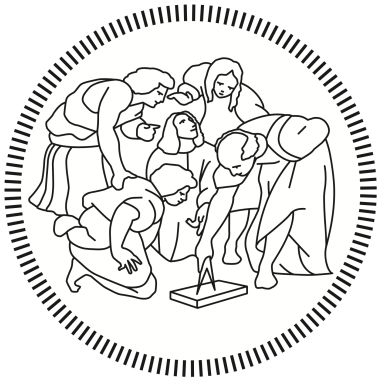
POLITECNICO DI MILANO
School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering
Master of Science in Computer Science and Engineering



Dialogue self-play and crowdsourcing to collect annotated data for a chatbot

Supervisor: Prof. Emanuele Della Valle
Co-Supervisor: Prof. Mikko Kurimo

Master Thesis by:
Luca Molteni, 913275

Academic Year 2020-2021

Abstract

The impact of chatbot technologies on our economies, services, and society as a whole is becoming more evident year by year and it's deeply influencing the future prospects of human-computer interaction. Conversational agents are not a new thing. Nonetheless, this area of research and development is still fertile ground for new innovation, especially from the field of Artificial Intelligence.

Historically, ad-hoc scripting languages were built to develop rule-based agents able to react to specific keywords or elements contained in the user input. Nowadays, deep learning based systems are becoming trending because they allow for more flexible and robust conversational models.

One of the curses of recent-years Artificial Intelligence, especially in the sub-field of deep learning, is the constant need for large datasets used to train models and algorithms. Gathering and annotating dialogues is a time and resource consuming process; knowledge is also not always transferable, especially when it deals with task-specific data as in the case of goal-oriented chatbots. On the other hand, pattern-based chatbots for goal-oriented tasks are relatively easy to design and implement but fail in achieving the natural feel of interaction demanded by today's users.

The goal of this work is to explore the applicability of Machines talking to Machines (M2M), a new framework to collect annotated datasets that can be used to train neural-based dialogue models. M2M makes use of a technique called dialogue self-play to build dialogue templates on top of computer-generated semantic annotations; in a successive phase, real users perform paraphrases of the templates to build up natural dialogues reflecting the underlying annotations. In this thesis, for the crowdsourcing phase, we tried to involve and compare the outcomes of classical workers from online platforms, and the real users of a chatbot called Siirtobot developed for the company 20Hexagons Oy.

The results show that it is possible to integrate this data collection approach within a rule-based chatbot. The datasets collected from professional

workers and actual users of a service show that the latter introduces a considerably higher dialogue diversity and linguistic richness, useful to train robust and flexible neural models. We also showcased how the collected data can be later used to bootstrap a neural agent by implementing and training a state-of-the-art module for Natural Language Generation.

Sommario

L'impatto dei chatbot sulle nostre economie, sui servizi e sulla società nel suo complesso sta diventando anno dopo anno più evidente influenzando profondamente i futuri prospetti dell'interazione tra uomo e macchina. I chatbots non rappresentano una novità recente, tuttavia questa area di ricerca continua ad essere terreno fertile per l'innovazione, specialmente nel campo dell'Intelligenza Artificiale.

Storicamente, linguaggi di scripting ad-hoc sono stati costruiti per sviluppare agenti rule-based capaci di reagire a specifiche parole chiave o elementi contenuti nell'input dell'utente. Ad oggi, prodotti basati sul deep learning sono molto in voga perché permettono di realizzare dei modelli più flessibili e robusti.

Uno dei maggiori problemi nel campo dell'intelligenza Artificiale degli ultimi anni, specialmente nella sotto-area del deep learning, è la costante necessità di trainare modelli e algoritmi con enormi moli di dati. Raccogliere ed annotare dei dialoghi per questo scopo è un processo lento e costoso; la conoscenza acquisita dai modelli inoltre non è facilmente trasferibile, soprattutto quando i dati fanno riferimento a scenari molto specifici come nel caso dei chatbot task-oriented. D'altra parte, i chatbot basati su regole pensati per risolvere specifici problemi sono relativamente facili da realizzare ma non permettono di ottenere quella sensazione di interazione naturale ormai richiesta dagli utenti.

Lo scopo di questa tesi è di esplorare l'applicabilità di Machines talking to Machines (M2M), un nuovo framework che permette di raccogliere dataset annotati per l'implementazione di chatbot basati su modelli di deep learning. M2M sfrutta una tecnica chiamata dialogue self-play per creare templates di dialoghi sulla base di annotazioni semantiche generate automaticamente; in una fase successiva, questi templates vengono parafrasarti da utenti in carne ed ossa in modo da ottenere dei dialoghi naturali che riflettono il contenuto delle annotazioni semantiche sottostanti. In questo progetto di tesi, per la fase finale di crowdsourcing, abbiamo provato a comparare i risultati ottenuti

con utenti assoldati su piattaforme online come Amazon Mechanical Turk, e i veri utenti di un chatbot chiamato Siirtobot sviluppato per la compagnia 20Hexagons Oy.

I risultati mostrano come sia possibile integrare il processo di raccolta dati sopra descritto all'interno di un chatbot rule-based. I dataset raccolti dagli utenti assoldati e i veri utilizzatori di un servizio rivelano che questi ultimi garantiscono una diversità dei dialoghi e una ricchezza linguistica considerevolmente più elevata, utile per realizzare modelli neurali robusti e flessibili. Abbiamo inoltre implementato e trainato un modulo di Natural Language Generation evidenziando come questi dati possano essere utilizzati in pratica.

Riconoscimenti

The motivation behind this thesis work stemmed from my interest in the world of conversational AI. I truly believe that chatbots and other dialogue systems will change the way in which we interact with our technology. I am especially grateful for the opportunity to delve deeper in this field by integrating into this project both my internship work and academic efforts.

I want to thank my supervisor at Polimi, Emanuele Della Valle, for his excellent and balanced guidance, and his great availability.

I also want to mention my supervising Professor at Aalto University, Mikko Kurimo and my advisors from the Aalto Speech Recognition Research group: Mittul Sing, Juho Leinonen, and Katri Leino; your comments and support have been inestimable.

I would also like to thank all the good friends that supported (and endured) me in various ways; is it with advice, criticism, or just with their comforting presence.

To conclude, I am grateful to my parents and my brother for always having my back in all circumstances. I hope to make you proud every day.

Milano, 24 settembre 2020
Luca Molteni

Contents

List of Acronyms

AI Artificial Intelligence

AIML Artificial Intelligence Markup Language

AMT Amazon Mechanical Turk

API Application Programming Interface

BLEU BiLingual Evaluation Understudy

CA Conversartional agents

CSV Comma Separated Value

DA Dialogue Act

DFA Deterministic Finite Automata

FSM Finite State Machine

HIT Human Intelligence Task

HTML HyperText Markup Language

M2M Machines talking to Machines

MLP Multi-Layer Perceptron

NGL Natural Language Generation

NLP Natural Language Processing

NLU Natural Language Understanding

RNN Recurrent Neural Network

SRT Service Registration Task

TF-IDF Term Frequency - Inverse Document Frequency

XML eXtensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

Chatbot industry has been expanding at an astonishing rate in the past 5 years and conversational agents are expected to gain more and more relevance in the context of human-computer interaction. Even if the first chatbots date as far as 1966, this research field is still going through a phase of deep evolution and mutation with many innovations coming especially form the field of Artificial Intelligence. Among the most popular applications of this technology, we may find customer support, booking systems, news outlets, and in general all those services covering a specific need.

Chatbots may be defined as that class of computer programs capable of carrying out a conversation with a human entity. Usually, the interaction takes place through a textual interface or in the form of spoken natural language. Developing chatbots, and in particular dealing with conversational design is not a trivial task. Chatbot systems can be divided into two broad categories. On one side, there are goal-oriented systems, such as customer support services, booking systems, and database querying applications. On the other hand, there are the so-called general-purpose, or *chit-chat*, models which rarely serves any practical uses but are used either for entertainment or as a way to showcase the current state of Artificial Intelligence.

Chatbots consist mainly of two components. A knowledge-base, which acts just like the brain and stores knowledge to reply to user input, and a dialog management module, used to control the dialogue flow, including receiving and understanding user input and searching for the right response from the knowledge base. The technology that characterizes the dialogue management module can be seen as an additional dichotomy splitting all chatbots systems into two separate classes: Rule-based and Model-based chatbots.

Rule-based chatbots, also called pattern-based chatbots, have been known since the appearance of the first advanced dialogue system, ELIZA, in 1966. Their knowledge base contains a set of templates that are matched against user input to select an appropriate response. Many could be negatively surprised seeing that such sophisticated systems, that should be virtually able to sustain a human conversation, do not leverage any of the hyped technologies of these days. Nonetheless, many chatbots winners of the Loebner prize for the most human-like agent are still based on these premise.

Model-based or AI-based chatbots are characterised by machine learning and reinforcement learning algorithms; most efforts of recent research are pointing in this direction. In particular, neural networks have been used to build more robust and flexible dialogue managers able to cope with the variety and unpredictability of human language. However, the biggest obstacle to the development of these systems is the ever-increasing amount of data required to train the models. The construction of a chatbot knowledge base is time-consuming and difficult to adapt to new domains especially when dealing with task-oriented systems.[1]

For goal-oriented chatbots, the rule-based approach represents a viable development solution, allowing to obtain an acceptable baseline with limited resources and time investment. In the long run, however, rule-based chatbots lack the flexibility, language variety, and natural feeling necessary to achieve a satisfying conversational experience.

As briefly mentioned, the biggest hurdle preventing the large scale diffusion of neural-based systems is the scarcity and costliness of task-specific datasets. Various approaches have been explored to mitigate this problem, ranging from the extraction of annotated knowledge from online forums [2], to social network scraping [3], and usage of general-purpose pre-trained models.[4] Typically, the annotated data required to train those systems share the following characteristics. A natural language utterance (e.g., "I want to order 3 pizzas") is associated with an intent, which entails the goal of the user ("Order food"), while a number of key-value slots carry the information required to complete the task ("Name", "Quantity", "Time", "Address")

Crowdsourcing is a technique that involves real people, generally recruited through online platforms, e.g Amazon mechanical Turk[1], to perform data

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

annotation tasks. The workers read and manually annotate given utterances to produce datasets compatible with the training of machine learning models. By itself, this technique, although not particularly expensive, mitigates only partially the issue of data annotation, since there is still the need for a dialogue corpora to annotate. Flaws in the quality of both these data and the resulting annotations may severely limit the performance of any neural-based model.[5]

Recent studies [6, 7] have tried to combine crowdsourcing techniques with dialogue self-play as a way to generate a complete annotated dataset from scratch. Instead of annotating pre-existing data, the workers are asked to contextually rewrite utterances from given templates built on top of automatically generated annotations. Dialogue-self play consists in generating a vast number of dialogue skeletons made of sequences of dialogue acts thanks to the interaction of a user and a system simulator. Dialogue acts are data structures able to encapsulate the semantic meaning of a given utterance or piece of text.

1.1 Methodology

This thesis work explores the possibility of integrating a new annotated data collection framework, called Machines talking to Machines (M2M), into a rule-based chatbot system, where a crowdsourcing phase is performed by real users. Said chatbot system has been developed during my internship for 20 Hexagons Oy, a company that provides the municipality of Helsinki with a service to notify residents about roadworks and car towings. The service, called SiirtoSoitto, has been integrated into a rule-based chatbot system and made available through the most popular instant messaging platforms. While in the original paper M2M has been tested in the context of database querying tasks, e.g. movie tickets booking, we tried to apply it on a set of different tasks with the integration in this rule-based chatbot.

A task-oriented rule-based chatbot is first developed to set an initial baseline while allocating a limited number of resources and time. Over time, the system is used to collect annotated data required to train and bootstrap a neural-based dialogue system and replace the original one. Two groups of users took part in the crowdsourcing stage. One was composed of real customers of the SiirtoSoitto service, the other was made of professional crowdsource workers from Amazon Mechanical Turk. The richness and variety of the collected dialogues and their respective wordings have

been evaluated on a number of quantitative metrics and compared between two datasets gathered with the aforementioned user groups. Following the same principle, the subjective qualities of the two resulting datasets have been also scored and compared by asking external participants to rate each dialogue between 1 to 5 on various linguistic dimensions. The hypothesis is that since the professional workers usually hired for this kind of task lack in-depth knowledge of the context they are dealing with, the resulting dialogues will be less diverse and more similar to the presented utterance templates.

1.2 Research questions

This entire work arises from a number of issues characterising current chatbot technology development, in particular related to annotated data collection. The research questions addressed within this thesis project are the following:

R1: Can the recently presented M2M approach be effectively applied to other tasks apart from database querying applications?

R2: What are the differences in the resulting dataset quality comparing two different crowdsourcing sources: generic professional workers from online platforms, and current users of the already existing system?

R3: Could M2M be integrated into a rule-based chatbot with the ultimate goal of crowdsourcing an annotated dataset to bootstrap a new neural-based dialogue system?

1.3 Thesis structure

The structure of this thesis is organised in the following manner:

- **Background**: acts as an introduction to the necessary background required to understand the functioning and the disputes behind chatbots and their main components (2.1, 2.2). A more specific section will be dedicated to data annotation and collection techniques (2.3).
- **Problem setting**: provides the context in which this thesis work has been developed, including an overview of the use case (3) and a description of the class of problems specifically tackled with the experiments (3.2).

- **Method**: in this chapter, Machines Talking to Machines, the data annotation and collection method that has been taken into consideration is explained in detail along with its technical implementation.
- **Results**: starts by describing how user testing has been carried out to gather two datasets (5.1). Later in the chapter are exhibited the results of the experimental phase carried out using the aforementioned data by mean of a quantitative and qualitative analysis (5.2, 5.4).
- **Proof-of-concept**: presents the implementation of an NGL module based on a modified architecture of a state-of-art model. The module is trained with the data collected during the experiments as a proof of concept of the applicability of the designed data annotation method (6.4).
- **Discussion and Conclusions**: here, the outcomes of the project are discussed and commented in relation to the research questions (7.1), posing particular consideration to the implications and limitations of the work. (7.2, 7.3)

Chapter 2

Background

This chapter is divided in two main parts. It serves as an introduction to the main concepts covered in the rest of the work and provides the context to understand the issues and choices that we are going through. The first portion provides a brief introduction to chatbots and the defining characteristics that allow to group them in distinguished classes. Next, follows an overview of the most common data annotation and collection methods for chatbot knowledge. There, we highlight how there is no definitive or most efficient solution to approach the task.

2.1 Chatbots

There is a lack of agreement on a standard meaning for what a chatbot is.[8] In particular it is unclear whether chatbots can indicate the same class of programs of dialogue systems and conversational agents. These are computer algorithms intended to converse with a human through a natural language interface. Some say the chatbots are generic computer programs that mimic human conversation. [9, 10, 11] Nonetheless, [12] highlights how people behave very differently between textual and spoken conversations, implying that chatbots and spoken dialogue systems represent two distinct classes of programs. The debate is not restricted to the use of a specific communication interface but concerns the purpose of the system as well. [13] claims that chatbots are meant to carry open domain conversations while task-specific agents should be called goal-oriented dialog systems.

For the purpose of this thesis, we will refer to chatbots as that class of computer programs able to carry on human-computer interaction by means of a text-based natural language interface, and we will interchangeably refer to them as conversational agents or dialog systems.

2.1.1 Historical notes

The history of conversational agents (CA) traces back to 1966 when Joseph Weizenbaum programmed ELIZA, a chatbot that answered to textual input by mimicking the behavior of a Rogerian psychologist.[14] Most of the time, the program replied simply by reformulating the user's sentences, creating the impression that it was interested to know more about the current topic (Fig. 2.1). ELIZA mainly worked by analyzing and substituting keywords with a number of predetermined transform rules. Sometimes, the result was so effective that people were convinced to talk with a human interlocutor for several minutes. ELIZA opened the doors to the development of increasingly intelligent and complex chatbots, such as PARRY, in 1972, impersonating a paranoid schizophrenic.[15]

During 1977, in the context of the DARPA Project in the US [16], dialogue systems became also able to communicate with the user through a spoken dialogue interface. The first commercial applications aimed at providing complementary services through phones in the telecommunication industry, e.g., train tables service. Most of the subsequent research has focused on developing languages to implement dialogue systems for the industry, such as customer support applications.

```
YOU: I don't like my family so much
ELIZA: That is quite interesting
YOU: My family members aren't friendly
ELIZA: Why do you think that?
YOU: I don't know who you are
ELIZA: Why do you ask?
YOU: I'm sorry
ELIZA: That's okay. Please continue
```

Figure 2.1: An example of conversation between ELIZA and a user

In 1995, Richard Wallace released A.L.I.C.E [17], a general purpose chatbot, on the same style of ELIZA, but capable of achieving even more impressive results. One of the biggest improvements compared to its predecessors was the ability to reply with linguistic deflections. These are phrases used to avoid breaking the dialogue when the response for a certain sentence is unknown, e.g, "Why everything needs to have an explanation?", "Do I really

need to answer this question?". Combining this ability with a hugely expanded variety of pre-determined sentences, A.L.I.C.E was able to keep a conversation running even if a user tried to ask a few times the same question.

In 2008, Apple introduced Siri[1], a smart assistant for iPhone devices, able to combine the interactivity of general-purpose dialog systems with the ability to accomplish a wide variety of tasks, such as searching information on the web, scheduling reminders and initiating calls. From this point in time, the capability of CA to act on the surrounding environment became an essential part of Human-Computer Interaction. Siri has been quickly followed by its counterparts from major competitors, such as Samsung's Bixby[2] and Microsoft's Cortana[3].
The diffusion of chatbot systems for customer support, online banking, marketing and enterprise branding, just to mention a few, was largely supported by the biggest platforms for online instant messaging. Facebook launched in 2016 its bot platform called "bots on Messenger"[18], following the trend set by other companies such as Telegram[4] and Line[5]. The biggest online messaging service, Whatsapp, in 2017, also added support for chatbot based solutions for enterprise customer service[6].

In recent years, smart assistants have moved from our phones to our houses. In 2014, Amazon launched Alexa[7], a home assistant capable to interact not only with resources on the internet but also with several smart devices around the house. As the precision and capabilities of chatbot systems grow, the trend is to entrust them with higher and higher responsibilities such as managing the usage of home appliances or performing payments.

2.1.2 General-purpose chatbots vs Goal-oriented chatbots

General-purpose chatbots comprise those systems which are not aimed at satisfying a user goal but simply carrying out a generic conversation. These agents act within an open domain, meaning without having any specific expert knowledge, and are therefore generally considered more challenging to build.[19] Nonetheless, open-ended models of this kind were the first to be

[1]Apple Machine Learning Journal, URL: https://machinelearning.apple.com/
[2]What is Bixby, URL: https://www.samsung.com/global/galaxy/what-is/bixby/
[3]Cortana, Your personal assistant, URL: https://www.microsoft.com/en-us/cortana
[4]Telegram Bot Platform, URL: https://telegram.org/blog/bot-revolution
[5]Bot Designer, URL: https://developers.line.biz/en/services/bot-designer/
[6]WhatsApp Business API, URL: https://developers.facebook.com/docs/whatsapp
[7]What is Alexa?, URL: https://developer.amazon.com/en-US/alexa

realized back in the 1960s.

The rise of interest in the topic of replicating human behavior was largely caused by the publication of the notorious paper "Computing machines and intelligence" by Alan Turing.[20] Turing posed the question "Can machines think?", where thinking means having the same causal properties that characterize humans brains. He also proposed a mechanism to answer the question in the form of an experiment called the "imitation game". During the imitation game three people, a man, a woman and an interrogator are sitting in separate rooms. The goal of the interrogation is to understand which of the two is a man and who is a woman by directing questions to them. The goal of the man is to fake being a woman in order to mislead the interrogator. Turing suggests that substituting the man with a computer and playing the same game would answer the question "Can machines think?", or in a different way "Can a machine win the imitation game?"

There has been and there is still going a never-ending debate on whether Turing's proposal is acceptable or even if the initial question is well-posed.[21] However, this question largely inspired the first researchers that applied themselves in building programs able to mimic human conversations, which was the requirement to pass the Turing test. All the first notable chatbots were general-purpose, starting from ELIZA, A.L.I.C.E, PARRY and going on. Over time general-purpose systems have become a way to showcase the current advances in the field of AI and it's unlikely that current research is still driven by the original Turing's question: "Can machines think?". While having little practical applicability, general-purpose chatbots continue to play such an important role in current research because they comprise many of the technologies that lie at the core of Artificial Intelligence and find relevant applications in, e.g., machine translation and text summarization.

In 1990 Hugh Loebner and The Cambridge Centre for Behavioural Studies established a competition based on implementing the Turing test. A Grand Prize of $100.000 was offered to the first program which cannot be distinguished from humans. Up to this day no chatbot has ever achieved the golden medal and passed the test to win the Loebner Prize.

Writing a general-purpose chatbot is very difficult because it needs a very large knowledge base and must give reasonable answers to all interactions. The first models, such as ELIZA, made use of a handwritten corpus of rules combined with linguistic tricks, while recent approaches apply supervised learning techniques to learn what sequence of tokens, or words, are better suited as a reply to the input sequence.[22] This method, in order to gen-

erate reasonable answers, requires to learn over a huge corpus of data, such as the Wikipedia dump. Recent developments include information retrieval and active learning, which means saving new interactions from the users and then using them later to give answers for similar sentences. While the first chatbots could only rely on the information contained in their knowledge bases, nowadays online chatbots can access information from the web. In 2011, IBM Watson was able to win several games of Jeopardy, a classic American quiz show where participants are asked to formulate the correct question given the corresponding answer. Watson leveraged information retrieval over a huge database of over 200 million pages of contents including a full Wikipedia dump.[23] Amazon Alexa and Google Assistant also make use of this kind of approach to reply to generic interactions while combining it with a number of goal-oriented sub-systems sometimes called "skills"[8].

Goal-oriented chatbots support users in achieving a predefined goal within a closed domain, e.g. booking a flight for a certain destination. Natural language understanding techniques including supervised learning algorithms trained with words ensembles or simpler keyword matching mechanisms are used to understand the user's goal. After the goal is known, the chatbot must manage the dialogue in order to collect the information required to achieve the goal. These chatbots can be divided into different categories, e.g., health, finance, news, food. Various banks, airlines, retailers, government bodies and restaurant chains use chatbots as a mean to offer simple services in a timely manner increasing customer engagement and satisfaction

In 2017, a study conducted among users showed that the most frequently reported motivational factor for the use of chatbots is "productivity".[24] Chatbots help users to obtain timely and efficient assistance and information. While general-purpose chatbots are considered more difficult to implement given their unconstrained working context, goal-oriented chatbots are required to resolve users' needs fast and in a satisfactory way in order to boost productivity. Developers are also faced with the complex task of considering how the users will possibly interact with the system and designing conversations able to take account of all the possible nuances of human spoken language. Some of these features are memory and persistence, context switching and the ability to retrieve the information required to complete the task.

[8]URL: https://developer.amazon.com/en-US/alexa/alexa-skills-kit

Goal-oriented systems are hardly scalable because usually, the knowledge required to solve a specific task is not reusable outside its scope. Some attempts have been conducted to apply transfer learning between two different task-oriented chatbots but relevant results have been obtained only when the source and the target domain overlapped significantly.[25]

2.1.3 Rule-based vs Model-based chatbots

In this section, we are going to discuss about the different ways in which a chatbot performs actions when the user writes something. One way to react to user's input is to first determine the meaning of the sentence. Semantic structures called dialogue act are often used to identify the function and the contents of a user utterance extracted at each dialogue turn. There are a number of standardized set of dialogue acts for generic conversations, such as the Cambridge dialogue act schema.[26] Domain-specific dialogue acts are called intents.

Different utterances may convey the same intent. The sentence: *"I want to book a restaurant at 9"* and *"Tonight I want to go to the restaurant"* should trigger the same command in a chatbot. Apart from the intent, which represents the goal of the user at a given dialogue turn, the dialogue manager usually uses two other elements. Entities, which are the relevant information contained inside a sentence, and contexts, which may be used to represent the general state of the conversation from a wider perspective. For example, if we imagine a restaurant booking chatbot, in the sentence *"I want to eat sushi at nine"*, the entities would be (cousine="sushi", time=9:00). Contexts are useful to know how to follow up to an interaction. When given a "Yes" or "No" answer the chatbot should perform different actions depending on the context of the dialogue. If the aforementioned booking chatbot asks *"Do you want to book a table for 4 people?"* or *"Do you like sushi?"*, a negative answer should trigger different actions. As previously stated in the introduction there are two classes of approaches: rule-based and model-based.

In a rule-based approach, the user input is matched against some pre-defined patterns and an action is selected accordingly. For each intent, a unique pattern must be available in the database to provide a suitable response. Entities are also extracted either through explicit elicitation, using some handcrafted rules or regular expressions. The context of the dialogue is usually accounted for by considering the few previous user utterances. The defined rules can be very simple to very complex and give rise to complex

hierarchical structures. The creation of chatbots is relatively straightforward using some rule-based approach, but the bot is not capable of reacting correctly to utterances whose pattern does not match any of the rules in the chatbot's knowledge base.

Many different scripting languages have been conceived with the purpose of specifying rules for this kind of chatbots. One of the most famous, based on the XML specification, is AIML (Artificial Intelligence Markup Language). AIML was initially developed by Richard Wallace to lay the foundations for his A.L.I.C.E chatbot and has since become the de-facto standard to develop rule-based chatbots. The most important elements of the AIML language are categories, patterns, and templates.

The goal of categories is to form the fundamental unit of knowledge of the conversation. Categories enclose patterns and templates.

A pattern is a string intended to match one or more user inputs also thanks to the addition of wildcards and special characters such as *.

Templates contain the replies to a matched pattern. They may make use of variables, conditional responses (if-then/else), and random replies.

Due to their nature, carrying out conversations with rule-based chatbots may sound artificial and constrained. For these reasons, developers have to make good use of alternative strategies to maintain a fluid conversation. Chatscripting focuses on the best syntax to build a sensible default answer and gives a set of functionalities such as variable concepts, facts, and logical and/or. Language tricks are sentences, phrases, or even paragraphs available to make the conversation still convincing when no rules match the user input. They include proposing a new topic or action, asking open questions, confirming information and telling jokes.[10]

Rule-based conversational agents parse user message, find synonyms, tag parts of speech and find out which rule matches the user input. However, they do not run machine learning algorithms.

Model-based chatbots extract user's intents using machine learning classifiers or use neural network models to generate replies. To provide an example, in a Bayesian approach the extracted intent is the one that maximizes the conditioned probability of having that intent I given an utterance U.

$$I_{max} = argmax_I P(I|U)$$

using Bayes rule:

$$I_{max} = argmax_I \frac{P(I)P(U|I)}{P(U)}$$

Since we maximize with respect to I we can get rid of the denominator

$$I_{max} = argmax_I P(I)P(U|I)$$

Now, in order to put this formula into use we add the so called Naive Bayes assumption, meaning that we consider all the subsequent words W_i in the utterance as independent. In general this assumption does not hold, since would mean that the probability of each word is independent from the previous which openly contrasts with, e.g, the word distributions induced by grammar rules. Nonetheless, even with this assumption, we can reach good results.

$$I_{max} = argmax_I P(I) * \prod_{i=1..N} P(W_i|I)$$

$P(I)$ and $P(W_i|I)$ can be computed from the training data. This is called the unigram model because each word is treated independently and has been shown to reach an accuracy close to 75%. [27]

Other approaches to intent classification include the use of memory-based learning algorithms such as K-nearest-neighbor. Training utterances are stored into memory with their respective intents; when a new sentence needs to be labeled the algorithm retrieves the N closest sentences and returns a classification based on majority voting. Measuring similarity between sentences is another discussed topics. Some simple methods include counting co-occurring words or overlapping groups of N-words (BLEU).

A more advanced approach consists in obtaining word representation in a lower dimensional space and using these features as the input for a neural network or any other classifier, such as a random forest. The outputs of the model will correspond to the probabilities of each possible output intent and can be computed with a Softmax function. In order for this method to work, it is extremely important that the representations of the words in the new vector space reflect as closely as possible the semantic relations between words. Word2vec skip-gram [28] is a widely used algorithm to obtain pre-trained vector representations for input words.

The advent of deep learning has transformed the field of conversational modelling as it did for many other applications. The latest trend is to build end-to-end models able to take simultaneously care of NLP, NLU and response generation. More specifically, the base architecture that is dominating the field is the encoder-decoder recurrent neural network (RNN) model,

called Seq2Seq [29], introduced by [30]. More details about this model will be presented in section 2.2.3. One of the issues with end-to-end neural-based models is that by nature they lack an explicit representation of intents and extracted entities. While it does not represent a relevant problem for chit chat models, it poses some limits especially in the implementation of goal-oriented systems. Several tweaks have been proposed. In [31] an entity indexing step is introduced before feeding the source utterance into a seq2seq model. In this phase all the pre-processing steps that are generally needed for entity extraction are carried out, such as keyword matching and named entity recognition. Using a technique called delexicalization, whose concept is shown in figure 2.2, the identified words are replaced with generic tokens, while the connection between the specific term and the token is stored in a dictionary. The Seq2Seq model produces responses that may include special tokens indicating API calls and generic tokens referencing stored values.

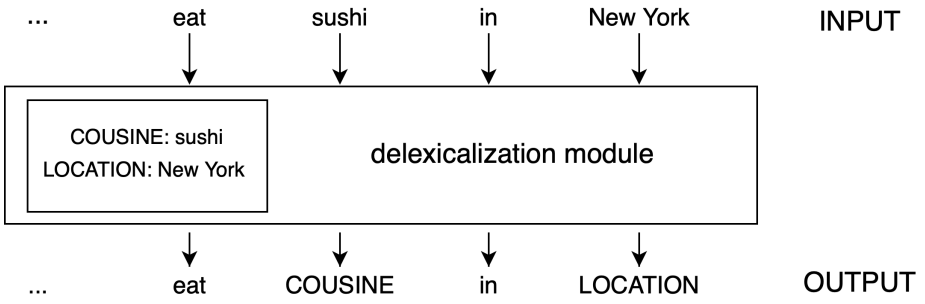


Figure 2.2: Delexicalization is used to remove specific entity values and make the dataset more homogeneous

2.2 Dialogue-state architecture for goal-oriented chatbots

In this section, we will focus on that subclass of goal-oriented systems based on the concept of dialogue state. A dialogue state can be imagined as a frame that needs to be filled with three elements, context, intent and entities, and allows for a complete representation of various dialogue aspects. We target this specific chatbot architecture mainly for two reasons. It has been astonishingly long-lived and underlies most, if not all, modern systems.[32] Secondly, Siirtobot, the rule-based chatbot whose implementation will be described in 4.2 is unsurprisingly part of the same category. Since the development of said chatbot has been part of an internship job for 20Hexagons Oy, and it is going to serve on the side of a well tested and utilised system,

it was a technically wise decision to choose a well-known architecture. In the previous section, we introduced a general notion of chatbot and highlighted which main features allow to separate different classes of dialogue systems. We also already introduced and started discussing some concepts of relevance for this new section, namely the notion of intent and entity extraction. In the following subsections, we will present in more detail the information pipeline and the main components that typically constitute the architecture of this kind of goal-oriented dialogue systems.

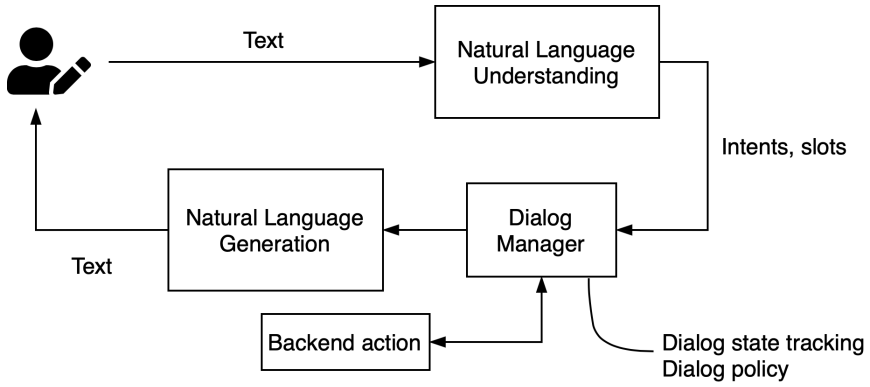


Figure 2.3: Information pipeline in dialogue-state goal-oriented chatbots

As shown in figure 2.3, the user textual input is first fed to the NLU model which takes care of extracting all the relevant information. Subsequently, intents and entities are used by the dialog manager, in combination with the current dialog state and the defined policy, to select a response action. During this phase, additional operations, such as database or API queries, are performed. At this point, the selected response action is passed to the NGL module which will translate it in a natural language utterance. Finally, the generated textual output is sent back to the user.

2.2.1 Natural Language Understanding for intents and slot filling

The natural language understanding component in a dialogue state-based architecture serves the role of using user's utterances to determine intents and perform slot filling. In this case, we are only considering single-domain systems, specialised on single tasks. Otherwise, a third feature, namely domain classification, would be required. The most common slot-filling method, which is still popular in industrial applications is to use handwritten rules. Such rules can be expressed in terms of hand-designed semantic grammars.

The left side of each rule corresponds to a slot name whose possible values are expressed in the rightmost part. Here follows an example:

```
phone_number <= ^[+]*[(]{0,1}[0-9]{1,4}[)]{0,1}[-\s\./0-9]*
cousine <= {Chinese, Japanese, Italian, French}
```

A different approach makes use of a stricter conversational design in which the system takes the initiative and explicitly asks the user for a piece of information that is later validated. Such paradigm is typically used only when, given the amount of information that needs to be retrieved from the user, the dialogue would end up in a multi-turn conversation. Nonetheless, this kind of approach could be perceived as frustrating and lead to the user being stuck in a bad dialogue system that asks a question and gives no opportunity to do anything until a valid answer is provided.

Finally, a modern approach to the slot filling and intent detection task employs supervised machine learning methods instead of rules. In this case, a dataset where utterances are paired with the respective intents and set of slots is used to train a model. Intent detection can be easily carried out using a 1-of-N classifier where each utterance is assigned to one of the N intents. A number of algorithms are suitable for the task such as Multi Layer Perceptrons (MLP) and decision trees. Slot filling represents a more challenging task and rule-based approaches such as dictionaries, regular expressions and grammars present strong limitations, especially at the presence of a non-finite or non well-defined set of entities. Among the statistical approaches to this problem, we may find hidden vector state models, support vector machines, conditional random fields and deep learning algorithms. IOB is a well-grounded practice that consists in tagging each word in a given sentence either with a begin (B-slot), inside (I-slot), or (O) outside label. Entity extraction is reduced to a classification problem. First, a new sentence representation is computed, for example by passing it through a contextual embedding network like BERT.[4]
In such a way we obtain a lower dimensional representation of each word called word embedding. An interesting characteristic of word embeddings is that semantically similar words, meaning they occur in similar contexts, tend to lay close in the new vector space. Each of these new representations can be then passed through a MLP with one or two feed-forward layers followed by a softmax function. The output of this architecture is one IOB tag for each token in the input sequence. (figure 2.4) The terms or expressions

enclosed by different types of B,I, and O tags indicate the presence of a specific entity.

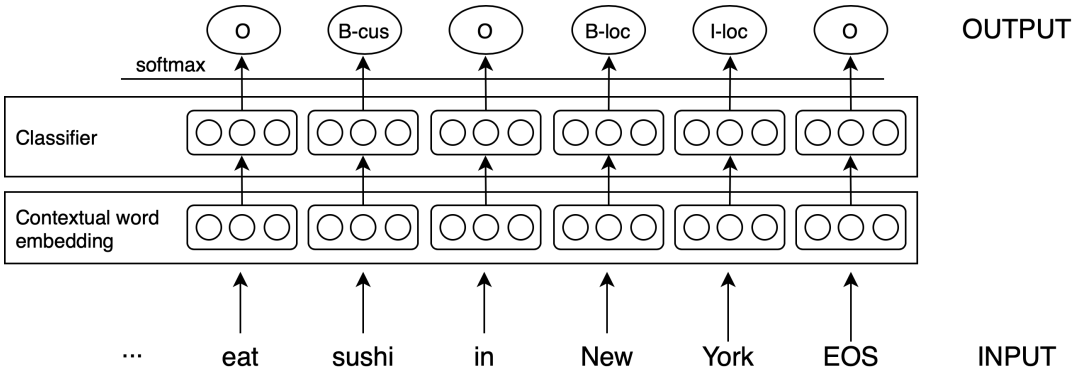


Figure 2.4: Example of IOB tagging architecture, extracted entities are cousine (cus) and location (loc)

2.2.2 Dialogue manager

After the NLU module generates a semantic representation of the user input, further processing is carried out by the Dialog Management component. It performs typically two tasks: dialogue state tracking and dialogue policy management. Tracking dialogue states means to estimate the user goal at every turn of the dialogue, combining both the input from the NLU unit and the dialogue history. On the other hand, policy management deals with the task of choosing the next action based on the current dialogue state. Traditional methods for dialogue state tracking consist of simple rule-based components keeping track of the slot filling process after each utterance. Each possible intent is associated with a semantic frame, which is an object containing all the entity slots required to complete the task. Given a semantic frame and it's respective slot-value pairs, after each utterance, the representation of the frame is updated with the new extracted slot values. The previous or few previous utterances are also stored to give contextual memory. Alternative techniques use statistical models maintaining a distribution over several possible states of the dialogue, such as in [33]. These are the methods generally implemented in commercial products.

The goal of dialogue policy is to decide which action to carry out next given the current dialogue state. At each turn n of the conversation we want to decide which action A_n the system will take based on the current dialogue

state representation S_{n-1}.

$$\hat{A_n} = argmax_{A_n \in A} P(A_n | S_{n-1})$$

If, as mentioned previously, we represent the current state just as the semantic frame containing all the slot-value pairs elicited from the user, and the previous few system A_j, and user actions U_j we can rewrite the conditioned probability as:

$$\hat{A_n} = argmax_{A_n \in A} P(A_n | \text{Frame}_{n-1}, A_{n-1}, U_{n-1}, ...)$$

Explicit policies make use of handcrafted rules to select the most likely response [34], for example matching the previous utterances against some handwritten templates and keeping an explicit representation of the dialogue state in a database. More recently new research has explored the application of deep learning, reinforcement learning, and end-to-end models also for policy learning. [35] [36] These new approaches have also been driven by the necessity to deal with structural issues of traditional techniques such as the difficulty to adapt to new domains, frequent errors, and sensitivity to noise or ambiguity.

2.2.3 Natural Language Generation

The last step in the information processing pipeline of chatbots is Natural Language Generation (NGL). The agent needs to put the chosen action into words by structuring all the information provided by the dialogue manager into a grammatically and semantically correct sentence. NGL needs to generate a specific textual representation from all the possible representations. This step can be split into several sub-task such as content determination and sentence realization.[37]

Content determination decides what information include in the output and it's a task generally carried out by the dialogue manager during action selection.

Sentence planning deals with establishing how to structure the selected contents into a natural language sentence. As with most NLP related tasks, NGL as well presents two approaches, leveraging respectively deterministic rules and machine learning.

If we look back in history, one of the characteristics that determined the success of the most famous chatbot, ELIZA, was the ability to generate very convincing and sophisticated responses. ELIZA sentence generation follows a number of pattern/transform rules associated with keywords that

may occur in user sentences. Each transform rule is associated with a priority level that allows to first execute rules that match with more specific keywords.

```
Input: I feel scared

Pattern/transform rules:
(I *) -> (Why you say *) (0)
(* feel *) -> (What do you think when you feel * ?) (5)

Output: What do you think when you feel scared?
```

Figure 2.5: Example of ELIZA NGL rules

As shown in the example of fig. 2.5, the input sentence matches both the pattern/transform rules but only the one with higher priority (5) is applied to generate the following output sentence: "What do you think when you feel scared?". ELIZA also had a clever memory mechanism that stored the content of sentences containing the keyword *my*. Whenever this keyword has the highest matching priority some content from the memory is extracted to simulate the presence of interest and attention in the chatbot. In case no rules are matched, a number of predefined output sentences are used to keep the conversation flowing.

Similar techniques are commonly used in goal-oriented chatbots with industrial applications by adding support to API and database queries to introduce customised variables. This template-based generation is suitable for restricted domains with specific tasks and provides high precision. Nevertheless, high competencies are required by the dialogue designer; moreover, the system's responses tend to lack variety and do not contextually adapt to reflect the input wording.

Machine learning methods, and in particular deep learning algorithms, are generally used to build generative model to create responses as a direct function of specific input words, and therefore have the potential to generate more natural outputs. This area of research owes a lot of accomplishments to the field of statistical machine translation where the same algorithms are used with a different purpose. A widely spread solution is based on the Seq2Seq model already mentioned in 2.1.3 in the context of end-to-end models. It is an encoder-decoder model that uses two recurrent neural

networks (RNN). The first takes as an input a variable-length sequence of words and encodes it in what is sometimes called toughs vector. The second RNN uses the encoded representation of the input to generate a response (see fig. 2.6). RNN are neural networks with a hidden state h and an output y which is a function of the input sequence $X_1...X_n$. RNNs can use the hidden state as a memory to process variable-length sequences of inputs. At each time step, the hidden state of the encoder RNN h is updated by the function:

$$h_t = f(g_{t-1}, x_t)$$

After the whole input sequence has been read the final hidden state contains an encoded representation of the input state that we call c. The second RNN is an auto-regressive model that updates its hidden state as a function of the the encoded input c, its previous state h_t, and the previous output y_{t-1}

$$h_t = f(h_{t-1}, c, y_{t-1})$$

At each timestep the current hidden state is used, e.g. in a linear regression, to generate a probability distribution over all the possible output tokens using a Softmax function.

$$y_t = \text{Softmax}(Wh_t + b)$$

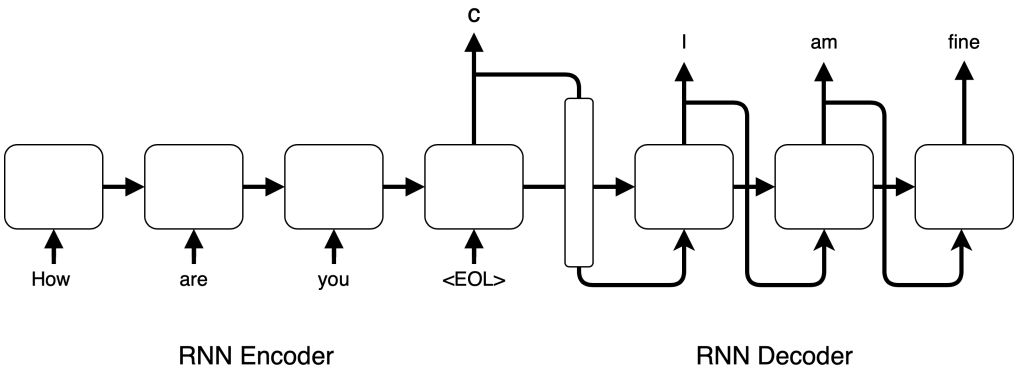


Figure 2.6: Seq2Seq RNN model architecture

The main issue with this kind of approach is the lack of an explicit representation of the extracted information or the ability to insert database queries. This is due to the fact that the output sentence is computed directly as a function of the input utterance. Various workarounds have been developed to use generative models also with goal-oriented chatbots. In [38]

and [39], instead of the user utterance, the input for the encoder-decoder model is the representation of the system action selected by the dialogue policy. By training on many examples of representation/sentence pairs from a large corpus of labeled dialogues, the Seq2Seq model is able to produce output sentences. One technique to increase the generalization power of such algorithms is delexicalization, already mentioned in 2.1.3, which consists in replacing specific words in both training set and input sentences with generic placeholders. The output sentence will also contain this kind of placeholders that can be substituted with the right values in a post-processing phase.

2.3 Data collection and annotation methods

It is not trivial to make use of supervised learning approaches in chatbots. They are not easy to develop as they require a huge set of annotated human-chatbot data that are not usually available. Gathering the amount of labeled data required to conduct supervised machine learning is a complex and costly task. As we have seen in the previous sections, chatbots may use supervised learning algorithms to accomplish many different tasks during the information processing pipeline. Annotated data are required for entity extraction, response generation, dialogue act tracking, etc.

Over the years, researchers have sought new methods to ameliorate the problem of data collection and annotation. In the next few sections, we will go through the most common techniques as well as presenting some experimental approaches. Nevertheless, the automatic collection of annotated data is still an open problem and the most popular technique is by far crowdsourcing, which can be seen as a distributed outsourced version of manual data annotation.

2.3.1 Manual data collection and annotation

While talking about manual data annotation we refer to annotation jobs carried out by domain experts, e.g. the engineers or the developers who are implementing a chatbot system. Manual annotation is sometimes required to build what is generally addressed as the gold standard, or ground truth, which is a dataset that can be used to bootstrap machine learning algorithms and overcome the "cold-start" situation. There are undoubtedly some advantages in manual data annotation. Firstly, domain experts understand the data better than other professionals, increasing the quality of the data, especially in highly specific contexts where broad knowledge about the task is required. Domain experts might also uncover interesting insights about

data that can be later incorporated into the algorithms resulting in better performances. There are several tools aimed at improving the easiness and speed of manual data annotation. ANVIL [40] is a generic annotation tool for multi-modal dialogue supporting different annotation schemes to map auditory (words, dialogue acts, rhetorical structure) and visual (gesture, posture) inputs. BRAT (fig. 2.7) is an intuitive and user-friendly web-based annotation tool for NLP which integrates machine learning technology to speed up the information processing. For example, in a NER task, type selection has been shown to be over 30% faster thanks to automated suggestion.[41]

Even with the support of semi-automated tools, manual-expert annotation presents numerous disadvantages. First, it is not scalable to large collections. Nowadays, deep learning algorithms need to be trained over thousands or even millions of samples. Moreover, manual annotation conducted by experts is expensive, in particular if the researchers themselves are carrying out the process. Lower data quality and slow processing can be determined by an overwhelmed small annotation team. Finally, especially in the context of paraphrasing or response generation, having a limited number of people engaged in the task may result in a reduced variety of utterances, which is a key for ensuring robust performance of any NLU model.[42]

It is relevant to highlight how, until this point, we only discussed about how to annotate an already existing dataset, but often the availability of such raw data can not be given for granted, especially for narrow and specific domains.

Figure 2.7: BRAT online annotation tool GUI

2.3.2 Data-driven data collection and annotation

Some techniques allow the collection of an already annotated dataset, where, with annotated, we mean having both the content itself (a dialogue or an utterance) and its semantic representation (dialogue acts, query-answer tuples, templates). Several pieces of research have attempted to show how to extract chatbot knowledge from online discussion forums. Discussion forums are digital platforms where people discuss, exchange ideas, and share information in certain domains, such as videogames, movies, lifestyle, etc. They can be seen as having a tree-shaped hierarchical structure. At a higher level, there are macro-categories, which further divide into sub-topics for the relative discussions. At the lower level, users can start what is defined as a thread, meaning they can post a message where they discuss a specific topic in the context of a certain sub-domain. The first message in a thread is called root and contains the original user question or opinion. Other users can reply to the root message giving origin to the typical thread structure.

[2] presents an approach to extract high quality ¡thread-title, reply¿ pairs where ¡thread-title¿ can be matched against user input to retrieve appropriate answers in domain-specific chatbot systems. Given a thread title, all the relevant replies part of that thread are extracted using an SVM classifier which simply labels the messages as relevant or not using features such as overlapping words and distance from the root messages. Then, a second SVM ranks all the extracted replies based on their contents and on the characteristics of the user, like reputation, and number of upvotes[9]. Finally, a subset of the top-N replies is used as chatbot knowledge.

Similarly, [43] collects ¡question-answer¿ pairs from community QA websites. In order to favor short replies, multiple sentences that are part of the same answer are considered separately. Relevance between sentences and question is computed based on a set of features working at different levels of granularity, including word, sentence, document, and topic level. A similar outcome is obtained by [1] by using rough sets and ensemble learning. Overall, the main limitations of this approach include the absence of context-dependent answers and noisy texts, full of irregular spelling usage and grammar mistakes.

Other studies attempted to extract chatbot knowledge from less structured documents, such as transcripts and books. in [44], the Dialog Diversity Corpus, containing a variety of transcripts in different domains, has been used to extract topics, categories, patterns, and templates, to build an ELIZA-like

[9]The number of positive feedbacks received on a particular message

chatbot using AIML. (fig. 2.8)

```
TUTOR [Opening remarks and asks student to read out aloud]
STUD [Reads problem] Mike starts a job at McDonald's that
will pay him 5 dollars and hour, Mike gets dropped off by his
parents at the start of is shift. Mike works a \h" hour shift.
Write an expression for how much he makes in one night?
[Writes \h*5 = how much he makes"]
TUTOR That's right number.
```

Figure 2.8: A snippet from the CIRCLE corpus part of the Dialog Diversity Corpus

In this case, the main drawback is the lack of a standard format for the documents. The absence of clear indicators, extra-linguistic annotations, too many speakers and irregular turn-taking, led to an extensive pre-processing phase. The transcripts had to be filtered to keep only the ones exhibiting a structured format, obvious turns without overlapping, and without any unnecessary notes.

Comparably, in [45], for the same purpose, different text corpora were used, such as the British National Corpus of English, and the Koran, the holy book of Islam, in which, given its nonlinear structure, verse and following verse are considered as turns. Again, consecutive turns are used to build pattern-template rules and the most significant words from each utterance are used as categorical triggers for that same rule.

2.3.3 Crowdsourcing

As anticipated in the introduction, to date, crowdsourcing is the most effective mean for the collection and annotation of datasets for chatbot systems. Crowdsourcing consists in the practice of engaging a 'crowd' or group of people to solve some kind of task. This concept has been promptly applied to data collection, which is considered generally as a tedious task. The advantages of having a large group of people may include improved costs, speed, quality, flexibility, scalability, and diversity of data. Contrarily to outsourcing, where the task is assigned to well-defined and known external entities, crowdsource workers are usually anonymous and with varying backgrounds, demographics, geographic location, and goals.

Amazon Mechanical Turk[10] (AMT) and Crowdflower[11] are some of the many online platforms connecting requesters, people proposing assignments, with workers.

In AMT, requesters can create short tasks and pay small amounts of money to registered workers, also called "Turkers", to complete them. A requester can create a group of tasks, defined as Human Intelligence Tasks (HITs) by choosing among a number of baseline templates customized for various domains, such as image tagging, utterance collection, intent detection, or audio transcription just to mention a few. Moreover, requesters can customize the layout and design of their task by editing a single-page HTML document where they can add descriptions, examples, questions, input forms, and other ad-hoc HTML elements used to collect worker's responses. The data shown in each HIT is fed as a single row from a CSV formatted document. Additionally, requesters can control how much money to pay for each completed HIT and how many unique workers should process each HIT. (Fig. 2.9) Furthermore, some filtering options allow selecting workers with a specific set of qualifications. Finally, requesters can review and approve each completed HIT or either reject them by providing a valid reason.

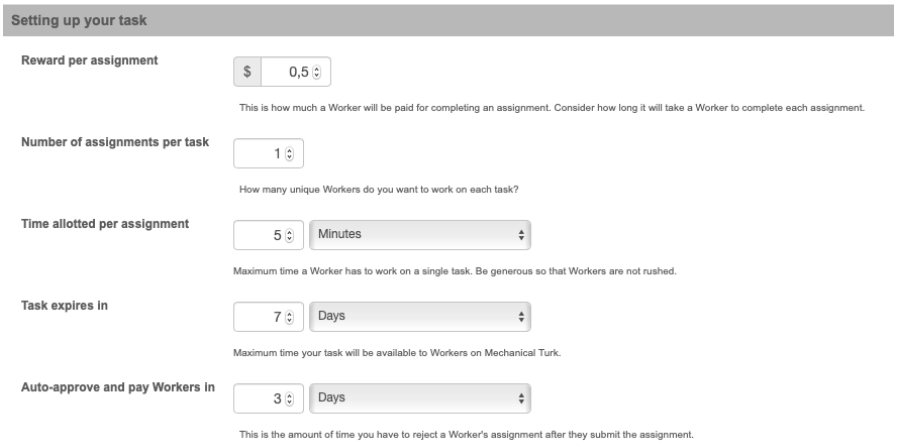


Figure 2.9: Amazon Mechanical Turk requester UI

AMT, as also Crowdflower and other platforms, provides researchers and developers with access to a seemingly unlimited amount of distributed resources, perfectly suited for the lengthy task of data collection and annotation.

[10] Amazon Mechanical Turk, URL: https://www.mturk.com
[11] Figure-eight, URL: https://appen.com

Crowdsourcing does not come without its drawbacks. One major problem regards the implementation of quality-control mechanisms to verify the job of each Turker, resulting in additional consumption of resources and manual checking. It is not uncommon to find Turkers who do not read the instructions or follow the guidelines, or simply try to trick the requesters in accepting large batches of responses by filling the forms with random data.

In [46], Turkers were tasked to write simple mathematical problems, providing a rationale of the solution and a selection of possible responses. Researchers found out that many workers, either proposed only small changes in the values of the examples or copied math problems from the web, which was not allowed from the instructions, resulting in duplicate problems.

Other research tried to assess the number of Turkers who actually read the instructions for the task they complete by hiding the answer to the questions in the text of the assignment. Only 28% of the workers proved to have read the instruction carefully and provided the right solution.[47]

Since we are focusing on chatbots and data collection for dialogue systems, an additional point of concern is represented by the bias introduced when designing the task, and the correlation between the lexical content of the prompt presented to the workers and the data collected.

In a study focusing on the key factors influencing crowdsourced paraphrase collection, which is tightly connected to the focus of this thesis work, researchers explored the effects of various factors on correctness, grammaticality and linguistic diversity; These aspects were variations in instructions, incentives, and different workflows.[42] The outcomes showed how simple examples result in lower diversity but better semantic equivalence, meaning that the original utterance and the paraphrase bore a similar significance. Also, feeding Turkers with paraphrases collected by other workers increased language diversity.

Other research explored the impact of using different elicitation methods to collect natural language sentences.[48] The proposed approaches were: providing a corresponding example sentence, adopting a storytelling approach by using multiple sentences to convey the goal of the elicited utterance, and a list-based method where a specific objective was presented alongside with a list of slots and values. The results reveal that the list-based method introduces the lowest bias, with a Spearman rank correlation coefficient between prompt and output of 0.08. The method providing example sentences showed the highest correlation with a coefficient of 0.43.

Despite its systematic issues, crowdsourcing has gained immense popularity in recent years for the development of dialog systems and has been vastly applied for NER, dialogue act annotation, paraphrasing, and response generation for NGL. Unmanaged crowds have been used by [47] to annotate entities in different domains and surprisingly exceeded the quality of in-house annotations. In [49], crowdsourcing has been used to maximize the lexical divergence between an original sentence and its valid paraphrasing by sequentially feeding crowd workers with the result of the previous paraphrasing round. Keeping at each round the N most dissimilar paraphrase showed a monotonic increase of linguistic divergence, allowing the creation of a good quality dataset in a cost-effective way. Natural language sentences for multi-turn conversations are collected in [36] by setting up a restaurant-finding scenario in which multiple workers take the role of user and system.

2.3.4 Interactive Learning

While the prevalent approach of crowdsourcing is to collect and annotate data to train a dialogue system, attempts have been made to improve system performances and enriching their knowledge bases after deployment. Actual users of the chatbots are asked to provide feedback, simulate missing interactions, or provide unknown answers. This idea takes the name of interactive learning and is often used in combination with reinforcement learning algorithms. The intent is to deploy a baseline system with a reasonable task-completion capability which will be further enhanced thanks to the interaction with real users.

In [50] interactive learning is used in three different phases. First, the system can ask the user to paraphrase the messages it does not understand, with the goal of learning different wordings for the posed question. Thenceforth, the system may ask for a more detailed explanation of the question and learn a policy where there is no need to ask extra information in the future. Finally, if no answer is found, the user is asked to provide the correct solution to the proposed question to expand the system's knowledge base.

Some other studies applied reinforcement learning to train a task-oriented chatbot through online interactions and user feedback. The results show that the agent corrects the mistake it makes by learning from user teaching, and thanks to reinforcement learning based on user feedback the task-completion capability improves.[51]

2.3.5 Data annotation pipelines

In the previous sections, we have described and exemplified different practices that help dealing with data collection and annotation, with particular interest for research focused on dialogue systems. Regardless, those techniques are rarely used alone but are often combined together into a pipeline to improve manageability and decrease costs. For example, it is unlikely to manually generate and annotate from scratch entire datasets in-house; instead, it is more reasonable to mix automated or semi-automated mechanisms in the early stage and then refine the results in the end with human supervision. Combining different approaches guarantees a good level of quality while sticking at the same time to a reasonable budget.

Automated methods can be used to generate or scrape from the web large corpora of data and dialogues. Labels can be extracted either from structured documents, such as forum threads, online tags, or by using an API-driven approach. Many APIs for different purposes, e.g, sentiment analysis, NER, POS tagging or image classification, have been made available by big companies, like Google Cloud Natural Language[12], Amazon Comprehend[13] and IBM Watson[14]. Those services can be sometimes integrated into the annotation process while considering the added noise and the cost of each API call.

All these data can be further processed or verified using manual in-house annotation or some relatively inexpensive methods like crowdsourcing.

[12]Google NLU, URL: https://cloud.google.com/natural-language

[13]Amazon NLU, URL: https://aws.amazon.com/comprehend/

[14]IBM NLU, URL: https://www.ibm.com/cloud/watson-natural-language-understanding

Chapter 3

Problem Setting

This chapter explains the details required to understand the context of this work. I will provide an overview of the company I have worked in while carrying out this thesis work. In particular, the software that will be implemented and described in chapter 4 has been developed using the company products and users as a real-world use-case. Thus, the importance of gaining a high-level outline of what this business does and what their service is about. I will also give a detailed description of what we called Service Registration Task which defines the class of problems specifically tackled with the experiments that have been carried out.

3.1 The company: 20Hexagons

20Hexagons is a company specialized in the development of software solutions with expertise in many different areas of software engineering and development. In the last few years, they piloted a project, called Siirtosoitto, in collaboration with the municipality of Helsinki and Stara, the Helsinki City Construction Services. Most of the effort of the company has been invested in developing and publicly launching this project which will be described in detail in the following section.

3.1.1 SiirtoSoitto

SiirtoSoitto is a free service offered to the residents of the city of Helsinki. Car owners can register to Siirtosoitto through a mobile app available for iOS and Android and start receiving reminders about upcoming street cleanings. After registering the license plate of their cars, users will be notified about scheduled or upcoming maintenance or roadworks in their selected areas of interest. If their car is still on the road when cleanings are about

to start, users will receive an automated call prompting them to remove the vehicle immediately and avoid it to be towed away.

The goal of the system is to provide benefits both to the municipality and the private users. The city administration will provide faster cleanings and reduce indirect costs of towings and inefficient use of machinery and workforce. On the other hand, registered users will be informed about upcoming parking limitations and avoid the unfortunate situation of having a car towed away and an 80 euro fine.

SiirtoSoitto doesn't only comprise of a mobile app for end-users (Fig. 3.1), but it's a complete ecosystem that includes:

- A web application for customer service and management
- A separate app for maintenance operators to scan the plates of cars still parked in areas with parking restriction and trigger an automated call
- The consumer app used by the user to register to the service and monitor the updates

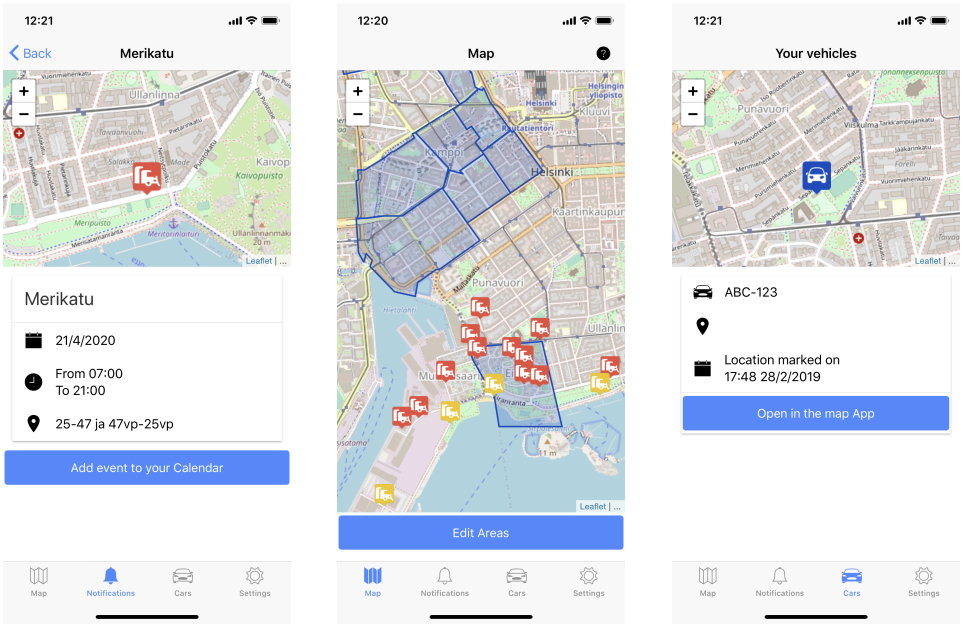


Figure 3.1: SiirtoSoitto consumer app UI

3.1.2 Internship work

The company wants to expand its product and I have been hired as a thesis worker with the goal of investigating the use of a chatbot system to bring the Siirtosoitto service to a larger number of people, in a seamless way, on an expanded set of communication channels such as Whatsapp, Telegram and Facebook Messenger. For what concerns this thesis work, the same rule-based chatbot developed to make SiirtoSoitto available on instant messaging app has been used as a gateway for some crowdsourcing experiments and integrated in the data generation and annotation pipeline described in chapter 4.

3.2 Service Registration Task

In this paragraph we provide a brief overview of the Service Registration Tasks (SRT), the class of interactions employed during the experiments to generate dialogues between an hypothetical user and a service's system.

Nowadays, most services require users to identify themselves or to provide relevant information for a correct functioning of the service itself. We, therefore, think that such procedure represents an important class of dialogue interactions between users and industrial chatbot systems that are characterized by a number of shared sub-tasks acquiring different values from system to system. While for the database querying tasks described in [7], the structure consisted in gathering preferences, querying a database, offering the results, and eventually selecting an item, for SRT we can identify the following sub-tasks. Gathering user information, validating user information, confirming user information, querying an API, and presenting the results of the operation.

3.3 Addressing the research questions

This work is revolving around the research of an answer to the three research questions stated in the introduction.

- **R1:** Can the recently presented M2M approach be effectively applied to other tasks apart from database querying applications?

To tackle this research question we will first implement the M2M automated dialogue generation system following the description provided in [7]. We will then validate the effective use of the proposed framework on the new set of

tasks described in 3.2 in a real-case scenario made available by the company service.

- **R2:** What are the differences in the resulting dataset quality comparing two different crowdsourcing sources: generic professional workers from online platforms, and current users of the already existing system?

To answer this question first, we will integrate the crowdsourcing step described in [7] into SiirtoSoitto rule-based chatbot. It will enable to gain feedback from a novel source of workers, the users of the service them-self. We will get the opportunity to evaluate the effect of extremely different sources of crowdsourcing by comparing the data collected via Amazon Turkers to the SiirtoSoitto data coming from the chatbot integration.

- **R3:** Could M2M be integrated into a rule-based chatbot with the ultimate goal of crowdsourcing an annotated dataset to bootstrap a new neural-based dialogue system?

We will explore the feasibility of combining the M2M framework and the rule-based chatbot by creating a pipeline that allows to integrate the dialogue outlines generated using M2M with the chatbot itself in order to crowdsource rewrites directly from SiirtoSoitto users. The annotated dataset obtained from these experiments will be used to train an *NLG* module to showcase if, and how, these data can be practically used to train neural-based dialogue systems.

Chapter 4

Method

4.1 Introduction

Machine Talking to Machines is a framework that combines automation in dialogue simulation and crowdsourcing to generate and collect high-quality annotated datasets for the training of dialogue systems. This approach has been proposed in [6, 7] and successfully applied on database-querying tasks for goal-oriented chatbots. The data collected with M2M showed greater diversity and coverage compared to similar datasets while guaranteeing the naturalness of the utterances and the quality of the annotations.

As stated in section 3.2, we now decide to focus on a slightly different kind of task which has been defined as (SRT).

The project consists in implementing the M2M method applied to this kind of task contextualized for the SiirtoSoitto service described in 3.1.1 and integrating it in a rule-based chatbot in order to harvest data with the help of actual users of the system. The work can be divided into three main phases. First, we implemented the rule-base chatbot with the goal to deploy a baseline solution that will provide users with an acceptable grade of goal completion and satisfaction. Then, we implemented the M2M framework as described in [6] with small variations to adapt it to the newly chosen class of tasks. Finally, we integrated M2M with the rule based-chatbot by adding additional conversational interactions that allowed users to contribute to the data collection. In chapter 5 I will then proceed to analyze the data harvested as part of the experiment and discuss how such data can be potentially used to bootstrap a new neural-based model substituting the rule-based chatbot. The intent of ultimately having a neural-based model is

to provide more robustness, variety, and flexibility in the Human-Machine interaction.

4.2 Rule-based chatbot

This section provides a design and implementation overview of the rule based-chatbot that has been deployed as a baseline solution to initially make the service available. In particular, we will present some of the important software and design requirements, how the new addition to the SiirtoSoitto family has been integrated in the legacy ecosystem, and how the rule-based dialogue manager has been implemented.

4.2.1 Requirements and constraints

the complete compatibility of the new module with the pre-existing system is, among some others, the imperative requirement identified during an introductory phase with Twenty Hexagons executives. The goal is to use the same database and the same schemas with as little disruption as possible.

The new chatbot system supports the most common communication channels for online messaging and allows maximum scalability in case a new interface needs to be added. Currently, the selected messaging platforms are Whatsapp, Telegram, and Facebook Messenger, because together they cover the biggest share of the audience. To guarantee scalability, all the platforms communicate back and forth with the underlying dialogue manager by extending an interface that implements a number of pre-defined methods (fig. 4.1). The interaction between the various communication platforms and the chatbot server makes use of webhooks. For each channel, a new endpoint is exposed and all incoming requests are redirected to it (fig. 4.2).

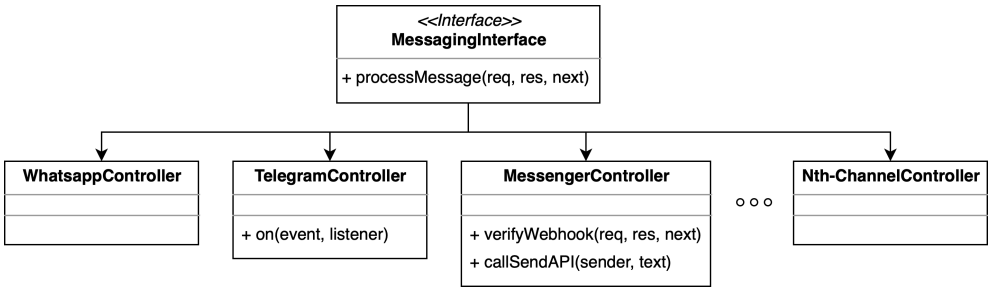


Figure 4.1: Messaging interface schema

Another feature of paramount importance is the flawless support of dia-

logues in different languages. While Finnish is expected to represent the preferred choice of most users, Swedish localization is required as it is the second official national language in Finland and it's spoken by a minority of people. Finally, English support is added for completeness and can be used by whoever doesn't fall into the two previous categories. To support these languages, three separate instances of the same dialogue manager have been created, each using the same rules used to match replies and answers but supported by a separated knowledge base, one for Finnish, one for Swedish, and one for English.

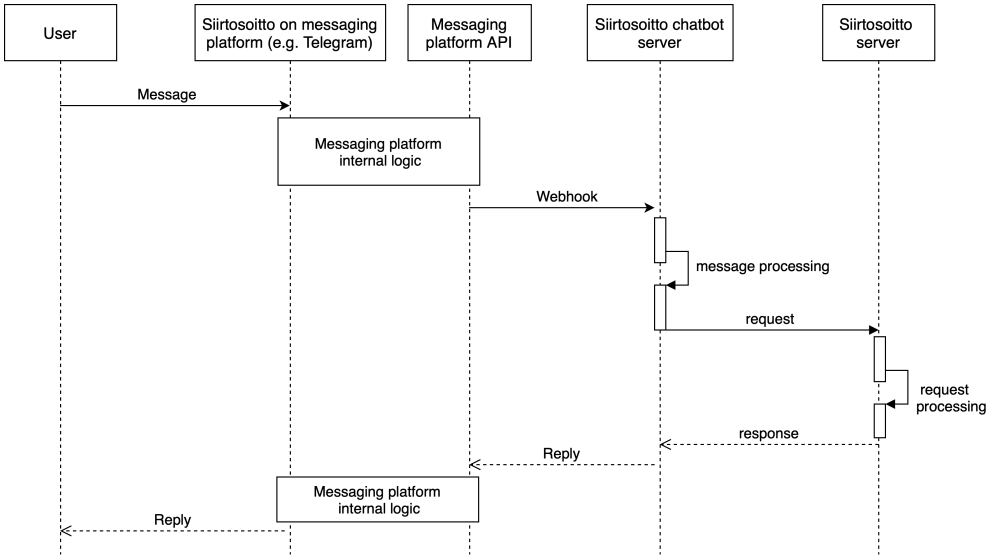


Figure 4.2: Message flow diagram

The chatbot server only acts as a dialogue system, all API calls are issued to the pre-existing SiirtoSoitto back-end. Additional requirements regard the secure communication between the chatbot and legacy server. This goal has been accomplished by signing all requests with a 128-bit API key and validating it server-side.

During an interaction between the chatbot and a user, the latter may be asked to prompt some information such as its phone number, license plate or the name of areas of interest. Such data must comply with certain requirements and characteristics, as shown in table 4.1; for example, phone numbers must be Finnish phone numbers and any area must exist in the database of the streets and parking areas of the city of Helsinki. While with a traditional application this information would be filled into forms and can

be independently validated, for a chatbot system this parallelized and asynchronous process cannot take place since the human-machine interaction is bound to a sequential textual feed. Therefore, in the new system, each piece of information is validated as soon as it's prompted and the user is immediately notified whether there are any issues. Here follows a table detailing all the different kinds of data handled by the SiirtoSoitto rule-based chatbot, what are the requirements for this information, and how they are validated on the go.

Datatype	**Description**	**Constraint**	**Validation**
Phone number	The user's personal phone number used by the system to perform automated calls	Must be a valid Finnish phone number	RegEx
Confirmation pincode	4-Digit code sent to the user's phone via SMS used to validate phone number ownership	Must match the code generated by the system	Exact match
License plate	The license plate of the car for which the user want to be notified in case of imminent towing	Must be either a Finnish (FI), Estonian (ES) or Swedish (SW) license plate	RegExes
Area of interest	The name of the street, parking area or zipcode for which the user want to receive notifications in case of scheduled or imminent cleanings/roadworks	The area must exist in the system database. Search must support presence of spelling mistakes	Non-exact match with fuzzy search based on n-grams

Table 4.1: Different kinds of data required by SiirtoSoitto and how they are validated

4.2.2 Integration with pre-existing system

The chatbot only represents the last addition to the already existing Siirtosoitto service. As mentioned in 3.1.1 the ecosystem is comprised of mobile applications for citizens and operators and a web app for the city administrative services. All these applications communicate with a centralized back-end system that takes care of managing the access to the database, exposing an API for various kinds of requests, and sending push notifications to user's devices. It also periodically downloads and processes data about cleanings from Helsinki city services and interacts with the automated call service to send warnings about imminent towings.

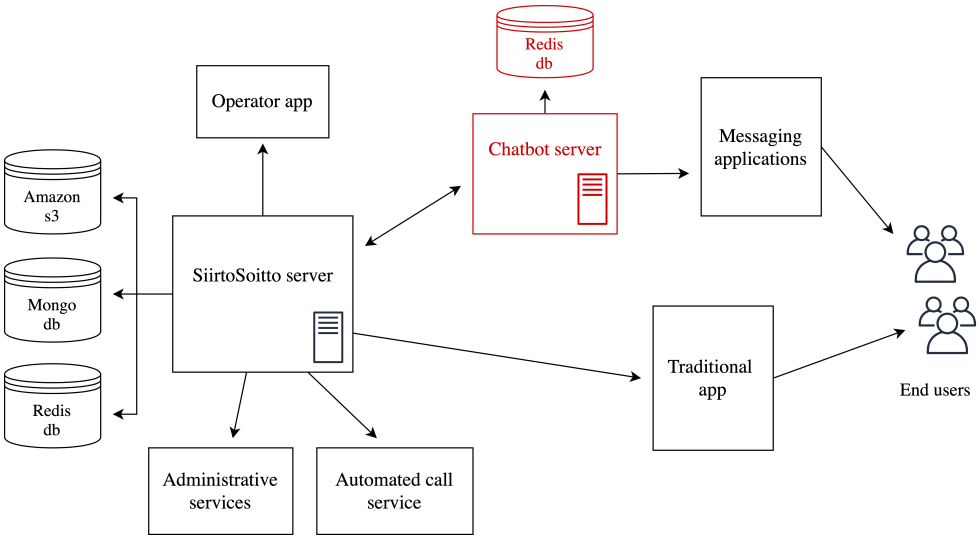


Figure 4.3: SiirtoSoitto architecture. The new components are shown in red

Apart from developing the chatbot system itself, a few major updates have been also applied to the existing server in order to integrate the new software. In the following listing, we present the database schema for the User object.

A new field *channel* has been introduced to discriminate between users

```
user: {
    name: String,
    email: String,
    password: String,
    role: enum,
    channel: enum,
    chatUserID: String,
    state: enum,
    phoneNumber: string,
    preferredLanguage: enum,
    pin: String,
    pinSentAt: Date,
    vehicles: [ObjectId],
    areas: [String],
    streets: [String],
    resetToken: String,
    resetSentAt: Date,
    deviceId: String,
    termsAcceptedAt: Date
}
```

using the mobile app, the chatbot on Whatsapp, Telegram, or Messenger. An attribute $chatUserID$ is associated with chatbot users and it is required to forward replies to the right user through the messaging apps. The notification module has also been heavily reworked. While previously it was just thought to send push notification to iOS and Android devices using a third-party service, now its architecture has been made more scalable. Depending on the user channel the same notification object is sent either through the external service or through the online messaging app interfaces.

4.2.3 Rivescript

Rivescript[1] is an open-source chatbot scripting language meant for the development of rule-based dialogue systems, that has been used to implement the dialogue manager of the SiirtoSoitto chatbot. It was developed by Noah Petherbridge with the goal of providing an easy-to-learn and immediate way to write triggers and rules using pattern matching. Inspired to the AIML language described already in 2.1.3, Rivescript aims to give up its obscure and complicated XML-like syntax with the objective of writing dialogue manager scripts that are quick to type, and easy to read and maintain. The syntax is very linear and the scripts are saved inside files with a *.rive* extension; there are a number of command characters that appear at the beginning of each line of the script to indicate the function of the remaining text, as shown in Fig. 4.4:

- + indicates a trigger and it's followed by an expression that is matched to the user input using wildcards, optional words, alternative phrasings, and arrays.
- - indicates the system response template, it supports random replies, redirections, and dynamic replies by processing programming code directly inside the document.
- * support conditionals and trigger different behaviors based on variable values or function calls
- % is used to force a trigger to be fired only when the previous last response matches the subsequent inline pattern.

Obviously, the Rivescript language represents only a high-level abstraction and in order to practically use it inside a program, an interpreter written in that particular programming language is required. Since the whole system implementation is based on Node.js[2], an interpreter available on the Node Package Manager registry[3] was used. As mentioned before, separate knowledge bases containing the rules in .rive documents have been written for each supported language. We have encountered some issues with the parsing of Finnish and Swedish characters because they are not part of the ASCII character encoding standard upon which the parsing module is based. As part of the solution we first enabled an option to parse using the UTF-8

[1]Rivescript, URL: https://www.rivescript.com
[2]Nodejs, URL: https://nodejs.org/en/about/
[3]NPM, URL: https://www.npmjs.com

encoding standard[4] which in turn presented a few bugs that prevented some patterns from being correctly matched. The problem was overcome by using a special "?" command character which was not part of the original formulation of the scripting language but was included in the Node.js parser to fix those issues.

```
+ [*] (remove|delete) [*] (street|area|parking zone) [*]
* <get registered> == false => You need to register first!
- Here is the list of your areas: \n
^ <call> print_area <id> </call> \n
^ Enter the one you want to remove:{topic=delete_area}
```

Figure 4.4: An example of how command characters and other syntactic elements are used to define rules

4.2.4 Deployment

A deployment pipeline allows every time a new version of the program is pushed to the master branch of the project git repository, to automatically execute unit and end-to-end tests and finally deploy and run the updated application on Heroku servers. Heroku is a Platform as a Service (PaaS) that helps to easily deploy programs written in a variety of supported language, Node.js included, without coping with the underlying infrastructure or any software update. All Heroku instances, also called dynos, are hosted on Amazon EC2 servers, an Infrastructure as a Service (IaaS) that provide a 99,99% availability. We use a Redis database instance to keep track of the dialogue sessions between users and chatbot, while a Mongo database is used to store persistent data about users, vehicles, towings, and all the remaining data processed by the system.

4.3 Machines Talking to Machines

The introduction of the Machines talking to Machines (M2M) approach rises from the difficulty in collecting and annotating dataset for goal-oriented dialogues. Different instances of the same task involve different schemas and entities.

[4]UTF-8, URL: http://www.utf-8.com

One popular approach for task-oriented dialogue collection is called Wizard-of-Oz. The user interacts with what he thinks being a software agent but is, in fact, a human "wizard" disguised by a software interface. The wizard converses with the user and outputs sentences by typing and using text-only interactions. Even if broadly used, this method lack scalability, is too dependent on the initiatives of the speakers, and may not provide sufficient coverage of relevant dialogue flows. It is also very difficult to simulate the errors, limitationsm, and responsiveness of real software systems.

Inspired by this practice the authors of the paper[6] proposed a new framework that guarantees more variety and dialogue paths coverage while maintaining naturalness in the utterances and high-quality annotations.

First, a task specification schema is used as a support for the exhaustive generation of dialogue acts sequences enclosing the semantic content of a dialogue. The sequences are generated by a simulated user and a system bot whose operation is described in detail in sections 4.3.4 and 4.3.5. Dialogue templates are then built on top of these semantic parses, using a simple domain grammar. The results are slightly unnatural computer-generated dialogue utterances paired with their semantic representation in the form of dialogue acts. The dialogue templates are then fed to a crowdsourcing platform where users provide contextual rewrites of the machine-generated utterance.

The main advantages are the following: the quality of the semantic annotations is guaranteed by this reverse processing. Instead of annotating natural utterances, we are building dialogues upon annotations. The automated generation of the outlines guarantees greater diversity and the exploration of all the relevant path conceived by the task designer. Finally, the naturalness of the utterances is guaranteed by the presence of human writers, and the variety is boosted by asking to rephrase highly generic machine-generated sentences.

M2M is conceived as being a domain-independent framework, which generates dialogues centered on completing a certain task. The developer provides the task-specific information used by the system.

$$F(T) \rightarrow D = \{d_i, i \in 1...N\}$$

$$d_i = [(u_1^i, ..., u_{n_i}^i), (a_1^i, ..., a_{n_i}^i)]$$

The framework F receives as an input a task specification T from the developer, containing all the information needed to generate the dialogues.

The output is a sequence of arbitrary size N of dialogues D generated from the interactions of the user and the system simulator. Each dialogue d_i contains a sequence of turns made up of a natural language utterance u_n and the corresponding semantic annotations a_n. The content carried out by the annotations includes dialogue acts and, for example, additional information about the speaker.

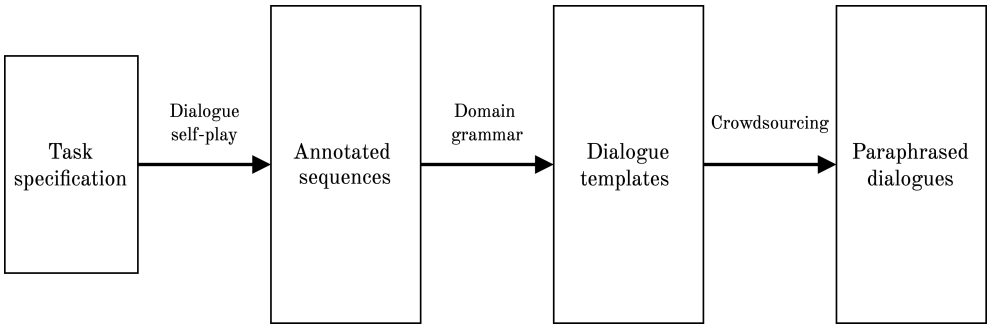


Figure 4.5: High-level M2M architecture

4.3.1 Task specification

As explained in the section above, the input for the M2M framework is a task specification that drives the user and system simulators during dialogue generation. In this thesis, the class of tasks we decided to tackle has been named service registration tasks (SRTs) and involves the domain-agnostic user act of registering to a service. The key elements of this phase are the collection and validation of information and preferences from the user. We can imagine these entities to induce a schema of slots S that needs to be filled with their corresponding value during the conversation. While the number and type of slots are derived directly from the particular kind of SRT instance, the slot schema specification is still issued to the M2M framework by a developer which acts as an intermediary between the high-level abstraction of the task itself and its projection on a task schema. The developer also provides a second element, that together with the task schema form the task specification T. It is a domain-dependent client API C which can be queried to validate the slot values.

$$S : taskschema$$

$$C : clientAPI$$

$$T = (S, C)$$

The ultimate goal of the system is to generate annotated dialogue built around the task specification provided by the developer. This process can be divided in two steps, $F = F_2 \circ F_1$. First, F_1 uses the task specification and map it to a set of outlines O. Then, F_2 makes use of the help of crowdsourcing to map the contents of each outline to a sequence of natural language utterances d.

$$F_1(T) \rightarrow O = \{o_i, i \in 1...N\}$$
$$o_i = [(t_1^i, ..., t_{n_i}^i), (a_1^i, ..., a_{n_i}^i)]$$
$$F_2(o_i) \rightarrow d_i$$

The contents of an outline o_i are two sequences of equal length of turn annotations a_j^i and corresponding template utterances t_j^i. Turns annotations contain one or more dialogue acts that encapsulate the semantic meaning of a particular dialogue turn. Template utterances are used to represent the flow of the dialogue using very simple and generic sentences that reduce the biases caused by variation in natural language. The templates provide two main benefits. They are easier to automatically generate because they don't require complex and diverse language and implicitly increase the lexical richness in the paraphrases generated by crowd workers.

The SRT instance taken into account for this thesis work, based on the Siirtosoitto service registration process, resulted in a task specification containing the following slot types: *notification_area, phone_number, license_plate, pincode, acceptance_of_terms_and_conditions*. The client API is instead based on the validation criteria highlighted in table 4.1 and it is also able to provide suggestions of the true value of notifications areas in case of minor spelling mistakes.

4.3.2 Scenario generator

During the first phase, a scenario is sampled from the task specification to encapsulate user goals and behaviour. The goal of the user is to complete a specific task characterised by an exchange of information. For each slot in the task specification, one or more values are sampled from all the possible accepted values for that slot. They can be retrieved from a knowledge base containing a finite set of values (e.g. categories, as for the Helsinki areas in our use case) or be generated following a regular expression (e.g. IDs, dates or, in this work, license plates and phone numbers). In addition, with a probability P_{err} some non-existent values are chosen in order to create unsatisfiable goals.

Apart from the user goals, a scenario is also characterized by a user profile that defines the behaviour of the user during the conversation. Different personalities have an impact on the dialogue flow and help increasing the diversity of the generated outlines and simulating a wider range of interactions. Users may differ for the number of slot information provided at each turn of the conversation. More extrovert users specify more than one entity slot within each sentence while more introvert profiles prefer to give one information at a time following the guidance of the system. Some users may be more or less prone to change their ideas and updating the values of some of the already specified fields. Different profiles also correspond to different probabilities of introducing spelling mistakes which reflect different typing abilities. The user profile p is practically implemented as a vector of probabilities concerning some task-independent characteristics of the user behaviour. The profile is later passed to the user simulator to influence its behaviour accordingly.

$$s_i = (p_i, g_i)$$

In this thesis work, for each scenario s_i, we build the set of user goals g_i by accompanying each *task_slot* class with a suited method to generate corresponding values. For example, Finnish phone numbers are generated using an appropriate regular expression with probability $(1 - P_{err})$ while an invalidating prefix is added otherwise. Similarly, one or more license plate numbers are generated using regular expressions, the 4-digits pincode is extracted from a uniform distribution with support 1000-9999 and the values of the notification areas are randomly selected from a database. The user profile takes into account two characteristics, the typing errors and the predisposition to provide more than one piece of information during each dialogue turn.

$$p = (p_{typo}, p_{verbose}, d_{verbosity})$$

$$d_{verbosity} : f(k) = (1 - p_{verbose})^{k-1} p_{verbose}$$

The first element p_{typo} is just a very low valued probability applied to each letter of the notification area slots. It can be seen as the mutation probability used in genetic algorithms; we go through each letter of the entity and substitute it with a random character with probability p_{typo} ¡ .1 else we leave it as is. The second and third element are respectively the probability $p_{verbose}$ of victory in a *Bernoulli trial* and a geometric distribution $d_{verbosity}$ with a density function $f(k)$. At each turn, the user simulator samples an integer value k from the distribution that indicates the number of trials required before obtaining a victory when the victory probability is $p_{verbose}$.

The value k is used to decide how many slots the user will fill in during a single turn. To model an extrovert and chatty user a low value of $p_{verbose}$ will result in generally higher k values and more information per turn. On the opposite, a higher probability means that only a single piece of information will be provided in each sentence. The geometric distribution was a good choice to represent the behaviour of different kinds of users at each turn. Different types of distribution have also been taken into account such as the *Poisson*. In this case, the main problem is that a k equal to 0 is often extracted, basically resulting, based on our modelling, in an unresponsive user. (fig. 4.6)

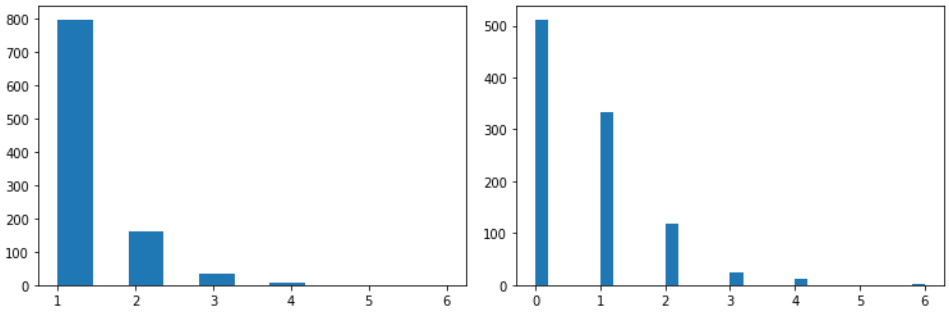


Figure 4.6: Histograms with the values extracted after 1000 samples from a geometric (left) and Poisson (right) distribution using p=0.8 (introvert user)

4.3.3 Dialogue self-play

Once a scenario has been sampled from the task specification the M2M framework uses it to generate an annotated dialogue structure. The dialogue is built with the support of two simulators: a user bot and a system bot. In a task-oriented dialogue, information flows between the two actors. In this specific case, the goal of the system is to collect and validate values for all the slots generated in the scenario.

The interaction between the two bots is sequential; at each turn one of the two simulators take an action and outputs a turn annotation made of a sequence of dialogue acts. Each action is depending on the dialogue history and the current scenario.

$$U_{sim} = P(a^i_j | a^j_1, ..., a^i_{j-1}, p_i, g_i)$$
$$S_{sim} = P(a^i_{j+1} | a^i_1, ..., a^i_j, S, C)$$

The user simulator at each turn can be represented as a distribution over the possible dialogue annotations conditioned by the dialogue history, the

previous turns annotations $a_1^i, ..., a_{j-1}^i$ and also the scenario in terms of user profile p_i and goals g_i. In a similar way, the system simulator maps the dialogue history, the task schema S, and the client API C on a distribution over possible annotated turns.

The user simulator U_{sim} has been implemented as an agenda-based simulator while the system simulator S_{sim} has been conceptualized as a finite state machine that reacts to the input coming from the user. The behaviour and realization of both simulators will be detailed in dedicated paragraphs 4.3.4 and 4.3.5.

The interaction between the two agents stops based on several criteria. The conversation ends successfully only if the goal is accomplished, meaning that all the slots contained in the dialogue frame have been filled successfully. Otherwise, either the conversation exceeds a pre-defined number of turns *turn_lim*, or an inconsistent state is reached because the scenario contains invalid values. In this work, the system and the user simulate the SiirtoSoitto registration process using the data sampled with the scenario. The goal is reached when all the required information, phone number, license plates, notification areas, pincode, and acceptance of terms and condition are collected and validated. The task will fail, and the dialogue will be interrupted if, for example, the terms and conditions are not accepted or the scenario generator introduced invalid data, such as a non-Finnish phone number.

This dialogue generation process is repeated iteratively as new scenarios are sampled from the task specification. Different slots, user profiles, and simulator actions result in a rich set of dialogue annotations characterised by diversity and excellent path coverage.

4.3.4 User simulator

In this section, the implementation and functioning of the user simulator module will be explained in detail. It is an agenda-based user simulator largely based on the work in [52].

An agenda is a stack structure that contains all the dialogue acts that the user needs to convey in order to reach the desired goal. The goal is made of two components, a set of constraints C which are the values drawn from the scenario provided to the simulator, and a request R that reflects the user desires.

$$G = (C, R)$$

The agenda is built from the constraints contained in the user's goal by converting all the values into $inform$ acts and by adding at the bottom of the stack the user request R as a $request$ act. An additional *bye* act element is added at the end to wrap-up successful dialogues. In this work, a typical agenda (fig. 4.7) contains $inform$ acts for phone number, license plates, areas, and all the information that the user must provide to the system. The $request$ act that should be satisfied at the end is a confirmation of the successful registration to the SiirtoSoitto system. In each turn, the user state is defined by the current content of the agenda A and the user goal G. For simplicity we assumed a static goal G, meaning that the user won't change constraints as the dialogue unrolls, e.g, changing a notification area, suddenly rejecting previously accepted terms and conditions or changing its objective to anything else apart from registering to the service.

$$S = (A, G)$$

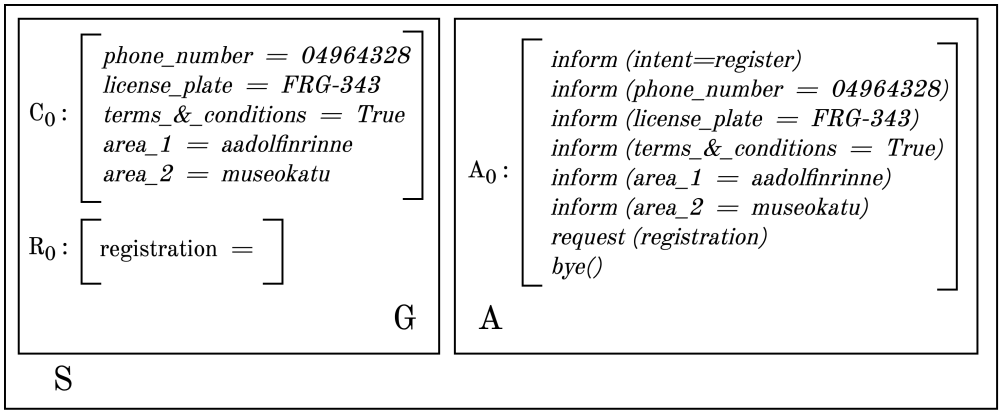


Figure 4.7: A possible user state at t=0. The user goal G is composed by constraints (C_0) and requests (R_0), while the agenda (A_0) encapsulates all the information into dialogue acts

At each step, the simulator outputs a dialogue annotation a_u, that we will call user action, obtained by removing and concatenating the first n dialogue acts from the top of the agenda. The user action is then fed as the input of the system simulator described in the next section. The corresponding system action received by the latter serves as a new input of the user simulator and concludes a full turn cycle. At each phase of this process, the user state gets updated, first when the user action is selected, and then when a system action a_m is received.

$$S \rightarrow a_u \rightarrow S' \rightarrow a_m \rightarrow S''$$

The dialogue acts in the agenda are ordered in a priority-wise fashion, $A[N]$ denotes the top element while $A[1]$ the bottom dialogue act. We can define a user act by indicating with the signature $a_u[i]$ the i-th element of the dialogue annotation. Since the elements are popped from the top of the agenda stack the complete content of a user act selection can be defined as follows:

$$a_u[i] := A[N - n + i] \;\; \forall i \in [1..n], 1 \leq n \leq N$$

The value of n indicates the user propensity to provide information and in this thesis corresponds to the value k sampled from the geometric distribution defined in the user's profile. An extrovert user will be characterised by a distribution d with expected values larger than 1, thus sampling from there a value for n will results in user actions sometimes containing multiple dialogue acts. The action selection process that has just been described can be modelled as a Dirac delta function δ.

$$P(a_u|S) = P(a_u|A, G) = \delta(a_u, A[N - n + 1..N])$$

This can be seen as a probability distribution that takes a value of 0 for each element of the support unless the value of a_u corresponds to $A[N-n+1..N]$, for which the probability is 1. (fig. 4.8)

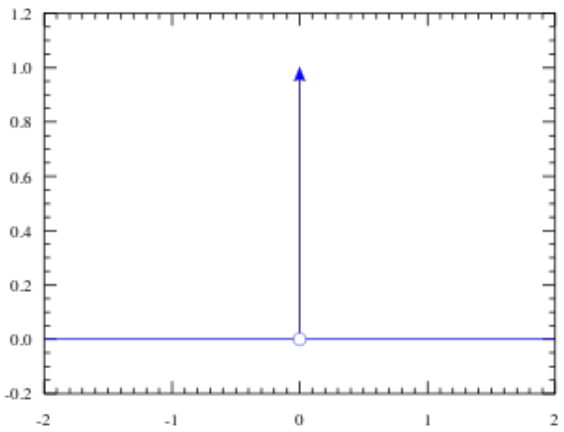


Figure 4.8: Dirac delta function representation

Once the user action has been selected the internal state of the simulator must be updated $S \rightarrow S'$. Since the state S is characterized by two elements, the agenda, and the user goal, we define the updated content of the agenda as A'. The updated agenda contains all the elements from the previous state in the same order without additions and with the dialogue acts selected for

the user action removed.

$$N' = N - n$$

$$A'[i] := A[i] \;\; \forall i \in [1..N']$$

$$|A'| = N'$$

$$A \supset A'$$

Using the above formalization of A' and re-stating the assumption that the user goal is static the state transition after user action selection can be modelled as:

$$P(S'|a_u, S) = P(A', G'|a_u, A, G) = \delta(A', A[1..N'])\delta(G', G)$$

Once again the notation with the Dirac delta function indicates that the only accepted user state transition is when the goal remain the same, $G = G'$, and the new agenda A' contains only the N' elements form the previous one.

The second state transition happens when the user simulator receives as an input a system annotation a_m. The system annotation always comes as a direct reply to the previously generated user action, (meaning that the conversation is initiated by the user), and updates the user agenda and goal. By assuming conditional independence between the new agenda A'' and the updated goal G'' the second state transition can be modelled as

$$P(S''|a_m, S') = P(A''|a_m, A', G'')P(G''|a_m, G')$$

where the first term indicates the update of the agenda and the second how the goal is changed in response to the dialogue acts contained in the system action. In order to simplify the treatment of the agenda update, we can simply consider that the initial N' elements remain the same and that the system action translates into a sequence of push operations on top of the agenda.

$$\begin{aligned} P(A''|a_m, A', G'') &= P(A''|a_m, A') \\ &= P(A''[1..N'']|a_m, A'[1..N']) \\ &= P(A''[N'+1..N'']|a_m)\delta(A''[1..N'], A'[1..N']) \end{aligned}$$

The delta function indicates that the first N' elements of the agenda remains the same with probability 1 while the content of the additional elements depends on the system action. As mentioned before the system action induces a sequence of push operations on top of the agenda introducing new

elements. Nonetheless not each dialogue act contained in the system annotation corresponds to a push operation. As a consequence, if M is the number of dialogue acts, then the new agenda will contain $N'' \leq N' + M$ elements. Denoting as D the number of new elements $N'' - N'$ the push operations can be modelled as follows.

$$P(A''[N'+1..N'']|a_m) = P(A''[N'+1..N'+D]|a_m[1..M])$$
$$= \prod_{i=1}^{D} P(A''[N'+i]|a_m[f(i)])$$
$$F \subset \{1..M\}, |F| = D$$
$$f(i) : \{1..D\} \rightarrow F$$

with F being a subset of the indices from 1 to M with cardinality D and f(i) mapping an index from 1 to D to a corresponding value in F Let's imagine that a system action contains three dialogue acts (m_1, m_2, m_3) but only the first and last of them will trigger an update in the user agenda. In this case $F = \{1, 3\}$, $D = 2$, $f(1) = 1$, $f(2) = 3$. The newly added elements can be defined using a set of simple heuristics, for example, *request* acts are transformed into $inform$ and $confirm$ acts are translated either into $affirm$ or *negate* acts based on the value contained in the slot. There may be the case in which the push operation introduced some duplicate elements in the agenda, e.g, the system asked to provide immediately some information that the user planned to provide at a later stage. To prevent inconsistencies the new agenda is scanned from top to bottom and cleared of any duplicates acts.

$$A'' = clear(A'')$$

Finally, the only missing transition that still requires a formalization is the user goal update. We can assume the conditional independence between requests and constraints and split the model into two parts using the chain rule of probability.

$$P(G''|a_m, G') = P(R''|a_m, R', C'')P(C''|a_m, R', C')$$

but since the constraints have been assumed as static it can be rewritten as

$$P(G''|a_m, G') = P(R''|a_m, R', C'')\delta(C'', C')$$

In the dialogue task considered in this work the only user request is receiving the confirmation of a successful or failed registration to SiirtoSoitto and the new goal state actually depends only on the presence of the corresponding $inform$ act among the system actions.

4.3.5 System simulator

The system simulator receives as an input a user action and reacts accordingly, based on its current state. The system is more passive compared to the agenda-based simulator used to mimic the user. As a matter of fact, the conversation is always initiated by the latter, and the system only reacts to the user messages. This doesn't mean that the conversation is totally driven by the user since at each turn the system explicitly tries to elicit some of the missing information. This scenario models a realistic case where the user does not know in advance all the information required during the SRT and the system suggests what is still missing.

Given these premises, the simulator has been implemented as a finite state machine (FSM) (fig. 4.9). In particular, it can be represented as an instance of a Mealy machine. A mealy machine is a deterministic finite automaton (DFA), which is a subclass of FSM. A FSM is a mathematical model of computation, which maps a list of states to the transitions it can make as a response of provided input. A Mealy machine is a subclass of FSM additionally characterised by an output at each transition step. Differently from a Moore machine, in a Mealy machine, the output does not only depend on the current state but also on the input. It can be formally defined by a sextuple of elements:

$$M : \{S, S_0, \Sigma, \Lambda, T, G\}$$

- S is a finite set of states
- S_0 is an initial state, part of S
- Σ is a finite set of inputs
- Λ is a finite set of outputs
- T is a transition function mapping the current state and an input to a new state ($T : S \times \Sigma \rightarrow S$)
- G is an output function that maps the current state and an input to a output value ($G : S \times \Sigma \rightarrow \Lambda$)

In the implementation of the system simulator, the set of states S corresponds to the potential intermediary states of the bot while performing elicitation and validation of the user data. S_0 is a starting context while the system is yet to receive the first message. The input set Σ corresponds to the finite set of possible user actions generated by the user simulator, while the output set Λ is the finite set of possible system actions. Similarly to the user annotations, the system actions are made of a sequence of one or more concatenated dialogue acts. T and G are projected into a number of case-specific rules that match the dialogue acts received as an input and the current state to a response made of one or more dialogue acts. In the practical implementation these rules are supported by an internal representation of the dialogue frame, built from the task schema, that collects all the information required from the user. For example if the system received an *inform* act it will transition to a validation state where it matches the value with the criteria defined in table 4.1 using the client API cited in section 4.3.1; in case of positive result it goes to a requesting state while a *request* act for an empty slot of the dialogue frame is generated and sent back to the user.

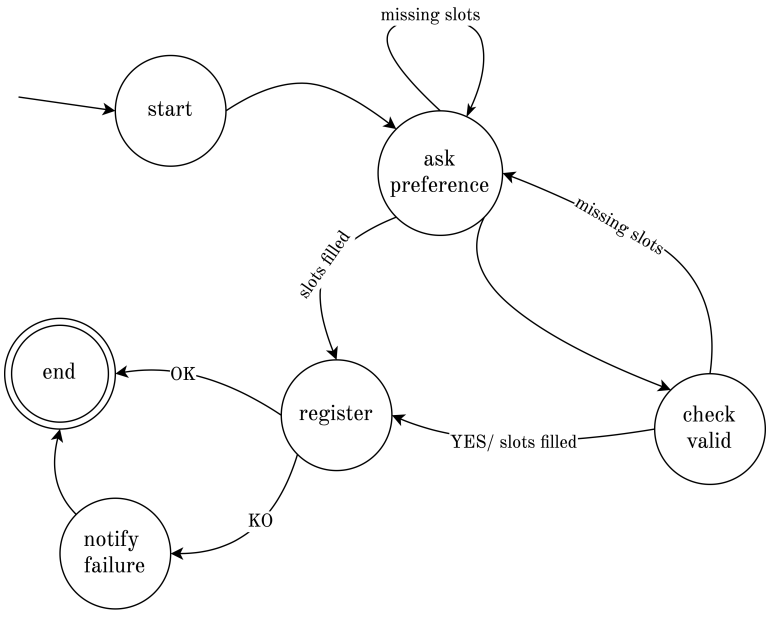


Figure 4.9: System simulator transition diagram

4.3.6 Template utterance generation

The dialogue self-play between user and system simulator results in a sequence of turn annotations $a_1, ...a_j$ each made of a sequence of dialogue acts in the form of $\{act, slot, value\}$. The list of possible acts is encoded in the rules implemented to build the user agenda in 4.3.4 and the system replies in 4.3.5 and is a subset of the dialogue acts proposed in [26]. The slots instead correspond to the categories of entities for which a value is requested from the user and are listed in the task schema. A slot can optionally be paired with a value. Noticeably, not all acts must have slots, and not all slots must have a value; but all values correspond to a slot and all slot are associated with some act. Here follows a list of all the acts and slots associated with the SiirtoSoitto use case.

acts
inform
request
confirm
negate
affirm
notify success
bye
greeting
propose

slots
phone_number
license_plate
notification_area
pincode
terms_and_conditions

At this point, we want to translate the turn annotations which are hardly understandable for human users, to utterance templates with a generic but clear structure. We apply a domain-general grammar G customized with a list of pre-determined templates to have some better control over the meaning conveyed in some annotations. The purpose of the grammar is to map turn annotations a into template utterances t. First, we introduce a hierarchy in the content of each dialogue act contained in the annotations. At the top level we place the act type, then optionally a slot type which in turn may contain a value.

$$\{act, slot, value\} \rightarrow act(slot(value))$$

The left part of each grammar rule consists of a triplet of previous, current, and next dialogue act and maps the current act, contextualised by

the previous and next, to an utterance skeleton containing references to the slot types and their values. Subsequently, the slots' references and values are substituted using the actual values and a set of customised templates that helps in providing the correct semantic meaning to the sentence.

$$(prev, curr, next) \rightarrow \text{utterance skeleton}$$

In this two-steps approach the act-level grammar rules provide the syntactical structure while the custom templates refine the semantic meaning. Tables 4.2 and 4.3 show a list of some of the grammar rules implemented in this thesis work for the generation of SiirtoSoitto-related dialogue templates.

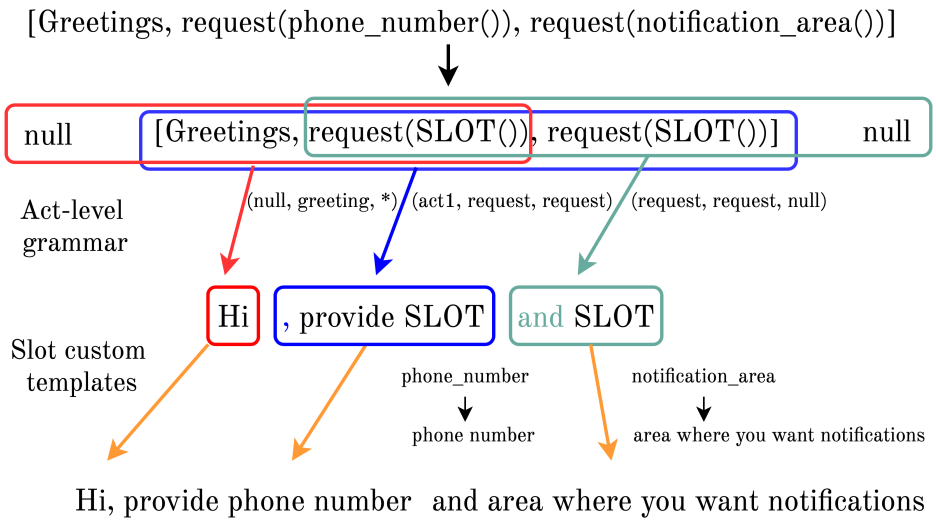


Figure 4.10: Example of generation of a template utterance using grammar rules and custom templates

Left	Utterance skeleton
(null, greeting, *)	Hi
(*, request, *)	provide SLOT
(request, request, *)	and SLOT
(*, inform, *)	SLOT is VALUE
(null, negate, *)	Wrong format

Table 4.2: Example of act-level grammar rules providing syntactic structure

Slot	Custom value
license_plate	license plate
area	area where you want notifications
terms_and_conditions	acceptance of terms and conditions

Table 4.3: Some of the custom templates used to make generated sentences sound more natural

4.4 Crowdsourcing step

The last step required for the complete fulfillment of the M2M framework involves collecting paraphrases for the previously generated utterance templates. In the original paper, this task is assigned to Amazon Mechanical Turk workers, and the results compared with a similar dataset for the same task. In this thesis work, we instead want to explore the differences in quality for datasets collected with expert and non-expert users. This comparison gains particular relevance when the inspected task is not straightforward. While generally papers tend to consider widely known assignments such as booking movie tickets, obtaining information about a trip or finding a restaurant, some more specific tasks as in the case of the registration to

a particular service may include non-obvious steps that only become clear when the user already has a good understanding of the current context. In this case, AMT workers represent non-expert users, who can base their paraphrasing work only on a brief description of the SiirtoSoitto service. On the other hand, the users of the rule-based chatbot described in 4.2 are expert users who already possess some contextual knowledge on how the service actually works and how the SRT looks like.

4.4.1 M2M integration with rule-based chatbot

This section describes how the output of the outline generation step of the M2M framework is integrated with the rule-based chatbot in order to collect paraphrases from SiirtoSoitto users. First of all the contents of the generated dialogues have been made accessible by storing them in a document-based non-relational database. The format of each dialogue and the associated extra fields have been detailed in appendix A. The choice of a document-based database seemed to fit the varying-length nature of the content of the dialogues.

New routines have been added on the rule-based chatbot server to handle the interaction between the chatbot and the users in this new experimental phase. After a user registers he is asked whether he'd like to help improving the chatbot by providing rewrites for computer-generated dialogues. In affirmative case, a random dialogue gets sampled from the database and assigned to the user, and the first utterance template is presented to him. After each rewrite is sent back, it is stored in the corresponding dialogue document and the next template utterance is sent to the user. This routine goes on until the dialogue has been completely paraphrased. At this point, the user is prompted with a couple of follow-up questions regarding the perceived length of the task, his feeling of frustration, and whether the rewritten dialogues resemble clearly a possible conversation between him and an assistant. An optional comment is also collected to gain further insights about the whole process. Appendix A.3 shows some glimpses of how the crowdsourcing task looks from the user perspective.

4.4.2 M2M integration with Amazon Mechanical Turk

A single page HTML form (appendix A.1) has been designed to perform the same paraphrasing task on Amazon Mechanical Turk. First, a brief header introduces the worker to the goal of the HIT and provide some basic context about Siirtosoitto. Then, a short example provides a practical overview of

the task followed by a table where each row contains a message template with a field to type the contextual paraphrase. Since AMT requires CSV formatted documents to contain the input values for each HIT instance, a small Python script has been written to map the contents of each dialogue in the database to the required scheme. Finally, at the bottom of the page, the same questions shown to the chatbot users are asked to the Turkers as well as the possibility to leave an optional comment to provide feedback about the HIT.

Chapter 5

Results

In this chapter, we are going through the results of the analysis of the data collected with the M2M framework. The first sections describe how users of the SiirtoSoitto service and Amazon Turkers have been enrolled to perform the experimental task described in the previous paragraph. Then, a quantitative analysis will compare the dialogue diversity and language richness between the data collected during the crowdsourcing phase both from the AMT online platform and trough the Siirtosoitto chatbot integration. In addition, an investigation on the users' feedback, collected at the end of the experiments, allows to gain some additional insights and perspective. Finally, a subjective human evaluation will assess the overall quality of the collected annotated dialogues.

5.1 User testing

Once the implementation and integration of the M2M framework were ready, a large number of dialogue skeletons have been generated with the framework and used for crowdsourcing experiments with users from AMT and SiirtoSoitto. Both groups have been tasked with dialogue templates coming from the same machine-generated dataset in order to additionally increase the comparability of the resulting paraphrases.

5.1.1 SiirtoSoitto users

The new conversational routines described in 4.4.1 have been made available publicly to the users of SiirtoSoitto chatbot for a period of approximately one week. At the end of the registration phase, all the new users were asked to contribute to the crowdsourcing of dialogue rewrites. Unfortunately, there is no demographic information on how the average user of the platform

looks like, e.g its age group, because this kind of information is not already part of Siirtosoitto user profiles. Asking those kind of information during the rewrite phase seemed unnecessary and could have dissuaded some users from participating due to the disclosure of personal details. A total of **133** participants showed interest in the task but only **83** actually completed the dialogue rewriting.

5.1.2 Amazon Turkers

Experiments with Amazon Turkers have been splits into several small batches. Initially, it was mainly due to the unfamiliarity with the platform in question, but later, since a significant manual checking work became necessary due to the unreliability of the workers, splitting the job into batches of 15-25 dialogues each helped to keep a manageable work.

AMT batches					
	Batch 1	Batch 2	Batch 3	Batch 4	Batch 5
Accepted HITs	10	20	20	20	35
Rejected HITs	3	8	7	6	5
Total HITs	13	28	27	26	40
Total dialogues	10	20	20	20	35
% rejected	23.1	28.6	25.9	23.1	12.5

Table 5.1: Overview of AMT crowdsourcing results

5.1 shows how a large number of Turkers task outcomes had to be consistently rejected resulting in a thorough and lengthy validation process. The rejected tasks were either a result of individuals trying to make HITs filled with random values slip through large result batches or users failing to comply with the provided instructions, e.g. copy-pasting the templates or coming up with semantically unrelated rewrites. There are several ways to cope with the verification of crowdsourced data. The known-answer technique consists in adding to the task some questions for which the answer is already known. By matching the outcomes with the already known ground truth all the HITs that fails this validation step can be automatically discarded. A second approach involves assigning the same HIT to multiple Turkers and automatically validating the results based on an agreement

level. Unfortunately, these validation methods were unsuitable for this specific task since the act of paraphrasing itself involves the presence of diverse results, even for the same task. An alternative option could have been using a second crowdsourcing step in order to validate the compliance of the paraphrases with the task instructions and correspondence between the semantic meaning of the proposed sentences and the original templates. This could have probably been the go-to solution but would have made the process more expensive. Given the limited amount of data handled for this work, manual checking was considered an acceptable solution.

Similarly to what happened for the SiirtoSoitto users, no demographic information is available about AMT workers. Amazon allows to assign HITs based on numerous parameters such as age group, gender, nationality, level of instruction etc. but given the lack of information about the other group this additional filtering was deemed unnecessary.

5.2 Dialogue diversity

One of the benefits brought by the employment of the M2M approach is an improved dialogue path coverage alongside with increased variety of language, presence of co-references, and other language tricks. We additionally wanted to investigate whether the presence of expert users would bring benefits to the quality of data coming from the crowdsourcing step. One dataset containing 98 dialogues has been collected from Amazon Mechanical Turk workers, which are professional task completioners, but yet they lack knowledge and context about the system around which the task is built on. They thus represent those non-expert users, which are often specifically hired to complete a crowdsourcing task in an NLP-related scenario. The other dataset, containing 83 dialogues, has been collected from real users of SirtoSoitto which accessed the crowdsourcing phase through the baseline rule-based chatbot deployed publicly. Once again, we remind how both expert and non-expert users have paraphrased the same set of dialogues generated in the self-play step of M2M to further increase the comparability of data.

The contextual rewrites for each turn have been extracted from the dialogues of both datasets. Each sentence has been split into tokens and the average number of tokens per turn has been computed. The AMT dataset has obviously a higher overall count of tokens because it comprises a greater amount of dialogues. Interestingly, even with a lower dialogue count, the

SiirtoSoitto dataset presents a larger amount of unique token appearances. This is the first indicator of a greater language variety associated with expert user rewrites. Accordingly, the coefficients of lexical richness, computed as the ratio between unique tokens and total tokens, are higher for the SiirtoSoitto data.

Metric	**AMT**	**Siirtosoitto**
Dialogues	98	83
Total turns	898	718
Total tokens	7723	5069
Avg. token per turn	8.60	7.06
Unique tokens	800	**816**
Unique tokens/Total tokens (Lexical richness)	0.104	**0.161**

Table 5.2: Comparison between AMT and Siirtosoitto on diversity of language

5.2.1 N-gram analysis

In the field of computational linguistics, n-grams are sequences of n consecutive items from a sample of text. In this case, the items are the tokens, words, and other punctuation symbols extracted from each rewrite. For each turn of every dialogue, we computed all the n-grams associated with the corresponding contextual rewrite. An n-gram of size 1 is referred to as *unigram*, size 2 as *bigram* and size 3 as *trigram*. (fig. 5.1) In this work, we focused on the analysis of bigrams and trigrams while the unigrams analysis simply coincides with the token metrics presented in the previous paragraph.

```
"I do accept the terms and conditions"

"I do" "do accept" "accept the" "the terms" "terms and"
"and conditions"
```

Figure 5.1: An example of bigram (n=2) extracted from a user paraphrase

Both bigrams and trigrams have been extracted from each contextual rewrite

using ntlk[1], a popular Python library for NLP. Duplicates have been removed from the results in order to handle only unique n-grams. Following the suggestion coming from [6], two additional metrics have been computed as the ratio between n-grams and the number of tokens.

Metric	**AMT**	**Siirtosoitto**
Unique bigrams / Total tokens	0.103	**0.122**
Unique trigrams / Total tokens	0.28	**0.387**
Unique rewrites / rewrites (Outline diversity)	0.885	0.837

Table 5.3: N-gram comparison between AMT and Siirtosoitto

5.2.2 TF-IDF diversity

Each rewrite R have been scored using the sum of TF-IDF scores of n-grams. TF-IDF reflects the importance of an n-gram. n-grams with lower frequency in the collected data have higher IDFs. Thus the Tdiv metric denotes the extent of diversity of an expression in the dataset. The TF-IDF and TF-IDF Diversity (Tdiv) [53] metrics are defined as:

$$\mathrm{tf_{x,d}} = \frac{n_{x,d}}{|\mathrm{N}_d|}$$

$$\mathrm{idf_{x,D}} = \log \frac{|D|}{|\{d : x \in d\}|}$$

$$\text{TF-IDF(x,d,D)} = tf \times idf$$

$$Tdiv(R) = \sum_{n=1}^{N} \frac{\sum_{\text{n-gram}\in R} \text{TF-IDF(n-gram)}}{V_n}$$

$$V_n = \frac{1}{|D|} \sum_{R\in D} \sum_{\text{n-gram}\in R} \text{TF-IDF(n-gram)}$$

Term frequency-inverse document frequency (TF-IDF) is the function that measures the importance of a term with respect to a document or a collection

[1]Natural Language Toolkit, URL: https://www.nltk.org

of documents. The TF-IDF value of a term grows proportionally with its frequency inside the document but with the inverse of the frequency of the term inside the collection. TF-IDF is composed of two factors that account for the previously described properties, tf and idf. The term frequency tf of a certain n-gram x is computed as the ratio between the number of appearances of that n-gram inside a document d and the number of n-grams N_d of that document. In this case, each rewrite is considered as a document. The inverse document frequency idf factor indicates the general importance of an element in the collection. It is formulated as the ratio between the total number of documents $|D|$ (the rewrites in the dataset) and the number of documents in which the n-gram x appears. V_n in the Tdiv formula is a normalization term. Using N = 3 we considered for the computation unigrams, bigrams, and trigrams coherently with the analysis performed in 5.2.1.

The Tdiv score for a sentence by itself has little meaning, it needs to be compared with Tdiv scores of sentences that entail the same semantic content. Similarly to what has been done by the authors of the formula, which have compared various machine-generated captions for a picture extracted from an image dataset, we compared only sentences based on the same underlying utterance template. Given two rewrites for the same turn r_i^{AMT} and r_i^{Sirto}, one from the AMT dataset and one from Siirtosoitto, if r_i^{Sirto} has higher $Tdiv$ score it generally has richer expressions than r_i^{AMT} The diversity scores have been computed by first merging together the two datasets. Then, for each available pair of rewrites of the same turn, the scores have been computed using the above formula and compared. Even if both datasets comprise rewrite of the very same dialogues, not all the turns in the bigger AMT dataset had always a corresponding paraphrase in the SiirtoSoitto one. This is due to the smaller number of dialogues rewritten with the real users and the fact that sometimes they didn't completely finish the paraphrasing task, leaving some rewrites blank. For each pair of corresponding sentences, we kept track of the one with the highest $Tdiv$ score. The results show how SiirtoSoitto wins the comparison almost 2 out of 3 times thus having overall turn rewrites with richer expressions.

Figure 5.2 shows some examples from the scoring script that evaluated the rewrites in the combined dataset.

Metric	AMT	Siirtosoitto
Tdiv comparison	155	**270**

Table 5.4: Tdiv comparison between AMT and Siirtosoitto. Given two rewrites of the same turn a point is given to the one with haighest score

```
Dataset Tdiv    Rewrite
Siirto  (4.16)  oops there is something wrong with the
                licence plate format could you please write
                that again
AMT     (3.55)  Sorry. The format is wrong. provide License
                plate again

Siirto  (4.47)  good to what number should we send
                notifications
AMT     (2.51)  And phone number?

Siirto  (2.79)  my phone number is 049925805 and i want to
                get notified for marjaniemenranta i accept
                the terms and conditions
AMT     (2.97)  My phone number is 049925805 and I have
                accepted the terms and conditions. I'd like
                to be notified in Marjaniemenranta
```

Figure 5.2: Some sentence pairs with their corresponding Tdiv scores

5.2.3 Jaccard similarity

The analysis of the quality of the data collected with the M2M framework proceeded with the computation of the Jaccard similarity coefficient. The Jaccard similarity coefficient measures the similarity between two finite sets of elements, in this thesis the words, or tokens, contained inside a particular sentence. More specifically, this coefficient has been used to compute the word similarity between the user rewrites and the proposed templates. Jaccard similarity coefficient is computed as the ratio of the shared words between two sentences and the total number of unique words.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

In this work, we actually care of making sure that the proposed template generated by the self-play step of M2M and the rewrite proposed in the crowdsourcing phase are as much different as possible. Therefore we actually compute the Jaccard distance, defined as $1 - J(A, B)$ to measure how dissimilar the two sentences are. Jaccard distance has been chosen as an evaluation metric because there is no interest in evaluating similarity at a semantic level (which is assumed to come from crowdsource workers) but rather evaluate how much effort the users put into phrasing rewrites with different wordings from the original.

Metric	**AMT**	**Siirtosoitto**
Average Jaccard distance	0.432	**0.490**

Table 5.5: Jaccard distance comparison between AMT and Siirtosoitto. A higher score means a more diverse wording compared to the original templates

Results show that, on average, template rewrites coming from real SiirtoSoitto users present a larger rephrasing effort. Having dialogue rewrites which are dissimilar from the original computer-generated sentence is important for the overall quality of the dataset since the flexibility and robustness of any neural model trained on those data largely depends on the diversity and variety of sentences.

5.3 User feedback

As part of the crowdsourcing step entrusted to Amazon Turkers and users of the SiirtoSoitto rule-based chatbot, we asked to provide feedback on a number of additional questions. The goal of this inquiry is earning some more insights about the crowdsourcing process. In particular, we were interested in understanding the user's perception about the duration and the frustration derived from the task completion.
Even if these questions have been posed not to make comparisons, but to better understand what the users thought of the rewriting task, we still computed average metrics between the two datasets as shown in table 5.7.

With respect to Q1, AMT workers almost unanimously confirmed that the proposed machine-generated sentences and their corresponding paraphrases led to a plausible user-system dialogue. SiirtoSoitto users are also very positive about the final outcome of their task with a reasonable 81% of sat-

ID	Question
Q1	Does it seem like a conversation between a user that sounds like you and a chatbot that sounds formal?
Q2	On a scale from 1 (not long) to 7 (excessively long), how long do you think the task was?
Q3	On a scale from 1 (not at all) to 7 (a lot), did you feel frustrated while doing the task?
Q4	Optional comment about the experiment

Table 5.6: The feedback questionnaire proposed to crowdsource users

isfaction. The AMT number looks a bit too optimistic especially compared to the feedback obtained from the chatbot users. The hypothesis is that, since each HIT completed by AMT workers is a paid task that must undergo an approval process, there is a bias toward a positive answer.

From Q2 we obtained an average perceived duration of task respectively of 3.37 from Turkers and 3.82 from chatbot users. The values have a comparable magnitude but a t-test performed on the data highlighted a slightly statistically significant difference between the values coming from the two groups ($t(179) = -2.03, p = .043$). In any case, this feedback does not reflect by any means the real duration of the paraphrasing practice.
The perceived frustration levels derived from Q3 also present intermediate values. The results from the two datasets are absolutely comparable and do not indicate any statistically significant difference. Overall, the feedback obtained both from Q2 and Q3 is encouraging and indicates that the time spent to complete the paraphrasing of one dialogue is not excessively long. The average level of frustration also seems to indicate that the tasks do not present any particular burden or overwhelming difficult but we further investigate these aspects by analyzing the optional comments left by the users.

Q4 refers to an optional comment that users could leave to point out some additional relevant aspects which are not covered in the previous questions. As expected, Amazon Turkers had not much to say, their few comments mainly regarded the excessive length of the HIT. Only two users complained about the fact that some statements are difficult to paraphrase. On the SiirtoSoitto users side, the comments are much more diverse and interesting.

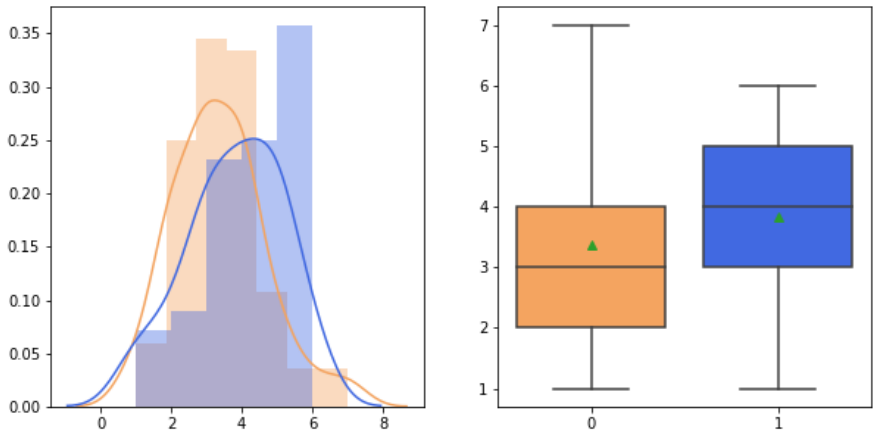


Figure 5.3: Distribution (left) and boxplot (right) of the results of Q2 for AMT (orange) and SiirtoSoitto (blue).

ID	Metric	AMT	SiirtSoitto
Q1	% of positive answers	96.94%	80.72%
Q2	range 1-7	3.37	3.82
Q3	range 1-7	3.06	3.11

Table 5.7: Results from feedback analysis

A few argue that the instructions and the purpose of the task are hard to understand. Some other users say that they needed a few messages to catch up with the task. During the set up of the experiment, we tried to carefully find the most appropriate way to introduce the users to the goal of the rewriting process by trying to provide them with a bit of context about what they were doing and why and how they were supposed to carry out their paraphrasing job. Balancing the expressiveness of the instructions while at the same time trying not to overwhelm the users with too much information and complex explanation has proven to be a challenging effort. The user's feedback suggests that there is still some work to be done in order to formulate meaningful but concise instructions. A correct understanding of the task is of paramount importance for the successful collection of high-quality data and the reduction of additional content checking.

Source	Comment
AMT	Some statements are difficult to paraphrase.
AMT	I think its a good and understandable task!
AMT	too long to do
SiirtoSoitto	hard to understand instructions and purpose
SiirtoSoitto	would need to understand better what the chatbot is meant to do
SiirtoSoitto	i didn't quite understand what i was supposed to do
SiirtoSoitto	i did not understand at first that i needed to be user and chatbot thats why it doesn't sound like real conversation at start then i realised what i should done"

Table 5.8: Results from feedback analysis

5.4 Qualitative evaluation

In section 5.2 we evaluated the properties of the data collected using M2M from the perspective of dialogue diversity and language richness. The outcome showed how generally M2M is more effective if its crowdsourcing step is performed with the efforts of expert users, that are, real users of the system in consideration. Similarly, we want to also assess the perceived quality of the generate and paraphrased dialogues from a human standpoint. Even if some automatic metrics about word richness and diversity look encouraging they do no say anything about whether the dialogue themselves are realistic and appreciable from a human perspective. For example, one could reach marvelous scores on these metrics while actually writing non-sense gibberish. Thus the importance of accompanying the previous quantitative assessment with a subjective human evaluation proving the effective quality of the data.

The evaluation of the human-perceived quality of the dataset has been carried out on 4 different dimensions that entail the overall quality of a dialogue from a human standpoint. Naturalness indicates whether the sentences do not sound artificial and robotic but rather resemble typical human expressions. It is somehow related to the first question asked to the crowdsource users as reported in section 5.3. Clearness refers to the extent to which the meaning conveyed by the dialogue turns is easily understandable. The

grammaticality score reflects the absence of misspellings or badly formatted sentences. Finally, the optimality measure is a reference of how quickly the generated dialogue and the proposed rewrites seem to go right to the point.

A group of participants (N=20) has been asked to grade the dialogues inside the two datasets on these 4 dimensions with a score ranging from 1 to 5.

	AMT	**Siirtosoitto**
Naturalness	4.05 (0.74)	4.15 (0.65)
Clearness	4.30 (0.71)	4.05 (0.80)
Grammaticality	3.85 (0.65)	4.20 (0.67)
Optimality	4.05 (0.49)	4.00 (0.63)

Table 5.9: Results of human evaluation on the dialogues collected using M2M from AMT and SiirtoSoitto. Numbers shows average scores of per dialogue grading. Standard deviation in brackets.

The results, summarised in Table 5.9 show that the human perceived quality on all dimensions of evaluation is satisfactorily high. In detail, scores for naturalness and optimality are absolutely comparable. The AMT dialogues are (statistically) more clear. This may be an effect of AMT constructs being more concise and repetitive while the other users put in more effort in recreating more varied paraphrases. With respect to grammaticality, the SiirtoSoitto dataset marks a significant increase. Once again this could be a symptom of the supposedly less care and attention put by AMT workers. Fig. 5.4 presents a side-by-side comparison of all the numbers among the two groups. Overall these results prove the fitness of the M2M framework in generating and produce high-quality annotated data for NLP tasks.

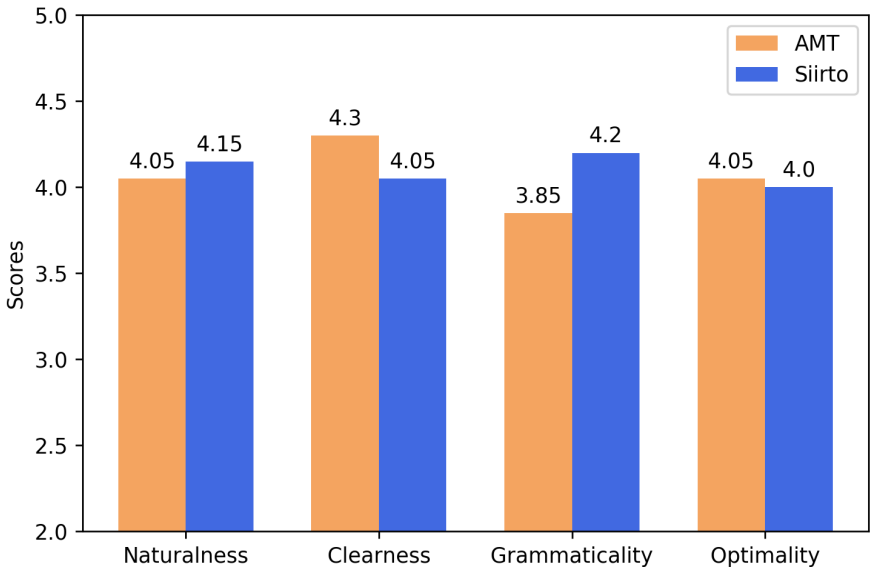


Figure 5.4: Graphical comparison between AMT (orange) and SiirtoSoitto (blue) subjective scores.

Chapter 6

Proof-of-Concept

As stated several times during the writing of this work, the annotated dialogues collected through the M2M framework will eventually serve the training of neural dialogue models. Even if this was not part of the main scope of the project, a Natural Language Generator NGL neural model has been implemented and trained to showcase how the data collected with the framework can be effectively used in practice. Though it does not represent a complete end-to-end neural dialogue system, this NGL module works as a proof-of-concept of the goodness of the data collected with M2M. In the next subsection a general overview of a Transformer, the deep learning model used to implement the NGL module, will be presented. Successively, we will show how we tweaked its original architecture to make use of the properties of the data collected with M2M to their maximum extent. Then we present the format in which the input data have been pre-processed for the training and prediction phase. Finally, we discuss and evaluate the outcomes of the trained models.

6.1 Transformer

A transformer is a deep learning model that currently represents the state-of-art for statistical machine translation. It was first proposed in 2017 in the notorious paper "Attention is all you need".[54] As all of the latest models for varying-length sequence modelling it leverages an encoder module to encapsulate the meaning of the input sequence in a fixed-length dimensional space and a decoder that is an auto-regressive module that generates an output sequence of symbols based on the context of the encoded input. The main difference between the transformer and other relevant models such as Seq2Seq and ConvS2S [55] is the total absence of any RNN or convolutional

layer in favor of a fully attention-based solution.

Attention is a mechanism used to model long term dependencies inside sequences of symbols. In its first applications, attention was used in Seq2Seq as a way to use encoded representations of varying lengths instead of just having the last hidden state of the encoder. During the decoding phase at each step, the decoder chooses, using a weighted sum, which encoded vector z use. In the transformer model a mechanism called self-attention is also used to substitute the RNN performing the encoding step and converting input sequence $(x_1, ..., x_n)$ into continuous representations $(z_1, ..., z_n)$. Any attention function can be seen as mapping a query and a set of key-value pairs to an output. The output is computed as a weighted sum of the values, where the weight assigned to each value is measured by a compatibility function of the query with the corresponding key. We can visualize self-attention as a matrix where the words of a sentence form the columns, and the same words form the rows. (fig. 6.1) The values in the matrix indicate how some parts of that sentence relate to others, for example, the pronouns with their antecedents.

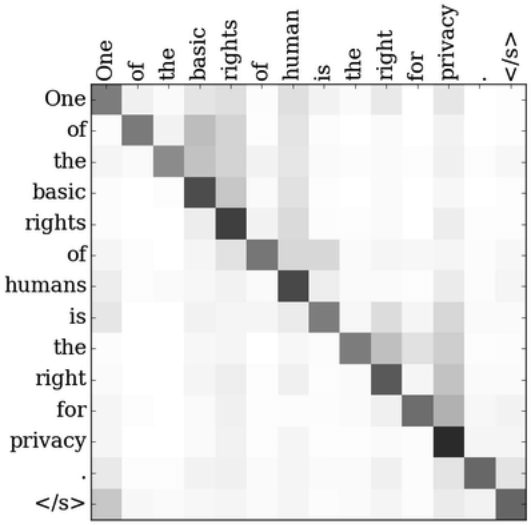


Figure 6.1: Visualization of self-attention

The particular attention formula used in the Transformer model is called "Scaled Dot-Product Attention" and can be computed as:

$$z_i = \sum_{j=1}^{n} \alpha_{ij} v_j$$

$$\alpha_{ij} = \frac{exp(k_j^T q_i/\sqrt{d_k})}{\sum_{j^i=1}^{n} exp(k_j^T q_i/\sqrt{d_k})}$$

or in matrix notation:

$$\text{Attention}(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where Q, K, V are the packed sets of queries, keys and values. The authors also found beneficial to apply what they called $multi-head\ attention$ which consists in linearly projecting the queries, keys and values in h multiple different smaller vector spaces, computing the scaled dot product attention, then concatenating the outputs and project them back in the original dimensional space.

$$\text{MultiHead}(Q, K, V) = (head_1 \ \oplus head_2 \ ... \ \oplus \ head_h)W^O$$

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_{model} \times dk}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v} and\ W^O \in \mathbb{R}^{h_{dv} \times dmodel}$$

This attention mechanism is used in each of the 6 stacked blocks that made up the encoder module. Inside each block, after self-attention, the representation in each position is processed with a small multi-layer perceptron (MLP). The encoder is also a stack of 6 identical blocks; self-attention is used at first to keep track of all the symbols already generated in the output sequence, and a masking trick is used to avoid attending not yet generated tokens during training. The outputs of the masked self-attention layer are then used as the values for the following attention block, where queries and keys are instead the encoded input sequences. Through the whole model (fig. 6.2) some standard deep learning tricks are used for regularization, such as skip connections and layer normalization. The model was first proposed as an advancement in statistical machine translation but can be easily adapted for several NLP tasks such as text summarization and response generation as we did in this work. Additionally, a hard-coded positional encoding is added to infuse the significance of different ordering in words.

6.2 Updated architecture

One of the unique characteristics of the M2M framework that we have implemented is that the paraphrases of the templates of each dialogue are completely performed by the same person. It means that, given the personality

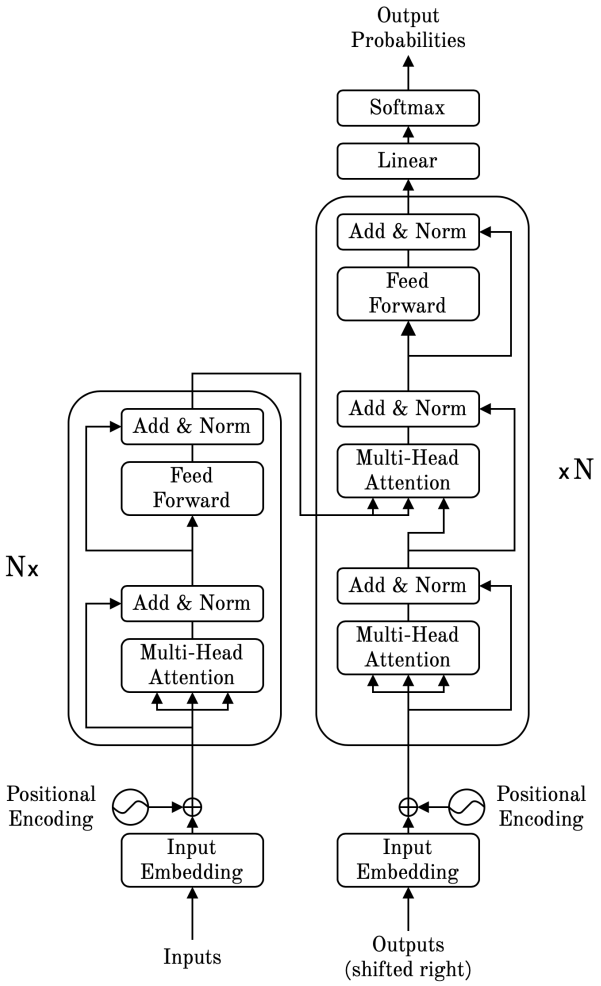


Figure 6.2: Original architecture of the transformer model

profile of a particular user, its consequent traits in conversational behaviour are reflected in the system answers which are still written by that person. To make it more clear, imagine that the paraphrasing task is being carried out by a particularly extrovert and expressive person. His personality, which is reflected in the way the user writes, will also appear in the paraphrases of the system turns because the user will rewrite the system replies in the way he would expect to be answered in its idealised conversation. This kind of contextual information could not be elicited with other annotation methods such as Wizard of Oz (because each of the two actors will maintain their personality in the way they write) or for example in the manual annotation of transcripts of online conversations (for similar reasons). Having a complete dialogue with the turn-taking of both actors being written down by

the same person is the only way to encapsulate in the system's responses the contextual information of the user writing style. If someone has a short and dry writing habit such markers will also appear in their systems turns rewrites. Users who tend to express more clearly their thankfulness or disappointment will similarly echo this tendency in both actors turns.

Given this premises, it seemed valuable to try to generate system utterances not only based on the dialogue act annotations that lie at the bottom of each template rewrite but also on the previous user turn utterance, with the goal of obtaining contextually different sentences based on the user's tone. In this PoC NGL module, we can imagine the DA sequence as the output of the dialogue manager of a frame-based chatbot system, similar to the one described in 2.2, that goes to define the structure and content of the generated utterance. The user utterance is instead used to encapsulate the tone of the last user message.

At this point, considering the significant difference of meaning carried out of the two inputs, it seemed clear that it was not wise to encode both of them together using a single encoder module. We decided to encapsulate the meaning of these inputs in two different vector spaces using two separate encoders in order to preserve their logical and semantic difference (fig. 6.3). It also makes sense to encode what we would like to call the user's *current typing persona* z^u in a much smaller dimension space compared to the DAs sequence z^a. The outputs of the two encoders are then concatenated on their first dimension to be fed to the decoder block. Similarly, the transposed of the Boolean masks computed to be able to process the data in mini-batches are also combined to make sure that the decoder knows what elements are relevant and what have been added just as a padding.

$$z_i^u = \sum_{j=1}^{n} \frac{exp(u_j^T u_i / \sqrt{d_u})}{\sum_{j^i=1}^{n} exp(u_j^T u_i / \sqrt{d_u})} u_j$$

$$z_i^a = \sum_{j=1}^{n} \frac{exp(a_j^T a_i / \sqrt{d_a})}{\sum_{j^i=1}^{n} exp(a_j^T a_i / \sqrt{d_a})} a_j$$

$$z_i = z_i^u \oplus z_i^a$$

$$y_i = f(y_{i-1}, ..., y_1, z_1, ..., z_n)$$

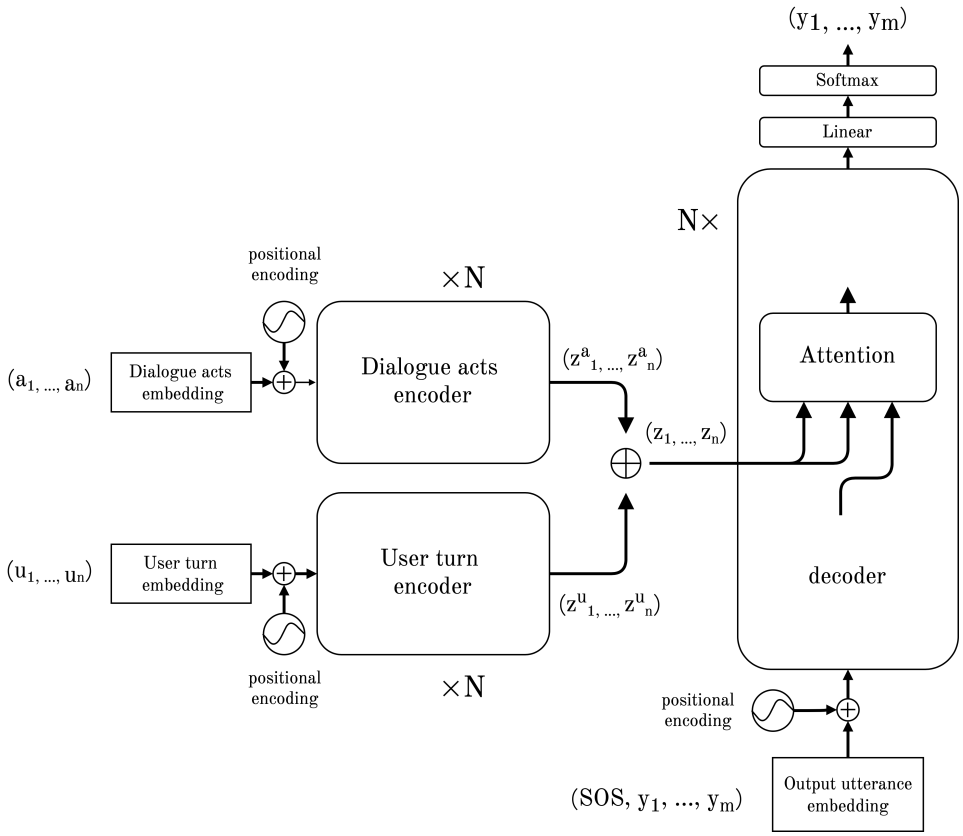


Figure 6.3: The updated architecture of the transformer with two different encoder modules

6.3 Data pre-processing

The transformer processes input sequences of varying length and outputs at each time-step a distribution over all possible symbols in the output dictionary. The set of symbols contained in the training output sequences form the output dictionary. Let's first consider the first of the two inputs, the sequence of dialogue act objects. As the reader will remember each dialogue act a_i has the form:

$$a_i := \begin{Bmatrix} act & x_{i1} \\ slot & x_{i2} \\ value & x_{i3} \end{Bmatrix}$$

The object structure got transformed into a sequence of symbols using the following notation. Brackets and colons have been stripped away and the

structure had been flattened.

$$a_i := \ 'act\ x_{i1}\ slot\ x_{i2}\ value\ x_{i3}\ '$$

It is important for the new representation to keep entailing the meaning of the original object, that's why we decided to keep the apparently redundant symbols, *act*, *slot* and *value*, which always appear in all dialogue acts. We saw that by maintaining this regularity between such terms and their varying values, the transformer was able to retain the information coming from each dialogue act. Here is an example of how a turn annotation made of two dialogue acts would get preprocessed before being used in the model.

$$\left\{\begin{matrix} act & \text{affirm} \\ slot & \text{none} \\ value & \text{none} \end{matrix}\right\}, \left\{\begin{matrix} act & \text{request} \\ slot & \text{phone} \\ value & \text{none} \end{matrix}\right\}$$

$$\Downarrow$$

$$'\ act\ \text{affirm}\ slot\ \text{none}\ value\ \text{none}\ act\ \text{request}\ slot\ \text{phone}\ value\ \text{none}\ '$$

Both annotations and previous turn user utterance are now represented as a sequence of symbols. They are then further processed by forcing lowercase characters so that the same words but with a capital initial are not misinterpreted as different. Punctuation is stripped away from user turns and sequences that exceed a pre-defined maximum length are removed. In order to facilitate the manipulation of data, the input symbols are embedded into one-dimensional vectors and the correspondence between those indexes and the symbols is kept by building a word-dictionary that facilitates the conversion in both ways.

The same pre-processing steps are also applied to the output system sentences. Eventually, the data used to train the transformer model are made of triplets containing: the previous turn user utterance u, the current turn annotations a, and the corresponding system reply s.

$$\langle u, a, s \rangle$$

6.4 Contextual responsiveness

We trained both a standard Transformer, using only the sequence of turn annotations, and the model described in 6.2 on the joint datasets (with an 80-20 training-test split) collected during the experiments with AMT workers and SiirtoSoitto chatbot users. The intent of training an NGL model

was twofold. On one side we were trying to prove that the collected data and its corresponding annotations had the right qualities to be used to train deep learning dialogue systems. On the other hand, we tried to capitalize on an additional property of the M2M collected datasets fostering the model to contextually adapt its answers to the user's tone.

Fig. 6.4 shows some results coming from the vanilla Transformer when used to generate natural utterances from semantic frames made of dialogue act sequences.

```
> act notify success slot none value none  act inform slot
registration value true  EOS
= thank you . your registration is finished . EOS
< thank you your registration is completed successfully EOS

> act request slot phone number value none  EOS
= please provide phone number EOS
< please provide phone number . EOS
```

Figure 6.4: Examples of some test set sentences generated with the model

To evaluate the goodness of the generated sentences we chose not to rely on one of the popular automatic scoring metrics such as BLEU. This metric relies on the word similarity between a human-generated translation (in our case a paraphrase coming from one of the users) and the model generated sentence. In the field of machine-translation and text generation there is a huge debate whether this kind of metrics effectively correlates with the human judgement about the quality of a "translation".[56] In principle, there are multiple evaluation dimensions for a sentence, scores about utility, or semantic correspondence (how much the sentence matches the meaning of the reference), likability (which words have been chosen to carry the message), readability, and grammaticality. A metric is of interest if it can approximate and predict these different aspects of human language. Ultimately, we care about whether an NGL system produces high-quality texts, not that it has an exact word correspondence with its original counterpart. The BLEU score presents a systemic problem since it is based on making comparisons with a reference sentence: in real life, sentences can be translated in many different ways, sometimes with no overlap. Therefore, if we were just to compare how much the sentences generated with the NGL model differ from

their true counterparts we would definitely not be evaluating the quality of the text from a semantic perspective. Hence, the evaluation of the outputs of the model has been carried out by subjectively comparing the semantic meaning of the generated and true utterances. We especially care not that the two are exactly the same but that, even with different wordings, the meaning is comparable. The before-mentioned problem of the disconnection between similarity-based metrics and the true essence of a sentence is even more remarked in our case study. Our input is in fact, not a natural language sentence in the traditional way but it is obtained by processing sequences of object-like structures, the dialogue acts. It is not uncommon then, that the dataset contains multiple dialogue's turns with the same sequence of DAs (e.g. notifying a successful registration) and that the crowdsource workers provided paraphrases with different wordings. Observing some of the results presented in Fig. 6.4 we can see how the sentences generated by the transformer nicely entail the semantic meaning encoded in the input sequence of DAs.

We then trained the updated Transformer, which uses an additional separate encoder to encapsulate the user's previous dialogue turn, on the very same joint dataset.

```
> act greeting slot none value none  act propose slot
registration value none  EOS
> hi ! EOS
= greetings! would you like to register to siirtosoitto ? EOS
< hello ! would you like to register to this service ? EOS

> act greeting slot none value none  act propose slot
registration value none  EOS
> hi EOS
= hi can i register to siirtosoitto EOS
< hello would you like to register to siirtosoitto EOS
```

Figure 6.5: Examples of some test set sentences generated with the updated architecture showing how the same dialogue act sequence is materialized with different expressions based on the user's attitude

Each sample in figure 6.5 shows four distinct elements; the first two lines correspond to the system inputs, the sequence of dialogue acts to translate and the previous user turn. The third line indicates the corresponding

rewrite gathered from the crowdsourcing. Finally, the fourth line shows the output of the NGL model. The examples show that now, even for the same dialogue act sequence input, the outputs, being also conditioned on the contents of the previous user turns, are different. Since in our training samples each ¡user turn - system reply¿ pair is written by the same person the model tries to learn how to build, given the structure dictated by the DAs, an utterance with the wording that the user expects. The first user approaches the chatbot with seemingly more excitement by saying "*hi*! and the same tone is also reflected in the system reply that the same user rewrote; accordingly, the model output also follows the same open and friendly mood by means of a warm greeting. On the other side, the participant from the second sample approaches the system with a neutral opening and the chatbot contextually adapts its reply to be more formal.

Chapter 7

Discussion and Conclusions

In this conclusive chapter, we analyse and discuss the outcome of this thesis work. The first section is dedicated to restating what were the main objectives and the rationale behind the whole project. In this section, we also answer all the research questions stated in the introduction, in addition to some a posteriori comments. The next section is dedicated to discussing and speculating about the possible implications and applications of the work with respect to data collection for dialogue systems and chatbot development. Then, we move on to the analysis of the main limitations of the whole work and reflect on what could be done in the future to further improve the project. Finally, as a wrap-up, we summarize the major phases of the project and its primary accomplishments.

7.1 Answering the research questions

As stated in the introduction the main objective of this work rises from the current difficulties in procuring high-quality annotated data for the training of neural-based dialogue systems. Some of the main obstacles concern the data greediness of the newest algorithms resulting in dilated development times for data collection, and excessive costs. It is also difficult to monitor the quality of data annotations that for dialogue datasets span over multiple levels (semantic, contents, structure). The lack of already available data for task-specific and service-specific use cases further dissuade developers and companies to leverage neural-based solutions for chatbot architectures. Currently, the most diffused architecture for dialogue systems leverages a combination of modules making use of machine learning or pattern matching algorithms to fill a semantic frame that drives the conversation. This approach, while guarantying a safe task-completion rate, lacks the flexibility

typical of human dialogues and language and it's often linked to an absence of chatbot personality.

We explored and experimented the applicability of a recently proposed framework called Machines talking to Machines that completely flips the classical data collection and annotation workflow. Instead of gathering dialogues, either from pre-existing sources or with other methods, and subsequently annotating them, M2M starts by automatically generating annotations and conversation skeletons and successively uses crowdsourcing to add on top the natural language utterances typical of human dialogue.

The original paper proposing the M2M framework showcased its use only on a particular set of recurrent tasks defined as database querying tasks. Moreover, the experiments were carried out in a rigidly regulated context. The research question **R1** targeted the effective applicability of the M2M approach to other classes of tasks. We tackled this question by selecting a new kind of task, that we named Service Registration Task, which models the frequent act of registering to a service by providing and validating information. Experimentation has been carried out with a specific instance of this task built around the use-case provided by the company where I've been doing my internship. Specifically, we have been working on the user registration procedure of SiirtoSoitto, an online service sponsored by the administration of the city of Helsinki used to inform its residents about roadworks and imminent car towings. We were able to successfully implement the core components of the framework, such as the user and system simulators, by following the high-level description available from the paper and adapted them to the new class of tasks. We successively used the implemented framework to generate dialogue templates and integrate them in the last step, the crowdsourcing of natural language paraphrases, both with AMT and a rule-based chatbot made available to the public. In the end, we were successfully able to collect for the SRT two annotated datasets whose quality was confirmed both by quantitative and qualitative inspections. In particular, when asked about the outcomes, a combined average of 89.5% of the users confirmed that the dialogues sounded realistic. Some other external participants contributed to a human evaluation aimed at judging the resulting dialogues from a number of relevant conversational dimensions. The results hinted that the whole process was able to satisfy all expectations resulting in an average grading of more than 4 out of 5 on most features such as naturalness and clearness.

Since all the most widespread data collection and annotation methods make use of crowdsourcing through external platforms to speed up the process, with research question **R2**, we were interested in evaluating the impact of expert and non-expert users on the contextual rewrites coming from the final step. In our view, this element is especially important whenever the particular use case or task instance is not completely straightforward. Having to rewrite templates about an unknown service with very little context could have been in our eyes a key element for differences in data quality that developers should not oversee. With the second research question we specifically intended to investigate the difference of the datasets collected using two different sources. On one side the professional AMT workers that lack of contextual knowledge and on the other side users that are already part of the service and familiar with its mechanics and purpose. From a human-level perspective, dialogue from both groups showed a similar level of goodness. In particular, after asking external participants to rate the conversations on various defining characteristics, such as clearness and grammatically, we obtained largely satisfying scores on both datasets. On the other hand, with regard to dialogue diversity and linguistic richness, which is key for the flexibility and robustness of any neural model built with these data, there were some evident differences. The dataset collected with the help of user from the SiirtSoitto system showed consistently richer linguistic features overtaking AMT data both on the count of unique words and on the variety at sentence's structure level. Here follows a table summarizing all the key metrics that have been computed:

The research question **R3** is about the possible integration of the M2M framework within a rule-based dialogue system with the ultimate goal of bootstrapping a new neural-based model using the crowdsourced data. The rationale behind this question is the following. Quickly building a rule base dialogues system with the intent of getting straight to the goal of a certain task is possible. The down sight is that these chatbots ultimately fail in one of the primary goals behind this technology which is not only to be present on a different communication channel but also to provide a new way to interact by mimicking a realistic human conversation. Neural-based models can overcome the limitation of rule-based systems, especially for task-oriented situations but require a lot of data. The integration of M2M into a rule-based system would enable, over time, the transition to a more flexible and robust neural-based model with limited effort. In the end, we feel that we answered the last research question partially. On one side we showed how the framework can be easily integrated to make chatbot users

Metric	**AMT**	**Siirtosoitto**
Dialogues	98	83
Total turns	898	718
Total tokens	7723	5069
Avg. token per turn	8.60	7.06
Unique tokens	800	**816**
Unique tokens/Total tokens (Lexical richness)	0.104	**0.161**
Unique bigrams / Total tokens	0.103	**0.122**
Unique trigrams / Total tokens	0.28	**0.387**
Unique rewrites / rewrites (Outline diversity)	0.885	0.837
Tdiv comparison	155	**270**
Avg. Jaccard distance	0.432	**0.490**

Table 7.1: Summary of the quantitative evaluation

participate in the crowdsourcing step. On the other hand, we didn't go as far as developing a fully-fledged end-to-end model with the collected data, but we successfully implemented and trained a state of the art deep learning model for natural language generation, which is one of the many components in a modular dialogue system.

7.2 Implications

With the experience gained during this thesis project, we can now suggest M2M as a viable option for companies who want to deploy a dialogue-based service while keeping a safe task completion rate and reducing the costs of data collection.

During the experimental phase we identified an additional property of the data collected with this method; the crowdsourced dialogues could be used to learn a dialogue-policy that adapts to how the user likes to converse. Once again, this is only possible because during the contextual rewrite phase a

users paraphrase both its turns and the system replies writing the sentences in the way he would like the conversation to be.

The results also suggest that, while it has not evident reflections on the human perceived quality of single dialogues, the source of the crowdsourced rewrites has a substantial impact on the diversity and richness of the dataset as a whole.

7.3 Limitations and future research

One of the main limitations of this whole project is the scale of the user-study. As mentioned a few times, in order to confidently conduct complete conversational tasks using neural-based dialogue agents, there would be the need of a substantial amount of data, which it was not possible to gather due to limitations both in time and resources. Such constraints also affected the complexity of the task the work focused on; for example in the generated dialogues, whenever the sampled scenario contains an inconsistent piece of information (e.g. a malformed car plate), once being notified by the system the only possible behaviour modelled by the user simulator is the sudden interruption of the registration task.

The use of content from the user's turn to provide contextual system responses must be also validated with a much larger dataset and with different dialogue tasks. In case of encouraging results, a future improvement would consist in trying to pre-train a task-independent user turn encoder that is able to predict the preferred style for the system response, without using any task-specific contents.

An additional consideration revolves around how M2M automatic dialogue generation works, specifically on the concept of class of tasks as a way to abstract from a specific use case and make the framework as scalable as possible. It would be convenient to conduct some field study involving companies and developers to conceive a set of critical classes of tasks that should be implemented and made available to be used with the M2M framework.

7.4 Conclusions

The main outcomes of this project are the implementation and validation of the M2M framework on a new class of tasks. We also integrated the

crowdsourcing step into a rule-based chatbot and tested it with real users highlighting how different kinds of workers have a substantial impact on the characteristics of the final datasets. Finally, we showcased the capacity and quality of the collected dialogues by implementing and training an NGL module based on state-of-art deep learning models. We also reasoned about an additional beneficial property of the datasets coming from M2M as suggested by some trials conducted with an ad-hoc NGL model based on a Transformer with modified architecture.

Bibliography

[1] Yuehua Wu et al. ≪Automatic Chatbot Knowledge Acquisition from Online Forum via Rough Set and Ensemble Learning≫. In: *2008 IFIP International Conference on Network and Parallel Computing* (2008), pp. 242–246.

[2] Jizhou Huang, Ming Zhou, and Dan Yang. ≪Extracting Chatbot Knowledge from Online Discussion Forums≫. In: *IJCAI*. 2007, pp. 423–428.

[3] Hiroshi Yamaguchi, Maxim Mozgovoy, and Anna Danielewicz-Betz. ≪A Chatbot Based On AIML Rules Extracted From Twitter Dialogues≫. In: *FedCSIS*. 2018.

[4] Jacob Devlin et al. ≪BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding≫. In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[5] Martin Schmitt et al. ≪Joint Aspect and Polarity Classification for Aspect-based Sentiment Analysis with End-to-End Neural Networks≫. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 1109–1114. DOI: 10.18653/v1/D18-1139. URL: https://www.aclweb.org/anthology/D18-1139.

[6] Pararth Shah et al. ≪Bootstrapping a Neural Conversational Agent with Dialogue Self-Play, Crowdsourcing and On-Line Reinforcement Learning≫. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*. New Orleans - Louisiana: Association for Computational Linguistics, June 2018, pp. 41–51. DOI: 10.18653/v1/N18-3006. URL: https://www.aclweb.org/anthology/N18-3006.

[7] Pararth Shah et al. ≪Building a Conversational Agent Overnight with Dialogue Self-Play≫. In: *CoRR* abs/1801.04871 (2018). arXiv: 1801.04871. URL: http://arxiv.org/abs/1801.04871.

[8] Anu Venkatesh et al. ≪On Evaluating and Comparing Open Domain Dialog Systems≫. In: *arXiv: Computation and Language* (2018).

[9] Sameera A. and Dr John. ≪Survey on Chatbot Design Techniques in Speech Conversation Systems≫. In: *International Journal of Advanced Computer Science and Applications* 6 (July 2015). DOI: 10.14569/IJACSA.2015.060712.

[10] Jack Cahn. ≪CHATBOT: Architecture, design, & development≫. In: *University of Pennsylvania School of Engineering and Applied Science Department of Computer and Information Science* (2017).

[11] Menal Dahiya. ≪A Tool of Conversation: Chatbot≫. In: *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING* 5 (May 2017), pp. 158–161.

[12] Bayan Abu Shawar and Eric Atwell. ≪Different Measurements Metrics to Evaluate a Chatbot System≫. In: *Proceedings of the Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technologies*. NAACL-HLT-Dialog '07. Rochester, New York: Association for Computational Linguistics, 2007, 89–96.

[13] Zhou Hao Yu et al. ≪Chatbot Evaluation and Database Expansion via Crowdsourcing≫. In: *Proc RE- WOCHAT Workshop of LREC*. 2016.

[14] Joseph Weizenbaum. ≪ELIZA—a Computer Program for the Study of Natural Language Communication between Man and Machine≫. In: *Commun. ACM* 9.1 (Jan. 1966), 36–45. ISSN: 0001-0782. DOI: 10.1145/365153.365168. URL: https://doi.org/10.1145/365153.365168.

[15] Kenneth Mark Colby. ≪Modeling a paranoid mind≫. In: *Behavioral and Brain Sciences* 4.4 (1981), 515–534. DOI: 10.1017/S0140525X00000030.

[16] DH KLATT. ≪REVIEW OF ARPA SPEECH UNDERSTANDING PROJECT≫. In: *The Journal of the Acoustical Society of America* 60 (Nov. 1976), S10–S10. DOI: 10.1121/1.2003088.

[17] Richard S. Wallace. ≪The Anatomy of A.L.I.C.E.≫ In: *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*. Ed. by Robert Epstein, Gary Roberts, and Grace Beber. Dordrecht: Springer Netherlands, 2009, pp. 181–210. ISBN: 978-1-4020-6710-5. DOI: 10.1007/978-1-4020-6710-5_13. URL: https://doi.org/10.1007/978-1-4020-6710-5_13.

[18] *Introducing Bots on Messenger*. URL: https://developers.facebook.com/videos/f8-2016/introducing-bots-on-messenger/.

[19] Vinayak Mathur and Arpit Singh. ≪The Rapidly Changing Landscape of Conversational Agents≫. In: *ArXiv* abs/1803.08419 (2018).

[20] A. M. TURING. ≪I.—COMPUTING MACHINERY AND INTELLIGENCE≫. In: *Mind* LIX.236 (Oct. 1950), pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433.

[21] John R. Searle. ≪Minds, brains, and programs≫. In: *Behavioral and Brain Sciences* 3.3 (1980), 417–424. DOI: 10.1017/S0140525X00005756.

[22] Oriol Vinyals and Quoc Le. ≪A Neural Conversational Model≫. In: *ICML Deep Learning Workshop, 2015* (June 2015).

[23] D. A. Ferrucci. ≪Introduction to "This is Watson"≫. In: *IBM Journal of Research and Development* 56.3.4 (2012), 1:1–1:15.

[24] Petter Bae Brandtzaeg and Asbjørn Følstad. ≪Why People Use Chatbots≫. In: *Internet Science*. Ed. by Ioannis Kompatsiaris et al. Cham: Springer International Publishing, 2017, pp. 377–392. ISBN: 978-3-319-70284-1.

[25] Vladimir Ilievski et al. ≪Goal-Oriented Chatbot Dialog Management Bootstrapping with Transfer Learning≫. In: *CoRR* abs/1802.00500 (2018). arXiv: 1802.00500. URL: http://arxiv.org/abs/1802.00500.

[26] Blaise Thomson Matthew Henderson and Jason Williams. *Dialog state tracking challenge 2 & 3*. 2013. URL: http://camdial.org/\textasciitildemh521/dstc/.

[27] Norbert Reithinger and Martin Klesen. ≪Dialogue Act Classification Using Language Models≫. In: *In Proceedings of EuroSpeech-97*. 1997, pp. 2235–2238.

[28] Tomas Mikolov et al. ≪Efficient Estimation of Word Representations in Vector Space≫. In: *Proceedings of the International Conference on Learning Representations*. 2013.

[29] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. ≪Sequence to Sequence Learning with Neural Networks≫. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3104–3112.

[30] Kyunghyun Cho et al. ≪Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation≫. In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[31] Tiancheng Zhao et al. ≪Generative Encoder-Decoder Models for Task-Oriented Spoken Dialog Systems with Chatting Capability≫. In: *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*. Saarbrücken, Germany: Association for Computational Linguistics, Aug. 2017, pp. 27–36. DOI: 10.18653/v1/W17-5505.

[32] Dan Jurafsky and James H. Martin. *Speech and language processing*. 3rd ed. Prentice Hall, Pearson Education International, 2014. URL: https://web.stanford.edu/~jurafsky/slp3/.

[33] Steve Young et al. ≪The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management≫. In: *Computer Speech Language* 24 (Apr. 2010), pp. 150–174. DOI: 10.1016/j.csl.2009.04.001.

[34] D. Goddeau et al. ≪A form-based dialogue manager for spoken language applications≫. In: *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96*. Vol. 2. 1996, 701–704 vol.2.

[35] Nikola Mrkšić et al. ≪Multi-domain Dialog State Tracking using Recurrent Neural Networks≫. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 794–799. DOI: 10.3115/v1/P15-2130. URL: https://www.aclweb.org/anthology/P15-2130.

[36] Tsung-Hsien Wen et al. ≪A Network-based End-to-End Trainable Task-oriented Dialogue System≫. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 438–449.

[37] Ehud Reiter and Robert Dale. ≪Building Applied Natural Language Generation Systems≫. In: *Nat. Lang. Eng.* 3.1 (Mar. 1997), 57–87. ISSN: 1351-3249. DOI: 10.1017/S1351324997001502. URL: https://doi.org/10.1017/S1351324997001502.

[38] Tsung-Hsien Wen et al. ≪Stochastic Language Generation in Dialogue using Recurrent Neural Networks with Convolutional Sentence Reranking≫. In: *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Prague, Czech Republic: Association for Computational Linguistics, Sept. 2015, pp. 275–284.

DOI: 10.18653/v1/W15-4639. URL: https://www.aclweb.org/anthology/W15-4639.

[39] Tsung-Hsien Wen et al. ≪Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems≫. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1711–1721. DOI: 10.18653/v1/D15-1199. URL: https://www.aclweb.org/anthology/D15-1199.

[40] Michael Kipp. ≪ANVIL - A Generic Annotation Tool for Multimodal Dialogue≫. In: *Proceedings of the 7th European Conference on Speech Communication and Technology*. Jan. 2001, pp. 1367–1370.

[41] Pontus Stenetorp et al. ≪BRAT: A Web-Based Tool for NLP-Assisted Text Annotation≫. In: *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*. EACL '12. Avignon, France: Association for Computational Linguistics, 2012, 102–107.

[42] Youxuan Jiang, Jonathan K. Kummerfeld, and Walter S. Lasecki. ≪Understanding Task Design Trade-offs in Crowdsourced Paraphrase Collection≫. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 103–109. DOI: 10.18653/v1/P17-2017. URL: https://www.aclweb.org/anthology/P17-2017.

[43] Zhao Yan et al. ≪DocChat: An Information Retrieval Approach for Chatbot Engines Using Unstructured Documents≫. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 516–525. DOI: 10.18653/v1/P16-1049. URL: https://www.aclweb.org/anthology/P16-1049.

[44] Eric Atwell and Bayan Shawar. ≪Using dialogue corpora to train a chatbot≫. In: *Proceedings of the Corpus Linguistics 2003 conference*. Jan. 2003, pp. 681–690. DOI: 10.13140/2.1.1455.7122.

[45] Bayan Abu-Shawar and Eric Atwell. ≪Automatic Extraction of Chatbot Training Data from Natural Dialogue Corpora≫. In: *RE-WOCHAT: Workshop on Collecting and Generating Resources for Chatbots and Conversational Agents-Development and Evaluation Workshop Programme*. 2016, p. 29.

[46] Wang Ling et al. ≪Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems≫. In: *CoRR* abs/1705.04146 (2017). arXiv: 1705.04146. URL: http://arxiv.org/abs/1705.04146.

[47] Spencer Rothwell et al. ≪Data collection and annotation for state-of-the-art NER using unmanaged crowds≫. In: *INTERSPEECH*. 2015.

[48] W. Y. Wang et al. ≪Crowdsourcing the acquisition of natural language corpora: Methods and observations≫. In: *2012 IEEE Spoken Language Technology Workshop (SLT)*. 2012, pp. 73–78.

[49] Matteo Negri et al. ≪Chinese Whispers: Cooperative Paraphrase Acquisition≫. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Istanbul, Turkey: European Language Resources Association (ELRA), May 2012, pp. 2659–2665. URL: http://www.lrec-conf.org/proceedings/lrec2012/pdf/772_Paper.pdf.

[50] Miroslav Vodolán and Filip Jurcícek. ≪Data Collection for Interactive Learning through the Dialog≫. In: *CoRR* abs/1603.09631 (2016). arXiv: 1603.09631.

[51] Bing Liu et al. ≪Dialogue Learning with Human Teaching and Feedback in End-to-End Trainable Task-Oriented Dialogue Systems≫. In: *CoRR* abs/1804.06512 (2018). arXiv: 1804.06512.

[52] Jost Schatzmann et al. ≪Agenda-Based User Simulation for Bootstrapping a POMDP Dialogue System≫. In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*. Rochester, New York: Association for Computational Linguistics, Apr. 2007, pp. 149–152. URL: https://www.aclweb.org/anthology/N07-2038.

[53] L. Liu et al. ≪Generating Diverse and Descriptive Image Captions Using Visual Paraphrases≫. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 4239–4248.

[54] Ashish Vaswani et al. ≪Attention is All you Need≫. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 5998–6008.

[55] Hirokazu Kameoka et al. ≪ConvS2S-VC: Fully convolutional sequence-to-sequence voice conversion≫. In: *CoRR* abs/1811.01609 (2018). arXiv: 1811.01609. URL: http://arxiv.org/abs/1811.01609.

[56] Jekaterina Novikova et al. ≪Why We Need New Evaluation Metrics for NLG≫. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 2241–2252. DOI: 10.18653/v1/D17-1238. URL: https://www.aclweb.org/anthology/D17-1238.

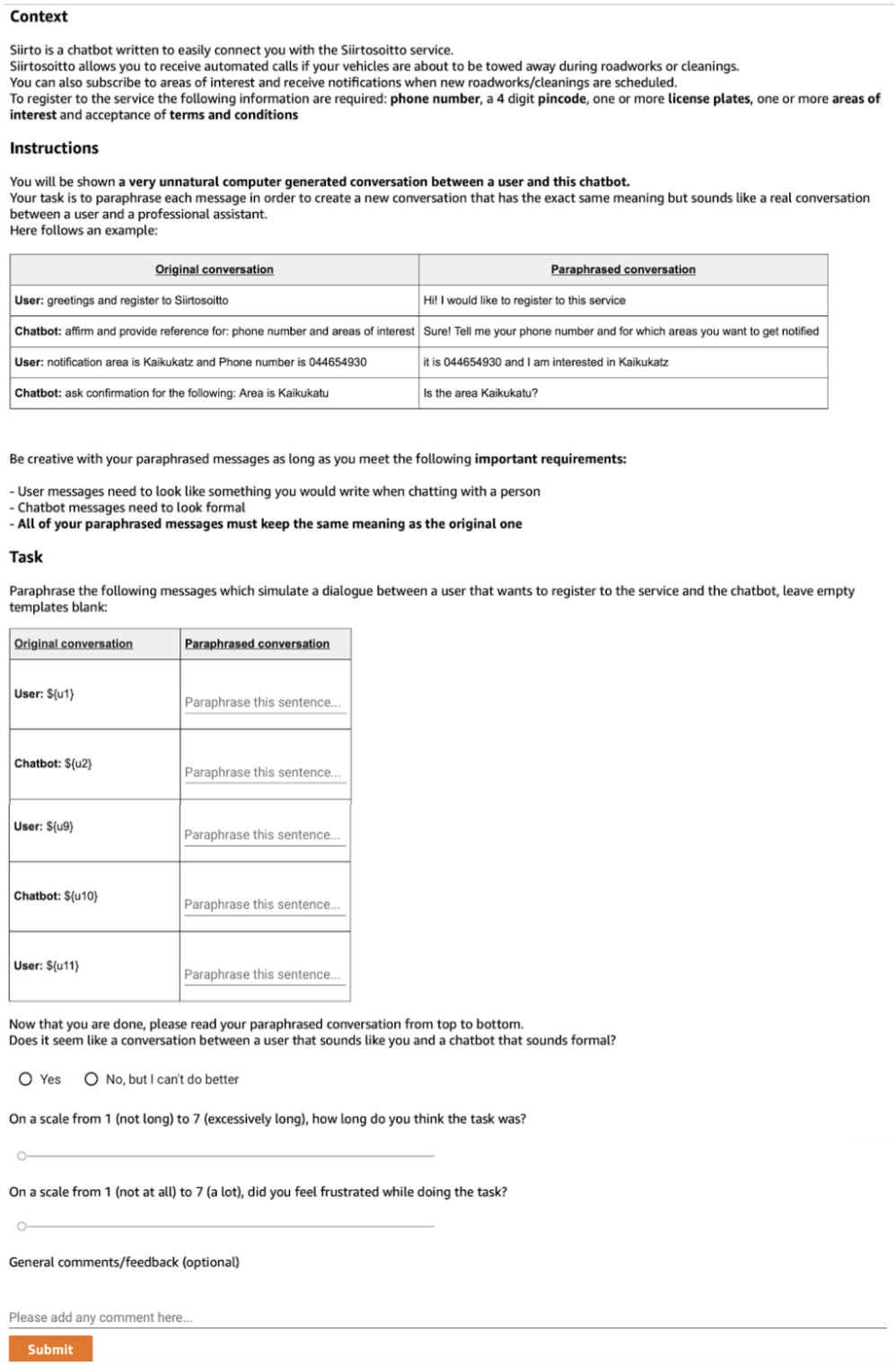
Appendix A

Supplementary material

```
"_id": String,
"createdAt": 2020-04-16T13:47:05.761+00:00,
"annotatedMessages": [
    {
        "sender": User
        "annotations": [
            {
                "act": greeting
                "slot": none
                "value": none
            }
        ],
        "template": greetings
        "rewrite": hi there
    },
    ...
    {
        "sender": Chatbot
        "annotations": [
            {
                "act": request
                "slot": Phone number
                "value": none
            },
            ...
            {
                "act": request
                "slot": License plate
                "value": none
            }
        ],
        "template": provide reference for: phone
           number and license plates
        "rewrite": good, what is your phone number?
           and the license number of your car?
    }
],
"processed": true,
"assigned": false
```

```
"feedback": [
    {
        "question": Does it seem like a conversation
            between a user that sounds like you and
           a chatbot that sounds formal?
        "answer": yes
    },
    {
        "question": How long do you think the task
           was?
        "answer": 3
    },
    {
        "question": Did you feel frustrated while
           doing the task?
        "answer": 2
    },
    {
        "question": Do you have any additional
           comment?
        "answer": it would have been great to have
           an example of the whole discussion
           somewhere
    },
]
```

Listing A.1: Dialogue document format in MongoDB



Context

Siirto is a chatbot written to easily connect you with the Siirtosoitto service.
Siirtosoitto allows you to receive automated calls if your vehicles are about to be towed away during roadworks or cleanings.
You can also subscribe to areas of interest and receive notifications when new roadworks/cleanings are scheduled.
To register to the service the following information are required: **phone number**, a 4 digit **pincode**, one or more **license plates**, one or more **areas of interest** and acceptance of **terms and conditions**

Instructions

You will be shown **a very unnatural computer generated conversation between a user and this chatbot.**
Your task is to paraphrase each message in order to create a new conversation that has the exact same meaning but sounds like a real conversation between a user and a professional assistant.
Here follows an example:

Original conversation	Paraphrased conversation
User: greetings and register to Siirtosoitto	Hi! I would like to register to this service
Chatbot: affirm and provide reference for: phone number and areas of interest	Sure! Tell me your phone number and for which areas you want to get notified
User: notification area is Kaikukatz and Phone number is 044654930	it is 044654930 and I am interested in Kaikukatz
Chatbot: ask confirmation for the following: Area is Kaikukatu	Is the area Kaikukatu?

Be creative with your paraphrased messages as long as you meet the following **important requirements:**

- User messages need to look like something you would write when chatting with a person
- Chatbot messages need to look formal
- **All of your paraphrased messages must keep the same meaning as the original one**

Task

Paraphrase the following messages which simulate a dialogue between a user that wants to register to the service and the chatbot, leave empty templates blank:

Original conversation	Paraphrased conversation
User: ${u1}	Paraphrase this sentence...
Chatbot: ${u2}	Paraphrase this sentence...
User: ${u9}	Paraphrase this sentence...
Chatbot: ${u10}	Paraphrase this sentence...
User: ${u11}	Paraphrase this sentence...

Now that you are done, please read your paraphrased conversation from top to bottom.
Does it seem like a conversation between a user that sounds like you and a chatbot that sounds formal?

○ Yes ○ No, but I can't do better

On a scale from 1 (not long) to 7 (excessively long), how long do you think the task was?

On a scale from 1 (not at all) to 7 (a lot), did you feel frustrated while doing the task?

General comments/feedback (optional)

Please add any comment here...

Submit

Figure A.1: The visual design of the rewriting task proposed to AMT workers

Annotation	**Template utterance**	**Paraphrase**
greeting(), inform(intent= register)	Greetings and register to Siirtosoitto	hello i would like to join siirtosoitto service
affirm(), request(phone_number), request(license_ plate), request(areas), request(terms_and_conditions)	affirm and provide reference for: phone number and license plates and areas and acceptance of terms and conditions	please list the areas you are interested to get notified among with your phone number (these areas could be changed later)
inform(area= Sylvesterintie)	notification area is Sylvesterintie	I would like to get notification for Sylvesterintie
request(license_plate)	provide reference for: License plate	good, please tell me your license plate number
inform(license_plate= VBM-612)	License plate is VBM-612	my license plate is vbm612
request(phone_number)	provide reference for: Phone number	please give me your phone number
inform(phone_number= 045523273)	Phone number is 045523273	my phone number is 045523273
request(pincode)	provide reference for: received Pincode	What is the pincode you received?
inform(pincode=2963), inform(terms_and_conditions=true), inform(area= museokatu)	pincode is 2963 and terms and conditions are accepted and notification area is museokatu	pincode is 2963 and i accepted the terms and conditions and a new notification area is museokatu
notify_success(), inform(registration=true)	success! and registration is completed	thank you all information are successfully registered
bye()	thank you, bye	thanks take care

Table A.1: A full dialogue sample showcasing the three main steps in M2M: generation of annotations, translation of annotation to template utterance with a domain grammar and, contextual paraphrasing of the templates

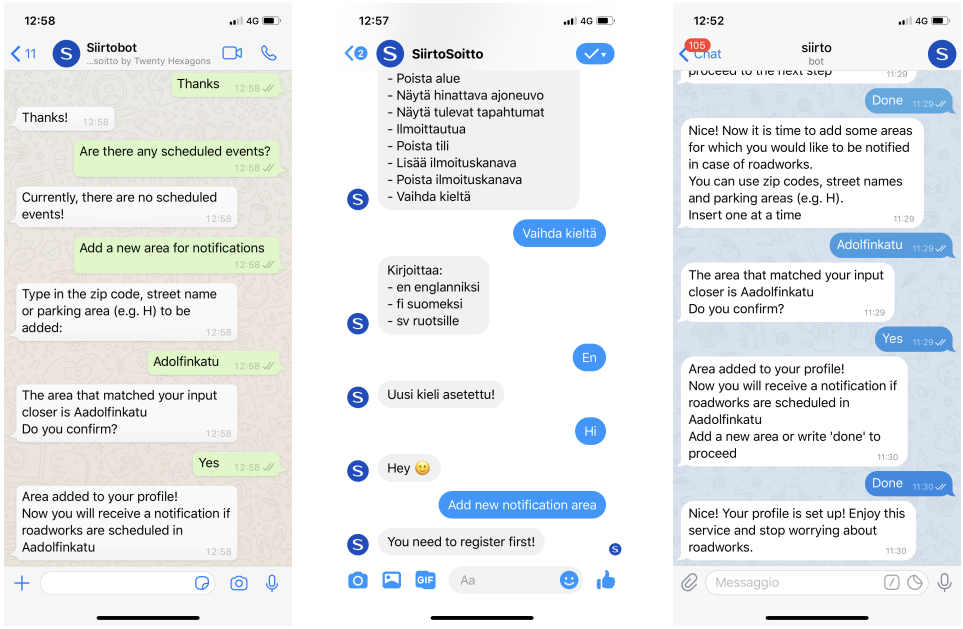


Figure A.2: SiirtoSoitto rule-based chatbot on Whatsapp (left), Facebook Messenger (center) and Telegram (right)

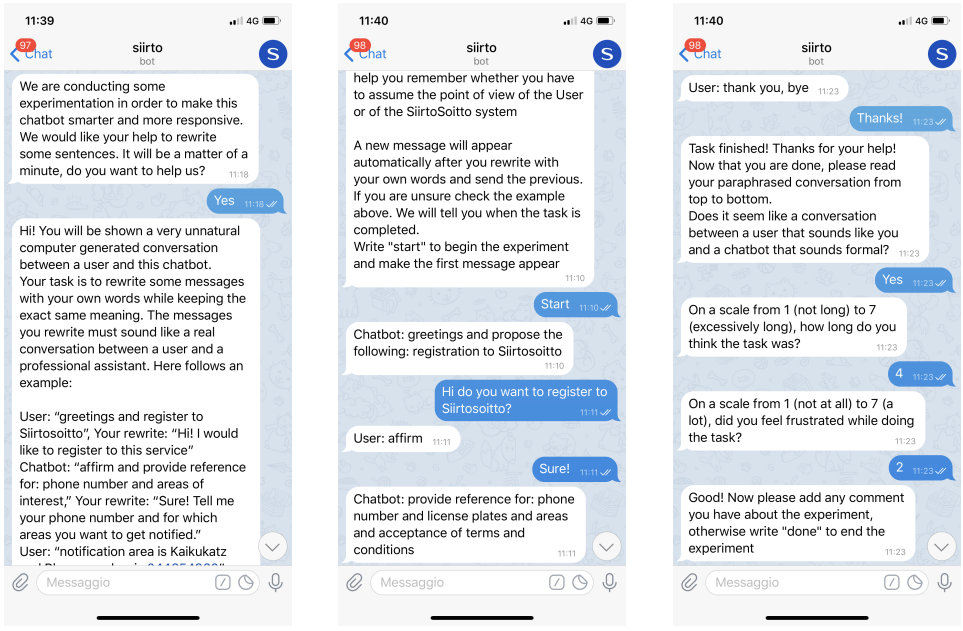


Figure A.3: Crowdsourcing phase using the rule-based SiirtoSoitto chatbot on Telegram