

Umeå University  
Department of Computing Science  
SE-901 87 UMEÅ  
Sweden

# **Dialogue Systems Using Web-based Language Tools**

André Niklasson

Spring 2017  
Master's Thesis in Computing Science, 30 credits  
Supervisors at CS-UmU: Suna Bensch and Thomas Hellström  
Examiner: Henrik Björklund



## Abstract

Chatbots in commercial environments are on the rise with the release of several web-based language understanding tools. The vast majority of the dialogue systems deployed today uses very primitive state-machine architectures to model their interactions. These primitive approaches are reliable and easy to implement but the dialogue becomes very unnatural and the system always has the initiative in the conversation. The positive features of being easy to build, and the ability to easily retain control over the system normally supersedes the shortcomings.

This thesis proposes a dialogue model that utilizes new approaches for dialogue modeling but aims to be easy to configure. The proposed dialogue management strategy is implemented in a prototype dialogue system. Developers are able to model their dialogues using an XML dialogue description. The system utilizes LUIS.ai, a recently launched web-based language tool for sentence analysis. LUIS.ai is evaluated together with the prototype dialogue system.



## **Acknowledgements**

I would like to thank Suna Bensch and Thomas Hellström for presenting the idea of this thesis and providing great feedback. I would also like to thank my parents for their support throughout this thesis.



# Contents

<b>Abstract</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Thesis outline . . . . .	1
<b>2 Background</b>	<b>3</b>
2.1 Dialogue . . . . .	3
2.1.1 Turn-taking . . . . .	4
2.1.2 Grounding . . . . .	4
2.1.3 Implicature . . . . .	5
2.1.4 Accommodation . . . . .	5
2.2 Dialogue systems . . . . .	6
2.2.1 Components . . . . .	6
2.3 Natural language understanding . . . . .	7

---

2.3.1	Semantic formats . . . . .	7
2.3.2	Machine learning . . . . .	8
2.4	Dialogue management . . . . .	8
2.4.1	Finite-state approaches . . . . .	9
2.4.2	Frame-based approaches . . . . .	10
2.4.3	Plan-based approaches . . . . .	10
2.4.4	Statistical approaches . . . . .	11
2.4.5	Handcrafted vs. Statistical approaches . . . . .	12
2.5	Evaluation of dialogue systems . . . . .	12
2.5.1	Dialogue cost . . . . .	12
2.5.2	Task success . . . . .	13
2.5.3	Interaction quality . . . . .	13
<b>3</b>	<b>Problem description</b>	<b>14</b>
3.1	Purpose . . . . .	15
<b>4</b>	<b>Related work in dialogue management</b>	<b>16</b>
4.1	E-form . . . . .	16
4.2	Information-state . . . . .	16
4.2.1	Issue-based dialogue management . . . . .	17



---

4.3	The Topic Forest plan-based structure . . . . .	17
4.4	RavenClaw . . . . .	18
<b>5</b>	<b>Proposal of a frame-based dialogue management strategy</b>	<b>20</b>
5.1	Weaknesses of frame-based systems . . . . .	20
5.2	The frame . . . . .	21
5.3	Dialogue state . . . . .	22
5.4	Information merging . . . . .	23
5.5	Questions . . . . .	24
5.6	Summary . . . . .	24
<b>6</b>	<b>LUIS.ai</b>	<b>26</b>
6.1	Natural language understanding . . . . .	26
6.1.1	Intents and entities . . . . .	26
6.2	Creating and managing a LUIS.ai application . . . . .	27
6.2.1	Training and test tool . . . . .	29
6.3	Integrating LUIS.ai with a system . . . . .	30
6.4	Limitations . . . . .	31
<b>7</b>	<b>Implementation of the prototype system</b>	<b>33</b>
7.1	Architecture . . . . .	33

7.2	Dialogue document . . . . .	34
7.2.1	Model with LUIS.ai . . . . .	36
7.3	Device.py . . . . .	36
7.4	Summary . . . . .	37
<b>8</b>	<b>Evaluation</b>	<b>38</b>
8.1	Evaluation method . . . . .	38
8.2	Evaluation results . . . . .	39
8.2.1	Wizard of Oz test . . . . .	39
8.2.2	Talkamatic Dialogue Strategies examples . . . . .	40
8.2.3	Dialogue cost and task success . . . . .	45
8.2.4	Dialogue modeling . . . . .	46
<b>9</b>	<b>Conclusion</b>	<b>48</b>
9.1	Purpose . . . . .	48
9.2	Future work . . . . .	49
	<b>Bibliography</b>	<b>50</b>
<b>A</b>	<b>Dialogue document</b>	<b>53</b>

# List of Figures

2.1	Example of grounding. . . . .	5
2.2	Example of implicature . . . . .	5
2.3	The traditional architecture of text-based dialogue systems . . . . .	6
2.4	A simple example of a finite-state dialogue management approach. . .	9
4.1	Part of a topic forest (similar to [19, p. 2]) . . . . .	18
5.1	A simple frame that covers the topic of reserving tickets for a movie .	22
5.2	A simple frame that covers the topic of acquiring of personal information. . . . .	23
5.3	Example of how the system can switch between different topics. . . .	23
6.1	A view listing the current intents. . . . .	27
6.2	Adding an intent with an unique identifier. . . . .	28
6.3	A view listing the current entities and the different entity options. . .	28
6.4	An example of when creating a custom simple entity. . . . .	29

---

6.5	The interface when training an application. . . . .	29
6.6	The training and test tool. . . . .	30
6.7	A sample JSON response of the prediction of the utterance ””My airline is SAS”. . . . .	31
7.1	Architecture overview . . . . .	34
7.2	An example of a specified simple dialogue for reserving movie tickets. ”[value]” can be used in the Confirmation element to present the obtained information to the user. . . . .	35
8.1	Example of over-answering provided by Talkamatic . . . . .	40
8.2	Example of the prototype system handling over-answering . . . . .	40
8.3	Example of other-answering provided by Talkamatic . . . . .	41
8.4	Example of the prototype system handling other-answering . . . . .	41
8.5	Example of a system correcting an error provided by Talkamatic . . . . .	42
8.6	Example of the prototype system when the user is correcting a mistake . . . . .	42
8.7	Example of a topic shift provided by Talkamatic . . . . .	43
8.8	Example of a topic shift by the prototype system . . . . .	43
8.9	Example of several topic shifts by the prototype system . . . . .	44
8.10	Example of task recognition provided by Talkamatic . . . . .	44
8.11	Example of the prototype system successfully performs task recognition . . . . .	45

8.12 The results of the first try. . . . .	45
8.13 The results of the second try. . . . .	46



# Chapter 1

## Introduction

Chatbots have existed for more than 60 years but the area has recently seen a massive upswing. The ambition of this project is to design and implement a dialogue system utilizing modern tools for sentence analysis and developing a domain-independent dialogue-model for construction of simple web-based dialogue applications.

### 1.1 Thesis outline

The remainder of the thesis is structured as follows:

**Chapter 2: Background**

Introduces the fundamental concepts and methods used throughout the thesis.

**Chapter 3: Problem description**

Introduces the problem statement and the overall goals of the project.

**Chapter 4: Related work in dialogue management**

Introduces relevant work in the field of dialogue management.

**Chapter 5: Proposal of a frame-based dialogue management strategy**

Describes the proposed dialogue management strategy of the thesis.

**Chapter 6: LUIS.ai**

Introduces the web-based language tool LUIS.ai. Describes the functionality and limitations of the web-based language service.

**Chapter 7: Implementation of the prototype system**

Describes the implemented prototype dialogue system and how it integrates with LUIS.ai.

**Chapter 8: Evaluation**

Presents the evaluation of the work done in this thesis.

**Chapter 9: Conclusion**

Conclusions of the thesis are presented and tied back to the project goals. Discusses future development.



# Chapter 2

## Background

This chapter introduces the fundamental concepts and methods used throughout the thesis. We start off with relevant linguistic concepts that are important in the field of dialogue management. We then move on to discuss how modern text-based dialogue systems are designed and what components are of importance in modern architectures. The thesis then moves on to definitions of dialogue management and introduces the most common approaches and strategies used today. Finally we discuss methodologies of how to evaluate a dialogue system in terms of task success and dialogue cost.

### 2.1 Dialogue

Humans communicate with one another everyday through both text and speech. A dialogue is an interaction between two or more agents that communicate with each other through natural language. The dialogue fulfills social needs such as exchange of information, building trust and relationships or coordination and collaboration. The ability to engage and maintain a dialogue is something we start developing through early childhood [1, p. 9]. One of the fundamentals of a dialogue is that it is a collaborative activity with two or more agents that share information with each other. The agents collaborate to obtain a mutual understanding of each other's desires and goals.

### 2.1.1 Turn-taking

One often-neglected aspect of dialogue is that agents take turns in sharing their information with each other. The roles of who is writing or talking and who is reading or listening change during the course of the dialogue. The agent who is talking or writing has the initiative. The agent releases the control of the dialogue when it is the other agent's turn to have the initiative to control the conversation.

How these turns are passed between the agents during the conversation depends heavily on the situation and on the participants. Techniques for how to allocate turns are usually divided into two groups: (1) the next agent to hold the initiative and speak is allocated by the current speaker; and (2) the next agent to hold the initiative is allocated by self-selection [2, p. 696-735]. An example of the first technique is when a person ask a direct question or is making an offer to another person. With that action, the turn is passed to the other person to respond. An example of the second technique is when a person decides to interrupt the current speaker.

### 2.1.2 Grounding

Problems arise when utterances by the sender are perceived and understood incorrectly by the receiver. The agents must have a mutual understanding of each other's contribution to the dialogue. The agents must share a *common ground* of knowledge, beliefs and assumptions that are established during the interaction [1, p. 14].

To obtain a common ground in a dialogue the sender can not simply send a message and take it for granted that the receiver will understand the context in the correct way. The sender and receiver must send feedback back and forth about the reception, comprehension and acceptance of the information that is communicated. This process is commonly known as *grounding* [3, p. 33]. A typical phrase for grounding if the receiver cannot make sense of a message is "Excuse me?" or "What did you say?", to give the sender a chance to rephrase his message. A common approach of grounding is that receiver shows that the utterance is understood. An example is shown in fig 2.1 below. Agent *A* shares information that he does not have any friends and the respondent *B* shares how he interpreted the statement. Agent *A* acknowledges that the interpretation of the information is correct and they now share a common ground regarding that information.

A: *"I don't have any friends."*  
B: *"So you are lonely?"*  
A: *"Yes...."*

Figure 2.1: Example of grounding.

### 2.1.3 Implicature

Semantic conclusions can be drawn from a message even though it is not specifically stated but implied by the context [1, p. 13]. An implicature is something that is inferred from an utterance and the context of the dialogue. An example is shown below:

A: *"Is Bob coming to work today?"*  
B: *"Bob is sick."*  
A: *"Ah, okay!"*

Figure 2.2: Example of implicature

B does not specify directly that Bob is not coming to work today, but given the context that Bob is sick he implicitly says that Bob will not show up at work. In this case, B counts on A being able to draw that conclusion.

### 2.1.4 Accommodation

Adaptation or adjustment in dialogues is commonly known as accommodation. An example of this could be a change of topic in the discussion or jumping in and out of different contexts [31, p. 1]. A dialogue system's ability to handle multiple topics is not the same as supporting accommodation. For instance, a system may have a static dialogue flow that is not affected by the user input. A system supporting accommodation features can jump back and forth between topics and contexts by recognizing the intention of the user.

## 2.2 Dialogue systems

A dialogue system is a software that communicates with a human user through natural language on a turn-to-turn basis. The main purpose of a dialogue system is to provide an interface between a user and a computer-based application [4, p. 3]. The sophistication of different dialogue systems varies immensely, ranging from systems that are functioning on a question-answer basis, to systems that strives to achieve a human-like performance of conversational interaction.

The goal of conversational dialogue systems is not to handle all kinds of human dialogue due the complexity of the human language. Allen et al. [5, p. 3] claims that the construction of a system with full understanding of the human language is such a difficult task that it will not be achieved in the foreseeable future. A constraint for most dialogue systems is therefore that the systems are focused on accomplishing specific and concrete tasks [3, p. 12].

### 2.2.1 Components

The most common approach of constructing a text-based dialogue system is by dividing it into three components. The three components are the *natural language understanding unit*, the *dialogue manager* and the *natural language generator*.

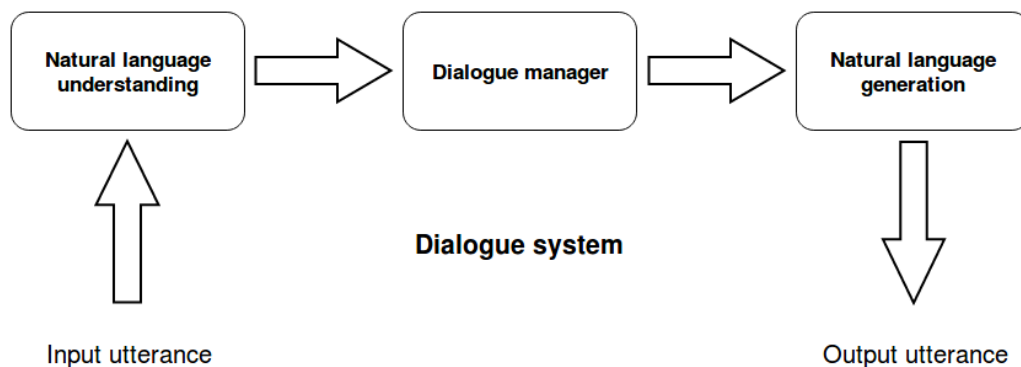


Figure 2.3: The traditional architecture of text-based dialogue systems

The *natural language understanding unit* reads the input from the user and extracts the semantic information of that utterance. The information is then modeled such that it is readable by the dialogue manager.

The *dialogue manager* is the controller of the dialogue system. It is responsible of maintaining and updating the state of the dialogue. Given a state it decides the next action to be taken and the overall behavior of the system.

Given a chosen action that is decided by the dialogue manager, the *natural language generator* chooses the best linguistic realization for the abstract communicative goal expressed in the action [1, p. 21]. One of the easiest approaches is to have a variety of prepared answers that are mapped to specific actions [3, p. 28]. More sophisticated approaches, such as statistical methods, have recently been an active topic of research [1, p. 21].

## 2.3 Natural language understanding

The task of the natural language understanding unit is to parse the textual input from the user into a semantic representation that can be interpreted by the dialogue manager. A common approach is to use context-free grammar that is enhanced with semantic information to parse the input [3, p. 22]. Another traditional approach is to use keyword or pattern matching, where each word or pattern of words are paired with semantic representations. A major drawback with these approaches is that their simplicity can over-generalize the semantic context of complex utterances.

### 2.3.1 Semantic formats

A common format to represent the semantics of an utterance is key-value frames [3, p. 23]. With this approach many aspects of the semantics are lost but the basic information necessary to update the state of the dialogue is well represented. The approach is quite primitive but can be improved by using nested frames. With nested frames, complex utterance can be interpreted and modeled. A regular frame that models the interpretation of the utterance: *"I want a cheeseburger with fries and a coke!"* could be presented as:

```
meal: { dish : cheeseburger
        side : fries
        drink : coke
      }
```

With nested frames it is easier to model utterances with more information. A nested frame model that models the interpretation of the utterance: *"I want one cheeseburger with fries and a coke, and then I want to order a Hawaii pizza with Fanta to drink!"* could be presented as:

```
order: { [ dish : cheeseburger
         side : fries
         drink : coke
       ]
        [ dish: Hawaii pizza
         side : none
         drink : Fanta
       ]
      }
```

### 2.3.2 Machine learning

A modern approach is to use data-driven approaches who utilizes machine learning algorithms. These approaches are deemed to be more robust but requires a large corpus of interactions and utterances that can be classified by the machine learning algorithms [3, p. 23].

## 2.4 Dialogue management

The dialogue manager (DM) has a central role in a dialogue system and coordinates the activity of all components, controls the dialogue flow, and communicates with external applications [12, p. 3]. The DM is responsible of maintaining and updating a representation of the dialogue state as it receives new input from the natural language understanding unit (NLU). The dialogue state should represent all information that is relevant to the system [1, p. 20]. Based on the state of the dialogue, the DM is responsible for selecting the next action to be performed by the system. The action could range from a communicative action to a query search of information, or no action at all. The DM is also responsible of handling errors and unexpected input from the NLU.

Several popular approaches have been studied and used for dialogue management. All common approaches represent the dialogue state as a data structure and use a decision mechanism dependent on the state that selects an appropriate action [1, p. 23]. In the following four sections, four common approaches will be briefly described: Finite-state, Frame-based, Plan-based and Statistical.

### 2.4.1 Finite-state approaches

The most simple and most common approach for dialogue management is the finite-state, approach which is based on a finite-state automaton. This approach statically maps each dialogue state to a state in a state machine. Each state has directed edges to other states. Each edge represents dialogue moves and is labeled with conditions. When a condition of an edge is satisfied the current state will move from the source of the edge to its target state [1, p. 23]. All of the states and their possible transitions must be designed in advance.

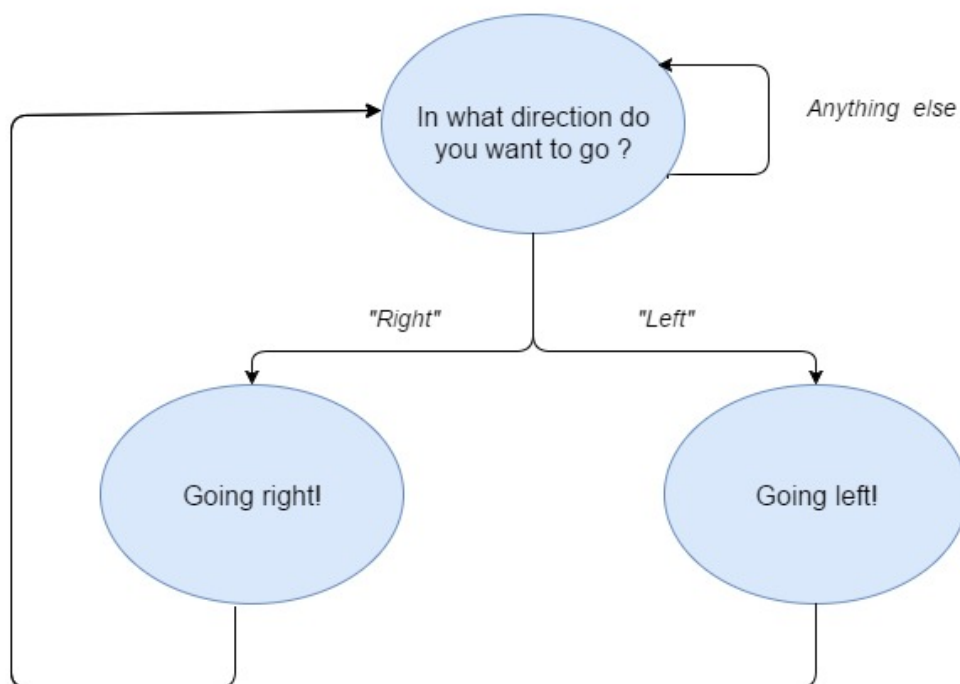


Figure 2.4: A simple example of a finite-state dialogue management approach.

Figure 2.4 shows a very primitive example of a finite-state architecture. The modeled domain asks the user for a direction but only covers the input "left" or "right". If

the user input is "forward" or "backwards" the system would ignore the input and repeat the question of the first state.

The finite-state approach is appropriate when the relevant semantic information of the dialogue may be represented as states in the state machine. It is not a flexible approach in terms of dialogue and does not allow the user to ever hold the initiative of the conversation [3, p. 26].

### **2.4.2 Frame-based approaches**

In a frame-based, or slot-filling, approach the state of the dialogue is represented by a frame of information. The frame comprises empty slots of desired information that are gradually filled with user inputs during the course of the dialogue [1, p. 24]. In each turn the system decides what its next best suited question or response is to fill the remaining slots of desired information. When sufficiently many slots in the frame are filled with information the system performs an action, such as a query search.

A system utilizing this approach does not have a fixed dialogue flow as in the finite-state approach. The flow depends on the context of the dialogue, and the initiative of the conversation may move between the user and the system throughout the dialogue [14, p. 23]. If an input from the user contains information for multiple slots, even though the system didn't ask for it, the system acknowledges this and fills the appropriate slots.

Systems utilizing the frame-based approach are primarily suited for domains such as ticket booking or reservations. It offers advantages over the finite-state approach in domain modeling and dialogue control, but can still be improved. One such improvement is to incorporate several contexts. With separate frames for different contexts, the system can be used for more complex domains [14, p. 23-24].

### **2.4.3 Plan-based approaches**

A plan-based system treats each utterance as a performed action to achieve a specific goal. The system identifies the goal and intention of the user and then dynamically builds a communicative plan that is believed to obtain the goal [3, p. 27]. A developer



modeling a dialogue in the spirit of the plan-based approach must therefore break down the overall task into smaller plans and goals [21, p. 24]. The system has access to a library with dialogue moves, where each move is mapped with preconditions, constraints and effects. These moves are then put together to fulfill the desired goal and tasks [3, p. 27].

Plan-based systems that dynamically build the dialogue are usually complex. Most dialogue systems operate in limited domains where such generalized and flexible models are not required. Skantze [3, p. 27] argues that an easier way is to define the plans in advance, given that usually a limited number of plans are needed in a given domain. See section 4.3 for an example of this, presented in the Topic-Forest approach.

#### 2.4.4 Statistical approaches

Statistical approaches to dialogue management are popular in dialogue systems research. The statistical, or data-driven, approaches utilize learning algorithms to develop a function that associates possible dialogue states to system actions [1, p. 25]. The key motivation for this approach is to reduce the cost and complexity of hand-crafted approaches such as the finite-state approach when the dialogue has to be more complex.

One of the most popular methods is to utilize partially observable Markov decision processes (POMDPs) [13, p. 1]. They work as a probabilistic transition model; given the current dialogue state  $A$ , what is the likelihood that the system will go to state  $B$ ? [11, p. 13] All statistical approaches apply learning algorithms on a large set of interaction data.

One of the major benefits of using statistical approaches is the improved ability to handle unexpected events. Probabilistic reasoning techniques typically account for uncertainty of input interpretations [1, p. 28]. However, one of the most apparent challenges is the amount of data needed for learning. Statistical methods typically need big data sets to estimate the parameters [1, p. 28]. Big corpora of interactions are expensive and hard to obtain. Another issue is that the standard POMDP methods are not tested enough of bigger domains and does not scale well enough with the complexity of real-world dialogues [13, p. 3].

## 2.4.5 Handcrafted vs. Statistical approaches

The finite-state, frame-based, and plan-based approaches are all commonly referred to as handcrafted approaches since, the dialogue needs to be modeled by an expert. Most of today's research focus on statistical approaches. As mentioned in the previous section, a very big drawback of these approaches is that they require large corpora of interaction data to train for a particular domain. Acquiring this type of data can be very difficult. Dialogue managers utilizing machine learning are also not deterministic and can be very unpredictable [21, p. 30]. This makes them unfit for applications where safety and security are big issues.

Handcrafted approaches are easier to implement in simple domains. They are also to prefer when the dialogue has a clear goal, such as booking tickets or reservations. In these cases, the actions can normally be derived from the input and the actions that caused them [21, p. 30]. The weaknesses of the statistical approaches make them unfit for commercial use as of today.

## 2.5 Evaluation of dialogue systems

There are many proposed methodologies on how to evaluate the quality of a dialogue system. One of the most prominent is the PARADISE framework that has been developed for spoken dialogue systems [15, p. 271-280]. Walker et al. presents the idea of finding a relation between the satisfaction of the user and objective measures of the systems. These measures are mainly *task success* and *dialogue cost*. Studies have also shown that interaction quality, how enjoyable and smooth the conversation is, may be of more importance than efficiency [16, p. 2].

### 2.5.1 Dialogue cost

One of the measurements we will focus on in this thesis is dialogue cost. This measures the efficiency of a task-based dialogue system. This is typically measured by the number of conversational turns or elapsed time to complete a task [15, p. 275].

### **2.5.2 Task success**

The second measurement is task success. Task success measures how well the system completed the underlying goal of the dialogue [15, p. 273]. Did the system present the right information or perform the right action at the end of the dialogue? Was the user satisfied with the result?

### **2.5.3 Interaction quality**

Interaction quality is usually evaluated by user testing. According Silvervag et al. [16, p. 2], user testing can be problematic since users tend to enjoy testing and talking to a developed dialogue system and give positive feedback and thus do not take factors such as elapsed time into account.

# Chapter 3

## Problem description

The vast majority of the commercial dialogue systems deployed today use simple state machine architectures to model interactions [23, p. 216]. Stoyanchev et al. claims that the main reasons for this is the need to retain control over the behavior of the system, being able to easily modify the internal models and for the system to scale to large number of users [23, p. 216]. These properties are often regarded as more important than shortcomings such as only system-initiative dialogue. Another reason for the use of primitive strategies is that the developers otherwise would have to invest a lot of efforts to learn the more academic approaches to dialogue management [27, p. 1]. An additional major drawback of the state machine approaches is the lack of abstraction between the dialogue strategy and the domain. If the developer wants to create a new dialogue in a different domain, the whole dialogue structure and strategy must be remodeled.

Markus M. Berg claims that there is a lack of systems where users easily can configure their dialogues without implementing a whole system [27, p. 1]. During the last two decades several toolkits and innovations utilizing more advanced dialogue strategies has been developed but very few are being used in commercial environments [23, p. 216].

With the recent launches of several web-based language tools provided by major companies, Lison et al. proposed a dialogue system that utilizes a simple single frame-based dialogue management approach [23]. The system uses trained web-based language tools such as LUIS.ai for the language understanding and lets the user configure a single frame for an information-seeking dialogue system. The pro-

---

prototype received a lot of positive feedback from fellow researchers [23, p. 218].

## 3.1 Purpose

The overall goal of the project is to investigate how modern tools for language analysis can be used in combination with improved versions of common dialogue system architectures to create a flexible and powerful dialogue system. In order to reach this goal, the following objectives are defined.

- Evaluation of LUIS.ai as a natural language understanding module in a dialogue system for simple domains used by non-experts in machine-learning.
- Identification of the benefits and drawbacks of using a web-based language tool such as LUIS.ai for sentence analysis.
- Investigation of how the known weaknesses of the trivial frame-based approach of dialogue management can be addressed.
- Proposal of a dialogue management strategy that makes it easy to define dialogues for new applications. The strategy should be able to handle multiple topics, accommodation, and mixed- initiative dialogues.
- Implementation of a prototype dialogue system that utilizes LUIS.ai for sentence analysis and the proposed dialogue management strategy. Dialogue applications should be easily programmed using a markup or script language.

# Chapter 4

## Related work in dialogue management

This chapter presents studies relevant for improvement of frame-based dialogue management strategies.

### 4.1 E-form

One early variation of the trivial frame-based approach is the *E-form* [20]. In the E-form approach, the slots of the frame have associated prompts and priorities. The priority dictates the order in which the system tries to acquire information of the slot given the current state of the dialogue. The slots can be either optional or mandatory.

### 4.2 Information-state

*TrindiKit* and *Talkamatic Dialogue Manager* are two dialogue toolkits utilizing the *information-state* approach of dialogue management [29]. A dialogue manager utilizing the information-state approach has a universal state of information that is based on observed dialogue moves. The control-algorithm has a set of update-rules with preconditions related to the information-state [3, p. 27]. Based on the current

information-state, the control-algorithm chooses the next dialogue move to react on the input of the user [27, p. 2].

### 4.2.1 Issue-based dialogue management

The information-state is a very general approach that could be implemented in different ways. A popular way of implementing it is by utilizing the *issue-based* dialogue strategy proposed by Larsson et al. [30].

The issue-based strategy models the dialogue as issues that are raised during the dialogue. Each issue contains a plan of how to be resolved. A plan can contain other plans which forms a hierarchy of plans. A plan usually consists of desired information that is to be extracted from the user by printing mapped prompts.

The information-state of an issue-based dialogue manager contains a stack where all the issues that are to be resolved are stored. When a dialogue move from the user raises an issue that already exists on the stack, it is taken from the stack and set as the active issue. The dialogue system then proceeds to follow the plan of that issue, until the plan is finished or the user raises another issue with a dialogue move. The information stored by the issue is not forgotten when a change of issue occurs. When an issue is resolved, the system fetches a new issue from the top of the stack and proceeds with its plan.

The information-state extended with the issue-based strategy provides great inspiration of how to provide accommodation features. The information-state is however a complex approach and requires more theoretical knowledge to configure simple dialogues [27, p. 2].

## 4.3 The Topic Forest plan-based structure

In 2001 Xiao-Jun et al. [19] proposed a plan-based dialogue management model that could handle a dialogue with multiple topics called *Topic Forest*. The model consist of one or several topic trees, where each tree starts with a topic node. A topic node represents a dialogue topic and has three types of branches: Primary property (PP), Secondary Property (SP) and Auxiliary property (AP). The leaves

of these branches represent information that is to be stored during the course of the dialogue. The leaf nodes are all connected to a shared information index (SSI) that gathers the information of all the leaves in a shared entity. Fig 4.1 shows a part of a topic forest that models a dialogue for a flight booking application.

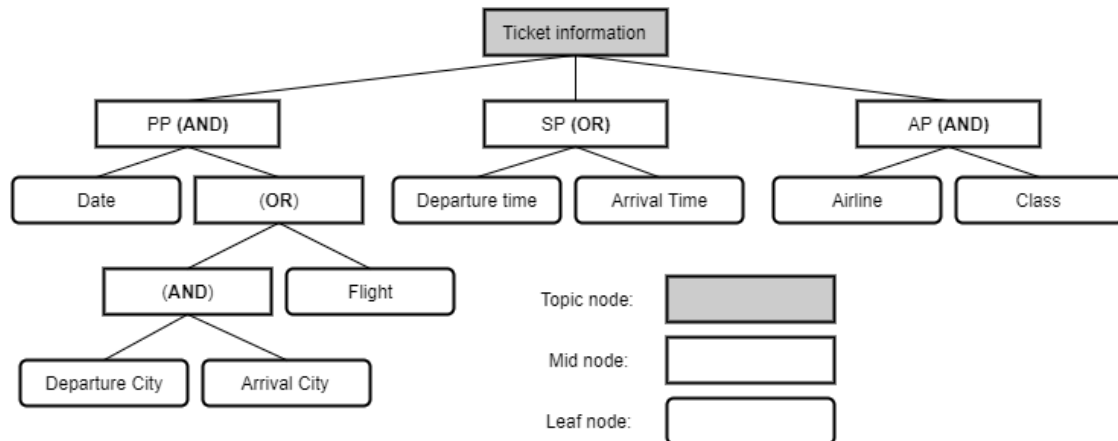


Figure 4.1: Part of a topic forest (similar to [19, p. 2])

The idea behind the different kind of branches is to divide the information into groups of different importance. The leaf nodes under the PP branch represents dominant information that needs to be present before any kind of query is executed. The SP branch represents information that is deemed relevant but not mandatory. The nodes in the AP branch represents optional information that only will be relevant if the user mentions it specifically. As shown in 4.1, the mid nodes have a logical operation that dictates if one or both branches below need to be filled with information.

## 4.4 RavenClaw

Rudnicky et al. [22] have developed a framework *RavenClaw*, which is a plan-based and task-independent dialog management framework. One of the key features of RavenClaw is the separation of the task specification and the conversational strategies. Since the conversational strategy is not dependent on the task specification, a set conversational strategy could be used for various tasks [18, p. 15].

The tasks are arranged in a hierarchy and utilize a frame-based-like approach to acquire information during the dialogue. The frames do not need to be filled in a



---

specific order; the dialogue can jump between different contexts due to the hierarchical structure [21, p. 24].

# Chapter 5

## Proposal of a frame-based dialogue management strategy

This chapter describes the proposed strategy developed inspired by the work reviewed in Chapter 4. We have developed a simple dialogue management design for an information-seeking dialogue system that utilizes the frame-based approach of dialogue management. The solution addresses known weaknesses of a trivial frame-based system. The design is domain-independent. We wanted to achieve these goals while maintaining a simple design such that developers with minimal knowledge in the field can easily model new dialogues.

### 5.1 Weaknesses of frame-based systems

Trivial frame-based systems do not support handling of multiple topics. One solution is to have several frames where each frames covers one topic, but these systems usually do not support any kind of accommodation. They require a frame to be finished before moving on to the next frame. This is where the issue-based dialogue strategy have an advantage since these systems can jump between issues that are not yet resolved [32, p. 15].

The necessity of the slots in the frame should depend on context. The system should be able to adapt if the triggered system action of a finished frame does not present a proper result. It should also be able to avoid asking for redundant information in

situations where information in certain slots do not affect the result of the system action.

Frame-based approaches do not have any standardized way to answer *wh-questions*. A *wh*-question is a sentence that starts with *what*, *where* or *when* [30, p. 26]. An example could be: Where do you want to go?. The proposal must have an efficient way to model the answering to these questions.

To summarize, to address the weaknesses of the trivial frame-based approach, the proposed strategy should fulfill the following:

- The strategy must be able to handle multiple topics.
- Dynamically check to see if slots are necessary to avoid improper system action results and to avoid having the system asking for unneeded information.
- Provide accommodation features: Be able to jump between contexts and remember input over topic boundaries.
- Able to answer user *wh*-questions.

## 5.2 The frame

We will start by introducing the design of the frame. Each possible topic of the dialogue is represented by a frame of its own. Just as in the standard approach, each slot in the frame defines one or more prompts that the system prints to ask the user for the missing information. The frame has three different types of slots, namely *primary*, *secondary* and *auxiliary*. Inspired by the Topic Forest approach described in Section 4.3, the three types of slots represents their augmented priority. Primary slots store mandatory information and the frame will not be complete until all of these slots are filled. The system will initiate and print suitable prompts for each of these slots that are currently empty. The secondary slots will be initiated by the system if the information from the primary slots alone is not able to generate a proper result. One example is a case where the information in the primary slots generates a query search that returns one hundred items and has to be narrowed down. The auxiliary slots are optional and will only be relevant if the user mentions them specifically. If the user does not mention them, neither will the system.

Each slot can also have a defined confirmation prompt that is printed if the system is uncertain about the input or if the information of a particular slot is of more importance. This is to maintain a common ground of knowledge with the user. The system should ask for a confirmation that lists the relevant acquired information when the frame is complete.

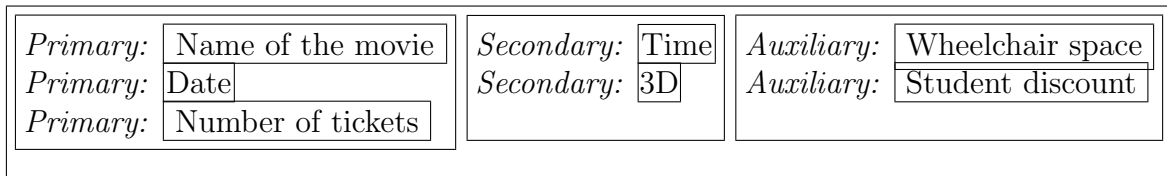


Figure 5.1: A simple frame that covers the topic of reserving tickets for a movie

An example of a frame is shown in fig 5.1. This frame covers a topic in a very simple movie ticket reservation application where the objective is to identify and reserve tickets to a movie. The primary slots cover the name and date of the movie, as well as the number of tickets. If the name and date of the movie results in more than one possibility, the system will print suitable prompts for the secondary slots, namely time or 3D to narrow the search. These are of course also filled if the user mentions them by his/her own initiative before the primary slots are filled. The auxiliary slots cover the options of having extra space due to a wheelchair and student discounts.

### 5.3 Dialogue state

Similar to the Information-state approach described in Section 4.2, the design will utilize an universal state that manages the overall flow of the conversation and handles all the possible topics. As mentioned in Section 5.2, each possible topic is represented by a dialogue frame. The dialogue state contains a *topic-queue* that stores the topics in the order the system should initiate them. The topic that should be discussed first is to be placed on the top of the queue. If a user utterance matches a different topic different from the one currently active, the state will switch to the new topic and set it as active. This will not however, change the position of the topics in the queue and the filled slots will not be cleared following a change of topic. This behavior will provide accommodation in a similar fashion as the issue-based approach described in Section 4.2.1. Instead of issues with plans, our proposal treats a topic as an issue and the frame represents the plan.

When a topic is finished it is removed from the queue and the system fetches the next topic that is on the top of the queue to be set as active. This could be a new topic that has not yet been introduced or a topic that is partially finished due to a change of topics. A topic does not have to be in the queue at the start of the dialogue. If the utterance matches a topic that is not present in the queue, the topic is added and set as active.

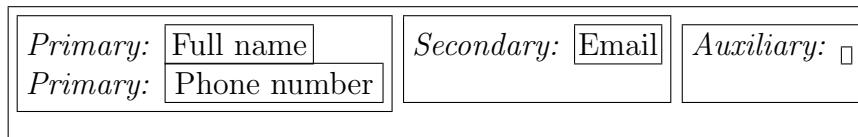


Figure 5.2: A simple frame that covers the topic of acquiring of personal information.

An example of how the state operates can be shown by introducing a second topic to the movie ticket reservation application from the previous section. The second topic covers personal information and the frame is shown in 5.2. A good way to model the dialogue could be to have both topics placed in the queue, with the topic of reserving a movie on top. The dialogue will start with the system initiates the movie reservation topic. If the frame is not complete and the user write something similar to "and my name is...", the topic of personal information will activate. The system then proceeds to focus on filling the slots of the new active frame until it is complete or another topic change occurs. We introduce a third topic that covers buying snacks for the film and is not in the queue to begin with. If the user write something similar to "..and i want some chips", the topic will be set as active and be placed at the end of the topic queue.

```
S: Hello, what movie do you want to see? [Movie reservation topic]
U: My name is Sven.
S: Okay, your name is Sven. What is your email? [Personal information topic]
U: I want to buy some chips.
S: Okay, you want to buy chips. What do you want to drink? [Snacks topic]
```

Figure 5.3: Example of how the system can switch between different topics.

## 5.4 Information merging

Two different topics may share the same partial information. One example is a travel application with one topic for booking flights and one for booking a hotel.

Both the destination city of the flight and the location city of the hotel will share the same information. In this case the slots can be designed as pointers to a shared index where both slots can change and acquire the same information.

## 5.5 Questions

Frames can be designed to handle questions by modeling the different kind of wh-questions as topics of their own. A frame can consist of only auxiliary slots. Since such frame has no primary or secondary slots, the system action triggers as soon as the topic is brought up. This means that the topic will never be placed in the topic-queue since it is deemed finished as soon as it is brought up. If an utterance brings up the topic but do not fill any of the slots with information, a standardized response could be prompted. If the utterance do fill any of the slot the system action can use them to give a more specific answer.

An example could be shown by introducing a new topic in the movie reservation application that is to answer questions about the available films. The topic is named *WhatMovies?* and the frame consists of two auxiliary slots; a date and the name of a movie. An utterance like "What movies are showing tomorrow?", would be identified as the topic *WhatMovies?* and "tomorrow" would be extracted as information for the date-slot. With this information the system can list the movies that are showing tomorrow. The utterance "When is spider-man showing on Saturday?" would result in listing the timestamps when spider-man is showing upcoming Saturday, hence the utterance fills both slots.

## 5.6 Summary

A frame-based dialogue management design has been presented. It is domain-independent and is able to handle dialogues with multiple topics. The design provides accommodation features in a similar fashion as the issue-based strategy where the frame represents the plans of the topics. It utilizes a universal state where the developer can specify a general dialogue flow by putting initial topics in the topic-queue. Inspired by the Topic-Forest approach, the design utilizes augmented

priorities to achieve a dynamic use of slots. The design of the frame makes it possible to handle wh-questions in an easy way.

By utilizing frames the design aims to be simple enough for novice users to model their dialogues. The design is flexible in the sense that the developer can utilize the augmented priorities of the slots in different ways.

# Chapter 6

## LUIS.ai

*LUIS.ai*, or *Language Understanding Intelligent Services*, is a web-based tool for sentence analysis that can be trained for a particular domain. It is developed by Microsoft and strives to be the most comprehensive and easy-to-use tool for language understanding for applications and devices [8].

### 6.1 Natural language understanding

LUIS utilizes machine learning based methods to process and analyze textual utterances. To be able to apply machine learning methods, LUIS breaks down the utterances into smaller pieces called tokens [6], by a process called tokenization [7].

One of the major benefits of a LUIS application is that it employs active learning to improve itself after deployment. LUIS identifies what utterances it finds difficult to classify and lets the user label them [6].

#### 6.1.1 Intents and entities

LUIS represents the extracted semantic information of an utterance as *intents* and *entities*. An intent represents what the user wants to achieve with the utterance, the overall goal [11, p. 30]. An intent of the utterance "*I want to go to Germany!*" could be that the user wants to book a flight. The intent is heavily influenced by the

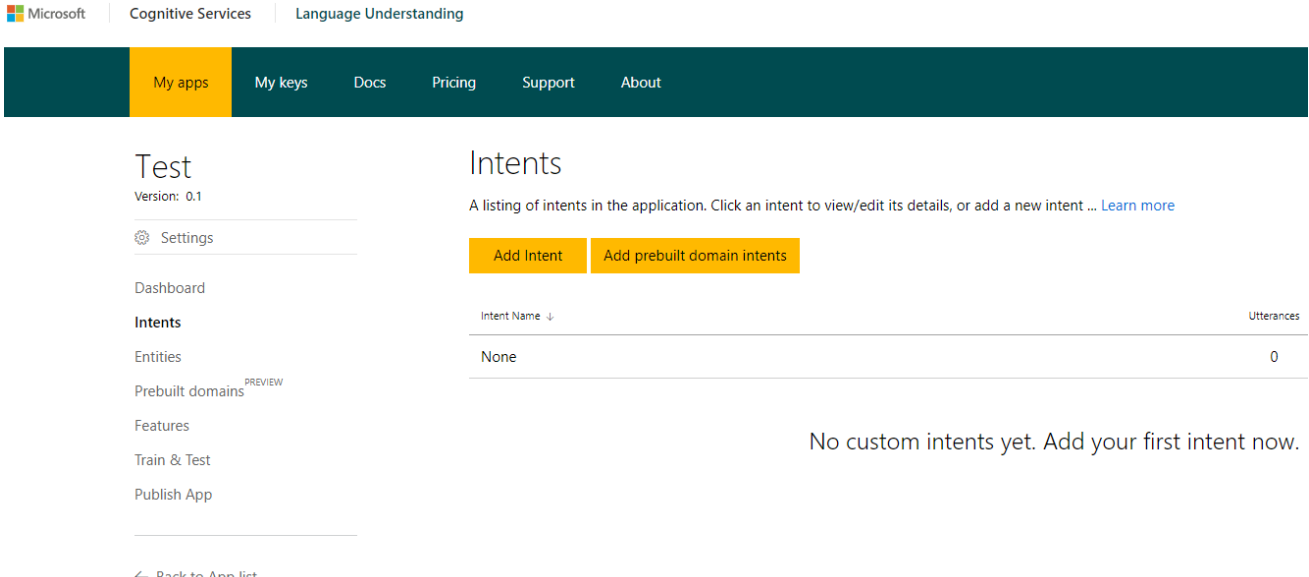


context and the domain. The same utterance could also mean that the user needs a car or buy a train ticket.

Every important fact of a sentence is mapped to an entity. The intent represents the purpose of an utterance while the entities represents the specific details and information. Examples of entities are dates, locations or specific products. In the previously mentioned utterance, "I want to go to Germany!", the intent is to book a flight while the entity is Germany.

LUIS lets the user create its own sets of intents and entities. LUIS presents a decimal value between 0 and 1 indicating how sure it is that the identified intent is the actual intent of an utterance. This is to make it easier for developers to handle unexpected events and misunderstandings.

## 6.2 Creating and managing a LUIS.ai application



The screenshot shows the Microsoft LUIS.ai application management interface. The top navigation bar includes the Microsoft logo, 'Cognitive Services', and 'Language Understanding'. Below this is a dark green navigation bar with 'My apps' highlighted in yellow, along with 'My keys', 'Docs', 'Pricing', 'Support', and 'About'. The main content area is split into two columns. The left column shows the application details for 'Test' (Version: 0.1) and a sidebar with options like 'Settings', 'Dashboard', 'Intents', 'Entities', 'Prebuilt domains', 'Features', 'Train & Test', and 'Publish App'. The right column is titled 'Intents' and contains a table listing the current intents. The table has two columns: 'Intent Name' and 'Utterances'. There is one row with 'None' in the 'Intent Name' column and '0' in the 'Utterances' column. Above the table are two buttons: 'Add Intent' and 'Add prebuilt domain intents'. Below the table is a message: 'No custom intents yet. Add your first intent now.'

Intent Name	Utterances
None	0

Figure 6.1: A view listing the current intents.

Microsoft provides a graphical web interface for creating and training an application. When a new application is created there is one intent present, namely the *None* intent. The *None* intent fires if none of the added intents were recognized in the utterance. As shown in fig 6.1 there are two choices when creating an intent; a regular intent or a prebuilt domain intent. A prebuilt intent is pretrained with

10 utterances in a popular domain such as reserving tickets or booking flights. A prebuilt intent is a good choice for the novice user to learn how to use the service. A regular intent requires a unique name and has no prior training.

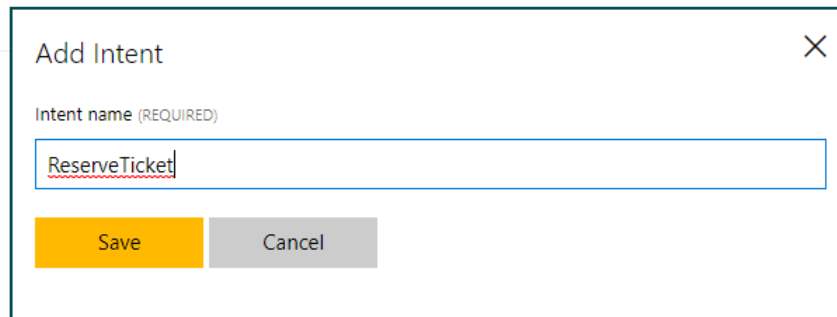


Figure 6.2: Adding an intent with a unique identifier.

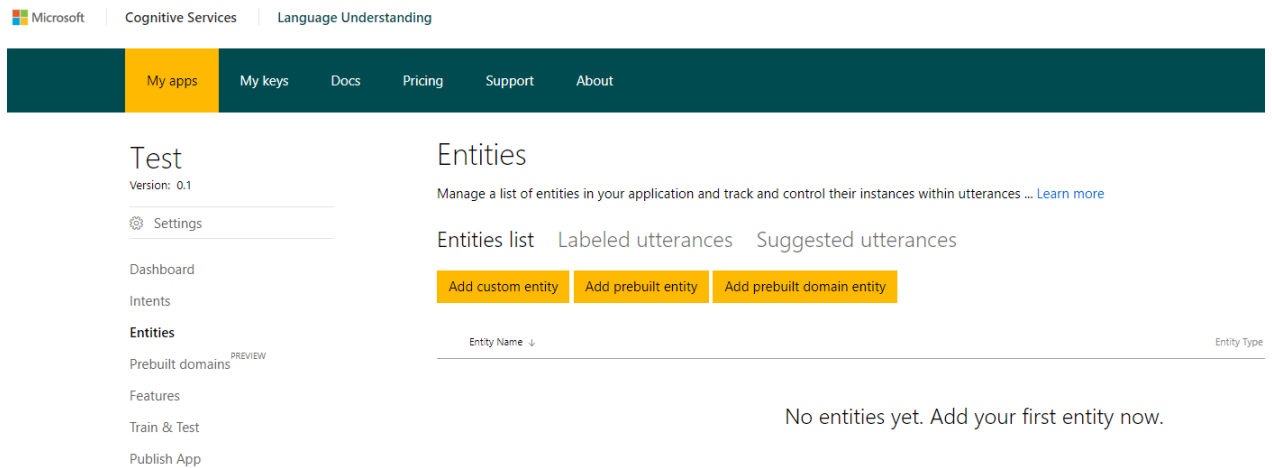


Figure 6.3: A view listing the current entities and the different entity options.

There are no entities present when a new application is created. As shown in Figure 6.3 there are three different kind of entities to choose from. The first kind is a custom entity, which the developer names by an identifier. There are two types of entities. A simple entity is a freetext entity where the value can be any string the developer teaches the system. For a list entity, the user can define the values that the entity may take, e.g. the names of cities or countries. The developer can also choose from a set of prebuilt entities. These are already learned by the engine to identify different type of values commonly used by information seeking dialogue systems, such as email and phone numbers.

Training is done by providing example utterances for each intent of the application.

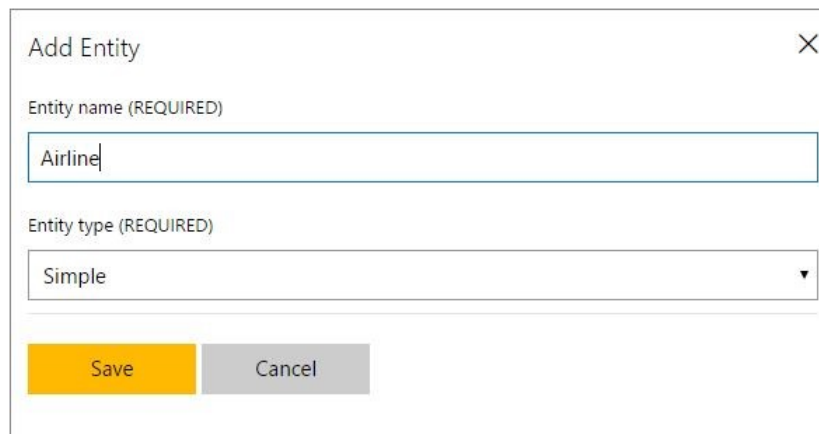


Figure 6.4: An example of when creating a custom simple entity.

Figure 6.5 shows an example where the utterance *"I want to fly with sas"* has been provided for the intent ReserveTicket. When the utterance has been provided the developer can mark the word or words that should be mapped to specific entities. In our example the string *"sas"* is marked as an airline entity.

## ReserveTicket

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Utterances   Entities in use   Suggested utterances

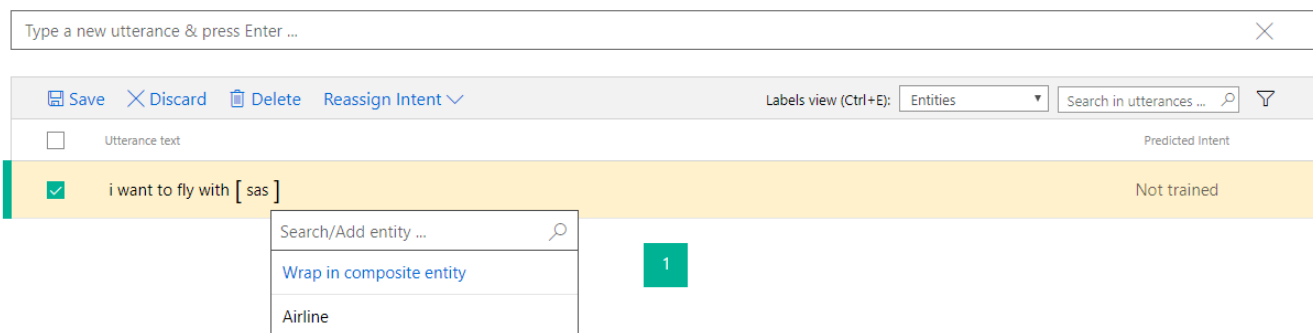


Figure 6.5: The interface when training an application.

### 6.2.1 Training and test tool

The train and test tool provides a useful interface to evaluate the behavior of the application. The developer can type an utterance and see how the application

understands it. The identified entities are shown directly in the utterance and the top scoring intent is shown in the result window to the right as shown in Figure 6.6. The resulting intent is also imbued with the certainty of the system.

## Test your application

Use this tool to test the current and published versions of your application, to check if you are progressing on the right track ... [Learn more](#)

**Train Application**

Last train: Jul 30, 2017, 1:07:22 AM | Last publish: **Not published yet.**

Interactive Testing   Batch Testing

The screenshot shows the LUIS.ai training and test tool interface. At the top, there is a 'Train Application' button and status information: 'Last train: Jul 30, 2017, 1:07:22 AM | Last publish: Not published yet.' Below this, there are two tabs: 'Interactive Testing' (selected) and 'Batch Testing'. The main area is split into two panels. The left panel has a text input field with the placeholder 'type a test utterance & press Enter' and a blue arrow button. The input field contains the text 'i want to fly with [\$Airline ]'. The right panel is titled 'Current version results' and shows the following information: 'Top scoring intent: ReserveTicket (1)' and 'Other intents: None (0.05)'. There is also a 'Labels view (Ctrl+E)' dropdown menu set to 'Entities' and a 'Reset console' link.

Figure 6.6: The training and test tool.

## 6.3 Integrating LUIS.ai with a system

The published application serves as an HTTP endpoint utilizing a REST API. The communication is done using JSON packages. For the application to accept the HTTP request, the *subscription key* and *application ID* must be attached as headers. The subscription key is the identifier of the developers Microsoft account and provides access to the API. The application ID is the identifier of the specific application.

A prediction of an utterance is simply made by sending a GET request to the application with the utterance attached. The sample JSON response is shown in Figure 6.7. The top scoring intent is set in its own element and all the identified entities are present in the entities list.

```
{
  "query": "My airline is SAS",
  "topScoringIntent": {
    "intent": "ReserveTicket",
    "score": 0.93267872
  },
  "intents": [
    {
      "intent": "ReserveTicket",
      "score": 0.93267872
    },
    {
      "intent": "None",
      "score": 0.06732128
    }
  ],
  "entities": [
    {
      "entity": "SAS",
      "type": "Airline",
      "score": 0.9303394
    }
  ]
}
```

Figure 6.7: A sample JSON response of the prediction of the utterance "My airline is SAS".

## 6.4 Limitations

There are always drawbacks with depending on external systems and services. A dialogue system relying on web-based language tools will render useless if the service of the tools are down or under maintenance. Alan Nichol [9] believes that the future of web-based language tools lies in prototyping but not in industrial use. Companies want to be independent and not having their fate decided by a third party. More open source tools will probably be released in the future by big players.

LUIS.ai provides 1000 free endpoint hits a month. If the developer wishes to use it more extensively, he or she has to pay a monthly fee.

There is a set limit for the number of intent and entities the developer can specify. An application may have at most 80 intents and 30 entities. These constraints are

set to reduce the chance of over-fitting. A way for the developer to work around this is using multiple applications in parallel [10].

# Chapter 7

## Implementation of the prototype system

This chapter describes the implemented proof of concept prototype dialogue system. The dialogue manager of the prototype utilizes the proposed dialogue management strategy described in Chapter 5. The system the users to configure their own dialogues by providing an XML-based dialogue description. It is required by the user to provide a trained and published LUIS.ai application for language understanding. The system can be integrated with a Facebook-page by utilizing the Facebook messenger application.

### 7.1 Architecture

An overview of the system architecture is shown in Figure 7.1. The prototype system consists of four different modules. The *Messenger Communicator* contains the endpoint that receives and handles the Facebook messenger webhook requests. The *LUIS.ai Communicator* requests and reads the responses of the provided LUIS.ai application. The *DialogueManager* handles the dialogue management of the system and maintains the dialogue state.

The webhook endpoint of the Messenger-communicator is implemented by utilizing the python Flask Micro framework. When a user utterance is received through the webhook it is sent via the LUIS-Communicator module to the LUIS.ai application.

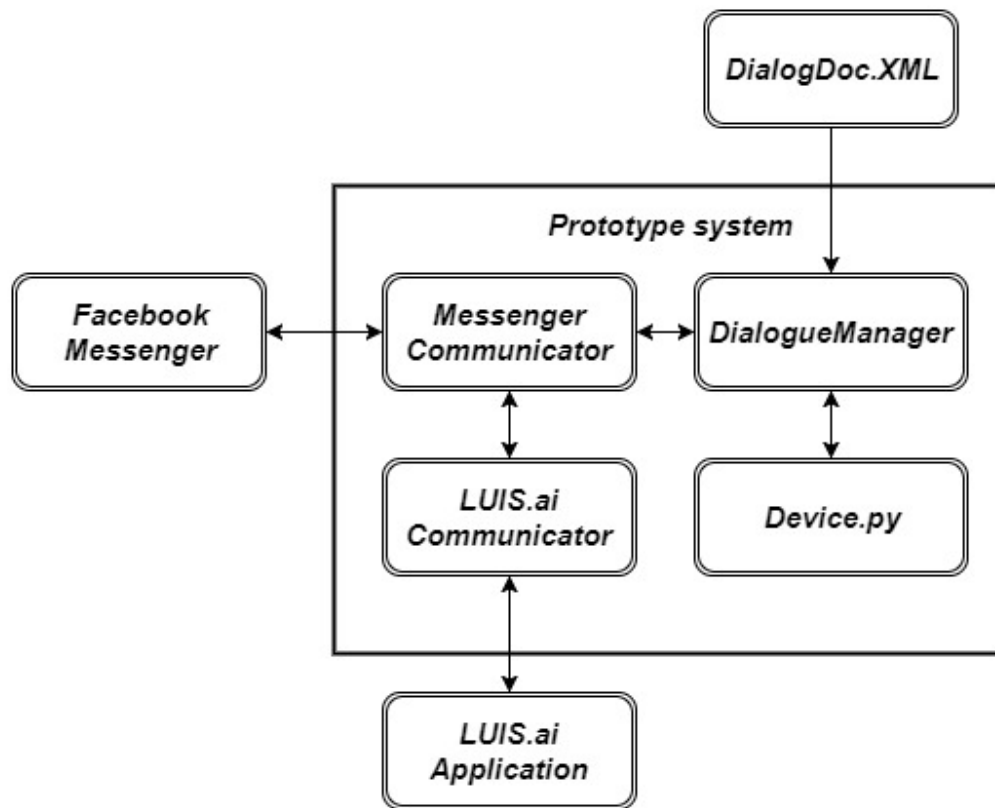


Figure 7.1: Architecture overview

The LUIS-Communicator then receives a response with the extracted intent and entities of the utterance. The intent and entities are then passed on to the DialogueManager that updates the dialogue state and returns a textual response to be presented to the user.

The dialogue manager is very loosely coupled to the rest of the system such that it can be extracted and integrated to another system. It parses the DialogDoc.XML when instantiated. A new instance of the DialogueManager is instantiated for each unique user.

## 7.2 Dialogue document

In the prototype system we propose a way to configure the dialogue structure according to an XML-based dialogue description. The choice of XML is motivated with the positive results obtained by the dialogue system NADIA [27] and the wide



usage of VoiceXML [18]. The document covers the whole dialogue between the user and the system. It defines all possible topics and the the required information to perform the system actions. A description of a simple dialogue in XML is shown in Figure 7.2.

```

<Dialogue>
  <InitGreeting>Hello! You are now talking to a chatbot!</InitGreeting>
  <EndPrompt>That was all i needed!</EndPrompt>
  <UnkownTopic>I ’m sorry I did not that!</UnkownTopic>
  <Frame Topic=’ReserveTicket’>
    <Slot Entity=’MovieName’>
      <Prompt lang=’en’>What movie do you want to see?</Prompt>
      <Type>primary</Type>
      <Confirmation lang=’en’>you want to see [value]</Confirmation>
    </Slot>
    <Slot Entity=’NumberOfTickets’>
      <Prompt lang=’en’>How many tickets do you want?</Prompt>
      <Type>primary</Type>
      <Confirmation lang=’en’>you want [value] tickets</Confirmation>
    </Slot>
  </Frame>
  <Frame Topic=’PersonalInfo’>
    <Slot Entity=’FullName’>
      <Prompt lang=’en’>What is your full name?</Prompt>
      <Type>primary</Type>
      <Confirmation lang=’en’>your name is [value]</Confirmation>
    </Slot>
  </Frame>
  <TUD>
    <Topic>ReserveTicket</Topic>
    <Topic>PersonalInfo</Topic>
  </TUD>
</Dialogue>

```

Figure 7.2: An example of a specified simple dialogue for reserving movie tickets. ”[value]” can be used in the Confirmation element to present the obtained information to the user.

The document consists of the *Dialogue* root element and five different child elements. The first three child-elements have textual values as they are prompts used by the system during the interaction. The *InitGreeting* and *EndPrompt* elements specifies the initial greeting and the goodbye message. They are presented by the system to the user at the start and the end of the dialogue respectively. The *UnkownTopic*

element specifies textual response from the system if no relevant information can be extracted from the given user utterance.

The *Frame* element has two types of child-elements; the unique name of the frame topic of and the slots of the frame. The frame contains one or more slots where each slot element has a unique specified *Entity*. The slot also needs a specified *Type*, which indicates the importance of the slot, see Section 5.2 for more information. The type is set by one of the lower case values; *primary*, *secondary* and *auxiliary*. The slot can also have a *Prompt* and a *Confirmation* element. The Prompt element describes the textual act that the system presents to help the user filling the corresponding slot with information. The Confirmation element is presented when the system has filled the slot with information. While these elements are not mandatory, they are useful for grounding and to navigate the dialogue in the right direction.

The *TUD* element represents the queue of topics that are to be brought up by the system during the interaction. It only has one type of child-element which is the name of the topics that are to be placed in the queue. Every topic specified in the queue must have a matching frame element.

### 7.2.1 Model with LUIS.ai

This dialogue system utilizes LUIS.ai for sentence analysis. As mentioned in chapter 6, a LUIS.ai application structures the semantic information as *intents* and *entities*. We interpret the intents of the LUIS.ai application as the possible topics of the dialogue and the entities as the slot entities.

The dialogue document in Figure 7.2 shows a small LUIS.ai application with two intents, namely ReserveTicket and PersonalInfo. Furthermore, it describes a total of three different entities; MovieName, NumberOfTickets and FullName.

## 7.3 Device.py

A major feature of the dialogue system is the *Device.py* file. It has to be implemented by the developer and is responsible for determining if a topic is finished or not. The file can be configured by the developer to have the system communicate with external

tools and systems such as databases and web services. The developer can also utilize it to validate the input of the frames; if an input is not deemed valid the slot can be emptied and the system will proceed the dialogue to reacquire information for that slot.

The python file must define a class that extends an abstract class called *AbstractDevice*. Doing so, it inherits one abstract method called *frameAction*. The method is called by the system every dialogue turn when all the primary slots of the active frame are filled. The method has two input parameters; the frame topic in the format of a string and a dictionary of the filled slot entities.

The *frameAction* method must return an instance of the class *SystemActionResult*. *SystemActionResult* is a class that specifies how the system should behave following a filled frame. The developer can set a prompt that should be presented as a response to the user and specify if any slots of the frame should be emptied because of invalid input.

*SystemActionResult* takes one out of three defined parameters in the constructor, namely *complete*, *continue* or *error*. The *complete* value is set as the parameter if the information of the frame in question is sufficient. The topic is then deemed to be finished and the topic is removed from the topic queue. The *continue* value is set as the parameter if the result of the information stored in the frame is not enough for the application and the system needs to continue pursuing the topic. Examples of this could be when a slot has obtained an invalid value and needs to be emptied, or when a query search using the frame information in a database results in multiple rows. The *error* value is set as the constructor parameter if a serious error has occurred with the external systems and the dialogue system should terminate.

## 7.4 Summary

This chapter has described the implemented prototype that is to be used for the evaluation of the dialogue management strategy proposed in chapter 5 together with LUIS.ai as a natural language understanding module. A developer that intends to use the system and configure a new chatbot should at have at least basic knowledge of Python and XML.

# Chapter 8

## Evaluation

This chapter describes the evaluation of the proposed and developed dialogue management strategy and of LUIS.ai as a natural language understanding module. First we present the method of how the evaluation is performed, followed by the results.

### 8.1 Evaluation method

Williams et al. [24, p. 1] claim that a developer can train and publish a working natural language understanding module using LUIS.ai within minutes. To test this claim we obtained a small training set for a chosen simple domain by utilizing the *Wizard of Oz* method. Wizard of Oz is a method to evaluate unimplemented software by having a person simulating the behavior of the system [25, p. 277]. In the presented experiments the author of this thesis acted as a wizard and thus simulated the behavior of the proposed dialogue model. A LUIS.ai application was trained by using roughly 70% of the recorded interactions, and evaluation was done by testing the remaining of the examples using the LUIS.ai training interface mentioned in Section 6.2.1. This tested if the sample of interactions was enough to fire the correct intents and entities. This also served as a good indicator of the effectiveness of the proposed dialogue model.

The trained language understanding application was then integrated with the implemented prototype dialogue system that utilizes the proposed dialogue model. The system was tested with scenarios listed in *Talkamatic Dialogue Strategies examples*

[33]. These examples covers a number of dialogue functionalities together with a rough idea of how common it is that they are provided by commercial dialogue systems. Talkamatic AB is a company that develops conversational interfaces and works with researchers from the computational linguistic department at Gothenburg University [34]. The Talkamatic dialogue manager was described briefly in Section 4.2.

The dialogue system was further evaluated by user-testing in terms of dialogue cost, task success and interaction quality as stated in Section 2.5. The proposed dialogue model was then evaluated by having participants modeling a dialogue from a trained LUIS.ai application. The result was evaluated by semi-structured interviews.

## 8.2 Evaluation results

A simple domain for reserving movie tickets was chosen as evaluation. The dialogue document modeling the desired information can be found in Appendix A. The LUIS.ai application used simple entities for the information of MovieName and FullName, while prebuilt entities were utilized for the information of numbers, email and dates.

### 8.2.1 Wizard of Oz test

The Wizard of Oz method provided 21 interactions with a total of 148 utterances. 16 of these interactions were used to train the created LUIS.ai application. The rest of the interactions were used to observe if the LUIS.ai application would fire the correct intents and entities.

The results showed that 55.7% of the utterances fired the correct intent. This result may sound discouraging but closer analysis revealed that the 90% of one-word utterances failed and utterances containing more than three words had a 90% success rate. Evidently, LUIS.ai is much more efficient in identifying the intent of the utterance if it contains more words.

## 8.2.2 Talkamatic Dialogue Strategies examples

The Talkamatic Dialogue Strategies examples were tested by integrating the trained LUIS.ai application from the previous section with the developed prototype system. Below we present dialogue examples provided by Talkamatic covering each functionality feature followed by an example of the prototype system covering the same functionality in the movie ticket reserving domain.

### Over-answering

Over-answering is the ability to acknowledge more information than was initially requested by the system. This was not a problem since it is a standard feature of the frame-based approach. This is according to Talkamatic a feature that is provided by several but not all other commercial solutions.

```
U (User): Make a call  
S (System): OK, who do you want to call?  
U: John's mobile [mobile not requested by the system]  
S: OK, placing the call
```

Figure 8.1: Example of over-answering provided by Talkamatic

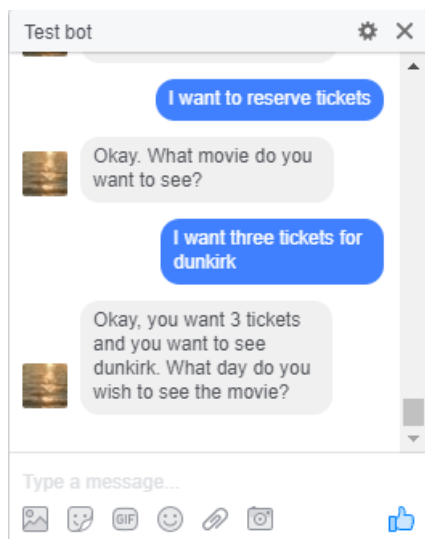


Figure 8.2: Example of the prototype system handling over-answering

## Other-answering

Other-answering is the ability to acknowledge information that is different than the actually requested information. This is something state-machine architectures are unable to do. For this example, the prototype system has `PersonInfo` as the active topic and requests the name of the user. The user presents his email instead and the system acknowledges the information. According to Talkamatic, this functionality is covered by a few other commercial solutions.

```
U: Make a call
S: Who do you want to call?
U: The mobile number [not requested by the system]
S: OK. Who do you want to call?
U: John
S: Calling John's mobile
```

Figure 8.3: Example of other-answering provided by Talkamatic

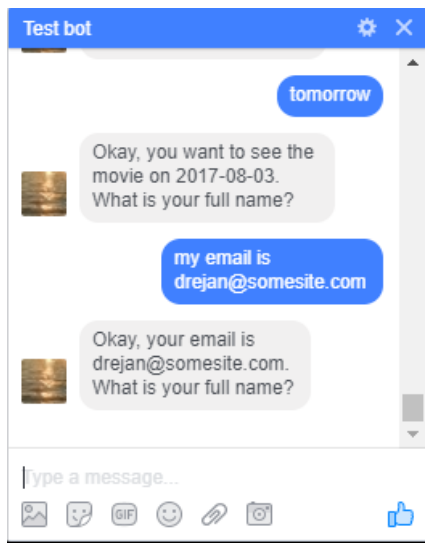


Figure 8.4: Example of the prototype system handling other-answering

## Correcting the system

A dialogue system providing this functionality makes it possible for the user to correct mistakes or incorrect input. The prototype system has no direct way to clear a slot of information by an utterance but the user is able to provide new information as long the topic is not finished. According to Talkamatic, this functionality is provided by very few other commercial solutions.

S: Who do you want to call?  
U: Call John  
S: John's home or mobile number?  
U: Not John, Steve!  
S: OK, Steve's home or mobile number?

Figure 8.5: Example of a system correcting an error provided by Talkamatic

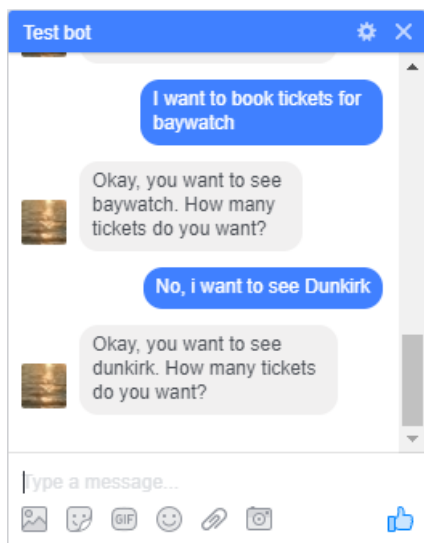


Figure 8.6: Example of the prototype system when the user is correcting a mistake



## Topic shifts

A dialogue system providing this functionality makes it possible for the user to jump between different topics or tasks before they are finished. In the second example of the prototype system presented in Figure 8.9 the system jumps between three topics. The WhatMovies topic is finished and removed from the queue as soon it is presented since it is designed to only answer primitive wh-questions. The system then proceeds with the previous topic. According to Talkamatic, this functionality is provided by extremely few other solutions.

U: Call John  
S: What number? Work, home or mobile?  
U: What time is it? [Time and date domain]  
S: 10 pm. Returning to calling. What number do you want to call?  
U: The home number

Figure 8.7: Example of a topic shift provided by Talkamatic

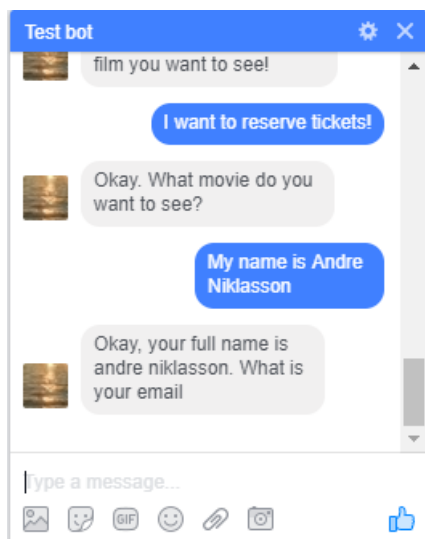


Figure 8.8: Example of a topic shift by the prototype system

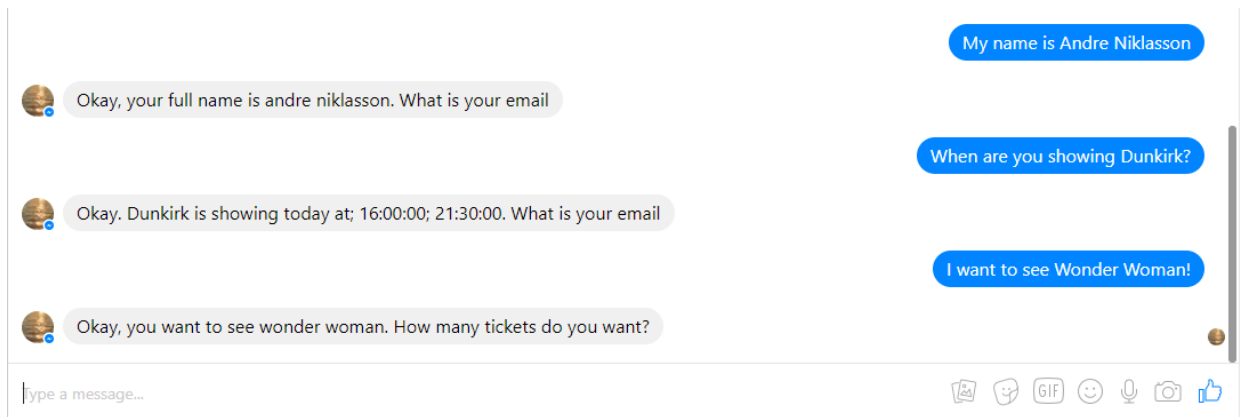


Figure 8.9: Example of several topic shifts by the prototype system

## Task recognition

The functionality of task recognition is the ability to recognize a task or topic by a single word that states information. The prototype system has no way of doing this by its dialogue manager alone, this functionality depends on the natural language understanding module. As stated in Section 8.2.1, LUIS.ai was found to have a hard time to understand one-word utterances. The prototype system does therefore not support the functionality reliably.

```
U: 7 pm
S: Do you want add to the calendar or set the alarm?
U: Set the alarm
S: You alarm is set for 7pm.
```

Figure 8.10: Example of task recognition provided by Talkamatic

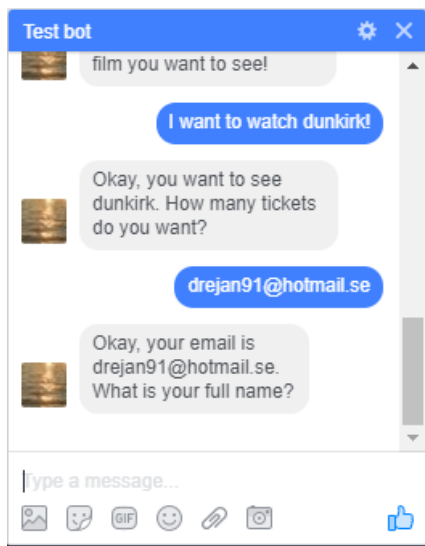


Figure 8.11: Example of the prototype system successfully performs task recognition

### 8.2.3 Dialogue cost and task success

The prototype system was evaluated by having four users conversing with the dialogue system to reserve tickets to a movie. All users were given two tries to complete the overall task. This was to see if prior knowledge of what information the system required would speed up the process by having the users utilizing more advanced dialogue functionalities.

The modeled domain is the same as in the previous sections. The dialogue document can be found in Appendix A. If the user were to fill one mandatory slot of information each turn, the dialogue cost would result in five turns.

	Number of dialogue turns	Task succeeded?
User1	10	Yes
User2	9	Yes
User3	<i>N/A</i>	No
User4	15	Yes

Figure 8.12: The results of the first try.

	Number of dialogue turns	Task succeeded?
User1	4	Yes
User2	6	Yes
User3	6	Yes
User4	6	Yes

Figure 8.13: The results of the second try.

The language understanding posed a problem in the first test. The users were using a lot of one word utterances to provide the information. This led to a lot of wasted dialogue turns and one of the users got frustrated and gave up. This motivated a modification to the dialogue document by adding extra information to the UnkownTopic element in the dialogue document stating that the user should use whole sentences.

The second test results shows a major decrease in the number of dialogue turns needed when the users got prior knowledge of the information required. The users did not need to ask the system questions about what movies that were showing and they utilized over-answering functionalities. When using whole sentences, the language understanding posed no longer a problem and **User3** was able to complete the task.

#### 8.2.4 Dialogue modeling

The dialogue modeling was evaluated by having two developers, both with a bachelor degree in computer science, to model a dialogue using the XML dialogue document. They did not develop any further functionalities using Device.py due to time constraints. They were both provided a trained LUIS.ai application with three intents and five entities. Both participants spent one hour each and were able to ask the author simple questions when problems occurred.

The overall idea of the frame-based approach was easy for both participants to understand. However, due to being beginners in the field of dialogue management, the different type of slots in the frames made no sense to them in the start. They both experienced a fast learning curve for how the confirmation prompts would be presented by the system. Both participants were quickly able to configure working

dialogue systems. One of the participants stated during the development that the dialogue model felt designed to fit the functionality of LUIS.ai.

The dialogue model received criticism because the dialogue felt non-alive with only one specified response to be presented when the system did not understand an utterance. Since neither of the participants designed a topic to handle primitive questions, they both found the topic-queue unnecessary. The reason was that they thought the system should just bring up every specified topic. Overall both participants felt that the system had a good design that was easy to learn, but both stated that they had nothing to compare it with.

# Chapter 9

## Conclusion

This chapter summarizes what goals we have achieved and what we have learned. We end the conclusion with a brief look at future work.

### 9.1 Purpose

LUIS.ai proved to be a very useful tool for rapid creations of dialogue systems. An application does not require a lot of training to be a sufficient natural language understanding module. We have discussed the drawbacks and limitations of using external tools. LUIS.ai is tool that still is the development phase, the API and functionality changes continuously.

A dialogue model that supports functionalities of competitive commercial dialogue systems has been presented. It is easy to configure for non-experts. By this combination of LUIS.ai with improved versions of state-of-the-art techniques for dialogue management, the set goal of the project is seen as fulfilled.

We have, inspired by related work, proposed solutions to how some weaknesses of the trivial frame-based approach can be addressed. We have achieved accommodation features and the handling of multiple topics by representing each topic as a frame and by using a universal dialogue state. The state keeps track of the current topic and stores the rest of the topics that are to be discussed in a queue. The slots of the frame have augmented priorities to dynamically check their necessity.

## 9.2 Future work

The developed dialogue model shows promising performance but should be evaluated further to identify possible ways of improvements. One of the biggest observed weaknesses of the prototype system is the inability to handle one word utterances in a reliable manner. The system is fully dependent on the LUIS.ai application for language understanding. A solution to address the problem could be to equip the prototype with a simple keyword or pattern matcher mechanism to be able to handle one word utterances when the LUIS.ai application fails.

Even if no participant mentioned it specifically during the interviews, writing code in XML can be quite tedious. A graphical interface to configure the dialogue document could improve the experience of designing dialogues.

# Bibliography

- [1] Pierre Lison, *Structured Probabilistic Modelling for Dialogue Management*, PhD Thesis, 2013.
- [2] Harvey Sacks, Emanuel A Schegloff, Gail Jefferson. *A simplest systematics for the organization of turn-taking for conversation*, Language, 1974.
- [3] Gabriel Skantze , *Error handling in spoken dialogue systems*, PhD Thesis, 2007.
- [4] Michael F. McTear, *Spoken Dialogue Technology: Enabling the Conversational User Interface*, 2002.
- [5] James F. Allen, Donna K. Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, Amanda Stent, *Towards Conversational Human-Computer Interaction*, 2001.
- [6] Microsoft Cognitive Services, <https://www.microsoft.com/cognitive-services/en-us/luis-api/documentation/home#Localization>, visited 06/04/2017.
- [7] The Natural Language Processing Group at Stanford University, <https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html>, visited 06/04/2017.
- [8] Language Understanding Intelligent Service (LUIS), <https://www.luis.ai/about>, visited 06/04/2017.
- [9] Alan Nichol, PhD in machine learning from the University of Cambridge, <https://www.developereconomics.com/nlp-wit-luis-api-ai>, visited 07/05/2017.
- [10] Jayme M Perlman, Denise Mak, Robert standefer, Den Delimarsky, *Troubleshooting general problems*, <https://docs.microsoft.com/en-us/bot-framework/troubleshoot-general-problems>, visited 07/05/2017.



- [11] Roland Meertens, *A Scalable Mixed Initiative Dialogue Manager*, Master's Thesis, 2015.
- [12] Cheongjae Lee, Sangkeun Jung, Kyungduk Kim, Donghyeon Lee, Gary Geunbae Lee, *Recent Approaches to Dialog Management for Spoken Dialog Systems*, 2010.
- [13] Steve Young, Fellow, IEEE, Milica Gasic, Member, IEEE, Blaise Thomson, Member, IEEE, Jason D Williams, Member, IEEE, *POMDP-based Statistical Spoken Dialogue Systems: a Review*, 2013.
- [14] A.F. van Woudenberg, *Chatbots and Dialogue Systems: A Hybrid Approach*, Master's Thesis, 2014.
- [15] Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, Alicia Abella, *PARADISE: A Framework for Evaluating Spoken Dialogue Agents*, 1997.
- [16] Annika Silvervag, Arne Jonsson, *Subjective and Objective Evaluation of Conversational Agents in Learning Environments for Young Teenagers*, 2011.
- [17] Xiao-Jun Wu, Fang Zheng, Wen-Hu Wu, *A hybrid dialogue management approach, a hybrid dialogue for a flight spoken dialogue system*, 2002.
- [18] Pavel Cenek, *Hybrid Dialogue Management in Frame-Based Dialogue Systems Exploiting VoiceXML*, Ph.D. thesis proposal, 2004.
- [19] Xiaojun Wu, Fang Zheng, Mingxing Xu, *Topic Forest: a plan-based dialog management structure*, 2001.
- [20] David Goddeau, Helen Meng, Joe Polifroni, Stephanie Seneff, Senis Busayapongchai, *A form-based dialogue manager for spoken language applications*, 1996.
- [21] Deeno Burgan, *Dialogue Systems & Dialogue Management*, 2017.
- [22] Alexander I. Rudnicky, Dan Bohus, *The RavenClaw dialog management framework: Architecture and systems*, 2008.
- [23] Svetlana Stoyanchev, Pierre Lison, Srinivas Bangalore, *Rapid Prototyping of Form-driven Dialogue Systems Using an Open-source Framework*, Proceedings of the SIGDIAL 2016 Conference, 2016.

- [24] Jason D. Williams, Eslam Kamal, Mokhtar Ashour, Hani Amr, Jessica Miller, Geoff Zweig, *Fast and easy language understanding for dialog systems with Microsoft Language Understanding Intelligent Service (LUIS)*, Microsoft Research
- [25] David Maulsby, Saul Greenberg, Richard Mander, *Prototyping an intelligent agent through wizard of oz*, In Proceedings of the INTERACT93 and CHI93 conference on Human factors in computing systems, 1993.
- [26] Jenny Brusik, Torbjörn Lager, Anna Hjalmarsson, Preben Wik, *DEAL Dialogue Management in SCXML for Believable Game Characters*, 2007.
- [27] Markus M Berg, *NADIA: A Simplified Approach Towards the Development of Natural Dialogue Systems*, 2015.
- [28] Peter Ljunglöf, Staffan Larsson, *A Grammar Formalism for Specifying ISU-based Dialogue Systems*, 2008.
- [29] Staffan Larsson, Alexander Berman, *Domain-specific and General Syntax and Semantics in the Talkamatic Dialogue Manager*, Empirical Issues in Syntax and Semantics 11, 2016.
- [30] Staffan Larsson, *Issue-based Dialogue Management*, PhD Thesis, 2002.
- [31] Fredrik Kronlid, Jessica Villing, Alexander Berman, Staffan Larsson, *Comparing system-driven and free dialogue in in-vehicle interaction*, Proceedings of Interspeech 2011, Conference paper 2011.
- [32] Robin Persson, *Constructing a Prototype Spoken Dialogue System for World Wide Named Places*, Master's Thesis, 2012.
- [33] Talkamatic AB, *TDM Dialogue Strategy Examples*, 2016.
- [34] Talkamatic AB, <http://www.talkamatic.se/>, visited 08/06/2017.

# Appendix A

## Dialogue document

```
<?xml version="1.0" encoding="UTF-8"?>
<Dialogue>
  <InitGreeting>Hello! You are now talking to a chatbot that will help
    you to book tickets to the film you want to see!</InitGreeting>
  <EndPrompt>That was all i needed! Now go and enjoy your movie!</
    EndPrompt>
  <UnknownTopic>I'm sorry I did not understand what you said , please
    use whole sentences to communiante with me!</UnknownTopic>
  <Frame Topic="ReserveTicket">
    <Slot Entity="MovieName">
      <Prompt lang="en">What movie do you want to see?</Prompt>
      <Type>primary</Type>
      <Confirmation lang="en">you want to see [value]
      </Confirmation>
    </Slot>
    <Slot Entity="builtin.number">
      <Prompt lang="en">How many tickets do you want?</Prompt>
      <Type>primary</Type>
      <Confirmation lang="en">you want [value] tickets</
        Confirmation>
    </Slot>
    <Slot Entity="builtin.datetimeV2.date">
      <Prompt lang="en">What day do you wish to see the movie?</
        Prompt>
      <Type>primary</Type>
      <Confirmation lang="en">you want to see the movie on [value
        ]</Confirmation>
    </Slot>
```

```

    <Slot Entity="builtin.datetimeV2.time">
      <Prompt lang="en">At what time do you wish to see the movie
        ?</Prompt>
      <Type>secondary</Type>
      <Confirmation lang="en">you want to see the movie at [value
        ]</Confirmation>
    </Slot>
  </Frame>
  <Frame Topic="PersonalInfo">
    <Slot Entity="FullName">
      <Prompt lang="en">What is your full name?</Prompt>
      <Type>primary</Type>
      <Confirmation lang="en">your full name is [value]</Confirmation
        >
    </Slot>
    <Slot Entity="builtin.email">
      <Prompt lang="en">What is your email</Prompt>
      <Type>primary</Type>
      <Confirmation lang="en">your email is [value]</Confirmation>
    </Slot>
  </Frame>
  <Frame Topic="WhatMovies">
    <Slot Entity="MovieName">
      <Type>auxiliary</Type>
    </Slot>
    <Slot Entity="builtin.datetimeV2.date">
      <Type>auxiliary</Type>
    </Slot>
  </Frame>
  <TUD>
    <Topic>ReserveTicket</Topic>
    <Topic>PersonalInfo</Topic>
  </TUD>
</Dialogue>

```