

Towards a Deeper Understanding of Current Conversational Frameworks through
the Design and Development of a Cognitive Agent

by

Prashanti Priya Angara
B.Sc., Gandhi Institute of Technology and Management, 2012

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

© Prashanti Priya Angara, 2018
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by
photocopying or other means, without the permission of the author.

Towards a Deeper Understanding of Current Conversational Frameworks through
the Design and Development of a Cognitive Agent

by

Prashanti Priya Angara

B.Sc., Gandhi Institute of Technology and Management, 2012

Supervisory Committee

Dr. Hausi A. Müller, Co-Supervisor
(Department of Computer Science)

Dr. Ulrike Stege, Co-Supervisor
(Department of Computer Science)

Supervisory Committee

Dr. Hausi A. Müller, Co-Supervisor
(Department of Computer Science)

Dr. Ulrike Stege, Co-Supervisor
(Department of Computer Science)

ABSTRACT

In this exciting era of cognitive computing, conversational agents have a promising utility and are the subject of this thesis. Conversational agents aim to offer an alternative to traditional methods for humans to engage with technology. This can mean to reduce human effort to complete a task using reasoning capabilities and by exploiting context, or allow voice interaction when traditional methods are not available or inconvenient. This thesis explores technologies that power conversational applications such as virtual assistants, chatbots and conversational agents to gain a deeper understanding of the frameworks used to build them.

This thesis introduces FOODIE, a conversational kitchen assistant built using IBM Watson technology. The aim of FOODIE is to assist families in improving their eating habits through recipe recommendations taking into account personal context, such as allergies and dietary goals, while helping reduce food waste and managing grocery budgets. This thesis discusses FOODIE's architecture, and derives a design methodology for building conversational agents.

This thesis explores context-aware systems and their representation in conversational applications. Through FOODIE, we characterize the contextual data and define its methods of interaction with the application.

FOODIE reasons using IBM Watson's conversational services to recognize users' intents and understand events related to the users and their context. This thesis discusses our experiences in building conversational agents with Watson, including features that may improve the development experience for creating rich conversations.

Contents

Supervisory Committee	ii
Abstract	iii
Table of Contents	iv
List of Tables	vii
List of Figures	viii
Acknowledgements	ix
1 Introduction	1
1.1 Motivation	1
1.2 Problem Definition and Research Questions	3
1.3 Contributions	4
1.4 Research Methodology	4
1.5 Thesis Outline	5
2 Background and Related Work	6
2.1 Concepts	6
2.1.1 Terminology: Conversational Interfaces, Bots and Agents	6
2.1.2 Context-Awareness	9
2.2 Requirements for a Rich Conversation	12
2.2.1 Usability	12
2.2.2 Personability	12
2.2.3 Methods of Interaction: Text versus Voice	13
2.3 Classification	14
2.3.1 Response Models	14
2.3.2 Intent Recognition	16

2.3.3	Entity Extraction	18
2.3.4	Domain	19
2.4	AI-as-a-Service	20
2.4.1	IBM Watson Services	20
2.5	Summary	22
3	Conversational Frameworks	23
3.1	Motivation	23
3.2	Key attributes of conversational frameworks	25
3.3	Dialogflow	25
3.3.1	Key Attributes	27
3.4	IBM Watson Assistant	29
3.4.1	Key Attributes	30
3.5	Rasa NLU	31
3.5.1	Key Attributes	31
3.6	Microsoft Bot Framework	33
3.6.1	Key Attributes	33
3.7	Summary	34
4	Smart Conversations	36
4.1	Design Methodology	36
4.2	High-level goals/objectives of FOODIE	38
4.3	Intents	38
4.3.1	Utility Intents	41
4.4	Entities	42
4.5	Dialog Design	44
4.5.1	Spring Expression Language (SpEL)	45
4.6	Context Variables	48
4.6.1	Linguistic Context	48
4.6.2	Personal Context	50
4.6.3	Physical Context	50
4.7	Summary	52
5	Application Architecture and Implementation	53
5.1	Requirements	53
5.2	Application Overview	54

5.2.1	Components	54
5.2.2	Architecture	54
5.3	Data Sources	56
5.3.1	Spoonacular	56
5.3.2	FoodEssentials	59
5.3.3	Recalls	59
5.4	Watson Assistant	60
5.5	Context Management	62
5.5.1	Maintaining state	62
5.6	Orchestration	65
5.7	FOODIE Voice User Interface (VUI)	66
5.8	Summary	67
6	Validation and Discussion	68
6.1	Validation Methodology	68
6.2	Conversational Goals	69
6.2.1	Scenario: Managing users' dietary preferences	70
6.3	Requirement 2: Personalization	74
6.3.1	Recipe Suggestions	74
6.3.2	Discussion: Initialization of Context Variables	75
6.4	Requirement 3: Voice User Interface	76
6.4.1	Unknown Requests	76
6.4.2	Discussion: Building rich conversations	77
6.5	Summary	78
7	Conclusions	79
7.1	Thesis Summary	79
7.2	Contributions	80
7.3	Future Work	81
	Bibliography	84

List of Tables

Table 4.1	List of goals mapped to their intents	39
Table 4.2	List of utility intents	42
Table 4.3	List of entities	43
Table 4.4	List of dialog nodes of FOODIE	47
Table 4.5	Examples of linguistic context variables	50
Table 4.6	Examples of linguistic context variables	51
Table 5.1	Query filters for the complex recipe search endpoint	58
Table 5.2	List of Recipe Search Endpoints in Spoonacular	58
Table 5.3	Request parameters for the <code>message</code> endpoint	61
Table 5.4	Response details for the <code>message</code> endpoint	62
Table 6.1	Goal-Requirement mapping	69

List of Figures

Figure 2.1 Gartner’s Hype Cycle for emerging technologies (2018)	7
Figure 2.2 An example of the personal context variable represented as a JSON string	12
Figure 2.3 Types of Conversational Applications	15
Figure 2.4 A sample AIML script.	17
Figure 2.5 Pseudocode for training an Artificial Neural Network	18
Figure 2.6 An Artificial Neural Network	18
Figure 2.7 A conversational application built using Watson Services	21
Figure 3.1 Chatbot landscape 2017	24
Figure 3.2 Dialogflow	27
Figure 3.3 Rasa component lifecycle	32
Figure 4.1 A subset of nodes in FOODIE’s Conversation	42
Figure 4.2 The flow of conversation as described by IBM Watson Assistant	44
Figure 4.3 Configuration of a dialog node	46
Figure 4.4 Types of context.	48
Figure 4.5 JSON code snippet depicting the linguistic context variable along with output from Watson Assistant	49
Figure 4.6 JSON code snippet depicting personal context variables	50
Figure 5.1 UML Sequence diagram of FOODIE	55
Figure 5.2 FOODIE Component Architecture	56
Figure 5.3 A simplified diagram showing flow of context between the appli- cation and the Watson Assistant workspace	64
Figure 6.1 Selection of dietary preferences	70
Figure 6.2 Identifying and setting entities for dietary preferences	73

ACKNOWLEDGEMENTS

- I would like to express my gratitude to my supervisors Dr. Hausi A. Müller and Dr. Ulrike Stege for the continuous support, mentoring, patience, and encouragement to pursue research.
- I would like to thank my lab mates for all the insightful discussions we've had and for providing such a collaborative atmosphere. I have learnt so much at the Rigi Lab.
- I would like to thank IBM Centre for Advanced Studies for supporting this project.
- I'm grateful to my family for everything, for encouraging and inspiring me to follow my dreams. I could not have done this without them.

Chapter 1

Introduction

We're moving from us having to learn how to interact with computers to computers learning how to interact with us.

Sean Johnson

1

Conversational agents aim to offer alternative, more intuitive methods of interaction with technology compared to traditional methods such as the command line and the graphical user interface. This thesis demonstrates the significance of conversational agents in the era of cognitive computing through *FOODIE: a smart, personalized, context-aware conversational agent* for the modern kitchen. *FOODIE* augments the capabilities of users by helping them with recipes and reasons about dietary needs, constraints and other preferences using IBM Watson technology. Our vision for *FOODIE* is for it to be a central hub of communication for the kitchen, and for related activities such as grocery shopping or other similar food-related domains. This chapter provides our motivation behind this research, the problem definition and our research contributions.

1.1 Motivation

Conversational agents are applications that make use of natural language interfaces, such as text or voice, to interact with people, brands or services. Popular exam-

¹Sean Johnson is a partner and leads product development and growth at Digital Intent, a firm that conceives, builds and grows new digital startups. <https://digintent.com/bots/>

ples of such agents are Apple’s Siri,² Microsoft’s Cortana,³ Google’s Assistant,⁴ and Amazon’s Alexa.⁵ They represent a new trend in digital gateways for accessing information, making decisions, and communicating with technology through sensors and actuators. The concept of conversing with a computing machine has been around for a long time. In 1966, Weizenbaum created ELIZA, the first natural language processing computing program [48]. ELIZA made use of directives provided via scripts along with pattern matching and substitution to respond to queries by humans. The idea of bots, such as ELIZA, was to not replace human intellect, but rather to have such tools to support the human capabilities.

We have come a long way since ELIZA. Natural language processing and artificial intelligence have advanced to such a degree that computers are able to almost accurately predict what a user’s intentions are [23]. For this reason, many traditional command-line and graphical user interfaces are now giving way to conversational agents for a variety of applications. While the former rely on specific input from the user’s side, conversational agents make use of natural language understanding and infer the user’s intent from linguistic sentences. Conversational agents provide more capability than conversational interfaces: they find practical application in areas where users need quick access to information, especially when the information is collated from different sources.

A variety of commercial frameworks have been developed to provide services to define behaviours for conversational agents, including IBM’s Watson Assistant,⁶ Google’s Dialogflow,⁷ Amazon’s Alexa,⁸ and Microsoft’s Bot Framework.⁹ With such frameworks, it is becoming easier not only to spin up simple conversational agents but also to add capabilities of conversation to existing applications.

²<https://www.apple.com/ca/ios/siri>

³<https://www.microsoft.com/en-ca/windows/cortana>

⁴<https://assistant.google.com>

⁵<https://developer.amazon.com/alexa>

⁶<https://www.ibm.com/watson/developercloud/conversation.html>

⁷<https://dialogflow.com/>

⁸<https://developer.amazon.com/alexa-skills-kit>

⁹<https://dev.botframework.com/>

1.2 Problem Definition and Research Questions

There has been a growing trend in making services and frameworks for building conversational agents. However, these are still in their nascent stages. There is a need to make conversational agents sophisticated, i.e., they are meant to be useful and relevant as opposed to agents that have only an initial appeal and have limited capabilities.¹⁰ To become useful, conversational agents have to be robust and serve a purpose that outweighs the usage of more traditional methods of interaction. The following are our questions guiding this research.

RQ1 What are some of the frameworks that provide AI-as-a-Service that are useful in building conversational agents?

Advances in machine learning, artificial intelligence and cloud computing have led to the availability of outsourced AI offered as Artificial Intelligence as a Service (AIaaS), AI Infrastructure (such as Hadoop and Spark) and AI Tools (such as notebooks, libraries and Integrated Development Environments (IDE)). In this thesis, we explore a few frameworks that provide AI-as-a-service for conversational AI, i.e., natural language understanding and dialog building.

RQ2 Given the frameworks to build conversational agents, what are the challenges in designing a sophisticated conversational agent?

We explore this area of research by building a text and voice-enabled conversational agent for the kitchen, FOODIE which uses IBM's AI-as-a-Service called Watson Assistant for natural language understanding. By iteratively building and improving on FOODIE, we identify some good design and development practices for conversational agents. We categorize the challenges based on the conversational method of interaction (i.e., text versus voice). We identify the requirements for a voice-based conversational agent and argue the need for additional capabilities in conversational frameworks for such agents.

RQ3 What role does contextual information play in the design and development of a conversational agent? How best can we represent context for our conversational agent, FOODIE?

For this research question, we recognize various context variables that play an

¹⁰<https://www.artificial-solutions.com/blog/slot-filling-chatbots-will-never-meet-human-expectations>

important role in the personalization of our application, FOODIE. We formulate a representation for the contextual variables and define their interaction with various parts of FOODIE.

1.3 Contributions

The following are the contributions of this thesis.

- C1** A study of popular conversational frameworks: IBM Watson Assistant, Google Dialogflow, Microsoft Bot Framework and Rasa NLU.
- C2** A design methodology for building conversational applications.
- C3** A Voice & Text enabled Conversational Agent: FOODIE, built using IBM Watson Services, serving as a case study for understanding the state-of-the-art in conversational applications.
- C4** A representation and classification of contextual variables for our conversational agent FOODIE

1.4 Research Methodology

Throughout this thesis, we aim to answer the research questions by implementing our smart, context-aware cognitive conversational agent, FOODIE. The chosen area of application for our investigations, the kitchen, is an ideal environment for our purposes, since the domain is reasonably self-contained but has complex contextual data—thus enabling us to research cognitive conversational agents. We chose to use Watson’s speech and conversational services to set up the “natural language” part of our application. By building FOODIE, we gain insights into these conversational services which in turn help us evaluate these services. We augment our application with contextual data, which keeps track of the user’s preferences. Some selected publicly available nutrition databases such as Spoonacular and FoodEssentials are used as the data sources for nutritional and recipe information.

Through our first research question, we aim to study a few popular conversational frameworks, identify their key attributes and compare them on the basis of those attributes. We choose the IBM Watson Assistant framework, Rasa NLU, Microsoft

Bot Framework and Google Dialogflow for our comparative study. For our second research question regarding the design challenges in building conversational agents, we develop a conversational agent FOODIE identifying its requirements and evaluating each of them. For our third research question, we identify and characterize the context variables we need for FOODIE. We validate the requirements of this research question by comparing the different results achieved with and without contextual variables in Chapter 6.

1.5 Thesis Outline

This chapter described our motivation and highlighted our research questions and contributions. The rest of this thesis is organized as follows.

Chapter 2 formalizes key concepts of this research, describes the background and the state-of-the-art in the literature for this thesis.

Chapter 3 describes the features of a select few popular frameworks that provide conversational services.

Chapter 4 elucidates the orchestration of conversations between the user and FOODIE. Additionally, it describes the usage of context to facilitate such conversations.

Chapter 5 describes the application's set up, features and the data sources used.

Chapter 6 presents an analysis of conversational services, gathers insight and discusses some improvements that can be made to existing frameworks.

Chapter 7 concludes with a summary of this research done and discusses ideas for future work.

Chapter 2

Background and Related Work

There has been a steady increase in the number of applications that use conversations as a means of interacting with users. 2016 was dubbed as the year of the chatbot, according to a market research study by MindBowser, in association with the Chatbots Journal.¹ Two years later, conversational applications and virtual assistants are still popular—according to Gartner’s hype cycle for emerging technologies.² Figure 2 shows the hype cycle of emerging technologies in 2018. Virtual Assistants, Conversational AI and AI Platform-as-a-Service are featured in the hype-cycle and predicted to be adopted within the next decade. Today, these conversational applications are used for a number of tasks across industries such as E-Commerce, Insurance, Banking, Healthcare, Telecom, Logistics, Retail, Leisure, Travel and Media. This chapter summarizes the research related to conversational applications in general and home automation applications using conversations in particular. We outline the core concepts regarding conversational agents, followed by the state-of-the-art applications clustered by the different ways to classify these agents.

2.1 Concepts

2.1.1 Terminology: Conversational Interfaces, Bots and Agents

This section explains some of the terms that are often used, sometimes interchangeably, in the literature. We make the following distinctions between a conversational

¹<https://chatbotsjournal.com/global-chatbot-trends-report-2017-66d2e0ccf3bb>

²<https://www.gartner.com/en/newsroom/press-releases/2018-08-20-gartner-identifies-five-emerging-technology-trends-that-will-blur-the-lines-between-human-and-machine>

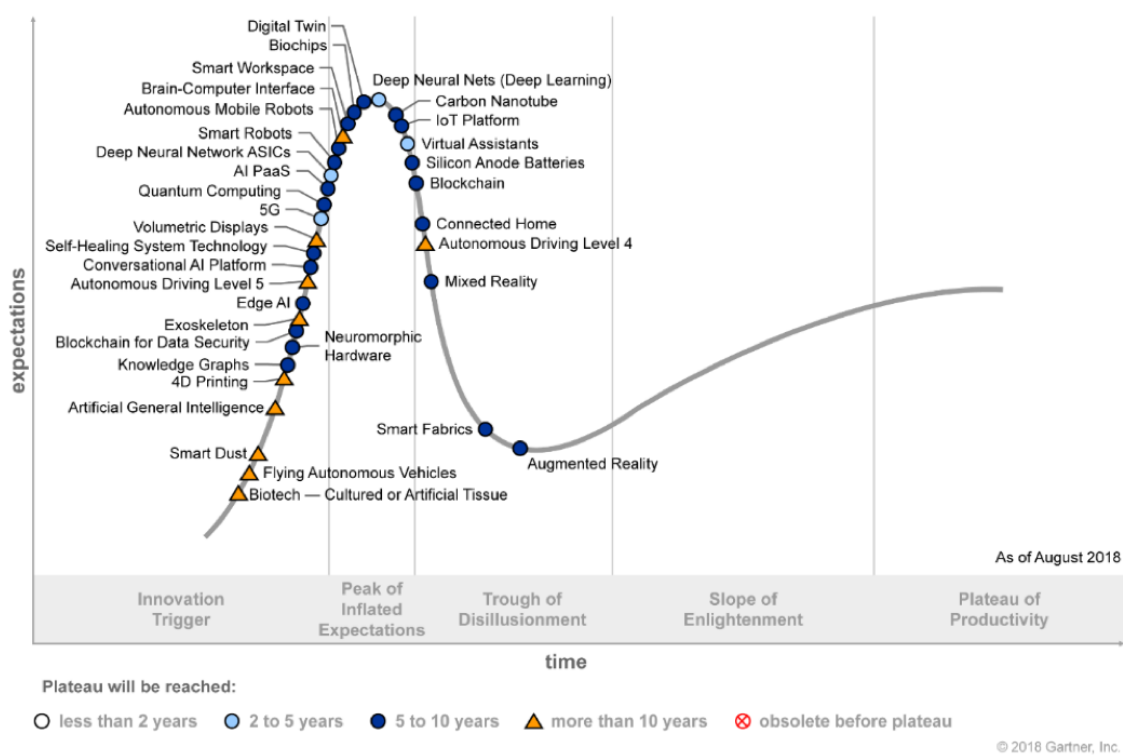


Figure 2.1: Gartner's Hype Cycle for emerging technologies (2018)

interface, a chatbot, a conversational agent, and an embodied conversational agent.

Conversational Interface

In software, an interface represents the shared boundary between components or components and humans across which information is passed [1]. Conversational interfaces rely on natural language for communication, much like how humans interact with each other. In contrast to programming languages, which are unambiguous and follow simple rule-based patterns, natural languages like English often require contextual background and leave a lot for interpretation. Therefore, it is an arduous task for machines to be able to interpret natural language as opposed to programming languages. However, there has been an increase in the usage of conversational interfaces only recently due to emerging technologies [37].

- **Advances in Artificial Intelligence.** Various machine learning and deep learning techniques have improved natural language processing and understanding. Collobert et al. [15] use multitask deep learning to process predictions for Part-of-Speech tags, chunks, named entity tags, semantic roles, semantically similar words and the likelihood that a sentence makes grammatical and semantic sense. Botha et al. [9] use Feed-Forward Networks to show that state-of-the-art results on natural language processing can be achieved even in resource-constrained environments.
- **Powerful Devices.** The power of parallel computing in new processors has gradually increased, from single processor to multiple processor architectures, cores, and threads [42]. Most deep learning and artificial intelligence tasks require significant processing power, and they are now possible with increased performance of small computing devices such as mobile phones.
- **Improved Connectivity.** With the ubiquitous nature of the internet, faster speeds coupled with the ability to delegate compute clouds for processor intensive tasks [6], it is possible to gain traction for conversational interface viability.

Chatbots and Conversational Agents

Chatbots and conversational agents realize (or give form to) a conversational interface. It is the application logic tied to the user interface that parses language to deduce the intent of the user and act on the intention of the user. Although the

terms “chatbot” and “conversational agent” are used interchangeably in the literature, we make the following distinction. A conversational application is a “chatbot” when its sole purpose is to form a conversational bridge between the application and the user interface. We define conversational agents to be more complex than a chatbot, with an additional layer of logic to support various subsystems that the agent could connect to. Additionally, we look at chatbots with more emphasis on the “chat” part, i.e., making these interfaces more human-like. In conversational agents, we focus on the abilities of conversations that might help us perform more complex tasks.

In chatbots, on the one hand, there is a direct mapping between user intents and the capabilities of an application. For example, Pizza Hut’s chatbot on Facebook Messenger can be used to customize and order a pizza, which is exactly what can be done on their website through a graphical interface. On the other hand, in conversational agents, the mapping between the intent and capabilities of the application can be much more complex. Fast et al. [18] describe Iris, a conversational agent that helps users with data science tasks. Iris draws on human conversational strategies to combine commands, allowing users to achieve more complex goals than the ones it has been explicitly designed to support.

Embodied Conversational Agent

Embodied conversational agents exhibit lifelike behaviours and use a range of verbal and nonverbal behaviors for communication that is akin to face-to-face human conversation [12]. More advanced embodied agents, called multi modally interactive embodied agents, must also know how to produce multi-modal output, i.e., respond using visual gestures and not just text and voice. They should be able to deal with conversational functions such as turn taking and feedback, as well as contribute new propositions to the discourse. They are far more complicated since they involve other components of advanced intelligence and vision and are not discussed this thesis.

2.1.2 Context-Awareness

Context-aware computing refers to a general class of mobile systems that can sense their physical environment, and adapt their behavior accordingly [40]. These are applications that make use of situational information and provide a tailored and

personalized user experience.

Context

Villegas et al. [46] provide a definition of context as follows:

“Context is any information useful to characterize the state of individual entities and the relationships among them. An entity is any subject which can affect the behavior of the system and/or its interaction with the user. This context information must be modeled in such a way that it can be pre-processed after its acquisition from the environment, classified according to the corresponding domain, handled to be provisioned based on the system’s requirements, and maintained to support its dynamic evolution.”

Context modeling for conversational agents

Companies are increasingly opting for individualized efforts to engage customers in the era of information abundance and intelligence [2][41]. Personalization is one of the aspects through which customer retention and engagement can be achieved [8]. Personal contextual data can be leveraged by conversational agents to make responses tailored to an individual user. In our case, FOODIE makes use of the following types of contextual data to augment the user’s experience.

- **Physical Context** relating to time of day and location
- **Personal Context** relating to an individuals preferences, in our case, information such as cuisine, dietary restrictions and goals.
- **Linguistic Context** relating to the state information exchanged between our conversational agent and the user.

Representation of context

Contextual data can be classified by the scheme of data structures which are used to exchange contextual information within a system [44] as follows:

- **Key-Value Models:** Key-value pairs are one of the most simple data structures for modeling contextual information. The context variable along with the value of the variable form the key-value pair in this model. Key-value pairs are

easy to manage, but lack capabilities for sophisticated structuring for efficient context retrieval. The JavaScript Object Notation [16] can be used to create a Key-Value Model for representation of context

- **Markup Scheme Models:** These models consist of a hierarchical data structure consisting of markup tags with attributes and content (where content can be recursively defined as a markup model). The Resource Description Framework (RDF) is one example of a markup scheme model [32].
- **Graphical Models:** The Unified Modeling Language is a general purpose modeling tool that can be used to model context as well. Henriksen et al. [22] define a modeling approach based on Object-Role Modeling (ORM) where the basic modeling concept is a *fact* and relationships are established between these facts.
- **Object Oriented Models:** Object Oriented Models employ the concepts of encapsulation and reusability to cover parts of the problems arising from the dynamic nature of context. Access to contextual information is provided through specified interfaces only.
- **Logic Based Models:** Context in logic based models is defined as facts, expressions and rules. There is a high degree of formality when defining these rules. Contextual information is added to, updated in or deleted from a logic based system in terms of facts inferred from the rules in the system.
- **Ontology Based Models:** Ontologies are explicit formal specifications of the terms in a domain and the relations among them [21]. Languages such as OWL (Web Ontological Language) have been developed to represent knowledge bases and can be used to represent contextual information [36]

For our application, FOODIE, contextual data is modeled as a key-value pair represented using JavaScript Object Notation (JSON) as shown in Figure 2.1.2

```

{
  "personal_context": {
    "cuisine": ["Japanese"],
    "allergy": ["peanut", "shellfish"],
    "diet": "pescaterian"
  }
}

```

Figure 2.2: An example of the personal context variable represented as a JSON string

2.2 Requirements for a Rich Conversation

2.2.1 Usability

A recent survey conducted by Zamora et al. [49] explored how conversational agents can find a place in routine daily lives. They argue convincingly that currently the usage of chatbots/conversational agents reduces as its novelty wears off. Their research collected qualitative insights from diverse participants about perspectives on digital personal assistants. Their findings suggest that users expect these assistants to be smart, high performing, seamless and personable.

2.2.2 Personability

In studies done by Milhorat et al. [38], users like the idea of assistants getting to know their personal quirks and highlighted in their work, that the digital assistants are yet to become personal.

Wanner et al. [47] presented KRISTINA, a knowledge-based conversation agent, capable of communicating in multiple languages, with users from different social and cultural backgrounds. It also takes into account the emotional sensitivity of the user, while modeling response to the conversation. Even though FOODIE currently communicates only in English, personability is one of our main objectives. Although not a current feature, we would like to augment FOODIE with support for other languages as well. For FOODIE to be more personable, we make use of a personal context sphere [46] that maintains a list of user preferences.

2.2.3 Methods of Interaction: Text versus Voice

In the aforementioned survey by Zamora et al. [49], users were asked what method of input they preferred. Speaking to an agent was found to be best when the user was multi-tasking or had hands or eyes occupied. Typing seemed to be best when the activity was complex. Users also found that they preferred to interact with bots for common administrative/menial needs and for emotional needs to provide motivation. We took this into consideration while building FOODIE. A kitchen is a place where a user is multi-tasking and their hands are occupied. In such an environment, one would want to interact using voice, and would also want answers to be reliable.

Klopfenstein et al. [30] observed that many of the platforms avoid voice processing and choose text as the most direct and unambiguous form of communication. They also talk about studies on interaction using voice such as those described by McTear et al. [37] and Bastiennali et al. [5], and how unexpected turns of phrase and simple misunderstandings from the users can lead to a misunderstanding of context and therefore yield a breakdown in the conversation. At the same time, speech is becoming a more powerful and reliable way of interacting with devices [31]. There have been breakthroughs in this area such as the speech recognition engine “Deep Speech 2,” developed by Baidu, which recognizes spoken words and interprets users queries with high accuracy [3].

A variety of IoT applications discussed by Cabrera et al. [11] and Kim et al. [29] stress the importance of voice and text-based control for their devices. When an application involves a number of decentralized interconnected devices, it makes sense to have an interface that seamlessly understands a user’s request. For example, in a home automation scenario, it is easier to say “Hey Siri, turn on the lights,” than opening up an application and clicking a few times to turn on lights.

With IoT starting to play an important role in kitchens [13], FOODIE could be connected as a seamless interface between a user and all these devices. For instance, FOODIE could serve as a gateway for the prototypes developed by Ficocelli et al. [19] or Blasco et al. [7], which assist elderly people in the kitchen activities, such as retrieving and storing items or acquiring recipes for preparing meals.

2.3 Classification

This section describes the ways of classifying conversational applications according to the perspectives of functionality, intelligence, methods of parsing input and the environment in which they operate. We also present some of the noteworthy applications under each kind of application. The following types of classification are described in this section.

- **Response Models:** Classification based on the kind of model that defines the response from the conversational agent.
- **Intent Recognition:** Classification based on the method in which users' intents are recognized.
- **Domain:** Classification related to the behaviour of the conversational agent according to their area of expertise.

Note: This is by no means an exhaustive classification. Lebeuf et al. [33] describe a detailed taxonomy for classifying software bots and conversational agents based on their characteristics. Here we focus on the categories that are useful to consider when *building* a conversational agent.

2.3.1 Response Models

Retrieval-Based Models

A conversational application is known to be retrieval-based when a set of rules defines its behaviors. Retrieval based models have a repository of pre-defined responses that are picked based on the input of the user and the context. The input of the user is matched to an intent by an intent classifier (described in detail in Subsection 2.3.2)

The ability of retrieval based models is directly related to the number of responses that have been defined. While the interaction is robust for the scenarios it is trained on, a retrieval based application fails to perform in unknown scenarios.

Generative Models

Generative models build (or generate) responses from scratch. They use machine translation and training data to create responses on the fly. Generative models are not very robust and are prone to make grammatical errors. They are still primitive

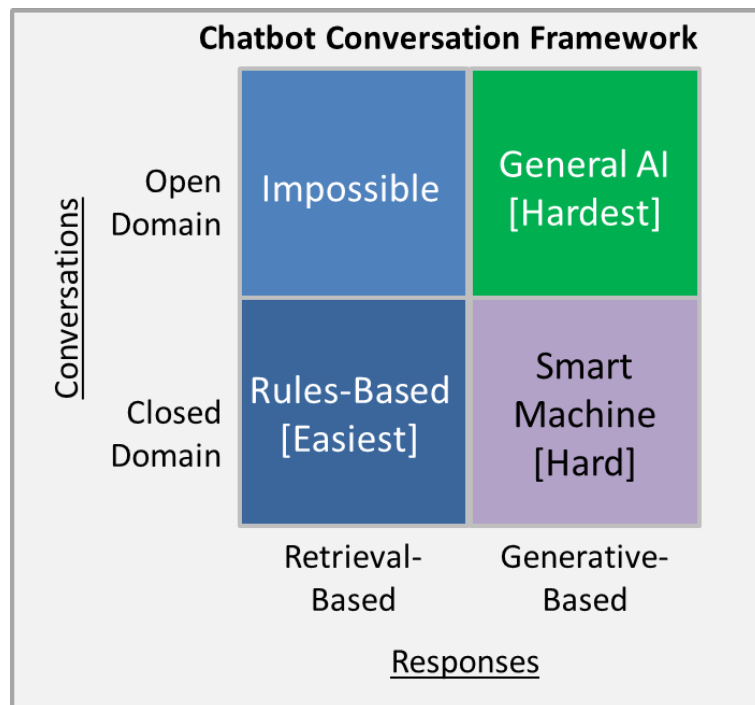


Figure 2.3: Types of Conversational Applications

and use deep learning and machine translation to parse sentences. The responses are generated from extensive training data and therefore are good for unseen data. However, these responses can be highly varied and unexpected [23] (e.g., Microsoft’s twitterbot Tay whose responses turned racist [4]). For conversational applications to appear more human, generative models are better suited. For applications that are more transactional, retrieval-based models are preferred.

A combination of retrieval-based and generative models

For conversational applications to be viable, the robustness of retrieval based models and the ability to generate responses on the fly is ideal. Some applications like Google’s smart reply [27] make use of both generative models and retrieval based models to generate automatic email replies.

2.3.2 Intent Recognition

For any generative or retrieval-based model, input from the user is obtained in terms of linguistic sentences. This means that such input has to be broken down to parse the intent of the sentence. For example, sentences such as “What is the weather like today?”, “Do I need to carry an umbrella today?” or “What is today’s forecast?” although phrased differently convey the intention of enquiring about the weather. The following sections describe the different kinds of intent classifiers.

Pattern Matching

Conversational applications in the early years used “brute force” techniques of pattern matching to classify text and produce an output. Developers used the Artificial Intelligence Machine Language (AIML) to describe the patterns of their conversational application. Each pattern is matched to a response in AIML. The example in Figure 2.4 shows the structure of an AIML script. Basic units of an AIML dialog are called categories. Pattern tags represent possible user input (including * which is a wildcard) and template tags represent possible answers from the conversational application. Even though it is brute force, given enough patterns, a conversational application built this way can appear smart. [45]

```

1     <aiml version="1.0.1" encoding="UTF-8" ?>
2     <category>
3         <pattern> * Foodie </pattern>
4         <template>
5             <random>
6                 <li> Hi there, what would you like to do today? </li>
7                 <li> What can I do for you? </li>
8             </random>
9         </template>
10    </category>
11    </aiml>

```

Figure 2.4: A sample AIML script.

Algorithm Based

Pattern matching classifiers require one response per pattern, i.e., if two sentences conveyed the same intent, they would still need to be specified as two separate instances with individual responses. Algorithm-based approaches for intent classification use probabilistic classifiers such as the Multinomial Naïve Bayes [35] that models the word frequency in documents. Given a set of sentences belonging to a class, and a new input sentence, the Multinomial Naïve Bayes algorithm can account for commonality between the user input and sentences and assign scores. The probability of a document d being in class c is computed as follows:

$$P(c|d) \propto P(c) \prod_{k=1}^{n_d} P(t_k|c) \quad (2.1)$$

where $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c . In the case of text classification, the Multinomial Naïve Bayes algorithm is termed multinomial since sentences or “bag of words” fit a multinomial distribution.

Neural Networks

Artificial Neural Networks, motivated by the structure of our brain, are networks of processing units (neurons) with weighted connections (synapses) between them [39]. Multiple layers between the input and output are trained by the input data in repeated iterations to produce outputs with greater accuracy and lower error rate (shown in Figure 2.3.2). Even though this is not a new concept, processors and memory are much faster and cheaper now, which makes neural networks a viable option for classification

```

1 Initialize weights
2 For each training case i
3   Present input i
4   Generate output  $O_i$ 
5   Calculate error ( $D_i \hat{=} O_i$ )
6   Do while output incorrect
7     For each layer j
8       Pass error back to neuron n in layer j
9       Modify weight in neuron n in layer j
10  EndDo
11  i = i + 1

```

Figure 2.5: Pseudocode for training an Artificial Neural Network

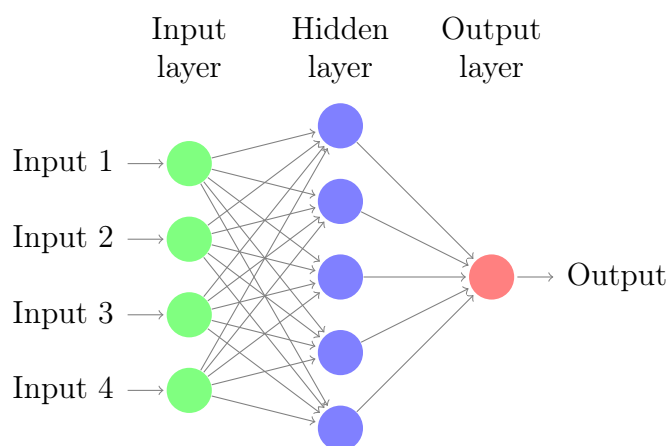


Figure 2.6: An Artificial Neural Network

of intents.

2.3.3 Entity Extraction

Entity extraction, also known as Named Entity Recognition (NER) is a task related to information extraction that seeks to locate and classify named entities in text into pre-defined categories such as names of persons, organizations, locations, expressions of times among others [10]. From investigations into open source conversational AI services,³ named-entity recognition is performed with libraries such as `spaCy`,⁴ `MITIE`,⁵ `sklearn-crfsuite`,⁶ and `duckling`.⁷ The `sklearn-crfsuite` and `MITIE`

³<https://rasa.com/docs/nlu/entities/>

⁴<https://spacy.io>

⁵<https://github.co>

⁶<https://github.com/TeamHG-Memex/sklearn-crfsuite>

⁷<https://duckling.wit.ai/>

is used for recognizing custom entities in a conversational agent [20]. The `spaCy` library is known to have pre-trained entity-recognizers which are useful for extracting places, dates, people, and organisations [14]. The `duckling` library can recognize dates, amounts of money, durations, distances and ordinals.

2.3.4 Domain

Generalist (Open Domain)

Conversational applications that are not restricted to a particular context can be termed as generalists. They are usually built using generative models making use of a large body of knowledge for training.

Digital assistants developed by major tech giants such as Google’s Assistant and Amazon’s Alexa serve as examples for Open Domain applications. Users can perform many tasks, such as set alarms and reminders, search for nearby restaurants, send text messages, and get real-time updates on weather, traffic, and sports. For home automation, Google Home and Amazon’s Echo are quite popular [17]. Amazon’s Alexa is an Internet-based voice assistant for home automation. It can accomplish tasks like switching lights off and on, playing music, and maintaining a thermostat. Google Home is another voice-based personal assistant driven by Google Assistant. They have built-in applications that give customized results (e.g., weather, news). However, for information unavailable in these apps, the assistants redirect a user to a collection of web results.

Specialist (Closed Domain)

Closed domain conversational applications are used for answering specific queries in relation to a product or a service. These can be easily created using retrieval based models, by modeling frequently occurring scenarios, thereby reducing human intervention. Commercial applications allow user-driven two-way conversations that lead to more engaged and vested users.⁸

⁸<https://chatbotsjournal.com/global-chatbot-trends-report-2017-66d2e0ccf3bb>

2.4 AI-as-a-Service

The popularity of conversational applications to a high extent is due to the services available for their easy creation. A number of services like IBM’s Watson Assistant, Microsoft’s LUIS, Amazon’s Alexa Skills, Facebook’s wit.ai, Google’s Dialogflow provide the basic building blocks to augment an application with conversation. There are multiple smart services available including sentiment analysis, retrieve web results, analyze the tone of the conversation, recognize objects and process voice. Companies like IBM, Microsoft, Google and Amazon provide working and scalable implementations of these services. Together, they make up what is called AI-as-a-service (AIIaaS).

This thesis focuses on services that aid the development of conversational applications. For FOODIE, we made use of IBM Watson’s services of Conversation, Speech and Retrieve and Rank. The following section describes these Watson services in detail.

2.4.1 IBM Watson Services

IBM offers enterprise-grade AI services to their clients and offers academic licenses to university students. Popular conversational applications build using Watson include Staples’ “Easy System” which features an intelligent ordering interface and product support ⁹ and Autodesk’s virtual agent¹⁰ which applied deep learning and natural language processing techniques to answer customer enquiries.

Watson Assistant

Watson Assistant provides cloud-based services for developers to architect complete conversational solutions for their applications. At its core, Watson Assistant processes natural language to figure out user intents and direct the conversation accordingly. The framework is supported in major programming languages such as Java, Node, Python, .NET and Unity through their Software Development Kits (SDKs). The IBM Cloud is utilized for storage of dialog nodes, rules and carrying out the processing. Watson Assistant provides a helpful graphical interface to make the dialog skeleton and specify the parameters for the conversational components.

⁹Staples’ Easy Button Case Study <http://ecc.ibm.com/case-study/us-en/ECCF-WUC12550USEN>

¹⁰Autodesk’s Virtual Agent: <http://ecc.ibm.com/case-study/us-en/ECCF-WUC12544USEN>

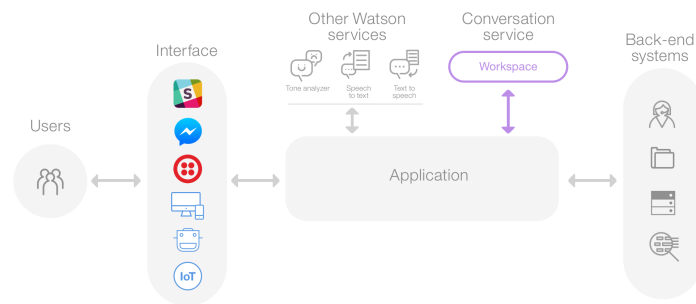


Figure 2.7: A conversational application built using Watson Services

Voice Services

IBM provides the state-of-the-art voice services in the form of Speech-to-Text¹¹ and Text-to-Speech¹² that can be connected to a conversational application in the same way Watson Assistant does. Voice services are supported in major formats such as MP3, Web Media (WebM), Waveform Audio File Format (WAV) and Free Lossless Audio Codec (FLAC). It provides services to recognize users, spot keywords, accent recognition and has support for multiple languages. This is useful add-on when conversational applications do not have information regarding a user query. Watson Discovery can be configured to query and crawl data to direct the user to better information sources.

2.5 Summary

This chapter described the background and the relevant related work for this thesis. We first described the concepts related to conversational technologies and highlighted the differences between chatbots, conversational agents and conversational interfaces. We described the importance of context-awareness, the various models used for the representation of context, and the requirements for a rich conversation. Next, we described ways of classifying conversational agents based on the perspectives of functionality, intelligence, methods of parsing input and the environment in which they operated in. At the end, we described AI-as-a-service that is introduced again in detail in Chapter 3.

¹¹<https://www.ibm.com/watson/services/speech-to-text/>

¹²<https://www.ibm.com/watson/services/text-to-speech/>

Chapter 3

Conversational Frameworks

3.1 Motivation

Gartner predicts that by 2020, customers will manage 85% of their relationship with an enterprise without interacting with a human.¹ The article also says that intelligent conversational interfaces will drive customer interaction. With this, there has been a rise in the number of conversational AI platforms that provide support to build conversational agents. Figure 3.1 shows the different kinds of conversational AI platforms that were available in 2017. Since then the number of platforms and frameworks has only increased. While a study of the entire ecosystem of tools available for building conversational frameworks would be a never-ending task, we discuss a few of the conversational frameworks that are effective and offer distinguishing features. This chapter, although not exhaustive, is intended as a guide to choosing the kind of framework for building a conversational agent. We choose to study the following frameworks in this chapter: IBM Watson Assistant,² Rasa NLU,³ Dialogflow,⁴ and Microsoft Bot Framework.⁵

¹https://www.gartner.com/imagesrv/summits/docs/na/customer-360/C360_2011_brochure_FINAL.pdf

²<https://www.ibm.com/watson/ai-assistant/>

³<https://Rasa.com>

⁴<https://dialogflow.com/>

⁵<https://dev.botframework.com/>

[H]

Chatbots Landscape

March 2017
Powered by Lihsin Chen



Figure 3.1: Chatbot landscape 2017

3.2 Key attributes of conversational frameworks

We look at the following attributes of a conversational framework for our comparative study.

- **Ease of use:** Ease of use is defined with reference to beginner to intermediate developers of conversational agents.
- **Algorithms used for natural language understanding:** Every framework makes use of natural language understanding algorithms that power the conversational agents.
- **Processing:** We look at where the processing of the natural language understanding is happening, which can either be on cloud servers, local servers or on-premises.
- **Context integration:** This lets us know whether contextual variables can be integrated out-of-the-box or need additional logic to define them.
- **Data security:** This feature describes where the conversational metadata and the dialog data is stored when using the framework.
- **Distinguishing features:** We also describe some of the distinguishing features or selling points of the framework.
- **Integrations:** This describes out-of-the-box front-end integrations and third-party database support by the frameworks.
- **Usage license:** This parameter describes the usage licenses for the frameworks and their natural language understanding algorithms

3.3 Dialogflow

Dialogflow⁶ is the developer of Google's Natural Language Understanding service that provides capabilities of building conversational agents. Dialogflow was formerly known as API.AI and Speaktoit. Speaktoit is the technology behind the Google

⁶<https://dialogflow.com/>

Assistant.⁷ It was later made open to developers via their API.AI service before becoming into the Dialogflow service. Dialogflow provides a web-based graphical user interface to design conversational agents. It uses the concepts of Agents, Intents, Entities, Context, and Fulfillment. Figure 3.2 shows the workflow of a Dialogflow Agent. Dialogflow also provides a RESTful API to integrate it with existing applications.

Agents

Agents are Natural Language Understanding modules in Dialogflow. These modules can be included in an app, website, product, or service and translate text or spoken user requests into actionable data.⁸ This translation occurs when a user's utterance matches an intent within the agent. Agents consist of the definitions for intents, entities, context, and details of fulfillment. Agents can be imported and exported as JSON files and can be merged together by importing one agent into another.

Intents and Entities

An intent represents one dialog turn within the conversation. Dialogflow provides a graphical interface to define intents and example utterances for each of them. Dialogflow also provides Welcome and Fallback intents. Welcome intents can be configured to greet a user at the start of the conversation. Fallback intents are triggered if a user's input is not matched by any of the regular intents or if it matches the training phrases below. Dialogflow uses training phrases as examples for a machine learning model to match users' queries to the correct intent. The machine learning model checks the query against every intent in the agent, gives every intent a score, and the highest-scoring intent is matched. If the highest scoring intent has a very low score, the fallback intent is matched.

Context

Dialogflow provides contextual integration in its web development environment itself. Contextual data is defined in terms of input and output. An input context tells Dialogflow to match the intent only if the user utterance is a close match and if the context is active. An output context tells Dialogflow to activate a context if it's not already active or to maintain the context after the intent is matched. Contexts are

⁷https://assistant.google.com/intl/en_ca/

⁸<https://dialogflow.com/docs/agents>

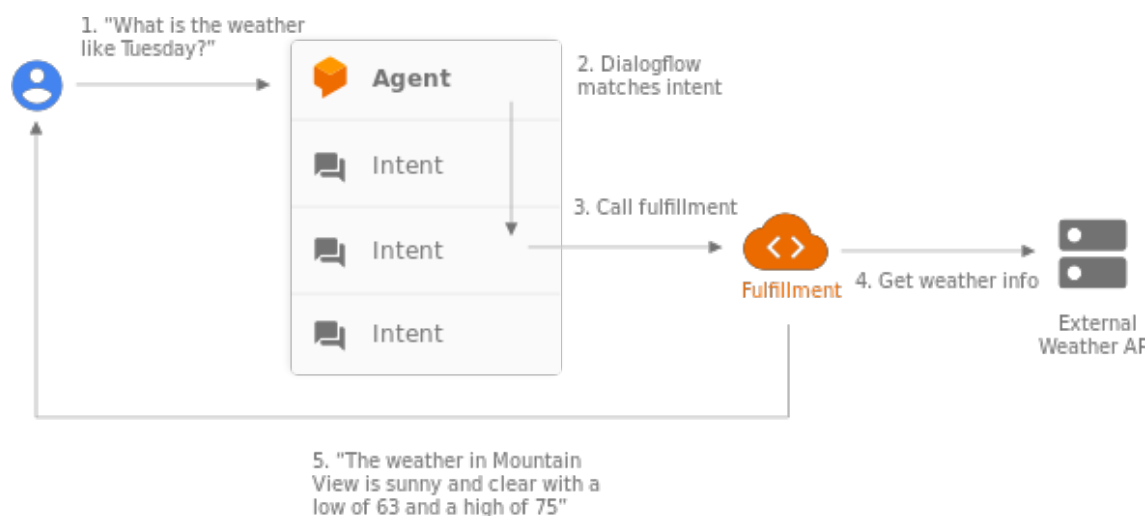


Figure 3.2: Dialogflow

helpful to control the order of intent matching and to create different outcomes for intents with the same training phrases.

Fulfillment

Dialogflow uses the concept of fulfillment for integration with third-party databases to generate dynamic responses to users queries. Fulfillment can be used in any Dialogflow agent with the help of webhooks which services requests, processes them, and returns responses. When an intent with fulfillment enabled is matched, Dialogflow makes an HTTP POST request to the webhook with a JSON object containing information about the matched intent.

3.3.1 Key Attributes

The following are the key attributes of the Dialogflow conversational AI framework:

- **Ease of use:** Dialogflow is easy to use for building prototypes of conversational agents. There is a lot of functionality that is built-in which makes the development of a conversational agent easier. However, the addition of many intents, entities and context may make the development of a conversational agent more challenging.
- **Algorithms used for Natural Language Understanding:** Dialogflow makes use of machine learning to identify intents from utterances. The details

of specific algorithms used are closed-source. However, Dialogflow gives the developer the opportunity to define machine learning thresholds and set match modes for intents. Dialogflow allows the developer to switch between hybrid (rule-based and machine learning) and Machine Learning only modes for intent matching.

- **Processing:** The computational power required by the Dialogflow Agent is provided by the Google Cloud.⁹ No processing is done on the device where Dialogflow is invoked.
- **Context integration:** Dialogflow provides context integration out-of-the-box in the form of input and output contexts.
- **Data security:** All data related to the Dialogflow agent is stored in the Google Cloud. Creating an Agent requires Google credentials. User data is stored per session of the conversation. Persistent storage can be added externally to store specific users and their conversations.
- **Distinguishing features:** One of the features of Dialogflow is that it provides Small Talk. Small Talk is used to provide responses to casual conversation. When Small Talk is enabled, the Dialogflow agent can respond to queries like “How are you?” and “Who are you?” without defining them as separate intents. Dialogflow also provides the feature of Pre-built agents. Pre-built agents are a collection of agents developed to cover specific use cases. These can be appended to a Dialogflow agent and modified according to the developers’ needs.
- **Integrations:** Dialogflow can be integrated with Actions on Google¹⁰ which is a platform where developers can create software to extend the functionality of the Google Assistant. Dialogflow also provides Cortana and Alexa integration, with the ability to import/export Intent schemas and sample utterances to Dialogflow. Dialogflow provides out-of-the-box support for platforms such as Facebook Messenger, Slack, Line, and Kik.
- **Usage license:** Dialogflow is provided as a Standard (free) version and an Enterprise (paid) version. The dialogflow software is closed-source and uses proprietary algorithms for machine learning.

⁹<https://cloud.google.com/>

¹⁰<https://dialogflow.com/docs/integrations/actions>

3.4 IBM Watson Assistant

Watson Assistant ¹¹ is the natural language understanding and bot building tool provided by IBM. It is built on a neural network of one billion Wikipedia words, understands intents, and interprets entities and dialogs. IBM Watson also provides services for tone analysis, speech-to-text, text-to-speech, machine learning and visual recognition among other services.

Intents

Intents define the purpose of the user's input. The intent of a user is defined by providing the intent classifier in Watson training data. For example, "What is the weather today?" and "What is the temperature today?" are designated for training Watson to classify the intent as finding out the weather.

Entities

Entities are specific keywords in a conversation. Entities provide additional context to an intent. Given a sentence, "What is the weather in Hyderabad?", we have additional information, there is a location that is specified along with the intent of finding the weather.

Context Variables

Context variables are JSON snippets that are transferred back and forth between the application and Watson Assistant which persists only during the duration of the conversation.

Dialog

Dialog provides the structure for possible flows in a conversation in the form of *nodes* connected by *directed edges*. These flows define how the application is to respond when it recognizes the defined intents and entities. Nodes are conditionally triggered usually based on intents or entities or a combination of both.

¹¹<https://www.ibm.com/watson/ai-assistant/>

Workspace

The workspace is where the entire conversation including dialog, intents, entities, and context variables are defined. The entire workspace is translated to JSON. An application connects to this workspace to enable the conversational components.

3.4.1 Key Attributes

- **Ease of use:** Easy. Watson Assistant provides a graphical user interface to design and develop intents, entities, and dialog which makes the building of a prototypical conversational agent easy.
- **Algorithms used for natural language understanding:** Watson Assistant makes use of proprietary algorithms which employ deep learning techniques for natural language understanding. Most of the intelligent services are derived from IBM Watson, which is a question-answering computer system capable of answering questions posed in natural language, developed in IBM's DeepQA project.
- **Processing:** All processing related to natural language understanding happens in the IBM Cloud.¹²
- **Context integration:** Watson Assistant provides support for contextual variables while building dialog. Expressions in context are powered by the Spring Expression Language (SpEL).
- **Data security:** Conversational data is stored only per session and databases can be integrated for persistence (i.e., to store previous conversations and users). The metadata the conversational agent is stored in the IBM Cloud. Continuous integration and delivery can be set up out-of-the-box if it is connected to a version controlled repository (such as those on GitHub)
- **Distinguishing features** Watson Assistant provides sophisticated intent recognition since it is trained on a large amount of data. It provides the entire stack of services, for example, with IBM Node-Red, Watson Assistant, IBM Speech-to-Text and Text-to-Speech a fully functional conversational agent can be set up easily without leaving the IBM umbrella.

¹²<https://www.ibm.com/cloud/>

- **Integrations** Watson Assistant can be integrated with Slack, Facebook messenger, Kik or custom user interfaces. Third party databases can also be integrated via their APIs for dynamic responses.
- **Usage license** Watson Assistant operates on a freemium model. A certain number of API calls to Watson Assistant are free, after which each call is charged. The Watson Assistant framework and algorithms used are closed-source. However, Watson Assistant has Software Development Kits supporting multiple major languages such as Java and Python.

3.5 Rasa NLU

The Rasa stack,¹³ developed by Rasa Technologies provides natural language understanding for intent classification, entity extraction and managing contextual dialog in conversational agents. The Rasa stack is open-source in its entirety. The Rasa NLU employs the concepts of intents, entities and context similar to that of Watson Assistant.

3.5.1 Key Attributes

- **Ease of use:** Moderate. Developing a prototype using the Rasa stack is straightforward. However, there is no graphical user interface. Instead, the paths a conversation might take (called stories) are defined in markdown format. Then a domain is defined in which the intents and entities of the conversational agent are defined. After this, the model is trained on the example stories defined. Rasa offers customizations for the minutest details due to which the difficulty of use increases as the conversational agent gets more custom.
- **Algorithms used for Natural Language Understanding:** Rasa provides the ability to configure the natural language understanding pipeline. The two most important pipelines are `spacy_sklearn` and `tensorflow_embedding`. The `spacy_sklearn` pipeline uses pre-trained word vectors from either GloVe¹⁴ or fastText.¹⁵ The `tensorflow_embedding` pipeline doesn't use any pre-trained

¹³<https://Rasa.com>

¹⁴<https://github.com/stanfordnlp/GloVe>

¹⁵<https://github.com/facebookresearch/fastText>

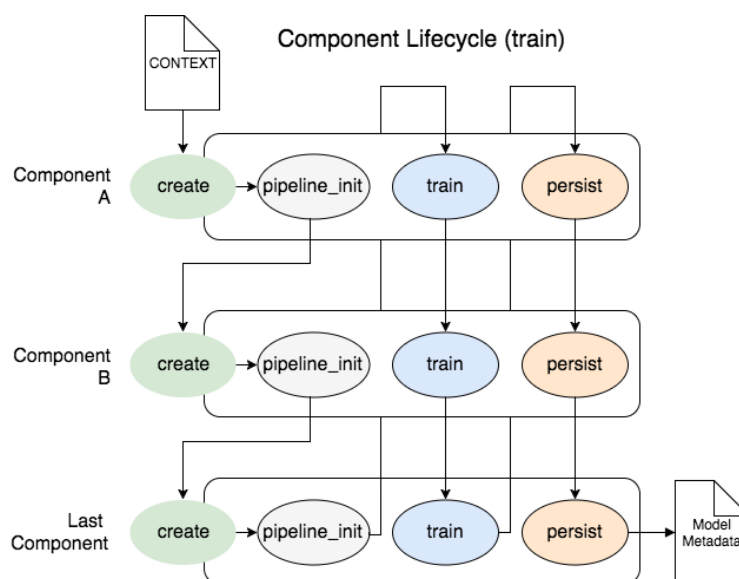


Figure 3.3: Rasa component lifecycle¹⁶

word vectors but instead fits these specifically for the dataset defined. Other pipelines such as `mitie`, `mitie_sklearn` or a custom pipeline can be used to provide necessary functionality.

- **Processing:** The Rasa stack can be configured on local servers or a private cloud.
- **Context integration:** Context is provided as a dictionary variable and is used to pass information between components. Initially, the context is filled with all configuration values, Figure 3.5.1 shows the call order and visualizes the path of the passed context. After all the components are trained and persisted, the final context dictionary is used to persist the model's metadata.
- **Data security:** Rasa provides full control over user data. Since it allows deploying on-premises or in a private cloud, valuable training data is not sent to third-party APIs.
- **Distinguishing features:** Using Rasa, natural language understanding models can be tweaked and customized. It is entirely open-source, unlike most frameworks. It provides complete data security. When it is run locally, an advantage is that an extra network request need not be made for every message.

¹⁶https://Rasa.com/docs/nlu/master/choosing_pipeline/#section-component-lifecycle

- **Integrations:** Rasa can be integrated with messaging platforms like Facebook, Slack or Telegram by using “standard connectors”. Third party databases can be integrated with connectors for dynamic responses to users.
- **Usage license** The Rasa stack is entirely open-source. Paid enterprise grade solutions and custom APIs are available for large scale solutions.

3.6 Microsoft Bot Framework

The Microsoft Bot Framework¹⁷ is an open-source framework that can be used to build conversational agents. Apart from Natural Language Understanding, the cognitive services of Spell Checking, QnA maker, Speech API, Text analytics and Web search are also provided by Microsoft for integrating with the Bot Framework. The Framework supports direct line API support for integrating existing software with a conversational agent. The Microsoft Bot Framework also employs the concepts of intents, entities and context similar to that of Watson Assistant and will not be redefined here.

3.6.1 Key Attributes

- **Ease of use** The Microsoft Bot Framework has a moderate learning curve. Large parts of the conversational agent have to be programmed, only the creation of the service and templates are provided out-of-the-box with the Graphical User Interface of Microsoft Azure. The platform for dialog creation is provided as a separate service, called LUIS(Language Understanding Service).¹⁸ Microsoft Bot Framework allows development in Node.JS and C# through their built-in code editor. They provide SDKs for other languages like Java and Python.
- **Algorithms used for Natural Language Understanding:** The Microsoft Bot Framework uses the Microsoft Language Understanding Service (LUIS) to recognize intents and entities. LUIS is a machine learning-based service to build natural language understanding into apps, bots, and IoT devices. The

¹⁷<https://dev.botframework.com/>

¹⁸<https://luis.ai>

algorithms used by LUIS for natural language understanding are closed-source and proprietary. However, the Bot Framework is open-source.

- **Processing:** Like Dialogflow and IBM Watson Assistant, all processing related to machine learning and natural language understanding happens on the cloud (Microsoft Azure, in this case), while the Edge (devices accessing the Agent) does not perform any computation.
- **Context integration:** The Microsoft Bot Framework does not currently support context integration out-of-the-box.
- **Data security:** All data related to the conversational agent is stored in the Azure Cloud. Creating an Agent requires Microsoft credentials. User data is stored per session of the conversation. Persistent storage can be added externally to store specific users and their conversations.
- **Distinguishing features:** Azure Bot Service provides a scale-on-demand, integrated connectivity, and development service for intelligent bots that can be used to reach customers across multiple channels. The Azure Bot Services speeds up development by providing an integrated environment that's purpose-built for bot development with the Microsoft Bot Framework. With the Azure cloud services, auto-scaling for the conversational agents is enabled. This means that the agents that are deployed on Azure perform well because the application is dynamically scaled based on demand. Continuous Integration and continuous delivery can be integrated by connecting the agent to a version control system like Github.
- **Integrations:** The Microsoft Bot Framework supports front-end integration for applications like Skype, Slack, and Facebook Messenger.
- **Usage license:** The Microsoft Bot Framework is free to use, however, the Azure Bot Service is priced according to the pricing of Azure services. It is provided as a free tier as well as a premium paid tier.

3.7 Summary

In this chapter, we discussed some of the popular and effective frameworks for building conversational agents namely IBM Watson Assistant, Rasa NLU, Microsoft Bot

Framework, and Google Dialogflow. Each framework was judged by its ease of use, internal algorithms used, distinguishing features, integrations, contextual support, data security, and the usage license.

Chapter 4

Smart Conversations

FOODIE is a cognitive text-and-voice based conversational agent that augments the capabilities of home cooks by incorporating health-related information to facilitate healthy eating habits. Our conversational agent is logically divided into two components—FOODIENLU and FOODIECORE. This chapter focuses on the conversational component of FOODIE, FOODIENLU. FOODIE makes use of IBM Watson’s conversational services (Watson Assistant, Text-to-Speech, and Speech-to-Text) for its state-of-the-art natural language understanding. We first briefly define our design methodology. Then, for each step in our design methodology, we describe how FOODIE’s conversations are orchestrated.

4.1 Design Methodology

We propose the following design methodology for the conversational component of our application FOODIENLU. FOODIENLU utilizes the services of IBM Watson Assistant to architect its conversational component. As described in Chapter 2, Watson Assistant provides the concepts of *intents*, *entities*, *dialog*, *context* and *workspaces* to develop conversational applications. We identify five key steps in the development of the framework for FOODIENLU which can be generalized to any conversational application.

Step 1: Identify objectives. The first step is to identify the objectives that we would like our conversational application to accomplish, analogous to requirements gathering in software engineering. These are defined as high-level goals

with a set of acceptance criteria for success. The conversational model for FOODIENLU is fleshed out based on these goals.

Step 2: Identify the intents for each objective. For each of the high-level objectives defined in Step 1, we identify intents, which are fine-grained descriptions of the goals. All the intents defined for the goals should be logically mutually exclusive from each other. This is done to ensure that identifying and classifying intents is not ambiguous.

Step 3: Identify the representative questions/statements for each intent.

Next, we identify the utterances that a user might say to communicate the goal or intent. These utterances are used to train the models that recognize the intent accurately. These examples are clustered under each intent. Ideally, five to ten examples or utterances are identified for training each intent. Like the previous step, sentences expressing similar ideas should not be grouped into different intents to reduce errors of classification.

Step 4: Identify entities corresponding to each intent. In each question or utterance, we identify the keywords that need to be recognized. The central idea of keyword identification is this: if the agent's response varies because of a certain word in the sentence, then those words must be recognized as entities. Similar entities can be grouped together. For example, `cuisine` is an entity and can recognize values like `African` and `Indian`.

Step 5: Design dialog for each intent. Once the intents and the entities have been identified, we design the conversation flow for each intent. Intent nodes are the first nodes that are encountered when the conversation starts unless there is a dependency between them (in which case, some intent nodes can be in the later parts of the conversation workflow).

Step 6: Identify context variables for each node in the dialog. For each of the dialog nodes, we identify whether there is a need for a context variable to be set. Context variables are used when there is more information that has to be sent to the application in addition to intents and entities.

The next few sections define each of these steps in the context of FOODIENLU.

4.2 High-level goals/objectives of FOODIE

We identify the following objectives that are to be accomplished by our conversational agent.

- G1: Manage dietary requirements and preferences.** FOODIE allows users to set and update parameters according to their diet and taste preferences. These parameters include diet requirements (e.g., vegan, vegetarian), food allergies, excluded ingredients (e.g., dislikes), cuisine style, and budget.
- G2: Manage dietary goals.** Users may ask FOODIE to add or remove or update their dietary goals. For instance, “Foodie, I would like to decrease my weight.”
- G3: Check for available ingredients.** Users may ask FOODIE for details about the current ingredients in the fridge (e.g., when a certain product expires or how much of a product is left). Ideally, FOODIE would be connected to a Smart Fridge that provides services to obtain such details.
- G4: Retrieve recipe recommendations.** Users may ask at any time for new recipe recommendations. These recommendations take into account personal preferences such as allergies and cuisine while retrieving recipes.
- G5: Get nutritional information.** Users may ask FOODIE to retrieve nutritional information about a food product or a dish.
- G6: Check for recalls.** Users may ask FOODIE to identify products that have been recalled or might want to be alerted when a food item they possess has been recalled.
- G7: Manage medical conditions.** Users may ask FOODIE to set a new medical condition, such as diabetes and obesity, although this is out of scope for our current prototype.

4.3 Intents

Next, we define our list of intents based on our goals. Intents represent atomic actions that can be defined to accomplish a goal or a part of the goal. Intents define the purpose of the user’s input. For example, for the goal G7, “retrieve recipe recommendation”, the following intents are identified:

Table 4.1: List of goals mapped to their intents

Goal	Intent	Description
G1	diet_preference	Manage user’s dietary preferences such as allergens and cuisine
G2	dietary_goal	Manage the user’s goals (set/update/delete)
G3	ingredient_availability	Retrieve information about available ingredients at home
G4	recipe_recommendation	Provide a recipe recommendation based on the user’s request and preferences
G4	recipe_steps	Provide the steps to a particular recipe.
G5	nutrition_info	Provide nutritional information for the product or dish asked for by the user
G6	recall	Provide details about a publicly recalled item
G7	medical_condition	Manage the user’s medical conditions (set/update/delete)
G8	anything_else	Manage response for unknown requests

- Retrieve a suitable recipe (`#recipe_recommendation`)
- Provide step-by-step instructions to cook the recipe `#recipe_steps`

For example, if a user says “Can you suggest a recipe to me?” or “I’d like to cook something,” FOODIE recognizes that the intention here is to retrieve a recipe. We define a few examples for each goal or intent that can be communicated to the users. Watson Assistant uses state-of-the-art deep learning classifier in the IBM Cloud for providing Natural Language Understanding to classify an intent. Our workspace on Watson Assistant classifies this intent as `recipe_recommendation`. In our *workspace*, we define intents and different examples for each intent. This trains Watson to recognize intents of the user’s input sentence with certain probabilities. Since Watson Assistant comes with natural language understanding, when a user says “I’m hungry,” although this does not feature in the predefined examples for the intents, Watson correctly classifies this as a `recipe_recommendation` intent. In a similar fashion, we define different intents for the different functionalities of FOODIE. Some of them include the `dietary_goal` intent for setting, updating or removing the user’s dietary goals or the `nutrition_information` intent that is defined for recognizing if the user has asked for nutrition information regarding a recipe or a product.

Listing 4.1: The “start_cooking” intent and its example utterances

```
1 {
2   "intent": "start_cooking",
3   "examples": [
4     {"text": "what should I cook?"},
5     {"text": "lets cook something"},
6     {"text": "let's cook"},
7     {"text": "i want to cook"},
8     {"text": "i want a recipe"},
9     {"text": "i need a recipe suggestion"},
10    {"text": "I don't know what to cook"},
11    {"text": "give me a recipe"},
12    {"text": "what's a good recipe"},
13    {"text": "I'm famished!"},
14    {"text": "I'm starving!"},
15    {"text": "what should I eat?"}
16  ],
17  "description": "Intent to retrieve a recipe"
18 }
```

Listing 4.2: The “no” utility intent and its examples

```
1 {
2   "intent": "no",
3   "examples": [
4     {"text": "I don't know"},
5     {"text": "not interested"},
6     {"text": "I don't like it"},
7     {"text": "not"},
8     {"text": "nah"},
9     {"text": "no"},
10    {"text": "nope"},
11    {"text": "no thanks"},
12    {"text": "no way"},
13    {"text": "I don't know what I want"},
14    {"text": "don't like"},
15    {"text": "i disagree"}
16  ]
17 }
```

4.3.1 Utility Intents

We identify utility intents that are generic to every text-and-voice based conversational agent and help keep a smooth flow of the conversation. A list of the utility intents used by FOODIE are listed in Table 4.2. One example of a utility intent is the **repeat** intent. Since FOODIE supports voice, there may be instances where a user does not understand or hear what FOODIE says and would like FOODIE to repeat the sentence. The **repeat** intent is recognized when a user says something along the lines of “Could you repeat that?” which results in the repetition of the sentence. Another key utility intent is **reset**, when the user changes her mind and wants to restart the conversation. There are intents like **yes**, **no**, **previous**, and **next** which are also conveyed often and thus are recognized as intents.

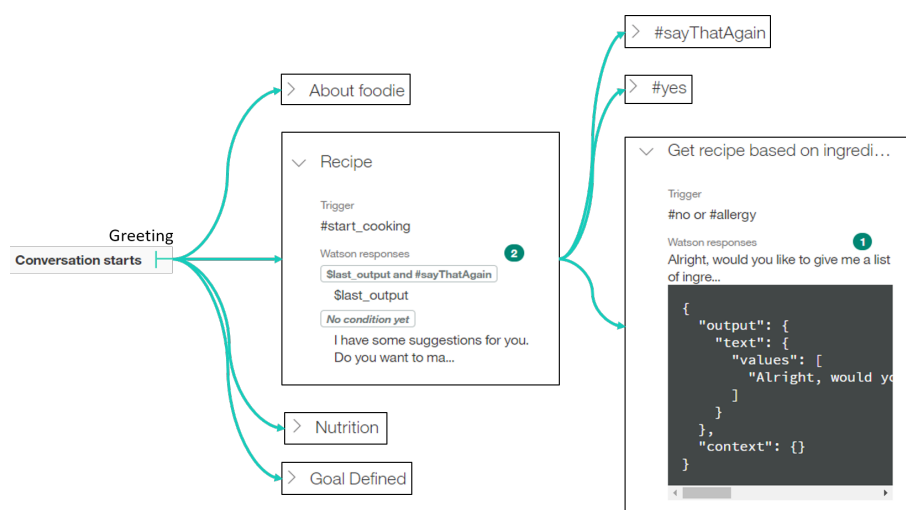


Figure 4.1: A subset of nodes in FOODIE's Conversation

Table 4.2: List of utility intents

Utility intent	Description
#yes	Positive response
#no	Negative response
#reset	Reset the conversation
#repeat	Repeat sentence
#next	Next step
#previous	Previous step
#about	About FOODIE's functionality
#anything_else	Unknown/out-of-context intents

4.4 Entities

Entities are meant for keyword identification. Entities provide additional context to an intent. Often, when a user wants to convey something, they provide details about their intention in the sentence itself. For example, there is a difference between “I want to eat” and “I want to eat a French dinner.” Although the intent in both of these scenarios is recognized as `recipe_recommendation`, there is additional information in the second sentence, `french` is a cuisine and `dinner` is a type of meal. Some of the entities in FOODIE's conversation workspace are cuisine, type of meal, allergies, kind of diet, goals and goal types.

Listing 4.3: The entity description for the possible diets that a person can be on

```

1 {
2   "entity": "diet",
3   "values": [
4     {"value": "primal"},
5     {"value": "vegan"},
6     {"value": "ovo vegetarian"},
7     {"value": "paleo"},
8     {"value": "pescaterian"},
9     {"value": "vegetarian"},
10    {"value": "lacto vegetarian"}
11  ],
12  "metadata": null,
13  "description": "The types of diets"}
14 }
```

Table 4.3: List of entities

Entity	Description	Values
@cuisine	Types of cuisine	American, British, Caribbean, Chinese, Indian, Thai
@diet	Types of diets	vegetarian, lacto-vegetarian, vegan, paleo, keto
@expectation	Rate of change of given goal	increase, decrease, maintain
@goal	Goal Modification	set, update, remove
@goalType	Type of Goal	sodium, cholesterol, protein, fat, weight
@intolerance	Types of commonly occurring intolerance	egg, gluten, seafood, sesame,
@mealType	Type of meal	breakfast, lunch, dinner, salad, soup, drink, dessert

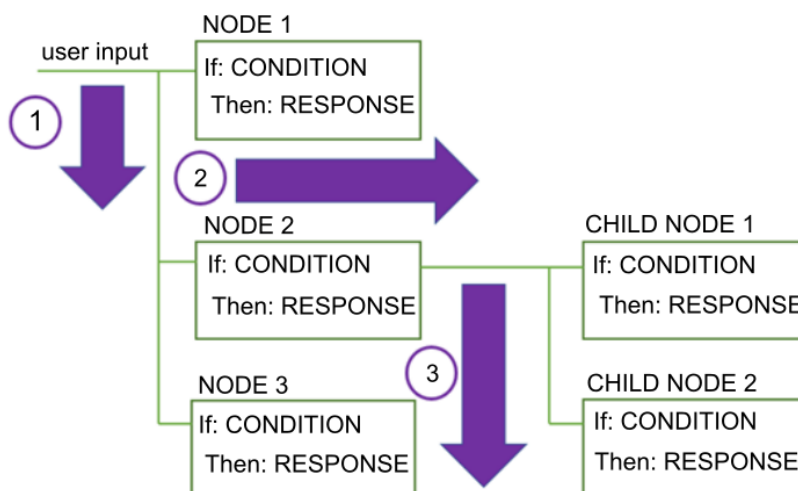


Figure 4.2: The flow of conversation as described by IBM Watson Assistant

4.5 Dialog Design

The *dialog* is the third building block for a conversation. It provides the structure for possible flows in a conversation in the form of *nodes* connected by *directed edges*. These flows define how the application is to respond when it recognizes the defined intents and entities. Nodes are conditionally triggered usually based on intents or entities or a combination of both. Figure 4.1 shows a subset of the conversation nodes of FOODIE. A conversation starts with a node for greeting (at Level 0). This is followed by nodes originating from the greeting node for the different things that a user may ask (these are the nodes at Level 1). For FOODIE, we define nodes for recipe recommendations, nutrition information, allergy information and goals among others. Each of these nodes branches off to other nodes based on how the conversation flows.

We also define a few utility nodes at each level that correspond to the utility intents. At every *level*, we define nodes for repeating (**repeat**), for resetting (**reset**) and for incomprehensible input (**anything_else**).

For every dialog node in Watson Assistant, there are four parts that have to be configured. The node in Figure 4.3 represents the “Anything Else” dialog flow. These are the actions that need to be taken when FOODIE cannot generate an appropriate response and information is not available to relay to the user (defined in Table 4.4). The syntax is in the form of an if-then-finally statement.

1. The first section describes the condition due to which the node gets triggered. For example, in the aforementioned figure, this node is triggered if `anything_else` is detected.
2. The second section describes the context variables that need to be set when this node is triggered. Figure 4.3 sets two context variables `from` and `request`. `from` represents the current node (which is `anything_else`) and `request` describes the kind of information that needs to be requested from back-end databases (which in this case is `food_trivia`; this redirects the request to an endpoint in the database that provides food facts).
3. The third section describes the possible responses that can be stated to the user (given that the user intent has been recognized as `anything_else`). As described in the figure, response variations are set to either sequential or random. Each response input is appended at run-time to responses from back-end databases, if one exists. In this case, the output of the request `food_trivia`.
4. The last section describes what is to be done at the end. Watson Assistant allows waiting on the user for input, skipping user input or jumping to a specific node. In our case, after providing food trivia to the user, there is a jump to the `greeting` node (which takes the user back to the start of the conversations).

4.5.1 Spring Expression Language (SpEL)

The Spring Expression Language (SpEL for short), a part of the Java Spring framework is a powerful expression language that supports querying and manipulating an object graph at runtime. It provides a short-hand notation that enables concise conditions in the dialog nodes. Watson Assistant supports the Spring Expression Language for expressions that access objects and properties of objects. With SpEL, we accomplish the following in FOODIENLU:

- **Evaluation of conditions.** Intents, entities and context variables can be evaluated in the “if” conditional part in a dialog node.
- **Manipulation of context variables (key-value pairs).** SpEL provides an evaluation syntax to expand variables inside other variables, or apply methods to output text or context variables. SpEL provides methods to process values of different data types such as arrays, date, time, numbers, objects and strings.

If bot recognizes:

anything_else ⊖ ⊕ 1

Then set context:

Variable	Value	
\$ from	"anything_else"	🗑
\$ request	"food_trivia"	🗑

⊕ Add variable 2

And respond with:

Text
Move: ^ v 🗑

I'm foodie. Ask me anything food related! Did you know that ⊖

I do not know the answer to that! Ask me anything food related! Did you know that ⊖

Food questions only! Here is a fun food fact: ⊖

Enter response variation

Response variations are set to random. [Set to sequential](#) 📘 3

And finally

Jump to... "Greeting" (User input) ⊖ 4

Figure 4.3: Configuration of a dialog node

Table 4.4: List of dialog nodes of FOODIE

Dialog Nodes	Description
Greeting	Represents the start of conversation
Anything Else	Actions needed to be taken when appropriate information is not available
Modify Preferences	Triggered when a user wants to manage their dietary preferences
Allergy	Triggered when the user asks information about allergies
Nutrition	Triggered when the user asks information about nutritional values in a product
Recipe	Triggered when the user would like a recipe
Goal	Triggered when the user tries to set/update a specific goal
Recalls	Triggered when there is a nationwide recall of a product

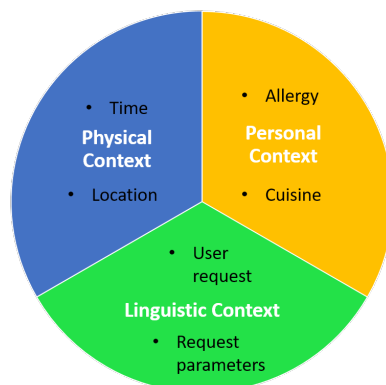


Figure 4.4: Types of context.

- **Accessing user input.** Input from the user can be accessed in the dialog nodes itself using the variable “\$output”.

4.6 Context Variables

Context is a key part of the Dialog. Context variables provide the mechanism for passing information between the dialog and the application. The relevant context variables for FOODIE can be classified into linguistic, personal and physical context variables. Each of these categories serves different functions and is set and updated in different components of the system (described in further detail in Section 5.5).

4.6.1 Linguistic Context

We define linguistic context variables as those that are passed between the Watson Assistant workspace and the application. These context variables are also passed and updated in between nodes in the dialog. The *linguistic context* is the dialog context and can be accessed in the conversation workspace using the syntax `$variable_name` (which is shorthand for `context.variable_name`).

For example, the following piece of JSON code represents contextual information that gets sent to FOODIE. The `context` key is used to pass the details of the request to FOODIECORE. The `output` key is used to identify the output that is to be sent to the user (FOODIECORE’s response, if it exists, gets appended to this output). In the case of FOODIE, decisions on the responses to the user are directed by the `request`

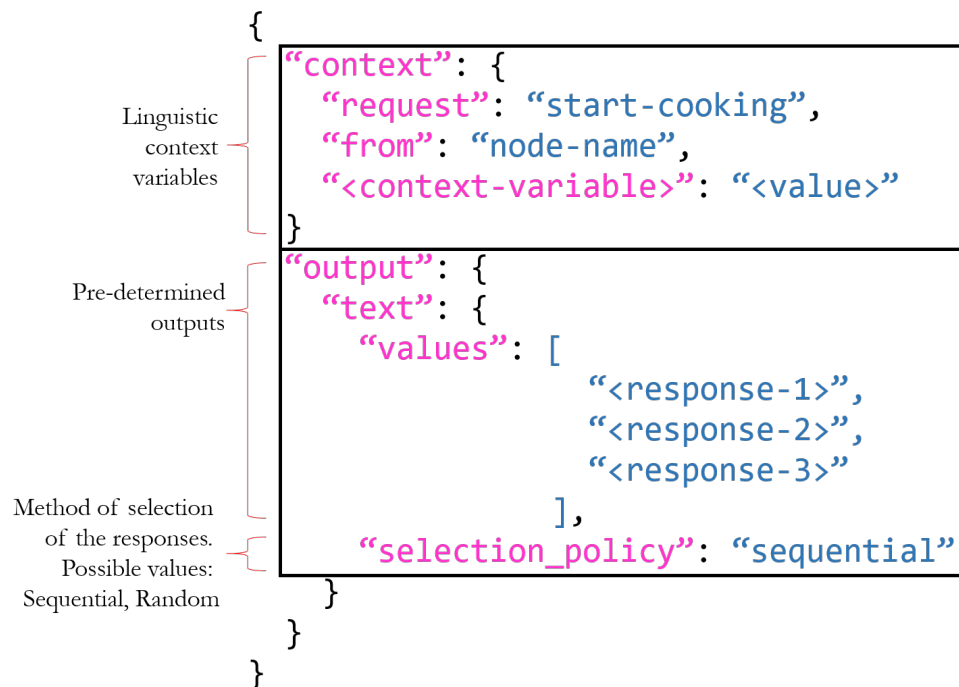


Figure 4.5: JSON code snippet depicting the linguistic context variable along with output from Watson Assistant

```

{
  "cuisine": ["Indian", "Chinese"],
  "allergy": ["peanut", "shellfish"],
  "diet": "vegetarian"
}

```

Figure 4.6: JSON code snippet depicting personal context variables

key within the context. The remainder of the keys serve as the parameters to this request. Figure 4.5 shows one such example of context variables within a dialog node. This node sends a request to FOODIE requesting to start cooking (“request”: “start-cooking”)

Table 4.5: Examples of linguistic context variables

Context Variables	
Linguistic Context	Description
\$start-cooking	When the user asks for a recipe recommendation
\$nutrition-information	When the user asks for the nutrition information
\$food-trivia	When the node defaults to Anything Else

4.6.2 Personal Context

Personal Context refers to the context variables that convey the users’ preferences. These variables are useful in terms of constraining the results obtained from our recipe and nutrition databases. Typically, the personal context would be unique to each individual and need not be set and updated each time the conversation happens. In our prototype, we specify a static JSON file that describes the personal context and serves as one of the inputs to FOODIECORE.

4.6.3 Physical Context

Physical Context refers to the context variables related to date, time and location. Watson Assistant provides system entities related to date, time and location that we leverage for physical context variables.

Table 4.6: Examples of linguistic context variables

Physical Context	System Entity	Description
date	@sys-date	The date entity recognizes various descriptions of date mentions, such as today , Friday or thirteenth July . The date can be inferred as a string of the format “MM/DD/YYYY”
time	@sys-time	The time entity is recognized with utterances such as now or 30 minutes from now . The time can be inferred as a string of the formate “HH:mm:ss”
location	@sys-location	The @sys-location system entity extracts place names (e.g., country, state/province, city, town) from the user’s input.

4.7 Summary

This chapter provided a detailed description of the building of one of the components of FOODIE, responsible for natural language understanding, called FOODIENLU. IBM's Watson Assistant is the framework that provided sophisticated natural language processing which was used by FOODIENLU for conversational support. This chapter also described FOODIENLU's building blocks which include intents, entities, dialog, and context variables. This chapter also introduced new concepts such as utility intents and provided an additional classification of the context variables in the context of FOODIE. The next chapter delves into the architecture and implementation of the whole system and provides a description of the data sources used in FOODIE.

Chapter 5

Application Architecture and Implementation

This chapter discusses the details of the implementation of FOODIE. While Chapter 4 focuses on architecting smart conversations (FOODIENLU), this chapter focuses on the overall application with FOODIENLU being one of the components of the system. In the following sections, we detail FOODIE’s requirements, application components, data sources, and application logic.

5.1 Requirements

One of the contributions of this thesis is a prototype of a conversational agent which aids users in their homes with regard to their nutritional and dietary needs. This section outlines the requirements for the software implementation of our conversational agent, FOODIE.

Requirement 1 A conversational agent that provides a capability to interact in a natural language (English) which accomplishes kitchen related goals such as suggesting a recipe, setting and updating dietary goals, retrieving nutritional information and checking for ingredient availability.

Requirement 2 A capability to have interactions using the medium of text as well as voice.

Requirement 3 A capability to augment the personalization using personal context.

5.2 Application Overview

5.2.1 Components

FOODIENLU

FOODIENLU is the natural language understanding component of FOODIE. Input is in the form of a textual sentence from the front-end. In case of Voice interaction with FOODIE, the input is first converted into text with the help of Watson Speech-to-Text. FOODIENLU exists as a Watson Assistant workspace on the IBM Cloud.¹ The workspace is entirely represented in JSON and can be accessed and updated through Watson Assistant's Graphical User Interface. The details regarding the content of the workspace, such as its intents or entities are presented in Chapter 4. Within the dialog modeling segment of FOODIENLU, the Spring Expression Language (SpEL) (cf. Section 4.5.1) is used as a shorthand notation that enables concise node conditions.

FOODIECORE

The core application code resides within FOODIECORE. This is the piece of application code that interacts with our Watson Assistant workspace (FOODIENLU). Java is used as the primary programming language to write the application code. FOODIECORE is comprised of integrations to the back-end recipe and food databases, and `foodie-orchestrator` which is responsible for parsing and redirecting requests to the appropriate databases. The data sources are explained in detail in Section 5.3 and `foodie-orchestrator` is described in Section 5.6.

5.2.2 Architecture

The complete solution architecture of FOODIE is depicted in Figure 5.2.2. The back-end comprises of integrations to our data sources (i.e., Spoonacular, Recalls and FoodEssentials) and FOODIECORE which encompasses all of the application code. FOODIECORE is connected to FOODIENLU which is our natural language understanding component. Figure 5.2.2 depicts the sequence diagram which shows the information/message flow between components of FOODIE (components are depicted as rectangles).

Users ideally interact with FOODIE via a hybrid interface (i.e., voice and text). FOODIE can be integrated with services, such as Slack, Facebook Messenger, Google

¹<https://console.bluemix.net/docs/services/conversation/services-information.html#services-information>

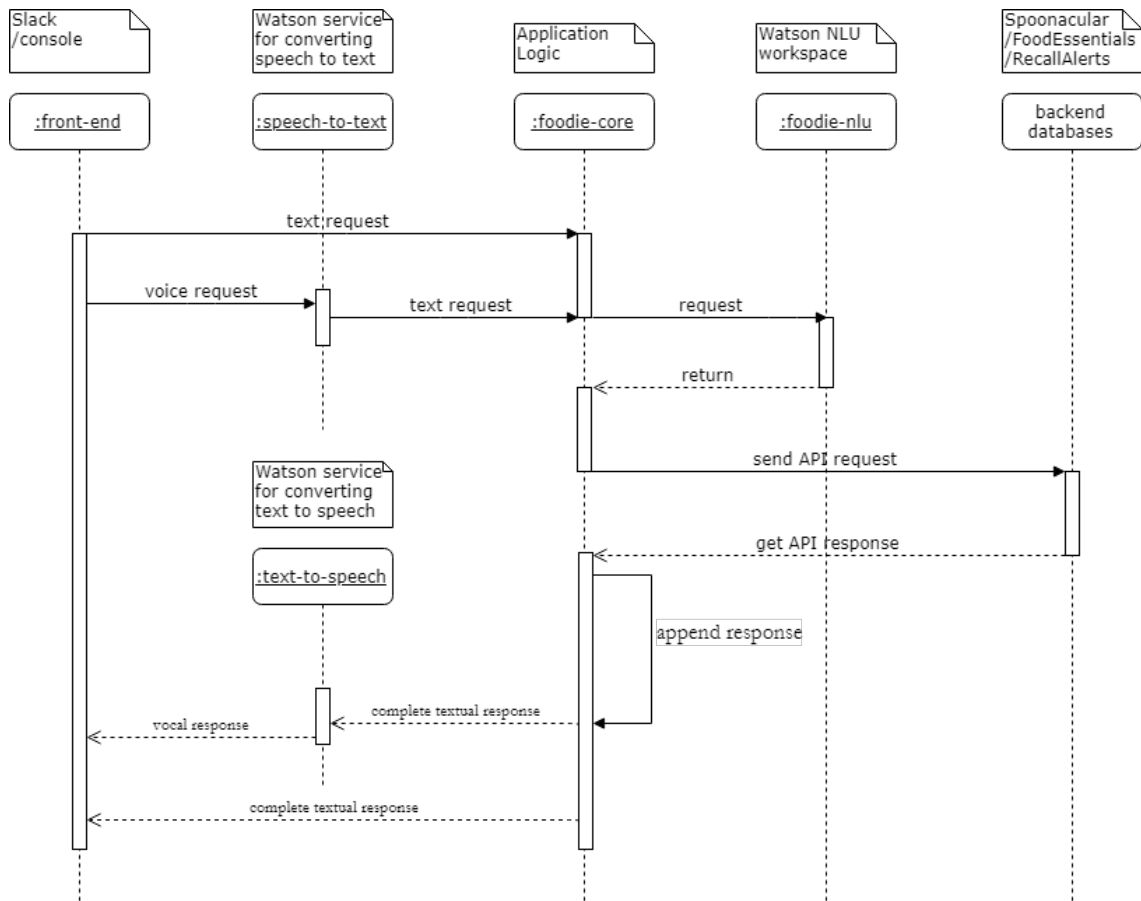


Figure 5.1: UML Sequence diagram of FOODIE

Assistant or Siri. In any case, text and voice serve as the basis of input at the front-end. If speech is used, then IBM Speech-to-Text service is used for converting voice to text. Each textual sentence serves as a request to FOODIE’s core engine. This engine takes into account, along with the text input, the user’s personal preferences from their personal context. FOODIECORE is connected to FOODIENLU, which uses the Watson Assistant platform. This platform provides services that can understand natural language input (cf. Chapter 4). Watson Assistant provides *workspaces* that are containers for the artifacts of a conversation (i.e., the dialog structure, relevant entities and intents). Input from the user is redirected to the Conversation Workspace, which responds by returning the intent of the sentence as a JSON Object.

Information from the user’s personal context and the user’s conversation with FOODIE is used to make appropriate requests to back-end systems. API requests are sent to one or more of these systems based on the user’s input as parsed by Watson

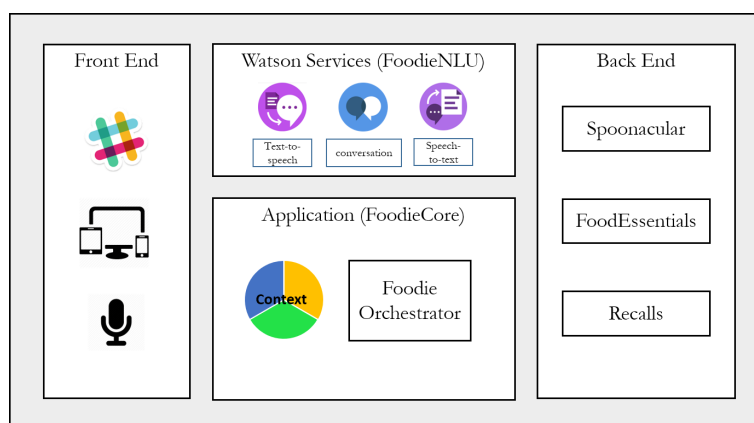


Figure 5.2: FOODIE Component Architecture

Assistant.

5.3 Data Sources

This section describes the data sources that feed our application. Our back-end comprises *Spoonacular*: a recipe database, *FoodEssentials*: an allergen database, and *Recalls and Safety Alerts*: the government of Canada’s recalls service. All of these data sources are accessed via their Application Programming Interface (API).

5.3.1 Spoonacular

Spoonacular provides a comprehensive recipe database with over 365,000 recipes and 86,000 food products. Spoonacular provides the functionality of finding recipes, nutritional requirements, ingredient specifications, and computing meal plans. Among the numerous recipe databases, Spoonacular was found to be the most useful, not only because of its vast collection of recipes but also because of its powerful search with support for natural language queries and its extensive filters to classify these recipes. We use the following services of Spoonacular for FOODIE:

- **Recipe Search:** Spoonacular provides multiple endpoints to search for recipes. The following endpoints are used by FOODIECORE to search for recipes. The endpoint to be used is decided by a user’s query.

Complex recipe search is provided as a GET method that provides a list of

recipes which can be parameterized by cuisine, diet, included/excluded ingredients, intolerance, controlled nutritional amounts, keywords and its type. Table 5.4 provides some more details regarding the query parameters used in a complex recipe search. This query is used when a user generally asks for a recipe without giving any specifications. For example, when the user says “I’m feeling hungry” or “I’d like to cook something”, the complex recipe search is parameterized by the user’s profile. When the user has a certain kind of specification like ingredients and cuisine, the following endpoints are used.

Find by Ingredients is the endpoint which is used when a recipe is to be searched by ingredients specified by the user. Although a similar result can be achieved by using the complex recipe search endpoint, the number of API calls this takes is lesser than the latter. This API Endpoint is used only when a user does not want her default recipe but wants to cook with specific ingredients.

Find by Cuisine is another endpoint and is used when the user is wishing to try out a new cuisine. Like the “find by ingredients” endpoint, results for this can be achieved by using the complex recipe search endpoint but is chosen for reducing the number of API calls (and consequently, time).

Random Recipe Search is used when the user doesn’t have any specific cuisine or ingredients in mind but uses random phrases to describe her dish. The random recipe endpoint takes

- **Nutritional Information:** FOODIE uses the recipe search endpoint provided by Spoonacular for getting a recipe’s nutritional information asked for by the user. This endpoint takes the ingredients of the recipe as input and provides the caloric breakdown for the dish.

FOODIE also makes use of “Visualize Product Nutrition” if a user wants to look up the nutritional information of a food product. This endpoint takes a product UPC (Universal Product Code) and returns its nutritional information. Internally, FOODIE makes use of the FoodEssentials API to map a product to its UPC (described in further detail in Subsection 5.3.2).

- **Trivia:** Spoonacular provides the trivia endpoint for delivering random food facts to the user. When the user asks for information that FOODIE cannot an-

Table 5.1: Query filters for the complex recipe search endpoint

Query Filters	Description
Cuisine	The cuisine(s) of the recipes. E.g. Indian, Mexican and Chinese
Diet	Recipes that must be filtered by a diet type such as vegetarian and pescaterian
Ingredient inclusion	Two parameters for included ingredients and excluded ingredients. Included ingredients could be items that a user wants to eat and excluded ingredients could be those that are unavailable in the fridge.
Intolerance	Allergies and food items that cause allergy can be provided in this filter
Controlled nutritional amounts	Minimum and maximum amounts for nutritional content such as calcium, folic acid, and zinc for users that have stricter control over their diets
Query	A generic list of keywords that returns a fuzzy search of the database
Type	The type of meal - e.g. breakfast, lunch and dinner

Table 5.2: List of Recipe Search Endpoints in Spoonacular

HTTP Method	Description	Endpoint
GET	Complex Recipe Search	https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/searchComplex
GET	Find by Ingredients	https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/findByIngredients
GET	Find by Cuisine	https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/findByCuisine
GET	Random Recipe Search	https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/random

swer, a response from the Trivia endpoint is directed to the user. This endpoint does not take any parameters and just returns a fact whenever it is accessed.

Recipe Recommendations with CAPRecipes

Eventually, in our implementation, we would like to add the option to choose between Spoonacular and CAPRecipes [24], for recipe recommendations. After the input is parsed by FOODIE, relevant requests are sent to CAPRecipes instead of Spoonacular. CAPRecipes is a recipe recommender system that takes into account personal context such as health-related information (e.g., age, weight, height, diseases, and allergies), faith or belief restrictions, cuisine style, browsing history, likes on social media, and recipe ratings to make personalized recipe recommendations. The kitchen context features ingredients available in the user’s refrigerator and cupboards (i.e., through digital information from receipts provided by grocery stores) including their quantities and expiry dates. Given this context (personal and kitchen), CAPRecipes makes recommendations using collaborative filtering [43]. For example, if CAPRecipes knows that a certain product in the kitchen, say eggs, is expiring soon, it will recommend making an appropriate dish using eggs before suggesting other dishes, thereby reducing the user’s food wastage and optimizing the use of groceries.

5.3.2 FoodEssentials

The FoodEssentials API primarily provides allergen information for grocery products. It also provides the capability of searching product information by looking up the Product UPC or the Product name. FOODIE uses FoodEssentials for looking up allergen information for a food product or to find out the UPC of a food product. The endpoint “searchprods” takes the product name as input and provides a list of UPCs that match the product name being searched for.

5.3.3 Recalls

Another data source that FOODIE connects to is the Recalls and Safety Alerts database provided by the Government of Canada. Typically, recalls are issued on products that have become unsafe to consume or use. The Recalls and Safety Alerts API issues these alerts for consumer products, vehicles, food and health products of which we access the alerts for food products since that is FOODIE’s area of concern. This API allows

developers to access that information in JSON format.

5.4 Watson Assistant

Section 2.4.1 gave a brief introduction of Watson Assistant and its services. In this chapter, we discuss how Watson Assistant is integrated with FOODIE. Watson's conversational services are hosted on IBM's Cloud platform which also powers Watson's other cloud offerings. Watson's services enable developers to plug in Watson's cognitive capabilities for their own applications. Watson offers a Graphical User Interface (GUI) for developing the conversational skeleton via workspaces. The design of our conversation has been described in detail in Chapter 4.

Watson Assistant's services can be accessed by applications via their Application Programming Interface.²

Versioning: All API requests to Watson Assistant take a version parameter, which is specified as a date (`version=YYYY-MM-DD`). Every time a backward-incompatible, new version of Watson Assistant is released, this date is updated.

Message endpoint: The message endpoint is the primary endpoint for all communication between Watson Assistant and the application code, which in our case is FOODIENLU and FOODIECORE respectively. Requests to the endpoint are in the form of a String (these are the sentences that the user speaks). These requests are then interpreted by the natural language understanding component of Watson Assistant. Every time a user interacts with FOODIE, a request is sent to the message endpoint to interpret it. Relevant information about the sentence such as the intents recognized and the entities extracted are given as the response. The format of the response is in the JavaScript Object Notation (JSON) which is illustrated in Listing 5.1.

²<https://www.ibm.com/watson/developercloud/conversation/api/v1/curl.html?curl>

Table 5.3: Request parameters for the `message` endpoint

Request Parameter	Description
<code>workspace_id</code>	The conversation workspace that is currently in use, in our case, FOODIENLU's <code>workspace_id</code>
<code>version</code>	Specified as the date when a backwards-incompatible version is released.
body	The details about the user's input sentence that is to be parsed. The data type of this field is <code>MessageRequest</code> .
<code>nodes_visited_details</code>	Additional diagnostic information (default: <code>false</code>)

Table 5.4: Response details for the `message` endpoint

Response	Description
intents	The various intents recognized by Watson Conversation, in order of their confidence. In most cases, if defined well, there is usually only one intent that is recognized for a user input.
entities	The keywords extracted from the users input. These keywords are predefined in our conversation workspace. Apart from the exact match for the keyword, Watson Conversation matches its synonyms as the entity.
output	The actual response that is to be given by FOODIE is mentioned in the output parameter. This response is defined in our conversational workspace, FOODIENLU
context	The context variables set during the conversation

5.5 Context Management

Context plays a key role in conversation management. Context is the mechanism to store information between our Watson Assistant workspace (FOODIENLU) and the application that processes the user input (FOODIECORE). Messages sent between FOODIECORE and FOODIENLU involve a contextual component (represented as a JSON). As discussed in Section 2.1.2 there are three kinds of context are used by FOODIE, namely, Physical, Linguistic and Personal Context. Personal and Physical context are accessed by FOODIECORE, whereas, linguistic context is set and updated with in FOODIECORE and FOODIENLU. This is represented in Figure 5.3 which shows how the context is sent back-and-forth between our workspace in Watson Assistant and the application.

5.5.1 Maintaining state

Watson Assistant is a stateless service, meaning that contextual data is not remembered between requests (to the message endpoint of Watson Assistant). The appli-

Listing 5.1: A sample response for the message endpoint in Watson Assistant

```
1 {
2   "intents" : [ {
3     "intent" : "conversation\_start",
4     "confidence" : 0.9755029201507568
5   } ],
6   "entities" : [ ],
7   "input" : {
8     "text" : "Hi"
9   },
10  "output" : {
11    "generic" : [ {
12      "response_type" : "text",
13      "text" : "Hello there, what can I do for you?"
14    } ],
15    "text" : [ "Hello there, what can I do for you?" ],
16    "nodes_visited" : [ "greeting" ],
17    "log_messages" : [ ]
18  },
19  "context" : {
20    "conversation_id" : "a96ec62f-773c-4e84-8be9-f9dbca9f8
21      3d0",
22    "system" : {
23      "dialog_stack" : [ {
24        "dialog_node" : "root"
25      } ],
26      "dialog_turn_counter" : 1,
27      "dialog_request_counter" : 1,
28      "_node_output_map" : {
29        "greeting" : {
30          "0" : [ 0, 0 ]
31        }
32      },
33      "branch_exited" : true,
34      "branch_exited_reason" : "completed"
35    }
36  }
```

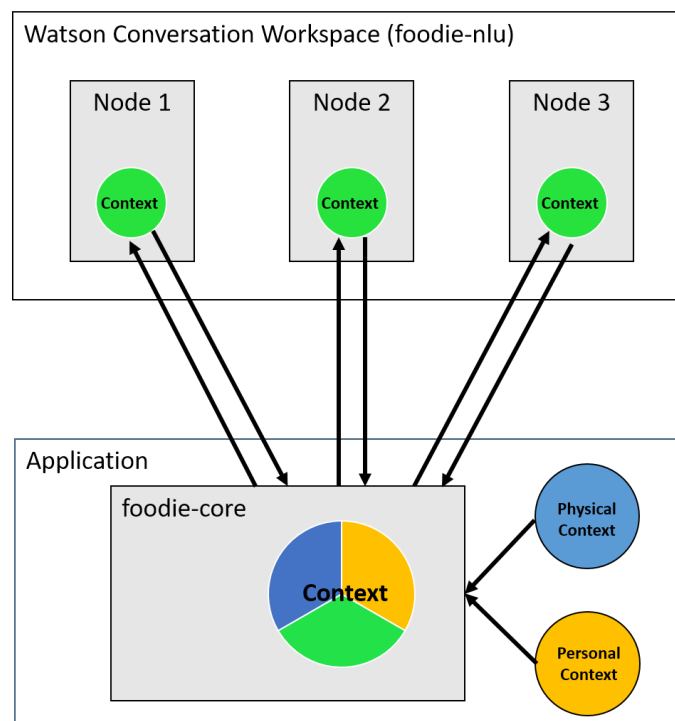


Figure 5.3: A simplified diagram showing flow of context between the application and the Watson Assistant workspace

cation (FOODIECORE) is responsible for maintaining state information during the conversation. This is maintained by passing the JSON object back and forth between FOODIENLU and FOODIECORE. Requests to the message endpoint include a response which includes the context object. This context object is sent back in the next request to FOODIENLU. The context received in a previous node is sent back as the input context for the next node to ensure that context is maintained from one turn to the next. It should be noted that the state information is not saved after the conversation has terminated. Section 7 presents some ideas for implementing a persistent store for context which would improve the application greatly.

5.6 Orchestration

One of the main components of FOODIECORE is its orchestrator (`foodie-orchestrator`) which is responsible for receiving requests from users as input, redirecting the request to appropriate back-end databases and formulating the response for the user.

Algorithm 1 Foodie’s orchestrator

```

1: procedure FOODIE-ORCHESTRATOR(input_sentence  $s$ , watson_workspace  $w$ )
2:   Sanitize input
3:   while  $s \neq \text{"bye"}$  do                                     ▷ While the conversation is active
4:      $message\_response \leftarrow message(w, s)$ 
5:     if message_response contains key “context” then
6:        $context \leftarrow message\_response.get(context)$ 
7:       if message_response contains key “request” then
8:          $request \leftarrow context.get(request)$ 
9:         Match request with existing request types
10:        Send back-end requests based on the request type
11:         $output \leftarrow output + \text{pre-defined response from watson workspace}$ 
12:   return  $output$                                              ▷ return-output

```

The specifics of `foodie-orchestrator` are captured in Algorithm 1. The orchestrator receives the input sentence and the watson workspace as input. After sanitizing the input and extracting the sentence, it is sent to the workspace which uses IBM Cloud services for natural language understanding. The message endpoint of Watson Assistant is used to gather important information about the sentence (intents, entities and context variables). After the intents, entities and the context variables have been extracted, the appropriate back-end database and its parameters are chosen. The

Listing 5.2: Response from the Watson Speech to Text Service

```

1 {
2   "results": [
3     {
4       "alternatives": [
5         {
6           "confidence": 0.8691191673278809,
7           "transcript": "Hi I would like to cook something
8             "
9         }
10      ],
11      "final": true
12    }
13  ],
14  "result_index": 0

```

response from the database is appended to the pre-defined response from the current node in the workspace.

5.7 FOODIE Voice User Interface (VUI)

Input to Foodie is via text as well as voice. There are additional requirements to set up voice in our application. Voice interaction is enabled by using the Speech-to-text and Text-to-Speech services offered by the IBM Cloud. In FOODIE, speech is first converted to text, which is then sent as an input to `foodie-orchestrator`. The response from FOODIE-ORCHESTRATOR is received in textual form, which is then converted using Text-to-Speech.

IBM Speech to Text

The IBM Speech to Text is a service that converts human voice to the written word.³ Requests sent to this service return a metadata object as a JSON response, which includes the transcription as well as confidence scores, start/end times and alternate hypotheses.

IBM Text to Speech Text to speech is a service that processes text and natural language to generate synthesized audio output complete with appropriate cadence

³<https://console.bluemix.net/catalog/services/speech-to-text>

Listing 5.3: Response from the Watson Text to Speech Service

```
1 curl -X POST -u {username}:{password} \  
2 --header "Content-Type: application/json" \  
3 --header "Accept: audio/wav" \  
4 --data "{\"text\":\"hello world\"}" \  
5 --output hello_world.wav \  
6 "https://stream.watsonplatform.net/text-to-speech/api/v1/  
  synthesize"
```

and intonation.⁴ The method “/v1/synthesize” (POST) is used to synthesize English input into audio.

5.8 Summary

This chapter described the implementation details regarding FOODIE. While Chapter 4 focused on the conversational part, this chapter focused on the entire implementation. We logically divided FOODIE into FOODIECORE and FOODIENLU and described these components in detail. We described the requirements that are to be met by our application, FOODIE. We also outlined the different publicly available databases that we use for recipe recommendation, finding allergens and searching for recalled items. We defined the contextual variables that are used and how context is maintained in FOODIE. We concluded the chapter with the implementation details of FOODIE’s Voice User Interface.

⁴<https://console.bluemix.net/catalog/services/text-to-speech>

Chapter 6

Validation and Discussion

6.1 Validation Methodology

This chapter describes the validation of the research work done for this thesis. Our research questions as posited in Section 1.2 are summarized below. For each of these research questions, we identify a set of requirements that we validate in this chapter.

- **Platforms that provide AI-as-a-service**

- C1.1** Identify significant conversational frameworks

- C1.2** Identify key attributes for describing features of conversational frameworks

- C1.3** Describe each of the conversational frameworks in terms of the key attributes.

- **Challenges in building a conversational agent**

- C2** Define a design methodology for building conversational agents

- C3** Incrementally improve the design methodology by building FOODIE: a conversational agent for the smart kitchen.

- **Personalization**

- C4.1** Identify the method of representation for contextual variables

- C4.2** Identify contextual variables for FOODIE

Table 6.1: Goal-Requirement mapping

Goal	Description	Requirement
G1	Manage users' dietary preferences	Conversational Goals
G4	Suggest a recipe based on the users' dietary preferences	Personalization
G8	Manage responses to unknown requests	Voice User Interface

We follow a three-pronged validation approach for the research questions of this thesis. For our first research question, Chapter 3 describes in detail, four conversational frameworks namely, IBM Watson Conversation, Google Dialogflow, Microsoft Bot Framework and Rasa NLU. These were chosen from a large pool of services that provide AI-as-a-service since they fairly represent the types of conversational AI services in this domain. For our second and third research questions, our design methodology and the context representation is evaluated by building FOODIE and is the content of this chapter. FOODIE's requirements are described in detail in Section 5.1. In the following sections we summarize the requirements of FOODIE that are to be satisfied and then conduct their validation which in turn, serve as a user manual for FOODIE. From the seven goals listed in Table 4.1 we pick three interesting cases for our validation such that they provide coverage of the requirements of FOODIE. Table 6.1 shows each requirement to be validated and is mapped to one goal. The following sections describe the validation of our requirements of conversational goals, voice user interface, and personalization.

6.2 Conversational Goals

Conversational goals are those high-level objectives that conversational agents aim to accomplish. FOODIE's conversational goals are described in Section 4.2. Note: The validation of this requirement is tied closely with the requirement related to personalization. Personalization is used in this context, however, specific details regarding personalization and context-awareness of FOODIE are described in Section 6.3.

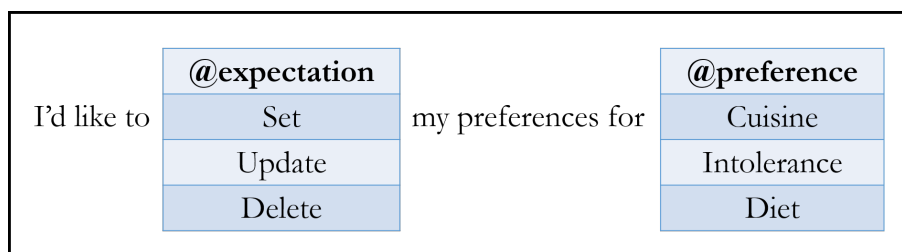


Figure 6.1: Selection of dietary preferences

6.2.1 Scenario: Managing users' dietary preferences

In this scenario, we describe how the user can go about managing her dietary preferences. As an evaluation to our design methodology (cf. Section 4.1), we describe the design methodology step-by-step for this scenario:

Identify Objectives. The high-level requirement of this scenario is for the user to be able to set, update or delete her dietary preferences. The user must be able to manage preferences regarding cuisine, diet, and intolerance.

Identify representative statements. The following are some examples of statements that the user may use in a conversation regarding dietary preferences:

- *Foodie, I'd like to set my preferences for cuisine*
- *Set my preference to vegetarian.*
- *I usually like to have Indian food.*
- *Add peanut as my intolerance.*

Identify Intents. Based on the utterances defined above, FOODIE must be able to recognize that there is a dietary preference to be managed. We identify similar statements and group them together under `#dietary_preference`. Listing 6.1 shows possible utterances for the dietary preferences intent. Some of the examples include the entities associated with this intent which will be further described in the following step. Examples of the intent are represented using a JSON script. Each key-value pair contains the sentence itself, and the “mentions” of entities that might occur in the sentence.

Listing 6.1: The intent description for the possible ways of indicating dietary preferences

```

1 {
2   {
3     "intent": "dietary_preference",
4     "examples": [{
5       "text": "I'd like to update my dietary preferences
6         "
7     }, {
8       "text": "I like to have cuisine",
9       "mentions": [{
10        "entity": "cuisine",
11        "location": [15, 22]
12      }]
13    }, {
14      "text": "I'm allergic to allergy",
15      "mentions": [{
16        "entity": "intolerances",
17        "location": [4, 12], [15, 22]
18      }]
19    }, {
20      "text": "Set my dietary preferences",
21      "mentions": [{
22        "entity": "goal",
23        "location": [0, 3],
24      }]
25    }, {
26      "text": "Open preferences"
27    }],
28   "description": "Manage dietary preferences of cuisine,
29     intolerance, and diet."

```

Identify Entities. The dietary preferences required to be managed are cuisine, intolerance, and diet. Therefore, these three parameters are identified as the entities that need to be recognized in utterances related the intent `#dietary_preference`. The key “location” (in Listing 6.1) specifies where in the sentence an entity might be mentioned.

Dialog Design. The next step in the design methodology is to design the dialog skeleton for the conversation. This involves identifying the triggers and paths to take in the conversation. The initial trigger is the intent itself. We include the parameters diet, intolerance and cuisine as the entities responsible to recognize dietary preferences. These are recognizable as entities in a sentence from the user. An example of the conversation between the user and FOODIE is shown below:

User: I’d like to manage my dietary preferences.

Foodie: Sure. You can set your preferred cuisine, your intolerances or your dietary restrictions.

User: Set my cuisine to Indian. “cuisine” = “indian”

Foodie: Done. Your preferred cuisine is set to Indian. Anything else?

User: Update my allergies to include peanuts. “intolerance” = “milk, peanuts”

Foodie: Done. Your intolerances include milk, peanuts. Anything else?

User: Delete my dietary restrictions “diet” = “vegetarian”

Foodie: Done. Your dietary restrictions vegetarian has been deleted. “diet” = “”

In the cases where the user specifies the entities in the first sentence (along with the trigger `#dietary_preferences`) itself, FOODIE must be able to pick up these entities and its expectation (whether the dietary preference is to be set, updated or deleted). Figure 6.2 shows how the relevant entities are extracted and parsed. **Identify Context Variables.** In this scenario, the following personal, linguistic and physical variables are recognized.

- Personal Context Variables: Listing 6.2 shows the personal context variables that are set by the end of the conversation on dietary preferences.

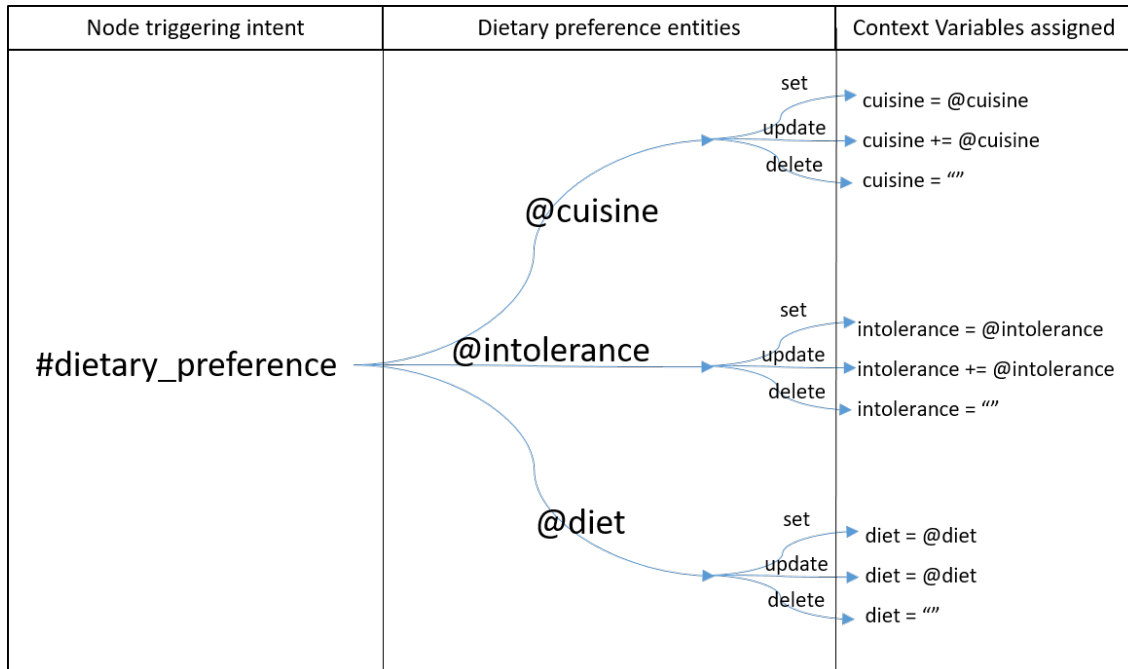


Figure 6.2: Identifying and setting entities for dietary preferences

Listing 6.2: Personal Context Variables

```

1 {
2   "context": {
3     "cuisine": "@cuisine"
4     "intolerance": "@intolerance"
5     "diet": "@diet"
6   }
7 }

```

Listing 6.3: Linguistic Context Variables

```

1 {
2   "context": {
3     "request": "dietary\_preferences"
4   }
5 }

```

- Linguistic Context Variables: Listing 6.3 identifies the linguistic context variable that is set during the conversation on dietary preferences.
- Physical Context Variables: No physical context variable is set while managing dietary preferences.

Although only a logical distinction is made between these context variables, they serve different roles in our application core. This is outlined further in the next section on Personalization.

6.3 Requirement 2: Personalization

We make use of personal context variables to tailor personalized experiences to the user during their interaction with FOODIE. The personal context variables that were set in the previous section are used here to show how they are used in our application code, FOODIECORE.

6.3.1 Recipe Suggestions

We pick the goal of suggesting recipes to show how personalization affects the results of a users query. Personalization is directly related to the personal context variables. The entities “@cuisine”, “@intolerance” and “@diet” that are set while updating dietary preferences serve as the parameters for personalization for requests to back-end databases.

The conversation related to the suggestion of recipes is as follows:

User: Hi Foodie. `#conversation_start`

Foodie: Hi there! How can I help you today?

User: I’m hungry. `#recipe_recommendation` “request”：“recipe_recommendation”

Foodie: Would you like to make **Butternut Squash & Chickpea Coconut Curry**?

User: No.

Foodie: What cuisine would you prefer?

User: Mexican

Foodie: Great! You have chosen **mexican** as the cuisine. Would you like to make **Taco Night with Homemade Corn Tortillas**?

User: Yes.

Foodie: Here are the steps to prepare **Taco Night with Homemade Corn Tortillas**:

6.3.2 Discussion: Initialization of Context Variables

Personal context variables are set for the first time during initialization. We use the physical context variable “location” to determine the cuisine that would most likely be preferred by the person. By default, this is set to “Canada”. The current system time is used to determine the meal that the user wants to have. For example, if a user invokes FOODIE at 4 PM in the afternoon, and asks for a recipe suggestion, the recommended meal will be a Canadian dinner item. The conversation related to the suggestion of a recipe in contrast to the conversation described above is as follows:

User: Hi Foodie.

Foodie: Hi there! How can I help you today?

User: I’m hungry. “request”:“recipe_recommendation”

Foodie: Would you like to make **Quinoa Chili with Slow Cooker Option**?

User: Yes, thanks!

Foodie: Here are the steps to prepare **Quinoa Chili with Slow Cooker Option**
... ..

The user can decline the suggestion, which prompts FOODIE to ask for other preferences such as cuisine, which would then override the default value set for the cuisine.

6.4 Requirement 3: Voice User Interface

The third important requirement is that of the Voice User Interface so that users can interact with FOODIE via voice. We use Watson’s Speech-to-Text and Text-to-Speech for enabling the Voice feature of FOODIE. In its current implementation, FOODIE can be run as a Java Archive (JAR) using the command line. Through the implementation and testing of Voice in FOODIE, it was realized that there were additional requirements that were required to make the interaction more natural.

6.4.1 Unknown Requests

We choose the management of responses to unknown requests as the goal which is uncomplicated and does not have too many connections so that the focus is on Voice. The intent “#anything_else” is a pre-defined intent in Watson Assistant and is typically the last node in a level of nodes in the conversation. This node is triggered when none of the nodes above it can recognize the intent of the user’s input. FOODIE responds with a short trivia about Food and reminds the user that the conversational agent can only understand sentences that are related to diet.

A typical conversation is as follows:

User: Hi Foodie. `#conversation_start`

Foodie: Hi there! How can I help you today?

User: How is the weather today?. `#anything_else`

Foodie: Sorry, food related questions only! Did you know that **Square watermelons sell for about \$85?**

`last_output = output`

User: Could you repeat that? `#repeat`

Foodie: Sorry, food related questions only! Did you know that **Square watermelons sell for about \$85?**

`last_output = output`

...

...

The linguistic context variable `last_output` saves the results of the current output each time so that if a user asks to repeat the sentence, FOODIE can give the same response. In a case when the last output is not saved, the pre-defined part of the response remains the same, but the part of the sentence which is a response from the back-end database (in this instance, the food trivia which is gathered from the Spoonacular database), will change when the node is revisited. It is also an issue if there are multiple pre-defined responses. For example, FOODIE's responses could be "Sorry, food-related questions only!" or "Sorry, I did not understand that!". This scenario brings to light the fact that voice interaction presents additional challenges which need to be looked at in conversational frameworks, which would contribute to suggestions that make the development of conversational agents that depend on voice much better. The following subsection describes some of these additional requirements that we found while implementing voice into FOODIE. Although these features can be built into the conversational skeleton, it is cumbersome to add these nuances at every step of the conversation. Therefore, these requirements are phrased as suggestions for conversational frameworks, in this case, for Watson Assistant.

6.4.2 Discussion: Building rich conversations

Global Nodes

We define global nodes to be the nodes that get repeated often in a conversation. In the case of FOODIE, the nodes to `repeat` and `reset` are repeated at every level of the conversation, at every branch. These nodes are triggered by the utility intents `#repeat` and `#reset`. At any point in the conversation, a user may say "What did you say?" or "Let's start over," especially with the input/output is in voice. The number of these nodes increases exponentially as the conversation branches out and becomes complicated. Having global nodes by default would be nice to have on platforms like Watson Assistant.

Conditional Jumps

Jumps are a feature in Watson Assistant that allows going from a node in one branch to a node in another branch at any level. When a jump is configured at the end of a node, the conversation transitions to another node regardless (i.e., jumps are not yet conditional). At many points during the design phase of our conversation, we

found that conditional jumps would have made the conversation easier to structure. For example, in the middle of the recipe recommendation, a user may ask for allergy information for one of the ingredients. This is much more likely in the case of voice because the user’s train of thought can switch to other tasks in the middle of a conversation, as opposed to conversations happening textually which augment the user with the part of the conversation that has already occurred.

Switching Context

The notion of conditional jumps is closely related to switching context. As discussed above, users may want to change something in the input or switch topics in the middle of a conversation. While it is possible to do this using jumps (better, conditional jumps), adding support to context switching at every node is cumbersome.

Entity Negation

In FOODIE’s conversation workspace, we describe many entities like goals, allergens, meal types and cuisine. When a user says “I’d like to prepare a snack,” the meal type “snack” is recognized. However, when a user says “I’m hungry, can you suggest something that is not a snack?”, the meal type is still recognized as “snack”. The problem is similar to that of context switching or global nodes; it is something that gets repeated often and even though setting it up is possible, it is cumbersome to do so.

6.5 Summary

This chapter described the validation of the requirements of FOODIE. We first picked a few goals which together would maximally characterize FOODIE. We then picked one goal/scenario per requirement and described the nature of output from FOODIE. The first requirement was that of the design of conversations which was demonstrated by setting and updating dietary preferences through the design methodology described in Chapter 4. The second requirement of personalization was described with the help of the recipe suggestion goal. The third requirement for the voice user interface was demonstrated by managing the response to unknown requests.

Chapter 7

Conclusions

This chapter summarizes this thesis, the addressed challenges and the contributions. We conclude with a discussion on the future work in the area of conversational agents.

7.1 Thesis Summary

IBM posits that we are in the era of cognitive systems— these are systems that learn at scale, reason with purpose, and interact with humans naturally. This makes the interaction between human and machine more co-operative [28]. Although the concept of computers interacting with humans has been around for a long time [48], it has only very recently become popular in terms of ease of use and ease of development [34].

In this thesis, we discuss conversational agents and how they are changing the landscape of user engagement in this era of cognition. This thesis aims to demonstrate the significance of conversational agents through the investigation of conversational technologies and through the design and development of our conversational agent, FOODIE. FOODIE is a text and voice-enabled smart conversational agent that assists users in the kitchen. FOODIE is augmented with personal context which manages user preferences, which leads to better engagement. FOODIE serves as a use case for modern conversational agents.

Advances in artificial intelligence, powerful devices, and better connectivity are the contributing factors to this. There are various conversational frameworks such as IBM Watson, Rasa NLU and Google Dialogflow that provide AI-as-a-service in the

areas of intent and entity recognition, and dialog building. This has made the process of getting a conversational agent up and running straightforward. However, to get a conversational agent from this initial proof of concept to a stage of sophistication is challenging and is the area of research for this thesis.

We describe how conversations between FOODIE and the user are orchestrated using Watson Assistant, the importance of having richer conversations and how that would make conversational agents a truly useful tool to augment human cognition. We also look at the differences in conversational structure between text and voice and stress on how conversational platforms have to be improved to support the latter.

7.2 Contributions

This section summarizes the research questions identified and the contributions that answer these research questions.

AI-as-a-service platforms

RQ1 What frameworks that provide AI-as-a-Service are useful in building conversational agents?

C1 We conducted a systematic review of some of the popular frameworks available for building conversational agents apart from Watson Assistant such as Dialogflow, Rasa NLU, and Microsoft Bot framework. We evaluated them on the basis of their ease of use, context integration, distinguishing features and voice integration. This is described in Chapter 3.

FOODIE: A conversational agent for the smart kitchen

RQ2 Given the frameworks to build conversational agents, what are the challenges in designing a sophisticated conversational agent?

C2 To address the challenges of the design of a conversational agent, we developed an iterative design methodology to build various kinds of conversational agents. This is described in Chapter 4.

- C3** One of the main contributions to this thesis is our conversational agent FOODIE as a proof-of-concept using the steps outlined in our design methodology (Contribution 2). FOODIE makes use of the conversational framework, Watson Assistant for natural language understanding which is deployed on the IBM Cloud platform.

Personalization

- RQ3** How can we leverage contextual information for the design and development of our conversational agent, FOODIE? How best can we represent context for our conversational agent, FOODIE?
- C4** Context plays an important role in the personalization of conversational agents. We use the JavaScript Object Notation (JSON) for representing context variables. We logically classify our context into three types: personal, physical and linguistic context. These are described in Chapters 4 and 5.

7.3 Future Work

This section concludes this thesis by discussing some directions for future work for this research.

SmartDialog: A module for enriching Voice-based interactions

There has been an increase in the use of the voice-based method of interaction with mobile applications due to the better accuracy of speech recognition systems. We identified the challenges in building a voice-based conversational agent in Section 6.4.2. We propose an external plug-in that would add functionality to existing workspaces, that enhances the conversational capabilities of conversational agents that interact with voice. For example, one of the features of SmartDialog would be creating reset nodes at every level in the conversation, creating **repeat** nodes at every level of the conversation. Since the workspace can entirely be represented in the JSON format with a logical structure to define the position of nodes in the dialog, it would be possible to insert the **reset** and **repeat** nodes. Since these utility nodes/intents

are required at every point in the conversation, it is worth investigating this area to make the process of making the dialog skeleton easier.

Integration with other digital assistants

“The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.” -Mark Weiser

There has been an increase in the usage of personal assistants, especially the Google Assistant ¹ in phones that use the Android operating system and Siri ² which is the personal assistant in Apple phones. Both ecosystems provide the ability for developers to integrate custom functionality into their assistants. In the future, we would like to integrate FOODIE with such personal assistants rather than it being a separate stand-alone application.

Persistent storage for personal context

In our current implementation, FOODIE stores conversational information only for the current conversation. Previous conversations and contextual information of users is not stored in a persistent storage. Addition of a database that stores context and users would improve the functionality of FOODIE greatly in the following ways:

- Addition of different profiles, for example, usage of FOODIE by family members.
- Storage of previous conversations which would document a user’s dietary habits over time.

Usage of sophisticated contextual frameworks

In this thesis, we made use of the JavaScript Object Notation for describing context. JSON provides many advantages in terms of its ease of use and understandability across platforms. However, JSON models offer limited capability in terms of expression. Our chosen application domain, the kitchen has complex contextual data. It would be useful to study whether the usage of more expressive contextual frameworks in this domain would lead to the better management of context. We would like to

¹<https://assistant.google.com/>

²<https://www.apple.com/ca/ios/siri/>

augment FOODIE with contextual models such as the Resource Description Framework (RDF) or ontological languages which are able to handle more complexity in context.

MealBuddy: A Cognitive IoT Meal Maven for Minimizing Food Waste at Home

We propose FOODIE to be a part of a larger ecosystem of software that caters to a smart kitchen. We term this to be “MealBuddy: A cognitive IoT Meal Maven for the Smart Kitchen”. MEALBUDDY collects information about food available in homes and grocery stores, understands users’ dynamic personal context, and provides personalized meal-planning services. Our motivation for this research is two-fold. First, there are various sources of personal information that can be exploited to address food-related concerns. Second, there is a need for systems that are capable of aiding the decision making the process for the achievement of personalized goals.

In its initial stage, we propose three components:

- CAPRECIPES [25] exploits personal preferences, health goals, and users’ personal IoT to provide personalized recipes with users’ available food;
- SMARTGROCER [26] exploits the grocery stores’ IoT data (i.e., prices and expiry date) to improve users budget and provide personalized coupons and discounts; and
- FOODIE augments the cognitive capabilities of users by providing a voice and text-enabled conversational interface tailored to understand food-related topics;

At this point in time, CAPRecipes, SmartGrocer, and FOODIE are functional as stand-alone applications. As future work, we would like to integrate these systems to form an ecosystem of software components that cater to the dietary requirements of an individual and their families.

Bibliography

- [1] IEEE authoritative dictionary of IEEE standards terms, seventh edition. *IEEE Std 100-2000*, pages 574–575, 2000.
- [2] Elizabeth Aguirre, Anne L Roggeveen, Dhruv Grewal, and Martin Wetzels. The personalization-privacy paradox: implications for new media. *Journal of Consumer Marketing*, 33(2):98–110, 2016.
- [3] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in English and Mandarin. *CoRR*, abs/1512.02595, 2015.
- [4] Ricardo Baeza-Yates. Data and algorithmic bias in the web. In *Proceedings 8th ACM Conference on Web Science (WebSci 2017)*, pages 1–1. ACM, 2016.
- [5] Emanuele Bastianelli, Daniele Nardi, Luigia Carlucci Aiello, Fabrizio Giacomelli, and Nicolamaria Manes. Speaky for robots: The development of vocal interfaces for robotic applications. *Applied Intelligence*, 44(1):43, 2016.
- [6] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (IaaS). *International Journal of Engineering and Information Technology*, 2(1):60–63, 2010.
- [7] Rubén Blasco, Álvaro Marco, Roberto Casas, Diego Cirujano, and Richard Picking. A smart kitchen for ambient assisted living. *Sensors*, 14(1):1629–1653, 2014.

- [8] Alexander Bleier, Arne De Keyser, and Katrien Verleye. *Customer Engagement Through Personalization and Customization*, pages 75–94. Springer International Publishing, 2018.
- [9] Jan A. Botha, Emily Pitler, Ji Ma, Anton Bakalov, Alex Salcianu, David Weiss, Ryan T. McDonald, and Slav Petrov. Natural language processing with small feed-forward networks. *CoRR*, abs/1708.00214, 2017.
- [10] Daniel Braun, Adrian Hernandez-Mendez, Florian Matthes, and Manfred Langen. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185, 2017.
- [11] Julio Cabrera, María Mena, Ana Parra, and Eduardo Pinos. Intelligent assistant to control home power network. In *IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC 2016)*, pages 1–6. IEEE, 2016.
- [12] Justine Cassell, Tim Bickmore, Lee Campbell, Hannes Vilhjálmsson, and Hao Yan. Conversation as a system framework: Designing embodied conversational agents. *Embodied Conversational Agents*, pages 29–63, 2000.
- [13] Jen-Hao Chen, Peggy Pei-Yu Chi, Hao-Hua Chu, Cheryl Chia-Hui Chen, and Polly Huang. A smart kitchen for nutrition-aware cooking. *IEEE Pervasive Computing*, 9(4):58–65, 2010.
- [14] Jinho D Choi, Joel Tetreault, and Amanda Stent. It depends: Dependency parser comparison using a web-based evaluation tool. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 387–396, 2015.
- [15] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167. ACM, 2008.
- [16] Douglas Crockford. The application/json media type for JavaScript Object Notation (JSON). Technical report, 2006.
- [17] M. R. Ebling. Can cognitive assistants disappear? *IEEE Pervasive Computing*, 15(3):4–6, July 2016.

- [18] Ethan Fast, Binbin Chen, Julia Mendelsohn, Jonathan Bassen, and Michael S. Bernstein. Iris: A conversational agent for complex tasks. *CoRR*, abs/1707.05015, 2017.
- [19] Maurizio Ficocelli and Goldie Nejat. The design of an interactive assistive kitchen system. *Assistive Technology*, 24(4):246–258, 2012.
- [20] Kelly Geyer, Kara Greenfield, Alyssa Mensch, and Olga Simek. Named entity recognition in 140 characters or less. In *# Microposts*, pages 78–79, 2016.
- [21] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
- [22] Karen Henriksen, Jadwiga Indulska, and Andry Rakotonirainy. Generating context management infrastructure from high-level context models. In *In 4th International Conference on Mobile Data Management (MDM)-Industrial Track*, 2003.
- [23] Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015.
- [24] Harshit Jain. CAPRecipes: Context-aware personalized recipes recommender. In *Poster: 26th IBM Center for Advanced Studies Conference: Annual International Conference on Computer Science and Software Engineering*, 2016.
- [25] Harshit Jain. CAPRECIPES: a context-aware personalized recipes recommender for healthy and smart living. Master’s thesis, Department of Computer Science, University of Victoria, Canada, June 2018.
- [26] Roshni Jain. SmartGrocer: a context-aware personalized grocery system. Master’s thesis, Department of Computer Science, University of Victoria, Canada, June 2018.
- [27] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, László Lukács, Marina Ganea, Peter Young, and Vivek Ramavajjala. Smart reply: Automated response suggestion for email. *CoRR*, abs/1606.04870, 2016.
- [28] John Kelly III and Steve Hamm. Computing, cognition and the future of knowing. Technical report, IBM Corp. White Paper, 2015.

- [29] Younsun Kim, Hyunggoy Oh, and Sungho Kang. Proof of concept of home IoT connected vehicles. *Sensors*, 17(6):1289, 2017.
- [30] Lorenz Cuno Klopfenstein, Saverio Delpriori, Silvia Malatini, and Alessandro Bogliolo. The rise of bots: A survey of conversational interfaces, patterns, and paradigms. In *Proceedings 2017 Conference on Designing Interactive Systems (DIS 2017)*, DIS 2017, pages 555–565, New York, NY, USA, 2017. ACM.
- [31] Will Knight. Conversational interfaces: Powerful speech technology from China’s leading internet company makes it much easier to use a smartphone. *MIT Technology Review*, 2016.
- [32] Ora Lassila and Ralph R Swick. Resource description framework (RDF) model and syntax specification. *World Wide Web Consortium Recommendation*, 1999.
- [33] Carlene R. Lebeuf. A taxonomy of software bots: towards a deeper understanding of software bot characteristics. Master’s thesis, Department of Computer Science, University of Victoria, Canada, August 2018.
- [34] Brian Manusama Magnus Revang, Van L. Baker. Market guide for conversational platforms. Technical report, Gartner, 2018.
- [35] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, volume 752, pages 41–48. Madison, WI, 1998.
- [36] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C Recommendation*, 10(10):2004, 2004.
- [37] Michael McTear, Zoraida Callejas, and David Griol. *The Conversational Interface: Talking to Smart Devices*. Springer, 2016.
- [38] Pierrick Milhorat, S Schlogl, Gérard Chollet, Jerome Boudy, Anna Esposito, and Gianni Pelosi. Building the next generation of personal digital assistants. In *1st IEEE International Conference on Advanced Technologies for Signal and Image Processing (ATSIP 2014)*, pages 458–463. IEEE, 2014.
- [39] Tom M Mitchell. Artificial neural networks. *Machine Learning*, 45:81–127, 1997.

- [40] Rosslin John Robles and Tai-hoon Kim. Context aware tools for smart home development. *International Journal of Smart Home*, 4(1), 2010.
- [41] Roland T Rust and Ming-Hui Huang. The service revolution and the transformation of marketing science. *Marketing Science*, 33(2):206–221, 2014.
- [42] Jaime Salvador, Zoila Ruiz, and Jose Garcia-Rodriguez. Big data infrastructure: A survey. In José Manuel Ferrández Vicente, José Ramón Álvarez-Sánchez, Félix de la Paz López, Javier Toledo Moreo, and Hojjat Adeli, editors, *Biomedical Applications Based on Natural and Artificial Computing*, pages 249–258. Springer, 2017.
- [43] JHJB Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. *The Adaptive Web*, pages 291–324, 2007.
- [44] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp*, volume 4, pages 34–41, 2004.
- [45] Lance Ulanoff. The life and times of eugene goostman, who passed the Turing test. Technical report, Mashable, 2014.
- [46] Norha M Villegas and Hausi A Müller. Managing dynamic context to optimize smart interactions and services. *The smart internet*, 6400:289–318, 2010.
- [47] Leo Wanner, Elisabeth André, Josep Blat, Stamatia Dasiopoulou, Mireia Farràs, Thiago Fraga, Eleni Kamateri, Florian Lingensfelder, Gerard Llorach, Oriol Martínez, et al. Kristina: A knowledge-based virtual conversation agent. In *International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2017)*, pages 284–295. Springer, 2017.
- [48] Joseph Weizenbaum. ELIZA - A computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.
- [49] Jennifer Zamora. Rise of the chatbots: Finding a place for artificial intelligence in India and US. In *Proceedings 22nd International Conference on Intelligent User Interfaces (IUI 2017)*, IUI 2017 Companion, pages 109–112, New York, NY, USA, 2017. ACM.