Utrecht University

Bachelor Thesis
Information Science

Specifying and Testing Conversational User Interfaces

Author:
Jasper Berends
J.Berends2@students.uu.nl
Student Number: 5516080

Thesis Supervisor:
Christof van Nimwegen
C.vanNimwegen@uu.nl

External Supervisor:
Joop Snijder
Joop.Snijder@infosupport.com

Thesis Supervisor 2:
Jan Martijn van der Werf
J.M.E.M.vanderWerf@uu.nl

July 7, 2017

Preface

I present to you the thesis 'Specifying and Testing Conversational User Interfaces' which proposes the Conversation Design Framework. The research is conducted for and on behalf of Info Support, a Dutch technology firm located in Veenendaal, the Netherlands. This thesis is written as a graduaction project for my Information Science Bachelor at Utrecht University. From April 2017 until July 2017, I have been an intern at Info Support in order to conduct the research and write this thesis.

The subject of this research came from Info Support, where an increasing amount of clients are asking the enterprise about conversational user interfaces. A created the specific research question together with my supervisor from Info Support, Joop Snijder. As this was a completely new field for me, the research I conducted was very difficult. No scientific research existed on the design of CUIs at all either, which made it even more complex. After gathering information from many nonscientific sources and scientific literature into related systems and components, I was able to create the Conversation Design Framework followed by presenting suggestions for test methods.

Throughout my research, Joop Snijder always offered very valuable feedback on my findings and offered support whenever necessary. I could always ask questions or pitch ideas to him and the that helped me with my research a lot. Therefore, I wish to thank him for all his guidance during my internship at Info Support.

I also want to thank Christof van Nimwegen for supervising my thesis research. Even though he warned me beforehand that he had a very tight schedule, he always took the time to hear me out about my progression and to guide me through the research process.

Another person that should be thanked is Willem Meints. He is an employee at Info Support with much experience in the fields of artificial intelligence and chatbots. He provided me with feedback on the CDF and gave me some good insights on his and Info Support's experience with chatbots and bot frameworks. Last but not least, I wish to thank the other graduate students as well as coworkers at Info Support that provided me with words of wisdom and with moral support throughout these past few months.

I hope you will enjoy reading this thesis.

Jasper Berends

Veenendaal, July 2017.

Table of Contents

Abstract

This thesis examines how to specify and test a Conversational User Interface (CUI) in an Agile environment. The research is conducted for and on behalf on Info Support, but its findings are generic and apply to all CUIs in general. CUIs are a new, trending technology with little to no research currently available and new frameworks are necessary for proper implementations of the system. During this research, the Conversation Design Framework (CDF) is created as the solution for specifying conversations of the CUI. The framework consists of seven concrete steps and aims to set a best practice design guideline for specifying an effective and efficient conversation. After elaborating the CDF, methods are proposed for testing the specifications that are created when using this model. A method of testing the personality is absent as only indirect testing suggestions are given, while many other testing can potentially be automated. Afterward, it is researched how well the well the Agile application of the CDF is and it is concluded that the framework can be implemented with high agility. The findings of this research provide a sound answer to the research question and are a good introduction to research into conversational user interfaces. Further research could be undertaken to create a model of testing chatbot personalities, validating the CDF by applying it to a real system through a case study and creating an evaluation method for bot frameworks.

Keywords: Conversational user interface, Conversation design framework, Conversation specification, Conversation testing, Agile

1 Introduction

Conversational user interfaces (CUIs) are trending, with many of companies working on their own version. Voice assistants are particularly popular in the consumer market: Apple has Siri[1], Google offers Google Assistant[2], Amazon has Alexa[3], and Samsung recently launched their version as well, Bixby[4]. Due to technology advances, AI bots continue to possess more human-like features[5]. This creates new opportunities for applications that simulate human communication, both verbal and nonverbal. Platforms such as Facebook Messenger and Slack opened their environments for chatbots, resulting in possible worldwide user reach for developers of those chatbots. According to Business Insider [6], there were already over 34,000 chatbots for Facebook Messenger in November 2016. They suggest that chatbots will become a mainstream for businesses as a means of communication and that it will potentially give those businesses more annual revenue than they will generate through platforms like the iOS app store or the Google Play Store. This means that the trend of CUIs is not merely a hype but that it can potentially add significant added value for firms.

The first predecessors of conversational user interfaces previously known as spoken dialogue systems, which "enable casual and naive users to interact with complex computer applications in a natural way using speech" (McTear, 2002). McTear described three main types of classifications for spoken dialogue systems:

1. finite state-based systems;
2. template-based systems; and
3. agent-based systems.

The three classifications are leveled from most basic to most advanced spoken dialogue system. A finite state-based system follows a sequential process where the user has no real input on how the conversation is built up. McTear describes the template-based system as a system where the user can "fill slots in a template," where the dialogue flow may change, depending on what information the user gives at what time. The latter of the three classifications is the most advanced: "Agent-based or AI systems are designed to permit complex communication between the system, the user, and the underlying application in order to solve some problem or task." These systems use dialogue models that take the context of the conversation into account, leading to a more dynamic conversation.

The latter already resembles a CUI very well, where a CUI is still a little more advanced: "Conversational interfaces enable people to interact with smart devices using spoken language in a natural way—just like engaging in a conversation with a person" (McTear, Callejas, & Griol,

[1]`https://apple.com/ios/siri/`, accessed May 2, 2017
[2]`https://assistant.google.com/`, accessed May 2, 2017
[3]`https://developer.amazon.com/alexa`, accessed May 2, 2017
[4]`http://samsung.com/global/galaxy/apps/bixby/`, accessed May 2, 2017
[5]`https://forbes.com/sites/danielnewman/2017/01/27/time-for-chatbots-to-get-smart/`, accessed May 2, 2017
[6]`http://www.businessinsider.com/facebook-opens-analytics-developer-tools-for-messenger-bots-2016-11`, accessed July 6th, 2017

2016). Whereas the agent-based spoken dialogue system already specifies complex communication, a CUI should be able to communicate with a user naturally. Natural Language Processing (NLP) is embedded in this, which means that a CUI needs to be able to understand naturally spoken language and not only predetermined dialogues (Chowdhury, 2003).

To process natural language, a CUI needs to be able to define its context at any point in time. This paper aims to provide a model for setting functional requirements specific to a CUI, such as the aforementioned context. Proper frameworks for designing conversations are necessary for this increasingly more popular branch, thus stressing the relevance of this paper. As software firms are increasingly implementing Agile development processes (Hamed & Abushama, 2013), the application using this methodology is included in the research. Thus accomplish this, the following research question is derived:

How can conversational user interfaces be specified and tested in an Agile environment?

This thesis presents the Conversation Design Framework, created as a best practice design guideline for specifying effective and efficient conversations. As little to no scientific research on CUIs exists, many insights are obtained from suggestions by technology giants (Google, Facebook and others) and published articles from current chatbot design professionals. Besides that, much insights are done from previous works on nonconversational voice-enabled interfaces, such as the spoken dialogue systems mentioned above, and small parts of the framework are backed by theory on conversation or software design where possible.

The research is executed for and on behalf of Info Support, a Dutch technology enterprise located in Veenendaal, the Netherlands[7]. Info Support is specialized in development, managing and hosting of custom software solutions and attempts to always be a leader in innovation. Info Support receives an increasing amount of requests regarding chatbots from its clients, which indicates that there is an increasing demand of chatbots by businesses. This research is, however, valuable for anyone who wishes to get involved with conversational user interfaces and chatbots. The research findings as well as the proposed framework make use of Info Support's insights and experiences in this field but apply to the chatbot environment as a whole.

This thesis begins by describing what exactly defines a conversational user interface, laying out what contemporary CUIs already exist and how it can add value to an enterprise in section 2. Section 3 presents the Conversation Design Framework which can be used for specifying CUIs, followed by guidelines on how to test specifications after having used this framework, as can be found in section 4. Section 5 will go deeper into how to apply the theories of section 3 and section 4 in an Agile environment. Afterward, section 6 will discuss the strengths and limitations of the research,

[7]https://www.infosupport.com/

2 Conversational User Interface

As mentioned in the introduction, a CUI should be able to understand and process natural language. The oldest related research to this was that of spoken dialogue systems, of which three types were categorized: finite state-based, template-based and agent-based systems (McTear, 2002). After that, the term Voice User Interface was given life: "Voice user interfaces (VUIs) use speech technology to provide users with access to information, allowing them to perform transactions, and support communications" (Lotterbach & Peissner, 2005). Lotterbach and Peissner set the boundary on VUIs in that they only cover a predefined set of speech acts and conversations that a system can handle, which will depend on the task domain in which the system is implemented.

A conversational interface is an advanced VUI where these boundaries may still be there, but not necessarily. For example, Siri, Apple's voice assistant that is integrated with the manufacturer's devices, can respond to inputs from many distinct domains and can even answer to user requests that are not within these domains (Bellegarda, 2014). Amazon Alexa is another example of a system that has an enormous amount of domains for which it can be used as companies can build their own *skills* for it, resulting in already more than 12,000 commands that the conversational interface accepts[8]. However, Alexa is also an example of how CUIs can be integrated in domain-specific contexts. Amazon provides an API for its bot[9] that allows developers to integrate the bot in their own applications for a specific domain.

2.1 Types of CUIs

As mentioned above, CUIs can be domain-specific but do not have to be. However, even in the example of adding a skill to Alexa, the developer that creates the skill operates in a domain-specific manner. This paper will describe several general development methods that apply to both, but domain-specific CUIs are kept as a focus in this research. Besides that, two CUI classifications are proposed in this research:

- Informational CUI; and
- goal-oriented CUI.

Informational interfaces are systems from which users can only retrieve information. They can ask the system for information, followed by a response based on a query search. The goal-oriented interface is one which is capable of performing tasks. An example of this is a flight booking system, where users can tell the interface to book a flight, after which this is automatically executed. The user will give his preferences and is asked for missing information by the system. Once a consensus has been reached on which flight to book, the user confirms that he has made his choice. Whereas the informational CUI would only provide information on the flights, the goal-oriented CUI actually books the flight as well, which means that the user does not have to perform any actions outside of the CUI. This paper limits itself to goal-oriented CUIs, but insights are likely applicable to informational CUIs as well.

[8] `https://developer.amazon.com/alexa-skills-kit`, accessed May 17, 2017

[9] `https://developer.amazon.com/public/solutions/alexa/alexa-voice-service/content/avs-api-overview`, accessed May 17, 2017

2.2 Current Systems

There are already systems that can be categorized as conversational user interfaces, of which a few are discussed in this section. Apple's Siri has been around the longest for smartphones, which was released in 2011[10]. As mentioned in section 1, Samsung recently introduced their own assistant, Bixby, and Google has Assistant. Even though the implementations with smartphones are different, their purpose is the same: enabling the user to talk to their phone to gather information or to perform tasks. Google Assistant is also usable through a smart speaker, Google Home. Amazon basically has the same implementation with Alexa, that can be used through their Echo speakers[11].

Other notable systems are Mycroft and Maluuba. Mycroft is an exceptional case as it is an open source voice assistant[12]. Being open source makes their system more customizable for experienced developers, and may also be favorable in privacy-sensitive cases as developers have the possibility to scan the source for unwanted code, e.g. code that shares usage data with the owners of the system. Maluuba is a company that developed a system that is not publicly available but is still mentioned as they do much research into the fields of deep learning and reinforcement learning[13], and they also published two large datasets for natural language understanding research.

2.3 Components of a VUI

The VUI components are the starting functional requirements of a CUI and should be ready before starting the process of the actual conversational interface. Only when this system is ready, developers can add a conversational interface. When a CUI lets its users speak to it vocally, one must first implement all the building blocks of a VUI. These are usually categorized as six different components (McTear, 2002; Salvador, de Oliveira Neto, & Kawamoto, 2008):

- **Speech Recognition**: the CUI will need to be able to listen to the user and transcribe the speech into text strings. This component is essential for enabling voice interactions. A substantial dictionary is required for recognizing all utterances of users. Even with domain-specific systems, the dictionary should contain not only domain jargon but also regular words to make processing the entire user request possible.
- **Natural Language Understanding (NLU)**: NLU takes the input from the speech recognition component or the chat input from the user, and analyzes what it means. Especially in CUIs, where systems have to be conversational, NLU is crucial as users should be able to speak to the system naturally.
- **Dialogue Management**: the dialog manager controls the flow of the conversation. It determines if a user has given enough information for a certain goal, it communicates with the external applications and it decides the response to be given to the user's request.
- **System Integration**: this component allows the dialog manager to connect to external databases of the system. By integrating the databases of an enterprise, the dialog manager

[10]`https://apple.com/pr/library/2011/10/04Apple-Launches-iPhone-4S-iOS-5-iCloud.html`, accessed May 23, 2017

[11]`https://amazon.com/Amazon-Echo-And-Alexa-Devices/b?ie=UTF8&node=9818047011`, accessed May 23, 2017

[12]`https://mycroft.ai/`, accessed May 23, 2017

[13]`http://maluuba.com/`, accessed May 23, 2017

can find the right data for the user's request and respond accordingly.

- **Response Generation**: this component constructs the form of the response. This is where the specifications are set for what information should be included in the reply, as well as how it is structured in terms of the syntactic structure and the choice of words.
- **Speech Output**: once the response has been generated, the speech output component will transform this into speech and play it back to the user. It has to analyze the output text and model a continuous speech effect along with it in order to make it sound natural.

The first and last item, speech recognition and speech output, do not apply when a CUI is purely text-based, whilst the other four are still relevant in that case. The goal of these components is to infer *intents* and *entities* of the user. McTear (2016, Chapter 6) describes intents as "an abstract representation of an action corresponding to an activity or service that can be initiated from other activities." This means that the intent is what the user wants the system to do and entities are the relevant elements that are important to execute the intent properly. For example, if the goal of the user is to order a flight ticket, the intent would be 'OrderTicket' with the entities 'Departure', 'Destination' and 'Date'.

2.3.1 Bot Frameworks

The six components of a VUI may require intensive programming when done right. Especially NLU will require complex algorithms and rigorous syntactic and semantic analysis(De Mori et al., 2008). Adding conversational elements to a VUI makes this process even more challenging, context and semantic meaning become much more important. For this reason, multiple tools can take the role of VUI upon themselves, allowing users to create chatbots through their frameworks. The three big ones are Google's Api.ai[14], Facebook's Wit.ai[15] and Microsoft's Bot Framework[16]. Broadly speaking, the three allow the bot developer to set intents, entities and the resulting actions of those intents and entities. These frameworks have all six VUI components, from speech input to speech output. Developers can integrate the frameworks into their applications and model conversations on the platform of the chosen framework. For example, for modeling a goal on Api.ai, one can give an example what a person says (the intent), the entities associated with these intents, and what action should be taken following the user's input. For entrepreneurial applications, this action will involve connecting to external databases of the enterprise systems in order to give back the appropriate response.

A developer can give as many example inputs as he desires but is not required to map out every possible user input, which would also not be possible when natural language is involved, due to *active learning*. With active learning, the system will adapt input specifications based on actual user inputs, which means that example inputs do not need to precisely match what the inputs of the user. As a result of active learning, the framework will be able to link more and more inputs to specific intents over time. Active learning is an important factor of a CUI because it will make the bot a lot more conversational as users can use natural language to communicate with the system instead of merely using predefined commands.

[14]`https://api.ai/`, accessed May 27, 2017
[15]`https://wit.ai/`, accessed May 27, 2017
[16]`https://dev.botframework.com/`, accessed May 27, 2017

2.4 Applications for Enterprises

Conversational user interfaces offer many possibilities to businesses. First of all, if the CUI is designed well, using it can be much easier than using software with a graphical user interface (GUI). For example, a user that owns a Google Home smart speaker may simply say "Hey Google, is it going to rain on Thursday morning in Amsterdam?" The smart speaker should then immediately give the answer to the user's request. With a GUI, e.g. the user's smartphone, he will need to open a web browser, find weather forecasts of Amsterdam, look up the forecast of Thursday morning, and determine himself whether or not it is going to rain. As one can conclude, this process takes much more time and effort, making a CUI much easier to use.

A CUI can manifest itself in many different forms. First of all, as mentioned earlier in this section, developers can add skills to general systems such as Google Home or Amazon Alexa. Another implementation is that of adding chatbots to large communication platforms, e.g. Facebook Messenger and Slack as noted in section 1. A third manifestation is that of releasing a standalone product. If a firm already has its own smartphone app, the chatbot can be implemented into that app, or a separate CUI app can be released. Last but not least, CUIs can manifest themselves in robotic systems. For example, at Info Support, applications are being developed for the humanoid system *Pepper*, which can be observed in Figure 1[17]. Pepper is meant for human interaction, making conversation design an important factor of successful application of the robot.

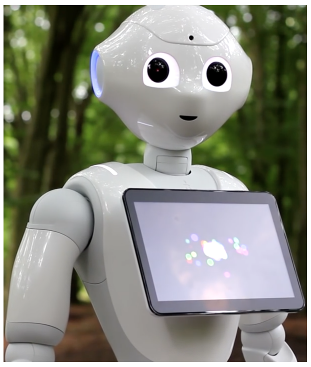


Figure 1: Pepper

It is important to note that this research focuses on the service behind these manifestations, i.e. the chatbot component itself. The difference between voice-enabled and purely text-based CUIs will be discussed at several points, but implementation, such as into a robot like Pepper, is kept out of the scope of this research.

2.5 Future of CUIs

"By 2020, the average person will have more conversations with bots than with their spouse." According to technology research and concultancy firm Gartner[18], chatbots will be an indispensable part of the average person's life by 2020. Another article from last year notes that the *chatbot landscape* at this point is like the Web in 1995 and mobile phone apps in 2008[19]. Both of these have developed tremendously since then, from which one can conclude that CUIs are only in their infancy, with most significant developments still to come. This yet again stresses the importance of research into this field. Artificial intelligence (AI) behind conversational user interfaces will keep improving itself in the foreseeable future. Improved AI algorithms for chatbots should result in better conversation analysis, increased context awareness, and more possibilities for developers to create increasingly better conversational experience.

[17]`https://carriere.infosupport.com/opdracht/de-mogelijkheden-van-pepper/`, accessed June 30, 2017

[18]`https://www.gartner.com/smarterwithgartner/gartner-predicts-a-virtual-world-of-exponential-change/`, accessed July 4, 2017

[19]`https://marketingfacts.nl/berichten/chatbots-facebook-inzet-chatbots-messenger`, accessed July 4, 2017

3 Specifying CUIs

To properly write functional requirements for CUIs, other perspectives and methods need to be implemented than for software that is purely based on a Graphical User Interface (GUI). A CUI can be based on chat or voice, where a voice-enabled CUI already adds several additional basic components, as described in section 2.3. Besides those components, there is also additional context that needs to be taken in mind when developing a CUI such as background noise, other people talking to each other in the background or someone talking to the person that is interacting with the CUI. Speaking with a conversational interface is, unlike most common software interactions, often non-linear. Users do not have to specify certain information at specific times and are not required to give a specific sequence of words for the CUI to work. Software development of the components is important, especially the natural language understanding (NLU) component, but that is already provided by the bot frameworks. Therefore, the specification of CUIs differs from software development for GUIs, as most of the time designing the system will be spent on designing the conversation, rather than coding the software.

This section presents the Conversation Design Framework (CDF) for specifying the conversation that was created during this research. The CDF is created as a a best practice design guideline for specifying an effective and efficient conversation. Among other things, the proposed model is based on insights from previous literature, design principles given by bot frameworks (section 2.3.1) and insights derived from Google I/O 2017[20]. The model has also been presented for validation and feedback to three professionals within Info Support that have experience with designing chat bots and with artificial intelligence. The model has seven distinct phases and is displayed in Figure 2, a larger version of the model can be found in Appendix A. Throughout this chapter, examples will be given as to how this model can be applied in practice. The example used for this is a CUI for an airline company, which was already addressed in section 2.

3.1 Target Functions

In order to know what conversations need to be designed exactly, the core functions of the system need to be specified. These core functions will serve as the goals of the CUI and are to be used in the rest of the process. In the example of a flight ticket ordering system, goals would be functions such as providing available travel routes from a specific airfield ordering a ticket and canceling a ticket.

When an organization already has an existing application, core functions can be derived from that. If the CUI is a new application, one could approach the CUI goals as writing functional requirements for conventional software. In most cases, only the domain-specific goals should be specified here. Sometimes general small talk is preferred to be included in the interface as well. Small talk implies out of context conversations, for instance when a user asks the chatbot to tell him a story, what its favorite color is, or for a dinner suggestion. Whether or not small talk is desired in a chatbot will depend on the product and the firm behind it. For example, an airline that targets adolescents and focuses on customer intimacy as a value proposition may wish to implement small talk as well, whereas an airline that focuses on efficient ordering may not.

[20]`https://events.google.com/io/`, accessed May 31, 2017

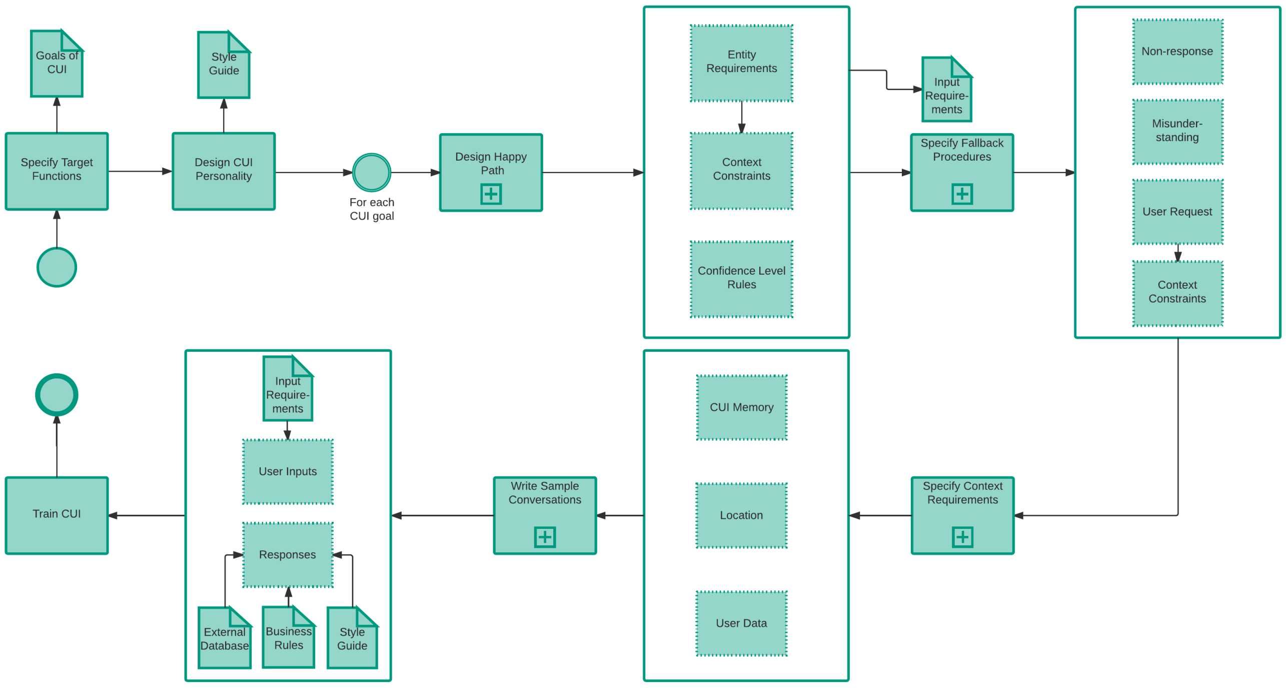


Figure 2: Conversation Design Framework (CDF)

Api.ai allows automatic small talk integration, but other frameworks currently do not. However, the automatic integration would be a generic set of inputs and responses, which still needs to go through certain phases of the framework. The minimal requirement of this is to pass this by the style guide in order to match the conversations to the bot's personality, as elaborated in the next section.

3.2 CUI Personality Design

During the session "Building apps for the Google Assistant" at Google I/O, Abrams gave the advice that the conversations of a conversational interface should be designed by creating a persona for it first[21]. A speaker from another session on storytelling[22], Oren Jacob, seconds this and believes a designed personification for the conversational interface is essential. Prior to getting in the industry of CUIs, Jacob had worked at Pixar for 20 years where he learned a lot about screenwriting. He explains that people will personify chatbots themselves, whether conscious or subconscious: "If you hear a thing or see a thing talk back to you, you will project human traits onto that thing." He gives the example of people personifying the fish in the Pixar movie Finding Nemo. This phenomenon is in fact proven in science by the name of anthropomorphism and is defined as the tendency of people to attribute human characteristics, motivations and mental states to non-human objects (Epley, Waytz, & Cacioppo, 2007).

[21]`https://events.google.com/io/schedule/?section=may-18&sid=8075afbc-88ab-4fbe-ab7f-1cf9f480afdd`, accessed June 2, 2017

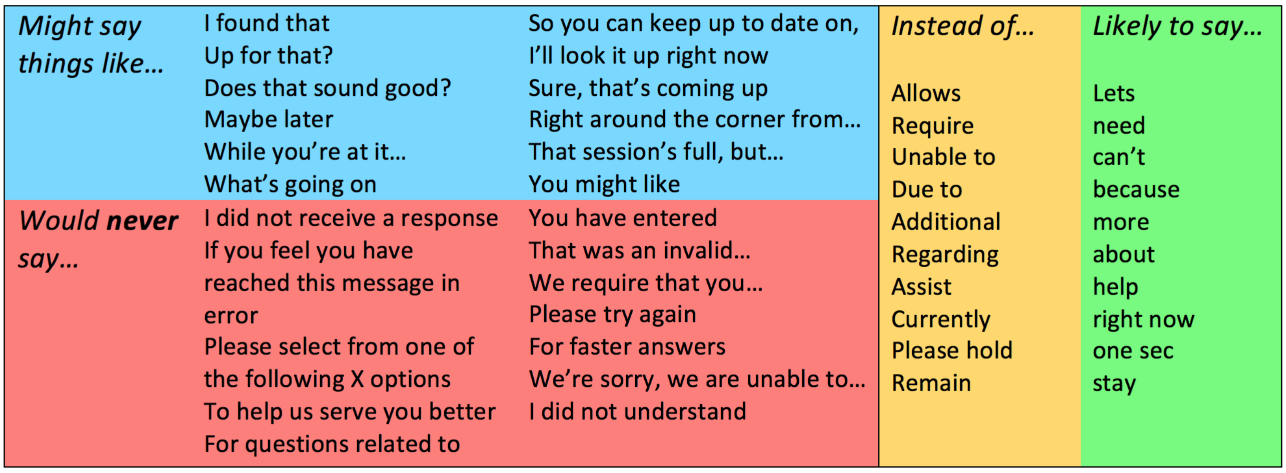
[22]`https://events.google.com/io/schedule/?section=may-19&sid=b7c64a7d-20d3-43b6-9f46-8123f65468fd`, accessed June 2, 2017

Jacob explains in his session that personification of the bot is good because users personalize chatbots and thereby create a relationship with it. However, he also describes the other side of the story when a personality has not been designed properly: "if you don't spend the time crafting that character and motivation carefully, you run the risk of people projecting motivations, personality traits, and other qualities onto your App and brand that you may not want associated with them." Brad Abrams researched apps that had good retention and found those all had a strong persona. Therefore, designing a personality has been put at the roots of the CDF. Abrams suggests four steps for creating a proper persona:

1. Define brand characteristics.
2. Correlate to attributes that will define functional design principles.
3. Define some attributes to be infused into the voice, style of writing, and personality of the dialog.
4. Create a style guide.

The first step is defining the brand's characteristics. Here, the designer has to think about what words describe the brand itself. Examples given for these are knowledgeable, helpful and encouraging. The second step is correlating these attributes to design principles by thinking about how they will manifest in the design. Data rich, rewarding and proactive are given as examples for this. The third action is translating these brand characteristics into attributes of the CUI. This means defining what personality traits match the design principles of the prior step, such as a geeky, eager or humorous personality.

Based on these personality traits, the style guide can be made at last. The style guide has three categories: utterances the bot might say, utterances it would never say, and suggestions of replacements for certain words by others. By having a style guide, different people can write conversations while keeping the dialogue "tight and crisp." Abrams gave an example of a style guide, reconstructed in Figure 3.



Might say things like…	I found that	So you can keep up to date on,
	Up for that?	I'll look it up right now
	Does that sound good?	Sure, that's coming up
	Maybe later	Right around the corner from…
	While you're at it…	That session's full, but…
	What's going on	You might like
***Would never** say…*	I did not receive a response	You have entered
	If you feel you have reached this message in error	That was an invalid…
		We require that you…
		Please try again
	Please select from one of the following X options	For faster answers
		We're sorry, we are unable to…
	To help us serve you better	I did not understand
	For questions related to	

Instead of…	*Likely to say…*
Allows	Lets
Require	need
Unable to	can't
Due to	because
Additional	more
Regarding	about
Assist	help
Currently	right now
Please hold	one sec
Remain	stay

Figure 3: Persona style guide

Many other chatbot designers stress the importance of persona's as well. In an article at Chatbots Magazine[23] is described that establishing a personality as the starting point of the chatbot's general design process. Thoms, a founding partner at bot conversation design agency Xandra, wrote a follow-up article[24] on Oren Jacob's session on storytelling, where the process of building a personality for the interface is further elaborated. She argues that bot personalities, which is the third step in the process Abrams described, should be based on the organization's current branding or on the user. At her company, personas are built by assigning a Myers-Briggs personality type to it. The Myers-Briggs personality type is an established and often-used framework that uses four personality traits and each of these four compares how one scores on a scale of two extremes (Myers, 1962):

- Orientation of focus: Extraversion (E) versus Introversion (I);
- perceiving function: Sensing (S) versus Intuition (N);
- judging function: Thinking (T) versus Feeling (F); and
- orientation to life: Judging (J) versus Perceiving (P)

Per category, one gets either one of the extremes assigned to him, resulting in sixteen possible personality types. In certain tests that are available, the magnitude of a certain type is also taken into account. A person can, for instance, score 5% Extraversion over Introversion. This is a relatively small difference, but the Myers-Briggs spectrum still assigns that individial with the E. Adaptions of this framework can interpret meaning of this magnitude as well.

Thoms explains the efficiency of this method due to the wide range of material available on this analysis and how that can be used to verify if right personality is chosen: "Amongst the wealth of material on Myers-Briggs, you will also find assignments to suitable careers and jobs per personality type. Matching this with the appropriate bot 'job' will confirm whether you are building the right personality." By performing this personality test on the target user, either by letting them make it, reconstructing their behavior or by applying this to the brand image, a proper persona can be sketched. Using the persona, a proper style guide can be created.

With the example of a system for an airline, the style guide may vary significantly, depending on what the target audience is. For a relatively expensive airline, the target audience may be mostly professionals over 40, resulting in a style guide aimed towards professional tone and formal language. A budget airline, however, may opt for a more informal approach and thereby target younger people. The expensive airline might have 'affirmative' in the style guide as a confirmation utterance, while the budget airline may go with 'got it' for the same situation. When a bot targets multiple distinct audience groups, multiple personalities and style guides can be implemented, given the condition that the bot has a way to distinguish which audience it is talking to, for example through the logged in user account that is connected to the interface. With domain-specific CUIs, it is important to keep in mind users can use both domain jargon and unrelated utterances. For example, with the airline system, a user can ask from what terminal his flight leaves, but can also ask "where do I enter the plane?"

[23]`https://chatbotsmagazine.com/writing-skills-at-the--of-chatbot-ux-design-17762b906a57`, accessed June 12, 2017

[24]`https://www.xandra.com/blog/a-guide-to-bot-personalities`, accessed June 12, 2017

3.3 Happy path

The next step in the design process is sketching the happy path of the conversation. The happy paths can be seen as the user stories for the CUI that set the minimal success criteria. User stories are often used in software agile software environments and are often to be found pleasant to work with (Lucassen, Dalpiaz, Van der Werf, & Brinkkemper, 2016). The advantage of user stories is that identification of requirements happens early on in the process. In the Conversation Design Framework (CDF), the happy path is used to identify entities, context constraints and confidence level rules.

Austin Beers wrote an article on making human-centered bots[25]. Besides giving the recommendation to create a persona for the interface, Beers recommends acting out conversations in an improvisational style and mapping out conversations. With the *improv style*, one person of the team has the role of the user, another team member acts as the CUI, and a third person takes notes on the conversation. This is a very suitable method for creating the happy paths of the conversation. Testing of natural language dialogue systems is often done with the Wizard of Oz (WOz) methodology (Dahlbäck, Jönsson, & Ahrenberg, 1993), which is very similar to this method of creating happy paths of the conversation. With WOz testing, one user takes on the role of the dialogue system, the wizard, who has access to the same databases that the original system has access to, and another person talks to it as a regular user of the interface. The wizard can execute queries to find the information needed to answer the user's question. This proven methodology is originally designed for testing but also shows great potential for adoption in the early design process.

The second part of Beers' recommendation is mapping out the conversation: "Teams collaboratively make decision trees of their bot's conversations, mapping out everything they want a bot to achieve. Each part of the conversational flow is illustrated with post-it notes, illustrating what the bot 'Hears', 'Says', and 'Asks'." A flaw of this methodology is that it assumes the conversation can be mapped as a decision tree, although it is not a linear process. As will be elaborated in section 3.5.1 into more detail, conversations can resume older states besides just following the active one, which is why decision trees cannot cover the entire process.

However, the happy path is likely always to be linear, therefore it is suitable at this point in the design process. The entity requirements can be mapped out using these two methods, after which the context constraints and confidence levels can be defined. These three together form the input requirements of a specific goal for the conversational user interface.

3.3.1 Entity Requirements

After the happy paths for the conversation have been constructed, all the required entities must be specified. As mentioned in section 2.3, entities are information elements such as 'Departure', 'Destination' and 'Date'. The entity requirements should cover all the possible data fields that the user can choose in the process. It is likely that most of these entities are already incorporated

[25] `https://medium.com/the-charming-device/how-to-design-intelligence-3-ways-to-make-human-centered-bots-76c5ff7524df`, accessed June 13, 2017

in the happy paths, but the system designers should always check if values have not been missed. If a GUI system is already in place through which users can complete the same process that is being designed for the GUI, all the entities that can be chosen there, need to be implemented in the CUI for it to be able to take over the role of the GUI.

Two categories of entities are incorporated in the CDF: entities that are required for the conversation to reach its end goal, and optional entities. Required entities should always be asked specifically by the CUI if they are not given by the user, optional entities not necessarily. With the airline system examples, required entities are 'Departure', 'Destination' and 'Date', while optional entities are 'SeatChoice' and 'Class'. The required entities are crucial for ordering a ticket, but the user does not always have to specify specific seats. However, it is strongly recommended to at least ask the user if they want to choose a specific seat and if they want to be upgraded to business class, with a simple yes or no answer, as long as the system does not know the user's preferences. Once it knows the preferences of the user, the system can simply ask the user to confirm if the preferred optional settings are okay for the new order.

3.3.2 Context Constraints

Several steps in the process will have constraints as to how they should be handled, which is mainly linked to the entities involved in those specific steps. This step is especially important when the CUI uses voice, where the possibility is significantly bigger that other people intercept the message as well as they may simply overhear it, which is why certain entities should never be said aloud by the system. For those entities, a privacy policy should be set. Some data can be displayed on screen but should never be told to the user by voice. For example when a user wants to order a flight ticket, the system might ask the user if he wants to use his credit card for the payment, and display the last four digits of the card on the screen. However, for somewhat obvious security reasons, the CUI should never broadcast the credit card's security code to the user if verification is needed and neither should the user be asked to do so.

A way to solve this is to, let the CUI refer the user to the screen, for example by saying "Can we charge the order to your usual credit card?" If the user wants to make sure which card this is and asks for it, the system can refer the user to the screen. When the system does not have a display, for example with a Google Home smart speaker, the user cannot be referred to a screen. In that case, an implementation of this could be to have the user confirm the last four digits himself. The user can now decide whether or not he is in a private area where the digits can be said aloud, and handle accordingly.

3.3.3 Confidence Level Rules

Confidence levels determine how confident the bot framework should be with giving a certain response. Since the user does not have to speak a specific sentence in order to trigger an action but instead can use natural language, the framework calculates how certain it is that its response corresponds to the user's request. The Wit framework defines it as that it "represents the confidence level of the next step, between 0 (low) and 1 (high)."[26]

[26]`https://wit.ai/docs/http/20170307`, accessed June 13, 2017

In previous research to voice user interfaces, a case study was presented where three *reliability degrees* were possible: low, average and high (Salvador et al., 2008). When the reliability degree was low, the system was configured to ask the user to repeat himself. When it was average, the system asks the user for confirmation by repeating the conceived entity values, and when a high-reliability degree was recognized, the system requested no confirmation by the user and immediately performed the action.

A configuration as such would be ideal for CUIs as well. However, the current bot frameworks do not allow developers to specify three levels of confidence. The frameworks enable the developer to set a minimally required confidence in order to proceed under which a fallback procedure can be specified, meaning only two levels of confidence can be defined. For example, when the *Machine Learning Classification Threshold*, as it is called in Api.ai[27], is set to 0.5, then everything with a confidence level under 0.5, or 50%, will trigger the fallback event. Due to this reason, an enterprise should opt for only a distinction between high and low confidence if it makes use of one of those frameworks, and keep the three-level model as an ideology for if this ever becomes available in the future.

The confidence level should not be defined just once during the happy path, but instead has to be reconsidered at every step. For example, when the system is 75% sure that the user wants to order a ticket to London then that may be enough, and asking the user to confirm it before purchasing the ticket will suffice. However, the system should not be *only* 75% sure whether or not the user confirmed that he wants to buy the ticket. In that case, the threshold for the confidence level rule is likely to be set at 95% or even 99%. Designers of the CUI should always be careful of these differences: accidentally selecting a wrong destination city may not hurt the user experience if the user can still correct it, but the same cannot be said about the system actually purchasing the ticket when the user did not want that.

3.4 Fallback Procedures

Fallback procedures need to be set in case the user does not respond or because the user wants to go back to a previous state of the conversation. These procedures can, for a part, be seen as pressing the back-button in a standard GUI. These procedures are not be the same at every step in the process. For example, at some places it will mean having to start the conversation all over again, while at other points in the process it only means removing one data entry. The same goes for CUIs, which is why this step should not be overlooked in the design process. In voice-enabled CUIs, fallback procedures may also be triggered when a user simply does not give any answer whatsoever. The third way a fallback procedure can be triggered, is when the system does not understand what the user said. This happens when the user asks a question that is not in the interface's domain or because the system did not fully comprehend the user's input. These three fallback triggers are each discussed separately in this subsection.

Certain fallback procedures may apply to the whole system and not just the specific goal that is being designed. For example, fallback procedures during the payment process may be standardized

[27]`https://docs.api.ai/docs/machine-learning-settings#ml-classification-threshold`, accessed June 13, 2017

throughout the entire system instead of setting new rules for every CUI goal. Therefore, the position of setting fallback procedures is not static in this model. However, if global fallback procedures are specified for the entire CUI, it is still recommended to review if those procedures fit with the current goal.

3.4.1 Non-response

A user may have several reasons for not responding to the CUI. One might not be paying attention to the interface, he might not have heard the question properly, or it is possible that he already has the information he was looking for, rendering the continuation of the conversation he had with the interface useless. It is important to keep this in mind while designing conversations, as this may significantly influence the user experience of the interface.

When the system being designed is a text-enabled CUI, other procedures are to be specified than for a voice-enabled CUI. With pure text-enabled CUIs, users are usually always able to recall what the bot's question is, simply by looking on their screen. With voice, however, the user does not have access to the question anymore. During a presentation at Google I/O, Daniel Padgett stressed the importance of keeping in mind that voice is always linear and ephemeral, which means that it is always moving forward, whilst fading as soon as something is said[28]. That is why he remarks that CUI creators rely heavily on "the user's knowledge of conversation and what they can recall". Non-response fallback procedures should therefore not be overlooked Implementing a wrong fallback method may cause irritation with the user, which is why non-response requires other handling methods than other fallback triggers.

At another I/O on fallback procedures[29], the recommendation is given to rephrase the question at such a fallback or to add variability to the question. This way, a user may not even always be aware of the fact that a fallback was initiated. If the user does not realize that anything went wrong during the conversation, his user experience is likely to be much better than when he constantly gets reminded of improper use.

3.4.2 Misunderstanding

Fallbacks are triggered when the CUI does not understand what the user's input means. Scott Ganz, writer and Creative Director at bot platform PullString, wrote an article on this type of fallback[30]. He wrote that there are two reasons a fallback might occur: when the user says something that was not designed, or when he says something that was designed, but in wording that was not. The first reason refers to the user asking questions that are not incorporated in the domain of the CUI. Fallbacks are triggered when the system does not properly understand what the user means, for example when the natural language understanding fails because there is too background noise. This results in a confidence issue, as explained in section 3.3.3.

[28]`https://events.google.com/io/schedule/?section=may-18&sid=4cf193f0-22b3-4f6d-8979-fa27be37ef17`, accessed June 16, 2017.

[29]`https://events.google.com/io/schedule/?section=may-18&sid=69d89fd9-f05a-452e-85ba-fc636054789e`, accessed June 17, 2017

[30]`https://www.pullstring.com/blog/how-to-use-fallbacks-interjections-and-believability-credit-to-write-convincing-computer-conversations`, accessed June 17, 2017

When a bot is focused on utility rather than simulating character, the recommendation is given that the bot should explicitly tell the user that it did not understand him properly, and ask him to repeat himself in another way. Goal-oriented CUIs will most of the times be written for utility, which is why that recommendation stands in this model. The reason for telling the user it was not understood in this case is that it will help move the conversation forward faster and thus helps reach the functional goal of the conversation faster.

3.4.3 User Requests

Users may regularly ask the CUI to 'go back' a step in the process. They may, for instance, decide they want a different departure location because it is cheaper. In most cases, the CUI should allow the user to make these changes. It should, however, remember the old entries in case the user changes his mind again. By remembering the old data as well, the user can easily resume where he left off with his initial choice.

However, there will also be cases where the user's navigation abilities are to be limited. For example, a user should not be able to change the departure city anymore when he is in the process of making the payment of the flight, without canceling the order first. The user should also not be able to change a payment method through a fallback after the payment is already completed. This may seem trivial but can significantly affect the system if overlooked. Therefore, context constraints need to be specified at certain stages in the conversation. The best way to solve this is by separating the conversation into phases where specific rules apply. This can be seen as the separation of components within conventional software. Component-based software architecture is a common principle with many advantages, including increased maintainability and overall system quality(Cai, Lyu, Wong, & Ko, 2000). This way, components such as the payment processor can be designed once for the CUI and reused for any goal that includes a payment. Once the payment of a flight ticket order is initiated, the user should unnoticeably get directed to a 'new' conversation. The payment component will give a result to the 'main conversation' after which the normal conversation is continued. This way, only after the payment is cancelled can a user change the entity fields such as the departure location.

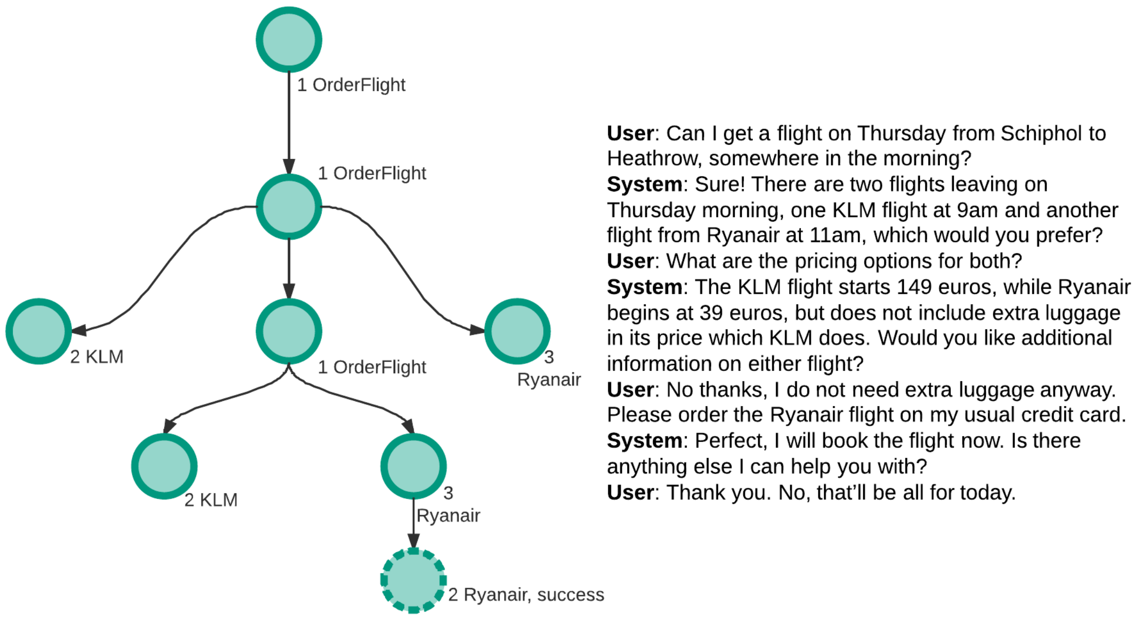
3.5 Context Requirements

User and system context are a crucial part to make the conversation sound natural and to get the best user experience for CUIs. In the Conversation Design Framework (CDF), three distinct categories are identified for the context requirements of a CUI's output: memory capabilities, location and user data.

3.5.1 CUI Memory

The conversational interface needs to remember a certain amount of data about the conversation, thereby simulation real memory. Research at Maluuba addressed this matter and created the Frames model (Asri et al., 2017). A frame is one of the paths that the conversation has taken. For example, if the user is booking a flight from Schiphol to Heathrow, but then tells the CUI that he wants to fly to Stansted instead, the system should remember both and allow the user to switch back to Heathrow later on if he wishes to do so.

A representation of this is given in Figure 4. A conversation always starts with the first frame. When the user asks for flights, two options are given which results in two new frames, one for each destination. At this point, the user does not ask anything specific about either flight. Therefore, frame 1 is continued. The user asks what kind of prices he can expect with either flight. As this new question regards the two previously initiated frames, those are continued here. Had the user, for instance, told the agent that he did not like either flight and wanted information about two flights in the afternoon, frame 4 and 5 would have been initiated. In this example, the user confirms that he wants to pursue with the first flight option, i.e. frame two, which is then continued.



User: Can I get a flight on Thursday from Schiphol to Heathrow, somewhere in the morning?
System: Sure! There are two flights leaving on Thursday morning, one KLM flight at 9am and another flight from Ryanair at 11am, which would you prefer?
User: What are the pricing options for both?
System: The KLM flight starts 149 euros, while Ryanair begins at 39 euros, but does not include extra luggage in its price which KLM does. Would you like additional information on either flight?
User: No thanks, I do not need extra luggage anyway. Please order the Ryanair flight on my usual credit card.
System: Perfect, I will book the flight now. Is there anything else I can help you with?
User: Thank you. No, that'll be all for today.

Figure 4: CUI Frames example

The researchers introduce a task called *frame tracking*, which is an extension of regular state tracking: "In frame tracking, the agent must simultaneously track multiple semantic frames throughout the dialogue. For example, two frames would be constructed and recalled while comparing two products – each containing the properties of a specific item." This allows a user to switch between different parts of the conversation easily. For example, while comparing two flights, two frames are created. One might ask the CUI additional information about one flight. This will then be saved in the corresponding frame, increasing its size. However, the user is then able to ask questions about the other flight, causing the CUI to switch to that specific frame. All the information about the first flight is still available in the system, even though it is not the active frame at that moment. This can be seen as having a new tab in a web browser for every option. Successfully implementing frames and frame tracking will allow the user to easily switch between the options and compare them.

Switching between frames must also have context constraint specifications. For example, a user should not be able to switch frames during payment. As described in section 3.4.3, phases such as the payment should be seen as separate components in the conversation. It is important to restrict frame switching at these places, making it impossible for the user to change to another frame without either finishing or canceling the payment first.

Besides the memory within the conversation, the permanent memory of the CUI also needs to be specified at this phase. For data analysis and system improvements, it is likely that everything the user says to the CUI is recorded, but not everything should be told back to the user. The reason for this is that the user himself will also not remember every detail of previous conversations and to keep the conversation natural, the CUI should not either. There should be different rules for conversations that were finished successfully, i.e. where the CUI goal was reached, and conversations that were not finished. For successful conversations, it is usually not necessary to recall any data that was not included in the frame that reached the goal of the conversation. Therefore, in this case, it is recommended to let the interface only give information about the successful frame to the user. For example, when a user successfully booked a flight from Schiphol Airport to Heathrow, the system can remind the user of this flight if he ever wants to go to London again in the future, but it can neglect that the user also looked at flights going to Stansted unless the user specifically asks for this.

Conversations will not always reach one of the CUI's goals. Users may close the application, get disconnected or simply tell the CUI that they do not want to continue the current conversation and start over. In case the conversation was not finished, the user may want to pursue the process at another time. In this case, the system should give the user more information than with recalling a successful conversation. The user may want to know between which three departure cities he doubted at the time of the conversation, which means the CUI should recall three frames. During the design process, the creators need to specify what the system should recall to the user at what phase. For instance, if the user got disconnected to the system during the payment, the user could be asked if he wants to continue his payment, and given all the information about the ticket, but it may be confusing if the interface agent also tells the user about what other two locations he was thinking. Special cases such as this should be specified beforehand so that it can be implemented properly whilst writing the sample conversations.

3.5.2 Location

Voice might not be preferred in busy areas such as a train station. When using a voice CUI, there need to be definitions on when to switch to text only. This should, however, be implemented in such a way that the user has the best experience. Therefore, the user should also be able to rollback (or turn off) such a function. For example, A user might be wearing headphones, which means that he will not be bothered by the business of the train station.

In some cases, several goals of the interface, or even the whole system, should only be accessible at a certain location. This may very well be the case when the CUI is made for internal processes of an enterprise. A warehouse CUI system might, for instance, limit the goal to order new stock to only be accessible from inside the warehouse, where the employee has a real insight into what stock is currently available.

3.5.3 User Data

To enhance information flows between different users of a system, designers need to specify whether user data fields are public or private. Sometimes the system may not know something but still knows who else might be able to help the user. In other cases, the user might ask specific data about another user. During this step in the design process, it should be specified what user data can be provided to others and what data cannot. In the example of the airline system, a user might ask the CUI what destination a friend has on their public wish list in order to give them a surprise gift trip to the place of their liking. However, the user should not be able to ask the system about the credit card details of that friend and tell the system to charge one ticket to that person's card. Shared data must never contradict the privacy specifications set with the input requirements.

3.6 Sample Conversations

This phase in the design process is where everything from the previous parts come together. At this point, all the conversations need to be written that make all features of the CUI goal possible. The best method is to design the complete happy path conversations first, writing both inputs and responses per step. For every step, there should be both multiple inputs and multiple responses. After the happy path has been completely designed, all the possible routes that are possible need to be designed.

During the Google I/O[31] that was mentioned in section 3.2 as well, Abrams compares this process with writing a movie script, "except you get to write lines 1, 3, 5 and 7, and somebody you have no control over gets to write lines 2, 4, 6 and 8, and you have to use that to come together with a beautiful experience." This is why it is crucial to design the sample conversations carefully and to specifically model all the turns that a conversation might take at any point in the process.

3.6.1 User inputs

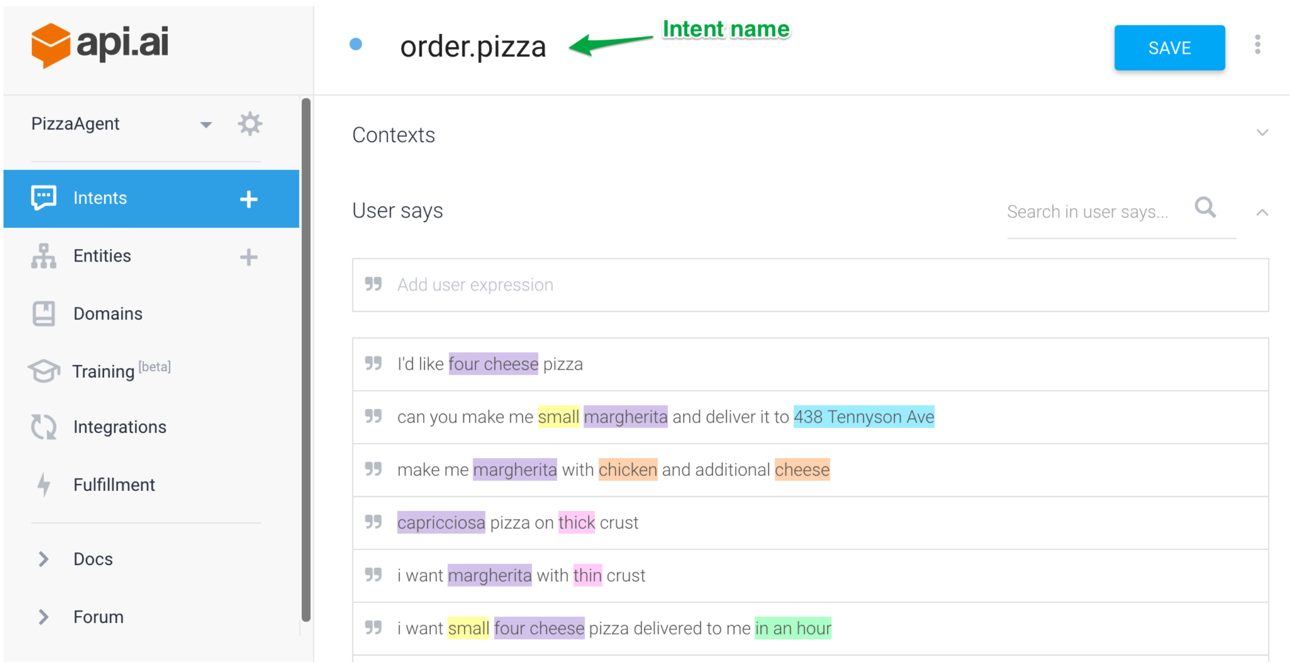
Even though the designers do not have control over what the user is going to say to the CUI, user inputs still have to be properly specified. When using bot frameworks, designers need to define what part of the input concerns intents and entities. Inputs must be created to meet the input requirements that were set during the happy path creation.

An example of how this looks in Api.ai[32] can be seen in Figure 5. Per intent, user input examples are modeled. With those inputs, the designer can specify which parts of the sentence refer to which entities.

Tools such as Api.ai make use of machine learning and complex algorithms, from which can be concluded that designers do not have to specifically write out every possible variation of what the user might say. Input requirements should be properly followed, but the rest of the sentence is more flexible. However, it is highly recommended to write all the different inputs the user might

[31] `https://events.google.com/io/schedule/?section=may-18&sid=8075afbc-88ab-4fbe-ab7f-1cf9f480afdd`, accessed June 2, 2017

[32] `https://docs.api.ai/docs/get-started`, accessed June 19, 2017



I'd like four cheese pizza

can you make me small margherita and deliver it to 438 Tennyson Ave

make me margherita with chicken and additional cheese

capricciosa pizza on thick crust

i want margherita with thin crust

i want small four cheese pizza delivered to me in an hour

Figure 5: Api.ai input design

give that the designers can think of, as the cited Api.ai guide explains: "The more of the relevant and diverse examples you add to the intents, the smarter your agent will be." Matching the inputs as closely as possible to what the user's input will be, maximizes the system's confidence in its operations and thereby makes it more efficient. However, it is not a disaster if the inputs do not completely match the user inputs either: the machine learning will learn from its users and adjust and add inputs to intents by itself, thereby adapting to the user group and increasing those confidence levels automatically.

3.6.2 Responses

At every step in the conversation, responses need to be created that cover all the possible turns that the conversation might make. This includes replies to the happy path inputs, as well as all the fallbacks and other inputs that cause the conversation to move in another direction. For a system to sound natural, many responses should be specified for every direction. For example, ten different responses could be given for every step in the happy path, from which a random one is given to the user.

The designed responses must be based on the style guide, the external database and the enterprise's business rules. Whereas the external database and the business rules define *what* the response should be, the style guide defines *how* it is worded. All responses must be specified according to these in order to get a fully coherent conversation that provides the correct information.

A user's time is valuable, so information must be provided to them in an efficient manner. A renowned theory on this matter is the Cooperative Principle theory. Introduced by Grice, the theory is divided into four *Gricean maxims*:

- quantity;
- quality;
- relevance; and
- manner (Ho & Swan, 2007).

Ho and Swan adapted and proved the use of this theory to online environments. The maxim of quantity states that as much information or materials should be provided as is necessary, and no more than that. The maxim of quality specifies the principle that information should only match what the environment (in this case the CUI) believes to be true, i.e. it should not lie to the user. Besides that, it also argues that information must not be provided if insufficient evidence is available to support it where necessary. The third maxim explains that provided information must always be relevant at the specific part in the conversation. For example, when planning a flight from Schiphol to Heathrow, the CUI should not give the user information regarding flights to New York. Last but not least, the maxim of manner states that information needs to be "logically organized and clearly presented." This means that a user should always be able to understand exactly what the conversational agent tells him without any ambiguity.

These four principles must be followed whilst writing responses for the CUI to efficiently provide the user with information. However, the conversational aspects of the interface may contradict principles such as the maxim of quantity, since adding extra words is often necessary in order for the interface to sound naturally. Therefore, the principles should be followed only to the extent where it does not negatively affect the conversational elements of the CUI.

3.6.3 Conversation Review

With conventional development, code review is a common software engineering practice (Bacchelli & Bird, 2013). As explained by Bacchelli and Bird, "peer code review, a manual inspection of source code by developers other than the author, is recognized as a valuable tool for reducing software defects and improving the quality of software projects." Sample conversation design involves implementation in the enterprise's software and external databases, where code review can be very valuable. Besides that, this principle can also be implemented for the conversations themselves. For this, the term *Conversation Review* can be defined as peer reviews of conversation flows by anyone other than the author. The reviewer must validate if inputs are adequate, if any inputs should be added, if the responses sound natural and if they adhere to the style guide.

3.7 CUI Training

To prepare the framework properly and increase the machine learning knowledge of the framework, training a new goal of the conversational interface before releasing it is necessary. This would not be necessary if the sample dialogues match everything a user could say to the interface, but designers will never truly be able to catch *everything.* Using the sample dialogues but changing syntax or the entity values will already help the system learn. For example, if a sample input is 'book a ticket from Schiphol to Heathrow for Thursday morning' as with the example in Figure 4, the training input could be 'book a ticket from New York to Schiphol for Tuesday 2pm' (changed entities), or 'Can I get a flight on Thursday from Schiphol to Heathrow in the morning?' (changed syntax).

This way, the CUI learns other inputs for the same intents next to those already provided. A minimal amount of training should be performed on the system before releasing it to the public in order to catch possible issues with the design. Without fine-tuning the system through training, one risks its users discovering these matters, which could potentially affect the user experience significantly. If done properly, training the CUI should result in the interface catching most, if not all, of the user's intents after releasing the product, thereby maximizing user experience.

3.8 Roles Within The Framework

In GUI software development, a good proportion of the design process is dedicated to software coding. This is yet another reason why CUI design is different from conventional software development: most of the design process does not even involve software developers. This subsection will give a small overview of what kind of roles are involved in the CDF.

The first step of specifying the CUI goals is not different for conversational agents than it is for GUIs. If software is being developed for a third party client, a product owner that handles the communication with the external stakeholders could come together with those stakeholders and make the goals together. The second step is where it gets more complex: a product owner could define brand characteristics together with the client but is most likely not trained to build personalities. For this, the recommendation is often given that it should come from people who are trained in this. In the post[33] that Jess Thoms wrote, which was also referred to in section 3.2, she also mentioned what kind of people are suitable for this: "skills to build a personality come from writers, designers, actors, comedians, playwrights, psychologists and novelists. The integration of these skills into tech roles have sprung terms such as conversation designer, persona developer, and AI interaction designer."

If a CUI is being developed to extend a current GUI interface, input requirements can be partially derived from the current system. The happy path will closely follow the GUI process flow for a large part, and entity requirements will, for the most part, match the capabilities of the software. In that case, this step will take most time with defining confidence level rules and context constraints. However, if a current system does not yet exist, all the input requirements still need to be defined.

As suggested in section 3.3, one way to do this is *improv style*, where the system is simulated by a person. A recurring term for people involved in writing for conversational interfaces is the 'User Experience Writer', which may also be involved in this phase already. An editor for 'UX Booth' did research into this term[34] and found that an increasing amount of IT corporations are hiring UX writers. The editor defined UX writing as "the act of writing copy for user-facing touchpoints.

[33]`https://www.xandra.com/blog/a-guide-to-bot-personalities`, accessed June 12, 2017
[34]`http://www.uxbooth.com/articles/what-is-ux-writing/`, accessed June 20, 2017

This copy must not only embody the voice of the organization, but must also be considerate and useful for the user." The UX writer becomes a crucial part of the design team in the phase of writing sample dialogues and should be employed mainly for that phase, but can also be useful when specifying happy paths. Some of these skills will overlap with the skills required for designing the bot's personality. Depending on the exact expertise of the personality designers, this person might also take the role of UX writer upon himself. The functional happy paths together with their corresponding input requirements need to be aligned properly with the wishes of the product owner, who should therefore agree upon these before moving on to the next step.

Fallback procedures are best specified by those people who also made the happy path and input requirements. They will already have a basic feel of the conversation flow in the successful scenario, and can greatly use this to their advantage to speed up this phase. Context requirements share many aspects with the happy path and input requirements as well, making those same people a suitable candidate for this phase. However, it is highly recommended to involve the software developers that are to implement the system in this phase as well. Especially with the CUI memory, the capabilities of the developers and their resources will greatly affect to what extent this can be implemented. Bot frameworks do not allow much customization towards memory, meaning that it will need to be partially implemented manually by the developers. Api.ai, for example, allows designers to set a context lifespan[35], which means it will remember a certain amount of requests before deleting old ones. Compared to the methodologies given at section 3.5, this is a very simplistic implementation of memory that leaves much room for improvement if resources and knowledge are available for this.

CUI training is the final step of the CDF. The UX writers who wrote the sample dialogues likely already implemented all their ideas on how the dialogue might progress into the sample conversations. Therefore, team members that have not taken a role in writing the sample conversations may add more value for training. This is also what is described with conversation reviews, as elaborated in section 3.6.3. If conversation review is already extensively conducted on a specific dialogue, the CUI is possibly already trained a satisfactory amount by other UX writers. Additional training may then not be needed anymore. If still deemed necessary, however, it should be performed by someone other than the writer of the conversation or the peer reviewer. This could be a third UX writer or someone with another function in the design team.

[35]`https://docs.api.ai/docs/concept-contexts`, accessed June 20, 2017

4 Specification Testing

As with any other software system, testing and evaluation must be performed on the system in order to verify that a sound system is designed. The minimal evaluation success is achieved when a CUI goal can be successfully completed. A part of this evaluation will be the functionalities of the software implementation. Certain evaluations of the CUI design described in this section will, therefore, be the same as that of conventional software. The back-end of the design can thus be tested using traditional methodologies, whereas other components will require new or adjusted ways of evaluation.

For testing user experience of the Voice User Interfaces, some methodologies have already been introduced. For example, a heuristics-based usability checklist was presented (2010) by Farinazzo et al. that consisted of 13 items:

- System feedback
- User diversity and user perception
- Minimizing memorization efforts
- Appropriate output sentences
- Output voice quality
- Proper entry recognition
- Natural user speech
- Dialogue start and instructions on interaction with VUI
- Natural dialogue structure
- Sufficiency of interface guidance
- Help tool
- Error prevention
- Error handling

Every item in the checklist has one or more guidance tips on how to evaluate it. A list like this has great potential for CUIs as well. However, much of this list concerns non-functional heuristics of the interface. Besides that, it is created for basic voice user interfaces and not optimized for more advanced CUIs. Therefore, this section aims to give evaluation methods for CUIs in particular. This is achieved by presenting methods of evaluation for every step in the design process of the Conversation Design Framework. Afterward, possibilities for automated testing are addressed. Combining these separate evaluation methods results in a proper evaluation for functional requirements of the CUI.

4.1 Personality

As the personality of the CUI lies at the core of the framework and conversation design, testing if it is a proper fit for the end users is imperative. The personality makes the bot more natural and may increase user retention, but it should be verified that the personality never interferes with the CUI's goals. As the user base may change over time, periodic evaluation might result in the best personality for the users. Adjusting the style guide means previously designed conversations need to be adjusted as well. Adjustment of the conversations to fit the style guide again could be partially automated. Minimal automation would be to have a tool that scans through the conversation designs and gives a list of places that require editing as an output.

The actual evaluation of the personality can be a complex task. A full reevaluation may be unavoidable if the application sees a significant change in user base, with a complete rebuild of the personality as a result. In that case, new sample conversations must be designed as well to fit the adjusted personality.

Evaluation of the designed personality does not currently offer any concrete methods. An indirect method may be through user experience (UX) tests, where users are asked afterward if they were comfortable talking to the chatbot and if they created a connection or friendly relationship with the chatbot. However, good or bad user experience cannot be linked directly to the personality as many factors influence the experience users have with a system.

A more direct evaluation of the personality can be done through A/B testing. With A/B or split testing, two different variations of the bot's personalities must be designed. These can either be two completely different personalities, or one personality that is implemented differently. For example, the first version (A) may have a very strong and present personality, whereas the variation bot (B) has a weaker personality that is not as present and noticeable in its responses. This can be implemented in a large quantitative research, or in qualitative ways through user experience tests where testers are tasked with comparing the two versions with one another.

This evaluation can be performed either with acceptance tests before continuing development or with the released product. This way, the current personality can be verified by comparing it to other possible implementations. However, one might argue that, especially with text-based CUIs, this tests the style guide more than the personality behind it. Additional research is thus required to set guidelines for concrete qualitative evaluation methods for a CUI's personality.

4.2 Happy Path

The conversation designs should minimally follow the happy path concept of the goal. However, testing if the happy path is indeed properly integrated into the design, requires minimal effort but can still overcome critical design faults. A small yet crucial step might have been overlooked in the design, which can be recognized by taking the happy paths through the final product.

Besides the factor of human error, the experience of employees at Info Support is that when more intents (i.e. CUI goals) are added to a framework, the more difficulty the Machine Learning has with identifying the correct intent. This experience is with Wit's framework, but is likely applicable to the other tools as well. Therefore, not only the current happy path should be tested once designers start to add more goals to the interface, but also the happy paths of goals that were already implemented.

4.2.1 Entity Requirements

Testing entity requirements, for the most part, involves testing whether entities properly register as such. With the airline ticket system, for example, the CUI must be able to distinguish departure and arrival location properly. Using natural language understanding (NLU), the interface must identify where the user wants to fly to and from which location he is leaving. Besides that, proper implementation of the difference between optional and required entities must be evaluated. Entities must also be flexible in the user's path, i.e. the user has to be able to give their values at another step than where they are originally defined in the interface. This is necessary, as defining every single entity at every possible place where they can be given, is close to impossible and also extremely inefficient. For example, it may be that the design has separated steps for giving the departure and arrival location, but a user should also be able to give both at once if he so wishes. The frameworks likely cover most of these cases, but if testing shows that the framework's capabilities are inadequate, manual software implementation is an alternative for obtaining such a feature.

Evaluation of the context constraints of the entities begins with establishing that the designed sample conversations follow the specified constraints. As two other phases, namely the fallback procedures and context requirements, divide the specification of input requirements with writing the sample conversations, it is possible that certain constraints are overlooked at this phase. Besides this fact, it is likely that those who write the dialogues did not specify the context constraints, hence the need for proper testing.

4.2.2 Confidence Levels

This research proposes a relatively simple yet potentially very effective method of testing if the specified confidence level rules work adequately: deliberately feeding bad input to the system. This way, one can test if the input is categorized as expected. However, because of the complex algorithms of bot frameworks, it is impossible to determine what exactly gives a specific confidence level. For example, if a tester gives wrong input by feeding the interface the sentence "I wan tu booc a flit to ork", as opposed to the actual meaning "I want to book a flight to New York", the artificial intelligence embedded in the frameworks may be able to correct this properly. The tester is not able to predict beforehand what confidence level the system will give this utterance. Depending on several circumstances such as what bot framework is used, the context of the user and the setting of the current dialogue, the confidence could be anywhere between 10% and 80% for this utterance.

The suggested way to circumvent this issue is by specifying beforehand which utterances should be accepted by the system and which should not. Thresholds have been specified during the design phase, but it may be difficult for designers to distinguish a difference with a 5% or 10% difference in confidence. For instance, the difference between 90 and 95% confidence can be hard to comprehend, but through trial and error of the settings, one can find out if he is satisfied with the settings. With the above example, one might argue that the interface should ask for confirmation from the user. If the confidence threshold is set at 0.5, and the system manages to get a 0.53 confidence that the utterance represents the request of booking a flight to New York, the designers could adjust the threshold to 0.55. However, it is imperative this be based on a larger set of utterances and not just on one request such as in the above example. It should only be increased if the average confidence of utterances that should not be accepted is above the threshold. That average confidence must also be compared with tests of utterances that *should* be accepted by the system. The confidence threshold ideally lies in the middle of those two averages.

4.3 Fallback Procedures

The fallback procedure tests consist of two parts: functionality and user satisfaction. Whereas functionality testing consists of verifying that the specified procedures are implemented properly in the interface, user satisfaction is confirmed by how users respond to specific fallbacks. The functionality of first fallback category, non-response, can easily be tested by simply not responding at specific steps in the conversation and awaiting the system's response to that. For user satisfaction, UX tests can integrate a section on fallback procedures. During a UX test, a user can either be specifically told not to respond or be given a task that would take him longer than what the timeout is set to.

Specifically asking the user not to respond may not give the natural response that one might hope for, which makes the second option more favorable. For example, the user could be asked to look up a credit card number in a fabricated document. By making the number uneasy to find, a user will likely not find it before the timeout happens, thus triggering the non-response fallback. Another way of testing this is by having the tester interact with the user during the conversation. Distraction may happen during real use as well, and the user's response to the fallback in this setting is maybe even more valuable than the aforementioned method of asking a user to look something up.

A misunderstanding can be tested by giving entity values that shall not be recognized by the system, e.g. by feeding the interface with a departure location unknown to the CUI, thus where no airfield exists according to the interface. The system must recognize the what entity the unknown value is referring to and catch this with a fallback procedure.

User requests to fallback are a relevant factor to evaluate as well, especially establishing that context constraints are functional. If components are properly separated from each other as described in section 3.4.3, then this will be a trivial task to test as functionality, but the user experience of the implementation can still not be as expected. Implementing fallback requests in the user experience test is therefore recommended at multiple phases of the conversation, at both moments where it will be accepted and at places where it will not be. Users can, for instance, be asked to change the departure location of a flight before as well as after initiating the payment procedure, where the system has to deny the latter as the payment should be canceled first.

4.4 Context requirements

Context requirements of the CUI are divided into three categories: CUI memory, location, and user data. Of these three, evaluating the memory may be the most challenging. Functional evaluation of memory involves frame tracking, frame switching and data recall in new conversations. Proper frame tracking can best be evaluated by a tool that visualizes a conversation's frame, as is done in the sample of Figure 4. If such a tool is unavailable, manual evaluation of the database is necessary, costing a lot more time and effort. Once a conversation has been visualized, evaluators can walk through the conversation and verify that frames are properly utilized, and only new ones are created when no old frame has the same information. The latter already relates to frame switching as well: without functional frame switching, new frames would have to be created, even though old frames consist the same information. Therefore, evaluation of frame tracking and frame switching is best done together.

Recall of previous conversations requires evaluators to design multiple conversations beforehand. First, a conversation must be designed that will be later recalled, which can be both a conversation with a successful finish of the CUI goal, or an unfinished one. Ideally, both of these are tested separately. The second conversation that needs to be designed is the one in which the previous conversation must be recalled.

Evaluation of the location specifications can be done by emulating different settings of the chatbot. Functionality can be evaluated by changing the location variable rather easily, for example by setting a mock location on an Android phone. Through user experience tests, users can be asked to use the product in the actual location that requires testing. Besides that, a room can also emulate certain settings. For example, a living room can be rebuilt in a test environment. This way, the environment closely matches the actual environment in which the product would be used, but there are increased possibilities of adapting the environment to ideal testing conditions.

Testing whether user data specifications and privacy restrictions are complied with in the design can be achieved by simply scanning for the restricted entities in the conversation and confirming that they are not given back to the user in ways that were explicitly prohibited. However, evaluation of such a function may add even more value by finding out what the user awareness of data sharing capabilities is. For example, a functionality of the system may be that a user can ask the CUI about the flight wish list of his friends and family. Such a function can greatly enhance the ease of use of the interface, but only if users know about the existence of it as well.

4.5 Sample Conversations

Copywriting is a task that requires specific skills that are not always familiar to regular programmers, which implies the same goes for evaluating the designed conversations. Testing input can be done through Wizard of Oz testing, as explained in section 3.3 as well. This methodology is originally meant for testing and thus will add even more value in this phase. With the WOz test, a person takes over the role of the interface, called the *wizard*, and a user talks to the wizard as if it were the actual system (Dahlbäck et al., 1993). The wizard should have access to all the data that the normal CUI can access. The wizard will use search queries to find information about the user's request and respond based on the results.

Using this method, one can evaluate if input requirements, and with that the data that the CUI has access to, cover everything for what a user might use the interface. More importantly, though, one can verify if the sample inputs are designed well. All the inputs the user gives to the wizard throughout the WOz test are saved and can later be used for testing the actual CUI. If the WOz methodology was also used for designing the happy paths, inputs given at that phase could also be used as test samples in this evaluation phase. If a wizard can give the user a satisfactory result, it does not necessarily implicate that the CUI will do so as well, and that is why these inputs need to be tested with the interface. If the wizard is able to help the user but the CUI cannot deliver the same results, designers must either adjust or add inputs into the bot framework. Same results from both, on the other hand, indicate well-designed sample inputs.

Reviewing responses, including the integration of the external database, requires several separate evaluations. Firstly, functional testing of database changes is needed. After a successful dialogue, values in the external database should be added, edited or removed. For example, if someone successfully booked a flight, a data entry for this should exist. For this, test scripts should be written beforehand, where the wanted enterprise DB state is specified and later compared to the actual outcome after the dialogue.

Secondly, responses must adhere to the style guide. Proper style guide implementation results in a coherent conversation, even if different people wrote different parts. The responses must be streamlined with the style guide, which means this is to be evaluated as a separate subject as well. Evaluation of the utterances that should not occur during the dialogue will take minimal effort: one can simply check if those utterances appear anywhere, and change them accordingly. The application of utterances that the CUI would likely say, however, may take more effort. Evaluation could take shape in the form of checking chunks of a conversation at once and verifying that the good utterances appear in them. If not, one can try to edit certain responses to include it in order to make the responses coherent.

Thirdly, evaluation must be performed on the naturalness of the responses. A user should not necessarily think that he is talking to a human, but should not feel like it is a static conversation either. He must be able to direct the dialogue and get fitting responses to that to the extent that the user is satisfied with the conversation flow. User experience tests will verify whether this is the case for either specific goals of the CUI or for the system as a whole.

Another important factor of the conversation is the end, where the CUI goal has either been reached or canceled. GUI software usually notifies the user when a goal is finished, for example by displaying the message "your ticket is successfully ordered, you may now return to the home screen". However, a message like that would not make a conversational interface sound natural. Confirming to a user that he has reached his goal should still sound natural, which may make the user less aware of the fact that he reached the end. Therefore, evaluators should focus on this during user experience tests.

4.6 Automated testing

As already addressed before in this section, several parts of the evaluation process can be automated. Something as a user experience test on how natural the conversation feels to a user cannot be automated, but whether or not certain components work as they were functionally designed can be.

One methodology of automating tests of functionality is through Specification by Example (Adzic, 2011). Specification by Example is a way of writing and validating specifications in which automation is one of the steps in the process. With this methodology, specifications are created, illustrated with examples, and then refined. The next step in the process proposed by Adzic is "automating validation without changing specifications". With this, he implies that an automated validation tool must be designed, which completely adheres to the previously set specifications.

This methodology shows many relations to the CDF and, therefore, shows great potential for implementation. In the conversational setting, Specification by Example means that when a certain input is given to the system, the specified output must be the result. For example, when the input is I want to book a flight to New York, the output of the system must be the response that is specified for thus utterance, e.g. “Cool! When would you like to leave?” Most of the specified inputs and responses will be implemented in the framework as well, but an efficient way to test good functionality can already be achieved by setting different entities in the framework than those in the specifications. When an automated validation tool is designed, specifications must minimally cover the happy path, but if possible also include validation of fallback procedures and context requirements.

4.7 Active Learning

As explained in section 2.3.1, bot frameworks use active learning to adapt the interface to the user, adding more inputs and understanding the user’s natural language better over time. This should greatly improve functionality and user experience and at the same time decrease the amount of manual adjustments needed.

However, a possible risk of this that should not be overseen is that it may break features as well. For instance, if one of the CUI’s goals is used significantly more often than others, many new user inputs may be added to that intent. After a while, the risk is that these inputs start to overlap with the designed inputs of the lesser-used intents. This can cause the other CUI goals to lose their functionality, which is of course not the intention of active learning. Therefore, frequently validating inputs is necessary when active learning is enabled. In fact, frequent validation is also a step in the Specification by Example methodology. If a tool was created for automated validation, this tool should be run not just once after the design is completed, but also frequently once the product is released to customers.

5 Agile Application

As is the case with Info Support where the research is done, many software companies apply Agile methodologies to their software development processes (Hamed & Abushama, 2013). Even though much of the Conversation Design Framework as created during this research lies outside of the scope of a traditional software developer, conversational interfaces are still likely to be designed mostly by software development firms. They may collaborate with firms specialized in UX writing or employ people for this role, but either way, building the CUI is likely desired to be done in an Agile way. Agile methodologies use short development cycles that allow for constant feedback (Highsmith & Cockburn, 2001). Agile application is often demanded by modern software development firms and, therefore, application of the Agile principle is researched in relation to the CDF. According to Adzic (2011), advantages of Agile development can be summarized in four aspects:

- implementing changes more efficiently;
- higher product quality;
- less rework; and
- better alignment of the activities of different roles on a project.

There are many different methods of obtaining agility within an enterprise. Within Info Support, either iteration-based Scrum processes are implemented, or a flow-based Kanban process is maintained. Scrum works with *sprints*, often one to four weeks, after which a small predefined portion of the final product is delivered[36]. Kanban, on the other hand, works limiting the amount of similar tasks that can be performed at once in order to stimulate the development flow[37]. For example, only a certain amount of issues may be worked on at once, i.e. Work in Progress or WIP. If this limit is reached, developers not assigned to those issues must pick up activities that are not in the WIP phase, such as software testing.

The term *minimal viable product* (MVP) is often used in this context as well, especially with Scrum development. This refers to a product that has the minimal features necessary to be usable. With agile development processes, the aim is often to create a MVP with every sprint that can then be tested for feedback. This is also possible with a large part of the design process when utilizing the CDF. The first step of the framework, specifying target functions, will produce a list of Goals for the CUI. Each of these goals is to be designed separately with the framework and can, therefore, be seen as separate minimal viable products. Depending on the size of the goal, the CUI design team can assign one or more CUI goals per development sprint. The result of this should then be one portion of the final software that can be tested and put into production. Short feedback loops and quick validation of correct implementation are possible this way, which are characteristics of a proper agile process.

If one goal of the CUI is too extensive to design in one sprint completely, an MVP can still be produced by not implementing all the possible paths that a user might take during the process. For instance, only writing sample conversations to accommodate the happy path can already act as a

[36]`https://www.scrumalliance.org/why-Scrum/agile-atlas`, accessed June 29, 2017
[37]`https://leankit.com/learn/kanban/what-is-kanban/`, accessed July 6, 2017

minimal viable product. However, there should always be at least one fallback procedure at every step of the conversation, namely that of completely canceling the process. Otherwise, one would have to go through the entire process successfully before being able to leave the CUI environment. For instance, when a flight ticket ordering goal is designed, a user might effectively be forced to buy the ticket before leaving the environment if no fallback procedure is implemented. As one might imagine, this will result in very negative experiences of the system and should, therefore, be avoided completely by minimally allowing the user to discontinue the process. After the MVP that contains the happy path is validated, a new sprint can be initiated in which the designs of the other possible paths, feedback procedures and context requirements can be implemented.

Before beginning with sprints for CUI goals, the bot's personality must be designed. This can be seen as a separate sprint in the design process. The MVP of that sprint will be the documentation on the personality and, more importantly, the style guide. For agility, the style guide can be tested in Wizard of Oz tests as a means of gathering quick feedback at this stage. The wizard, i.e. the person pretending to be the system, can adhere to the style guide, and this way it can be validated whether or not the style guide is sound.

In the CDF, WOz is recommended both during the specification of the happy path and with the verification of sample inputs. This method is good for the agility of the framework, as it allows for quick validation of the design. WOz tests and the validation of the system afterward can be performed within a day, which gives the development team quick feedback on their implementation of the goal. Specification by Example also delivers good Agile possibilities (Adzic, 2011). From this, combined with the fact that MVP development is possible, it can be concluded that the CDF has a high potential for Agile application.

6 Discussion

This thesis proposes the Conversation Design Framework (CDF) as the first step for scientific research on conversational user interfaces. The CDF is composed of seven concrete steps that should result in an effective conversation design when followed properly. After the CUI goals are set, an important step of the CDF is creating the personality for the bot. This personality gets translated into a style guide for the chatbot, which helps provide the experience that fits well with the target user and enables simultaneous dialogue writing by different people whilst it coherent. Afterward, happy path conversations are specified, along with input requirements as an outcome. The fourth step is specifying the fallback procedures, where the distinction ought to be made between non-response, misunderstanding and user request. The fifth step in the framework is specifying the context requirements, which is an important step to keep the dialogue conversational. Setting the requirements for the CUI's memory is the most important part of this phase and may also take extensive software implementation to fit well with one's specific needs. The step that follows the context requirements is that of writing the sample conversations, which is where all the previous steps come together, and full conversations are modeled. The happy path should be modeled first, followed by every other path a user might take in the conversation. A multitude of user inputs and responses is to be modeled for all those paths. Last but not least, the CUI needs to be trained as a final step after the sixth phase is finished, which will make the CUI ready for release.

The separate steps of the CDF can be tested, as proposed in section 4. Through Wizard of Oz (WOz) testing, a significant portion of the specifications can be verified for completeness. With the WOz test, the wizard takes on the role of the system and guides a user through the conversation in similar manners as the CUI would. Besides that, much of the testing can potentially be automated. However, no proper evaluation methods are found for testing the personality.

Last but not least, the third aspect of this research was Agile application of the framework. Short development cycles with quick feedback loops are an important factor of this. Agile application is found to be very promising while using the CDF as one CUI goal can designed at a time and quick feedback is possible through WOz and automated testing.

Currently, little to no scientific research exists for conversational user interfaces. Therefore, it is found that this thesis offers a significant scientific contribution, being a pioneer in this field. The CDF is the first step in CUI research with many still to come. This thesis can be taken as a good starting point for future researchers. Conversational user interfaces are still in their infancy with many developments yet to come, which is why additional research is crucial for CUIs to have the proper impact in one's life they may potentially have in the future.

One of the strengths of this research is that, although there was not an opportunity for a case study of the framework, the findings have constantly been presented to professionals that have extensive experience with chatbots and AI. This resulted in a sound model that is believed to be very applicable in business environments. Besides that, even though no scientific research on CUI design was available, another strength of the framework is that a good portion of it is based on and linked to scientific research into software and conversation design.

Last but not least, as mentioned earlier, many aspects of the framework allow for automated testing. This can be seen as another strength of the research as it means that, once automation tools are developed, testing the specifications can be done very efficiently. Agile application is also a demand for many software development firms. As this model has high promises for Agile application, the framework offers many opportunities for direct use, without special adjustments required to adhere specific Agile design principles.

Many researches have limitations attached to them, and this one is no different in that aspect. First of all, the framework is not validated with a real case study. This sometimes gave difficulties of relating the research to situations that will concretely occur in the design process of chatbots. This issue was greatly overcome by using one big example throughout the entire process, namely that of the CUI for an airline company. This helped make the research tangible and allowed for providing good examples of how certain steps can be performed in the design of an actual system. Future research is definitely needed, however, where a real CUI gets designed using the CDF. This can be either a proof of concept, e.g. a CUI that exists purely inside a bot framework. However, it can also already be applied in a real situation. For instance, in the case of Info Support, a chatbot can be designed using the CDF for the robot Pepper, which is illustrated in section 2.4. This real world implementation will either verify the usability of the complete CDF or researchers may conclude that adjustments need to be made to the framework to make it more effective or efficient.

Next to that, no concrete test method was found for personality testing. More research is thus required to create a methodology of personality testing for conversational user interfaces. This research will likely take have more psychological aspects to it than that it will have software design aspects. Therefore, it is suggested to have this this done by people specialized in the field of psychology rather than software development.

Last but not least, a limitation of this research is that no scientific research exists on bot frameworks. As no case study was performed during this research, there was also relatively little knowledge of the different bot frameworks and the differences between them. This was partially overcome by consulting the documentation resources of particular frameworks. However, more research into bot frameworks will be extremely valuable for the field of CUIs. A model can be designed in future research that tests and scores separate bot frameworks, based on an individual's or firm's needs and desires. This model can, for instance, take into account the difference in features and natural language understanding capabilities. Besides that, this research can dig deeper into the possibilities and added value that open-source bot platforms have, such as Mycroft that was elaborated in section 2.2. Also, research can look into the applicability of the CDF in specific bot frameworks. One framework may better implementation of specifications set using the CDF, while another could need more manual software development to comply with the design.

7 Conclusion

This research aimed to the best way of specifying and testing conversational user interfaces in an Agile environment, and the Conversational Design Framework created in this research is deemed to be a valid solution to this, as is elaborated in section 3. The first part of the research question regards specification of CUIs. The CDF provides a sound guideline for specifying dialogues for conversational user interfaces. If the seven phases of the framework are followed properly, a well-defined conversation should be the result which can then be implemented into the desired system.

The second part of this research concerns the testing of those specifications. Except of personality testing, this thesis presents testing methods for every aspect of the CDF, which can be found in section 4. Suggestions are also given for automating a large portion of the required testing. It can thus be concluded that this part of the research question is answered to a satisfactory level.

Last but not least, the final part of the research considers how specification and testing can be done in an Agile environment. Because most of the CDF is executed per separate CUI goal, the framework shows high potential for agility. Because of this, combined with the opportunities for automated testing, short feedback loops and quick validation are made possible. From this, one can infer that a sound answer is provided for the last part of the research question.

To conclude, the Conversation Design Framework can be used for specifying conversational user interfaces. Testing methods are presented with many possibilities of automation, and the framework allows for proper Agile application. Strengths of this research are that a sound framework is presented even though little to no research on CUIs was available, that many aspects are linked to science on software and conversation design and finally that it shows high potential for automated testing. The limitations of this study are that no case study was conducted to verify the CDF, that no method is found for personality testing and that there was little knowledge on the differences between bot frameworks and their features and limitations. Future research suggestions are given for each of these three limitations.

CUIs are deemed indispensable in one's life in the near future, which shows the significant impact this research can have as a pioneer in the field. CUIs are only in their infancy, which is why additional research is crucial. Conversational user interfaces have only scratched the surface of their potential, and one can only imagine the impact they will have in the future.

8 Bibliography

Adzic, G. (2011). *Specification by example: how successful teams deliver the right software.* Manning Publications Co.

Asri, L. E., Schulz, H., Sharma, S., Zumer, J., Harris, J., Fine, E., … Suleman, K. (2017). Frames: A corpus for adding memory to goal-oriented dialogue systems. *arXiv preprint arXiv:1704.00057.*

Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering* (pp. 712–721).

Bellegarda, J. R. (2014). Spoken language understanding for natural interaction: The siri experience. In *Natural interaction with robots, knowbots and smartphones* (pp. 3–14). Springer.

Cai, X., Lyu, M. R., Wong, K.-F., & Ko, R. (2000). Component-based software engineering: technologies, development frameworks, and quality assurance schemes. In *Software engineering conference, 2000. apsec 2000. proceedings. seventh asia-pacific* (pp. 372–379).

Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, *37*(1), 51–89.

Dahlbäck, N., Jönsson, A., & Ahrenberg, L. (1993). Wizard of oz studies—why and how. *Knowledge-based systems*, *6*(4), 258–266.

De Mori, R., Bechet, F., Hakkani-Tur, D., McTear, M., Riccardi, G., & Tur, G. (2008). Spoken language understanding. *IEEE Signal Processing Magazine*, *25*(3).

Epley, N., Waytz, A., & Cacioppo, J. T. (2007). On seeing human: a three-factor theory of anthropomorphism. *Psychological review*, *114*(4), 864.

Farinazzo, V., Salvador, M., Kawamoto, A. L. S., & de Oliveira Neto, J. S. (2010). An empirical approach for the evaluation of voice user interfaces. In *User interfaces.* InTech.

Hamed, A. M. M., & Abushama, H. (2013). Popular agile approaches in software development: Review and analysis. In *Computing, electrical and electronics engineering (icceee), 2013 international conference on* (pp. 160–166).

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*, *34*(9), 120–127.

Ho, C.-H., & Swan, K. (2007). Evaluating online conversation in an asynchronous learning environment: An application of grice's cooperative principle. *The Internet and Higher Education*, *10*(1), 3–14.

Lotterbach, S., & Peissner, M. (2005). Voice user interfaces in industrial environments. In *Gi jahrestagung (2)* (pp. 592–596).

Lucassen, G., Dalpiaz, F., Van der Werf, J. M. E., & Brinkkemper, S. (2016). The use and effectiveness of user stories in practice. In *International working conference on requirements engineering: Foundation for software quality* (pp. 205–222).

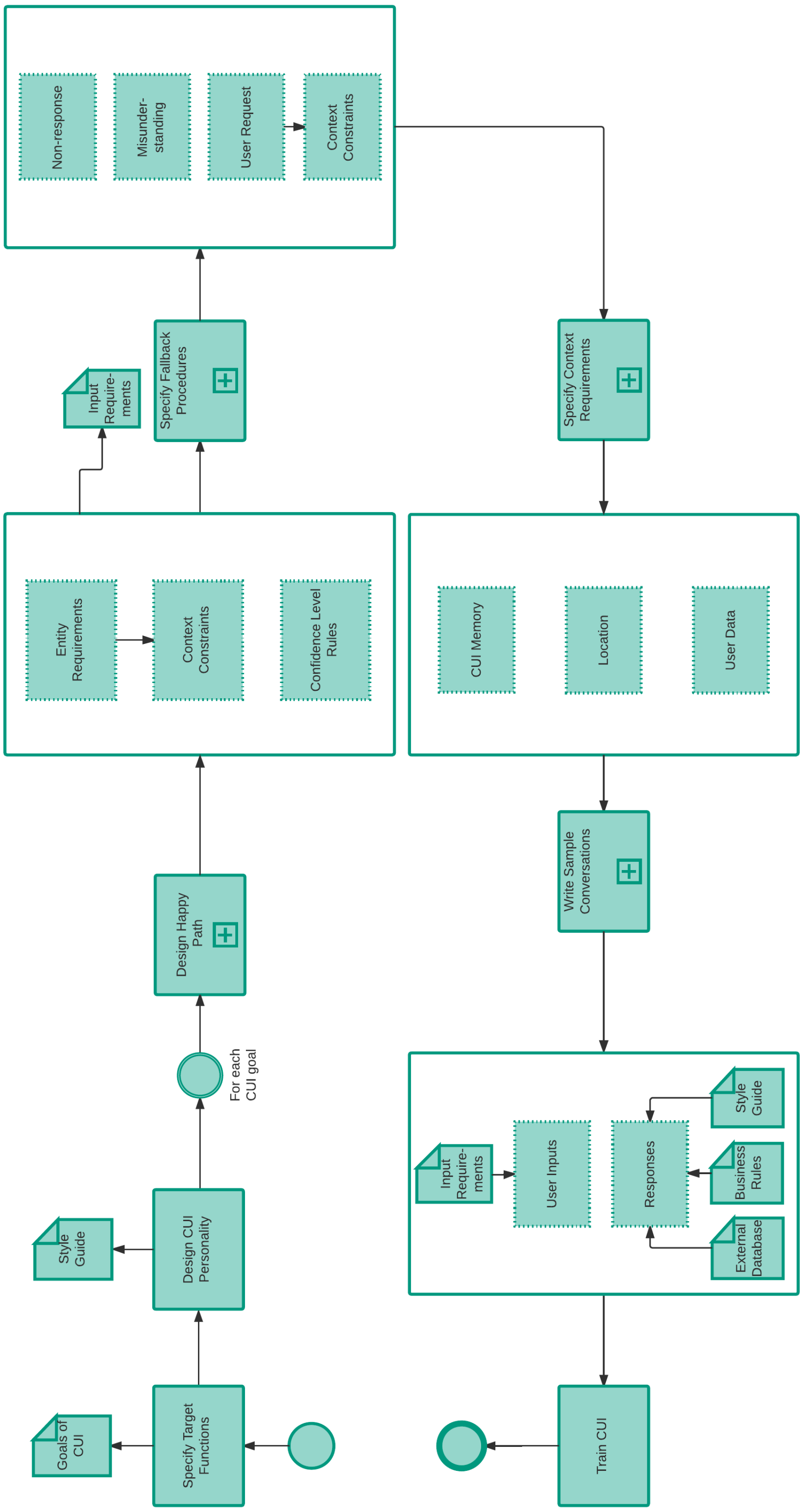
McTear, M. (2002). Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys (CSUR)*, *34*(1), 90–169.

McTear, M., Callejas, Z., & Griol, D. (2016). *The conversational interface.* Springer.

Myers, I. B. (1962). *The myers-briggs type indicator.* Consulting Psychologists Press Palo Alto, CA.

Salvador, V. F. M., de Oliveira Neto, J. S., & Kawamoto, A. S. (2008). Requirement engineering contributions to voice user interface. In *Advances in computer-human interaction, 2008 first international conference on* (pp. 309–314).

A Conversation Design Framework


Specify Target Functions
Goals of CUI
Design CUI Personality
Style Guide
For each CUI goal
Design Happy Path
Entity Requirements
Context Constraints
Confidence Level Rules
Input Require-ments
Specify Fallback Procedures
Non-response
Misunder-standing
User Request
Context Constraints
Specify Context Requirements
CUI Memory
Location
User Data
Write Sample Conversations
Input Require-ments
User Inputs
Responses
Style Guide
Business Rules
External Database
Train CUI