THESIS TITLE:

AUTHOR:

M.S. Thesis Approved as an elective towards the Master of Science Degree in Computer Science.

Thesis Advisor

Thesis Reader

Thesis Reader

Thesis Reader

Thesis Reader

GRADUATE SCHOOL APPROVAL:

Director, Graduate School

COPY RECEIVED IN GRADUATE SCHOOL OFFICE:

Recipient's Signature

Distribution: Once completed, this form should be scanned and attached to the front of the electronic dissertation document (page 1). An electronic version of the document can then be uploaded to the Northeastern University-UMI website.
CONTEXT AWARE CONVERSATIONAL AGENT FOR FLIGHT SEARCH

BY

HRIDYA RAMESH

THESIS

College of Computer and Information Science
Northeastern University

Thesis Committee:
Advisor: Prof. Adrienne Slaughter
Advisor: Prof. Rutu Mulkar-Mehta
Abstract

The Conversational Agents of today cannot indulge in a continuous conversation with a human due to the Conversational Agent’s inability to infer and apply context from previous interactions. The Master’s Thesis is intended to address this issue by building context modeling for the Conversational Agents for the scenario of “Searching for Flights” and answering the research question of what context is needed to provide a natural(-ish) conversation with an agent that will result in the user successfully searching for flights.
1. Introduction

The notion of human like virtual character has been framed and realised in many different ways. In this context, one might think of a chatbot, virtual companion, interface agent, embodied conversational agent, virtual assistant, etc often used interchangeably. A Conversational Agent (CA) can be defined as “computer interfaces that can hold up their end of the conversation, interfaces that realize conversational behaviors as a function of the demands of dialogue and also as a function of emotion, personality, and social conversation.” [Cassell, Sullivan, et al., 2000]. It can be thought of as the emergent form of dialog system that is becoming increasingly embedded in personal technologies and devices.

1.1 History

Conversational system development dates back to the early days of computer science when pioneering researchers started to take up serious projects aimed at having computers interact with people using natural language. Their endeavors have produced a broad range of theories, techniques, and systems, ranging from basic research to applications, from text to multimodal signals, from dialogue to story, and from computational to cognitive. This section presents a bird’s eye view of history of conversational agents.

ELIZA is an early natural language processing computer program created from 1964 to 1966 at the MIT Artificial Intelligence Laboratory by Joseph Weizenbaum. Created to demonstrate the superficiality of communication between man and machine, Eliza simulated conversation by using a 'pattern matching' and substitution methodology that gave users an illusion of understanding on the part of the program, but had no built in framework for contextualizing events.

SHRDLU is another program for understanding natural language, written by Terry Winograd at the M.I.T. Artificial Intelligence Laboratory in 1968-70. SHRDLU carried on a simple dialog (via teletype) with a user, about a small world of objects (the BLOCKS world) shown on an early display screen (DEC-340 attached to a PDP-6 computer). [http://hci.stanford.edu/winograd/shrdlu/]

Some of the other most popular early Natural Language Systems include: Lunar [Woods 1973], Baseball [Green 1961].

1.2 Current capabilities and drawbacks

Conversational Agents have become increasingly popular as human-computer interface in the past few years. With the launch of Siri (Apple, 2011), Google Now (Google, 2012), Cortana (Microsoft, 2015) and Alexa (Amazon, 2015), there is a spike in mainstream market commitment to this form of experience. This interface for human-computer interaction allows communication protocols that give a much richer communication channel than other means of communicating (such as interacting through UI). Conversational Agents also provide a social dimension to the interaction, similar to human/human
interactions which can raise the believability and trustworthiness of the agent thus increasing the user’s engagement with the system [van Mulken, Susanne; André, Elisabeth; Müller, Jochen, 1988]. Another effect of the social aspect of agents is that presentations given by an embodied agent are perceived as more entertaining and less difficult than the same presentations given without an agent. [Nass et al, 1991]. The perceived enjoyment, followed by perceived usefulness and ease of use, is the major factor influencing user’s adoption of embodied agents. [Serenko, 2008]. Although the emerging class of CAs may indeed have aspirations to become user’s artificial companion, their current functionality places them some way from realising this.

Recent research finds that in majority of use cases of a CA, the users commented on the lack of ability of the CA to bring to bear contextual understandings between interactions, even when the context was fairly close. Users expected the CA to be able to infer from previous interactions, the context of the current task. In particular, once an interaction/task was complete, the majority of users expected the CA to remember the context of preceding interactions. [Sellen & Luger, 2016]

Following are some examples of conversations I have had with Siri that shows Siri’s inability to infer context:

1) **User:** Find restaurants near me.
   **Siri:** I found 15 restaurants fairly close to you. <Displays 15 restaurants>
   **User:** Tell me more about the second one.
   **Siri:** I am sorry, I am afraid I cannot do that.

2) **User:** Find restaurants near me.
   **Siri:** I found 15 restaurants fairly close to you. <Displays 15 restaurants>
   **User:** Are any of them Italian?
   **Siri:** My web search turned this up. <Displays web search for “Are any of them Italian”>

The above mentioned potential benefits and current challenges have motivated this work defined precisely in next section.

### 1.3 Problem Statement

This thesis project addresses the drawback of the current Conversational Agent’s inability to infer contexts from previous interactions by implementing a Context Manager to extend the current functionality of the system. I intend to apply the context handling by focusing on a specific task of searching for flights.

**The task of searching for flights:** In order to use a system to look up flights, the user needs to provide, at a minimum, the **departure city**, the **arrival city** and **date of travel**. In addition, the user may give other parameters such as a **airline** of choice, preference for flights with **no layover**, etc. Thus, the parameters for flight search include both required (departure city, arrival city, date of travel) and optional (airline,
direct flights, etc.). The system takes these parameters, queries for flights that match these parameters, and returns the list to the user.

![Flight Search UI](image)

**Fig 1:** Specifying parameters to search for flights on a website (www.expedia.com)

A familiar approach for performing the flight search task is displayed in Figure 1, using a web-based UI (for example, using Expedia or Kayak.com). This thesis focuses on using a conversational agent to perform this task. This will require the agent to maintain inference to the context from previous interaction to enable a smooth flow of conversation.

## 2. Contexts in Conversational Agents

In this section, I define what a context is in a conversation, identify and explain contexts for a flight conversational agent.

Linguistically, a context is said to be “words or sentences that occur before or after a word or sentence and imbue it with a particular meaning”. Context is necessary in any conversation. As humans, context inference in a conversation is intuitive and we do it without thinking much about it. However, handling context while conversing with a CA needs to be designed into the agent to enable a smooth flow in the conversation.

I define two relevant Contexts for the Flight Search task:

1. Search and Refine Flight Context
   a. Required Parameters:
      i. Departure City
      ii. Arrival City
      iii. Date of Travel
   b. Optional Parameters
2. Follow Up Question Context
   a. Required Parameters:
      i. Departure City
      ii. Arrival City
      iii. Date of Travel
      iv. List of relevant flights
   b. Optional Parameters
      i. Preferred Airline
      ii. Fetch Direct flights

2.1 Search and Refine Flight Context

When a user is searching for flights, the context includes required parameters and optional parameters (Section 1.3). The conversational agent cannot fetch a list of flights without the required parameters. If a user does not provide any of the required parameters, the CA has to elicit those parameters while keeping the already provided parameters in context.

After the user has specified all the flight search parameters, the CA will provide a list of flights that satisfy those constraints. At this point, the user might want to try searching for flights of different prices by modifying the arrival or departure date parameters. In this case, the agent will execute a new query with the previous parameters and any updated parameters.

Here, the context can be thought of as an empty box with slots. Each slot in the context is filled up with the parameters obtained from the current user utterance.

There are 2 possible states: one in which the user has not provided all of the required parameters, and one where the user begins modifying the required parameters. Below, I provide examples of the conversation between the user and the CA for each of these states.

**State 1: CA is eliciting required parameters** This is the state in conversation between the CA and user to search for flights, where at least some of the required parameters have not been given by the user. Below are examples of this conversation state.

Example 1:
In this conversation, the initial utterance by the user to the CA provides no required parameters for flight search. As a result, the CA asks for each parameter in subsequent questions, keeping the bound parameters in context.
| User Says: | “I want to search for flights” | {departure city: ?? 
arival city: ?? 
date of travel: ?? } |
| Bot Says: | “Where are you flying out from?” | {departure city: .... 
arival city: ?? 
date of travel: ?? } |
| User Says: | “From Seattle” | {departure city: Seattle 
arival city: ?? 
date of travel: ?? } |
| Bot Says: | “What is your destination?” | {departure city: Seattle 
arival city: .... 
date of travel: ?? } |
| User Says: | “To NYC” | {departure city: Seattle 
arival city: New York 
date of travel: ?? } |
| Bot Says: | “When do you want to travel?” | {departure city: Seattle 
arival city: New York 
date of travel: .... } |
| User Says: | “Aug 20th” | {departure city:Seattle, 
arival city: New York, 
date of travel: 2017-08-20} |
| Bot Says: | <List of Flights from Seattle to New York on Aug 20th> | {departure city: Seattle, 
arival city: New York, 
date of travel: 2017-08-20} |

Example 2:
In this example, the conversation is initiated with the user providing the destination. The CA then asks for the other required parameters in subsequent questions, updating context at each step.

<table>
<thead>
<tr>
<th>User/Bot</th>
<th>Utterance</th>
<th>Context</th>
</tr>
</thead>
</table>
| User Says: | “I want to search flights to LA” | {departure city: ?? 
arival city: Los Angeles 
date of travel: ?? } |
| Bot Says: | “Where are you flying out from?” | {departure city: .... 
arival city: Los Angeles 
date of travel: ?? } |
| User Says: | “From Seattle” | {departure city: Seattle 
arival city: Los Angeles } |
Example 3: 
Here, the initial query from the user consists of 2 out of the 3 required parameters and so the CA only asks for the one missing parameter and updates the context.

<table>
<thead>
<tr>
<th>User/Bot</th>
<th>Utterance</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Says:</td>
<td>“Search flights to dallas on Aug 20th”</td>
<td>{departure city: ??, arrival city: Dallas, date of travel: 2017-08-20}</td>
</tr>
<tr>
<td>Bot Says:</td>
<td>“Where are you flying out from?”</td>
<td>{departure city: ...., arrival city: Dallas, date of travel: 2017-08-20}</td>
</tr>
<tr>
<td>User Says:</td>
<td>“From Seattle”</td>
<td>{departure city: Seattle, arrival city: Dallas, date of travel: 2017-08-20}</td>
</tr>
<tr>
<td>Bot Says:</td>
<td>&lt;List of Flights from Seattle to Dallas on Aug 20th&gt;</td>
<td>{departure city: Seattle, arrival city: Dallas, date of travel: 2017-08-20}</td>
</tr>
</tbody>
</table>

Note that in each of the examples, parameter values provided by the user are normalized when saved into the context. For example, when the user says “Aug 20th” in Example 3, the travel date is specified as “2017-08-20”. In Example 2, “LA” is translated into “Los Angeles”. This normalization is provided by the CA platform, described in more detail later in the paper. (Section 3.2)

**State 2: User is modifying the required parameters:** In this phase of the conversation, the user may be refining and exploring flights, thereby updating parameters. Unlike the State 1, context slots are not filled at each utterance, but updated according to the user’s utterance.
In this conversation, all the required parameters are provided in the initial user utterance. But, the departure date is updated by the user in the subsequent utterance.

<table>
<thead>
<tr>
<th>User/Bot</th>
<th>Utterance</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Says:</td>
<td>“Find Flights from Seattle to Dallas on Aug 20th”</td>
<td>{departure city: Seattle, arrival city:Dallas, date of travel: 2017-08-20}</td>
</tr>
<tr>
<td>Bot Says:</td>
<td>&lt;List of Flights from Seattle to Dallas on Aug 20th&gt;</td>
<td>{departure city: Seattle, arrival city:Dallas, date of travel: 2017-08-20}</td>
</tr>
<tr>
<td>User Says:</td>
<td>“Search flights for Aug 24th instead”</td>
<td>{departure city: Seattle, arrival city: Dallas, date of travel: 2017-08-24}</td>
</tr>
<tr>
<td>Bot Says:</td>
<td>&lt;List of Flights from Seattle to Dallas on Aug 24th&gt;</td>
<td>{departure city: Seattle, arrival city:Dallas, date of travel: 2017-08-24}</td>
</tr>
</tbody>
</table>

### 2.2 Follow Up Question Context

Once the user has obtained the list of flights he/she wishes to see, the user might have follow up questions about the result. In such queries, the context is the resultant list of flights. There are three possible conversational states explained below.

**ResultContext:** { Flight A: {price, duration, airline, …. } , Flight B: {price, duration, airline, … } …. }

**State 3:** This conversation state refers to user asking for details about a single flight from the list of flights fetched for the given locations and date.

In this conversation, after the list of flights is fetched by the CA, the user wants details on the second flight from the list. In this conversational state, the result flight list is the context.

<table>
<thead>
<tr>
<th>User/Bot</th>
<th>Utterance</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Says:</td>
<td>“I want to search for flights from Seattle to NYC on Aug 20th”</td>
<td>{departure city: Seattle, arrival city: New York, date of travel: 2017-08-20}</td>
</tr>
<tr>
<td>Bot Says:</td>
<td>&lt;List of Flights from Seattle to New York on Aug 20th&gt;</td>
<td>{departure city: Seattle, arrival city: New York, date of travel: 2017-08-20, }</td>
</tr>
<tr>
<td>User Says</td>
<td>Utterance</td>
<td>Context</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>User Says</td>
<td>“Show me the 2nd flight”</td>
<td>{departure city: Seattle, arrival city: New York, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
<tr>
<td>Bot Says</td>
<td>&lt;Flight B&gt;</td>
<td>{departure city: Seattle, arrival city: New York, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
</tbody>
</table>

**State 4:** This conversational state refers to user asking the chatbot about information on the fetched flight results for the given locations and dates.

Example 1:
In this example, after the user gets the result flight list, the user wants to know the price of 3rd flight in the result.

<table>
<thead>
<tr>
<th>User/Bot</th>
<th>Utterance</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Says</td>
<td>“I want to search for flights from NYC to Dallas on Aug 20th”</td>
<td>{departure city: New York, arrival city: Dallas, date of travel: 2017-08-20}</td>
</tr>
<tr>
<td>Bot Says</td>
<td>&lt;List of Flights from New York to Dallas on Aug 20th&gt;</td>
<td>{departure city: New York, arrival city: Dallas, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
<tr>
<td>User Says</td>
<td>“What is the price of 3rd flight”</td>
<td>{departure city: New York, arrival city: Dallas, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
<tr>
<td>Bot Says</td>
<td>&lt;price of Flight B&gt;</td>
<td>{departure city: New York, arrival city: Los Angeles, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
</tbody>
</table>

Example 2:
This example is very similar to the previous conversation (example 1). Instead of the price, the user wants to know the duration of 3rd flight from the result.

<table>
<thead>
<tr>
<th>User/Bot</th>
<th>Utterance</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Says</td>
<td>“I want to search for flights from NYC to Dallas on”</td>
<td>{departure city: New York,</td>
</tr>
<tr>
<td><strong>Aug 20th”</strong></td>
<td></td>
<td>arrival city: Dallas, date of travel: 2017-08-20</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>Bot Says:</strong></td>
<td><code>&lt;List of Flights from New York to Dallas on Aug 20th&gt;</code></td>
<td>{departure city: New York, arrival city: Dallas, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
<tr>
<td><strong>User Says:</strong></td>
<td>“What is the duration of 3rd flight”</td>
<td>{departure city: New York, arrival city: Dallas, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
<tr>
<td><strong>Bot Says:</strong></td>
<td><code>&lt;duration of Flight B&gt;</code></td>
<td>{departure city: New York, arrival city: Los Angeles, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
</tbody>
</table>

**State 5:** This conversational state refers to user’s utterances refining the result flight list obtained for the given locations and date.

In this conversation, the user wants to refine the result flight list to show only flights of a particular airline.

<table>
<thead>
<tr>
<th><strong>User/Bot</strong></th>
<th><strong>Utterance</strong></th>
<th><strong>Context</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Says:</strong></td>
<td>“I want to search for flights from NYC to Dallas on Aug 20th”</td>
<td>{departure city: New York, arrival city: Dallas, date of travel: 2017-08-20}</td>
</tr>
<tr>
<td><strong>Bot Says:</strong></td>
<td><code>&lt;List of Flights from New York to Dallas on Aug 20th&gt;</code></td>
<td>{departure city: New York, arrival city: Dallas, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
<tr>
<td><strong>User Says:</strong></td>
<td>“Can you show me only delta airline flights”</td>
<td>{departure city: New York, arrival city: Dallas, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
<tr>
<td><strong>Bot Says:</strong></td>
<td><code>&lt;List of Delta airline Flights from New York to Dallas on Aug 20th&gt;</code></td>
<td>{departure city: New York, arrival city: Los Angeles, date of travel: 2017-08-20, flights: ResultContext}</td>
</tr>
</tbody>
</table>
3. Implementation

With the task and the contexts identified in the section 2, the methodology of the solution implemented and the evaluation of the solution is captured in this section.

The thesis is aimed at accomplishing the carrying conversational context by an agent throughout a conversation to search for flights, thus making the agent more effective in helping a user accomplish the task.

3.1 Context Manager Architecture

To carry out the context in a conversation, the Natural Language Understanding (NLU) framework, API.AI, is extended to handle contexts, thereby allowing the user to have a continuous thread of conversation to search for flights.

The NLU framework only helps in extracting information from the user statements that is needed to form the response for isolated queries. The context manager creates a connection between a set of isolated queries by inferring the context from previous interactions thereby enabling the agent to have a continuous conversation.

The context manager is a Flask (micro web development framework for Python) Application hosted on Heroku. This web service is integrated with the API.AI platform. The response given back to the user from the Flight Conversational Agent is formed in the context manager by processing the incoming request and as well as the context stored from the previous interaction.

The API.AI platform parses the input statements made by the user. A request is then sent to the context manager. The request is sent as a json, contains data about the user’s statement. Based on the request data, the context to be updated or processed is identified (Section 2.1). the request is either passed on to the missing parameter context handler or the follow up question context handler. The respective context is then updated and stored as a json file and the incoming request and the context is processed to give an appropriate response. The response is sent back to the API.AI platform which is then displayed back to the user (See Fig.2)
The two types of contexts handled in the project: Missing Parameter Context and Follow up Questions context are handled separately in the Context Manager.

The Missing Parameter Context Module of the Context Manager is responsible for populating the Search and Refine Flight Context as defined in Section 2.1. This is the context which defines the parameters required to fetch a list of flights. When the context is completely populated, the flight data is fetched from Google Flight Availability API.

Initially, the request is processed to find which parameters are provided by the user in the current query. The current query parameters are also stored in the context.json file for following queries. Once the parameters in the current query is accessed, if there are missing parameters required to fetch the results, then an appropriate response asking for the missing parameters is sent back to the user. Next when the user provides the parameters asked for, the incoming request is processed to get the parameter in the current query and then the context.json file is processed to get the previous parameters. If all parameters are available, then the list of flights is fetched by making a call to the Google Flight Availability API. If there is still some parameters missing, then the same process is followed i.e asking for the parameter by appropriate response, parsing current request to get parameter and the previously provided parameters obtained by parsing the context.json (see Fig.3).

Once the results are fetched from Google Flight Availability API, these results are stored as context needed to answer the follow up questions.

The Follow Up Question Context Module of the Context Manager is responsible for populating and using the Follow Up Question Context as defined in Section 2.2. In this case the context that is processed is the list of flights displayed to the user in the previous interaction.

Fig.2: Control Flow Diagram
The follow up question request is processed to get the parameters which has the information about follow up questions. For example, if the user has asked for the price of the second flight, the ordinal key of the parameter will have the value 2 and the detail key will have the value price. Then the context having the flight results is processed and for the above example, the price value of the second result is fetched from the context and displayed to the user (see Fig. 3).

The Make Response module is responsible for formatting the output (response sent back to the NLU platform). After the response content is framed in the context handlers, the data is then sent to the make response module to format a json with the data and sent back to the NLU platform to display it back to the user.

![High Level Architecture Diagram of Context Manager](image)

**Fig.3: High Level Architecture Diagram of Context Manager**

### 3.2. Natural Language Understanding framework

A Natural Language Understanding framework is used to parse and interpret the course of action needed to be taken for statements made by the user. The framework chosen for the Thesis project implementation is API.AI.

The API.AI platform supports text input and output. It also supports speech input and output with third party integrations. There are various platform integrations available (accessible as HTTP request to API.AI agent’s query endpoint or connect using SDKs). It is also possible to extract or access data and perform business logic by integrating the API.AI agent with a web service via webhooks.
3.2.1 API.AI Architectural Overview

The API.AI platform receives user statements in the form of text or speech input. This input is parsed by the platform with the help of machine learning techniques, domain knowledge and NLP to understand and analyze the user’s intent. After the platform identifies the user’s intent, the platform determines the appropriate response defined within the API.AI platform or via an external web service. The API.AI platform connects to an external service via webhooks. The web service can call other APIs or connect to a database to fetch data needed for the response. The response is then sent back to the user as text or speech (Fig. 3).

![API.AI Architectural Diagram](https://api.ai/docs/agents)

3.2.2 API.AI Terminologies and Explanation

This section describes the concepts in API.AI and how they map to relevant concepts in our Context Aware Conversational Agent architecture.

1. **Agent**
   An Agent is a conversational application intended perform specific tasks. Examples include an Alarm Agent that will set an alarm, Restaurant Reservation Agent that will create a restaurant reservation, and so on. They are also known as NLU Modules that transform user input into actionable data. The process an agent follows from invocation to fulfilment is similar to someone answering a question.

2. **Intents**
   An Intent represents a mapping between what a user says and what action should be taken by the agent. Intents are used by the agent to provide training data which is used in training the API.AI’s internal machine learning model so that the user statements can be matched to the appropriate intent.

3. **Entities**
   Entities represent any useful data that needs to be extracted from the user statements. These pieces of data essentially gives an insight into what the user is talking about.
4. **Response**
After the user input statement is mapped to one of the intents and entities is extracted from the input statement, an appropriate response needs to be formed to be displayed to the user. The response can be defined in respective intents or formed in an external web service.

5. **Fulfilment**
The external web service is integrated into the API.AI platform via webhooks using fulfilment.

In the thesis project, the Agent is a context aware flight agent intended to answer questions relating to flight search. Some examples of the Intents for a flight agent are: flight search intent, flight check from location intent, flight check to location intent, flight answer follow up question intent, flight reset intent, etc. Entity values for this project are: from location, to location, ordinal value (which result flight is the user asking more information about), follow up details (price, duration, etc). A sample Response of the flight agent is: “Where are you flying out from”, “What is your destination”, etc. The fulfilment web service is the context manager (see Section 3.1).

### 3.3 Data Source

The data source used for this project is the Google Flight Availability API. The API returns full display and booking information with real time availability through an easy to implement json interface. A Google account used to create a project in the Google API Console.

The API supports two ways of authentication: OAuth 2.0 and API Keys. The authentication method used is the API Keys. The API is invoked by sending a HTTP request from the context manager and parsing the responses.

### 3.4 Evaluation Criteria

There are two ways of evaluating the performance of a Conversational Agent.

1. **Confusion Triggers**
2. **Extent of the proposal implementation**

#### 3.4.1 Confusion Triggers

Confusion triggers [https://blog.ubisend.com/optimise-chatbots/benchmarking-chatbot-user-engagement](https://blog.ubisend.com/optimise-chatbots/benchmarking-chatbot-user-engagement) refer to the CA’s version of “I don’t understand the question”. The encounter of “I don’t understand the question” for a CA is important because it represents an instance where the CA fails to understand the user’s utterance which leads to frustrating user experience.

I used the this criteria to evaluate two flight CAs: context aware flight CA and context unaware flight CA. I used 10 evaluation conversations to measure the confusion triggers. I then calculated a score using these conversations. The results are summarized below:
Context Aware Flight Agent:  
Successful Conversations: 9/10  
Score: 0.9 (90 %)

Context Unaware Flight Agent:  
Successful Conversations: 1/10  
Score: 0.1 (10 %)

NOTE: A successful conversation is one where the CA does not respond with “I don’t understand the question” to any of the user’s utterance (see Appendix for the 10 conversations used for evaluation).

Context Unaware Flight Agent’s successful conversation:  
User: I want to travel to phoenix from seattle on july 20th  
CA: I found 11 flights  
User: Can you show me only flights from United airlines  
CA: I found 5 flights

This conversation does not encounter “I don’t understand the question” because the initial user utterance has all of the required parameters. The agent does not need context to elicit parameters.

Context aware Flight Agent’s un successful conversation:  
User: Find flights from San Francisco to Dallas on july 20th  
CA: I have 10 flights from San Francisco to Dallas on july 20th  
User: what is the duration of 9th flight  
CA: 125 min  
User: find return flights from Dallas to Seattle on july 28th  
CA: I found 7 flights  
User: wait, can we go back to the previous flight search  
CA: I don’t understand the question

This conversation has an initial user utterance asking for flights and a follow up question about the duration of one of the flights. Then the user starts a new search in the next utterance, the user wishes to make changes to the first flight search. This requires the agent to bear the entire flight search as the context. This conversation fails for the context aware flight agent as this type of context is out of scope for the thesis, which handles only Search Flight context and refine flight list context. (See Section 2).

Thus, the context aware flight agent fails for 1/10 conversations and context unaware agent fails for 9/10 questions showing that having the context manager for the flight agent leads to a less frustrating user experience.

3.4.2 Extent of the proposal implementation  
In this section, I report on the completion of the original proposed milestones, summarized in the table below.
| May 2017          | Develop a CA with API.AI platform to answer simple questions about flight booking using sample flight data:  
|                  | 1. Define a new Agent (Travel bot)  
|                  | 2. Define intents (ex: source, destination, dates)  
|                  | 3. Define entities | To Do | Done |
| June 2017        | Identify and define various contexts with respect to flight booking. | To Do | Partial¹ |
| June 2017        | Incorporate and test the context in the travel bot to handle subsequent user questions to complete the end to end flight booking activity: Search for flight -> Book flights -> Remedy disruptions. | To Do | Done |
| July 2017        | Start with one/two contexts. Evaluate and iterate for specific contexts. The idea is to handle a few contexts well before trying to take on all of the contexts | To Do | Done |
| July 2017        | Create a webservice to fetch the flight data from the flight stats developer API and expose this web service via webhooks to the API.AI platform in order to allow the NLP platform to have access to flight data | To Do | Done² |

1. Contexts with respect to flight booking was not implemented as it turned out to be much more complicated with respect to the number of parameters to be considered. But the implementation idea is similar to the Missing Parameter Context handler for flight search but with more number of parameters. Hence the contexts were limited to questions only about flight search.

2. The data source used in the project is the Google Flight Availability API instead of flight stats because of the ease of use and easy to parse json response of the Google Flight API.

4. Future Work

This section identifies potential additions and improvements to this project.

One helpful addition to the project would be to identify and implement more contexts relating to flight conversations. With more number of users using the context aware flight agent, it is possible to gain more insight into what other contexts can be dealt with. With the current limitation of the number of users using the agent, the identified contexts are limited. Current Engineering and Linguistic limitations also contribute to the limited contextual use cases.

More complicated flight booking can be added to the current flight searching capability of the agent. The flight booking context use cases can be handled similar to the missing parameter use cases but the flight booking would consist of more parameters to be dealt with.

Personalization is a very powerful tool. With increase in the user base for the agent, it will be possible to gather the user data and use data science techniques to include capabilities like recommendations to the user.
Integrating the existing Flight agent with popular, commercially available platforms such as Alexa, Facebook Messenger would also improve user experience.

5. Conclusion

In this thesis project, I explored an approach to improve the user experience of a conversational agent by defining and designing a Context Manager to handle the conversational context during a conversation between a human and a CA. I provided a definition of the flight search task, proposed a general definition of context, identified specific contexts for the flight search task, and implemented a CA for the same. I evaluated the CA by comparing two CAs, one which used the Context Manager, and one that did not. I found that maintaining context resulted in fewer confusion triggers in the CA, thus enabling a better user experience.

References:

2. Xin Li & Dan Roth. 2004. “Learning Question Classifiers”: The role of Semantic Information
8. Neural Networks in plain english: www.ai-junkie.com
9. Early Facebook Messenger Bots - A critique and look forward
10. Expedia’s first bot for booking hotels: www.venturebeat.com
11. Technology behind travel bots: www.30secondstofly.com
12. The ultimate travelbot list: www.30secondstofly.com
13. How to plan your next vacation with a chatbot: www.Nytimes.com
15. Susanne van Mulken, Elisabeth André & Jochen Muller, 1988. The persona effect. How substantial is it?
17. Byron Reeves. The benefits of interactive online characters
18. SHRDLU: http://hci.stanford.edu/winograd/shrdlu/
Appendix

Evaluation Conversations:

1. User: I want to search for flights
   CA: Ok, where are you flying from?
   User: from Seattle
   CA: Sure, what is your destination?
   User: to San Francisco
   CA: When would you like to travel?
   User: July 20th
   CA: I have 6 flights from Seattle to SFO on july 20th

2. User: I want to search flights to New York
   CA: Ok, where are you flying from?
   User: Seattle
   CA: What is your departure date?
   User: July 20th
   CA: I have 9 flights from Seattle to New York on july 20th

3. User: I want to travel to Chicago from Seattle on july 20th
   CA: I have 8 flights from Seattle to Chicago on july 20th

4. User: I want to travel to Chicago from Seattle
   CA: Sure, when are you planning to travel?
   User: July 20th
   CA: I have 8 flights from Seattle to Chicago on july 20th
   User: What is the price of 2nd flight?
   CA: 250 USD

5. User: I want to travel to Chicago from Seattle
   CA: Sure, when are you planning to travel?
   User: July 20th
   CA: I have 8 flights from Seattle to Chicago on july 20th
   User: Is there a layover for the 4th flight?
CA: No

6. User: I want to travel to Charlotte from Seattle on July 20th
CA: I found 2 flights
User: These dates don’t work for me, can you try for July 27th instead?
CA: I found 4 flights on July 27th

7. User: I want to travel to Phoenix from Seattle on July 20th
CA: I found 11 flights
User: Can you show me only flights from United Airlines?
CA: I found 5 flights

8. User: I want to travel on May 15th
CA: Ok, where are you flying from?
User: from SFO
CA: Sure, what is your destination?
User: to Seattle.
CA: I have 6 flights from SFO to Seattle on May 15th

9. User: Find flights from San Francisco to Dallas on July 20th
CA: I have 10 flights from San Francisco to Dallas on July 20th
User: what is the duration of 9th flight
CA: 125 min
User: find return flights from Dallas to Seattle on July 28th
CA: I found 7 flights
User: wait, can we go back to the previous flight search
CA: I don’t understand the question

10. User: Find Flights for me
CA: where are you flying from
User: from Seattle
CA: what is your destination
User: I don’t know, can you help me in finding a destination
CA: I don’t understand the question

20.