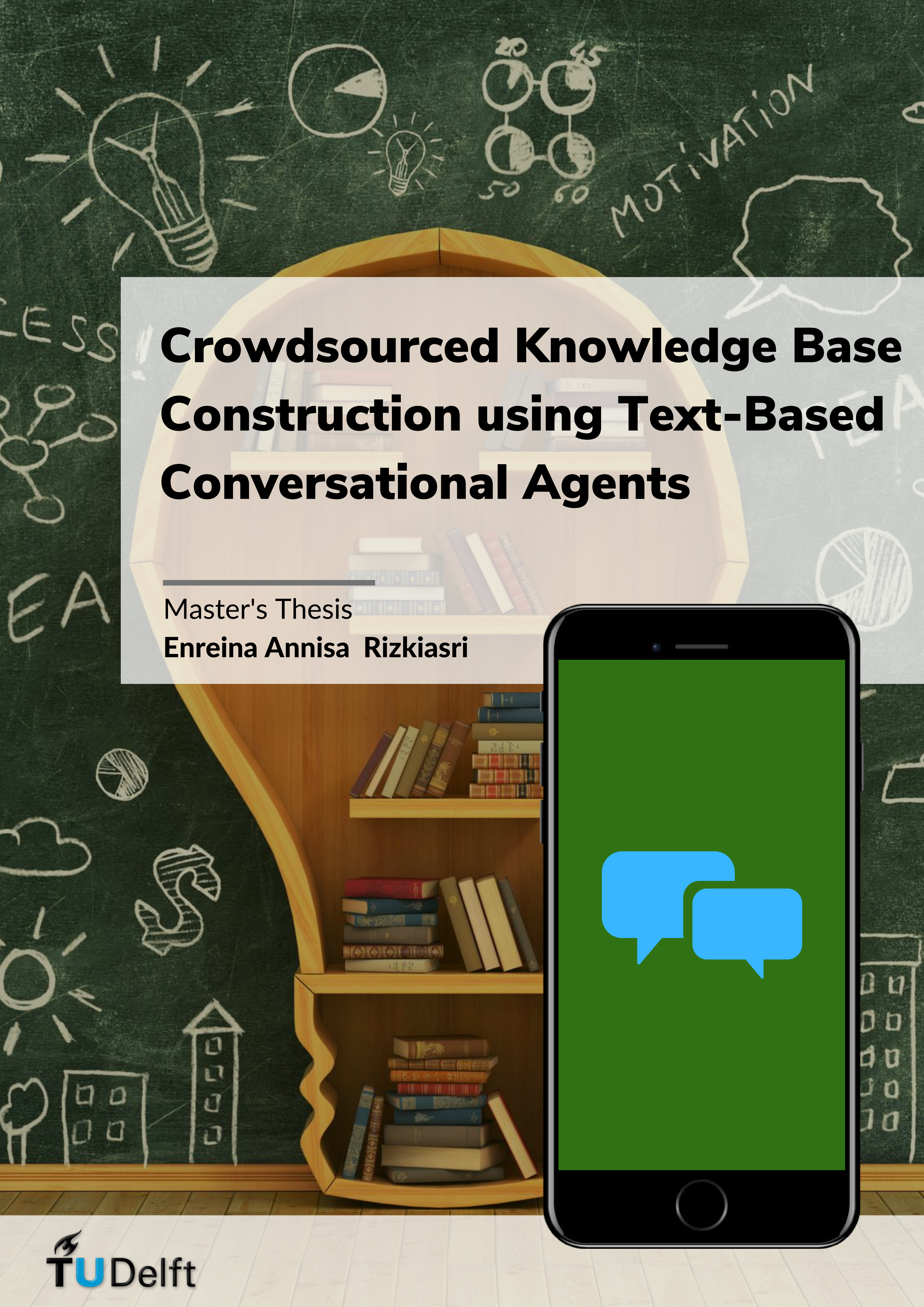


# Crowdsourced Knowledge Base Construction using Text-Based Conversational Agents

Master's Thesis

**Enreina Annisa Rizkiasri**

TU Delft

# Crowdsourced Knowledge Base Construction using Text-Based Conversational Agents

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
TRACK DATA SCIENCE & TECHNOLOGY

by

Enreina Annisa Rizkiasri
born in Jakarta, Indonesia

Web Information Systems
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
`http://wis.ewi.tudelft.nl`

# Crowdsourced Knowledge Base Construction using Text-Based Conversational Agents

| | |
|---|---|
| Author: | Enreina Annisa Rizkiasri |
| Student id: | 4701224 |
| Email: | e.a.rizkiasri@student.tudelft.nl |

**Abstract**

Knowledge Base Construction (KBC) is a challenging and complex task involving several substeps and many experts of a knowledge domain. Crowdsourcing approach has been used to support KBC with promising scalability and output quality, but to enable even more people to participate in KBC, there is a need to broaden the pool of workers beyond the ones who are already familiar with existing crowdsourcing platforms. Meanwhile, the number of people who use messaging platforms has been increasing. There is also a renowned popularity of text-based conversational agents – *chatbots* – existing on these messaging platforms. By leveraging the fact that there is a large number of users who are familiar with conversational interfaces, we see an opportunity to broaden the participants of crowdsourced KBC by using chatbots to execute KBC tasks.

In this thesis, we investigate the use of chatbots to enable crowdsourced construction of knowledge bases. We design a conversational crowdsourcing platform to support the execution of KBC tasks. An experiment involving 43 students using our system and interviews with 7 participants were conducted to evaluate the system within the context of constructing a knowledge base for the TU Delft campus. From the results, we show that the platform is suitable to be used for crowdsourced KBC with justifiable execution time, accuracy, and completeness. We also laid out the potential future work to improve and extend the functionalities of the chatbot system.

Thesis Committee:

| | |
|---|---|
| Chair: | Prof.dr.ir. G.J.P.M. Houben, Faculty EEMCS, TU Delft |
| University Supervisor: | Prof.dr.ir. A. Bozzon, Faculty EEMCS, TU Delft |
| Daily Supervisor: | S. Qiu, PhD Student, Faculty EEMCS, TU Delft |
| Committee Member: | Dr.ir. J.A. Pouwelse, Faculty EEMCS, TU Delft |

# Preface

This thesis report concludes my two years of study to obtain a Master Degree in Computer Science at the Delft University of Technology (TU Delft). The work towards completing this thesis has given me an opportunity to grow in so many ways: from being able to extend my knowledge beyond the courses I have taken prior to the thesis work; challenging myself to be persistence and critical toward my own work; as well as developing myself to be a better researcher and a better person.

I would like to give my gratitude to Alessandro Bozzon, who has allowed me to work under his supervision. Without his insightful input, this work wouldn't be able to be completed with sufficient quality. I would also like to thank Sihang Qiu, my daily supervisor who has always been available whenever I have doubts regarding my thesis work, giving on-point feedbacks as well as keeping the research direction on the track. Furthermore, I would like to thank the members of my thesis committee, Geert-Jan Houben and Johan Pouwelse, for taking their time to read the report as well as attending my thesis assessment.

Special thanks to my friend, Neha Sree Thuraka, whom I have been working with throughout finishing our thesis. We discussed, designed, and implemented the crowdsourcing system together to achieve the goal of our thesis. I also wish to thank all the participants who have volunteered to try and use the system, without them I would not have been able to conduct the experiments which are important to complete this thesis.

I would also like to give my special thanks to my friends and family. Without their emotional and motivational support, I would not have been able to reach this point of completing my thesis. Finally, thank you to TU Delft for supporting the funding of my study through the JvEffen Excellence Scholarship.

Enreina Annisa Rizkiasri
Delft, the Netherlands
August, 2019

# Contents

# List of Figures

# Chapter 1

# Introduction

These days, there is a vast amount of information available to be accessed by people. However, from the perspective of technology, not all information is directly usable because they are often presented in an unstructured format such as web pages, articles, and other formats. This kind of information would be more useful to support information-based systems, such as web-search and question-answering technologies, if it is stored in a semantically structured way (that is machine-processable) while allowing for consumption by humans.

A repository of structured information – or interchangeably, knowledge – is often referred to as a **Knowledge Base (KB)**. A KB consists of entities, representing concepts or real-world objects, with their relationships between each other. The relationship between entities can be in the form of object-property relationship (e.g. the entity **Netherlands** has a property **capital city** that links to the **Amsterdam** entity) or hierarchical relationship (e.g. **Amsterdam** is an **instance of** the concept **city**). Notable examples of KBs are DBPedia [35], Wikidata [57] (replacing its predecessor Freebase [43]), and Yago [53].

Producing and populating KBs has been one of the main challenges in the field of the Semantic Web. The process of populating a knowledge base is often referred to as **Knowledge Base Construction (KBC)**. KBC is a non-trivial process consisting of several substeps: starting from identifying concepts from the unstructured knowledge source, defining and classifying these concepts, determining relationships between concepts, and finally constructing the knowledge base from these definitions. With the complexity of the process, the output of a KBC process tends to have a trade-off between its quality (the correctness of the stored knowledge) and its quantity (how much information is included in the KB) [19].

Constructing a knowledge base can be done either by manual or automatic technique. A straightforward manual approach is to recruit domain experts and construct the knowledge base by hand. These experts are tasked to annotate unstructured information and input them into a structured KB. Though the quality of the KB can be controlled as each entity are input by human experts by hand, this approach is costly and time-consuming especially with large-scale complex knowledge [25].

Automatic approaches, on the other hand, employ information extraction techniques on the unstructured knowledge sources such as articles, books, and web pages. Although it has the advantage in term of cost and time, the automatic approach often yields to noisy output, leading towards a knowledge base which lacking in accuracy and coverage [25].

The crowdsourcing approach to KBC has been explored to overcome the limitation of the manual and automatic approach. The term crowdsourcing is used for any approach that tries to distribute a number of tasks to a pool of workers. A research manifesto [46] of the intersection between the field of Semantic Web and Crowdsourcing tries the opportunity to leverage crowdsourcing technique as a part of KBC workflow. This research manifesto leads to a number of works [11, 42, 50] investigating on to what extent crowdsourcing can support the task of KBC.

The crowdsourcing approach is more scalable compared to the manual approach, as we can break down the KBC task into smaller tasks and distribute them to a pool of workers who each can work independently to complete the task. Additionally, compared to the automatic approach, crowdsourced KBC has the advantage of better quality output, as it involves humans to work on the construction of a KB. The quality, though, might not be on par with the quality of a KB constructed by hand by experts, as the workers participating in crowdsourced KBC might also include people who are not experts of the KB domain. To improve the KB quality, tasks need to be assigned to workers optimally based on the level of expertise needed. This issue is addressed through task assignment and has been addressed for general crowdsourcing in [15, 20, 34] and specifically for KBC in [12].

Another issue of using crowdsourcing to KBC is the availability of workers. Crowdsourcing tasks are typically distributed on crowdsourcing platforms such as Mechanical Turk[1] and Figure Eight[2] or through a mobile app [40]. Meanwhile, constructing a KB with sufficient quality and coverage would need to involve a large number of people. There is a need to broaden the pool of workers beyond the ones who are already familiar with existing crowdsourcing platforms. We need to look into platforms which these people are already familiar with in order to reduce the learning curve of completing KBC tasks.

Meanwhile, interest on chatbots – text-based conversational agents that mimic human conversation to enable tasks completion [24] – has resurfaced because of the increased popularity of Messaging platforms such as Facebook Messenger, Whatsapp, and Telegram. Based on this popularity, chatbots can be seen as an opportunity to broaden the pool of workers to execute crowdsourcing tasks, by leveraging the fact that there exists a crowd of people using these messaging platforms. Additionally, familiarity with conversational interfaces can potentially lower the learning curve of completing crowdsourcing tasks.

From this opportunity, Mavridis et al. [37] did a work on the use of chatbot in microtask crowdsourcing. The result of their work shows that chatbots have comparable output quality and execution time, and gain higher user satisfaction compared with using traditional web-based interface. Based on their findings, we see the potential of employing their approach specifically to KBC process. Therefore, we would like to design KBC as microtasks and investigate to what extent can a chatbot support KBC process.

[1]https://www.mturk.com/

[2]https://www.figure-eight.com/

## 1.1 Problem Definition and Goal

The main goal of our thesis is to investigate to what extent chatbots can be used to enable the process of crowdsourced KBC. To achieve this goal, we need to take a look into the challenges that come from employing a crowdsourcing approach to KBC and presenting the KBC task using chatbots.

First of all, KBC is indeed a non-trivial task. There has been a need to formalize the methodology to construct a KB as well as bridging the gap between domain experts with no knowledge of KBC with the KBC process itself. The challenge lies in specifying a "step-by-step" guideline towards the construction of a KB. As there have been numerous works proposing methodologies for KBC, a follow-up challenge would be on how to bridge this methodology to a more refined decomposition of KBC task into microtasks that are suitable to be incorporated into a crowdsourcing workflow.

The second challenge comes from the goal of bringing conversational interface into supporting KBC process. The challenge is on how to map the decomposition of the KBC task to be presented in a conversational setting. Also, to bridge the gap between generic crowdsourcing workflow and KBC workflow, a conversational crowdsourcing platform specifically for KBC would have to be designed.

Finally, as mentioned in the previous section, one of the main challenges of constructing a KB is to maintain the trade-off between its quality and quantity. At the same time, there is also the need to reduce time and cost when constructing a KB. The challenge here lies in designing and validating a system such that it can support KBC while maintaining correctness, coverage, and scalability.

From the previous discussion, we can summarize the challenges that need to be addressed in order to achieve our goal: (1) decomposing the KBC task; (2) mapping and presenting the task in a conversational crowdsourcing system; and (3) retaining correctness and coverage of the constructed KB while maintaining the scalability of the process. In the next section, we will refine the research questions to address these challenges.

## 1.2 Research Questions

Drawing from the problem defined in the previous section, we would like to know how to decompose and organize KBC task and present them on a text-based conversational interface. We set our main focus of the thesis to answer the following main research question (RQ):

**Main RQ: How could a text-based conversational agent be designed to enable the crowdsourced construction of knowledge bases?**

To answer our main research question, we derive three research sub-questions. The questions are listed as follows along with what they would address in this work.

**RQ 1: What is the state of the art in knowledge base construction and conversational crowdsourcing?**

To define a decomposition of KBC task, we would need to look into existing state-of-the-art methodologies of constructing a knowledge base. We also look into existing works investigating the use of crowdsourcing approach to construct knowledge bases as well as the viability of conversational interface for crowdsourcing. We also highlight the limitations that come from previous works on crowdsourced KBC.

**RQ 2: How could a text-based conversational agent be designed and implemented to support the microtask crowdsourcing for knowledge base construction?**

Drawing from the answer for RQ1, we analyze and define KBC as smaller tasks – *microtasks* – that are decomposed from the task of KBC. We shall design the task such that it is small enough and appropriate to be presented in a text-based conversational interface. This part will address challenge (1) discussed in our problem definition.

As a proof-of-concept, we shall design a system specifically designed to support the execution of the previously defined microtasks. The system design would be based on existing state-of-the-art techniques of building text-based conversational interface. We will also have to consider both the conversational and user interface design based on the current standard of conversational User Experience (UX) design. This part of the system design shall address challenge (2) as discussed in our problem definition.

**RQ 3: How does the interaction style of the text-based conversational agent affect the construction time and quality of the KB?**

Our next concern comes to the quality and quantity of constructed KB as the output of the KBC process supported by the chatbot system. In order for us to understand to what extent the system can be used for KBC while still retaining scalability and the output quality (addressing challenge (3)), we shall run an experiment to assess both the system and the constructed KB. The experiment would be based on the variation of interaction style introduced in the system as well as task types and how they affect the correctness and coverage of the constructed KB. Additionally, we would also investigate on to what extent the chatbot is usable to construct a knowledge base.

## 1.3 Contributions

Our work sits at the intersection of three fields: knowledge base construction, crowdsourcing, and conversational interface. Into our work, we incorporate what we have learned from the three fields and made three main contributions as listed below.

- **C1:** We contribute a literature survey on three fields and their intersection. The three fields are well explored independently in previous studies, so our contribution would be summarizing the fields as well as drawing relation between them. We first look into the state-of-the-art methodology to construct knowledge base, before we delve into how crowdsourcing can support a KBC process. We also investigate how crowdsourcing and text-based conversational agent can support each other. The findings on what can be explored on conversational crowdsourcing for KBC shall be the groundwork on which our research would be based on.

- **C2:** We designed and implemented a conversational crowdsourcing system that is intended to enable the process of KBC. This contribution consists of: the decomposition and organization of KBC process into microtasks; the design of the crowdsourcing system; the design of the conversational interface; and an implementation of the system design as a proof-of-concept. The importance of this contribution is to show the feasibility to design a conversational crowdsourcing platform specialized for constructing knowledge bases. The implementation part especially would also enable us to carry out experiments in order to evaluate the system.

- **C3:** Finally we setup and conduct an experiment in order to evaluate our design and to understand better on how can it support a KBC process. An implementation of the designed chatbot called **CampusBot** is used to run the experiment. The experiment was run on tasks which were devised from 4 item types of a campus knowledge base: Places, Food, Course Questions, and Trash Bins. We recruited 43 students to try out and use CampusBot in their daily life to complete some tasks and measure the time they each took to complete the tasks. We also measure the quality and completeness of the resulting KB and how it relates to our system design. As a follow-up study, we also interviewed 7 participants for a guided one-to-one session where we observe and ask them questions while they use our chatbot system.

## 1.4 Thesis Outline

This thesis consists of seven chapters, where each chapter focuses on the following parts building the whole thesis. Figure 1.1 illustrates the flow of the chapters with their corresponding contributions.

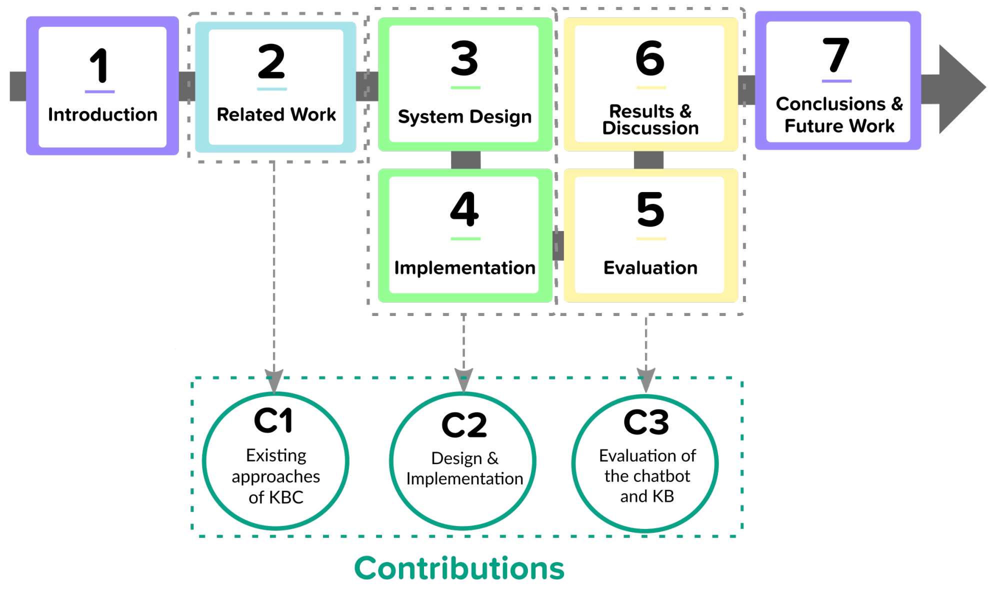


Figure 1.1: Thesis Outline

In chapter 2, we discuss the existing methodologies of constructing knowledge base and how crowdsourcing has been incorporated to support KBC. Chapter 3 and

4 will focus on the system design and implementation of the chatbot for constructing KB. In chapter 5, experimental design is described to evaluate the design of the chatbot, while chapter 6 discusses the experiment results. Finally in chapter 7, we summarize the contributions, conclude the thesis with final remarks as well as potential future work.

# Chapter 2

# Related Work

This chapter is a survey of previous works in the field of Knowledge Base Construction (KBC) in relation to crowdsourcing and specifically the potential of using conversational interface to support it. The survey answers to RQ1 described in Chapter 1. The gap between research found in this survey would be our starting point for our work to focus on.

As this survey involves multiple fields (KBC, Crowdsourcing, and Conversational Interface) at once, we structure this chapter as follows:

1. We will first survey on the definition of KBC and known KBC methodologies that have been used so far. This part would give us an initial guideline on what are the subtasks of constructing a knowledge base.

2. Then, we would focus on surveying previous works which utilize crowdsourcing as a mean to support the task of KBC. We would associate the works to each subtask of KBC from the previous section.

3. Finally, we would discuss related works in the field of conversational interface and its potential as an alternative crowdsourcing interface.

## 2.1 Knowledge Base Construction and Ontology Engineering

In a practical guide of "Ontology Development 101" [41], an **ontology** is defined as a formal description of concepts in a domain of discourse (called **classes**), with a set of properties of each concept to describe features (**slots**), and restrictions on slots (**facets**). Additionally, a set of instances built on top of an ontology constitutes a **knowledge base**.

The set of activities involving ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies is referred as **ontology engineering** [26].

As a knowledge base consists of both the ontology and its instances, ontology engineering is a part of knowledge base construction (KBC). While ontology engineering focuses more on concepts and relations between concepts, KBC involves the task of mapping these concepts to real entities referred as instances.

Note that the methodologies to develop ontology discussed below mostly involve the activities of creating instances of the ontology itself. As defined in [41], a knowledge base is an ontology with a set of individual instances, thus we will use the term KBC and ontology engineering interchangeably.

Methodologies of ontology engineering have been a subject of literature survey several times, such as in the book Ontological Engineering by Gomez-Perez [26] and a survey of collaborative ontology engineering by Simperl [51]. We will first discuss ontology engineering methodologies in general, and then methodologies which are intended for collaborative ontology engineering.

### 2.1.1 Methodology for Ontology Engineering

The ontology development process was identified along with the METHONTOLOGY framework in [23]. It refers to the set of stages which the ontology moves during its life, but does not dictate the order on the execution of each stage. Three categories of activities were advised in [13] to be carried in an ontology development process:

- **Ontology management activities** consisting of scheduling, control, and quality assurance.
- **Ontology development oriented activities** consisting of three subcategories of activities:
  - **Pre-development activities** consisting of environment and feasibility study.
  - **Development activities** consisting specification, conceptualization, formalization, and implementation.
  - **Post-development activities** consisting of maintenance of the ontology and how it is used by other ontologies or applications.
- **Ontology support activities** consisting of activities performed at the same time with development-oriented activities such as knowledge acquisition, ontology evaluation, integration, and alignment.

[26] gives a thorough overview of several known methodologies for ontology engineering. They compared several methodologies and methods used to build ontologies, either from scratch or by reusing other technologies. The methods discussed were: the Cyc method, the Uschold and King's method, the Grüninger and Fox's methodology, the KACTUS approach, METHONTOLOGY, the SENSUS method, and the On-To-Knowledge methodology.

Each of the discussed methodologies were mostly developed by experience or guidelines of developing a specific ontology. The Cyc method, for example, was the method used for building the ontology of Cyc, a knowledge base of common sense. The Uschold and Kingâs method, on the other hand, was developed from the their experience in developing the Enterprise Ontology of the Artificial Intelligence Applications Institute at the University of Edinburgh.

Most of these methodologies cover some part of the activity category in the ontology development process, but the most stable and comprehensive ones are the METHONTOLOGY and On-To-Knowledge methodology. For our work, we will use the METHONTOLOGY as our guideline for ontology engineering.

The METHONTOLOGY methodology comprises of 7 phases to build ontologies from scratch: specification, knowledge acquisition, conceptualization, integration, implementation, evaluation, and documentation. Each of the phase will be briefly described as follows.

1. **Specification**: the goal of this phase is to produce either an informal or formal ontology document specification. It consists of the purpose, level of formality, and scope of the ontology.

2. **Knowledge Acquisition**: this phase is an independent activity supporting the development of ontology. The goal is to collect knowledge by means of either interviews with domain experts and/or text analysis of the knowledge source in the form of books, articles, and so on.

3. **Conceptualization**: in this phase, the domain knowledge is structured into a conceptual model describing problem and solution in terms of the domain vocabulary.

4. **Integration**: the METHONTOLOGY framework encourages the re-usage of existing ontologies, with the integration phase, an integration document is produced to link between existing ontology and the ontology being developed.

5. **Implementation**: this phase's objective is to formalize the conceptual model into a formal computational language.

6. **Evaluation**: in this phase, a technical judgement of the ontology is carried out, in respect to the requirements specified in the specification phase.

7. **Documentation**: documentation is encouraged by the METHONTOLOGY framework on every phase of the ontology development,

Each phase was described briefly in the original paper of METHONTOLOGY, with the conceptualization phase described in detail in the follow-up paper of building ontology at the knowledge level [13]. The conceptualization phase can be broken down into 11 tasks which are shown in Figure 2.1.

First, a glossary of terms to be included in the ontology is built including each of their description and synonyms (**Task 1**). Each of this term has to be identified either as a concept or instance attribute. Then, the concepts are classified as such that a concept taxonomy is built (**Task 2**). After that, a binary relation diagram (**Task 3**) is built to identify relationships between concepts in the ontology. For the next task, a concept dictionary (**Task 4**) is built. The concept dictionary contains all the domain concepts with their relations, instances, and their class and instance attributes.

After the concept dictionary is built, the detail of relations (**Task 5**), instance attributes (**Task 6**), and class attributes (**Task 7**), as well as constants (**Task 8**) should be defined. Then, formal axioms (**Task 9**) and rules (**Task 10**) should be described for constraint checking and inferring attribute values. Instances could also be introduced (**Task 11**), thus making the ontology a full-fledged knowledge base.

Note that although some order must be followed to carry each of the task of the conceptualization phase, the process is not strictly sequential as an ontology engineer can return to any of previous task at point where a new term is introduced.

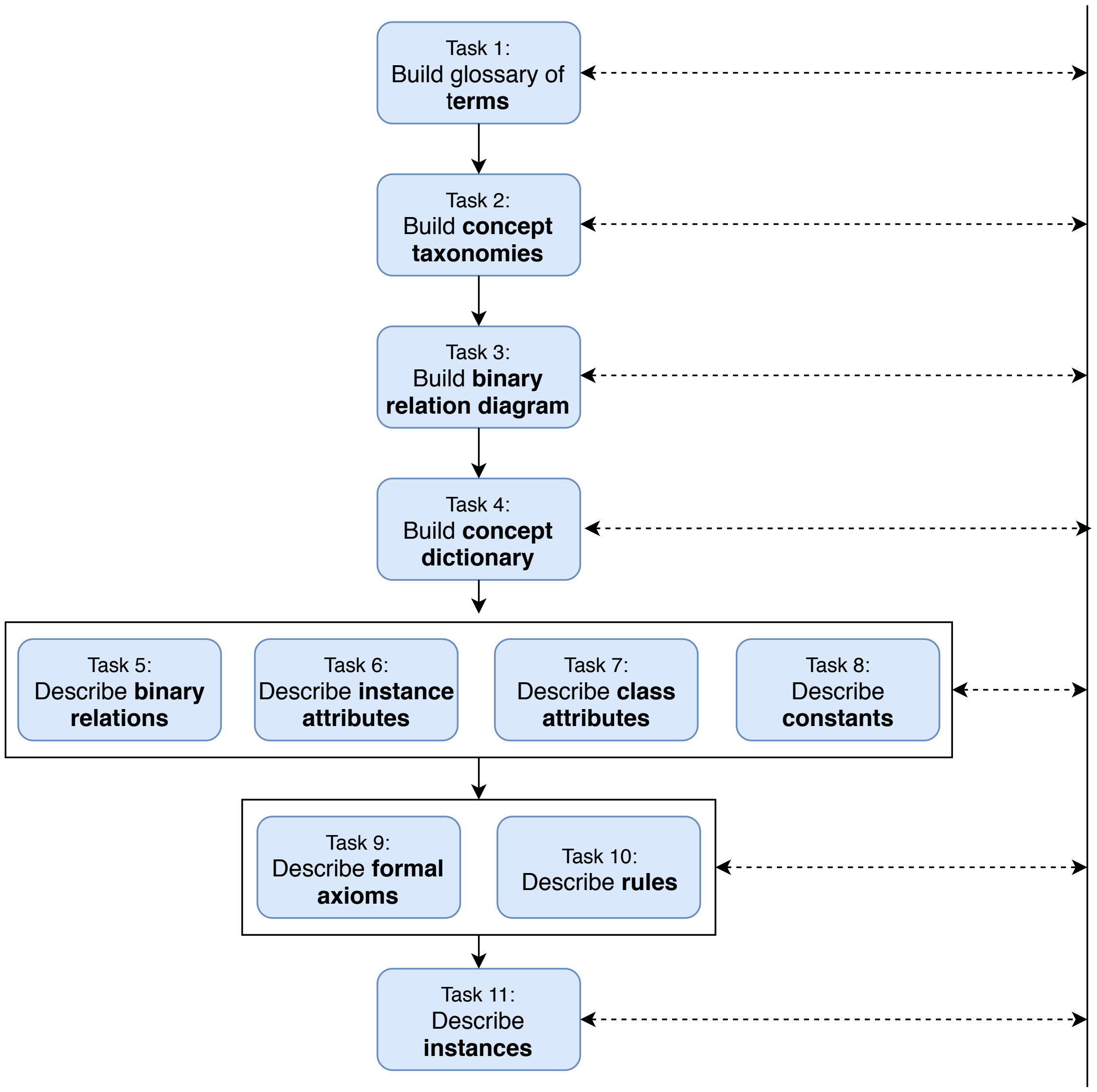


Figure 2.1: Tasks of Conceptualization Phase

Based on this tasks, we can relate each task based on the primitive element of an ontology that it involves. In general an ontology would consist of: classes, instances, attributes, relationships, hierarchy, axioms and rules. An overview table of these primitive elements is shown in Table 2.1. Each of this element would connect with one or more elements, thus building the whole ontology.

In the scope of our work, we will focus mostly on the **Conceptualization** phase, specifically on **Task 11**, which is to describe instances. The system design in our work would facilitate the creation of instances on top of a predefined ontology schema. Our work would also touch a part of the **Integration** and **Implementation** phase where we would integrate the knowledge base constructed by the system to an existing ontology. The system, however, could be easily extended to cover other phases.

The METHONTOLOGY framework has introduced a stable guideline in order to

Table 2.1: Primitive Elements of an Ontology

| Primitive Element | Description | Tasks | Example |
|---|---|---|---|
| Class | Represents a concept or group of similar concepts | Task 1, Task 2, and Task 4 | Building, Person, Animal, City |
| Instance | Represents a real-world entity or item | Task 11 | The Eiffel Tower, Taylor Swift, Delft |
| Attribute | A property of either a class or instance | Task 6, Task 7, and Task 8 | name, colour, location |
| Relationship | A special type of attribute with a value referencing to another instance | Task 3 | author, belongs to, leader, CEO |
| Hierarchy | A special type of relationship where a group of instances or class belongs to another class | Task 2 and Task 5 | Subclass of, instance of |
| Axiom | A logical expression to specify constraint in an ontology | Task 9 | "Every train departing from European location must arrive at another European location" |
| Rule | A conditional expression used to infer knowledge such as attribute values or relationships | Task 10 | "Every ship that departs from Europe is arranged by the company Costa Cruises" |

develop an ontology, but it does not consider when an ontology is built collaboratively. In the next section, we will discuss some methods and tools explicitly designed to support the construction of ontology collaboratively.

### 2.1.2 Methodologies for Collaborative Ontology Engineering

The emergence of Semantic Web has encouraged the practice of developing ontology in a community-driven manner. Methods and tools to support collaboration and contributions are needed. Simperl and Luczak-Rosch did a survey on collaborative ontology engineering methods that have emerged along with tools to facilitate the activities described in each methodology [51]. In this section, we will briefly discuss some of the mentioned methods and conclude on how do they relate to our work.

In [51], the methods that were discussed and compared are the Holsapple and Joshi method; Dogma-Dess; DILIGENT; Human-Centered Ontology Engineering Methodology (HCOME); and Ontology Maturing. Each of this method follows a participatory approach, emphasizing on collaborative effort with technological support that allows non-expert to participate in the ontology development.

The Holsapple and Joshi methodology consists of four phases: preparation, anchoring, iterative improvement and application. The preparation phase is similiar to the Methontology's specification phase, while the anchoring phase's objective is to produce a first version of the ontology. The collaboration happens in the iterative improvement phase where domain experts are asked for their feedback on the ontology and the consolidated results are implemented when there is a consensus among the experts.

The Dogma-Mess methodology are distinguished in 5 main phases: formulate vision statement; conduct feasibility study; project management; preparation and scoping; domain conceptualization; and application specification. This methodology does not describe on how to reach consensus on multiple view of ontology, but the collaboration happens in the form of knowledge negotiation which is a part of the conceptualization phase.

The DILIGENT is referred as an "argumentation-based" methodology based on IBIS argumentation model [54]. First, a core team create an ontology that is not required to be complete in the **build** phase. Then, this ontology is made available to users who can adapt it to their own local environment, while the original ontology remains unchanged. This second phase is called **local adaptation**. The **analysis** phase then involves an ontology engineering board to select which of the local branches' change to be carried to the next version of the original ontology. These agreed changes are implemented in the **revision** phase, resulting a new version of the shared ontology. Users may opt to align their local ontologies with the version in the **local updates** phase.

The HCOME, similar to DILIGENT, also focuses on distributed development of ontologies. Their approach is to distinguish between a personal information space, reflecting the view of an individual party, and a shared information space, reflecting the agreed conceptualization view of different parties. The collaboration is supported by allowing users to browse shared ontology, compare it to their own version, and discuss feedback on a moderated discussion thread.

Ontology Maturing [16] is a "Web 2.0" approach of ontology engineering, focusing on building lightweight ontology and views ontology engineering as an informal learning process. It takes from the Web 2.0 approach which empowers informal, lightweight, and easy-to-use aspect to ontology development. The methodology consists of 4 main phases: emergance of ideas; consolidation in communities; formalization; and axiomization. The core collaborative aspect lies in the first two phases where any participants can contribute terms by assignment of simple tags and then be consolidated by merging or splitting terms leading to a common terminology.

Overall, the mentioned methodologies have common challenges that they try to overcome: how to resolve differences of multiple versions of ontology produced by each participants viewpoint. They also take the iterative view of ontology life cycle: allowing the ontology to be improved at any point of time, thus producing multiple revised versions.

### 2.1.3 Collaborative Ontology Engineering Tools

In this section, we will discuss some tools (software; desktop and web application) used in order to support collaborative ontology engineering. The purpose of this section is to give an overview on how these tools work and what participation patterns they encourage that can later be translated to conversational interface for our work.

Ontology Development Environment (ODE) is the term used to refer to tools dedicated for ontology development and editing. The two most prominent tools are Protégé[1] and NeOn Toolkit[2].

To support collaboration, Protégé has an extended version called Collaborative Protégé[3] which has features for participants to interact and hold a discussion for conflict resolution. Multiple users can edit the ontology at the same time, and then use the discussion threads and direct chat channel to communicate their opinions. The discussion then can be followed by a voting, either with a five-star or agreement type, and the result will be in the form of annotation to the ontology.

NeOn Toolkit, on the other hand, is a set of Eclipse plugins dedicated for developing interconnected ontologies. Each plugin of the set is dedicated for a certain feature, from ontology editing, export and import feature, to an integrated system with a wiki system for collaboration. The wiki is dedicated as a part of the ontology engineering process, facilitating interaction between user to discuss on a certain ontology item (concepts and instances), and the result of the discussion can be transferred back into the ontology.

There are also dedicated wiki-based ontology engineering tools. In addition to being user-friendly, wiki system has gained attention in the Semantic Web field as it is used for the purpose of community-driven content curation focusing on collaborative interaction. Some examples of ontology engineering tools that adapted this Wiki system are myOntology [52], coefficientMakna [54], IkeWiki [47], and OntoWiki [27]. These Wiki-based tools present each instance and concept as a Wiki page, allowing user to annotate these pages with links with other pages, corresponding to relationships in the ontology, and adding property-value pairs, corresponding to attributes in the ontology.

There is also the Wikidata[4] project, which has its origin from Wikipedia[5], the largest peer-produced encyclopedia, with the purpose of building structured knowledge base allowing users to access and edit the data. Although it is not strictly an ontology editor, and more of a collaborative knowledge base itself, we will regard it as so, as the system, in a way, supports collaborative ontology engineering tasks (editing concepts, adding links, voting statements).

Müller-Birn et al. did a study on how Wikidata can be regarded both as a peer-production system and a collaborative engineering tool[39]. They analyzed the history of Wikidata edits, and group the edits as action sets.

Based on participation patterns, users of Wikidata can be categorized into 6 groups: Reference Editor, Item Creator, Item Editor, Item Expert, Property Editor, and Property Engineer.

Overall, the main challenge of developing an ontology collaboratively is how to reach consensus whenever there are multiple conflicting opinions from different contributors (both knowledge engineers and domain experts). In general, a contributor

[1]https://protege.stanford.edu/

[2]http://neon-toolkit.org/

[3]http://protegewiki.stanford.edu/index.php/Collaborative$_{protege}$

[4]https://www.wikidata.org/

[5]https://www.wikipedia.org/

can take part in the ontology development process by two type of actions: **suggesting** a creation, change, or removal of primitive entities (as discussed in Section 2.1.2) and **reacting** on a suggestion. The later action would depend on the agreement mechanism: it can be in the form of comments, voting, moderation, or a mix between them. For example, actions involving the primitive element class would be distinguished as follows:

- Suggesting
  - Suggesting a creation of concept/class
  - Suggesting a merge of two concepts
  - Suggesting a removal of a concept
- Reacting
  - Agreeing/disagreeing on a creation of concept
  - Agreeing/Disagreeing on a merge of two concepts
  - Commenting on the creation of a concept
  - Agreeing/disagreeing on a removal of a concept

The above example could also be applied to other primitive elements such as instances, relations, and attributes. In our work, we would derived our decomposition of KBC task upon the described two actions. Ultimately, the KBC task would be decomposed into two major subtasks corresponding to the two actions: the task of **creating** (suggesting) a primitive element (in our work, we would focus on instances) and assigning attributes and the task of **validating** (reacting) the created instances and attributes.

## 2.2 Crowdsourcing for Knowledge Base Construction

As with crowdsourcing research in general, crowdsourcing approach for knowledge base construction can also be divided into two types of approach: microtasking approach and Games with a Purpose (GWAP) approach. The microtasking approach tries to utilize microtask platform such as FigureEight and Amazon Mechanical Turk (AMT) to publish tasks for the crowd to work on. Usually this approach involves incentives in form of money which is paid to the crowdworkers based on the number of tasks they completed. On the other hand, the GWAP approach tries to avoid this monetary incentives by designing a game as such that crowdworkers, as players, will complete tasks for the intent of seeking entertainment.

Works focusing on crowdsourcing for KBC can be grouped based on which type of task of ontology engineering to be solved by crowdsourcing. The type of task is based on what ontology elements are involved and whether the crowdsourcing task is to create (collect) new elements or validate existing elements. As described previously, example of ontology elements are class, instance, attribute, relationship, and hierarchy.

A work by Simperl et al. proposes a framework utilizing crwodsourcing to support linked data management [50]. Linked Data is a way to publish structured data, so it is highly related to ontology and knowledge base. Simperl et al. use Amazon Mechanical Turk (AMT) to execute microtask involving identity resolution which is the task of deciding weather two resources (equivalent to instances) is the same with each other. This involves the creation of a sameAs links, which are equivalent to relationships in a knowledge base. On top of that, their work also involves metadata correction, the task of validating concept's attributes, and classification, related to the task of creating hierarchies.

Noy et al. instead focuses solely on the task of hierarchy-verification, presenting them to crowdworkers through AMT [42, 38]. Their main purpose is to find out the feasibility of crowdsourcing in ontology engineering by comparing the performance of students and AMT workers in answering hierarchy-verification questions. The research results that both groups perform similarly for verifying hierarchies of two ontologies: a business process model ontology and a general-purpose terms ontology. Additionally, they also investigate if the choice of the ontology affects the workers' performance, as well if crowdsourcing is suitable for ontologies in specialized domains. Their experiment results in high levels of accuracy achieved for common-sense ontologies such as WordNet, and fewer questions correctly answered than by domain experts for verifying hierarchies in specialized ontologies.

Crowdlink [11], utilizes crowdsourcing to create and validate triples in linked data. Triples is another term for links between concepts and instances in structured semantic data. AMT, is again used in this work as a platform to crowdsource tasks of creating and verifying properties (attributes), entity (instances), schema, and sameAs relationship in linked data. Their contribution is three-fold: the use of crowdsourcing to create links; a workflow management which is responsible to generate tasks based on existing ontologies; the implementation of their approach in Semantic Web.

Another line of work tries to incorporate crowdsourcing into existing ontology engineering workflow, specifically reolving around Protege as an ontology editor. Wohlgenannt et al. developed a plugin that allows ontology engineers to outsource ontology verification tasks [58]. The plugin supports the generation of relationship and hierarchy verification tasks right from within the Protege ontology editor. The tasks are generated through the uComp API. uComp [6] is a crowdsourcing platform facilitating knowledge acquistion tasks. It could flexibily allocate tasks to either GWAPs or mechanised labour platform such as CrowdFlower (now Figure8) and AMT.

Another work [21] tries to solve the problem of fine-grained entity type completion using crowdsourcing to aid the limitation of automatic machine-based algorithm. Entity type is equivalent to class, where an instance can be typed as a general type (such as SportPlayer, for example) and also as a more specific type (such as BasketballPlayer). Their work focuses on generating tasks that can be presented in crowdsourcing platform in order to complete the missing fine-grained type for entities in DBPedia[7], a large-scale knowledge base derived from Wikipedia.

There are several works which alternatively takes the GWAPs approach to crowdsource KBC tasks such as Climate Quiz [48], UrbanMatch [18], HIGGINS [29], and the Aparto Game [44]. Climate Quiz [48] is a web-based game for players to verify whether two concepts (e.g. "climate change" and "ecosystem") are related to each

[6]https://www.ucomp.eu/
[7]https://wiki.dbpedia.org/

other and what kind of relationship between the two terms. The game attracts player by leveraging social network, particularly Facebook.

UrbanMatch [18], on the other hand, is a mobile photo-coupling game. It presents the player photos of point-of-interests (POIs) and asks for their coupling. This couplings would be used to verify links between POIs and their respective images.

HIGGINS [29] is an attempt to combine Information Extraction and Human Computation, generating game questions to collect input from the players of relationships between entities as subjects and objects. They did an experiments using plots and character description from SparkNotes stories and movie articles from Wikipedia, resulting that HIGGINS is able to manage high-quality knowledge collection with low crowdsourcing costs.

The Aparto Game [44] focuses on harnessing the power of casual user to construct a knowledge base. They experiment with a knowledge base of a multilingual rental apartment Frequently Asked Questions (FAQ). The game itself is a part of the FAQ website, so users can post questions and answers regarding the apartment and also play the game. In the game, users are asked to answer apartment related quiz, collecting points for answering the quiz. The system then collects the answers from the players to update the knowledge base.

OntoPronto [55] is another GWAP, specifically a selection-agreement game where two players are asked to decide whether a Wikipedia article refers to a specific instance or to a class of entities. The players are also asked to categorize the entity refered by the article. The collection of answers from players eventually are used to create a general-purpose ontology. The GWAP approach used by OntoPronto is then compared to a similar settings but using microtask crowdsourcing through AMT.

Overall, works on crowdsourcing approach for constructing knowledge base is summarized in Table 2.2. The table groups the work based on the crowdsourcing approach used (microtask and GWAP), the ontology element involved, and the type of KBC task (either creating or validating tasks).

As for our work, we will base our system design upon the existing works that involves the following primitive elements: instance, attribute, relationship, and hierarchy with microtask as the crowdsourcing approach (blue-colored in Table 2.2). This will ensure that crowdsourcing approach for these primitive elements have been explored before, and our work will extend them as such with a conversational interface instead of a web-based interface. Though, for future work, we can see that there are opportunities of research here, where there are empty cells (gray-colored in Table 2.2) indicating types of task that have not been done for certain crowdsourcing approach and ontology primitives, at least as far as we know for now.

## 2.3 Conversational Interface and Crowdsourcing

Conversational interface, specifically the text-based one commonly called *chatbot*, is a type of user interface where a typical use case is carried through a series of text conversation mimicking that of with a human. Although chatbot is often referred as an AI-powered bot that is assumed to be able to interact with us as humane as possible, the term "Botplication" was introduced in [24]. The term refers to a category of conversational agent encouraging simplicity and effectiveness as replacement of tra-

Table 2.2: Summary of Works on Crowdsourcing for KBC (blue-colored cells highlight the scope of our work; gray-colored cells highlight approaches not yet explored)

| **Crowdsourcing Type** | **Element** | **Task Type** | **Approach** |
|---|---|---|---|
| Microtask | Class | Create | OntoPronto [55], CrowdLink [11], PROFIT[3] |
| | | Validate | uComp Protégé Plugin [58] |
| | Instance | Create | OntoPronto [55], CrowdLink [11] |
| | | Validate | Simperl [50], CrowdLink [11] |
| | Attribute | Create | CrowdLink [11], PROFIT [3] |
| | | Validate | Simperl [50], CrowdLink [11] |
| | Relationship | Create | Simperl [50], Crowdmap [45] |
| | | Validate | Crowdmap [45], Amini [5], uComp Protégé Plugin [58] |
| | Hierarchy | Create | [50], CrowdLink [11], Dong [21] |
| | | Validate | Noy [42] [38], CrowdLink [11], uComp Protégé Plugin [58] |
| | Axiom | Create | |
| | | Validate | |
| | Rule | Create | |
| | | Validate | |
| GWAP | Class | Create | OntoPronto[55] |
| | | Validate | |
| | Instance | Create | OntoPronto[55], Aparto Game[44] |
| | | Validate | |
| | Attribute | Create | |
| | | Validate | |
| | Relationship | Create | UrbanMatch[18], Climate Quiz[48], HIGGINS[29] |
| | | Validate | Kondreddi[30] |
| | Hierarchy | Create | |
| | | Validate | |
| | Axiom | Create | |
| | | Validate | |
| | Rule | Create | |
| | | Validate | |

ditional mobile applications. In our work, we are inclined more towards this type of chatbot to support KBC.

As our work investigates the potential of using conversational interface to support crowdsourced KBC, this section will discuss related work surrounding usage of crowdsourcing to support chatbots, ontology-based chatbots, and KBC using chatbot.

### 2.3.1 Crowdsourcing for Chatbot

Crowdsourcing approaches have been tried to be integrated to chatbot system. The basic idea is to incorporate human computation to the usual machine learning based and pattern-matching technology underlying the chatbot. The human computation module plays a role to decide what response would be appropriate to the user's input. Several implementations of this approach are Chorus [33, 32], Crowd-Intelligence-chatBot (CI-Bot) [36], and Evorus [28].

Chorus features a two-way natural language conversation between one end user and the crowd. The crowd in Chorus acts as a single agent. When a user inputs a query to Chorus, Chorus will forward it to a pool of crowd workers. Each worker would submit or vote on other worker's response. Once a response has reach consensus of agreement, the response would be forwarded back to the end user. In their test, Chorus correctly answered 84.62% out of the questions asked during conversations.

The difference between CI-Bot and Chorus is that CI-Bot would first tries to respond to end user automatically. Then if the question is beyond the knowledge of CI-Bot, it would forward the question to experts found by the expert recommender. The answers by the experts are collected and integrated to the final answer that would be sent to the end user. In a way, CI-Bot is a *hybrid* chatbot, incorporating both a corpus-based AI module and a CI module. The prototype of CI-Bot was built on top of the chinese messaging platform Wechat[8] and experiments were conducted to evaluate how effective CI-Bot is.

Another hybrid approach of crowd-powered and automated conversational assistant, Evorus [28] was built to be able to automate itself over time. The basic mechanism behind Evorus are three folds: (i) uses multiple chatbot and the system would learn how to select appropriate bots over time, (ii) reusing prior answers to similar queries, and (iii) automatically approve response candidates generated by the crowd workers. In Evorus, crowd workers can play a role to suggest a response (in addition to automatically generated responses) and also vote on candidate responses. Evorus is another case of hybrid chatbot, integrating the crowds and machine learning.

### 2.3.2 Ontology-based Chatbot

In this section, we will discuss several chatbot development approach where they incorporate the utility of ontology and/or knowledge base in order to generate response to end-user.

Augello et al. [8, 7, 9] built an intuitive chatbot using AIML (Artificial Intteligence Mark-Up Language) KB of Alice chatbot, DBPedia datasets, and Wikipedia repository. The chatbot exploits the knowledge from DBPedia by making SPARQL queries through AIML categories.

[8]https://www.wechat.com/

Another chatbot utilizing DBPedia is the DBPedia chatbot [6], aiming to optimize community interaction for answering recurrent questions. Rather than using AIML, the DBPedia chatbot utilize Rivescript to identify the intent of user's message. The chatbot handles three types of message: DBpedia questions, Factual questions, and regular Banter. For DBPedia question, they apply a rule-based approach by refering to DBPedia's mailing list. For factual question in the form of natural language, the bot utilizes combination of WolframAlpha and QANARY question answering system. While QANARY responds are already in the form of DBPedia URIs, WolframAlpha's response are still needed to be linked to DBPedia through DBPedia Spotlight. For banter, or casual conversation, a basic set of responses are used such as "Hi" or "What is your name" with the combination of the Eliza chatbot in case there is no applicable rule to a user's utterance.

There is also another ontology-based chatbot with specific purpose of being an argumentative agent of climate change. The chatbot, called ClimeBot [56], is hosted on the API.AI platform (now called Dialogflow[9]). The chatbot consist of two modules: an *ontology module* which answers question based on an ontoloty stored in OWL format and a *textual entailment module* in case the relevant knowledge cannot be found in the ontology. One of their main contribution is to incorporate OWL ontologies into the API.AI platform.

### 2.3.3 KBC using Chatbot

In this section, some chatbots whose purposes are to construct knowledge base are discussed. In contrast to the previous category of chatbot, where the ontology are used in order to support response generation, this section discusses category of chatbots specifically developed as an interface for KBC.

First, there are several ontology editors supporting the use of controlled natural language (CNL) to edit ontologies. CNL is a type of language used to hide the formal syntax and semantics of traditional ontologies. Some ontology editors supporting CNL are discussed in [1], and then there is AceWiki[31] and also Fluent Editor[49]. Although these editors are technically not chatbots, the way that they allow ontology editing using natural language is similar to a chatbot.

One of the most in-depth proof-of-concept for knowledge acquisition chatbot is Curious Cat [14]. Curious Cat is a context-aware crowdsourcing knowledge acquisition system using conversational interface. Curious Cat is developed as a mobile application with interface similar to messaging apps. The system was built based on Cyc, a knowledge base of common sense. It also uses the Cyc inference engine in order to answer user's question. Questions are mapped into a set of logical expressions and then answers are generated through the logical inference engine. The generated answers are translated back into natural language before sent back to the user. The context awareness is facilitated through the use of mobile sensors (e.g. location) in order to decide which user and what question to ask.

Our work differs from the work by Bradesko et al. because we are focusing on knowledge base construction starting from an almost empty knowledge base (only a partially defined ontology), while Curious Cat was developed on an already growing

[9]https://dialogflow.com/

knowledge base (Cyc). Additionally, Curious Cat heavily focuses on using context-based information to support acquisition of knowledge, while our work would focus more on defining KBC as microtasks and executing them in conversastional setting.

Similar to Curious Cat, KB-Agent [2] is also a chatbot with the purpose of completing the knowledge given an existing knowledge base. The chatbot tries to collect new entities and attributes through conversation with users. Put simply, KB-Agent works as follows: (i) first it generates a list of questions for each entity in a KB, (ii) KB-agent that ask questions to the users, (iii) finally, KB-agent identify entities in user's utterances and incorporate them into the KB. As far as our understanding, the system design of KB Agent does not incorporate any crowdsourcing workflow, thus the work did not explore much on how the chatbot users can contribute on ensuring the quality of the acquired knowledge. This is the gap that our work is trying to bridge, by incorporating a crowdsourcing workflow to the chatbot system to ensure the quality of the constructed knowledge base.

## 2.4 Summary

Up until now, numerous studies on how crowdsourcing are used for knowledge base construction and ontology engineering have been done. They mostly explored one of the two main approaches: either using a web-based microtask platform or a GWAP approach. Existing approaches used crowdsourcing to support part of KBC tasks to some extent, but not the whole knowledge base construction itself.

In the context of our work, we will mostly design our system upon the previously discussed existing works, specifically the following aspects:

**KBC Task Decomposition** We based our design of decomposing KBC Tasks on the methodologies discussed in section 2.1. The task decomposition would focus specifically on the conceptualization phase. In the case of which primitive ontology elements to be involved, the scope of our work would focus on the creation and validation of instances, attributes, and relationships.

**Crowdsourcing for KBC** Our work would also be a contribution to the state-of-the-art by extending existing works in the field of crowdsourcing for KBC. The extension tries to investigate the use of conversational interface as an alternative to the two crowdsourcing approaches in previous works: web-based microtasking and GWAP.

**Conversational Crowdsourcing** In the intersection of conversational interface and crowdsourcing, there are two main lines of work discussed above: crowdsourcing for chatbot and ontology-based chatbot.

Our work relates to the first line of work by using chatbot for crowdsourcing similar to the work of conversational microtasking by Mavridis et al [37]. Our work would targets microtasking specifically for KBC task.

In relation to the second line of work, instead of building a chatbot that uses an existing ontology or knowledge base, we intend to do the converse: building a knowledge base using a chatbot. Though, to some extent, the chatbot could continuously use the evolving knowledge base, this will remain as an opportunity for future work.

# Chapter 3

# Chatbot System Design for Knowledge Base Construction

In order to answer RQ2, this chapter will discuss the system design of the chatbot in detail. We first formalize how KBC process can be decomposed as microtask in our system. We then explain the crowdsourcing workflow to KBC using a set of the microtasks. After describing the microtask and crowdsourcing workflow, we describe how the architecture of the system as well as the conversation flows were designed.

## 3.1 KBC as Microtasks

As discussed in the Related Work (chapter 2), the KBC process is complex as it consists of several non-linear substeps such as identifying what to include into the KB, defining and classifying concepts, and determining relationships between concepts. With the complexity, it would be helpful if we define several small tasks that builds the whole KBC process. Defining these tasks would become the starting point before we discuss on how to execute these tasks using conversational crowdsourcing.

In this section, we will discuss how the process of KBC can be broken down into several small tasks (*microtasks*). This will be the basis on what types of KBC tasks to be presented in the conversational interface discussed in the later sections. First we will define several basic terms of KBC, then a task analysis would be described, before synthesizing the task types that would be used for our system.

### 3.1.1 Basic Terms of KBC

As KBC is a wide field of research and many literature refer to same concepts with different terms, we would like to explain the terms used in this thesis.

First of all, an **item** is a representation of either a real-world entity or a concept. If it represents a real-world entity (e.g. the Eiffel Tower), it can be referred to as an **instance**. A **concept** is a representation of a set of instances. For example, the Eiffel Tower (an instance) is an instance of a landmark (a concept). An item would have at least a **name** and optionally **description** and/or **aliases**.

Each item (either a concept or an instance) would have 0 or more **statements**. A **statement** consists of a **property-value** pair. A **property** is a descriptor of a data value. A property can be paired with one or more values; or even a special value such as "no value" or "unknown value".

A **relation** is a special case of statement depicting a connection between items. The target item of a relation becomes the value of the statement. A relation which depicts the property of *subclass of* or *instance of* is referred to as a **hierarchical relation**.

Each statement is accompanied by a **rank**, representing how probable that the statement is true. The rank is calculated based on the calculation of **upvotes** and **downvotes** from contributors.

### 3.1.2 Hierarchical Task Analysis

In this section, we attempt to decompose the task of constructing knowledge base using *hierarchical task analysis* (HTA) method. We carry the breakdown analysis based on synthesis of KBC methodology discussed in the related work chapter. Here, the task-breakdown is done to two major subtasks separately: the task of **constructing a knowledge base** (Table 3.1), and the task of **fixing and verifying a knowledge base** (Table 3.2). With each task-breakdown, we would also describe the task plan, explaining how each of the task should be executed.

#### HTA of Contructing a Knowledge Base

Table 3.1 depicts the task break-down of constructing a knowledge base. The task is carried under the assumption that there is no knowledge base created yet, e.g. the construction started from scratch.

The goal and plan for each task of constructing a knowledge base is as follows:

- Plan for **Task A**: Do each of **Task A.1**-**A.5** once in order. After that feel free to return to any task in any order to make the KB complete.
- Plan for **Task A.1**: Repeat **Task A.1.1** as many times as needed. This depends on how wide and specific the domain of the KB is intended. The goal of this task is to have a list of terms (keywords, or important words) within the the domain of the KB that could be potentially turned into concepts or instances in the next tasks.
- Plan for **Task A.2**: Do and repeat **Task A.2.1** and **Task A.2.2** in order until all concepts are identified. The goal is to define some concepts from the list of terms.
- Plan for **Task A.3**: Do and repeat **Task A.3.1** and **Task A.3.2** in order until all instances are identified. The goal is to define some instances from the list of terms.
- Plan for **Task A.4**: Do **Task A.4.1**, **A.4.2**, and **A.4.3** in any order or even in parallel. The goal is to have necessary relations between the defined concepts and instances.
  - Plan for each **Task A.4.1**, **A.4.2**, and **A.4.3**: Do each subtask in order (e.g. A.4.1.1-A.4.1.3) and repeat as necessary to assign all possible relations.

Table 3.1: HTA of Constructing a Knowledge Base

| | | | |
|---|---|---|---|
| **Task A**: Construct a knowledge base | | | |
| | **Task A.1**: Create a list of terms | | |
| | | **Task A.1.1**: Add a term to the list | |
| | **Task A.2**: Define concepts | | |
| | | **Task A.2.1**: Choose a term from the list as a concept | |
| | | **Task A.2.2**: Create a definition of the concept | |
| | **Task A.3**: Define instances | | |
| | | **Task A.3.1**: Choose a term from the list as an instance | |
| | | **Task A.3.2**: Create a definition of the instance | |
| | **Task A.4**: Assign relations | | |
| | | **Task A.4.1**: Assign relations between concepts | |
| | | | **Task A.4.1.1**: Choose two concepts |
| | | | **Task A.4.1.2**: Define relation |
| | | | **Task A.4.1.3**: Assign relation |
| | | **Task A.4.2**: Assign relations between instances | |
| | | | **Task A.4.2.1**: Choose two instances |
| | | | **Task A.4.2.2**: Define relation |
| | | | **Task A.4.2.3**: Assign relation |
| | | **Task A.4.3**: Assign relations between instances and concepts | |
| | | | **Task A.4.3.1**: Choose an instance and a concept |
| | | | **Task A.4.3.2**: Define relation |
| | | | **Task A.4.3.3**: Assign relation |
| | **Task A.5**: Assign statements | | |
| | | **Task A.5.1**: Choose an instance or a concept | |
| | | **Task A.5.2**: Add a statement | |
| | | | **Task A.5.2.1**: Define a property |
| | | | **Task A.5.2.2**: Set statement's property |
| | | | **Task A.5.2.3**: Define a value |
| | | | **Task A.5.2.4**: Set statement's value |

Skip A.4.1.2 (or A.4.2.2 and A.4.3.2) if the relation is already defined before, so it can just be reused and assigned immediately.

- Plan for **Task A.5**: Do **Task A.5.1** and **Task A.5.2** in order. Repeat as necessary for every instance. The goal is to have necessary statements describing the facts about each concept and instance.
  - Plan for **Task A.5.2**: Do **Task A.5.2.1 - A.5.2.4** in order. Repeat as necessary to assign all needed statements to the instance. Skip **Task A.5.2.1** and/or **Task A.5.2.3** if the property and/or value to be assigned has already been defined previously.

Note that **Task A.4** of assigning relations is actually a specific case of Task A.5 of assigning statements. A relation can be defined as a special statement, with which the property depicts the type of relation, and the value depicts another instance or concept.

Table 3.2: HTA of Fixing and Verifying a Knowledge Base

| | | | |
|---|---|---|---|
| **Task B**: Fix and verify a knowledge base | | | |
| | **Task B.1**: Fix and verify existing relation | | |
| | | **Task B.1.1**: Determine whether the relation is true or false | |
| | | **Task B.1.2**: Fix relation | |
| | | | **Task B.1.2.1**: Change relation |
| | | | **Task B.1.2.2**: Remove relation |
| | **Task B.2**: Fix and verify existing statement | | |
| | | **Task B.2.1**: Determine whether the statement is true or false | |
| | | **Task B.2.2**: Fix statement | |
| | | | **Task B.2.2.1**: Change statement (property and/or value) |
| | | | **Task B.2.2.2**: Remove statement |

### HTA of Fixing and Verifying a Knowledge Base

Table 3.2 depicts the task break-down of fixing and verifying an existing knowledge base. This major subtask is meant to be a continuation of the previous major task, with the goal of sustaining and maintaining the knowledge base. A knowledge base is assumed to be already constructed, but may contain invalid relation or statements. The task itself can be carried in parallel with the task of constructing the knowledge base.

The followings are the task plans for this task-breakdown:

- Plan for **Task B**: Do **Task B.1** and **Task B.2** in any order, and repeat as necessary for all relations and statements. The goal is to make sure the constructed knowledge base from Task A is correct.

- Plan for **Task B.1**: Do **Task B.1.1**. Only if the relation is thought to be false, do **Task B.1.2**. The goal is to make sure the chosen relation is correct. If it's deemed to be incorrect it should be changed or removed.

  – Plan for **Task B.1.2**: Either do one of **Task B.1.2.1** or **Task B.1.2.2**. Task B.1.2.1 of changing a relation can be done either by: changing the name of the relation, changing the target of the relation, or both.

- Plan for **Task B.2**: Do **Task B.2.1**. Only if the statement is thought to be false, do **Task B.2.2**. The goal is to make sure the chosen statement is correct. If it's deemed to be incorrect it should be changed or removed.

  – Plan for **Task B.2.2**: Either do one of **Task B.2.2.1** or **Task B.2.2.2**. Task B.2.2.1 of changing a statement can be done either by: changing the property of the statement, changing the value of the statement, or both.

In this section of task analysis, constructing, fixing and verifying a KB involves only a single individual carrying the task. When more than one contributors are involved, each contributor can carry each task. In order to resolve conflicts between opinion (e.g. contributor A thinks a statement is true while contributor B thinks otherwise), we can apply a voting mechanism, where all contributors can give an upvote or downvote of relation and/or statements.

For our work, we would focus on the following types of KBC microtasks derived from our HTA. We choose the task granularity on the second level from the hierarchical task analysis, and give them a new numbering (T1, T2, and so on) so we can refer it in the later sections. Note that we chose a limited number of the microtasks as the scope of our thesis, but we acknowledge that there are potential future work to explore how the other defined microtasks can presented in a conversational crowdsourcing system. We describe each microtask type briefly as follows.

**Creating a definition of an instance (T1)**. This task corresponds to Task A.3 from our HTA. In this task, worker creates an instance representing a real-world object. An example would be creating an item with the name "Lecture Hall Ampere" that represents a lecture hall in TU Delft campus. For our system, we wouldn't include Task A.1 and A.2 as we would have a predefined concepts (categories), and would focus on creation of instances. This is, of course, would be a potential future work to include the task of concept definition using a crowdsourced conversational system.

**Assigning a relation between instances (T2)**. This task corresponds to Task A.4.2 from our HTA. Given an instance and another instance (or list of instances), the worker has to specify the type of relation (if there is any) between the two instances. For example, a worker chooses the relation "building in which it is located at" between the instance "Lecture Hall Ampere" and the instance "EWI Building".

**Assigning a relation between an instance and a concept (T3)**. This task corresponds to Task A.4.3 from our HTA. Given an instance and a concept, the worker has to specify a relation (if there is any) between the instance and the concept. For example, a worker assigns the relation "categorized as" between the instance "Lecture Hall Ampere" and the concept "Lecture Hall".

**Adding a statement (T4)**. This task corresponds to Task A.5.2 from our HTA. Worker is given an already created instance and are asked to complete a statement regarding the instance. For example, the worker is given the instance "Lecture Hall Ampere" representing a place and is asked to complete the statement about the floor number of the place.

**Determining whether a relation is true or false (T5)**. This task corresponds to Task B.1.1 from our HTA. Worker is given with an already defined relation and they have to assess whether the relation is correct or not. For example, the worker is asked to verify if "Lecture Hall Ampere" can be categorized as "Lecture Room".

**Determining whether a statement is true or false (T6)**. This task corresponds to Task B.2.1 from our HTA. Worker is given with an already defined statement of an instance, and they have to assess whether the statement is correct or not. For example, the worker is asked to verify the statement "Lecture Hall Ampere is located on the first floor".

## 3.2 Crowdsourcing Workflow

In this section, we will describe the workflow on utilizing the crowd in order to construct a knowledge base. The workflow is designed based on the defined microtasks

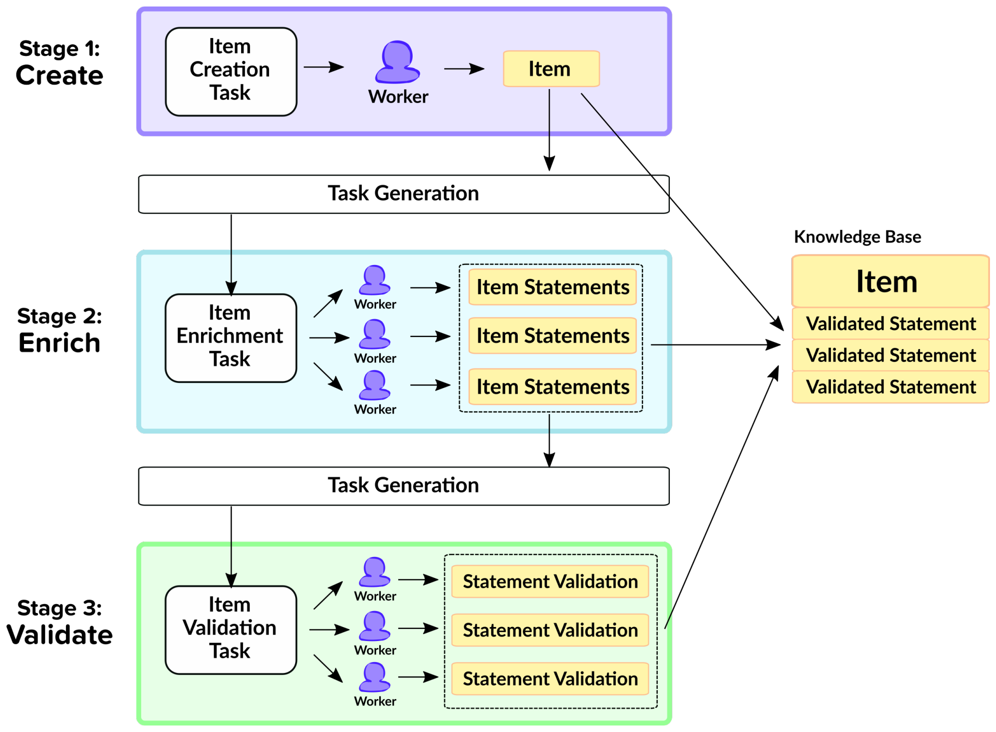


Figure 3.1: Crowdsourcing Workflow for Producing an Item for a Knowledge Base

from the previous section and can be divided into three stages: **Create**, **Enrich**, and **Validate**. This workflow would later be used by our system to enable crowdsourced construction of knowledge base. Figure 3.1 illustrates an overview of the stages to produce a single item for a knowledge base. In each of the stages, one or more workers executes some tasks to produce input for the item. We will now elaborate the details of each stage.

### Stage 1: Create

In the Create stage, a worker is assigned to an **Item Creation Task**. This Item Creation Task is a task consisting of multiple microtasks. Each of the microtasks can be of type **T1**, **T2**, **T3**, or **T4**. The output of this stage is for a worker to create an initial definition of an item representing a real-world object. The following is a list of example tasks that a worker might need to complete in this stage:

1. Creating an item with the name "Lecture Hall Ampere" (microtask type **T1**).
2. Assigning a relation 'building in which it is located at' between "Lecture Hall Ampere" and the "EWI" building (microtask type **T2**).
3. Assigning a relation 'categorized as' between "Lecture Hall Ampere" and the category "place" (microtask type **T3**).
4. Adding a statement "located on the 1st floor" for "Lecture Hall Ampere" (microtask type **T4**).

After the Item Creation Task is completed by the worker, thus producing an initial definition of an item, the workflow then proceeds with a task generation process done by our system. The task generation process would take an item as an input and produces a task to be executed in the next stage.

### Stage 2: Enrich

After an item is produced from the previous stage, a corresponding **Item Enrichment Task** would be generated. This task would be assigned to a defined number of workers other than the worker who created the item in Stage 1. These workers would add more information (in the form of statements) to the created item by executing the Item Enrichment Task that consist of multiple microtasks of type **T2**, **T3**, and **T4**. The microtasks contained in this stage can be similar with the ones from Stage 1, as the purpose of this stage is to complete missing statements which were not acquired in Stage 1. The following is a list of example tasks that a worker might need to complete in this stage:

1. Assigning a relation 'building in which it is located at' between "Lecture Hall Ampere" and the "EWI" building (microtask type **T2**).
2. Assigning a relation 'categorized as' between "Lecture Hall Ampere" and the category "place" (microtask type **T3**).
3. Adding a statement that "Lecture Hall Ampere" has electricity outlets for students (microtask type **T4**).
4. Adding a statement about the seat capacity of "Lecture Hall Ampere" (microtask type **T4**).

When the number of workers that have completed the task satisfies the defined number of answers required for each microtask, a new task would be generated to validate the statements in the next stage.

### Stage 3: Validate

In this stage, an **Item Validation Task** is generated by considering the answers from workers completing the item creation task and item enrichment task from the first two stages of the workflow. The goal of this stage is to ensure that the statement inputted regarding an item is valid to some extent. The Item Validation Task again would be assigned to a defined number of workers. These workers have to validate the statements gathered from stage 1 and stage 2. The task would consist of multiple microtasks of type **T5** and **T6**. A list of example tasks in this stage is as follows:

1. Verifying that "Lecture Hall Ampere" can be categorized as "Lecture Room" (microtask type **T5**).
2. Verifying that "Lecture Hall Ampere" is not located on the 1st floor (microtask type **T6**).

Finally, when the number of workers who completed the Item Validation Task satisfies the defined number of answers required, the final step of the workflow would be to aggregate the item, statements, and validated statements so it will produce an item along with relations and statements as a part of a knowledge base. All three stages would be executed numerous times for each newly created item, and ultimately the produced knowledge base would evolve with more items along with their validated statements and relations.

## 3.3 System Architecture

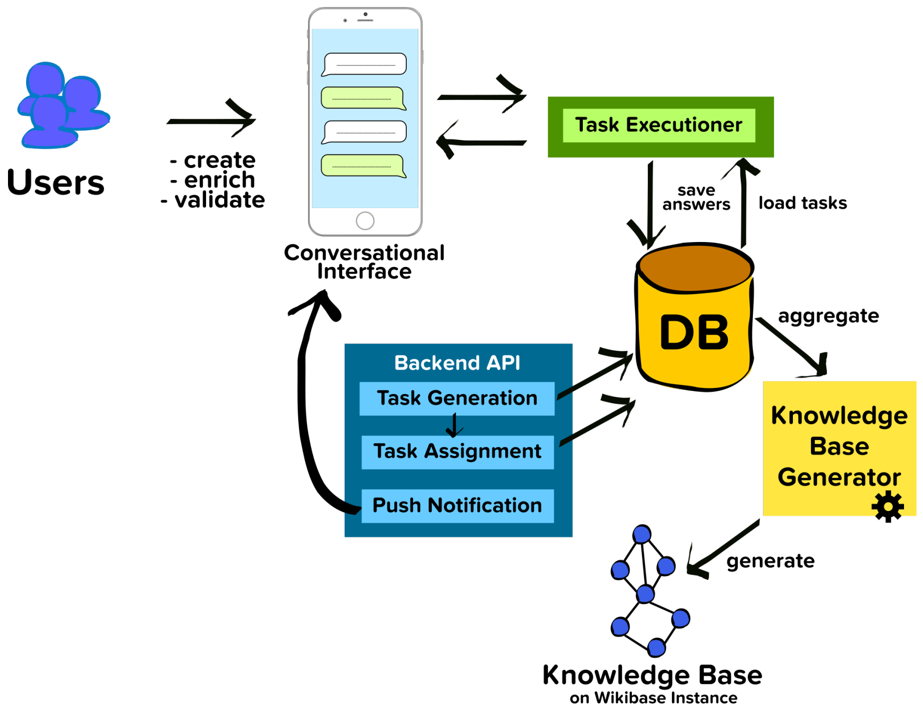


Figure 3.2: System Architecture Overview

We first describe the overall architecture of the system supporting the chatbot. Figure 3.2 illustrates the overview of the system. The main modules of the system consists of the **Conversational Interface**, the **Task Executioner** module, the database, the **Backend Application Programming Interface (API)** as the back-end layer, and the **Knowledge Base Generator** module. The Backend API consists of submodules to handle Task Generation, Task Assignment, and Push Notification.

### 3.3.1 Database

As we will refer to the collection in the database often throughout the following description of the system architecture, we will first briefly describe how our data model is structured.

We decided to use a NoSQL document database, instead of relational database, as it will allow us to flexibily define entity's properties without fixing the schema. So in the future, if we want to add more types of information to be stored for certain use case, we will be able to easily adjust the data model.

The complete entity relationship diagram is available in Appendix A. We will briefly explain each entity in this section.

### Users

First, we have the **users** collection, saving the data of our users. Data of each user is stored as a document in this collection. We store their basic data such as their `userId`, inferred from their Messaging platform account ID, and `createdAt`, which is the timestamp of when they started to chat with the chatbot. For logging purpose, we would also store the conversations between the user and the chatbot in the subcollection `utterances` of each user document.

### Items

Next, we have the **items** collection. This collection consists of documents of saved answers when user completed a Create task. Some attributes stored for each item would be the `author` attribute to indicate which user created an item; the `createdAt` attribute reprsenting when the item was created; and the `executionStartTime` attribute representing when the user start working on the corresponding create task.

### Tasks

A **task** document has a reference to an item that needs to be completed either by adding more information (an item enrichment task) or validating existing information (an item validation task). A task would have the following attributes: the id of the item (`itemId`), number of answers required (`numOfAnswersRequired`), the type of task (`type`; whether it is an enrichment or validation task), and optionally an expiration date (`expirationDate`).

### Task Instances

Every task would be assigned to several users. The assignment is done by generating a task instance document that links to the task document. Each of this task instance would then be stored as a subcollection **taskInstances** under the corresponding assigned user.

### Task Templates

Questions for each task type (create task, enrich task, and validate task) are stored in a task template document in the **taskTemplates** collection. For example, the questions for create task for items with type place would be saved as a document with id `create-place` in the taskTemplates collection. We will discuss more about how task template works in section 3.3.3.

#### Enrichments and Validations

While the answers from users who completed a create task is stored in the items collection, for enrichment and validation task, they would be stored in the **enrichments** collection and **validations** collection respectively. The documents stored in this collection have references to the completed task instances.

### 3.3.2 Conversational Interface

The **Conversational Interface** module is responsible to handle interaction between the user and the chatbot itself. There are three main user-initiated interactions that it would handle: when user starts the conversation (**Start Handler**); when user asks a list of task (Task List Handler); and when a user choose to start working on a selected task.

#### Start Handler

When a user starts a conversation with the chatbot, the Start Handler will first check if the user is a first time user. If they are, the start handler will save the user data to the users collection for future identification. The Start Handler then will send instruction for the user to choose the available item types. Types of item available are dependent on the domain of the knowledge base, in which for our work we will offer 4 item types (food, place, course question, trash bin), discussed in more detail later in Chapter 5.

#### Task List Handler

When a user selects one of the available item types, the Task List Handler will fetch available task instances assigned to the user from the database. It will then send a list of messages representing each of the available task instances. The user can then response to select which of the task they want to work on. This module will then instantiate a Task Executioner which will be discussed in more details in section 3.3.3.

#### Help Menu Handler

Additionally, we would also have a Help Menu Handler module which simply sends a text message explaining how to use the chatbot whenever a user asks for help.

### 3.3.3 Task Executioner

The task executioner module is responsible to handle the flow when a user is working on a task, starting from the user selecting a task instance until the user submits their answers. After a user selects a task instance, an instance of a task executioner will be spawned specifically for that user. Figure 3.3 shows the high level overview of how the task executioner works.

First, the task executioner instance will load the task template corresponding to the item type and the type of task (create, enrich, or validate) the user selected. The task template for each task type are stored in the `taskTemplates` collection in the database. Table 3.3 describes the main fields that should be stored in a task template and Figure

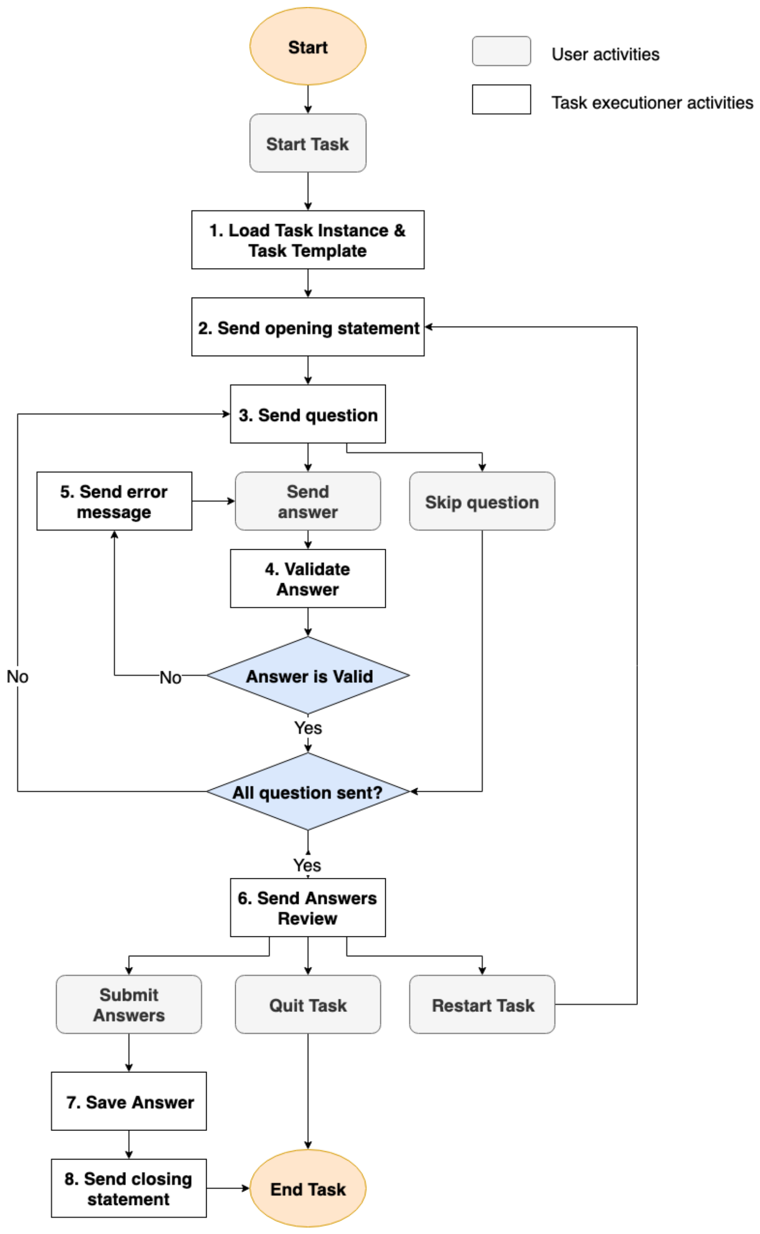


Figure 3.3: High Level Overview of Task Executioner Logic

Table 3.3: Task Template Fields

| **Field** | **Description** | **Example** |
|---|---|---|
| `openingStatements` | A list of text messages or images sent by the bot to the user when they start working on a task. | Here's the place that needs to be verified: |
| `closingStatements` | A list of text messages or images sent by the bot to the user when they finish working on a task and after they submitted their answer. We can also include data submitted from the user in the statement (e.g. to include name of the item, use the item[name] placeholder). | Thank you! We have updated the information about *item[name]* |
| `entryCommand` | The text command that should be sent by the user to trigger start of a task. In our system this defaults to "create" for create task. The task template will also accepts custom commands if this field is set. | create |
| `questions` | Stores a list of questions that will be sent by the bot. | *See types of question (Table 3.4) for examples.* |

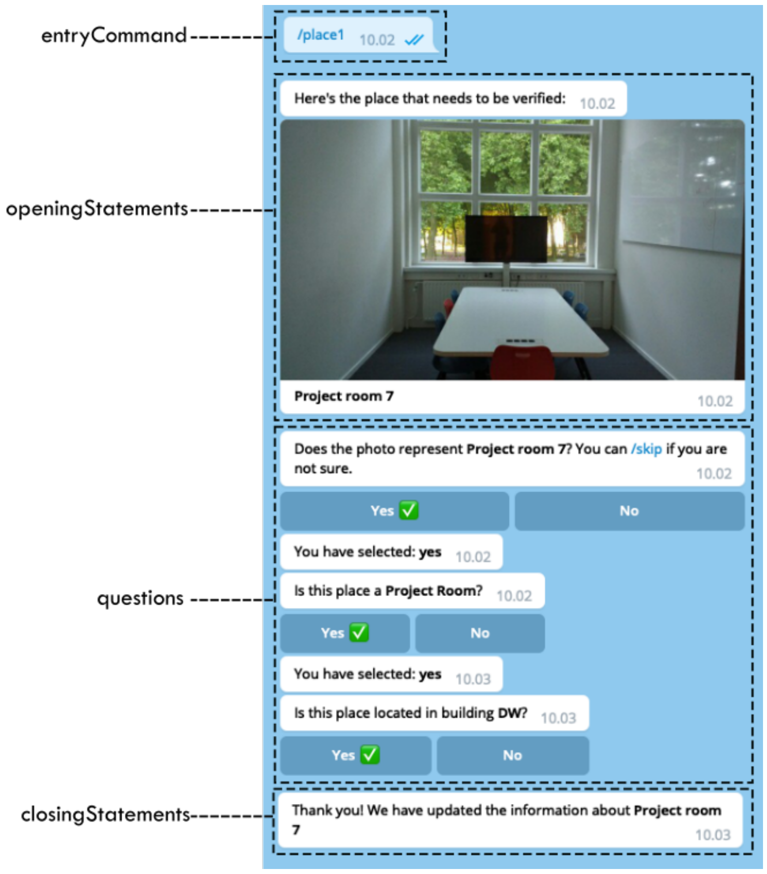
3.4 highlights how the corresponding elements of a task template are presented in the conversational interface.

If the type of the selected task instance is an enrichment or validation task, the task executioner will also load the data of the item corresponding to that task instance. After that, the task executioner will start the conversation and guide the user to complete the task. The task executioner will prepare each message that will be sent to the user based on the loaded task template and the data item.

First, it will send message built from each string in the `openingStatements` field. Then, it will send each question from the questions field, waiting the user's response for each question before moving on to the the next one. A question in which the answer is not required can be skipped by the user and the task executioner shall move on to the next question. User can also quit and abandon the task whenever they want, in which the task executioner will terminate.

Several types of question can be defined in the task template and it will determine how the Task Executioner will send the question and what kind of answer is accepted for the question. Table 3.4 shows the type of question that our chatbot would support. Figure 3.5 illustrates examples on how each question type would be presented in the chatbot.

There are three questions types which falls within the **"Multiple Choice" (MC)** question type: **MC Categorization**, **MC Item**, and **MC Custom**. We differentiate



/place1 10.02

Here's the place that needs to be verified: 10.02

Project room 7 10.02

Does the photo represent **Project room 7**? You can /skip if you are not sure. 10.02

You have selected: **yes** 10.02

Is this place a **Project Room?** 10.02

You have selected: **yes** 10.03

Is this place located in building **DW?** 10.03

Thank you! We have updated the information about **Project room 7** 10.03

Figure 3.4: Task Template Elements

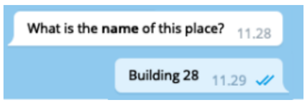
these question types based on their allowed input and choices generation. An MC Categorization question populate the choice options based on pre-defined categories for an item type. An MC Item question, on the other hand, show a list of certain item type (e.g. building items) based on certain criteria (e.g. location). Additionally, this type of question also allows user to give free text as a custom input if their answer is not listed. Finally, an MC Custom question has a fixed set of answers (e.g. "Yes", "No", and "Not Sure" options). Furthermore, later we would be having two variations of interaction style for these three question types for experimental purpose: whether to show a set of buttons (as shown in Figure 3.5f, 3.5g, and 3.5h) or to show a list of possible answer along with numbers corresponding to each answer (see section 5.2.3).

When the user finishes answering all questions, a question with the type of **Answers Confirmation** shall be sent by the task executioner. This question allows the user to either: submit their answers; start over the task; or quit the task. Although, the ideal feature to allow users editing their answer is to enable user to go back to any of the questions; in this thesis we simplify the feature in order to have a better control of experimental settings.

After the user submitted the answers, the task executioner will save the answers to the database. Then, it will notify the Backend API that a new answer has been submitted, so the API can generate new task accordingly as discussed in the Crowdsourcing

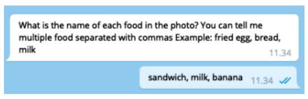
Table 3.4: Question Types

| **Question Type** | **Type of Message(s)** | **Answer Type** |
|---|---|---|
| Text | Text Message | Text |
| Multiple Input | Text Message | Comma-separated text |
| Numeric | Text Message | Text (Numeric Only) |
| Location | Text Message, A button to Send Current Location | Location |
| Image | Text Message | Image File |
| MC Categorization | Text Message, Buttons of Category Options | Selected Option |
| MC Item | Text Message, Buttons of Item Options | Selected Option or Text |
| MC Custom | Text Message, Buttons of Custom Options | Selected Option |
| Answers Confirmation | Text Message, Buttons to "Submit", "Restart", or "Quit" Task | Selected Option |


What is the name of this place? 11.28

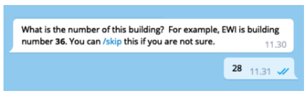
Building 28 11.29

(a) Text Question


What is the name of each food in the photo? You can tell me multiple food separated with commas Example: fried egg, bread, milk 11.34

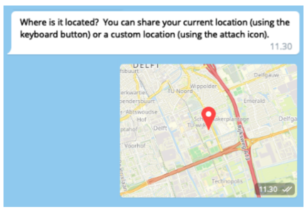
sandwich, milk, banana 11.34

(b) Multiple Input

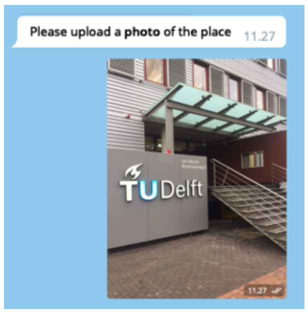

What is the number of this building? For example, EWI is building number **36**. You can /skip this if you are not sure. 11.30

28 11.31

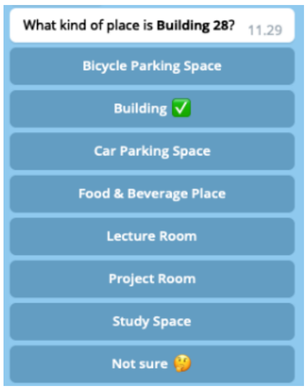
(c) Numeric


Where is it located? You can share your current location (using the keyboard button) or a custom location (using the attach icon). 11.30

(d) Location


Please upload a **photo** of the place 11.27

(e) Image


What kind of place is **Building 28?** 11.29

Bicycle Parking Space

Building ✅

Car Parking Space

Food & Beverage Place

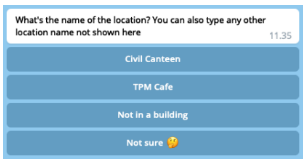
Lecture Room

Project Room

Study Space

Not sure 🤔

(f) MC Categorization


What's the name of the location? You can also type any other location name not shown here 11.35

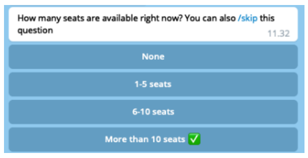
Civil Canteen

TPM Cafe

Not in a building

Not sure 🤔

(g) MC Item


How many seats are available right now? You can also /skip this question 11.32

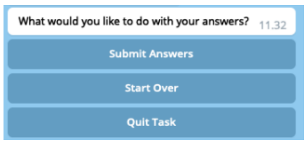
None

1-5 seats

6-10 seats

More than 10 seats ✅

(h) MC Custom


What would you like to do with your answers? 11.32

Submit Answers

Start Over

Quit Task

(i) Answers Confirmation

Figure 3.5: Example of Each Question Type

Workflow (Figure 3.1).

### 3.3.4 Backend API

The Backend API acts as our backend module consisting of three submodules: **task generation**; **task assignment**; and **push notification**. Endpoints to trigger task generation, task assignment, and push notification would be exposed as REST Endpoints, such that it can be triggered by the chatbot, the mobile app, or by a cron schedule.

#### Task Generation

The task generation process happens in between stages of our crowdsourcing workflow as discussed in Section 3.2. Two types of generation is handled by the task generation module: the generation of item enrichment tasks that happens between the Create stage and Enrich Stage and the generation of item validation tasks that happens between the Enrich Stage and Validate Stage.

A new item enrichment task would be generated every time a new item is created when the chatbot user completes an item creation task. The output of this task generation would be a new task document in the `tasks` collection that has the field `item` referencing to the newly created item.

For item validation tasks, they are generated whenever a sufficient number of answers of an enrichment task have been submitted. This number is defined in the task in the `numOfAnswersRequired` field, which we set to 5 for our evaluation setting described in Chapter 5. Every time a user completes an enrichment task, the task generation module will check the total number of answers, and proceed by generating the corresponding validation task if the number is more than or equal to 5.

#### Task Assignment

After either an item enrichment task or an item validation task is generated, the task assignment module would assign this new task to some existing users. The assignment is done by creating a `taskInstance` document in the `taskInstances` subcollection under each user.

For each task, the number of users to be assigned is determined as three times the number of answers required. We also prioritize users who have previously worked on similar tasks based on a predefined similarity depending on item types. For example, for a task corresponding to a place, we would group the places based on their location, so a task about a place in a library would be prioritized for assignment to users who previously has completed a task about another place in that same library.

#### Push Notification

As we want our users to spend time on completing task every day, we included a push notification module to ask the user whether or not they want to complete another task.

The module is a part of the Backend API and would be implemented as an endpoint that would be hit twice a day (for our experimental settings it would triggered at 10.00 and 15.00). The endpoint will send a message to each of all chatbot users who are not currently in the middle of completing a task.

After receiving the message, the user can response to indicate if they would want to complete another task or would like to do it later.

The module should also remove previously pushed notification to avoid clutter of several messages in a row. Additionally, this module would also keep track of users who has blocked the chatbot.

### 3.3.5 Knowledge Base Generation

As the main problem this thesis is trying to solve is to construct a knowledge base, we include a module which is responsible to aggregate the answers from our user to a more known knowledge base structure. We would call this module as the **Knowledge Base Generation** module. Figure 3.6 shows the high level overview on how the module works.

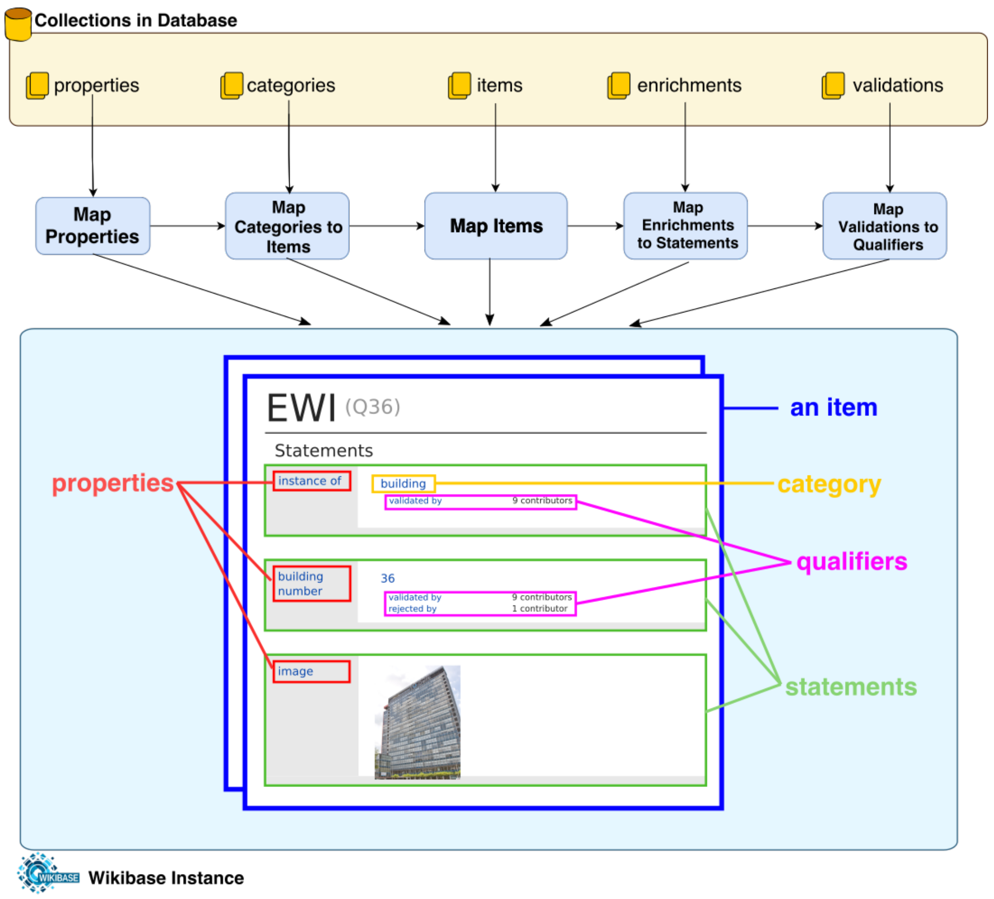


Figure 3.6: Knowledge Base Generation Module

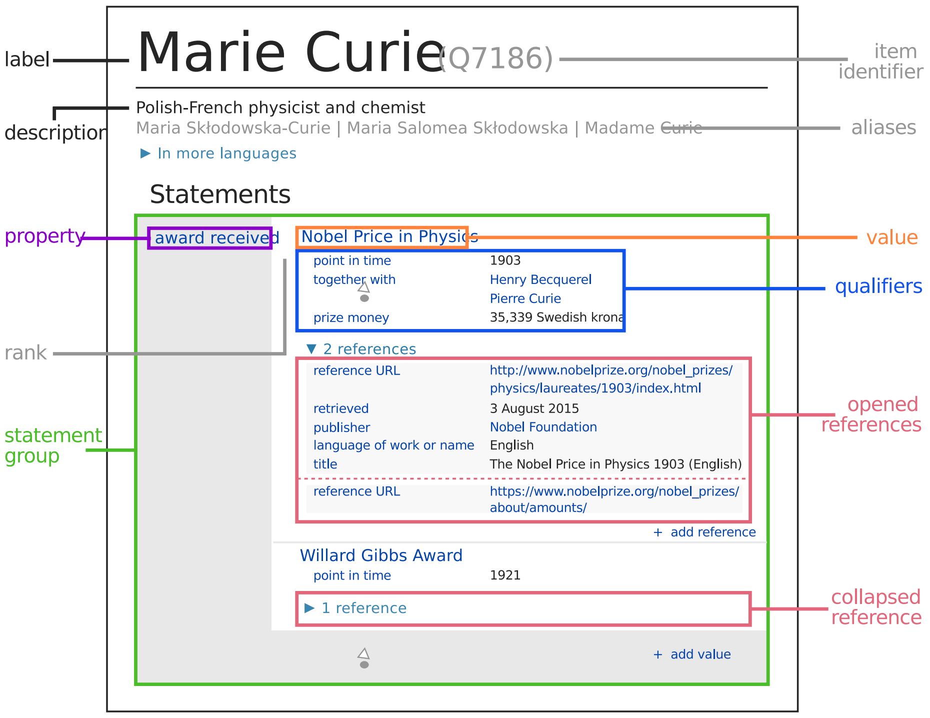
#### Wikibase Data Model

In our proof-of-concept we intend to store the aggregated collected answers into the data model used by Wikidata, a free and open collaborative knowledge base initiated by the Wikimedia Foundation [57]. Wikidata itself is an instance of Wikibase[1], the software which enables access and store data in a structured repository. Wikibase allows anyone to run their own KB along with a SPARQL endpoint to access the KB and a Wiki as the client to view, explore, and edit data.

The data in Wikibase is stored using the Wikibase Data Model[2] structure. In this data model, entity which can be both conceptual or phyisical are represented as an **Item**. For example, in Figure 3.7, Marie Curie is an Item represented by an **item identifier** of Q7186[3] in Wikidata.

An item would have several information stored with it such as a **Label**, a **Description**, and optionally some **Aliases**. An item would also have a list of **Statements**. Statements are responsible to represent factual data about the item. For example, here we have statements stating that Marie Curie had received the Nobel Price in Physics Award and Willard Gibbs Award. Here, the statement consists of a **property** "award received" and two values. Here, both the property itself and the values are also items.

Supporting the statements are a list of optional **qualifiers** and **references**. Qualifiers are meant to add additional facts to the main statement. For example in Figure 3.7, Marie Curie received the Nobel Price in Physics in 1903. The additional fact is the point of time on when the award was given, which is stored as a qualifier to the main statement. References, on the other hand, are publication and/or links to source which supports the fact represented by the statement.



Marie Curie (Q7186)

Polish-French physicist and chemist

Maria Skłodowska-Curie | Maria Salomea Skłodowska | Madame Curie

▶ In more languages

Statements

award received

Nobel Price in Physics

| | |
|---|---|
| point in time | 1903 |
| together with | Henry Becquerel |
| | Pierre Curie |
| prize money | 35,339 Swedish krona |

▼ 2 references

| | |
|---|---|
| reference URL | http://www.nobelprize.org/nobel_prizes/physics/laureates/1903/index.html |
| retrieved | 3 August 2015 |
| publisher | Nobel Foundation |
| language of work or name | English |
| title | The Nobel Price in Physics 1903 (English) |
| reference URL | https://www.nobelprize.org/nobel_prizes/about/amounts/ |

+ add reference

Willard Gibbs Award

| | |
|---|---|
| point in time | 1921 |

▶ 1 reference

+ add value

Figure 3.7: Wikibase Data Model, from http://w.wiki/32q

We decided to use Wikibase with the following plus points in mind:

[1]https://www.mediawiki.org/wiki/Wikibase

[2]https://www.mediawiki.org/wiki/Wikibase/DataModel

[3]https://www.wikidata.org/wiki/Q7186

- Wikibase allows us to use *bot* to create and edit items. This is an advantage to our intent on automating the aggregation phase of syncing our database to the Wikibase instance.
- It maintains history of changes. In this case, all edits to the knowledge base are recorded thus we can better analysis the edit activity corresponding to the user activity in KBC process by using the chatbot.
- The data model used by Wikibase is not rigid, especially with qualifiers and references which allows another level of clarification of a factual information. This enables us to adjust the usage of qualifiers for our system to represent the answer of validation tasks; depicting how many users approved or rejects a statement.
- Compatibility with Wikidata is also a great opportunity for future work to link the knowledge base constructed by using the chatbot to the Wikidata knowledge base.
- Wikibase has a similar vision to our thesis vision which is to promotes broader participation of KBC process. We can see this as a future opportunity to integrate the chatbot-based system with the Wikibase technology for constructing a knowledge base.
- Exporting the Wikibase data to the Resource Description Framework (RDF) format, which is a well-known standard model specified by World Wide Web Consortium (W3) for Semantic Web, is also supported [22]. This compatibility means that we can easily export the data to the Linked Data Web.

#### Properties Mapping

As we would like to start with a blank Wikibase, without any item or properties, we started defining properties for our campus knowledge base by storing it in our database. The Knowledge Base Generation module would then import this properties to our Wikibase instance. The complete list of properties is available in Appendix C.

#### Categories Mapping

For our domain use case, we would make a predefined list of categories so each created item can be categorized. This range from categories of places (study space, lecture room, and so on). Completed list of categories is available in Appendix C.

The module is then responsible to import this categories as items in Wikibase instance. This categories would then be values of "instance of" statement that would indicate category of created items.

#### Items Mapping

To avoid confusion, we would refer items stored in our the database as **Database Item**, while item in Wikibase instance as **Wikibase Items**.

Each database items would be mapped as wikibase items. It means that we would be able to view them as Wiki page. Properties of the item (such as image, gelocation, and so on) collected from users would then be included as statements.

### Enrichments Mapping

Enrichments from the database would be mapped as statements of the corresponding Wikibase item. The module would be responsible to map the property name to the corresponding Wikibase properties. Values are also adjusted depending on its data type.

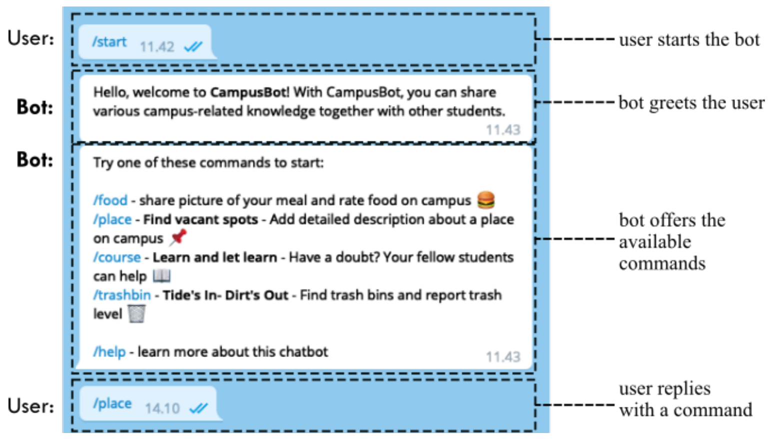
### Validations Mapping

As validations in our database represents whether a user approves or rejects a certain information about an item, we would need to represent this in Wikibase instance. We designed the module such that it maps validations into *qualifiers* to the corresponding statements. Two types of qualifiers are used: to represent approval we use "validated by", and to represent rejection we use "rejected by". The value of these two fields would be the number of users who gives their validation.

## 3.4 Conversation Flow

While previous section focuses on the system architecture, this section will focus on how the chatbot should carry the conversation with the users. We work on the conversation design iteratively, presenting the conversation in mockups, and improving the mockups by considering the potential problems that user might encounter when using a chatbot application. We enforce guided conversation into our chatbot design, suggesting actions and choices to the user in order to finish a task. This kind of design is a typical characteristic in chatbots that have specific goal in mind [24], in our case: user completes a KBC task.

### 3.4.1 Start Flow


User: /start 11.42

Bot: Hello, welcome to **CampusBot!** With CampusBot, you can share various campus-related knowledge together with other students. 11.43

Bot: Try one of these commands to start:

/food - share picture of your meal and rate food on campus 🍔
/place - **Find vacant spots** - Add detailed description about a place on campus 📌
/course - **Learn and let learn** - Have a doubt? Your fellow students can help 📖
/trashbin - **Tide's In- Dirt's Out** - Find trash bins and report trash level 🗑️

/help - learn more about this chatbot 11.43

User: /place 14.10

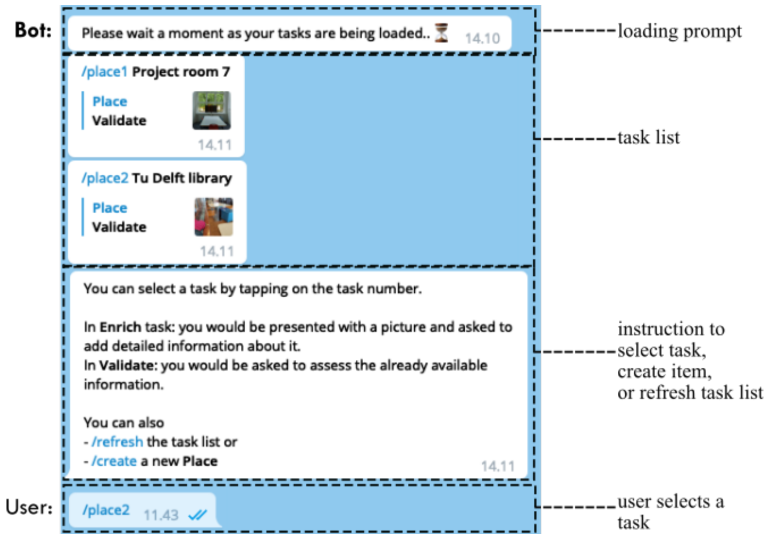
user starts the bot
bot greets the user
bot offers the available commands
user replies with a command

Figure 3.8: Start Flow

When the user starts a new conversation with the chatbot by tapping the Start button which triggers the command `/start`, the chatbot would send a message to give the user the options to choose among the available item types. When the user types or clicked on the corresponding command of a use case, the chatbot will then switch to the Task List Flow for the chosen use case. Figure 3.8 shows a sample dialog between the first-time user and the chatbot. In this example, the user types in the `/place` command which will lead the chatbot to the Task List (see section 3.4.2 for the place use case.

In addition, the user can also type in the '/help' command in which the chatbot would reply with a more detailed description of the chatbot.

### 3.4.2 Task List Flow


Bot:

Please wait a moment as your tasks are being loaded.. ⌛ 14.10

/place1 Project room 7
Place
Validate
14.11

/place2 Tu Delft library
Place
Validate
14.11

You can select a task by tapping on the task number.

In **Enrich** task: you would be presented with a picture and asked to add detailed information about it.
In **Validate**: you would be asked to assess the already available information.

You can also
- /refresh the task list or
- /create a new **Place**
14.11

User:

/place2 11.43

loading prompt
task list
instruction to select task, create item, or refresh task list
user selects a task

Figure 3.9: Task List Flow

After the user selects one of the item types, the chatbot would send consecutive messages depicting a list of available tasks instances for the user. Figure 3.9 shows a sample dialog where the bot sends the available tasks and explains how to select a task.

The user can select one of the task by typing in the task number command at which the chatbot would start Task Execution flow for the corresponding task.

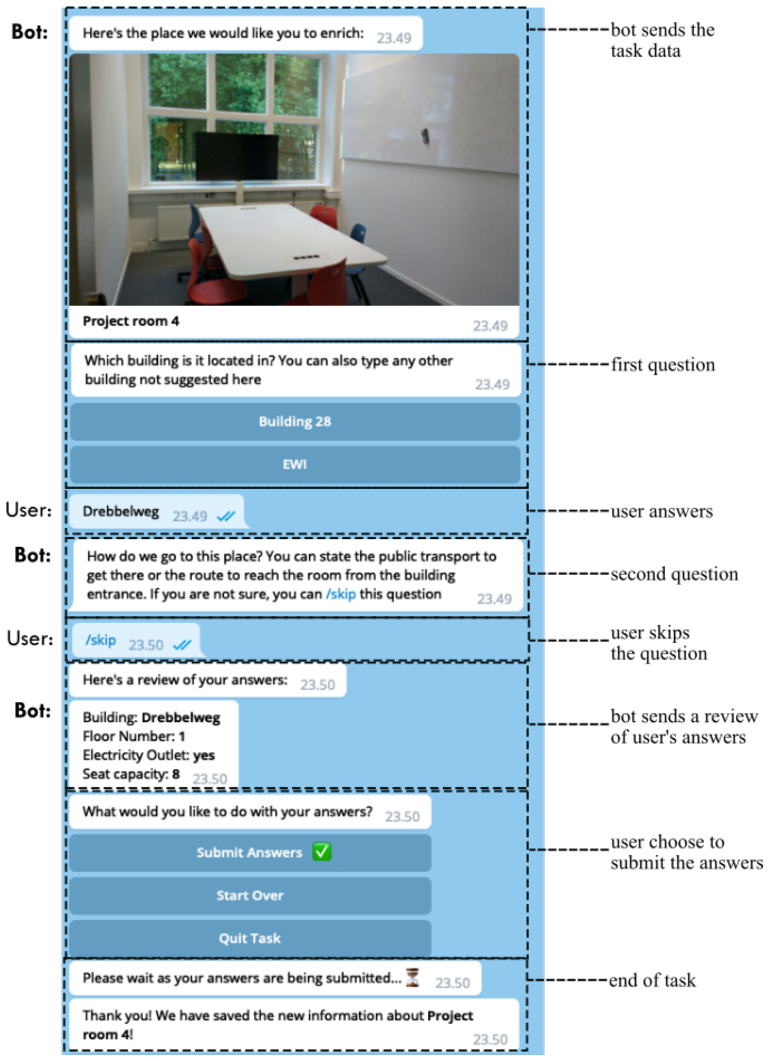
In addition to the task list, the chatbot would also send a message describing other actions that can be done aside from selecting a task. User can choose to do a create task by typing in the `/create` command. User can also request to see a different set of 5 task instances by typing in the `/refresh` command.

### 3.4.3 Task Execution Flow

When the task execution flow starts, the chatbot will send one question at a time to the user, waiting for the user to answer the question before moving on to the next. The user can either: answer the question or skip the question (if it's not a required question).

By the end of the task, the chatbot will offer the user to do one of the followings:

1. **Submit Answers**: The user can confirm and finalize their answers by submitting their answers. After the user submits the answer, the chatbot will terminate the


Bot: Here's the place we would like you to enrich: 23.49

Project room 4 23.49

Which building is it located in? You can also type any other building not suggested here 23.49

User: Drebbelweg 23.49

Bot: How do we go to this place? You can state the public transport to get there or the route to reach the room from the building entrance. If you are not sure, you can /skip this question 23.49

User: /skip 23.50

Here's a review of your answers: 23.50

Bot: Building: **Drebbelweg**
Floor Number: **1**
Electricity Outlet: **yes**
Seat capacity: **8** 23.50

What would you like to do with your answers? 23.50

Please wait as your answers are being submitted... ⏳ 23.50

Thank you! We have saved the new information about **Project room 4**! 23.50

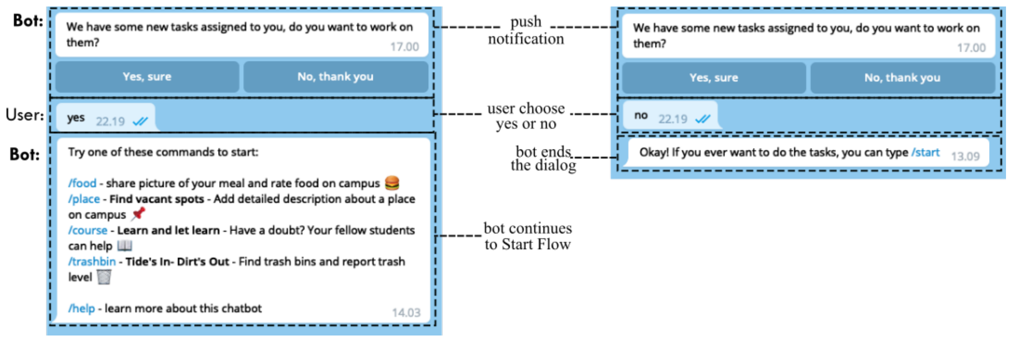
Figure 3.10: Task Execution Flow

current task execution flow, offering the user to do another task and then go back to the Start Flow.

2. **Start Over**: If the user thinks that they need to change their answer, the user can opt to start over the task. The chatbot will then restart the current task execution flow and starts from the first question again.

3. **Quit Task**: If the user does not want to submit the answer, for example they want to do another task instead, they can decide to quit the current task. The chatbot will then terminate the current task execution flow without saving the answers.

The overall flow of task execution is shown in Figure 3.10.



We have some new tasks assigned to you, do you want to work on them? 17.00

yes 22.19

Try one of these commands to start:

/food - share picture of your meal and rate food on campus 🍔
/place - Find vacant spots - Add detailed description about a place on campus 📌
/course - Learn and let learn - Have a doubt? Your fellow students can help 📖
/trashbin - Tide's In- Dirt's Out - Find trash bins and report trash level 🗑

/help - learn more about this chatbot 14.03

We have some new tasks assigned to you, do you want to work on them? 17.00

no 22.19

Okay! If you ever want to do the tasks, you can type /start 13.09

Figure 3.11: Push Notification Flow

### 3.4.4 Push Notification Flow

Once in a while, the chatbot will send a message, asking the user if they are interested to work on another task or not. The user can choose "yes" or "no", which the chatbot will execute the Start Flow for the former, or end the confirmation for the latter. Figure 3.11 shows the dialog samples when the user accepts or reject the notification.

## 3.5 Summary

This chapter describes the details of the design of our conversational crowdsourcing system for constructing knowledge bases. First, a decomposition of KBC task was done through a hierarchical task analysis approach, leading us to derive 6 types of KBC microtask to be covered in our work. Building upon these microtask types, we design a crowdsourcing workflow consisting of three main stages for crowdsourced KBC: Create, Enrich, and Validate. Each of these stages corresponds to a task which in turns consist of several microtasks.

To enable the execution of the crowdsourcing workflow, we design a conversational crowdsourcing system specifically for constructing knowledge bases. The system consists of four main modules: a conversational interface which users interact with to complete KBC task; the task executioner module that manages the states of a task execution; the Backend API module that handles task generation and assignment; and a Knowledge Base Generation module to construct a KB based on the collected answers. We also discuss the design of the conversation flow, describing how users would complete tasks using our system through the conversational interface.

# Chapter 4

# Chatbot Implementation

In this chapter, we would describe the technical implementation of the chatbot system and its supporting modules. First, the implementation of the conversational interface is explained followed by implementation of the logic modules: task executioner; task generation and assignment; as well as knowledge base generation. Finally we would describe how all of the modules are deployed to production.

## 4.1 Conversational Interface

We developed a Telegram Bot using Python 2.7, utilizing the Python Telegram Bot Framework[1] which is a Python interface to the Telegram Bot API[2]. The framework also has some high-level classes that are useful to develop typical chatbot application, such as managing conversation states and error handling, which helps us speeding up the process of developing the chatbot.

We will now discuss some of the key points of conversational interface implementation as follows.

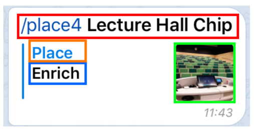
#### Identifying New User

When a user first starts a conversation with the chatbot, they would tap on the Start button in Telegram chat which will send a `/start` command message to the bot. In addition to searching the chatbot directly in Telegram, user can also open the bot by opening it through a unique link e.g. `telegram.me/v1_campusbot`.

In our experimental settings described in later chapter, we would need to track who is the user opening the chatbot. Thus, we append a unique id as the `/start` parameter to the end of the link, which Telegram Bot API will recognize as a parameter to the `/start` command. We process this unique id and save it to the `uniqueId` field of the new user. For example, the link `telegram.me/v1_campusbot?start=22` will pass the number 22 as unique id. So when a user opens the chatbot for the first time using that link and start the chatbot, it will be recognized and saved as that user's unique id.

[1] https://python-telegram-bot.org/

[2] https://core.telegram.org/bots/api


/place4 Lecture Hall Chip
Place
Enrich
11:43

(a) Task Preview Bubble

```
<title>Enrich</title>
<meta property="og:title" content="Enrich"/>
<meta property="og:image" content="https://campusbot.cf/images/place4.png"/>
<meta property="og:description" content=""/>
<meta property="og:site_name" content="Place" />
```

(b) Meta Tags

```
{
   ...,
   "message": "/place4 <b>Lecture Hall Chip</b>
<a href='https://campusbot.cf/task-preview?task=42312asd34123asd1'>\u200f</a>"
   ...
}
```

(c) Message Payload

Figure 4.1: How Task Preview Works

### Sending Message as a Bot

We use the `sendMessage` method of the Telegram Bot API to send text message to the chatbot users. We just have to supply the text and the id of the user to send a text message. This method also supports Markdown or HTML formatting so we sometimes use it to emphasize specific word of the text we're sending by using bold formatting.

There are also some cases that we would need to send a photo or location to our users. In these cases, we use the `sendPhoto` method and `sendLocation` method of the Telegram Bot API respectively.

Some of the questions that the chatbot will have to ask the users are questions with multiple choice answers. In this case, we use Inline Keyboard to give the possible options to the user. Inline Keyboard is a feature of Telegram to send buttons to user along with a text message. When a user taps on one of the buttons, it will trigger a predefined method to be processed by the chatbot. In our case, the chatbot will simply save the user's selected answer and response accordingly.

There is also a question that requires the chatbot to get the user's current location. In this case, we send a text along a keyboard button with the parameter `request_location` to true. By setting this parameter, when the user taps on this button, the chatbot will ask permission to the user and proceed saving the current location if the user agrees to send their location.

### Showing Task Preview

In the chatbot, when a user chooses a use case (e.g. `/place`), the bot will send messages representing previews of available task instances assigned to the user. An example of a task preview is shown in Figure 4.1a.

As Telegram does not yet support sending this type of message (text with image thumbnail), we took advantage of the Rich Link Preview[3] feature of Telegram. This feature processes links send in Telegram conversation and then show a preview of the link with title, short short description, and optionally image of the website.

We implemented the task preview by creating a webpage which dynamically process the task title, task type, and task image into HTML meta tags (shown in Figure

[3]https://telegram.org/blog/link-preview

4.1b) that Telegram will use to process link previews. The chatbot will then send this link to the user in HTML format (shown in Figure 4.1c), using blank character ( u200f) between a hyperlink tag, so the link itself won't be shown in the message bubble.

### Responding to User

We use predefined response templates in order to generate appropriate response to chatbot users according to their current state. The response templates mostly have placeholder to be replaced by the the current chosen item type.

When a user is in the middle of working on a task, the response would be generated based on the current question and the message sent by the user. Each question of a task has their own predefined response template, and it usually contains a placeholder to be replaced by user's previous answers of the whole. For example, given the following question template:

```
{
    ...,
    "text": "What kind of place is {item[name]}?"
    "responseOk": "I see that this place is a item[category][name]."
    ...,
}
```

The bot would use this question template to send the question text replacing `item[name]` with the name of item previously given by the user. If the user answered "TU Delft Library", then the question that would be sent would be:

```
What kind of place is TU Delft Library?
```

Next, the response of the user's answer to this question would depend on the given answer. The bot will replace the `item[category][name]` placeholder with the chosen category's name. So, if the user answered "building" as the answer to the question, then the response to the answer would be:

```
I see that this place is a building.
```

Each question also has their own predefined response dedicated to handle error when a user response to a question with unrecognized message type. For example, when the question asks a user to send an image but the user sends some text instead, the chatbot will response with a message to ask the user to send a photo instead of text. The response message to handle error is also defined in the question template, as shown in the following example with the `responseError` field:

```
{
    ...,
    "text": "Please upload a *photo* of the place?"
    "responseError": "Send a photo using the camera or from photo ...
        album by tapping the attachment (paper clip) icon"
    ...,
}
```

## 4.2 Task Executioner

As explained in Chapter 3, the Task Executioner module is responsible to manage the flow when a user is working on a task. This section descibes how we implemented the Task Executioner module.

### 4.2.1 Flow Handler Classes

We implemented the Task Executioner as 4 Python classes: (1) the **Generic Flow Handler** class which is an abstraction of a task flow handler. It handles generic flow when any type of task is executed; (2) the **Create Flow Handler** class, (3) the **Enrich Flow Handler** class, and the (4) **Validate Flow Handler** class. Each of Create, Enrich, and Validate Flow Handler class is a subclass of Generic Flow Handler, dedicated to handle specific logic for Create Task, Enrich Task, and Validate Task respectively.

Whenever a user selects and starts a task, an instance of one of the class corresponding to the task type is instantiated and would be dedicated to the said user. The instance of the class would be run on an individual thread process, thus each user would be independent from each other when completing a task.

In overall, these classes are responsible to handle the following logic:

**Loading the task template and task instance:** The class should load the corresponding task template and task instance to the memory. In the case of Create Flow Handler, there won't be any task instance associated, thus only the task template is loaded.

**Preparing the opening and closing statement(s) from the task template:** Whenever a task is started, an opening statement loaded from the task template is sent to the user through the `sendMessage` method of the Telegram Bot API. A closing statement is sent instead for whenever a task has been completed.

**Managing the conversation states:** We implemented the state management logic by utilizing the `ConversationHandler`[4] provided by the Python Telegram Bot Framework. We use the question number to represent the state of the module whenever the user is expected answer the corresponding question. Each state is associated with a handler depending on the type of answer is expected for each question. The class would also need to handle the state when a user wants to skip a question using the `/skip` command or they want to quit a task using the `/quit` command.

**Storing the (temporary) answers in memory:** Everytime a user answers a question, the answer should be saved to the memory. This way, it would be possible to include some of the temporary answers in the next questions or responses for the user (as described in Section 4.1).

[4]https://python-telegram-bot.readthedocs.io/en/stable/telegram.ext.conversationhandler.html

**Validating answer:** A certain type of answer would be expected for a question depending on the type of question (see Table 3.4). The class would be responsible to check and handle invalid answer from the user. In the case of a mismatch type of answer, an error message would be sent to prompt the user for answering with the correct type of message.

**Saving the submitted answers:** When a user submits their answer by the end of the task, the class would then save the answer to a database depending on the task type. The Create Flow Handler, for example would save the answers as a new document in the `items` collection, while the Enrich and Validate Flow Handler would save the answers in the `enrichments` and `validations` collection respectively.

### 4.2.2 Task Instance

When a user starts a task, the Task Executioner module should load the selected task instance if it's an Enrich or Validate task. A task instance is assigned to a user, i.e. in the database, it is stored in a subcollection under the user document. A task instance has the following schema (the schema depicts the name of the fields with their corresponding data type):

```
{
    taskId: String, // required
    task: Reference, // reference to Task, required
    createdAt: Timestamp,
    completed: Boolean,
    expired: Boolean
}
```

What we would like to highlight is the fact that the task field references to a task document in the `tasks` collection. The referenced task would in turn have a reference to an item document. To be more clear, here is the schema of a task document:

```
{
    itemId: String, // required
    item: Reference, // reference to Item, required
    type: Number, //0: ENRICHMENT_TASK_TYPE; 1: ...
        VALIDATION_TASK_TYPE; required,
    createdAt: Timestamp, // required,
    numOfAnswersRequired: Number,
    expirationDate: Timestamp,
    aggregatedAnswers: Object,
    answersCount: Object,
}
```

The task executioner would use the data from the referenced item to ask enrichment questions related to the referred item. For validation task, the task executioner would use the data from the `aggregatedAnswers`, which is the aggregated data from multiple answers based on majority count (see section 7.2).

### 4.2.3 Task Templates

The core element used by the Task Executioner to send questions of a task is the loaded task template. A task template represents how a conversation for completing the task should be carried: starting from sending opening messages, asking questions, and finally sending a message to indicate end of a task. We implemented the task template schema as follows:

```
{
   openingStatements: [String],
   closingStatements: [String],
   questions: [QuestionSchema]
}
```

Opening messages are defined as `openingStatements`, a list of message to be sent at the start of a task. Closing messages are defined as `closingStatements`, a list of message to be sent at the end of a task. The questions to be asked are stored in the `questions` field, which we will discuss its schema in more detail in the following section.

### 4.2.4 Question Template

The task template schema supports conditional logic flow which allows the conversation handler to "jump" from one question to another question in a fullfilment of certain condition. We defined the schema of a question as follows:

```
{
   property: String,
   text: String | [String],
   type: Number,
   responseOk: String,
   responseError: String,
   confirmationText: String,
   isRequired: Boolean,
   jumpRules: [QuestionJumpRuleSchema],
   imageUrl: String, //optional, if question includes image
   geolocation: LocationType, // optional, if question includes ...
       location,
   mustHavePropertyNames: [String]
}
```

We want to highlight the `jumpRules` field which is related to the implementation of how the task execution handles conditional jump. A jump rule is a rule consisting of 4 main elements which describe the condition that must be fulfilled in order for the conversation to proceed to a certain question number, instead of the next question. The schema is better depicted as follows:

```
{
   propertyName: String,
   propertyValue: (dynamic),
   isEqual: Boolean,
    jumpIndex: Number // question index
}
```

The **propertyName** field indicates which property of the condition to be checked, while **propertyValue** is the value to be matched. The **isEqual** field indicates whether we want the condition check to be equal or not for it to be evaluated to be true. If the evaluated condition is true, then the task executioner will proceed to question number indicated by the **jumpIndex** field (note that jumpIndex starts from 0 representing the first question). For example, if we define a question with the following jump rule:

```
{
    propertyName: "category",
    propertyValue: /categories/building,
    isEqual: false,
    jumpIndex: 8
}
```

This means that when a user's previous answer for the question asking the category is not **building**, then for the next question, the chatbot will ask the question numbered 8 (instead of the next question). In the case the condition is not fulfilled (i.e. the selected category is **building**), the next question will be asked instead.

## 4.3 Backend API

In order to execute the task generation and assignment strategy, we implemented API endpoints using Flask API[5]. These endpoints can be used by the chatbot to trigger automatic task generation and assignment after a user completes a certain task. We implemented the endpoints for the following logic:

### Generate Enrichment Task

The generation of enrichment task is implemented separately for each item type, with the following endpoint URI:

```
/api/[itemType]/generate-enrichment-task/<itemId>
```

This endpoint will be used by the Task Executioner module whenever a user completes a create task, passing the `id` of the new item by replacing the `itemId` parameter. The endpoint simply creates a new task document and save it to a corresponding task collection (so a place item would trigger a generation of a place task). The endpoint also predefines an expiration date for the generated task. After a task is generated, it automatically triggers a task assignment procedure implemented in a separate endpoint (see Assign a Task to Multiple Users).

We also implemented another variation of this endpoint that allow us to do task generation for a batch of multiple new items. We define the batch endpoint on the following URI:

[5]https://flaskapi.org

```
/api/[itemType]/generate-enrichment-task
```

This endpoint additionally sweeps all expired task, and generate tasks for existing place or trashbin items. This endpoint is not called by the task executioner itself, but scheduled (using cron) to be hit daily at midnight.

**Generate Validation Task**

For generating validation task, the endpoint is implemented with the following URI:

```
/api/[itemType]/generate-validation-task/<userId>
/<enrichmentTaskInstanceId>
```

This endpoint would be used by the Task Executioner module every time a user completes an enrichment. The API would first update the number of answers collected so far for the enrichment task. If the number of answers is sufficient (as defined in `numOfAnswersRequired` in the task), it will proceed to generate a validation task for the corresponding item.

As the case with enrichment task, the generated validation task would also be assigned to several users by the endpoint described in the following section.

**Assign a Task to Multiple Users**

Task assignment procedure is implemented in the following endpoint:

```
/api/[itemType]/assign-task/<taskId>
```

The task assignment simply generates a task instance for each user selected by our task assignment strategy.

**Assign Tasks to New User**

Every time a new user starts to use the chatbot, they will by default have no task instance assigned to them. To prevent showing an empty task list to the user, we implemented an endpoint to assign new task instances for the new user. The endpoint simply loads 15 most recent generated tasks and assign them as task instances to the new user. We use the following endpoint URI which is hit every time a new user is registered:

```
/api/[food|place|question|trashbin]/assign-task-to-user/<userId>
```

**Push Notification**

The push notification API is used to regularly send reminders to chatbot users. The reminder will ask whether they want to work on another task or not. The API is implemented with the following endpoint:

```
/api/push-notification
```

Additionally, we implemented the push notification API to take note if a user has blocked the bot or not, thus we can exclude them to not send the push notification in the future.

## 4.4 Knowledge Base Generation

For the Knowledge Base Generation module, the implementation consists of two components: the knowledge base instance and the script to aggregate and import the answers from the database to the knowledge base instance.

For the knowledge base instance, we setup a new installation of Wikibase to store the structured data. To import and map data from the database, we implemented a python script and utilize Pywikibot[6], a Python library that wraps a collection of script to automate editing activities of a Wikibase instance.

## 4.5 Deployment

We deployed most of our system components on a virtual machine running Ubuntu Server 18.04 as the operating system. We deploy the chatbot using Gunicorn[7] as our Web Server Gateway Interface (WSGI) and then exposed to the public network using ngrok[8]. The exposed url generated by ngrok is then used as a webhook which is used by our Telegram bot.

For the Backend API component, we also deployed it to the same virtual machine and also uses Gunicorn as the WSGI. The API is then exposed to public network using Nginx as reverse proxy with a previously set up domain address.

For the database component, we use Cloud Firestore[9] which is a NoSQL cloud database provided as part of Google Firebase. The chatbot and Backend API reads and writes to the database using the Python Client for Google Cloud Firestore.

The Knowledge Base Generation module is also deployed to the same virtual machine, as a separate API endpoint under the Backend API project. This allows us for improvement in the future where we can use the endpoints for immediate synchronization between the database and the KB whenever a user completes a task and a new answer is saved. For the Wikibase instance used to store the constructed knowledge base, we run it on top of Docker using the Wikibase Docker Image[10] provided by Wikimedia.

[6]ttps://doc.wikimedia.org/pywikibot/master/
[7]https://gunicorn.org/
[8]https://ngrok.com/
[9]https://firebase.google.com/docs/firestore
[10]https://github.com/wmde/wikibase-docker

# Chapter 5

# Evaluating the Chatbot System

This chapter discusses in detail on how we evaluated the implemented chatbot system that should enable users to participate in a KBC process. The evaluation would be divided into two experimental settings. In the first study, we publishes a prototype of our system into the public (in our case, to the students of TU Delft Campus), encouraged them to complete tasks, and evaluate the collected answers. As we weren't to able form a strong understanding based on quantitative findings from the first study, a follow-up study focusing on qualitiative findings is needed. We conducted a second study which consists of one-to-one guided sessions where we observe the participants closer while using the chatbot system to complete tasks. We will explain the details of both study in the following sections.

## 5.1 Goal of Experiments

Our experiments are conducted in order to answer RQ3. Specifically, we would like to investigate the execution time of KBC tasks using conversational interface as well as the quality of the constructed knowledge base. The quality would be measured on the accuracy of answers and completeness of the constructed KB. Additionally, we would also like to learn more on how different types of interaction style, specifically between buttons and free-text input, may influence the execution time and KB quality.

In Study 1, we included 3 task types relating to KBC process and 4 item types as well as two types of interaction style, denoted as button-based input and free-text input within our experimental design. The goal of the first study is to find out if the execution time and quality of KB is acceptable and aligns with the findings from previous work. We also investigate how the variations of interaction style and question types affect the constructed KB.

The second study is a follow-up study to the first one, with the main goal is to justify qualitatively on what is intuitive and what is not for the users to participate in a KBC process using conversational interface. The findings of the second study would be assessed qualitatively to complement the findings in Study 1.

## 5.2 Study 1: Campus Domain Use Case

In the first study, we implemented a chatbot called **CampusBot** to support the construction of a domain-specific knowledge. The system is intended to be used by students of TU Delft to answer questions revolving 4 domains of the campus: Food, Places, Course Questions, and Trash Bins. The answers we get from the students are then aggregated and constructed to be a knowledge base.

We set up the three versions of the system: a Telegram bot with two versions of interaction style (CampusBot 1 and CampusBot 2) and an equivalent mobile app version (Campus Manager app). The reason for developing the mobile app is that the experimental setup was run along with a simultaneous project to compare the use of chatbot and mobile app for crowdsensing. As this thesis would focus on investigating how the implemented chatbot is being used to support KBC process, the focus will be on the chatbot version, although we would mention the mobile app throughout the chapter as needed.

In this section we would discuss the experimental settings for the first study. We first discuss the domain in which our chatbot is implemented. After that we discuss the independent variables (**task types** and **interaction style**), the participant recruitment strategy, and the metrics and measurements that we would take to assess the system.

### 5.2.1 Domain Description

The implemented chatbot as our proof-of-concept has the purpose of building knowledge base in the domain of campus, specifically the TU Delft campus. We refer the chatbot as CampusBot, a bot running on the Telegram[1] messaging platform to collect campus-related knowledge with the following specific item types:

- **Food**: Users would be able to share photos of food that they eat and/or bought at the campus. With the photos, the chatbot would ask them to complete information about the food such as the place where they bought it and how much they paid for it.
- **Place**: The place use case encourages users to upload photos of places around campus such as: Building, Study Space, Education Room, Parking Space, and Food-Beverage Place. The chatbot would ask them to complete the information such as location of the place and available space.
- **Course Questions**: The course use case allows users to post questions regarding a certain course. A user can also answers question from other user while also promoting (or de-promoting) other's answer to a question.
- **Trash Bin**: The trash bin use case would be used for users to report location of trash bins and also the information whether a trash bin is full and need to be replaced.

A more robust description of what type of information to collect for each item type is available in Appendix B.

[1] http://telegram.me

### 5.2.2 Task Types

For our experiment, we have three different task types to be executed by our participants. This task types are derived from the Crowdsourcing Workflow of our system design (section 3.2). We descibe the task types again in this section within the context of campus domain for experimental purpose. The task types are as follows:

- **Item Creation Task**: In creation task, the chatbot users are expected to create an item from scratch by answering questions to complete the information of an item. In our defined campus domain, an item can be a set of meal (food), a place, a course question, or a trash bin. For each type of item, we define a set of questions concerning information of the item.
- **Item Enrichment Task**: In enrichment task, users are given an existing item, either prepopulated by us or created by another user, and then they need to answer questions to complete the information of the given item. Again, we have defined a set of questions for each type of item.
- **Item Validation Task**: In validation task, similar to enrichment, users are also given an existing item. The item already has some information, but not validated. The task would consist of a set of questions to confirm whether the given information is valid or not.

Each task would contain a set of predefined questions depending on the type of item. An example set of questions for item creation task for place items is shown in Table 5.1 along with the type of chatbot UI elements to facilitate the user to input their answers. Question 4-13 are also asked in Item Enrichment Task. Meanwhile, Table 5.2 shows an example set of questions for place item validation task.

Initially, our knowledge base is empty; thus there wouldn't be any enrichment task or validation task available until a user creates an item by executing the item creation task. As we want to avoid of showing an empty task list to our participants, we prepopulated some items along with their enrichment and validation tasks. Table 5.3 shows the number of item enrichment tasks and item validation tasks we created for each item type.

Note that we initially planned for the validation tasks to be generated according to our task generation strategy, where a sufficient number of answers for an enrichment task would generate the corresponding validation task. Though We expected there would be enough number of users to complete the enrichment tasks, it did not turn out as expected. To accommodate this, we manually generated some validations tasks even if the number of answers does not satisfy the minimum answers required.

### 5.2.3 Interaction Style for Multiple Choice Questions

One of the most common question type asked in our experiment is the multiple choice (MC) question, in which we have three specific defined question types: MC Categorization, MC Item, and MC Custom. We initially choose Telegram inline button as the interaction style for user to select their answer. To investigate how the choice of input UI for this type of question affects the task completion time and answer input, we include two variations of the interaction style for this question type. The variation

Table 5.1: Questions for Item Creation and Enrichment Task for Place Items

| No | Question | Question Type | UI Input | Required |
|---|---|---|---|---|
| 1 | Please upload a photo of the place | Image | File Upload | Yes |
| 2 | What is the name of the place? | Text | Free Text | Yes |
| 3 | What kind of place is *item[name]*? | MC Categorization | Inline Buttons | Yes |
| 4 | Where is it located? You can share your current location (using the keyboard button) or a custom location (using the attach icon). | Location | Custom Keyboard, Location Sharing | Yes |
| 5 | Which building is it located in? You can also type any other building not suggested here | MC Item | Inline Buttons, Free Text | No |
| 6 | Which floor number is it located on? You can /skip this question if you are not sure | Numeric | Free Text | No |
| 7 | What is the number of this building? For example, EWI is building number *36*. You can /skip this if you are not sure. | Numeric | Free Text | No |
| 8 | How do we go to this place? You can state the public transport to get there or the route to reach the room from the building entrance. | Text | Free Text | No |
| 9 | Does this place have any electricity outlet? | MC Custom | Inline Buttons | No |
| 10 | What is the estimated capacity of this place? | Numeric | Free Text | No |
| 12 | How many seats are available right now? | MC Custom | Inline Buttons | No |
| 13 | How clean is it right now in *item[name]*? | MC Custom | Inline Buttons | No |

Table 5.2: Questions for Item Validation Task for Place Items

| No | Question | Question Type | UI Input | Required |
|---|---|---|---|---|
| 1 | Does the photo represent *item[name]*? | MC Custom | Inline Buttons | No |
| 2 | Is this place a *item[categoryName]*? | MC Custom | Inline Buttons | No |
| 3 | Is this place located in building *item[buildingName]*? | MC Custom | Inline Buttons | No |
| 4 | Is this place located on floor *item[floorNumber]* of building *item[buildingName]*? | MC Custom | Inline Buttons | No |
| 5 | To get to *item[name]*: *item[route]*, is this correct? | MC Custom | Inline Buttons | No |
| 6 | Does this place have any electricity outlet for public use? | MC Custom | Inline Buttons | No |
| 7 | This place has *item[seatCapacity]* capacity, is this correct? | MC Custom | Inline Buttons | No |

Table 5.3: Number of Pre-populated Tasks Across Item Type and Task Type

| Item Type | Enrichment Task | Validation Task* |
|---|---|---|
| Food | 3 | 2 |
| Place | 11 | 7 |
| Course Question | 3 | 1 |
| Trash Bin | 13 | 9 |

depends on how a user can input their selected answer choice. Figure 5.1 shows the same question with the two interaction styles.

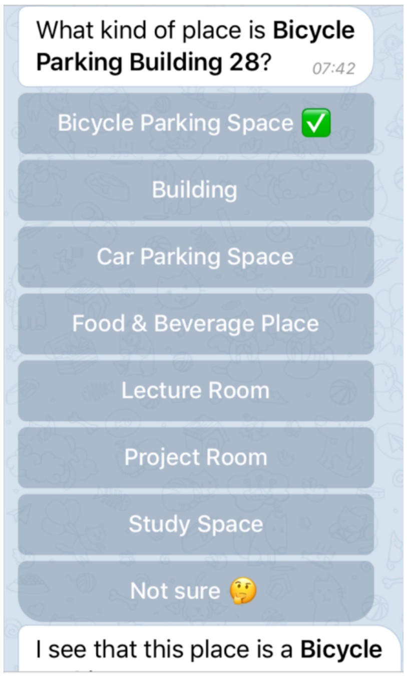
**Input through buttons (CampusBot 1)**

With this interaction style, user can simply input their answer by selecting their choice through the inline buttons provided by the chatbot. When a user sends their answer, a checklist mark would be shown next to the text to indicate the selected answer.

**Input through code and/or free text (CampusBot 2)**

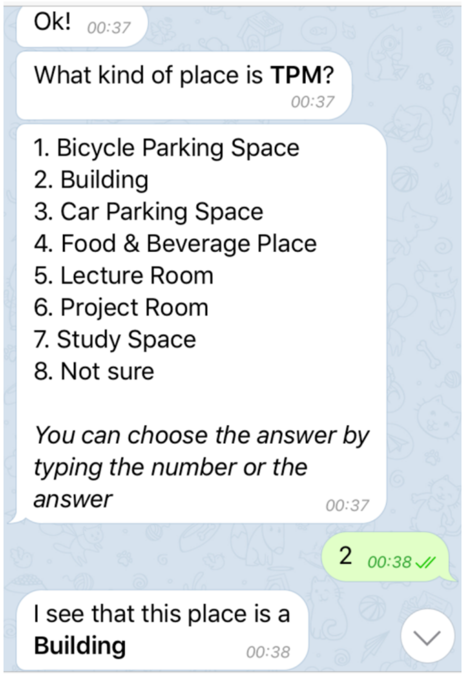
With this interaction style, the chatbot would send a list of possible answer along with code (numbers) corresponding to each answer. User can then select the answer by typing either the number of the answer or the text of the answer of itself.

When inputting some text, our chatbot would match the input with a set of predefined regular expressions (regex) corresponding to each answer. Additionally, we


What kind of place is **Bicycle Parking Building 28?** 07:42

I see that this place is a **Bicycle**

(a) Buttons


Ok! 00:37

What kind of place is **TPM?** 00:37

1. Bicycle Parking Space
2. Building
3. Car Parking Space
4. Food & Beverage Place
5. Lecture Room
6. Project Room
7. Study Space
8. Not sure

*You can choose the answer by typing the number or the answer* 00:37

2 00:38

I see that this place is a **Building** 00:38

(b) Code and Free Text

Figure 5.1: Interaction Style

would also ignore the case (lower and capital letters) when matching this regular expressions.

### 5.2.4 Recruiting Participants

For the first experiment, we conduct an open recruitment process to recruit our participants without offering any extrinsic incentives. We are well aware that this strategy has the risk of not having enough well-motivated participants, but we want to try avoiding confounding factors introduced by other variables (e.g. the presence of incentives) outside the independent variables that we are testing.

The open recruitment process was conducted by advertising our application throughout the TU Delft Campus. The advertisement was done by means of posters, screens advertisement, and short presentations in several lectures. The advertisement promotes the experiment study and invites students who are interested to sign up through Google Form. We include the poster and form for the recruitment in Appendix D.

When signing up, the student are asked to fill their email address and the Operating System (OS) of their mobile phone. Then, we use Google Apps Script to automatically assign the participant to a system version (Mobile App, CampusBot 1, or CampusBot 2). The assignment is done by a rotation, so the first participant who signs up will be assigned to mobile app, the second participant will be assigned to CampusBot 1, the third participant will be assigned to CampusBot 2, and so on.

The script will then send an email to the participant according to their assigned system and selected OS. The email contains guide on how to download the Mobile App (or Telegram for chatbot participants) and also a short step-by-step tutorial on how to use the chatbot. In the email, participants are encouraged to use the app for 2-3 weeks so we can gather sufficient data. Each email is tailored to the participant based on their assigned system such that it has a unique link to the chatbot so we can identify the user who opens the chatbot through the link. The email template is included in Appendix D.

### 5.2.5 Metrics and Measurement

By the end of our first study, we will measure the following as our dependent variables: Execution Time, Answer Accuracy, and Knowledge Base Completeness. Additionally we would also send out a post-experiment survey where we will ask our participants regarding the usability and possible improvements of our chatbot system.

#### Execution Time

The execution time would be measured in seconds between starting a task and submission of a task. In case of user selecting to start over a task, we would take the whole execution time from the first start, taking note on the times that the user has restarted the task.

The start of a task is when a user inputs the command to start a task. This command is either the `/create`, or the item type and number of the task (e.g. `/place1`). The end of a task is when a user submits the answer by selecting the "submit answer" button (in CampusBot 1) or inputting the "submit" keyword (in CampusBot 2).

Additionally, we would also measure the execution time on the question level based on the question types as described in Section 7.2. The time would be measured between when the question is sent by the bot and when the user sends their answer for that particular question.

#### Answer Accuracy

The accuracy of answers contributed by the user would be manually inspected and compare to the factual information of TU Delft campus as necessary. Because most tasks are generated from user's own created item, we won't have the actual ground truth from each task except that from information inferred from the photo uploaded by the user and our own knowledge about the campus referring to TU Delft website as necessary.

For food items, we would just check if the names of the food input by the user is indeed shown in the uploaded photo. For place items, we will mostly refer to information available on TU Delft website or our own knowledge. As for course questions, we would not measure the contribution quality as it would be difficult to confirm the answer validity across courses. For trash bin items, we would heavily focus on the photo of the trash bin and its related properties such as waste type and the color of the trash bin. As it would be difficult to know if the exact location of the trash bin is correct or not, we wouldn't take this into account when measuring the quality.

### Knowledge Base Completeness

In addition to quality, another important metric of evaluating a knowledge base is its completeness [4]. We would measure the completeness of the knowledge base constructed by means of chatbot. In a survey of knowledge base assesment metrics [60], completeness refers to the degree to which all required information is present. In our experiment we will use two dimensions to measure the completeness as defined in [59]: Schema Completeness ($m_{cSchema}$) and Column Completeness ($m_{cCol}$).

- **Schema Completeness** ($m_{cSchema}$): Schema completeness represents to what extent in which classes (categories) and relations (properties) are not missing. Formally, the schema completeness is defined as the ratio of the number of classes and properties represented in knowledge base $g$, $no_{clatg}$, to the number of classes and properties in the gold standard, $no_{clat}$. For our experiment, we set our gold standard for Place and Food classes as shown in Table 5.4.

$$m_{cSchema}(g) = \frac{no_{clatg}}{no_{clat}}$$

- **Column Completeness** ($m_{cCol}$): Column completeness represents the extent to which values of properties on item level (or statements) are not missing. Formally, it is defined as the ratio of the number of items under categories $k$ and having a value for property $p$, $no_{kp}$, to the number of all items having under category $k$. To calculate the column completeness of the whole knowledge base $g$, we average the completeness over all category-property pairs on the item level. Here the metric is defined formally letting $H$ as the set of combinations of the considered categories and considered properties:

$$m_{cCol}(g) = \frac{1}{|H|} \sum_{(k,p)\in H} \frac{no_{kp}}{no_k}$$

### Usability Survey

We would also measure how usable is the implemented chatbot to enable users participate in constructing a knowledge base. We sent out a post-experiment survey too all participants who signed up regardless of their activities level – we also sent it out to people who signed up but did not try the chatbot at all. The set of questions is included with this report in Appendix E

We based the survey on the System Usability Scale (SUS) [17], asking 10 questions with five response options in likert scale (1 means strongly disagree, 5 means strongly agree).

We also asked which specific use case the participants find easy to do as well as the ones they found challenging to do. They can choose which one of the following is hard or easy to do: starting the chatbot, finding a task, choosing a task, creating an item, completing an enrichment task, and completing a validate task.

Additionally, for people who signed up but did not try the chatbot, we asked a question to find out the reason behind why they did not try the chatbot. We give them a list of possible reasons to choose and they can input their own reason not listed on the survey.

Table 5.4: Gold Standard to Compute $m_{cSchema}(g)$ and $m_{cCol}(g)$

| Item Type | Categories | Properties |
|---|---|---|
| Place | Place | coordinate location |
| | Bicycle Parking Space | route |
| | Building | available seats |
| | Car Parking Space | floor |
| | Food Establishment | building |
| | Lecture Room | building number |
| | Project Room | has electricity outlet |
| | Study Space | image |
| | | maximum capacity |
| | | clean level |
| Food | Meal | image |
| | Food | food location |
| | Appetizer / Snack | meal item |
| | Main Course | price |
| | Beverage | rating |
| | Dessert / Fruit | |

## 5.3 Study 2: Qualitative User Study

In the second study, we would like to investigate even further, specifically in the qualitative perspective, on how the chatbot can be used to construct a knowledge base. We only got a small sample of task completed from Study 1, so we would need to to conduct a second study to interview the users and get a better understanding on how the usage of the chatbot affect the resulting answers and constructed KB. For this purpose, we invite participants for one-to-one interviews while guide and observe them to complete tasks using the chatbot.

### 5.3.1 Recruiting Participants

We invited some of the participants from the first by contacting them personally and asked if they are interested to participate in a follow-up interview. As to avoid bias on having previous experience on the chatbot before, we try to prioritize participants who either: signed up but did not download or try the chatbot; or tried the chatbot without completing any task. Though, user who has already used the chatbot before could still be invited and we will take consideration on this fact when analyzing the study results. Additionally, we also tried to recruit some additional participants who did not sign up for Study 1.

### 5.3.2 Study Setup

In the interview, a smartphone with Telegram installed and a laptop to browse the internet is provided. The interview would be carried as follows:

1. First, we would explain briefly about our thesis goal and the implemented chatbot. We would also explain to the participant that the screen of the provided smartphone is recorded and the conversation would also be recorded. The participant could withdraw from the study whenever they want.
2. Participant is then guided to open the chatbot using the provided smartphone.
3. After starting the chatbot, the participant is then guided to complete the following tasks: choosing a task, completing a validation task, completing an enrichment task, and creating an item. In this second study, we guide the user to complete the tasks for place item types.
4. While working on the task, participant is allowed to use the provided laptop to search information in order to complete their tasks, specifically for answering the questions asked by the chatbot. We also provide links to pages containing information of the place asked in each task.
5. The participant is encouraged to say their thoughts out loud when using the chatbot to complete the given tasks. Once in a while, when the participant seems to ponder on something, we would ask them some prompting questions in order to know what they think about the task at hand.
6. Finally, after completing all of the tasks, the interviewer would ask several questions to the participant in order to know more about what was easy and challenging while using the chatbot.

### 5.3.3 Post-Interview Survey

After completing the requested tasks, the participant is then asked to fill a survey on the usability of the system and also about the things that they found easy and challenging to do while using the chatbot. The survey also asked additional comments in order to ellicit any additional comments and/or suggestions not discussed in the verbal interview. The set of questions of the survey is similar to the survey sent for participants of Study 1, and is included in Appendix F.

### 5.3.4 Analysis Method

The main method to analysis the result of the second study would be qualitative analysis based on the recorded interview. We, as the interviewer, would also take notes on our observation on the screen when the user was executing the tasks. We summarize our findings based on common things that the participant found challenging to do as well as connecting it to the results found in Study 1.

# Chapter 6

# Results and Discussion

This chapter discusses the results obtained from running the two studies described from the previous chapter. We would divide discussion into two main sections: discussion on mainly quantitative result from the first study and discussion on qualitative analysis from results of the second study. With each study, we also discuss some issues we found from running the experiment.

## 6.1 Results and Discussion on Study 1

For the first study, we carried an analysis on the resulting completed tasks focusing on the execution time and quality of the knowledge base. Before we discuss the two collected measurements, to give a better overview of the data collected, we give a brief discussion on general statistics from our participants.

### 6.1.1 General Statistics on Participation and Tasks

#### Participants Conversion

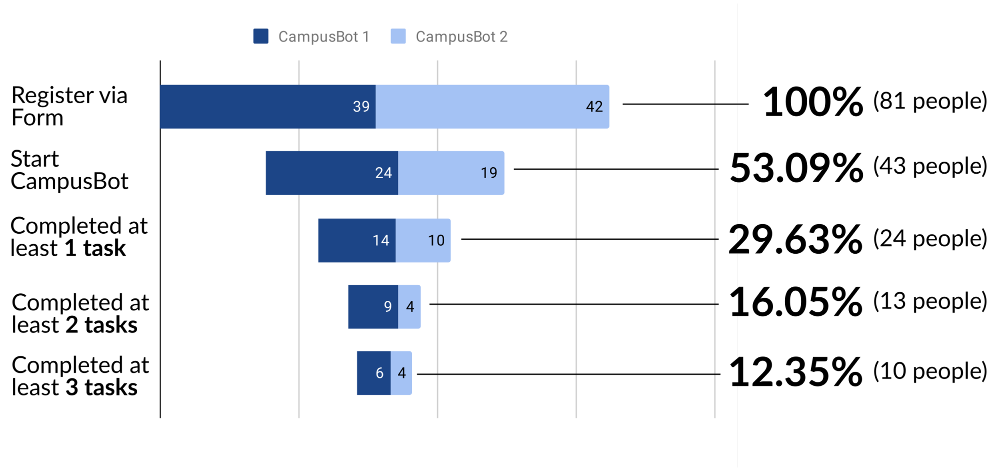


Figure 6.1: Participants Conversion

A total number of 81 people were recruited by means of registration through an online form, and 39 were assigned to use CampusBot 1, while 42 were assigned to use CampusBot 2. Not all of these people try the chatbot, as we record 53.09% of them (43 people) who actually open the link sent to their email to start the chatbot. Furthermore, among the 43 participants, there are 24 participants who at least completed one task. Figure 6.1 shows the participants conversion funnel starting from the number of recruited people who registers through the form until the participants who completed at least 3 tasks.

We are aware that the number of active users were not as high as expected, thus we analyze the collected data keeping in mind that we don't have that much data to do more of statistical analysis. We will also discuss some of the potential issues causing this low participation in section 6.1.6 as well as conducting Study 2 as a follow-up study for better understanding of our system's capability to support KBC process.

#### Task Completion

In the experiment for Study 1, a total of 49 tasks is deployed consisting of 30 item enrichment tasks and 19 item validation tasks. Each task contained a number of task instances assigned to the participants. The number of assignments is not fixed for each task because of our task assignment strategy which assigns a task according to the number of currently available users and the similarity of the task (i.e. location of the place) with their previously completed tasks.

A total of 105 task instances were executed by the participants in an experiment period of 50 days. The executed task instances consist of 30 creation task instances, 68 enrichment task instances, and 7 validation task instances. The number of task instances completed by each user grouped by the type of task is shown in Figure 6.2, with 2 most active users completed 19 task instances each. We note that the number of completed enrichment task instances is higher than creation task instances, and even way higher than the number of validation task. This might be caused by higher availability of enrichment task, because validation task is generated only after enrichment task has sufficient number of answers.

### 6.1.2 Execution Time

In this section, we discuss and compare the execution time across task types (Item Creation Task (Create), Item Enrichment Task (Enrich), Item Validation Task (Validate)), item types (food, place, course question, trash bin) as well as interaction style for multiple choice questions (buttons and code & free text). Table 6.1 summarize the average and standard deviation of task execution time across task types and interaction style along with the numbers of task executions (#TE).

In all task types (create, enrich, validate), we can note that average task execution for place items takes more time compare to the other item types. This is due to the number of defined questions is higher for place items then other item types. Meanwhile, on average, validating answers of a course questions takes the least time, that might be due to the fact that the users just have to decide to choose an "up-vote" or "down-vote" to several answers to a question regarding a course.

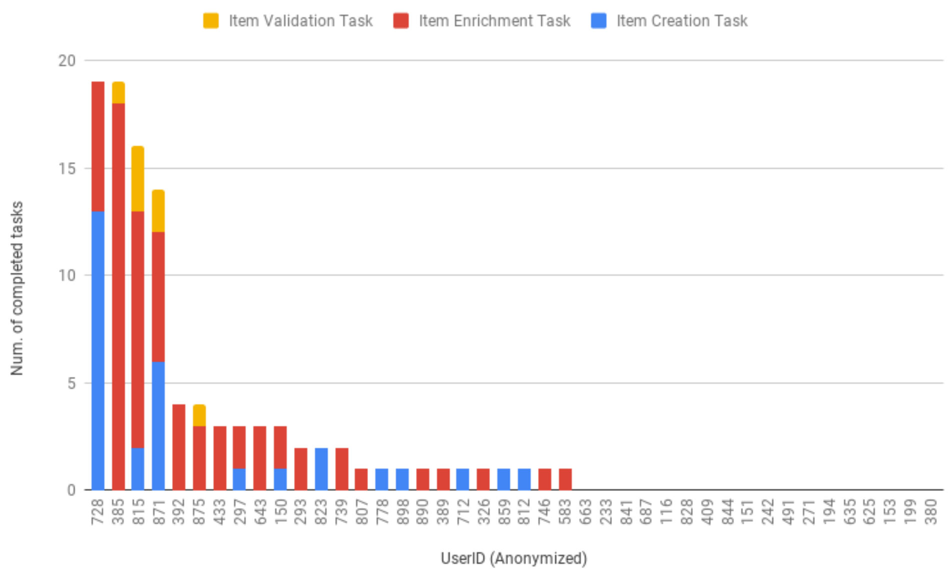


Figure 6.2: Task Instance Completion

Table 6.1: Execution Time (average ± standard deviation, unit: seconds) Across Task Types and Interaction Styles (#TE=number of task executions)

| | | Interaction Style | | | |
|---|---|---|---|---|---|
| **Task Type** | **Item Type** | **Buttons (B)** | **#TE** | **Code & Free Text (CF)** | **#TE** |
| **Create** | Food | 242 | 1 | 84 ± 17 | 4 |
| | Place | 374 ± 522 | 9 | 155 ± 93 | 12 |
| | Course Question | 104 ± 72 | 3 | *n/a* | 0 |
| | Trash Bin | *n/a* | 0 | 87 | 1 |
| **Enrich** | Food | 22 ± 15 | 12 | 33 ± 15 | 6 |
| | Place | 88 ± 97 | 35 | 92 ± 81 | 8 |
| | Course Question | 17 ± 5 | 4 | 94 ± 59 | 3 |
| | Trash Bin | 21 | 1 | *n/a* | 0 |
| **Validate** | Food | 30 | 1 | *n/a* | 0 |
| | Place | 34 ± 11 | 4 | *n/a* | 0 |
| | Course Question | 16 ± 4 | 2 | *n/a* | 0 |
| | Trash Bin | *n/a* | 0 | *n/a* | 0 |

Note that there are item types that we only get one sample for some settings, presented in the table with #TE equals to 1. There are also some cases that we did not even manage to acquire any sample at all, presented with #TE equals to 0 in the table, thus the average execution time is not available (*n/a*). Due to this lack of sufficient samples, we can only compare the average execution time across the two interaction styles for the following pairs of task-item type: Create-Place, Enrich-Food, Enrich-Place, and

Table 6.2: Execution Time (average ± standard deviation, unit: seconds) Across Non-Multiple Choice Question Types (B=Button Input, CF=Code & Free Text Input)

| **Metric** | **Text** | **Numeric** | **Location** | **Image** |
|---|---|---|---|---|
| Execution Time | 21 ± 23 | 12 ± 9 | 38 ± 69 | 47 ± 53 |

Table 6.3: Execution Time (average ± standard deviation, unit: seconds) Across Multiple Choice Question Types (B=Button Input, CF=Code & Free Text Input)

| | **MC Categorization** | | **MC Item** | | **MC Custom** | |
|---|---|---|---|---|---|---|
| **Metric** | **B** | **CF** | **B** | **CF** | **B** | **CF** |
| Execution Time | 8 ± 4 | 15 ± 19 | 8 ± 5 | 16 ± 6 | 4 ± 1 | 5 ± 3 |

Enrich-Course.

For Enrich-Food, Enrich-Place, and Enrich-Course, we can see that their average execution time are higher when using Code & Free Text (CF) interaction style compared to using Buttons interaction style. This is expected due to the fact that the user has to type in the answers to the KB questions using the CF input, while with buttons, user can tap on one of the buttons to give their answer. Intuitively, tapping on a button should take less time then typing the answer. Unexpectedly, though, we see the average execution time for creating place items using buttons seems to take more time compared to using CF. We suspect that this is because we don't have enough sample or because certain question takes more time to answer despite of the interaction style.

In order to have a more thorough analysis, we take a look into the execution time of a more finer level of the task execution: on the question level. A task consists of several questions with various types: Text, Numeric, Location, Image, and Multiple Choice (MC) questions. Users might need more time to answer a certain type of question compared to the others. In order to know whether the execution time was affected due to different interaction style for multiple choice questions or due to another type of question, we take a look into the average execution time for each question type. The average is calculated by sampling the execution time of each question from 12 task instances, in which each task instance would have around 11-12 questions.

Table 6.2 and Table 6.3 shows the average and standard deviation of execution time across non-multiple choice question types and multiple choice question types respectively. Figure 6.3 and 6.4 shows the distribution of execution time for non-multiple choice question types and for multiple choice question types respectively.

We can note from Figure 6.3, image questions need more time to be answered, because the user has to take or choose a photo and we also have to take into account the time needed for the image to be uploaded. There seems to be a similar case for answering Location questions, where user has to pick a location from a map. Though, for this case, user can speed up the process by sending their current location with the provided keyboard button.

Numeric question type takes less time to be answered, compared to Text question type. Questions with text as expected answer varies on the question itself, where

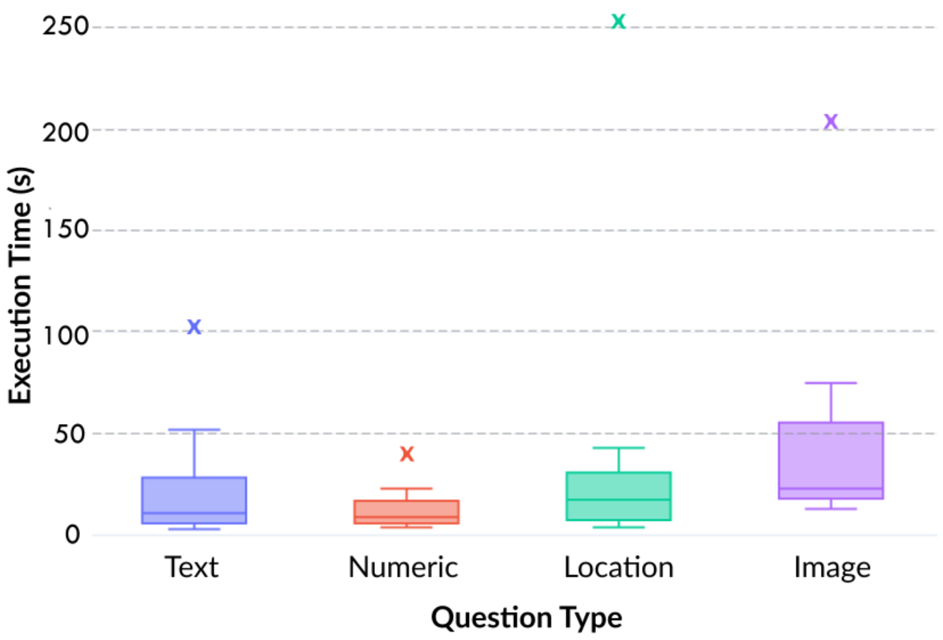


Figure 6.3: Execution Time (seconds) of Text, Numeric, Location, and Image Questions

answering question such as "*What is the name of the place?*" takes less time then answering questions "*What is the route to get to the place?*".

We would like to highlight on three question types under the Multiple Choice (MC) question types (MC Categorization, MC Item, and MC Custom) which we have set for each to have two variations of interaction style (Buttons (B) and Code & Free Text (CF)). From figure 6.4, we can see that the average execution time for these three question types is higher using Code & Free Text interaction style compared to using Buttons interaction style. Although we couldn't statistically test the significance of the difference due to small sample size, from looking into the conversation log, we suspect that answering questions with the CF interaction style takes more time due to having to input the answer by typing the code or the text of the answer. Especially with the MC Item, where the user can answer a custom answer not listed as a choice, in which user has to type the full answer instead of typing a shorter code corresponding to the answer. Intuitively, typing out a full answer would take more time comparing to typing out the code or tapping on a button. Interestingly, for MC Categorization and MC Custom using CF interaction style, we observe from conversation log that most users answer by typing the code of the answer (e.g. "5"), instead of the text of the answer (e.g. "Lecture Room"), which should speed up the typing time, but still takes longer time on average compared to interacting with buttons.

In a way, our findings in term of execution time is in line with the findings in [37] on using chatbots for microtask crowdsourcing, where they found that using button-based interaction pulls ahead in terms of execution time compared to text-based interaction. Our work extends the findings within the scope of KBC, though we lack

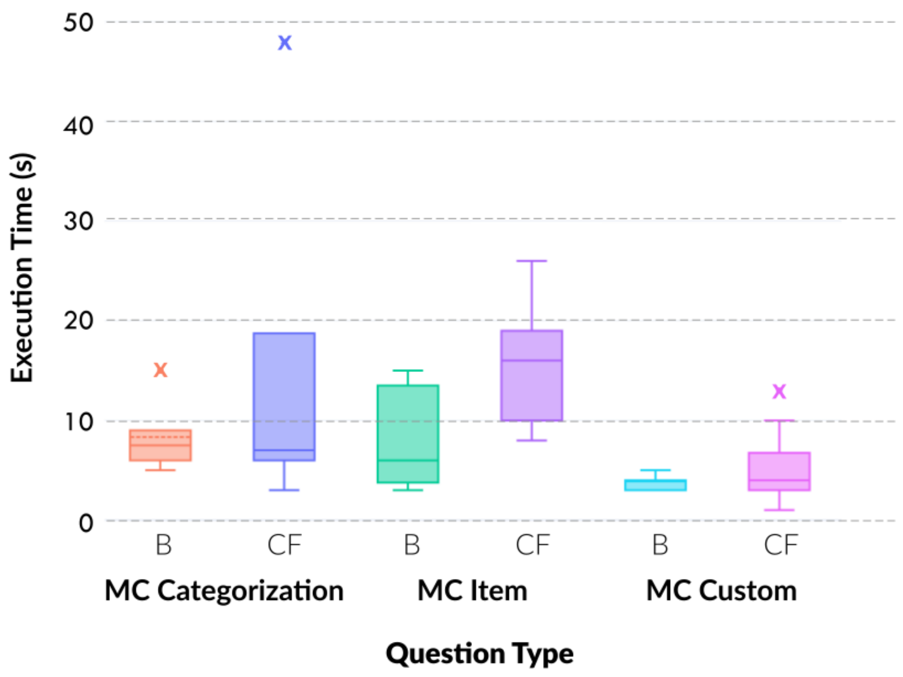


Figure 6.4: Execution Time (seconds) of Multiple Choice Questions (B=Buttons, CF=Code & Free Text)

statistical testing to know the significance of the difference due to small sample size as well as more variations of interaction style (e.g. mixed input of buttons and free text). This might be an opportunity for future work to measure the execution time with larger sample size and more variations of interaction style.

### 6.1.3 Answer Accuracy

In order to know the quality of the answers provided by the users, we measure the accuracy of their answers by calculating the percentage between the number of correct answers and the number of given answers. In other words, we do not take into account questions which the users have skipped the answer. Furthermore, we only consider the accuracy for the place and food item types, because we don't have the necessary ground truth for trash bin and course question items. In the case of place items, in order to determine if the answer is correct or not, we inspect each answer manually and refer to the factual information which is available on the TU Delft website. For food, we judge their answer subjectively (i.e. determining if the food mentioned contained in the image or not).

We examine a total of 397 answers from 21 place creations, 42 place enrichments, 4 place validations, 5 food creations, 18 food enrichments, and one food validation. Table 6.4 shows the answer accuracy across task types and interaction styles for completed place and food task instances.

At first look, the accuracy of place items for both creation tasks and enrichment tasks seems to be lower when using the CF interaction style compared to using the button-based interaction. After we checked the answers of both versions of the chatbot manually, the low accuracy turns out to be caused the answers for the question asking the route to get to the place *"How do we get to this place?"*. Most answers are deemed incorrect because their answer mentions only the number of the bus without the stop to get off at in order to reach the created place. We suspect that the cause is that the question might be unclear to the participants, so we would revisit this particular question in the second study, where we clarify with the participants whether the question is clear or not.

Answers of validation tasks have high accuracy both for place items and a single food item (we only managed to get one answer for food validation task). The high accuracy is expected due to the fact that the users only have to answer some "yes" or "no" questions to validate whether a statement is correct or not. However, we only got answers for validations using buttons as the interaction style, so we couldn't compare the accuracy between the two interaction styles.

Furthermore, we also look into the accuracy across question types to know whether answers to a certain type of question have the tendency to be more often correct or incorrect. Table 6.5 and 6.6 shows the accuracy of answers grouped by the type of question. We can note that answers for questions without choice expecting text, numeric, location, and image have lower accuracies compared to multiple choice questions. Questions expecting text input, in particular, have the lowest accuracy, which we suspect is due to the nature of allowing the user to input any text without any specific input validation, thus more prone to incorrect answers.

In the case of multiple choice questions, we have given a set of possible options of the answer, thus the chance of receiving incorrect answer is lower compared to typing the answer without any given choice of answer. As for the effect between using button-based interaction (Buttons) and using text-based interaction (Code Free Text), there seems to be no apparent difference in term of the accuracy of the inputted answers. This is the case because both interaction styles only accept the pre-defined choices as a valid answer, and would ask the user to re-input their answer again if they try to submit answer that is not included in the provided choice of answers. For MC Item questions, we do allow users to input a custom answer not listed in the choice list, but when we look into their answers, users tend to select one of the choices instead of typing their own custom answer.

Our findings regarding the answer accuracy are as follows. First, the accuracy seems to be comparable across all task types and also across place and food items. Second, between the two interaction style, there seems to be no apparent difference in terms of answers of accuracy. Finally, we found that answers for multiple choice questions tend to have higher accuracy compared to text, numeric, location, and image questions. This might be a motivation for future work to suggest possible answer choices for the users in order to enforce better answer accuracy.

### 6.1.4 Knowledge Base Completeness

In addition to execution time and accuracy, when we construct a knowledge base, one of the challenge is to have a high coverage of the acquired knowledge such that the information contained in the KB should be as complete as needed. Thus, we mea-

Table 6.4: Accuracy across different task types, item types, and interaction styles (#TE=number of task executions)

| | | **Interaction Style** | | | |
|---|---|---|---|---|---|
| **Task Type** | **Item Type** | **Buttons (B)** | **#TE** | **Code & Free Text (CF)** | **#TE** |
| **Create** | Place | 0.88 | 9 | 0.66 | 12 |
| | Food | 1.00 | 1 | 0.75 | 4 |
| **Enrich** | Place | 0.90 | 34 | 0.68 | 8 |
| | Food | 0.79 | 12 | 1.00 | 6 |
| **Validate** | Place | 1.00 | 4 | *n/a* | 0 |
| | Food | 1.00 | 1 | *n/a* | 0 |

Table 6.5: Accuracy Across Non-Multiple Choice Question Types (B=Button Input, CF=Code & Free Text Input)

| **Metric** | **Text** | **Numeric** | **Location** | **Image** |
|---|---|---|---|---|
| Accuracy | 0.66 | 0.74 | 0.76 | 0.85 |

Table 6.6: Accuracy Across Multiple Choice Question Types (B=Button Input, CF=Code & Free Text Input)

| | **MC Categorization** | | **MC Item** | | **MC Custom** | |
|---|---|---|---|---|---|---|
| **Metric** | **B** | **CF** | **B** | **CF** | **B** | **CF** |
| Accuracy | 0.87 | 0.90 | 0.98 | 0.92 | 0.98 | 0.90 |

sure and evaluate the completeness of the knowledge base constructed by the knowledge base generation module which aggregates the answers acquired from our chatbot users. The constructed knowledge base is available as RDF and JSON dump file in our github repository[1]. We measure the completeness using two dimensions: the schema completeness ($m_{cSchema}$) and the column completeness ($m_{cCol}$). Schema completeness represents how much of the schema (in our case categories and properties) are represented as items and statements in the generated KB. Column completeness represents how complete the KB on the instance level, making sure that each instance has the required properties based on their categories.

Table 6.7 shows the $m_{cSchema}$ and $m_{cCol}$ across item types and interaction styles. We have the completeness for each of the KB generated from each version of the chatbot with different interaction style, and also the completeness of the *Combined KB*, which is generated by the combined answers of both versions of the chatbot. We only include Places and Food in our analysis as we did not acquire enough task completion for Course Questions and Trash Bins.

We can note from Table 6.7 that the schema completeness is measured to be comparable across item types and interaction styles. While the schema completeness is

[1]https://github.com/enreina/campusbot-data-results

Table 6.7: Schema and Column Completeness Across Item Type and Interaction Style

| | | | Interaction Style | |
|---|---|---|---|---|
| **Item Type** | **Metric** | **Combined KB** | **Buttons** | **Code & Free Text** |
| Place | $m_{cSchema}$ | 0.94 | 0.83 | 0.83 |
| | $m_{cCol}$ | 0.63 | 0.42 | 0.28 |
| Food | $m_{cSchema}$ | 1.00 | 0.73 | 0.91 |
| | $m_{cCol}$ | 0.45 | 0.25 | 0.20 |

acceptable, with only missing some unrepresented categories (e.g. there is no "car parking space" place created) and properties (e.g. food, price, and rating are not represented in food knowledge base), the column completeness is fairly low especially of two KBs generated from answers from each versions of the chatbot. Though, as expected, the column completeness gets better when we combine the answers from both versions, as statements acquired from one version of chatbot complements and completed the missing statements from the other version.

To understand more about how such results of schema and column completeness is obtained, again we look into how much of answers did we acquire from each type of question. In this case, we measure the completeness simply as the ratio between the number of submitted answers (i.e. the user did not skip the question) and the number of total questions (within that question type) which were asked to the users. Table 6.5 and Table 6.6 shows the completeness percentage of each type of question.

Unsurprisingly, we have perfect completeness for Location and Image questions as we have set the questions of these two types to be required, thus the user wouldn't be able to skip the question. In the case of MC Categorization question (e.g. asking the user to categorize the category of a place), we actually did not require the user to answer the question, allowing them to answer "Not Sure" if they don't know category of the item. Nevertheless, all users answered to the categorization question, resulting in complete set of answers.

Between using Buttons and CF interaction style, there is not that much difference of completeness percentage between the two interaction styles. We can see that most users did not skip questions of type MC Item (e.g. *which building is the place located in*) and type MC Custom (e.g. *does the place has an electricity outlet or not*), thus the interaction style does not seem to matter much in regards to completeness.

Overall, across question types, the completeness percentage is surprisingly higher and seems to not align with the low $m_{cCol}$ of the resulting KB. When we look further into this, it turns out this is caused by item creation with incomplete information and also the lack of completion of corresponding enrichment task which should have complement the missing information. A better task assignment approach for enrichment tasks would have to be designed such that it enforces to collect high quality and complete answers from the users. Additionally, we also think that the column completeness should be able to be improved by making questions to be required, not allowing users to skip the questions. Although this has a downside specifically when the user doesn't really know the answer of the required question, thus blocking them to continue answering the other questions.

Table 6.8: Completeness Across Non-Multiple Choice Question Types (B=Button Input, CF=Code & Free Text Input)

| Metric | Text | Numeric | Location | Image |
|---|---|---|---|---|
| Completeness | 0.85 | 0.84 | 1.00 | 1.00 |

Table 6.9: Completeness Across Multiple Choice Question Types (B=Button Input, CF=Code & Free Text Input)

| | MC Categorization | | MC Item | | MC Custom | |
|---|---|---|---|---|---|---|
| **Metric** | **B** | **CF** | **B** | **CF** | **B** | **CF** |
| Completeness | 1.00 | 1.00 | 1.00 | 0.92 | 0.97 | 1.00 |

### 6.1.5 Usability

From the survey link that had been sent to the participants email address, we acquired 10 responses consisting of 7 responses and 3 responses from participants who were assigned to CampusBot 1 and CampusBot 2 respectively. One participant, who were assigned to CampusBot 1, did not manage to download and try the chatbot. He stated that he did not have Telegram on his phone in addition to not being in campus that often to be able to actively use the chatbot.

We managed to gather the perceived System Usability Score (SUS) [17] from the 9 participants who did use the chatbot and responded to the survey. Figure 6.5 shows the SUS of each of the 9 participants. The average SUS calculated from these 6 participants is 66.39. If we refer to [10] on how to interpret this score, our chatbot system can be interpreted as "marginally acceptable", so although it is usable to some extent, there are certainly things to be improved. In order to understand more about why such score were given by the participants, we look into each of the participant activities and their answers to the survey.

In the survey, we gave a list of options for the respondents to pick on what are the challenges they encounter while using CampusBot. Five respondents (P3, P4, P5, P6, and P7) stated that they had a hard time on completing enrichment tasks. By looking through their activities and interaction with the chatbot, we found that most of the struggles lies on answering questions that they might not know the answer of. This shows in the way they skipped most of the questions, and only submitting answer for the "easy" ones (such as whether a place has an electricity outlet or not).

We also notice that one of them tried to input something which the chatbot deemed to be invalid. For example, when the chatbot asked the estimated maximum capacity of a place, one of the respondent answers with a range (e.g. *"6-8"*) instead of a single number (e.g. *"6"*). Our chatbot does not recognize this kind of answer, thus it prompted the user to input a single number instead. Although the user managed to submit the correct answer, this might result as an awkward experience from their perspective. This aligns with their agreement with the statement "*I found CampusBot very awkward to use*" in the survey.

Additionally, we also asked the respondents to choose on what was easy to do

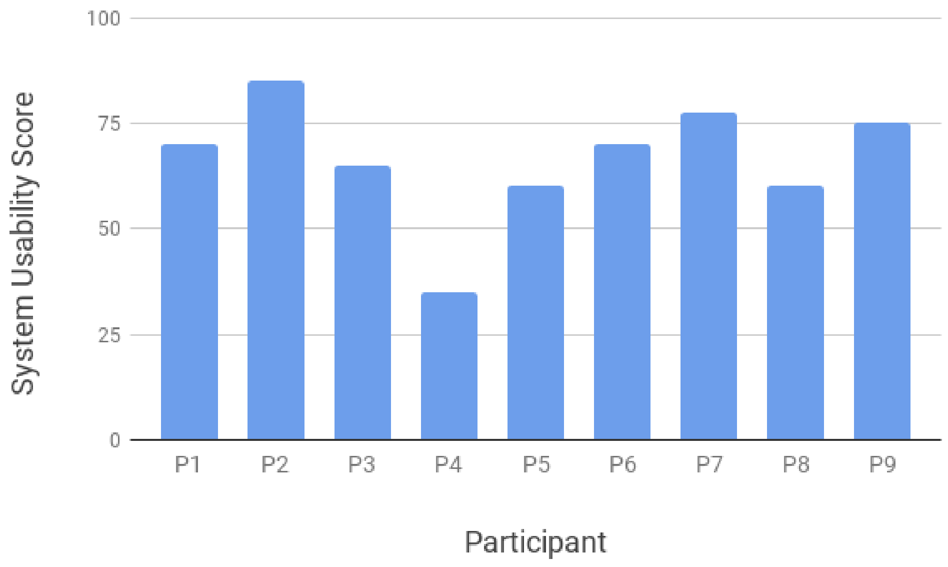


Figure 6.5: System Usability Score (SUS) Perceived by Survey Respondents of Study 1

when they use CampusBot. Five respondents (P4, P5, P6, and P7) chose "Starting the chatbot" as one of their answers. "Choosing a task" was also chosen by five respondents (P1, P3, P5, P6, and P7). Although this shows that these respondents perceived that starting the chatbot and choosing a task are straightforward, we still need to investigate whether this is really the case or because they might already be familiar with similar chatbots or Telegram bots in particular. Our survey did not collect the fact whether they have previous experience with chatbot and/or Telegram, so we would need to revisit this in Study 2.

We also received specific suggestions in regards on how to improve CampusBot. P2 discusses specifically on the timing of the push notification, which we set to be sent twice a day on 10 AM and 3 PM to remind users to do more tasks. P2 suggests to vary the time of the push notification as students tend to have a habit of being in the same place at certain time everyday, thus reducing the variation of places created and enriched everyday. Furthermore, P3's comment is somewhat aligned with P2's suggestion, stating the fact that they frequently gets the same place to enrich and validates everyday.

### 6.1.6 Issues Found on Study 1

From conducting the experiment for first study, we found several issues that results on the lack of completed tasks, causing the lack of data to do quantitative and statistical analysis. Each issue is discussed as follows.

**Many "no-show" participants**. As mentioned several times before, we found that there were many participants who signed up for the experiment but did not try the chatbot. Furthermore, there are also participants who tried the chatbot but did not complete any task. We found out through the survey we sent to the participants that one of the reason they did not try the chatbot is the need of installing Telegram and/or they are not often in campus. They assume they would need to be in campus when answering the question, which is ideally correct, though our experiment description should have mentioned that answering the question does not require the participants to be in the campus all the time.

**No suitable tasks for participants**. This is a follow up on the previous point, that might be the main reason on why some participants did not complete any task at all. We suspect that the tasks shown in the list are not suitable for them, for example the place tasks shown are not the places that they often come by, thus they did not complete any task.

**Same tasks everyday for place and trash bin tasks**. We notice that for place items and trash bin items, one of the participants commented through the online survey that they are offered the same task everyday resulting in having to input the same answer everyday for the same place. This might be caused by poor task generation and assignment strategy.

**Unbalanced number of completed tasks between the interaction styles**. By the end of the experiment, we found out that our participant assignment causes unbalanced number of completed tasks between CampusBot 1 and CampusBot 2. This might be the case that the number of active users in CampusBot 1 is higher than in CampusBot 2.

## 6.2 Results and Discussion on Study 2

The results from Study 1 were not sufficient to form a strong understanding on to what extent the designed chatbot could enable users to participate in a KBC process. The lack of enough number of task executions demands us to conduct a second experiment focusing on qualitative findings. We discuss the results of the second study in this section.

The audio recording of each interview session from Study 2 is transcribed and analyzed along with the recorded screen when the participant executes the requested tasks (create, enrich, and validate). We focus on analyzing what is intuitive to do and what is not based on the recorded interviews and the behavior of the user when interacting with the chatbot.

We interviewed a total of 7 people consisting of 3 participants who signed up for the first study but did not complete any tasks, as well as 4 additional participants who did not sign up for Study 1. 5 participants (P10, P11, P13, P14, P15) have stated that they haven't used Telegram before the experiment, while 2 participants (P12 and P16) are regular Telegram users. We assign 4 participants (P10, P11, P12, and P15) to use CampusBot 1, and 3 participants (P13, P14, P16) to use CampusBot 2. The followings are the findings that we have found from interviewing the 7 participants.

**Usability Score**. After the guided session, we asked the participants to fill a survey to measure their perceived SUS just like we did in Study 1. Figure 6.6 show the perceived SUS for each participant, resulting in an average SUS of 68.93. This confirms

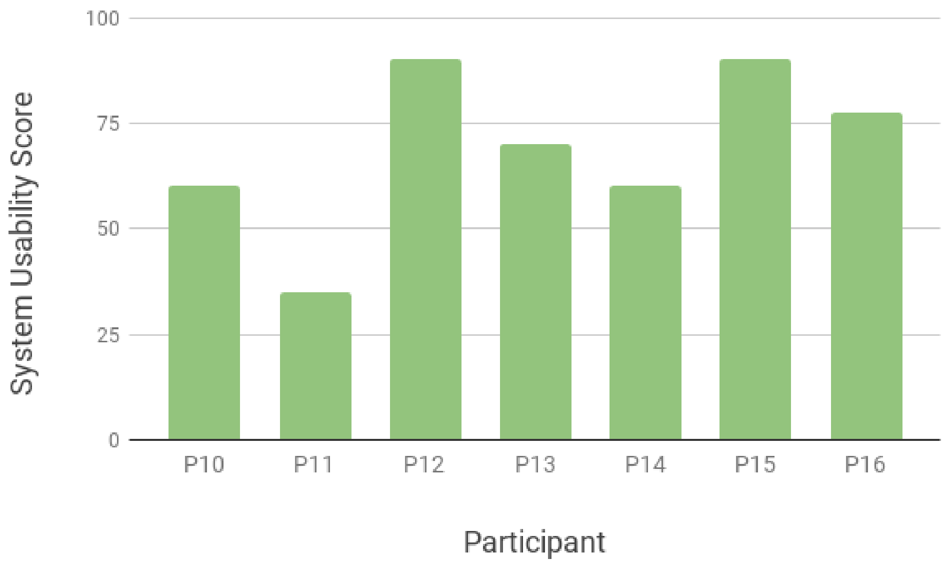


Figure 6.6: System Usability Score (SUS) Perceived by Survey Respondents of Study 2

the acceptable level of our system's usability as found in Study 1 – which can be interpreted as "marginally acceptable".

**Finding a task**. Apparently, we observe (P10) to be struggling, by the way they kept scrolling the task list back and forth, on what task should they choose among the ones shown in the task list. At the end of the session, we asked a follow-up question on why was it hard to choose a task. The participant instead suggested that the shown tasks should be relevant to them. This might indicate for a future work to a more context-aware task assignment, for example based on the user's current location. On regards on similar matter, P16 also expressed similar opinion, where they took their time to choose a task because they were not sure on what task to choose among the shown task list.

**Choosing a task**. The other observation on P10 shows that they weren't sure on how to choose the task, despite the instruction shown at the end of the task list (*"You can select a task by tapping on the task number"*). It turns out that, according to the participant, they were not aware that the instruction exists, because the instruction comes after the task list, instead of before the task list. P14 was also not sure at first on how to choose a task. They expected that tapping on the "chat bubble" of the task is the way to do it, but after reading through the instruction again, they managed to choose one of the task by typing the task number. On contrary, P13 and P15 didn't have such troubles on choosing a task. When asked about what they think about the interaction they had to make in order to choose a task, P13 stated that they are familiar with blue-colored "hyperlinks", so they just intuitively tapped on the task number which is blue-colored.

**Consistency of interaction style**. We also observed some struggles with different kind of interaction style offered by the chatbot. P13, who stated their familiarity with hyperlinks, expressed their opinion that the way to answer multiple choice questions should be similar to the way they choose the task, which is to tap on the answer's number instead of typing the number of the answer. P15, who was assigned to CampusBot 2 which uses CF interaction style for multiple choice questions, highlighted the fact that there is an inconsistency between when they were asked to choose a category (MC Categorization) and when they were asked to choose a building (MC Item). In the former, they could just type the number of the answer they intended to submit, while for the latter, the number of the answer was not presented so they had to type in the full answer.

**Unclear questions**. Once in a while, when we observed that a participant took some time to answer a certain question, we tried to ask whether the cause of their pause was because the question is unclear. Several participants had trouble on answering the "*How do we get to the place?*" question where they stated that the question is unclear and they were not sure on how to answer it not until we explain to them the intention of the question. This shows the importance of clearly defined questions, especially in conversational setting where the users expect to be asked in the most clear and natural way as possible.

**Getting used to the chatbot**. P10 expressed their opinion that, although the chatbot was fun to use, they think it was a little confusing at first especially on the navigation part. P11 also express similar opinion where they stated that they needed some time to get used to the chatbot and answer the questions. P13 expressed that they just needed a short time to get used on how to use the chatbot, and once they have gotten used to it, they think the chatbot is fairly easy to use. Both P15 and P16 also had similar opinion – they think the chatbot is not hard to use and people can get used to it real soon without detailed explanation.

**Other suggested improvements**. There are suggestions that some of the participants gave that they think should improve the overall chatbot experience. First, P13 suggests to vary the wording the chatbot used so it would feel more natural and not robot-like. Responding to user's answer naturally is also important, for example giving feedback like "*Nice to know!*" instead of a robot-like response like "*Ok*". P16 expressed their suggestion on how the chatbot should not only collect data but also allows users to check on the summary of the collected data so far. Additionally, they also suggested that the questions should show multiple choice options based on common answers in addition of allowing users to input their own answer.

## 6.3 Discussions on Study 1 and Study 2

In this section, we will discuss various findings from both studies and how they complement each other. The discussion will mostly focus on how the independent variables (interaction style and task type) affect the construction of a knowledge base in regards to its execution time, accuracy, and completeness. The discussion draws from mostly quantitative results of Study 1 and complemented by qualitative findings from Study 2.

Regarding the execution time, Study 1 shows while button-based interaction style enforces faster execution time compared to text-based interaction style, the execution time was mostly affected by the number of questions in a single task execution as well as the type of question being asked. For example, asking the question of "*What is the route to get to the place?*" takes more time than "*What is the name of the place?*". This was also confirmed in Study 2, where we observe that users seem to take more time on thinking for coming up with the answer for questions that demands answer for longer type. These finding can be a motivation for future work and aligns with one of the participant suggestion to incorporate more choice-based questions, where the chatbot shows a list of suggested answers. We think this improvement would speed up the execution time of task completion.

For the accuracy of the inputted answers, we found out that there is no apparent difference between using buttons and CF interaction style for multiple-choice questions. We found that the accuracy is mostly affected by the type of question asked, where non-multiple choice questions (text, numeric, location, and image) are more prone to producing incorrect answers compared to multiple-choice questions disregard of the interaction style.

The one-to-one guided sessions with some participants also show that incorrect answers tend to emerge when a user is faced with ambiguous questions (e.g. the route to the place, whether a place has electricity outlet). Some users find the need to ask for clarification for this kind of question to the interviewer before proceeding to answer them.

We also measure the completeness degree of KB generated from Study 1. We found that answers from different users for the same item complement each other, thus more users tend to result in more complete info for an item. This shows by the fact that the completeness degree of the combined KB from both CampusBot 1 and CampusBot 2 is higher than the KB generated from separate answers of each version of the chatbot.

Interaction style does not seem to matter much in regards to the completeness, as we can see in Table 6.9 that multiple-choice questions tend to results in high completeness degree. Perfect completeness degree (i.e. 1.0) mostly resulted from questions which are marked as required, in which the users could not skip.

We investigate the effect of required and non-required questions towards KB completeness in Study 2. We observed that most users tend to do their best answering all questions (despite the fact that the question is not required). They only skipped if they really think they do not know the answer to a question. Thus, although from Study 1 we can see a perfect degree of completeness for required questions, Study 2 shows that marking a question to be not required (i.e. allowing users to skip the question) does not necessarily encourage the user to skip the questions.

To understand more on whether the implemented system is usable for constructing a knowledge base, we collected survey responses to measure the System Usability Score (SUS). Study 1 and Study 2 results in an average perceived SUS of 66.39 and 68.93 respectively and both can be interpreted as "marginally acceptable".

After we look into this further, based on conversation log of users from Study 1 and observation on users in Study 2, we found that the usability issues mostly come from the fact that the users seem to not be able to found relevant tasks for them. This is more of a result on the task assignment, which we acknowledge can be improved for future work, and less of a UI or UX design issue as we have already, in fact, tested our prototype in a controlled experiment with some colleagues before publishing the chatbot. On the contrary, using the chatbot and executing the task itself weren't found to be a problem, as observed in Study 2 where users get used to the chatbot in a short time despite having no prior experience with Telegram.

## 6.4 Threats to Validity

We conducted two experiments in order to investigate to what extent our system can be used to enable participation of crowdsourced construction of knowledge bases. Although we try our best to minimize "external" factors outside our selected independent variables that might influence the results, we acknowledge that there are still potential threats to the validity of the results. We discuss the threats as follows.

**Representation of KBC participants.** Our participants were limited to students of TU Delft due to the design of the experiment which is to construct a knowledge base for the campus. Thus, our results might not be representative of other potential workers for constructing knowledge bases for other domains. However, our work can easily be extended to validate whether similar understandings can be found when applied to other domains.

**Majority of task executions from most active users.** There is quite a gap of completed task executions from Study 1 between the most active users (4 users completed around 14-19 task executions) and less active users (20 users completed less than 5 task executions). This might cause biased results where the majority of task executions for execution time and accuracy measurement were taken from the four most active users. However, when we measured the execution time and accuracy across question types, we try our best to sample across different users as well as confirming our understanding based on observation in Study 2.

**Telegram as the messaging platform.** We chose Telegram as the messaging platform of the chatbot because it requires the least time to set up compared to other platforms, such as Messenger, that requires more time for the application approval process. We are aware that this decision of using a specific messaging platform might introduce bias to our results caused by participants' familiarity (or non-familiarity) with Telegram. We try our best to reduce this bias by designing the chatbot to be as general as possible, using common features supported by other messaging platforms.

**Limited variations of interaction style and task type.** We acknowledge the fact that the experimental design only includes limited variations of interaction style as well as task type. Although we believe the experiments were well designed to answer our research question, there is a potential to extend the research to include more variety of interaction style and task type in the future.

# Chapter 7

# Conclusions and Future Work

In this chapter, we present our conclusion to answer our main research question by answering the three research subquestions stated in the Introduction chapter. We also discuss opportunities of future work in order to utilize the potential of conversational crowdsourcing for KBC even more.

## 7.1 Conclusions

In order to answer our main research question **"How could a text-based conversational agent be designed to enable the crowdsourced construction of knowledge bases?"**, we first summarize the conclusion for each of the three subquestions as follows.

#### RQ 1: What is the state of the art in knowledge base construction and conversational crowdsourcing?

We did a **literature survey (C1)** on previous work within the intersections of three fields: KBC, crowdsourcing, and conversational interface. We discussed several well-known methodologies for constructing a knowledge base and managed to identify primitive elements (classes, instances, attributes, relations, hierarchies, axioms, and rules) of a knowledge base as well as activities involved in order to construct a KB from scratch.

We also discuss work which investigates the viability of utilizing crowdsourcing technique for KBC process. Previous work in this research line mostly focuses on either the microtasking approach, which utilize monetary incentives or the Game with a Purpose (GWAP) approach, which promotes entertainment incentives. Both approaches try to present one or more KBC tasks into the crowdsourcing workflow.

Finally, we discuss a research direction on which text-based conversational agents (*chatbots*) are used to execute microtask crowdsourcing. We also discuss some previous work exploring the potential of chatbots for knowledge acquisition.

We based our work on these findings, designing a crowdsourcing workflow based on existing KBC methodologies as well as validating the findings of similar work of chatbot for microtask crowdsourcing but specifically in the context of constructing KBs.

**RQ 2: How could a text-based conversational agent be designed and implemented to support the microtask crowdsourcing for knowledge base construction?**

Our contribution to address this research question is a **System Design of a Conversational Agent for Crowdsourced KBC (C2)** and consist of three main components:

- A crowdsourcing workflow which incorporates KBC microtasks decomposed from state-of-the-art KBC methodologies. The workflow consists of three main stages: **Create**, **Enrich**, and **Validate**. In Create stage, KB items are defined and initial statements are added. In Enrich stage, additional statements are added to make the item more complete. Finally, in Validate stage, the correctness of added statements are validated before being aggregated into a constructed KB.

- A system design of a conversational crowdsourcing system which incorporates the designed crowdsourcing workflow in order to support the construction of KBs. The system consists of a conversational interface, a task executioner component to control the flow of task execution, a database, as well as a KB generator module to construct a KB from the acquired knowledge.

- An implementation of the system (called CampusBot) for the specific use case of Campus domain specifically Food, Places, Course Questions, and Trash Bins. This implementation enabled us to carry an evaluation on the system design, which helped us addressing RQ3.

**RQ 3: How does the interaction style of the text-based conversational agent affect the construction time and quality of the KB?**

We contribute an **Evaluation of Chatbot System for KBC (C3)** in order to address this research question. We carried out two experiments by recruiting participants to use CampusBot to participate in the construction of a campus knowledge base.

The first experiment involves publishing CampusBot to students of TU Delft and invites them to use CampusBot to create campus-related items and complete the several tasks. 43 participants were involved in trying out the chatbot in which 24 of them completed at least one task execution. Then we measure the execution time, answers accuracy, and completeness of the collected answers. We also measure the usability of the system by sending an online post-experiment survey, which results in an average System Usability Score (SUS) of 66.39, deeming our system to be marginally acceptable.

In the second experiment, we invite participants to a one-to-one guided session and observe and talk to them regarding their experience with the chatbot. This second experiment was carried to conduct a qualitative analysis and complement the quantitative findings from the first experiment.

We contribute to the existing line of work by confirming similar results with related work but specifically in the context of KBC. We found several findings such as: button-based interaction style enforces faster execution time (on average 5 seconds faster) then text-based interaction style; as well as the answers accuracy (varied between 0.66 and 0.98 across question types) are comparable despite different interaction type and question type. These findings align with previous work specifically on the resulting execution time and answers accuracy.

Additionally, we also include a KB-specific metric, the completeness metric, into the set of metrics to evaluate our system. We measure the schema and column completeness metric based on the definition in previous work on KB assessment [60], and use it to evaluate the KB generated from the collected information by our chatbot system. The constructed KB results in a schema completeness degree of 0.94 for place items and 1.00 for food items. The column completeness degrees are fairly low for KB generated from each version of chatbot independently (between 0.20 to 0.45), but once the combined KB is generated the column completeness improves (0.63 for place items and 0.45 for food items). This leads us to conclude that the Create and Enrich stages of our crowdsourcing workflow enable users to indirectly complement each other for completing missing information of a knowledge base item.

To summarize, to design a chatbot system for enabling crowdsourced construction of knowledge bases, we first need to know the state-of-the-art of KBC methodologies. The methodologies were used to decompose KBC task into microtasks which in turn are incorporated into a crowdsourcing workflow. To execute the workflow, we would need a conversational crowdsourcing system for KBC to be designed. Finally, the system was evaluated to make sure that it can produce a KB with acceptable execution time, quality, and completeness. The results of our experiments show that our system is viable to be used for constructing KBs, though of course with room for improvement which we left as potential future work.

## 7.2 Future work

As mentioned throughout the the whole report, there is a lot of future work opportunities based on our work. We summarize and describe each of them below.

**Experiment with more task execution.** The limitation of our work is the lack of enough samples to generate stronger findings by means of statistical analysis. A potential future work would be to carry a similar experiment within a more controlled setting, where a larger sample of task execution can be collected thus more quantitative measurements can be analyzed.

**Incorporate other types of KBC microtask.** Although we defined an exhaustive list of KBC microtasks in Section 3.1, for the sake of limiting the scope of our work, we only incorporate some of them into our crowdsourcing workflow. The microtask involving the definition of concepts has not yet been explored in conversational settings. It would be interesting to deploy this type of microtasks where users can contribute not only by creating instances but also defining new concepts.

**Integration with existing ontologies.** In our work, we briefly describe a knowledge base generation module which connects to a Wikibase instance. This is a bridge to the potential of linking the constructed knowledge base with existing ontologies such as the Wikidata ontology. This would promote a new way to contribute to the linked data available on the internet.

**Comparison with other KBC tools.** Although to some extent, we found that a chatbot system can be designed to crowdsource the construction of a knowledge base, there is a need to find a more evident proof that using a chatbot is comparable or more superior compared to using existing KBC tools such as Protege or wiki-based interface. An idea of future work is to compare the difference of the resulting KB between using the chatbot and using existing KBC tools.

**Querying the constructed KB through the conversational agent.** A recurring discussion from the participants of our experiment is that some of them expect to not only be able to submit answers to the chatbot but also acquire answers from it. Our work's scope is limited to executing microtasks to construct a KB, but it would be interesting if the chatbot functionalities are extended so it can be used to query the constructed KB.

# Bibliography

[1] Controlled and uncontrolled English for ontology editing. In Emmons I., Laskey K.B., Costa P.C.G., and Oltramari A., editors, *10th Conference on Semantic Technology for Intelligence, Defense, and Security, STIDS 2015*, volume 1523, pages 74–81, CUBRC, Buffalo, NY, United States, 2015. CEUR-WS.

[2] Chatbot who wants to learn the knowledge: KB-agent. volume 2241, pages 33–36, Semantic Web Research Center, School of Computing, KAIST, 291 Daehak-ro, Yuseong-gu, Deajeon, South Korea, 2018. CEUR-WS.

[3] Albin Ahmeti, Victor Mireles, Artem Revenko, Marta Sabou, and Martin Schauer. Crowdsourcing updates of large knowledge graphs. In *CEUR Workshop Proceedings*, volume 2169, pages 1–6, 2018.

[4] Albin Ahmeti, Simon Razniewski, and Axel Polleres. Assessing the completeness of entities in knowledge bases. In *European Semantic Web Conference*, pages 7–11. Springer, 2017.

[5] R Amini, M Cheatham, P Grzebala, and H B McCurdy. Towards best practices for crowdsourcing ontology alignment benchmarks. In *CEUR Workshop Proceedings*, volume 1766, pages 1–12, Data Semantics Laboratory, Wright State University, Dayton, OH, United States, 2016.

[6] Ram G. Athreya, Axel-Cyrille Ngonga Ngomo, and Ricardo Usbeck. Enhancing Community Interactions with Data-Driven Chatbots–The DBpedia Chatbot. *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18*, pages 143–146, 2018.

[7] A Augello, G Pilato, A Machi, and S Gaglio. An Approach to Enhance Chatbot Semantic Power and Maintainability: Experiences within the FRASI Project. In *2012 IEEE Sixth International Conference on Semantic Computing*, pages 186–193, 2012.

[8] A Augello, G Pilato, G Vassallo, and S Gaglio. A Semantic Layer on Semi-Structured Data Sources for Intuitive Chatbots. In *2009 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 760–765, 2009.

[9] A Augello, G Pilato, G Vassallo, and S Gaglio. Chatbots as interface to ontologies, 2014.

[10] Aaron Bangor, Philip T. Kortum, and James T. Miller. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, jul 2008.

[11] A Basharat, I B Arpinar, S Dastgheib, U Kursuncu, K Kochut, and E Dogdu. CrowdLink: Crowdsourcing for large-scale linked data management. In *Proceedings - 2014 IEEE International Conference on Semantic Computing, ICSC 2014*, pages 227–234, Large Scale Distributed Information Systems (LSDIS) Lab., Department of Computer Science, University of Georgia, Athens, GA 30602, United States, 2014.

[12] Senjuti Basu Roy, Ioanna Lykourentzou, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. Task assignment optimization in knowledge-intensive crowdsourcing. *The VLDB JournalâThe International Journal on Very Large Data Bases*, 24(4):467–491, 2015.

[13] Juan Blázquez, M. Fernández, JM. García-Pinar, and A. Gómez-Pérez. Building Ontologies at the Knowledge Level using the Ontology Design Environment, 1998.

[14] Luka Bradeško, Michael Witbrock, Janez Starc, Zala Herga, Marko Grobelnik, and Dunja Mladenić. Curious Cat–Mobile, Context-Aware Conversational Crowdsourcing Knowledge Acquisition. *ACM Transactions on Information Systems*, 35(4):1–46, aug 2017.

[15] Jonathan Bragg, Andrey Kolobov, Mausam, and Daniel Weld. Parallel task routing for crowdsourcing. In *Proceedings of the Second AAAI Conference on Human Computation and Crowdsourcing (HCOMP-14)*. AAAI Press, November 2014.

[16] Simone Braun, Andreas Schmidt, and Andreas Walter. Ontology Maturing: a Collaborative Web 2.0 Approach to Ontology Engineering. *WWW 2017*, (August), 2007.

[17] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[18] Irene Celino, Simone Contessa, Marta Corubolo, Daniele Dell'Aglio, Emanuele Della Valle, Stefano Fumeo, and Thorsten Krüger. UrbanMatch - linking and improving smart cities data. In *CEUR Workshop Proceedings*, volume 937, CEFRIEL, Italy, 2012.

[19] Philipp Cimiano and Heiko Paulheim. Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. Technical report, 2016.

[20] John P Dickerson, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. Assigning tasks to workers based on historical data: Online task assignment with two-sided arrivals. In *Proceedings of the 17th International Conference*

*on Autonomous Agents and MultiAgent Systems*, pages 318–326. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[21] Zhaoan Dong, Ju Fan, Jiaheng Lu, Xiaoyong Du, and Tok Wang Ling. Using Crowdsourcing for Fine-Grained Entity Type Completion in Knowledge Bases. pages 248–263. Springer, Cham, jul 2018.

[22] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. Introducing wikidata to the linked data web. In *International Semantic Web Conference*, pages 50–65. Springer, 2014.

[23] M Fernández-López, A Gómez-Pérez, and Natalia Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Technical report, 1997.

[24] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. The Rise of Social Bots. 2014.

[25] Evgeniy Gabrilovich and Nicolas Usunier. Constructing and mining web-scale knowledge graphs. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 1195–1197, New York, NY, USA, 2016. ACM.

[26] Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering*. Advanced Information and Knowledge Processing. Springer-Verlag, London, 2004.

[27] Martin Hepp, Daniel Bachlechner, and Katharina Siorpaes. Ontowiki: Community-driven ontology engineering and ontology usage based on wikis. In *Proceedings of the 2006 International Symposium on Wikis*, WikiSym '06, pages 143–144, New York, NY, USA, 2006. ACM.

[28] T.-H. Ting-Hao T.-H. Huang, J C Chang, Jeffrey P Bigham, Joseph Chee Chang, and Jeffrey P Bigham. Evorus: A Crowd-powered conversational assistant built to automate itself over time. In *2018 CHI Conference on Human Factors in Computing Systems, CHI 2018*, volume 2018-April, Language Technologies Institute, Human-Computer Interaction Institute, Carnegie Mellon University, United States, 2018. Association for Computing Machinery.

[29] Sarath Kumar Kondreddi, Peter Triantafillou, and Gerhard Weikum. Combining information extraction and human computing for crowdsourced knowledge acquisition. In *2014 IEEE 30th International Conference on Data Engineering*, pages 988–999. IEEE, mar 2014.

[30] Sarath Kumar Kondreddi, Peter Triantafillou, Gerhard Weikum, Sarath Kumar Kondreddi, Peter Triantafillou, and Gerhard Weikum. Human computing games for knowledge acquisition. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management - CIKM '13*, pages 2513–2516, New York, New York, USA, 2013. ACM Press.

[31] Tobias Kuhn. AceWiki: A natural and expressive semantic wiki. In *CEUR Workshop Proceedings*, volume 543, 2009.

[32] Walter S Lasecki. Real-Time Conversational Crowd Assistants. In Beaudouin-Lafon M., Baudisch P., and Mackay W.E., editors, *31st Annual CHI Conference on Human Factors in Computing Systems:, CHI EA 2013*, volume 2013-April, pages 2725–2730, ROC HCI, Computer Science, University of Rochester, United States, 2013. Association for Computing Machinery.

[33] Walter S Lasecki, Rachel Wesley, Jeffrey P Bigham, and Anand Kulkarni. Speaking with the crowd. In *25th Annual ACM Symposium on User Interface Software and Technology, UIST 2012*, pages 25–26, University of Rochester, Computer Science, ROC HCI, United States, 2012.

[34] Edith Law and Luis von Ahn. *Human Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2011.

[35] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[36] X Liang, R Ding, M Lin, L Li, X Li, and S Lu. CI-Bot: A hybrid chatbot enhanced by crowdsourcing, 2017.

[37] Panagiotis Mavridis, Owen Huang, Sihang Qiu, Ujwal Gadiraju, and Alessandro Bozzon. Chatterbox: Conversational interfaces for microtask crowdsourcing. In *UMAP*, 2019.

[38] Jonathan M Mortensen, Mark A Musen, and Natalya F Noy. Developing crowd-sourced ontology engineering tasks: An iterative process. In *CEUR Workshop Proceedings*, volume 1030, pages 79–88, 2013.

[39] Claudia Müller-Birn, Benjamin Karran, Janette Lehmann, and Markus Luczak-Rösch. Peer-production system or collaborative ontology engineering effort. In *Proceedings of the 11th International Symposium on Open Collaboration - OpenSym '15*, pages 1–10, 2015.

[40] Prayag Narula, Philipp Gutheim, David Rolnitzky, Anand Kulkarni, and Bjoern Hartmann. Mobileworks: A mobile crowdsourcing platform for workers at the bottom of the pyramid. In *Proceedings of the 11th AAAI Conference on Human Computation*, AAAIWS'11-11, pages 121–123. AAAI Press, 2011.

[41] Natalya F Noy and Deborah L Mcguinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical report, 2001.

[42] Natalya F. Noy, Jonathan Mortensen, Mark A. Musen, and Paul R. Alexander. Mechanical turk as an ontology engineer?: using microtasks as a component of an ontology-engineering workflow. In *Proceedings of the 5th Annual ACM Web Science Conference*, volume volume, pages 262–271, Stanford University, Stanford, CA 94305, United States, 2013. ACM Press.

[43] Thomas Pellissier Tanon, Denny Vrandečić, Sebastian Schaffert, Thomas Steiner, and Lydia Pintscher. From freebase to wikidata: The great migration. In *Proceedings of the 25th international conference on world wide web*, pages 1419–1428. International World Wide Web Conferences Steering Committee, 2016.

[44] Boonsita Roengsamut, Kazuhiro Kuwabara, and Hung Hsuan Huang. Toward gamification of knowledge base construction. In *INISTA 2015 - 2015 International Symposium on Innovations in Intelligent SysTems and Applications, Proceedings*, pages 1–7. IEEE, sep 2015.

[45] Cristina Sarasua, Elena Simperl, and Natalya F. Noy. CrowdMap: Crowdsourcing Ontology Alignment with Microtasks. pages 525–541. Springer, Berlin, Heidelberg, nov 2012.

[46] Cristina Sarasua, Elena Simperl, Natasha F. Noy, Abraham Bernstein, and Jan Marco Leimeister. Crowdsourcing and the Semantic Web: A Research Manifesto. *Human Computation*, 2(1):3–17, 2015.

[47] Sebastian Schaffert. Ikewiki: A semantic wiki for collaborative knowledge management. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE'06. 15th IEEE International Workshops on*, pages 388–396. IEEE, 2006.

[48] Arno Scharl, Marta Sabou, and Michael Föls. Climate quiz: a web application for eliciting and validating knowledge from social networks. … *symposium on Multimedia and the web*, pages 189–192, 2012.

[49] Alessandro Seganti, Paweł Kapłański, and Piotr Zarzycki. Collaborative editing of ontologies using fluent editor and ontorion. In *International Experiences and Directions Workshop on OWL*, pages 45–55. Springer, 2015.

[50] E Simperl, B Norton, and D Vrandečić. Crowdsourcing tasks in Linked Data management. In *CEUR Workshop Proceedings*, volume 782, Institute AIFB, Karslruhe Institute of Technology, Germany, 2011.

[51] Elena Simperl and Markus Luczak-Rösch. Collaborative ontology engineering: A survey. *Knowledge Engineering Review*, 29(1):101–131, 2014.

[52] Katharina Siorpaes and Martin Hepp. myontology: The marriage of ontology engineering and collective intelligence. *Bridging the Gep between Semantic Web and Web*, 2:127–138, 2007.

[53] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM.

[54] Christoph Tempich, Elena Simperl, Markus Luczak, Rudi Studer, and H Sofia Pinto. Argumentation-based ontology engineering. *IEEE Intelligent Systems*, (6):52–59, 2007.

[55] S Thaler, E Simperl, and S Wölger. An experiment in comparing human-computation techniques. *IEEE Internet Computing*, 16(5):52–58, 2012.

[56] D Toniuc and A Groza. Climebot: An argumentative agent for climate change. In *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 63–70, 2017.

[57] Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, September 2014.

[58] Gerhard Wohlgenannt, Marta Sabou, and Florian Hanika. Crowd-based ontology engineering with the uComp Protégé plugin. *Semantic Web*, 7(4):379–398, 2016.

[59] Amrapali Zaveri, Dimitris Kontokostas, Universität Leipzig, and Sebastian Hellmann. Linked Data Quality of DBpedia , Freebase, OpenCyc, Wikidata, and YAGO. Technical Report 0, 2017.

[60] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for linked data: A survey. *Semantic Web*, 7(1):63–93, 2016.

# Appendix A

# Entity Relationship Diagram

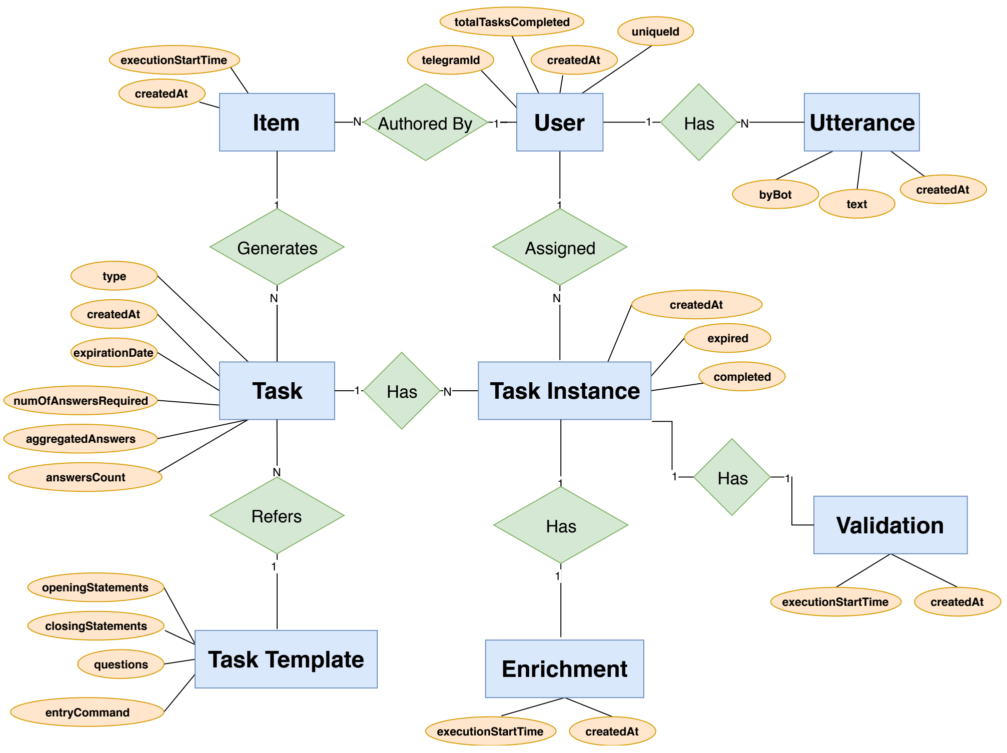


Figure A.1: CampusBot Entity Relationship Diagram

# Appendix B

# Knowledge Model

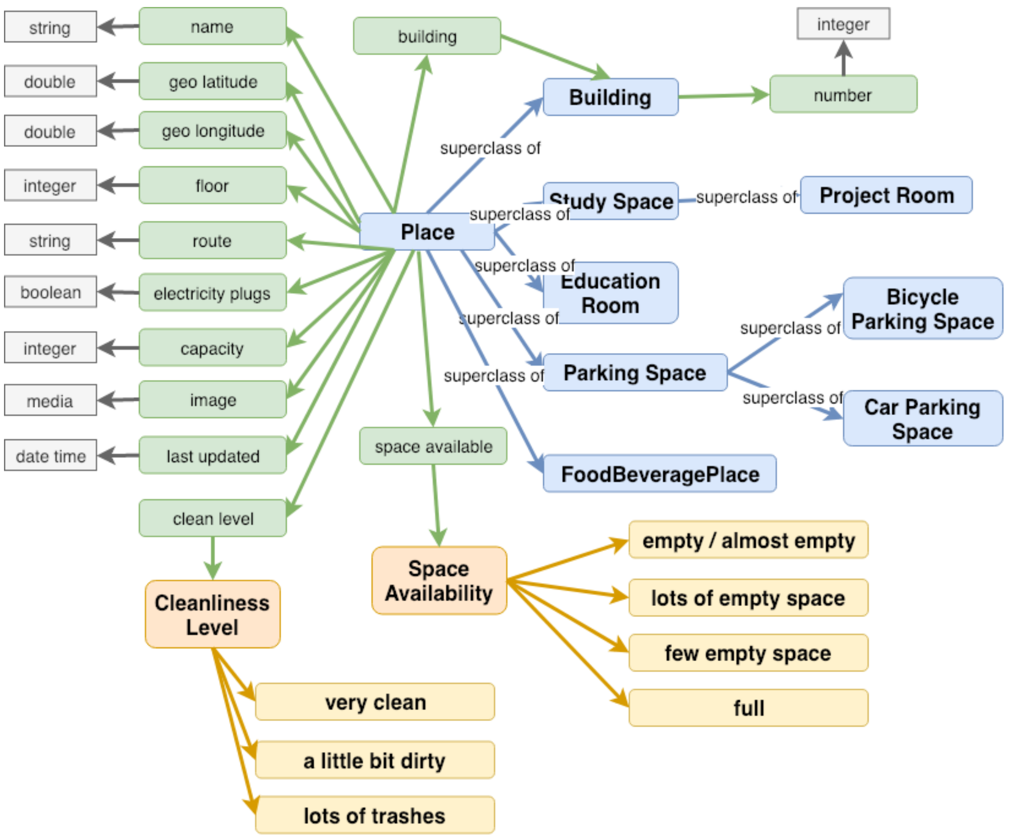


Figure B.1: Knowledge Model for Place Use Case

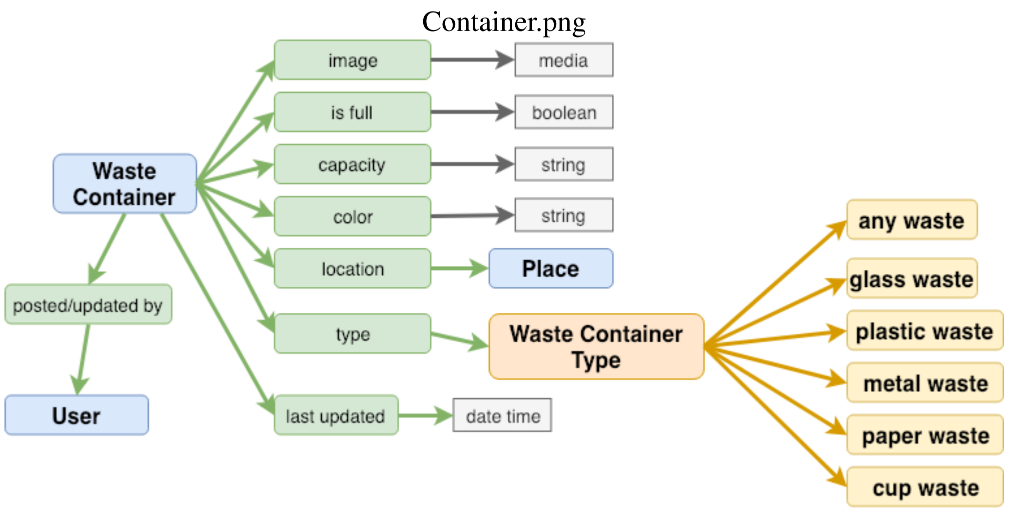


Figure B.2: Knowledge Model for Trash Bin Use Case

# Appendix C

# Wikibase Properties and Categories

| Category Name | Subclass of | Use Case Domain |
|---|---|---|
| place | - | place |
| building | place | place |
| parking space | place | place |
| car parking space | place | place |
| food establishment | place | place, food |
| lecture room | place | place |
| project room | place | place |
| study space | place | place |
| meal | - | food |
| food establishment | - | food |
| trash bin | - | trash bin |
| waste type | - | trash bin |
| general waste | waste type | trash bin |
| paper cup waste | waste type | trash bin |
| other waste type | waste type | trash bin |
| question | - | course |
| course | - | course |

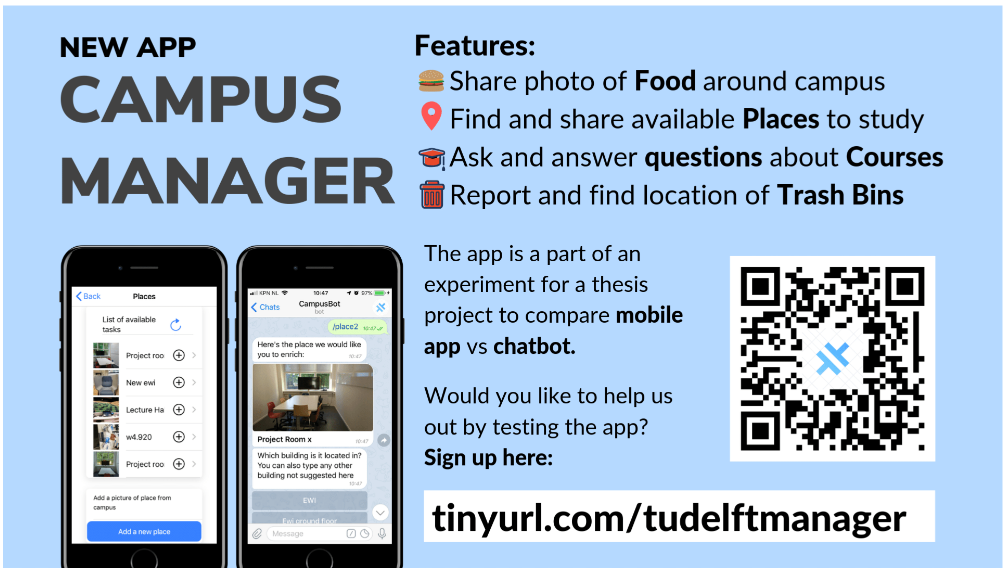
Table C.1: List of Predefined Categories in the Wikibase Instance

| **Property Label** | **Property Description** | **Data Type** |
|---|---|---|
| subclass of | all instances of these items are instances of those items; this item is a class (subset) of that item. | wikibase-item |
| instance of | that class of which this subject is a particular example and member (subject typically an individual member with a proper name label) | wikibase-item |
| image | image of relevant illustration of the subject | url |
| building | the building in which a place is a part of | wikibase-item |
| coordinate location | geocoordinates of the item | globe-coordinate |
| building number | the number associated with this building | string |
| floor number | the floor number which the place is located on | string |
| route | the route or direction on how to go to the place | string |
| has electricity outlet | depicts whether the place has electricity socket for public use or not | string |
| maximum capacity | number of people allowed for a venue or vehicle | quantity |
| available seats | the currently available number of space allowed for additional humans or vehicles | string |
| clean level | the subjective level of cleanliness of a place | string |
| point in time | time and date something took place, existed or a statement was true | time |
| waste type | type of a waste container | wikibase-item |
| size | size of a trash bin (subjective) | string |
| is full | indication whether a trash bin is currently full or not | string |
| color | the color of an item | wikibase-item |
| course code | the unique code of a study course | string |
| course | the study course associated with this item | wikibase-item |
| answer | the answer to a question | string |
| meal items | the (food) item contained in this meal | wikibase-item |
| price | published price listed or paid for a product (use with unit of currency) | quantity |
| food location | the location where the food was bought from | wikibase-item |

Table C.3: List of Predefined Properties in the Wikibase Instance

# Appendix D

# Participant Recruitment - Study 1



NEW APP

**CAMPUS MANAGER**

**Features:**

🍔 Share photo of **Food** around campus

📍 Find and share available **Places** to study

🎓 Ask and answer **questions** about **Courses**

🗑 Report and find location of **Trash Bins**

The app is a part of an experiment for a thesis project to compare **mobile app** vs **chatbot.**

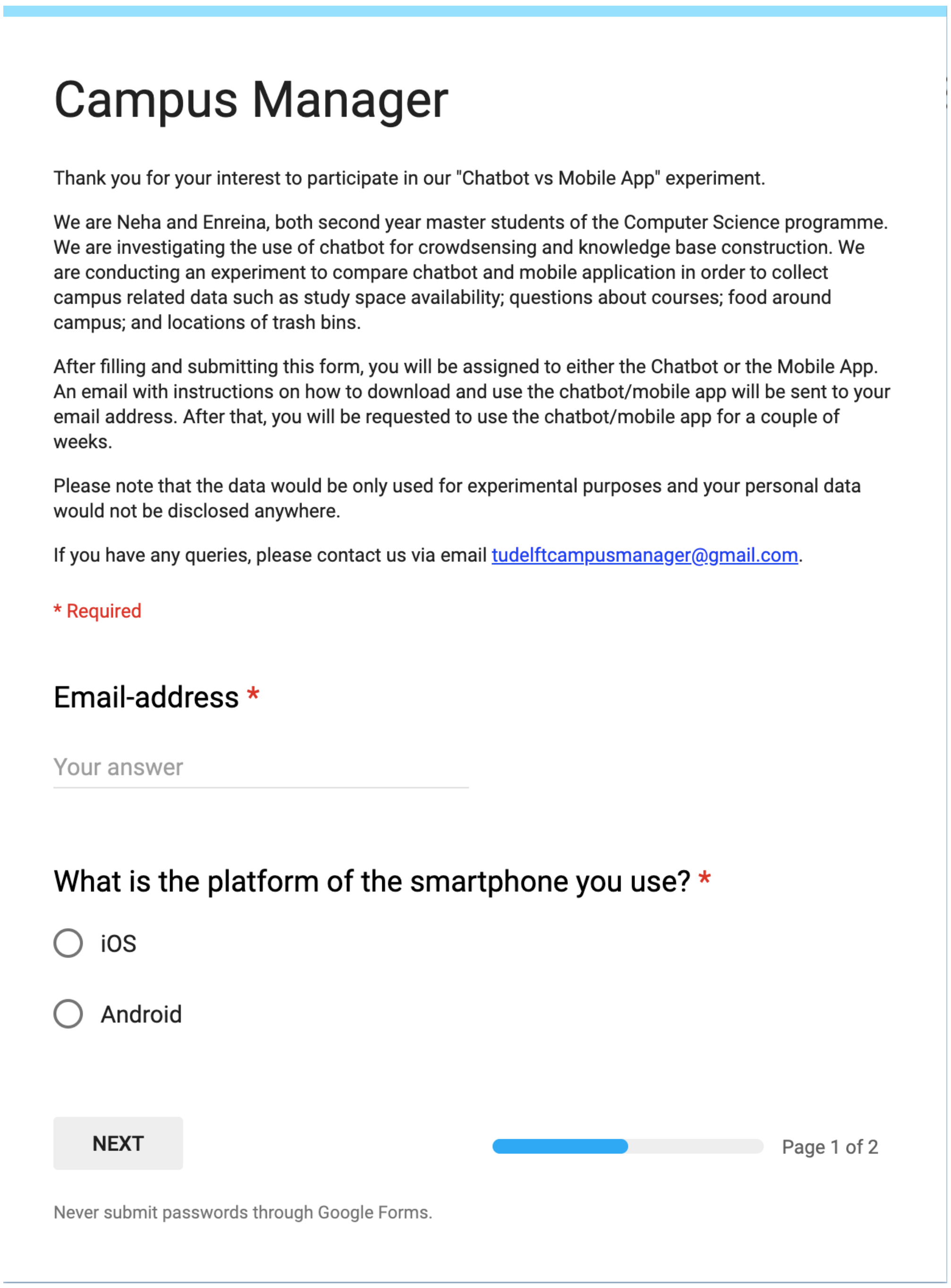
Would you like to help us out by testing the app?

**Sign up here:**

**tinyurl.com/tudelftmanager**

Back
Places
List of available tasks
Project roo
New ewi
Lecture Ha
w4.920
Project roo
Add a picture of place from campus
Add a new place
Chats
CampusBot
/place2
Here's the place we would like you to enrich:
Project Room x
Which building is it located in? You can also type any other building not suggested here
EWI
Ewi ground floor
Message

Figure D.1: Experiment Advertisement Poster



Campus Manager

Thank you for your interest to participate in our "Chatbot vs Mobile App" experiment.

We are Neha and Enreina, both second year master students of the Computer Science programme. We are investigating the use of chatbot for crowdsensing and knowledge base construction. We are conducting an experiment to compare chatbot and mobile application in order to collect campus related data such as study space availability; questions about courses; food around campus; and locations of trash bins.

After filling and submitting this form, you will be assigned to either the Chatbot or the Mobile App. An email with instructions on how to download and use the chatbot/mobile app will be sent to your email address. After that, you will be requested to use the chatbot/mobile app for a couple of weeks.

Please note that the data would be only used for experimental purposes and your personal data would not be disclosed anywhere.

If you have any queries, please contact us via email tudelftcampusmanager@gmail.com.

* Required

Email-address *

Your answer

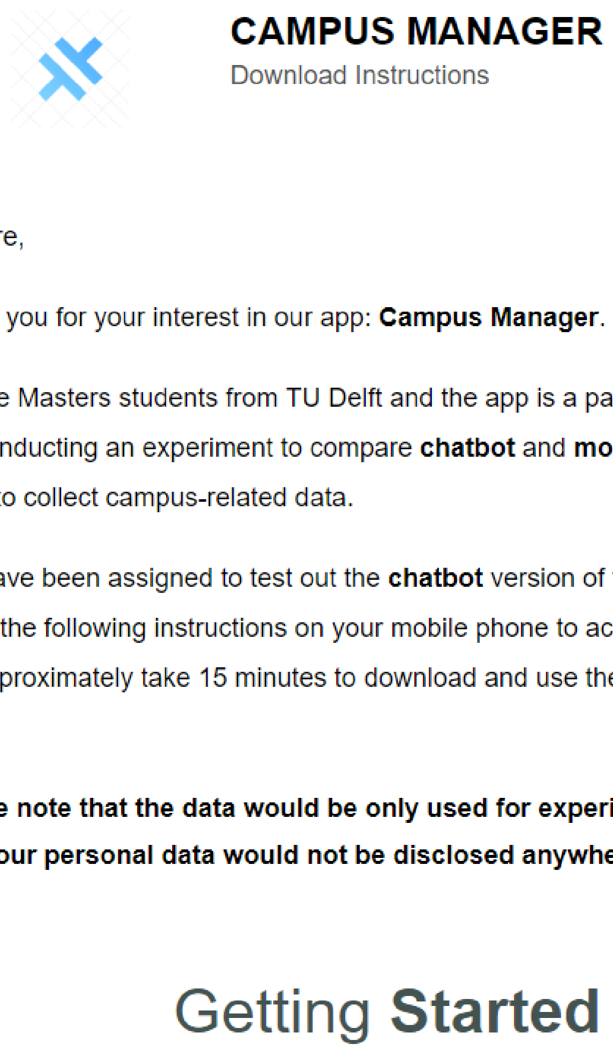
What is the platform of the smartphone you use? *

○ iOS

○ Android

Never submit passwords through Google Forms.

Figure D.2: Sign Up Form for Recruiting Participants



**CAMPUS MANAGER**

Download Instructions

Hi there,

Thank you for your interest in our app: **Campus Manager**.

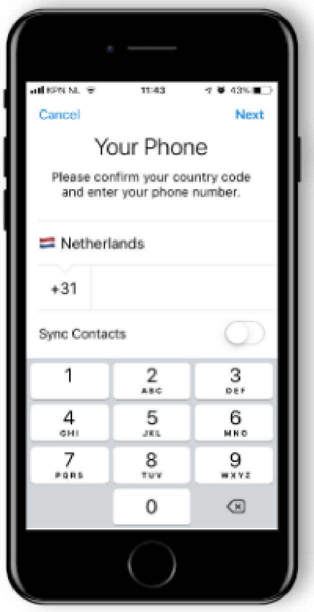
We are Masters students from TU Delft and the app is a part of our thesis. We are conducting an experiment to compare **chatbot** and **mobile application** in order to collect campus-related data.

You have been assigned to test out the **chatbot** version of the app. Please follow the following instructions on your mobile phone to access the chatbot. It will approximately take 15 minutes to download and use the app for the first time.

**Please note that the data would be only used for experimental purposes and your personal data would not be disclosed anywhere.**

Getting **Started**

Figure D.3: Download Instructions (Part 1)

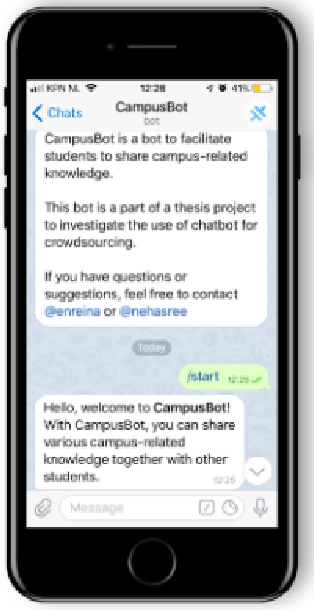


### 1 Download Telegram

Download Telegram (for iOS or Android) and register a new account.

You can skip this step if you already have Telegram on your phone.

**Download Telegram**



CampusBot is a bot to facilitate students to share campus-related knowledge.

This bot is a part of a thesis project to investigate the use of chatbot for crowdsourcing.

If you have questions or suggestions, feel free to contact @enreina or @nehasree

/start 12:25

Hello, welcome to **CampusBot**! With CampusBot, you can share various campus-related knowledge together with other students. 12:25

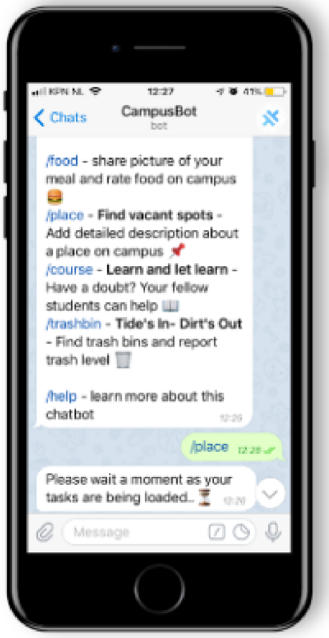
### 2 Open CampusBot

Open the chatbot by clicking the following button (or access directly at https://telegram.me/v1_campusbot?start=146) from your phone.

Then, tap on the "Start" button at the bottom. You will be greeted by the chatbot.

**Open CampusBot**

Figure D.4: Download Instructions (Part 2)



/food - share picture of your meal and rate food on campus 🍔
/place - **Find vacant spots** - Add detailed description about a place on campus 📌
/course - **Learn and let learn** - Have a doubt? Your fellow students can help 📖
/trashbin - **Tide's In- Dirt's Out** - Find trash bins and report trash level 🗑

/help - learn more about this chatbot 12:28

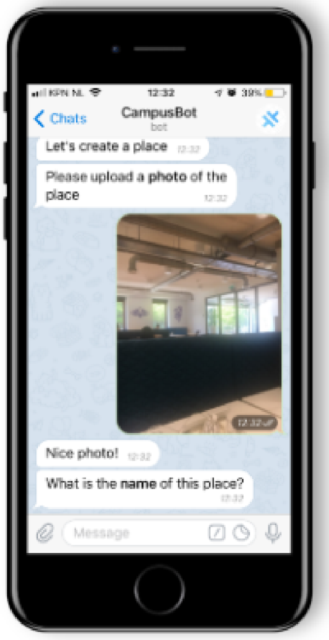
/place 12:28

Please wait a moment as your tasks are being loaded.. ⏳ 12:28

## 3 See list of tasks

Type in or select **"/place".**

**Wait a moment until the chatbot sends you a list of places.**



Let's create a place 12:32

Please upload a **photo** of the place 12:32

Nice photo! 12:32

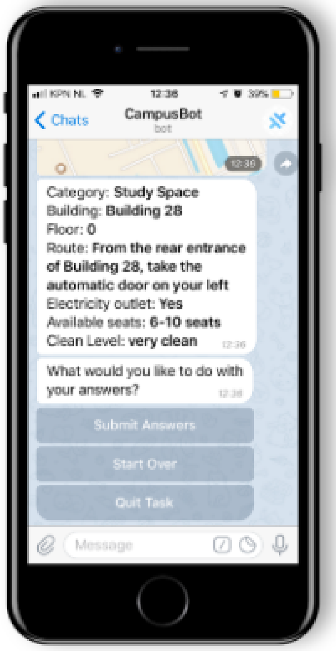
What is the **name** of this place? 12:32

## 4 Create a new place

Start creating a new place by typing **"/create"**.

Upload a photo of a place in campus and then answer several questions asked by the chatbot.

Figure D.5: Download Instructions (Part 3)



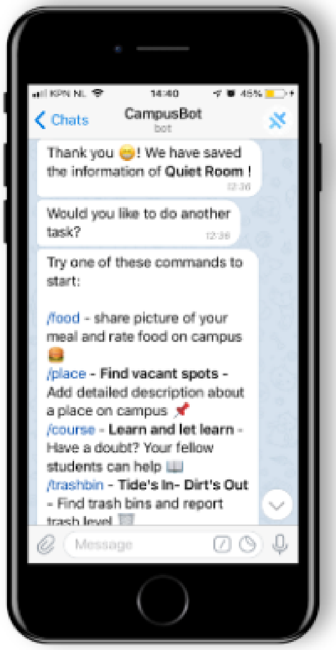
Category: **Study Space**
Building: **Building 28**
Floor: **0**
Route: **From the rear entrance of Building 28, take the automatic door on your left**
Electricity outlet: **Yes**
Available seats: **6-10 seats**
Clean Level: **very clean** 12:36

What would you like to do with your answers? 12:38

## 5 Submit your answers

When you have answered all the questions, submit your answers by tapping on "**Submit Answers**".

Congrats! You have helped us out by sharing information about a place in the campus.



Thank you 😁! We have saved the information of **Quiet Room** ! 12:36

Would you like to do another task? 12:36

Try one of these commands to start:

/food - share picture of your meal and rate food on campus 🍔
/place - **Find vacant spots** - Add detailed description about a place on campus 📌
/course - **Learn and let learn** - Have a doubt? Your fellow students can help 📖
/trashbin - **Tide's In- Dirt's Out** - Find trash bins and report trash level 🗑

## 6 Explore the chatbot

Feel free to explore the chatbot and do other things: sharing info about other places, post food photo, trash bin, and course questions.

We would really appreciate it if you could use the chatbot regularly for 2-3 weeks :D

Figure D.6: Download Instructions (Part 4)

# Appendix E

# Survey Questions (Study 1)

# CampusBot: Post-Experiment Survey

Thank you for participating in our CampusBot experiment.

We would like to ask a few questions regarding your experience in trying our app. It will take approximately 10 minutes to answer all the questions.

The data collected in this survey (excluding the information for identification) will be used only for research purpose.

* Required

1. **Email Address** *

   This will help us identify you in our participant list. Your email address won't be used for purposes outside the experiment.

   ______________________________

2. **Gender** *

   This data will only be used for statistical description purpose.
   *Mark only one oval.*

   - ◯ Female
   - ◯ Male
   - ◯ Prefer not to say

3. **What is your age?**

   This data will only be used for statistical description purpose.

   ______________________________

4. **What degree are you currently studying?**

   This data will only be used for statistical description purpose.
   *Mark only one oval.*

   - ◯ Bachelor
   - ◯ Master
   - ◯ PhD
   - ◯ Other / Not a Student

5. **Did you try to download the app?** *

   *Mark only one oval.*

   - ◯ Yes *Skip to question 19.*
   - ◯ No *Skip to question 20.*

## Please rate how much you agree with the following statements:

6. **I think that I would like to use CampusBot frequently.** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

7. **I found CampusBot unnecessarily complex.** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

8. **I thought CampusBot was easy to use.** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

9. **I think I would need the support of a technical person to be able to use CampusBot.** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

10. **I found the various features in CampusBot were well integrated.** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

11. **I thought there was too much inconsistency in CampusBot.** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

12. **I would imagine that most people would learn to use CampusBot very quickly.** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

13. **I found CampusBot very awkward to use.** *

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

14. **I felt very confident when I use CampusBot.** *

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

15. **I needed to learn a lot of things before I could get going with CampusBot.** *

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

16. **What were the challenges you found while interacting with the app?**

*Check all that apply.*

- ☐ Starting the app
- ☐ Choosing a task
- ☐ Creating an item (place, food, question or trash bin)
- ☐ Completing an Enrich Task
- ☐ Completing a Validate Task
- ☐ Other: ____________________

17. **What did you find easy to do while interacting with the app?**

*Check all that apply.*

- ☐ Starting the app
- ☐ Choosing a task
- ☐ Creating an item (place, food, question or trash bin)
- ☐ Completing an Enrich Task
- ☐ Completing a Validate Task
- ☐ Other: ____________________

18. **Any additional comments about the app? (e.g. why was it hard to use, or the app's purpose was unclear, how to make it better, etc)**

______________________________

______________________________

______________________________

______________________________

______________________________

*Stop filling out this form.*

19. **Did you use the app?** *

*Mark only one oval.*

- Yes *Skip to question 6.*
- No *Skip to question 22.*

20. **What was the reason you did not download the app?**

You can choose more than one reason. If the reason is not listed, you can also type in your reason by selecting "Other"

*Check all that apply.*

- I forgot to download the app
- The app did not work on my phone
- I was not interested
- I did not have Telegram on my phone and don't want to download it
- I did not have time to try out the app
- Other: ____________

21. **What would have convinced you to download the app?**

*Stop filling out this form.*

22. **What was the reason you did not use the app?** *

You can choose more than one reason. If the reason is not listed, you can also type in your reason by selecting "Other"

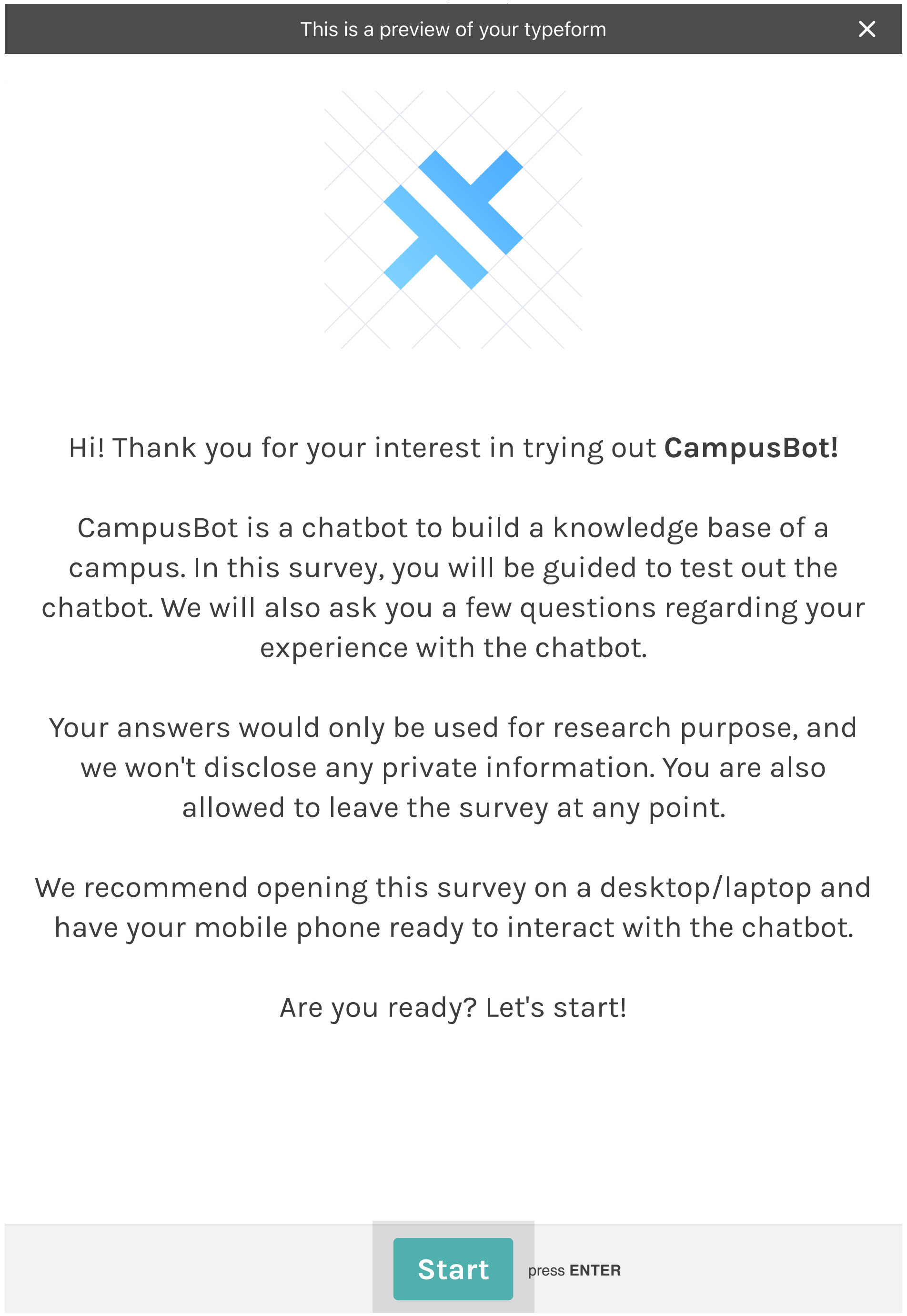
*Check all that apply.*

- I did not have time to use the app
- It was not interesting enough to use the app
- I forgot to use the app
- I would have used it if it was a chatbot instead
- The app was not useful for me
- Other: ____________

23. **What would have convinced you to use the app?**

# Appendix F

# Guideline and Survey Questions (Study 2)



This is a preview of your typeform

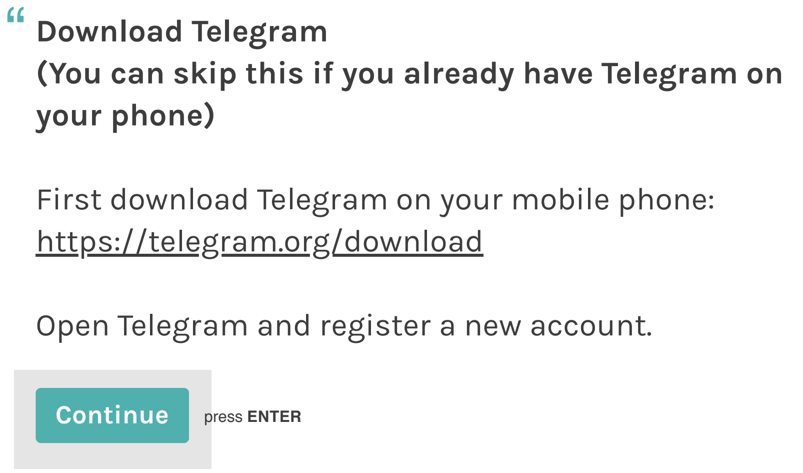
Hi! Thank you for your interest in trying out **CampusBot!**

CampusBot is a chatbot to build a knowledge base of a campus. In this survey, you will be guided to test out the chatbot. We will also ask you a few questions regarding your experience with the chatbot.

Your answers would only be used for research purpose, and we won't disclose any private information. You are also allowed to leave the survey at any point.

We recommend opening this survey on a desktop/laptop and have your mobile phone ready to interact with the chatbot.

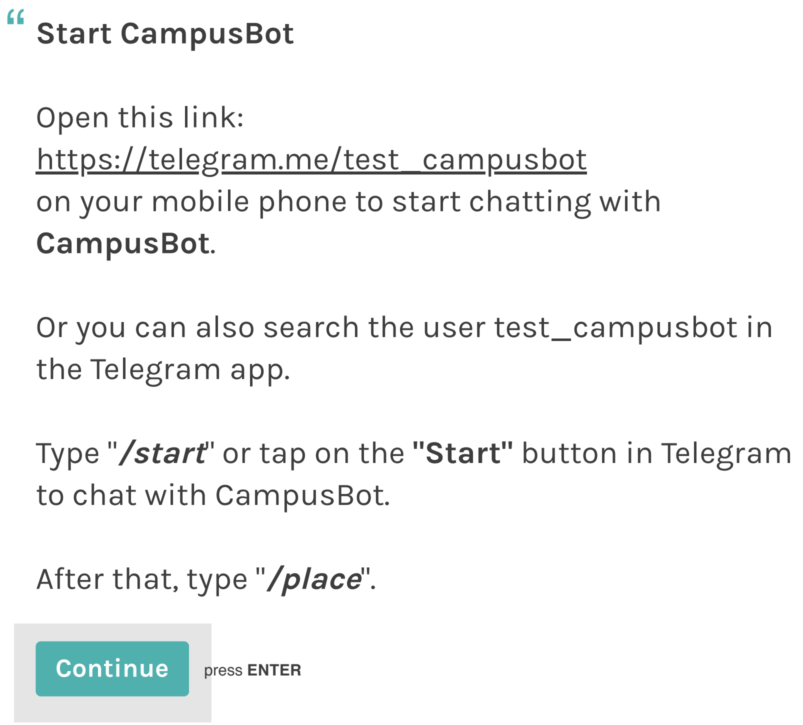
Are you ready? Let's start!



**Download Telegram**
**(You can skip this if you already have Telegram on your phone)**

First download Telegram on your mobile phone:
https://telegram.org/download

Open Telegram and register a new account.



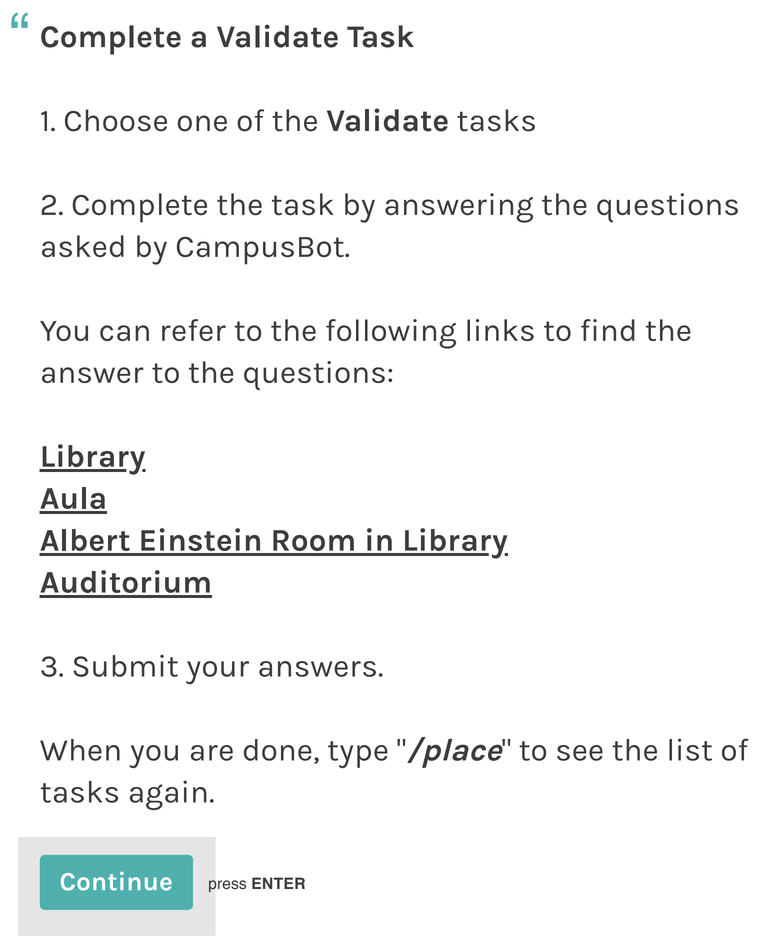
**Start CampusBot**

Open this link:
https://telegram.me/test_campusbot
on your mobile phone to start chatting with **CampusBot**.

Or you can also search the user test_campusbot in the Telegram app.

Type "***/start***" or tap on the **"Start"** button in Telegram to chat with CampusBot.

After that, type "***/place***".



“ **Complete a Validate Task**

1. Choose one of the **Validate** tasks

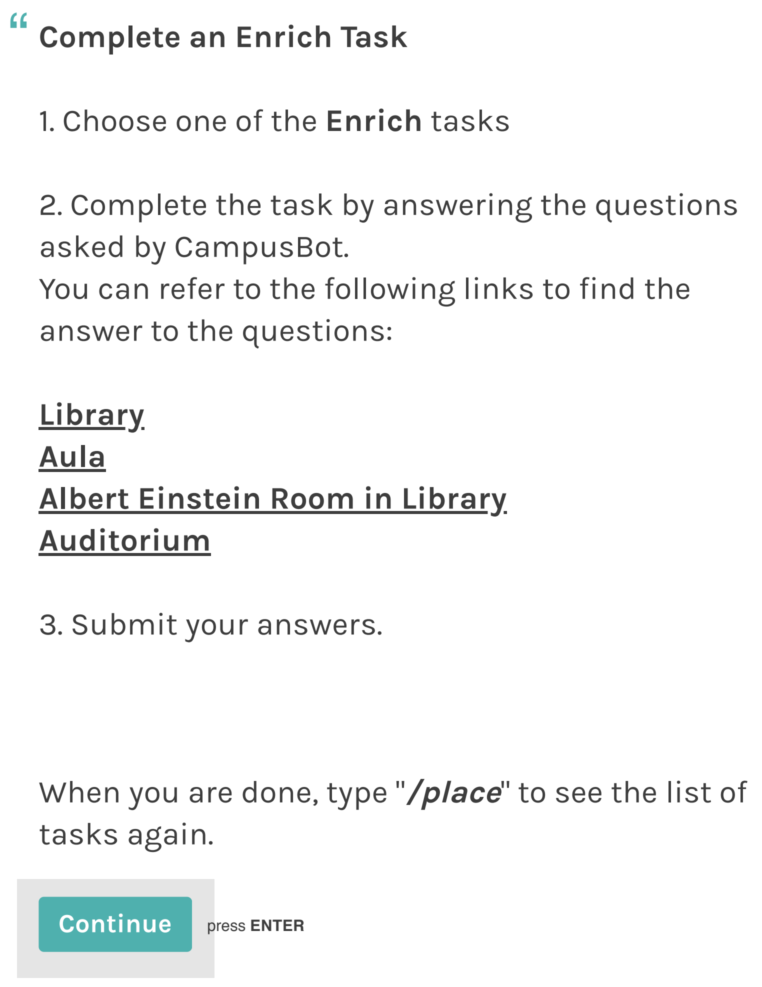
2. Complete the task by answering the questions asked by CampusBot.

You can refer to the following links to find the answer to the questions:

**Library**
**Aula**
**Albert Einstein Room in Library**
**Auditorium**

3. Submit your answers.

When you are done, type "***/place***" to see the list of tasks again.



“ **Complete an Enrich Task**

1. Choose one of the **Enrich** tasks

2. Complete the task by answering the questions asked by CampusBot.
You can refer to the following links to find the answer to the questions:

**Library**
**Aula**
**Albert Einstein Room in Library**
**Auditorium**

3. Submit your answers.

When you are done, type "***/place***" to see the list of tasks again.

“ **Create an Item**

1. Type "***/create***" to create an item

2. Complete the item creation by answering the questions asked by CampusBot.
You can create any place of your choice or refer to this link when answering the questions:
**Lecture Hall A**

3. Submit your answers.

**Continue** press **ENTER**

1 → Please rate how much you agree with the following statements:

**Continue** press **ENTER**

a. I think that I would like to use **CampusBot** frequently. *

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

Strongly Disagree Strongly Agree

b. I found **CampusBot** unnecessarily complex. *

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

Strongly Disagree Strongly Agree

c. I thought **CampusBot** was easy to use. *

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

Strongly Disagree Strongly Agree

d. I think I would need the support of a technical person to be able to use **CampusBot**. *

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

Strongly Disagree Strongly Agree

e. I found the various features in **CampusBot** were well integrated. *

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

Strongly Disagree Strongly Agree

f. I thought there was too much inconsistency in **CampusBot**. *

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

Strongly Disagree Strongly Agree

g. I would imagine that most people would learn to use **CampusBot** very quickly. *

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

Strongly Disagree Strongly Agree

h. I found **CampusBot** very awkward to use. *

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

Strongly Disagree Strongly Agree

i. I felt very confident when I use **CampusBot**. *

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

Strongly Disagree Strongly Agree

j. I needed to learn a lot of things before I could get going with **CampusBot**. *

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

Strongly Disagree Strongly Agree

2 → What are the challenges you found while interacting with **CampusBot**?

Choose as many as you like

- A Starting CampusBot
- B Finding a task
- C Choosing a task
- D Creating an item
- E Completing an Enrich Task
- F Completing a Validate Task
- G Other

3 → What did you found easy to do while interacting with **CampusBot**?

Choose as many as you like

- A Starting CampusBot
- B Finding a task
- C Choosing a task
- D Creating an item
- E Completing an Enrich Task
- F Completing a Validate Task
- G Other

4 → Any additional comments about **CampusBot**?

e.g. why was it hard to use, or the chatbot's purpose was unclear, how to make it better, etc

Type your answer here...

**SHIFT** + **ENTER** to make a line break