUT. The URI would include the resource name and the HTTP method defines the operation to be performed. The response would use the status codes indicating the successful or unsuccessful state of the response.

Level three in REST API implements HATEOAS, where the responses have links that would control the application state for the client. The client would not need to know the URIs available since they would be included in the response.

These design decisions can be divides in to existence, property and executive.

Existence decisions which defines the structural decisions should include decisions such as having a server that would prevail to serve the API requests. This should be an independent component and should be differentiated from the client. Resources should be defined for each service end point and each resource should support the CRUD operations under a resource aligning with the functional requirements requested.

Property decisions would maintain the quality of the system by following a format during the implementations. How the resources are defined following the standards and the conventions. What frameworks should be used in the system and how the data for the service is consumed, would be decided under this property.

Executive decisions are implied through the business domain. These would contain the authentication and authorization levels that the service would be exposed to. Also, would define constraints on the tools that would be used during the development.

The relationship between these decisions are mapped including the constraints, excludes, enables, subsumes, conflicts with, is bound to, is alternative to and many other relations. The extracted information are stored in RDF stores to be used in the next phase.

3.2 Architectural Element extraction from the software document to provide reasoning

The requirements of the system to be developed, forms the architectural elements as well as the design out comes. When designing a software, the process of considering the multiple design methodology that could be applied and selecting the process to follow from the design space is software design process. Hence there are multiple architectural rationale that would be proposed for a single requirement.

To extract the required details, form the documents natural language processing techniques, are used. First all the words in a PDF document was identified, then these words were taken into further processing. According to the ontology structured, a dataset was trained in to identifying the text patterns that needs to be extracted. The word sets that had a similar pattern was identified to be provided with the design rationale.

3.3 Map the extracted elements with the existing ontology elements

The word sets that were identified in the previous section were mapped to the current ontology. For this, the property values of the text were used. Based on the property values the word set was aligned to an existing property mapping in the Resource Definition file.

An exact match with the property values is needed for this as the identification is based on the property values. Based on these insertions using the word sets extracted a new Resource Definition file was constructed.

3.4 Perform design reasoning

The reasoning is need in evaluating the alternative design and provide justification in to selecting the design process. Designing reasoning can be divided into two forms as design rationale and motivational reasons [35].

The motivation in developing the software would initiate from providing the solution to fulfill the need of the requirement. The aspects of motivation can be to achieve a sub goal or a goal, it can be a stimulus for a design, it can be influential by either supporting, constraining or rejecting a decision and it could be an assumption or a driving factor.

These motivational reasons would be an input for the design decision phase, and they should be represented explicitly. They can sometimes be conflicting as there are assumptions, requirements and constraints all included in it.

The conflicts that arise in the design process along with the evaluation of weakness and benefits of the alternatives that are available would form the design rationale. The design reasoning is the part in-between the motivational reasoning and the design rationale from which the design decision if formed.

To conduct the design reasoning already available tools such as Hermit or Pellet can be used. These reasoners infer information that are not explicitly contained in the ontology. This inferencing is done through classifiers. They provide different kinds of reasoner services [36].

- Consistency checking
- Equivalence checking
- Subsumption checking
- Instantiation checking

Consistency checking is done initially when the ontology is loaded on to the editor. In the context of REST API services, it should ideally fall into the second or third level in the maturity model. The consistency checking would make the ontology adhere to the class properties where for an instance the service cannot fall in to level zero and anyone of the other levels at the same time. This would be identified as an inconsistent class.

Subsumption checking is done when building the hierarchy. Primitive classes would appear, and these should basically be the requirements of the system which should be extracted as the architectural elements. There could be different alternatives in how this could be implemented.

The design rationale for an architecture element will be indicated through subclasses. These rationales can be either disjoint or joint when it comes to combining classes. This signifies that the alternatives that have been proposed for an architecture element can either coexist if they are not disjoint properties or they cannot coexist if they are disjoint properties. Disjoint means that two classes cannot share the same instance, regardless of how the classes will be interpreted.

For a given architecture element if it cannot be reasoned into two out of the given three alternatives which are disjoint through the inference logic it can be inferred to fall under the third alternative.

Polyhierarchies is a principle to form the class, which needs to have multiple parents. Asserting polyhierarchies is a bad practice since all the subclasses needs to be updates. The reasoner will form this hierarchy when the required information is given.

When modeling the architectural element and the architectural rationale a direct connection between them will be formed, bridging the link of architectural rationale trace. The rationale will encapsulate the design decision and will have a one to one mapping between them. Therefore, a rationale will associate with at least one effect and one cause. The links between architectural element and architectural rationale or architectural rationale and architectural element should be mapped in an acyclic graph. For design reasoning purpose, each architectural element will be mapped to multiple architectural rationale. The linkage between the elements and rationale is depicted in Figure 3.2. The architectural rationale will comprise of both the quantitative reasoning and qualitative reasoning stated out. Each of the architectural rationale will result in either a design outcome or several design outcomes.

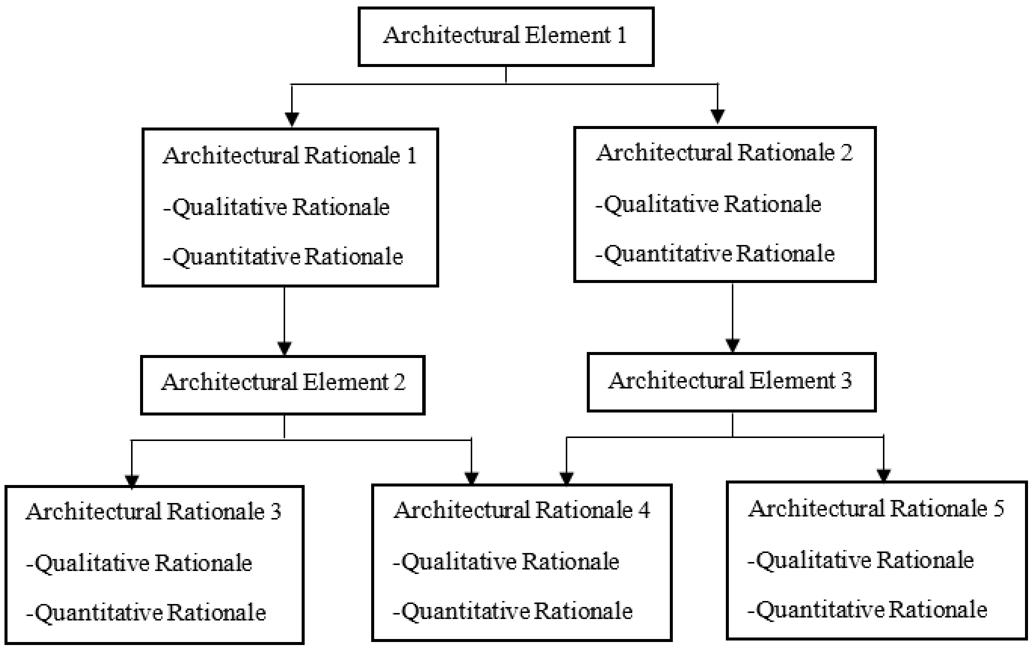


Figure 3.2: Architecture element and relation mapping

This flow will be documented in to providing a software design reasoning documentation for the software architectural elements presented.

These results would be the architectural elements that would be the input for the next architectural rationale process. Based on the rationale the design outcome can be selected, which is the design reasoning process. The reasoning process would consider the quantitative and qualitative facts. Based on these facts weights would be assigned and the outcome would be proposed.

When mapping the ontology, the architectural element will be a single entity. This entity would be extended by either a single or multiple architectural rationale. These architectural rationales will have a relationship with both qualitative and quantitative rationale. Under each of the rationale the rationale facts will be attributes. For qualitative attributes will contain assumptions, constraints and risks. The quantitative rationale will have attributes such as the costs and the benefits of acquiring the rationale.

For an example for a resource URI the best practice is to have one POST method. If there is a need to past many parameters for a GET request, then it will be better to pass them in the method body, converting it to a POST request. Then the practice would be to create another resource with the same name and appending query to it. This would cater for the querying of data under that resource.

If backtracking from the design outcome to the point of where the selection was made, the understanding on why that selection was made can be easily understood by the users. Using ontologies, the mapping can be tracked since it would be depicted in a hierarchical structure.

3.5 Generate design reasoning document

On the reasoning section multiple evaluations would be made. These evaluations would result in providing multiple design rationale. The rationale could also map an architectural element to the highest option available in the hierarchy. Hence when selecting the matching class, based on the ontology hierarchy the best option would be selected.

To capture the hierarchy structure in the ontology the super classes and subclasses existing in the RDF will stored in a data collection. Based on this mapping and the rationale provided on the reasoning section will be filtered out. This helped in providing the best justification for the elements. The filtered-out data was structured in a way that was convenient for a user to be read. The structured data was written to a PDF file finally resulting in a reasoning document.

3.6 Summary

The chapter provides a five-step approach into how the design reasoning documentation would be done through the research. The process would start with ontology creation. Next architectural element extraction from the documents which is. followed by, mapping the extracted elements with ontology words. Then design reasoning is applied and finally it would be documented.

Chapter 4

SOLUTION ARCHITECTURE AND IMPLEMENTATION

Documenting the architectural reasoning and the design decisions is a tedious task that consumes a lot of time and money. Since the time at the initial stage is not utilized for this purpose it is hard to recall the reasons why these decisions were taken in the initial stage. On this chapter the tools and techniques used in providing a solution to document the architectural design reasoning is presented. The implementation steps had to be broken down so that the tasks that are identified as dependent tasks would be executed at first. In achieving the implementation, the steps given below were followed.

- Extract the existing architectural elements included in the current architecture
- Map the existing elements to the knowledge base created using Part of Speech tagging
- Use the mapping logic to determine the relationship with the architectural elements and knowledge base elements.
- Based on the mapping extract the reasoning applicable for the architectural elements.
- Document the reasoning values that were extracted for the architectural elements.

4.1 Ontology creation

As the software architecture domain is a vast area the research was narrowed to a narrower scope in the software architecture domain. The REST service-oriented architecture was selected as the domain for the research.

To gather the knowledge needed in creating an ontology existing documents available on the REST service-oriented architecture was searched on the internet. The material available provided the best practices followed during the development and it also included white papers that were published on this area by the domain experts. Tutorials on the design followed when implementing restful services were looked at as a source that would provide domain knowledge. Using these materials, the information was captured that were necessary in providing the design rationale to the architectural decisions.

4.1.Ontology structuring

When defining an ontology using the OWL format, all the classes that would be defined in the ontology will be subclasses coming under the Owl, Thing. Extending from the OWL Thing, a hierarchical class structure for the rest service-oriented architecture was defined. The ontology was defined based on three main areas as the Architecture, Requirement and the Rest concepts.

4.1.1.1 Architecture

Under the Architecture the restful style, the client server interaction and the application programming interface will be captured. The application programming interface will include features like addressability which specifies the identifier of the URL, the stateless interaction which discusses the session state being stored only at the client server and should be passed in with the client request on each call. The stateless interaction will be further divided in to reliability, scalability and visibility. Also, it would include the uniform interface concept where each resource would be accessible with a HTTP end point.

4.1.1.2 Requirement

The requirement section would discuss regarding the features and the functionalities expected from the target system. This is divided in to three sections as functional, nonfunctional and concerns. The functional requirement defines the expected behavior for a given input and its resulting output. The nonfunctional requirements could be one specification where the system would be judged. The nonfictional requirements would consist of the quality of the application, security, speed, reliability and other domain specific factors. The concerns would capture the different perceptions of the stakeholders regarding the turn out of the application and how it will be perceived by the end users.

4.1.1.3 Restful service

The structure for the restful service was formed through breaking down the core concepts that are followed in during the development of a restful service. There are patterns that are followed which are considered as the standards and have proven to provide better results when applied on the real environment throughout the industry. The key topics were identified as are resources, requests, response, representation, headers, authentication, cache and URI.

The resource was divided further in to data model and HTTP method. The data model defines the different structures that the resource can be defines as. The resource end point can define results for a single entity which could be an atomic resource. The end point could also result in a group of entities which will be a collection resource. Also, there are composite resources which manipulated multiple entity types and provides an aggregated result to the end user. The controller resource is used when there are multiple resources to be manipulated but the result should be provided in one API call. The HTTP method consists of the four main requests types. A GET request retrieves the data values, POST request creates a new entity, PUT request updates an existing entity and DELETE request would delete an existing entity.

The representation of the resource could be in either CSV, JSON, XHTML or XML format. This format can be specified in the request header as the content type on the request and as the content type accepted from the response.

The resource headers could be broadly divided in to request headers and response headers. The request headers would contain the authorization mechanism which could be user name and password or could be a session token. It also includes the accepted response format, the format type of the request body. It also provides conditional responses like if the entities are modified from the time the request is made and could provide information if there was any match with the parameters that were passed. The response headers could include the content location of the response and the content type of the response body. It could also include the last modified timestamp of the entities. The response could comprise of an ETag, which is considered as the fingerprint of the resource in the current state. The authentication schema also is included in the response with the indication of the access level that could be used in determining if the user has access.

The response will include a response status which could either be a success status or a failure status. Different types of success status are captured under which could be either the request is accepted, or a new instance is created, either the response only returned okay status or either the request was successfully executed but there is no response. The success category falls in to the 20X response status. The redirect responses which could be either not modified or see other falls into the 30X response category. The 40X response category represents the client error. This error type could either be due to a bad request meaning the content passed could be in an unrecognized format or the parameters are incorrect which could also result in the request going to the not accepted state. Could also be the request resource is not found. The user could be unauthorized with the session expiring or not containing the required authentication level. The request could also have its preconditions failed or the requested acceptance type format might not be supported by the resource.

The responses with the 50X status are server error status types. When the service is unavailable or due to the response time lag the gateway timeout response can be returned. If the requested resource is not implemented, then the response will state that the resource is not implemented. When the server is acting as a proxy or gateway and gets a request for an invalid request then the bad gateway response will be returned.

During processing the request an error occurs due to bad data, inefficient code or a database error an internal server error would be returned in the response.

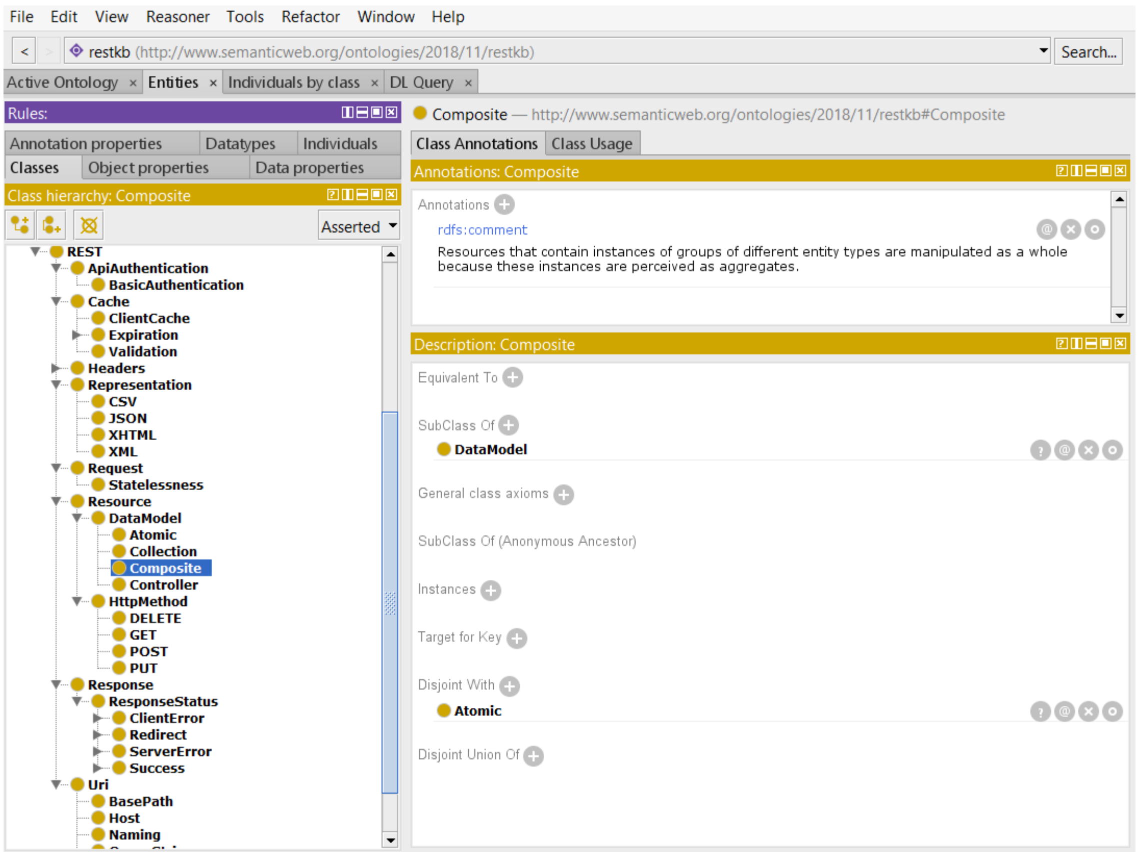
The format of the request URI will comprise of different types of information to the server. All the request URI would have a host which specifies the domain in which the requests will be processed. The host name will be followed with a base path in which the request resource is specified. It is a best practice when writing the base path to follow the method conventions such as always using plural nouns and using the hyphen annotation in between separating the words. The schema in the URI provides protocol that needs to be followed in the API. The query strings provide the conditional parameters which needs to be fulfilled in the request. The path parameters in the request will be processed as URI templates where the curly braces in the request path will be replaced with the parameter given. Also, the URI can follow a versioning schema which will provide backward compatibility of the resource to its consumers.

Cache is also a feature supported for RESTful services. This caching could be either on the client end or the server end. Client cache will be processed by the RAM of the client machine. The server cache will be handled in the server end and would contain an expiration time and a validation schema for the requests coming in.

4.1.2 Ontology format

According to the structure modeled to create the ontology different tools were available in modelling it to OWL format. Using the OWL API library, the ontology could be created using Java as well. Protégé which is an open-source ontology editor also allows in creating an ontology and since it gives a visual structure of the model it is easier to structure the ontology. It provides tools for creating domain-based knowledge models. Protégé is created based on the OWL API which helped in the latter process in the research to read the ontology and perform reasoning based on the model.

Protégé 5.2.0 editor was used in creating the ontology. Figure 4.1 displays the classes defined when creating the ontology. On the editor to start creating an ontology a new ontology option was selected and for every ontology a unique IRI must be given as the identification of the ontology. This IRI will be used as the prefix for all the classes and properties that would be added into the ontology.



```
REST
  ApiAuthentication
    BasicAuthentication
  Cache
    ClientCache
    Expiration
    Validation
  Headers
  Representation
    CSV
    JSON
    XHTML
    XML
  Request
    Statelessness
  Resource
    DataModel
      Atomic
      Collection
      Composite
      Controller
    HttpMethod
      DELETE
      GET
      POST
      PUT
  Response
    ResponseStatus
      ClientError
      Redirect
      ServerError
      Success
  Uri
    BasePath
    Host
    Naming
```

rdfs:comment
Resources that contain instances of groups of different entity types are manipulated as a whole because these instances are perceived as aggregates.

SubClass Of: DataModel

Disjoint With: Atomic

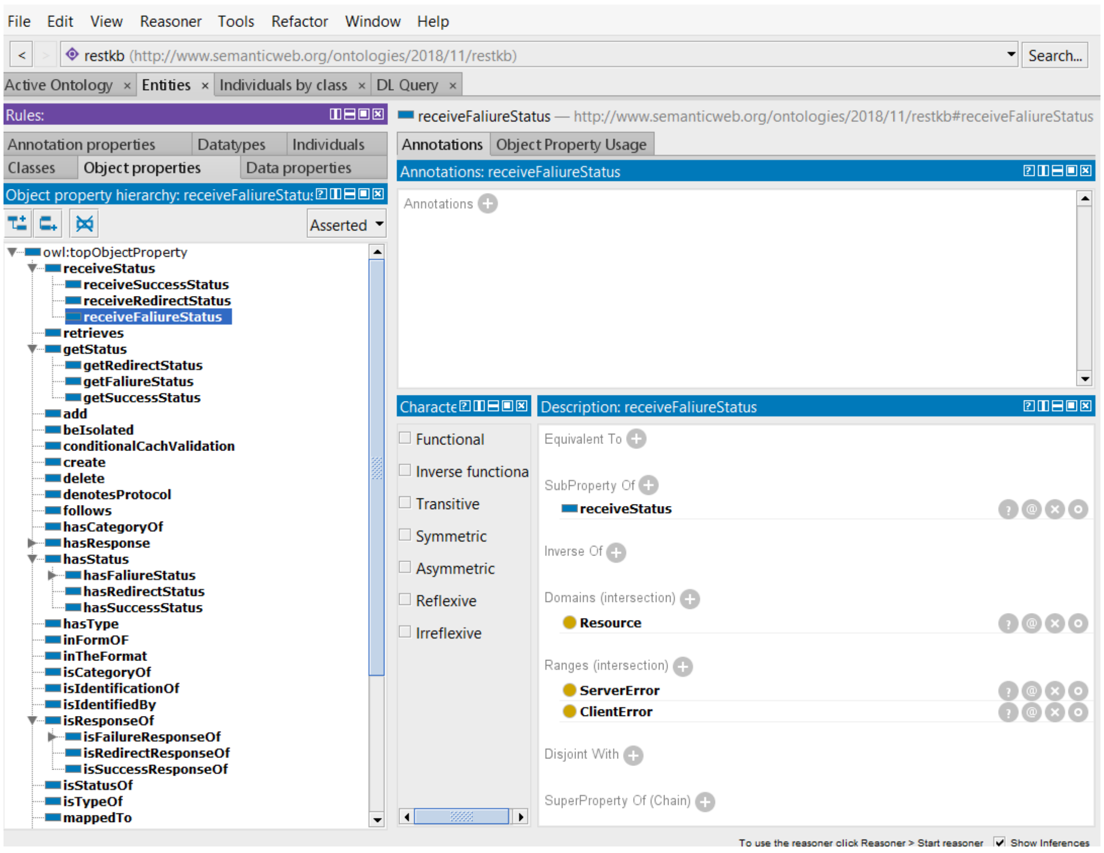
Figure 4.1: Class hierarchy of the knowledge model

The IRI name given to the REST domain knowledge base was "http://www.semanticweb.org /ontologies/2018/11/restkb#". The format that was used in saving the ontology was RDF/XML. On the editor the classes and the object properties were mainly used when populating the knowledge base.

The architectural elements or the final design decisions were extracted from the materials. These extracted decisions were structured in a hierarchical way to construct the ontology. The elements were added to the ontology as classes in Protégé editor. In representing the hierarchy, the subclass, superclass concept was used. There were instances were the same class could be a sub class of two different classes and this was mapped on the subclass adding both superclass entries.

For each class created the class annotation property was included with the annotation type of rdfs:comment. The comments were placed for each class explaining its relevance to the architecture and why it has been used. These comments were used in defining the reasoning for the architectural elements used.

The object properties were defined to map relationship among the classes. The object properties also followed the superclass, subclass format. A property can also be an inverse of an existing property. All the properties included a domain and data range of the classes that can be applied with that property. This restricted the instances that could map with a property. Figure 4.2 shows the object properties defined on the ontology modeled for the project.


File Edit View Reasoner Tools Refactor Window Help
restkb (http://www.semanticweb.org/ontologies/2018/11/restkb)
Search...
Active Ontology
Entities
Individuals by class
DL Query
Rules:
Annotation properties
Datatypes
Individuals
Classes
Object properties
Data properties
Object property hierarchy: receiveFaliureStatus
Asserted
receiveFaliureStatus — http://www.semanticweb.org/ontologies/2018/11/restkb#receiveFaliureStatus
Annotations
Object Property Usage
Annotations: receiveFaliureStatus
Annotations
Characte
Description: receiveFaliureStatus
To use the reasoner click Reasoner > Start reasoner
Show Inferences

```
owl:topObjectProperty
  receiveStatus
    receiveSuccessStatus
    receiveRedirectStatus
    receiveFaliureStatus
  retrieves
  getStatus
    getRedirectStatus
    getFaliureStatus
    getSuccessStatus
  add
  beIsolated
  conditionalCachValidation
  create
  delete
  denotesProtocol
  follows
  hasCategoryOf
  hasResponse
  hasStatus
    hasFaliureStatus
    hasRedirectStatus
    hasSuccessStatus
  hasType
  inFormOF
  inTheFormat
  isCategoryOf
  isIdentificationOf
  isIdentifiedBy
  isResponseOf
    isFailureResponseOf
    isRedirectResponseOf
    isSuccessResponseOf
  isStatusOf
  isTypeOf
  mappedTo
```

☐ Functional
☐ Inverse functiona
☐ Transitive
☐ Symmetric
☐ Asymmetric
☐ Reflexive
☐ Irreflexive

Equivalent To

SubProperty Of
receiveStatus

Inverse Of

Domains (intersection)
Resource

Ranges (intersection)
ServerError
ClientError

Disjoint With

SuperProperty Of (Chain)

Figure 4.2: Object property hierarchy of the Knowledge model

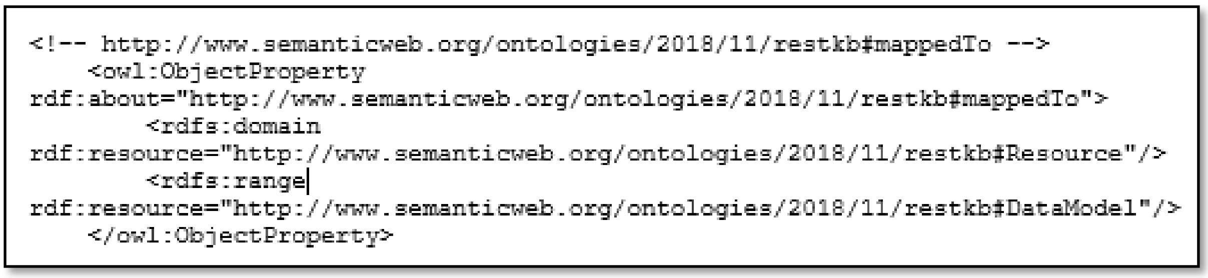
The knowledge base constructed for the REST service domain included a total of 94 classes and 52 object properties.

Figure 4.3 and figure 4.4 represents a sample architecture represented as a class and a sample object property defined in RDF/XML format.

```
<!-- http://www.semanticweb.org/ontologies/2018/11/restkb#Collection -->

    <owl:Class
rdf:about="http://www.semanticweb.org/ontologies/2018/11/restkb#Collection">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/ontologies/2018/11/restkb#DataModel"/>
        <rdfs: comment>Resources that contain atomic resources of the same type that
needs to be grouped into a set. </rdfs: comment>
    </owl:Class>
```

Figure 4.3: Sample architectural element represented as a class



```
<!-- http://www.semanticweb.org/ontologies/2018/11/restkb#mappedTo -->
    <owl:ObjectProperty
rdf:about="http://www.semanticweb.org/ontologies/2018/11/restkb#mappedTo">
        <rdfs:domain
rdf:resource="http://www.semanticweb.org/ontologies/2018/11/restkb#Resource"/>
        <rdfs:range
rdf:resource="http://www.semanticweb.org/ontologies/2018/11/restkb#DataModel"/>
    </owl:ObjectProperty>
```

Figure 4.4: Sample architectural element represented as a class

4.2 Solution Architecture

According to the analysis done in the previous section the solution architecture would comprise of the following components.

- OWL file reader
- Document reader
- Word pattern extractor
- Reasoning generator
- Document generator

Given in Figure 4.5 is a layered diagram of the proposed architecture for the project. The presentation layer will have a document reader. The business layer would have a word pattern extractor, a reasoning module and a document generator. The information needed for the processing is captured from the data access layer using the OWL file created and the cache maintained by reading the file. Other than these layers the cross-cutting concerns applied at all layers are the exception handling and logging for the tracking purpose if there are any errors.

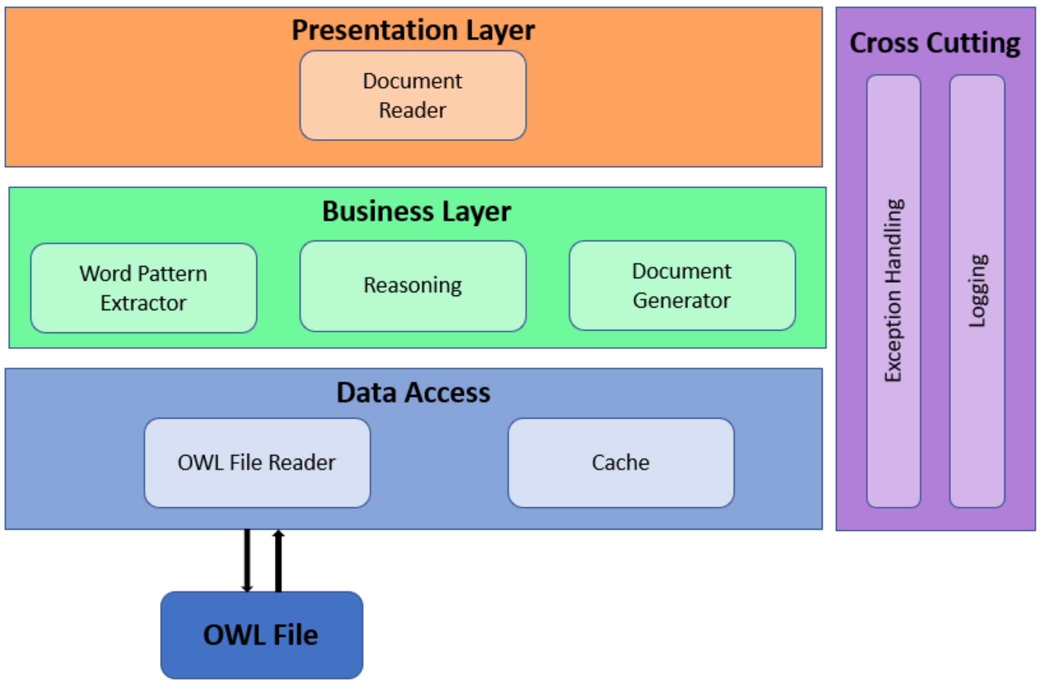


Figure 4.5: Layered diagram of the proposed system

For the development purposes Java 1.8.0_144 version was used. Java was chosen as the development language mainly since the OWL API that was used by Protégé when constructing the ontology is supported by Java. For the latter process for NLP there were libraries available in Java.

4.2.1 OWL file reader

The OWL API 5.1.7 version was used in processing the ontology created using the Protégé editor [37]. Using API, the ontology manager was created. This manager would be then used in reading the existing ontology and writing an inferred ontology in the later part. Using the manager, the RDF\XML formatted owl file was read by loading the ontology from existing OWL document. With the ontology reference created the classes defined in the ontology and the properties were extracted which will be needed for the processing.

4.2.2 File reader

PDF documents were chosen as the sources of documents that needed to be provided with reasoning. To read the PDF documents from Java Apache PDFBox 2.0.6 library [38] was used. This is an opensource library which allows to create new PDF documents, manipulate the existing document content and extract the content of an existing document.

Using this library, the document contents were extracted as a stream of words to be provided with reasoning documentation. To extract the words the “PDDocument” class was used and the file was loaded. Then using PDF text stripper by area method, the words were assigned to a String variable.

4.2.3 Word pattern extractor

The stream of words that were read using the PDF reader must be processed to extract the words needed. For the extraction the knowledge base that was created needs to be used. From the stream of words, the phrase of word patterns that would align with the knowledge model had to be extracted.

To process the stream of words extracted, natural language processing techniques were used. Apache OpenNLP 1.5.3 library helped in executing this task. Apache OpenNLP is from the name itself an opensource library that is based on machine learning which performs NLP on the text [39].

A sample architecture document was selected, and its content was copied into a text file. All the words in this file were then manually annotated with the necessary architectural references and matching knowledge base wordings. The words that needed to be disregarded were annotated with “unknown” text specifying that it is not needed to be considered when extracting the phrases. This annotated text was used as the training model and was the input for the Part of Speech (POS) model. Using the

PosModel available in the OpenNLP library, based on the annotated text a trained text was generated. This trained text was then used in extracting the text phrases needed from the documents.

The extracted stream of words was captured from the Apache PDFBox, was first broken down into sentences using the OpenNLP library. This list of sentences was then used in the processing to extract the word phrases. The trained POS model was used in extracting the word phrases from the list of sentences. The process was iterated three times and the most proposed tagging for the provided word, was selected as the matching tagging for a given word.

4.2.4 Reasoning generator

Using the word phrases that were extracted from the REST documents using POS tagging a match was performed first with the property values. When a match with the property values is found the words neighboring the property would be extracted. The neighboring words will be the words that would be included to construct the inferred ontology. The properties along with the two words associated with it will be extracted to a data structure. These words will be included in an inferred ontology as individuals associated with the property. After the individuals have been included in the ontology the reasoning will be applied.

HermiT reasoner version 1.3.8.4 was used in providing reasoning for the created individuals. Using the “ReasoningFactory” an owl reasoner was created. For the reasoner the classes, subclasses and the disjoint classes associated with the individual were included as the axioms that needed to be generated for the inferred ontology. Using the computed axioms for the individuals a new inferred ontology was created using the reasoner and the generated axioms. The newly inferred ontology was written to a new owl file in RDF/XML format so that it could be used during the documentation purposes.

4.2.5 Document generator for Architecture Reasoning

The final step in this project is to generate software architectural design documentation included with design reasoning. The inferred ontology generated in the previous step and the word phrases extracted for the reasoning were both used in to generating this document.

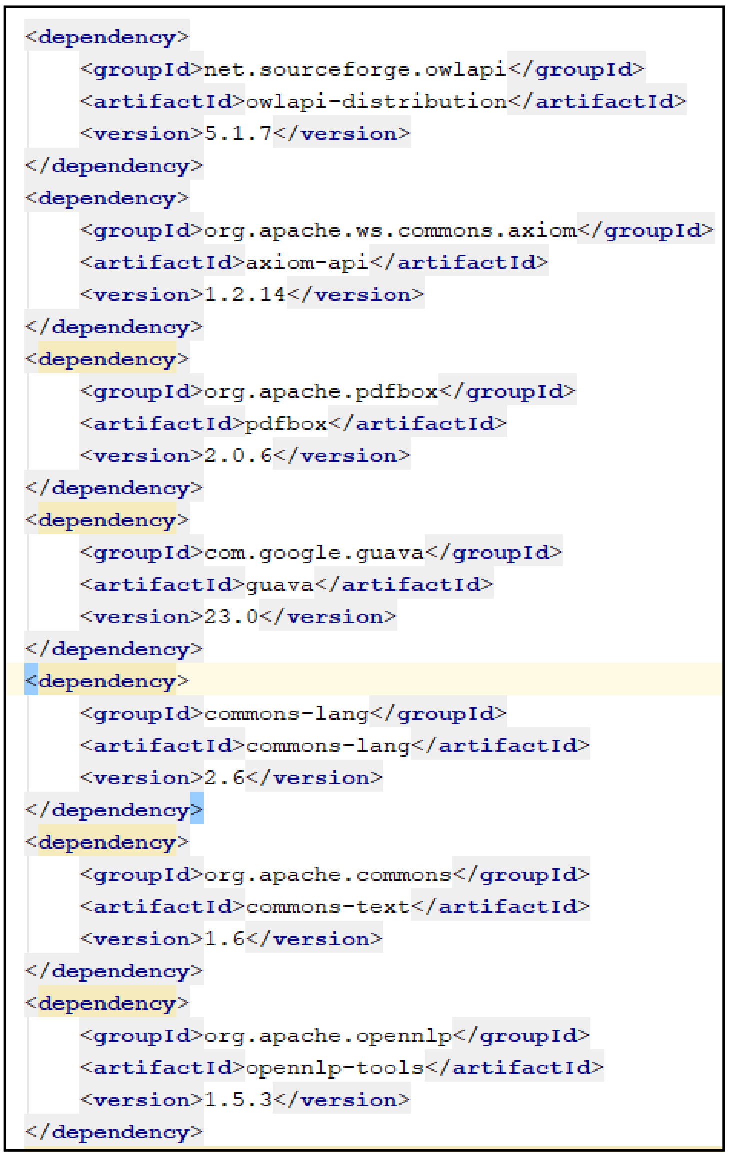
From each word phrase stored in a list, the two words passed in when generating the inferred ontology will be taken. For these words the Ontology IRI will be prefixed and using this the owl individual could be extracted. For this individual the owl classes that have been inferred could be extracted from the ontology. since the ontology is comprised of a super class, subclass basis the inferred class types for the individual would comprise of multiple classes starting for the superclass to the subclass. To find the most specific class type for the individual the generic classes needs to be disregarded.

To rule out the generic classes, the super class subclass hierarchy needs to be mapped. Using the ontology, the hierarchy of the superclass subclass will be mapped to a collection. Based on this collection values the generic classes from the inferred class types will be removed. The remaining class will be taken as the best match for the extracted words. For each class when creating the ontology, a comment annotation was added. This comment annotation will be taken for the inferred class type for the individual.

To document the extracted class types along with their comment annotation values, a new PDF document will be created. For this the Apache PDFBox library will again be used. The document will be structured in a way where it would first write the extracted word and then the class type that was inferred with reasoning. Then using the comment annotation, the architectural rationale will be documented.

4.3 Prototype Implementation

To develop the prototype project of the proposed solution maven, build automation tool was used. Some of the dependencies that were required for the development are depicted in figure 4.6. Other than these dependencies for logging purposes the sl4j dependency was added.



```xml
<dependency>
    <groupId>net.sourceforge.owlapi</groupId>
    <artifactId>owlapi-distribution</artifactId>
    <version>5.1.7</version>
</dependency>
<dependency>
    <groupId>org.apache.ws.commons.axiom</groupId>
    <artifactId>axiom-api</artifactId>
    <version>1.2.14</version>
</dependency>
<dependency>
    <groupId>org.apache.pdfbox</groupId>
    <artifactId>pdfbox</artifactId>
    <version>2.0.6</version>
</dependency>
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>23.0</version>
</dependency>
<dependency>
    <groupId>commons-lang</groupId>
    <artifactId>commons-lang</artifactId>
    <version>2.6</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-text</artifactId>
    <version>1.6</version>
</dependency>
<dependency>
    <groupId>org.apache.opennlp</groupId>
    <artifactId>opennlp-tools</artifactId>
    <version>1.5.3</version>
</dependency>
```

Figure 4.6: Sample list of maven dependencies

After adding those dependencies, the class structure of the project was formatted according to figure 4.7

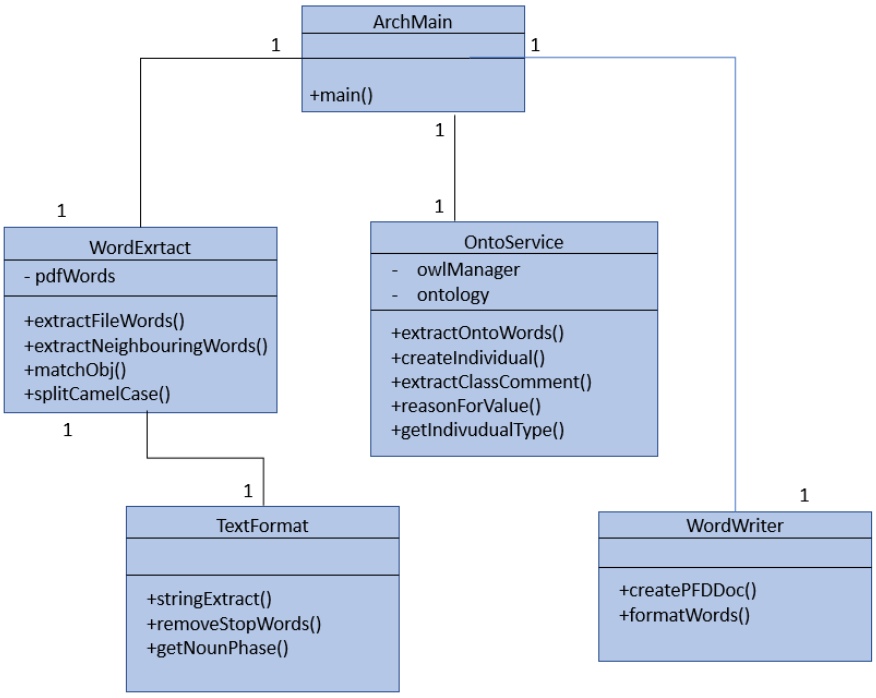
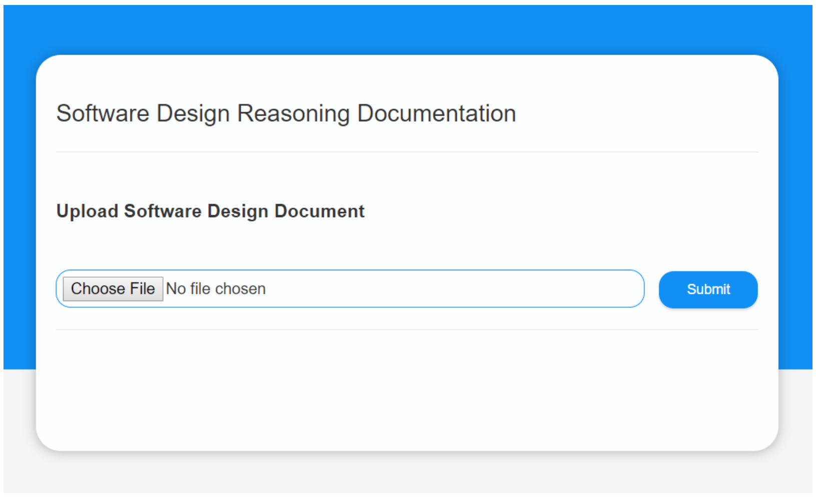


Figure 4.7: Class diagram of the prototype project

The main method which would trigger the entire reasoning process was included in the "ArchMain" class. Using Spring Boot the user interface was loaded using with localhost and on 8080 port. The user interface was designed using bootstrap for CSS and jquery was used in implementing the logic on the UI. Given in Figure 4.8 is the user interface that was developed for this tool.


Software Design Reasoning Documentation

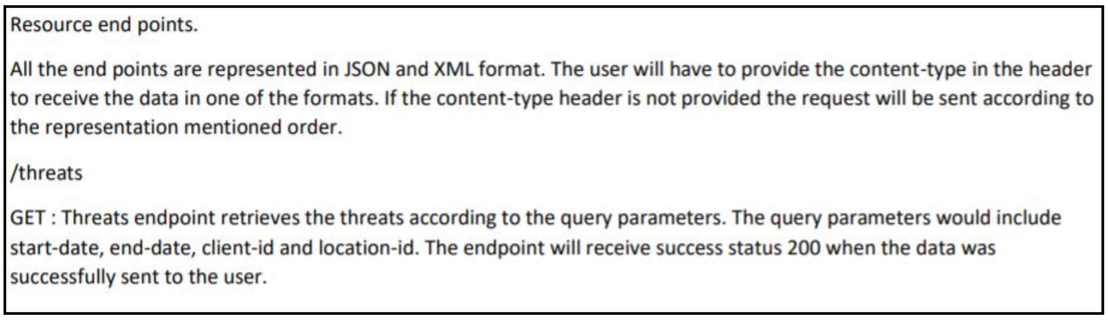
Upload Software Design Document

Choose File No file chosen

Submit

Figure 4.8: User interface for the design reasoning tool

Through the user interface a user can input a software design document to generate reasoning. As the implementation has been based on the RESTful architecture domain design material from that domain needs to be provided. Given in Figure 4.9 is a sample software document given to the tool.


Resource end points.

All the end points are represented in JSON and XML format. The user will have to provide the content-type in the header to receive the data in one of the formats. If the content-type header is not provided the request will be sent according to the representation mentioned order.

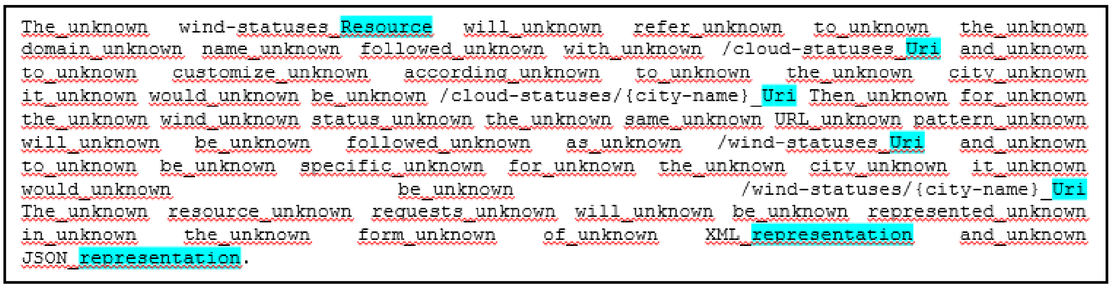
/threats

GET : Threats endpoint retrieves the threats according to the query parameters. The query parameters would include start-date, end-date, client-id and location-id. The endpoint will receive success status 200 when the data was successfully sent to the user.

Figure 4.9: Sample software design document

When the software document is given, to start the process first the “OntoService” class has to be instantiated. Using the service classes, words and properties included in the ontology owl file was extracted. These words were then provided to the “WordExtract” class. The PDF document that given was read using this class. Based on the words provided by the ontology, the words were extracted according to the matching class values and property values.

For the next extraction phase the annotated text-based training model was used. To train the annotated text model around twelve documents were used that were available online. Around six properties were annotated in the training model with the same name as the ontology class. The other words were annotated as “unknown”. Given below in figure 4.10 is a sample annotated text. Using this training model each word in the provided text was suggested with the most appropriate word. With the suggested words the architectural elements suggested were extracted,



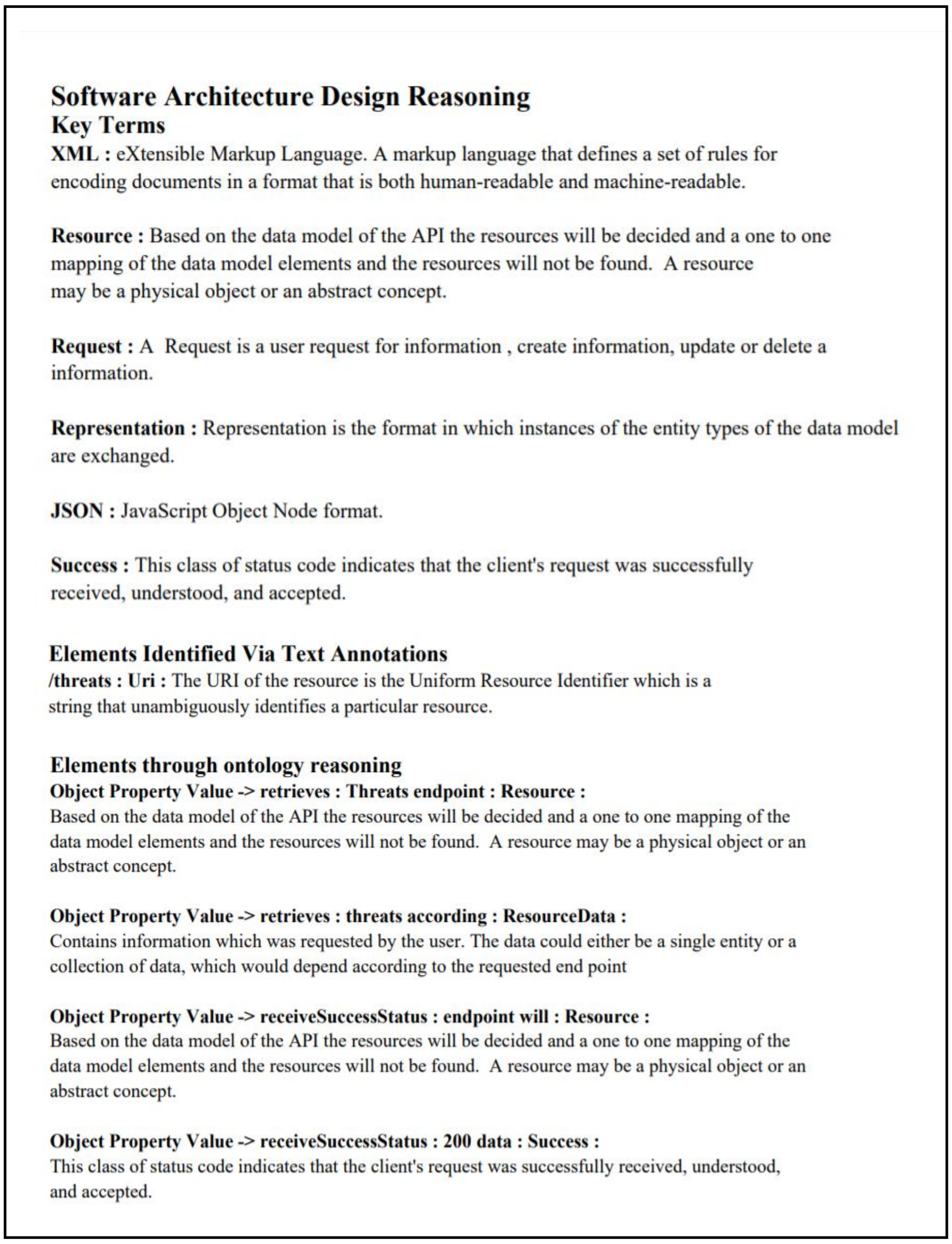
```
The_unknown wind-statuses_Resource will_unknown refer_unknown to_unknown the_unknown
domain_unknown name_unknown followed_unknown with_unknown /cloud-statuses_Uri and_unknown
to_unknown customize_unknown according_unknown to_unknown the_unknown city_unknown
it_unknown would_unknown be_unknown /cloud-statuses/{city-name}_Uri Then_unknown for_unknown
the_unknown wind_unknown status_unknown the_unknown same_unknown URL_unknown pattern_unknown
will_unknown be_unknown followed_unknown as_unknown /wind-statuses_Uri and_unknown
to_unknown be_unknown specific_unknown for_unknown the_unknown city_unknown it_unknown
would_unknown be_unknown /wind-statuses/{city-name}_Uri
The_unknown resource_unknown requests_unknown will_unknown be_unknown represented_unknown
in_unknown the_unknown form_unknown of_unknown XML_representation and_unknown
JSON_representation.
```

Figure 4.10: Sample annotated training model

The “TextFormat” class was used in extracting the sentences of the paragraphs in the documents. After the sentences were identified the stop words of the sentences were removed and noun phrase extraction was also used to process some words. This formatted text was again provided to the word extractor and here the object properties were mapped. For this the object properties which were listed in camel case notation was first formatted as normal words. Then these words were searched on the text that was formatted.

For the object properties the neighboring words also had to be extracted since they would then be provided to onto service to be reasoned with the ontology model to be mapped with the ontology classes. The object properties with its neighboring words would be used in mapping them as individuals in an inferred ontology.

Using the inferred individual types and the class properties extracted, the comment annotations for each type would be searched on the owl file. Finally using the “WordWriter” class the values that were extracted would be written to a PDF doc. Given in figure 4.11 is a reasoning document generated for a given software design document.


Software Architecture Design Reasoning
Key Terms
XML : eXtensible Markup Language. A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

Resource : Based on the data model of the API the resources will be decided and a one to one mapping of the data model elements and the resources will not be found. A resource may be a physical object or an abstract concept.

Request : A Request is a user request for information , create information, update or delete a information.

Representation : Representation is the format in which instances of the entity types of the data model are exchanged.

JSON : JavaScript Object Node format.

Success : This class of status code indicates that the client's request was successfully received, understood, and accepted.

Elements Identified Via Text Annotations
/threats : Uri : The URI of the resource is the Uniform Resource Identifier which is a string that unambiguously identifies a particular resource.

Elements through ontology reasoning
Object Property Value -> retrieves : Threats endpoint : Resource :
Based on the data model of the API the resources will be decided and a one to one mapping of the data model elements and the resources will not be found. A resource may be a physical object or an abstract concept.

Object Property Value -> retrieves : threats according : ResourceData :
Contains information which was requested by the user. The data could either be a single entity or a collection of data, which would depend according to the requested end point

Object Property Value -> receiveSuccessStatus : endpoint will : Resource :
Based on the data model of the API the resources will be decided and a one to one mapping of the data model elements and the resources will not be found. A resource may be a physical object or an abstract concept.

Object Property Value -> receiveSuccessStatus : 200 data : Success :
This class of status code indicates that the client's request was successfully received, understood, and accepted.

Figure 4.11: Generated reasoning document

4.4 Summary

Summarizing the steps mentioned in this chapter, first a knowledge base that could be used for the rest domain was constructed. Then using this knowledge base, the architectural elements were captured. Based on these elements the reasoning was applied and finally the reasoning outcome was documented for providing the architectural rationale.

Chapter 5

EVALUATION

The evaluation of the proposed implementation was carried out under several steps. This evaluation steps depended on the structure on how the documents that needed to be provided with was structured. Based on the data that was extracted from the documents, the productivity and the accuracy of the reasoning generated differed.

The key outcome of this research was to provide reasoning into the architectural approaches that were taken during the implementation of the projects. Basically, three approaches were taken in the identification. The key words used in the domain were identified from the text based on the knowledge base that was created. The technique used for this was to extract the word phrases and match them with the knowledge base to extract reasoning though the knowledge base. The next step that was followed was identifying the main architectural components through the word usage. Also find the architectural elements through the trained annotated model and discover the matching elements.

During the design phase of a project there could be multiple reasons into why a decision was made. But those decisions not being documented made it a tedious task for those who sought for those answers during the latter part of the project or during modification times or due to requirement changes.

5.1 Correctness of the prototype application

The flow of the prototype application is shown on figure 5.1. The flow can be defined as:

- Insert the software architecture document to the system.
- Load the restful service ontology.
- Identify architectural elements and key words.

- Document the architectural reasoning according to the elements.

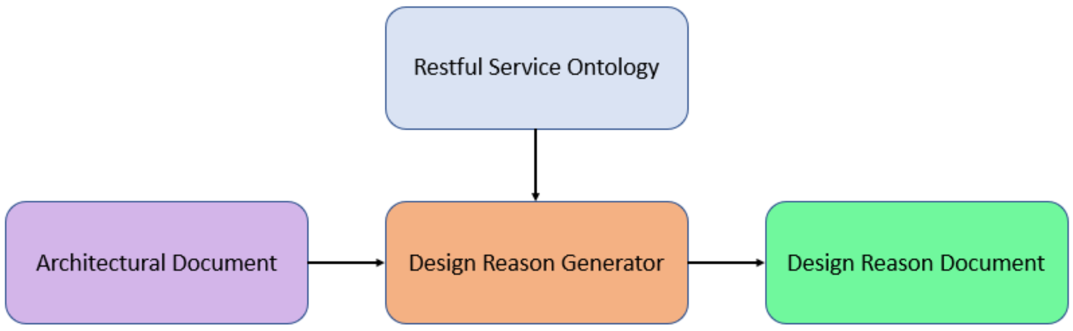


Figure 5.1: Basic flow of design reasoning generator

As given in the diagram the architectural document had to be processed first and the text extracted from that section will be processed. For this the restful service domain ontology will be used to provide the design reasoning. Using the knowledge base and the text that was extracted the design reasoning generator will provide reasoning to the architectural elements. The design reasoning that was captured will be documented in the next step. This documentation will be the final step in the design reasoning prototype project proposed.

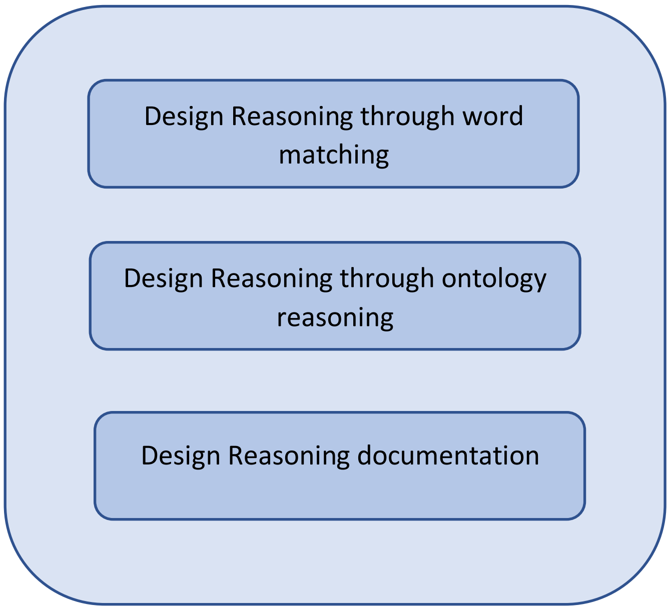

Design Reasoning through word matching
Design Reasoning through ontology reasoning
Design Reasoning documentation

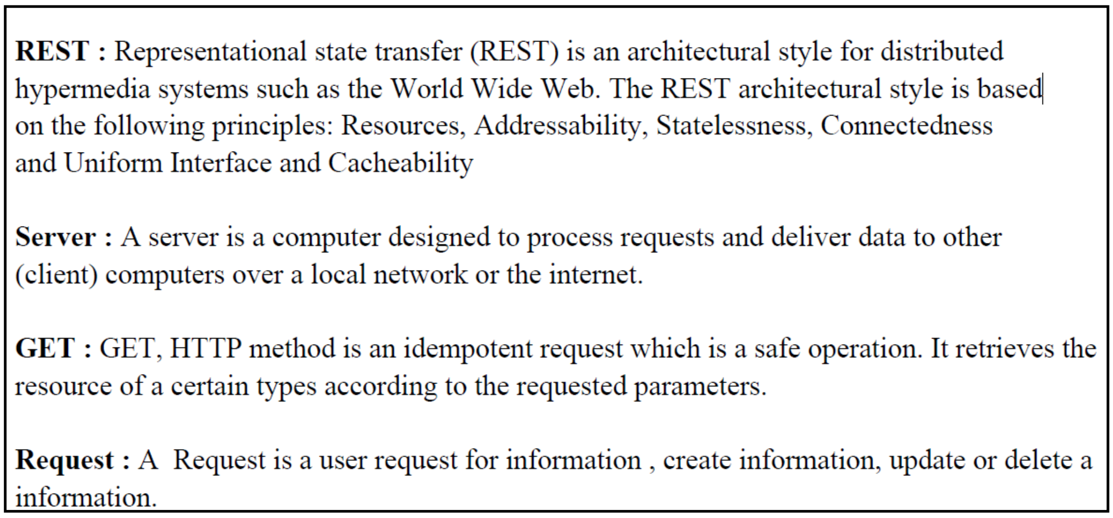
Figure 5.2: Basic components of the design reason generator

5.1.1 Evaluation of the success path of the application

To evaluate the application was functioning correctly on a happy path without any crashes and exceptions few steps were followed. As the application only accepts PDF documents a sample document was created.

- The created document included lot of matching architectural key words that were the knowledge base included.
- The architectural elements were used similar in structure that was included in the training documents.
- The words used in the properties were only included when used in between architectural elements that needed to be identified.
- The documents were clearly structured without diagrams making the processing easier.

The document was inserted to the system and the output document was presented to the user capturing the key words with along the explanations. Given below are sample output documents. In figure 5.3 the matching key terms that were identified on the text was selected and along with the key terms. The key terms are given with an explanation on its usage.



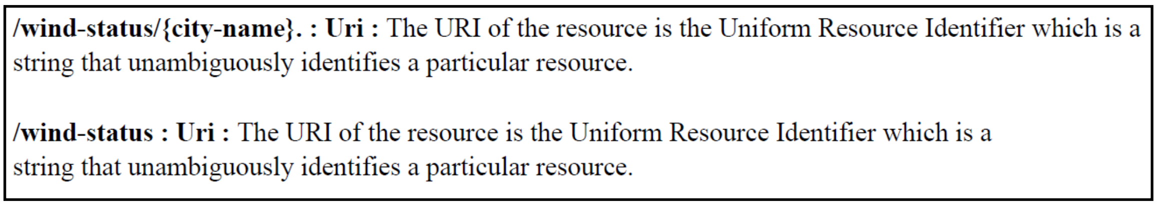
REST : Representational state transfer (REST) is an architectural style for distributed hypermedia systems such as the World Wide Web. The REST architectural style is based on the following principles: Resources, Addressability, Statelessness, Connectedness and Uniform Interface and Cacheability

Server : A server is a computer designed to process requests and deliver data to other (client) computers over a local network or the internet.

GET : GET, HTTP method is an idempotent request which is a safe operation. It retrieves the resource of a certain types according to the requested parameters.

Request : A Request is a user request for information , create information, update or delete a information.

Figure 5.3: Matching key terms being identified



/wind-status/{city-name}. : Uri : The URI of the resource is the Uniform Resource Identifier which is a string that unambiguously identifies a particular resource.

/wind-status : Uri : The URI of the resource is the Uniform Resource Identifier which is a string that unambiguously identifies a particular resource.

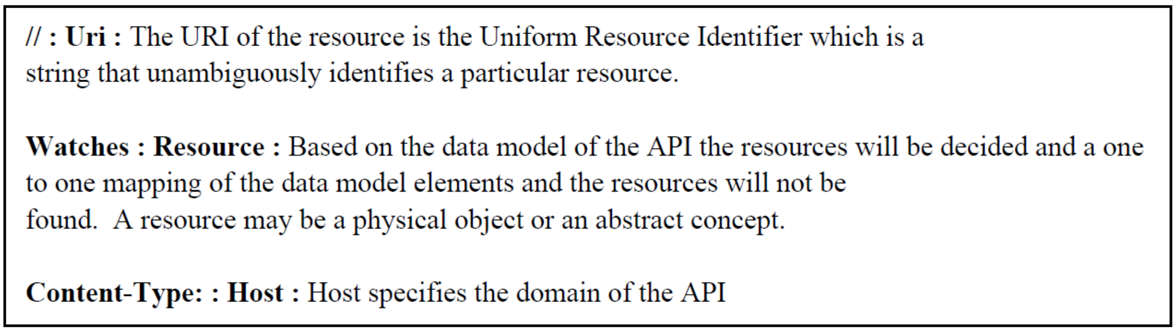
Figure 5.4: Derived architectural elements

Figure 5.4 displays the derived architectural elements that were captured from the application. Hence in a success path scenario the application would perform as accepted and provide accurate results.

5.1.2 Evaluation of the failure path of the application

To evaluate how the application would respond to the failure path, software documentation that were created for other domains were selected. Since the application consumes PDF documents the documents selected were converted to PDF and the evaluation was performed. Some key words were misidentified due to their usability in different contexts. Such words were “schema”, “delete”, “get”, “atomic” and some others.

Figure 5.3 are derived architectural elements that were misidentified during this failure path evaluation. These elements were identified since the similar patterned words were used in the training data. Since these were mentioned as actual elements it is misleading for the architectural reasoning purpose.



// **: Uri :** The URI of the resource is the Uniform Resource Identifier which is a string that unambiguously identifies a particular resource.

Watches : Resource : Based on the data model of the API the resources will be decided and a one to one mapping of the data model elements and the resources will not be found. A resource may be a physical object or an abstract concept.

Content-Type: : Host : Host specifies the domain of the API

Figure 5.5: Misidentified architectural elements

Hence rather than producing a document stating that design reasoning could not be generated for the produced document it would return to the user an invalid document. This invalid document would be misleading for users and hence the failure scenario should be improved in to detecting the domain the software documentation before generating the document.

5.2 Performance evaluation

The evaluation for the architectural design reasoning was carried out in several steps. This approach was taken to calculate the accuracy of the approaches separately and to get an estimation on the best approach that resulted in the highest accuracy. Given below are the three approaches that was followed.

- Identify the RESTful service-oriented key words using context words from the ontology
- Extract architectural elements from the context with annotated text
- Extract architectural elements using ontology reasoning

To evaluate the applications performance using actual documents, the application was deployed in a machine with the following parameters.

- Processor: Intel Core i7 7th gen processer with 2.70 GHz frequency containing 2 processor cores
- Random Access Memory: 8GB
- Operating system: 64bit, Windows 10
- JVM: 64bit, 1.8 JVM

The experiments were performed using several software architecture documents to measure the accuracy under each of the sections that are mentioned. The document paragraphs were first manually evaluated to gather the architectural elements that were included on them. Using these manually evaluated results and the values calculated from the experiment the precision was calculated along with increasing the number of architectural elements in each case.

To calculate the precision the true positive (TP) which will be the actual architectural elements and the false positives (FP) which will be the elements that would be incorrectly identified as architectural elements has to be calculated. Using these counts the precisions for each technique will be calculated using Equation 1.

$$precision = TP/(TP + FP) \tag{1}$$

5.2.1 Key word identification using ontology classes

To start with evaluation the knowledge base that was constructed as the domain-based ontology was first loaded into the system. The paragraphs of the selected documents were given to the application one after the other and the counts were taken for the correctly identified architectural key words. The count was also taken for the architectural key words that were proposed as architectural key words but were not actual key words.

Based on these counts the precision was calculated through increasing the actual architectural key words count of the paragraphs. The counts taken at each step is given under table 5.1.

Table 5.1: Counts taken for the evaluation on matching key words

Actual Key words	4	5	6	7	8	9	10
True Positive	3	4	4	7	8	7	7
False Positive	1	1	3	3	2	4	3
Precision	0.75	0.8	0.57	0.7	0.73	0.64	0.7

Figure 5.6 graph shows the precision calculated, along with the actual architectural key words at each scenario.

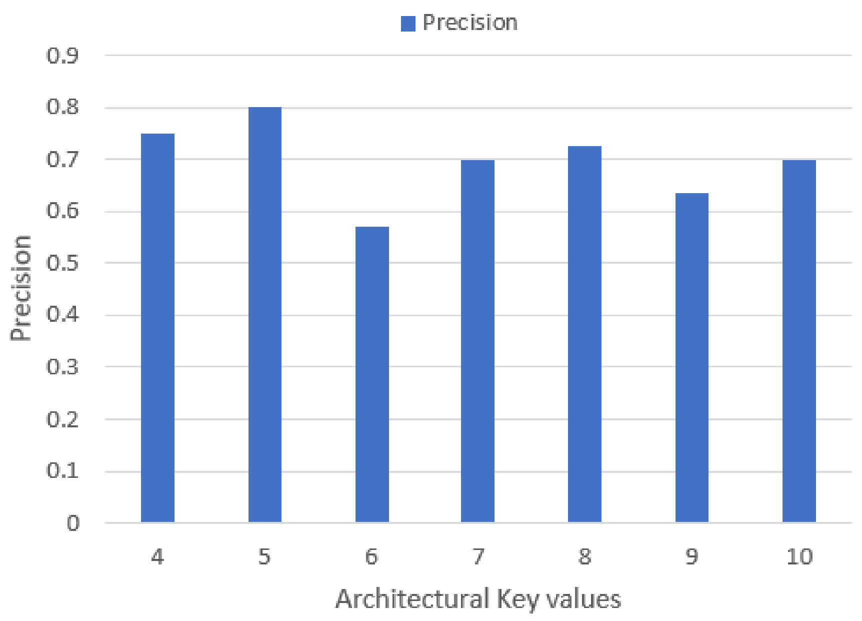


Figure 5.6: Precision of architectural key words identification for matching context words

According to precisions calculated, the value varied between 0.8 and 0.57. The average precision of this technique resulted in 0.7. Mostly the context words that were included in the ontology itself were identified and documented. Normal words on the paragraph were also incorrectly identified as architectural key words was the main reason for the accuracy to be lower at some evaluations.

Since an exact match of the words was performed when the words are used even for a different meaning they could be extracted as a correct match. The false identification was sometimes not visible to the user. This was since the unique key words were provided with the reasoning explanations and if an incorrectly identified key word was also included in the correctly identified set of key words. Then it would not be an incorrect identification.

5.2.2 Architectural element extraction with annotated text

The architectural elements that were captured using this technique were not explicitly defined as architectural elements. Hence using the word annotation technique, the words were derived. Mainly four architectural elements were used for the training of the annotated data. The derived architectural elements were based on these trained values.

The paragraphs with architectural elements were increased by one when the experimental counts were taken. The correctly identified architectural element count and the incorrectly identified architectural count were taken separately. Based on these values the precision for each round was calculated. The counts for each step are provided in table 5.2.

Table 5.2: Counts taken for the evaluation on derived elements based on trained data

Actual Arch. Elements	4	5	6	7	8	9	10
True Positive	2	2	3	4	4	5	5
False Positive	2	2	2	2	3	3	4
Precision	0.5	0.5	0.6	0.66	0.57	0.63	0.55

Based on the precision values the graph can be plotted along with the number of actual architectural elements at each step which is given in figure 5.7.

Analyzing the precision values calculated for this technique the values fluctuated between 0.5 and 0.66. A pattern on the increase or decrease in values was not visible as the precision was dependent on the words that are available on the text.

A higher precision was not achieved as in most cases words were incorrectly identified as architectural elements. The most misidentified element was the resource element type. When annotating paragraph, the words that were annotated as resources were plural nouns. Hence after a training model was created the plural nouns were evaluated as resource element types on the testing data set.

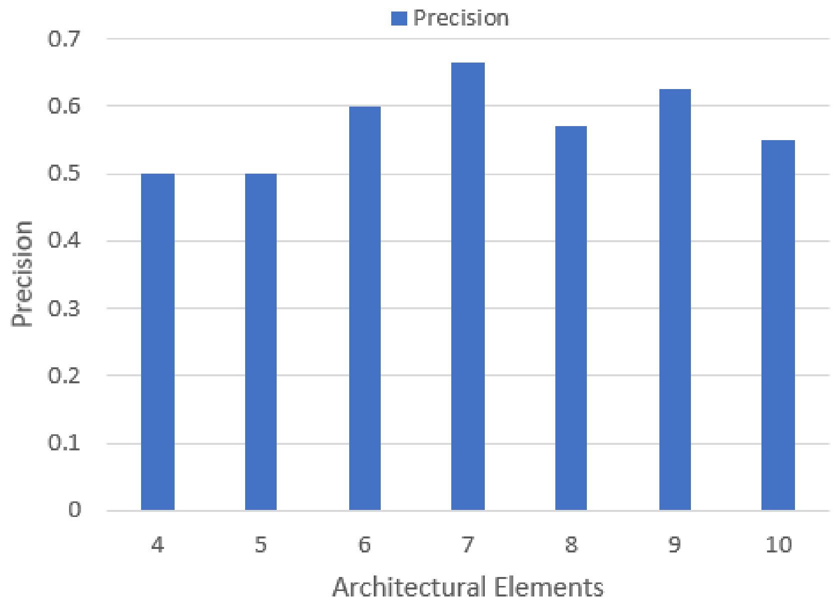


Figure 5.7: Precision of architectural element identification using annotated text

The precision was marginally on 0.5 and above. The average precision of this technique was calculated as 0.572. Annotating the sentences with POS technique and building the trained data should be done for a broader data set. Based on the training data set the test data would have provided a better result if the test data was also always selected from the same domain as the training data was selected from.

5.2.3 Architectural element extraction using ontology reasoning

The final technique used in extracting architectural elements was to use the ontology reasoning technique. In this technique also the elements were not defined as architectural elements and had to be derived from the paragraph using the ontology property values that were identified. As in the previous techniques the correctly identified architectural elements and the incorrectly identified architectural element counts were considered in calculating the precision in using this technique. The values calculated are given in table 5.3.

Table 5.3: Counts taken for the evaluation on derived elements based on ontology reasoning

Actual Arch. Elements	4	5	6	7	8	9	10
True Positive	1	2	2	3	3	3	3
False Positive	2	2	3	3	3	4	4
Precision	0.33	0.5	0.4	0.5	0.5	0.43	0.43

Based on the calculation of the precision the values were plotted in a graph according to the increasing architectural elements.

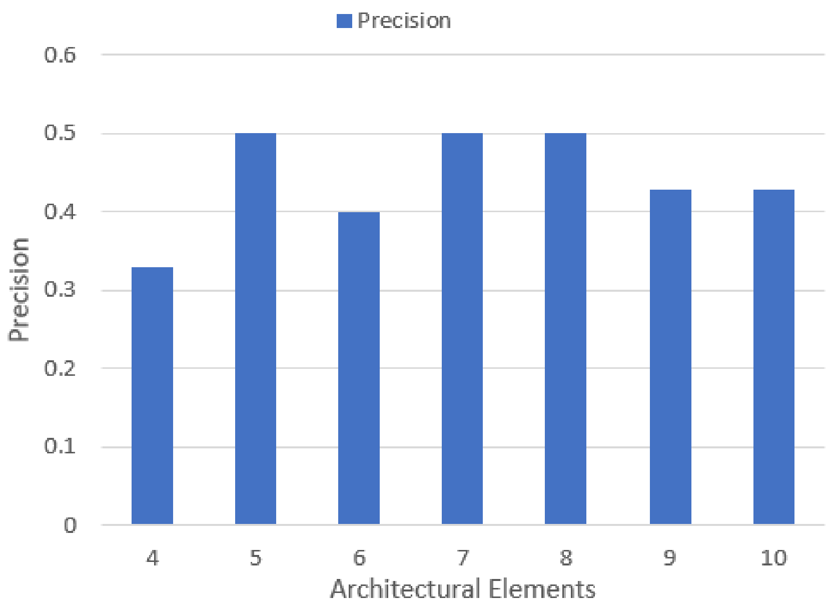


Figure. 5.8: Precision of architectural element identification using annotated text

According to the calculated precisions given in Figure 5.8, the maximum precision was recorded as 0.5 and the lowest precision was recorded as 0.33. The average of the precision values was calculated as 0.44. The values for this technique was gathered using the neighboring elements of the properties defined on the ontology. The property values are verbs that are used in between two architectural elements. Based on that assumption the neighboring words of the property word was extracted.

The precision using this technique did not achieve a high level as much of the elements identified were incorrectly proposed. The properties included in the ontology are common words that could be used in sentences for different purposes hence this misidentification was common in using this technique.

5.3 Expert Evaluation

To validate the accuracy levels achieved seven individuals were randomly selected from the industry, who are involved in the development and documentation of the software projects. Hence for the sample selection simple random sampling technique was used. Some software document that were created for projects that were using the RESTful architecture was provided. These documents contained between eight to thirteen key words and the architectural elements that needed to be derived were between six to ten. Based on these facts the documents were input to the tool and the architectural reasoning documents were generated.

Table 5.4 provides the counts that were taken during the evaluation and the precision according to the values were calculated.

Table 5.4: Counts taken for the expert evaluation

Document	1	2	3	4	5	6	7
Actual Arch. Elements and key words	15	17	14	15	16	14	16
True Positive	7	8	7	6	8	8	9
False Positive	6	5	5	5	7	8	7

Precision	0.54	0.61	0.58	0.54	0.53	0.5	0.53

The precision values calculated for the provided input is shown in Figure 9. According to the graph values it can be visible that the average precision value was around 0.54. The precision value varied based on the key wards and the elements that had to be captured from each document.

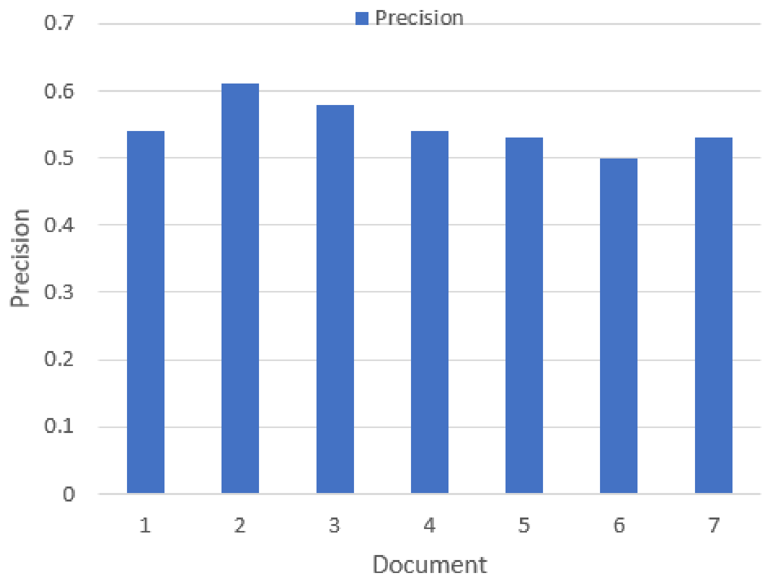


Figure 5.9: Precision for the expert evaluation

The precision for each document supplied varied between 0.5 and 0.61. This variation was minor and was due to some of the design reasoning not being captured by the ontology. Hence, this depicts that this tool is not subjective to the users writing patterns and techniques. The precision is achieved through the completeness of the ontology and how rich its knowledge base is comprised of.

5.4 Evaluation of the importance and usefulness of the application

To identify the importance of documenting the software architecture design reasoning and its current practice in the IT industry a survey was created and distributed among few members in the IT community.

The survey comprised of questions starting from the general information to capture the background of the participants of the survey. It then included questions regarding the usage of tools used for software documentation, time spent during the project for documentation and their opinion on the importance of documenting the design reasoning for a software project. The questionnaire is provided under Appendix A.

Summarizing the general background of the participants, most of the responses were collected from industrial people who were within the age range of 25 to 30 years. Around 60% of the participants are engaged in the industry as software engineers. The industry experience of the participants varied from 52.3% being from 2 to 5 years of experience range and 31.3% being from 6 to 10 years of work experience backgrounds.

On the questions regarding the software design documentation except for 3.1% the other responses stated that they maintain documentation on the projects that are developed. The key tools used for documenting purposes were Confluence and Jira. Since this was a multiple options question 50% and 50.8% provided those answers respectively. A percentage of 48.4% was found for using word documentation.

When questioned if the design documentations included the architecture on them 46.1% responded as they would often include the architecture while 35.2% responded as they would always include the architecture in their software project documentation. Figure 5.9 shows the percentage if the explanations for the design decisions were documented. The users often include the explanations in the documents but 32% rarely included them in the documentation.

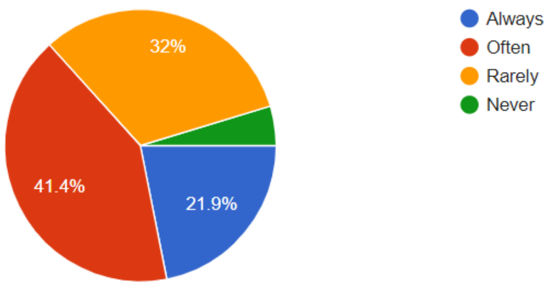


Figure 5.10: Responses in percentage on the inclusion of design reasoning in documentation

Even though the majority answered as they often included the design reasoning on the documents and not always, 55.5% strongly agreed that it is important to have design reasons during a requirement change or maintenance work.

Some of the tools that organizations have used for architectural designing are Visio, Mule Soft, Lucid Charts and AgroUML. 46.8% of the organizations allocate time often for software design documentation and 25.3% always allocate time for this purpose.

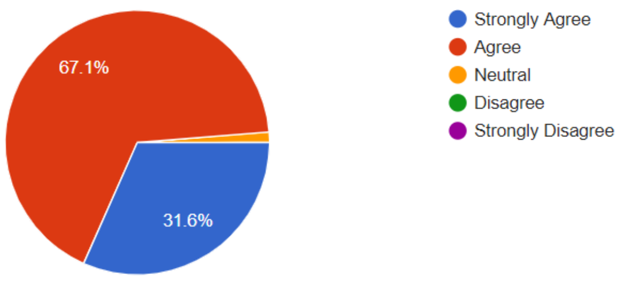


Figure 5.11: Response in percentage on the importance of design reasoning in software development

When questioned, if they would consider that software documentation as a waste of resources time 55% disagreed while 37.5% strongly disagreed on this question. The question raised on if the users considered that software design documentation is important the responses were 67.1% agreed and 31.6% strongly agreed that it is important. This is displayed in figure 5.10.

When the participants were questioned if they would consider using a tool to generate software design reasoning for the software project documentation the following was the response as given in figure 5.11. From the participants 47.5% would like to consider using a tool for this purpose and 25% would very likely use a tool.

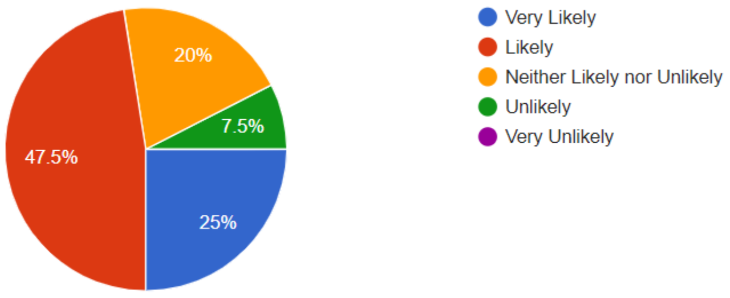


Figure 5.12: Response on using a tool to generate software design reasoning.

From the survey carried out, the participants from the IT industry do consider that software design reasoning is important for a project and would likely to allocate time for this purpose. Also, if a tool was available for this purpose there is a high probability of using such a tool to generate the document.

5.5 Summary

With the three techniques used into generating design reasoning, the overall average of the precision was calculated as 0.58. The design reasoning document was generated using the elements that were captured. The key word identification gave the best results when documenting while the other two techniques had to derive the elements using the knowledge base. These techniques heavily depended on the NLP techniques hence a high precision could not be achieved. The approach used in this research is not subjective to a user, as the terms used when documenting the software will not differ. The completeness of the ontology will increase the accuracy of the tool. The survey results proved that a design reasoning tool is useful and would be used in the industry if one has been developed.

Chapter 6

CONCLUSION

Developing a software application that would satisfy the requirements of the stakeholders complying with the best software practices in a timely manner is the main target of the software development companies. In achieving this target, the focus is given to the product. The documentation of the project details is sometimes considered as minor priority. Hence sometimes it is neglected as it is overlooked as a time-consuming task and that time could have been allocated in improving the features of the product. The project documentation is usually documented to capture the features but the reasoning behind the architectural decisions is not included on them.

6.1 Research contribution

The research focuses on creating a knowledge base with the architectural elements of the software designs with its architectural rationale. Using this knowledge base created and the software project documentation, an approach to document the design reasoning for the architectural elements presented in the documents is proposed. Different technical approaches have been suggested for this and in this research, an ontology-based approach is followed.

For the proof of concept, the knowledge base was created for the RESTful service domain. The knowledge base included the architectural elements and terms that were identified on this domain. With this knowledge base using three techniques the key words and the derived architectural elements of the software project documentation was identified. The values that were identified along with the reasoning provided was written to a document.

If this needs to be applied to other domains, the ontology needs to be constructed identifying the key words and elements of those domains. The properties that would be defined in the ontology needs to be refined so that the ambiguous words would not be included in them. For each domain, training data should be gathered and must be annotated with the architectural elements which are also captured in the ontology. The training should be chosen from a data set capturing the different writing patterns of the users. This will provide better results for the part of speech tagging technique as it could identify different patterns and extract the words correctly. It also avoids words ambiguously being extracted as architectural elements as the training data set would cover a variety of the patterns.

For design reasoning different approaches has been suggested but an ontological approach has not been implemented and evaluated. Hence through this research a different approach for design reasoning documentation has been suggested. This approach has been proven to be successful through the prototype application that has been developed.

6.2 Limitations of the research approach

The main research limitation under this research is that the domains concepts and techniques must be captured in the ontology and if they haven't been included, they will not be correctly identified. Hence when creating the knowledge base, the online material as well as domain expert knowledge must be captured and included in the knowledge base. To derive the architectural elements the property values included in the ontology must be selected carefully and not all the common words should be included. Since in the third technique used to derive the architectural elements, the properties of the ontology will play the key role. If common words ae used as properties it would misidentify elements. Hence for the ontology reasoning section, the exact wordings with the property value should be matched to correctly capture the architectural elements.

The research was limited to the software project documentation done on the RESTful service domain. Hence to apply this to other domains, ontologies needs to be constructed including the design decisions and elements. The application with all the domain knowledge together has not been tested and how the identification of the elements and its accuracy hence is doubtful since there could be misleading decisions.

6.3 Future work

The prototype developed was only for one domain hence this needs to be expanded to other domains as well. The knowledge base created needs to be more refined and inserted with concepts to improve on the ontology. The relationships between the key values need to be improved by defining unambiguous words which would result in better accurate results.

The NLP techniques that were used in deriving the elements need to be improved. Training of the data must be done for a variety of writing patterns capturing the different writing styles of the writers. The trained data should be used to derive elements when the test data results in a high accuracy. When deriving elements using properties, a similar approach to annotated values could be used to identify if the elements could be derived as property phrases. Then the values will not be raw since it will go through one scan. A format could be introduced when writing the project documentation so that the processing the document could be done in a similar format.

According to the prototype tool developed the proposed approach to document the architectural design reasoning proved to be a success. It could identify the key terms used in the documents and derive the architectural elements that was presented in the software project documents. This tool could be used by any senior or junior developer as through the evaluation process it was observed that this tool’s accuracy was not subjective to the person who would be using the tool. The architectural elements and the key terms used in a specific domain will be included in the ontology. These wordings will be used by the people documenting the software project hence the accuracy will differ only based on the completeness of the ontology.

To apply this tool into a different domain the OWL file must be created with the domain related knowledge. The training dataset must also be created with the material available for that domain to be inserted to the tool to derive architectural elements.

REFERENCES

[1] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," microfilm, Irvine, 2000.

[2] D. E. Perry and A. L. Wolf, "Foundations for the Study of Software Architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40-52, 1992.

[3] D. Garlan and D. Perry, "Introduction to the Special Issue on Software Architecture," *IEEE Transactions on Software Engineering - Special issue on software* , vol. 21, no. 4, pp. 269-274, 1995.

[4] V. Reyna, "IN TWO MINDS: DUAL-PROCESS THEORIES OF REASONING AND RATIONALITY," pp. 5–8, 2006.

[5] P. Calogero, "Dilemmas in a General Theory of Fieldwork," *Berkeley Planning Journal,* vol. 22, no. 1, 2009.

[6] H. Vliet and A. Tang, "Decision Making in Software Architecture," *Journal of Systems and Software,* pp. 638-644, 2016

[7] L. Karsenty, "An Empirical Evaluation of Design Rationale Documents," in *CHI Papers,* 1996.

[8] J. Lee and K. Y. Lai, "What's in Design Rationale?," *Human Computer Interaction,* vol. 6, no. 4,. pp. 251–280, 1991.

[9] A. MacLean, R. M. Young, V. M. E. Bellotti and T. P. Moran, "Questions, Options, and Criteria: Elements of Design Space Analysis," *Human-Computer Interaction,* vol. 6, no. 3, pp. 201- 0, 1991.

[10] R. Capilla, A. Jansen, A. Tang, P. Avgeriou and M. Ali Babar, "10 years of Software Architecture Knowledge Management," *Journal of Systems and Software,* pp. 191-205, 2016

[11] L. Bratthall, E. Johansson and B. Regnell, "Is a Design Rationale Vital when Predicting Change Impact? A Controlled Experiment on Software Architecture Evolution," in *Product Focused Software Process Improvement,* Oulu, Finland, 2000.

[12] D. Clements, D. Garlan, L. Bass and J. Stafford, "Documenting Software Architectures: Views and Beyond," in *ICSE'03,* 2002

[13] A. Tang, M. A. Babar, I. Gorton and J. Han, "A survey of architecture design rationale," *Journal of Systems and Software,* vol. 12, no. 1792-1804, p. 79, 2006.

[14] A. Tang and J. Han, "Architecture Rationalization: A Methodology for Architecture Verifiability, Traceability and Completeness," in *E C.B.S 2055,* USA, 2005..

[15] A. Tang, J. Han and R. Vasa, "Software Architecture Design Reasoning: A Case for Improved Methodology Support," *IEEE Software,* vol. 26, no. 2, pp. 43-49, 24 February 2009.

[16] W. De Neys, "Implicit Conflict Detection During Decision Making," *In: Proceedings of the Annual Conference of the Cognitive Science Society,* vol. 29, pp. 209-214, 2007

[17] H. W. J. Rittel, "The Reasoning of Designers," Working Paper A-88-4, Institut fiir Grundlagen der Phmung, Stuttgart, 1988.

[18] N. Cross, "Creative Thinking by Expert Designers," *The Journal of Design Research,* vol. 4, no. 2, 2004.

[19] A. Tang and H. Vliet, "Software Architecture Design Reasoning," in *Software Architecture Knowledge Management: Theory and Practice,* 2009.

[20] A. Tang and J. Han, "Architecture rationalization: A methodology for architecture verifiability, traceability and completeness," *Proc. 12th IEEE Int. Conf. Work. Eng. Comput. Syst. ECS 2005,* vol. Compendex, pp. 135–144, 2005.

[21] A. Tang and A. Aleti, "Human Reasoning and Software Design: An Analysis," 2014.

[22] R. Mizoguchi and J. . V. Welkenhuysen, "Task ontology for reuse of problem solving knowledge," *Towards Very Large Knowledge Bases,* 1995.

[23] R. Damasevicius, "Ontology of Domain Analysis Concepts in Software System Design Domain," in *Information Systems Development: Towards a Service Provision Society,* 2006, pp. 319-327.

[24] G. Guizzardi, "On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models.," 2006.

[25] P. Kruchten, "An Ontology of Architectural Design Decisions in Software-Intensive Systems," in 2nd Groningen Workshop on Software Variability, 2004.

[26] P. Kruchten, P. Lago and H. Vliet, "Building Up and Reasoning About Architectural Knowledge," in Quality of Software Architectures, Springer-Verlag, 2006, pp. 43-58.

[27] H. Alani et al., "Automatic ontology- extraction from web documents," IEEE Intel. Syst., pp. 14–21, 2003.

[28] W. C. Daya and D. Dou, "Ontology-based information extraction: An Introduction and a survey of current approaches," Journal of Information Science , vol. 36, no. 3, pp. 306-323, 2010.

[29] T. R. Gruber, "A translation approach to portable ontology specifications," KNOWLEDGE ACQUISITION, vol. 5, no. 2, pp. 199-220, 1993.

[30] I. Maedche and S. Staab, "Ontology Learning for the Semantic Web," Intelligent Systems, IEEE, vol. 16, no. 2, pp. 72-79, 2001.

[31] H.-J. Happel and S. Seedorf, "Applications of ontologies in software engineering," 2006.

[32] P. Kroha and J. Labra Gayo, "Using Semantic Web Technology in Requirements

Specifications," Chemnitzer Informatik-Berichte, CSR-08-0, 2008.

[33] M. Fowler, "Richardson Maturity Model," ThoughtWorks, 18 March 2010. [Online].Available: https://martinfowler.com/articles/richardsonMaturityModel.html. [Accessed 25 January 2006]

[34] B. C. Henry, "RESTFUL SERVICES – APPLYING THE REST ARCHITECTURAL STYLE," Denver, Colorado, 2011.

[35] A. Tang, "A Rationale-based Model for Architecture Design Reasoning," 2007

[36] J. Sun, H. H. Wang and T. Hu, "Design Software Architecture Models using Ontology," in *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011),* Miami, 2011.

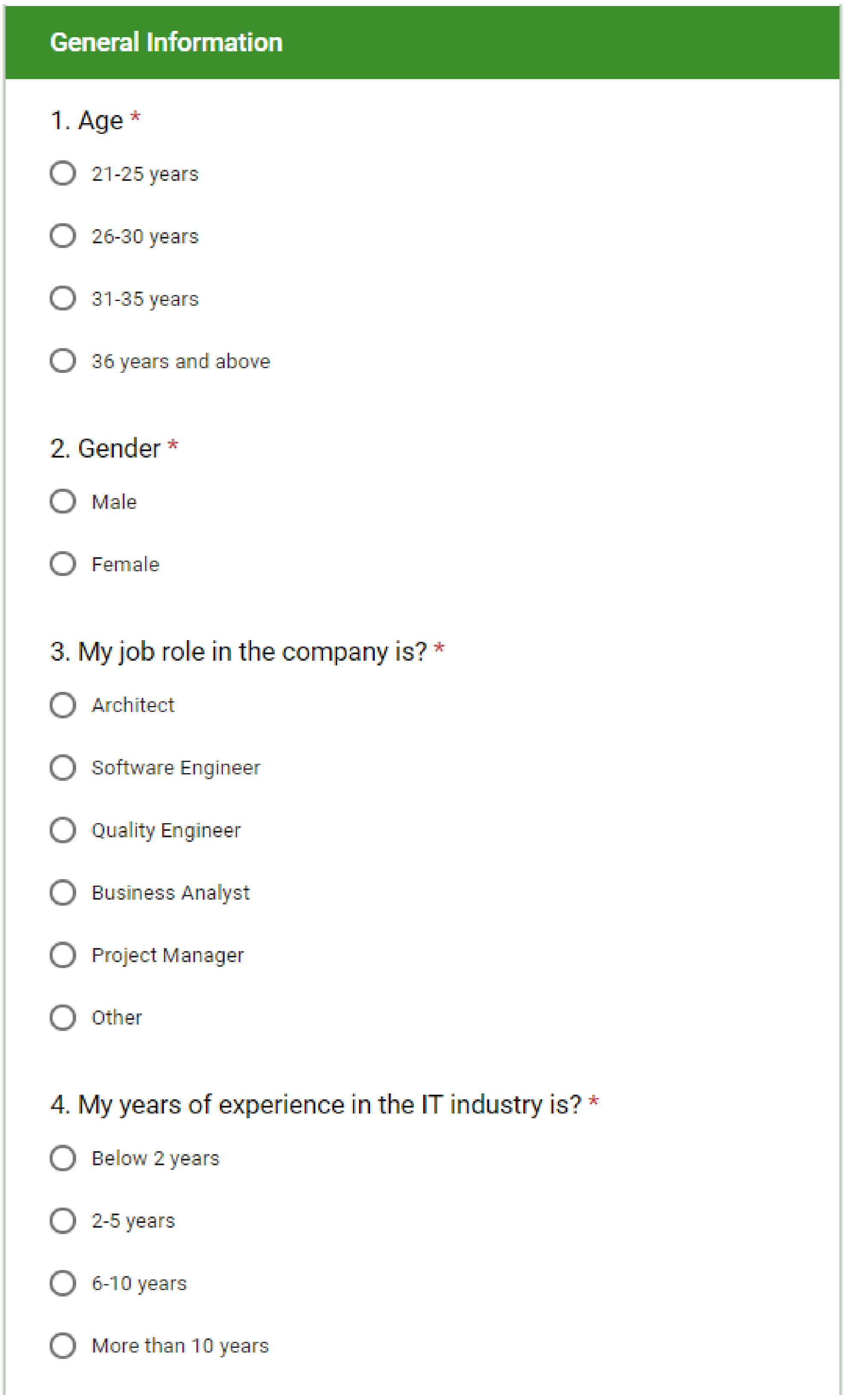
[37] "protégé," Stanford Center for Biomedical Informatics Research, Stanford University School of Medicine, [Online]. Available: https://protege.stanford.edu/. [Accessed 11 January 2019].

[38] "Apache PDFBox® - A Java PDF Library," The Apache Software Foundation, [Online]. Available: https://pdfbox.apache.org/. [Accessed 23 February 2019].

[39] "OpenNLP," The Apache Software Foundation, [Online]. Available: https://opennlp.apache.org/. [Accessed 23 February 2019].

APPENDIX A

Survey questions



General Information

1. Age *

◯ 21-25 years

◯ 26-30 years

◯ 31-35 years

◯ 36 years and above

2. Gender *

◯ Male

◯ Female

3. My job role in the company is? *

◯ Architect

◯ Software Engineer

◯ Quality Engineer

◯ Business Analyst

◯ Project Manager

◯ Other

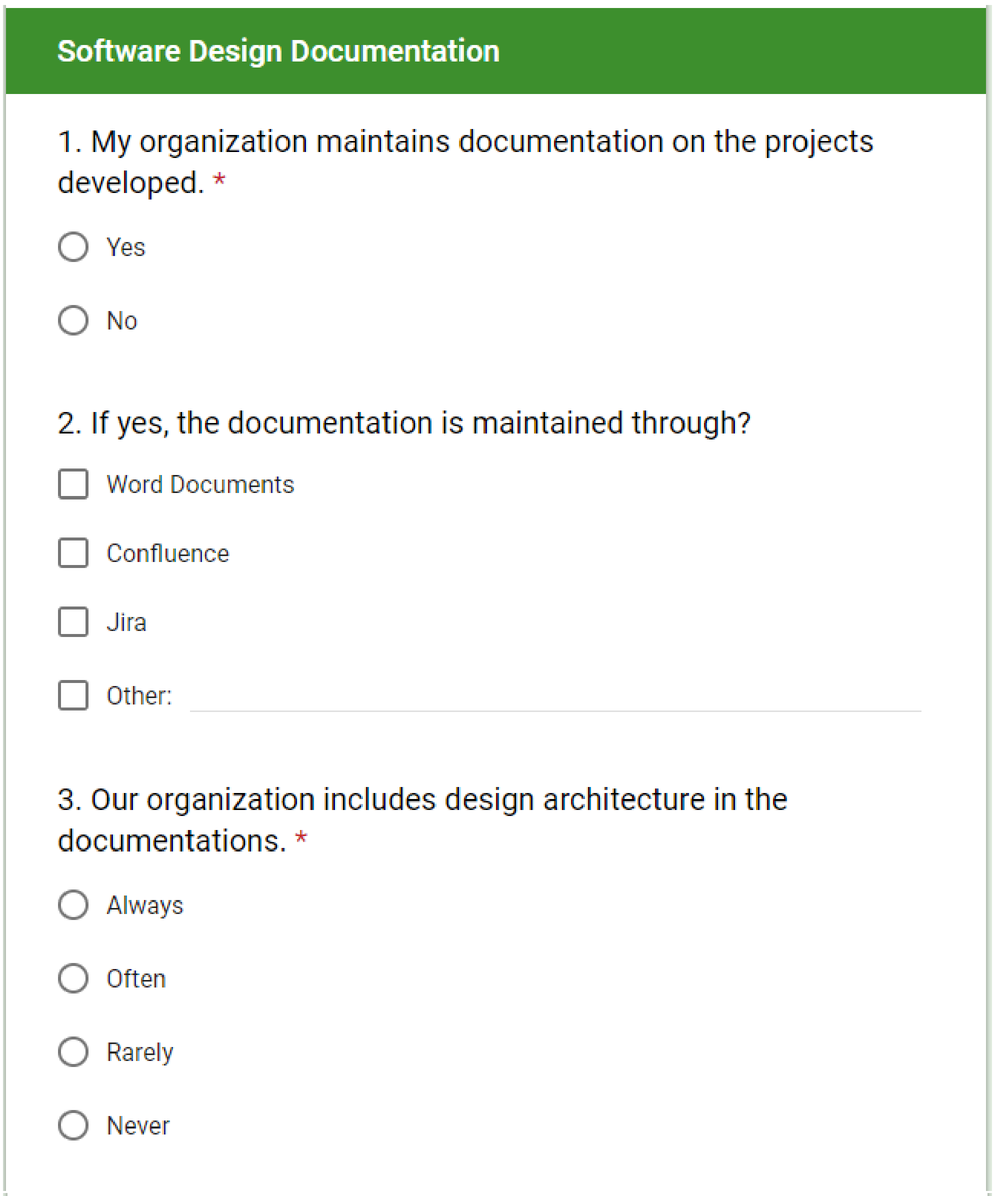
4. My years of experience in the IT industry is? *

◯ Below 2 years

◯ 2-5 years

◯ 6-10 years

◯ More than 10 years



Software Design Documentation

1. My organization maintains documentation on the projects developed. *

○ Yes

○ No

2. If yes, the documentation is maintained through?

☐ Word Documents

☐ Confluence

☐ Jira

☐ Other: ____________________

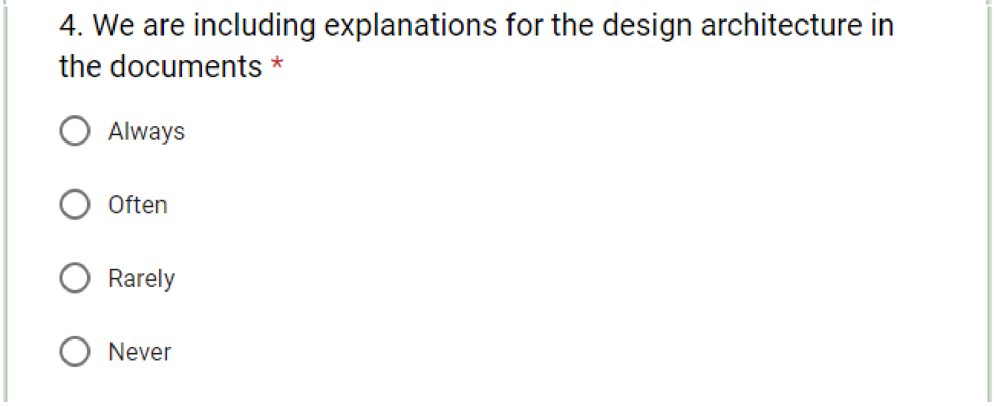
3. Our organization includes design architecture in the documentations. *

○ Always

○ Often

○ Rarely

○ Never



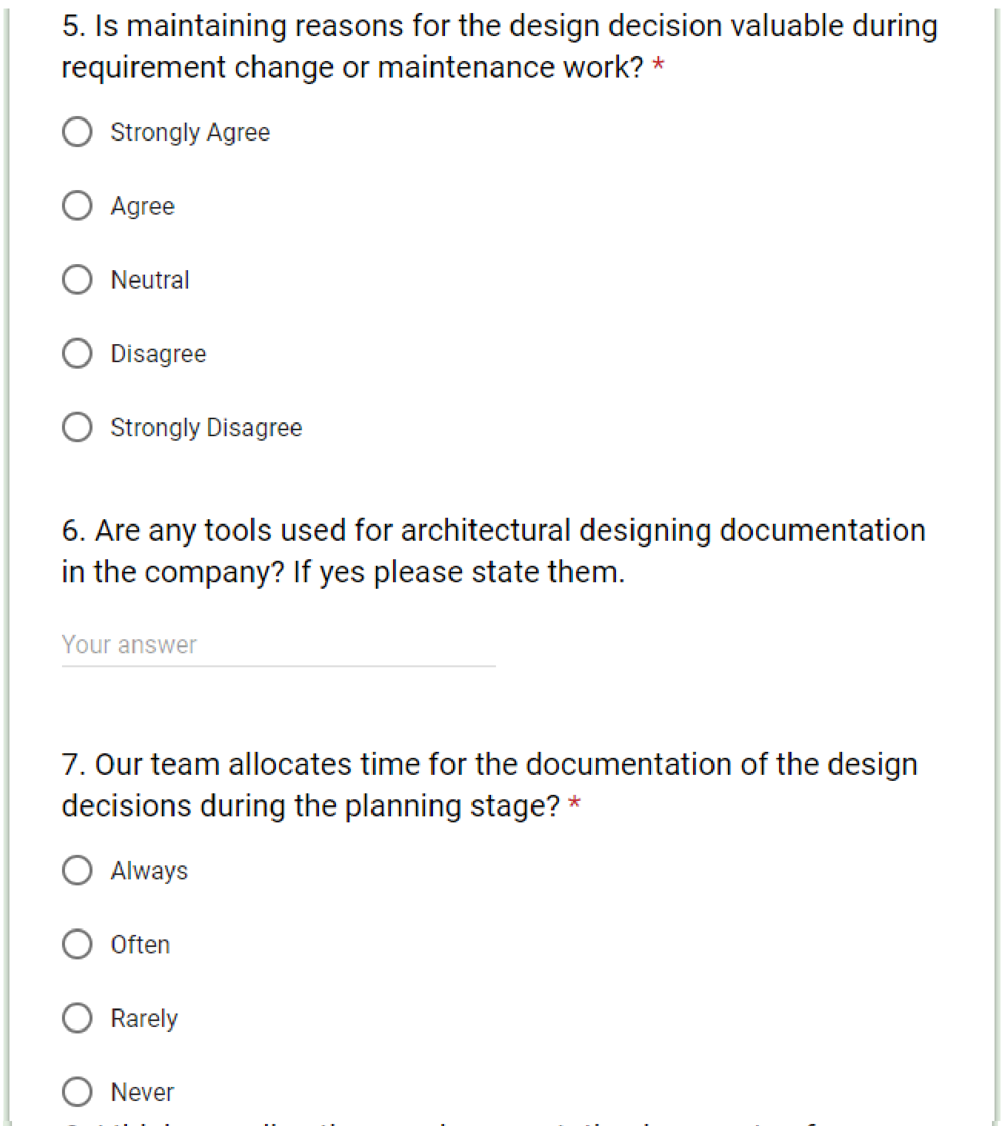
4. We are including explanations for the design architecture in the documents *

○ Always

○ Often

○ Rarely

○ Never



5. Is maintaining reasons for the design decision valuable during requirement change or maintenance work? *

○ Strongly Agree

○ Agree

○ Neutral

○ Disagree

○ Strongly Disagree

6. Are any tools used for architectural designing documentation in the company? If yes please state them.

Your answer

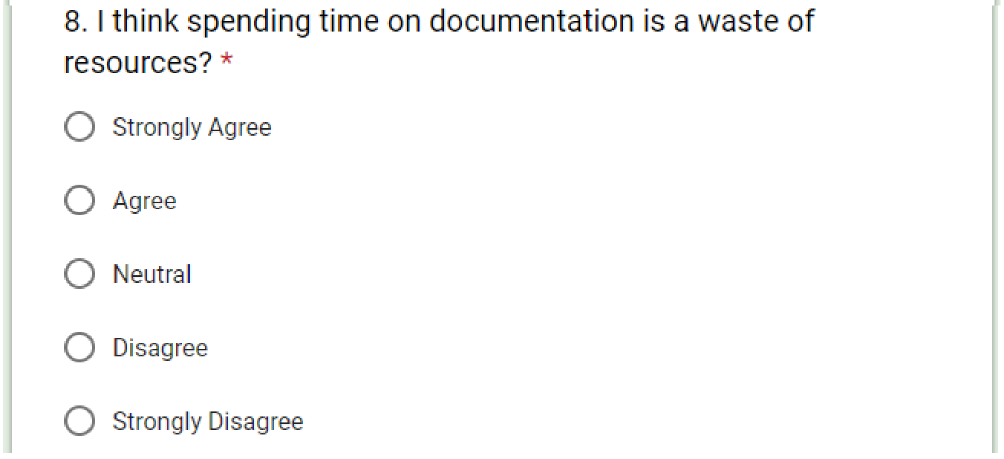
7. Our team allocates time for the documentation of the design decisions during the planning stage? *

○ Always

○ Often

○ Rarely

○ Never



8. I think spending time on documentation is a waste of resources? *

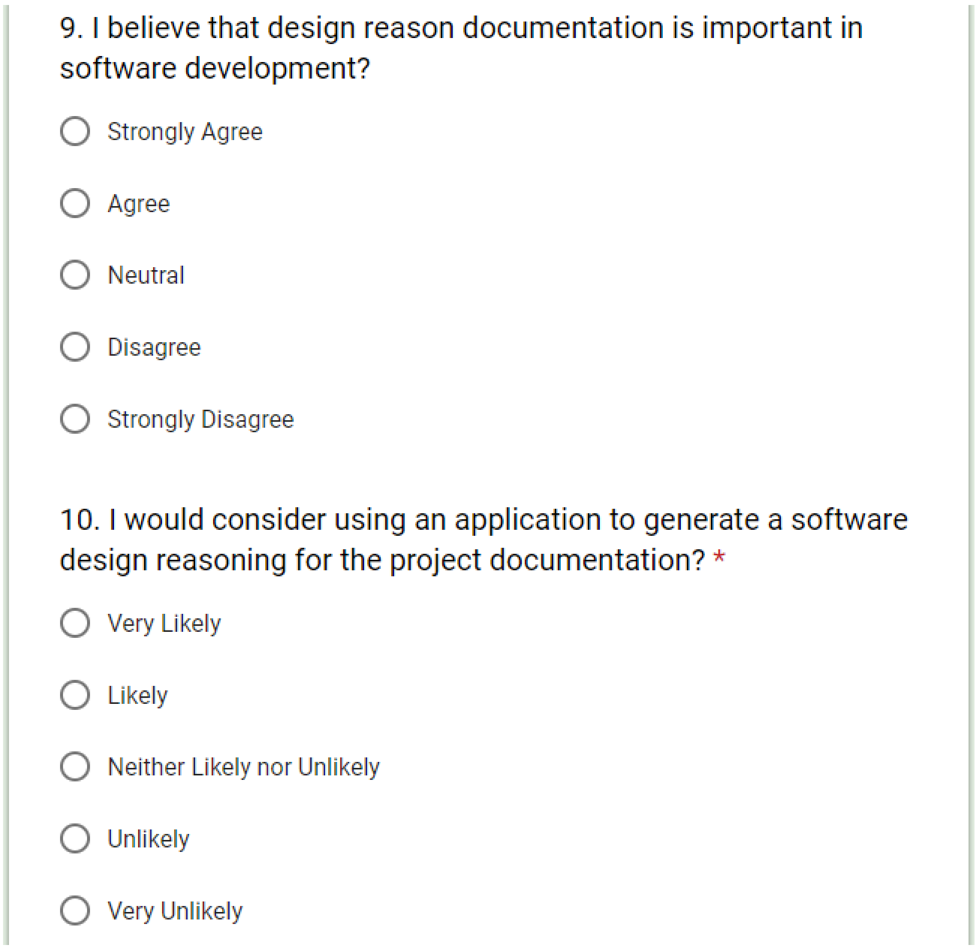
○ Strongly Agree

○ Agree

○ Neutral

○ Disagree

○ Strongly Disagree



9. I believe that design reason documentation is important in software development?

○ Strongly Agree

○ Agree

○ Neutral

○ Disagree

○ Strongly Disagree

10. I would consider using an application to generate a software design reasoning for the project documentation? *

○ Very Likely

○ Likely

○ Neither Likely nor Unlikely

○ Unlikely

○ Very Unlikely