Leiden University

ICT in Business and the Public Sector

An Interactive Chatbot For Software Requirements Elicitation

Name:	Niharika Bhandari
Student number:	s3068293
Date:	January 27th , 2023
1st supervisor:	dr. G.J. Ramackers
2nd supervisor:	prof dr. SA Raaijmakers

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1 — 2333 CA Leiden — The Netherlands

Abstract

Software development involves eliciting requirements in the early stages of the development cycle. Business analysts typically conduct interviews with end-users, and transcribe these into formal UML requirements models. This results in a complex and time-consuming manual process. Our research leverages NLP based chatbot technology to enable users to express their requirements directly, thereby bootstrapping the formal modelling process in the early stages. The chatbot aims to streamline elicitation for users not adept in the requirements process. It captures the different requirements components of an Agile SDLC process, specifically Use Cases, Classes, and Activities. In particular, Semantic Role Labeling is applied to extract relevant meta data from a knowledge graph-based conversation process. The chatbot uses a hybrid approach with a mixture of direct input processing using a conversation strategy component, and randomises variants of questions to the user in order to avoid repetitive, unnatural interaction. The thesis has been conducted through research by design. Based on an extensive literature study, an end-to-end prototype has been developed as part of the LIACS ngUML / Prose to Prototype development tool project. This enables the chatbot to generate UML diagrams which can be further refined in a 'human-in-the-loop' fashion by analysts, in conjunction with the user.

Acknowledgements

I would like to express my profound gratitude to my primary supervisor, Dr.Guus Ramackers , for all his guidance during this project. In conjunction with our weekly meetings, I developed a comprehensive and objective assessment of the research. I would like to thank my second supervisor, Prof.dr. Stephan Raaijmakers, for guiding me to important publications that aligned my thesis with current research and helped me shape the overall project.

Additionally, I would like to thank my colleagues from ngUML project, Willem-Pieter van Vlokhoven, Anna Roussou, Pepijn Griffioen, and Max Boone, for their patience and support during the implementation integration debugging process.

Finally, I would like to express my sincere gratitude to my parents and to my partner, Aditya, for their unfailing support and encouragement throughout my studies.

Contents

List of Figures

1 Introduction

Requirements elicitation is an essential part of the software development life cycle [Cha+12]. Ease and correctness of elicited requirements plays a major role in the development timelines of the omnipresent software products around us. There are a lot of implementations in the literature for requirements elicitation during different stages of software development life cycle using focussed techniques [SRV22] [Ram+21]. This work focuses on the development of a use case driven hybrid chatbot for the purpose of requirements elicitation during the early stages of the software development life cycle.

1.1 Problem Statement

Requirements elicitation is a complex process and is quite often error-prone for users not very accustomed to it. The requirements elicitation process, however, is a vital component of the software development life cycle (SDLC). There is a need to streamline the requirements elicitation process to make it easy to use for an average user in the early stages of SDLC. Additionally, agile processes are iterative and increasingly use case driven. The requirements elicitation flow needs to be use case driven but also capture all the other stages of elicitation flow.

Current implementations of automated requirements elicitation rely heavily on Natural language processing (NLP). The efficacy of these techniques depends heavily on the input data structure and are heavily case and extraction dependent. Eliciting requirements for use cases, classes and activities iteratively is a complex process. Proper structure needs to be enforced to capture and process the incoming data. Additionally, capturing all the requirements as a single file or user story creates a complex processing scenario. It is very difficult for any NLP algorithm to process it correctly and capture complete requirements data.

Even if the requirements elicitation process is broken down into stages, the overall extraction procedure and mapping it to a knowledge base which can correctly predict the next requirement to be asked is also a complex process. In-order to correctly elicit requirements, firstly, the correct extraction algorithm needs to be run. Then the metadata from the extraction run needs to be mapped correctly to a knowledge base. The structure for the knowledge base for best automation and extendability is also a non-trivial problem.

Additionally, if the requirements are elicited in a staged manner, the user might want to switch between providing different kinds of requirements. The strategy to determine the switching of the requirements elicitation which map to different sections of the knowledge graph is also a complex problem. During the whole flow, the monotonicity of the elicitation process needs to be avoided to make the procedure more engaging for the user as well.

1.2 Research Objective

The main objective of this research is the development of a methodology which can streamline and ease the early stage requirements elicitation in the software development life cycle (SDLC). The process of streamlining and ease of use is of absolute necessity since typical users involved in requirements elicitation do not have much expertise in the area. The derived research question is :

1. How can early stage requirements elicitation in SDLC be streamlined and improved using a chatbot?

The requirements elicitation in agile processes is typically use case driven and iterative. The chatbot needs to be able to have a use case component as the driving component for the requirements elicitation. At the same time, the class and activity related requirements elicitation also needs to be done. The chatbot should also not have a static and monotonous feeling when posing the questions for requirements elicitation. All these issues are tackled using the combination of knowledge graph and strategy component to create a hybrid chatbot. The derived research question is :

2. How can combination of strategy and knowledge graph be done to create a use case driven, dynamic and hybrid chatbot?

To answer the research questions posed as part of the thesis, the following research objectives have been set.

Development of a prototype chatbot with the following features:

1. **R-01**: The chatbot should be able to streamline the requirement elicitation process with different components of use case, classes and activity.

2. **R-02**: The chatbot should be use case driven and should be able to link to different components using a strategy component.

3. **R-03**: The chatbot should be able to do complex input metadata extraction and map it to knowledge base to correctly predict the next question.

4. **R-04**: The chatbot needs to link the strategy component to the knowledge base to create a hybrid chatbot.

5. **R-05**: The chatbot needs to be extendable for future research and complexity increment.

6. **R-06**: The chatbot needs to be dynamic in the questions posed to the user and not repeat same question always.

7. **R-07**: The chatbot needs to perform end to end requirements elicitation. The elicitation process starts from capturing the user inputs in a structured format using the chatbot. It then performs the necessary processing and extraction. At the end of the elicitation cycle, the user is presented with the metadata/UML diagrams.

1.3 Research Methodology

The research method applied in this thesis is research by design. In this research method, knowledge is produced in a domain by deriving it from the design and development. This research and development typically involves a new solution to a research problem. The approach does not focus on review and evaluation of existing solutions. This is a common approach implemented in Information Systems. According to Peffers et al. [5] design science "includes six steps: problem identification and motivation, objectives for a solution, design and development, demonstration, evaluation, and communication."

Problem identification and motivation: The research problem of streamlining the requirements elicitation during early stages of SDLC and making it more use case driven using the strategy component have been described in the first chapter. The background work and motivation to work in this specific direction have been done in the second chapter

Objectives of a solution: The objectives of this thesis have all been inferred from the derived research questions. The research questions have been derived from the problem statement. A solution has been proposed, and a prototype development has been done as part of this thesis. The comparison and novelty of the prototype as compared to the existing work in the literature has been done in the second chapter. Its value proposition for academics and future extendability for research and development has also been kept in focus as part of the prototype development.

Design and development: The third chapter describes the system and architectural level designs and development of the proposed chatbot. The system and architectural choices taken in the design and development have been related to the research objectives of this thesis.

Demonstration: The fourth chapter details the implementation of the chatbot in pseudocode. The chapter describes how research objectives have been achieved using different parts of the proof of concept demonstrator chatbot.

Evaluation: Chapter five focuses on case studies for different systems requirements elicitation performed. Specific focus has been kept on how the research objectives have been met with the proof of concept demonstrator.

Communication: Chapter six justifies and communicates the novelty of the proposed solution. It also dives into possible limitations and how they can be handled by the extendability of the chatbot already built into the chatbot proof of concept. The goal is to motivate the researchers and industry to further improve and develop this proof of concept research done by design.

1.4 Academic Contribution

This research facilitates the requirements elicitation in the early stages of the software development life cycle. It performs complex metadata extraction and streamlines the process using a hybrid chatbot implementation. The approach is extendable in various directions, and the extendability is built into the implementation. The complex metadata extraction flow and its mapping into the chatbot has been done for the first time in the academic literature. Additionally, a hybrid chatbot implementation with a complex strategy component and a knowledge graph for the purpose of requirements elicitation has also been done for the first time in the academic work. The work has been demonstrated for complex systems in an end two end integrated pipelines from user input to requirements diagrams.

1.5 Outline

Chapter 1 of this thesis provide an introduction to the thesis. It details at the problem statement of the thesis. Then it creates research questions and objective based on the problem statement. It provides the research methodology of the thesis and justifies the academic contribution. Chapter 2 reviews the background work done in the literature for the problem statement. It draws inspiration from the work literature to create a methodology to approach the research objectives. Chapter 3 focuses on system design of the overall end to end prototype of the chatbot. It proposes a system architecture using the methodology shortlisted in chapter 2 to solve the research objectives created in 1. Chapter 4 details the implementation for the prototype done for the proposed system architecture in 3. It hierarchically details each function implemented for the proposed architecture to achieve the research objectives. Chapter 5 details how different complex system use cases were validated using the developed chatbot in an end to end pipeline.

2 Background

2.1 Requirements Elicitation in Software Development Life Cycle

Requirements Engineering (RE) has become an important pivoting point for the software industry for more than a decade, [Hor+19][Lam00]. Requirements elicitation is one of the most important and key phases of the Software Development Life Cycle (SDLC) [Cha+12]. Correct management of a project requirements help in achieving the right solution within the time and budget the companies have been allocated [LB99]. Additionally, the validation of the elicited requirements is an important step to ensure the quality of requirement documents [Nae+13]. Therefore, elicitation of requirements require proper attention from requirement engineers and other related stakeholders [MMY08].

To this end, there are various techniques that have been used to elicit information from people in the literature. These techniques depend on the specific project or situation, [Fer+17][KZ09]. Some examples of the techniques used for the elicitation process are:

- **Interviews**: It is an open-ended or structured method to understand customer's expectations from a software. The stakeholder providing the requirements and the person conducting the interview both need to have a certain amount of expertise for this methodology to work well.
- **Brainstorming**: It is a group technique used when a lot of new ideas for development need to be generated. This also requires a highly trained facilitator to streamline the elicitation process.
- **Facilitated Application Specific Technique** : It is used as a bridging technique between developers to customer's expectation.
- **Quality Function Deployment** : This technique aims to emphasize and prioritize customer requirements.[LNZ01]
- **Use Case Approach** : The approach uses both textual and pictorial representations to provide a better understanding of the requirements. These may include major things like actor, use cases and use case diagrams.[RKW95]

Requirements elicitation is the method of learning, obtaining and compiling system requirements using some method of user engagement in the process [Zha07][PGR18][Dav+06]. There are various crucial components associated with it.

1. The process of providing is not always executed by a trained requirements engineer. The process is often run by personnel with only basic understanding of the requirements process. The fundamental knowledge of how a requirement might relate to any process is also often lacking.

2. The right question needs to be often posed to the person providing the requirements. Responses and therefore the elicited requirements have a heavy dependence on the questions posed for the requirements' elicitation.

3. Requirement elicitation is often done in the preliminary stages of the SDLC. This results in a lot of requirements being incomplete and imprecise. These incomplete and imprecise requirements get converted into subsequent processes incorrectly. Since the elicitation is done in the preliminary stages and any error in beginning can result in big delays in the overall SDLC.

4. Elicited requirements need to have certain amount of correctness for the SDLC cycle to carry forward without any hassles. Therefore, the elicited requirements need to be validated against some standards.

In-order to alleviate the above issues and streamline the process of requirements elicitation, in this work we have proposed the usage of a chatbot. The chatbot is expected to gather and process the requirements in a streamlined manner. The chatbot is expected to be simple, yet dynamic enough that a basic user can create requirements using it.

2.1.1 Agile Use Case Driven Approach

A lot of SDLC are increasingly transforming into agile approaches where use case requirements are of the highest importance [Rup10]. The steps are executed in short sprints iteratively, and small requirements need to be gathered at the start of the process and validated at the end. This process is done iteratively. Multiple implementations of agile use case driven requirements' elicitation can be found in the literature. [Hor+19] proposes the usage of requirements analysis methods underlying GORE (Goal Oriented Requirement Engineering) concepts. As part of the methodology, soft goals can be identified quickly at an early stage of software development using the concept of agility. It ensures active participation of stakeholders in identifying soft goals. [Fra+21] propose extension of agile requirements' elicitation by applying situational method engineering. There is a fundamental lack of work to make the agile use case driven elicitation process simpler and streamlined for the user.

In this work, we are proposing the use of the chatbot to streamline the process of requirements elicitation. The implementation focuses on a use case driven approach and manages the flow of the elicitation process using the strategy component of the implementation. Use cases are first elicited and then, depending on the user's responses to the chatbot, the chatbot dynamically asks for more use cases or classes or activities.

2.1.2 Chatbot

The work profile of a business analyst involves tasks like interviews, observations, document analysis and interface analysis [Kle+12]. These tasks are mundane and repetitive. There is a huge scope of automation in these tasks. Chatbots have been used quite widely for the automation of such tasks. High utilization has been observed in the retail industry [Cas+20] [AAG22][ZH18]. The combination of streamlined process and machine learning on existing datasets have been used to automate a lot of mundane processes [GHV20]. Using the chatbots, the error component created by having a human in the loop gets reduced, and the knowledge set of the human in the loop can be a lot diverse and less focussed on the specific task being handled by the user. This also enhances the user experience, since the streamlined process leads to fewer errors and unified flow. There have been multiple attempts to utilize the chatbots in other industries like the human resource work. Chatbots have streamlined the new employee onboarding process. There have also been attempts to automate the development of chatbots [Pér+21] due to their widespread utilization in business processes. They utilize multi component approach with basic and strategy components to automate the development of the chatbots.

There have been different attempts to create different architectures of chatbots[Abd+22][AAB20]. There are kinds of chatbots that a specific for a certain function[NL20]:

- **Menu/button-based chatbots** : These are based on static decision tree hierarchies presented to the user as a menu button.
- **Linguistic Based (Rule-Based Chatbots)** : They create a conversational automation using a series of if/then logic. Language conditions like order of words, synonyms etc. need to be defined for the chatbot.
- **Keyword recognition-based chatbots** : They are based on keyword recognition using Natural Language Processing (NLP) techniques. Complex responses from the user can be interpreted using NLP and an appropriate action can be executed by the chatbot.
- **Machine learning chatbots** : These chatbots evolve over time based on user-types, responses, processes etc.
- **Hybrid models** : These are typically used by businesses are they offer the best of both Rule and NLP based chatbots using hybrid implementations.
- **Voice bots** : These chatbots can respond and process speech along with other NLP post-processing features of the ML or hybrid chatbots.

Various machine learning models for chatbot have been proposed in the literature. [BEF21] and [Csa19] propose the use of transformer based approach. [JRG20] proposes the use of end to end memory network based approach. Taking inspiration from the [Tha+21] the chatbot can be implemented in the SDLC

cycle in exceedingly initial stages of product development. This iterative approach of gathering requirements and posing the next right question to the user can help companies to reduce repetitive work. Hybrid chatbot implementation based on NLP can be used to provide both dynamic and streamlined requirements' elicitation flow.

2.2 Natural Language Processing

The science of using artificial intelligence for the purpose of understanding text or spoken words is referred to as Natural language processing (NLP) [Lid01]. Computational linguistics-rule-based modeling of human language are combined with statistical, machine learning and deep learning models in this science. A combination of these technologies can process human language and make decisions based on those. It is used in computer programs across business world, research wherever automation is needed based on a human textual or audio input.

NLP can be used in various tasks like:

- **Speech recognition**: This involves methodologies and technologies that enable the recognition and translation of spoken language. This spoken language is translated into text by computers. It provides the main benefit of searchability.[Red76]
- **Part of speech tagging**: This process does the marking up a word in a text. This is done corresponding to a particular part of speech. Additionally, it is based on both its definition and its context.[BM]
- **Word sense disambiguation**: It is the process of identifying the sense of a word. It is meant to distinguish the sense in a sentence or in a segment. It is also meant to distinguish out of context usage.[Nav09]
- **Named entity recognition**: It seeks to locate and classify named entities mentioned in unstructured text. It classifies them into pre-defined categories such as person names, organizations, locations, quantities, monetary values, percentages, medical codes, time expressions etc. [Moh14]
- **Co-reference resolution**: This task of finds all expressions that refer to the same entities in a text.[DW15]
- **Sentiment analysis**: It is the use of natural language processing, text analysis, computational linguistics, and biometrics. It aims to systematically identify, extract, quantify, and study affective states and subjective information in the text. [MHK14]
- **Language generation**: It is a software process driven by artificial intelligence. It produces natural written or spoken language. This is derived from structured and unstructured data.[GK18]

For the purpose of creating a hybrid chatbot, combination of the techniques of NLP need to be deployed. For the purpose of this thesis, a pre-existing library set needs to be used to speed up the deployment of the proof of concept.

2.2.1 Allen NLP

For the purpose of automation and speed of deployment of NLP based tools, a combination of multiple library sets need to be used. AllenNLP provides a single combination of multiple NLP based tools which provide a precise definition of the approach, easy reproducibility of the results, and a basis for expanding the study. It is a widely used tool across the research community for the purpose of deploying NLP tools in their research projects [Gar+18]. The AllenNLP tool set contains a Semantic Role Labeling library with pretrained models. These models are a good fit for the hybrid chatbot.

2.2.2 Semantic role labeling

Semantic role labeling (SRL) labels parts of speech in a sentence. The goal is to understand what they represent. This method can be particularly helpful in determining how speakers of a certain language refer to subjects. It also helps identify other primary elements in a sentence. [CLS08] [He+17] It helps machines to understand the roles of words within sentences. Natural Language Processing programs benefit from this, as NLP needs to understand not just the words of languages, but how they can be used in varying sentences.

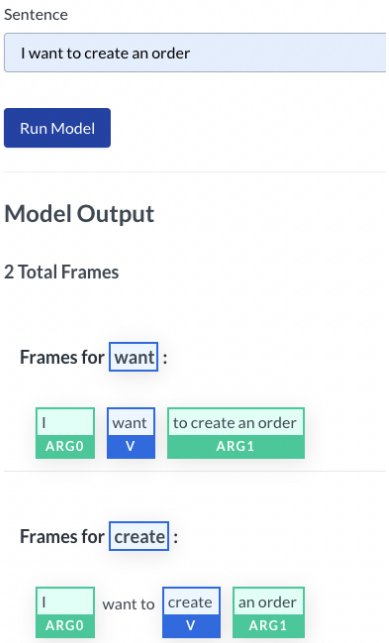

Figure 1: Example of Semantic Role Labeling

Figure 1 shows a snapshot of a typical usage of SRL. The algorithm creates a frame for each verb in the sentence. For each verb, the algorithm creates a list of arguments 0 and 1. The representation created by the algorithm is can be used for mapping the next question in the knowledge graph. Other techniques can also be used as an extension of the chatbot. For this thesis, SRL was chosen as the correct candidate.

2.3 Knowledge Graphs

Knowledge graphs are collections of interlinked descriptions of concepts, entities, relationships and events. [Hog+21] They are used in various fields, ranging from computational methodologies like artificial intelligence to information and data-management systems [Umu+20].

There are many possibilities for the design and complexity of a knowledge graph. Fundamentally, the graph comprises edges and nodes, as shown in figure 2. In the context of chatbot, the knowledge graph plays a crucial role in creating the dynamics of the questions posed to the user. [AJ20]. There can be highly complex and self learning knowledge graph implementations, but for the purpose of this thesis, only pre-compiled knowledge graphs have been used.

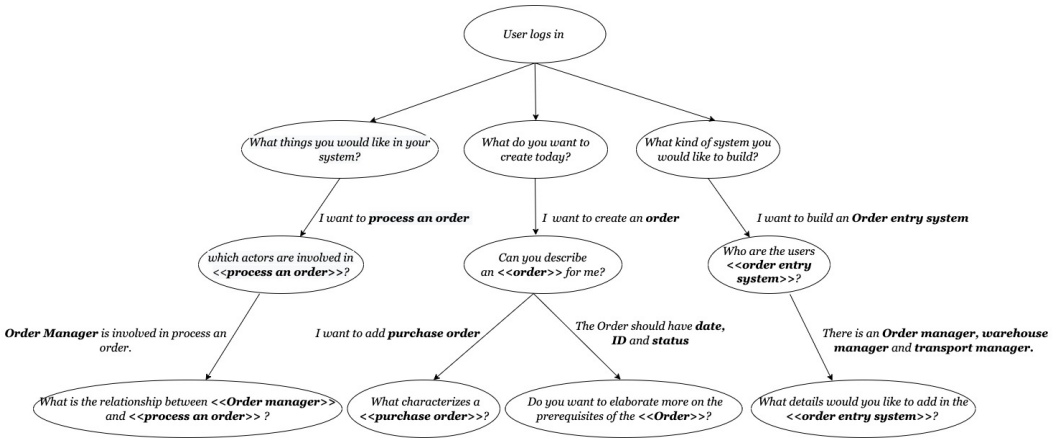

Figure 2: Example of Knowledge Graph

2.4 UML

In the field of software engineeering, UML is used as a general-purpose, developmental modeling language [Sel06] [Bai+05]. It provides a standard way to visualize the design of a system. Figure 3 shoes an example of use case UML diagram. Typical uses in software requirements elicitation are use case, class and activity visualization. In the context of this thesis, the diagrams are used as a method to visualize the metadata generated from the extraction of requirements by the chatbot. Other methods in the literature to visualize the software requirements as a complement to UML diagrams include C4 model [VGG20]. The generated metadata in this thesis is visualized using UML diagrams, but the approach can be translated to a C4 model as well.

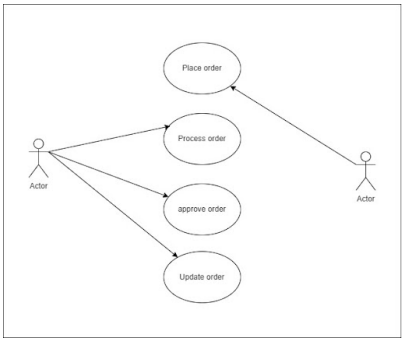

Figure 3: Example of UML Diagram for Use Case

3 System design

The design of the chatbot for requirements elicitation has been done as part of a complete end to end pipeline. This chapter explains design from an overall system design and integration perspective. Various design and integration choices have been explained in the subsequent sections.

3.1 High Level Architecture

Figure 4 shows the high level architecture of the implemented chatbot. From the user perspective, the user can provide a response on the frontend interface. The response in principle can be both text or audio. For the proof of concept implementation in this thesis, the response has been limited to a textual value.

The response received from the user passes through the interface chatbot component. This chatbot component has been kept focussed on the early stage requirements. There have been other works which focuses on modifying the already elicited requirements. The high level architecture shares an implementation and similarities with the chatbot, but the system level implementation has been implemented as a completely separate pipeline in both the frontend and backend.

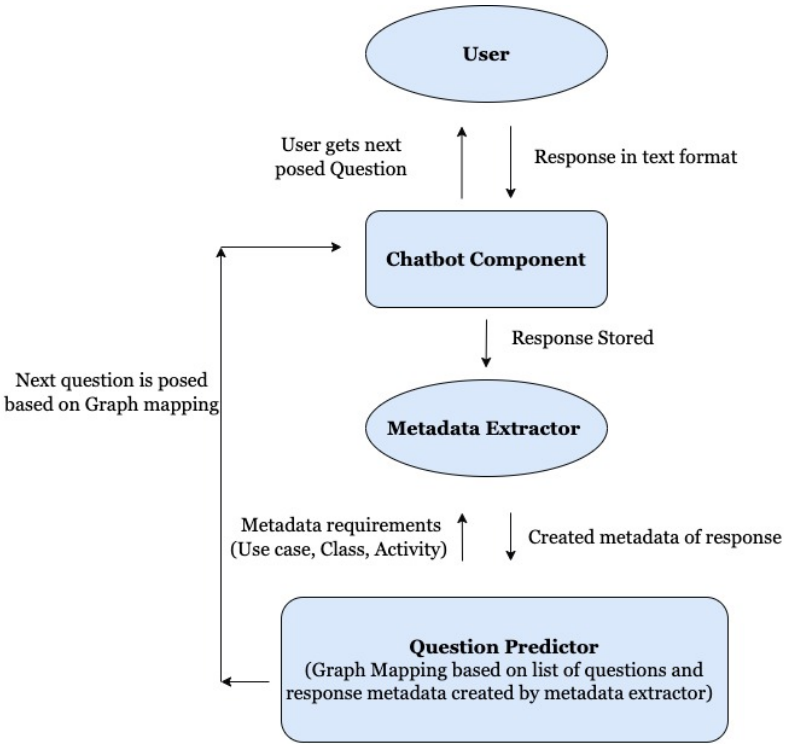

Figure 4: High Level Architecture

Fundamentally, the received response is in a structured format. The reason behind it is that the questions are targeted and streamlined to specific stages of the elicitation, as explained in the section 3.2. The response passes through an extraction algorithm. The extraction algorithm for the purpose of this thesis has been kept as semantic role labeling, as explained in section 2.2.2. The extracted metadata is passed through a process of knowledge graph matching and prediction of the next question. Once the next question is determined, the next question to be posed to the user is sent to the frontend. The metadata requirement for the extraction purposes to be extracted in the next question is sent to the metadata extractor.

3.2 UCA Architecture

The architecture to streamline the development requirements elicitation has been kept as a Use Case - Activity - Class (UCA) architecture as shown in the figure 5.

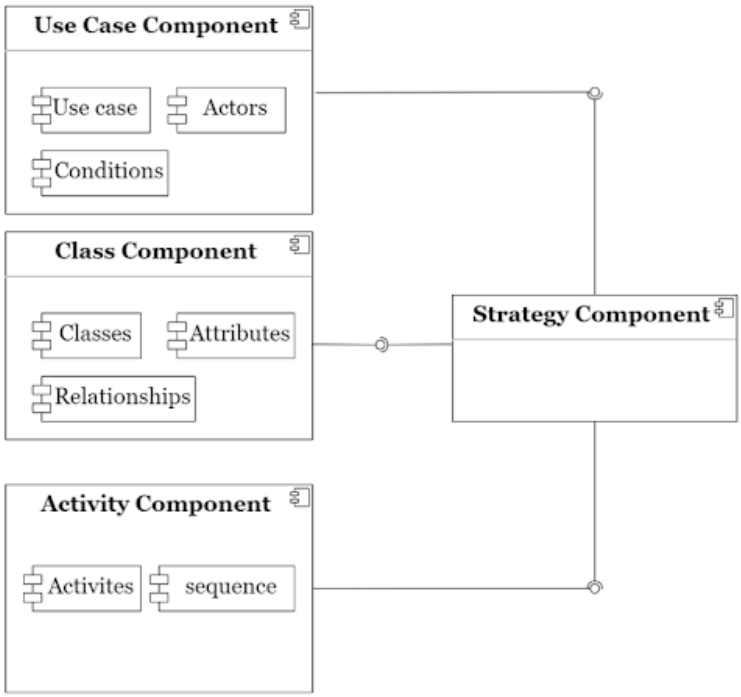

Figure 5: Component Architecture

The architecture has different components for use case, class and activity requirements elicitation. In each of the components, there are different properties of the component that can be elicited from the user input. The list of

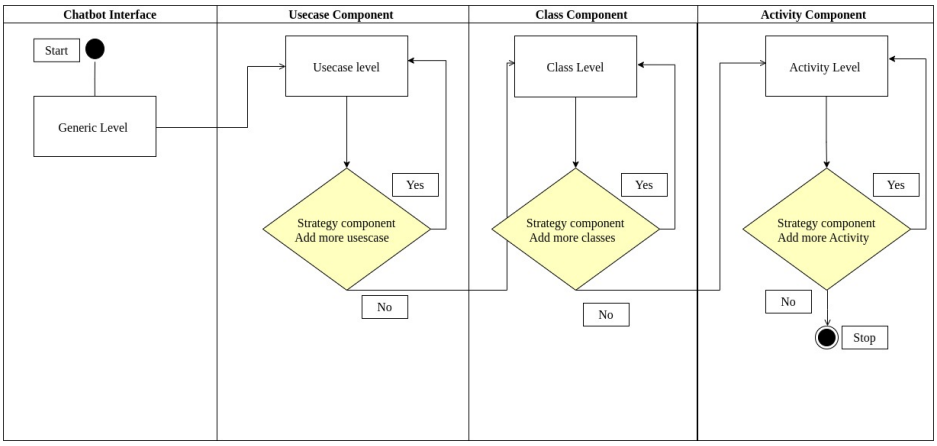

Figure 6: Interaction between UCA components and the strategy Component

properties has been kept limited for the purposes of proof of concept. They can be further expanded. The overall architecture does not limit it and is expandable. The strategy component controls how the process of elicitation is streamlined. The chatbot implementation is use case driven have by default the use case requirements are elicited first, then the next set of class and activity requirements are elicited. The interaction between the UCA components and the strategy components is shown in figure 6.

3.3 System Components

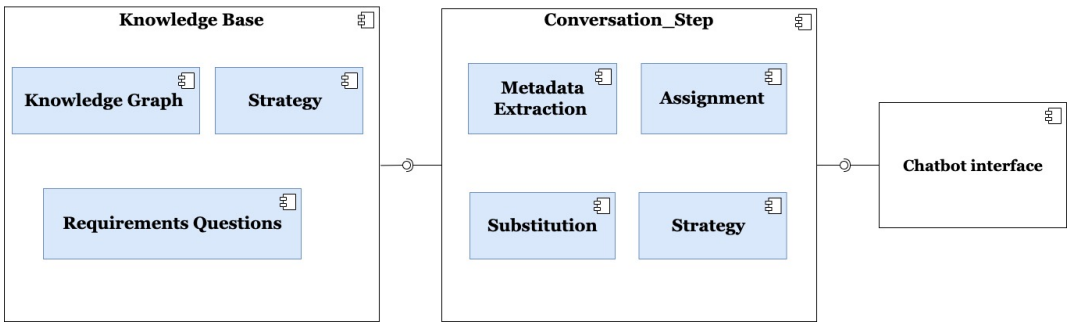

Figure 7: System Components

The actual implementation of the system for the proposed architecture in 3.2 is divided into multiple components, as shown in figure 7. The core of any system specific chatbot implementation is its knowledge base component. The knowledge base consists of the requirements questions and requirements knowledge graph. The knowledge base has both the multiple components of

use case, class, activity and strategy embedded within both the questions and the graph. The chatbot main component acts an interface to the frontend, where the input is provided by the user. The chatbot main interface calls the conversation step interface. The extraction of metadata, mapping to the next question using the knowledge base and sending the next question to the user through chatbot main interface are executed by the conversation step.

3.3.1 Requirements Questions

The first part of the knowledge base consists of the requirements questions. The questions are put in the knowledge base as a JSON file. The file is parsed by the chatbot each time a new requirements elicitation request is initialized from the frontend. The questions form the base of the requirements elicitation flow. There need to be a new set of questions formulated for a chatbot deployment for each new system. The questions are created based on following set of rules:

1. The questions for the format : **"Component_type_Q_level_count" : "Question language <<substitution_tag>>?"**
2. The key for the JSON file is the questions ID given by : **"Component_type_Q_level_count"**.
3. **Component_type** can be either of the following:
   (a) **g**: for generic questions
   (b) **u**: for use case component
   (c) **c**: for class component
   (d) **a**: for activity component
   (e) **s**: for strategy component
4. **level** is the depth of questions graph in the same component (count start from 0). This is defined differently for strategy component. It is detailed in section 3.3.3.
5. **count** is the count of questions at the same level in the graph, these will be used and explained in section 3.4.2 (count starts from 0). This is defined differently for strategy component. It is detailed in section 3.3.3.
6. **<<substitution_tag>>**: is the already extracted metadata from the previous questions which has to be substituted in this question before posing it to the user.

By using the ruleset described above, automation can be done for new questions generation for a new system. This automation was not done as part of this thesis. An example of a question can be:
"u_Q_1_1" : "What would you like to have in your <<system_name>>?"
The explanation for the example is :

- **u_Q_1_1**
  - Use case component
  - Second level of the questions in the graph
  - Second question in the same level

3.3.2 Knowledge Graph

The questions described in the section 3.3.1 are interconnected in the form of a knowledge graph. The knowledge graph is used in the chatbot to predict the next question to be posed to the user using the extracted information from the questions already answered and the knowledge graph itself. The knowledge graph needs to be formulated for each chatbot deployment for a new system. The knowledge graph is in a JSON format and follows the following rules:

```
{"nodes":
   {
      "component_type_Q_level_count":
      {
         "extract_type" : "extract_algo",
         "metadata"  : "extract_tag"
      },
      "component_type_Q_level_count":
      {
         "extract_type" : "extract_algo",
         "metadata"  : "extract_tag"
      }
   },
   "edges":
   [
      {
         "source" : "component_type_Q_level_count",
         "target" : "component_type_Q_level_count",
         "metadata" : "extract_tag"
      }
   ]
}
```

Figure 8: Format definition of Knowledge Graph used in Chatbot

1. The format for the rules is shown in figure 8.
2. The first level of keys either points to a **dictionary of nodes** or a **list of edges**.
3. Inside the dictionary of nodes, the keys are question ids which follow the same format as the requirements questions.

4. Individual questions are modeled as nodes in the graph. Each question (modelled as a node) has its own dictionary of features. Currently, this dictionary is limited to two features.

   (a) **extract_type**: This lists the possible algorithms that can be run on the input from the user. Currently, this is limited to Semantic role labeling and is run using the keyword "semrol" for the extraction type. For the strategy questions, there is a separate tag "direct" used. This is explained in section 3.3.3.

   (b) **metadata**: This lists the possible metadata to extract from the response received from the user when this node question gets asked. The possible keywords are listed in the section 4.2

5. Inside the list of edges, each edge contains a dictionary with the following features:

   (a) **source**: This refers to the question id of the source node

   (b) **target**: This refers to the question id of destination node

   (c) **metadata**: This is the feature of the edge which limits its access to only when the "metadata" tag explained in section 4.2 is met.

The proposed format for the knowledge graph is both extendible and regular for automation for any future improvements beyond the scope of this thesis.

Figure 9 shows an example of a simple knowledge graph. The graph consists of two questions with questions ids "g_Q_0_1" and "g_Q_1_1" as nodes and an starting from "g_Q_0_1" and targeting "g_Q_1_1". Figure 10 shows the actual knowledge graph structure that gets built with the convention described in this section.

3.3.3 Conversation Strategy

The conversation strategy component is needed for controlling the dynamic behavior of the chatbot. It controls the following behaviors in the chatbot:

1. Repetition of chatbot within the same component.
2. Jumping of chatbot from one component to another.
3. Generation of Metadata.
4. Generation of UML diagrams.

To implement the above behaviors reliably, the strategy is broken down into multiple parts. The strategy questions follow the following format.

1. The questions for the format : **"s_Q_component_level_count" : "Question language"**
2. The key for the JSON file is the questions ID given by : **"s_Q_component_level_count"**.

```
{"nodes":
   {
      "g_Q_0_1":
      {
         "extract_type" : "semrol",
         "metadata"  : "semrol"
      },
      "g_Q_1_1":
      {
         "extract_type" : "semrol",
         "metadata"  : "system_name"
      }
   },
   "edges":
   [
      {
         "source" : "g_Q_0_1",
         "target" : "g_Q_1_1",
         "metadata" : "semrol"
      }
   ]
}
```


Figure 9: Example usage of the Knowledge Graph format for the chatbot

3. **Component_level** can be either of the following, it is dependent on the level after which the strategy question asked to the user:
   (a) **u**: for use case component
   (b) **c**: for class component
   (c) **a**: for activity component
4. **count** is the count of questions at the same level in the graph. In the current implementation, count 0 is hard-coded for posing the questions to view the metadata generated till now. This will trigger both diagram generation and metadata lists to be printed in the chatbot frontend. Count 1 is hard for posing the question if the user wants to repeat more requirements in the same level or wants to switch to another level in the UCA framework.

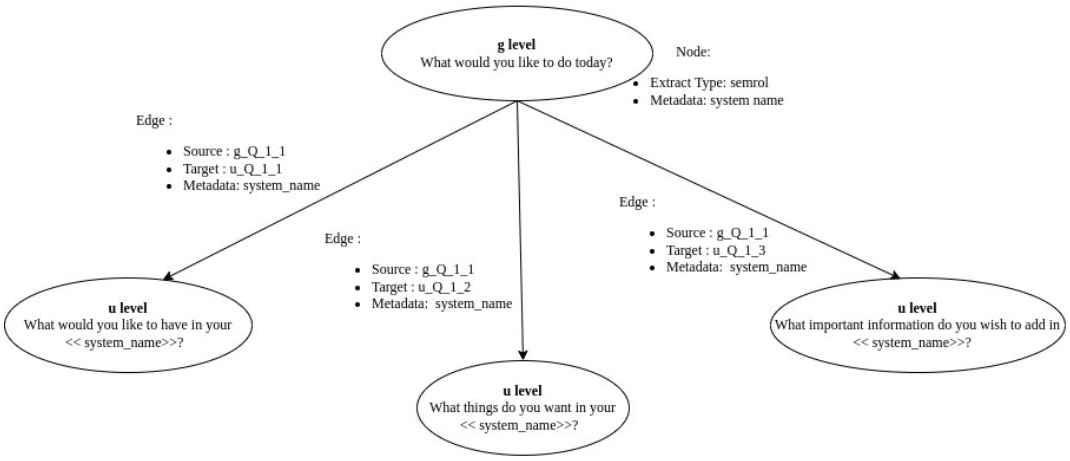

Figure 10: Detailed knowledge graph

An example of a strategy question **""s_Q_u_1" : "Would you like to add another use case?"**
The explanation for the example is :

- **s_Q_u_1**
  - It is a strategy question.
  - It is asked at the end of the use case level.
  - It is count 1 question. Currently, count 1 questions are hard-coded to ask if the user wants to repeat the previous level or jump to the next level.

The generation of metadata lists to be printed in the chatbot and the UML diagrams are currently triggered as a response to the same strategy questions at the end of each UCA level. An example of a strategy question used for generating the output metadata is :

""s_Q_u_0" : "Perhaps you would like to see the requirements so far (metadata)?"
The explanation for the example is :

- **s_Q_u_0**
  - It is a strategy question.
  - It is asked at the end of the use case level.
  - It is count 0 question. Currently, count 0 questions are hard-coded to ask if the user wants to generate the metadata of elicited requirements till now as lists and diagrams.

In the current implementation of the proof of concept, the interpretation of the responses received from the user for the strategy questions is not passed through an extraction algorithm (like SRL). Instead, it is directly interpreted. This limits the usage of responses to simple Yes/No like responses for the strategy questions. The implementation for the strategy decisions is discussed in chapter 4. The conversation strategy can be further extended to include strategy questions to resolve the conflict between the user position in the graph and where the user wants to be. This can be done with real-time visualization of the knowledge graph traversal and more control of traversal jumps through the strategy component.

3.3.4 Chat Conversation History

Chat conversation history is a log of all the nodes traversed by the user, as well as all the responses received from the user. Currently, this serves the purpose of a logging mechanism for each requirements elicitation project. This can be further extended to create a mechanism for self learning dynamic chatbot. The

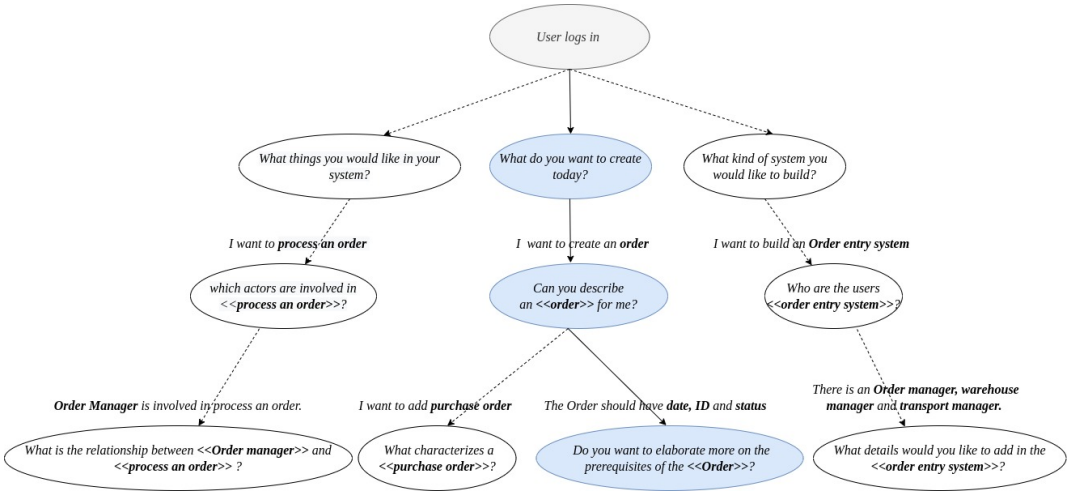

Figure 11: Example of partially traversed knowledge graph.

log is created in the form of a list of nodes traversed (chat_nodes) and a list of responses received from the user (chat_responses). As an example, figure 11 shows the partially traversed knowledge graph. The node IDs and the responses from the user for each of the questions gets logged.

3.4 Methodologies

The responses received from the user either go through the process of extraction or interpreted directly and passed through strategy_response as explained in 3.3.3. In the case where the responses go through the process of extraction, a relationship methodology needs to be setup between the questions asked, response received, extraction process and subsequent questions asked.

3.4.1 Metadata Complexity, Extraction and Mapping

For each of the questions posed in the, the responses are expected to be in a given format for the current elicitation methodology to work. This is detailed later in the 4. Assuming a given format is followed, the following steps need to be executed for correct extraction of a complex metadata.

- **Language to Semantic Role Labeling**: Each response when passed Semantic Role Labeling (SRL), generates the following:
  - List of verbs in the sentence.
  - Arguments list as done through identification and classification in [SL19].
  - Relative position of each of the Arg0, Arg1 and Verb, so that they can be extracted from the sentence.
- For each question's expected response, the information to be extracted and saved(either or all of Arg0, Arg1 and verb) needs to be known. This is coded in terms of **metadata** information associated with the node of the question.
- Based on the extracted information and **metadata** label of the question, the correct information is saved in the global variables as described in 4.2.
- **SRL to Knowledge Graph**: The extracted and saved information needs to be mapped in the knowledge graph. The mapping in the knowledge graph is done using the process of assignment to global variables and substitution to the tags using the global variable information. The implementation is detailed in chapter 4.

3.4.2 Randomization and Knowledge Graph

Each level of the knowledge graph used in the prototype chatbot has multiple counts of questions. For the current implementation, each question is a rephrased version of the questions asking the same software requirements information from the user. When the chatbot traverses the knowledge graph and reaches a given level, it randomizes the question it poses to the user if there are multiple options available. As an example:

1. **"u_Q_1_1" : "What would you like to have in your <<system_name>>?"**
2. **"u_Q_1_2" : What things do you want in your <<system_name>>?"**

Above are two questions in the level one of the use case component asking for the use case name from the user.

3.4.3 Extendable Framework

The system has been designed to be extendible from the proof of concept done in this thesis in the following ways :

- More Components can be added to the requirements elicitation flow. This is enabled by the regular expressions used to code a component. Additionally, more features can be added for the elicitation purpose in a given UCA level.
- For adding more features in requirements elicitation in a given level, the generation of questions can be automated due to the regular expression methodology used.
- Similar to requirements questions, the knowledge graph is also coded with regular expressions. This facilitates the extension using automation for new features and level or different chatbot.
- The strategy component currently is limited to a simple implementation, but is well integrated into the knowledge graph to provide a hybrid dynamic chatbot. Complex extraction and learning can be added to the strategy component standalone. This is not done as part of proof of concept design.
- Metadata extraction and mapping is semantic role labeling driven in the proof of concept. The knowledge graph encoding and interpretation scheme offers the extendible framework to add keyword for additional algorithms to improve the extraction and mapping.

4 Implementation

The High Level Architecture described in chapter 3 is achieved through a low level design methodology.

4.1 Information Flow

The sequence of events (also detailed in figure 12) in the chatbot are as following:

1. The chatbot poses a question to the user.
2. The user sends the response back to the chatbot.
3. The response is either directly interpreted or passes through an extraction process. The decision to either directly interpret or pass into extraction is based on the **extract_type** and **metadata** values associated with the question posed to the user.
4. If the response is directly interpreted, then next question is determined based on the **strategy_response**. This question is sent to the user and the sequence is repeated. Also, necessary action is taken depending on the **strategy_response**.
5. If the response passes into extraction, the extraction is done based on **extract_type** associated with the question's node in the knowledge graph.
6. After the extraction, the process of assignment and substitutions are carried out. The detailed description of these processes is described in 4.
7. For the assignment and substitution process to work, the chatbot uses tags, which are described in 4.2.
8. Once the substitution is completed, the next question is sent to the user and the process is repeated.

4.2 Globals and Tags

The information flow is managed with the use of the following:

- **Global Variables**: These are used to save the extracted metadata from the user's responses. Table 1 details and provides a description of each used variable. The global variable lists have one to one correspondence within the same level. For example, the second item in use case name and use case actor will correspond to the second use case. If the actor in the two are common, duplicates are removed during the UML diagram generation process.

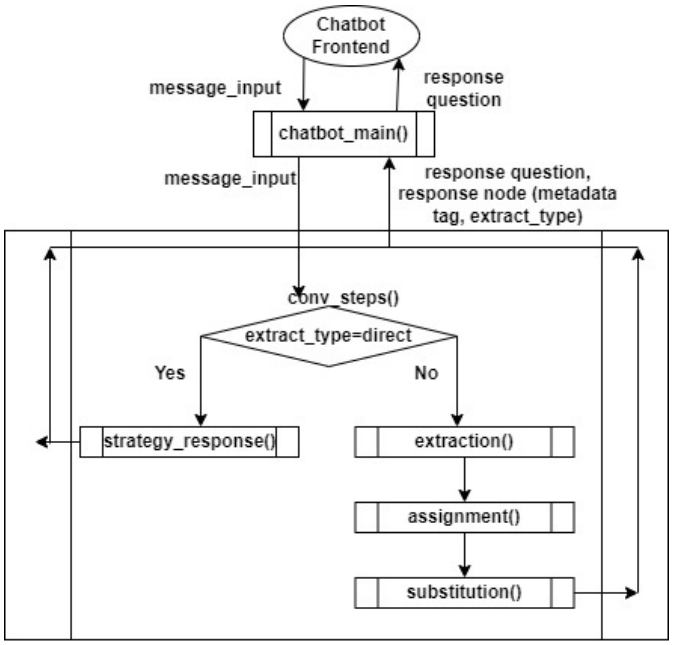

Figure 12: Information flow in the chatbot

- **Metadata Tag**: These are the tags in associated with the nodes and edges. With every node they signify what information needs to be extracted from the user's response. when associated with an edge, they signify that a particular edge is selected only if the metadata tag meets the value we expect. Table 2 details and provides the meaning of each tag for both nodes and edges.

- **Substitution Tags**: These are tags which are build into the questions stored in the knowledge base. The task of mapping is to find the correct information stored in the global variables and map them into these tags before posing the next questions to the user. Table 3 details and provides a description of each of the substitution tag.

Global Variable	Description	Example
system_name	It saves the system name of the SDLC in the generic level	[order_entry_system]
system_users	It saves the list of system users in the generic level	[Order Manager, Warehouse Manager]
use_case	It saves the list of use case names	[process an order, approve an order]
use_case_actors	It saves names of the actors involved in particular use cases.	[Warehouse Manager, Order Manager]
use_case_relationship	It saves names of relationships between use case actors and use case names. Extracted information is saved in these variables. Transformation to relationship types is done during the diagram generation process.	[interacts, interacts]
class_name	It saves extracted class names.	[order]
class_attribute	It saves the list of class attributes.	[ID,date,item]
class_related_name	It saves the class name to which the main class elicited is related	[product]
activity_name	It saves the list of activity names.	[receive an order]
activity_seq_a	It saves the first step in the activity sequence.	[places, an order]
activity_seq_b	It saves the second step in the activity sequence.	[approves, an order]
activity_seq_b	It saves the third step in the activity sequence.	[sends, details]

Table 1: Global Variables

Metadata Tags	Description
system_name	System name needs to be extracted from user response.
system_users	System users needs to be extracted from user response.
use_case	Use case name needs to be extracted from user response.
use_case_actors	Use case actors need to be extracted from user response.
use_case_relationship	Use case relationships needs to be extracted from user response.
class_name	Class name needs to be extracted from user response.
class_attribute	Class attributes need to be extracted from user response.
class_related_name	Class name related to the main class elicited to be extracted from user response.
activity_name	Activity name needs to be extracted from user response.
activity_seq_a	First activity step to be extracted from user response.
activity_seq_b	Second activity step to be extracted from user response.
activity_seq_c	Third activity step to be extracted from user response.

Table 2: Metadata Tags

Substitution Tags	Description
system_name	System Name needs to be found from the global variables and substituted.
system_users	System Users needs to be found from the global variables and substituted.
use_case	Use Case Name needs to be found from the global variables and substituted.
use_case_actors	Use Case Actors need to be found from the global variables and substituted.
use_case_relationship	Use Case relationship needs to be found from the global variables and substituted.
class_name	Class Name needs to be found from the global variables and substituted.
class_attribute	Class attributes need to be found from the global variables and substituted.
class_related_name	Class Name related to the main class needs to be found from the global variables and substituted.
activity_name	Activity Name needs to be found from the global variables and substituted.

Table 3: Substitution Tags

4.3 Code Structure

The code implementation database is part of the ngUML database. The main chatbot code is under the chatbot subdirectory. The code is broken down into

the following files:

- **nguml/chatbot/knowledge_base/requirements_Questions.json**: JSON file with all the questions for the requirements elicitation purpose. The format for the question is described in chapter 3. The detailed requirements questions are list in the appendix.
- **nguml/chatbot/knowledge_base/requirements_Kgraph.json**: JSON file with all the nodes and edges information necessary to create a knowledge graph of the requirements questions. The format for the nodes and edges is described in chapter 3. The detailed knowledge graph nodes and edges are listed in the appendix.
- **nguml/chatbot/utils/requirements.py**: Python file with all the functions necessary for the chatbot extraction process.
- **nguml/chatbot/consumers.py**: Python file to manage the link between the frontend interface of the chatbot to the backend implementation.

The code calls the following built in internal API for the purpose of the integration with the overall project and to create a proof of concept demonstrator. In the current implementation, the **internal_api_path** is set to **nguml/model/tools**

- **internal_api_path/project** : Library of internal functions to create and manage projects with ngUML.
- **internal_api_path/system** : Library of internal functions to create and manage systems with ngUML.
- **internal_api_path/usecase_model** : Library of internal functions to create and manage use case models with ngUML.
- **internal_api_path/class_model** : Library of internal functions to create and manage class models with ngUML.
- **internal_api_path/activity_model** : Library of internal functions to create and manage activity models with ngUML.

4.4 Functions and Pseudocode

The overall code is broken down into multiple functions to implement requirements elicitation chatbot. This section describes the pseudocode and functionality of those functions

4.4.1 Chatbot Main

Algorithm 1 chatbot_main() function

Require: *message_input*

if *system_state* == 0 **then**
  next_qid ← *start_qid*
  current_qid ← *start_qid*
  response ← *questions*[*current_qid*]
else if *system_state* == 1 **then**
  next_qid, *response* ← *conv_steps*(*message_input*, *current_qid*)
  node_history ← *append*(*current_qid*)
  current_qid ← *next_qid*
  chat_history ← *append*(*message_input*)
end if

The *chatbot_main*() function receives an input message from the user through the chatbot frontend connected via a web socket. The pseudo code is shown in algorithm 1. **system_state** variable controls the state of the chatbot. If the state is 0, it implies a new SDLC elicitation process has started. The starting node of the knowledge graph is set. The **node_history** variable logs the nodes traversed by the user in the knowledge graph. **chat_history** variable logs the overall messages inputs from the user for post-processing and analysis requirements.

4.4.2 Conversation Steps

The chatbot_main() function calls the conversation step function. The goal of the conv_steps() function is to execute all the steps necessary to interpret message from the user, elicitate requirements from it, and then pose the correct question to the user in response.

The pseudocode for the conv_steps() functions is shown in algorithm 2. The functions creates a decision on whether to pass the input through strategy_reponse or through the extraction cycle based on the extract_type tag associated with the node of the last posed question to the user. If the tag is **direct**, then strategy_response gets called otherwise the extraction flow gets called. The extraction flow is broken down into multiple stages called within the conversation step. First extraction using SRL is called, then the assignment process, then the next nodes are searched, finally the substitution process is called.

Algorithm 2 conv_steps() function

Require: $message\_input, requirements\_questions, knowledge_graph$

if $direct\_process == 1$ **then**
  $next\_qid, response \leftarrow strategy\_response(message\_input, current\_qid$

  $current\_qid \leftarrow next\_qid$
else if $direct\_process == 0$ **then**
  $arg0, arg1, verbs \leftarrow extraction(message\_input, current\_qid)$
  $global\_variables = assignment()$
  $next\_qid, response = substitution()$
  $current\_qid \leftarrow next\_qid$
end if

4.4.3 Strategy Response

The strategy response is currently responsible for making decisions on whether to generate the metadata output and whether to repeat a level in the UCA architecture. It gets called when the **extract_type** for the question posed to the user is **direct**. Strategy response algorithm follows a direct mapping based on user response to next question ID to be posed to the user. Currently, the complex language constructs are not interpreted within the response, but because of the overall modular structure, that can be added.

4.4.4 Extraction

The extraction algorithm pseudocode is shown in algorithm 3. The SRL output using the algorithm is used as an input for the extraction flow to proceed. The extraction flow first finds all the verbs from outputs. Then, it creates a lists of all the argument 0 and argument 1 associated with each verb. The code supports rejecting a list of verbs. Currently, it is common to all the questions. This can be extended to be metadata tag specific. The useful arguments and verbs are saved into a list (after removing the rejected ones).

Algorithm 3 extraction() function

Require: srl_result

$new\_verb \leftarrow find\_verbs\_from\_srl(srl\_result)$
$arg0, arg1 \leftarrow find\_arg0\_arg1\_from\_srl\_verbs(new\_verb, srl\_result)$
$useful\_verbs, useful\_arg0, useful\_arg1 \leftarrow$
$keep\_useful\_verbs\_arg0\_arg1(new\_verb, arg0, arg1, rejected\_verbs)$

4.4.5 Assignment

The assignment function is used to save the correct extracted metadata from the list of verbs and arguments into the correct global variables (detailed in table 1. The pseudocode for the assignment function is shown in algorithm 4.

Algorithm 4 assignment() function

Require: $useful\_verbs, useful\_arg0, useful\_arg1$

$global\_variables \leftarrow useful\_verbs, useful\_arg0, useful\_arg1$

4.4.6 Substitution

After the assignment process, the knowledge graph is searched for all the matching edges. One of the edges is picked at random. The target node for that edge has a question ID associated with it. The question associated with the ID has a tag present in it, which is detailed in table 3. These substitution tags need to be swapped with the correct information from the global variables. The pseudocode for the substitution process is detailed in algorith 5.

Algorithm 5 substitution() function

Require: $global\_variables$

$response \leftarrow reponse\_replace(global\_variables)$

4.4.7 Creating Metadata

The metadata creation is triggered by the strategy_response function based on a input from the user. Depending on which level strategy function triggers the metadata creation, different types of metadata are created.

1. **Use Case**:List of use cases, actors and their relationships are generated. A use case diagram is also generated.
2. **Class**:List of class and their attributes is generated. A class diagram is also generated.
3. **Activity**:List of activity name and activity sequences are generated. An activity diagram is also generated.

The generation of the UML diagrams is done through internal API in the ngUML project.

5 Prototype Validation

Organizations use Enterprise resource planning (ERP) to manage day-to-day business activities such as accounting, procurement, project management, risk management and compliance, and supply chain operations. Based on ERP systems used in logistics companies, the case studies are divided into three departments. [MW12]

- **Order Entry System**: The order entry management starts with a customer placing an order. Upon approval by the order manager, fulfillment is notified of the order.
- **Warehouse Management System**: Inventory is sorted, selected, and packed according to order entries. The warehouse manager updates the transport department once the product is ready for shipment.
- **Delivery Management System**: In the final stage, the transport manager selects mode of transport, updates the order status and the product is shipped to the end customer.

5.1 System Overview

The chatbot is integrated as part of larger SDLC requirements elicitation framework in ngUML. Figure 13 shows the interface to that chatbot visible to the user from the frontend. This has been integrated in the larger ngUML project. Figure 14 shows the default message displayed on every chatbot bootup. Figures 15 and 16 show the randomization process of the chatbot for each node as they show different questions asked in the first level of the generic questions.

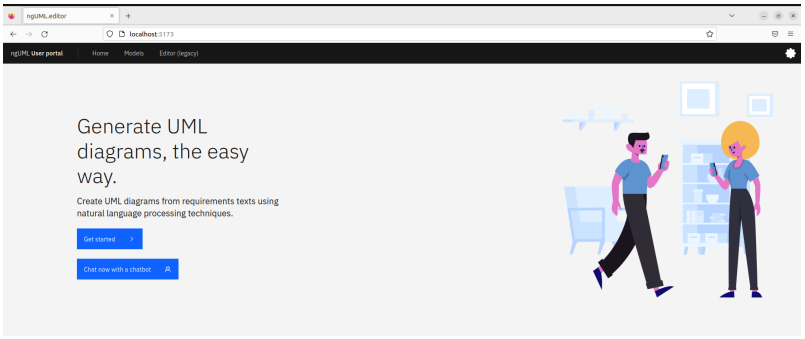

Figure 13: Chatbot Frontend Interface

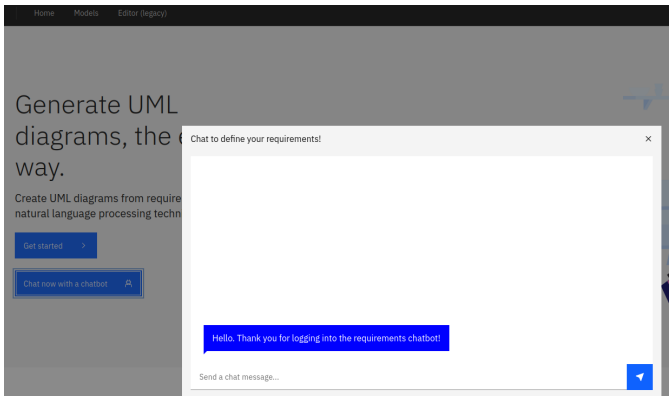

Figure 14: Chatbot Default Message on Bootup

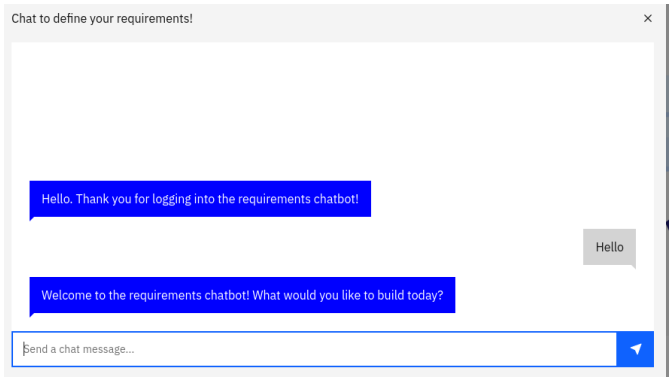

Figure 15: Generic Level Randomized Question 1

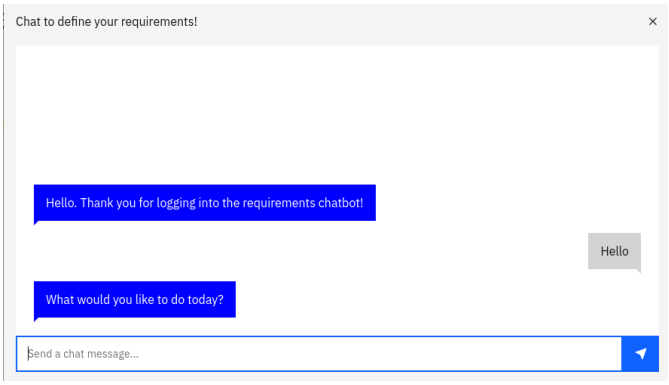

Figure 16: Generic Level Randomized Question 2

5.2 Case Study 1 : Order Entry System

The first case study focuses on the lifecycle of the order-to-cash process in which an order is received which is a sales order, the sales order has attributes like sales order ID, date, and item. The customer places an order online, The order manager processes the order. Multiple use cases are visualized in this case study.

5.2.1 General Requirements

The goal of the general questions is to gather the details of the system being generated. These details are later used in the UCA requirements elicitation. Figure 17 shows the chatbot capturing the generic requirements for the creation of an order entry system.

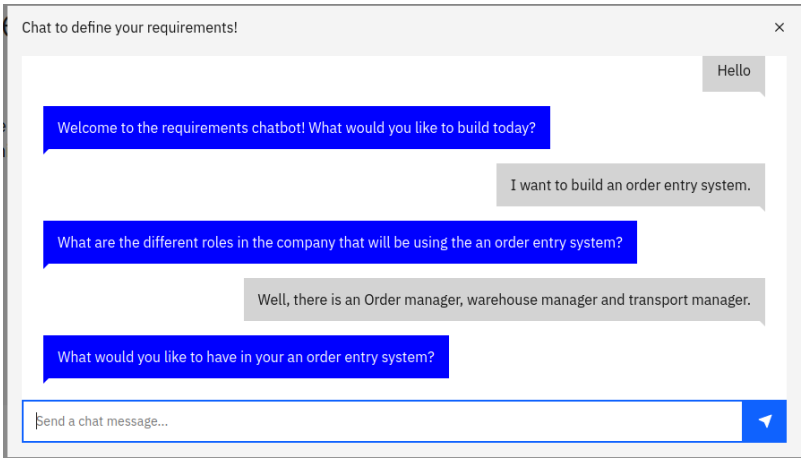

Figure 17: Generic requirements for order entry system

5.2.2 Use Case Requirements

Figure 18 shows that the chatbot elicitates the first set of use case requirements. Figure 19 shows the strategy component sending the chatbot to elicitate another set of use case requirements. The user response triggers the elicitation of the second set of use case requirements. Figure 20 shows that the chatbot elicitates the second set of use case requirements. Finally, figure 21 shows the metadata generated by the chatbot. It can be seen that the metadata generated has two sets of values for the two cycles of elicitation of use cases conducted in the case study.

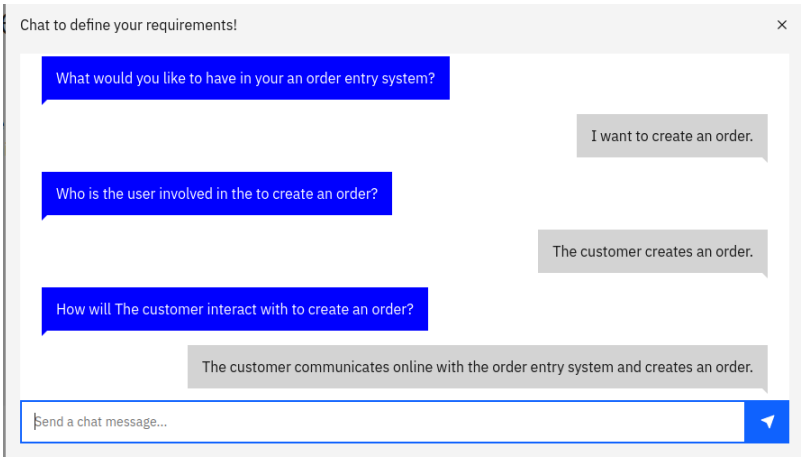

Figure 18: Use Case requirements for order entry system

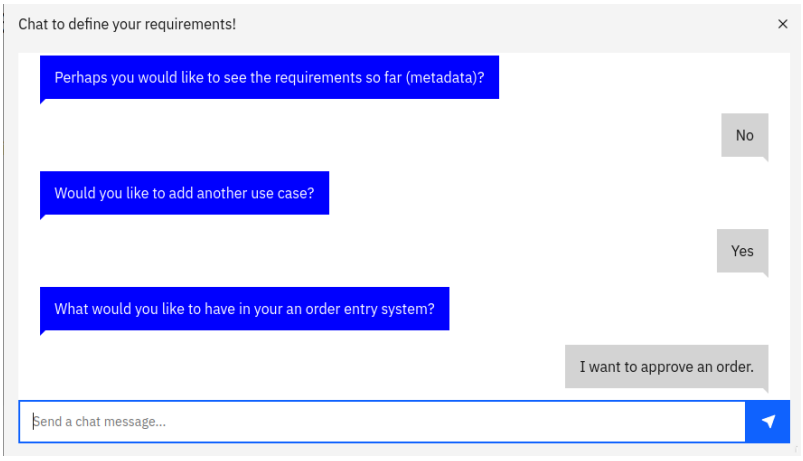

Figure 19: Strategy Questions for Use Case requirements

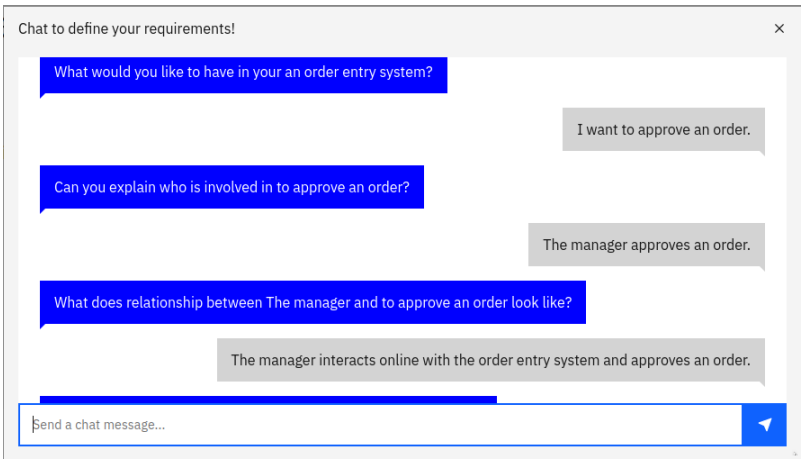

Figure 20: Second Use Case Requirements Elicitation

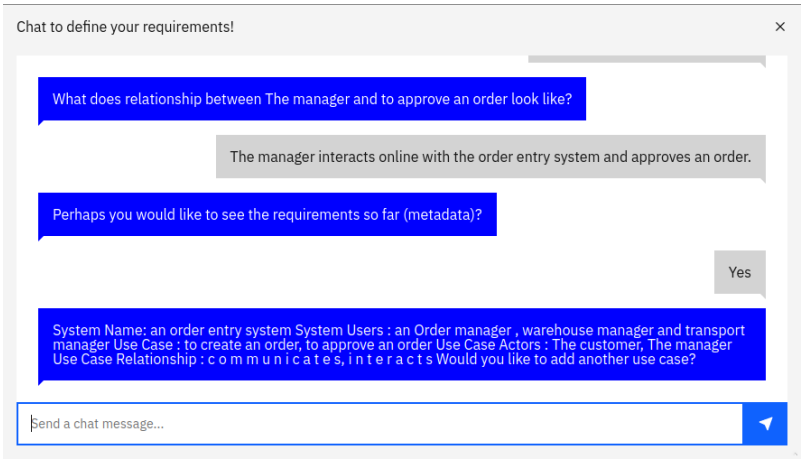

Figure 21: Use Case requirements output generation

5.3 Case Study 2 : Warehouse Management System

The second case study focuses on warehouse management system (WMS) or inventory management, which deals with fulfillment and update of stock and finally provides details to the delivery department. This case study showcases the elicitation of class requirements after the use case requirements.

5.3.1 General Requirements

Figure 22 shows the generic requirements, focusing on the generic level captured for the warehouse management system.

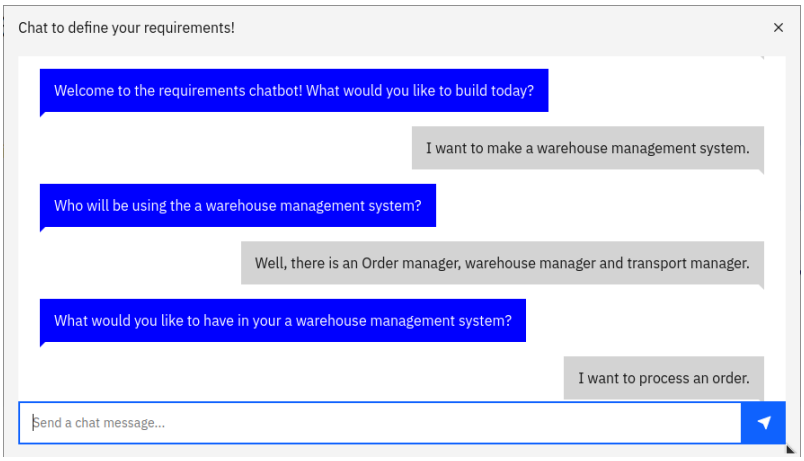

Figure 22: Generic Requirements for Warehouse Management System

5.3.2 Use Case Requirements

Figure 23 shows the elicitation of use case requirements for a warehouse management system. Figure 24 shows the generation of strategy component, generating the use case metadata and switching to class requirements elicitation based on the user response.

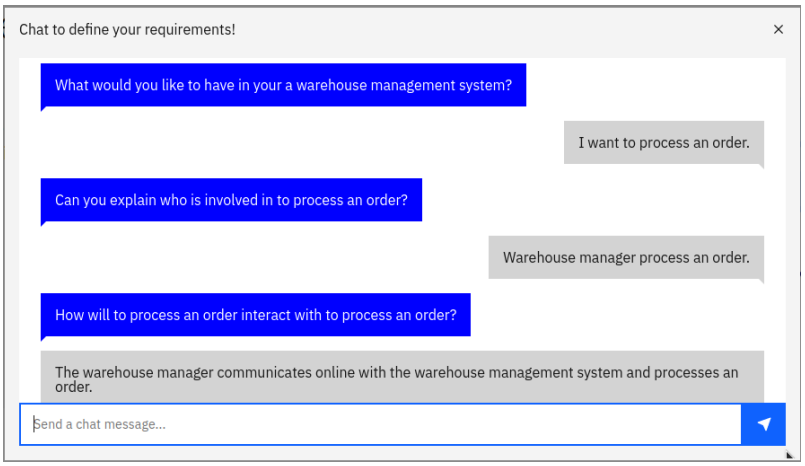

Figure 23: Use Case requirement for Warehouse Management System

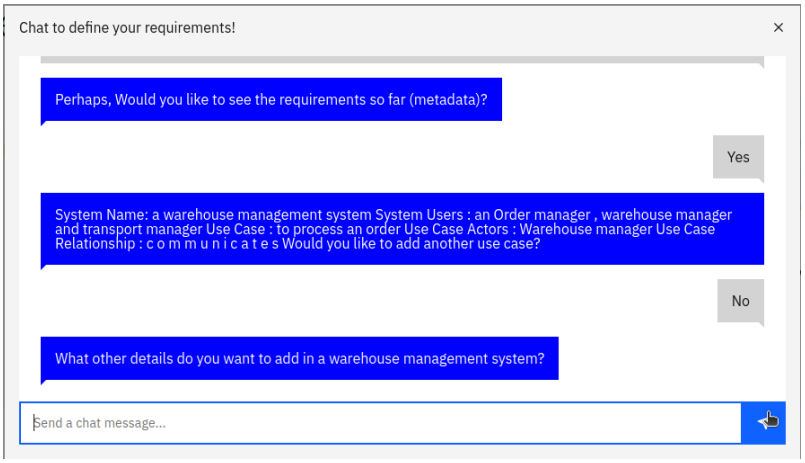

Figure 24: Conversation strategy component triggering class requirements

5.3.3 Class Requirements

Figure 25 shows the capture of the class requirements, and figure 26 shows the display of the metadata captured using the strategy questions.

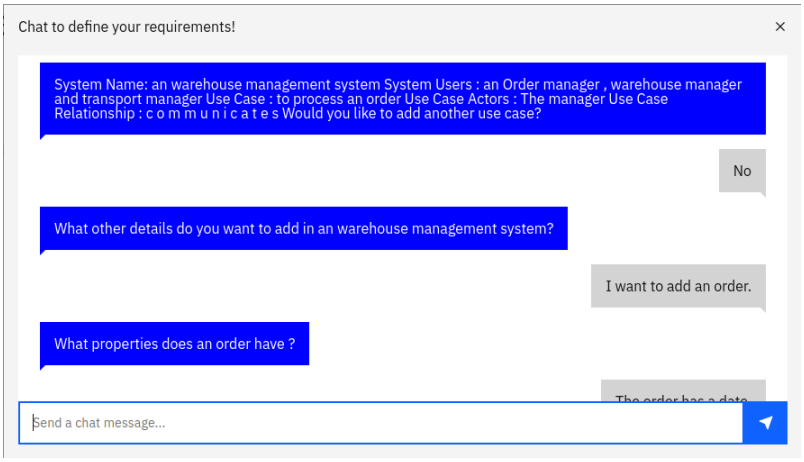

Figure 25: Class requirements for the Warehouse Management System

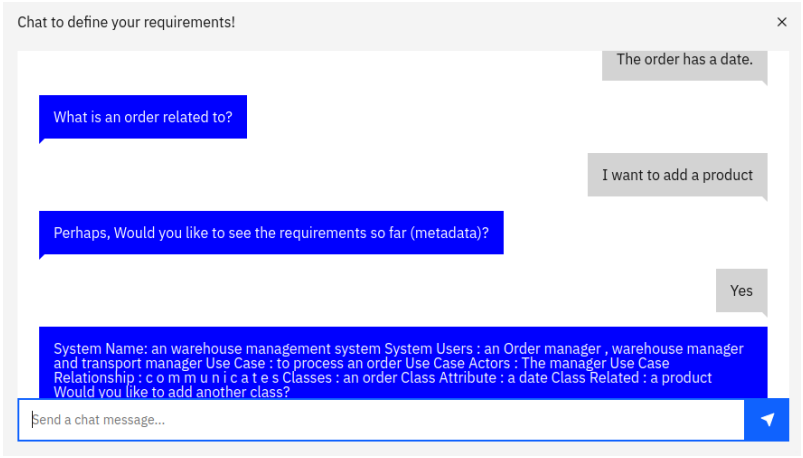

Figure 26: Strategy component triggering the display of requirements metadata

Figure 27 shows the interactive user interface to display new project created for class diagrams generation.

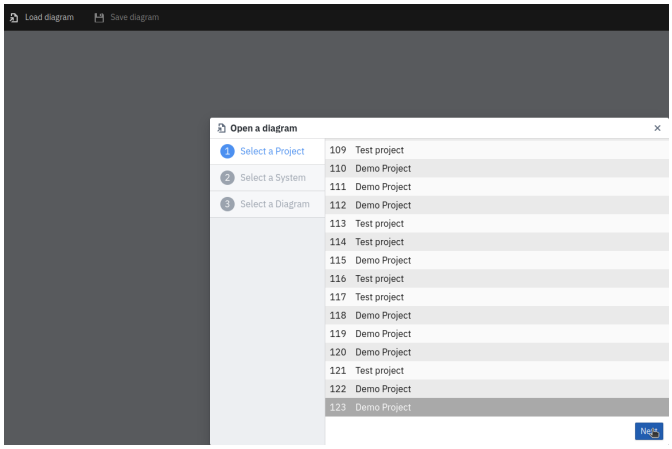

Figure 27: New project creation in the frontend user interface

Figure 28 shows the interactive user interface capturing the warehouse management system elicited using the user responses.

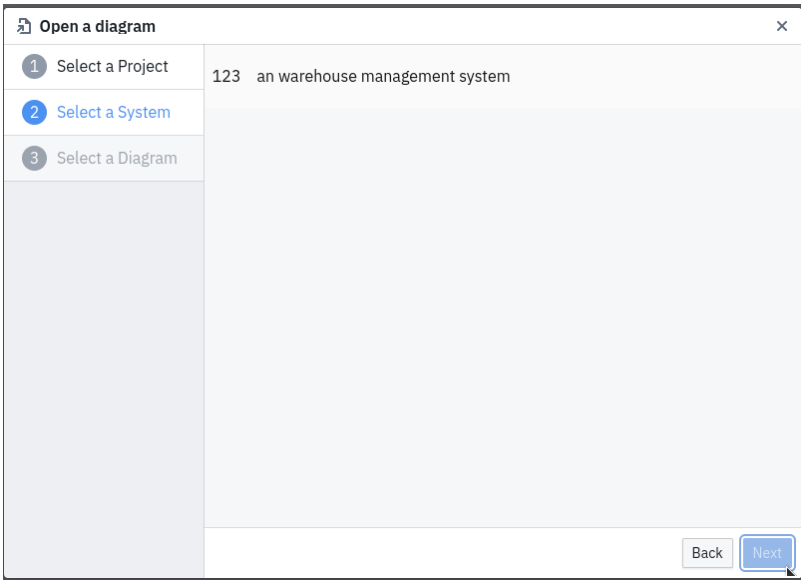

Figure 28: New system creation triggered by the user responses of warehouse management system in frontend

Figure 29 shows the interactive user interface with the class diagrams being displayed with two classes elicited from the user inputs and also the relationship between them.

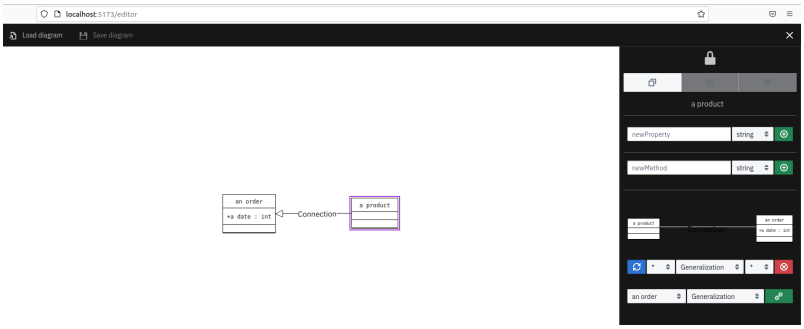

Figure 29: User interface display of the class requirements as a UML diagram

The elicited requirements can be modified further using multiple iterations within the requirements elicitation chatbot created in this thesis, or using the changes chatbot implemented in other work. Also, the user interface can be used to add more details and capture or change any new requirements as shown in the figure. Figure 30 shows attributes being added to the class diagram from a user interface screenshot.

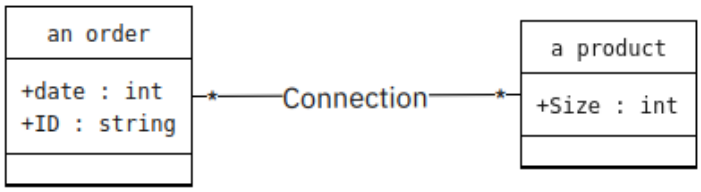

Figure 30: User Interface screenshot with class diagram

5.4 Case Study 3 : Delivery Management System

The third case study focuses on eliciting requirements to build a transportation system. The activity requirements are kept in focus for this use case.

5.4.1 Activity Requirements

Figures 31 and 32 show the activity requirements being captured for the delivery management system.

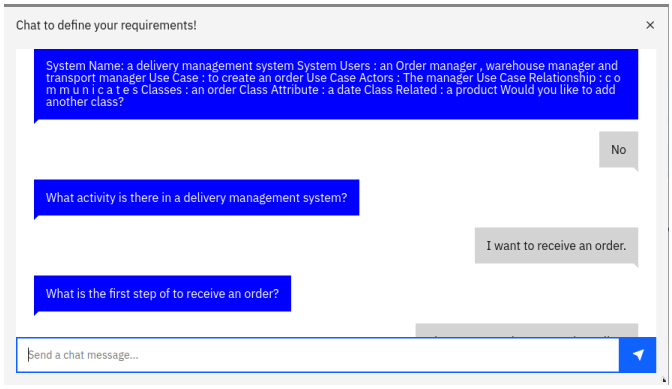

Figure 31: Activity requirements elicitation for delivery management system

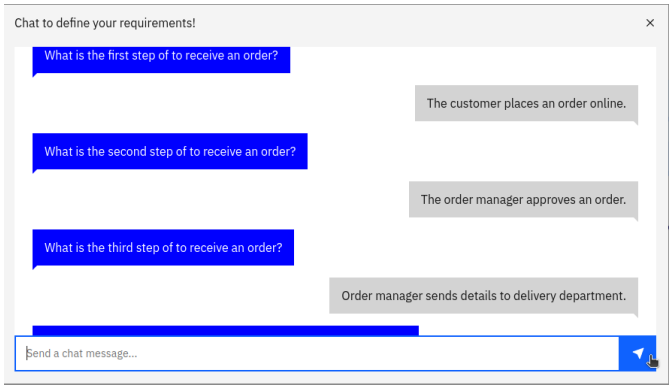

Figure 32: Sequence of activities in delivery management system

Figure 33 shows the metadata captured for the delivery management system triggered by the strategy question.Figure 34 shows the log of the user responses and nodes traversed by the user.

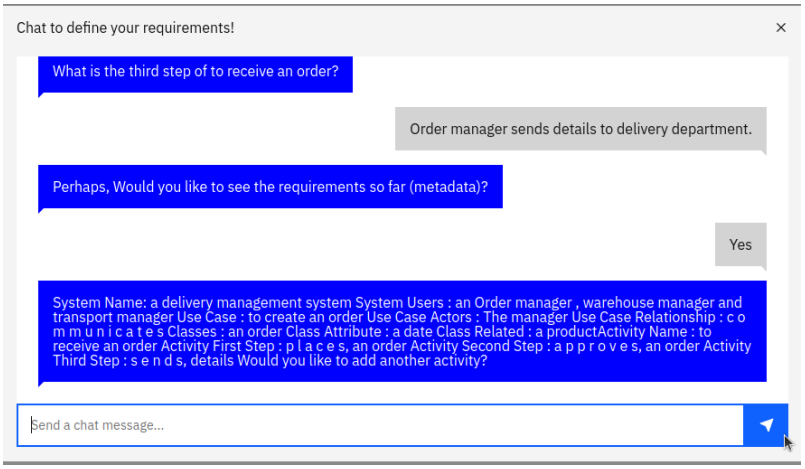

Figure 33: Conversation strategy component for metadata display

```
System Name:   a delivery management system   System Users  :    an Order manager , warehouse manager and transport manager
  Use Case  :   to create an order Use Case Actors   :  The manager Use Case Relationship  :  c o m m u n i c a t e s
  Classes   : an order Class Attribute   : a date Class Related   : a product
Activity Name   : to receive an order Activity First Step   : p l a c e s, an order Activity Second Step   :  a p p r o v e s, an order Activity Third Step    : s e n d s, details
['Hello', 'I want to make a delivery management system.', 'Well, there is an Order manager, warehouse manager and transport manager.', 'I want to create an order.', 'The manager creates an
 order.', 'The manager communicates online with the delivery management system and creates an order.', 'No', 'No', 'I want to add an order.', 'The order has a date.', 'I want to add a prod
uct.', 'Yes', 'No', 'I want to receive an order.', 'The customer places an order online.', 'The order manager approves an order.', 'Order manager sends details to delivery department.']
['q_0_1_1', 'q_0_2_2', 'u_0_1_2', 'u_0_2_2', 'u_0_3_1', 's_0_u_0', 's_0_u_1', 'c_0_1_1', 'c_0_2_2', 'c_0_3_2', 's_0_c_0', 's_0_c_1', 'a_0_1_1', 'a_0_2_2', 'a_0_3_2', 'a_0_4_1', 's_0_a_0']
```

Figure 34: Code log of user responses and the nodes traversed

5.5 Discussion

5.5.1 Research Objectives vs Prototype

The thesis had the objective to conduct design by research. As part of the design of the prototype, the following features were kept as part of the research objectives. Next, the status on achieving those features with respect to the prototype development done as part of this thesis is discussed.

1. **R-01** - *The chatbot should be able to streamline the requirement elicitation process with different components of use case, classes and activity.*
   The developed chatbot prototype comprises Use case, Class and Activity component. All the component execute the requirements elicitation for SDLC in a streamlined fashion. As part of the proof of concept, only a few features sets of these components were implemented. The proposed architecture has flexibility to add additional features in each of the components. The chatbot asks very basic questions in a streamlined manner to ensure that a beginner in the area of requirements elicitation can use it and still correctly elicitate the requirements.

2. **R-02**: *The chatbot should be use case driven and should be able to link to different components using a strategy component.*
   The chatbot designed chatbot has a use case driven design for requirements elicitation. All the other components can be accessed dynamically using the strategy component. The strategy component is under the control of the user. The chatbot starts from the use case component and can be diverted in different component directions based on user inputs.

3. **R-03**: *The chatbot should be able to do complex input metadata extraction and map it to knowledge base to correctly predict the next question.*
   The prototype chatbot provides complex NLP extraction flow using Semantic Role Labeling (SRL). The information extracted using SRL is mapped into metadata. It is further assigned and substituted into the correct questions and positions using a knowledge graph.

4. **R-04**: *The chatbot needs to link the conversation strategy component to the knowledge base to create a hybrid chatbot.*
   The developed prototype has an intricately woven conversation strategy component and the knowledge-base based design. The strategy component dictates how the chatbot is expected to jump around the knowledge graph based on inputs received from the user. The implementation does not yet give graph visualization and has limited traversal capabilities. The designed framework allows extension in this direction. Further, the noted down node history depth can further be used for adaptive conversation strategy. This can be used as a measure of timing in the conversation strategy.

5. **R-05**: *The chatbot needs to be extendable for future research and complexity increment.*
   The framework and methodology created as part of the chatbot development is completely extendible. More components and features of components can be added. Requirements questions and knowledge graph is based on a standard rules. They can be extended and automated. The strategy component can be extended to add more dynamics with respect to user inputs. Also, new algorithms can be used for metadata extraction and mapping.

6. **R-06**: *The chatbot needs to be dynamic in the questions posed to the user and not repeat same question always.*
   The chatbot prototype has multiple questions in the requirements questions for each level of each component. These questions are randomized when they are posed to a user whenever the chatbot reaches a given level for any component. This ensures non-monotonicity of the chatbot behavior.

7. **R-07**: *The chatbot needs to perform end to end requirements elicitation. The elicitation process starts from capturing the user inputs in a structured format using the chatbot. It then performs the necessary processing and extraction. At the end of the elicitation cycle, the user is presented with the metadata/UML diagrams.*
   The implemented chatbot prototype provides the capability of interacting with the user in the frontend to capture requirements in a streamlined manner. The collected inputs are automatically extracted and mapped into a database of metadata and UML diagrams. This is the first academic implementation of end-to-end pipeline of requirements elicitation using a chatbot, covering all the components of UML. To the best of our knowledge, the literature review was done for requirements elicitation processes and automated chatbots. According to the literature review, a dedicated chatbot showcasing the complete requirements elicitation process has not been done before this work.

Based on the review of the research objectives set for the research by design, a prototype of a chatbot has been developed. The chatbot meets the feature expectations set with respect to the research objectives of this thesis.

5.5.2 Review of the research questions

The research objectives for the prototype chatbot development were derived from a set of research questions in the thesis. This section discusses how the overall research answers the research questions of this thesis.

1. How can early stage requirements elicitation in SDLC be streamlined and improved using a chatbot?

A chatbot has been proposed as part of this thesis to streamline the process of requirements elicitation in early stage SDLC. The chatbot has a UCA architecture to ensure all aspects of early stage SDLC requirements elicitation is covered. The implementation has been done to cover the breadth of the topic and yet keep the requirements elicitation task simple and streamlined.

2. How can combination of strategy and knowledge graph be done to create a use case driven, dynamic and hybrid chatbot?

The implemented chatbot implements a combination of both strategy component and knowledge graph based chatbot development. The strategy component drives the overall process in terms of type and amount of requirements the user wants to elicitate at the given time. The knowledge graph component drives the overall elicitation of requirements for each of the UCA components. The chatbot prototype still has some missing text processing cleanup requirements for the improvement of display and finesse.

5.5.3 Validation Summary

The research in this thesis was conducted by design. As part of the design, a prototype chatbot was developed for requirements elicitation in early stages of SDLC. The performance of the chatbot was validated through multiple case studies done in this chapter. Additionally, the proof of concept prototype chatbot adheres to all the research objectives set as part of the thesis goals. This helped advance the research by answering the research questions posed in this thesis.

6 Conclusion

This thesis proposed the methodology for the development of a chatbot for requirements elicitation in the Software Development Life Cycle (SDLC). A hybrid chatbot with a strategy and knowledge graph-driven approach was created. The chatbot was used case driven and yet dynamic enough. This was achieved using the proposed multi-component UCA approach, connected by the strategy component.

6.1 Academic relevance

The proposed chatbot for the requirements elicitation streamlines the process as well as captures all the different components of SDLC requirements (use case, classes, and activity). All the different components (UCA) have been linked together via the strategy component. This work is the first-ever implementation in academics of an end-to-end streamlined requirements elicitation methodology using a chatbot.

The chatbot uses a hybrid approach with a mixture of direct input processing using the strategy component as well as complex metadata extraction and mapping. Using the hybrid approach, it is able to predict the next question using a knowledge graph. It is the first such implementation of a chatbot in academics. The hybrid approach linked with complex mapping provides an ideal extendible framework for the academic work to be executed and augmented in future iterations of the design.

The user experience has been kept in perspective while designing the chatbot. It is ensured that the same questions don't get posed to the user again and again. This removes the element of monotonicity. The chatbot also provides an end-to-end platform for the requirements elicitation flow through its integration into the ngUML project. The user can directly provide inputs and get the metadata and UML diagrams using an active interactive platform.

The ngUML project provides further advantages with its multiple other projects, where the requirements once created can be actively modified using other projects[cite]. This is the first such implementation in academics where a single platform provides a complete SDLC requirements elicitation solution.

6.2 Limitations

The thesis has been done as research by design. As part of the design goals in the thesis, a proof-of-concept chatbot has been developed for early-stage requirements elicitation in SDLC. Due to limitations in the design and research time, certain assumptions have been taken to simplify the development of the proof of concept. Currently, they put a limitation on the scope of requirements that can be elicited using the demonstrator. There are also limitations on the format in which the chatbot expects the inputs to be provided in order to successfully do the metadata extraction and map to the next questions in the knowledge graph.

During the course of the development of the chatbot, the limitations have been handled in a way that they can be easily improved upon. The design methodology has taken care that the overall approach and implementation is based on an extendible framework.

6.3 Future work

The research and implementation done as part of a chatbot implementation for requirements elicitation of SDLC open up multiple possibilities for future research and development in this area. Future research and development are two-pronged.

1. **Enabled as extensions of the proof of concept**:
   The proof of concept implementation in this thesis provides an extendible framework. It provides extendability with respect to adding more components to the chatbot. Additionally, both the requirements questions and the knowledge graph are created in a well-defined language. This helps on two fronts. One, if there is a need for chatbot implementation for a different system, the rules allow for the easy creation of the two files in the knowledge base. Second, it leaves the scope for future automation of the two files. This automation can be used for the automated development of system-specific chatbots and for chatbots with self-evolving strategies and dynamics. The architecture developed leaves an open room for more complex metadata extraction algorithms to be used. These algorithms can both improve and complement the existing metadata extraction done as a proof of concept.

2. **Research needed to augment and complement the current chatbot design**:
   As part of a long term research, visualization and systematic validation plays an important role. Due to limitations of time, certain parts of validation pipeline still need development. Additionally, even though the existing framework is open to automation and self learning algorithms, such implementation was not done. Extensions of this research can implement learning on the data collected from the users. This learning augmented with the validation can lead to evolving knowledge base questions and graphs. This can also lead to more fluid interactions between the strategy component and the knowledge graph metadata mapping. The input patterns which can be extracted and mapped as part of the thesis were limited as well. The extendible framework allows for more complex Semantic Role Labeling methodologies to be applied for extraction and subsequent mapping in the chatbot.

References

[AAB20] Soufyane Ayanouz, Boudhir Anouar Abdelhakim, and Mohammed Benhmed. "A Smart Chatbot Architecture Based NLP and Machine Learning for Health Care Assistance". In: Association for Computing Machinery, 2020. ISBN: 9781450376341. DOI: 10.1145/3386723.3387897. URL: https://doi.org/10.1145/3386723.3387897.

[AAG22] Shubhangi Agarwal, Bhawna Agarwal, and Ruchika Gupta. "Chatbots and virtual assistants: a bibliometric analysis". In: *Library Hi Tech* 40 (4 Aug. 2022), pp. 1013–1030. ISSN: 07378831. DOI: 10.1108/LHT-09-2021-0330.

[Abd+22] Ahmad Abdellatif et al. "A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering". In: *IEEE Transactions on Software Engineering* 48 (8 2022), pp. 3087–3102. DOI: 10.1109/TSE.2021.3078384.

[AJ20] Addi Ait-Mlouk and Lili Jiang. "KBot: A Knowledge Graph Based ChatBot for Natural Language Understanding Over Linked Data". In: *IEEE Access* 8 (2020), pp. 149220–149230. DOI: 10.1109/ACCESS.2020.3016142.

[Bai+05] Don Baisley et al. *Unified Modeling Language: Superstructure. Version 2.0*. Dec. 2005.

[BEF21] Jordan J Bird, Anikó Ekárt, and Diego R Faria. "Chatbot Interaction with Artificial Intelligence: human data augmentation with T5 and language transformer ensemble for text classification". In: *Journal of Ambient Intelligence and Humanized Computing* (2021), pp. 1–16.

[BM] Michele Banko and Robert C Moore. *Part of Speech Tagging in Context*.

[Cas+20] Mario Casillo et al. "Chatbot in Industry 4.0: An Approach for Training New Employees". In: 2020, pp. 371–376. DOI: 10.1109/TALE48869.2020.9368339.

[Cha+12] Abhijit Chakraborty et al. "The Role of Requirement Engineering in Software Development Life Cycle". In: 3 (5 2012). ISSN: 2079-8407. URL: http://www.cisjournal.org.

[CLS08] Xavier Carreras, Kenneth C Litkowski, and Suzanne Stevenson. *Semantic Role Labeling: An Introduction to the Special Issue*. 2008. URL: http://direct.mit.edu/coli/article-pdf/34/2/145/1798596/coli.2008.34.2.145.pdf.

[Csa19] Richard Csaky. "Deep learning based chatbot models". In: *arXiv preprint arXiv:1908.08835* (2019).

[Dav+06] Alan Davis et al. “Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review”. In: 2006, pp. 179–188. DOI: 10.1109/RE.2006.17.

[DW15] Sourav Dutta and Gerhard Weikum. “Cross-Document Co-Reference Resolution using Sample-Based Clustering with Knowledge Enrichment”. In: *Transactions of the Association for Computational Linguistics* 3 (Jan. 2015), pp. 15–28. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00119. URL: https://doi.org/10.1162/tacl_a_00119.

[Fer+17] D Méndez Fernández et al. “Naming the pain in requirements engineering”. In: *Empirical Software Engineering* 22 (5 2017), pp. 2298–2338. ISSN: 1573-7616. DOI: 10.1007/s10664-016-9451-7. URL: https://doi.org/10.1007/s10664-016-9451-7.

[Fra+21] Xavier Franch et al. “Data-Driven Agile Requirements Elicitation through the Lenses of Situational Method Engineering”. In: 2021, pp. 402–407. DOI: 10.1109/RE51729.2021.00045.

[Gar+18] Matt Gardner et al. “AllenNLP: A Deep Semantic Natural Language Processing Platform”. In: (Mar. 2018). URL: http://arxiv.org/abs/1803.07640.

[GHV20] Aishwarya Gupta, Divya Hathwar, and Anupama Vijayakumar. *Introduction to AI Chatbots*. 2020. URL: www.ijert.org.

[GK18] Albert Gatt and Emiel Krahmer. *Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation*. 2018, pp. 65–170. URL: https://www.narrativescience.com.

[He+17] Luheng He et al. “Deep semantic role labeling: What works and what’s next”. In: vol. 1. Association for Computational Linguistics (ACL), 2017, pp. 473–483. ISBN: 9781945626753. DOI: 10.18653/v1/P17-1044.

[Hog+21] Aidan Hogan et al. “Knowledge Graphs”. In: *ACM Comput. Surv.* 54 (4 July 2021). ISSN: 0360-0300. DOI: 10.1145/3447772. URL: https://doi.org/10.1145/3447772.

[Hor+19] Jennifer Horkoff et al. “Goal-oriented requirements engineering: an extended systematic mapping study”. In: *Requirements Engineering* 24 (2 2019), pp. 133–160. ISSN: 1432-010X. DOI: 10.1007/s00766-017-0280-z. URL: https://doi.org/10.1007/s00766-017-0280-z.

[JRG20] Raheleh Jafari, Sina Razvarz, and Alexander Gegov. “End-to-end memory networks: a survey”. In: *Science and Information Conference*. Springer. 2020, pp. 291–300.

[Kle+12] Ruth Klendauer et al. “Towards a competency model for requirements analysts”. In: *Information Systems Journal* 22 (6 2012), pp. 475–503. DOI: https://doi.org/10.1111/j.1365-2575.2011.00395.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2575.2011.00395.x.

[KZ09] Muhammad Umair Ahmed Khan and Mohammed Zulkernine. “On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software”. In: vol. 2. 2009, pp. 353–358. DOI: 10.1109/COMPSAC.2009.206.

[Lam00] Axel van Lamsweerde. “Requirements Engineering in the Year 00: A Research Perspective”. In: Association for Computing Machinery, 2000, pp. 5–19. ISBN: 1581132069. DOI: 10.1145/337180.337184. URL: https://doi.org/10.1145/337180.337184.

[LB99] Jerry Luftman and Tom Brier. *California Management Review Achieving and Sustaining Business-IT Alignment*. 1999.

[Lid01] Elizabeth D Liddy. *Natural Language Processing*. 2001. URL: https://surface.syr.edu/istpub.

[LNZ01] Xiaoqing Frank Liu, Kunio Noguchi, and Weihua Zhou. “Requirement Acquisition, Analysis, and Synthesis in Quality Function Deployment”. In: *Concurrent Engineering* 9 (1 2001), pp. 24–36. DOI: 10.1177/1063293X0100900103. URL: https://doi.org/10.1177/1063293X0100900103.

[MHK14] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. “Sentiment analysis algorithms and applications: A survey”. In: *Ain Shams Engineering Journal* 5 (4 2014), pp. 1093–1113. ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2014.04.011. URL: https://www.sciencedirect.com/science/article/pii/S2090447914000550.

[MMY08] Deepti Mishra, Alok Mishra, and Ali Yazici. “Successful requirement elicitation by combining requirement engineering techniques”. In: 2008, pp. 258–263. DOI: 10.1109/ICADIWT.2008.4664355.

[Moh14] Behrang Mohit. *Named Entity Recognition*. Ed. by Imed Zitouni. Springer Berlin Heidelberg, 2014, pp. 221–245. ISBN: 978-3-642-45358-8. DOI: 10.1007/978-3-642-45358-8_7. URL: https://doi.org/10.1007/978-3-642-45358-8_7.

[MW12] Ellen Monk and Bret Wagner. *Concepts in enterprise resource planning*. Cengage Learning, 2012.

[Nae+13] Muhammad Naeem et al. “Review of Requirements Management Issues in Software Development”. In: *Modern Education and Computer Science* (1 2013), pp. 21–27. DOI: 10.5815/ijmecs.2013.1.03. URL: http://www.mecs-press.org/.

[Nav09] Roberto Navigli. "Word Sense Disambiguation: A Survey". In: *ACM Comput. Surv.* 41 (2 Feb. 2009). ISSN: 0360-0300. DOI: 10.1145/1459352.1459355. URL: https://doi.org/10.1145/1459352.1459355.

[NL20] S Nithuna and C A Laseena. "Review on Implementation Techniques of Chatbot". In: 2020, pp. 157–161. DOI: 10.1109/ICCSP48568.2020.9182168.

[Pér+21] Sara Pérez-Soler et al. "Choosing a Chatbot Development Tool". In: *IEEE Software* 38 (4 2021), pp. 94–103. DOI: 10.1109/MS.2020.3030198.

[PGR18] Carla Pacheco, Ivan García, and Miryam Reyes. "Requirements elicitation techniques: a systematic literature review based on the maturity of the techniques". In: *IET Software* 12 (4 2018), pp. 365–378. DOI: https://doi.org/10.1049/iet-sen.2017.0144. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-sen.2017.0144.

[Ram+21] Guus J Ramackers et al. "From Prose to Prototype: Synthesising Executable UML Models from Natural Language". In: IEEE, 2021, pp. 380–389. DOI: 10.1109/MODELS-C53483.2021.00061. URL: https://doi.org/10.1109/MODELS-C53483.2021.00061.

[Red76] D R Reddy. "Speech recognition by machine: A review". In: *Proceedings of the IEEE* 64 (4 1976), pp. 501–531. DOI: 10.1109/PROC.1976.10158.

[RKW95] B Regnell, K Kimbler, and A Wesslen. "Improving the use case driven approach to requirements engineering". In: 1995, pp. 40–47. DOI: 10.1109/ISRE.1995.512544.

[Rup10] Nayan B Ruparelia. "Software Development Lifecycle Models". In: *SIGSOFT Softw. Eng. Notes* 35 (3 May 2010), pp. 8–13. ISSN: 0163-5948. DOI: 10.1145/1764810.1764814. URL: https://doi.org/10.1145/1764810.1764814.

[Sel06] Bran Selic. "Tutorial: An Overview of UML 2". In: Association for Computing Machinery, 2006, pp. 1069–1070. ISBN: 1595933751. DOI: 10.1145/1134285.1134510. URL: https://doi.org/10.1145/1134285.1134510.

[SL19] Peng Shi and Jimmy Lin. "Simple BERT Models for Relation Extraction and Semantic Role Labeling". In: (Apr. 2019). URL: http://arxiv.org/abs/1904.05255.

[SRV22] Martijn B J Schouten, Guus J Ramackers, and Suzan Verberne. "Preprocessing Requirements Documents for Automatic UML Modelling". In: ed. by Paolo Rosso et al. Vol. 13286. Springer, 2022, pp. 184–196. DOI: 10.1007/978-3-031-08473-7_17. URL: https://doi.org/10.1007/978-3-031-08473-7_17.

[Tha+21] Mansi Dinesh Thakkar et al. “Infini – A Keyword Recognition Chatbot”. In: 2021, pp. 1036–1042. DOI: 10.1109/ICAIS50930.2021.9395818.

[Umu+20] Umutcan et al. *Why We Need Knowledge Graphs: Applications*. Springer International Publishing, 2020, pp. 95–112. ISBN: 978-3-030-37439-6. DOI: 10.1007/978-3-030-37439-6_4. URL: https://doi.org/10.1007/978-3-030-37439-6_4.

[VGG20] Andrea Vázquez-Ingelmo, Alicia García-Holgado, and Francisco J. García-Peñalvo. “C4 model in a Software Engineering subject to ease the comprehension of UML and the software”. In: *2020 IEEE Global Engineering Education Conference (EDUCON)*. 2020, pp. 919–924. DOI: 10.1109/EDUCON45650.2020.9125335.

[ZH18] Darius Zumstein and Sophie Hundertmark. “Chatbots-An Interactive Technology for Personalized Communication, Transactions and Services”. In: *IADIS International Journal on WWW/Internet* 15 (1 2018), pp. 96–109. ISSN: 1645-7641. URL: https://www.researchgate.net/publication/322855718.

[Zha07] Zheying Zhang. *Effective Requirements Development-A Comparison of Requirements Elicitation Techniques Requirements analysis*. 2007. URL: https://www.researchgate.net/publication/228717829.

Appendices

A Knowledge Base

```
{
"g_Q_1_1" : "Welcome to the requirements chatbot! What would you like to build today?",
"g_Q_1_2" : "What would you like to do today?",
"g_Q_2_1" : "Who will be using the <<system_name>>?",
"g_Q_2_2" : "What are the different roles in the company that will be using the <<system_name>>?",
"u_Q_1_1" : "What would you like to have in your <<system_name>>?",
"u_Q_1_2" : "What things do you want in your <<system_name>>?",
"u_Q_2_1" : "Who is the user involved in the <<use_case>>?",
"u_Q_2_2" : "Can you explain who is involved in <<use_case>>? ",
"u_Q_3_1" : "How will <<use_case_actor>> interact with <<use_case>>?",
"u_Q_3_2" : "What does relationship between <<use_case_actor>> and <<use_case>> look like?",
"c_Q_1_1" : "What other details do you want to add in <<system_name>>?",
"c_Q_1_2" : "What other details do you want to add in <<system_name>>?",
"c_Q_2_1" : "What properties does  <<class_name>> have ?",
"c_Q_2_2" : "What characterizes <<class_name>>?",
"c_Q_3_1" : "What is <<class_name>> related to?",
"c_Q_3_2" : "What is <<class_name>> related to?",
"a_Q_1_1" : "What activity is there in <<system_name>>?",
"a_Q_1_2" : "What activity is there in <<system_name>>?",
"a_Q_2_1" : "What is the first step of <<activity_name>>?",
"a_Q_2_2" : "What is the first step of <<activity_name>>?",
"a_Q_3_1" : "What is the second step of <<activity_name>>?",
"a_Q_3_2" : "What is the second step of <<activity_name>>?",
"a_Q_4_1" : "What is the third step of <<activity_name>>?",
"a_Q_4_2" : "What is the third step of <<activity_name>>?",
"s_Q_u_0" : "Perhaps you would like to see the requirements so far (metadata)?",
"s_Q_c_0" : "Perhaps you would like to see the requirements so far (metadata)?",
"s_Q_a_0" : "Perhaps you would like to see the requirements so far (metadata)?",
"s_Q_u_1" : "Would you like to add another use case?",
"s_Q_c_1" : "Would you like to add another class?",
"s_Q_a_1" : "Would you like to add another activity?"
}
```

Figure 35: Requirements question in the knowledge base

B Knowledge Graph Nodes

```
{"nodes":
    {
        "g_Q_1_1":
        {
            "extract_type" : "semrol",
            "metadata"  : "system_name"
        },
        "g_Q_1_2":
        {
            "extract_type" : "semrol",
            "metadata"  : "system_name"
        },
        "g_Q_2_1":
        {
            "extract_type" : "semrol",
            "metadata"  : "system_users"
        },
        "g_Q_2_2":
        {
            "extract_type" : "semrol",
            "metadata"  : "system_users"
        },
```

Figure 36: Generic Nodes of the knowledge graph

```
},
"u_Q_1_1":
{
    "extract_type" : "semrol",
    "metadata"  : "use_case"
},
"u_Q_1_2":
{
    "extract_type" : "semrol",
    "metadata"  : "use_case"
},
"u_Q_2_1":
{
    "extract_type" : "semrol",
    "metadata"  : "use_case_actor"
},
"u_Q_2_2":
{
    "extract_type" : "semrol",
    "metadata"  : "use_case_actor"
},
"u_Q_3_1":
{
    "extract_type" : "semrol",
    "metadata"  : "use_case_relationship"
},
"u_Q_3_2":
{
    "extract_type" : "semrol",
    "metadata"  : "use_case_relationship"
```

Figure 37: Use Case Nodes of the knowledge graph

```
    "metadata"  : "use_case_relationship"
},
"c_Q_1_1":
{
    "extract_type" : "semrol",
    "metadata"  : "class_name"
},
"c_Q_1_2":
{
    "extract_type" : "semrol",
    "metadata"  : "class_name"
},
"c_Q_2_1":
{
    "extract_type" : "semrol",
    "metadata"  : "class_attribute"
},
"c_Q_2_2":
{
    "extract_type" : "semrol",
    "metadata"  : "class_attribute"
},
"c_Q_3_1":
{
    "extract_type" : "semrol",
    "metadata"  : "class_related_name"
},
"c_Q_3_2":
{
    "extract_type" : "semrol",
    "metadata"  : "class_related_name"
},
```

Figure 38: Class nodes of the knowledge graph

```
"a_Q_1_1":
{
    "extract_type" : "semrol",
    "metadata"  : "activity_name"
},
"a_Q_1_2":
{
    "extract_type" : "semrol",
    "metadata"  : "activity_name"
},
"a_Q_2_1":
{
    "extract_type" : "semrol",
    "metadata"  : "activity_seq_a"
},
"a_Q_2_2":
{
    "extract_type" : "semrol",
    "metadata"  : "activity_seq_a"
},
"a_Q_3_1":
{
    "extract_type" : "semrol",
    "metadata"  : "activity_seq_b"
},
"a_Q_3_2":
{
    "extract_type" : "semrol",
    "metadata"  : "activity_seq_b"
},
"a_Q_4_1":
{
    "extract_type" : "semrol",
    "metadata"  : "activity_seq_c"
},
"a_Q_4_2":
{
    "extract_type" : "semrol",
    "metadata"  : "activity_seq_c"
}
```


Figure 39: Activity Nodes of the knowledge graph

```
    },
    "s_Q_u_0":
    {
        "extract_type" : "direct",
        "metadata"  : "output-use_case"
    },
    "s_Q_c_0":
    {
        "extract_type" : "direct",
        "metadata"  : "output-class"
    },
    "s_Q_a_0":
    {
        "extract_type" : "direct",
        "metadata"  : "output-activity"
    },
    "s_Q_u_1":
    {
        "extract_type" : "direct",
        "metadata"  : "next-use_case-class"
    },
    "s_Q_c_1":
    {
        "extract_type" : "direct",
        "metadata"  : "next-class-activity"
    },
    "s_Q_a_1":
    {
        "extract_type" : "direct",
        "metadata"  : "next-activity-output"
    }
},
```


Figure 40: Strategy Nodes of the knowledge graph