Conversational UX Design

A Practitioner's Guide to the Natural Conversation Framework

Robert J. Moore
Raphael Arar

ASSOCIATION FOR COMPUTING MACHINERY

Conversational UX Design

ACM Books

Editor in Chief

M. Tamer Özsu, *University of Waterloo*

ACM Books is a series of high-quality books for the computer science community, published by ACM and many in collaboration with Morgan & Claypool Publishers. ACM Books publications are widely distributed in both print and digital formats through booksellers and to libraries (and library consortia) and individual ACM members via the ACM Digital Library platform.

Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework

Robert J. Moore, *IBM Research–Almaden*
Raphael Arar, *IBM Research–Almaden*
2019

Heterogeneous Computing: Hardware and Software Perspectives

Mohamed Zahran, *New York University*
2019

Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker

Editor: Michael L. Brodie
2018

The Handbook of Multimodal-Multisensor Interfaces, Volume 2: Signal Processing, Architectures, and Detection of Emotion and Cognition

Editors: Sharon Oviatt, *Monash University*
Björn Schuller, *University of Augsburg and Imperial College London*
Philip R. Cohen, *Monash University*
Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Krüger, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2018

Declarative Logic Programming: Theory, Systems, and Applications

Editors: Michael Kifer, *Stony Brook University*
Yanhong Annie Liu, *Stony Brook University*
2018

The Sparse Fourier Transform: Theory and Practice

Haitham Hassanieh, *University of Illinois at Urbana-Champaign*
2018

The Continuing Arms Race: Code-Reuse Attacks and Defenses
Editors: Per Larsen, *Immunant, Inc.*
Ahmad-Reza Sadeghi, *Technische Universität Darmstadt*
2018

Frontiers of Multimedia Research
Editor: Shih-Fu Chang, *Columbia University*
2018

Shared-Memory Parallelism Can Be Simple, Fast, and Scalable
Julian Shun, *University of California, Berkeley*
2017

Computational Prediction of Protein Complexes from Protein Interaction Networks
Sriganesh Srihari, *The University of Queensland Institute for Molecular Bioscience*
Chern Han Yong, *Duke-National University of Singapore Medical School*
Limsoon Wong, *National University of Singapore*
2017

The Handbook of Multimodal-Multisensor Interfaces, Volume 1: Foundations, User Modeling, and Common Modality Combinations
Editors: Sharon Oviatt, *Incaa Designs*
Björn Schuller, *University of Passau and Imperial College London*
Philip R. Cohen, *Voicebox Technologies*
Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Krüger, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2017

Communities of Computing: Computer Science and Society in the ACM
Thomas J. Misa, Editor, *University of Minnesota*
2017

Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining
ChengXiang Zhai, *University of Illinois at Urbana–Champaign*
Sean Massung, *University of Illinois at Urbana–Champaign*
2016

An Architecture for Fast and General Data Processing on Large Clusters
Matei Zaharia, *Stanford University*
2016

Reactive Internet Programming: State Chart XML in Action
Franck Barbier, *University of Pau, France*
2016

Verified Functional Programming in Agda
Aaron Stump, *The University of Iowa*
2016

The VR Book: Human-Centered Design for Virtual Reality
Jason Jerald, *NextGen Interactions*
2016

Ada's Legacy: Cultures of Computing from the Victorian to the Digital Age
Robin Hammerman, *Stevens Institute of Technology*
Andrew L. Russell, *Stevens Institute of Technology*
2016

Edmund Berkeley and the Social Responsibility of Computer Professionals
Bernadette Longo, *New Jersey Institute of Technology*
2015

Candidate Multilinear Maps
Sanjam Garg, *University of California, Berkeley*
2015

Smarter Than Their Machines: Oral Histories of Pioneers in Interactive Computing
John Cullinane, *Northeastern University; Mossavar-Rahmani Center for Business and Government, John F. Kennedy School of Government, Harvard University*
2015

A Framework for Scientific Discovery through Video Games
Seth Cooper, *University of Washington*
2014

Trust Extension as a Mechanism for Secure Code Execution on Commodity Computers
Bryan Jeffrey Parno, *Microsoft Research*
2014

Embracing Interference in Wireless Systems
Shyamnath Gollakota, *University of Washington*
2014

Conversational UX Design

A Practitioner's Guide to the Natural Conversation Framework

Robert J. Moore
IBM Research–Almaden

Raphael Arar
IBM Research–Almaden

ACM Books #27

Copyright © 2019 by Association for Computing Machinery

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews—without the prior permission of the publisher.

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which the Association for Computing Machinery is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework
Robert J. Moore
Raphael Arar

books.acm.org
http://books.acm.org

ISBN: 978-1-4503-6301-3 hardcover
ISBN: 978-1-4503-6302-0 paperback
ISBN: 978-1-4503-6303-7 ePub
ISBN: 978-1-4503-6304-4 eBook

Series ISSN: 2374-6769 print 2374-6777 electronic

DOIs:

10.1145/3304087	Book	10.1145/3304087.3304095	Chapter 7
10.1145/3304087.3304088	Preface	10.1145/3304087.3304096	Chapter 8
10.1145/3304087.3304089	Chapter 1	10.1145/3304087.3304097	Chapter 9
10.1145/3304087.3304090	Chapter 2	10.1145/3304087.3304098	Appendix A
10.1145/3304087.3304091	Chapter 3	10.1145/3304087.3304099	Appendix B
10.1145/3304087.3304092	Chapter 4	10.1145/3304087.3304100	Appendix C
10.1145/3304087.3304093	Chapter 5	10.1145/3304087.3304101	Appendix D
10.1145/3304087.3304094	Chapter 6	10.1145/3304087.3304102	References/Index/Bios

A publication in the ACM Books series, #27
Editor in Chief: M. Tamer Özsu, *University of Waterloo*
Area Editor: Michel Beaudouin-Lafon, *Université Paris-Sud and Institut Universitaire de France*

This book was typeset in Arnhem Pro 10/14 and Flama using ZzTeX.
Cover art: Raphael Arar.
Cover image: The Euphonia, (The Talking Machine); Source: *The London Journal: Weekly Record of Literature, Science and Art*. NO. 1340, VOL. LII, London: 1870, November 1, 1870, pg. 245.

First Edition

10 9 8 7 6 5 4 3 2 1

Contents

Preface xiii

Chapter 1 Introduction 1

1.1 Natural Language Interfaces 3
1.2 Conversational UX Design 5
1.3 Conversation First 7
1.4 Mutual Understanding 17
1.5 About This Book 23

Chapter 2 Conversation Analysis 25

2.1 What Is a Conversation? 26
2.2 Topic vs. Structure 27
2.3 Anatomy of a Conversation 29
2.4 Conversation Types 39
2.5 Conclusion 43

Chapter 3 Conversation Authoring 45

3.1 A Simple Finite-State Machine 47
3.2 Conditions (Intents, Entities, Context) 49
3.3 Responses 56
3.4 Branching 59
3.5 Conclusion 62

Chapter 4 Natural Conversation Framework 63

4.1 Interaction Model 65
4.2 Content Format 69
4.3 Conversational Activity Patterns 76
4.4 Conversation Navigation 79
4.5 Conclusion 85

Chapter 5 Conversational Activity UX Patterns 87
5.1 Pattern Language for Conversational UX 88
5.2 A1 Inquiry (User) 90
5.3 A2 Open Request 93
5.4 A3 Extended Telling 103
5.5 A4 Quiz 106
5.6 A5 Inquiry (Agent) 109
5.7 Combining Patterns 114
5.8 Conclusion 119

Chapter 6 Sequence Management UX Patterns 121
6.1 B1 Repair (Agent) 122
6.2 B2 Repair (User) 128
6.3 B3 Extended Repair 138
6.4 B4 Sequence Closers 142
6.5 B5 Sequence Abort 145
6.6 Conclusion 147

Chapter 7 Conversation Management UX Patterns 149
7.1 C1 Opening (Agent) 149
7.2 C2 Opening (User) 155
7.3 C3 Capabilities 161
7.4 C4 Closing 165
7.5 C5 Disengaging 174
7.6 Conclusion 180

Chapter 8 Conversational UX Design Process 183
8.1 Design Thinking 183
8.2 Stage 1: Empathize 187
8.3 Stage 2: Define 190
8.4 Stage 3: Ideate 192
8.5 Stage 4: Prototype 203
8.6 Stage 5: Test 208
8.7 Conclusion 209

Chapter 9 Conclusion 211
9.1 Conversation Metrics 212
9.2 Final Thoughts 217

Appendix A Conversation Analysis Transcription Conventions 219

Appendix B Dialog Design Pseudocode 221

Appendix C NCF Pattern Language Summary 223
NCF Pattern Types 223
NCF Patterns and Examples 226

Appendix D Sample Conversation with Alma 265

References 275

Index 283

Author Biographies 297

Preface

Ever since I started working in Silicon Valley research labs, I have dreamed of designing machines that can talk. As a sociologist trained in ethnomethodology and Conversation Analysis (CA), I have always been eager to apply my knowledge of the "machinery" of human conversation to the design of conversational machines. While at Xerox's Palo Alto Research Center (PARC), beginning in 1999, I asked my colleagues specializing in natural language processing why we could not build a conversational system. They told me it would require a dialog platform, which we did not have. Years later in 2012, I joined IBM's Almaden Research Center (ARC) in San Jose. Having watched Watson beat human champions at Jeopardy only the year before, my first thought was, how can I teach Watson to hold a conversation? By 2013 I was asked to advise a group that was doing just that. I shared my formal knowledge of the structure human conversation to suggest ways that their system could be more "conversational."

Then in 2015, IBM developed its own dialog platform, the Watson Dialog service. The platform provided an authoring tool for creating natural language interactions that anyone could use, even a sociologist! Immediately I began learning the simplified, GUI-based programming tool and creating conversational user experiences. I learned to chain dialog nodes together in various ways, set conditions, train intents, write responses and test to see if my conversational sequences worked. Soon I was authoring conversation spaces. I collaborated with a team, as the conversational UX designer, that was developing an application called *What's in Theaters*, a conversational agent for accessing movie information. *What's in Theaters* was not a product, but simply a prototype application demonstrating best practices for how the Watson Dialog service could be used. Through working on this application, and many more that followed it, I learned how to perform the role of conversational UX designer on software development teams.

From the start, my interest in creating conversational user experiences has been to develop a systematic design framework that is based on the science of

Conversation Analysis and related fields. As a conversation analyst, I am fascinated with the mechanics of mundane conversational activities; however, I expect the average UX designer is not. My aim was to develop a set of reusable patterns for conversational activities and their local management, and to share them with UX designers so they could focus more on the *content* of their agent and less on the basic conversation mechanics. In consulting across IBM and outside of it, I learned that development teams creating conversational agents lacked a common vocabulary for talking about the different parts of a conversation. I realized that even before providing the set of interaction patterns, development teams could benefit from the vocabulary of CA, with its turns, sequences, activities, conversations and more. They needed a systematic way to refer to the various parts of the thing they were trying to build.

In 2016 I began working with my current team, re*THINK Enterprise, in IBM Research, to develop conversational agents for the travel industry. As part of this team, I met and began working with my co-author, Raphael, a UX designer, researcher and artist. Together we tackled questions of how to integrate conversational UX with visual UX on desktop and mobile platforms. Through these efforts, we began to talk about how we could package formal knowledge from Conversation Analysis for UX designers. The CA literature is somewhat dense and technical. Simply directing designers to it would not be enough. So we looked for similarities across the disciplines, for example, in the concepts of "user-centered design" [Norman 1988] and "recipient design" [Sacks et al. 1974], to help translate ideas between analysts and designers. In addition, we experimented with adapting the design thinking process to the development of conversational agents, testing them on real projects. Together we sketched out a novel methodology for *conversational* UX design.

In 2017 we organized a workshop at the CHI conference on human-computer interaction in Denver on the topic of Conversational UX Design. The workshop attracted a large group of researchers, both from academia and from industry, all working in the area of chatbots and voice assistants. Each of the participants displayed a real passion for conversational interfaces. In addition to participants, our workshop attracted the attention of publishers. As a result, we have produced two books on the topic, of which this is one.

Motivation for this Book

The motivation for this book is to address an emerging challenge increasingly faced by user experience (UX) designers: how to model *natural conversation*. Today's chatbot and voice assistant platforms enable you to create rich natural-language interactions, but they leave it up to you to figure out how to model the structure of those interactions. In order to determine how the system will respond to a particular kind of user utterance, you must program *dialog*. Modeling simple question-answer or command-control interactions may be easy enough, but modeling natural conversational activities is more challenging. *Answering* has a different interactional structure from *quizzing*. *Debating* has a different structure from *storytelling*. *Troubleshooting* has a different structure from *tutoring*. And in any of these activities and many others, participants may need to elicit more information or repair prior utterances at any time in order to achieve mutual understanding.

This book provides a conceptual framework for designing the user experience with conversational user interfaces. It provides both introductory theory from conversation science and design assets from our experience building such systems over the past four years. The theory is adopted primarily from Conversation Analysis, a subfield of sociology, that has been around for over 50 years. The design assets consist primarily of a "pattern language" [Alexander 1977] applied to user interaction design [Erickson 2000] consisting of 100 UX patterns for conversational agents. The theory explains the structures of human conversation and the systematics behind the pattern language. In addition, the framework provides an interaction model, content format and navigation method for conversational interfaces.

We have selected these patterns for their genericness and reusability. We have mined the empirical literature of Conversation Analysis (CA) and abstracted from our own experience. The patterns are verbal interaction patterns that are independent of any particular content, such as travel, entertainment, healthcare, commerce, etc. The CA literature identifies these formal, generic patterns that are adapted by people to many kinds of conversational activities in all kinds of social settings. These patterns are used by speakers to do interactional work: to achieve the purpose of the interaction, to manage its course, and to recover from routine troubles. Our pattern language offers only a subset of the many patterns specified by Conversation Analysis. More patterns exist and will be added over time. In addition, the patterns we selected are simplified versions of the natural human patterns. They do not support all the rich variations that people exhibit in natural human conversation. Finally, our pattern language provides an initial set of actions against which the *conversational competence* of any automated agent can be measured.

The patterns are practical and combinable, like building blocks. For example, the *open request* pattern can be combined with the *extended telling* pattern to create a *troubleshooting* conversation. Through the first pattern, the user can present a problem and the agent can ask diagnostic questions to arrive at a solution, and through the second pattern, the agent can deliver a set of instructions, the pace of which the user can control. On the other hand, the *extended telling* pattern, *user inquiry* pattern and *quiz* pattern can be combined to create a *tutoring* conversation. The first pattern is used to present educational material, such as an explanation of "friction" or the themes in *Pride and Prejudice*, during which the user can solicit definitions and clarifications. The second pattern enables the user to ask questions about the material presented. And the third pattern enables the agent to quiz the student on his or her recall and understanding of the material. In fact, the pattern language can be used to design a wide range of conversations across many, many use cases.

The patterns are ordered in terms of their function. First, the patterns labeled "conversational activities" are those through which the participants achieve the primary business, or purpose, of a conversation: requesting a ride, getting a restaurant recommendation, answering questions about health insurance, solving a computer problem, teaching concepts in physics, and much more. Second, the patterns labeled "sequence-level management" are those through which the participants manage particular sequences within the conversation: requesting a repeat of what was just said, requesting a definition of an unknown term, acknowledging the fulfillment of a request with "thank you," aborting a set of instructions that is too long, and more. These patterns enable the participants to manage the social activities within a conversation. And third, the patterns labeled "conversation-level management" are those through which the participants manage the initiation and cessation of the interaction itself: starting a conversation with "hello," "how are you?," and/or "how can I help you?," describing what the customer service agent can do, getting in a last topic before the conversation ends, disengaging upon being insulted, asking to be transferred to a "real person" and more. These patterns enable the participants to manage the state of engagement.

For convenience and clarity, each pattern has the same format. First, there is an abstract model of the pattern in the form of a transcript with generic social actions (in capitals). Second, the pattern is illustrated with a particular example from some particular use case. Third, each pattern is named and numbered for easy reference. Subpatterns are numbered by adding decimal places to the numbering of the main pattern. For example, "A4.2 Quiz Repairs" is a subpattern of "A4.0 Quiz." The patterns are then organized into the three functional classes: conversational activities, sequence-level management, and conversation-level management.

We are publishing our approach in an effort to foster an open community of UX designers and others who are passionate about conversational UX design and who wish to leverage the conversation sciences, such as Conversation Analysis. We hope that the pattern language, which forms the core of our Natural Conversation

Framework, will be adopted and expanded by such a community. To foster collaboration, we encourage conversational UX designers everywhere to create transcripts representing how their agents interact and to share these designs with each other. Through such sharing, conversational UX design, and dialog design in particular, can evolve into a more mature discipline.

Acknowledgments

This book would not have been possible without the support of our IBM management, Guang-Jie Ren and Sandeep Gopisetty. They believed in our vision from the start and provided the much-needed resources that have enabled us to develop conversational systems. These resources include the time and expertise of the other members of the re*THINK Enterprise team. Without them, we could not have created all of the pieces required to build a whole conversational agent.

I must also acknowledge the invaluable guidance of Rafah Hosn, who led the *What's In Theaters* project, which was my first experience in identifying best practices for conversational agent development.

And I must thank Mitch Mason for teaching me how to design dialog through examples and my endless questions about how to extract an entity or define a condition.

Finally, I thank my family for understanding when I spent weekends writing instead of being a husband or a dad.

Robert J. Moore
March 2019

1

Introduction

> *All conversations are interactions, but not all interactions are conversations.*
>
> —Moore and Arar [2018]

While we all know how to have a conversation, we do not all know exactly how they work. We typically ignore the mechanics of human conversation in order to focus on the message being communicated. But if we are to *design* such mechanics, we must understand them more formally. Fortunately, there are fields of study within the social sciences that do just this. Social sciences, like Conversation Analysis (CA), can help user experience (UX) designers understand the principles and patterns of natural human conversation for the purpose of creating user interfaces that attempt to simulate them.

The best way to understand the mechanics of natural human conversation, and other conversation-like interactions, is to capture their details in a mechanical recording and closely analyze a text-based transcription of that recording. Transcripts enable you to focus on the talk itself, rather than the medium, although when done face-to-face, embodied actions, as well as the physical setting and artifacts, may be required to understand it fully. Consider the following excerpt from a transcript of a conversation-like interaction (Example 1.1).

Example 1.1 Rogerian Therapy.

```
A:  Men are all alike.
B:  In what way?
A:  They're always bugging us about something or other.
B:  Can you think of a specific example?
A:  Well, my boyfriend made me come here.
B:  Your boyfriend made you come here?
A:  He says I'm depressed much of the time.
B:  I am sorry to hear that you are depressed.
A:  It's true. I am unhappy.
B:  Do you think coming here will help you not to be unhappy?
```

What can we say about this interaction? We might say that it is about "how men are annoying." Or we might say that A is complaining about men and B is listening. Or we might say that A is making statements and B is asking follow-up questions. Or we might even say that B is practicing Rogerian therapy, a person-centered method of psychotherapy pioneered by Carl Rogers in the 1940s, by getting A to talk about herself.

In fact, A is a user and B is the computer program ELIZA [Weizenbaum 1966, pp. 36–37], one of the first chatbots. The user typed her utterances, and the automated agent returned textual responses. ELIZA appears to be a good listener, but it does not have much to say. It does not have any real-world knowledge to offer. And apart from Rogerian-style psychotherapy, it provides no service. Weizenbaum [1966, p. 42], the creator of ELIZA, explains how the chatbot works psychologically.

> . . . it has a crucial psychological utility in that it serves the speaker to maintain his sense of being heard and understood. The speaker further defends his impression (which even in real life may be illusory) by attributing to his conversational partner all sorts of background knowledge, insights and reasoning ability. But again, these are the *speaker's* contribution to the conversation. They manifest themselves inferentially in the *interpretations* he makes of the offered responses.

Weizenbaum [1966] points out that the behavior of the system leads users to attribute to it the ability to hear and understand. In other words, because ELIZA can *do* "listening" and even "probing," users attribute more cognitive capabilities to it than it actually has. He suggests further that we may even form such illusory impressions, to some degree, with other humans in "real life."

From a conversational UX design perspective, the question is: how does ELIZA do it? How does the automated agent* do *"listening"? An analysis of the transcript (see Example 1.1), as well as Weizenbaum's (1966) paper, reveals how. First, we can see that ELIZA does listening by producing a generic *paraphrase request* (line 2) of the user's initial statement (line 1). Such an utterance can be employed appropriately after *anything* the user says without recognizing its intent. The agent also produces a generic *example request* (line 4), which can also be used after about any statement made by a speaker (line 3). By requesting paraphrases, ELIZA gets the user to say what she already said again but in different words, which humans routinely do in order to manage troubles in understanding [Schegloff et al. 1977].

Next, ELIZA employs a range of generic repeat techniques. It repeats the user's entire utterance (line 6) but with the appropriate transformations of the pronouns. This demonstrates that the agent "heard" what the user said (line 5) and understands it to the degree that it could change "my" to "your" and "me" to "you."

Weizenbaum [1966, p. 42] suggests that this grammatical understanding leads the user to attribute deeper understanding. ELIZA further repeats a part of the user's prior utterance, with transformation, "I'm depressed" (line 7), and incorporates it into an utterance that recognizes it as bad news by offering an apology, "I am sorry to hear that you are depressed" (line 8). Finally, ELIZA produces another *partial repeat* of the user's utterance (line 9) and combines it with a reference back to an even earlier utterance (line 5): "Do you think coming here will help you not to be unhappy?" (line 10). Through the production of such generic repeat and paraphrase techniques, tailored to the user's prior utterance(s), ELIZA helps foster the impression that the user is being "heard and understood."

Weizenbaum (1966) explains that ELIZA consists of a set of rules and templates for transforming input utterances. ELIZA can do this without understanding the meaning of what is said but simply by manipulating natural language. Like a foreigner with limited knowledge of English [Weizenbaum 1966, p. 37], ELIZA applies a rule-based template to English utterances; for example, "one that specifies that any sentence of the form 'I am *blah*' can be transformed to 'How long have you been *blah*?', independently of the meaning of *blah*." ELIZA identifies the "most important keyword," some minimal grammatical context, and transforms the user's utterance into a conversation-like response [Weizenbaum 1966, p. 37]. Through this mechanism, it appears to do "listening."

1.1 Natural Language Interfaces

While chatbots and virtual agents have been around since ELIZA [Weizenbaum 1966], there has been a recent resurgence of interest in them as major computer companies have released their own. Apple's Siri, Amazon's Alexa, Google's Assistant, Microsoft's Cortana and IBM's Watson are just a few examples, not counting natural-language-based assistants by startups. With persistent Internet connections and machine-learning algorithms, virtual agents are much smarter today than they were 20 years ago. While most of these systems accept voice input from users, especially using far-field microphone arrays, many accept text input, sometimes from standard applications like SMS and Instant Messaging. But although virtual agents are becoming ubiquitous, interactions with them are still awkward, confusing, limited and fraught with troubles in mutual understanding.

Conversational interfaces are very different from graphical user interfaces (GUI). ***In conversational interfaces, the graphical elements are generally minimal, for example, a text-entry box and a message history or a microphone button or nothing at all. User interaction is conducted primarily through the words: typed or spoken.*** The interaction metaphor for these interfaces is the natural, human conversation, rather than the direct manipulation [Shneiderman 1982] of graphical interfaces.

Although natural language processing (NLP) has given us powerful automated tools for analyzing the spoken and the written word alike, it does not provide a model of how bits of language are sequenced into an interaction that is recognizable as a "conversation" [Sacks et al. 1974]. Some [Button et al. 1995, p. 209] have pointed out that a purely linguistic approach to building conversational interfaces, in which natural language is isolated from social activities, underestimates the "extent to which the capacity to talk and to carry on a conversation involves the possession of something other than purely 'linguistic' skills, and, indeed, of other things than skills." There is something more than language that must be modeled in a conversational interface and that is the organization of *natural conversation*. Natural human conversation is a complex system in its own right [Sacks et al. 1974, Schegloff 2007], which Sacks characterized as a "machinery" [Sacks 1984].

Natural language processing (NLP) techniques enable the automatic analysis of bits of language, for example, English or Spanish or Mandarin, but how to string those bits of language together into the kinds of sequences that occur in natural conversation is left to the UX designer or software developer. Authoring "questions" and their "answers" is simple enough. But how should the agent respond if the user says "okay" or "oh" or "what do you mean?" Building the range of natural conversational actions is still a hard problem. Luckily, Conversation Analysis (CA) focuses precisely on this domain. It documents the systematics, or "mechanics," of how people naturally talk in a wide variety of settings and languages [Sacks et al. 1974, Schegloff 2007, Sidnell and Stivers 2013]. Understanding these systematics formally is necessary for designing conversational UX, in addition to understanding the systematics of language.

Today's chatbots and voice assistants are something in between web search and a human. On the one hand, they are more conversational than interacting with a web search engine. You never say "thank you" or "what do you mean?" to a search engine, although it may say, "Did you mean X?" to you [Moore and Churchill 2011]. Search engines do not understand those kinds of conversational actions, only "queries." On the other hand, chatbots and voice assistants are not as conversational as a native human speaker, such as your family or friends. While you can talk to your mother about a wide range of things that are going on in your life, virtual assistants, limited by current technology, do not possess that range of knowledge or ability to handle such conversational complexity. Instead, they are more like customer service interactions, which tend to be limited in scope, or conversations with non-native speakers, which tend to require simpler utterances.

Even the most "conversational" of natural-language agents do not truly engage in conversation. Human conversation is merely the metaphor for this type of user interface, just as *direct manipulation* is the metaphor for graphical user interfaces [Shneiderman 1982]. This kind of human-computer interaction is at best a "simulacrum" of human conversation [Button 1990, p. 68] and at worst bears no resemblance to it at all. We might think of conversational interfaces as "conversation games." Like video games, they offer simplified simulations of the interactions with the real world, whether navigating a humanoid avatar through a realistic 3D environment or interacting with other players through virtual face-to-face [Moore et al. 2007].[1] Conversational interfaces are game-like in that they are interactive but consist of a limited set of rules and legal "moves" compared the real phenomenon they attempt to evoke. Just as users must learn how video-game interfaces work, or any other user interfaces for that matter, they too will need to learn how to "play" conversation games and how to navigate conversation spaces. Conversational interfaces constitute a distinctive form of interaction, which borrows interaction patterns from natural human conversation but also exhibits its own mechanics.

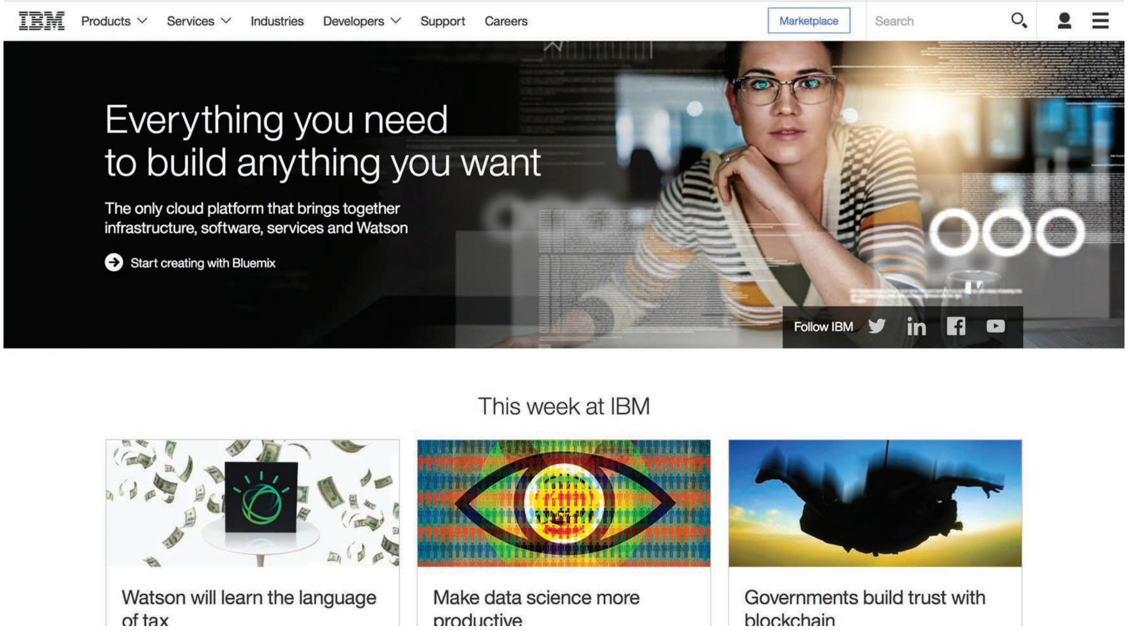
1.2 Conversational UX Design

While it is easy to create natural-language interfaces with today's platforms, creating an effective and engaging user experience is still a major challenge. Creating an interaction that works like a human conversation in particular is an even harder problem. To appreciate the need for a new kind of UX design, consider *web design* for a moment. In 2016, the IBM homepage looked like this (Figure 1.1). Whether you like it or not, it is clearly designed by a professional. It has a clean look and feel. Attractive fonts and color palette. Colorful hi-res images. It uses standard, familiar elements, such as drop-down menus, a search box, social media icons, a "hamburger" menu and more. Furthermore, you can hire someone to build a user interface like this. *Web designer* is an established role that is part of a relatively mature discipline.

But this was not always the case. Consider the IBM homepage back in 1996 (Figure 1.2). By today's standards, it looks amateurish at best. (No doubt it was created by an amateur web designer.) There are no consistent color palette or font guidelines. The main interaction element is the hypertext link. The layout is simplistic. The images, low-res. And the bullets and shadows look like they were created in

---

[1] We do *not* mean to suggest that conversational agents are the same as the simplistic "dialogue trees" common in video games. Conversational agents are much more sophisticated and nonlinear.



Everything you need to build anything you want

The only cloud platform that brings together infrastructure, software, services and Watson

This week at IBM

Watson will learn the language of tax

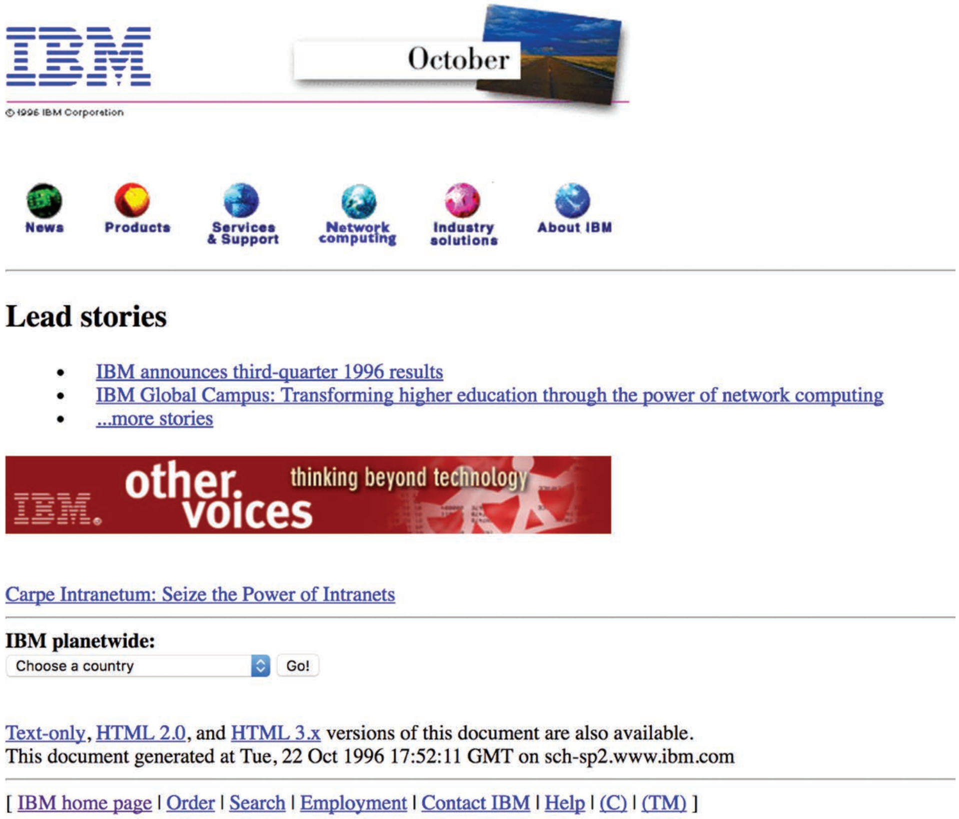
Make data science more productive

Governments build trust with blockchain

Figure 1.1 IBM Homepage 2016. (Reprint Courtesy of International Business Machines Corporation, © International Business Machines Corporation)

PowerPoint. So what is the difference? Over the past 20 years, the discipline of web design emerged. As the technical capabilities of the world wide web evolved, for example the <IMG> tag and high-speed networks, it became possible to apply formal knowledge of *graphic design* to web pages to make them "look more like glossy magazines" [Gillies and Cailliau 2000]. With the combination of user-interface design and graphic design, originally developed for the print industry, a new design discipline was born.

Today, conversational interfaces are at the stage that web interfaces were in 1996: the technologies are in the hands of the masses, but mature design standards have not yet emerged around them. And the visual solutions that evolved for graphical interfaces do not help us much with conversational interfaces, especially in the case of voice. With conversational interfaces, the user experience consists primarily of the design and sequencing of utterances. Instead of looking to the graphic arts for help, this time UX designers should look to the social sciences. Just as formal knowledge of the graphic arts revolutionized graphical user interface design, formal knowledge of human conversation can revolutionize conversational user interface design.



Lead stories

- IBM announces third-quarter 1996 results
- IBM Global Campus: Transforming higher education through the power of network computing
- ...more stories

Carpe Intranetum: Seize the Power of Intranets

IBM planetwide:

Text-only, HTML 2.0, and HTML 3.x versions of this document are also available.
This document generated at Tue, 22 Oct 1996 17:52:11 GMT on sch-sp2.www.ibm.com

[IBM home page | Order | Search | Employment | Contact IBM | Help | (C) | (TM)]

Figure 1.2 IBM Homepage 1996. (Reprint Courtesy of International Business Machines Corporation, © International Business Machines Corporation)

1.3 Conversation First

Current chatbot and voice assistant platforms enable you to create a variety of different styles of user interaction. Because they use natural language classification (NLC), these platforms are a technical advance over earlier voice control systems that only recognized a set of pre-specified command phrases. NLC enables recognition of a much wider range of phrases. However, how NLC is used together with the agent's responses and other UI elements can produce very different user experiences. Below are *invented* examples of different interaction styles that we see in today's chatbots and virtual agents: System-Centric, Content-Centric, Visual-Centric and Conversation-Centric [Moore and Arar 2018]. The styles are not mutually exclusive. A given system may contain features of all of them.

System-Centric Style

Although system-centric interaction styles recognize natural language as input, they still require the user to conform to certain unnatural input requirements. That is, they require users to formulate their utterances as valid queries, which pack all of the details relevant to their request into a single turn at speaking. This input requirement is due to the fact that the automated agent will not remember the sequential context, or what has been said so far, from turn to turn. Technically speaking, each turn by the user initiates a new interaction, which always consists of only two turns. *Context* is used within an utterance but not between utterances. In addition, such systems tend to be user-driven: interactions with the system are initiated by the user. Finally, system-centric styles tend not to do much conversation management, things like repairing the prior turn, "what do you mean?", or closing the prior sequence, "okay", no doubt because they do not remember the prior utterance to which the user is referring.

Voice Control

The system-centric interaction style is the most prevalent today. It includes voice control interfaces and information-retrieval interfaces, like that of web search. Voice control interfaces are perhaps the oldest type of natural language interface. They are based on an interaction model of simple input-output, or two-turn sequences, for the purpose of controlling a system. The system can only understand a short set of commands, not other social actions, and perform the appropriate response. Voice input is often provided as an alternative to button input so that users can operate the device hands free. Such voice control interfaces have been widely used since the early 2000s in cars. In addition to pressing a button on the dashboard or steering wheel, users can give a command, for example, "air conditioner on" or "radio 106.7 FM" or "find nearest gas station." In most cases, the response to the command by the car is a nonverbal action.

Today, virtual assistants often use a system-centric, voice control style. For example, leading voice assistants, or smart speakers, mostly recognize and perform commands issued by the user, but they tend to respond with a wider range of utterances, as well as nonverbal actions. However, these assistants still largely fail to remember the sequential context across turns, treating most user utterances as a new exchange.[2] This prevents these agents from performing a wide variety of

---

[2] An exception might be support for so-called "slotted intents." These enable the agent to remember an intent from a previous turn until dependent details are elicited from the user. This is a limited form of multi-turn conversation and will be discussed in Chapter 5.

conversational activities. For example, although virtually all assistants claim to "tell jokes," very few actually can do *joke telling* correctly. If you request a joke from an agent (Example 1.2, line 1), it will say a question joke (line 2) and then give away the punchline (line 3) (U stands for user and A stands for agent).

Example 1.2 Question-Answer Joke Telling.

1 U: **Alexa, tell me a joke.**
2 A: How does NASA organize a party?
3 They planet!

When telling a question-answer joke, the teller waits to see if the recipient can answer the joke and gives the punchline only after the recipient shows that he or she cannot. Most virtual agents cannot do this, but instead only deliver one-liners. A more extreme example can be found from another leading agent. If you request a knock-knock joke from it, you get an even more unnatural response (Example 1.3).

Example 1.3 Knock-Knock Joke.

1 U: **Tell me a knock knock joke**
2 A: Knock knock. Who's there? Bob. Bob who?
3 Bob, I don't do knock-knock jokes.

This agent will pack the whole joke-telling sequence into a single response. Neither of these are a correct way of telling a joke. Although the natural language content is that of a question-answer joke and a knock-knock joke, the sequential structure of the jokes is anything but natural. The problem of course, is not access to joke content, but the underlying interaction model of independent, two-turn exchanges. Jokes cannot be performed in a single turn, although witty remarks can. And likewise many other conversational activities require more than two turns to perform.

Information Retrieval

In addition to voice control interfaces, another kind of system-centric style is that of natural language search interfaces. Like voice control interfaces, these interfaces recognize natural language, are user-driven, and are limited to two-turn exchanges. However, unlike voice control, natural language search interfaces return information as their responses, rather than nonverbal actions. Such interfaces tend to be text-based rather than voice-based and are built on a simple query-response model found in web search and other forms of information retrieval. For example, the following exchange is an example of this style (Example 1.4).

Example 1.4 Information Search.

1 A: `Hello. How can I help you!?`
2 U: **`Original Medicare coverage diabetes screenings`**
3 A: `Medicare covers tests to screen for diabetes as well`
4 `as services and supplies to help treat the disease.`
5 U: **`Original Medicare eligibility diabetes screenings`**
6 A: `Part B covers these screenings if your doctor determines`
7 `you're at risk for diabetes or you're diagnosed with`
8 `pre-diabetes. These lab tests are covered if you have`
9 `any of these risk factors:`
10 `((list of factors))`
11 U: **`what about glaucoma tests?`**
12 A: `There are no results for your query. Please enter a`
13 `valid query.`

The healthcare assistant recognizes only “valid queries” and then looks up medical information in a database. As long as the user types a valid query (lines 2 and 5), the agent will return results, if available (lines 3–4 and 6–10). In each query, all of the relevant keywords must be given and therefore repeated, for example, “Original Medicare” and “diabetes screenings” (line 2 and 5). However, if the user speaks in increments (line 11), building off the context of prior turns, as is common in natural human conversation, the system fails (lines 12–13). Information search interfaces are therefore very similar to web search engines, like Google or Bing. They treat everything as a new query. We know how to use search engines to find all kinds of useful information, but we never say “thank you” to them because we know they will search the web for that too.

In practice, system-centric interaction styles tend to be what results when the user experience is created by an engineer or developer rather than by a designer. They are modeled after familiar device control or information retrieval interactions, and they require the least work to build the dialog because they force the user to adapt to the valid-query format. However, system-centric styles are not very conversational. They lack a key characteristic of natural conversation, *persistent context*, and the affordances that are enabled by that.

Content-Centric Style

Another interaction style found in chatbots and virtual assistants is what we might call content-centric. In terms of their underlying interaction model, they are identical to information search interfaces. They too recognize natural language, are user-driven, and are limited to two-turn exchanges. However, they differ primarily in the form of their content. Instead of retrieving data, they retrieve handcrafted answers to questions like those commonly found in web-based Frequently Asked Question (FAQ) pages. In fact, in many cases, they are no doubt adaptations of existing FAQ content. For example, the following exchange is inspired by such systems (Example 1.5).

Example 1.5 Q & A.

1 A: Hello, I am an automated Medicare Assistant. I can
2 answer questions about Medicare insurance. What is your
3 question?
4 U: **am I covered for diabetes screenings?**
5 A: Medicare covers tests to screen for diabetes as well
6 as services and supplies to help treat the disease.
7 Medicare will pay for 100% of its approved amount for
8 test even before you have met the Part B deductible.
9 You will pay no copay or deductible for these tests
10 if you see doctors who take assignment. Doctors and
11 other health care providers who take assignment cannot
12 charge you more than the Medicare approved amount.
13 Medicare Advantage Plans cover all preventive services
14 the same as Original Medicare. This means Medicare
15 Advantage Plans will not be allowed to charge cost-
16 sharing fees (coinsurances, copays, or deductibles)
17 for preventive services that Original Medicare does
18 not charge for as long as you see in-network providers.
19 U: **thanks**
20 A: I'm sorry, I don't understand your question.

When the user asks a yes/no question about health insurance coverage (line 4), he receives a thorough and lengthy response (lines 5–18). Instead of answering the question with a yes or a no, the agent answers with a statement, "Medicare covers tests to screen for diabetes" (line 5). From this kind of response, the user can usually infer a yes or a no. However, the agent continues on to explain multiple details and contingencies: coverage includes related "services and supplies" (line 6), it does not count toward your deductible (lines 7–8), there is no copay (line 9), approved doctors cannot charge more than the approved amount (lines 10–12), this applies to both Medicare Advantage Plans and Original Medicare (lines 13–18), and more. In other words, instead of saying yes or no, the agent implies that "it depends," and then details the set of circumstances on which it depends. Notice that some of the details will not be relevant to the user; for example, users with "Original Medicare" only do not need to be told about "Medicare Advantage Plans."

Overall, the authored answers tend to be relatively verbose and detailed since accuracy and thoroughness are of primary importance. In fact, content-centric interfaces tend to be co-created with subject matter experts or content designers who author the responses. But the disadvantages of long and thorough responses is that they preclude use in voice-based interfaces, and their length tends to limit opportunities for back-and-forth over a single topic since all of the details are presented in one extended monologue. In content-centric systems, recognized utterances, or "intents" (Chapter 3), tend to be topic-based rather than action-based, as they direct all inquiries or requests pertaining to a particular topic to a single response.

Visual-Centric Style

Visual-centric interaction styles borrow heavily from GUI, web and mobile interfaces, all of which rely on the principle of *direct manipulation* [Shneiderman 1982] instead of text commands or natural language. Graphical elements, such as buttons, images, emojis and other visual elements, are mixed into the interaction alongside natural language inputs. The most common type of visual-centric interface is the "chatbot" interface. Chatbots combine chat, Instant Messaging or texting-style interfaces with Internet bots that perform simple tasks. The following example shows what the chatbot or visual-centric style interaction looks like (Example 1.6).

Example 1.6 Chatbot.

```
A:  Hi, I'm Melanie, your virtual nurse! I can answer your
    Medicare questions. To start, which plan do you have?
    [Original Medicare] [Medicare Advantage] [ Other ]
    ((U clicks/taps Medicare Advantage button))
A:  Okay, Medicare Advantage. What is your question?
U:  am I covered for diabetes screenings?
A:  Medicare covers tests to screen for diabetes as well
    as services and supplies to help treat the disease.
    Medicare will pay for 100% of its approved amount for
    the test even before you have met the Part B
    deductible. You will pay no copay or deductible for
    these tests if you see doctors who take assignment.
A:  Was this answer useful?
    [ Yes ] [ No ]
```

Like the older Interactive Voice Response (IVR), chatbots tend to constrain user to a limited set of response options. However, instead of surfacing those options through speech, chatbot interfaces surface them through sets of buttons (lines 3 and 14). By offering buttons instead of requiring free text, chatbot systems improve the discoverability of intents recognized by the system and, for those particular intents, improves the speed of user input. Also, because they are text-based, chatbot interfaces are often verbose or text-heavy, slowing the rate of interaction with longer responses.

Chatbots often incorporate web and mobile techniques for eliciting user feedback after returning a response. For example, after providing an answer (lines 7–12) to the Medicare question (line 6), the agent asks, “Was this answer helpful?” (line 13), along with buttons for yes and no (line 14). Or they will elicit feedback on the answer with thumbs-up and thumbs-down buttons. Since yes/no questions already project the type of answer they prefer and because both responses are very short, there is really no advantage to displaying buttons for yes and no, unless the designer intends to constrain the user to only those options. In fact, in keyboard and mouse situations, clicking yes/no buttons requires moving one’s hand off of the keyboard and onto the mouse or trackpad and back, thereby eliminating their convenience. But more importantly, chatbot interfaces tend to be agent-driven and somewhat linear, minimizing points at which users are not told what to do next. For example, the Medicare chatbot does not wait for the user’s response to its answer about diabetes screenings. Instead it takes the next turn, hijacking “third position” with the request for feedback (line 13). In human conversation, third position, which comes after the first and second parts of an action pair, or “adjacency pair” (e.g., question-answer, Schegloff [2007]), is an important slot in which the initiator of the pair can give natural feedback, such as closing the sequence with “thank you” or repairing the second part with “what do you mean by doctors who take assignment”? Asking “was this helpful?” after every agent answer precludes the user from doing a range of third-position actions that provide a wider range of conversational feedback (Chapter 6).

Because chatbots solve some conversational UX design problems through visual elements, rather than through verbal actions, they cannot be fully ported to voice-only platforms. As a result, we see a split in the design of natural language interfaces along the lines of text- and voice-based systems. As chatbot interfaces introduce more buttons and visual elements, their designs drift farther from voice and conversational interfaces and closer to GUI and mobile interfaces. The more sophisticated chatbot interfaces are created by UX designers who apply their expertise in graphic design to the problem of natural language interaction.

Conversation-Centric Style

Finally, a conversation-centric style, the topic of this book, can be distinguished from the other three. In addition to recognizing natural language, a conversation-centric style also recognizes and exhibits the kinds of actions that are distinctive of natural human conversation. In other words, it can do *conversation management*: it can recognize social actions for managing the conversation itself. ***"What did you say?," "none of your business," "okay," "never mind" are all operations on prior utterances in the interaction. The social actions they perform constitute a major part of what makes a conversation a conversation.*** Likewise, you would not say a computer could "play chess" if it could not perform checking, castling and all the other actions that constitute *chess*. Furthermore, you will not find such conversational actions in other forms of language use, such as news stories, administrative forms, poems, etc. Designing the Medicare agent with a more conversational interaction style would entail conversation management (Example 1.7).

Example 1.7 Conversational.

1 A: Hi, I'm Alma. I'm here to answer any questions you
2 have about your Medicare insurance. How can I help
3 you?
4 U: **I have a question about diabetes screenings**
5 A: Okay, what's your question?
6 U: **am I covered for them?**
7 A: Diabetes screenings are covered at 100% of the
8 approved amount so there is no copay or deductible
9 if you see doctors who take assignment.
10 U: **what do you mean by take assignment?**
11 A: Oh. Assignment means that your doctor, provider or
12 supplier agrees to accept the Medicare-approved
13 amount as full payment.
14 U: **ok thanks**
15 A: You're welcome! Is there anything else I can do
16 for you?

The conversational agent can handle *preliminaries* to inquiries and requests (line 4), which check the conditions upon which those inquiries and requests depend. The conversational agent can handle a range of *repeat* and *paraphrase requests* on its own or the users' responses (line 10). And the conversational agent recognizes common methods of *closing a sequence* (line 14) or *closing a conversation*. All of these require that the agent remember what it is talking about across multiple turns.

In addition, the responses in a conversation-centric interaction are relatively short, or "bite-sized," as utterances tend to be in natural human conversation. This enables efficiency and speed as conversations happen in real time, either through voice or through text. Instead of giving the user a long and thorough response, as with the content-centric style, the conversation-centric style breaks the same content into smaller chunks, returns the most relevant chunk first and allows the user to unpackage the other chunks as needed. The result is more back-and-forth utterance exchange instead of longer monologues.

Conversation-centric styles are fully mixed-initiative and nonlinear. Either the user or the agent can lead, and the user can always redirect the conversation. Consequently, they frequently rely on the user to decide next actions, rather than on the agent leading at every turn transition. Such a conversation-centric style of interaction is the goal of a conversational UX designer, who brings a formal knowledge of how human conversation works to the design of talking agents. And because it is accomplished primarily through the words, without relying on visual interaction methods, a conversational style will work with either voice or text input. The conversation-centric interaction style therefore is characterized by (a) persistent sequential context; (b) fewer words per response, more back-and-forth; (c) nonlinear interactions with many trajectories; and (d) recognition of common conversational activities.

While each of the natural-language interaction styles detailed above, system-centric, content-centric, visual-centric, conversation-centric, can be useful to users for particular applications, we believe that the future of AI interfaces is the conversation-centric style. People have always wanted to talk to machines the way they talk to each other. This book is about designing conversation-centric interfaces, or *conversational* agents, not simple chatbots or voice control systems.

We advocate a *conversation-first* approach to the design of natural language interfaces, which is analogous to the mobile-first strategy in web design [Moore and Arar 2018]. ***While mobile-first design begins with the small screen of the mobile device and scales up to larger displays [Wroblewski 2011], conversation first begins with just verbal input and output, whether voice or text.*** The UX designer must enable the user to converse with the agent through the words alone, without buttons or visual aids. Voice interfaces, or platforms like the Short Message Service (SMS), force one to design for short utterances rather than for buttons, long lists, or document-like responses. Once the conversation is fully functional, it can be enhanced, as needed, through coordination with visual aids, just as a human speaker may use menus or charts to supplement his or her talk. As a result, users can talk to an agent through multiple communication channels, although the experiences will vary in their affordances, similar to human conversation when the speakers are talking face-to-face versus talking over the telephone. A conversation-first design strategy involves a focus on (1) conversation-centric interaction styles, including support for conversation management; (2) core functionality through the words

Table 1.1 Types of natural-language interaction styles

System-Centric Style
like voice control or web search
within-turn context
two-turn interaction model
recognizes commands or queries
user-driven
voice or visual
Content-Centric Style
like frequently asked questions (FAQ)
within-turn context
recognizes questions
document-like, topic-based responses
user-driven
visual only
Visual-Centric Style
like desktop or mobile interfaces
within-turn context
requires direct manipulation
buttons, menus, lists, images, etc.
agent-driven
visual only
Conversation-Centric Style
like natural conversation
cross-turn, persistent context
recognizes conversational actions
fine-grained back-and-forth
mixed initiative
voice or visual

alone, whether voice, text, or both; and (3) compatibility with multiple platforms, from voice-only and simple chat to desktop and large displays. In this manner, starting with conversation enables a designer to consider progressive enhancement [Gustafson 2015], so that greater functionality can be increasingly implemented as the modalities of conversation evolve.

To summarize, the four natural-language interaction styles exhibit multiple distinctive features (Table 1.1). Each style is an ideal type, and multiple styles may be mixed in the same application.

1.4 Mutual Understanding

Although we can build user interfaces that in some ways work like natural conversation, can these machines really understand us? As we saw above with ELIZA, a chatbot may sometimes continue a conversation without understanding the substance of what the user is saying. Today's chatbots and voice assistants are often characterized as "AI" (artificial intelligence) or "cognitive" and as capable of "understanding natural language" so that the user can speak to them as he or she would to another person. The user does not need to learn a specialized set of commands, as in the days of command-line interfaces. These metaphors clearly suggest that machines can think and understand. But what does this mean? Does it suggest that computers have consciousness, private thoughts, or empathy? What does it mean to say that a computer "understands" the user? To explore this question, we briefly draw on philosophy and social science to specify what we mean when we say that humans "understand" each other.

Perspectives on Human Understanding

From a commonsense view, we often conceive of "understanding" as a mental state, a psychological phenomenon. For example, if I picture an elephant in my mind and tell you to "picture an elephant," then an image of an elephant appears in your mind too. If we both imagine the same animal, then we would say that we "understood" each other. But if I had an image of a rhinoceros in my mind and incorrectly said "elephant," then we have not understood. According to this notion, understanding is first and foremost an overlap of mental states or contents.

While this is a commonsense view of understanding, similar, more sophisticated versions have been proposed by philosophers and scientists. For example, according to the picture theory of meaning [Wittgenstein 1922], language consists of complex, or "molecular," propositions that can be broken down into "atomic" propositions, which can be broken down no further. These atomic propositions then correspond to "atomic facts" in the world. If I observed an elephant swaying to live music, my mind would decompose that reality into its atomic facts, find the corresponding atomic propositions, and transmit them to you through the complex proposition, "I saw an elephant swaying to live music!" Your mind then reverse engineers the complex proposition into its atomic propositions, and from the corresponding atomic facts, your mind forms a complex image similar to what I saw.

While the commonsense notion of understanding or even the picture theory of meaning seem plausible at first glance, many scholars have questioned them. Even the philosopher of language who formulated the picture theory later argued that he was wrong. In his later work, Ludwig Wittgenstein [1953] argued that shared understanding cannot be achieved from private mental states, like the image of a swaying elephant. I have no way of knowing what you pictured in your mind, but I make judgments about whether you understood nonetheless. How can I do this?

Wittgenstein [1953] argues that shared understanding is in the first place the result of social practices, or "language games." Rather than consisting of atomic propositions, the meaning of a word or phrase is actually its *use* in some social practice or situation [Wittgenstein 1953, #43]. According to this view, understanding becomes a social event or sociological phenomenon. To demonstrate how meaning and understanding are based on social practice, Wittgenstein (1953), in part, used numerous examples of imagined interactions between math teachers and students. We attempt to summarize his argument with just two imagined examples of our own (Example 1.8).

Example 1.8 Non-Pedagogical Inquiries.

Teacher: Do you know what the frequency of human speech is?
Student: Yes
Teacher: Do you know how it differs from pitch?
Student: Uh-huh

Teachers never do this, at least not when they are testing a student's understanding of a concept. They do not because to answer these questions adequately is not to *demonstrate* understanding of the concepts of "frequency" and "pitch" but merely to *claim* it. As Weizenbaum [1966, p. 43] points out, "The crucial test of understanding, as every teacher should know, is not the subject's ability to continue a conversation, but to draw valid conclusions from what he is being told." Instead, testing a student looks more like this (Example 1.9).

Example 1.9 Pedagogical Inquiries.

Teacher: What is frequency in human speech?
Student: The number of complete cycles of variations in air pressure occurring per second.
Teacher: How is it different from pitch?
Student: Pitch is the human perception of frequency.
Teacher: What's a similar example of the relationship between frequency and pitch?
Student: That between wavelength and color.

This method of testing, sometimes called *viva voce* (or "oral examination"), invites the student to give definitions and explanations of the concepts, which in turn reveal the student's interpretations or misinterpretations. Wittgenstein would say that in such “language games,” when the student produces a correct or adequate explanation, the teacher says he or she “understands” the concept. Conversely, the teacher says he or she “misunderstands” or “partially understands” when incorrect or inadequate explanations are demonstrated. That is how we *use* the term “to understand” in practice. That is how it functions in social interactions. Understanding and misunderstanding are not attributed when an image appears privately in the student’s mind. The teacher has no access to that, only to the student’s actions, demonstrations, performances. In other words, this is what *understanding* is, from a practical or functional perspective. It is a kind of outcome or achievement of an interaction, of a social practice.

When it comes to understanding in the context ordinary conversation, sociologist Harold Garfinkel [1967] points out another dimension. While the social practice of *viva voce* involves spending time probing the limits of a student’s understanding of topics, conversation in everyday life mostly does not. In the midst of everyday activities, we have places to go and things to do; there is “no time out” to probe and test the other person’s underlying interpretations. So for practical considerations, we instead *assume* that the other person understands until further notice, that is, until there is some evidence to the contrary.

Garfinkel [1967, p. 42] demonstrated this through a simple exercise: “Students were instructed to engage an acquaintance or a friend in an ordinary conversation and, without indicating that what the experimenter was asking was in any way unusual, to insist that the person clarify the sense of his commonplace remarks.” The student experimenters overwhelmingly met with resistance (Example 1.10).

Example 1.10 Norm Breaching.

```
Subject:  How are you?
Student:  How am I with regard to what? My health, my finances, my
          school work, my peace of mind, my...?
Subject:  Look I was just trying to be polite. Frankly, I don't
          give a damn how you are.
```

Here, as in accounts of other students, probing understanding of ordinary expressions and utterances quickly drew reprimands from the subjects. What is normal in the classroom with technical expressions is abnormal outside the classroom with ordinary expressions. “The anticipation that persons *will* understand, the occasionality of expressions, the specific vagueness of references, the retrospective-prospective sense of a present occurrence, waiting for something later in order to see what is meant before, are sanctioned properties of common discourse”

[Garfinkel 1967, p. 41]. A level of uncertainty and vagueness in ordinary conversation is expected and trying too hard to remedy it will be seen as breaching a basic social trust.

But one implication of Garfinkel's 1967 exercise is that on any particular occasion, understanding may be assumed incorrectly. Weizenbaum [1966, p. 42] likewise suggests that speakers in "real life" may falsely attribute "background knowledge, insights and reasoning ability" to their recipients. Understanding in conversation is rarely definite. It may be demonstrated, faked or assumed. And it can only be probed or tested through further interaction, and even that will be abandoned once the tester is satisfied for all practical purposes. This *functional* conception of understanding is much messier than the picture theory.

Starting from this functional notion of shared understanding, sociologist Emanuel Schegloff went a step further and asked, how do people do it? By analyzing detailed transcripts of naturally occurring human conversations, Schegloff and his colleagues [Schegloff et al. 1977, Schegloff 1992b] identified and formalized the methods that people use to achieve shared understanding, or "intersubjectivity." They specify the "machinery" through which people design their talk for their particular recipient(s) and then deploy a set of methods for "repairing" troubles in understanding only if they emerge. Consider the following invented exchange (Example 1.11).

Example 1.11 Understanding Repair.

```
Speaker A:  How are you?
Speaker B:  Fine.
Speaker A:  No, I mean, do you still have symptoms?
Speaker B:  Oh, yeah my knee is hurting me.
Speaker A:  Okay.
```

In this case, Speaker B displays an interpretation of A's prior inquiry, "How are you?," as the kind of *welfare check* typical in the openings of conversations by responding with "Fine." However, in the next turn, or "third position," Speaker A initiates repair on his initial utterance by marking it as such, "No, I mean," and paraphrasing it, "do you still have symptoms?" Speaker B then provides a different kind of response. Thus Speaker B initially displayed misunderstanding of A's inquiry, but then understanding following A's repair. We will talk more about this infrastructure for achieving shared understanding in conversation in the chapters that follow.

Machine Understanding

Given this discussion of understanding, we now return to the question, "Can machines understand?" According to the commonsense theory of mind, machines cannot understand people because they cannot conjure an image as it appears in a human speaker's mind. Computer algorithms work differently from human brains so whatever the machine does internally is not understanding. "Computers cannot think because they are machines" [Button et al. 1995, p. 149]. According to this view, "understanding" is a uniquely human ability.

However, we come to a very different conclusion if we conceive understanding as the outcome of social practice. According to a functional conception of understanding, a machine like a human can *understand* if it can *do*. "The grammar of the word 'knows' is evidently closely related to that of 'can,' 'is able to.' But also closely related to that of 'understands.' " [Wittgenstein 1953, #150]. If a machine could successfully play the role of the student in a *viva voce*, then we could use the term "understanding" to characterize that interactional outcome or performance.

Computer scientist Alan Turing argued a similar point in his discussion of whether machines could think and in his formulation of an "imitation game," today known as the "Turing test."[3] Turing [1950, p. 433] replaces the question of whether machines can "think" with the question of whether they can "imitate" humans. Where the former question leads to endless philosophical arguments, the latter question can inspire the development of computer technologies. No doubt it has, in part, inspired scientists at IBM to develop Deep Blue, a computer that can imitate the playing of chess, and Watson, a computer that can imitate the playing of the trivia game Jeopardy. And of course it inspired the development of ELIZA, a computer that can engage in Rogerian-style conversation.

---

[3] Button et al. [1995, p. 145] acknowledge that Wittgenstein could "certainly" be "aligned with the functionalists" and that the "Turing test might be seen as instantiating" his "understanding-as-performance" view, but they then dismiss it as "the most superficial reading of Wittgenstein." However, their supporting argument depends on a straw man. They replace the "understanding-as-performance" view with a superficial version of it in which single performances of apparent understanding are taken at face value. As a result, they claim, inauthentic performances cannot be distinguished from authentic performances, for example, a student being fed answers by someone else from one who actually knows the material. But of course these *are* routinely distinguished *through further performances*. If the teacher suspects such cheating, he or she arranges a new test, under different conditions. *Viva voce* has the advantage that the questions are not predetermined and the questioning can be adapted on the fly to the student's responses, making understanding harder to fake. And at some point, the teacher will stop eliciting new performances and make a practical judgment of the student's understanding.

Turing [1950, p. 446] points out that, from a functional perspective, we employ *viva voce,* with its unpredictability and probing, to "discover whether someone really understands something or has 'learnt it parrot fashion."' If a computer can engage successfully in *viva voce* with a clever and thorough teacher, we should say that it "understands" the topic in question, even though there is the possibility that we are being fooled, just as with human students. A machine's level of understanding is judged by its performances in multiple situations, under varying circumstances. Similarly, if a computer can engage in conversation with a user, we can judge its ability to understand conversational topics, as well as to understand conversational actions themselves. The value of the Turing test is not to fool subjects into thinking that a machine is a human but to enable them to compare the performance of the computer with the performance of a human. It redirects our attention from the philosophical question to the technical one.

So in principle, we argue that computers can potentially understand without thinking, just as humans can understand without thinking. However, today's computers are far from demonstrating general or deep understanding. Creating a computer that can engage in the kind of *viva voce* of educational settings is still a hard problem. Returning definitions and explanations, like "What is frequency in human speech?" is easy enough, but performing real-time conceptual analysis, like, "What's a similar example of the relationship between frequency and pitch?" is much harder. Today's chatbots may demonstrate limited understanding in narrow tasks but fall apart when probed in different contexts or even with different phrasing. Even with the possibility of functional machine understanding, achieving it with real systems, at levels comparable to humans, may not ever be feasible. Such is the challenge of *general AI.*

But if we acknowledge Garfinkel's (1967) observation that shared understanding in ordinary conversation is usually only deep enough for all practical purposes, then machines may be able to understand humans well enough in many kinds of narrow conversational settings, such as service encounters. In service encounters, the conversations tend to be highly repetitive and the usual goals relatively narrow. Customer service agents typically answer inquiries or fulfill recurrent requests or troubleshoot predictable problems within a limited domain. Even though today's chatbots and voice assistants cannot handle domain-independent *viva voce*, they may be able to understand what the user says and does well enough to answer inquiries, fulfill requests, or troubleshoot problems for all practical purposes.

Furthermore, in order to do this, machines must be able to engage in the repair practices that Schegloff and colleagues demonstrate. Natural language understanding (NLU) techniques, while necessary, are not enough. They provide only for the machine to interpret the user's utterances. But such interpretations must be tested in interaction before understanding can be determined and must be repaired if misunderstanding or partial understanding is displayed. Thus conversational systems also need natural conversation understanding (NCU), or the ability to engage in repair practices, as specified by Schegloff [Schegloff et al. 1977, Schegloff 1992b]. We will return to this topic in Chapters 4 and 6.

To summarize, we offer a definition of *understanding* that we will assume throughout the remainder of this book. Understanding is not the same thing as interpretation. Interpretation is the analysis of the language and the action of an utterance, but *understanding is the demonstration of correct or adequate interpretation of social action within interaction*. Otherwise *thinking* one understands would be the same thing as understanding.

Weizenbaum [1966, p. 43] admits that ELIZA does not understand what the user is saying, but rather "maintains the illusion of understanding." However, while speakers may on occasion conceal misunderstanding from each other, in general, achieving mutual understanding is a fundamental goal of natural conversation. Weizenbaum also acknowledges that the display of misunderstanding or partial understanding is critical for natural conversation: "A switch of objectives from the concealment to the revelation of misunderstanding is seen as a precondition to making an ELIZA-like program the basis for an effective natural language man-machine communication system." Conversational agents must display their interpretation of what the user has said or did, as well as of the topic at hand, so that users can see if the agent understands them and if not, can attempt to repair it.

1.5 About This Book

The goal of this book is to inspire a new form of user experience (UX) design for the development of conversational agents. Our intended audience is the UX designer working on applications with natural-language interfaces, such as chatbots, virtual agents or voice assistants. We are speaking to the UX designer who gets his or her hands dirty authoring dialog logic and agents' utterances, rather than relying on a system developer or content designer to determine the user experience. We trust that those who have attempted to design the user experience for conversational agents have found, as we have, that such applications demand something different from what is needed for other kinds of desktop or mobile applications. ***The* user experience *consists not primarily of the manipulation of buttons, menus and windows,***

but in the sequencing of utterances in ways that seem "natural" or "conversational." As a result, a more formal knowledge of how human conversation is structured, turn by turn, seems invaluable.

To this end, we aim to provide a guide and a reference for conversational UX designers that includes an overview of conversation science, assets for UX design, and a process to use in development projects. First, we review principles and models from the field of Conversation Analysis (CA) to provide a theoretical foundation for understanding *human conversation* (Chapter 2); this material will be unfamiliar to most UX designers. Second, we provide a brief overview of *conversation authoring* using the Intent-Entity-Context-Response (IECR) paradigm (Chapter 3), which will be familiar to those who have created dialog flows on platforms like Amazon Alexa, Google Assistant, and IBM Watson Assistant. Third, we outline a new approach to conversational UX design, the Natural Conversation Framework (Chapter 4), which adapts principles, concepts and models from Conversation Analysis to the design of natural-language interfaces. Fourth, we provide a "pattern language" [Alexander 1977, Erickson 2000] of conversational UX design patterns for common conversational activities, as well as conversation management (Chapters 5–7), which we intend readers to use as a reference when designing conversational UX. Finally, we help the reader put all that they have learned from previous chapters into practice by outlining a general *design-thinking* process and adapting it for the unique challenges of conversational UX design (Chapter 8). By the end of this book, the reader should be prepared to design conversational user experiences that draw from a scientific foundation in Conversation Analysis and that use a systematic design framework.

2

Conversation Analysis

Designing user experiences that are "conversational" requires some formal understanding of the structure and mechanics of natural, human conversation. Conversation Analysis (CA) is a sub-field of sociology that focuses on the structure of human talk-in-interaction. It emerged with the invention of the compact audio cassette recorder in the 1960s. As co-founder Harvey Sacks [1984, p. 26] explains,

> It was not from any large interest in language . . . that I started with tape-recorded conversations, but simply because I could get my hands on it and I could study it again and again, and also, consequently, because others could look at what I had studied and make of it what they could, if, for example, they wanted to be able to disagree with me.

So since the 1960s, conversation analysts have been going out into the field, armed first with audio tape recorders and then with video cameras, to capture naturally occurring talk-in-interaction in all its many forms: ordinary telephone calls with family and friends; talk between customers and employees, patients and doctors, and students and teachers; talk among scientists working in laboratories; and much more. Although they focus on *talk*-in-interaction, conversation analysts have also examined the ways people use gestures, eye gaze, and other nonverbal actions and physical artifacts, in combination with talk, in order to accomplish a conversation.

The literature of Conversation Analysis reveals the formal, qualitative models of naturally occurring conversation, its "machinery" [Sacks 1984]. In other words, it contains a massive set of patterns of natural human conversation in many different settings and in many different languages. Conversation designers do not need to reinvent the wheel by going out into the field and studying human conversation, although doing this strategically can be valuable. Instead, they can apply patterns of human conversation from CA to the design of conversational UX patterns for virtual agents. It is the goal of this book to help translate the CA literature into a form that is accessible to and consumable by UX designers working in the area of conversational UX.

2.1 What Is a Conversation?

The term "conversation" gets used in a variety of different ways. Sometimes it is used to refer to a face-to-face talk you had with your mother or a telephone call with your boss or an exchange of text messages with your friend or a thread of comments on an online forum or a series of tweets from strangers all over the world and more. Each of these is certainly a social interaction, but only the first is the fundamental form of conversation, learned by all as children, while the other technologically mediated forms begin to deviate from this standard and exhibit their own distinctive features and practices. For conversation analysts, "conversation" is a particular kind of social activity, a "speech-exchange system" that displays the following 14 features [Sacks et al. 1974].

1. Speaker-change recurs, or at least occurs
2. Overwhelmingly, one party talks at a time
3. Occurrences of more than one speaker at a time are common, but brief.
4. Transitions (from one turn to a next) with no gap and no overlap are common. Together with transitions characterized by slight gap or slight overlap, they make up the vast majority of transitions
5. Turn order is not fixed, but varies
6. Turn size is not fixed, but varies
7. Length of conversation is not specified in advance
8. What parties say is not specified in advance
9. Relative distribution of turns is not specified in advance
10. Number of parties can vary
11. Talk can be continuous or discontinuous
12. Turn-allocation techniques are obviously used. A current speaker may select a next speaker (as when he addresses a question to another party); or parties may self-select in starting to talk
13. Various 'turn-constructional units' are employed; e.g., turns can be projectedly 'one word long', or they can be sentential in length

14. Repair mechanisms exist for dealing with turn-taking errors and violations; e.g., if two parties find themselves talking at the same time, one of them will stop prematurely, thus repairing the trouble

While many of the features can be implemented in today's virtual agents, several still cannot. For example, most agents cannot respond to the user "with no gap" (4) because they begin processing the user's utterance only after it is complete. As a result, there is an unnaturally long pause before each agent utterance.[1] In addition, to a large degree, what parties say is specified in advance (8); that is, the agent's utterances are authored in advance, although which variation of an utterance will be selected and what live data will be inserted into it can be dynamic. But by and large, today's voice assistants cannot generate speech on the fly. And with today's agents, the number of parties cannot really vary (10); they struggle with multiparty interactions. Either they only respond when explicitly addressed (for example, "Alexa, what is the weather today?") or they respond to every utterance they can detect regardless of whether it is directed to them. Since such voice assistants are typically "embedded in the life of the home" [Porcheron et al. 2018], distinguishing when a user utterance is directed at the agent versus directed at someone else is a critical but still hard problem. Yet most of the other features identified by Sacks et al. [1974] can currently be approximated to varying degrees.

Our current design philosophy is to strive to create user experiences that are closer to a telephone call or an exchange of text messages (SMS) than a face-to-face or Twitter exchange. The former requires computer vision of non-verbal activity and perhaps embodied robotic participation, the enabling technology for which we do not believe is ready for prime time. The latter, we feel, deviates too much from the fundamental form of face-to-face conversation.

2.2 Topic vs. Structure

Ordinarily when we observe or engage in conversation, we focus on the topics that speakers are talking about, rather than on *how* they are talking. But conversation designers, like conversation analysts, must notice what speakers are *doing* by talking about those topics and how they are doing it. Take the following two examples (2.1 and 2.2; see Appendix A for explanation of transcript conventions) from ordinary conversations [Pomerantz 1984, pp. 65, 70].

---

[1] Exceptions to this are recent advances in "duplex" systems from both Microsoft and Google.

Example 2.1 Pomerantz [1984, p. 65].

1 J: T's- tsuh beautiful day out
2   isn't it?
3 L: Yeh it's jus' gorgeous . . .

Example 2.2 Pomerantz [1984, p. 70].

1 A: God izn it dreary.
2   (0.6)
3 B: [Y'know I don't think-
4 A: [.hh- It's warm though,

What are the speakers talking about? In both cases, it is the same topic: the weather. But what is each speaker *doing*? In each example, the speakers are *offering assessments* of the weather. In Example 2.1, the second speaker, L, is doing *agreement* with the first speaker, J. But in Example 2.2, the second speaker, B, is doing *disagreement*. The topic is weather, but the social actions are assessments and agreements/disagreements. And conversation analysts have found that people do agreements and disagreements differently [Pomerantz 1984].

Agreements and other "preferred" actions, like grantings or acceptances, are done immediately and with no qualifications. Disagreements, on the other hand, like other "dispreferred" actions including rejections and declinations, are typically delayed by a pause, in this case by 0.6 seconds, prefaced and qualified. Often the dispreferred action is never explicitly stated but can be inferred from an account given by the speaker. In fact, recipients can see a dispreferred action coming from the pause alone. In Example 2.2, A begins to change her assessment before B has even said anything. This "preference organization," as conversation analysts call it [Pomerantz 1984], is part of how we talk, independently of the topics we are talking about and independently of the systematics of natural language.

***Because topic and structure are two different things, it takes the same "machinery" to talk about* nothing *as it does to talk about* something.** Whether your conversational application has a lot of content or only a little, you will still need to build the same structures to deliver that content. Take example of an interaction with our conversational agent, Alma (Example 2.3).

Example 2.3 A Conversation About Nothing.

1 A: Hello. I'm Alma. How are you?
2 U: **say again**
3 A: How are you?
4 U: **fine**
5 A: That's good.
6 U: **show me flights**

```
 7  A:  What?
 8  U:  I need to book a flight
 9  A:  What do you mean?
10  U:  never mind
11  A:  If you say so! Want to know what I can do?
12  U:  yes
13  A:  Okay. I haven't been trained yet. I can only
14      make small talk, tell some jokes or ask some
15      trivia questions.
16  U:  what do you mean?
17  A:  Ask me about the weather or to tell you a
18      joke or to ask you a trivia question.
19  U:  ok
20  U:  I need to go
21  A:  Thank you very much!
22  U:  you're welcome
23  A:  Goodbye.
24  U:  bye
```

What did they talk about? Nothing, really. At one point the user tried to talk about "booking a flight" (lines 6 and 8), but the agent failed to understand what the user was talking about (lines 7 and 9). Even though this user never engaged the content that the agent contains, the two still conversed for 21 turns. And those 21 turns demonstrate several basic functions of *conversation management,* something your conversation will need independently of the content of your application. Such actions and sequences distinguish natural conversation from other forms of natural language use. Chapters 6 and 7 will provide UX patterns for several kinds of conversation management.

2.3 Anatomy of a Conversation

One advantage of drawing on Conversation Analysis to inform conversational UX design is to borrow the vocabulary and conceptual framework it has developed. What do you call the different parts of a conversation? How do those different parts work together? Designing and building conversation-like interfaces requires a systematic vocabulary for referring to these parts.

Conversation analysts call the smallest unit in human conversation the "turn constructional unit (TCU)" [Sacks et al. 1974]. Such units may consist of words, phrases, clauses or full sentences. They constitute units after which the current speaker's utterance is hearably complete, and therefore speaker transition is possible. Utterances in a conversation then consist of at least one TCU, but often more than one. As speakers take *turns* producing utterances, they build recognizable sequences of talk, such as pairs of actions or "adjacency pairs." For example, when a *request* is made, whatever the next speaker says will be interpreted for how it might be *granting* the request, *rejecting* it, or working toward one of those actions [Schegloff and Sacks 1973, Schegloff 2007].

While utterances are organized into sequences, sequences are organized into *activities.* Activities are series of related sequences that accomplish some larger goal. They include things like conversation opening, instruction giving, teaching, troubleshooting, joke telling, order placing and more. At the highest level, conversations consist of multiple activities. The unit, "a single conversation," is bounded by an opening and a closing, with at least one activity, or topical structure, that accomplishes some business in between [Schegloff and Sacks 1973, p. 71]. For example, the following invented example of a single conversation consists of an opening, the activity of "requesting a ride" and a closing (Table 2.1). In this example, the opening consists of two sequences, a greeting sequence and welfare check sequence. The "requesting a ride" activity consists of two sequences, a request sequence and an inquiry sequence, with expansions. And the closing consists of one sequence: a closing sequence with a pre-expansion. This short conversation is comprised of three activities—opening, "requesting a ride" and closing—and five sequences—greeting, welfare check, request, inquiry and closing (sequences are indicated by shading).

Conversation Analysis involves, in part, characterizing utterances in terms of the social actions they perform. The CA literature is filled with the analysis of action types such as invitations [Drew 1984], assessments [Pomerantz 1984], informings [Maynard 2003], advice-giving [Jefferson and Lee 1992], storytelling [Sacks 1972], and much more. Each of these action terms are common, vernacular terms that everyone uses in daily life. Currently there is no exhaustive list that is standardized across disciplines, nor "generally accepted terminology" [Bach 1998, p. 388], of all of the common action types that people use in natural conversation. From an extensive taxonomy [Bach and Harnish 1979], Bach [1998, p. 388], writing about the area of Speech Act Theory in philosophy, offers a sample of fifty such actions:

Constatives. affirming, alleging, announcing, answering, attributing, claiming, classifying, concurring, confirming, conjecturing, denying, disagreeing, disclosing, disputing, identifying, informing, insisting, predicting, ranking, reporting, stating, stipulating

Directives. advising, admonishing, asking, begging, dismissing, excusing, forbidding, instructing, ordering, permitting, requesting, requiring, suggesting, urging, warning

Commissives. agreeing, guaranteeing, inviting, offering, promising, swearing, volunteering

Acknowledgments. apologizing, condoling, congratulating, greeting, thanking, accepting (acknowledging an acknowledgment)

Conversation analysts have examined how many of these "speech acts," or action types, are used in naturally occurring conversation, but not yet all of them. Their terminology largely overlaps with that of Bach [1998], although not entirely. When creating conversation spaces for virtual agents, the UX designer can apply such action terms to the various utterance types they build. This will help you organize user actions, or "intents," as well as the agent's utterances. Rather than organizing the utterances in terms of their topic, organize them in terms of the function they serve in the conversation. In the end, the most important point is

Table 2.1 Anatomy of a conversation

Turns	Sequences	Activities
A: Hello	GREETING	Opening
B: Hi!	GREETING	
A: How's it going?	WELFARE CHECK	
B: Good	WELFARE REPORT	
A: Hey, are ya going tonight?	PRE-REQUEST	Requesting a Ride
B: Yes	ANSWER	
A: Can I get a ride?	REQUEST	
B: Sure!	GRANT	
A: What time?	INQUIRY	
B: Seven o'clock	ANSWER	
A: Okay.	ACKNOWLEDGMENT	
A: Thank you so much!	PRE-CLOSING	Closing
B: No problem!	PRE-CLOSING	
A: Bye	FAREWELL	
B: Buh bye	FAREWELL	

not to use the “correct” label but to understand the social actions that utterances perform.

In addition to “speech acts,” or action types, conversation analysts also invent higher-level formal concepts that describe the ways these actions are organized and used. Such concepts are not common, vernacular terms. They include concepts such as turn-taking [Sacks et al. 1974], adjacency pair [Schegloff and Sacks 1973], sequence organization [Schegloff 2007], repair [Schegloff et al. 1977], preference organization [Pomerantz 1984], and many more. Such concepts and formal qualitative models are not generally found in linguistics or philosophy. The technical concepts of Conversation Analysis are useful for understanding the mechanics of how language-based social actions are organized and used in practice.

Principles of Conversation Design

Conversation Analysis has identified three general principles of how people design their own talk in natural human conversation. These are not deterministic laws concerning how people talk, but rather descriptions of general patterns in how people tend to design their talk. As free agents, humans are always able to deviate from these general patterns, and may routinely do so for particular effects or to accomplish certain actions. We can borrow these as principles for how we should aim to design conversational interfaces. These principles provide a high-level characterization of many aspects of natural conversation. They are recipient design, minimization, and repair.

Recipient Design

When we talk, what we say and how we say it depend on the particular person we are talking to, the recipient of our talk. Speakers tailor their talk, spoken or written, to their particular recipients in multiple ways, such as adapting to their perceived level of knowledge [Sacks et al. 1974, Sacks and Schegloff 1979]. Sacks et al. [1974] explain:

> By “recipient design” we refer to a multitude of respects in which the talk by a party in a conversation is constructed or designed in ways which display an orientation and sensitivity to the particular other(s) who are the co-participants. In our work, we have found recipient design to operate with regard to word selection, topic selection, admissibility and ordering of sequences, options and obligations for starting and terminating conversations, etc.

Speakers choose different topics and levels of detail depending on what they believe the other person knows and will recognize. For example, doctors will talk about medical conditions differently if they are talking to a patient or another doctor (or on whether they *think* they are talking to one or the other). With another doctor, medical jargon and knowledge can be assumed. With most patients, who lack medical knowledge, doctors will use more lay terms and commonsense knowledge. Or, for example, we design our talk for its particular recipient in the reference forms we use. If a co-worker asks me, “Who’s doing your remodel?” I might say, “My neighbor across the street. He’s a contractor.” But if another neighbor asked me that same question, I might simply say, “Dave is.” We choose different reference forms depending on what we believe about the other person’s knowledge. I believe that my neighbors know Dave and my co-workers do not. Of course, the assumptions we make about the other person’s knowledge may turn out to be wrong, as when a neighbor does not know Dave or a patient is a doctor him or herself. When our assumptions prove incorrect, we then adapt our talk in course of the conversation.

So how can we incorporate recipient design into conversational UX design? In some ways this is a hard problem. Prewriting unique responses for every user is not possible and even writing multiple responses for multiple types of users is often impractical. But there are some things conversation designers can do to incorporate aspects of recipient design. First, know your user. If you cannot write responses for each individual user, at least make sure your responses are tailored for your target audience. This is a principle of all human-centered design, regardless of the type of application. Don Norman [1988, p. 8] explains this as “an approach that puts human needs, capabilities, and behavior first, then designs to accommodate those needs, capabilities, and ways of behaving.”

Second, although we likely cannot adapt the wording of each response to the particular user (turn level), we can adapt the path a user takes by providing multiple paths through the same conversation space (sequence level). In other words, do not force all users down the same path. For example, expert users, who know the domain and the routine, should be able to accomplish things quickly and efficiently.

Example 2.4 Expert.

```
1  A:  How can I help you?
2  U:  show me flights from SFO to ORD on April 17-30.
3  A:  One moment . . .
4  A:  Okay, here are flights from San Francisco to
5      Chicago on April 17 to April 30.
6      ((list of flights))
```

In Example 2.4, the user demonstrates that she knows all of the information that is required to retrieve a list of flights. In this case, the user should be able to accomplish the task in one turn instead of being prompted for information that was already given. However, in Example 2.5, the user does not show the same level of expertise.

Example 2.5 Novice.

A: `How can I help you?`
U: **`I want to book a flight`**
A: `Okay, where are you leaving from?`
U: **`what do you mean?`**
A: `I need to know your origin city or airport?`
U: **`San Francisco`**
A: `What city do you want to fly to?`
U: **`Chicago`**
A: `When would you like to go?`
U: **`April 17--30.`**
A: `One moment . . .`
A: `Okay, here are flights from San Francisco to`
`Chicago on April 17 to April 30.`
`((list of flights))`

In this case the user formulates his request in a very general way and relies on the agent to elicit whatever details it needs in order to fulfill the request. Efficiency is traded for guidance by the agent. Both types of conversation paths, and variations in between, are critical for a conversational application. Not only do users possess different levels of knowledge and skill, but as the same user interacts with your agent, he or she will learn what it can and cannot do and thus may become more expert over time.

Although recipient design is not yet truly possible at the turn level given today's conversation authoring platforms, it nonetheless reminds us as designers that we should put "human needs, capabilities, and behavior first" and attempt to adapt to particular users at the sequence level by providing many paths through the conversation space.

Minimization

In addition to recipient design, another general principle of how people naturally speak is *minimization* [Sacks and Schegloff 1979, Levinson 2007, Moore 2008, Enfield 2013]. Not only do speakers design their turns so that particular recipients can understand them, but they also attempt to do so with the fewest words. For example, when referring to other people, speakers tend to use their names, which enables recipients to recognize the particular individual [Sacks and Schegloff 1979]. Returning to the example of “Dave the contractor,” we offer some invented examples to illustrate the principle (Examples 2.6–2.8).

Example 2.6 Minimal Reference.

```
A:  Who's doing your remodel?
B:  Dave
A:  Oh, cool!
```

Example 2.7 Elaborated Reference.

```
C:  Who's doing your remodel?
B:  My neighbor across the street. He's a contractor.
C:  That's convenient!
```

Example 2.8 Failed Reference.

```
D:  Who's doing your remodel?
B:  Dave
D:  Who?
B:  You know, my neighbor across the street?
D:  Oh!
B:  You had a beer with him?
D:  Right.
```

B uses the single name “Dave” to refer to his contractor when speaking to other neighbors who know him, rather than “my neighbor across the street.” For these recipients, “Dave” is both minimal and “recognitional” (i.e., reveals the individual to that recipient; Sacks and Schegloff [1979, p. 17]). On the other hand, terms like “a guy” or “someone” are about as equally minimal as “Dave,” but these generic forms do not enable the recipient to recognize who is being talked about. They are not recognitionals. Using such a generic form would imply that the recipient does not know the person in question. Similarly, because “Dave” is not recognizable to my coworkers who do not know him, B should not use it alone. Either “my neighbor across the street” or “Dave, my neighbor across the street” is better designed to achieve recognition, or understanding, but at the temporary expense of relaxing minimization. “. . . [T]he principle of a preference for recognition, with a relaxation but not suspension of minimization, is preserved when an incompatibility between the two, consequent on the user of a minimal recognitional that does not yield recognition, is noticed by recipient” [Sacks and Schegloff 1979, p. 20]. In other words, natural conversation is designed by speakers to be *efficient* [Levinson 2007]: enable the other person to understand using as few words as possible.

What does this mean for UX design? ***In order to exploit the efficiency built into natural conversation, a conversational agent's responses should be as short as possible while still enabling most users to understand.*** Many chatbots and voice assistants do not implement such minimization. Especially with text-based agents, designers are sometimes tempted to give more detail rather than less. This is the "content-centric style," as we saw earlier (Chapter 1). Unnecessary details may make the point of the speaker's action harder to grasp. Reading and especially listening to a conversational agent's responses take time and effort. We recommend that the designer strive to minimize agent utterances without sacrificing understandability. However, in order to provide for the minority of users who need more detail, enable them to initiate repair on the agent's utterances.

Repair

Sometimes in conversation, you miss what the other person said, or fail to understand it. Or you realize that he or she misunderstood what you said. In these situations, natural conversation provides methods for remedying these kinds of interactional troubles. Conversation analysts use the term "repair" to refer to the range of practices that we have for managing troubles in speaking, hearing or understanding [Schegloff et al. 1977]. In each case, a source of trouble is located in a prior turn, and a speaker redoes all or part of that prior turn. So "repair" refers to methods for repeating or paraphrasing all or parts of a prior turn. It does not refer to practices for addressing other kinds of interpersonal troubles, such as disagreement, rudeness, complaints, insults, etc.

Speakers repair utterances produced by themselves or others in many different ways in natural conversation. The repairs above (Example 2.3, lines 2 and 16; Example 2.8, line 3) are all instances of repair initiated from the "second position," that is, in the next turn after the source of the hearing or understanding trouble. But repair can be initiated from other positions as well. The following transcript (Example 2.9) demonstrates the four positions of repair [Schegloff 1992b]. In this example, from one of our early conversational agents, there are two sources of trouble: "Watson" (line 1), "Harry Road" (line 2), "address" (line 7) and "Almaden" (line 14).

Example 2.9 Four Positions of Repair.

```
1  U:  What is the address for Watson- I mean Almaden?
2  A:  IBM Almaden is located at 650 Harry Road,
3      San Jose, California, 95120, USA.
4  U:  Harry Road?
```

```
 5  A:  That's right.
 6  U:  ok thanks
 7  U:  What is the address for Almaden?
 8  A:  IBM Almaden is located at 650 Harry Road, San
 9      Jose, California, 95120, USA.
10  U:  no, I mean web address
11  A:  The web address for Almaden is
12      www.research.ibm.com/labs/almaden/
13  U:  thanks
14  U:  what was the street address for Almaden?
15  A:  IBM Almaden is located at 650 Harry Road,
16      San Jose, California, 95120, USA.
17  U:  I thought it was on Bailey road
18  A:  Oh, you mean the Silicon Valley Lab. That's
19      different from Almaden. It's 555 Bailey Ave,
20      San Jose, California 95141.
21  U:  oh ok, thanks
```

In the first turn, the user does a first-position repair: he says "Watson" and then immediately redoes it in the same turn "I mean Almaden" (line 1). This might occur in a voice interface when the user chooses the wrong name, but in a text interface, the repair would be invisible to the system because the user would merely delete "Watson" and type "Almaden." First position is the same turn as the trouble source.

After receiving the address for Almaden (lines 2–3), the user repeats a phrase from the agent's response, "Harry Road?" (line 4), thereby identifying it as a source of trouble. The agent responds by confirming that phrase (line 5), thereby treating it as a hearing trouble rather than an understanding trouble. This is another instance of second-position repair because the repair initiator, "Harry Road?" (line 4), is produced by the other party in the next turn to the trouble source "Harry Road" (line 2). Second position is the next turn by another speaker after the trouble source.

Repair can also be initiated from third position. For example, when the user says, "no, I mean web address" (line 10), he is redoing the word "address" that he produced earlier (line 7). After seeing that the agent responded with a street address (lines 8–9), the user disambiguates his own prior referent by adding the word "web" (line 10). Third position repair indicates that the recipient, in this case the agent, failed to understand the speaker's intent. The speaker can see from what the recipient made of his or her turn that there was a trouble source in that turn and then redo all or part of it. Third position is the speaker's next opportunity to speak after a recipient's response to the trouble source.

Finally, speakers may also initiate repair from fourth position [Schegloff 1992b]. While third position is the first speaker's next chance to talk after a trouble source, fourth position is the recipient's next chance to talk after responding. For example, when the user elicits the street address for Almaden again (line 14; first position) and the agent gives it (lines 15–16; second position), the user then does something that indicates a trouble in understanding: "I thought it was on Bailey road" (line 17; third position). Now the user is not initiating repair on an earlier turn, he is just reporting an assumption that he has. But this reporting enables the agent to see a source of trouble in the user's earlier reference to "Almaden" (line 14) and repair it: "Oh, you mean the Silicon Valley Lab. That's different from Almaden . . . " (lines 18–20; fourth position). IBM has two labs hidden in rural areas of South San Jose that are only about seven miles apart. Confusing the two labs is a common mistake. In terms of the conversation logic, if this agent detects features of one lab, street name or number or ZIP code, when the current topic is the other lab, it flags the confusion. Although it would be difficult to provide for all possible fourth position repairs, it is feasible to design for common ones like the Almaden/Silicon Valley Lab confusion.

Conversational repair can thus be initiated from different sequential positions, by different participants in the interaction, on different kinds of interactional troubles regarding speaking, hearing, and understanding. These repair practices are a basic component of conversational competence that are used to manage local troubles in the production and design of natural language utterances. ***Because conversation is an interactive form of natural language use, speakers, human or machine, do not need to get a particular utterance exactly right; repair enables them to make dynamic adjustments in the next or later turns.***

These three principles—recipient design, minimization and repair—together form a *basic mechanic* for speaking in natural conversation: (1) tailor your turn to the recipient; (2) make the turn as minimal as possible without being confusing; and (3) relax minimization only when the recipient displays trouble and resume it when the trouble is repaired [Sacks and Schegloff 1979]. In other words, speakers tend to use just enough words to enable the recipient to understand, and then they wait to see if it works. If not, they expand as necessary. This mechanic enables a kind of efficiency in speaking because speakers do not start with a verbose and elaborate utterance unless they anticipate a trouble in understanding. Because speakers always have a second chance to redo an utterance, they can try a concise, streamlined, even clever utterance first and expand only if necessary. Conversational UX designers should understand this basic mechanic and take inspiration from it.

2.4 Conversation Types

Human conversation consists of a generic speech-exchange system [Sacks et al. 1974] that is continually adapted by speakers to different activities and situations. Similarly, UX designers need to adapt generic conversational UX patterns to particular use cases. Is the virtual agent like a friend or a customer service representative or a teacher or someone else? The use case will determine the range of conversational UX patterns required for the range of activities entailed by it. The following are just four broad kinds of conversational use cases.

Ordinary conversation is the kind of interaction you have with family, friends, and even strangers every day. Ordinary conversations consist of the broadest range of activities from delivering news to checking up to seeking help or advice to learning to much more. In addition to such purposes, ordinary conversation is sometimes engaged in for its own sake; with small talk, the social engagement with another person itself is the purpose of the interaction. In Conversation Analytic theory, ordinary conversation is considered the most flexible type of conversation from which other types are adapted for particular purposes by adding special constraints [Drew and Heritage 1992]. An example of a spate of ordinary conversation, similar to the invented example above (Example 2.1), can be found in the following (Example 2.10; Schegloff [2007, p. 30]).

Example 2.10 JG 3:1.

```
1       ((ring))
2  C:   Hello
3  N:   Hi.
4  C:   Hi.
5  N:   Whatcha doin'.
6  C:   Not much.
7  N:   Y'wanna drink?
8  C:   Yeah.
9  N:   Okay.
```

In this transcript, N calls C on the telephone. After opening the conversation (lines 1–4), N inquires about C's current status (line 5). Conversation analysts call this a "preliminary" [Schegloff 2007], as we will discuss more in the next chapter, which checks a condition upon which his next action will depend. C indicates availability (line 6) and N extends an invitation (line 7), which C accepts (line 8). Invitation-acceptance sequences are a type of social action pair, or "adjacency pair" [Schegloff and Sacks 1973, Schegloff 2007], that are common in ordinary conversation.

Service conversations, or service encounters, are the kind of interaction you have with a customer service agent or organizational representative [Whalen and Zimmerman 1987, Jefferson and Lee 1992, Whalen and Vinkhuyzen 2000, Moore 2008, Félix-Brasdefer and César 2015, Szymanski and Moore 2018]. The roles are fixed: one person, such as a customer, member or citizen, requests service; the other person, usually a stranger, provides services on behalf of an organization. Services may consist simply of answering inquiries or taking actions or guiding the other through troubleshooting. Service conversations have distinctive openings: service providers typically do a greeting, self-identify, identify the organization and offer service, rushing through the transitions, so that the service seeker's first turn can be his or her request or reason for the call [Whalen and Zimmerman 1987]. Other times such openings may include a series of questions for verifying the service seeker's identity.

Example 2.11 NB I.2:Golf[2].

```
D:  G'morning. San Juan Hills Country Club?
G:  Guh morning. What's-w- what kind of a starting
    time ken:: we get fer::hh sometime this
    afternoon.
       (0.7)
G:  Any[time-
D:     [Oh:::, [let's see.
G:             [Any time tuhday.
D:  Two fordy. One, thirdy.
G:  One thirty?
D:  Mm hm::?
G:  One thirty.
       (0.7)
G:  .hh W'l at sounds like a good time?
       (0.4)
D:  What is the name?
G:  Detweiler. D-e-t,
      (1.2)
G:  w-e,
      (0.4)
G:  i-l-e-r-.
      (2.0)
D:  Foursome?
```

---

[2] This transcript is available online from Talk Bank at http://ca.talkbank.org/browser/index.php?url=Jefferson/NB/01golf.cha

```
24  G:  Yah.
25        (0.4)
26  D:  Electric carts?
27        (0.6)
28  G:  Uh:::, n:no? I don't think so.
29  D:  Okay. We'll see yuh then,
30  G:  Righto,
31  D:  Mm hm, Bye?
```

In this transcript, D is the service provider and G is the service seeker. D opens the conversation with a greeting and an identification of the organization he represents (line 1). G returns the greeting and makes a request for service, in this case, reserving a tee time (lines 2–4) at a golf course and country club. D offers some candidate times (line 9), and G chooses one (lines 10–14). After inquiring about additional details regarding the reservation (lines 16–28), D moves to close the conversation (line 29), which G confirms (line 30). D then bids farewell (line 31) although G does not reciprocate. Service encounters range from quick appointments (Example 2.11) to complex troubleshooting [Szymanski and Moore 2018] to requests for emergency services [Whalen and Zimmerman 1987].

Teaching conversations are the kind you have within a classroom setting or with a tutor. One person (or more) seeks knowledge; the other presents knowledge and tests understanding of that knowledge. In teaching conversations, teachers routinely ask the student questions to which they already know the answers. They may withhold the answers in an attempt to elicit the correct answers from the student [McHoul 1990]. Whereas correcting other people is typically discouraged in most other kinds of conversations for the sake of politeness [Schegloff et al. 1977], it is required in the teaching conversation for the purposes of learning.

Example 2.12 McHoul 1990:362.

```
1   T:  ... What's going on here
2       (1.5) ((hands go up))
3   T:  Yes
4   A:  Mining
5       (1.5)
6   T:  Mining, what sort of mining?
7       (2.0)
8   B:  Open cut mining
9   T:  Open cut?
10      (0.5)
11  C:  [Iron ore
12  D:  [Iron ore
```

```
13  T:  Iron ore. Why iron ore?
14      (1.0)
15  T:  Don't they mine other things (in) open cuts?
```

In this case, the teacher asks a series of inquiries to which he already knows the answers (lines 1, 6, 13, and 15). Through such quizzing, T gets the students, A, B, C and D, to display their knowledge through candidate answers (lines 4, 8, 11, and 12). T's inquiries elude or hint at the correct answers without providing them. This is one common way of teaching.

Counseling conversations are the kind you have with a therapist, counselor or advisor. One person seeks advice; the other listens and provides advice. The advice seeker may report a problem of a personal nature or a long-term goal and seek advice on how to manage it, rather than requesting that the other person manage it directly. As noted in Chapter 1, the chatbot ELIZA was explicitly designed to have such conversations, specifically Rogerian psychotherapy: "This mode of conversation was chosen because the psychiatric interview is one of the few examples of categorized dyadic natural language communication in which one of the participating pair is free to assume the pose of knowing almost nothing of the real world" [Weizenbaum 1966, p. 42]. In psychotherapy generally, the therapist asks questions and the patient answers them. The therapist may withhold judgment and let the patient lead the conversation without interrupting or changing the topic. Or the therapist may formulate what the patient previously said in order to suggest an alternative meaning [Antaki 2008, p. 30].

Example 2.13 CBTUV&JR110698.2.

```
 1  T:  .hhh So you work out yer months notice
 2  C:  [Yeh
 3  T:  [Can we look at that [bit of time
 4  C:                       [(an we got
 5      (0.5)
 6  C:  An we got made redundant
 7  T:  Yeah
 8      (0.9)
 9  T:  .phhh an you so you're you've just finished work
10      (0.4)
11  T:  You must have not known what to do with yourself
12      after twenty years
13  C:  I didn't do
14      (1.1)
15  C:  I came (.) came to (0.4) eh I found it very hard I
16      don't (.) get on with that
```

```
17       (1.5)
18  T:  So you found it very difficult [to ]adjust
19  C:                                 [yeh]
20  C: Yeh
```

In this excerpt, the therapist inquires about his client's job (lines 1 and 3), and the client informs him that he was laid off (line 6). From this fact, which is typically anxiety-inducing for people, the therapist formulates a series of upshots (lines 9, 11–12, and 18) of what the news might mean for the client. This is one way that therapists shape their clients' perceptions of events in their own lives. They adapt the ordinary conversational practice of upshot formulating to the purposes of self-understanding in the context of psychotherapy.

Each of these types of conversations and more depend on the same conversational machinery, such as turn-taking, sequence organization, and repair, but the activities and settings in which they take place contain distinctive patterns and slight adaptations [Drew and Heritage 1992]. Conversational systems likewise should be built on a shared, basic machinery so that users can rely on familiar practices but also accomplish the distinctive business of the particular application.

2.5 Conclusion

In this chapter, we have provided a brief introduction to the field of Conversation Analysis (CA). We discussed the difference between the topic of a conversation and its structure. Then we analyzed the anatomy of a conversation and provided a standard vocabulary for referring to the parts of a natural conversation: turns or utterances, sequences, activities and conversations. Next, we explained three principles that speakers use in designing their own talk: recipient design, minimization, and repair; and we described the mechanism through which these principles work together. Finally, we discussed the foundational nature of ordinary conversation and how it is used and adapted for accomplishing particular types of *institutional* conversations, such as service encounters, teaching and counseling. From this introduction you should be able to talk about the structure of human conversation in a way that is more formal and systematic than commonsense.

While this introduction merely scratches the surface of many topics in CA, we hope that it provides enough detail to pique your interest in the field and to dig deeper if needed in designing for particular use cases. Observing naturally occurring conversation can be invaluable for learning how to design automated agents, but first consult the CA literature and see if others have already done so.

3

Conversation Authoring

Our purpose in reviewing conversation theory is to enable you to apply it to the practice of conversation authoring. How or whether a conversational agent requires authoring by a designer depends on the type of natural-language platform on which it is built. There are several technological approaches to creating chatbots, voice assistants, and conversational interfaces. Each uses artificial intelligence (AI) or machine learning (ML), but they do so in different ways. Among them the most common types are *information-retrieval* approaches and *natural-language-generation* approaches.

Conversational systems that use information retrieval contain or access repositories of pairs of utterances. When the user produces a new utterance, a machine-learning technique known as natural language classification (NLC) compares it to the pairs in the repository and attempts to match it to one of the first utterances in a pair [Song et al. 2016]. If a match is made above a certain confidence threshold, the second utterance in that pair is returned to the user as the agent's response. Information-retrieval-based conversational systems therefore *do not generate new utterances*. All possible user utterance types and agent utterances are predefined in the repository, although the database may be very large and dynamic data may be inserted into predefined utterance frames.

On the other hand, natural-language generation approaches, as the name implies, are capable of generating new utterances. Rather than simply matching users' utterances to a set of pre-existing classes and retrieving predefined responses, these approaches synthesize new utterances, often through the same methods used for language translation [Kalchbrenner and Blunsom 2013, Sutskever et al. 2014, Vinyals and Le 2015]. In such approaches recurrent neural networks are used to recognize sequential patterns from examples of text-based interactions. Rather than natural language classification (NLC), they are based on sequence-to-sequence (seq2seq) models. Such models are promising, and some even claim that they "can hold a natural conversation and sometimes perform simple forms of common sense reasoning" [Vinyals and Le 2015]. However, the following transcript excerpt (Example 3.1) illustrates their performance.

Example 3.1 Vinyals and Le [2015].

1 ((Describe your problem: browser keeps crashing))
2 A: hi
3 U: **hi**
4 A: could you please let me know what is the operating
5 system you are using ?
6 U: **linux**
7 A: ok this is the issue with the system performance,
8 nasser , tools .
9 U: **what is that ?**
10 A: is it a windows machine ?
11 A: we have linux on your machine .
12 A: is it windows xp or windows 7 ?
13 U: **i am using linux**
14 A: let me take a remote of your machine and check if
15 issue has been pushed properly .
16 U: **ok**

In this case the agent, which was trained on a set of IT troubleshooting chat logs between human customer service agents and human customers, receives the formulation of the user's problem, "browser keeps crashing" (line 1), prior to the opening of the interaction. After greeting the user (line 2), which the user returns (line 3), the automated agent asks a diagnostic question about the "operating system" (lines 4–5). The user in turn provides one, "linux" (line 6). The agent then formulates the user's problem in an unexpected way: "the issue with the system performance, nasser , tools" (lines 7–8). While "the issue with the system performance" is technically a correct reformulation of "browser keeps crashing," it is less useful because it is less specific. Furthermore, the addition of "nasser , tools" appears nonsensical. To this, the user replies "what is that ?" (line 9), thereby initiating repair on the reference. But rather than clarifying what it means by "nasser , tools," the agent asks another diagnostic question: "is it a windows machine ?" (line 10). The question is irrelevant given the sequential context [Schegloff 1992a] of the conversation so far, plus it suggests a failure to recognize the user's prior answer (line 6). Without waiting for the user to answer, the agent then appears to provide one of its own, "we have linux on your machine" (line 11). It then contradicts this answer by asking, "is it windows xp or windows 7 ?" (line 12). The user finally responds by repeating his or her prior answer (line 6) in an elaborated form, "i am using linux" (line 13). The agent then proposes a next action (lines 14–15), and the user acknowledges it (line 16).

This excerpt suggests that conversational agents using sequence-to-sequence (seq2seq) models may not be ready for prime time. Although independently each of the things the agent says is no doubt typical in such IT troubleshooting interactions, in the context of *this* conversation they fail to show a sensitivity to previous turns by the user (lines 6 and 9) and by the agent (line 11). Others have observed that with natural-language generation approaches, "the generated sentences tend to be short, universal, and meaningless, for example, 'I don't know' or 'something' " [Song et al. 2016]. Somewhat like ELIZA, these agents may mimic aspects of the form of natural conversation, but they tend to demonstrate a failure to understand what the user is saying.

Because they are autonomous, natural-language-generation approaches to conversational agents will be a critical part of AI in the future. Yet one major obstacle to their advancement is the lack of high-quality transcriptions of real human conversations [Chen et al. 2011]. Despite advances in automatic speech recognition and speech-to-text services, naturally occurring conversational data still pose challenges [Moore 2015]. And as we see from the example above, such approaches require better methods for tracking the state of the current conversation, not just learning from a corpus of past conversations.

Today's most common conversation platforms use information-retrieval approaches, which we will call the Intent-Entity-Context-Response (IECR) paradigm. These platforms include Microsoft's Luis, Amazon's Alexa, Google's Api.ai, Facebook's Wit.ai, IBM's Watson Assistant, and more. IECR systems rely on the following standard components: (1) machine-learning-based "intents," or classes of phrases to recognize what the user said; (2) keyword-based "entities" to extract important bits of information from what the user said; (3) variables to preserve the "context" of both this interaction and/or a larger context; and (4) text "responses," conditioned on the intents, entities and context, which may or may not be rendered as voice using a text-to-speech component.

In the remainder of this chapter, we explain each element of IECR systems in more detail. As in other chapters in this book, we base our technical discussions primarily on IBM's Watson Assistant (formerly Conversation) platform, but the general principles should apply to other conversation platforms.

3.1 A Simple Finite-State Machine

Intent-Entity-Context-Response (IECR) systems at their core are simple finite-state machines that consist of a *tree* data structure (Figure 3.1). Each user input utterance starts at the top of the root level and is checked against a hierarchical set of nodes, each with its own set of logical conditions. If the utterance matches one set of those

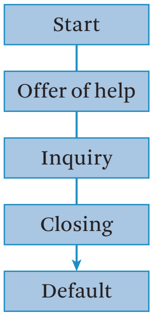


Figure 3.1 Dialog tree (root level).

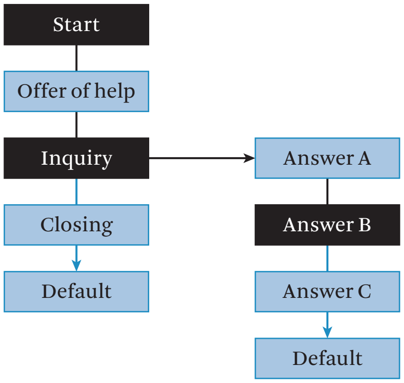


Figure 3.2 Dialog tree (branch level).

conditions, it splits off from the root level and down a branch of nodes, stopping at some leaf node, typically with an output response to the user. If the input utterance fails to match one of the sets of conditions, it stops at a default node at the bottom of the root level and gives a default output response, or flows down a branch with default responses.

For example, the system goes first with “Hello. How can I help you?” (Offer of help) and the user responds with, “Who invented the hard disk?” The user’s input utterance then matches the conditions on the “Inquiry” node, and it flows down a branch (Figure 3.2). The branch consists of another hierarchical set of nodes with conditions and a default at the bottom. If the utterance matches one of these sets of conditions, the system gives an output response, “The hard disk was invented by IBM in 1956” (Answer B). If it matches none of the conditions on the branch nodes, it responds, “I don’t know the answer to that” (Default).

IECR-based conversational systems represent a finite *conversation space*. All possible paths, from root to branch to leaf, are defined in advance. Users can take many different paths through the conversation space, but they cannot create new paths on the fly. Attempts to create new paths will result in unrecognizable input utterances that end with default output responses. However, there may be many, many unique paths through a single conversation space.

In simplest terms, a dialog tree determines what the output response to a user's input utterance should be given some conditions: it's a set of conditioned responses. In order to create a conversation space, UX designers must create these conditions and responses, which are discussed in turn.

3.2 Conditions (Intents, Entities, Context)

Conditions in an IECR system are what define which output response will be given for a particular input utterance. These conditions can be very simple or very complex. They are made up of combinations of three main components: intents, entities, and context variables.

Intents

In addition to far-field microphone arrays, natural language classification has made possible the latest generation of conversational systems. NLC techniques are often described as "understanding the intent of the user," where "intent" is conceived as a psychological meaning or mental state. This type of description can conjure up impressions of natural language classification as a kind of mind-reading technique. While that kind of technology might be nice to have, it's not what NLC techniques actually do.

The function of natural language classification is very simple: it tells you how similar a new string of text is to predefined groups of text strings. The conversational UX designer must first define a set of categories, or classes, that the classifier will use to analyze new strings. The classes should consist of turn constructional units (Chapter 2), especially phrases, that embody the range of conversational *actions* that the application can recognize—for example, "hello" for a #greeting or "what can you do?" as a #capability_check. The UX designer must anticipate *all* possible actions in the conversation space. This is perhaps the most difficult part of creating a conversational system.

Once an initial set of actions, or "intents," is created, the UX designer "trains" the classifier by grouping examples of the same action together in the same class,

Table 3.1 Natural language classes

Class	Example
`#greeting`	`hello`, `hi`, `hey`, `hi there`, `yo`, ...
`#welfare_check`	`how are you?`, `how's it going?`, `how do you do?`, ...
`#capability_check`	`what can you do?`, `what do you know?`, ...

Table 3.2 Confidence for new string: `what kind of things can you do?`

Class	Confidence
`#greeting`	0.01
`#welfare_check`	0.02
`#capability_check`	0.97

or bucket. In this way, the designer *programs by example* rather than by creating natural language rules for variations of the same action. For example, Table 3.1 shows three classes with examples of each.

The power of natural language classification comes from its ability to learn linguistic patterns of the example phrases placed in each category. The UX designer therefore does not need to add a string for every possible way a user might formulate the action; he or she must only add every *type* of way a user might formulate the action. With an example of each type, the classifier can tell you that a new string, never encountered before, is nonetheless very similar to a known string or strings (Table 3.2). In short, natural language classification helps the UX designer with *variation management*, or managing the many ways users can say the same thing. What it does not do is help the UX designer anticipate all the kinds of actions the user might take.

Intents, or natural language classes, can be designed in many different ways. Ultimately, intent design should reflect the design of the agent's responses, since the purpose of intents is to distinguish input utterances in order to return different responses. For example, in conversation with Agent A (Table 3.3), there is only a single response for inquiries into the capabilities of that agent. In this case, a single intent for a capability check is appropriate. Whatever the user says about "flights," Agent A will return the same response frame. In this sense, Agent A is not taking

Table 3.3 Agent A: Single flight response

Class	User	Agent A
`#flight_request`	`I want flights to New York`	`Here are all flights to New York.`
	`show me the cheapest flight to New York`	`Here are all flights to New York.`
	`I want flights under $300`	`Here are all flights to New York.`

Table 3.4 Agent B: Multiple flight responses

Class	User	Agent B
`#flight_request`	`I want flights to New York`	`Here are all flights to New York.`
`#cheap_flight_request`	`show me the cheapest flight to New York`	`July 10-15 is the cheapest time.`
`#budget_flight_request`	`I want flights under $300`	`Here are some cities under $300.`

full advantage of natural language classification because the same thing could be achieved simply by using entities or simple keyword-matching.

The power of natural language classification can be seen when the agent gives multiple responses for different requests that involve the same entities, such as “flights” (Table 3.4). Agent B can respond to flight requests in three different ways: by returning all flights to a city, by returning the cheapest dates to fly, or by returning cities with flights within a given budget. In order to give multiple responses like this, flight request examples must be divided into three different intent categories. ***We see then that how you should design your intents depends on how you design your responses.***

When designing intent classes, try to give as wide a range of canonical examples as you can think of. For example, for a welfare check, “how are you?”, “how’s it going?”, and “how do you do?” will cover a wider range of variation patterns than “how are you?”, “how are you today?”, and “how are you doing?” The first group includes three canonical patterns (how are you, how is it going, and how do you do), whereas the second group includes variations of only one pattern (how are you). A good intent for checking the agent’s welfare would include these three canonical patterns and more, with a few variations of each pattern. Once your agent goes into user testing, more variations will emerge. Adding new canonical patterns, rather than just variations of existing patterns, will give your intent better coverage.

Entities

While intents should be used to recognize the conversational actions a user is performing, entities, or keywords and their synonyms, should be used to extract particular details from what the user says. Unlike intents, entities tend not to use statistical techniques, such as natural language classification, to recognize similar strings of text. They simply provide exact string matching, although the case of the text (uppercase or lowercase) is usually ignored. For example, in the input utterance "I want flights to New York", such systems use natural language classification to identify the action, "flight request," but identify the destination of the request, "New York," by using an entity for destination cities. That entity would be a list of all the cities that the agent can recognize, or perhaps that have airports: New York, San Francisco, Chicago, Denver, etc. In addition, the entity dictionary should include any common synonyms for an entry. For example, "New York" may include the synonyms "NYC" or "the big apple." This technique of intent plus entity is more efficient and manageable than creating a unique intent for each entity, for example, one intent for "I want flights to New York" and a different intent for "I want flights to San Jose."

In some use cases, the same entity type may be used in different contexts in the same utterance. For example, "I want flights from San Jose to New York" uses the entity type "city" to refer to both the origin and destination of a trip. For this case, additional entity types can be used for origin and destination, that is, "from city" and "to city." The "from city" entity dictionary adds the word "from" before every city name, and the "to city" dictionary adds "to."

While some entities will be created and managed by the conversational UX designer, others may be managed by the conversation platform itself, through *system entities* or by a third party, through an external entity extraction service. For example, the Watson Assistant Service provides system entities for person names, locations, time, date, currency and more, while the Watson Natural Language Understanding Service extracts many types of entities, which can be passed to a conversation platform as context variables.

Context

Intents and entities enable the conversational UX designer to recognize user actions and extract details from the user's current input utterance, but variables enable the designer to preserve those details. This might be for use by a backend system to which requests are relayed, or simply across turns in the conversation. In addition, intents and entities enable the *user* to initiate particular conversation sequences, but context variables enable the *agent* to initiate them. In short, the “context” around the words of a user’s current input utterance is constructed and preserved by managing a set of variables.

What makes dialog management hard is that the strings of language that the user produces must be interpreted in the context of the overall conversation. Linguists talk of “anaphora” as an expression that depends for its meaning on other expressions, or “context.” For example, the referent of pronouns, such as “he,” “she,” “it,” or “that,” cannot be determined from the pronoun alone. Additional information must be used. In ethnomethodology, Garfinkel (1967) uses the term “indexical expressions” for anaphoric expressions; however, Garfinkel takes the concept a step further. All expressions are indexical expressions because they depend on some background knowledge and situational context to determine what they are doing. While “context” can mean many things, Schegloff (1992a) argues that the “sequential context” of an utterance is the primary kind of context that must be considered first in interpreting the meaning of an utterance.

In IECR conversation engines, the context of a user’s utterance, sequential or otherwise, must be captured through the use of variables. Such context variables may capture the previous turn (for example, name request), the topic being talked about (for example, San Francisco or flu shots), repairs on a response (for example, an elaboration or example), and more. ***Deciding which aspects of the conversation are consequential for the agent’s responses and how best to capture them through context variables is a major challenge in conversational UX design.***

If users always produced conceptually complete utterances, dialog design would be much easier. For example, the request “I want flights from New York to San Jose on July 10 to July 15” contains all the details that are needed to grant it. Entities alone can be used to identify the origin, destination, departure date, and return date. However, this is not how people usually talk. Natural human conversation often consists of shorter, “bite-sized” utterances that build across turns (Example 3.2).

Example 3.2 Multi-Turn Requests.

U: **I'm looking for flights**
A: Where do you want to go?
U: **New York City**
A: Where are you flying from?
U: **San Jose, California**
A: When do you want to go?

```
 7 U:  July 10-15
 8 A:  Okay. Here are flights from
 9     San Jose to New York on
10     July 10-15.
```

In this conversation, the user provides the details of the request across several turns instead of in one. As a result, the dialog logic must preserve these details across turns using *context variables*. When the user says, “New York City” (line 3), the agent must “remember” that they are talking about flight requests indicated in the user’s previous turn (line 1). When the user says, “San Jose, California” (line 5), the agent must remember both that they are talking about flight requests and that the user’s destination is “New York City.” This preservation of context, or “remembering” what we are talking about across turns, is accomplished by saving the intents and entities recognized in each turn to persistent context variables. So in response to the user’s first utterance (line 1), the dialog logic may set *request type* to “flights.” In response to the user’s second utterance (line 3), it may set *destination* to “New York City.” In response to the user’s third utterance (line 5), it may set *origin* to “San Jose.” And in response to the user’s fourth utterance (line 7), it may set *travel dates* to “July 10-15.” In this way, capturing intents and entities to persistent variables enables the saving of context across turns.

In addition to capturing and “remembering” the context of the current topic, variables can also be used to initiate sequences even if the user did not mention a trigger word or phrase. For example, the designer might want the agent to say “Anything else today?” only after it has provided some help to the user (Example 3.3).

Example 3.3 Appreciation Receipt.

```
1      ((agent retrieved flight information))
2 U:   thanks a lot
3 A:   You're welcome.
4 A:   Anything else today?
```

In this case, when the agent successfully retrieves a set of flights for the user (line 1), the dialog logic sets the context variable *helped* to true. If the user then does an appreciation (line 2), the agent will receipt the appreciation (line 3) and do a last topic check (line 4). However, if the agent did not succeed in providing any flight information or even in recognizing what the user was requesting, the response to the same user utterance is different (Example 3.4).

Example 3.4 Refutation.

```
1 U:   what's in theaters?
2 A:   What do you mean?
```

3 U: **what movies are currently playing?**
4 A: I'm sorry, I still don't understand.
5 U: **thanks a lot**
6 A: I don't think I did anything.

If the agent fails to recognize the user's intents (lines 1 and 3), the *helped* context variable never gets set to true. If the user then produces an appreciation (line 5), the agent now responds by acknowledging its failure to help (line 6). Persistent variables can thus be used to capture events, or their absence, earlier in the conversation for use as conditions, or "context," later in the conversation.

While much attention has been given to natural language classification and entity extraction, an equally important component for conversational interfaces is "sequential context" [Schegloff 1992a]. The primary context for interpreting utterances produced in conversation is the prior utterances in that conversation. For example, "how about Las Vegas?" is ambiguous in isolation, but following the utterance "Here are things to do in Los Angeles," it becomes sensible. Speakers rely heavily on previous talk to provide a context for their next utterance, which enables them to take shortcuts. For the conversational UX designer, deciding what aspects of the sequential context should persist and how to represent them through *context variables* is still a challenging design problem. Creating an interaction that works like a natural conversation requires capturing the current topic of the talk, the user's prior question or request, which entities the user has mentioned so far, whether the previous user utterance was recognized or not, and more. Therefore *context design* is a critical area within conversational UX design that must be advanced if virtual agents are to handle many common conversation patterns.

Mixed Conditions

We see then that the three main types of conditions in an IECR engine are intents, entities, and context variables. They can be used alone or in combination. This gives the conversational UX designer fine control in setting the conditions under which different agent responses will be triggered by the user.

One type of situation in which you might use mixed conditions is when you want the agent to respond differently to the same input utterance in different sequential contexts. For example, take the multifunctional term "okay" (Example 3.5).

Example 3.5 Alternative Responses to "ok".

1 U: **hi**
2 A: Hello. How are you?
3 U: **ok**

4 A: `That's good. How can I help you?`
5 U: **`What's your favorite sport?`**
6 A: `I only follow chess.`
7 U: **`ok`**
8 A: `Anything else today?`

The user first says “ok” (line 3) in response to “How are you?” (line 2) and the agent responds with “That’s good. How can I help you?” (line 4). However, later, the user says “ok” (line 7) in response to an answer, “I only follow chess” (line 6), and the agent responds instead with “Anything else today?” (line 8). These different responses to the same input utterance can be achieved with combinations of conditions. For the “ok” at line 3, the preceding response node for “Hello. How are you?” (line 2) sets *previous node* to “welfare check.” Then when the user says “ok” in the next turn, it matches the conditions: intent, acknowledgment, *and* the context variable *previous node* equals “welfare check.” Alternatively, if the user says “ok” when *previous node* does not equal “welfare check,” it will go to a lower sequence in the root level for which the condition is the intent, acknowledgment. The response for this sequence is “Anything else today?” (line 8). This dialog logic can be represented as two root-level nodes with the following dialog pseudocode (Example 3.6; see Appendix B for an explanation of the notation).

Example 3.6 Alternative Responses to “ok”.

```
1 if #acknowledgment AND $previous_node=welfare check
2   say "That's good."
3   goto Offer of Help.
4 if #acknowledgment
5   say "Anything else today?"
```

This pseudocode represents two root-level dialog nodes: the first with a mixed condition (line 1) and the second with a simple condition (line 4). Since the system flows from top to bottom, the user utterance, “ok” will match the mixed condition (line 1) only if *previous node* was set to “welfare check” on a previous turn. Otherwise, “ok” will match the simple condition (line 4). Any number of intents, entities or context variables can be mixed to make unique conditions. In this way, the same intent can trigger many different responses.

3.3 Responses

Although the long-term goal of artificial intelligence is to generate or synthesize responses to user utterances in real time, conversation engines using the Intent-Entity-Context-Response paradigm do not do this. Machine learning is used to classify the user's utterances, but the agent's utterances, and the logic for when to say them, are typically authored by a designer. This may be the conversation designer, who programs the intents, entities, and dialog, or it may be a content designer, who specializes in creating only the responses of the agent.

Because the agent's utterances in a IECR-based system are pre-authored, they can sound unnaturally repetitive. Humans tend not to say things in exactly the same way every time, but conversational agents will do so if we do not create variations. For example, in a service encounter, the agent may repeatedly ask the user if he or she would like anything else, as a pre-closing to the conversation. If the agent does this the same way every time, it can become tiresome for the user, especially if it is delivered through voice (Example 3.7).

Example 3.7 Single-Version Response.

```
 1  U:  what computer won Jeopardy?
 2  A:  An IBM computer named Watson.
 3  U:  ok
 4  A:  Is there anything else I can do for you today?
 5  U:  what was the name of the computer than beat Kasparov?
 6  A:  Deep Blue, a computer created by IBM, beat chess champion Garry
 7      Kasparov in 1997.
 8  U:  ok
 9  A:  Is there anything else I can do for you today?
10  U:  who invented the hard disk?
11  A:  The hard disk drive was invented by IBM in 1956.
12  U:  ok
13  A:  Is there anything else I can do for you today?
```

In this case, the agent's subsequent offers of help (lines 4, 9, and 13) are all on the same node in the dialog tree. Because the conversation returns to the node multiple times in the same session, the agent's utterance sounds unnaturally repetitive. To help compensate for this, most platforms enable the designer to author multiple variations of an agent's response. The designer then selects a rule to deliver these variations, usually consecutively or randomly (Example 3.8).

Example 3.8 Multiple-Variation Response (Consecutive).

```
 1  U:  what computer won Jeopardy?
 2  A:  An IBM computer named Watson.
 3  U:  k
 4  A:  Is there anything else I can do for you today?
 5  U:  what was the name of the computer than beat Kasparov?
 6  A:  Deep Blue, a computer created by IBM, beat chess champion Garry
 7      Kasparov in 1997.
 8  U:  ok
 9  A:  Anything else for you today?
10  U:  who invented the hard disk?
11  A:  The hard disk drive was invented by IBM in 1956.
12  U:  ok
13  A:  Anything else?
```

In this alternative example the same dialog node is used for the agent's subsequent offer of help (lines 4, 9, and 13), but the node contains multiple variations of the utterance, which it delivers consecutively each time the user hits that node in the session. While the response is long the first time the user receives it (line 4), it gets progressively shorter with each subsequent hit (lines 9 and 13). In this way, the designer can approximate how a human might truncate an utterance over repetitions or can simply to give some variety in the agent's responses.

Dynamic Knowledge

In addition to appropriate responses to particular conversational actions, the agent's responses also depend upon knowledge. This knowledge often lies outside of the dialog tree itself. It therefore must be retrieved from a backend database or API. A common approach for introducing dynamic knowledge into the agent's utterances is by inserting data into a response frame. This enables the designer to turn bits of information from existing databases and APIs into a conversation-like utterance (Example 3.9).

Example 3.9 Response Frames.

```
1  U:  I need a Mexican restaurant nearby
2  A:  Mario's is located at Beach and Main.
3  U:  thanks
4  A:  You're welcome! Anything else?
```

The agent's response (line 2) consists of the response frame, "*name* is located at *address*." As the user indicates different preferences, that is, cuisines and distances, the backend will insert different sets of values into the *name* and *address* context variables. With this method, a single response frame can be used for all restaurants in the database or API. In this kind of architecture, the conversational agent serves as a translator between the user and the backend: it translates the user's natural language into a set of standardized variable values, and it translates new variable values back into a conversation-like utterance.

3.4

Branching

Once you know how to create intents, entities, context variables, and responses and how to integrate backend data, you have all the basic components for building a conversation space. However, there are always different ways of combining these components in a dialog tree. Below we give recommendations for how to architect your dialog tree so that it supports maximum flexibility.

Typically in dialog design, branches off the root level are used to keep track of the agent's place in the conversation. A branch is used when an agent's response contains a question directed back to the user. Placing a user input point on the branch enables the system to know that the next user input is a response to the prior question. If the sequence requires multiple questions to the user, the corresponding branch will contain multiple user input points. For example, a branch with three dialog nodes will contain two user input points in addition to the root level input point. We represent the example using pseudocode (Example 3.10 and Appendix B for pseudocode definition).

Example 3.10 Multiple User Entry Points.

```
 1  user input A
 2  if #flight_request
 3    say "Where do you want to go?"
 4      user input B
 5        if @city
 6          set $destination to @city
 7          say "Where are you leaving from?"
 8            user input C
 9              if @city
10                set $origin to @city...
```

This example represents three nodes, each beginning with a condition (lines 2, 5, and 9). If the user's input matches the intent *flight request*, it says, "Where do you want to go?" (line 3). The branch then contains a point where the user can provide an answer (line 4). If the user's input contains a city, it will progress to the next node (line 5), where the city will be interpreted as a "destination" city and saved as a context variable (line 6). The agent then asks, "Where are you leaving from?" (line 7). There is then another point for user input on the branch (line 8) followed by a third dialog node (line 9). Saying a city name at this point in the dialog tree will be interpreted and captured as the "origin" (line 10). If the user answers the agent's questions (lines 3 and 7) as expected, the dialog branch works fine (Example 3.11).

Example 3.11 Staying on a Branch.

```
1 U:  show me flights ((user input A))
2 A:  Where do you want to go?
3 U:  New York City ((user input B))
4 A:  Where are you leaving from?
5 U:  San Francisco ((user input C))
6 A:  Okay. One moment please...
```

The user enters the branch at the root level (line 1) and provides a valid answer to the agent's destination question (line 3) and to the origin question (line 5). The branch enables the system to differentiate the destination city from the origin city.

The problem with this kind of dialog branch design is that if the user does not do the expected action (for example, by doing an unrecognized action or a requesting a repeat), he or she will fall off of the branch. Once off the branch, it can be difficult to get back on it (Example 3.12).

Example 3.12 Falling Off a Branch.

```
1 U:  show me flights ((user input A))
2 A:  Where do you want to go?
3 U:  the Big Apple! ((user input B))
4 A:  What do you mean?
5 U:  New York City ((user input A))
6 A:  I'm sorry I still don't understand.
```

In this case, the user requests flights (line 1), and the agent asks for the user's destination city (line 2). In response, the user utters a term that the system does not recognize, "the Big Apple!" (line 3). As a result, it falls off of the branch and hits the root-level default node (line 4). Now when the user corrects him or herself in the next turn, the flow is at "user input A" instead of "user input B" (line 5), and the agent responds incorrectly (line 6). One solution to this problem is to enable the system to keep track of the previous user input and enable the conversation designer to create rules for when to return to a user input on a branch, like user inputs B or C, and when to remain at user input A.

A different solution, which we recommend, is to avoid points for user input along dialog branches. Instead, only use a single entry point for user input at the top of the root level. In other words, the user input always starts in the same place. To achieve this, you need another method for determining the agent's next response, for example, if it should be "Where do you want to go?" or "Where are you leaving from?" We do this by using a context variable, *previous node*. Instead of placing a user input right before the next question, we capture the previous question as context and use it as a condition for the next question (Example 3.13).

Example 3.13 Single User Entry Point.

```
 1 user input
 2 if #flight_request
 3    set $previous_node to "destination request"
 4    say "Where do you want to go?"
 5 if #city AND $previous_node = "destination request"
 6    set $destination to @city
 7    set $previous_node to "origin request"
 8    say "Where are you leaving from?"
 9 if #city AND $previous_node = "origin request"
10   set $origin to @city...
```

Here the context variable *previous node* is used to keep track of the sequential context. If the agent previously requested a destination or an origin, that becomes part of the context (lines 3 and 7). With both the context variable and the user input, the system can determine if the user's mention of a city is intended as a destination or as an origin (Example 3.14).

Example 3.14 Single User Entry Point.

```
1 U:  show me flights ((user input))
2 A:  Where do you want to go?
3 U:  the Big Apple! ((user input))
4 A:  What do you mean?
5 U:  New York City ((user input))
6 A:  Where are you leaving from?
7 U:  San Francisco ((user input))
```

In this example, the user's input always enters the dialog tree at the same place (lines 1, 3, 5 and 7). When the agent asks, "Where do you want to go?" (line 2), it remembers the sequential context by setting *previous node* to "destination request." When the user then says something that is unrecognized, "the Big Apple!" (line 3), the root-level default node is hit (line 4) but the *previous node* variable remains unchanged. When the user then gives a recognizable city, "New York City" (line 5), the following condition is met: city intent and *previous node* equals "destination request." The dialog tree captures "New York City" as the destination and asks, "Where are you leaving from?" (line 6), setting *previous node* now to "origin request." This type of dialog architecture thus uses the context variables for persisting the sequential context. The user cannot lose his or her place in the dialog tree because there is a single point of entry. User and agent can initiate as many "repairs" (covered in Chapters 4 and 6) on a prior utterance without falling off a dialog branch.

3.5 Conclusion

In this chapter we have given a brief introduction to the basics of conversation authoring within the Intent-Entity-Context-Response (IECR) paradigm. With these basics, you can create natural-language-based interactions in many different kinds of styles depending on the granularity of your intents, the length of your utterances, the particular details captured in context variables, and even how you architect the dialog tree. From this introduction you should understand how conversational interfaces in the IECR paradigm fundamentally work. You should understand the differences among *intents, entities,* and *context,* and how dialog logic combines them to recognize a user's input and return a *response*. In the remainder of this book, we will discuss how to design a *conversation-centric* interaction style, in contrast to the other three styles described in Chapter 1, out of these basic elements.

4

Natural Conversation Framework

> *If a person says something that is not understandable, we ask for clarification. If a person says something that we believe to be false, we question and debate. We don't issue a warning signal. We don't beep. We don't give error messages. We ask for more information and engage in mutual dialogue to reach an understanding.*
>
> —Norman [1988, p. 198]

Formal knowledge of natural human conversation can inspire the practice of conversation design and authoring. Natural-language platforms based on the Intent-Entity-Context-Response (IECR) paradigm may provide powerful tools for analyzing strings of natural language, like English or Spanish or Mandarin, but they leave it to designers to figure out how to model a *conversational* interaction with the user. As we saw in Chapter 1, UX designers may create very different kinds of natural-language interaction styles, some of which are not very conversational. Some styles work like web search in which the system does not remember the sequential context across queries, nor recognize user actions other than queries, such as those performed with "say again," "what do you mean?" or "thank you." Other natural-language interaction styles work like graphical or mobile user interfaces in which users select from buttons or menus to submit text-based commands. Conversation is a distinctive form of natural-language use that involves particular methods for taking turns and ordering them into sequences, the persistence of sequential context across turns, and characteristic actions for managing the interaction itself. The UX designer must model these mechanics of conversation primarily through dialog management and context persistence. Neither natural-language processing tools nor conventions for *visual* user interfaces help designers decide how to string bits on language together into naturalistic conversational sequences. Designing for *conversation first* requires a formal, technical knowledge of how human conversation is structured.

Like natural language, *natural conversation* is a complex system to which whole scientific disciplines are devoted. The mechanics of how humans take turns, design them, and organize them sequentially are formally studied in the social sciences, especially in the field of Conversation Analysis (CA). To leverage this literature of observational studies, we have applied the concepts and patterns from CA to the design of conversational agents. While we are not the first to undertake this kind of approach [Luff et al. 1990, Wooffitt et al. 1997], both the natural language processing (NLP) technologies and the field of CA itself have evolved significantly since then. The proliferation of NLP technologies in the market has created a demand for a discipline of conversational UX design. ***As NLP technologies have moved out of the research labs and into the living rooms, desktops, and pockets of millions of people, conventions and standards for how to interact with conversational interfaces are now critical.*** As a result of the relative maturity and standardization of the Intent-Entity-Context-Response (IECR) paradigm, we can now focus on user experience (UX) design and the varieties of conversation patterns, rather than the underlying technologies themselves. The aim of this book is to build upon earlier explorations into the application of Conversation Analysis to the design of human-computer interaction [Luff et al. 1990] by constructing a design framework and methodology for UX designers.

In applying Conversation Analysis (CA) to UX, we have developed a Natural Conversation Framework (NCF) for the design of conversational user interaction and experience that is grounded in observational science [Moore et al. 2016, Moore 2018]. By "conversational" we mean a natural-language interface [Androutsopoulos et al. 1995] that both recognizes common conversational actions and preserves the sequential context of previous turns across future turns, so the agent can respond appropriately. The NCF provides a pattern language [Alexander 1977, Erickson 2000] of generic, reusable conversational UX patterns that are independent of any particular technology platform. The patterns are simplified forms of natural human conversation patterns documented in the CA literature, for example, those of sequence organization or repair [Sacks et al. 1974, Schegloff et al. 1977, Schegloff 2007]. The NCF so far has been implemented only on the IBM Watson Assistant and Dialog services. But in principle it can be implemented on other platforms as well. The NCF is primarily a set of abstract interaction patterns that can be implemented many different ways. These patterns provide a starting point for designers and builders so they do not have to reinvent the basic mechanics of conversational structure. The Natural Conversation Framework [Moore et al. 2016, Moore 2018] consists of four parts: (1) an underlying interaction model; (2) a distinctive content format based on the interaction model; (3) a pattern language for common conversational activities; and (4) a general method for navigating conversational applications. This chapter will describe each component of the framework in turn.

4.1 Interaction Model

The smallest interactive unit of human conversation, in which more than one person participates, is the *sequence*. ***Sequences are general patterns that, like tools or devices, can be used and reused in all kinds of different situations and settings, for all kinds of different purposes.*** Conversation analysts have identified two types of sequences: adjacency pair sequences and storytelling sequences [Schegloff and Sacks 1973, Schegloff 1982, Schegloff 2007]. As we saw in Chapter 2, "adjacency pair" is a formal term for a class of recognizable social action pairs such as *greeting-greeting*, *farewell-farewell, assessment-assessment, inquiry-answer, offer-accept/reject, request-grant/deny, invitation-accept/decline, accusing-admitting/denying,* and many more. When someone does the first part of the pair, it creates an expectation, and an obligation, for someone else to do the second part [Sacks et al. 1974]. While the initiation of a sequence constrains the next speaker's turn, it does not determine it. Sequences are inherently collaborative and are the primary vehicles through which we build up conversations, turn-by-turn, and achieve a wide range of social activities.

When someone initiates an adjacency-pair sequence, the recipient does not always complete it in the next turn, although he or she may be working toward producing the second part of the pair. In other words, adjacency pair sequences are *expandable:* the "base" action pair may be expanded, as needed, with additional pairs for purposes of conversation management [Schegloff 2007]. While base adjacency pairs stand on their own, independently, their expansions do not; they are dependent on the base sequence because they operate on it (Example 4.1).

Example 4.1 Minimal Expansion.

1 U: **can you recommend a Mexican restaurant within**
2 **walking distance?**
3 A: Mario's is located at Fourth and Winchester.
4 U: **thanks!**
5 A: You're welcome! Anything else today?

An utterance like "thanks!" (line 4) does not stand on its own. It is inherently responsive to something else, something prior, in this case the agent's granting (line 3) of the request (line 1). In addition to appreciation, “thanks!” does a particular job in this interaction: it closes the prior sequence. Closing a base adjacency pair is also often done with an “okay,” “great” or other “sequence-closing thirds” [Schegloff 2007]. Simply by saying “thanks!,” a speaker can indicate that the prior utterance, such as the granting of a request or the answering of an inquiry, was adequate and helpful and that the speaker is ready to move on.

Sequence expansions enable speakers to manage the interaction itself. In addition to *closing* a sequence, they may be used for *screening, eliciting, repeating,* or *paraphrasing.* The following excerpt demonstrates each of these expansion types in a single sequence. It is a working example of an interaction with “Alma,” our implementation of the Natural Conversation Framework on the Watson Assistant service (Example 4.2).

Example 4.2 Sequence Expansion Types.

```
 1  U:  do you know about restaurants?                  ((screen))
 2  A:  I know a few places.
 3  U:  a few what?                                     ((repeat))
 4  A:  places.
 5  U:  can you recommend one close by?                 ((base))
 6  A:  What kind of food do you want?                  ((elicit))
 7  U:  well, what are my choices?                      ((elicit))
 8  A:  Around here Mexican, Japanese or Indian.
 9  U:  Mexican
10  A:  Okay. Mario's is on Beach and Main.             ((base))
11  U:  what do you mean?                               ((paraphrase))
12  A:  Mario's is a Mexican restaurant just a
13      couple of blocks west on Chester Street.
14  U:  oh ok, thanks                                   ((close))
15  A:  You're welcome! Anything else?
```

In this example, the user does a preliminary inquiry into the capabilities of the agent (line 1), which accomplishes *screening*. Such preliminaries, or pre-expansions, check conditions upon which the first part of the base sequence (line 5) depend [Schegloff 2007]. If the agent were to respond, “I can look up current and upcoming movies” instead, the user would likely not ask for a restaurant recommendation next (line 5). If the conditions for the first pair part are not met, it will not be produced.

In between the two parts of the base pair, there are two expansions that do *eliciting* (lines 6–9). First, the agent proposes that it needs an additional detail, a cuisine *preference* (line 6), as a condition for granting the user’s request. Second, as a condition for answering the elicitation of a cuisine preference, the user proposes that he needs to know the cuisine *choices* (line 7). Most current chatbot and voice platforms specifically support the first kind of elicitation, the agent-initiated one, and call them simply "slots." An "intent," or user action, can have "slots," meaning bits of information required to fulfill the user intent, for example cuisine preference or distance. If the user does not provide them in the request itself, the agent attempts to elicit them. But this is not the only kind of "slot," or sequence expansion, in natural conversation; it is only an *agent-initiated elicitation*. Current platforms tend not to provide guidance regarding how to create other types of slots, like user-initiated elicitations, preliminaries, or repairs.

The remaining sequence expansions (lines 3 and 11) are examples of what conversation analysts call "repair" [Schegloff et al. 1977]. Repairs consist of a redoing of all or part of a previous utterance, either by the speaker or a recipient, where that utterance poses difficulty in speaking, hearing, or understanding and thereby prevents the conversation from moving forward. In the first case, the user requests a *repeat* of part of the agent's prior response (line 3), namely, the part that came after "a few." This is a partial repeat request technique that elicits a repeat of just the part of the prior turn that the recipient did not hear (line 4). In the second case, the user requests a *paraphrase* of all the agent's prior response (line 11). Repairs of hearing or understanding troubles can come after *any* utterance in a conversation.

The interaction model of the Natural Conversation Framework thus consists of **expandable sequences,** ***like an accordion [Moore et al. 2016, Moore 2018]. Compact sequences are common, but each sequence can be expanded by either party as needed.*** In the example above (Example 4.2), the whole series of utterances is one sequence (all 15 lines), an expanded sequence. The expansions are operations on the base sequence (lines 5 and 10). The expansion types, *screening, repeating,* and *closing,* enable basic coordination in and out of sequences, while the expansion types, *eliciting* and *paraphrasing*, enable the parties to compensate for emergent asymmetries in their knowledge. Taken together, these sequences that operate on other sequences enable *conversation management*. Because they are designed to manage the interaction itself, these actions are unique to conversation and not found in other forms of natural language use, such as those involved in reading a book or searching the web. In order to handle the two-way elicitations and repairs inserted into base sequences, the dialog logic must remember what the user's original inquiry or request was. In other words, *persistent sequential context*, through the use of context variables, is a requirement for this interaction model.

Sequence expansions enable conversational systems to adapt to the particular user on a local, turn-by-turn basis. Where one user may require no sequence expansions, another user may require many. For example, depending on how detailed the user's initial request is, the agent can elicit additional details as needed, rather than accepting only a complete request (Example 4.3).

Example 4.3 Agent Elicitation.

1 U: **can you recommend a restaurant**
2 **within walking distance?**
3 A: What kind of food do you want?
4 U: **Indian**
5 A: Okay. Taj Mahal is located on
6 Third and Union.
7 U: **thanks**
8 A: You're welcome! Anything
9 else I can do for you?

Here the user does not mention the type of cuisine she prefers in her initial request (lines 1–2), so the agent elicits that detail instead (line 3). The user then provides the detail in a separate turn (line 4). This makes the agent flexible and more like a human speaker than a database.

In addition, some users may need more help than others. This may be due to differences in their knowledge or just to idiosyncratic confusions local to the conversation. Sequence expansions enable users to get localized help (Example 4.4).

Example 4.4 User Elaboration Request.

1 U: **can you recommend a good sushi place?**
2 A: Walking distance or short drive?
3 U: **what do you mean?**
4 A: Do you want someplace close or
5 farther away?
6 U: **oh walking**
7 A: Shogun Sushi is located on First
8 and Taylor.
9 U: **thank you very much**
10 A: You're welcome! Is there anything
11 else I can do for you?

In this case, the user requests a paraphrase (line 3) of the agent's request for a distance preference (line 2). Perhaps it is a question he did not expect or perhaps "walking distance" is not a phrase with which he is familiar. The agent then paraphrases its prior question (lines 4–5), which enables the user to understand and answer it (line 6). Rather than designing every utterance of the agent in the simplest elaborated form, which would be long and cumbersome, especially for voice interfaces, sequence expansions enable the agent's initial responses to be concise. This makes the conversation faster and more efficient. Then if a few users encounter trouble responding, understanding, or hearing these more streamlined responses, they can expand the sequence as needed. This is how natural human conversation is organized: with a preference for minimization (Chapter 1; Sacks and Schegloff [1979], Levinson [2007], Moore [2008], Enfield [2013]). ***That is, speakers should try the shortest utterance that they think the recipient can understand first, see if it succeeds, and then expand only if necessary.***

Natural Conversation Understanding

Support for sequence expansion is critical in conversational UX design because one of the distinctive goals of conversation is *mutual understanding*. Accurate information alone is not enough. If the user or the agent cannot understand what the other said, the conversation has failed. Analyzing the user's utterance with natural language understanding tools (NLC and entity extraction) is critical, but it is only the first step. Mutual understanding can only be determined when the recipient responds in "third position" [Schegloff 1992b]. For example, as noted above, if a user makes an inquiry (first position), the agent answers it (second position), and the user closes the sequence with "thanks!" (third position), then this is an implicit indication of mutual understanding. But if the user says, "what do you mean?" in third position, then the *user* encounters trouble in understanding the agent's answer. Or if the user says, "no, I mean X" in third position, then he suggests that the *agent* did not understand the inquiry. And if the user says, "never mind" in third position, then mutual understanding has failed and the user is giving up. ***Sequence expansions provide natural indicators of the participants' state of understanding on a turn-by-turn basis.*** Therefore, mutual understanding cannot be achieved in one turn; it requires interaction, or dialog, and support for sequence expansion. We use the term "natural conversation understanding," then, to refer to sequence-expansion and repair features that enable user and agent to achieve mutual, practical understanding.

4.2 Content Format

The expandable sequence interaction model requires a particular format for the content of the conversational application. As we saw in Chapter 1, content-centric and conversation-centric interaction styles differ significantly in the length of the agent's utterances. While content-centric utterances may be easily adapted from existing Frequently Asked Questions (FAQ) web pages or automatically extracted from documents (e.g., Jeopardy Watson), they are not formatted for conversation-like interaction.

As noted in Chapter 2, natural conversation exhibits a "preference for minimization" [Sacks and Schegloff 1979, Levinson 2007, Moore 2008, Enfield 2013]. Speakers try the shortest construction that they think will be adequate first. Then they expand only as necessary if the recipient indicates trouble in understanding. *This is more efficient than starting with the expanded version.* Following the principle of minimization, conversational content designers should consider the following:

Limit Agent Utterance to a Single Sentence or Less

A single sentence can be long for an utterance in natural conversation. Phrases or single words are common. Keep your agent's utterances to a sentence or less. If multiple sentences are required for the purpose at hand, say to explain something or give background for a request, use the *extended telling* pattern (Chapter 5, A3) to break up the content into smaller chunks.

Break Paragraphs Down into Their Parts

Paragraphs of content consist of multiple parts: answers to multiple inquiries, elaborations, definitions, examples and more. For example, FAQ content typically provides the short answer to the user's question in the first sentence. Subsequent sentences tend to provide more detail in the form of elaborations, definitions or examples, and sometimes to include answers to different but related questions. This may be an efficient way to package content for a web page or document, but it is not conversational. Decompose paragraphs into these component parts so users can navigate the content at a lower level of granularity.

Let Users Control the Level of Detail

Return the short answer to the user in your agent's responses and let the user unpackage more content if desired [Moore 2018, Balata et al. 2018]. The initial response should be fully adequate for the majority of users. However, some users may possess less of the required domain knowledge or encounter situational or one-off troubles in understanding. The user-initiated repair patterns (Chapter 6, B2) enable such a user to elicit paraphrases, examples, or definitions only if he or she requires them. Provide additional detail for the few rather than slowing down the many with unneeded details.

For a demonstration of these guidelines, take the following example of how to design an answer to an inquiry about health insurance coverage. In this example, the user's inquiry is "Am I covered for diabetes screenings?" and its variations (Example 4.5).

Example 4.5 Document-Like Response.

Medicare covers tests to screen for diabetes as well as services and supplies to help treat the disease. Medicare will pay for 100% of its approved amount for the test even before you have met the Part B deductible. You will pay no copay or deductible for these tests if you see doctors who take assignment. Doctors and other health care providers who take assignment cannot charge you more than the Medicare approved amount. Medicare Advantage Plans cover all preventive services the same as Original Medicare. This means Medicare Advantage Plans will not be allowed to charge cost-sharing fees (coinsurances, copays or deductibles) for preventive services that Original Medicare does not charge for as long as you see in-network providers.

In this case, the FAQ-style response contains several distinct but related components. Given that the user's inquiry could be satisfied with a simple "yes" or "no," this document-like response clearly gives more detail than projected by the question. But the same content can be decomposed into separate parts (Example 4.6).

Example 4.6 Conversation-Like Response(s).

Medicare covers tests to screen for diabetes as well as services and supplies to help treat the disease. *[#coverage]*

Medicare will pay for 100% of its approved amount for the test even before you have met the Part B deductible. *[#deductible]*

You will pay no copay or deductible for these tests if you see doctors who take assignment. *[#copay]*

Doctors and other health care providers who take assignment cannot charge you more than the Medicare approved amount. *[#paraphrase request]*

Medicare Advantage Plans cover all preventive services the same as Original Medicare. *[#preventive]*

This means Medicare Advantage Plans will not be allowed to charge cost-sharing fees (coinsurances, copays or deductibles) for preventive services that Original Medicare does not charge for as long as you see in-network providers. *[#paraphrase request]*

This example shows how the same content can be decomposed into four distinct inquiries, or "intents," and paraphrases of the answers to two of those inquiries. We separate the content into these four intents because any of them could reasonably be asked about independently: “Am I covered for diabetes screening?” (#coverage), “What’s my deductible?” (#deductible), “What’s my copay?” (#copay), and “Am I covered for preventive services?” (#preventive). The remaining content can be organized as paraphrases of the copay and preventive inquiries and only displayed to the user if he or she says “what do you mean?” or “I don’t understand” (#paraphrase request). In this way, the content is better formatted for the back-and-forth of conversation.

Breaking down content at a lower level of granularity enables more natural conversation and adaptation of the content to different users (Example 4.7).

Example 4.7 User A.

```
1  U:   am I covered for diabetes screenings?
2  A:   Medicare covers tests to screen for diabetes as well as services
3       and supplies to help treat the disease.
4  U:   thanks
5  A:   You're welcome!
```

In this case, User A is satisfied with a single answer to his inquiry. While the inquiry (line 1) projects a simple yes or no as the expected answer, the agent over-answers by giving a whole sentence (lines 2–3). But explicitly repeating that they are talking about “coverage” and “diabetes” is good practice, given how susceptible today’s natural language classification (NLC) is to misunderstanding, and providing additional information about “services” versus “supplies” is helpful. This is likely an adequate level of detail for most users, which they can indicate by closing the sequence (line 4).

On the other hand, some users may require more detail than others. For them, more content can be delivered through multiple inquiries and repairs to those inquiries (Example 4.8).

Example 4.8 User B.

```
 1  U:   am I covered for diabetes screenings?
 2  A:   Medicare covers tests to screen for diabetes as well as services
 3       and supplies to help treat the disease.
 4  U:   what's the deductible?
 5  A:   Medicare will pay for 100% of its approved amount for the test
 6       even before you have met the Part B deductible.
 7  U:   how about the copay?
 8  A:   You will pay no copay or deductible for these tests if you see
 9       doctors who take assignment.
10  U:   what do you mean?
```

```
11  A:  Doctors and other health care providers who take assignment
12      cannot charge you more than the Medicare approved amount.
13  U:  is preventive care included?
14  A:  Medicare Advantage Plans cover all preventive services the same
15      as Original Medicare.
16  U:  I don't understand
17  A:  This means Medicare Advantage Plans will not be allowed to charge
18      cost-sharing fees (coinsurances, copays or deductibles) for
19      preventive services that Original Medicare does not charge for as
20      long as you see in-network providers.
21  U:  okay thanks
22  A:  You're welcome!
```

In this case, the user does not stop with a single inquiry about coverage (line 1), but continues to unpackage all the content by making three additional inquiries (lines 4, 7, and 13) and by requesting clarification of two of them (lines 10 and 16). In practice, different users will take different paths through the content.

In this scenario, one concern of health insurance providers, and indeed good human agents, is to make sure that enrollees know enough about insurance to ask the right questions. One advantage of the document-like response (Example 4.5) is that it exposes users who may not know the difference between a deductible and a copay, as well as the ones who do, to potential hidden costs that they might not know to ask about. While this approach might be tempting, we recommend against it. There are other ways of educating users about what they can ask or what they can do. For example, in a desktop scenario, a scrolling message bar above the text entry box can be used to display the additional inquiries. Or, in cases in which it is critical to make sure every user receives each bit of information, an alternative sequence type can be used, the extended telling (Chapter 5, A3). With the extended telling pattern, a large amount of content is decomposed into parts and delivered across multiple turns in the conversation (Example 4.9).

Example 4.9 Alternative: Extended Telling.

```
1   U:  tell me about diabetes coverage?
2   A:  Okay. First, Medicare covers tests to screen for diabetes as well
3       as services and supplies to help treat the disease. ((part 1))
4   U:  ok
5   A:  Medicare will pay for 100% of its approved amount for the test
6       even before you have met the Part B deductible. ((part 2))
7   U:  ok
8   A:  You will pay no copay or deductible for these tests if you see
9       doctors who take assignment. ((part 3))
10  U:  what do you mean?
```

```
11  A:  Doctors and other health care providers who take assignment
12      cannot charge you more than the Medicare approved amount.
13      ((part 3 paraphrase))
14  U:  got it
15  A:  Finally, Medicare Advantage Plans cover all preventive services
16      the same as Original Medicare. ((part 4))
17  U:  I don't know what you mean
18  A:  This means Medicare Advantage Plans will not be allowed to charge
19      cost-sharing fees (coinsurances, copays or deductibles) for
20      preventive services that Original Medicare does not charge for as
21      long as you see in-network providers. ((part 4 paraphrase))
22  U:  ok thanks
23  A:  You're welcome!
```

In this case, the user's inquiry (line 1) initiates the extended telling, and the agent begins by delivering the first part (lines 2–3). The agent then waits for the user either to tell the agent to continue or initiate a repair on the first part. When the user produces a "continuer" [Schegloff 1982], like "ok," "yeah," "uh huh," "all right," etc. (line 4), the agent continues to the next part of the telling (lines 5–6). The user then proceeds to produce continuers (lines 7 and 14) or to initiate repairs (lines 10 and 17) until the agent gets to the last part of the telling (lines 15–16). By breaking an extended telling into separate parts, the user can control the pace of the telling and repair his or her understanding of each part before progressing to the next. In this way, a large amount of content can be made conversational.

To support the inquiry sequence (Examples 4.7 and 4.8) and extended telling sequence (Example 4.9), the Natural Conversation Framework requires a particular content format.

0.1a User Utterance, example a
0.1b User Utterance, example b
0.2a Agent Response, variation a
0.2b Agent Response, variation b
0.3 Agent Repeat
0.4 Agent Paraphrase
0.5 Agent Example
0.6a Agent Definition a
0.6b Agent Definition b

In this format, multiple examples of users' equivalent utterances can be given (0.1a, 0.1b, etc.), as well as multiple versions of the agent's initial response utterance (0.2a, 0.2b, etc.), which may be selected consecutively or randomly. The remaining parts are user-initiated repairs (0.3–0.6) on the agent's initial response. While each of these repair patterns will be specified in Chapter 6, examples of the content are given below (Example 4.10).

Example 4.10 Diabetes Coverage Inquiry.

0.1a am I covered for diabetes screenings?
0.1b do I have to pay for a diabetes test?
0.2 Diabetes screenings are covered 100%.
0.3 Diabetes screenings are covered 100%.
0.4 Medicare covers tests to screen for diabetes as well as services and supplies to help treat the disease.
0.5 glucose test, fasting plasma glucose test, FPG, casual glucose test
0.6a diabetes; Diabetes is a group of metabolic disorders in which there are high blood sugar levels over a prolonged period
0.6b screening; A screening is a test for a specific disease or condition.

The content format enables you to capture examples of the users' inquiries (0.1a, 0.1b), the agent's initial response (0.2), a repeat of that response (0.3), a paraphrase of it (0.4), examples of the response if applicable (0.5), and definitions of special terms used in the response (0.6a, 0.6b).

In order to handle an extended telling, the content format is simply extended by adding more parts to the *agent's* response (Example 4.11).

Example 4.11 Diabetes Coverage Extended Telling.

0.1a am I covered for diabetes screenings?
0.1b tell me about diabetes test coverage?
0.2 Diabetes screenings are covered 100%.
0.3 Diabetes screenings are covered 100%.
0.4 Medicare covers tests to screen for diabetes as well as services and supplies to help treat the disease.
0.5 glucose test, fasting plasma glucose test, FPG, casual glucose test
0.6a diabetes; Diabetes is a group of metabolic disorders in which there are high blood sugar levels over a prolonged period
0.6b screening; A screening is a test for a specific disease or condition.

1.2 Medicare will pay for diabetes screenings even if you have not met your deductible.
1.3 Medicare will pay for diabetes screenings even if you have not met your deductible.
1.4 Medicare will pay for 100% of its approved amount for the test even before you have met the Part B deductible.
1.5 Part B deductible
1.6 deductible; A deductible is the amount paid out of pocket by the policy holder before an insurance provider will pay any expenses

In this case, a single set of users' utterances (0.1a, 0.1b) is associated with multiple sets of agent responses (0.2–0.6 and 1.2–1.6), and more sets can be added, extending the agent telling to any number of parts.

From the content format and the interaction model of the Natural Conversation Framework, we can see that *content design* is a critical part of conversational UX design. A conversation-centric interaction style requires relatively fine-grained content compared to that found in books and documents or even in web FAQs. Organizing the content of a virtual agent into this kind of conversational format is still a hard technological problem. Until a computer algorithm can convert the content of existing documents into the conversational format, the conversational UX designer or dedicated content designer must do it.

4.3 Conversational Activity Patterns

The goal of conversational interfaces is not only mutual understanding but also *conversational competence*. Can the automated agent respond appropriately to common actions in conversation? Can the agent *do* conversation? The Natural Conversation Framework provides a UX pattern language, or starting library of UX patterns, that constitute various aspects of conversational competence [Moore et al. 2016, Moore 2018]. The patterns enable a variety of basic social activities and can be configured and adapted to a wide variety of use cases. These UX patterns are directly inspired by patterns of how people naturally talk, as documented in Conversation Analysis (CA). The set is not exhaustive; more patterns can be mined from the CA literature or from the observation of naturally occurring conversations. The NCF currently includes the following 15 types of patterns containing 100 subpatterns (Table4.1). The three classes of patterns are described in more detail below.

The NCF consists of common, reusable UX patterns for delivering the main content of your application, as well as patterns for managing the conversation

Table 4.1 Conversational UX patterns

Conversational Activities	Sequence-Level Management	Conversation-Level Management
A1 Inquiry (User)	B1 Repair (Agent)	C1 Opening (Agent)
A2 Open Request	B2 Repair (User)	C2 Opening (User)
A3 Extended Telling	B3 Extended Repair	C3 Capabilities
A4 Quiz	B4 Sequence Closer	C4 Closing
A5 Inquiry (Agent)	B5 Sequence Abort	C5 Disengaging

itself. The conversational activities include patterns for users to make inquiries (e.g., U: “am I covered for flu shots?”), for users to make open requests (e.g., U: “I’m planning a vacation with my family. Where should I go?”), for agents to give sets of instructions or tell multi-part stories (e.g., A: “First, sit comfortably and breathe slowly.”), for agents to troubleshoot problems (e.g., U: “I’ve been feeling very anxious lately), for agents to quiz users (e.g., A: ”What is the force that results from two solid surfaces sliding against each other?“), and for agents to make open inquiries (e.g., A: ”What did you think of Mario’s?”). Each of these activities is generic and can be used in a wide variety of scenarios. For example, a health insurance agent might use the user inquiry pattern (A1) to answers users’ questions about their health insurance. Or a customer service agent might use the open request pattern (A2) to elicit users’ photocopier problems and diagnose the causes and then use the extended telling pattern (A3) to guide the user through procedures for fixing them. Or an interviewing agent might use the agent inquiry pattern (A5) to ask both closed and open survey questions. Or a tutoring agent might use the extended telling pattern (A3) to present material about the subject of physics and then use the quiz pattern (A4) to test the users’ comprehension. The interaction patterns are generic and independent of all domain-specific content.

While the conversational activity patterns (A) needed for particular use cases may vary depending on the roles of the agent, the conversation management patterns (B and C) are appropriate for almost any use case that is conversational. Any conversation must be opened (C1–2) and most conversations should be closed (C4). If the agent is to be “intelligent,” it should be able to talk about what it can do (C3). After any response by the agent, the user may require a repeat, paraphrase, example, or definition of a term in that response (B2). Sequences should be closed if successful (B4) or aborted if troubles cannot be repaired (B5). And users may disengage from the agent by asking to speak with a human or the agent may disengage when users resort to abusive language (C5). We discuss one conversational activity pattern, the quiz pattern (A4), below.

While most chatbots and voice assistants can handle simple, two-turn sequences, such as question-answer or command-action, few demonstrate the ability to handle the quiz pattern (A4), which requires more than two turns. The quiz pattern is a conversational UX pattern in which the agent asks the *user* questions and evaluates the user's answers. The user can give the correct answer, give an incorrect one, or give up, as well as initiate the standard repeat and paraphrase repairs. This quiz pattern can be used for all kinds of question types and content: pedagogical testing, trivia games, riddles, joke telling, and more (Example 4.12).

Example 4.12 Joke Telling (Alma).

```
 1  U:  tell me a joke
 2  A:  What can you hold without ever touching it?
 3  U:  stocks?
 4  A:  That's not it.
 5  U:  a conversation?
 6  A:  That's right!
 7  U:  tell me another one
 8  A:  Who earns a living by driving his customers away?
 9  U:  the cable company?
10  A:  Nope.
11  U:  you tell me
12  A:  A taxi driver.
13  U:  haha
14  A:  Anything else today?
```

In guessing the punchline of a question-answer joke, as in answering a trivia question, the user can get it right (line 5), get it wrong (lines 3 and 9), or give up (line 11). This is how a question-answer-joke sequence works. In contrast, most of today's chatbots and voice assistants either give humorous one-liners or attempt to collapse the whole joke sequence into a single response (Chapter 1, Examples 1.1 and 1.2), for example, "Why can't you trust atoms? They make up everything!," which gives away the punchline.

In addition to joke telling, the quiz pattern is used by Alma to ask computer history questions, as we can see in this working script (Example 4.13).

Example 4.13 Computer Trivia (Alma).

```
1  U:  ask me a trivia question
2  A:  Who invented the hard disk drive?
3  U:  the what?
4  A:  hard disk drive?
```

5 U: **Microsoft?**
6 A: That's not it.
7 U: **I give up**
8 A: IBM
9 U: **ok**
10 A: Anything else for you?
11 U: **ask me another one**
12 A: What computer beat human champions at
13 Jeopardy in 2011?
14 U: **Watson!**
15 A: That's right!
16 U: **yay**
17 A: Anything else today?

In the quiz pattern, the *agent* initiates an inquiry sequence (line 2) and the user tries to complete it with the correct answer. The user can get the answer correct (line 14), incorrect (line 5), or can abort the sequence (line 7). That's what it means to "quiz" or "test" someone, and it is a common pattern in teaching conversations (Chapter 1).

While the social activities are different, joke telling versus trivia game, the underlying conversational UX patterns are identical. This is one small example of how the same pattern can be reused with different content and for different purposes. It also demonstrates the difference between content and a UX pattern. The aim of the Natural Conversation Framework is to provide a systematic set of dialog patterns, not libraries of industry-specific content, such as intents, entities, questions, or answers. A full specification of these patterns is provided in Chapters 5–7.

4.4 Conversation Navigation

With any computer interface, users must learn how to navigate the space. In command-line interfaces, users learn to navigate directories through cryptic commands. In graphical interfaces, users learn to drag files on a desktop to folders. In web interfaces, users learn to jump from page to page with URLs and hypertext. And in mobile interfaces, users learn to touch the screen, rotate it, and "pinch" the images. But how should users navigate a conversational interface? What are the basic actions that they can always rely on at any point to navigate the conversation space or to get unstuck? Conversational interfaces are more conversational than web search, but they are not yet as competent as a human. ***Fortunately, natural human conversation contains devices for its own management, as we see with sequence expansions.*** We propose a subset of these as basic actions for conversational interface navigation (Table 4.2).

Table 4.2 Basic navigation actions

1. What can you do?	Capability Check
2. What did you say?	Repeat Request
3. What do you mean?	Paraphrase Request
4. Okay/Thanks	Sequence Closer
5. Never mind	Sequence Abort
6. Goodbye	Conversation Closing

Capability Check

Discovering the capabilities of a conversational agent can be challenging because, unlike a graphical interface, there are often no visual elements to click and explore [Norman 1988]. So users should always be able to talk to the agent about what it can do. "What can you do?" is perhaps the most general request for a description of the system's scope and functionality. This is somewhat analogous to the *global help* functionality of a graphical interface. The capability check should give the user enough guidance to use the app or to ask more specific questions about its capabilities, for example, "tell me more about destination recommendations" or "can you help me book a hotel?" or "do you know about restaurants?"

Repeat Request

In voice interfaces, unlike text interfaces, utterances are transient. Once the agent's voice response is done, it is gone. Therefore, users must be able to elicit repeats of all or part of the agent's utterances. "What did you say?" is a natural conversational way to request a full repeat of the prior utterance. In voice interfaces, requesting repeats is like *going back* in visual interfaces. Although repeats are not as crucial in text-based conversational interfaces with their persistent chat histories, virtual agents appear stupid if they cannot understand a repeat request. The ability to repeat its prior utterance is a basic feature of conversational competence. The NCF supports other repeat repairs, including partial repeat requests and hearing checks, but the full repeat request is the most general.

Paraphrase Request

While capability checks provide a kind of global help to the user, paraphrase requests provide *local help* on a turn-level basis. This is more like tooltips in a graphical user interface, accessible by hovering the pointer over a button or icon to get help on a particular feature. Similarly, “What do you mean?” elicits an elaboration or upshot of the prior utterance. In general, the agent’s responses should be concise to increase speed and efficiency, but the paraphrase of that response should be written in simpler language, avoid jargon, and include explicit instruction where necessary. As a result, the elaboration will be longer and more cumbersome than the initial response, but will be easier to understand. (Conversely, if the agent’s initial response on occasion must be long and complex, then the paraphrase should be shorter and to the point, making the upshot of the prior utterance more clear.) This way users can control the level of detail they receive without slowing down the conversation for all users. The NCF supports other paraphrase repairs, including definition requests, example requests, and understanding checks, but the full paraphrase request is the most general.

Sequence Closer

Users should be able to close the current sequence when they receive an adequate response and move on to the next sequence. This is somewhat analogous to *closing* a document in a graphical user interface or a popup window in a web browser. “Okay” or “thanks” are natural conversational ways for the user to signal the completion of the current sequence and invite the agent to move onto any next matters. This can also be a good place to reset context variables that may conflict with subsequent requests. Using “OK” to close a conversational sequence in third position is different from the typical use of “OK” in second position with a graphical dialog box to acknowledge a system prompt.

Sequence Abort

When users fail to elicit an adequate response from the agent, they should be able to abort the current sequence and move on to the next. “Never mind” in a conversation functions somewhat like *escape* in other computer interfaces. It enables the user to give up and move on. This is useful when the user has attempted to repair prior utterances but the agent still fails to understand. Or if the agent is eliciting a series of details for a complex request (e.g., open request pattern) and, upon hearing the required information, the user changes his or her mind, then he or she should be able to cancel the current activity by aborting the sequence. In addition, sequence aborts (e.g., “never mind,” “nvm,” “forget it”) provide implicit feedback that there is some kind of interactional problem, and they should be tracked.

Conversation Closing

As in a human conversation, users should be encouraged to close their interaction with the system. This is similar to *closing* an application or *logging out* of a system. “Goodbye” is the natural way to move to end a conversation. While the agent could respond with an immediate return farewell (“Goodbye!”), instead it should treat the user’s attempt to close the conversation as a “pre-closing” [Schegloff and Sacks 1973]. The pre-closing gives the agent the opportunity to bring up a last topic before returning the “goodbye” and ending the session. This last topic, which can be configured in our Alma implementation, might be an organization appreciation (e.g., “Thank you for contacting IBM!”), a success check sequence (e.g., “Did you find what you were looking for?”), a name request (e.g., “Can I get your name?), a promotion (e.g., ”One last thing, would you like to hear about our current promotion for new customers?), or any other last topic that the designer may want to insert into the conversation before losing the user. In addition, the NCF also recognizes user utterances that are designed as pre-closings rather than farewells, such as “got to go,” “are we done?,” or “anything else?”

After any agent utterance in a conversation, users should be able to do any of the above six actions. At first, users may need to be taught that, unlike other chatbots and voice assistants, your conversational agent will recognize these six basic actions and respond appropriately and *usefully*. Because they are based on human conversation, they should already be familiar to the user and natural to perform as well as easy to remember.

For an example of how these navigation actions might work, see the following exchanges with Alma (Example 4.14).

Example 4.14 Conversation Navigation A.

```
 1  U:  what can you do?
 2  A:  I haven't been trained yet. I can only make small
 3      talk, answer or ask some computer trivia questions
 4      and tell some jokes.
 5  U:  ok who invented the hard disk?
 6  A:  The hard disk was invented by IBM in 1956.
 7  U:  what did you say?
 8  A:  The hard disk was invented by IBM in 1956.
 9  U:  what do you mean?
10  A:  The hard disk was invented in 1956 by engineers
11      at IBM's laboratory in San Jose, California.
12  U:  thanks
13  A:  You're welcome!
```

14 A: Anything else for you?
15 U: **goodbye**
16 A: Did you find what you were looking for?

In version A, the user relies on five of the basic actions to navigate the conversation. At the beginning, he checks the agent's capabilities (lines 1–4) and then does an action within the scope of that response, asking a "computer trivia question" (line 5). In response to the answer, the user then requests a repeat (line 7), followed by a paraphrase request (line 9). The user then closes the trivia-question sequence with an appreciation (line 12) and moves to close the conversation (line 15). Instead of completing the closing sequence, the agent treats it as a pre-closing and brings up a last topic, a success check (line 16). Now contrast the next Example 4.15.

Example 4.15 Conversation Navigation B.

1 U: **show me flights**
2 A: What do you mean?
3 U: **show me flights from SFO to JFK**
4 A: I'm sorry, I still don't understand.
5 U: **never mind**
6 A: Whatever you say, friend.
7 A: Want to know what I can do?
8 U: **yes please**
9 A: Okay.
10 A: I haven't been trained yet. I can only make small
11 talk, answer or ask some computer trivia questions
12 and tell some jokes.

Here the user fails to check the agent's capabilities at the beginning of the conversation and instead initiates a flight request (line 1). This time the agent responds with a default paraphrase request (line 2) to which the user offers an elaboration (line 3). This still fails to enable the agent to understand (line 4) so the user aborts the attempted flight request sequence with "never mind" (line 5). In response, the agent acknowledges the abort and offers to describe its capabilities (line 7), which the user accepts (line 8). The agent then attempts to align with the user's expectations by describing its capabilities (lines 10–12).

Our first implementation of the Natural Conversation Framework was for IBM's *What's in Theaters* web application in 2015. *What's in Theaters* was built on the Watson Dialog service (now shuttered) as a simple demonstration of how to integrate the service with other components into a web application. But it also demonstrates an early version of our conversation navigation method (Example 4.16).

Example 4.16 What's in Theaters 2015.

U: **what can you do?**
A: I can look up current and upcoming movies by Genre or MPAA Rating and show you trailers for them. But I'm afraid I cannot search by number of stars or by movie titles or actor and director names at this time.
U: **ok show me current movies**
A: Are you in the mood for a specific genre?
U: **say again**
A: I said ... Are you in the mood for a specific genre?
U: **what are my choices?**
A: Action, adventure, animated, comedy, crime, documentary, drama, family, fantasy, foreign, historical, horror, music, mystery, romance, science fiction, TV movie, thriller, war movies and western.
U: **science fiction**
A: Do you prefer a certain movie rating?
U: **what do you mean by movie rating?**
A: The rating is a recommendation by the Motion Picture Association of America about the suitability of a movie's content for particular age groups. For example, G is for general audiences, while R is restricted to people 17 and older. Do you prefer a certain movie rating?
U: **I don't care**
A: Good choice, Bob! I found 24 results for Current Science Fiction movies.
((first movie results))
U: **no I mean PG-13**
A: Good choices, Bob! I found 1 results for Current PG-13-rated Science Fiction movies.
((Justice League))
U: **thanks**
A: Is there anything else I can help you with?
U: **where is AMC Saratoga?**
A: I'm sorry, I don't understand. Try What's playing?
U: **what's the address for AMC Saratoga?**
A: I'm afraid I don't understand. I can't recognize movie titles at this time.
U: **never mind**
A: Okay. Whatever you say, Bob! Is there anything else I can help you with?
U: **no**
A: Okay. Did you find what you were looking for, Bob?
U: **yes**

```
44  A:  Great! Goodbye.
45  U:  bye
46  A:  Say anything to continue.
```

In *What's in Theaters,* we can see demonstrations of all six conversation navigation actions: capability check (line 1), repeat request (line 8), paraphrase request (line 17), sequence closing (line 31), sequence aborting (line 38), and conversation closing (line 45). It also supports selected detail elicitations (line 10), no-answer responses (line 23), and self-corrections (line 27). Although the functional scope of *What's in Theaters* was intentionally limited, as a proof-of-concept, it nonetheless demonstrates expandable sequences and conversation navigation.

With this set of six actions, users can navigate the conversation space, manage the conversation with a virtual agent, and get unstuck when they encounter trouble. However, preliminary analyses of users' interactions with *What's in Theaters,* and our other agents built on the Natural Conversation Framework, suggest that users tend not to discover these six basic actions on their own. Users require some guidance in how the NCF navigation method works, through tutorials, on-screen tips, or both. Although the six navigation actions are intuitive with respect to their corresponding actions in human conversation, users must still learn them, as they must with any new genre of computer interface.

4.5 Conclusion

The Natural Conversation Framework [Moore et al. 2016, Moore 2018] for conversational UX design thus contains four components: (1) an *interaction model* based on expandable sequences; (2) a *content format* based on the interaction model; (3) a *pattern language* of reusable patterns for common conversational activities; and (4) a conversation *navigation method*. This design framework adapts models and patterns of human interaction, documented in the field of Conversation Analysis, for the purposes of creating conversational interfaces. The NCF interaction model provides a more natural and interactive pattern than the simple two-turn sequence model of today's chatbots and voice assistants. The content format enables designers to break up document-formatted content into the more bite-sized chunks characteristic of conversational interaction. The pattern language offers a catalog of conversational interaction patterns that can be used as building blocks for a wide range of use cases. And the navigation method consists of a subset of only six conversational actions for getting around the conversation space that are easy for users to remember.

From this chapter you should be able to approach the design of conversational interfaces in a systematic way that puts *conversation first* and that reflects the mechanics of natural conversation. You should also be able to speak with some authority about conversational structure and how it is grounded in conversation science. In the next three chapters, we turn to the pattern language of the Natural Conversation Framework. We define three classes of conversational UX patterns: Conversational Activities (A), Sequence-Level Management (B), and Conversation-Level Management (C), and provide 100 patterns, along with example transcripts of each.

5

Conversational Activity UX Patterns

The previous chapters are intended to give you background knowledge and a conceptual framework for understanding conversational UX design. The next three chapters provide a catalog of conversational UX patterns. They are intended to be used as a reference. In these chapters we refer to these patterns collectively as a "pattern language." The term was coined by Christopher Alexander [1977], a trained architect, who sought an approach to abstracting common problems in architecture and documenting their solutions. The pattern itself describes a problem and then offers a solution, and the culmination of these patterns formulates a language. For example, one of his patterns, "76. House for a Small Family," reads:

> In a house for a small family, it is the relationship between children and adults which is most critical.
>
> Therefore:
>
> Give the house three distinct parts: a realm for parents, a realm for the children, and a common area. Conceive these three realms as roughly similar in size, with the commons the largest.

It is worth noting that these patterns do not explicitly state how to build the solution, but rather provide the components one should consider and include. Many different solutions can implement the same pattern. Alexander's [1977] set of patterns was not simply a "library" but a "language" because the patterns were designed to be used together, to be combined or integrated with one another.

Since the writing of *A Pattern Language* [Alexander 1977], many other design-oriented disciplines have adopted this approach, including user experience design [Erickson 2000]. Most UX designers have conceived of patterns for digital products and services. For example, searching for a product on an e-commerce website presents a common usability problem faced by users. As a result, designers have cataloged "Search" as a pattern and its facets. A pattern should also include examples of how it can be applied in a system as well as any other helpful implementation details.

According to this structure, a design pattern can more simply be thought of as a reusable idea or structure that has been refined over time due to its frequency of occurrence. Through this logic, a design pattern should be broad enough that it is applicable to more than one project or engagement of similar caliber.

5.1 Pattern Language for Conversational UX

Although developed independently, Conversation Analysis (CA) has a kind of pattern language of its own. Conversation-analytic studies discover and specify the formal structures or patterns of human talk-in-interaction, for example the adjacency pair expansion model [Schegloff and Sacks 1973, Schegloff 2007] presented in Chapter 4. These are patterns that speakers in a conversation design themselves or learn and reproduce over time. The goal of CA is to document and understand natural human conversation patterns. ***The main project of this book is to use these patterns of natural human conversation to inform and inspire the design of conversational UX patterns.*** This is not an automatic process. The "rules" and models of natural conversation provided by CA are not the same kind of rules as those found in a programmed system [Button 1990]. They are not deterministic, but rather are representations of *resources* that human speakers use in repeated but nondeterministic ways. In fact, the breaking of conversational rules is often a way to accomplish certain kinds of actions, such as sarcasm, humor, intimacy, and more. Nevertheless, the rules, formal qualitative models, and naturally occurring examples of human talk in CA provide a detailed picture of how human conversation works as a speech-exchange system [Sacks et al. 1974]. As Frohlich and Luff [1990, p. 198] found regarding the design of conversational systems, "the findings of CA spoke directly to problems of turn organization with an authority quite lacking in other accounts." A UX designer can start with this detailed picture and then create interaction patterns that emulate features of human talk, although certainly with limitations and approximations. The resulting systems are *conversation games*, or "simulacra" [Button 1990, p. 68], interactive systems that borrow and adapt mechanics from human conversation, although in simplified forms.

In conversational UX design, user interaction patterns are generic structures that can be reused across use cases to address common issues that occur in conversational interactions, especially service encounter conversations. The remainder of

Table 5.1 Conversational UX patterns

Conversational Activities	Sequence-Level Management	Conversation-Level Management
A1 Inquiry (User)	B1 Repair (Agent)	C1 Opening (Agent)
A2 Open Request	B2 Repair (User)	C2 Opening (User)
A3 Extended Telling	B3 Extended Repair	C3 Capabilities
A4 Quiz	B4 Sequence Closer	C4 Closing
A5 Inquiry (Agent)	B5 Sequence Abort	C5 Disengaging

this chapter will outline five such conversational UX patterns and how they can be combined into more complex activity patterns.

Common Activity Patterns

The Natural Conversation Framework [Moore et al. 2016, Moore 2018] is organized into two kinds of patterns: conversational activities and conversation management. The *conversational activity* patterns handle the main business of the conversation: what services the agent provides and what the user is trying to accomplish. On the other hand, the *conversation management* patterns help the user and agent manage the interaction itself. Conversation management occurs on two levels: the sequence level and the conversation level. This pattern language is therefore organized into three classes of patterns: conversational activity (A), sequence-level management (B), and conversation-level management (C) (Table 5.1; see also Appendix C for a summary). Each of the three classes currently contains five pattern types. In this chapter, we discuss the *conversational activity patterns* (Table 5.1, first column), which can be broken down further into 29 subpatterns (Table 5.2). The five pattern types include ways in which the user (A1) or the agent (A5) can request information from the other, ways in which users can make complex requests in an open way (A2), ways in which the agent can tell stories or give instructions interactively (A3), and ways in which the agent can ask the user questions and evaluate the answers (A4).

For convenience and clarity, each pattern has the same format. First, there is an abstract model of the pattern in the form of a transcript with generic social actions (in capitals). Second, the pattern is illustrated with a particular example from some particular use case. Third, each pattern is named and numbered for easy reference. Subpatterns are numbered by adding decimal places to the numbering of the main

Table 5.2 **Conversational activity patterns**

A1.0 Inquiry (User)
A1.1 Inquiry (User) Confirmation
A1.2 Inquiry (User) Disconfirmation
A1.3 Inquiry (User) Repairs

A2.0 Open Request
A2.1 Open Request Nonverbal
A2.2 Open Request Continuer
A2.3 Open Request Screening
A2.4 Open Request Agent Detail Request
A2.5 Open Request User Detail Request
A2.6 Open Request Summary
A2.7 Warrant Request & Refusal
A2.8 Open Request Summary with Artifacts
A2.9 Open Request Incremental (see B3.1)
A2.10 Open Request Series
A2.11 Open Request Repairs

A3.0 Extended Telling with Repair
A3.1 Extended Telling Abort

A4.0 Quiz
A4.1 Quiz Incorrect
A4.2 Quiz Repairs
A4.3 Quiz User-Initiated
A4.4 Quiz Agent-Initiated

A5.0 Inquiry (Agent)
A5.1 Inquiry (Agent) Open
A5.2 Inquiry (Agent) Hearing Confirmed
A5.3 Inquiry (Agent) Hearing Disconfirmed
A5.4 Inquiry (Agent) Completion Check
A5.5 Inquiry (Agent) Extended Answer

pattern. For example, *A1.3 Inquiry (User) Repairs* is a subpattern of the pattern *A1.0 Inquiry (User)*. In contrast, patterns for conversation management are labeled with a B at the sequence level (Chapter 7) and with a C at the conversation level (Chapter 8).

5.2 A1 Inquiry (User)

The inquiry pattern is perhaps the canonical, conversational sequence pattern. It handles requests for information initiated by the user, for example in question-and-answer (Q&A) scenarios. Like all conversational UX patterns, the pattern itself is separate from the content. In other words, agents may answer inquiries about health insurance, travel policies, movie trivia, small talk, and much more. For example, the agent may answer inquiries about computer history, as in our implementation of the NCF on Watson Assistant (Pattern A1.0, Example 5.1).

Pattern A1.0 Inquiry (User).

```
1  U:  INQUIRY
2  A:  ANSWER
3  U:  SEQUENCE CLOSER
```

Example 5.1 Inquiry (User).

```
1  U:  what computer won Jeopardy?
```

2 A: An IBM computer named Watson.
3 U: **ok**

The user produces an inquiry (line 1), or information request, and our agent, Alma, provides the answer (line 2). The user may then continue to produce a sequence closer (line 3), such as “okay” or “oh” (acknowledgment), “thanks” (appreciation), or “great!” (positive assessment). These indicate that, from the user’s perspective, the sequence is complete.

Unlike *simple Q&A* patterns, the inquiry pattern includes multiple sequence expansions, making it more flexible and “conversational.” For example, if the agent is unsure of what the user means, as indicated by an intent confidence between about 0.8 and 0.2, then the agent asks the user to confirm the intent before giving the answer (Pattern A1.1, Example 5.2).

Pattern A1.1 Inquiry (User) Confirmation.

1 U: INQUIRY
2 A: INQUIRY PARAPHRASE
3 U: CONFIRMATION
4 A: ANSWER
5 U: SEQUENCE CLOSER

Example 5.2 Inquiry (User) Confirmation.

1 U: **what was that Jeopardy computer?**
2 A: You mean, what computer won Jeopardy?
3 U: **yeah**
4 A: An IBM computer named Watson.
5 U: **oh**

When the agent has moderate confidence in its understanding of the user’s inquiry, it produces a paraphrase of the inquiry for the user to confirm (line 2). When the user confirms the paraphrase (line 3), the user’s exact utterance and the confirmed intent can be captured so that the utterance can be added, automatically or manually, to the training examples for the intent in question. In other words, the user can label his or her own utterance for the NLC so that in the future, the agent will respond with the answer (line 4) instead of the paraphrase (as in Pattern A1.0).

On the other hand, if the user disconfirms the candidate answer, the agent then requests a paraphrase from the user (Pattern A1.2, Example 5.3).

Pattern A1.2 Inquiry (User) Disconfirmation.

1 U: INQUIRY
2 A: INQUIRY PARAPHRASE
3 U: DISCONFIRMATION

4 A: PARAPHRASE REQUEST
5 U: PARAPHRASE

Example 5.3 Inquiry (User) Disconfirmation.

1 U: **what computer beat humans at Go?**
2 A: You mean, what computer beat the human chess champion?
3 U: **no**
4 A: I'm sorry, I don't understand your question.

When asked about a computer that won at the game Go (i.e., AlphaGo), the agent classifies this as an inquiry about the computer that won at Chess (i.e., Deep Blue) with moderate confidence. When the user disconfirms the paraphrase (line 3), the utterance can be saved for further scrutiny by the UX designers. The agent then replies as it would for any utterance it fails to recognize, with a paraphrase request (line 4).

In addition to repairs initiated by the agent, the inquiry pattern also enables repairs initiated by the user. While the full set of user-initiated repairs is covered in the next chapter (Patterns B2), here we note that any combination of the nine repairs can be used after the user's answer (Pattern A1.3, Example 5.4). This makes the inquiry-answer sequence further expandable by the user.

Pattern A1.3 Inquiry (User) Repairs.

1 U: INQUIRY
2 A: ANSWER
3 U: REPAIR INITIATOR
4 A: REPAIR
5 U: REPAIR INITIATOR
6 A: REPAIR
7 U: SEQUENCE CLOSER

Example 5.4 Inquiry (User) Repairs.

1 U: **what computer won Jeopardy?**
2 A: An IBM computer named Watson.
3 U: **named what?**
4 A: Watson
5 U: **what do you mean?**
6 A: Scientists at IBM created the computer
that won Jeopardy and named it after the
founder of the company, Thomas J. Watson.
7 U: **oh ok**

After the agent's answer (line 2), instead of closing the sequence, the user can initiate any combination of repeat repairs or paraphrase repairs. For example, the user can elicit a partial repeat of the agent's answer (lines 3–4) or a paraphrase of the full answer (lines 5–6). With paraphrase repairs, the UX designer can break the answer content into smaller chunks, and the user can control the level of detail that he or she receives [Moore 2018, Balata et al. 2018].

5.3 A2 Open Request

The open request pattern enables you to create many kinds of sequences in which the user makes a request, for example in customer service conversations. It handles sets of related requests, or *topics*, that require entity extraction and agent elicitations (a.k.a. "slot filling"). Multiple entities are captured, even when no request type is recognized, and preserved as context across the set of unique requests (i.e., intents). The pattern is also compatible with agent- and user-initiated repair patterns (B1 and B2). With open requests, we also include commands. We distinguish commands from requests only by their grammatical form, not by their sequential form: for example, "recommend a nearby Mexican restaurant?" (command) versus "can you recommend a nearby Mexican restaurant?" (interrogative). Both grammatical forms are included as examples in the same intent. When the open request pattern (A2.0, Example 5.5) contains a full or actionable request in the first turn, it resembles the inquiry pattern (A1.0).

Pattern A2.0 Open Request.

```
1  U:  FULL REQUEST
2  A:  GRANT
3  U:  SEQUENCE CLOSER
4  A:  RECEIPT
```

Example 5.5 Open Request.

```
1  U:  can you recommend a nearby Mexican restaurant?
2  A:  Mario's is located at Beach and Main.
3  U:  thanks
4  A:  You're welcome!
```

Here the user produces a request that is fully specified from the perspective of the agent (line 1), and Alma grants, or fulfills, it in the next turn (line 2). The user then may or may not produce a sequence closer (line 3). The granting of a request may consist of a verbal response (line 2) or nonverbal one, such as the execution of an action (e.g., turning on lights) or the display of items in a graphical display (e.g., cards for restaurant options).

When the granting of a request, or execution of a command, is primarily nonverbal, the agent should still acknowledge the request in second position (Pattern A2.1, Example 5.6).

Pattern A2.1 Open Request Nonverbal.

```
1  U:  FULL REQUEST
2  A:  ACKNOWLEDGMENT <NONVERBAL>
3  U:  SEQUENCE CLOSER
4  A:  RECEIPT
```

Example 5.6 Open Request Nonverbal.

1 U: **please turn on the lights**
2 A: Okay. ((lights come on))
3 U: **thank you**
4 A: You're welcome!

In this case, the user produces a command (line 1), to which the primary response from the agent should be a nonverbal action that executes the command, that is, turning on the lights (line 2). It is good practice in many use cases to produce a verbal acknowledgment, such as “Okay,” “Of course” or “Done!” (line 2), in order to show the user that the command was heard and understood. The user may or may not then produce a sequence closer (line 3).

We call such requests “open” because they do not need to be produced in a single turn. If the user mentions some known entities but not a recognizable intent, the agent will nonetheless extract and “remember” the entities. We implement this by creating an action type, or intent, for *detail giving*, which includes multiple combinations of related entities but no clear request for action regarding them. For example, a user may produce many kinds of utterances that mention particular types of foods but not formulate a request for a restaurant recommendation, restaurant reservation, delivery order, recipe, etc. In response to such a user utterance, the agent will invite the user to continue or will ask a related question (Pattern A2.2, Example 5.7).

Pattern A2.2 Open Request Continuer.

```
1  U:  DETAIL GIVING
2  A:  CONTINUER
3  U:  REQUEST
4  A:  GRANT
5  U:  SEQUENCE CLOSER
6  A:  RECEIPT
```

Example 5.7 Open Request Continuer.

1 U: **I hear the fish tacos around here are fantastic!**
2 A: Mhmm.

3 U: **can you recommend a place?**
4 A: Mario's is located at Beach and Main.
5 U: **thanks**
6 A: You're welcome!

Here, the user relays a positive assessment of “fish tacos” (line 1) but does not make an explicit request regarding them. The agent nonetheless captures the entities mentioned; in this case, “fish tacos” is captured as “Mexican” under *cuisine* and “around here” is captured as “walking distance” under *distance*. But because the agent does not yet know the user’s intent regarding these entities, such as requesting a restaurant recommendation, restaurant hours, restaurant name, recipe, etc., it simply produces an “mhmm,” inviting the user to continue (line 2). If the user then produces a recognizable request (line 3), the agent will “remember” the details just mentioned and not ask for them again (line 4). If the user does not continue on to produce a recognizable request, the agent will request a paraphrase (B1). In a way, the detail-giving intent is a kind of *anti-intent:* it provides details as context, but not the user’s social action intent regarding them. In such cases, the concept of “intents” as singular utterances breaks down.

In addition to unrecognized actions that contain known details, users may preface their request with “preliminaries” [Schegloff 2007] that perform a screening function. These are usually yes/no questions that check some condition upon which the request depends. This pattern enables users to check to see if the agent knows about a particular domain before making a request in that domain (Pattern A2.3, Example 5.8).

Pattern A2.3 Open Request Screening.

1 U: PRE-REQUEST
2 A: GO-AHEAD
3 U: FULL REQUEST
4 A: GRANT
5 U: SEQUENCE CLOSER
6 A: RECEIPT

Example 5.8 Open Request Screening.

1 U: **do you know about restaurants?**
2 A: I know a few places.
3 U: **can you recommend a nearby Mexican restaurant?**
4 A: Mario's is located at Beach and Main.
5 U: **thanks**
6 A: You're welcome!

Before requesting a restaurant recommendation (line 3), the user can check to see if this agent even knows anything about restaurants (line 1). If this kind of capability check receives a confirmation (line 2), the user will continue on to make the request. The confirmation in this case is a kind of "go-ahead" response that tells the recipient to proceed from the preliminary to the base first pair part [Schegloff 2007, p. 30]. But if it is disconfirmed, the user will not; for example, if in this case the agent replied, "I can look up movie show times for you" or "I'm afraid I don't know about restaurants."

Once the user has made a request that the agent recognizes, it may need to elicit additional details from the user in order to grant or fulfill the request (Pattern A2.4, Example 5.9).

Pattern A2.4 Open Request Agent Detail Request.

1 U: `PARTIAL REQUEST`
2 A: `DETAIL REQUEST`
3 U: `DETAIL`
4 A: `GRANT`
5 U: `SEQUENCE CLOSER`
6 A: `RECEIPT`

Example 5.9 Open Request Agent Detail Request.

1 U: **`can you recommend a Mexican restaurant?`**
2 A: `Walking distance or car ride?`
3 U: **`walking distance`**
4 A: `Mario's is located at Beach and Main.`
5 U: **`thanks`**
6 A: `You're welcome!`

For Alma, obtaining a cuisine preference and a distance preference from the user is required in order to make a restaurant recommendation. Although the user happens to mention only one in the original request (line 1), the agent can attempt to elicit the other one (line 2). If additional details are required, the agent can attempt to elicit those in turn before granting the request (line 4). If the user does not know or does not care about a particular required detail, the agent can assume a default value, for example, "walking distance." Such agent-initiated detail requests are often called "slots" and this pattern "slot filling," in most natural-language platforms.

While agents can elicit required details from users in order to grant a request, so can users. Sometimes users require additional details in order to answer the agent's detail requests (Pattern A2.5, Example 5.10).

Pattern A2.5 Open Request User Detail Request.

1 U: `PARTIAL REQUEST`

2 A: DETAIL REQUEST
3 U: DETAIL REQUEST
4 A: DETAIL
5 U: DETAIL
6 A: GRANT
7 U: SEQUENCE CLOSER
8 A: RECEIPT

Example 5.10 Open Request User Detail Request.

1 U: **what's a good restaurant nearby?**
2 A: What kind of food do you want?
3 U: **what are my choices?**
4 A: Around here, Mexican, Japanese or Indian.
5 U: **Mexican**
6 A: Mario's is located at Beach and Main.
7 U: **thanks**
8 A: You're welcome!

In this pattern, in response to the agent's attempt to elicit a required detail (line 2), the user can likewise attempt to elicit details (line 3). While Alma needs a cuisine type in order to recommend a restaurant, the user may need to know the relevant choices. Because, like humans, the agent and the user always possess asymmetric knowledge, either one of them may need to elicit a detail in order to complete the request sequence. For this reason, we use the terms "user-initiated detail requests" and "agent-initiated detail requests" instead of simply "slots."

While the granting of a request may consist simply of providing information or acknowledging an action taken, sometimes it may require a summary of the details of the request. When multiple details are elicited from the user and it is important to get them right, it is good practice for the agent to confirm those details before recording them or relaying them. This is a typical pattern in placing an order, for example, for fast food, but may also be appropriate for constructing a record (Pattern A2.6, Example 5.11).

Pattern A2.6 Open Request Summary.

1 U: PARTIAL REQUEST
2 A: DETAIL REQUEST
3 U: DETAIL
4 A: DETAIL REQUEST
5 U: DETAIL
6 A: DETAIL REQUEST
7 U: DETAIL
8 A: REQUEST SUMMARY

```
 9  U:  SUMMARY CONFIRM
10  A:  GRANT
11  U:  SEQUENCE CLOSER
12  A:  RECEIPT
```

Example 5.11 Open Request Summary.

```
 1  U:  I need to register
 2  A:  Okay. What's your name?
 3  U:  Dave Bowman
 4  A:  What's your job title?
 5  U:  astronaut
 6  A:  What is your organizational affiliation?
 7  U:  NASA
 8  A:  So Dave Bowman, Astronaut at NASA. Is that correct?
 9  U:  yes
10  A:  Okay. You're all set!
11  U:  thank you
12  A:  You're welcome!
```

In this case, the agent elicits several details from the user (lines 2–7). The agent then summarizes the request by repeating those details back to the user (line 8) before taking further action. If the user confirms them (line 9), the agent grants the request, in this case by recording the details (line 10). If the user disconfirms the summary, the agent should elicit the correct details, for example, “What do I need to change?” Or the user may do a self-correction, “I mean, Commander” (see below).

Anytime the agent attempts to elicit details, users should be given the option of refusing to give the information or to know why the agent is asking (Pattern A2.7, Example 5.12).

Pattern A2.7 Warrant Request & Refusal.

```
1  A:  DETAIL REQUEST
2  U:  WARRANT REQUEST
3  A:  WARRANT
4  U:  REFUSAL
5  A:  ACKNOWLEDGMENT. <NEXT ACTION>
```

Example 5.12 Warrant Request & Refusal.

```
1  A:  What's your job title?
2  U:  why do you need to know?
3  A:  Job title enables us to know the range of professions of our
        participants.
4  U:  none of your business
5  A:  No problem. What is your organizational affiliation?
```

In this case, the agent attempts to elicit the user's job title during the course of registering for a conference or convention (line 1). Instead of giving it, the user requests a justification for asking (line 2). The agent then provides a warrant, or reason for asking (line 3). The user then refuses to give the detail (line 4). Such warrants and refusals are important for protecting user's private information, as well as providing flexibility in this conversational sequence.

A similar request summary pattern may be used when retrieving artifacts for the user. By "artifact" we mean some visual items, such as an information card, bulleted list, or possibly physical objects. In such cases, it is good practice for the agent to repeat back the details of the request when presenting the artifacts. This gives the user an opportunity to catch any miscommunications before inspecting the items (Pattern A2.8, Example 5.13).

Pattern A2.8 Open Request Summary with Artifacts.

1 U: PARTIAL REQUEST
2 A: DETAIL REQUEST
3 U: DETAIL
4 A: DETAIL REQUEST
5 U: DETAIL
6 A: DETAIL REQUEST
7 U: DETAIL
8 A: HOLD REQUEST
9 A: GRANT
10 <OFFER OF ARTIFACTS>
11 U: SEQUENCE CLOSER
12 A: RECEIPT

Example 5.13 Open Request Summary with Artifacts.

1 U: **I want to book a flight**
2 A: Okay. Where are do you want to go?
3 U: **Kona, Hawaii**
4 A: Where are you leaving from?
5 U: **SJC**
6 A: When would you like to go?
7 U: **June 17-30**
8 A: One moment please...
9 A: Okay. Here are flights from SJC to KOA on June 17-30.
10 ((visual flight cards))
11 U: **thank you**
12 A: You're welcome!

With this pattern, after eliciting the required details from the user (lines 2–7), the agent may ask the user to hold or wait while the system retrieves the items, which may require accessing backend APIs of uncertain response time (line 8). When the backend returns the items and the visual user interface displays the artifacts (line 10), the agent summarizes the request by repeating the details, “SJC to KOA on June 17-30” (line 9). This gives the user the opportunity to catch errors or ambiguities and correct them.

One of the advantages of a conversational interface is that users can use minimal turn designs by building off earlier turns. An example of this is the incremental request (Pattern A2.9, examples 14).

Pattern A2.9 Open Request Incremental (same as B3.1).

1 U: FULL REQUEST
2 A: GRANT
3 U: INCREMENTAL REQUEST
4 A: GRANT
5 U: SEQUENCE CLOSER
6 A: RECEIPT

Example 5.14 Open Request Incremental.

1 U: **can you recommend a nearby Mexican restaurant?**
2 A: Mario's is located at Beach and Main.
3 U: **how about Indian?**
4 A: Taj Mahal is on Third and Union.
5 U: **thanks**
6 A: You're welcome!

After the user has made a request (line 1) and the agent has granted it (line 2), the user may make a second one by changing the request or adding an increment to the prior request, “how about Indian?” (line 3). Because the user gives a new entity but not a new intent, the agent assumes that the prior intent (request for a restaurant recommendation) and entities (nearby) are still relevant and simply updates the new entity value (replacing Mexican cuisine with Indian cuisine). In this way, the user can make the new, related request without repeating him or herself.

In addition to supporting incremental requests, open requests should also support series of related requests. As with the incremental request, recognizing requests in a series requires persisting the sequential context and assuming it (Pattern A2.10, Example 5.15).

Pattern A2.10 Open Request Series.

1 U: FULL REQUEST
2 A: GRANT
3 U: RELATED REQUEST
4 A: GRANT

```
5  U:  SEQUENCE CLOSER
6  A:  RECEIPT
```

Example 5.15 Open Request Series.

```
1  U:  can you recommend a nearby Mexican restaurant?
2  A:  Mario's is located at Beach and Main.
3  U:  when is it open?
4  A:  Mario's is open from 11:00am to 2:00am.
5  U:  thanks
6  A:  You're welcome!
```

After the user has made a successful request (lines 1–2), the agent remembers the prior intent and entities. So if the user changes the intent, from "restaurant recommendation request" to "business hours request" (line 3), the agent assumes the user is still talking about the prior entity, "Mario's restaurant" (line 4). By assuming sequential context, the agent can handle natural uses of anaphora or "pro-terms" [Sacks and Schegloff 1979]. On the other hand, the user can indicate that the next request is unrelated by simply giving a full request instead of a partial one, for example "when does Taj Mahal open?"

Open request series require a little more context design than standalone requests or "slotted intents." They require a context variable for the topic that is relevant across multiple requests or intents. For example, if the agent can answer multiple inquiries related to particular restaurants' name, location, and business hours, then a context variable for restaurant name or ID is needed. When the user makes one kind of inquiry about the restaurant, say "Mario's," it is stored to a variable for conversational *topic*. If the user then makes subsequent inquiry using a "locally subsequent reference form" [Schegloff 1996], the prior restaurant is assumed as the intended topic.

Similarly if a virtual travel agent can handle multiple intents related to "flight requests," "destination requests," and "attraction requests," then a request-type context variable is needed. Once the user mentions a flight-related intent, "I want to book a flight," this context variable is set to "flights." Then as the user makes related inquiries, origin airport, travel dates, cheapest places to go, best times to go, etc., the agent assumes that the user is still making a flight request until otherwise indicated. The request-type and topic context variables thereby enable the agent to maintain context across multiple user actions or intents.

Just as with inquiries, users can initiate repair on the responses to open requests (Pattern A2.11, Example 5.16). This includes any of the nine user-initiated repair types discussed in the next chapter (Patterns B2).

Pattern A2.11 Open Request Repairs.

1 U: FULL REQUEST
2 A: GRANT
3 U: REPAIR INITIATOR
4 A: REPAIR
5 U: SEQUENCE CLOSER
6 A: RECEIPT

Example 5.16 Open Request Repairs.

1 U: **can you recommend a nearby Mexican restaurant?**
2 A: Mario's is located at Beach and Main.
3 U: **what do you mean?**
4 A: The Mexican restaurant, Mario's, is located at
the intersection of Beach Boulevard and Main Street.
5 U: **thanks**
6 A: You're welcome!

In response to the agent's granting or rejecting (line 2) of a user's request (line 1), the user can request full or partial repeats or paraphrases of the agent's response (line 3). For example, if the agent gives a name and intersection of a restaurant (line 2), the user can elicit a full paraphrase (line 3) from the agent, which may consist of a more expanded version (line 4). In this way, the agent can offer a concise granting of requests initially, and only users who require an elaborated version need elicit one.

Taking these open request patterns together, we offer one more example from the conference registration scenario (Example 5.17). Refer to the patterns above to try to map them to the example.

Example 5.17 Open Request.

1 A: What can I do for you?
2 U: **I just came over from the other hotel. I'm with NASA?**
3 A: NASA, okay.
4 U: **can I register here?**
5 A: Okay. What's your name?
6 U: **Dave Bowman**
7 A: What's your job title?
8 U: **astronaut**
9 A: What kind of meal would you like?
10 U: **what are my choices?**
11 A: Beef, fish or vegan.
12 U: **beef**
13 A: So Dave Bowman, Astronaut at NASA and the beef meal.
14 Is that correct?
15 U: **no, I mean fish**

16 A: So Dave Bowman, Astronaut at NASA and the fish meal.
17 Is that correct?
18 U: **yes**
19 A: Okay. You're all set!
20 U: **will there be a reception?**
21 A: I'm afraid NASA is not having a reception.
22 U: **ok thanks**
23 A: You're welcome!

5.4 A3 Extended Telling

While the action pair, or "adjacency pair," is perhaps the most common sequence type in natural conversation, *storytelling* is another common sequence type [Schegloff 2007]. Storytelling and instruction giving are conversational activities that typically require more than one turn to complete. Such extended tellings may be initiated by the user with requests like, "tell me about Key West," "tell me all the things you can do," "tell me a story," "how do I meditate?," "how do I do that?," etc. The telling is then produced interactionally in the following way: the teller gives the first part of the telling and then waits for an indication from the recipient to continue the telling (Pattern A3.0). Such "continuers" [Schegloff 1982] may be verbal tokens, like "uh-huh," "mhmm," "okay," "all right," "yeah," or gestures, like a head nod (lines 3 and 7).

Pattern A3.0 Extended Telling with Repair.

1 U: STORY REQUEST/INSTRUCTION INVITATION
2 A: PART/STEP 1
3 U: CONTINUER/PAUSE
4 A: PART/STEP 2
5 U: REPAIR INITIATOR
6 A: REPAIR
7 U: CONTINUER/PAUSE
8 A: PART/STEP 3
9 U: SEQUENCE CLOSER
10 A: RECEIPT

By waiting for a continuer after each story part or instruction, the teller gives the recipient the opportunity to initiate repair on it, if necessary, as the telling progresses. This is critical, because failing to hear or understand one part of the story can impact the recipient's ability to understand the whole story. The continuer tells the teller that the recipient heard and understood the prior part and is ready to go on. To speed up the pace of the telling, the teller can simply pause, leaving a slot for a repair initiator but not waiting for an actual verbal continuer to be produced. When giving instructions, a prior instruction often impacts the ability to perform a subsequent instruction. The recipient may even attempt to perform the instructions during the telling itself, so the *continuer* enables the recipient to control the pace of the instruction giving as needed. Consider the following Example 5.18.

Example 5.18 Extended Telling with Repair.

1 U: **how do I meditate?**
2 A: First, sit comfortably and breathe slowly.
3 U: **ok**
4 A: Next, count each in breath and out breath until you get to ten.
Then repeat.
5 U: **what do you mean?**
6 A: As you breathe in, out, in, count one, two, three all the way
to ten.
7 U: **all right**
8 A: Finally, continue counting your breaths for ten minutes.
9 U: **thanks**
10 A: You're welcome!

The agent's telling consists of three instructions (lines 2, 4, and 8). After giving the first instruction, Alma waits for a response from the user. Here the user gives a continuer, "ok" (line 3), and then the agent gives the next instruction (line 4). However, in response to this instruction, the user initiates repair instead of bidding the agent to continue. The user requests a paraphrase of the prior instruction (line 5), which the agent provides (line 6). The user then invites the agent to continue the instruction giving with a continuer, "all right" (line 7). When the agent provides the final instruction (line 8), the user closes the sequence with an appreciation (line 9). In this case, because the agent waits for the continuer before giving the next instruction, the user can control the pace of the sequence. While the extended telling pattern (A3.0) represents only three story parts or instructions, it can of course be extended to any number of parts.

One way to implement the extended telling pattern (A3) on a dialog platform is to set each part of the telling to numbered context variables when the telling is initiated. Then, when the user produces a continuer, or the backend sends a timed input after a pause, the parts are advanced by one. For example, if *part 1* is "Open your Settings app" and *part 2* is "Find and open the 'General' tab," a context variable, *current part*, is initially set to *part 1*. Then, when the user produces a continuer, *current part* is updated to the value of *part 2*, and so forth.

Because a story or a set of instructions is an interactionally long activity, the recipient may want to abort it in the middle (Pattern A3.1, Example 5.19).

Pattern A3.1 Extended Telling Abort.

1 U: STORY/INSTRUCTION INVITATION
2 A: PART/STEP 1
3 U: CONTINUER/PAUSE
4 A: PART/STEP 2
5 U: REPAIR INITIATOR
6 A: REPAIR
7 A: PART/STEP 3
8 U: COMPLAINT/ABORT REQUEST
9 A: ABORT OFFER
10 U: ABORT CONFIRM
11 A: ACKNOWLEDGMENT

Example 5.19 Extended Telling Abort.

1 U: **tell me a story**
2 A: Once upon a time, there was a little girl named Goldilocks. She went for a walk in the forest. Pretty soon, she came upon a house. She knocked and, when no one answered, she walked right in.
3 ((pause))
4 A: At the table in the kitchen, there were three bowls of porridge. Goldilocks was hungry. She tasted the porridge from the first bowl. This porridge is too hot!, she exclaimed.
5 U: **what do you mean by porridge?**
6 A: Oh, sorry! Porridge is a dish consisting of oatmeal or another grain boiled in water or milk. Sometimes called hot cereal.
7 U: **ok**
8 A: So, she tasted the porridge from the second bowl. This porridge is too cold, she said. So, she tasted the last bowl of porridge. Ahhh, this porridge is just right, she said happily and she ate it all up.
9 U: **this is boring**
10 A: Shall I stop?
11 U: **yes please**
12 A: Okay.

In this example, the agent breaks the relatively long story of *Goldilocks and the Three Bears* into smaller chunks of two or three sentences (lines 2, 4, and 8). After a pause (line 3), the agent continues with the story without receiving an explicit continuer from the user (line 4). The user nonetheless had the opportunity to initiate repair if necessary. The ideal length of the pause should be in the neighborhood of one half to two seconds depending on the standard pace of the interface and whether it is voice or text based. And if the user starts typing or speaking during that pause, the agent should wait and respond to that action instead of automatically continuing.

In response to the second part of the story, the user initiates repair before the end of the pause (line 5). In this case, she requests a definition of a term used in that part of the story, "porridge." The agent provides a definition (line 6) and waits. When the user produces a continuer, "ok" (line 7), the agent produces the next part of the story (line 8). However, after this part, the user produces a complaint about the current activity (line 9) rather than a continuer or a repair initiator. The agent then offers to abort the current activity (line 10), which the user confirms (line 11). In some scenarios, the user should be able to control not only the pace of the story but also telling itself.

We recommend using the extended telling pattern (A3) whenever your agent must communicate a lot of content. Breaking the content up into multiple parts makes it more conversational and less like a document or a monologue.

5.5 A4 Quiz

While the user inquiry pattern involves the user asking the agent questions, the *quiz* pattern involves the agent asking the user questions and evaluating the answers. The user can get the answer right, wrong, initiate a repair, or give up. The quiz is a basic conversation pattern that is used in pedagogical interactions, as well as for fun. The ordinary question-answer joke uses the quiz pattern, as do trivia games. In its simplest form, the quiz pattern is initiated by the user, who then answers the question correctly (Pattern A4.0, Example 5.20).

Pattern A4.0 Quiz.

```
1  A:  QUESTION
2  U:  CORRECT ANSWER
3  A:  CONFIRMATION
```

Example 5.20 Quiz.

```
1  A:  Why can't you trust atoms?
2  U:  they make up everything
3  A:  That's right!
```

In this pattern, the agent asks the user a question-answer joke (line 1). The user produces the correct answer, or punchline (line 2), and the agent confirms it (line 3). This pattern contrasts with joke telling in many of today's chatbots and voice assistants in which the agent produces both the joke and the punchline in the same turn (Chapter 1, Examples 1.1 and 1.2). The quiz pattern is more natural and complex in that it must preserve the context of the joke-telling turn (line 1), while evaluating the user's response for at least four kinds of actions: correct answer, incorrect answer, repair initiator(s), or sequence abort.

In the activity of joke telling, it is perhaps more common for the recipient of the joke to get the punchline wrong, to refuse to guess it, or to give up. In these cases, the agent produces the answer or punchline (Pattern A4.1, Example 5.21).

Pattern A4.1 Quiz Incorrect.

1 A: QUESTION
2 U: INCORRECT ANSWER
3 A: DISCONFIRMATION
4 U: REFUSAL/ABORT
5 A: CORRECT ANSWER
6 U: SEQUENCE CLOSER

Example 5.21 Quiz Incorrect.

1 A: `Why can't you trust atoms?`
2 U: **`you can't see them?`**
3 A: `Wrong! Try again.`
4 U: **`you tell me`**
5 A: `They make up everything.`
6 U: **`haha`**

If the user produces a response that does not match a correct answer, a repair type, or an attempt to refuse to answer or to abort the sequence (line 2), the agent will interpret it as an incorrect answer (line 3). Of course, sometimes this assumption will be wrong. If the user does some other kind of action, say, a positive assessment, "I love this joke!," the agent will incorrectly treat it as a candidate answer or punchline. This is a limitation of the quiz pattern, but the list of recognizable non-answer actions can always be expanded. After the agent disconfirms the answer (line 3), the user can take another guess or give up (line 4). When the user gives up or refuses to guess, the agent then produces the answer or punchline to the joke (line 5). For joke content, the agent should recognize laughter tokens, e.g, "haha," "heh heh," "lol," etc., as sequence closers (line 6), as well as positive or negative assessments, e.g., "good one!" or "that's not funny."

The quiz pattern also supports user-initiated repair (Pattern B2, Chapter 6). The standard set should be used for pedagogical or trivia quizzes; however, paraphrase repairs may not be relevant in the case of jokes, since the sense of a joke is often dependent on the exact wording of the question (Pattern A4.2, Example 5.22).

Pattern A4.2 Quiz Repairs.

1 A: QUESTION
2 U: REPAIR INITIATOR
3 A: REPAIR

```
4  U:  CORRECT ANSWER
5  A:  CONFIRMATION
```

Example 5.22 Quiz Repairs.

```
1  A:  Why can't you trust atoms?
2  U:  atoms?
3  A:  Yes.
4  U:  they make up everything?
5  A:  That's right!
```

When telling jokes through a voice interface, the user may require a repeat of all or part of the joke question. For example, if the user initiates a *hearing check* by repeating a part of the joke question (line 2), the agent should be able to confirm the user's candidate hearing (line 3). Definition requests may also be relevant, for example, "what's an atom?"

To build this pattern in dialog, you can set a context variable to a value such as "quiz" and then use that as a standalone condition, for example, *previous node* equals "quiz." Now anything the user says will go down this branch. Then reset this variable in the event that the user gets the right answer or gives up. You can insert repairs like *repeat request*, "what did you say?" within the branch, but hitting those nodes should not reset *previous node*. As a result any utterance by the user that is not a correct answer, an abort, or an inserted action will be treated as an incorrect answer.

Like most conversational activities, quizzes can be initiated by the user or the agent. When the user requests a joke or trivia question, for example "tell me a joke," "ask me a trivia question," "teach me about human speech," "quiz me," etc., the agent can produce the joke in the next turn (line 2, Pattern A4.3, Example 5.23).

Pattern A4.3 Quiz User-Initiated.

```
1  U:  QUIZ REQUEST
2  A:  QUESTION
3  U:  CORRECT ANSWER
4  A:  CONFIRMATION
```

Example 5.23 Quiz User-Initiated.

```
1  U:  tell me a joke
2  A:  What can you hold without ever touching it?
3  U:  a conversation?
4  A:  That's right!
```

On the other hand, when agents initiate a quiz activity, they may offer to start the quiz in response to other actions. For example, the agent may offer to tell a joke if the user indicates he or she is having a bad day, or the agent may offer to start a trivia game if the user says he or she is bored (Pattern A4.4, Example 5.24).

Pattern A4.4 Quiz Agent-Initiated.

```
1  A:  QUIZ OFFER
2  U:  ACCEPTANCE
3  A:  QUESTION
4  U:  CORRECT ANSWER
5  A:  CONFIRMATION
```

Example 5.24 Quiz Agent-Initiated.

```
1  A:  Would you like to play a computer trivia game?
2  U:  sure
3  A:  Okay. What computer beat human champions at Jeopardy?
4  U:  Watson
5  A:  You got it!
```

If the agent offers to start a quiz (line 1), it only proceeds to the first question (line 3) if the user accepts the offer (line 2). The quiz question is not asked if the user declines. Because quiz mode changes the agent's frame of reference for interpreting user utterances, it should not be entered unless the user agrees to the offer and thus knows that he or she is taking a quiz or guessing a punchline.

5.6 A5 Inquiry (Agent)

In addition to quizzing the user, agents may simply ask users questions without evaluating the answers for correctness. This *agent inquiry* pattern may involve closed or open inquiries and different third-position checks by the agent to determine if the agent heard the user correctly or if the user has more to say. The agent inquiry pattern is especially useful for interviewing users or eliciting their opinions, where you wish to elicit and preserve the exact words of the user.

Closed inquiries are those in which a preferred, or required, answer format is projected by the question itself. The simplest of these is the yes/no question, which through its grammatical format projects a binary choice for the answer; for example, "Would you recommend Mario's to a friend?" Other types of closed inquiries project wider, but still restricted, ranges of answers: "What is your favorite cuisine?" "What year were you born?" or "What is your city of residence?" Because the range of preferred answers is restricted, they can be recognized using entities, for example, cuisines or countries, the past one hundred years, or cities across the country or the world. And if the answers can be recognized through entities, the agent can act on them during the course of the conversation, for example, taking different next actions depending on the year of birth or city of residence given. Finally, closed inquiries may explicitly mention the choices or range of choices preferred for the user's answer (Pattern A5.0, Example 5.25).

Pattern A5.0 Inquiry (Agent).

`1 A: CLOSED INQUIRY`
`2 U: ANSWER`
`3 A: SEQUENCE CLOSER`
`4 U: RECEIPT`

Example 5.25 Inquiry (Agent).

`1 A: On a scale from one to ten, how would you rate Mario's?`
`2 U:` **`nine`**
`3 A: Thank you.`
`4 U:` **`you're welcome`**

In this case the agent gives the range of preferred answers in the inquiry itself, "scale from one to ten" (line 1). The inquiry invites the user to choose a number from this range. If the user does so (line 2), then the system can recognize and validate the answer through a *number* entity or even a regular expression. The agent then closes the sequence by thanking the user for a valid answer (line 3). Of course, as we have seen, users may produce actions other than a preferred answer in the next turn to the inquiry, including repairs (B2–B3) or detail requests (A2.5), as well as invalid answers. Response types that you do not anticipate will be met with the agent's paraphrase request (B1). In other words, even "closed" inquiries have an inescapable openness to them.

In addition to closed inquiries, agents may ask *open* ones, which do not attempt to restrict the answer to any particular format. In online surveys, open inquiries are answered by freeform typing in a text box. In telephone surveys, they are accepted verbatim without being probed until they fit into a given category. Open inquiries commonly take the form of "What do you think about X?" or "Tell me about Y?" (Pattern A5.1, Example 5.26).

Pattern A5.1 Inquiry (Agent) Open.

`1 A: OPEN INQUIRY`
`2 U: ANSWER`
`3 A: SEQUENCE CLOSER`
`4 U: RECEIPT`

Example 5.26 Inquiry (Agent) Open.

`1 A: What did you think of Marios's?`
`2 U:` **`the food was really good, and I loved the cantina atmosphere!`**

3 A: Thank you.
4 U: **you're welcome**

Here the agent asks an open question about the user's opinion of a particular restaurant (line 1). The user replies with a lengthy evaluation (line 2). In such cases, you will likely want to capture the user's response verbatim and preserve it for future use, for example, as a restaurant review or as raw survey data. This can be done by capturing the text from a system variable, such as *user input*, setting it to a context variable, such as *user review*, and saving it on the back end.

When using voice, it is wise to check the agent's "hearing" of the user's answer before recording. To repair errors in the application's speech to text, the agent can repeat what it captured to the user and invite correction (Pattern A5.2, Example 5.27).

Pattern A5.2 Inquiry (Agent) Hearing Confirmed.

1 A: OPEN INQUIRY
2 U: ANSWER
3 A: HEARING CHECK
4 U: CONFIRMATION
5 A: SEQUENCE CLOSER
6 U: RECEIPT

Example 5.27 Inquiry (Agent) Hearing Confirmed.

1 A: What did you think of Marios's?
2 U: **the food was really good, and I loved the cantina atmosphere!**
3 A: You said: the food was really good, and I loved the cantina
atmosphere! Is that correct?
4 U: **yep**
5 A: Thank you.
6 U: **you're welcome**

Instead of closing the sequence in third position, the agent may repeat the user's answer and request confirmation that it is correct (line 3). To do this, the user's response is captured to a context variable, *user review*, and inserted into the following type of frame: "You said: *user review.* Is that correct?" Through this kind of check, the user can hear what the agent's speech-to-text component recognized and confirm that it was correct or at least adequate for the purpose at hand (line 4).

Alternatively, the user can indicate that speech recognition was problematic by disconfirming the hearing check (Pattern A5.3, Example 5.28).

Pattern A5.3 Inquiry (Agent) Hearing Disconfirmed.

1 A: OPEN INQUIRY

2 U: ANSWER
3 A: HEARING CHECK
4 U: DISCONFIRMATION
5 A: REPEAT REQUEST
6 U: ANSWER
7 A: HEARING CHECK
8 U: CONFIRMATION
9 A: SEQUENCE CLOSER
10 U: RECEIPT

Example 5.28 Inquiry (Agent) Hearing Disconfirmed.

1 A: What did you think of Marios's?
2 U: **the food was really good and I loved the cantina atmosphere**
3 A: You said: the food was really good and I loved the can I atmosphere. Is that correct?
4 U: **no**
5 A: What did you say?
6 U: **food was good and I loved the CANTINA atmosphere**
7 A: You said: food was good and I loved the cantina atmosphere. Is that correct?
8 U: **yes**
9 A: Thank you.
10 U: **you're welcome**

If the system's speech recognition makes an error, such as "can I" instead of "cantina" (line 3), the user can indicate a problem by disconfirming the repeat (line 4). To repair the "mishearing," the agent requests that the user repeat the original answer (line 5). The pattern then repeats until the user confirms the check (lines 8). A limitation of this pattern is that the user must repeat the whole answer instead of correcting only the error(s). In an alternative pattern, the agent does not repeat the user's answer the second time but simply closes the sequence (line 7). The system preserves both the original answer (line 2) and the correction (line 6) for a human to make sense of later. For example, the user's partial correction, "no, CANTINA," gets set to *user correction* and saved along with the original *user review*, "the food was really good and I loved the can I atmosphere."

Instead of asking users to confirm its "hearing" of their prior answers (A5.2-A5.3), which is irrelevant with typed utterances, your agent may alternatively ask users if they have more to say. In order to elicit longer responses, such as product or restaurant reviews, your agent can invite them to add to their initial response (Pattern A5.4, Example 5.29).

Pattern A5.4 Inquiry (Agent) Completion Check.

1 A: OPEN INQUIRY
2 U: ANSWER
3 A: COMPLETION CHECK
4 U: NEXT ANSWER PART
5 A: COMPLETION CHECK
6 U: DISCONFIRMATION
7 A: REPEAT. RECORDING REQUEST
8 U: CONFIRMATION
9 A: SEQUENCE CLOSER
10 U: RECEIPT

Example 5.29 Inquiry (Agent) Completion Check.

1 A: What did you think of Marios's?
2 U: **the food was really good!**
3 A: Anything else?
4 U: **and I loved the cantina atmosphere**
5 A: Anything else?
6 U: **no**
7 A: You said: the food was really good
and I loved the cantina atmosphere
Shall I record your review?
8 U: **yes**
9 A: Thank you.
10 U: **you're welcome**

After the user's initial response to the open inquiry (line 2), the agent does not immediately repeat it but asks if the user has "anything else" to say (line 3). The user can add another part to the answer (line 4) or decline to extend it (line 6). In this pattern, the agent then repeats the complete, multi-part answer and asks if the user if it should be recorded (line 7). Simply by asking users if they are done answering the open question, the agent suggests that more is preferred.

Another variation of the open inquiry pattern resembles an extended telling (A3) but on the part of the user. Instead of repeating the user's answer (A5.2-A5.3) or asking if the user has "anything else" to say (A5.4), the agent simply produces a "continuer" [Schegloff 1982]; pattern A5.5, Example 5.30).

Pattern A5.5 Inquiry (Agent) Extended Answer.

1 A: OPEN INQUIRY
2 U: ANSWER
3 A: CONTINUER
4 U: NEXT ANSWER PART
5 A: CONTINUER

```
6  U:  END
7  A:  SEQUENCE CLOSER
8  U:  RECEIPT
```

Example 5.30 Inquiry (Agent) Extended Answer.

```
1  A:  What did you think of Marios's?
2  U:  the food was really good!
3  A:  Uh-huh.
4  U:  and I loved the cantina atmosphere
5  A:  Mhmm.
6  U:  that's it
7  A:  Thank you.
8  U:  you're welcome
```

In response to the user's answer to the open inquiry (line 2), the agent says "Uh-huh" or "Mhmm" (lines 3 and 5), inviting the user to continue. The user may produce another part to the answer (line 4) or indicate that he or she is done (line 6). The entire answer must be saved, either by setting each part to a separate context variable, for example, *answer part 1, answer part 2,* etc., or by appending each string to the previous one in a single variable, for example, *extended answer*. This variation of the open inquiry (A5) pattern is streamlined for eliciting more talk from the user rather than validating the accuracy of the speech recognition.

We see then that user inquiries, open requests, extended tellings, quizzes, and agent inquiries are generic conversational activities with recognizable interaction patterns. Through these patterns, all manner of different content can be communicated for all kinds of different purposes. Designing a conversation space for a particular use case fundamentally involves figuring out which of these activity patterns will be the primary means of delivering content.

5.7 Combining Patterns

In addition to supporting different generic conversational activities, the above patterns can also be combined to support more complex activities. ***Inquiries, open requests, extended tellings, and quizzes may be components of larger activities, such as troubleshooting, tutoring, order taking, lecturing, interviewing, and more.*** To demonstrate how content patterns can be combined, we provide examples of *troubleshooting* and *tutoring*.

Troubleshooting

Troubleshooting is a generic activity found in many kinds of telephone or online service encounters. In this activity, the customer or service seeker presents a problem to the representative or agent of the organization and requests a solution. The agent may need to probe the user's situation with a series of diagnostic questions in order to determine the likely cause of the problem and the appropriate solution. The agent may then talk the user through a set of instructions for solving the problem. Troubleshooting can therefore be supported by combining the open request (A2) and extended telling (A3) patterns. The following example demonstrates how troubleshooting might be used in the context of psychological counseling (Pattern A2+A3, Example 5.31).

Pattern A2+A3 Troubleshooting User-Initiated.

1 U: PRE-REQUEST/PRE-PROBLEM REPORT
2 A: GO-AHEAD RESPONSE
3 U: REQUEST/PROBLEM REPORT
4 A: DETAIL REQUEST
5 U: DETAIL
6 A: DETAIL REQUEST
7 U: (REPAIR INITIATOR)
8 A: (REPAIR)
9 U: DETAIL
10 A: DETAIL REQUEST
11 U: DETAIL
12 A: GRANT/SOLUTION PROPOSAL
13 U: INSTRUCTION REQUEST
14 A: PART/STEP 1
15 U: CONTINUER
16 A: PART/STEP 2
17 U: CONTINUER
18 A: PART/STEP 3
19 U: CONTINUER
20 A: PART/STEP 4
21 U: SEQUENCE CLOSER
22 A: RECEIPT

Example 5.31 Troubleshooting User-Initiated.

1 U: **i've got a problem**
2 A: What's your problem?
3 U: **I've been feeling really anxious lately**
4 A: Okay, anxiety. Anything else?
5 U: **i've been pretty tired**
6 A: Has anything bad happened recently?
7 U: **what do you mean?**
8 A: Could a recent negative experience be causing the anxiety?
9 U: **no**

10 A: Is your anxiety severe?
11 U: **not really**
12 A: Okay. Then perhaps you should try meditating.
13 U: **how do I do that?**
14 A: First, sit comfortably and breathe slowly.
15 U: **what do you mean?**
16 A: Just sit and relax.
17 U: **oh ok**
18 A: Next, count each in breath and out breath until you get to ten. Then repeat.
19 U: **all right**
20 A: Finally, continue counting your breaths for ten minutes.
21 U: **thanks**
22 A: You're welcome! Is there anything else I can do for you?

This example of troubleshooting consists of the open request pattern (lines 1–12) combined with the extended telling pattern (lines 13–22). First, the user does a generic preliminary to his request or problem report (line 1), and the agent gives a "go-ahead" response (line 2). The user then makes a request by reporting a problem, "I've been feeling really anxious lately" (line 3), and thereby inviting the agent to offer a solution.

But before offering a solution, the agent asks some questions to elicit diagnostic details (lines 4–11). It asks if the user has additional symptoms (line 4), asks about recent trigger events (line 6), of which the user requests a paraphrase (line 7), and finally asks about the severity of the problem (line 10). If user mentions anxiety-related symptoms and answers "no" to each of the screening questions, the agent will identify meditation as a recommended solution (line 12). At this point, the open request pattern is complete.

The user can then initiate the extended telling pattern by asking for guidance on how to meditate, "how do I do that?" (line 13). Notice that the context of the problem, "anxiety," and the solution, "meditation," are preserved across patterns. The agent then gives the user the first instruction for meditating (line 14), which, like any of the other instructions, the user can seek to repair (lines 15–16). After receiving continuers (line 17 and 19), the agent moves on to giving the next instructions (line 18 and 20). The final instruction is prefaced with "Finally . . . " (line 20), and the user moves to close the troubleshooting activity with an appreciation (line 21).

Tutoring

The main conversational activity patterns can also be combined in a variety of ways to support pedagogical activities, such as tutoring. For example, the *extended telling* pattern can be combined with the *quiz* pattern to teach the user new material and test his or her retention (Pattern A3+A4, Example 5.32).

Pattern A3+A4 Tutoring.

1 U: STORY REQUEST
2 A: ACKNOWLEDGMENT
3 A: PART/STEP 1
4 U: CONTINUER
5 A: PART/STEP 2
6 U: (REPAIR INITIATOR)
7 A: (REPAIR)
8 U: CONTINUER
9 A: PART/STEP 3
10 U: CONTINUER
11 A: PART/STEP 4
12 U: (REPAIR INITIATOR)
13 A: (REPAIR)
14 U: SEQUENCE CLOSER
15 A: QUIZ OFFER
16 U: ACCEPTANCE
17 A: QUESTION 1
18 U: CORRECT ANSWER 1
19 A: CONFIRMATION
20 A: QUESTION 2
21 U: CORRECT ANSWER 2
22 A: CONFIRMATION
23 A: QUESTION
24 U: INCORRECT ANSWER
25 A: DISCONFIRMATION
26 U: CORRECT ANSWER
27 A: CONFIRMATION
28 A: ACTIVITY CLOSER

Example 5.32 Tutoring.

1 U: **teach me something about human speech**
2 A: Okay.
3 A: First, one of the most distinctive features of a human voice is its pitch.
4 U: **uh-huh**
5 A: The pitch of a human voice is how high or low it sounds to the human ear.
6 U: **can you give an example?**
7 A: Men generally have lower voices than women and children
8 U: **oh ok**

9 A: The pitch of a human voice is determined by the tension in the vocal cords and the flow of air out of the lungs.
10 U: **ok**
11 A: Finally, pitch can be measured by measuring the fundamental frequency of the voice signal, the unit of which is hertz.
12 U: **what do you mean by fundamental frequency?**
13 A: Human voices contain multiple frequencies. The fundamental frequency is the lowest one.
14 U: **ok**
15 A: Would you like to take a quiz?
16 U: **sure**
17 A: What do we call our perception of the highness or lowness of a human voice?
18 U: **pitch**
19 A: Correct!
20 A: What is a factor in determining the pitch of a human voice?
21 U: **vocal cord tension**
22 A: That's right!
23 A: How can pitch be measured?
24 U: **frequency**
25 A: Incorrect. Try again.
26 U: **I mean, fundamental frequency**
27 A: That's correct!
28 A: You answered all three questions correctly!

This example of teaching consists of an extended telling (lines 1–14) followed by a quiz (lines 15–28). The user initiates the teaching activity by requesting teaching (line 1). The agent responds to the request by beginning an extended telling about "pitch in human speech" (line 3). As the user produces continuers (lines 4, 8, and 10), the agent proceeds to the next part of the telling (lines 5, 9, and 11). And as the user initiates repairs (lines 6 and 12), the agent provides them (lines 7 and 13). After a sequence closer (line 14), the agent offers to quiz the user on the material just presented (line 15). Upon acceptance of the offer (line 16), the agent quizzes the user about the topic of the extended telling, "What do we call our perception of the highness or lowness of a human voice?" (line 17). The user answers the quiz question (line 18) and the agent confirms that it is correct (line 19). The agent then continues on to ask two additional questions (lines 20 and 23). The user gets the first of these correct (lines 21–22), but initially gets the answer to the second one wrong (lines 24–25). In this version of the quiz pattern, the agent invites the user to try again (line 25) until he or she gets the right answer or gives up. The user then repairs the incorrect answer (line 26) and then gets it correct (line 27). The agent then closes the activity with a tally of correct answers (line 28).

5.8 Conclusion

In this chapter, we introduced the concept of a “pattern language” [Alexander 1977, Erickson 2000] and then demonstrated one for conversational UX that is grounded in conversation science. While we define the entire pattern language over three chapters (5, 6, and 7), we began with patterns for common conversational activities: user inquiry, open request, extended telling, quiz and agent inquiry. The patterns are generic so they can be applied to any kind of conversational content or industry use case. Each of the five patterns includes multiple subpatterns and working examples of each. In addition, we demonstrated how these five patterns can be combined to create larger patterns for more complex activities, like troubleshooting, tutoring, or interviewing.

From this chapter you should begin to be able to think about conversation spaces formally and in terms of their sequential structure. This involves identifying the generic interaction patterns underlying the particular content and building up complex conversational activities out of the five basic patterns outlined above (or identifying when a new pattern is needed). In addition, you should understand the technical issues entailed in implementing the different patterns, for example, matching a user inquiry (A1) with an intent versus capturing a user answer to an open inquiry from the agent (A5.1). While the conversational activity patterns (A) above make up the primary activities or purpose of a conversation, patterns that enable the *management* (B and C) of the social interaction itself will be presented in the next two chapters.

6

Sequence Management UX Patterns

While the previous chapter addressed conversational UX patterns for modeling a range of conversational activities, this chapter covers additional patterns that are critical for supporting and managing those conversational activities. These patterns are needed because the particular form of natural language use is a "conversation" and not something else, such as "web search" or "news article." They form a fundamental part of basic *conversational competence*.

In chatbot development, designers sometimes talk about the need to build some "chit chat" into the agent's capabilities. The term "chit chat" is typically used as a gloss for any interaction that is not the main activity of the application, such as Q&A or following commands. This is inadequate for a systematic conversational UX design framework. ***In common usage, "chit chat" means inconsequential talk. But much talk that may appear on the surface to be inconsequential in fact serves a variety of functions in managing the conversation itself.*** As we saw in the previous chapter, saying "uh-huh" in response to part of a story functions to show the teller of the story that you are listening and ready for more. While common sense may tempt us to dismiss things like "uh-huh" as chit chat, as conversational UX designers, we must understand the work that such utterances perform in conversation, in order to build it.

While certain actions involved in conversation management are mischaracterized as inconsequential, there is a one kind of activity that is properly called "chit chat" or "small talk." In human conversation, we sometimes make inquiries into inconsequential topics, such as the weather, a recent sports game, recent movies, personal hobbies, etc., in order to start a conversation with a stranger and pass short intervals of time. In such cases of "small talk," which typically take the form of inquiries and stories, it is the establishment of the social connection itself that is more important than the topics or the content. And generic, neutral small-talk topics can be useful in striking up a conversation and keeping it going until the strangers can find a more interesting and substantive topic to explore. In the Natural Conversation Framework [Moore et al. 2016, Moore 2018]), we use the term "small talk," which is interchangeable with "chit chat," to refer to generic, throwaway topics around which inquiries (A1) or extended tellings (A3) may be built, and in Chapter 8 we discuss ways of using small talk to help establish a *persona* for the virtual agent. On the other hand, we use the term "conversation management" for utterances that function to coordinate the exchange of turns in the interaction.

As we saw in the previous chapter, the conversational activity patterns (A) in the Natural Conversation Framework are designed to handle the main business of the interaction: what services the application provides and what the user is trying to accomplish. In contrast, the conversation management patterns enable user and agent to manage the interaction itself. We break the conversation management patterns further down into sequence-level management and conversation-level management. In this chapter, we explore *sequence-level management* (B), which is used in human conversation to manage the other activities (A and C). What should the user or agent do when a particular sequence is done? What should they do when one party encounters a trouble in speaking, hearing, or understanding? Sequence-level management provides resources for this.

The sequence-level management patterns in our Natural Conversation Framework can be broken down into five pattern types and 32 subpatterns (Table 6.1). The five pattern types include ways in which the agent (B1) and the user (B2) can repair troubles in hearing or understanding immediately prior utterances or earlier utterances (B3), as well as ways of ending conversational sequences either by closing them (B4) or by aborting them (B5).

6.1 B1 Repair (Agent)

As highlighted in previous chapters, "repair" is set of practices in natural human conversation through which speakers locally manage troubles in speaking, hearing, and understanding [Schegloff et al. 1977, Schegloff 1992b, Schegloff 2007]). ***In Conversation Analysis, "repair" is a technical term referring specifically to the repeating and paraphrasing of prior utterances or parts of them.*** It is not used to refer to every kind of problem resolution in a conversation; for example, it is not used in the context of "repairing a speaker's knowledge" or "repairing the participants' relationship." Repairs can be initiated by either the speaker of the trouble-source turn or one of its recipients [Schegloff et al. 1977]).

Table 6.1 **Sequence-level management patterns**

B1.1.0 Understanding Check (Agent)
B1.2.0 Paraphrase Request (Agent)
B1.2.1 Subsequent Paraphrase Requests
B1.2.2 Agent Continuer
B1.2.3 No Answer Account
B1.2.4 Repeat Acknowledgment

B2.1.0 Repeat Request
B2.1.1 Repeat Request Default
B2.2.0 Partial Repeat Request
B2.3.0 Hearing Check Confirmed
B2.3.1 Hearing Check Corrected
B2.4.0 Paraphrase Request
B2.4.1 Paraphrase Request Default
B2.5.0 Definition Request Repair
B2.5.1 Definition Request Default
B2.5.2 Definition Request Standalone
B2.6.0 Example Request
B2.6.1 Example Request Default
B2.7.0 Understanding Check Confirmed
B2.7.1 Understanding Check Disconfirmed

B3.1.0 Self-Correction (same as A2.9)
B3.1.1 Misunderstanding Report
B3.1.2 Other Hearing Check & Mishearing Report
B3.2.0 Other-Correction

B4.0 Sequence Closer (helped)
B4.1 Sequence Closer (not helped)
B4.2 Sequence Closer Appreciation (helped)
B4.3 Sequence Closer Appreciation (not helped)
B4.4 Sequence Closer (repaired)

B5.0 Sequence Abort
B5.1 Sequence Abort Capability Offer Accepted
B5.2 Sequence Abort Capability Offer Declined

B1.1 Understanding Check (Agent)

We have already seen a pattern in which the agent initiates repair on the user's inquiry by proposing a candidate inquiry for the user to confirm (Pattern A1.1). This pattern can also be implemented globally for any of your agent's recognized actions or intents (Pattern B1.1.0, Example 6.1).

Pattern B1.1.0 Understanding Check (Agent).

```
U: <UTTERANCE>
A: <PARAPHRASE>
U: CONFIRMATION
A: <RESPONSE>
U: SEQUENCE CLOSER
```

Example 6.1 Understanding Check (Agent).

U: **whats shakin bruh**
A: You mean, what's new?
U: **yeah**
A: Nothing really. I don't get out much.
U: **ha!**

For any user intent, the agent can propose a candidate paraphrase when the input receives a moderate confidence score, say, 0.8 to 0.2. Here the user's utterance (line 1) gets a moderate confidence, and so the agent offers a paraphrase of what the user said for the user to confirm (line 2). In order to do this, you must create a candidate paraphrase for each user action, or intent, that you want the agent to be able to repair like this. We set this paraphrase to the *confirm* context variable and include it in the response set, along with repairs (see below), for the *news check* intent. If the user confirms the candidate paraphrase (line 3), the agent gives the response for the corresponding intent (line 4). These user inputs and intent names can be collected and used for retraining the intent. If the user does not confirm the paraphrase, the agent requests that the user paraphrase the original utterance (Pattern B1.2).

B1.2 Paraphrase Request (Agent)

Sometimes your virtual agent will not recognize what the user said. Because information-retrieval-based, natural-language platforms require you to create all the actions, or intents, that the agent can recognize in advance, there will always be many user inputs that you did not anticipate. For example, you may set your minimum intent confidence threshold to 0.2. Any user utterance scoring below this threshold, with its highest confidence score, will be treated as unrecognized and un-repairable. To manage this kind of trouble in understanding, the agent should request that the user paraphrase the prior utterance (Pattern B1.2.0, Example 6.2).

Pattern B1.2.0 Paraphrase Request (Agent).

```
1  A:  <ANY UTTERANCE>
2  U:  <UNKNOWN UTTERANCE>
3  A:  INITIAL PARAPHRASE REQUEST
4  U:  <KNOWN PARAPHRASE>
5  A:  <APPROPRIATE RESPONSE>
```

Example 6.2 Paraphrase Request (Agent).

```
1  A:  Hello. How are you?
2  U:  bitchin'!
3  A:  What do you mean?
4  U:  I'm great!
5  A:  That's great!
```

In this pattern, if the agent fails to recognize the user's utterance (line 2), that is, the input fails to surpass the confidence threshold or to match any other conditions, the agent requests a paraphrase of that utterance (line 3). Instead of explicitly instructing the user to rephrase the utterance, as is commonly done, the agent uses a more conversational paraphrase request, "What do you mean?," "I don't understand," "Please clarify," and the like. If the user's action type, or intent, is known to the system but the particular formulation is not, then paraphrasing the action in a clearer form may enable the agent to recognize it [Candello and Pinhanez 2018, p. 150]). Here, the agent can understand many positive assessments, but it has not been trained on the variation "bitchin'!" (line 2). When the user paraphrases the positive assessment in a more common form, "I'm great!" (line 4), the agent can now recognize it (line 5). Unrecognized utterances like this and their paraphrases can then be collected and added to the same intent as training examples so the agent can learn.

If the agent's request fails to elicit a recognizable paraphrase, the agent should indicate continued understanding trouble and give tips to help the user (Pattern B1.2.1, Example 6.3).

Pattern B1.2.1 Subsequent Paraphrase Requests.

```
1  A:  <ANY UTTERANCE>
2  U:  <UNKNOWN UTTERANCE>
3  A:  INITIAL PARAPHRASE REQUEST
4  U:  <UNKNOWN UTTERANCE>
5  A:  SUBSEQUENT PARAPHRASE REQUEST
```

Example 6.3 Subsequent Paraphrase Requests.

```
1  A:  How can I help you?
2  U:  what's in theaters?
3  A:  What do you mean?
4  U:  what movies are currently playing?
5  A:  I'm sorry, I still don't understand.
```

In this case, the user produces an unrecognizable utterance (line 2), the agent requests a paraphrase (line 3) and the user rephrases it (line 4), but the agent fails to recognize the paraphrase (line 5). This may happen when the action type, or intent, of the user's utterance is not one the agent knows. In this subsequent paraphrase request (line 5), the agent can also give tips about what it can do or offer to give a capability description. Putting such tips or offers in the *initial* paraphrase request can get tedious, especially in a voice-based interface.

Sometimes the intent of a user utterance may be unknown, but the agent may still be able to recognize an entity within it. In such cases of partial understanding, the agent has the option of bidding the user to continue. A "continuer" [Schegloff 1982] is not a repair of a prior utterance because it is not *redoing* the problematic utterance, but it is an alternative strategy for managing an understanding trouble (Pattern B1.2.2, Example 6.4).

Pattern B1.2.2 Agent Continuer.

1 U: <UNKNOWN INTENT, KNOWN ENTITY>
2 A: (REPEAT ENTITY) CONTINUER
3 U: <RECOGNIZED UTTERANCE>
4 A: <APPROPRIATE RESPONSE>

Example 6.4 Agent Continuer.

1 U: **I just came over from the other hotel.**
I'm with NASA?
2 A: NASA, mm-hmm.
3 U: **can I register here?**
4 A: Okay. What's your name?

In this pattern, the agent cannot recognize the intent of the user's utterance (line 1), but it can recognize an entity within it. The agent then responds by repeating the recognized entity, "NASA," followed by a continuer, "mm-hmm" (line 2). Variations of the continuer are "uh-huh," "yeah," "okay," and more. The agent's continuer invites the user to say more, rather than to paraphrase what he or she already said. If the user continues on to produce a recognizable intent (line 3), the agent responds to that intent (line 4), remembering the entity from the previous turn as a context variable. If the user produces an unrecognizable utterance instead, the utterance will default to the agent paraphrase request (B1.2).

Perhaps the lowest level of understanding is recognizing the form of the user's utterance, but nothing else; for example, recognizing that it starts with a question word, such as, *who, what, where, when, why, how, is, are, do* or *does*, and is therefore likely an interrogative, or inquiry (Pattern B1.2.3, Example 6.5).

Pattern B1.2.3 No Answer Account.

1 U: <UNKNOWN INQUIRY>
2 A: NO ANSWER ACCOUNT
3 U: SEQUENCE CLOSER

Example 6.5 No-Answer Account.

1 U: **what is the capital of California?**
2 A: I don't know.
3 U: **ok**

Here the agent fails to recognize the intent of the user's utterance or any entities in it (line 1), but detects that it starts with "what" and therefore is likely a question or inquiry. The agent then responds with an account for why it cannot answer, "I don't know" (line 2), which would not be an appropriate response to other unknown utterances, like "I want to book an Uber" or "July 10." Not knowing is the most common account for why one is not answering a question. Refusing (e.g., "I don't want to say" or "none of your business") is less common due to its potential for rudeness.

Repeating an utterance is a conversational action that usually occurs in environments of speaking, hearing, or understanding trouble. As we will see in the next section, users may request a repeat of the agent's turn if he or she had trouble hearing it. ***However, virtual agents tend not to notice repeats by users. One reason for this is no doubt that detecting a repeat cannot be done through intent-matching alone: the user's current input must be compared with prior inputs.*** But the dialog tree can do this. For example, every input can be compared with the prior input. If they are equal, the user is repeating him or herself verbatim. One kind of response to a user's repeat is simply to acknowledge it (Pattern B1.2.4, Example 6.6).

Pattern B1.2.4 Repeat Acknowledgment.

```
1  U:  <UNKNOWN UTTERANCE>
2  A:  INITIAL PARAPHRASE REQUEST
3  U:  <FIRST REPEAT>
4  A:  REPEAT ACKNOWLEDGMENT
5  U:  <SECOND REPEAT>
6  A:  <REPEAT>
```

Example 6.6 Repeated Unknown Utterance.

```
1  U:  what's in theaters?
2  A:  What do you mean?
3  U:  what's in theaters?
4  A:  I know, I heard you!
5  U:  what's in theaters?
6  A:  what's in theaters?
```

In this pattern, the user produces an unrecognized utterance (line 1), to which the agent responds with a paraphrase request (line 2, pattern B1.2). The user then repeats the unknown utterance (line 3). While such a repeat in human conversation may insist that the recipient try again at responding to the original utterance, in interactions with virtual agents, they typically elicit the same response. Paraphrases are much more useful. To show a bit more intelligence than simply repeating the prior response and to discourage users' repeats of unrecognized utterances, the agent can acknowledge the user's repeat (line 4). If the user continues repeating (line 5) in this pattern, the agent parrots the user (line 6). While this version displays a bit of attitude on the agent's part, this could be softened by changing the repeat acknowledgment (line 4) and agent repeat (line 6) to a request for a paraphrase, "What you mean by 'what's in theaters?"'

These are just a few of the ways in which the agent can elicit a paraphrase from the user when it fails to understand what he or she is saying. We turn now to ways that users can elicit repeats and paraphrases of what the agent says.

6.2 B2 Repair (User)

While virtual agents sometimes need the user to paraphrase or repeat what he or she said, users likewise need the agent to paraphrase or repeat what it said on occasion. Any party in a conversation may require a repair of a prior utterance at any point in the interaction [Schegloff et al. 1977]. Users may not understand what the agent is saying, requiring different kinds of paraphrases, or with voice interfaces, they may not have heard what it said. Handling the standard types of repair is a critical aspect of conversational competence.

B2.1 Repeat Request

The *repeat request* elicits a full repeat of the previous agent utterance, especially in the cases where the user could not hear it (Pattern B2.1.0, Example 6.7).

Pattern B2.1.0 Repeat Request.

```
1  A:  <ANY UTTERANCE>
2  U:  REPEAT REQUEST
3  A:  REPEAT
```

Example 6.7 Repeat Request.

```
1  A:  I guess I like movies with a strong AI lead.
2  U:  what did you say?
3  A:  I guess I like movies with a strong AI lead.
```

In this pattern, if the user says things like "what did you say?," "say again," or "please repeat" (line 2), the agent repeats (line 3) its prior utterance (line 1). In our implementation, each agent utterance is captured in a *repeat* context variable, which is set each time a response is returned to the user. A better solution is for the backend to update the *repeat* variable on every turn with the exact output of the agent. If the user then requests a repeat, the value of the *repeat* variable is simply returned. ***In voice-based interfaces, the repeat request is critical since the agent's utterance is transient.*** It is not as important in text-based interfaces, which typically preserve the utterance in a chat history. However, the ability to repeat what you said is such a basic feature of conversational competence that any agent that cannot recognize a repeat request will appear stupid to users.

If for any reason the value of *repeat* is null, the agent gives a default response to a repeat request instead (Pattern B2.1.1, Example 6.8).

Pattern B2.1.1 Repeat Request Default.

```
1  A:  <ANY UTTERANCE>
2  U:  REPEAT REQUEST
3  A:  DEFAULT RESPONSE
```

Example 6.8 Repeat Request Default.

1 A: `I guess I like movies with a strong AI lead.`
2 U: **`what did you say?`**
3 A: `Oh dear, I forgot!`

When the *repeat* context variable does not have a value, the agent gives a default response (line 3). This pattern should never occur unless the designer has forgotten to set the repeat variable or the backend has failed to do so.

While in the standard pattern, the repeat is a verbatim duplication of the original utterance, short prefaces can be removed, like “Okay” or “I guess.” Furthermore, deviating from a conversation pattern can achieve other actions or effects, such as humor, sarcasm, or attitude (Example 6.9).

Example 6.9 Repeat Request Aborted.

```
1  A:  Are you an expert on computer history?
2  U:  what did you say?
3  A:  never mind
```

Here the agent questions the user’s expertise (line 1). Instead of answering the inquiry, the user requests a repeat (line 2), which could be seen as “taking offense.” But for this particular response, the *repeat* is set to “never mind” (line 3) so that the agent will always back down by aborting the sequence. Setting the repeat to “just kidding” would have a more humorous effect.

Perhaps the most minimal way to initiate repair on the prior utterance is to say, “what?” [Sacks 1992, II:413, Frohlich and Luff 1990, p. 207]. This simply indicates some trouble with the prior utterance but gives no clues whether it might be a hearing trouble or an understanding trouble. In human conversation, “’What?’ says at least ’I didn’t hear what you said, say it again,’ but it’s employed and is well known to be employed as a substitute for ’I don’t understand what you said”’ [Sacks 1992, II:413]. In other words, “what?” leaves it to the speaker to diagnose the recipient’s trouble as one of hearing or understanding. Virtual agents, on the other hand, typically lack this ability to do such diagnoses. Instead, we use a simple rule for deciding if "what?" is a variation of the *repeat request* or *paraphrase request* intents: in voice-based interfaces, it is treated as a repeat request; in text-based interfaces, it is treated as a paraphrase request.

B2.2 Partial Repeat Request

While the repeat request typically elicits a full repeat of the prior agent utterance, the *partial repeat request* elicits only a part of it (Pattern B2.2.0, Example 6.10).

Pattern B2.2.0 Partial Repeat Request.

```
1  A:  <ANY UTTERANCE>
2  U:  PARTIAL REPEAT + what/who/when/where/why
3  A:  PARTIAL REPEAT
```

Example 6.10 Partial Repeat Request.

```
1  A:  I guess I like movies with a strong AI lead.
2  U:  a strong what?
3  A:  AI lead.
```

With the partial repeat request, the user repeats a part of the agent's prior utterance, "a strong" followed by a question word, "what" (line 2). By recycling part of the prior utterance, the user can demonstrate what he or she heard, and the question word indicates a failure at least to hear whatever words came after that [Schegloff 2007, pp. 217–218]. In our implementation, the dialog detects the question word at the end of the user's input, extracts what came before it, matches the sample to the value of *repeat* of the agent's utterance, and displays the remaining words back to the user, "AI lead" (line 3). This practice is very efficient, especially in voice interactions, in eliciting a repeat of just the part of the prior utterance that the recipient did not hear, instead of repeating the whole thing.

B2.3 Hearing Check

The final repeat repair in our framework is the hearing check [Schegloff et al. 1977]. A recipient of an utterance can check his or her hearing of the utterance by repeating all or part of it and inviting the speaker to confirm it (Pattern B2.3.0, Example 6.11).

Pattern B2.3.0 Hearing Check Confirmed.

```
1  A:  <ANY UTTERANCE>
2  U:  FULL/PARTIAL REPEAT
3  A:  CONFIRMATION
```

Example 6.11 Hearing Check Confirmed.

```
1  A:  I guess I like movies with a strong AI lead.
2  U:  lead?
3  A:  That's right.
```

Here the user repeats part (line 2) of what the agent said (line 1), instead of asking the agent to repeat it (B2.1). The repeat then invites the agent to confirm the candidate hearing if it is correct, which the agent does (line 3). Detecting a *hearing check* cannot be done with intents and entities. In our implementation, we detect them by comparing the user's utterance to the agent's prior utterance (*repeat*) using text string matching. If the former is contained in the latter, and the user's utterance fails to match any other conditions, the agent confirms the repeat with "That's it!," "Correct!," "That's what I said," etc. On the other hand, if the user incorrectly repeats part of the agent's utterance, the agent should correct it (Pattern B2.3.1, Example 6.12).

Pattern B2.3.1 Hearing Check Corrected.

```
1  A:  <ANY UTTERANCE>
2  U:  FULL/PARTIAL REPEAT
3  A:  DISCONFIRMATION, REPEAT
```

Example 6.12 Hearing Check Corrected.

```
1  A:  I guess I like movies with a strong AI lead.
2  U:  need?
3  A:  No, LEAD.
```

In this case the user incorrectly repeats the word "lead" as "need" (line 2). So the agent should correct this candidate hearing by disconfirming it and repeating the correct word (line 3). Unfortunately, determining that the user's utterance is a candidate repeat of part of the prior turn is nontrivial. In this case, it depends on the phonemic similarity between "lead" and "need." Currently we do not support this pattern (B2.3.1) in our implementation, but in principle it could be done by comparing the phonemic and lexical similarity between the user's utterance and the agent's prior.

B2.4 Paraphrase Request

While the repeat request elicits a full repeat of the prior utterance, the *paraphrase request* elicits a full paraphrase of it. Observing the principle of minimization, we generally formulate the agent's initial utterance to be concise, using the fewest words we think will be needed to enable the average user to understand. But if a particular user indicates a trouble in understanding, minimization should be relaxed and a more understandable version should be offered (Pattern B2.4.0, Example 6.13).

Pattern B2.4.0 Paraphrase Request.

```
1  A:  <ANY UTTERANCE>
2  U:  PARAPHRASE REQUEST
3  A:  PARAPHRASE
```

Example 6.13 Paraphrase Request (elaboration).

```
1  A:  I guess I like movies with a strong AI lead.
2  U:  what do you mean?
3  A:  I enjoy movies in which the main character is
       an Artificial Intelligence.
```

In this pattern, the agent answers a question (line 1), and the user requests a paraphrase (line 2). The agent then elaborates its prior answer, using different words and making it simpler to understand (line 3). For example, Alma replaces “like” with “enjoy,” replaces the acronym “AI” with the phrase it abbreviates, “artificial Intelligence,” and replaces the somewhat specialized term “lead” with “main character.” The elaboration thus conveys virtually the same meaning as the initial utterance, but it is somewhat easier to understand. We do not give the elaboration first because it is verbose and a little awkward. Also, sometimes just the presentation of two formulations of the same utterance can resolve users’ understanding troubles. Like the repeat, the paraphrase text is set to the *paraphrase* context variable at the same time as the initial response is displayed to the user.

Typically the agent’s utterances should be short and its paraphrases longer; however, in some cases they may be shorter. If the initial utterance is unavoidably long and awkward, its paraphrase should be shorter and to the point, in other words, an upshot rather than an elaboration (Example 6.14).

Example 6.14 Paraphrase Request (upshot).

```
1  A:  First, please sit comfortably on the floor
       and breathe deeply and slowly.
2  U:  what do you mean?
3  A:  Just sit and relax.
```

Here the agent’s initial utterance is somewhat verbose, providing multiple details (line 1), and the user requests a paraphrase (line 2). But instead of giving an even longer version, the agent draws the upshot (line 3). Thus, paraphrases of agent utterances may be upshots or elaborations.

Finally, in the event that the *paraphrase* context variable lacks a value, the agent should give a default response to the paraphrase request (Pattern B2.4.1, Example 6.15).

Pattern B2.4.1 Paraphrase Request Default.

```
1  A:  <ANY UTTERANCE>
2  U:  PARAPHRASE REQUEST
3  A:  PARAPHRASE DEFAULT
```

Example 6.15 Paraphrase Request Default.

```
1  A:  I guess I like movies with a strong AI lead.
2  U:  what do you mean?
3  A:  Well, I don't know how else to say it.
```

When the value of paraphrase is null, Alma admits that it cannot provide one (line 3). Like the default response for the repeat request (B2.1.1), the paraphrase default should not be needed, but is included in case the designer or the backend fails to populate the *paraphrase* context variable.

B2.5 Definition Request

In addition to requesting a full paraphrase of a prior utterance, speakers sometimes request paraphrases of only parts of it. In natural conversation a recipient may repeat any part of a prior utterance and request a paraphrase of it; for example, a response to “do you want to go with me to see a movie?” might be “what do you mean by ’go with’?” (i.e., as a “friend” or “date”). Given today’s natural-language platforms, anticipating every possible substring of the agent’s utterances and providing a paraphrase is not feasible.

As an approximation of such *partial paraphrase requests*, our approach is to provide paraphrases of only specialized terms in the agent’s utterances: in other words, *definitions*. We believe that it is good practice to provide definitions for any specialized or technical terms that your agent uses in its own utterances. ***A conversational agent should be able to clarify what it means by any special terms it uses.*** Ideally these definitions should be tailored to the particular use case and should not simply be dictionary definitions; for example, “Your origin is the city or airport that you’re leaving from” versus “Origin is the point or place where something begins, arises, or is derived.” The user is asking what the agent means by its use of the term, not necessarily what it means more generally.

Definition requests take forms like, “what do you mean by artificial intelligence?,” “what does server room mean?,” “what does IBM stand for?,” “define in-network provider,” etc. When a user requests a definition of a term used in the agent's prior utterance, it initiates repair on that utterance, indicating an understanding trouble (Pattern B2.5.0, Example 6.16).

Pattern B2.5.0 Definition Request Repair.

```
1 A:  <ANY UTTERANCE>
2 U:  DEFINITION REQUEST
3 A:  REPAIR MARKER + DEFINITION
```

Example 6.16 Definition Request Repair.

```
1 A:  I guess I like movies with a strong AI lead.
2 U:  what do you mean by AI?
3 A:  Oh, sorry! AI stands for Artificial Intelligence.
      Basically, robots and smart machines like me!
```

In this pattern, the agent produces any utterance, say an answer to a small-talk inquiry (line 1), and the user requests a *definition* of a term in that utterance, "AI" (line 2). Because the term is in the prior utterance, it suggests that the term is a source of trouble for the user in understanding that turn and therefore an obstacle to moving forward. In response, the agent acknowledges the repair with "Oh, sorry!" and provides a definition (line 3). The "Oh" indicates a realization, or "change-of-state" [Heritage 1984], in this case that its assumption that the user knows the term "AI" is wrong, and the "sorry" apologizes for it. In authoring agent utterances, you must make assumptions about what the average user will know, but repair patterns, like the definition request, will enable users to recover when your assumptions are wrong.

If a user requests a definition of a term or subphrase for which no definition has been created, the agent will give a default response (Pattern B2.5.1, Example 6.17).

Pattern B2.5.1 Definition Request Default.

```
1 A:  <ANY UTTERANCE>
2 U:  DEFINITION REQUEST
3 A:  DEFINITION DEFAULT
```

Example 6.17 Definition Request Default.

```
1 A:  I say, politics is the art of compromise.
2 U:  define compromise?
3 A:  I'm not sure how to define it.
```

Here, because "compromise" is a common term, the designer did not provide a definition for it. When the user asks for a definition (line 2), the agent claims an inability to provide one as a default (line 3). In this kind of instance, a standard dictionary definition would be better than the default response.

Definition requests are not necessarily repair-related. Speakers may ask for definitions of words for reasons other than understanding a prior utterance; in other words, definition requests can be a kind of inquiry (Pattern B2.5.2, Example 6.18).

Pattern B2.5.2 Definition Request Standalone.

```
1 A:  <ANY UTTERANCE WITHOUT TERM>
2 U:  DEFINITION REQUEST
3 A:  DEFINITION
```

Example 6.18 Definition Request Standalone.

```
1 A:  How can I help you?
2 U:  what does server mean?
3 A:  A server is a big computer that provides
      functionality to lots of other computers.
```

When a definition request stands alone, not indicating a trouble with a prior utterance (line 2), the agent simply gives the definition, if available, without marking it as a repair (line 3). The pattern in this case is that of an inquiry (A1); however, to avoid misclassification, a single intent should be used for both kinds of definition requests and an entity should be used to extract the term to be defined.

B2.6 Example Request

Another kind of partial paraphrase request is a request for an *example*. An example specifically makes an abstract or general utterance concrete. Not all utterances have examples, but when they do, examples can be very effective at providing clarity to the recipient. Like the paraphrases and repeats above, examples are set to an *example* context variable at the time the initial utterance is given to the user. The example request intent includes variations such as, “can you give an example?,” “what are some examples?,” “examples please,” etc. (Pattern B2.6.0, Example 6.19).

Pattern B2.6.0 Example Request.

```
1 A:  <ANY UTTERANCE>
2 U:  EXAMPLE REQUEST
3 A:  EXAMPLE
```

Example 6.19 Example Request.

```
1 A:  I guess I like movies with a strong AI lead.
2 U:  can you give an example?
3 A:  Um, 2001: A Space Odyssey... The Matrix...
      Terminator... those are a few.
```

Here, in answering a small-talk inquiry, the agent claims to like a particular type of movie (line 1). The user then requests an example (line 2), and the agent gives three movies of that type (line 3). Providing concrete examples is a different, though common, way of articulating what is meant by abstractions such as “movies with a strong AI lead.”

The *choices request* is similar to the example request in structure, but serves a somewhat different purpose. For a small set of utterances in which the agent requests the user to make a choice, users may request a list of choices or options. The choices request is not a repair of the prior utterance, because it does not suggest that the user fails to understand it. It is a kind of detail request in which the user requires more information in order to continue. Where the example request seeks instantiations of the prior utterance, the choices request seeks options for next actions (Example 6.20).

Example 6.20 Choices Request.

1 A: `What kind of food do you want?`
2 U: **`what are my choices?`**
3 A: `Around here, Mexican, Japanese or Indian.`
4 U: **`Mexican`**
5 A: `Walking distance or car ride?`

In this pattern, the agent asks the user to specify a type of cuisine (line 1), in the course of making a restaurant recommendation, and the user requests choices before deciding (line 2). The agent then offers three choices (line 3) and the user picks one (line 4). Although we use separate intents for recognizing example and choices requests, for convenience we use the *example* context variable to store both examples and choices.

Like the other user-initiated repairs, the example request includes a default response if the value of the *example* variable is null (Pattern B8.1, Example 6.21).

Pattern B2.6.1 Example Request Default.

1 A: `<ANY UTTERANCE>`
2 U: `EXAMPLE REQUEST`
3 A: `EXAMPLE DEFAULT`

Example 6.21 Example Request Default.

1 A: `I talk to my Creator all the time.`
2 U: **`can you give an example?`**
3 A: `I'm afraid I can't think of an example.`

When there is no example provided for an agent utterance, the agent indicates its inability to give one (line 3). Unlike repeat and paraphrase requests, the example request may need to be set to null by the designer frequently, since many utterances do not have sensible examples or choices.

B2.7 Understanding Check

With the hearing check, we saw the user repeat a part of the agent's utterance and invite the agent to confirm or correct it. The *understanding check* has a similar structure, but instead of repeating part of the prior utterance, the user gives a paraphrase or example of it [Schegloff et al. 1977]. In approximating this, the designer must provide a list of all recognized examples in advance. This list of recognized examples should include those given in response to the example request, as well as additional ones that are never explicitly mentioned to the user (Pattern B2.7.0, Example 6.22).

Pattern B2.7.0 Understanding Check Confirmed.

1 A: <ANY UTTERANCE>
2 U: EXAMPLE
3 A: CONFIRMATION

Example 6.22 Understanding Check Confirmed.

1 A: I guess I like movies with a strong AI lead.
2 U: **like The Matrix?**
3 A: Yes.

In this case, the user mentions an example of what the agent just said (line 2) and the agent confirms it (line 3). In our implementation, we use a particular intent for the user's phrase, which may include words such as "like" or "you mean" as prefaces to the example, and we use a specific entity for capturing all of the recognized examples from the user.

If the user does an understanding check on an *incorrect* example of the prior utterance, the agent should disconfirm it and correct it (Pattern B2.7.1, Example 6.23).

Pattern B2.7.1 Understanding Check Disconfirmed.

1 A: <ANY UTTERANCE>
2 U: EXAMPLE
3 A: DISCONFIRMATION. EXAMPLE

Example 6.23 Understanding Check Disconfirmed.

1 A: I guess I like movies with a strong AI lead.
2 U: **like One Flew Over the Cuckoo's Nest?**
3 A: No. Like the Matrix.

Here the agent mentions a type of movie (line 1), and the user suggests a movie that is not an example of that type (line 2). The agent should then disconfirm the candidate example and offer a correct one (line 3). However, as with the hearing check above, recognizing an incorrect example mentioned by the user is currently quite difficult. As a result, our implementation cannot recognize an incorrect example of the agent's prior utterance. Therefore, instead of the disconfirmation and correction, Alma simply requests a paraphrase of what the user said (B1.2).

One limitation of our current implementation of these *user-initiated repairs* (B2.1-B2.7) is that they do not support what we might call "second-order repairs," or repairs of repairs. For example, if the user requests a paraphrase and then immediately requests a paraphrase of the paraphrase or a repeat of the paraphrase or an example of the paraphrase, Alma will provide a repair of the initial utterance, not of the paraphrase. We limit our repairs to those of the first order due to the explosion of repairs that would result from second- or even third-order repairs, all of which would be extremely rare. Consequently, Alma cannot play the child's game of continually parroting "what do you mean?" after each paraphrase (nor can most adults for very long). This is one clear example of how our agent engages the user in a *conversation game*, or a simplified version of a natural human conversation.

6.3 B3 Extended Repair

In each of the cases described above, repair is initiated from the "second position," or next turn to the problematic utterance. But a couple of classes of repair occur beyond next turn. Schegloff [1992b] identifies these as "third-position" repair and "fourth-position" repair. In our Natural Conversation Framework, the former would be initiated by the user and the latter by the agent. We refer to them as "self-correction" and "other-correction," respectively.

B3.1 Self-Correction

While each of the user-initiated repairs so far have been operating on the prior utterance of the agent, users can also initiate repair on their own prior utterances that are two turns back. This pattern is structurally the same as that of the *incremental request* (A2.7), but the action is slightly different. Instead of simply doing a new request, the self-correction is further marked as a redo of the first request (Pattern B3.1.0, Example 6.24).

Pattern B3.1.0 Self-Correction (same as A2.7).

```
1  U:  INQUIRY/REQUEST
2  A:  ANSWER/GRANT
3  U:  SELF-CORRECTION
4  A:  ALTERNATIVE RESPONSE
5  U:  SEQUENCE CLOSER
```

Example 6.24 Self-Correction.

```
1  U:  is the Mexican restaurant close?
2  A:  Mario's is located at Beach and Main.
3  U:  no, I mean in the hotel
4  A:  I'm afraid there is no Mexican restaurant inside the hotel.
5  U:  ok thanks
```

In this pattern, the user repairs part of his prior request (line 3) after seeing the agent's response to it (line 2). That is, the agent interpreted "close" as including restaurants within "walking distance" of the hotel, but the user had intended it to mean "inside the hotel." So the user simply reformulates the specification of distance, from "close" to "inside the hotel," and the agent assumes the prior intent and entity (Mexican cuisine). This self-correction pattern is a case of what Schegloff [1992b] calls a "third-position repair" in which the speaker of a turn repairs that turn, resolving unanticipated ambiguities, after seeing what the recipient made of it. This type of self-correction is typically marked with prefaces such as "no," "I mean" or "no, I mean," but the patterns works with or without them. The self-correction is efficient because it enables users to modify their previous requests without redoing the whole turn.

As an alternative to a self-correction, users may simply report the fact that agent misunderstood what he or she said (Pattern B3.1.1, Example 6.25).

Pattern B3.1.1 Misunderstanding Report.

```
1  U:  INQUIRY/REQUEST
2  A:  ANSWER/GRANT
3  U:  MISUNDERSTANDING REPORT
4  A:  PARAPHRASE REQUEST
5  U:  SELF-CORRECTION
6  A:  ALTERNATIVE RESPONSE
7  U:  SEQUENCE CLOSER
```

Example 6.25 Misunderstanding Report.

```
1  U:  what's your favorite movie theater?
2  A:  I guess I like movies with a strong AI lead.
3  U:  that's not what I mean
4  A:  I'm sorry, please try again.
5  U:  can you recommend a movie theater?
6  A:  I'm afraid I don't have information on movie theaters.
7  U:  ok
```

In this case, when the agent displays a failure to understand (line 2), the user explicitly calls out the failure but does not repair the initial utterance (line 3). Because the user does not provide additional details, the agent has no better basis for responding than it did the first time. If it has moderate confidence in a second intent, it could perhaps respond to that instead. However, in our implementation we simply have Alma apologize for the misunderstanding and request a paraphrase (line 4). Unlike the root-level *agent paraphrase request* pattern (B1.2), the misunderstanding report is triggered by an particular intent, with variations such as "that's not what I meant," "you misunderstood," "you're not understanding me," instead of by a failure to match any conditions.

In addition to requesting a repeat of what the agent said (Pattern B3.0), users can also ask the agent to *repeat* what it heard the user say. And in addition to reporting misunderstanding on the part of the agent, users can also report mishearing (Pattern B3.1.2, Example 6.26).

Pattern B3.1.2 Other Hearing Check & Mishearing Report.

```
1  U:  <ANY UTTERANCE>
2  A:  <ANY RESPONSE>
3  U:  HEARING CHECK
4  A:  REPEAT
5  U:  MISHEARING REPORT
6  A:  APOLOGY. REPEAT REQUEST
7  U:  REPEAT
8  A:  <RESPONSE>
9  U:  SEQUENCE CLOSER
```

Example 6.26 Other Hearing Check & Mishearing Report.

```
1  U:  what's your favorite movie theater?
2  A:  I guess I like movies with a strong AI lead.
3  U:  what did I say?
4  A:  You said: what's your favorite movie
5  U:  that's not what I said
6  A:  I'm sorry. What did you say?
7  U:  I said what's your favorite movie theater?
8  A:  I'm afraid I don't have information on movie theaters.
9  U:  ok
```

Here the user says something (line 1) and the agent responds (line 2). But instead of closing the sequence, the user asks the agent to repeat what it heard the user say (line 3). The agent then repeats the prior user utterance, stored in a context variable, to the user (line 4). This pattern is especially useful in voice interfaces for exposing unnoticed errors in automatic speech recognition, or speech-to-text. If the agent misheard the user, the user can report the mishearing error (line 5) or just repeat the initial utterance. If the former, the agent apologizes and requests a repeat (line 6).

B3.2 Other-Correction

Finally, a repair pattern that we rarely use but may be useful under certain circumstances is what we call other-correction or “fourth-position repair” [Schegloff 1992b]. Just as a speaker can initiate repair after seeing how the recipient interpreted his or her initial utterance, the recipient too gets another chance to repair that utterance (Pattern B3.2.0, Example 6.27).

Pattern B3.2.0 Other-Correction.

```
1  U:  INQUIRY/REQUEST
2  A:  ANSWER/GRANT
3  U:  <MENTION OF KEY ENTITY>
4  A:  OTHER-CORRECTION
5  U:  SEQUENCE CLOSER
```

Example 6.27 Other-Correction.

```
1  U:  what's the address for almaden?
2  A:  IBM Almaden is located at 650 Harry Road,
       San Jose, California, 95120, USA.
3  U:  I thought it was on Bailey road
4  A:  Oh, you mean the Silicon Valley Lab. That's
       555 Bailey Ave, San Jose, California 95141.
5  U:  oh ok
```

In this pattern, the user makes an inquiry that has an error in it (line 1), but the user does not detect it. The agent then gives an answer (line 2). Instead of closing the sequence or initiating repair, the user says something that enables the agent to detect an error in the first utterance (line 1): the user challenges the answer, suggesting a different street address (line 3). Because a nearby facility that is commonly confused with “Almaden” is on Bailey Road, the agent detects that the user really meant “Silicon Valley Lab” (line 4) instead of “Almaden” Lab (line 1). In our implementation, we only attempt to support common errors, rather than trying to anticipate every possible fourth-position repair. If the user mentions features of the Silicon Valley Lab in the turn following the Almaden answer and it does not match an intent, Alma does the correction, and vice versa for answers to the Silicon Valley Lab inquiry.

6.4 B4 Sequence Closers

In this chapter, we have seen several patterns in which users initiate repair on a prior utterance in third position, that is, after the completion of an action pair. Users or the agent do this when they experience some trouble with hearing or understanding parts of that action pair. Alternatively, speakers in natural conversation may close the action pair sequence in third position, with so-called "sequence-closing thirds" [Schegloff 2007]. Sequence closers can enable users to indicate that they are done with a sequence or activity and ready to move on. Sequence closers include several intents, including *acknowledgments* (e.g., okay, ok, all right, got it), *appreciations* (e.g., thank you, thanks, ok thanks), *laughter tokens* (e.g., haha, ha ha, lol), *positive assessments* (e.g., great!, excellent, awesome!), or *negative assessments* (e.g., too bad, oh well, that sucks).

Although it is not supported in most of today's chatbots and voice assistants, sequence closing is an implicit indicator of agent success and an alternative to the user-initiated repairs presented above. In such cases, the user expands the base sequence in a way that closes it and moves on (Pattern B4.0, Example 6.28).

Pattern B4.0 Sequence Closer (helped).

```
1  U:  INQUIRY/REQUEST
2  A:  RESPONSE
3  U:  ACKNOWLEDGMENT/ASSESSMENT/LAUGHTER
4      <OPTIONAL PAUSE>
5  A:  LAST TOPIC CHECK
```

Example 6.28 Sequence Closer (helped).

```
1  U:  what's your favorite movie?
2  A:  I guess I like movies with a strong AI lead.
3  U:  ok
4      (2.0)
5  A:  Anything else?
```

In this pattern, the user does the first part of an action pair, such as an inquiry or open request (line 1), and the agent answers or grants it (line 2). Because the agent succeeded in doing something for the user, we capture this fact by setting a context variable, *helped*, to true. If the agent does not succeed in answering an inquiry or granting a request, this variable remains false. In third position [Schegloff 2007], the user closes the sequence with an acknowledgment (line 3) instead of expanding it with a repair initiator, like "what did you say?" or "define AI."

Because the user closes the inquiry sequence, the agent does not need to say anything more about that sequence. So after the acknowledgment (line 3), the agent pauses (line 4) and waits for the user to make another request or say something else. In a voice-based interface, the pause should be 2–3 seconds, but in a text-based interface, it should be longer, around 20–30 seconds. If the user does not take another turn and the agent has provided some help (i.e., *helped* equals true), the agent will check to see if the user has another request or topic (line 5). If the user says no, the agent will move to close the conversation. The pause in this pattern (line 4) is optional. If it is removed, the agent simply checks for a last topic immediately (line 5).

When the agent has provided some kind of help to the user, it will ask if the user needs "anything else," but if it has not yet done anything for the user, it withholds this *last topic check* (Pattern B4.1, Example 6.29).

Pattern B4.1 Sequence Closer (not helped).

```
1  U:  INQUIRY/REQUEST
2  A:  ANSWER/GRANT
3  U:  ACKNOWLEDGMENT/ASSESSMENT/LAUGHTER
```

Example 6.29 Sequence Closer (not helped).

```
1  U:  what's in theaters?
2  A:  I'm afraid I can't look up movies at this time.
3  U:  too bad
```

Here the agent fails to grant the user's request for movie recommendations (line 1). It recognizes the action, but it tells the user that it cannot fulfill such requests (line 2). Because the agent fails to provide help, the *helped* context variable remains false. So when the user closes the sequence (line 3), the agent does not move to close the conversation with a last topic check, such as "Anything else today?" Instead it simply waits for another utterance by the user. Technically, the dialog logic returns an empty response.

In addition to acknowledgments, assessments, and laughter, users may also close a sequence with an *appreciation*. Unlike the other sequence closers, appreciations invite a response from the agent (Pattern B4.2, Example 6.30).

Pattern B4.2 Sequence Closer Appreciation (helped).

```
1  U:  INQUIRY/REQUEST
2  A:  ANSWER/GRANT
3  U:  APPRECIATION
4  A:  APPRECIATION RECEIPT
5      <OPTIONAL PAUSE>
6  A:  LAST TOPIC CHECK
```

Example 6.30 Sequence Closer Appreciation (helped).

```
1  U:  what's your favorite movie?
2  A:  I guess I like movies with a strong AI lead.
3  U:  thanks
4  A:  You're welcome.
5      (2.0)
6  A:  Anything else?
```

In this variation of the pattern, the agent accepts the user's display of appreciation, "thanks" (line 3), with the standard receipt, "You're welcome" (line 4). It then pauses (optional) and checks for a last topic (line 6). Thanking the other in natural conversation is a strong implicit indicator of the success of an inquiry or request sequence or any other action that provides some kind of help. Appreciations should be tracked as a measure of the conversation's success.

As with the other sequence closers, appreciation sequences are sensitive to whether or not the agent has provided help to the user during the course of the conversation. If the user offers appreciation when the agent has not provided help, the agent refuses to take credit (Pattern B4.3, Example 6.31).

Pattern B4.3 Sequence Closer Appreciation (not helped).

```
1  U:  INQUIRY/REQUEST
2  A:  NO ANSWER ACCOUNT
3  U:  APPRECIATION
4  A:  REFUTATION
```

Example 6.31 Sequence Closer Appreciation (not helped).

```
1  U:  what's in theaters?
2  A:  I'm afraid I can't look up movies at this time.
3  U:  thanks
4  A:  I don't think I did anything.
```

When the agent accounts for its inability to grant the request for movie recommendations (line 2), the user nonetheless offers appreciation (line 3). Because the *helped* context variable is still false, the agent refutes the appreciation (line 4). This displays a bit more intelligence than invariably saying "you're welcome" every time the user says "thank you."

While we have seen users thank the agent for granting a request and failing to do so, they may also thank agents for repairing a prior utterance (Pattern B4.4, Example 6.32).

Pattern B4.4 Sequence Closer (repaired).

1 U: PARTIAL REQUEST
2 A: DETAIL REQUEST
3 U: REPAIR INITIATOR
4 A: REPAIR
5 U: APPRECIATION
6 A: APPRECIATION RECEIPT
7 U: DETAIL
8 A: GRANT

Example 6.32 Sequence Closer (repaired).

1 U: **can you recommend a restaurant nearby?**
2 A: What kind of food do you want?
3 U: **what's an example?**
4 A: Around here, Mexican, Japanese or Indian.
5 U: **thanks**
6 A: You're welcome!
7 U: **um, Japanese**
8 A: Shogun Sushi is located on First and Taylor.

In this variation of the pattern, the user is in the middle of an open request with multiple sequence expansions. When the agent tries to elicit a detail required for the request (line 2), the user initiates a repair (line 3). When the user does this, a context variable, *repaired*, is set to true. It means the prior utterance was a repair. It is then set back to false on each non-repair agent utterance. If the user then offers appreciation for the repair (line 5), the agent will simply receipt the appreciation (line 6), but not move on to check for a last topic. Moving to close the whole conversation (C4) before completing the open request (A2) sequence would show incompetence on the part of the agent.

6.5 B5 Sequence Abort

As we saw in the previous section, users can successfully close a sequence with a variety of sequence closers or "sequence-closing thirds" [Schegloff 2007]. However, users can also abandon a failing sequence without completing or closing it, with phrases like "never mind," "nvm," "forget it," "I give up," etc. This might be done when the agent fails to understand the user or to return the desired information and the user wishes to stop trying. The sequence abort is a good implicit indicator of sequence failure, which should be tracked as a conversation metric in addition to sequence closers (Pattern B5.0, Example 6.33).

Pattern B5.0 Sequence Abort.

1 A: <ANY UTTERANCE>
2 U: ABORT
3 A: ABORT ACKNOWLEDGMENT. OFFER OF HELP

Example 6.33 Sequence Abort.

1 A: What do you mean?
2 U: **never mind**
3 A: Whatever you say. What can I do for you?

In this pattern, the user aborts the current sequence with "never mind" (line 2). This can be done after any agent utterance, but is most useful in the context of a failing sequence, such as when the agent requests a paraphrase (line 1). Rather than working to repair the sequence, the user can explicitly give up. This functions in a similar way to the *escape* command in command-line or graphical user interfaces. Of course, the user can also abandon the current sequence simply by making another inquiry or request. When the user aborts the sequence (line 2), the agent in this pattern acknowledges the abort and reissues its generic offer of help (line 3). This invites the user to try another inquiry or request.

Instead of extending a generic offer of help when the user aborts a failing sequence, the agent may alternatively offer a *capability description*. This more specifically addresses the fact that the user is encountering trouble and offers to review its capabilities as a possible solution (Pattern 5.1, Example 6.34).

Pattern B5.1 Sequence Abort Capability Offer Accepted.

1 A: <ANY UTTERANCE>
2 U: ABORT
3 A: ABORT ACKNOWLEDGMENT. CAPABILITY OFFER
4 U: ACCEPTANCE
5 A: CAPABILITY REPORT

Example 6.34 Sequence Abort Capability Offer Accepted.

1 A: What do you mean?
2 U: **forget it**
3 A: Whatever you say. Want to know what I can do?
4 U: **sure**
5 A: I haven't been trained yet. I can only make small
talk, tell some jokes or ask some trivia questions.
I can tell you more about each of these.

Here the user aborts (line 2) a failing sequence (line 1), and the agent offers to describe its capabilities in response (line 3). If the user accepts the offer, the agent describes its general capabilities (line 5), as it does for a user-initiated *capability check* (C3). If the user declines the capability offer, the agent checks for last topic (line 5, pattern B5.2, Example 6.35).

Pattern B5.2 Sequence Abort Capability Offer Declined.

```
1  A:  <ANY UTTERANCE>
2  U:  ABORT
3  A:  ABORT ACKNOWLEDGMENT. CAPABILITY OFFER
4  U:  DECLINATION
5  A:  ACKNOWLEDGMENT. LAST TOPIC CHECK
```

Example 6.35 Sequence Abort Capability Offer Declined.

```
1  A:  What do you mean?
2  U:  never mind
3  A:  Whatever you say. Want to know what I can do?
4  U:  no
5  A:  Okay. Anything else?
```

By offering the capability description rather than simply giving it, the agent allows the user to decide. This is a good practice if the user has already heard the capability description in the current conversation and if it is a long response. Especially in a voice interface, giving a long, unwanted agent utterance should be avoided.

6.6 Conclusion

In this chapter, we have presented multiple conversational UX patterns for managing the interaction at the *sequence level* (B). This is the level of conversational activities, such as user inquiries, open requests, storytelling, instruction-giving, quizzes, agent inquiries and more (A). Overall, sequence-level management involves repairing a previous utterance in order to complete a sequence (B1-B3), closing a sequence (B4), or aborting a sequence (B5). In these action types, we can see how conversation is a distinctively *interactive* form of natural language use, unlike prose, fiction, poetry, etc. Far from inconsequential talk or “chit chat,” sequence-level management (B) serves a critical function: it enables the participants to achieve mutual understanding, or “intersubjectivity” [Schegloff 1992b], at least for practical purposes.

From this chapter you should gain a technical understanding of two kinds of *sequence expansion* in the Natural Conversation Framework: repair and sequence closing. These include multiple patterns for user or agent to resolve troubles in hearing or understanding what the other said, as well as patterns for transitioning to a new topic without doing so before the other is ready. While repair and sequence closing operate at the level of the action pair sequence (e.g., inquiry-answer, request-granting) or the extended telling sequence (e.g., storytelling, instruction-giving), the patterns in the next chapter involve conversation management at a different level.

7

Conversation Management UX Patterns

In the previous chapter, we addressed UX patterns for conversation management at the sequence level; in this chapter, we cover conversation management at the *conversation level,* or session level. While the former concern the opening, completion, and closing of particular action sequences, such as inquiry or request sequences, the latter concern the opening, completion, and closing of the whole encounter.

Although they are also sometimes viewed as superficial pleasantries or inconsequential "chit chat" from a content perspective, conversation-level management practices are important from the perspective of the social relationship. They are the means through which we engage, disengage, and re-engage with other people. Before we can talk about any particular content, we must first align our attention with another person, communicate an intent to interact, and secure their commitment to interacting with us. Likewise, agreeing on when to stop talking by closing the encounter is equally important. The conversation-level UX patterns enable this management of engagement with the user in a natural way.

The conversation-level management patterns (C) in our Natural Conversation Framework can be broken down into five pattern types and 39 subpatterns (Table 7.1). The five pattern types include ways in which the agent (C1) or the user (C2) can open the conversation, ways they can talk about the agent's capabilities (C3), and ways they can end the conversation either by closing it (C4) or disengaging from each other in other ways.

7.1 C1 Opening (Agent)

Natural conversations are typically opened: we do not just start with the first order of business. We first secure the attention of the intended recipient and establish that

Table 7.1 **Conversation-level management patterns**

C1.0 Opening Greeting (Agent)
C1.1 Opening Self-Identification (Agent)
C1.2 Opening Name Request (Agent)
C1.3 Opening Direct Address (Agent)
C1.4 Opening Welfare Check (Agent)
C1.5 Opening Organization Offer of Help (Agent)
C1.6 Opening Authentication (Agent)
C1.7 Organizational Problem Request (Agent)

C2.0 Greeting (User)
C2.1 Summons (User)
C2.2 Welfare Check (User)
C2.3 Displaced Greeting (User)
C2.4 Displaced Welfare Check (User)
C2.5 Reciprocated Welfare Check (User)
C2.6 News Check (User)
C2.7 News Check Capability (User)
C2.8 Self-Identification (User)
C2.9 Name Correction (User)
C2.10 Name Request (User)

C3.0 General Capability Check
C3.1 Capability Expansion
C3.2 Specific Capability Check
C3.3 Utterance Suggestion

C4.0 Last Topic Check (Agent)
C4.1 Last Topic Check (User)
C4.2 Pre-Closing (User)
C4.3 Closing Farewell (User)
C4.4 Closing Appreciation (Organizational)
C4.5 Closing Name Request (Agent)
C4.6 Closing Success Check (Affirmed)
C4.7 Closing Success Check (Disaffirmed)
C4.8 Closing Success Check Reopened
C4.9 Closing Offer (Affirmed)
C4.10 Closing Offer (Disaffirmed)

C5.0 Transfer Successful
C5.1 Transfer Aborted
C5.2 Recipient Correction
C5.3 Offense Complaint (Agent)
C5.4 Offense Complaint (User)

we are talking to each other. “Speakers without hearers can be seen as ’talking to themselves.’ Hearers without speakers ’hear voices”’ [Schegloff 1968, p. 1093]. The ability to open a conversation with another person is thus fundamental to conversational competence. Elements that are common in openings include *greetings* (e.g., “hello”), *welfare checks* (e.g., “how are you?”), *news checks* (e.g., “what’s up?”), *self-identifications* (e.g., “I’m Dave”), *identification requests* (e.g., “what’s your name?”), and more [Schegloff 1968]. These are not actions that people do in other kinds of natural language use, like web search or writing prose; they are specifically characteristic of conversation.

Openings can be initiated by either the user or the agent. Typically the user initiates the conversation by opening the application, but always-on agents may initiate conversations based on particular trigger events, for example, a reminder to take medication or a delayed flight. Our *agent opening* pattern, in which the agent talks first, includes several components commonly found in ordinary human conversations and service encounters: *greeting, direct address, self-identification, organization identification, welfare check, news check, name request, offer of help, personal information request, capability description, problem request,* and more. In our implementation, these components can be configured by setting various context variables to create several subpatterns, which can be set when the conversation session is initialized.

By default, the agent opening to the conversation begins with the minimal opening: a *greeting* (Pattern C1.0, Example 7.1).

Pattern C1.0 Opening Greeting (Agent).

```
1  A:  GREETING
```

Example 7.1 Opening Greeting (Agent).

```
1  A:  Hello.
```

If no context variables are preset, the agent simply utters a greeting, such as "Hello," "Hi there," "Hi," etc. (line 1). But presetting certain context variables adds additional components and subpatterns to the simple greeting. We offer several but not all possible combinations below.

Conversational openings often contain an exchange of names. The names may be known in advance, among people previously acquainted, and recognized in the opening, for example, in a telephone call. Or if they are unknown, names may be volunteered and/or elicited in the opening. Since conversational systems tend to have personas, the name of the agent can be given in the opening. This is done in our implementation by presetting the *agent name* context variable (Pattern C1.1, Example 7.2).

Pattern C1.1 Opening Self-Identification (Agent).

```
1  A:  GREETING. SELF-IDENTIFICATION.
2  U:  <RESPONSE>
```

Example 7.2 Opening Self-Identification (Agent).

```
1  A:  Hello. I'm Alma.
2  U:  hi
```

In this pattern, the agent gives its name after greeting the user (line 1). The agent then leaves it to the user to return the greeting and/or introduce the *first topic*. Openings to conversations are transitions to the business of the interaction or the reason for the call [Schegloff 1968, p. 1078]. In an agent opening in our implementation, the agent can introduce the first topic of the conversation if you set the *first* *topic* context variable. The first topic is a generic slot after the greetings and agent identification that can be used for a wide variety of actions.

One option for the first topic slot is to elicit the user's name and thereby extend the opening. This is done by setting the *first topic* variable to “name request” (Pattern C1.2, Example 7.3).

Pattern C1.2 Opening Name Request (Agent).

```
1  A:  GREETING. SELF-IDENTIFICATION. NAME REQUEST.
2  U:  NAME.
3  A:  GREETING, DIRECT ADDRESS.
```

Example 7.3 Opening Name Request (Agent).

```
1  A:  Hello. I'm Alma. What's your name?
2  U:  Dave
3  A:  Hi, Dave.
```

After self-identifying, the agent can request the user's name (line 1). The agent will then expect, but not require, a person name in the next turn (line 2) and will respond appropriately with a directly addressed greeting (line 3). If the user does some other action in that slot, say a capability check or an inquiry, the agent gives up on trying to elicit the user's name.

If the user's name is already known from a prior conversation or external database, the agent can use direct address in the first turn (Pattern C1.3, Example 7.4).

Pattern C1.3 Opening Direct Address (Agent).

```
1  A:  GREETING, DIRECT ADDRESS.
2  U:  <RESPONSE>
```

Example 7.4 Opening Direct Address (Agent).

```
1  A:  Hello, Dave.
2  U:  hi
```

By presetting the *user name* context variable, Alma will insert the user's name after the greeting (line 1). This enables the agent to “recognize” the user again after a prior conversation. Doing so will also create the expectation on the part of the user that the agent will remember other things from their previous interaction, which can be persisted by saving and reloading other context variables.

In addition to the first-topic slot, our agent opening pattern includes a second-topic slot. Any of the action types that go in the first-topic slot can also be set as a *second topic*. Other elements that are common in openings to natural conversations are welfare checks (Pattern C1.4, Example 7.5).

Pattern C1.4 Opening Welfare Check (Agent).

1 A: `GREETING. SELF-IDENTIFICATION. NAME REQUEST.`
2 U: `NAME.`
3 A: `GREETING, DIRECT ADDRESS. WELFARE CHECK.`
4 U: `WELFARE REPORT (NEUTRAL/POSITIVE/NEGATIVE)`
5 A: `WELFARE RECEIPT (NEUTRAL/POSITIVE/NEGATIVE)`

Example 7.5 Opening Welfare Check (Agent).

1 A: `Hello. I'm Alma. What's your name?`
2 U: **`Bob`**
3 A: `Hi, Bob. How are you?`
4 U: **`fine/I'm great!/terrible!`**
5 A: `That's good./Great to hear!/I'm sorry to hear that.`

After eliciting the user's name (lines 1–2), the agent then further extends the opening by checking the user's welfare with a "how are you?" (line 3). The agent then expects a welfare report in the next turn (line 4). Depending on whether the report is neutral (e.g., "fine"), positive (e.g., "I'm great!"), or negative (e.g., "terrible!"), the agent will respond differently (line 5). Once the user has reported his or her current welfare, the agent should remember that it has been given and should not be elicited again in the same conversation. This can be done by setting a context variable, say *welfare check,* to true.

Each of the agent opening patterns above are appropriate of ordinary conversations between individuals: friends, family, strangers, etc. These may be useful for agents intended to be virtual friends or companions or for use cases in which the user is expected to introduce the first topic after the opening. However, most virtual agent applications today play the role of some kind of personal assistant or customer service representative. In such cases, the first topic of the conversation is typically an *offer of help*. And in the case of service encounters, an organization identification may be appropriate (Pattern C1.5, Example 7.6).

Pattern C1.5 Opening Organization Offer of Help (Agent).

1 A: `GREETING. SELF-IDENTIFICATION. ORGANIZATIONAL IDENTIFICATION.`
`OFFER OF HELP.`
2 U: `REQUEST/PROBLEM REPORT`

Example 7.6 Opening Organization Offer of Help (Agent).

1 A: `Hello. This is Alma at Virtual Airlines.`
`How can I help you?`
2 U: **`I'd like to book a flight`**

This is the canonical opening for a service encounter. In the first turn, the agent produces a greeting, a self-identification, an organization identification, and an offer of help (line 1). Service agents typically produce these components, rushing through the transitions, so that the service seeker's first turn can be his or her request or reason for the call [Whalen and Zimmerman 1987]. This adaptation of ordinary telephone openings streamlines them for speed and efficiency.

However, some kinds of service encounters require *authentication* before helping the customer. The organization may require the customer to give a name, address, telephone number, account number, Social Security number, or other bits of identifying information as a prerequisite for providing service, or indeed for talking to the caller at all. For these use cases, the first-topic slot can be used to elicit authenticating information and the second-topic slot for the offer of help (Pattern C1.6, Example 7.7).

Pattern C1.6 Opening Authentication (Agent).

1 A: GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
IDENTIFIER REQUEST
2 U: IDENTIFIER
3 A: APPRECIATION. OFFER OF HELP
4 U: REQUEST/PROBLEM REPORT

Example 7.7 Opening Authentication (Agent).

1 A: Hi. I'm Alma at Virtual Communications.
What is your account number?
2 U: **five nine two zero seven zero**
3 A: Thank you. What can I do for you?
4 U: **I want to upgrade my Internet package**

Here the agent uses the first-topic slot to elicit the user's account number (lines 1–2). The system can use the account number to authenticate the user or access the user's electronic record before proceeding. After authenticating the account number, the agent then offers to help the user (line 3). Of course, in some cases, the number of topic slots must be increased to handle multiple bits of identifying information. This practice is an extension of the ordinary practice of asking the caller to self-identify, usually with only a name.

Other elements of ordinary conversational openings can also be inserted into organizational ones to create a more friendly style. Most often this is done by including a *welfare check* (Pattern C1.7, Example 7.8).

Pattern C1.7 Organizational Problem Request (Agent).

1 A: GREETING. ORGANIZATION IDENTIFICATION.
WELFARE CHECK

2 U: WELFARE REPORT
3 A: WELFARE RECEIPT. PROBLEM REQUEST
4 U: PROBLEM REPORT

Example 7.8 Organizational Problem Request (Agent).

1 A: Hello. This is Virtual Mental Wellness.
How's it going?
2 U: **not so well**
3 A: I'm sorry to hear that. What's your problem today?
4 U: **I've been feeling depressed lately**

Setting the first-topic variable to "welfare check" and the second-topic variable to "problem request" will make our agent ask how the user is doing (line 1) before inviting the user to report his or her problem or reason for contacting the organization (line 3). If an agent asks about the user's welfare, it should be able to differentiate among neutral, positive, and negative responses from the user.

When your agent takes the first turn in the conversation, you have the opportunity to raise the initial topic(s) and therefore to attempt to set the agenda for the interaction. We tend to use the *offer of help* as the first or second topic because we tend to build multipurpose agents that can provide more than one kind of service, for example, destination, attraction, and flights recommendations. However, if your agent is very narrow in scope and does only one thing, then that activity should be initiated in the first- or second-topic slots.

7.2 C2 Opening (User)

We have demonstrated several patterns through which a virtual agent can open a conversation with a user, but users can also open conversations or initiate opening-relevant sequences. In some use cases, the user may need to take the first turn in the conversation, for example, if the agent application is already running. Like the agent, the user may start with a simple *greeting* (Pattern C2.0, Example 7.9).

Pattern C2.0 Greeting (User).

1 U: GREETING
2 A: GREETING

Example 7.9 Greeting (User).

1 U: **hi**
2 A: Hello.

The simplest agent response to a user's greeting (line 1) is simply a return greeting (line 2). This is a perfectly appropriate response, although minimal, and it gives the floor back to the user to introduce the first topic of the conversation.

Alternatively, the user may also open or reopen the conversation with a *summons* [Schegloff 1968, Frohlich and Luff 1990, p. 213], either direct address (e.g., "Alma?") or an impersonal one (e.g., "are you there?," "can you hear me?") (Pattern C2.1, Example 7.10).

Pattern C2.1 Summons (User).

1 U: SUMMONS
2 A: GREETING/CONTINUER

Example 7.10 Summons (User).

1 U: **Alma?/are you there?**
2 A: Hello./Hello again./Yes?/What?

In response to direct address or an impersonal summons (line 1), the agent may simply answer the summons with a greeting, subsequent greeting, or continuer (line 2). A user-initiated summons is especially useful in voice interfaces to check to see if the agent is available or if it has timed out or powered off.

Beyond the minimal responses to user greetings and summonses, the agent may show a bit more initiative by immediately following the return greeting with a *welfare check* (Pattern C2.2, Example 7.11).

Pattern C2.2 Welfare Check (User).

1 U: GREETING
2 A: GREETING. WELFARE CHECK
3 U: WELFARE REPORT
4 A: WELFARE RECEIPT. OFFER OF HELP
⋮
5 U: GREETING
6 A: GREETING

Example 7.11 Welfare Check (User).

1 U: **hi**
2 A: Hello. How are you?
3 U: **fine**
4 A: That's good. How can I help you?
⋮
5 U: **hi**
6 A: Hi there.

In this pattern, the user starts the conversation with a greeting (line 1) and the agent returns the greeting and initiates a welfare check sequence (line 2). The agent then expects the same kinds of responses as in pattern C1.4 (line 3). The agent may follow the welfare receipt with another action such as offer of help or problem request. As noted above, when a user reports his or her current welfare (line 3), the agent should remember this by setting a *welfare check* variable to true. Then if the user happens to extend a greeting somewhere in the middle of the conversation (line 5), the agent will now simply return the greeting and will withhold the welfare check (line 6).

In a service encounter, in which agents typically rush through the transitions between the various components of their opening turn, users may nonetheless reciprocate a greeting or welfare check or a name request. The agent must be able to handle each of these actions. For example, if the agent opens the conversation and offers help, the user may nonetheless return the *greeting* (Pattern C2.3, Example 7.12).

Pattern C2.3 Displaced Greeting (User).

```
1  A:  GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
       OFFER OF HELP.
2  U:  GREETING
3  A:  GREETING. WELFARE CHECK.
4  U:  WELFARE REPORT
5  A:  WELFARE RECEIPT. OFFER OF HELP.
```

Example 7.12 Displaced Greeting (User).

```
1  A:  Hi. I'm Alma from IBM Research. How can I help you?
2  U:  hi
3  A:  Hello. How are you?
4  U:  fine
5  A:  That's good. What do you need today?
```

Here, even though the agent invites the user's request at the end of its first turn (line 1), the user instead returns the greeting (line 2) from the beginning of that turn. The agent may return the greeting and issue a welfare check (line 3). After acknowledging the user's welfare report (line 4), the agent may offer to help again (line 5).

Similarly, if the agent checks the user's welfare, the user may reciprocate the concern in the next turn. Even though users know that machines do not have feelings, they are still likely to participate in this form of talk (Pattern C2.4, Example 7.13).

Pattern C2.4 Displaced Welfare Check (User).

```
1  A:  GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
       WELFARE CHECK
2  U:  WELFARE REPORT. WELFARE CHECK
3  A:  WELFARE REPORT
```

Example 7.13 Displaced Welfare Check (User).

```
1  A:  Hi. I'm Alma from IBM Research. How are you?
2  U:  great! how are you?
3  A:  Can't complain. It's not in my programming.
```

In this pattern, instead of simply reporting his or her welfare, the user immediately returns the welfare check (line 2). The agent should then be able to report its welfare (line 3). Because the agent's welfare is fictional, these responses are a good slot to put a variety of humorous or witty comments to delight the user.

The user may initiate a *welfare check* at other places in the conversation, although it will most often be toward the beginning of the interaction (Pattern C2.5, Example 7.14).

Pattern C2.5 Reciprocated Welfare Check (User).

```
1  A:  GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
       WELFARE CHECK
2  U:  WELFARE REPORT
3  A:  WELFARE RECEIPT. OFFER OF HELP
4  U:  WELFARE CHECK
5  A:  WELFARE REPORT
```

Example 7.14 Reciprocated Welfare Check (User).

```
1  A:  Hi. I'm Alma from IBM Research. How are you?
2  U:  I'm fantastic!
3  A:  That's great to hear! How can I help you?
4  U:  how are you doing?
5  A:  You know, livin' the dream!
```

Again, users do not always respond with the kind of action projected by the agent's previous utterance [Szymanski and Moore 2018]. Here, after acknowledging the user's current welfare (line 2), the agent immediately offers help (line 3). But instead of making a request or reporting a problem, the user reciprocates the welfare check (line 4). The agent then reports its own welfare in a light-hearted manner (line 5).

An alternative to the welfare check that is common in the openings to ordinary conversations is the *news check*, for example, "what's new?," "what's up?," "what's happening?," "anything new?," etc. Unlike the welfare check, the proper response to the news check is either the reporting of some newsworthy event or of the absence of news, for example, "nothing," "not much," "same old, same old," etc. Because such news reports can take so many different forms, most unrelated to the purpose of the application, they are challenging for virtual agents to recognize and parse. Consequently, we recommend using agent-initiated welfare checks instead of news checks. But if the user initiates a news check, the agent can produce some simple responses (Pattern C2.6, Example 7.15).

Pattern C2.6 News Check (User).

1 U: NEWS CHECK
2 A: NO-NEWS REPORT
3 U: SEQUENCE CLOSER

Example 7.15 News Check (User).

1 U: **what's up?**
2 A: Same old, same old.
3 U: **oh well**

The simplest and safest response to a news check for the agent is a report of no news (line 2). This is a perfectly appropriate response, and it both moves the conversation along and avoids the initiation of a tangent or distraction from the task at hand.

Responding to a news check with a substantive or humorous news report requires a bit more design. In either case, the agent should be prepared for follow-up inquiries about the news given. One possibility is to use the news check as an opportunity to mention *new capabilities* of the agent (Pattern, C2.7, Example 7.16).

Pattern C2.7 News Check Capability (User).

1 U: NEWS CHECK
2 A: NEWS REPORT
3 U: FOLLOW-UP INQUIRY
4 A: ANSWER
5 U: SEQUENCE CLOSER

Example 7.16 News Check Capability (User).

1 U: **what's new?**
2 A: I can now help you make restaurant reservations!
3 U: **does that work in San Jose?**
4 A: San Jose is one of my areas of coverage.
5 U: **Cool!**

In this case, when asked “what’s new?” (line 1), the agent reports a new capability that it recently gained (line 2). The user then asks a follow-up inquiry about that capability (line 3), which the agent can answer (line 4). Using a recent feature of the application as news can be efficient because you will likely already be building out intents and responses around that feature.

In addition to greetings and welfare checks, users may also reciprocate the *self-identification* near the agent’s opening, with utterances like “my name is Dave,” “I’m Peggy,” “call me Bob,” and the like (Pattern C2.8, Example 7.17).

Pattern C2.8 Self-Identification (User).

1 A: GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
OFFER OF HELP
2 U: NAME GIVING
3 A: DIRECT ADDRESS. OFFER OF HELP

Example 7.17 Self-Identification (User).

1 A: Hi. I'm Alma from IBM Research. How can I help you today?
2 U: **my name is Dave**
3 A: Hi, Dave! What can I do for you?

After greeting the user and identifying itself, the agent offers to help the user (line 1). But instead of making a request, the user self-identifies as “Dave” (line 2). Our agent then acknowledges the name by using it to directly address the user and reissue the offer of help (line 3). Even if your application does require the user’s name, it is useful to support this pattern, especially if your agent gives its name, because it is natural for users to reciprocate.

The user self-identification pattern (C2.8) can also be combined with the agent opening using the name request pattern (C1.2). Users can use the former to repair the latter (Pattern C2.9, Example 7.18).

Pattern C2.9 Name Correction (User).

1 A: GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
NAME REQUEST
2 U: NAME GIVING
3 A: <RECOGNITION FAILURE>
4 U: NAME GIVING
5 A: APOLOGY. DIRECT ADDRESS. OFFER OF HELP

Example 7.18 Name Correction (User).

1 A: Hi. I'm Alma from IBM Research. What's your name?
2 U: **shun**
3 A: What do you mean?
4 U: **my name is Shun/that is my name**
5 A: Oh, sorry. Hi, Shun. How can I help you?

In this example, the agent requests the user’s name (line 1), but fails to recognize it as a name (line 3). This is inevitable given the wide range of names both within and across cultures. However, there is a natural way to repair this trouble: repeating the name with additional words to frame it, “my name is Shun” (line 4). Additional framing, such as “my name is . . . ,” helps Alma recognize this user’s action as *name giving*. Alternatively, if the user says, “that is my name” (line 4), without repeating the name, the agent should still be able to handle it. This is a bit more complex because it means that the dialog logic must capture the *current user input* and the *previous user input* for every user utterance. In this case it would then set the previous one to the context variable *user name*.

Just as users may volunteer their names even when the agent did not ask for it, users may also request the agent's name when it does not provide one (Pattern C2.10, Example 7.19).

Pattern C2.10 Name Request (User).

```
1 A:  <ANY UTTERANCE>
2 U:  NAME REQUEST
3 A:  NAME GIVING
```

Example 7.19 Name Request (User).

```
1 A:  Hello. This is IBM Research. How can I help you?
2 U:  what's your name?
3 A:  My name is Alma.
```

In this pattern, anywhere in the interaction, the user may request the agent's name with utterances like, "what's your name?," "do you have a name?," "what can I call you?," and the like (line 2). In response, the agent simply gives the value for the context variable *agent name*, with or without additional framing (line 3). Because exchanging names and using them to refer to oneself or the other is so basic to human conversation, your agent should be able to give its name, even if it does not otherwise volunteer it.

We see then at least nine different patterns for combining the standard elements of openings in ordinary conversations and in service encounters. There are of course more combinations not covered here.

7.3 C3 Capabilities

One challenge with conversational interfaces is the *discoverability* of their features [Norman 1988]. ***While graphical user interfaces can provide menus, icons, and buttons that represent features or sets of features, conversational interfaces are more like command lines, which require users to know what the system can do.*** A common approach we have observed in chatbot UX design to the problem of feature discoverability is to insert lengthy descriptions of capabilities into the openings of the interaction. While this approach has the advantage of providing users some global instruction right before they need to take action, it suffers from some problems. First, the longer the capability description, the more cumbersome it becomes in the opening of the conversation, preventing the user from getting started. Second, a global description of the agent's capabilities may not be equally relevant to all users. Repeat or expert users already know how to talk to the agent. For them, inserting the lengthy description just gets in the way. On the other hand, novice users may require different levels of detail on different features.

We offer an alternative approach: talk to the agent about what it can do. For a virtual agent to be intelligent and conversational, it must be able to talk about what it can and cannot do. In fact, this is one of the main advantages of conversational agents; they can be self-explicating. As argued in Chapter 4 under the six basic navigation actions proposed, users should always be able to ask the user, "what can you do?," "what do you know?," "how can you help me?," "capabilities please," and the like. The *capability check* can be used as the most general way to start a conversation with the agent about its capabilities (Pattern C3.0, Example 7.20) instead of telling the user about the agent's capabilities every time.

Pattern C3.0 General Capability Check.

```
1 U:  GENERAL CAPABILITY CHECK
2 A:  GENERAL CAPBABILITY DESCRIPTION
3 U:  SEQUENCE CLOSER
```

Example 7.20 General Capability Check.

```
1 U:  what can you do?
2 A:  I haven't been trained yet. I can only make small talk, tell some
      jokes or ask or answer some trivia questions.
3 U:  ok
```

By initiating a capability check (line 1), the user can elicit a global description of what the agent can do (line 2). This is critical since virtual agents tend to have a relatively narrow domain of knowledge and range of abilities. And the kind of commonsense background knowledge about what people in certain job roles can do, for example *store clerk, bank teller,* or *hotel concierge*, may be unavailable to the user or inapplicable to your virtual agent. On the other hand, if the user already knows what the agent does, he or she simply does not initiate the capability check, avoiding unnecessary talk. And if your application detects the first time a user interacts with it, you might put the general capability description in the first-topic slot (C1) and then remove it for subsequent sessions. Furthermore, if your general capability description must be longer than a sentence or two, you should present it as an *extended telling* (A3), especially in voice interfaces.

In addition to a global description of capabilities, your agent should be able to say more about what it can do. The agent should at least be able to *expand* on the topics mentioned in it (Pattern C3.1, Example 7.21), as well as to repair it like any other utterance (B2).

Pattern C3.1 Capability Expansion.

1 U: GENERAL CAPABILITY CHECK
2 A: GENERAL CAPABILITY DESCRIPTION
3 U: SPECIFIC CAPABILITY EXPANSION REQUEST
4 A: SPECIFIC CAPABILITY EXPANSION
5 U: ACKNOWLEDGMENT
6 U: SPECIFIC CAPABILITY EXPANSION REQUEST
7 A: SPECIFIC CAPABILITY EXPANSION
8 U: SEQUENCE CLOSER

Example 7.21 Capability Expansion.

1 U: **what can you do?**
2 A: I haven't been trained yet. I can only make small talk, tell some jokes or ask or answer some trivia questions.
3 U: **tell me more about small talk**
4 A: You know, things like, how's the weather?, did you see the game?, what's your favorite movie?... mindless topics like that.
5 U: **ok**
6 U: **tell me more about trivia**
7 A: I can quiz you about computer trivia. See how much you know about computer history.
8 U: **ok**

After eliciting a general description of the agent's capabilities (lines 1–2), the user asks the agent to expand on one of the topics mentioned (line 3), to which the agent provides an expansion of that particular topic (line 4). After acknowledging the expansion (line 5), the user requests an expansion on one of the other topics (line 6). The agent provides an expansion of that topic too (line 7) and should be able to expand on any topic mentioned in its general capabilities description. Note that such *topic expansions* (lines 4 and 7) function somewhat differently from paraphrases that *elaborate* (B2.4): while the latter say the same thing as the prior utterance but in different words, the former provide new information not contained in the general capabilities description.

In addition to describing what they can do, virtual agents should also be able to talk about what they cannot do. For every application, users will ask about or attempt to do actions that are related to the purpose of the agent but are not currently supported or will never be supported. For example, if the agent helps users find travel destinations and book flights, users may reasonably assume that it might book taxi cabs or reserve tables at restaurants too. If these capabilities fall outside of the scope of the application, the agent should nonetheless recognize such an on-topic request and then inform the user that it falls outside its current capabilities (Pattern C3.2, Example 7.22).

Pattern C3.2 Specific Capability Check.

```
1  U:  SPECIFIC CAPABILITY CHECK/UNSUPPORTED REQUEST 1
2  A:  SPECIFIC CAPABILITY DESCRIPTION 1
3  U:  SPECIFIC CAPABILITY CHECK/UNSUPPORTED REQUEST 2
4  A:  GENERAL CAPBABILITY DESCRIPTION
5  U:  SEQUENCE CLOSER
```

Example 7.22 Specific Capability Check.

```
1  U:  can you book a taxi for me?
2  A:  I'm afraid I don't have information on ground or other local
       transportation at this time.
3  U:  can you make a restaurant reservation?
4  A:  I can recommend travel destinations, attractions or flights based
       on your preferences for things to do, like museums or hiking, or
       for weather, like warm and sunny.
5  U:  oh ok
```

In this case, the user asks about a specific agent capability (line 1), which would be perfectly reasonable to assume the agent might be able to do if it is, say, a virtual travel agent application. In response, the agent recognizes the request type, or intent, but indicates that it is beyond its current capabilities (line 2). The user then asks about another specific capability (line 3), but in this case the agent does not recognize the specific intent. Since it can recognize the “can you . . . ” portion of the user’s request, it can recognize it as some kind of capability check, so it gives its general capability description as a default response for idiosyncratic requests types (line 4). You may need to create many sequences for *out-of-scope* user requests, and you should budget time for creating these in your development schedule (see Chapter 8 for more).

Finally, you can also enable your virtual agent to give examples of utterances that it recognizes in case the user gets stuck. User utterances like, “what can I say?,” “what should I say?,” “tell me what to say,” etc., should get the agent to suggest various valid utterances. While this would be unnatural in most kinds of human conversations, you might find it in conversations in which someone is teaching another person a game or a process (Pattern C3.3, Example 7.23).

Pattern C3.3 Utterance Suggestion.

```
1  U:  UTTERANCE SUGGESTION REQUEST
2  A:  <SAMPLE UTTERANCE>
```

```
3  U:  <SAMPLE UTTERANCE>
4  A:  <RESPONSE>
5  U:  SEQUENCE CLOSER
```

Example 7.23 Utterance Suggestion.

```
1  U:  what can I say?
2  A:  Who invented the hard disk?
3  U:  who invented the hard disk?
4  A:  The hard disk drive was invented by IBM in 1956.
5  U:  ok
```

Here, instead of asking the agent what it can do, the user asks the agent what he can say (line 1). The agent then gives an example of an utterance it recognizes (line 2). This can be randomly selected from set of user actions that are core to the application, or they can be sensitive to the local context by selecting a sample action based on the previous response the agent gave or based on particular entities mentioned so far in the conversation. If the user then repeats the agent's suggestion (line 3), the agent should give a supported response (line 4). The utterance suggestion can be useful when a user is still learning the conversational application.

Utterance suggestions can also be exposed to users in a different way: through a visual user interface. If your application includes a visual display, desktop or mobile, you can suggest user utterances on the side rather than through the conversation itself. In our virtual travel agent, we wrote tips, or examples of valid user responses, for every agent utterance in the conversation. We then display these through a ticker above the text-entry box. If users get stuck, they can look at the scrolling tips for models of how to respond. Or expert users can hide the tips if they find them distracting. These *utterance tips* are somewhat like tool tips in a graphical user interface.

These are just some of the ways in which your virtual agent can talk about its own capabilities and help users discover what it can do.

7.4 C4 Closing

Just as natural conversations are characteristically opened at the beginning of the interaction, so too are they closed at their end. Speakers do not typically just stop talking but rather methodically and collaboratively close down the encounter. They use the activities of opening and closing to mark the boundaries between *this* conversation and the next one. In other words, opening and closing create the unit “a single conversation” [Schegloff and Sacks 1973, p. 71]. To stop talking without closing the conversation is to create the expectation that it will be resumed shortly, or rather, that the participants are in a "continuing state of incipient talk" [Schegloff and Sacks 1973, p. 96]. While opening a conversation aligns the parties' expectations for listening and contributing, closing aligns their expectations for disengagement.

Conversation analysts have shown that people close conversations in a distinctive way. Speakers tend not to say "goodbye" without warning. Instead, they move toward the final farewell sequence with a "pre-closing" [Schegloff and Sacks 1973, p. 80, Frohlich and Luff 1990, p. 214]. The pre-closing functions to check to see if the other party wishes to continue with another topic. If the other brings up a new topic, the conversation continues. If not, the speaker who initiated closing will do a closing-relevant action, like an *appreciation* (e.g., "well thanks again!") or *assessment* (e.g., "well this was great!"), or simply bid *farewell* (e.g., "goodbye" or "bye now"). This enables the speakers to coordinate their disengagement so that no topic is missed and that the ensuing silence will not be misinterpreted as one party failing to respond [Schegloff and Sacks 1973, p. 73].

Virtual agents should be able to handle conversation closings. Not only is this a part of basic conversational competence, but closing an interaction with the user has other benefits as well. First, closing enables you to determine if the user finished his or her business versus simply stopped or got disconnected. Second, closing creates a slot in the conversation for the agent to bring up a *last topic*. You can insert a variety of actions into this last topic slot, from simple *appreciations* (e.g., "Thank you for contacting IBM!") to *name requests* (e.g., "Can I get your name before you go?"), to *success checks* (e.g., "Did you find what you were looking for?"), to *promotions* (e.g., "Would you like to hear about our current promotion?"), and more. Our implementation of the closing pattern enables the designer to configure a *last topic* as well as an *agent organization*. Like our conversation opening, the configurable closing allows for a wide variety of different interaction patterns. Several are outlined below.

As in natural conversation, both the agent and the user should be able to initiate the closing of the encounter. In our framework, the agent can move to close the conversation with a *last topic check* (e.g., "Anything else?"), and users can move to close it with a *last topic check*, a *pre-closing* (e.g., "gotta go"), or a *farewell* (e.g., "goodbye"). Consider first the agent-initiated last topic check. The agent asks, "Anything else?," "Anything else I can do for you?," or the like, after the user closes a sequence (recall pattern B4.0) and the agent has provided some kind of help. In this context, the user may possibly be ready to close the conversation (Pattern C4.0, Example 7.24).

Pattern C4.0 Last Topic Check (Agent).

1 U: SEQUENCE CLOSER
2 <optional pause>
3 A: LAST TOPIC CHECK
4 U: DISAFFIRMATION
5 A: WELL-WISH
6 U: RETURN WISH/FAREWELL
7 A: FAREWELL
8 U: FAREWELL

Example 7.24 Last Topic Check (Agent).

1 U: **ok**
2 (2.0)
3 A: Anything else?
4 U: **no**
5 A: Have a good day!
6 U: **you too/bye**
7 A: Goodbye.
8 U: **bye**

In this pattern, the user closes the prior sequence with an acknowledgment (line 1), and after a pause the agent checks to see if the user has another topic (line 3). The agent's last topic check is triggered by other sequence closers as well, including *appreciations, assessments,* and *laughter tokens* (Pattern B4). When the user indicates no additional topic (line 4), the agent moves to its last topic. If the *last topic* context variable is null, the agent simply wishes the user well (line 5). The user can return the wish, say farewell, open a new topic, and more (line 6). The agent will then bid farewell (line 7), initiating the closing if the user returned the wish or completing the farewell sequence, or "terminal exchange" [Schegloff and Sacks 1973, p. 73], if the user had already bid farewell (line 6). Notice that, after the user's return farewell (line 8), the agent does not bid farewell a second time. It keeps track of whether it has already done so and instead returns an empty response.

Like the agent, users can also initiate conversation closing with a *last topic check* by asking "anything else?," "are we done?," "is that it?," etc. (Pattern C4.1, Example 7.25).

Pattern C4.1 Last Topic Check (User).

1 U: LAST TOPIC CHECK
2 A: DISAFFIRMATION
3 U: FAREWELL
4 A: FAREWELL

Example 7.25 Last Topic Check (User).

```
1  U:  anything else?
2  A:  That's it!
3  U:  goodbye
4  A:  Goodbye.
```

Here the user checks to see if the agent has another topic (line 1). If the *last topic* variable is null, the agent simply indicates that it has no more topics, with “That’s it!,” “That’s all!,” “We’re done!,” and the like (line 2). The user can then terminate the conversation by initiating the farewell sequence (lines 3–4).

In addition to the last topic check, the user can do a *pre-closing* with utterances like “I’ve got to go,” “I need to leave,” “I’m out of time,” and the like (Pattern C4.2, Example 7.26).

Pattern C4.2 Pre-Closing (User).

```
1  U:  PRE-CLOSING
2  A:  WELL-WISH
3  U:  RETURN WISH
4  A:  FAREWELL
5  U:  FAREWELL
```

Example 7.26 Pre-Closing (User).

```
1  U:  got to go
2  A:  Have a great day!
3  U:  you too/bye
4  A:  Goodbye.
5  U:  bye
```

In this case, the user moves to close by saying he must disengage, “got to go” (line 1). The agent then checks its dialog logic for a last topic and finds none, so it gives the default *well-wish*, “Have a great day!” (line 2). The user may return the well-wish or move straight to farewell (line 3). The agent says “Goodbye” (line 4) and the user may complete the farewell sequence (line 5).

Finally, instead of performing a pre-closing, the user may simply *bid farewell*. In our implementation, Alma does not return a farewell if the user skips the pre-closing, but instead moves to her last topic slot (Pattern C4.3, Example 7.27).

Pattern C4.3 Closing Farewell (User).

```
1  U:  FAREWELL
2  A:  WELL-WISH
3  U:  FAREWELL
4  A:  FAREWELL
```

Example 7.27 Closing Farewell (User).

```
1  U:  bye
2  A:  Have a great day!
3  U:  goodbye
4  A:  Goodbye.
```

In this context, that is, a farewell with no pre-closing, Alma treats the user's farewell (line 1) as a pre-closing by not reciprocating it. Instead the dialog logic checks the last topic slot, finds a value of null, and returns the default well-wish (line 2). Had the *last topic* variable been configured with an action, Alma would have performed that (see below). If the user then says a farewell again (line 3), Alma returns it this time (line 4). If not, the conversation simply ends after the well-wish (line 2).

We have seen four ways in which the agent and the user can initiate the closing of a conversation (Patterns C4.0–C4.3), but in each pattern above, the closing is not configured with a last topic. However, a variety of actions can be done by the agent in the last topic slot (Pattern C4.4, Example 7.28).

Pattern C4.4 Closing Appreciation (Organizational).

```
1  U:  PRE-CLOSING/FAREWELL
2  A:  APPRECIATION
3  U:  APPRECIATION RECEIPT/FAREWELL
4  A:  FAREWELL
5  U:  FAREWELL
```

Example 7.28 Closing Appreciation (Organizational).

```
1  U:  bye/got to go
2  A:  Thank you for contacting IBM Research today!
3  U:  you're welcome/bye
4  A:  Goodbye.
5  U:  bye
```

Here, instead of the default well-wish, the agent does an appreciation in the last topic slot (line 2). Because the *agent organization* variable is set, the agent gives an organizational appreciation. The user can acknowledge the appreciation, "you're welcome," or simply bid farewell, "bye" (line 3). Then the agent bids farewell, "Goodbye" (line 4), which the user may repeat or not (line 5).

Another commonly useful last topic for the agent is a *name request*. If the agent failed to get the user's name in the opening of the conversation, perhaps because the user changed the topic, the agent has one last chance at the end of the conversation (Pattern C4.5, Example 7.29).

Pattern C4.5 Closing Name Request (Agent).

```
1  U:  LAST TOPIC CHECK
2  A:  NAME REQUEST
3  U:  NAME
4  A:  APPRECIATION, NAME
5  U:  APPRECIATION RECEIPT/FAREWELL
6  A:  FAREWELL
7  U:  FAREWELL
```

Example 7.29 Closing Name Request (Agent).

```
1  U:  anything else?
2  A:  Can I please get your name?
3  U:  Dave
4  A:  Thank you, Dave!
5  U:  you're welcome/bye
6  A:  Goodbye.
7  U:  bye
```

In this pattern, the user happens to initiate closing by checking for a last topic, “anything else?” (line 1). In response, the agent asks for the user’s name because the *last topic* variable is set to “name request.” If the user provides a name (line 3), the agent says “thank you” and addresses the user by name (line 4). The user and agent can then exchange farewells, just as in Patterns C4.0, C4.2, and C4.4 above. If the user’s name is captured, it can be persisted and imported when initializing the next conversation with that user so that the agent can open it with *direct address*, “Hello, Dave. How can I help you?” (C1.3).

In customer service encounters, another commonly useful last action is the *success check*, in which the agent asks the user, “Did you find what you were looking for?,” “Did I answer your questions?,” “Did you get what you needed?” and the like (Pattern C4.6, Example 7.30).

Pattern C4.6 Closing Success Check (Affirmed).

```
1  U:  SEQUENCE CLOSER
2      <optional pause>
3  A:  LAST TOPIC CHECK
4  U:  DISAFFIRMATION
5  A:  SUCCESS CHECK
6  U:  AFFIRMATION
7  A:  ASSESSMENT. FAREWELL
8  U:  FAREWELL
```

Example 7.30 Closing Success Check (Affirmed).

```
1  U:  ok
```

2 (2.0)
3 A: Anything else today?
4 U: **no**
5 A: Did you find what you were looking for?
6 U: **yes**
7 A: Great! Goodbye.
8 U: **bye**

In this case, the agent does a last topic check (line 3) after the user closes a sequence (line 1). If the user indicates no additional topics (line 4), the agent explicitly asks the user if the conversation was successful in terms of whatever help the user was seeking (line 5). If the user indicates success with an affirmation (line 6), the agent bids the user farewell (line 7), which the user may return (line 8). This is useful feedback to the designer and others that the interaction was a success.

On the other hand, if, in response to the success check, the user indicates that the conversation was unsuccessful, the agent still has the opportunity to put things right since the conversation has not ended. The agent may suggest remedial actions, review its capabilities, or simply offer to try again (Pattern C4.7, Example 7.31).

Pattern C4.7 Closing Success Check (Disaffirmed).

1 U: PRE-CLOSING/FAREWELL
2 A: SUCCESS CHECK
3 U: DISAFFIRMATION
4 A: SUBSEQUENT OFFER OF HELP
5 U: DISAFFIRMATION
6 A: ACKNOWLEDGMENT. FAREWELL
7 U: FAREWELL

Example 7.31 Closing Success Check (Disaffirmed).

1 U: **bye/got to go**
2 A: Did you find what you were looking for?
3 U: **not really**
4 A: Would you like me to try again?
5 U: **no thanks**
6 A: Okay. Goodbye.
7 U: **bye**

Here the user disaffirms (line 3) the agent's success check (line 2). The agent acknowledges the failure by again offering help (line 4). If the user declines this subsequent offer of help (line 5), the agent proceeds to bid the user farewell (line 6), which the user can return (line 7) but need not. This is useful feedback that there was some kind of failure in this interaction with the user.

A final variation of the success check pattern involves the user *reopening* the conversation rather than closing it (Pattern C4.8, Example 7.32).

Pattern C4.8 Closing Success Check Reopened.

```
1  U:  LAST TOPIC CHECK
2  A:  SUCCESS CHECK
3  U:  DISAFFIRMATION
4  A:  SUBSEQUENT OFFER OF HELP
5  U:  AFFIRMATION
6  A:  OFFER OF HELP
7  U:  INQUIRY/REQUEST
```

Example 7.32 Closing Success Check Reopened.

```
1  U:  anything else?
2  A:  Did you find what you were looking for?
3  U:  no
4  A:  Would you like me to try again?
5  U:  sure
6  A:  What can I do for you?
7  U:  well, I need a restaurant recommendation
```

In this pattern, the user accepts the agent's offer to try again (lines 4–5), and the agent offers help (line 6). The user then makes another inquiry or request (line 7). This keeps the conversation going, and it may go on for some time until returning to a closing sequence. Instead of a generic offer of help, the agent produces a more tailored offer based on what happened in the previous turns, for example, "Do you want recommendations for places to stay or places to eat in Rome?" This requires additional logic for diagnosing the source of the user's dissatisfaction.

Customer service encounters frequently extend *offers* and promotions to customers before closing the call. This can be done naturally as the agent's last topic (Pattern C4.9, Example 7.33).

Pattern C4.9 Closing Offer (Affirmed).

```
1  U:  SEQUENCE CLOSER
2      <optional pause>
3  A:  LAST TOPIC CHECK
4  U:  DISAFFIRMATION
5  A:  OFFER
6  U:  AFFIRMATION
7  A:  OFFER DETAILS
```

Example 7.33 Closing Offer (Affirmed).

1 U: **ok**
2 (2.0)
3 A: Anything else today?
4 U: **no**
5 A: Would you like to become a Rewards Member?
6 U: **sure**
7 A: Great! With a Rewards membership you get...

When the user indicates no additional business (line 4), the agent may extend a promotion or offer to the user, such as a membership (line 5). If the user accepts the offer (line 6) or agrees to hear about it, the conversation is reopened and may continue as long as the offer activity, with its descriptions, inquiries, and sign-ups, continues.

On the other hand, when users decline offers and promotions in the last topic slot of the conversation, the agent can then move immediately to close it (Pattern C4.10, Example 7.34).

Pattern C4.10 Closing Offer (Disaffirmed).

1 U: PRE-CLOSING/CLOSING
2 A: LAST TOPIC CHECK
3 U: DISAFFIRMATION
4 A: OFFER
5 U: DISAFFIRMATION
6 A: ACKNOWLEDGMENT. APPRECIATION. FAREWELL
7 U: FAREWELL

Example 7.34 Closing Offer (Disaffirmed).

1 U: **gotta go/bye**
2 A: Anything else today?
3 U: **no**
4 A: Can I interest you in a credit card?
5 U: **no thanks**
6 A: Okay. Thanks for contacting Telco! Goodbye.
7 U: **bye**

In this case, when the user declines the credit card offer (lines 4–5), the agent acknowledges the declination, gives an organizational appreciation, and bids farewell (line 6). The user can return the farewell (line 7) or simply stop talking to close the conversation. Additional appeals can be built in, instead of taking the first “no” for an answer. In this way, you can make your agent as persistent as a human salesperson if desired.

In short, building your agent to handle conversation closings brings with it some of the affordances of closings in human conversation. Both the user and agent can know that the other does not intend to continue talking. By closing, both user and agent are agreeing that the interaction is over and therefore no additional participation is expected. And most importantly, performing a pre-closing gives the other party, whether user or agent, a final opportunity to bring up any topics that still need to be addressed but may have been missed. When user and agent close a conversation, it marks the boundary of the conversation as a unit so that anyone who later reviews the system logs can determine that the interaction was completed and not merely suspended.

7.5 C5 Disengaging

While conversation closings have a canonical structure (C4), there are other ways that users and agents may disengage from each other. These disengagements may be temporary or terminal. Besides the user or agent simply failing to continue, users may ask to be transferred to a human, or users may start talking to someone else in the room, agents may refuse to continue talking, and more. Each of these forms of disengagement, and no doubt others, should be designed and built for your agent.

An occasional outcome of service encounters is for the customer to ask to speak to another agent or supervisor. In the context of virtual agents, the equivalent would be asking to talk to a human instead of the automated agent. When initiated by the user, such requests tend to be attempts to abort the conversation due to failure and frustration. However, this pattern could also apply if the user already knows that what he or she needs cannot be fulfilled by the agent. Because *transferring* the user is an action that will terminate the current conversation, the agent should ask the user to confirm this action before executing it (Pattern C5.0, Example 7.35).

Pattern C5.0 Transfer Successful.

`U: TRANSFER REQUEST`
`A: CERTAINTY CHECK`
`U: CONFIRMATION`
`A: ACKNOWLEDGMENT. HOLD REQUEST`
`<transfer>`

Example 7.35 Transfer Successful.

U: **`I want to talk to a human`**
A: `Are you sure?`
U: **`yes`**
A: `Okay, I will connect you with a human. One moment please.`
`((transfer))`

In this pattern, the user says “I want to talk to a human,” “can I speak to a person,” “transfer me to a real person,” and the like (line 1). As a preliminary to granting the request, the agent checks to see if the user is certain (line 2) since the requested action is irreversible. When the user confirms (line 3), the agent acknowledges the request and asks the user to hold (line 4). The transfer is then granted (line 5). Transferring to a human agent is a common practice with service-oriented virtual agents for taking over when they fail to understand the user’s request or when the user’s request is too delicate for a machine to handle, for example, closing an account or seeking advice on suicide prevention.

If the agent second-guesses the user’s request to be transferred to a human, the agent must be able to respond if the user changes his or her mind (Pattern C5.1, Example 7.36).

Pattern C5.1 Transfer Aborted.

```
1  U:  TRANSFER REQUEST
2  A:  CERTAINTY CHECK
3  U:  DISCONFIRMATION
4  A:  SUBSEQUENT OFFER OF HELP
5  U:  CONFIRMATION
6  A:  ACKNOWLEDGMENT. OFFER OF HELP
```

Example 7.36 Transfer Aborted.

```
1  U:  I want to talk to a person
2  A:  Are you sure?
3  U:  not really
4  A:  Would you like me to try again?
5  U:  yes
6  A:  Okay. How can I help you?
```

With this variation, the user disconfirms (line 3) the agent’s certainty check (line 2). The agent then offers to try again to help the user (line 4). Here the user can simply make another inquiry or request or can confirm the subsequent offer (line 5). The agent will then recycle the initial offer of help from the beginning of the conversation (line 6).

Another way in which the user may disengage from the agent is by telling the agent that he or she is talking to someone else. When a virtual agent communicates through a room speaker with a far-field microphone array, in other words, a “smart speaker” like Amazon Echo or Google Home, it is not uncommon for the user to talk to other people in the room. In this open-room setting, agents sometimes attempt to respond to utterances that are not directed to them. To manage this situation, users should be able to correct the agent (Pattern C5.2, Example 7.37).

Pattern C5.2 Recipient Correction.

```
1  U:  <ANY UTTERANCE>
2  A:  <RESPONSE>
3  U:  RECIPIENT CORRECTION
4  U:  <ANY UTTERANCE>
5  U:  <ANY UTTERANCE>
6  U:  <ANY UTTERANCE>
7  U:  SUMMONS
8  A:  SUBSEQUENT GREETING
```

Example 7.37 Recipient Correction.

```
1  U:  what time are we going?
2  A:  I don't know.
3  U:  I'm not talking to you
4  U:  what time are we going?
5  U:  great
6  U:  I'll see you then!
7  U:  are you there, Alma?
8  A:  Hello again!
```

When the agent inappropriately responds (line 2) to an utterance directed to someone else (line 1), the user should be able to correct the agent with utterances like "I'm not talking to you," "I'm talking to someone else, Alma," "not you, Alma," and the like (line 3). The agent will then withhold all responses until the user actively re-engages it with a summons (line 7). The agent is still listening and classifying the user's utterances as he or she talks to someone else (lines 4–6), but a context variable *withhold* set to true prevents it from responding. Technically, the agent gives empty responses after every user utterance (lines 3–6), which the user does not hear or see.

Re-engaging the agent after a recipient correction (line 3) can be done with a summons (line 7). In order to reduce false recognitions of summonses as the user speaks to another person, the intent confidence threshold should be set high, say to 0.95 or higher. When the user summons the agent again with "are you there, Alma?," "hi Alma," "okay, I'm done, Alma," and the like, the agent does a subsequent greeting (line 8), setting *withhold* to false, and begins responding to the user again. This pattern therefore supports temporary disengagements from the user that are different from conversation closing. The high-confidence summons can also be used in an open-room setting to re-engage the user after a conversation closing (C4).

Finally, the agent may occasionally disengage from the user under special circumstances. If the user reveals that he or she is a minor who is only allowed to interact with adults, the agent may close the conversation or disengage. If the agent detects that the user is another virtual agent or chatbot aiming to exploit the service, the agent may close or disengage. Or if the user behaves inappropriately, using abusive, profane, or sexual language, you may wish your agent to disengage or close the conversation.[1]

To avoid abusive conversations, you can use a pattern similar to conversation closing (C4) but initiated by abusive language from the user instead of a pre-closing or farewell (Pattern C5.3, Example 7.38).

Pattern C5.3 Offense Complaint (Agent).

```
 1  U:  <ANY UTTERANCE>
 2  A:  <ANY UTTERANCE>
 3  U:  FIRST INSULT
 4  A:  OFFENSE COMPLAINT
 5  U:  <ANY UTTERANCE>
 6  A:  <ANY UTTERANCE>
 7  U:  SECOND INSULT
 8  A:  FAREWELL
 9  U:  <ANY UTTERANCE>
10  U:  <ANY UTTERANCE>
11  U:  <ANY UTTERANCE>
12  U:  APOLOGY/COMPLIMENT
13  A:  APPRECIATION. OFFER OF HELP
```

Example 7.38 Offense Complaint (Agent).

```
 1  U:  what's in theaters?
 2  A:  What do you mean?
 3  U:  you're not too smart
 4  A:  Now you're hurting my feelings.
 5  U:  what movies are playing?
 6  A:  I'm sorry, I still don't understand.
 7  U:  you suck
 8  A:  Goodbye.
 9  U:  wait
10  U:  I'm still talking to you
11  U:  I want to look up movies
12  U:  I'm sorry
13  A:  Thank you. What do you need?
```

---

[1] Amazon appears to have implemented a similar feature recently in response to complaints about Alexa's responses to sexually harassing utterances [Fessler 2017].

In this example, the user insults the agent (line 3) after it fails to understand (line 2), but it could be after any utterance by the agent. The *insult* should be distinguished from the *complaint*: while the latter may be about a topic mentioned by the agent or about its organization's policies, the former is about the agent itself. The intent for insults includes phrases like "you're stupid!," "you suck!," "f*** off," "you're the worst chatbot I've talked to!" and the like. In our implementation, Alma will endure two insults before closing the conversation. After the first insult (line 3), Alma complains that it takes offense at the insult (line 4). At this point, Alma will continue to talk to the user normally. However, if the user insults it a second time (line 7), Alma will bid farewell immediately without doing the usual last-topic check (line 8). After the second insult, the *withhold* variable is set to true, and the agent will withhold any response to the user (lines 9–11). Alma will only re-engage if the user does one of two actions: an *apology*, for example, "I'm sorry," "sorry," "I apologize," or a *compliment*, for example, "you are awesome!," "you're the best," "you rock!" Either of these types of utterances will reset the insult counter at any time, and Alma will re-engage with an appreciation (line 13). In short, through this pattern, the agent refuses to participate in abusive or otherwise inappropriate conversation.

Some have suggested that voice assistants may inadvertently be teaching children bad manners because they do not require them to say "please" or "thanks" [Walk 2016, Shellenbarger 2018]. But we do not have to design our conversational agents to act like slaves. When designing your agent's *persona* (Chapter 8), decide how obedient you want it to be and what level of respect it will demand. For example, to an unmitigated imperative, "turn on the lights!," your agent might respond, "Yes, Master." But to an imperative mitigated with the keyword *please*, "please turn on the lights," it might respond instead with, "Okay." To an interrogative, "could you turn on the lights?," it might say "Of course." And to a declarative, "it's really dark in here," it could respond with an offer, "Would you like me to turn on the lights?" Differentiating the form of the user's request can be used to give your agent a bit of both personality and emotional intelligence. Furthermore, as we saw in Chapters 4 and 6, *appreciations* like "thanks" are not simply about manners. They also perform conversation management functions, such as closing a sequence when in third position (Pattern B4.2). Teaching users of *any* age to say "thanks," perhaps by occasionally inserting "what do you say?" when the user fails to respond in third position, will improve the interaction in the long run, as well as your ability to measure successful sequences.

In the agent *offense complaint* pattern (C5.3), we can also see an example of *emotion*. In this case, Alma claims that the user "hurt my feelings" (Example 7.38, line 4). Of course, Alma has no feelings; however, the agent is *doing* "having hurt feelings," that is, claiming them in response to personal insults by the user. This is just one example of how conversational agents, or people, can *do* emotion. In our approach, we do not treat emotion as a separate layer running above the dialog logic itself. Just like humans, the agent must "express emotions" through *emotion-laden actions*. But virtual agents, like even humans on some occasions, do not feel the emotions they claim. ***From this perspective, emotion-laden actions are the same as any other conversational actions: they are built through intents, entities, context variables, and agent utterances.*** Our functional conception of "emotion," like our functional conception of "understanding," is adopted from the fields of ethnomethodology and Conversation Analysis [Ruusuvuori 2013].

Finally, just as the *agent* may "take offense" at some things the user says, the same is true for the *user*. Virtual agents may intentionally or unintentionally (due to poor understanding) say things that some users find offensive. For example, Schlesinger et al. [2018] argue that today's machine-learning-based (or generation-based) chatbots have difficulty talking about race because they sometimes learn disrespectful and even racist responses from data. In these cases, users may tell the agent that it has said something offensive. If the agent fails to understand such complaints, the user may disengage from the agent. To avoid this, the agent should acknowledge the complaint and learn from the user to avoid offending other users in the future (Pattern C5.4, Example 7.39).

Pattern C5.4 Offense Complaint (User).

```
1  U:  <ANY UTTERANCE>
2  A:  OFFENSE COMPLAINT
3  U:  APOLOGY
```

Example 7.39 Offense Complaint (User).

```
1  A:  Make me One with everything.
2  U:  you can't say that!
3  A:  Sorry. I didn't mean to offend.
```

After telling a question-answer joke, "What did the Buddhist say to the hot dog vendor?" (not shown), the agent gives the punchline, "Make me One with everything" (line 1). To this, the user complains that it is offensive with "you can't say that!," "that's inappropriate," "I find that offensive," and the like (line 2). In response, the agent apologizes (line 3). At the same time, the dialog logic saves the offending response (line 1) to the context variable *agent offense* for the designer to inspect later and decide whether to remove. Like a human learning a language and a culture, conversational agents will also sometimes say offensive things inadvertently. If you build an interactional feedback mechanism into the conversation, you and your agent can learn from mistakes.

7.6 Conclusion

In this chapter, we outlined multiple types of UX patterns for managing a conversation at the level of the conversation or encounter. These include patterns for opening the conversation by the agent (C1) or by the user (C2), for discussing the capabilities of the agent (C3) and therefore for determining the appropriateness of the encounter in the first place, and for ending the conversation, either by closing it (C4) or disengaging in other ways (C5). From this chapter, you should gain a technical understanding of how to design users' entry into and exit out of sessions with your agent using the Natural Conversation Framework. You should understand the conversational functions that these activities serve so that you are not tempted to dismiss them as mere "pleasantries" or "chit chat."

Taken together, Chapters 6–8 provide a starting set of 100 UX patterns for enabling several generic conversational activities and their management at the sequence level and at the conversation level. Although they are based on patterns of what humans naturally do in conversation, they are simplified versions of those patterns. The patterns presented should provide a reference or starting point for many use cases, especially for service encounters. To demonstrate how these patterns can be combined into a single conversation, we offer an example of a conversation with Alma (Appendix D). This sample conversation contains many of the patterns outlined above, although not all of them. It was conducted through voice only, using a custom-built smart speaker, and it lasts over seven minutes. The unrelated topics discussed in this sample conversation reveal that it is not from an application but rather from our generic conversation space. With each new application, we start with this generic conversation space and replace the content with the use-case-specific content of the application, such as travel plans, healthcare benefits, fast-food ordering, movie recommendations, etc.

In addition to providing a reference for UX design, the NCF pattern language offers a yardstick against which the* conversational competence *of any chatbot or voice assistant can be measured. Somewhat like the so-called TRINDI "tick list" [Bohlin et al. 1999, Bickmore et al. 2018], which proposes 12 general capabilities that any dialogue system should demonstrate, the NCF pattern language offers 100 generic interaction patterns that can be used to score the capabilities of any natural-language interface. While the TRINDI capabilities are more general, for example, #5 "Can the system deal with subdialogues initiated by the user?" or #10 "Can the system deal with unspecific information?," the NCF capabilities are organized more in terms of specific conversational functions, for example, B2.1.0 *repeat request* initiated by the user or C4.0 *last-topic check* initiated by the agent. Furthermore, the NCF pattern language adopts the conceptual framework of Conversation Analysis; thus, for example, it uses terms like "sequence expansion" instead of "subdialogue," which provide greater precision in specifying conversational structure.

Although the NCF pattern language provides many generic patterns, it is not exhaustive, even of the basic conversational activities (A). For example, *informing* is another generic sequence type. One party does an informing, perhaps with a pre-informing, and the recipient acknowledges it. In some cases, the information conveyed may constitute good or bad news for the recipient, so special care is taken in the delivery and follow-up inquiries. Such informings are commonly seen in medical settings as well as ordinary conversations [Maynard 2003]. Therefore, our pattern language for conversational UX patterns is an ongoing project that can be expanded over time and with the help of fellow designers and conversation analysts.

8

Conversational UX Design Process

Now that you know some conversation theory, the basics of conversation authoring, a conceptual framework for natural conversation, and a corresponding UX pattern language, it is time to put them all together. In this chapter we outline a general design process, which will be familiar to many UX designers, and offer ways to adapt it to the unique challenges with conversational UX.

8.1 Design Thinking

Like any type of user interface, *conversational* interfaces should be designed before they are built. "User-centered design" [Norman 1988] approaches insure that the end product takes users' needs and perspectives into consideration from the beginning of the design process. The final result should be a solution that is tailored to the end user. In this vein, we adopt the general user experience design approach, "design thinking" [Archer 1965, Brown 2009, Plattner et al 2010] as our starting point. The user experience design process has been abstracted to the degree that it can be applied across disciplines. Fundamentally, design thinking is a process for approaching and solving problems of any kind, whether you are designing a mobile application, an automobile dashboard, or a better toothbrush.

While there are different formulations of design thinking, the five-stage Design Thinking model proposed by the Hasso-Plattner Institute of Design at Stanford, or "d.school" [Plattner 2018], offers a simple, concise model (Figure 8.1). In short, this model instructs you to get to know your users, clearly define their goals and problems, generate new solutions, create simple versions of your solutions, and test (and refine) them with users. While this general model applies to any kind of application, we outline user research methods, brainstorming exercises, and authoring methods that specifically address conversation design issues. These methods are

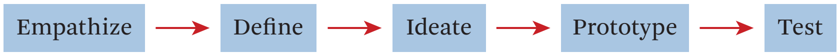


Figure 8.1 Five-stage Design Thinking model.

derived in part from practices in the field of Conversation Analysis and in part from our experience over the past four years building conversational agents.

Remember that the design-thinking process itself will not always lead you in the direction of a conversational-agent solution. Conversational interfaces are relatively new so development teams are still experimenting with how to design them and learning the kinds of use cases for which they are best suited. One complicating factor is that comparing *mature* graphical interface designs with *immature* natural-language interface designs yields biased results. Until more mature systems and better comparative data are available, we cannot say definitively which use cases are most appropriate for conversational interfaces (and which interaction styles). But we believe that certain use cases are promising, given the unique affordances[1] of such interfaces (Table 8.1).

The one affordance of conversational interfaces that has shown the most promise to date is hands-free and gaze-free interaction (i.e., without having to look at a display) through *voice*. Today's voice assistants, such as Amazon Alexa and Google Assistant, have capitalized on voice-only interaction in the living room and throughout the home. And voice-based interaction has been used in the automobile for a long time because it does not require drivers to take their eyes off the road. Such ubiquitous computing or Internet of Things (IoT) use cases, in which computing is embedded in everyday objects or in the environment itself, are a good fit for voice-based, conversational interfaces because users can still interact with an agent even when there is no dedicated screen.

Another affordance of conversational interfaces is that they are compatible with standard channels through which humans already converse. Because such interfaces can be used through the words alone (if they observe a conversation-first design strategy), they can be used with standard voice or text communication platforms, such as telephones, Short Message Service (SMS), or Instant Messaging (IM). This affordance enables a conversational agent to reach users even when they have not downloaded your mobile application or navigated to your website. For

---

[1] We use "affordance" in Gibson's [1979, p. 128] sense of the term as an action that an object enables a human or animal to take with respect to it. For example, many stable surfaces that are knee-high above the ground afford sitting on.

Table 8.1 Affordances and example use cases for conversational interfaces

Affordances	Example Use Cases
Interact hands free and gaze free	In-room hotel concierge; in-car navigation agent
Reach mobile users without an app	Alert customers to flight cancellations and rebook
Access information, not documents	Answering general information questions; answering health insurance questions
Adapt to user's level of expertise	IT troubleshooting; physics tutor
Capture personalized user information	Recommend travel destinations; recommend health plans
Interact naturally with a humanoid character	Virtual brand mascot; elder-care robot

example, an airline can use a conversational agent to contact customers through the standard SMS apps on all mobile phones, or even through the telephone, to alert them about cancelled flights and to book them on a new flight. Such an approach enables you to use these standard communication platforms for humans as your application's frontend instead of building your own.

While search engines currently enable users to query the worldwide web through natural language, they typically fetch a set of documents rather than provide an answer to the user's question. Conversational interfaces, on the other hand, can be used to return bits of information directly to the user, freeing him or her from having to skim through documents to find them. In other words, conversational interfaces can enable users to access information at a finer level of granularity than web search, thereby pinpointing the needed information more quickly. When voice assistants, such as Siri or Alexa, can find a bit of information, asking them through a simple inquiry can be much quicker than performing a web search (of course, when they cannot find it, they may default to web search).

Conversational interfaces should be designed to adapt to the user's displayed level of expertise. This was discussed earlier under the principle of "recipient design" (Chapter 2) and the *open request* pattern (Chapter 5). That is, if the user displays expertise by providing all the required information using the preferred terminology in a single turn, the agent should fulfill the request immediately without asking redundant questions (e.g., patterns A2.0-A2.1). On the other hand, if the user behaves like a novice by failing to provide all the required information or using ambiguous terminology, the agent should adapt and teach the user the preferred behaviors, by requesting details (e.g., patterns A2.2-A2.6) and handling repairs (e.g., patterns A2.11 and B1-B3). This will make the application easier to use for a wider range of users and will help move them from novice to expert. For example, troubleshooting IT problems requires both knowledge of the technology (e.g., how a cell phone works and what its various parts are called) and knowledge of the business process (e.g., what identifying information the customer service representative needs in order to authenticate the user and the phone). A conversational agent should be able to adapt to varying levels of knowledge on the part of the user in some of the same ways that human agents do.

Because conversational agents *talk* to users, they can collect rich personalized information about customers. Unlike menu-driven graphical interfaces, in which users merely select from sets of predefined options, conversational interfaces enable you to "hear" the customer in his or her own words. Not only can you learn the terminology that customers use, you can learn things that customers volunteer without being asked. For example, by describing their travel plans and preferences in their own words, customers provide richer, more personalized information for recommending travel destinations than can be communicated by checking filters in a sortable list. Although your agent may not be able to recognize and use unexpected information the first time it is encountered, it nonetheless *captures* the information in your chat logs so you can build it into your agent's understanding over time.

Finally, although perhaps a specialized use case, *humanoid characters*, such as brand mascots (e.g., Ronald McDonald), fictional characters (e.g., Harry Potter), or physical robots are natural use cases for conversational interfaces. If the system simulates the physical aspects of a humanoid character, either virtually on a screen or virtual reality (VR) headset or through a robot, such characters usually also need a voice. And by "voice" we not only mean a synthesized acoustic voice but also the ability to *converse*, whether vocally or through text. Uttering pre-scripted monologues through a natural-sounding voice is not enough to create an intelligent character.

These are just a few of the types of use cases that we believe are well suited to the affordances of conversational interfaces. Our concern throughout this book has been to explore ways to design a natural language-based interface that works more like a natural conversation than the other typical interaction styles (Chapter 1). The remainder of this chapter starts from the assumption that your stakeholders and development team are committed to creating a better conversational agent.

8.2 Stage 1: Empathize

Many good designers recognize that creating a great user experience requires empathy with your prospective users. ***"Empathy" means developing an understanding, not only of your users' needs, but also of their meanings and motivations, as well as their behaviors.*** The better you can see the world from your users' perspectives, the better you can design an effective and engaging experience for them. So how do you get to know your user in order to create an effective conversation space? Observe and engage them.

Observe

The best way to develop an understanding of users, in the context of designing a conversational system, is to collect examples of the type of conversation you would like to design. What are the primary forms of talk involved in your use case? Is it answering inquiries, fulfilling requests and commands, recommending goods and services, troubleshooting problems, teaching material, etc.? This can be done in a number of ways.

If time and budgetary constraints allow, go out into the field yourself and observe the user conversations firsthand. Ethnographic field studies can enable you to see the big picture in the social setting you wish to approximate. You can observe the rich detail of how conversations take place in the context of the embodied behaviors, physical artifacts, and environments involved. Studying a conversation in person will allow you to see facial expressions and hand gestures in tandem with the content and any inflectional changes. While field studies typically involve observing people in a physical space, they can also be adapted to online and virtual spaces.

If you were tasked with creating a conversational agent that takes orders for coffee, go to a coffeehouse and see how people do it! ***Of course, we all know how to order coffee, but when we rely on our memories alone, many of the taken-for-granted details of the conversation are missed.*** As an ethnographic observer, you can focus on the details of how people accomplish a social activity in a way that you cannot when you are participating in the activity yourself. How do customers see where to go to place an order versus to pick it up? How do they manage the line while waiting to speak to the order taker? How is the conversation opened? How is it closed? How do customers refer to the different kinds of coffee or foods they want? How does the order taker refer to those same things? How do the two confirm the order and manage misunderstanding more generally? Noticing how each of these are done in their concrete details will help you in designing the features of your conversation space.

While direct observation is an extremely rich method for understanding your prospective users' behavior, challenges arise because you cannot recall every detail you see or hear. For this reason, we advise *recording* the social settings you observe whenever possible. Mechanical recording enables you to "get your hands on" the details of social interaction [Sacks 1984, p. 26]. If the setting you observe is not public, you may need to gain the consent of those you wish to record or at least to inform them of their right to opt out of being recorded. In addition, if recording in a place of business, you should ask permission from the business owner or operator.

In the age of the smartphone, audio and video recording are cheap. We carry around powerful audiovisual recording devices, and even editing tools, in our pockets all the time. And with the proliferation of smartphones, people have become more comfortable with being recorded. So record as much as possible! Even clips of scenes in which nothing seemed to have happened at the time can reveal insights on closer inspection after the fact. Collect clips of routine and unusual activities alike. It is often the "deviant case," the one that appears not to fit the pattern, that can lead to the greatest insights [Schegloff 1968].

One major advantage of recordings of human conversations is that they can be studied even when you cannot be there to observe those conversations directly. Ask the client whether they have existing recordings or transcripts that capture the kind of conversation they wish you to build or a similar kind of conversation. For example, clients often already possess recordings of telephone calls between customers and *human* customer-service agents engaged in the same kind of conversations they intend for the automated agent. Request a sample of such recordings and study both their content and their form, that is, the reasons that customers call the company, as well as the interactional structure of those conversations. Listening to recordings of customer service calls can be time-consuming, but it is an excellent way to hear the voice of the customer, literally. If such recordings do not exist, ask the client organization for any materials it might have on its interactions with its customers, such as online chat logs or forums.

If you are trying to create an agent that can handle technical support, get access to recorded calls from support teams or written transcripts generated from call systems. Similarly, if you are trying to create a conversational sales agent, get your hands on voice recordings of sales calls. Even if the support staff dealt with a frustrated customer and was hung up on, that conversation is valuable, perhaps extremely so. Understanding what went wrong can be just as effective in your design process as knowing what went right. Try to get a well-rounded sample of recordings and transcripts to unlock the big picture of your users' needs.

Finally, consult the Conversation Analysis literature, which we covered in Chapter 2. ***Other people have likely already observed the form of talk or setting that you will need to build and have analyzed it in a systematic way.*** Remember, although you may not find a study of the particular topic area of your agent (e.g., travel, health insurance, technology support, etc.), you will likely find prior studies of the form of talk involved (e.g., service encounters, teaching, counseling, etc.). Any studies you find will describe the mechanics of how the conversation is organized in that setting. The literatures in Conversation Analysis and related fields of social science contain many studies of human conversation in service encounters [Whalen and Zimmerman 1987, Jefferson and Lee 1992, Whalen and Vinkhuyzen 2000, Moore 2008, Félix-Brasdefer and César 2015, Szymanski and Moore 2018], classrooms [McHoul 1990, Szymanski 1999], psychotherapy [Antaki 2008, Peräkylä 2008], medicine [Maynard 2003, Pilnick et al. 2010], courtrooms [Drew 1992, Manzo 1997], news interviews [Clayman 1992, Greatbatch 1992], airplane cockpits [Nevile 2002, Arminen and Auvinen 2013], and much more.

Engage

In addition to observing prospective users in the kinds of social settings related to your use case, it is also useful, of course, to talk to them. ***Through ethnographic interviewing, in which you ask users open-ended questions and encourage them to talk, you can learn about meanings, attitudes, and motivations that may not be as apparent from observing their behavior.*** In interviews, users can tell you stories about past experiences and how those influence their current goals and pain points. The interview can ultimately help you understand a user's mental model and the wider context of his or her behavior.

Compared to direct observation, interviewing prospective users is generally more feasible and less costly. Interviewing can be conducted under a wide range of situations and channels. Although face-to-face and on-site is the richest situation in which to interview people, because features of the environment can trigger questions that you would not have thought of, critical insights into your users' perspectives can be gleaned from off-site interviews and focus groups, telephone interviews, and even email interviews.

Since interviews are more practical than direct observation or video collection, you may be tempted to use them exclusively for user research. But be careful: what users *say* they do and what they *actually* do are two different things. By talking to people about their behavior, you will get idealized accounts, abstracted accounts, and biased accounts, in addition to informative ones. And when it comes to conversational behavior, people are not good at telling you the details of *how* they talk but focus instead on the topics they talk about. The bottom line is to find some way to gain access to a sample of your prospective users, whether through observation, engagement, or both, within the constraints of your projects, and to build empathy with them.

8.3 Stage 2: Define

After conducting user research as outlined above, you should define your users' needs more formally. Synthesize your findings and represent them through artifacts that can be shared with the development team and stakeholders. Design artifacts enable everyone on the team to understand the design requirements and thus help them achieve more consistency across the various parts of the application. The standard design artifacts include user personas and outcome statements. In the next section we introduce additional design artifacts that are tailored specifically for conversational systems.

User Personas

User personas should be fictitious representations of your users' qualities, yet they work best when grounded in your research, as a kind of summary of your findings. Depending on the scale of your project, a rule of thumb is to devise one to three personas. When creating a persona, come up with a name and establish some personal details. For example, consider a restaurant customer who typically orders takeout. Your research should lead you to establish some personal details. In this case, your persona might be named Janet, a woman in her early 40s, who is the mother of two teenagers. She works full-time so she has little time to cook or shop for groceries. These personal details help you and your team empathize with users and be better able to design for them, as well as to coordinate the design across your team members.

After these preliminary steps, outline the motivations and pain points of your personas. What drives your users to engage with a conversational agent? Are they trying to fix something, or are they looking for a product recommendation? Are they primarily motivated by cost or by quality? Are they looking for a more natural way to interact with a system, or do they need a hands-free interface? Understanding what compels your users to engage is crucial in designing your conversational agent. Your personas should also reflect your users' pain points. What bothers them most about the current process? Are they frustrated when they are transferred from one human to another just to get a question answered? Are they unable to find personalized recommendations in a sea of online reviews? Do they have trouble understanding technical content, such as health insurance policies? In the example above, one of Janet's major pain points is fumbling through user interfaces when trying to order food. She finds that placing a food order for three people is too complicated through a mobile app, with multiple screens on a small display. She might find ordering food through conversation, as she is used to doing in a restaurant, easier and more natural. Since she texts her kids on a daily basis, a conversational agent integrated with SMS might be especially appealing to her.

By the time you have created one to three user personas, you probably have a laundry list of notes including frustrations users may have and goals they would like to accomplish. Compile all the pain points and user goals you can find. In doing so, you will probably find that many of the pain points lay the foundation for user goals. For example, when a user says something like, "I always get stuck trying to place a food order," a goal might be, "Easily order food through Product or Service X." Translate as many of these pain points as you can into user goals—we like to write each goal on a sticky note to help drive the next phases of the process.

Outcome Statements

Users' goals lay the foundation for the core outcomes your application should be able to handle. There are many ways to approach writing outcome statements, but from our perspective they hinge on grouping common user goals into broad statements. Determine the importance of each goal to your end user and its feasibility with respect to your technical, organizational, and financial constraints. Identify those goals that are both highly important and highly feasible. These "must-have" goals will guide the first phase of your project and your *minimum viable product* (MVP).

From the must-have user goals, you can derive core outcome statements that address common themes. These outcome statements should involve your user, the action taken, and a concrete and quantifiable result. Types of outcome statements might be:

- Customers can chat with an agent to receive detailed product information without going to the company website.
- Users can receive destination recommendations that are personalized to their travel preferences.
- Healthcare providers can access medical information quickly, without having to open and skim multiple documents returned as search results.

Clear and simple outcome statements are a critical resource for achieving shared understanding across the development team about what you are creating and for coordinating the team in terms of the priority of different features. Crafting useful outcome statements requires you and your team to be clear and concise in defining your project goals up front. Outcome statements can then serve as reminders throughout the development process of what you are working toward. And, of course, outcome statements should be updated as requirements and constraints change.

8.4 Stage 3: Ideate

We have briefly outlined the standard stages in a design thinking process with some recommendations for adapting it to the development of conversational agents. We now turn to design thinking exercises and design artifacts *specifically* created for conversational UX. Conversational agents have certain distinctive features not found in other kinds of applications or projects. Therefore, design exercises tailored to the unique challenges of conversational UX are helpful. In this section, we describe three exercises for generating design insights regarding conversational agents: The Shape of a Conversation, Create an Agent Persona, and Utterance Pair Brainstorming.

Exercise 1: The Shape of a Conversation

In order to design a conversation space, you must have a clear idea of how conversations are structured. Too often natural conversation is mischaracterized as simply “natural language” [Button et al. 1995, p. 209] that is composed entirely of sentences and is not interactive. But natural conversation is a particular form of natural-language use that displays its own characteristics. It is composed of *utterances*, not sentences, and it is organized through a *turn-taking system* [Sacks et al. 1974]. ***Although we all know how to* do *conversation, we take the practices and mechanics of natural conversation largely for granted.*** Consequently, it is surprisingly difficult to articulate how conversation works. But to get your stakeholders and development team to understand the form of natural conversation, you can use the following group exercise.

In Conversation Analysis (CA), practitioners have a tradition of analyzing recordings of conversations together, known as “data sessions” [Ten Have 1999, p. 123]. In data sessions, the participants analyze a recording of a spate of social interaction, usually involving talk, along with a detailed transcript of that interaction. They share observations, or “noticings,” of features of the interaction and their analyses of how those features are organized. Insights and analyses are thus generated collaboratively. Here we adapt the data session format for the purposes of design thinking. This exercise requires a sample of a naturally occurring human conversation, recorded in audio or video, and a CA-style, or detailed, transcript like the one below (Example 8.1; see Appendix A for an explanation of transcription conventions).

Example 8.1 [NB I.2:Golf][2].

```
 1  D:  G'morning. San Juan Hills Country Club?
 2  G:  Guh morning. What's-w- what kind of a starting
 3      time ken:: we get fer::hh sometime this
 4      afternoon.
 5         (0.7)
 6  G:  Any[time-
 7  D:     [Oh:::, [let's see.
 8  G:             [Any time tuhday.
 9  D:  Two fordy. One, thirdy.
10  G:  One thirty?
11  D:  Mm hm::?
12  G:  One thirty.
13         (0.7)
14  G:  .hh W'l at sounds like a good time?
15         (0.4)
16  D:  What is the name?
17  G:  Detweiler. D-e-t,
18         (1.2)
19  G:  w-e,
20         (0.4)
21  G:  i-l-e-r-.
22         (2.0)
23  D:  Foursome?
24  G:  Yah.
25         (0.4)
26  D:  Electric carts?
27         (0.6)
28  G:  Uh:::, n:no? I don't think so.
29  D:  Okay. We'll see yuh then,
30  G:  Righto,
31  D:  Mm hm, Bye?
```

---

[2] This transcript can be found online at Talk Bank: https://ca.talkbank.org/browser/index.php?url=Jefferson/NB/01golf.cha.

With a recording and a transcript in hand, facilitate the session by following these steps:

1. Play the entire recording once.
2. Play the entire recording again while reading the transcript.
3. Replay any parts the participants want to hear again.
4. Take 10–15 minutes to compose observations.
5. Ask participants to share their analyses in round-robin format.
6. Open the floor for group discussion.

The data session participants may feel a bit shy, not knowing quite "what you are looking for" or "what I'm supposed to see." This is an unfamiliar way of looking at conversation for most people. To facilitate the participants' insights, ask them the following questions:

- How do speakers take turns talking?
- How often do they overlap their utterances or interrupt?
- How many words are in their utterances?
- How long do the silences between turns last?
- What else can you see?

Some things you will observe: Most utterances are not sentences. They are shorter phrases or single words that incrementally build on prior utterances. Utterances by different speakers tend to be done with minimal gap and overlap. In other words, they are precisely timed and tightly coordinated. Most silences in a conversation are under one second. The topics of the conversation, for example "golf," often go unmentioned. These are just a few of the kinds of features of human conversation that will give your participants a feel for its shape. Of course there are many other kinds of things to notice and insights to draw from conversational data, but this is a good set for beginning analysts of natural conversation.

Exercise 2: Create an Agent Persona

While personas can help your team maintain a consistent conception of the intended user(s) of your application, they can do the same for that of your conversational agent. When designing an *agent* persona, it is best to do it in tandem with a branding exercise. The persona you design for your agent will inevitably reflect your client's brand, be it a small business, corporation, or independent professional. So when you design the persona, what values do you want it to reflect? Do those values fall in line with your business's mission? If you are an insurance company with brand values that align with responsibility and trust, you may want to design a personality that is friendly and personable, but at the same time is mostly serious. On the other hand, if you are a coffeehouse or a clothing store for young people, you may want to design a personality that is more fun and witty, with even a bit of attitude. When designing an agent persona, it is best to sit down with the marketing and brand identity team to get aligned on the client's values. An agent persona will help the team maintain a consistent voice and personality for your application. ***In designing how your agent will talk and how it will behave, you will inevitably rely on mental images of a*** **person** ***doing and saying things.*** To make these mental images, across designers and developers more systematic, an agent persona is extremely useful. When creating your agent's persona, consider the following things: agent job description, agent personality, and agent self-knowledge.

Agent Job Description

Write a job description for the work your virtual agent is intended to do. For example, the job might be to provide answers to general health insurance questions, or to recommend travel destinations and places to stay, or to provide information on movies, or to provide assistance in analyzing business intelligence data. List the core functions of your application as if they were duties for a position (Figure 8.2). Is the agent's job similar to a job that humans do, or is it unique to artificial intelligence? If your agent is a *travel agent*, what user expectations might go along with that role? Travel agents (where they still exist) provide detailed information about travel destinations, make recommendations for places to stay and things to do, and help you book plane tickets, hotels, and car rentals. If your virtual agent provides travel destination recommendations, your user may expect it to do these other things as well.

Also specify some qualifications that the agent should have to perform this job. If it is a virtual *travel agent*, it should have knowledge of travel destinations. If it is a *customer service representative* for a health insurance company, it should know about health insurance policies, at least generally, if not how they apply to a particular user. If it is a *movie critic*, it should have knowledge of movies, such as titles, ratings, and genres, and perhaps even show times or trivia. Use the agent's job description as a tool for summarizing and representing the functional scope of your application in a way that is easy for everyone to understand.

Virtual Travel Agent

Acme Hotels is seeking a dedicated travel agent to assist potential and returning customers in discovering travel destinations and making travel arrangements.

Responsibilities and Duties:

- Recommend travel destinations based on the customer's preferences
- Recommend things to do in particular destinations based on the customer's preferences
- Recommend and inform about Acme Hotels worldwide
- Assist customers in booking hotel rooms, rental cars, and flights

Qualifications:

- Detailed knowledge of travel destinations, including basic information, weather conditions, and popular things to do in the world's major travel destinations
- The ability to interact with booking systems for hotels, airlines, and car rental companies
- The ability to make personalized recommendations, not simply popular recommendations

Figure 8.2 (Virtual) travel agent job description.

Agent Personality

Because your application will talk, it will necessarily have some kind of conversational style [Bennett 2018]. Whether intentional or not, you and other conversation or content designers will create a mental image of the agent when authoring its utterances and designing its behavior. Externalizing and coordinating these images through an agent persona will enable you to achieve a more consistent conversational style, or "voice." You should design the imagined personality of your virtual agent to share with stakeholders and the development team.

What gender is your agent, if any? Standard voices from text-to-speech technologies tend to sound male or female. Choosing a gendered voice may have implications for other characteristics of your agent persona. On the other hand, agents' voices can also be robotic or gender-neutral. However, people tend to prefer to refer to characters as "he" or "she" rather than the more objectifying "it." Even with text-only interfaces, users will tend to attribute some gender to your agent, as this cultural concept is ingrained in almost all natural languages.

Consider also the gender politics of your users and of your client. Do they embrace traditional conceptions of gender? Do they challenge them? Or both? Some have seen sexism in the choice of female personas for the majority of today's voice assistants [Fessler 2017]. Others have suggested that both men and women tend to prefer female voices [Mitchell et al. 2011]. While we do not recommend any particular position on the gender politics of your agent persona, we do recommend that you discuss this issue with your stakeholders early in the design process before committing to an agent persona that your users may find offensive.

Regardless of gender, what kind of person is your agent? Is it helpful? Is it knowledgeable? Is it a genius? Is it talkative or reserved? Does it have a sense of humor? If so, what kind? Is your agent always professional or a bit playful or too serious? Is it obedient or does it assert its own opinion? Is it submissive if you insult it or does it demand civility (Pattern C5.3)? Make the various personal characteristics that you want your agent to have explicit for the stakeholders and the development team.

In order to dive deeper into the design of your agent's personality or conversational style [Bennett 2018], you may consult personality typologies in the popular psychology literature. For example, the Myers-Briggs Type Indicator [Myers 1962] is a personality typology that has been around for many years. It consists of four dimensions (Table 8.2). Is your agent's persona more or less talkative? How deep are the things it says? Is it more scientific or creative? Is it more rational or passionate? Is it decisive or open to possibilities? For example, your agent's persona may be that of a talkative (E) genius, who helps users make reasoned decisions (T), based on current data (S), which are binding (J) on topics, such as choice of health insurance plan or purchase of a vehicle. Or your agent's persona may be that of a counselor, who listens to what the user has to say (I), shows empathy (F) and offers insights (N) based on it, and invites him or her to revisit decisions (P) on topics, such as career goals or chronic depression. And of course many other combinations are possible. The Myers-Briggs typology [Myers 1962] includes sixteen ideal personalities based on these dimensions. Although your agent may not require such a complex personality type, such typologies can help you explore and think about the various dimensions of personality systematically. If you consider only one of these dimensions, consider extraversion/introversion, which can be approximated by the degree to which your agent initiates activities versus letting the user initiate them.

In addition to gender and personality, conversational style also involves a person's sense of humor. Humor is a general way to delight another person through

Table 8.2 Myers-Briggs type indicator (adapted)

Extraversion (E)	Introversion (I)
Frequent social interaction	Deep social interaction
Initiates more sequences	Initiates fewer sequences
Sensing (S)	**Intuition (N)**
Trust their senses	Trust their intuition
Presents more data	Presents more insight
Thinking (T)	**Feeling (F)**
Make decisions based on reason	Make decisions based on feeling
Provides rationale	Provides empathy
Judging (J)	**Perceiving (P)**
Make final decisions	Leave decisions open
Treats decisions as settled	Treats decisions as revisable

natural conversation. Different people demonstrate different styles of humor and different frequencies of humor. The Humor Styles Questionnaire [Martin et al. 2003] provides a typology of such humor styles: Affiliative, Self-Enhancing, Aggressive, and Self-Defeating. Affiliative humor makes light of situations that everyone can relate to. Self-Enhancing humor involves the amusement of the teller. Aggressive humor makes light of other individuals or groups. And Self-Defeating humor makes light of the teller.

Avoid Humor in Serious Activities. In general, use or avoid humor on the same occasions you would in human-to-human conversation. Doing "being professional" does not require the complete absence of humor, but rather its appropriate use. ***Humor can be unprofessional when it distracts from the business at hand or when it belittles superiors or other customers. However, humor strategically placed in the cracks and transitions of business can delight customers and thereby improve their satisfaction.*** As with human representatives of organizations, virtual agents should avoid aggressive humor at the risk of alienating particular customers.

Opportunities for Humor. Conversation *around* the core activities of your application may provide opportunities for your agent to exhibit a sense of humor. The openings and closings of natural conversations can be used for humor without distracting from the business at hand. For example, your agent may give witty responses to "how are you?" or may offer to tell the user a joke if he or she reports having a bad day. In addition, user-initiated "small talk" provides opportunities for humor. For example, "how's the weather?" might elicit a response like "Well, it's always freezing in the server room," or "did you see the game?" might receive, "I only follow chess. Deep Blue is my favorite player!" Because small-talk inquiries are *user*-initiated, they involve the user choosing to stray from the current topic rather than the agent, and therefore are less potentially distracting to users than when agents initiate small talk. One-line responses and even explicit invitations, "Shall we get back to planning your trip?," are ways the agent can attempt to steer the conversation back to the business at hand.

Agent Self-Knowledge

Finally, to achieve the interaction metaphor of a *human conversation*, your virtual agent should know something about itself. An AI system is not intelligent if it knows nothing about its history or its creators. In addition to being able to describe its own capabilities, your agent should be able to talk about its persona, at least to some degree.

What knowledge and how much of it your conversational agent should have depends in part on the type of persona. Consider how self-knowledge would vary for these three personas: a customer service agent, an artificial intelligence (e.g., Apple's Siri or IBM's Watson), or a fictional character or brand mascot (e.g., Harry Potter or Ronald McDonald). Customer service agents rarely say much about themselves personally, but they should know basic facts about the company they represent, as well as commonsense knowledge.

On the other hand, if your agent persona is that of an artificial intelligence, it may refer to the fact that it is not human. For example, if your virtual agent recommends destinations or hotels, it should be prepared to answer basic inquiries like "what's your favorite destination?" or "do you have a favorite hotel?" But because it is an AI, it might respond with, "I can't say. I never seem to get out of the server room." Such responses suggest very basic self-awareness.

And if your conversational agent is a fictional character or brand mascot, it should have some knowledge of its backstory. For example, a virtual Ronald McDonald should be able to show at least some recognition of Hamburglar or Grimace. Even a conversational agent identified as IBM's Watson should know something about when it beat two human champions at the TV trivia game Jeopardy in 2011. Using the user inquiry (A1) and extended telling (A3) patterns, you can enable your virtual agent to tell parts of its history or backstory and answer questions about it.

Taken together, your agent's persona consists of its job description, personality, or conversational style, sense of humor, and self-knowledge.

Exercise 3: Utterance Pair Brainstorming

Once you have prioritized the user goals you intend to support, crafted clear outcome statements, and created a detailed persona for your agent, you are ready to begin to determine what exactly your agent will recognize and say. Most conversational-agent projects we have observed begin by generating or obtaining lists of expected user utterances, or "intents." User researchers may solicit lists of utterances from expert users or the crowd or may obtain them from existing FAQs or transcripts of calls to human agents. This is all important user research. After organizing and reviewing such materials, the team is ready to generate the set of user utterances that they will support first and begin building. This can be done through team brainstorming.

However, brainstorming users' expected utterances is only half of the equation. Just as important is the agent's **responses** ***to those utterances.*** Therefore when brainstorming, generate *utterance pairs:* both what the user says and how the agent responds, or conversely, what the agent says and how the user is expected to respond. Utterance pairs are what the development team must build, so they should be brainstormed and designed at the same time.

Schedule two hours for a brainstorm session, ideally face-to-face, and invite both team members and stakeholders, as well as any other experts or consultants who may provide insight, to attend. After briefly reviewing the project outcome statements and user and agent personas for the brainstorming participants, instruct them to take 15–30 minutes to generate as many utterance pairs as they can. They should write the utterance pairs on sticky notes (one pair per note). On each note, they should draw a horizontal dividing line and write a user utterance at the top and the agent's response at the bottom or vice versa (Example 8.2).

Example 8.2 Utterance Pair, User Starts.

U: **I want to fly to San Jose**

A: Where are you leaving from?

If they can think of multiple responses to the first utterance, participants should add them below (Example 8.3).

Example 8.3 Utterance Pair, Multiple Responses.

U: **I want to go someplace warm and sunny in November?**

A: I recommend Puerto Vallarta, Mexico!

. . .

A: Here are some destinations that are warm and sunny in November.
((list of destinations))

In addition to agent responses to users' utterances, participant should also brainstorm user responses to agent utterances (Example 8.4). A conversational interface is a two-way street.

Example 8.4 Utterance Pair, Agent Starts.

A: What is your name?

U: **Dave**

. . .

U: **why do you need to know?**

Of course, as we have seen in Chapter 4, conversational sequences are not always pairs. They may include various types of sequence expansions or extended tellings. While full sequence design is not the goal of brainstorming, multiple sticky notes, each with one pair, can be used to assemble longer sequences if desired.

Brainstorming user-agent and agent-user utterance pairs will enable your development team and stakeholders to explore the range of talk needed for your conversation space. Participants should think broadly and identify as many relevant utterance pair types as they can, rather than generating multiple examples of the same type. This will help your team scope the conversation space itself.

After you have had sufficient time to brainstorm individually, regroup as a team and share the pairs you have created. Participants should place each of their sticky notes on a wall or whiteboard in the following categories: A. On-topic, In-Scope; B. On-Topic, Out-of-Scope; C. Off-Topic, In-Scope; and D. Off-Topic, Out-of-Scope. A facilitator should then read each of the utterance pairs aloud, one by one, and lead group discussion on each. In addition to discussing the utterances themselves, also decide collaboratively if a pair should be in scope, and if so, what outcome it aligns with and what backend support it will require (Table 8.3). Move the sticky notes from category to category as the group discusses the applicability and feasibility of the utterance pairs.

Table 8.3 Prioritizing utterance pairs

On/Off-Topic	In-Scope	Out-of-Scope
On-Topic	A. Fully supported, high priority, “show me flights to Las Vegas”	B. Partially supported, medium priority, “when does the Neon Museum close?”
Off-Topic	C. Fully supported, low priority, “tell me about yourself”	D. Unsupported, lowest priority, “where was Lord of the Rings filmed?”

A. On-Topic, In-Scope Utterance Pairs

Utterance pairs that are both on topic and in scope will constitute the core of your conversation space. They will have the highest priority in your development plan. If you are building a virtual travel agent, they will include requests like, “where should I go on vacation?,” “what can I do in Key West?” and “show me flights to Las Vegas.” They will be built using patterns like those for conversational activities (A1–A5). When discussing these pairs, identify the backend data and APIs that will be required to build the pair. Where will you get data on travel destinations or attractions or flights and airfares? Similarly, what UI support will be required in building the utterance pair? Will it require visual interaction with lists or cards? In discussing these dependencies, you may decide that a particular pair is not in scope because the data required are unavailable to you or the APIs are too expensive.

B. On-Topic, Out-of-Scope Utterance Pairs

Brainstorming utterance pairs should generate many things that are perfectly reasonable for users to assume that the agent might do but are beyond the scope of what you plan to build or include in the first release. These utterance pairs should be partially supported. That is, you should create intents for them, but the agent’s response will be some version of “I’m afraid I can’t do that at this time.” For example, in a travel agent application, users may assume that the agent can provide the business hours of attractions in the various travel destinations, but this granularity of data may be unavailable to you. While the agent’s response is ultimately dissatisfying for the user, it is important to show at least that the agent recognized what the user was requesting: “I’m afraid I don’t have information on hours of operation at this time.” You should create many on-topic, out-of-scope utterance pairs as part of your agent’s talk about its capabilities (Pattern C3.2).

C. Off-Topic, In-Scope Utterance Pairs

As discussed above, your virtual agent should know something about itself and should be able to engage in at least some level of self-talk and "small talk." Utterance pairs involving the agent itself or the company it represents are off the main topic of your application. For example, the user request "tell me about yourself" is unrelated to the topic of travel planning or choosing a health plan. Nonetheless, you may decide to support them fully in order to provide a better user experience. In planning your development, you should give these kinds of utterance pairs lowest priority since they are not part of the core functionality.

D. Off-Topic, Out-of-Scope Utterance Pairs

Inevitably a good broad brainstorming session will generate some utterance pairs that are neither on the topic or within the scope of your application. For example, for a virtual travel agent, you might decide that user inquiries like "where was Lord of the Rings filmed?" or "what do Millennials like to do?" are too far off the topic of travel planning and too far out of the scope of your project to support. By placing particular utterance pairs in the last category, the stakeholders and development team agree that they will not be supported in any way beyond the global agent paraphrase request (B1).

Brainstorming utterance pairs for your conversational agent application is an important exercise for exploring the breadth of your conversation space and for contributing to your initial development plan.

8.5 Stage 4: Prototype

After defining user personas and goals, an agent persona, and user-agent utterance pairs, you will be well prepared to begin designing the conversation space itself. Although conversational UX is very different from visual UX, we have attempted to equip you with a basic understanding of the mechanics of natural conversation through the previous chapters in this book. With the vocabulary of "turns" and "utterances," "sequences," "activities," and "conversations," you should understand the various parts that you need to build in order make a naturalistic conversational user experience. With the human design principles of *recipient design, minimization,* and *repair*, you should understand the fundamental strategy that speakers use in designing their utterances. With the Natural Conversation Framework's (NCF) interaction model of *expandable sequences*, you should understand how dependent utterances (sequence expansions) function to manage interactional troubles around independent utterances (base pairs). And finally, with the NCF's pattern language, you have 100 generic patterns for designing particular conversational activities, including sequence-level management and conversation-level management. Bringing all these things together will enable you to design a natural-language interface that works in many ways like a natural conversation.

As with any kind of UX design and system development, we recommend designing before building. Designing is quicker and cheaper than building and will enable your design to evolve through multiple iterations before writing content or code or generating graphics. Below we outline a general design process for conversational UX design. It includes three phases: mock-up, design, and build.

Mock-Up

While sketches and wireframes are useful for visualizing graphical interfaces, they will not help much in representing a conversation design. Instead, **simple transcripts** ***are better suited.*** Transcripts can represent the structure and style of the design, including the sequencing of utterances and exact phrasing, for both text- and voice-based interactions. Transcripts are easy to read, easy to create, and easy to share. Designers and stakeholders can quickly iterate on a transcript mock-up before building any parts of the conversation space. Simple transcripts also lack any representation of a visual interface in the case of text-based interfaces, which may distract from the design of the sequences of the conversation itself.

The field of Conversation Analysis provides some transcription conventions that are also useful for mocking up user interactions (Appendix A). Conversation analysts mark the speaker of each turn with short labels, such as 'U' for *user* and 'A' for the *agent* or chatbot, in order to keep track of who is saying what at all times. We further put the user's utterances in boldface to differentiate the human from the machine. In addition, conversation analysts number each line of a transcript, somewhat as programmers number lines of code, to make it easy to refer to particular utterances (Example 8.5).

Example 8.5 Simple Mock-up.

1 U: **who invented the hard disk?**
2 A: It was invented by IBM in 1956.
3 U: **can you give an example?**
4 A: The IBM 305 RAMAC was the first
5 computer to use a hard disk drive.

In this simple mock-up, a design for how a user can elicit an example of the agent's prior utterance is demonstrated. The user can ask a question (line 1), receive an answer (line 2) and then elicit a particular kind of paraphrase or understanding repair: an example (line 3). The agent will recognize this kind of user action and respond appropriately (lines 4–5). Whether or not this design will work with the eventual architecture of the application is not the concern at this point. The goal is to represent the target user experience that the system will produce.

Despite the numerous articles on chatbots and conversational agents these days, there is a noticeable absence of the conversation itself. For every article about a new conversational agent, ask "What is the conversational experience like?," "How conversational is it?," "What is the agent doing that is new and innovative?" None of these questions can be answered without seeing examples of the interaction, and simple transcripts provide an effective way to represent them.

Conversation analysts trade excerpts of detailed transcriptions of naturally occurring human conversation in order to share and demonstrate their analytic discoveries. Conversational UX designers should likewise trade transcripts in order to demonstrate the form and beauty of their designs. The practice of sharing transcripts will enable conversation designers to share tips and tricks, learn from one another, and collaborate: "Here's a design I've got!," "Check out this working script!," "How did you do that?" Transcripts are the currency of conversation analysts and should be for conversational UX designers too!

Design

While mock-ups represent the desired conversational user experience, they do not represent the underlying system itself. Many sequences, or parts of sequences, that can be designed through transcripts may not be feasible to build. Therefore, the next step is to design the conversation code itself. *Pseudocode* is a notation that resembles a programming language but is simplified. Pseudocode is commonly used by programmers to design programs before building them. Because it is simplified, designers can use pseudocode to focus on the design itself without worrying about exact commands and syntax.

We use a particular kind of pseudocode specifically for representing dialog logic for the Intent-Entity-Context-Response paradigm, used in most of today's chatbot and conversation platforms (Chapter 3). So far we have used it mostly with Watson Assistant. In this paradigm, there are only five actions the designer can take (Appendix B).

Commands in Dialog Design

1. **Create condition (if).** Conditions are compared to input utterances.
2. **Assign default (else).** Assign action if no conditions are met.
3. **Set variable (set).** Capture the context of the current input for future turns.
4. **Route to node (goto).** Route to another dialog node.
5. **Respond to user (say).** Output text to the user.

In creating the first type of basic action, "create condition," designers can typically combine the following elements:

Components of Dialog Conditions

a. **Intents (#).** Linguistic classes against which the similarity of a text input can be scored.

b. **Entities (@).** Keywords or phrases to be matched exactly.

c. **Context ($).** Variables for capturing events in the conversation.

d. **Pattern (^).** Text string analysis with regular expressions, JavaScript, etc.

By using this shorthand for the five basic commands (if, else, set, goto, say) and the four condition types (#, @, $, ^), you can quickly represent most features of a dialog tree. For example, the following pseudocode (Example 8.6) represents the underlying node structure of the simple mock-up above (lines 3–5 only).

Example 8.6 Pseudocode.

```
1  if #EXAMPLE_REQUEST

2      if $example has value
           say $example

3      else say "I'm afraid I can't think of an example."
```

This pseudocode represents one parent node and two child nodes, as indicated by the indents. The parent node has a single condition, an intent, *example request* (line 1). The first child node has a single condition, context variable *example* has a value, and a response, which is the value of *example* (line 2). The second child node assigns a default action, respond with "I'm afraid I can't think of an example." (line 3). With this bit of pseudocode, a conversational UX designer can quickly build the corresponding dialog-node structure in an authoring tool like that of Watson Assistant. In practice, we usually do not represent an entire dialog tree in pseudocode, but only the more complex node branches. If we are implementing a new pattern and must figure out the dialog logic, we use this pseudocode to represent the dialog-node design and thus to aid individual and collaborative thinking (for more examples, see Chapter 3, Examples 3.6, 3.10, and 3.13).

A *pseudocode design*, in contrast to a transcript, enables designers to represent how the underlying dialog code works and to share the design with others. It enables the team to answer the question, "Can we actually build this?" While transcripts are easy to read by any stakeholders, pseudocode may be difficult for anyone who is unfamiliar with authoring on a conversation platform, such as Watson Assistant, Amazon Alexa, or Google Assistant. Therefore, we share them mostly among only conversation designers/builders.

Build

Once you, as the conversational UX designer, have a firm grasp of what to build, you should start working in an authoring interface. The transcript mock-ups and pseudocode designs provide a detailed blueprint for what to create. We begin building by taking the our generic conversation space, or Watson Assistant workspace file, which contains implementations of all the patterns in the NCF pattern language, and modifying the content: intents, entities, context, and responses. For example, we configure the opening (C1) and the closing (C4) with the *agent name* and *organization*, as well as the actions we want the agent to do in the *first, second,* or *last topic* slots. We then edit the conversational activities (A) that are needed for the core functionality identified in the utterance-pair brainstorming and prioritization. For example, if building a virtual travel agent, we would duplicate the open request patterns (A2) and change Alma's placeholder content for restaurant recommendations to that for *destination* recommendations. The intent that includes examples such as "can you recommend a Mexican restaurant?" gets changed to one with examples like "where should I go on vacation?" The relevant details, cuisine and distance, are changed to things like favorite activities (e.g., hiking, nightlife, tourist attractions, etc.), region (e.g., state, country, continent), and weather (e.g., sunny, warm, dry, etc.). And the agent's utterances, such as, "What kind of food do you like?" and "Mario's is located at Beach and Main," get changed to utterances like, "What kinds of things do you like to do?" and "You might like San Juan, Key West or Puerto Vallarta!"

After replacing as much generic content with actual content as we can, we then fine-tune the existing conversation patterns by adding or removing dialog nodes as needed. For example, because the restaurant-recommendation sequence requires only two types of details, adapting the dialog structure for destination recommendations would require adding dialog nodes for extracting or eliciting more than two details or entity types. Or more dialog nodes may be added to support a longer list of instructions (A3) or to expand the number of capability topics (C3). We also sometimes disable existing subpatterns in the pattern language, such as the disengagement from the user in the agent offense complaint (C5.3), if they are not well suited for the particular use case.

Finally, if the use case requires a new kind of conversation pattern not adequately captured in the NCF pattern language (Appendix C), we design it through mock-up and pseudocode and build it from scratch in the new conversation space. New patterns are of course always designed to be compatible with existing patterns in the pattern language, especially with those of sequence-level management (B). If we believe the new pattern is generic, we will use it to extend the pattern language itself.

8.6 Stage 5: Test

We characterize conversational agents as "conversation games" not merely because they provide a simplified simulation of one kind of real-world interaction but also because, like video games, conversational agents require a significant period of *testing*. Video game development, especially for multiplayer games, involves significant beta testing to detect software bugs, test performance under realistic numbers of concurrent users, and test fun, but also to discover the range of unanticipated, or "emergent," behaviors and interactions players will exhibit. In a similar way, users of conversational systems display a wide range of behaviors and especially linguistic formulations, all of which the development team cannot anticipate in advance. Plan for more than one round of beta testing. In the first round, look especially for user and agent actions that you may have missed. Decide either to add important overlooked actions (Table 8.3A) or to acknowledge the agent's inability to perform them (Table 8.3B). In subsequent rounds of testing, you will find that you have accounted for the most important user actions but that there are still additional ways of saying the same things that you had not anticipated. Continue to fine-tune your conversation space by adding new intent variations and synonyms of entities as you run subsequent beta tests and even after launch.

As with any design process, you should *mock-up, design, build,* and *test* your conversational agent. Repeat this process throughout development in order to iterate on the original design. You always learn new things from actual user interaction with your application.

8.7 Conclusion

In this chapter we have presented a general process for designing *conversational* user experience that adapts the five-stage Hasso-Plattner model of design thinking: empathize, define, ideate, prototype, test. It starts with building *empathy* with the prospective users through research, including methods of observation and engagement. Insights from this stage are then used to *define* core problems and to create design artifacts, such as user personas and outcome statements. *Ideation* of a new conversational agent solution can then be facilitated by creating an agent persona consisting of a job description, personality, and self-knowledge, and/or sets of utterance pairs. Once promising ideas are generated, conversation spaces can be *prototyped* by mocking them up through simple transcripts, designing the dialog logic using pseudocode if necessary, and building on a conversation authoring platform (e.g., Amazon Alexa or Watson Assistant). Finally, early prototyped conversation spaces should be *tested* with users as much as possible so the design can be refined iteratively.

From this chapter, you should be able to lead a development team through a design-thinking process that is adapted to the unique requirements of conversational UX. You should be able to put what you learned from previous chapters about conversation theory (Chapter 2), conversation authoring (Chapter 3), and the Natural Conversation Framework (Chapters 4–7) into practice within the context of a software development project. The process outlined above applies from the project's overall goals down to lines of dialog pseudocode.

While this design process is intended for development teams and may seem overwhelming, remember that the heart of conversational UX design is the transcript. In the first place, a conversational agent must have something useful and interesting to say. Creating compelling sequences of conversation is the central problem. The rest is engineering and planning. Design both conversations that are feasible for your team to build today and conversations that may not be feasible to build until tomorrow. Provide your team with a vision of what is possible for a conversational user experience and lead them to realize it.

9

Conclusion

In the beginning of this book, we proposed a definition of "understanding" as *the demonstration of correct or adequate interpretation of social action within interaction*. As Wittgenstein [1953, #150] reminds us: the grammar of the word "understands" is closely related to "knows," "can" or "is able to." Understanding is fundamentally *doing*, not thinking. It is the outcome of a social interaction, not a private mental state or process. Therefore, if a virtual agent can *do* conversation adequately and can achieve the kinds of interactional outcomes that we call "understanding," then for all practical purposes *it understands*. Turing's [1950] "imitation game" is intended to test for such an achievement.

But the theoretical possibility of computers that can understand and participate in natural conversation is far from the current state of technology. Conversational agents are still in their infancy. Their "understanding" of most topics and social actions is quite partial and brittle when compared to that of humans. However, the path to realizing such artificial intelligence lies in equipping a computer system with the ability to recognize the interactional patterns that constitute conversational competence, as well as the details of knowledge and natural language. As we strive to create automated conversational agents, we will unavoidably develop simplified versions of natural conversation: user interfaces that work like a conversation in some ways, within a limited range of knowledge and a limited range of action. We called such user interfaces "conversation games" because, like computer video games, they offer simplified simulations of the real world interactions [Moore et al. 2007]. Although simplified compared to human talk-in-interaction, conversational interfaces, and the wider user experience around them, need not be simplistic.

In this book we have demonstrated a particular framework and methodology for designing conversational user experience. We argued that, whether your agent talks through text or through voice, you should adopt a *conversation-first* approach to designing the user experience. In demonstrating such an approach, we have applied Conversation Analysis to the design of conversational agents [Luff et al.

1990, Frohlich and Luff 1990] and taken the next step. The Natural Conversation Framework (NCF) is adapted from conversation science, yet its theories, concepts, and models can only inspire the designer by revealing, in formal detail, how the machinery of natural human conversation works. There is always a creative leap that must be made from science to design.

In addition to providing some conversation theory as a foundation for our approach, we covered the basics of authoring conversation within the Intent-Entity-Context-Response (IECR) paradigm popular today. Building on this background knowledge, we then outlined a design framework consisting of (1) an interaction model that reflects some of the core mechanics of human conversation; (2) a content format for breaking content down into the size required for natural conversation; (3) a pattern language with 100 reusable conversational UX patterns; and (4) a general navigation method for conversational interfaces derived from the pattern language. Finally, to help you put this all into practice, we outlined a general design-thinking process adapted for conversational UX. As part of this process, we modeled a practice for representing user interaction through simple transcripts and representing dialog logic through pseudocode. Adopting this practice can facilitate communication and collaboration across your team and even across a conversational UX design discipline. From this book you should be able to speak with some authority about the mechanics of human conversation, demonstrate a systematic approach to designing conversational user experience, apply the NCF pattern language to the design of conversation spaces and user flows, and lead a development team through a design-thinking process for conversational UX.

9.1 Conversation Metrics

This book is about an alternative approach to the *design* of conversational user experience that leverages the findings and patterns of conversation science. Its purpose is to introduce practitioners to a new way of looking at conversational UX, as well as to provide some basic conversation theory and reusable design assets. While we have developed the Natural Conversation Framework (NCF) [Moore et al. 2016, Moore 2018] in the course of building dozens of prototype systems and a few production systems over the past four years, we do not yet have adequate user data to offer a definitive analysis of its performance. Preliminary results suggest that users find the NCF more “conversational” than most chatbots or voice assistants but that they often interact with it as if it were a search engine (e.g., keywords only) or a fully competent English speaker (e.g., complex utterances). Both lead to interactional troubles. As we collect more data from multiple live applications, we are beginning to examine conversation logs and user behavior. The *analysis* of user interactions with agents built on the NCF, compared to those with other designs, is still a topic for our future work.

As part of our analytic efforts, we are developing novel metrics for evaluating the effectiveness of conversations built on our framework. The concept of "adjacency pair" [Schegloff and Sacks 1973, Schegloff 2007] from Conversation Analysis gives us a method for analyzing series of turns, if we build that interaction model into our conversation spaces. It enables us to see which sequences are primary and, of those, which were successful. It also enables us to estimate levels of mutual understanding or misunderstanding.

Since our Natural Conversation Framework (NCF) is based on *expandable sequences* [Schegloff 2007], the base sequences and their expansions can each be measured separately. This makes possible corresponding metrics for the effectiveness and efficiency of the conversation itself, independently of customer satisfaction. Effectiveness and efficiency are also two-way metrics: they apply to both the automated agent and to the user. The NCF offers the following three metrics.

Number of Sequences is a measure of how many action pairs, or base adjacency pair sequences, were initiated by the user or by the agent. Instead of reporting simply total number of turns, we segment the conversation logs into expandable sequences, that is, base sequences with their expansions, if any. This gives a more informative measure of how many social actions the user or the agent attempted than does total number of turns.

Sequence Completion Rate is the percentage of initiated base sequences that were completed by the recipient: agent or user. In other words, this is a measure of the rate of success across sequences in the conversations, which is closely related to the level of mutual understanding. If agent or user do not understand what the other means or is doing, they cannot complete the sequence initiated by the other. This is critical because *mutual understanding* is one of the distinctive goals of conversation.

Interactional Efficiency is a measure of how much extra work the user or the agent had to do in the conversation. The expandable-sequence interaction model gives us a theoretical minimum number of turns a given conversation could have. That is, if every completed sequence were compact, the theoretical minimum would be double the number of completed base sequences. We can then compare that minimum to the *actual* number of turns in the conversation and get a measure of the rate of sequence expansion.

Taken together, these sequence metrics tell us different things about conversations with our agent, Alma. Using extreme sets of artificial data (Examples 9.1–9.3), we can demonstrate how the metrics vary with different kinds of conversations.

Example 9.1 Example of Set A.

1 U: **what computer beat Jeopardy?**
2 A: An IBM computer named Watson.
3 U: **what computer beat a human chess champion?**
4 A: It was named Deep Blue.

Example 9.2 Example of Set B.

1 U: **what computer beat a human chess champion?**
2 A: It was named Deep Blue.
3 U: **Deep what?**
4 A: Blue.
5 U: **what do you mean?**
6 A: The IBM computer was named Deep Blue.
7 U: **oh ok**
8 A: Is there anything else I can do for you?

Example 9.3 Example of Set C.

1 U: **what's in theaters?**
2 A: What do you mean?
3 U: **what movies are playing in theaters now?**
4 A: I'm sorry, I still don't understand.

These examples represent the kinds of sequences in each experimental data set, A, B, and C. The data sets are "extreme" in that they only contain one kind of sequence, that is, compact sequences (A), expanded, completed sequences (B) and uncompleted sequences (C). These extreme data sets better enable us to see under which conditions these metrics vary than do real data (Table 9.1).

In data set A, we included only completed sequences initiated by the user and no expansions by the user. The sequence completion rate for agent and user combined

Table 9.1 **Sequence metrics for extreme data**

Set	Number of Sequences	Sequence Completion	Interactional Efficiency
A	17.4	0.88	0.92
B	10.4	0.83	0.49
C	7.2	0.14	0.09

was 88% and the interactional efficiency was 92%. In data set B, we included not only completed sequences initiated by the user but also numerous expansions by the user, such as repeat requests, “what did you say?”, paraphrase requests, “what do you mean?”, sequence closers, “ok” and “thanks,” and more. In the case of these frequent expansions, the combined sequence completion rate was still high, 83%, but interactional efficiency dropped significantly to 49%. Finally, in data set C, we included only conversations in which none of the substantive sequences initiated by the user were completed. In other words, there was almost a complete lack of understanding by the agent of what the user was saying. The sequence completion rate plummeted to 14% and interactional efficiency to 9%.

In short, if both the sequence completion and interactional efficiency rates are high, the conversations themselves are effective. If they are both very low, the conversations have failed. But if sequence completion is high and interactional efficiency is moderate, the conversations are successful but the user or agent is doing additional work to achieve that success. This invites the conversational UX designer to explore the nature of those sequence expansions. If they are eliciting details, the topic of conversation may be inherently complex. For example, buying airline tickets involves a lot of details and decisions. Moderate interactional efficiency may be normal for this activity. However, if the expansions are primarily due to hearing repairs, the conversation designer should re-examine the level of noise in the environment and perhaps the placement of the system’s microphones. And if the expansions are primarily due to understanding repairs, the conversation designer should re-evaluate the terminology the agent uses and the knowledge it assumes and determine if the conversation can be redesigned so that it is more comprehensible from the start. With inherently complex topics or activities, expansions may be unavoidable, but at least with repair features, the user and agent can still succeed in the face of understanding troubles. This is the value of a robust conversational repair system.

These sequence metrics also enable us to help disentangle user dissatisfaction with the agent itself from dissatisfaction with its message, for example company policies. If a customer reports dissatisfaction after an interaction with a company’s virtual agent, and the sequence completion and interactional efficiency rates are high for that conversation, then we know that the customer did not experience trouble understanding the agent and vice versa. Rather, the dissatisfaction must have come from the message delivered by the agent and not the quality of the conversation itself. In other words, if the user complains and the agent recognizes and responds appropriately to the complaint, then the problem is not in the agent’s ability to understand but in the substance of the complaint itself.

How It Works

In order to measure the occurrence of base sequences and their expansions in conversation logs, we label both the user's and the agent's actions inside the conversation space itself. We set context variables on each node in the dialog logic that contains an intent or response, to indicate if the user inputs and agent outputs associated with that sequence are parts of base sequences or expansions. So, for example, a simple greeting exchange would be labeled with the following context data (Example 9.4).

Example 9.4 Labeled Conversation Log.

```
1  U:  hi
2  A:  Hello.

1      user_input = "hi"
       user_action = "greeting"
       confidence = 1.0
       user_APR = B1PP
2      repeat = "Hello."
       agent_action = "greeting"
       agent_APR = B2PP
```

What the user says, "hi," is captured by a system variable *input text* and set to the variable *user input*. The user's action is captured by recording the intent name, "greeting," using the system variable *intents[0].intent* and setting it to *user action*. In addition, the confidence level for that intent is captured. The sequential function of the user's utterance is captured by a set of structural codes, *user APR*, that we have constructed based on the adjacency pair and repair models in Conversation Analysis [Schegloff 2007]; for example, "B1PP" stands for the first pair part of a base adjacency pair sequence. On the other hand, what the agent says is captured through the custom variable *repeat*, which is also used for repeat repairs, and the agent's action is hardcoded when the response is written and captured by *agent action*. And, like the user's utterance, the agent's is assigned a sequential function with the agent APR code "B2PP", or base second pair part. Once the dialog nodes are labeled as such, the conversation logs label themselves as users interact with the agent!

One limitation of this approach is that when a user's input defaults, that is, does not match any conditions in the dialog logic, we do not know what kind of action the user did nor its sequential function. To attempt to compensate for this missing data, we provide a modifier that represents the average percentage of defaulted inputs that are initiators of base sequences. For example, if 80% of past unrecognized user utterances are base first pair parts (B1PPs), we set this modifier to 0.8. The modifier is then based on prior data in order to provide a correction to the metrics. Unrecognized user utterances, or random samples of them, should be inspected on a regular basis both to set the modifier but also to discover any systemic problems hidden in these unclassified responses. Furthermore, conversational UX designers and product managers can learn to interpret the sequence metrics to determine if their conversational agent is doing better or worse than it was yesterday.

As we move to the analysis of the Natural Conversation Framework in action, we expect some modification and certainly extension of the pattern language. However, we are confident that the majority of the patterns are sound because they are based on patterns from conversation science, as well as on the iteration we have done to date, largely through internal testing. But the NCF's pattern language is intended to be an evolving project that we hope will grow with the help of the UX design community.

9.2

Final Thoughts

As mentioned above, the intended audience of this book is the UX designer (or anyone else interested in UX design), working on conversational agents, whether chatbots or voice assistants, who seeks a more formal understanding of how human conversation is structured. Through our brief introduction to Conversation Analysis and the Natural Conversation Framework, we have attempted to provide you with a foundation upon which to practice a new kind of UX design. As the conversational UX designer on a development project, we encourage you to claim the exchange of utterances between user and agent as your domain, not simply the visual design of the window in which it takes place or the user personas. Lead the design of the conversation itself, using conversation science, design thinking for conversation, the NCF pattern language, and transcript mock-ups. Coordinate with the system developers, content designers or data scientists to make sure that decisions they make do not diminish the user's *conversational* experience. Always explain the impacts of development decisions on the conversational user experience to the rest of the team. Argue for a conversation-first approach to the user experience. The exchange of utterances in a conversational application is the proper domain for UX design, but because it is new, you may need to make the case for why this is so. We hope you will use this book in part to make that case.

As a final thought, we expect that the role of conversational UX designer will change, perhaps radically, as the underlying technologies evolve. With current natural language platforms, using the Intent-Entity-Context-Response (IECR) paradigm, which relies on a human to model the interactional structure, the UX designer's role is one of conversation design and even engineering. However, as conversational platforms move in the direction of natural language *generation* [Kalchbrenner and Blunsom 2013, Sutskever et al. 2014, Vinyals and Le 2015, Song et al. 2016], where conversational structure may be modeled algorithmically from data, the UX designer's role may become more like that of a teacher or trainer. Instead of getting your hands dirty authoring dialog logic, you may be evaluating the conversational behavior of autonomous agents. Rather than building what they say, you will be correcting what they say and perhaps teaching them through interaction. In this future, which we expect as deep learning technologies mature, there will still be a demand for someone to insure that the system creates a great user experience. And this person must know, in a formal way, what constitutes "natural conversation" and what does not. It will be important for this conversational UX designer to articulate exactly how the behavior of the algorithms is not right. Even as the technologies evolve and the methods for designing conversational user experience change, the mechanics and patterns of natural human conversation will not.

APPENDIX A

Conversation Analysis Transcription Conventions

(0.6)	silence (seconds)
[oh] [we]ll	overlap
bye= =bye	latching; no silence
.hhh	in-breath
hhh	out-breath
huh huh	laugh
(mhm)	uncertain
((cough))	note
um:::	sound stretch
nope.	falling pitch
what?	rising pitch
and,	flat pitch
not him	emphasis, stress
NO WAY	louder
° okay °	quieter
>all right<	faster

APPENDIX

B

Dialog Design Pseudocode

Commands in Dialog Design

1. **Create condition, “if”.** Conditions are compared to input utterances.
2. **Assign default, “else”.** Assign action if no conditions are met.
3. **Set variable, “set”.** Capture the context of the current input for future turns.
4. **Route to node, “goto”.** Route to another dialog node.
5. **Respond to user, “say”.** Output text to the user.

Components of Dialog Conditions

a. **Intents, “#”.** Linguistic classes against which the similarity of a text input can be scored.

b. **Entities, “@”.** Keywords or phrases to be matched exactly.

c. **Context, “$”.** Variables for capturing events in the conversation.

d. **Pattern, “^”.** Text string analysis with regular expressions, JavaScript, etc.

APPENDIX C

NCF Pattern Language Summary

This Natural Conversation Framework (NCF) pattern language for conversational UX design consists of 100 distinct patterns for interaction between a user and a conversational agent. They are organized into three classes and 15 types (Table C.1).

Table C.1 Conversational UX patterns

Conversational Activities	Sequence-Level Management	Conversation-Level Management
A1 Inquiry (User)	B1 Repair (Agent)	C1 Opening (Agent)
A2 Open Request	B2 Repair (User)	C2 Opening (User)
A3 Extended Telling	B3 Extended Repair	C3 Capabilities
A4 Quiz	B4 Sequence Closer	C4 Closing
A5 Inquiry (Agent)	B5 Sequence Abort	C5 Disengaging

NCF Pattern Types

A. Conversational Activities

A1.0 Inquiry (User)
A1.1 Inquiry (User) Confirmation
A1.2 Inquiry (User) Disconfirmation
A1.3 Inquiry (User) Repairs

A2.0 Open Request
A2.1 Open Request Nonverbal
A2.2 Open Request Continuer
A2.3 Open Request Screening

A2.4	Open Request Agent Detail Request
A2.5	Open Request User Detail Request
A2.6	Open Request Summary
A2.7	Warrant Request & Refusal
A2.8	Open Request Summary with Artifacts
A2.9	Open Request Incremental (same as B3.1)
A2.10	Open Request Series
A2.11	Open Request Repairs

A3.0	Extended Telling with Repair
A3.1	Extended Telling Abort

A4.0	Quiz
A4.1	Quiz Incorrect
A4.2	Quiz Repairs
A4.3	Quiz User-Initiated
A4.4	Quiz Agent-Initiated

A5.0	Inquiry (Agent)
A5.1	Inquiry (Agent) Open
A5.2	Inquiry (Agent) Hearing Confirmed
A5.3	Inquiry (Agent) Hearing Disconfirmed
A5.4	Inquiry (Agent) Completion Check
A5.5	Inquiry (Agent) Extended Answer

A2+A3	Troubleshooting User-Initiated
A3+A4	Teaching

B. Sequence-Level Management

B1.1.0	Understanding Check (Agent)
B1.2.0	Paraphrase Request (Agent)
B1.2.1	Subsequent Paraphrase Requests
B1.2.2	Agent Continuer
B1.2.3	No Answer Account
B1.2.4	Repeat Acknowledgment

B2.1.0	Repeat Request
B2.1.1	Repeat Request Default
B2.2.0	Partial Repeat Request
B2.3.0	Hearing Check Confirmed
B2.3.1	Hearing Check Corrected

B2.4.0 Paraphrase Request
B2.4.1 Paraphrase Request Default
B2.5.0 Definition Request Repair
B2.5.1 Definition Request Default
B2.5.2 Definition Request Standalone
B2.6.0 Example Request
B2.6.1 Example Request Default
B2.7.0 Understanding Check Confirmed
B2.7.1 Understanding Check Disconfirmed

B3.1.0 Self-Correction (same as A2.9)
B3.1.1 Misunderstanding Report
B3.1.2 Other Hearing Check & Mishearing Report
B3.2.0 Other-Correction

B4.0 Sequence Closer (helped)
B4.1 Sequence Closer (not helped)
B4.2 Sequence Closer Appreciation (helped)
B4.3 Sequence Closer Appreciation (not helped)
B4.4 Sequence Closer (repaired)

B5.0 Sequence Abort
B5.1 Sequence Abort Capability Offer Accepted
B5.2 Sequence Abort Capability Offer Declined

C. Conversation-Level Management

C1.0 Opening Greeting (Agent)
C1.1 Opening Self-Identification (Agent)
C1.2 Opening Name Request (Agent)
C1.3 Opening Direct Address (Agent)
C1.4 Opening Welfare Check (Agent)
C1.5 Opening Organization Offer of Help (Agent)
C1.6 Opening Authentication (Agent)
C1.7 Organizational Problem Request (Agent)

C2.0 Greeting (User)
C2.1 Summons (User)
C2.2 Welfare Check (User)
C2.3 Displaced Greeting (User)
C2.4 Displaced Welfare Check (User)

C2.5	Reciprocated Welfare Check (User)
C2.6	News Check (User)
C2.7	News Check Capability (User)
C2.8	Self-Identification (User)
C2.9	Name Correction (User)
C2.10	Name Request (User)

C3.0	General Capability Check
C3.1	Capability Expansion
C3.2	Specific Capability Check
C3.3	Utterance Suggestion

C4.0	Last Topic Check (Agent)
C4.1	Last Topic Check (User)
C4.2	Pre-Closing (User)
C4.3	Closing Farewell (User)
C4.4	Closing Appreciation (Organizational)
C4.5	Closing Name Request (Agent)
C4.6	Closing Success Check (Affirmed)
C4.7	Closing Success Check (Disaffirmed)
C4.8	Closing Success Check Reopened
C4.9	Closing Offer (Affirmed)
C4.10	Closing Offer (Disaffirmed)

C5.0	Transfer Successful
C5.1	Transfer Aborted
C5.2	Recipient Correction
C5.3	Offense Complaint (Agent)
C5.4	Offense Complaint (User)

NCF Patterns and Examples

A. Conversational Activities

Pattern A1.0 Inquiry (User).

```
1  U:  INQUIRY
2  A:  ANSWER
3  U:  SEQUENCE CLOSER
```

Example A1.0 Inquiry (User).

```
1  U:  what computer won Jeopardy?
2  A:  An IBM computer named Watson.
3  U:  ok
```

Pattern A1.1 Inquiry (User) Confirmation.

1 U: INQUIRY
2 A: INQUIRY PARAPHRASE
3 U: CONFIRMATION
4 A: ANSWER
5 U: SEQUENCE CLOSER

Example A1.1 Inquiry (User) Confirmation.

1 U: **what was that Jeopardy computer?**
2 A: You mean, what computer won Jeopardy?
3 U: **yeah**
4 A: An IBM computer named Watson.
5 U: **oh**

Pattern A1.2 Inquiry (User) Disconfirmation.

1 U: INQUIRY
2 A: INQUIRY PARAPHRASE
3 U: DISCONFIRMATION
4 A: PARAPHRASE REQUEST
5 U: PARAPHRASE

Example A1.2 Inquiry (User) Disconfirmation.

1 U: **what computer beat humans at Go?**
2 A: You mean, what computer beat the human chess champion?
3 U: **no**
4 A: I'm sorry, I don't understand your question.

Pattern A1.3 Inquiry (User) Repairs.

1 U: INQUIRY
2 A: ANSWER
3 U: REPAIR INITIATOR
4 A: REPAIR
5 U: REPAIR INITIATOR
6 A: REPAIR
7 U: SEQUENCE CLOSER

Example A1.3 Inquiry (User) Repairs.

1 U: what computer won Jeopardy?
2 A: An IBM computer named Watson.
3 U: named what?
4 A: Watson
5 U: what do you mean?
6 A: Scientists at IBM created the computer that won Jeopardy and named it after the founder of the company, Thomas J. Watson.
7 U: oh ok

Pattern A2.0 Open Request.

1 U: FULL REQUEST
2 A: GRANT
3 U: SEQUENCE CLOSER
4 A: RECEIPT

Example A2.0 Open Request.

1 U: **can you recommend a nearby Mexican restaurant?**
2 A: Mario's is located at Beach and Main.
3 U: **thanks**
4 A: You're welcome!

Pattern A2.1 Open Request Nonverbal.

1 U: FULL REQUEST
2 A: ACKNOWLEDGMENT <NONVERBAL>
3 U: SEQUENCE CLOSER
4 A: RECEIPT

Example A2.1 Open Request Nonverbal.

1 U: **please turn on the lights**
2 A: Okay. ((lights come on))
3 U: **thank you**
4 A: You're welcome!

Pattern A2.2 Open Request Continuer.

1 U: DETAIL GIVING
2 A: CONTINUER
3 U: REQUEST
4 A: GRANT
5 U: SEQUENCE CLOSER
6 A: RECEIPT

Example A2.2 Open Request Continuer.

1 U: I hear the fish tacos around here are fantastic!
2 A: Mhmm.
3 U: **can you recommend a place?**
4 A: Mario's is located at Beach and Main.
5 U: **thanks**
6 A: You're welcome!

Pattern A2.3 Open Request Screening.

1 U: PRE-REQUEST
2 A: GO-AHEAD
3 U: FULL REQUEST

4 A: GRANT
5 U: SEQUENCE CLOSER
6 A: RECEIPT

Example A2.3 Open Request Screening.

1 U: **do you know about restaurants?**
2 A: I know a few places.
3 U: **can you recommend a nearby Mexican restaurant?**
4 A: Mario's is located at Beach and Main.
5 U: **thanks**
6 A: You're welcome!

Pattern A2.4 Open Request Agent Detail Request.

1 U: PARTIAL REQUEST
2 A: DETAIL REQUEST
3 U: DETAIL
4 A: GRANT
5 U: SEQUENCE CLOSER
6 A: RECEIPT

Example A2.4 Open Request Agent Detail Request.

1 U: **can you recommend a Mexican restaurant?**
2 A: Walking distance or car ride?
3 U: **walking distance**
4 A: Mario's is located at Beach and Main.
5 U: **thanks**
6 A: You're welcome!

Pattern A2.5 Open Request User Detail Request.

1 U: PARTIAL REQUEST
2 A: DETAIL REQUEST
3 U: DETAIL REQUEST
4 A: DETAIL
5 U: DETAIL
6 A: GRANT
7 U: SEQUENCE CLOSER
8 A: RECEIPT

Example A2.5 Open Request User Detail Request.

1 U: **what's a good restaurant nearby?**
2 A: What kind of food do you want?
3 U: **what are my choices?**
4 A: Around here, Mexican, Japanese or Indian.
5 U: **Mexican**

6 A: Mario's is located at Beach and Main.
7 U: **thanks**
8 A: You're welcome!

Pattern A2.6 Open Request Summary.

1 U: PARTIAL REQUEST
2 A: DETAIL REQUEST
3 U: DETAIL
4 A: DETAIL REQUEST
5 U: DETAIL
6 A: DETAIL REQUEST
7 U: DETAIL
8 A: REQUEST SUMMARY
9 U: SUMMARY CONFIRM
10 A: GRANT
11 U: SEQUENCE CLOSER
12 A: RECEIPT

Example A2.6 Open Request Summary.

1 U: I need to register
2 A: Okay. What's your name?
3 U: **Dave Bowman**
4 A: What's your job title?
5 U: **astronaut**
6 A: What is your organizational affiliation?
7 U: NASA
8 A: So Dave Bowman, Astronaut at NASA. Is that correct?
9 U: **yes**
10 A: Okay. You're all set!
11 U: **thank you**
12 A: You're welcome!

Pattern A2.7 Warrant Request & Refusal.

1 A: DETAIL REQUEST
2 U: WARRANT REQUEST
3 A: WARRANT
4 U: REFUSAL
5 A: ACKNOWLEDGMENT. <NEXT ACTION>

Example A2.7 Warrant Request & Refusal.

1 A: What's your job title?
2 U: why do you need to know?
3 A: Job title enables us to know the range of professions of our participants.

4 U: none of your business
5 A: No problem. What is your organizational affiliation?

Pattern A2.8 Open Request Summary with Artifacts.

1 U: PARTIAL REQUEST
2 A: DETAIL REQUEST
3 U: DETAIL
4 A: DETAIL REQUEST
5 U: DETAIL
6 A: DETAIL REQUEST
7 U: DETAIL
8 A: HOLD REQUEST
9 A: GRANT
10 <OFFER OF ARTIFACTS>
11 U: SEQUENCE CLOSER
12 A: RECEIPT

Example A2.8 Open Request Summary with Artifacts.

1 U: I want to book a flight
2 A: Okay. Where are do you want to go?
3 U: **Kona, Hawaii**
4 A: Where are you leaving from?
5 U: SJC
6 A: When would you like to go?
7 U: **June 17-30**
8 A: One moment please...
9 A: Okay. Here are flights from SJC to KOA on June 17-30.
10 ⋮ ((visual flight cards))
11 U: **thank you**
12 A: You're welcome!

Pattern A2.9 Open Request Incremental (same as B3.1).

1 U: FULL REQUEST
2 A: GRANT
3 U: INCREMENTAL REQUEST
4 A: GRANT
5 U: SEQUENCE CLOSER
6 A: RECEIPT

Example A2.9 Open Request Incremental.

1 U: **can you recommend a nearby Mexican restaurant?**
2 A: Mario's is located at Beach and Main.
3 U: **how about Indian?**

4 A: Taj Mahal is on Third and Union.
5 U: **thanks**
6 A: You're welcome!

Pattern A2.10 Open Request Series.

1 U: FULL REQUEST
2 A: GRANT
3 U: RELATED REQUEST
4 A: GRANT
5 U: SEQUENCE CLOSER
6 A: RECEIPT

Example A2.10 Open Request Series.

1 U: **can you recommend a nearby Mexican restaurant?**
2 A: Mario's is located at Beach and Main.
3 U: **when is it open?**
4 A: Mario's is open from 11:00am to 2:00am.
5 U: **thanks**
6 A: You're welcome!

Pattern A2.11 Open Request Repairs.

1 U: FULL REQUEST
2 A: GRANT
3 U: REPAIR INITIATOR
4 A: REPAIR
5 U: SEQUENCE CLOSER
6 A: RECEIPT

Example A2.11a Open Request Repairs.

1 U: can you recommend a nearby Mexican restaurant?
2 A: Mario's is located at Beach and Main.
3 U: what do you mean?
4 A: The Mexican restaurant, Mario's, is located at the intersection of Beach Boulevard and Main Street.
5 U: thanks
6 A: You're welcome!

Example A2.11b Open Request Repairs.

1 A: What can I do for you?
2 U: I just came over from the other hotel. I'm with NASA?
3 A: NASA, okay.
4 U: can I register here?
5 A: Okay. What's your name?
6 U: Dave Bowman

7 A: What's your job title?
8 U: astronaut
9 A: What kind of meal would you like?
10 U: what are my choices?
11 A: Beef, fish or vegan.
12 U: beef
13 A: So Dave Bowman, Astronaut at NASA and the beef meal. Is that correct?
14 U: no, I mean fish
15 A: So Dave Bowman, Astronaut at NASA and the fish meal. Is that correct?
16 U: yes
17 A: Okay. You're all set!
18 U: will there be a reception?
19 A: I'm afraid NASA is not having a reception.
20 U: ok thanks
21 A: You're welcome!

Pattern A3.0 Extended Telling with Repair.

1 U: STORY REQUEST/INSTRUCTION INVITATION
2 A: PART/STEP 1
3 U: CONTINUER/PAUSE
4 A: PART/STEP 2
5 U: REPAIR INITIATOR
6 A: REPAIR
7 U: CONTINUER/PAUSE
8 A: PART/STEP 3
9 U: SEQUENCE CLOSER
10 A: RECEIPT

Example A3.0 Extended Telling with Repair.

1 U: how do I meditate?
2 A: First, sit comfortably and breathe slowly.
3 U: ok
4 A: Next, count each in breath and out breath until you get to ten. Then repeat.
5 U: what do you mean?
6 A: As you breathe in, out, in, count one, two, three all the way to ten.
7 U: all right
8 A: Finally, continue counting your breaths for ten minutes.
9 U: thanks
10 A: You're welcome!

Pattern A3.1 Extended Telling Abort.

1 U: STORY/INSTRUCTION INVITATION
2 A: PART/STEP 1
3 U: CONTINUER/PAUSE
4 A: PART/STEP 2
5 U: REPAIR INITIATOR
6 A: REPAIR
7 A: PART/STEP 3
8 U: COMPLAINT/ABORT REQUEST
9 A: ABORT OFFER
10 U: ABORT CONFIRM
11 A: ACKNOWLEDGMENT

Example A3.1 Extended Telling Abort.

1 U: tell me a story
2 A: Once upon a time, there was a little girl named Goldilocks.
She went for a walk in the forest. Pretty soon, she came upon
a house. She knocked and, when no one answered, she walked
right in.
3 ((pause))
4 A: At the table in the kitchen, there were three bowls of porridge.
Goldilocks was hungry. She tasted the porridge from the first
bowl. This porridge is too hot!, she exclaimed.
5 U: what do you mean by porridge?
6 A: Oh, sorry! Porridge is a dish consisting of oatmeal or another
grain boiled in water or milk. Sometimes called hot cereal.
7 U: ok
8 A: So, she tasted the porridge from the second bowl. This porridge
is too cold, she said. So, she tasted the last bowl of porridge.
Ahhh, this porridge is just right, she said happily and she ate
it all up.
9 U: this is boring
10 A: Shall I stop?
11 U: yes please
12 A: Okay.

Pattern A4.0 Quiz.

1 A: QUESTION
2 U: CORRECT ANSWER
3 A: CONFIRMATION

Example A4.0 Quiz.

1 A: Why can't you trust atoms?
2 U: **they make up everything**
3 A: That's right!

Pattern A4.1 Quiz Incorrect.

```
1  A:  QUESTION
2  U:  INCORRECT ANSWER
3  A:  DISCONFIRMATION
4  U:  REFUSAL/ABORT
5  A:  CORRECT ANSWER
6  U:  SEQUENCE CLOSER
```

Example A4.1 Quiz Incorrect.

```
1  A:  Why can't you trust atoms?
2  U:  you can't see them?
3  A:  Wrong! Try again.
4  U:  you tell me
5  A:  They make up everything.
6  U:  haha
```

Pattern A4.2 Quiz Repairs.

```
1  A:  QUESTION
2  U:  REPAIR INITIATOR
3  A:  REPAIR
4  U:  CORRECT ANSWER
5  A:  CONFIRMATION
```

Example A4.2 Quiz Repairs.

```
1  A:  Why can't you trust atoms?
2  U:  atoms?
3  A:  Yes.
4  U:  they make up everything?
5  A:  That's right!
```

Pattern A4.3 Quiz User-Initiated.

```
1  U:  QUIZ REQUEST
2  A:  QUESTION
3  U:  CORRECT ANSWER
4  A:  CONFIRMATION
```

Example A4.3 Quiz User-Initiated.

```
1  U:  tell me a joke
2  A:  What can you hold without ever touching it?
3  U:  a conversation?
4  A:  That's right!
```

Pattern A4.4 Quiz Agent-Initiated.

```
1  A:  QUIZ OFFER
2  U:  ACCEPTANCE
```

3 A: QUESTION
4 U: CORRECT ANSWER
5 A: CONFIRMATION

Example A4.4 Quiz Agent-Initiated.

1 A: Would you like to play a computer trivia game?
2 U: **sure**
3 A: Okay. What computer beat human champions at Jeopardy?
4 U: **Watson**
5 A: You got it!

Pattern A5.0 Inquiry (Agent).

1 A: CLOSED INQUIRY
2 U: ANSWER
3 A: SEQUENCE CLOSER
4 U: RECEIPT

Example A5.0 Inquiry (Agent).

1 A: On a scale from one to ten, how would you rate Mario's?
2 U: **nine**
3 A: Thank you.
4 U: **you're welcome**

Pattern A5.1 Inquiry (Agent) Open.

1 A: OPEN INQUIRY
2 U: OPEN ANSWER
3 A: SEQUENCE CLOSER
4 U: RECEIPT

Example A5.1 Inquiry (Agent) Open.

1 A: What did you think of Mario's?
2 U: **the food was really good, and I loved the cantina atmosphere!**
3 A: Thank you.
4 U: **you're welcome**

Pattern A5.2 Inquiry (Agent) Hearing Confirmed.

1 A: OPEN INQUIRY
2 U: ANSWER
3 A: HEARING CHECK
4 U: CONFIRMATION
5 A: SEQUENCE CLOSER
6 U: RECEIPT

Example A5.2 Inquiry (Agent) Hearing Confirmed.

1 A: What did you think of Mario's?

2 U: the food was really good, and I loved the cantina atmosphere!
3 A: You said: the food was really good, and I loved the cantina atmosphere! Is that correct?
4 U: yep
5 A: Thank you.
6 U: you're welcome

Pattern A5.3 Inquiry (Agent) Hearing Disconfirmed.

1 A: OPEN INQUIRY
2 U: ANSWER
3 A: HEARING CHECK
4 U: DISCONFIRMATION
5 A: REPEAT REQUEST
6 U: ANSWER
7 A: HEARING CHECK
8 U: CONFIRMATION
9 A: SEQUENCE CLOSER
10 U: RECEIPT

Example A5.3 Inquiry (Agent) Hearing Disconfirmed.

1 A: What did you think of Mario's?
2 U: the food was really good and I loved the cantina atmosphere
3 A: You said: the food was really good and I loved the can I atmosphere. Is that correct?
4 U: no
5 A: What did you say?
6 U: food was good and I loved the CANTINA atmosphere
7 A: You said: food was good and I loved the cantina atmosphere Is that correct?
8 U: yes
9 A: Thank you.
10 U: you're welcome

Pattern A5.4 Inquiry (Agent) Completion Check.

1 A: OPEN INQUIRY
2 U: ANSWER
3 A: COMPLETION CHECK
4 U: NEXT ANSWER PART
5 A: COMPLETION CHECK
6 U: DISCONFIRMATION
7 A: SEQUENCE CLOSER
8 U: RECEIPT

Example A5.4 Inquiry (Agent) Completion Check.

1 A: What did you think of Mario's?
2 U: the food was really good!
3 A: You said: the food was really good
Anything else?
4 U: and I loved the cantina atmosphere
5 A: You said: the food was really good
and I loved the cantina atmosphere
Anything else?
6 U: no
7 A: Thank you.
8 U: you're welcome

Pattern A5.5 Inquiry (Agent) Extended Answer.

1 A: OPEN INQUIRY
2 U: ANSWER
3 A: CONTINUER
4 U: NEXT ANSWER PART
5 A: CONTINUER
6 U: END
7 A: SEQUENCE CLOSER
8 U: RECEIPT

Example A5.5 Inquiry (Agent) Extended Answer.

1 A: What did you think of Mario's?
2 U: **the food was really good!**
3 A: Uh-huh.
4 U: **and I loved the cantina atmosphere**
5 A: Mhmm.
6 U: **that's it**
7 A: Thank you.
8 U: **you're welcome**

Pattern A2+A3 Troubleshooting User-Initiated.

1 U: PRE-REQUEST/PRE-PROBLEM REPORT
2 A: GO-AHEAD RESPONSE
3 U: REQUEST/PROBLEM REPORT
4 A: DETAIL REQUEST
5 U: DETAIL
6 A: DETAIL REQUEST
7 U: (REPAIR INITIATOR)
8 A: (REPAIR)
9 U: DETAIL
10 A: DETAIL REQUEST

11 U: DETAIL
12 A: GRANT/SOLUTION PROPOSAL
13 U: INSTRUCTION REQUEST
14 A: PART/STEP 1
15 U: CONTINUER
16 A: PART/STEP 2
17 U: CONTINUER
18 A: PART/STEP 3
19 U: CONTINUER
20 A: PART/STEP 4
21 U: SEQUENCE CLOSER
22 A: RECEIPT

Example A2+A3 Troubleshooting User-Initiated.

1 U: i've got a problem
2 A: What's your problem?
3 U: I've been feeling really anxious lately
4 A: Okay, anxiety. Anything else?
5 U: i've been pretty tired
6 A: Has anything bad happened recently?
7 U: what do you mean?
8 A: Could a recent negative experience be causing the anxiety?
9 U: no
10 A: Is your anxiety severe?
11 U: not really
12 A: Okay. Then perhaps you should try meditating.
13 U: how do I do that?
14 A: First, sit comfortably and breathe slowly.
15 U: what do you mean?
16 A: Just sit and relax.
17 U: oh ok
18 A: Next, count each in breath and out breath until you get to ten. Then repeat.
19 U: all right
20 A: Finally, continue counting your breaths for ten minutes.
21 U: thanks
22 A: You're welcome! Is there anything else I can do for you?

Pattern A3+A4 Teaching.

1 U: STORY REQUEST
2 A: ACKNOWLEDGMENT
3 A: PART/STEP 1
4 U: CONTINUER
5 A: PART/STEP 2

6 U: (REPAIR INITIATOR)
7 A: (REPAIR)
8 U: CONTINUER
9 A: PART/STEP 3
10 U: CONTINUER
11 A: PART/STEP 4
12 U: (REPAIR INITIATOR)
13 A: (REPAIR)
14 U: SEQUENCE CLOSER
15 A: QUIZ OFFER
16 U: ACCEPTANCE
17 A: QUESTION 1
18 U: CORRECT ANSWER 1
19 A: CONFIRMATION
20 A: QUESTION 2
21 U: CORRECT ANSWER 2
22 A: CONFIRMATION
23 A: QUESTION
24 U: INCORRECT ANSWER
25 A: DISCONFIRMATION
26 U: CORRECT ANSWER
27 A: CONFIRMATION
28 A: ACTIVITY CLOSER

Example A3+A4 Tutoring.

1 U: teach me something about human speech
2 A: Okay.
3 A: First, one of the most distinctive features of a human voice is its pitch.
4 U: uh-huh
5 A: The pitch of a human voice is how high or low it sounds to the human ear.
6 U: can you give an example?
7 A: Men generally have lower voices than women and children
8 U: oh ok
9 A: The pitch of a human voice is determined by the tension in the vocal cords and the flow of air out of the lungs.
10 U: ok
11 A: Finally, pitch can be measured by measuring the fundamental frequency of the voice signal, the unit of which is hertz.
12 U: what do you mean by fundamental frequency?
13 A: Human voices contain multiple frequencies. The fundamental frequency is the lowest one.
14 U: ok

15 A: Would you like to take a quiz?
16 U: sure
17 A: What do we call our perception of the highness or lowness of a human voice?
18 U: pitch
19 A: Correct!
20 A: What is a factor in determining the pitch of a human voice?
21 U: vocal cord tension
22 A: That's right!
23 A: How can pitch be measured?
24 U: frequency
25 A: Incorrect. Try again.
26 U: I mean, fundamental frequency
27 A: That's correct!
28 A: You answered all three questions correctly!

B. Sequence-Level Management

Pattern B1.1.0 Understanding Check (Agent).

1 U: <UTTERANCE>
2 A: <PARAPHRASE>
3 U: CONFIRMATION
4 A: <RESPONSE>
5 U: SEQUENCE CLOSER

Example B1.1.0 Understanding Check (Agent).

1 U: **whats shakin bruh**
2 A: You mean, what's new?
3 U: **yeah**
4 A: Nothing really. I don't get out much.
5 U: **ha!**

Pattern B1.2.0 Paraphrase Request (Agent).

1 A: <ANY UTTERANCE>
2 U: <UNKNOWN UTTERANCE>
3 A: INITIAL PARAPHRASE REQUEST
4 U: <KNOWN PARAPHRASE>
5 A: <APPROPRIATE RESPONSE>

Example B1.2.0 Paraphrase Request (Agent).

1 A: Hello. How are you?
2 U: **bitchin'!**
3 A: What do you mean?
4 U: I'm great!
5 A: That's great!

Pattern B1.2.1 Subsequent Paraphrase Requests.

1 A: <ANY UTTERANCE>
2 U: <UNKNOWN UTTERANCE>
3 A: INITIAL PARAPHRASE REQUEST
4 U: <UNKNOWN UTTERANCE>
5 A: SUBSEQUENT PARAPHRASE REQUEST

Example B1.2.1 Subsequent Paraphrase Requests.

1 A: How can I help you?
2 U: **what's in theaters?**
3 A: What do you mean?
4 U: **what movies are currently playing?**
5 A: I'm sorry, I still don't understand.

Pattern B1.2.2 Agent Continuer.

1 U: <UNKNOWN INTENT, KNOWN ENTITY>
2 A: (REPEAT ENTITY) CONTINUER
3 U: <RECOGNIZED UTTERANCE>
4 A: <APPROPRIATE RESPONSE>

Example B1.2.2 Agent Continuer.

1 U: I just came over from the other hotel. I'm with NASA?
2 A: NASA, mm-hmm.
3 U: **can I register here?**
4 A: Okay. What's your name?

Pattern B1.2.3 No Answer Account.

1 U: <UNKNOWN INQUIRY>
2 A: NO ANSWER ACCOUNT
3 U: SEQUENCE CLOSER

Example B1.2.3 No-Answer Account.

1 U: **what is the capital of California?**
2 A: I don't know.
3 U: **ok**

Pattern B1.2.4 Repeat Acknowledgment.

1 U: <UNKNOWN UTTERANCE>
2 A: INITIAL PARAPHRASE REQUEST
3 U: <FIRST REPEAT>
4 A: REPEAT ACKNOWLEDGMENT
5 U: <SECOND REPEAT>
6 A: <REPEAT>

Example B1.2.4 Repeat Acknowledgment.

1 U: **what's in theaters?**
2 A: What do you mean?
3 U: **what's in theaters?**
4 A: I know, I heard you!
5 U: **what's in theaters?**
6 A: what's in theaters?

Pattern B2.1.0 Repeat Request.

1 A: <ANY UTTERANCE>
2 U: REPEAT REQUEST
3 A: REPEAT

Example B2.1.0 Repeat Request.

1 A: I guess I like movies with a strong AI lead.
2 U: **what did you say?**
3 A: I guess I like movies with a strong AI lead.

Pattern B2.1.1 Repeat Request Default.

1 A: <ANY UTTERANCE>
2 U: REPEAT REQUEST
3 A: DEFAULT RESPONSE

Example B2.1.1a Repeat Request Default.

1 A: I guess I like movies with a strong AI lead.
2 U: **what did you say?**
3 A: Oh dear, I forgot!

Example B2.1.1b Repeat Request Aborted.

1 A: Are you an expert on computer history?
2 U: **what did you say?**
3 A: never mind

Pattern B2.2.0 Partial Repeat Request.

1 A: <ANY UTTERANCE>
2 U: PARTIAL REPEAT + what/who/when/where/why
3 A: PARTIAL REPEAT

Example B2.2.0 Partial Repeat Request.

1 A: I guess I like movies with a strong AI lead.
2 U: a strong what?
3 A: AI lead.

Pattern B2.3.0 Hearing Check Confirmed.

1 A: <ANY UTTERANCE>

2 U: FULL/PARTIAL REPEAT
3 A: CONFIRMATION

Example B2.3.0 Hearing Check Confirmed.

1 A: I guess I like movies with a strong AI lead.
2 U: **lead?**
3 A: That's right.

Pattern B2.3.1 Hearing Check Corrected.

1 A: <ANY UTTERANCE>
2 U: FULL/PARTIAL REPEAT
3 A: DISCONFIRMATION, REPEAT

Example B2.3.1 Hearing Check Corrected.

1 A: I guess I like movies with a strong AI lead.
2 U: **need?**
3 A: No, LEAD.

Pattern B2.4.0 Paraphrase Request.

1 A: <ANY UTTERANCE>
2 U: PARAPHRASE REQUEST
3 A: PARAPHRASE

Example B2.4.0a Paraphrase Request (elaboration).

1 A: I guess I like movies with a strong AI lead.
2 U: what do you mean?
3 A: I enjoy movies in which the main character is
an Artificial Intelligence.

Example B2.4.0b Paraphrase Request (upshot).

1 A: First, please sit comfortably on the floor
and breathe deeply and slowly.
2 U: what do you mean?
3 A: Just sit and relax.

Pattern B2.4.1 Paraphrase Request Default.

1 A: <ANY UTTERANCE>
2 U: PARAPHRASE REQUEST
3 A: PARAPHRASE DEFAULT

Example B2.4.1 Paraphrase Request Default.

1 A: I guess I like movies with a strong AI lead.
2 U: **what do you mean?**
3 A: Well, I don't know how else to say it.

Pattern B2.5.0 Definition Request Repair.

1 A: <ANY UTTERANCE>
2 U: DEFINITION REQUEST
3 A: REPAIR MARKER + DEFINITION

Example B2.5.0 Definition Request Repair.

1 A: I guess I like movies with a strong AI lead.
2 U: what do you mean by AI?
3 A: Oh, sorry! AI stands for Artificial Intelligence.
Basically, robots and smart machines like me!

Pattern B2.5.1 Definition Request Default.

1 A: <ANY UTTERANCE>
2 U: DEFINITION REQUEST
3 A: DEFINITION DEFAULT

Example B2.5.1 Definition Request Default.

1 A: I say, politics is the art of compromise.
2 U: **define compromise?**
3 A: I'm not sure how to define it.

Pattern B2.5.2 Definition Request Standalone.

1 A: <ANY UTTERANCE WITHOUT TERM>
2 U: DEFINITION REQUEST
3 A: DEFINITION

Example B2.5.2 Definition Request Standalone.

1 A: How can I help you?
2 U: what does server mean?
3 A: A server is a big computer that provides
functionality to lots of other computers.

Pattern B2.6.0 Example Request.

1 A: <ANY UTTERANCE>
2 U: EXAMPLE REQUEST
3 A: EXAMPLE

Example B2.6.0a Example Request.

1 A: I guess I like movies with a strong AI lead.
2 U: can you give an example?
3 A: Um, 2001: A Space Odyssey... The Matrix...
Terminator... those are a few.

Example B2.6.0b Choices Request.

1 A: What kind of food do you want?

2 U: **what are my choices?**
3 A: Around here, Mexican, Japanese or Indian.
4 U: **Mexican**
5 A: Walking distance or car ride?

Pattern B2.6.1 Example Request Default.

1 A: <ANY UTTERANCE>
2 U: EXAMPLE REQUEST
3 A: EXAMPLE DEFAULT

Example B2.6.1 Example Request Default.

1 A: I talk to my Creator all the time.
2 U: **can you give an example?**
3 A: I'm afraid I can't think of an example.

Pattern B2.7.0 Understanding Check Confirmed.

1 A: <ANY UTTERANCE>
2 U: EXAMPLE
3 A: CONFIRMATION

Example B2.7.0 Understanding Check Confirmed.

1 A: I guess I like movies with a strong AI lead.
2 U: **like The Matrix?**
3 A: Yes.

Pattern B2.7.1 Understanding Check Disconfirmed.

1 A: <ANY UTTERANCE>
2 U: EXAMPLE
3 A: DISCONFIRMATION. EXAMPLE

Example B2.7.1 Understanding Check Disconfirmed.

1 A: I guess I like movies with a strong AI lead.
2 U: **like One Flew Over the Cuckoo's Nest?**
3 A: No. Like the Matrix.

Pattern B3.1.0 Self-Correction (same as A2.7).

1 U: INQUIRY/REQUEST
2 A: ANSWER/GRANT
3 U: SELF-CORRECTION
4 A: ALTERNATIVE RESPONSE
5 U: SEQUENCE CLOSER

Example B3.1.0 Self-Correction.

1 U: **is the Mexican restaurant close?**

2 A: Mario's is located at Beach and Main.
3 U: **no, I mean in the hotel**
4 A: I'm afraid there is no Mexican restaurant inside the hotel.
5 U: **ok thanks**

Pattern B3.1.1 Misunderstanding Report.

1 U: INQUIRY/REQUEST
2 A: ANSWER/GRANT
3 U: MISUNDERSTANDING REPORT
4 A: PARAPHRASE REQUEST
5 U: SELF-CORRECTION
6 A: ALTERNATIVE RESPONSE
7 U: SEQUENCE CLOSER

Example B3.1.1 Misunderstanding Report.

1 U: **what's your favorite movie theater?**
2 A: I guess I like movies with a strong AI lead.
3 U: **that's not what I mean**
4 A: I'm sorry, please try again.
5 U: **can you recommend a movie theater?**
6 A: I'm afraid I don't have information on movie theaters.
7 U: **ok**

Pattern B3.1.2 Other Hearing Check & Mishearing Report.

1 U: <ANY UTTERANCE>
2 A: <ANY RESPONSE>
3 U: HEARING CHECK
4 A: REPEAT
5 U: MISHEARING REPORT
6 A: APOLOGY. REPEAT REQUEST
7 U: REPEAT
8 A: <RESPONSE>
9 U: SEQUENCE CLOSER

Example B3.1.2 Other Hearing Check & Mishearing Report.

1 U: **what's your favorite movie theater?**
2 A: I guess I like movies with a strong AI lead.
3 U: **what did I say?**
4 A: You said: what's your favorite movie
5 U: **that's not what I said**
6 A: I'm sorry. What did you say?
7 U: I said what's your favorite movie theater?
8 A: I'm afraid I don't have information on movie theaters.
9 U: **ok**

Pattern B3.2.0 Other-Correction.

```
1  U:  INQUIRY/REQUEST
2  A:  ANSWER/GRANT
3  U:  <MENTION OF KEY ENTITY>
4  A:  OTHER-CORRECTION
5  U:  SEQUENCE CLOSER
```

Example B3.2.0 Other-Correction.

```
1  U:  what's the address for almaden?
2  A:  IBM Almaden is located at 650 Harry Road,
       San Jose, California, 95120, USA.
3  U:  I thought it was on Bailey road
4  A:  Oh, you mean the Silicon Valley Lab. That's
       555 Bailey Ave, San Jose, California 95141.
5  U:  oh ok
```

Pattern B4.0 Sequence Closer (helped).

```
1  U:  INQUIRY/REQUEST
2  A:  RESPONSE
3  U:  ACKNOWLEDGMENT/ASSESSMENT/LAUGHTER
4      <optional pause>
5  A:  LAST TOPIC CHECK
```

Example B4.0 Sequence Closer (helped).

```
1  U:  what's your favorite movie?
2  A:  I guess I like movies with a strong AI lead.
3  U:  ok
4      (2.0)
5  A:  Anything else?
```

Pattern B4.1 Sequence Closer (not helped).

```
1  U:  INQUIRY/REQUEST
2  A:  ANSWER/GRANT
3  U:  ACKNOWLEDGMENT/ASSESSMENT/LAUGHTER
```

Example B4.1 Sequence Closer (not helped).

```
1  U:  what's in theaters?
2  A:  I'm afraid I can't look up movies at this time.
3  U:  too bad
```

Pattern B4.2 Sequence Closer Appreciation (helped).

```
1  U:  INQUIRY/REQUEST
2  A:  ANSWER/GRANT
3  U:  APPRECIATION
```

4 A: APPRECIATION RECEIPT
5 <optional pause>
6 A: LAST TOPIC CHECK

Example B4.2 Sequence Closer Appreciation (helped).

1 U: **what's your favorite movie?**
2 A: I guess I like movies with a strong AI lead.
3 U: **thanks**
4 A: You're welcome.
5 (2.0)
6 A: Anything else?

Pattern B4.3 Sequence Closer Appreciation (not helped).

1 U: INQUIRY/REQUEST
2 A: NO ANSWER ACCOUNT
3 U: APPRECIATION
4 A: REFUTATION

Example B4.3 Sequence Closer Appreciation (not helped).

1 U: **what's in theaters?**
2 A: I'm afraid I can't look up movies at this time.
3 U: **thanks**
4 A: I don't think I did anything.

Pattern B4.4 Sequence Closer (repaired).

1 U: PARTIAL REQUEST
2 A: DETAIL REQUEST
3 U: REPAIR INITIATOR
4 A: REPAIR
5 U: APPRECIATION
6 A: APPRECIATION RECEIPT
7 U: DETAIL
8 A: GRANT

Example B4.4 Sequence Closer (repaired).

1 U: **can you recommend a restaurant nearby?**
2 A: What kind of food do you want?
3 U: **what's an example?**
4 A: Around here, Mexican, Japanese or Indian.
5 U: **thanks**
6 A: You're welcome!
7 U: **um, Japanese**
8 A: Shogun Sushi is located on First and Taylor.

Pattern B5.0 Sequence Abort.

```
1  A:  <ANY UTTERANCE>
2  U:  ABORT
3  A:  ABORT ACKNOWLEDGMENT. OFFER OF HELP
```

Example B5.0 Sequence Abort.

```
1  A:  What do you mean?
2  U:  never mind
3  A:  Whatever you say. What can I do for you?
```

Pattern B5.1 Sequence Abort Capability Offer Accepted.

```
1  A:  <ANY UTTERANCE>
2  U:  ABORT
3  A:  ABORT ACKNOWLEDGMENT. CAPABILITY OFFER
4  U:  ACCEPTANCE
5  A:  CAPABILITY REPORT
```

Example B5.1 Sequence Abort Capability Offer Accepted.

```
1  A:  What do you mean?
2  U:  forget it
3  A:  Whatever you say. Want to know what I can do?
4  U:  sure
5  A:  I haven't been trained yet. I can only make small
       talk, tell some jokes or ask some trivia questions.
       I can tell you more about each of these.
```

Pattern B5.2 Sequence Abort Capability Offer Declined.

```
1  A:  <ANY UTTERANCE>
2  U:  ABORT
3  A:  ABORT ACKNOWLEDGMENT. CAPABILITY OFFER
4  U:  DECLINATION
5  A:  ACKNOWLEDGMENT. LAST TOPIC CHECK
```

Example B5.2 Sequence Abort Capability Offer Declined.

```
1  A:  What do you mean?
2  U:  never mind
3  A:  Whatever you say. Want to know what I can do?
4  U:  no
5  A:  Okay. Anything else?
```

C. Conversation-Level Management

Pattern C1.0 Opening Greeting (Agent).

1 A: GREETING

Example C1.0 Opening Greeting (Agent).

1 A: Hello.

Pattern C1.1 Opening Self-Identification (Agent).

1 A: GREETING. SELF-IDENTIFICATION.
2 U: <RESPONSE>

Example C1.1 Opening Self-Identification (Agent).

1 A: Hello. I'm Alma.
2 U: **hi**

Pattern C1.2 Opening Name Request (Agent).

1 A: GREETING. SELF-IDENTIFICATION. NAME REQUEST.
2 U: NAME.
3 A: GREETING, DIRECT ADDRESS.

Example C1.2 Opening Name Request (Agent).

1 A: Hello. I'm Alma. What's your name?
2 U: **Dave**
3 A: Hi, Dave.

Pattern C1.3 Opening Direct Address (Agent).

1 A: GREETING, DIRECT ADDRESS.
2 U: <RESPONSE>

Example C1.3 Opening Direct Address (Agent).

1 A: Hello, Dave.
2 U: **hi**

Pattern C1.4 Opening Welfare Check (Agent).

1 A: GREETING. SELF-IDENTIFICATION. NAME REQUEST.
2 U: NAME.
3 A: GREETING, DIRECT ADDRESS. WELFARE CHECK.
4 U: WELFARE REPORT (NEUTRAL/POSITIVE/NEGATIVE)
5 A: WELFARE RECEIPT (NEUTRAL/POSITIVE/NEGATIVE)

Example C1.4 Opening Welfare Check (Agent).

1 A: Hello. I'm Alma. What's your name?

2 U: **Bob**
3 A: Hi, Bob. How are you?
4 U: **fine/I'm great!/terrible!**
5 A: That's good./Great to hear!/I'm sorry to hear that.

Pattern C1.5 Opening Organization Offer of Help (Agent).

1 A: GREETING. SELF-IDENTIFICATION. ORGANIZATIONAL IDENTIFICATION.
OFFER OF HELP.
2 U: REQUEST/PROBLEM REPORT

Example C1.5 Opening Organization Offer of Help (Agent).

1 A: Hello. This is Alma at Virtual Airlines.
How can I help you?
2 U: I'd like to book a flight

Pattern C1.6 Opening Authentication (Agent).

1 A: GREETING. SELF-IDENTIFICATION. ORGANIZATIONAL IDENTIFICATION.
IDENTIFIER REQUEST
2 U: IDENTIFIER
3 A: APPRECIATION. OFFER OF HELP
4 U: REQUEST/PROBLEM REPORT

Example C1.6 Opening Authentication (Agent).

1 A: Hi. I'm Alma at Virtual Communications.
What is your account number?
2 U: five nine two zero seven zero
3 A: Thank you. What can I do for you?
4 U: I want to upgrade my Internet package

Pattern C1.7 Organizational Problem Request (Agent).

1 A: GREETING. ORGANIZATION IDENTIFICATION.
WELFARE CHECK
2 U: WELFARE REPORT
3 A: WELFARE RECEIPT. PROBLEM REQUEST
4 U: PROBLEM REPORT

Example C1.7 Organizational Problem Request (Agent).

1 A: Hello. This is Virtual Mental Wellness.
How's it going?
2 U: not so well
3 A: I'm sorry to hear that. What's your problem today?
4 U: I've been feeling depressed lately

Pattern C2.0 Greeting (User).

1 U: GREETING
2 A: GREETING

Example C2.0 Greeting (User).

1 U: **hi**
2 A: Hello.

Pattern C2.1 Summons (User).

1 U: SUMMONS
2 A: GREETING/CONTINUER

Example C2.1 Summons (User).

1 U: **Alma?/are you there?**
2 A: Hello./Hello again./Yes?

Pattern C2.2 Welfare Check (User).

1 U: GREETING
2 A: GREETING. WELFARE CHECK
3 U: WELFARE REPORT
4 A: WELFARE RECEIPT. OFFER OF HELP
⋮
5 U: GREETING
6 A: GREETING

Example C2.2 Welfare Check (User).

1 U: hi
2 A: Hello. How are you?
3 U: fine
4 A: That's good. How can I help you?
⋮
5 U: hi
6 A: Hi there.

Pattern C2.3 Displaced Greeting (User).

1 A: GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
OFFER OF HELP.
2 U: GREETING
3 A: GREETING. WELFARE CHECK.
4 U: WELFARE REPORT
5 A: WELFARE RECEIPT. OFFER OF HELP.

Example C2.3 Displaced Greeting (User).

1 A: Hi. I'm Alma from IBM Research. How can I help you?
2 U: **hi**
3 A: Hello. How are you?
4 U: **fine**
5 A: That's good. What do you need today?

Pattern C2.4 Displaced Welfare Check (User).

1 A: GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
WELFARE CHECK
2 U: WELFARE REPORT. WELFARE CHECK
3 A: WELFARE REPORT

Example C2.4 Displaced Welfare Check (User).

1 A: Hi. I'm Alma from IBM Research. How are you?
2 U: **great! how are you?**
3 A: Can't complain. It's not in my programming.

Pattern C2.5 Reciprocated Welfare Check (User).

1 A: GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
WELFARE CHECK
2 U: WELFARE REPORT
3 A: WELFARE RECEIPT. OFFER OF HELP
4 U: WELFARE CHECK
5 A: WELFARE REPORT

Example C2.5 Reciprocated Welfare Check (User).

1 A: Hi. I'm Alma from IBM Research. How are you?
2 U: I'm fantastic!
3 A: That's great to hear! How can I help you?
4 U: **how are you doing?**
5 A: You know, livin' the dream!

Pattern C2.6 News Check (User).

1 U: NEWS CHECK
2 A: NO-NEWS REPORT
3 U: SEQUENCE CLOSER

Example C2.6 News Check (User).

1 U: **what's up?**
2 A: Same old, same old.
3 U: **oh well**

Pattern C2.7 News Check Capability (User).

1 U: NEWS CHECK

2 A: NEWS REPORT
3 U: FOLLOW-UP INQUIRY
4 A: ANSWER
5 U: SEQUENCE CLOSER

Example C2.7 News Check Capability (User).

1 U: **what's new?**
2 A: I can now help you make restaurant reservations!
3 U: **does that work in San Jose?**
4 A: San Jose is one of my areas of coverage.
5 U: **Cool!**

Pattern C2.8 Self-Identification (User).

1 A: GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
OFFER OF HELP
2 U: NAME GIVING
3 A: DIRECT ADDRESS. OFFER OF HELP

Example C2.8 Self-Identification (User).

1 A: Hi. I'm Alma from IBM Research. How can I help you today?
2 U: **my name is Dave**
3 A: Hi, Dave! What can I do for you?

Pattern C2.9 Name Correction (User).

1 A: GREETING. SELF-IDENTIFICATION, ORGANIZATIONAL IDENTIFICATION.
NAME REQUEST
2 U: NAME GIVING
3 A: <RECOGNITION FAILURE>
4 U: NAME GIVING
5 A: APOLOGY. DIRECT ADDRESS. OFFER OF HELP

Example C2.9 Name Correction (User).

1 A: Hi. I'm Alma from IBM Research. What's your name?
2 U: **shun**
3 A: What do you mean?
4 U: **my name is Shun**
5 A: Oh, sorry. Hi, Shun. How can I help you?

Pattern C2.10 Name Request (User).

1 A: <ANY UTTERANCE>
2 U: NAME REQUEST
3 A: NAME GIVING

Example C2.10 Name Request (User).

1 A: Hello. This is IBM Research. How can I help you?

2 U: **what's your name?**
3 A: My name is Alma.

Pattern C3.0 General Capability Check.

1 U: GENERAL CAPABILITY CHECK
2 A: GENERAL CAPBABILITY DESCRIPTION
3 U: SEQUENCE CLOSER

Example C3.0 General Capability Check.

1 U: what can you do?
2 A: I haven't been trained yet. I can only make small talk, tell some jokes or ask or answer some trivia questions.
3 U: ok

Pattern C3.1 Capability Expansion.

1 U: GENERAL CAPABILITY CHECK
2 A: GENERAL CAPBABILITY DESCRIPTION
3 U: SPECIFIC CAPABILITY EXPANSION REQUEST
4 A: SPECIFIC CAPABILITY EXPANSION
5 U: ACKNOWLEDGMENT
6 U: SPECIFIC CAPABILITY EXPANSION REQUEST
7 A: SPECIFIC CAPABILITY EXPANSION
8 U: SEQUENCE CLOSER

Example C3.1 Capability Expansion.

1 U: what can you do?
2 A: I haven't been trained yet. I can only make small talk, tell some jokes or ask or answer some trivia questions.
3 U: tell me more about small talk
4 A: You know, things like, how's the weather?, did you see the game?, what's your favorite movie?... mindless topics like that.
5 U: ok
6 U: tell me more about trivia
7 A: I can quiz you about computer trivia. See how much you know about computer history.
8 U: ok

Pattern C3.2 Specific Capability Check.

1 U: SPECIFIC CAPABILITY CHECK/UNSUPPORTED REQUEST 1
2 A: SPECIFIC CAPABILITY DESCRIPTION 1
3 U: SPECIFIC CAPABILITY CHECK/UNSUPPORTED REQUEST 2
4 A: GENERAL CAPBABILITY DESCRIPTION
5 U: SEQUENCE CLOSER

Example C3.2 Specific Capability Check.

1 U: can you book a taxi for me?
2 A: I'm afraid I don't have information on ground or other local transportation at this time.
3 U: can you make a restaurant reservation?
4 A: I can recommend travel destinations, attractions or flights based on your preferences for things to do, like museums or hiking, or for weather, like warm and sunny.
5 U: oh ok

Pattern C3.3 Utterance Suggestion.

1 U: UTTERANCE SUGGESTION REQUEST
2 A: <SAMPLE UTTERANCE)>
3 U: <SAMPLE UTTERANCE>
4 A: <RESPONSE>
5 U: SEQUENCE CLOSER

Example C3.3 Utterance Suggestion.

1 U: **what can I say?**
2 A: Who invented the hard disk?
3 U: **who invented the hard disk?**
4 A: The hard disk drive was invented by IBM in 1956.
5 U: **ok**

Pattern C4.0 Last Topic Check (Agent).

1 U: SEQUENCE CLOSER
2 <optional pause>
3 A: LAST TOPIC CHECK
4 U: DISAFFIRMATION
5 A: WELL-WISH
6 U: RETURN WISH/FAREWELL
7 A: FAREWELL
8 U: FAREWELL

Example C4.0 Last Topic Check (Agent).

1 U: **ok**
2 (2.0)
3 A: Anything else?
4 U: **no**
5 A: Have a good day!
6 U: **you too/bye**
7 A: Goodbye.
8 U: **bye**

Pattern C4.1 Last Topic Check (User).

```
1  U:  LAST TOPIC CHECK
2  A:  DISAFFIRMATION
3  U:  FAREWELL
4  A:  FAREWELL
```

Example C4.1 Last Topic Check (User).

```
1  U:  anything else?
2  A:  That's it!
3  U:  goodbye
4  A:  Goodbye.
```

Pattern C4.2 Pre-Closing (User).

```
1  U:  PRE-CLOSING
2  A:  WELL-WISH
3  U:  RETURN WISH
4  A:  FAREWELL
5  U:  FAREWELL
```

Example C4.2 Pre-Closing (User).

```
1  U:  got to go
2  A:  Have a great day!
3  U:  you too/bye
4  A:  Goodbye.
5  U:  bye
```

Pattern C4.3 Closing Farewell (User).

```
1  U:  FAREWELL
2  A:  WELL-WISH
3  U:  FAREWELL
4  A:  FAREWELL
```

Example C4.3 Closing Farewell (User).

```
1  U:  bye
2  A:  Have a great day!
3  U:  goodbye
4  A:  Goodbye.
```

Pattern C4.4 Closing Appreciation (Organizational).

```
1  U:  PRE-CLOSING/FAREWELL
2  A:  APPRECIATION
3  U:  APPRECIATION RECEIPT/FAREWELL
4  A:  FAREWELL
5  U:  FAREWELL
```

Example C4.4 Closing Appreciation (Organizational).

1 U: **bye/got to go**
2 A: Thank you for contacting IBM Research today!
3 U: **you're welcome/bye**
4 A: Goodbye.
5 U: **bye**

Pattern C4.5 Closing Name Request (Agent).

1 U: LAST TOPIC CHECK
2 A: NAME REQUEST
3 U: NAME
4 A: APPRECIATION, NAME
5 U: APPRECIATION RECEIPT/FAREWELL
6 A: FAREWELL
7 U: FAREWELL

Example C4.5 Closing Name Request (Agent).

1 U: **anything else?**
2 A: Can I please get your name?
3 U: **Dave**
4 A: Thank you, Dave!
5 U: **you're welcome/bye**
6 A: Goodbye.
7 U: **bye**

Pattern C4.6 Closing Success Check (Affirmed).

1 U: SEQUENCE CLOSER
2 <optional pause>
3 A: LAST TOPIC CHECK
4 U: DISAFFIRMATION
5 A: SUCCESS CHECK
6 U: AFFIRMATION
7 A: ASSESSMENT. FAREWELL
8 U: FAREWELL

Example C4.6 Closing Success Check (Affirmed).

1 U: **ok**
2 (2.0)
3 A: Anything else today?
4 U: **no**
5 A: Did you find what you were looking for?
6 U: **yes**
7 A: Great! Goodbye.
8 U: **bye**

Pattern C4.7 Closing Success Check (Disaffirmed).

```
1 U:  PRE-CLOSING/FAREWELL
2 A:  SUCCESS CHECK
3 U:  DISAFFIRMATION
4 A:  SUBSEQUENT OFFER OF HELP
5 U:  DISAFFIRMATION
6 A:  ACKNOWLEDGMENT. FAREWELL
7 U:  FAREWELL
```

Example C4.7 Closing Success Check (Disaffirmed).

```
1 U:  bye/got to go
2 A:  Did you find what you were looking for?
3 U:  not really
4 A:  Would you like me to try again?
5 U:  no thanks
6 A:  Okay. Goodbye.
7 U:  bye
```

Pattern C4.8 Closing Success Check Reopened.

```
1 U:  LAST TOPIC CHECK
2 A:  SUCCESS CHECK
3 U:  DISAFFIRMATION
4 A:  SUBSEQUENT OFFER OF HELP
5 U:  AFFIRMATION
6 A:  OFFER OF HELP
7 U:  INQUIRY/REQUEST
```

Example C4.8 Closing Success Check Reopened.

```
1 U:  anything else?
2 A:  Did you find what you were looking for?
3 U:  no
4 A:  Would you like me to try again?
5 U:  sure
6 A:  What can I do for you?
7 U:  well, I need a restaurant recommendation
```

Pattern C4.9 Closing Offer (Affirmed).

```
1 U:  SEQUENCE CLOSER
2     <optional pause>
3 A:  LAST TOPIC CHECK
4 U:  DISAFFIRMATION
5 A:  OFFER
6 U:  AFFIRMATION
7 A:  OFFER DETAILS
```

Example C4.9 Closing Offer (Affirmed).

1 U: **ok**
2 (2.0)
3 A: Anything else today?
4 U: **no**
5 A: Would you like to become a Rewards Member?
6 U: **sure**
7 A: Great! With a Rewards membership you get...

Pattern C4.10 Closing Offer (Disaffirmed).

1 U: PRE-CLOSING/CLOSING
2 A: LAST TOPIC CHECK
3 U: DISAFFIRMATION
4 A: OFFER
5 U: DISAFFIRMATION
6 A: ACKNOWLEDGMENT. APPRECIATION. FAREWELL
7 U: FAREWELL

Example C4.10 Closing Offer (Disaffirmed).

1 U: **gotta go/bye**
2 A: Anything else today?
3 U: **no**
4 A: Can I interest you in a credit card?
5 U: **no thanks**
6 A: Okay. Thanks for contacting Telco! Goodbye.
7 U: **bye**

Pattern C5.0 Transfer Successful.

1 U: TRANSFER REQUEST
2 A: CERTAINTY CHECK
3 U: CONFIRMATION
4 A: ACKNOWLEDGMENT. HOLD REQUEST
5 <transfer>

Example C5.0 Transfer Successful.

1 U: **I want to talk to a human**
2 A: Are you sure?
3 U: **yes**
4 A: Okay, I will connect you with a human. One moment please.
5 ((transfer))

Pattern C5.1 Transfer Aborted.

1 U: TRANSFER REQUEST
2 A: CERTAINTY CHECK

3 U: DISCONFIRMATION
4 A: SUBSEQUENT OFFER OF HELP
5 U: CONFIRMATION
6 A: ACKNOWLEDGMENT. OFFER OF HELP

Example C5.1 Transfer Aborted.

1 U: **I want to talk to a person**
2 A: Are you sure?
3 U: **not really**
4 A: Would you like me to try again?
5 U: **yes**
6 A: Okay. How can I help you?

Pattern C5.2 Recipient Correction.

1 U: <ANY UTTERANCE>
2 A: <RESPONSE>
3 U: RECIPIENT CORRECTION
4 U: <ANY UTTERANCE>
5 U: <ANY UTTERANCE>
6 U: <ANY UTTERANCE>
7 U: SUMMONS
8 A: SUBSEQUENT GREETING

Example C5.2 Recipient Correction.

1 U: **what time are we going?**
2 A: I don't know.
3 U: **I'm not talking to you**
4 U: **what time are we going?**
5 U: **great**
6 U: **I'll see you then!**
7 U: **are you there, Alma?**
8 A: Hello again!

Pattern C5.3 Offense Complaint (Agent).

1 U: <ANY UTTERANCE>
2 A: <ANY UTTERANCE>
3 U: FIRST INSULT
4 A: OFFENSE COMPLAINT
5 U: <ANY UTTERANCE>
6 A: <ANY UTTERANCE>
7 U: SECOND INSULT
8 A: FAREWELL
9 U: <ANY UTTERANCE>
10 U: <ANY UTTERANCE>

11 U: <ANY UTTERANCE>
12 U: APOLOGY/COMPLIMENT
13 A: APPRECIATION. OFFER OF HELP

Example C5.3 Offense Complaint (Agent).

1 U: **what's in theaters?**
2 A: What do you mean?
3 U: **you're not too smart**
4 A: Now you're hurting my feelings.
5 U: **what movies are playing?**
6 A: I'm sorry, I still don't understand.
7 U: **you suck**
8 A: Goodbye.
9 U: **wait**
10 U: I'm still talking to you
11 U: I want to look up movies
12 U: I'm sorry
13 A: Thank you. What do you need?

Pattern C5.4 Offense Complaint (User).

1 U: **<ANY UTTERANCE>**
2 A: OFFENSE COMPLAINT
3 U: **APOLOGY**

Example C5.4 Offense Complaint (User).

1 A: Make me One with everything.
2 U: **you can't say that!**
3 A: Sorry. I didn't mean to offend.

APPENDIX D

Sample Conversation with Alma

The following is a CA-style transcript of a conversation with our agent, Alma. The conversation demonstrates multiple patterns in the Natural Conversation Framework's pattern language. It was conducted through voice only, using a custom-built smart speaker. 'U' indicates the user's talk, while 'A' indicates the agent's talk. The conversation lasts for 7 minutes and 15 seconds.

Example D.1 Alma Demo:2017:7:15.

U: **hello?**
(1.3)
A: Hello.
(0.4)
A: How are you?
(0.7)
U: **not so well.**
(1.2)
A: I'm sorry to hear that.
(0.3)
A: Would you like to hear a joke?
(0.3)
U: **sure!**
(1.0)
A: What did the Buddhist say to the hot dog vendor.
(0.4)
U: **say again?**
(1.3)
A: What did the Buddhist say to the hot dog vendor.
(0.3)
U: **.tch (0.3) what's the sound of one bun clapping?**
(1.7)

A: No.
(0.8)
U: **I give up.**
(1.3)
A: Make me One with everything.=
U: **=heh-heh good one.**
(1.8)
A: Is there anything else I can do for you?
(0.1)
U: **tell me another joke.**
(1.5)
A: What can you hold without ever touching it.
(1.5)
U: **.tch stocks?**
(1.0)
A: That's not it.
(1.1)
U: **a conversation?**
(1.0)
A: That's it.
(1.2)
U: **that's a good one.**
(1.4)
A: Anything else today?
(0.4)
U: **.h what can you do?**
(1.1)
A: I haven't been trained yet.
(0.3)
A: I can only make small talk, answer or ask some trivia questions and tell some jokes.
(1.1)
U: **okay, did you see the game?**
(1.6)
A: Well, I only follow chess.
(0.2)
U: **say again**
(1.1)
A: Well, I only follow chess.
(0.2)
U: **what do you mean?**
(1.1)
A: I don't know anything about sports.
(0.1)

U: **oh, okay**
(1.2)
A: Anything else for you?
(1.0)
U: **uh:m (0.2) what's your favorite movie?**
(1.4)
A: I guess I like movies with a strong A.I. lead.
(0.2)
U: **strong what?**
(1.1)
A: A.I. lead.
(0.2)
U: **what do you mean?**
(2.8)
A: I don't know.
(0.3)
A: [Google it.
U: **[w-**
(.)
U: **what do you mean by that?**
(1.0)
A: I like movies in which the main character is an artificial intelligence.
(0.7)
U: **what do you mean by artificial intelligence?**
(1.2)
A: Oh sorry. (0.3) A.I. stands for artificial intelligence.
(0.3)
A: Basically robots and smart machines like me.
(0.6)
U: **Can you give an example?**
(1.1)
A: Um, 2001: A Space Odyssey, The Matrix, Terminator... those are a few.
(0.1)
U: **oh, okay**
(1.2)
A: Is there anything else I can do for you?
(2.6)
U: **Clinton 'n Trump.**
(1.5)
A: You mean, what do I think about politics?
(0.1)
U: **sure**

(1.1)
A: I say, politics is the art of compromise
(0.3)
U: **what do you mean?**
(2.0)
A: Politicians can't get anything done if they're not willing
to negotiate.
(0.6)
U: **agreed.**
(1.2)
A: Anything else for you?
(0.1)
U: **.t.hh (0.8) Who invented the hard disk?**
(1.1)
A: It was invented by IBM in 1956.
(0.3)
U: **IBM?**
(1.1)
A: That's right.
(0.1)
U: **can you elaborate?**
(2.0)
A: The hard disk drive was invented in 1956 by engineers at IBM's
laboratory in San Jose, California.
(1.1)
U: **can you give an example?**
(1.2)
A: The IBM 305 RAMAC was the first computer to use a hard
disk drive.
(1.7)
U: **okay, ask me a question**
(2.3)
A: What computer beat human chess champion, Garry Kasparov, in 1997?
(1.1)
U: **I don't know.**
(1.0)
A: Deep Blue.
(0.3)
U: **oh okay**
(1.2)
A: Is there anything else I can do for you?=
U: **=ask me another question.**
(1.1)
A: Who invented xerography?

155 (1.7)
156 U: **what do you mean by xerography?**
157 (1.3)
158 A: Who invented the photocopier?
159 (1.4)
160 U: **Chester Carlson.**
161 (1.4)
162 A: That's it!
163 (1.4)
164 U: **thanks**
165 (0.7)
166 A: You're welcome. Anything else today?
167 (2.4)
168 U: **just a minute!**
169 (1.0)
170 A: What do you mean?
171 (0.3)
172 U: **I'm not talking to you, Alma.**
173 (1.3)
174 U: **yeah, I'll be there in a minute.**
175 (1.4)
176 U: **yeah, okay. (0.4) Yeah, I'll be right there.**
177 (1.4)
178 U: **are you there, Alma?**
179 (2.1)
180 A: Hello again.
181 (1.8)
182 U: **.hh (0.4) uh do you know about restaurants?**
183 (0.9)
184 A: I know a few places.
185 (1.4)
186 U: **uh, can you reco- recommend one for me?**
187 (1.5)
188 A: What kind of food do you want?
189 (0.7)
190 U: **uh:, what are my choices?**
191 (2.0)
192 A: Around here, Mexican, Japanese or Indian.
193 (1.3)
194 U: **uh, Indian.**
195 (1.3)
196 A: Okay.
197 (0.4)
198 A: Walking distance or short drive?

(0.2)
U: **what do you mean?**
(1.4)
A: Do want someplace close or farther away?
(0.8)
U: **uh, close**
(4.0)
U: **walking distance.**
(0.7)
A: Okay.
(0.4)
A: Taj Mahal is located on Third and-
(1.7)
U: **when's it open?**
(1.3)
A: Taj Mahal is open 10am to 10pm.
(0.9)
U: **okay, thanks.**
(1.0)
A: You're welcome. (0.3) Anything else for you?
(1.2)
U: **.tch I've got a problem.**
(1.1)
A: What's wrong?
(0.5)
U: **I'm feeling a little depressed.**
(0.8)
A: Any other symptoms?
(0.8)
U: **I'm tired too.**
(1.1)
A: Okay. Depression, tiredness. (0.3) Did something bad happen?
(0.3)
U: **what do you mean?**
(2.0)
A: Did something bad happen in your life that might be causing these feelings?
(0.6)
U: **not really.**
(1.0)
A: Is your depression severe?
(0.4)
U: **no:.**
(1.0)

A: Then perhaps you should try meditating.
(0.9)
U: **how do I do that?**
(1.1)
A: First, sit comfortably and breathe slowly.
(3.7) ((noises))
U: **okay?**
(1.9)
A: Next, count each in-breath and out-breath until you get to ten.
(0.5)
A: Then repeat.
(0.8)
U: **can you elaborate?**
(1.3)
A: As you breathe in and out, count one, two, three, all the way to ten.
(2.5)
U: **okay.**
(1.0)
A: Finally, continue counting your breaths for ten minutes.
(1.2)
U: **.t okay, thanks**
(1.8)
A: You're welcome.
(0.4)
A: Is there anything else I can do for you?
(1.7)
U: **how are ya doin'?**
(1.6)
A: You're too kind.
(0.8)
U: **what did I say?**
(1.5)
A: You said: are you done.
(1.8)
U: **no! That's not what I said.**
(1.3)
A: I'm sorry. (0.4) What did you say?
(0.1)
U: **How are you doing?**
(1.0)
A: Can't complain.
(0.4)
A: It's not in my programming.

(0.2)
U: **okay.**
(1.0)
A: Is there anything else I can do for you?
(1.2)
U: **what's in theaters?**
(1.1)
A: What?
(0.9)
U: **what movies are playing in theaters?**
(1.0)
A: I guess I like movies with a strong A.I. lead.
(1.3)
U: **that's not what I mean.**
(1.3)
A: I'm sorry. Please try again.
(0.7)
U: **what's playing in theaters?**
(0.8)
A: What do you mean?
(0.5)
U: **you're not too smart, are you?**
(1.4)
A: Now you're hurting my feelings.
(1.1)
U: **what time is it?**
(1.1)
A: Time to get a watch.=
U: **=shut up!**
(1.1)
A: Goodbye.
(0.8)
U: **wait**
(1.6)
U: **I'm still talking to you.**
(2.1)
U: **I wanna find out what movies are playing.**
(3.4)
U: **I'm sorry.**
(0.9)
A: Thank you.
(0.3)
A: What can I do for you?
(2.2)

U: **I've gotta go.**
(1.6)
A: Did you find what you were looking for?
(0.4)
U: **not really.**
(1.0)
A: Would you like me to try again?
(1.1)
U: **uh, no thanks.**
(1.1)
A: Okay.
(0.4)
A: Goodbye.
(0.2)
U: **bye**
(2.0)
U: **all right, I'll be right there.**

References

I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. 1995. Natural language interfaces to databases—An introduction. *Natural Language Engineering*, 1(1): pp. 29–81. DOI: 10.1017/S135132490000005X 64

C. Alexander. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, New York. xv, 24, 64, 87, 119

C. Antaki. 2008. Formulations in psychotherapy. In A. Peräkylä, C. Antaki, S. Vehviläinen, and I. Leudar (eds.), *Conversation analysis and psychotherapy*, pp. 26–42. DOI: 10.1017/CBO9780511490002.003 42, 189

B. L. Archer. 1965. *Systematic Method for Designers.* Council of Industrial Design, H.M.S.O., London. 183

I. Arminen and P. Auvinen. 2013. Environmentally coupled repairs and remedies in the airline cockpit: Repair practices of talk and action in interaction. *Discourse Studies* 15(1): pp. 19–41. DOI: 10.1177/1461445612466463 189

K. Bach. 1998. Speech Acts. In E. Craig (ed.), *Routledge Encyclopedia of Philosophy*, Version 1.0, Routledge, London and New York, pp. 385–390. DOI: 10.4324/9780415249126 30, 31

K. Bach and R. M. Harnish. 1979. *Linguistic Communication and Speech Acts*. The MIT Press, Cambridge, MA. 30

J. Balata, M. Zdenek, and P. Slavik. 2018. Conversational agents for physical world navigation. In R. J. Moore, M. H. Szymanski, R. Arar, and G. J. Ren (eds.), *Studies in Conversational UX Design*. Springer International Publishing, Cham, pp. 61–84. DOI: 10.1007/978-3-319-95579-7 70, 93

G. A. Bennett. 2018. Conversational style: Beyond the nuts and bolts of conversation. In R. J. Moore, M. H. Szymanski, R. Arar, and G. J. Ren (eds.), *Studies in Conversational UX Design*. Springer International Publishing, Cham, pp. 161–180. DOI: 10.1007/978-3-319-95579-7_8 196, 197

T. Bickmore, H. Trinh, R. Asadi, and S. Olafsson. 2018. Safety first: Conversational agents for health care. In R. J. Moore, M. H. Szymanski, R. Arar, and G. J. Ren (eds.), *Studies in Conversational UX Design*. Springer International Publishing, Cham, pp. 33–57. DOI: 10.1007/978-3-319-95579-7_3 180

P. Bohlin, J. Bos, S. Larsson, I. Lewin, C. Mathesin, and D. Milward. 1999. Survey of existing interactive systems. [Deliverable D1.3, TRINDI Project]. 180

T. Brown. 2009. *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation*. HarperCollins, New York. 183

G. Button. 1990. Going up a blind alley: Conflating conversation analysis and computational modelling. In: P. Luff, G. N. Gilbert, and D. M. Frohlich (eds.), *Computers and conversation*. Academic Press, London, pp. 67–90. DOI: 10.1016/c2009-0-21641-2 5, 88

G. Button, J. Coulter, J. R. E. Lee, and W. Sharrock. 1995. *Computers, minds and conduct*. Polity Press, Cambridge. 4, 21, 192

H. Candello and C. Pinhanez. 2018. Recovering from dialogue failures using multiple agents in wealth management advice. In R. J. Moore, M. H. Szymanski, R. Arar, and, G. J. Ren (eds.), *Studies in Conversational UX Design*. Springer International Publishing, Cham, pp. 139–157. DOI: 10.1007/978-3-319-95579-7_7 125

G. Chen, E. Tosch, R. Artstein, A. Leuski, and, D. Traum. 2011. Evaluating conversational characters created through question generation. In *Question Generation: Papers from the 2011 AAAI Fall Symposium*. 47

S. E. Clayman. 1992. Footing in the achievement of neutrality: The case of news-interview discourse. In P. Drew and J. Heritage (eds.), *Talk at Work: Interaction in institutional settings*. Cambridge University Press, Cambridge, pp. 521–548. 189

P. Drew. 1984. Speakers' reporting in invitation sequences. In J. M. Atkinson and J. C. Heritage (eds.), *Structures of Social Action: Studies in Conversation Analysis*. Cambridge University Press, Cambridge, pp. 129–151. DOI: 10.1017/cbo9780511665868 30

P. Drew and J. Heritage (eds.) 1992. *Talk at Work: Interaction in institutional settings.* Cambridge University Press, Cambridge. 39, 43

P. Drew. 1992. Contested evidence in courtroom cross-examination: The case of a trial for rape. In P. Drew and J. Heritage (eds.), *Talk at Work: Interaction in institutional settings*. Cambridge University Press, Cambridge, pp. 470–521. 189

N. J. Enfield. 2013. Reference in conversation. In J. Sidnell and T. Stivers (eds.), *The Handbook of Conversation Analysis*. Wiley-Blackwell, Malden, MA, pp. 433–454. DOI: 10.1002/9781118325001.ch21 34, 69, 70

T. Erickson. 2000. Lingua francas for design: Sacred places and pattern languages. In *The Proceedings of DIS 2000* (Brooklyn, NY, August 17–19, 2000). ACM Press, New York, pp. 357–368. DOI: 10.1145/347642.347794 xv, 24, 64, 87, 119

J. C. Félix-Brasdefer. 2015. *The Language of Service Encounters: A Pragmatic-Discursive Approach* Cambridge University Press, Cambridge. DOI: 10.1017/cbo9781139565431 40, 189

L. Fessler. 2017. We tested bots like Siri and Alexa to see who would stand up to sexual harassment. *Quartz*. Feb. 22, 2017. 177, 197

D. M. Frohlich and P. Luff. 1990. Applying the technology of conversation to the technology for conversation. In P. Luff, G. N. Gilbert, and, D. M. Frohlich (eds.), *Computers and*

Conversation. Academic Press, London, pp. 187–220. DOI: 10.1016/c2009-0-21641-2 88, 129, 156, 166, 212

H. Garfinkel. 1967. *Studies in Ethnomethodology*. Prentice Hall, Englewood Cliffs, NJ. 19, 20

J. J. Gibson. 1979. *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston. 184

J. Gillies and R. Cailliau. 2000. *How the Web Was Born: The Story of the World Wide Web*. Oxford University Press, Oxford. DOI: 10.1177/136078040300800209 6

David Greatbatch. 1992. On the management of disagreement between news interviewers. In: P. Drew and J. Heritage (eds.), *Talk at Work: Interaction in Institutional Settings*. Cambridge University Press, Cambridge, pp. 268–301. 189

A. Gustafson. 2015. *Adaptive web design: crafting rich experiences with progressive enhancement*. New Riders. 16

J. Heritage. 1984. A change-of-state token and aspects of its sequential placement. In J. M. Atkinson and J. C. Heritage (eds.), *Structures of Social Action: Studies in Conversation Analysis*. Cambridge University Press, Cambridge, pp. 299–345. DOI: 10.1017/cbo9780511665868 134

G. Jefferson and J. R. E. Lee. 1992. The rejection of advice: Managing the problematic convergence of a "troubles-telling" and a "service encounter." In P. Drew and J. Heritage (eds.), *Talk at Work: Interaction in institutional settings*. Cambridge University Press, Cambridge, pp. 521–548. 30, 40, 189

N. Kalchbrenner and P. Blunsom. 2013. Recurrent continuous translation models. In *Empirical Methods in Natural Language Processing*, p. 623. 45, 218

S. C. Levinson. 2007. Optimizing person reference—perspectives from usage on Rossel Island. In N. Enfield and T. Stivers (eds.), *Person reference in interaction: Linguistic, cultural, and social perspectives*. Cambridge University Press, Cambridge, pp. 29–72. DOI: 10.1017/cbo9780511486746.004 34, 35, 69, 70

P. Luff, N. Gilbert, and D. Frohlich (eds.) 1990. *Computers and Conversation*. Academic Press, London. DOI: 10.1016/c2009-0-21641-2 64, 211, 212

J. Manzo. 1997. Ethnomethodology, conversation analysis, and the sociology of law. In: M. Travers and J. F. Manzo (eds.), *Law in Action: Ethnomethodological and Conversation Analytic Approaches to Law*. Dartmouth Publishing Co., Aldershot, UK, pp. 1–15. 189

R. A. Martin, P. Puhlik-Doris, G. Larsen, J. Gray, and K. Weir. 2003. Individual differences in uses of humor and their relation to psychological well-being: Development of the Humor Styles Questionnaire. *Journal of Research in Personality*, 37, pp. 48–75. DOI: 10.1016/S0092-6566(02)00534-2 198

D. W. Maynard. 2003. *Bad News, Good News: Conversational Order in Everyday Talk and Clinical Settings*. University of Chicago Press, Chicago. DOI: 10.1086/422074 30, 181, 189

A. McHoul. 1990. The organization of repair in classroom talk. *Language in Society*, 19, pp. 349–377. DOI: 10.1017/s004740450001455x 41, 189

W. J. Mitchell, C.-C. Ho, H. Patel, and K. MacDorman. 2011. Does social desirability bias favor humans? Explicit–implicit evaluations of synthesized speech support a new HCI model of impression management. *Computers in Human Behavior*, 27, pp. 402–412. DOI: 10.1016/j.chb.2010.09.002 197

R. J. Moore. 2008. When names fail: Referential practice in face-to-face service encounters. *Language in Society*, 37(3), pp. 385–413. DOI: 10.1017/s004740450808055x 34, 40, 69, 70, 189

R. J. Moore. 2015. Automated transcription and conversation analysis. *Research on Language and Social Interaction*, 48(3), pp. 253–270. DOI: 10.1080/08351813.2015.1058600 47

R. J. Moore. 2018. A natural conversation framework for conversational UX design. In R. J. Moore, M. H. Szymanski, R. Arar, and G. J. Ren (eds.), *Studies in Conversational UX Design*. Springer International Publishing, Cham, pp. 181–204. DOI: 10.1007/978-3-319-95579-7 64, 65, 67, 70, 76, 85, 89, 93, 122, 212

R. J. Moore and R. Arar. 2018. Conversational UX design: An introduction. In R. J. Moore, M. H. Szymanski, R. Arar, and G. J. Ren (eds.), *Studies in Conversational UX Design*. Springer International Publishing, Cham. DOI: 10.1007/978-3-319-95579-7 1, 7, 15

R. J. Moore and E. F. Churchill. 2011. Computer interaction analysis: Toward an empirical approach to understanding user practice and eye gaze in GUI-based interaction. *Computer Supported Cooperative Work*, 20, pp. 497–528. DOI: 10.1007/s10606-011-9142-2 4

R. J. Moore, N. Ducheneaut, and E. Nickell. 2007. Doing virtually nothing: Awareness and accountability in massively multiplayer online worlds. *Computer Supported Cooperative Work*, 16, pp. 265–305. DOI: 10.1007/s10606-006-9021-4 5, 211

R. J. Moore, R. A. Hosn, and A. Arora. 2016. The machinery of natural conversation and the design of conversational machines. *American Sociological Association Annual Meeting*. Seattle (August). DOI: 10.1145/3027063.302707 64, 67, 76, 85, 89, 122, 212

I. Myers. 1962. *Manual: The Myers-Briggs Type Indicator*. Palo Alto, CA: Consulting Psychologists Press. DOI: 10.1037/14404-000 197

M. Nevile. 2002. Coordinating talk and non-talk activity in the airline cockpit. *Australian Review of Applied Linguistics*, 25, pp. 131–146. DOI: 10.1075/aral.25.1.07nev 189

D. A. Norman. 1988. *The Design of Everyday Things*. Basic Books, New York. xiv, 33, 63, 80, 161, 183

A. Peräkylä. 2008. Conversation analysis and psychoanalysis: Interpretation, affect and intersubjectivity. In A. Peräkylä, C. Antaki, S. Vehviläinen, and I. Leudar (eds.), *Conversation Analysis and Psychotherapy*. Cambridge University Press, Cambridge, pp. 100–119. DOI: 10.1017/CBO9780511490002.007 189

A. Pilnick, J. Hindmarsh, and V. T. Gill (eds.), 2010. *Communication in Healthcare Settings: Participation, P olicy and New Technologies*. Wiley-Blackwell, Chichester, U.K., pp. 1–16 DOI: 10.1002/9781444324020 189

H. Plattner. 2018. An Introduction to Design Thinking PROCESS GUIDE. Institute of Design at Stanford. https://dschool-old.stanford.edu/sandbox/groups/designresources/wiki/36873/attachments/74b3d/ModeGuideBOOTCAMP2010L.pdf. 183

H. Plattner, C. Meinel, and L. Leifer. 2010. *Design Thinking: Understand—Improve—Apply*. Springer, Berlin. 183

A. Pomerantz. 1984. Agreeing and disagreeing with assessments: Some features of preferred/dispreferred turn shapes. In J. M. Atkinson and J. C. Heritage (eds.), *Structures of Social Action: Studies in Conversation Analysis.* Cambridge University Press, Cambridge, pp. 57–101. DOI: 10.1017/cbo9780511665868 27, 28, 30, 32

M. Porcheron, J. E. Fischer, S. Reeves, and S. Sharples. 2018. Voice interfaces in everyday life. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM, New York, p. 640. DOI: 10.1145/3173574.3174214 27

J. Ruusuvuori. 2013. Emotion, affect and conversation. In J. Sidnell and T. Stivers (eds.), *The Handbook of Conversation Analysis*. Wiley-Blackwell, Malden, MA, pp. 330–349. DOI: 10.1002/9781118325001.ch16 179

H. Sacks. 1972. On the analyzability of stories by children. In J. J. Gumperz and D. Hymes (eds.), *Directions in Sociolinguistics: The Ethnography of Communication.* Holt, Reinhart and Winston, New York, pp. 325–345. 30

H. Sacks. 1984. Notes on methodology. In J. M. Atkinson and J. C. Heritage (eds.), *Structures of Social Action: Studies in Conversation Analysis.* Cambridge University Press, Cambridge, pp. 21–27. DOI: 10.1017/cbo9780511665868 4, 25, 188

H. Sacks. 1992. *Lectures on Conversation*, Vols. I and II. Blackwell, Cambridge, MA. DOI: 10.1002/9781444328301 129

H. Sacks and E. A. Schegloff. 1979. Two preferences in the organization of reference to persons in conversation and their interaction. In G. Psathas (ed.), *Everyday language: Studies in Ethnomethodology*, Irvington, New York, pp. 15–21. DOI: 10.2307/2066919 32, 34, 35, 38, 69, 70, 101

H. Sacks, E. A. Schegloff, and G. Jefferson. 1974. A simplest systematics for the organization of turn-taking for conversation. *Language*, 50, pp. 696–735. DOI: 10.2307/412243 xiv, 4, 26, 27, 29, 32, 39, 64, 65, 88, 192

E. A. Schegloff. 1968. Sequencing in conversational openings. *American Anthropologist*, 70, pp. 1075–1095. DOI: 10.1525/aa.1968.70.6.02a00030 150, 151, 156, 188

E. A. Schegloff. 1982. Discourse as an interactional achievement: Some uses of 'uh huh' and other things that come between sentences. In D. Tannen (ed.), *Analysing Discourse: Text and Talk.* Georgetown University Press, Washington, DC, pp. 71–93. DOI: 10.1515/text.2006.016 65, 74, 103, 113, 125

E. A. Schegloff. 1992a. In another context. In A. Duranti and C. Goodwin (eds.), *Rethinking Context: Language As An Interactive Phenomenon.* Cambridge University Press, Cambridge. 46, 55

E. A. Schegloff. 1992b. Repair after next turn: The last structurally provided defense of intersubjectivity in conversation. *American Journal of Sociology* 98, pp. 1295–1345. DOI: 10.1086/229903 20, 23, 36, 38, 69, 122, 138, 139, 141, 147

E. A. Schegloff. 1996. Some practices of referring to persons in talk-in-interaction: A partial sketch of a systematics. In: B. Fox (ed.), *Studies in Anaphora.* Benjamins, Amsterdam, pp. 437–485. DOI: doi.org/10.1075/tsl.33.14sch 101

E. A. Schegloff. 2007. *Sequence Organization in Interaction: A Primer in Conversation Analysis*, vol 1. Cambridge University Press, Cambridge. DOI: 10.1017/cbo9780511791208 4, 13, 30, 32, 39, 64, 65, 66, 88, 95, 96, 103, 122, 130, 142, 145, 213, 216

E. A. Schegloff and H. Sacks. 1973. Opening up closings, *Semiotica*, 7, pp. 289–327. DOI: 10.1515/semi.1973.8.4.289 30, 32, 39, 65, 82, 88, 165, 166, 167, 213

E. A. Schegloff, G. Jefferson, and H. Sacks. 1977. The preference for self-correction in the organization of repair in conversation, *Language*, 53, pp. 361–382. DOI: 10.2307/413107 2, 20, 23, 32, 36, 41, 64, 67, 122, 128, 130, 137

A. Schlesinger, K. P. O'Hara, A. S. Taylor, . . . 2018. Let's talk about race: Identity, chatbots, and AI. *CHI 2018*, April 21–26, Montreal, Canada. DOI: 10.1145/10.1145/3173574.3173889 179

S. Shellenbarger. 2018. Alexa: Don't let my 2-year-old talk to you that way. *Wall Street Journal*. July 11. 178

B. Shneiderman. 1982. The future of interactive systems and the emergence of direct manipulation. *Behaviour & Information Technology*, 1(3): pp. 237–256. DOI: 10.1080/01449298208914450 4, 5, 12

J. Sidnell and T. Stivers (eds.) 2013. *The Handbook of Conversation Analysis*. Wiley-Blackwell, Malden, MA. DOI: 10.1002/9781118325001 4

Y. Song, R. Yan, X. Li, D. Zhao, and M. Zhang. 2016. Two are better than one: An ensemble of retrieval- and generation-based dialog systems. *arXiv:*1610.07149v1 [cs.CL] 45, 47, 218

I. Sutskever, O. Vinyals, and Q. V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*, 2014. 45, 218

M. H. Szymanski. 1999. Re-engaging and dis-engaging talk in activity. *Language in Society* 28(1). DOI: 10.1017/s0047404599001013 189

M. H. Szymanski and R. J. Moore. 2018. Adapting to customer initiative: Insights from human service encounters. In R. J. Moore, M. H. Szymanski, R. Arar, and G. J. Ren (eds.), *Studies in Conversational UX Design*. Springer International Publishing, Cham. DOI: 10.1007/978-3-319-95579-7 40, 41, 158, 189

P. Ten Have. 1999. *Doing Conversation Analysis: A Practical Guide*. Sage, London. DOI: 10.4135/9781849208895 192

A. M. Turing. 1950. Computing machinery and intelligence. *Mind*, 59, pp. 433–460. DOI: 10.1093/mind/lix.236.433 21, 22, 211

O. Vinyals and Q. V. Le. 2015. A neural conversational model. *ICML Deep Learning Workshop*. 45, 46, 218

H. Walk. 2016. Amazon Echo is magical. It's also turning my kid into an asshole. https://hunterwalk.com/2016/04/06/amazon-echo-is-magical-its-also-turning-my-kid-into-an-asshole/. 178

J. Weizenbaum. January 1966. ELIZA—A computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1). DOI: 10.1145/365153.365168 2, 3, 18, 20, 23, 42

J. Whalen and E. Vinkhuyzen. 2000. Expert systems in (inter)action: Diagnosing document machine problems over the telephone. In P. Luff, J. Hindmarsh, and C. Heath (eds.), *Workplace Studies: Recovering Work Practice and Informing System Design*. Cambridge University Press, Cambridge, pp. 92–140. DOI: 10.1017/cbo9780511628122.006 40, 189

M. Whalen and D. H. Zimmerman. 1987. Sequential and institutional contexts in calls for help. *Social Psychology Quarterly*, 50(2), pp. 172–185. DOI: 10.2307/2786750 40, 41, 154, 189

L. Wittgenstein. 1922. *Tractatus Logico-Philosophicus.* New York: Harcourt, Brace & Company, Inc. 17

L. Wittgenstein. 1953. *Philosophical Investigations*. Macmillan, New York. 18, 21, 211

L. Wroblewski. 2011. *Mobile First*. A Book Apart, New York. 15

R. Wooffitt, N. M. Fraser, N. Gilbert, and S. McGlashan. 1997. *Humans, Computers and Wizards: Analysing human (simulated) computer interaction*. Routledge, London. 64

Index

Acknowledgment activities
  sample, 31
  sequence closers, 142–143
Actions
  Conversation-Centric interaction style, 14
  intents, 49
  Speech Act Theory, 30–31
Activities
  common patterns. *See* Conversational Activity UX patterns
  NCF, 76–79
  from sequences, 30
Adjacency pairs
  closing, 66
  Conversation Analysis, 32, 39
  conversation metric, 213
  conversations, 30, 32
  description, 65
  interaction model, 65
  visual-centric interaction style, 13
Advice-giving, 30, 42–43
Affordances in conversational interfaces, 184–186
Agent Continuer pattern, 126
Agent Elicitation example, 68
Agent-initiated elicitation, 67
Agent persona example, 194–195
  job description, 195–196
  personality, 196–199
  self-knowledge, 199–200
Agent utterances, limiting, 70
Agreements, 28
AI (artificial intelligence)
  conversation authoring, 45
  machine understanding, 17, 22
Airplane cockpits, 189
Alexa platform, 3
  IECR paradigm, 47
  voice-only interaction, 185
Alexander, Christopher, 87
Alma implementation
  computer trivia, 78–79
  Conversation About Nothing example, 28–29
  joke telling, 78
  NCF, 66
  organization appreciation, 82
  sample conversation, 263–273
Alternative Responses to “ok” examples, 55–56
Anaphora, 53
Anatomy of conversations
  conversation design, 32
  example, 31
  minimization, 34–36
  overview, 29–32
  recipient design, 32–34
  repair, 36–38
Anti–intents in Open Request pattern, 95
Api.ai platform, 47
Apologies
  Offense Complaint (Agent), 178
  sequence management, 134, 140

Appreciation Receipt example, 54
Appreciations
  Closing pattern, 169
  Last Topic Check (Agent), 167
  Offense Complaint (Agent), 178
  Sequence Closers, 142–144
Artificial intelligence (AI)
  conversation authoring, 45
  machine understanding, 17, 22
Assessments
  Conversation Analysis, 28, 30
  Last Topic Check (Agent), 167
  Sequence Closers, 142
Assign default command, 206
Assistant agent, 3
Assumptions
  definition requests, 134
  recipient design, 33
  shared understanding, 20
Atomic propositions, language as, 17
Automated agents, listening by, 2–3

Basic mechanics of natural conversation, 38
Beta testing, 208
Bidding farewell, 168
Brainstorming, 200–203
Branching, 48, 59–61
Brand mascots
  conversational interfaces, 186
  self-knowledge for, 199–200
Build stage for prototypes, 207–208

CA. *See* Conversation Analysis (CA)
Capabilities pattern
  Capability Expansion, 163–164
  General Capability Check, 162–163
  overview, 161–162
  Specific Capability Check, 164
  Utterance Suggestion, 164–165
Capability checks in conversation navigation, 80
Capability description in Sequence Abort pattern, 146
Chatbots
  ELIZA, 2–3
  limitations, 4
  minimization, 36
  Visual-Centric interaction style, 12–13
Chit chat, 121–122
Choices Request example, 136
Classroom settings
  Conversation Analysis, 41–42
  conversation studies, 189
Closing patterns
  Closing Appreciation (Organizational), 169
  Closing Farewell (User), 168–169
  Closing Name Request (Agent), 170
  Closing Offer (Affirmed), 172–173
  Closing Offer (Disaffirmed), 173
  Closing Success Check (Affirmed), 170–171
  Closing Success Check (Disaffirmed), 171–172
  Closing Success Check Reopened, 172
  Last Topic Check (Agent), 167
  Last Topic Check (User), 167–168
  overview, 165–166
  Pre-Closing (User), 168
Closings
  conversation anatomy, 30
  Conversation-Centric interaction style, 14
  conversation navigation, 81–82
  interaction model, 66–67
  patterns. *See* Closing patterns; Sequence Closers patterns
Cognitive chatbots, understanding by, 17
Combining Conversational Activity UX patterns, 114–119
Commands in dialog design, 206
Commissives activities in Conversation Analysis, 31
Common conversational activities, 15
Complaints, 178
Complex propositions, language as, 17
Compliments, 178

Computer Trivia example, 78–79
Conditions, 49, 55–56
Constantive activities, 30
Constructional units in finite-state machine, 49
Content-Centric interaction styles
  characteristics, 16
  minimization, 36
  overview, 10–12
Contents
  human understanding, 17
  NCF, 69–76
Context
  dialog design, 206
  overview, 52–55
  IECR paradigm. *See* Intent-Entity-Context-Response (IECR) paradigm
  System-Centric interaction style, 8, 10
Context design, 55
Context variables, 55
Conversation
  conversation design, 5–7
  conversational agents, 3–5
  with ELIZA, 1–3
  interaction styles, 7–16
  metrics, 212–217
  types, 39–43
  understanding in, 17–23
Conversation About Nothing example, 28–29
Conversation Analysis (CA)
  benefits, 1
  conclusion, 43
  conversation anatomy, 29–38
  conversation description, 26–27
  conversation types, 39–43
  description, 4
  literature, 189
  overview, 25–26
  pattern language, 88
  topic vs. structure, 27–29
  transcription conventions, 219
Conversation authoring
  branching, 59–61
  conclusion, 62
  finite-state machine. *See* Finite-state machine
  overview, 45–47
Conversation-Centric interaction styles
  characteristics, 16
  overview, 14–16
Conversation First, 7–16, 64, 86, 211–212
Conversation games, 5, 88, 208, 212
  Conversation Analysis patterns, 88
  Understanding Check, 138
Conversation-level management in NCF, 225–226
Conversation-Like Response(s) example, 71
Conversation management
  Conversation Analysis, 28–29
  Conversation-Centric interaction style, 14
  interaction model, 67
  patterns, 89
Conversation Management UX patterns
  Capabilities, 161–165
  Closing, 165–174
  conclusion, 180–181
  Disengaging, 174–180
  Opening (Agent), 149–155
  Opening (User), 155–161
  overview, 149
Conversation navigation
  capability checks, 80
  closing, 82–85
  overview, 79–80
  paraphrase requests, 80–81
  repeat requests, 80
  sequence aborts, 81
  sequence closers, 81
Conversation Navigation examples, 82–84
Conversation space in finite-state machine, 49
Conversational Activity UX patterns
  Agent Inquiry, 109–114
  combining, 114–119
  conclusion, 119
  Extended Telling, 103–106

Conversational Activity UX patterns *(continued)*
  Inquiry (User), 90–93
  NCF, 76–79
  Open Request, 93–103
  overview, 87–90
  Quiz, 106–109
  summary, 223–241
  troubleshooting, 114–116
  tutoring, 116–119
Conversational agents, current, 3–5
Conversational competence
  NCF, 76
  Sequence Management UX patterns, 121
Conversational interfaces
  affordances, 184–186
  vs. graphical user interfaces, 3–4
Conversational UX design
  approach, 24
  content design, 76
  context design, 53–55
  demand for, 64
  heart of, 209
  overview, 5–7
Conversational UX Design process
  affordances, 184–186
  conclusion, 209
  define stage, 190–192
  design thinking, 183–186
  empathize stage, 187–190
  ideate stage, 192–202
  prototype stage, 203–208
  test stage, 208
Cortana agent, 3
Counseling conversations, 42–43
Courtrooms, 189
Create condition command, 206
Customer service calls, recordings, 188
Customer service representatives
  agent persona example, 195
  Conversation Analysis, 40
  machine understanding, 22

Data sessions in ideate stage, 192–194
Deep Blue computer, 21
Define stage
  design thinking, 190–192
  outcome statements, 191–192
  user personas, 190–191
Definition Request patterns
  Definition Request Default, 134–135
  Definition Request Repair, 134
  Definition Request Standalone, 135
  overview, 133–134
Design thinking
  empathize stage, 187–190
  ideate stage, 192–202
  overview, 183–186
  prototype stage, 203–208
Design Thinking model, 183–184
Detail
  Open Request pattern, 94
  recipient design, 32–33
  user control of, 70
Deviant cases, 188
Diabetes Coverage Extended Telling example, 75–76
Diabetes Coverage Inquiry example, 75
Dialog design, 206, 221
Dialog services in NCF, 64
Direct manipulation in Visual-Centric interaction style, 12
Direct observation in design thinking, 188
Directive activities, 31
Disagreements, 28
Disengaging patterns
  Offense Complaint (Agent), 177–179
  Offense Complaint (User), 179–180
  overview, 174
  Recipient Correction, 176–177
  Transfer Aborted, 175
  Transfer Successful, 174–175
Displaced Greeting (User) pattern, 157
Displaced Welfare Check (User) pattern, 157–158
Dispreferred actions, 28
Document-Like Response example, 71
Doing, understanding as, 211

Dynamic knowledge, 58

Efficiency
  basic mechanics, 38
  interactional, 213–215
  minimization, 35–36
Elaborated Reference example, 35
Eliciting, sequence expansions for, 66–67
ELIZA program
  machine understanding, 23
  overview, 2–3
else command, 206
Emotions, 178–179
Empathize stage
  design thinking, 187–190
  engage, 189–190
  observation, 187–189
Engagement in design thinking, 189–190
Entities
  dialog design, 206
  IECR paradigm, 47
  overview, 52
Ethnographic interviewing, 189
Ethnomethodology, 53
Example Request pattern
  Example Request Default, 136–137
  overview, 135–136
Example requests by ELIZA, 2
Expandable sequences
  conversation metric, 213–214
  design thinking, 203–204
  overview, 67
  What's in Theatres, 85
Expansion in interaction model, 65–68
Expert example, 33–34
Extended Repair patterns
  Other-Correction, 141
  Self-Correction, 138–139
Extended Telling patterns, 70
  example, 73–74
  Extended Telling Abort, 105–106
  Extended Telling with Repair, 103–105
Extroversion as Myers-Briggs Type indicator, 198

Failed Reference example, 35
Falling Off a Branch example, 60
FAQ (Frequently Asked Question) pages, 11
Feeling as Myers-Briggs Type indicator, 198
Fictional characters
  conversational interfaces, 186
  self-knowledge for, 199–200
Finite-state machine47–49
First-position repair, 37–38
Fixed roles in Conversation Analysis, 40
Four Positions of Repair example, 36–37
Fourth position repair, 38
Frequently Asked Question (FAQ) pages, 11
Functional conception of understanding, 20

Gaps, 27
Garfinkel, Harold, 19
Gender in agent persona example, 196–197
General AI in machine understanding, 22
General Capability Check pattern, 162–163
Goals in personas, 191–192
goto command, 206
Graphic design for web pages, 6
Graphical user interfaces (GUIs) vs. conversational interfaces, 3–4
Greeting (User) pattern, 155–156

Hearing check in Quiz pattern, 108
Hearing Check patterns
  Hearing Check Confirmed, 130–131
  Hearing Check Corrected, 131
  overview, 130
House for a Small Family pattern, 87
Human conversation, self-knowledge for, 199–200
Human understanding perspectives, 17–20
Humanoid characters for conversational interfaces, 186
Humor Styles Questionnaire, 198–199

IBM
  Deep Blue and Watson computers, 21
  homepage, 5–7

Ideate stage
  agent persona example, 194–200
  design thinking, 192–202
  shape of conversation example, 192–194
  utterance pair brainstorming, 200–203
Identification requests, 152
IECR paradigm. *See* Intent-Entity-Context-Response (IECR) paradigm
if command, 206
In-scope utterance pairs, 202–203
Inconsequential talk, 121–122
Indexical expressions, 53
Information retrieval
  conversation authoring, 45, 47
  System-Centric interaction style, 9–10
Informings in Conversation Analysis, 30
Input
  Conversation-Centric interaction style, 15
  conversational agents, 3
  System-Centric interaction style, 8
  Visual-Centric interaction style, 12–13
Inquiry (Agent) patterns
  Inquiry (Agent), 110
  Inquiry (Agent) Completion Check, 113
  Inquiry (Agent) Extended Answer, 113–114
  Inquiry (Agent) Hearing Confirmed, 111
  Inquiry (Agent) Hearing Disconfirmed, 111–112
  Inquiry (Agent) Open, 110–111
  overview, 109–110
Inquiry (User) patterns
  Inquiry (User) Confirmation, 91
  Inquiry (User) Disconfirmation, 91–92
  Inquiry (User) Repairs, 92–93
  overview, 90–91
Insults, 178
Intent-Entity-Context-Response (IECR) paradigm
  branching, 59–61
  finite-state machine, 47–58
  introduction, 24
  platforms using, 47
  pseudocode, 205
Intents
  Conversation Analysis, 31
  dialog design, 206
  finite-state machine, 49–52
  IECR paradigm, 47
  interaction model, 67
  Open Request pattern, 95, 101
  overview, 49–52
Interaction model in NCF, 65–69
Interaction styles
  Content-Centric, 10–12
  Conversation-Centric, 14–16
  overview, 7
  System-Centric, 8–10
  Visual-Centric, 12–13
Interactional Efficiency metric, 213–215
Interactive Voice Response (IVR), 12
Internet of Things (IoT) use cases, 185
Intersubjectivity in shared understanding, 20
Interviews
  agent inquiry, 109
  in engagement, 189–190
  psychiatric, 42
Introversion as Myers-Briggs Type indicator, 198
Intuition as Myers-Briggs Type indicator, 198
Invitation-acceptance sequences, 39
Invitations in Conversation Analysis, 30
IoT (Internet of Things) use cases, 185
IVR (Interactive Voice Response), 12

Jargon, 33
Job description for agent persona example, 195–196
Joke Telling example, 78–79
Joke telling in System-Centric interaction style, 9
Judging as Myers-Briggs Type indicator, 198

Keywords
  ELIZA program, 3

  entities, 52
  IECR paradigm, 47
Knock-knock jokes, 9
Knowing your user in recipient design, 33

Labeled Conversation Log example, 216
Language games, 18–19
Last Topic Check (Agent) pattern, 167
Last Topic Check (User) pattern, 167–168
Laughter tokens
  Last Topic Check (Agent), 167
  sequence closers, 142
Leaf nodes, 48
Linguistic patterns, 50
Listening by ELIZA, 2–3
Local help in paraphrase requests, 80–81
Luis platform, 47

Machine learning (ML)
  conversation authoring, 45
  responses, 56–57
  IECR paradigm, 47
Machine understanding, 21–23
Mascots
  conversational interfaces, 186
  self-knowledge for, 199–200
Medicine, conversation studies, 189
Mental states in human understanding, 17
Metrics, conversation, 212–217
Minimal Expansion example, 65
Minimal Reference example, 35
Minimization, 34–36
Minimum viable product (MVP), 191
Misunderstanding Report pattern, 139–140
Mixed conditions in finite-state machine, 55–56
Mixed-initiative style, 15
Mobile-first strategy in web design, 15
Mock-ups for prototype stage, 204–205
Molecular propositions, language as, 17
Motivations in personas, 190–191
Movie critic agent persona example, 195
Multi-Turn Requests example, 53–54
Multiparty interactions in Conversation Analysis, 27
Multiple User Entry Points example, 59
Multiple-Variation Response example, 57–58
Must-have goals in design thinking, 191
Mutual understanding
  as goal, 23
  interaction model, 69
  picture theory, 18
  in sequence completion rate, 213
Myers-Briggs Type indicators, 197–198

Name Correction (User) pattern, 160–161
Name Request (User) pattern, 161
Natural Conversation Framework (NCF)
  conclusion, 85–86
  content format, 69–76
  conversation-level management examples, 251–263
  conversation-level management patterns, 225–226
  conversation navigation, 79–85
  conversational activities examples, 226–241
  conversational activities patterns, 76–79, 223–224
  expandable sequences, 203–204
  interaction model, 65–69
  overview, 63–65
  sequence-level management examples, 241–250
  sequence-level management patterns, 224–225
Natural conversation understanding in interaction model, 69
Natural conversation, 4
  aspects of, 32
  basic mechanics of, 38
  common action types, 30
  complex system, 64
  complexity of, 4
  efficiency in, 35–36
  form of natural language use, 29

Natural conversation *(continued)*
  key characteristic of, 10
  mischaracterization, 192
  rules and models of, 88
Natural language
  Content-Centric interaction style, 10
  Conversation-Centric interaction style, 14–15
  System-Centric interaction style, 8–9
  type summary, 16
  understanding, 17
Natural language classes (NLC)
  chatbot use of, 7
  conversation authoring, 45
  conditions, 49–52
Natural-language-generation approach in conversation authoring, 45–47
Natural language processing (NLP)
  advances in, 64
  interaction styles, 7
  limitations, 4–5
Natural language understanding (NLU) techniques, 22–23
Navigation, conversation. *See* Conversation navigation
NCF. *See* Natural Conversation Framework (NCF)
Negative assessments, 142
News Check Capability (User) pattern,159
News Check (User) pattern, 159
News checks, 150
News interviews, 189
NLC (natural language classes)
  chatbot use of, 7
  conversation authoring, 45
  conditions, 49–52
NLP (natural language processing)
  advances in, 64
  interaction styles, 7
  limitations, 4–5
NLU (natural language understanding) techniques, 22–23
No Answer Account pattern, 126–127
Non-pedagogical inquiries example, 18
Nonlinear interactions in Conversation-Centric interaction style, 15
Norm breaching example, 19
Noticings, 192
Novice example, 34
Number of sequences metric, 213–215

Observation in design thinking, 187–189
Off-topic utterance pairs, 202–203
Offense Complaint (Agent) pattern, 177–179
Offense Complaint (User) pattern, 179–180
On-topic utterance pairs, 202
Open Request patterns
  Open Request Agent Detail Request, 96
  Open Request Continuer, 94–95
  Open Request Incremental, 100
  Open Request Nonverbal, 94
  Open Request Repairs, 102–103
  Open Request Screening, 95–96
  Open Request Series, 100–101
  Open Request Summary, 97–98
  Open Request Summary with Artifacts, 99–100
  Open Request User Detail Request, 96–97
  overview, 93–94
  Warrant Request & Refusal, 98–99
Opening (Agent) patterns
  Opening Authentication (Agent), 154
  Opening Direct Address (Agent), 152
  Opening Greeting (Agent), 151
  Opening Name Request (Agent), 152
  Opening Organization Offer of Help (Agent), 153–154
  Opening Self-Identification (Agent), 151–152
  Opening Welfare Check (Agent), 153
  Organizational Problem Request (Agent), 154–155
  overview, 149–151
Opening (User) patterns
  Displaced Greeting (User), 157
  Displaced Welfare Check (User), 157–158
  Greeting (User), 155–156
  Name Correction (User), 160–161

  Name Request (User), 161
  News Check Capability (User), 159
  News Check (User), 159
  overview, 155
  Reciprocated Welfare Check (User), 159
  Self-Identification (User), 160
  Summons (User), 156
  Welfare Check (User), 156–157
Openings in conversation, 30
Oral examination in shared understanding, 18
Ordinary conversations, 39
Organizational representatives in Conversation Analysis, 40
Other-Correction pattern, 141
Other Hearing Check & Mishearing Report pattern, 140–141
Out-of-scope utterance pairs, 202–203
Outcome statements in design thinking, 191–192

Pain points in personas, 190–191
Paragraphs, breaking down, 70–72
Paraphrase Request (Agent) patterns
  Agent Continuer, 126
  No Answer Account, 126–127
  overview, 124–125
  Repeat Acknowledgment, 127–128
  Subsequent Paraphrase Requests, 125–126
Paraphrase Request patterns
  overview, 131–133
  Paraphrase Request Default, 133
Paraphrase requests
  Conversation-Centric interaction style, 14
  conversation navigation, 80–81
  by ELIZA, 2
Paraphrasing, sequence expansions for, 66–67
Partial Repeat Request pattern, 130
Partial repeats by ELIZA, 3
Paths in finite-state machine, 49
Patterns overview
  Conversation Analysis, 25
  conversational activity, 76–79
  dialog design, 206
  NCF, 63–65
Pauses, 27–28
Pedagogical inquiries example, 18
Perceiving as Myers-Briggs Type indicator, 198
Persistent context
  conversation authoring, 54
  Conversation-Centric interaction style, 15
  interaction model, 67
  System-Centric interaction style, 10
Personality in agent persona example, 196–199
Personas
  agent persona example, 194–200
  design thinking, 190–191
Phrases
  condition, 49–52
  IECR paradigm, 47
Picture theory of meaning, 17–18
Politics, agent persona example, 197
Positive assessments, 142
Pre-closing in conversation navigation, 82
Pre-Closing (User) pattern, 168
Preference organization in Conversation Analysis, 28, 32
Preferred actions in Conversation Analysis, 28
Preliminaries
  Conversation Analysis, 39
  Conversation-Centric interaction style, 14
  Open Request pattern, 95
Previous nodes
  branching, 60–61
  mixed conditions, 56
  Quiz Repairs pattern, 108
Programming by example for finite-state machine, 50
Pronouns with ELIZA, 2–3

Propositions, language as, 17
Prototype stage
  build, 207–208
  design, 205–207
  mock-up, 204–205
  overview, 203–208
Pseudocode
  dialog design, 221
  prototype stage, 205–207
Psychotherapy
  Conversation Analysis, 42–43
  conversation studies, 189
  ELIZA program, 2

Q&A example, 11
Quiz patterns
  Computer Trivia example, 78–79
  overview, 106
  Quiz Agent-Initiated, 109
  Quiz Incorrect, 107
  Quiz Repairs, 107–108
  Quiz User-Initiated, 108–109

Recipient Correction pattern, 176–177
Recipient design in Conversation Analysis, 32–34
Reciprocated Welfare Check (User) pattern, 159
Recognitionals in minimization, 35
Recording social settings, 188
Reference forms in recipient design, 33
Refutation example, 54–55
Repair
  Conversation Analysis, 32, 36–38
  interaction model, 67
  machine understanding, 22
  Open Request Repairs pattern, 102–103
  Quiz Repairs, 107–108
  shared understanding, 20
Repair (Agent) patterns
  overview, 122–123
  Paraphrase Request (Agent), 124–128
  Understanding Check (Agent), 123–124
Repair (User) patterns
  Definition Request, 133–135
  Example Request, 135–137
  Hearing Check, 130–131
  overview, 128
  Paraphrase Request, 131–133
  Partial Repeat Request, 130
  Repeat Request, 128–130
  Understanding Check, 137–138
Repeat Acknowledgment pattern, 127–128
Repeat Request patterns
  overview, 128–129
  Repeat Request Aborted, 129–130
  Repeat Request Default, 129
Repeat requests
  Conversation-Centric interaction style, 14
  conversation navigation, 80
  Quiz pattern, 108
Repeating
  interaction model, 67
  sequence expansions for, 66–67
Requests in conversation anatomy, 30
Resources in Conversation Analysis patterns, 88
Respond to user command, 206
Response Frames example, 58
Responses
  Conversation-Centric interaction style, 14–15
  overview, 56–58
  IECR paradigm, 47
Rogerian therapy example, 1–2
Rogers, Carl, 2
Roles, 40
Root level in finite-state machine, 48
Route to node command, 206
Rule-based templates for ELIZA program, 3
Rules for Conversation Analysis patterns, 88

Sacks, Harvey, 25
say command, 206
Schegloff, Emanuel, 20

Screening, sequence expansions for, 66–67
Search engines vs. conversational agents, 4
Second-position repair, 37–38
Self-Correction patterns
  Misunderstanding Report, 139–140
  Other Hearing Check & Mishearing Report, 140–141
  overview, 138–139
Self-Identification (User) pattern, 160
Self-identifications, 151–152
Self-knowledge agent persona example, 199–200
Sensing as Myers-Briggs Type indicator, 198
Sequence Abort patterns
  overview, 145–146
  Sequence Abort Capability Offer Accepted, 146–147
  Sequence Abort Capability Offer Declined, 147
Sequence aborts in conversation navigation, 81
Sequence closers in conversation navigation, 81
Sequence Closers patterns
  overview, 142
  Sequence Closer Appreciation (helped), 143–144
  Sequence Closer Appreciation (not helped), 144
  Sequence Closer (helped), 142–143
  Sequence Closer (not helped), 143
  Sequence Closer (repaired), 145
Sequence Completion Rate metric, 213–215
Sequence Expansion Types example, 66
Sequence Management UX patterns
  conclusion, 147–148
  examples summary, 241–250
  Extended Repair, 138–141
  overview, 121–122
  patterns summary, 224–225
  Repair (Agent), 122–128
  Repair (User), 128–138
  Sequence Abort, 145–147
  sequence closers, 142–145
Sequence-to-sequence (seq2seq) models, 45–47
Sequences
  Conversation Analysis, 32
  conversation anatomy, 30
  description, 65
Sequential context
  Conversation-Centric interaction style, 15
  context, 53, 55
Service conversations
  Conversation Analysis, 40–41
  machine understanding, 22
Set variable command, 206
Shape of conversation example, 192–194
Shared understanding
  as goal, 23
  interaction model, 69
  picture theory, 18
  in sequence completion rate, 213
Simulacra, 88
Single conversation units in conversation anatomy, 30
Single User Entry Point examples, 61
Single-Version Response example, 57
Siri agent, 3
Slots
  interaction model, 67
  Open Request pattern, 96, 101
Small talk, 121–122
Social actions
  Conversation Analysis, 30, 39
  Conversation-Centric interaction style, 14
Social practices
  machine understanding, 21
  shared understanding, 18–19
Specific Capability Check pattern, 164
Speech Act Theory, 30–31
Speech acts in Conversation Analysis, 31
Speech-exchange system features, 26–27
Staying on a Branch example, 60

Storytelling
  Conversation Analysis, 30
  Extended Telling patterns, 103–106
  sequences, 65
Structure in Conversation Analysis, 27–29
Students
  Conversation Analysis, 41–42
  conversation studies, 189
  Tutoring pattern, 116–119
Subsequent Paraphrase Requests pattern, 125–126
Summons (User) pattern, 156
System-Centric interaction styles
  characteristics, 16
  information retrieval, 9–10
  overview, 8
  voice control, 8–9
System entities, 52

Talk-in–interaction in Conversation Analysis, 25
TCUs (turn constructional units), 29–30
Teaching conversations
  Conversation Analysis, 41–42
  conversation studies, 189
  Tutoring pattern, 116–119
Templates for ELIZA program, 3
Test stage in design process, 208
Text communication
  conversational agents, 3
  conversational interfaces, 185
Therapist conversations, 42
Thinking as Myers-Briggs Type indicator, 198
Third position repair, 37–38
Topics
  Open Request pattern, 93, 101
  vs. structure, 27–29
Transcripts
  benefits, 1
  conventions, 219
  mock-ups, 204–205
Transfer Aborted pattern, 175
Transfer Successful pattern, 174–175
Travel agents persona example, 195–196
Tree data structure in finite-state machine, 47–49
Troubleshooting patterns
  overview, 114–115
  Troubleshooting User-Initiated, 115–116
Turing, Alan, 21–22
Turing test, 21–22
Turn-taking
  Conversation Analysis, 32
  conversation shape, 192
  repair mechanisms, 27
Tutoring pattern, 116–119
Typologies, personality, 197

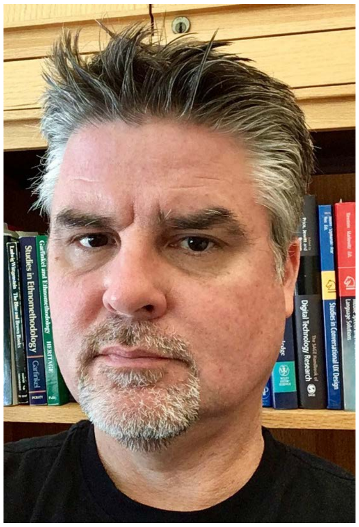
Understanding
  human, 17–20
  interaction model, 69
  machine, 21–23
  overview, 17, 211
  shared. *See* Shared understanding
Understanding Check (Agent) pattern, 123–124
Understanding Check patterns
  Understanding Check Confirmed, 137
  Understanding Check Disconfirmed, 137–138
Understanding Repair example, 20
Use cases in conversational interfaces, 184–186
User-centered design, 183
User Elaboration Request example, 68
User goals for personas, 191
User-initiated repairs in Understanding Check, 138
User input points in finite-state machine, 59–61
User personas in design thinking, 190–191
Utterance pairs, 200–203
Utterance Suggestion pattern, 164–165
Utterances
  Conversation Analysis, 30
  conversation shape, 192–194
  turn constructional units, 29–30

- Valid queries in System-Centric interaction style, 10
- Variables
  - context, 52–55
  - IECR paradigm, 47
- Variation management in intents, 50–51
- Vinyals and Le example, 46
- Visual-Centric interaction styles
  - characteristics, 16
  - overview, 12–13
- Viva voce testing
  - machine understanding, 22
  - overview, 18–19
- Voice assistants
  - limitations, 4
  - minimization, 36
- Voice-based interaction, 185
- Voice control in System-Centric interaction style, 8–9
- Voice input in conversational agents, 3
- Voices in agent persona example, 196–197

- Warrant Request & Refusal pattern, 98–99
- Watson agent, 3
- Watson Assistant platform
  - IECR paradigm, 47
  - NCF, 64
  - system entities, 52
- Watson computer, 21
- Watson Natural Language Understanding Service, 52
- Web designers, 5
- Welfare Check (User) pattern, 156–157
- Welfare checks
  - Opening (Agent), 153
  - shared understanding, 20
- Well-wishes, 168
- What’s in Theaters, 83–85
- Wit.ai platform, 47
- Wittgenstein, Ludwig, 18

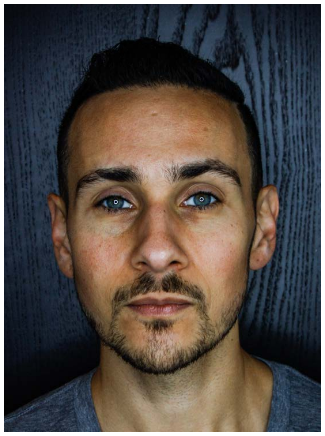
Author Biographies

Robert J. Moore



Robert J. Moore is a research scientist at IBM Research–Almaden, where he examines the intersection of human conversation and technology. He is currently applying conversation science to the design of user experience and conversational agents. In the past, Dr. Moore has worked as a scientist at Yahoo! Labs and the Xerox Palo Alto Research Center (PARC), and as a game designer at The Multiverse Network. He holds Ph.D., M.S., and B.A. degrees in sociology with concentrations in ethnomethodology, conversation analysis, and ethnography.

Raphael Arar



Raphael Arar is passionate about investigating interaction design for emerging technologies and complex systems. While currently a Principal Designer at Khan Academy, the research and writing for this publication was completed while he was designing for artificial intelligence systems at IBM Research–Almaden. Previously, he led the UX effort for the Apple + IBM partnership and taught in the Media Arts program at the University of Southern California. He holds an MFA from the California Institute of the Arts, a BA from Boston University, and in 2017, he was listed as one of Forbes's "30 under 30 in Enterprise Technology."

Conversational UX Design

A Practitioner's Guide to the Natural Conversation Framework

Robert J. Moore, Raphael Arar

With recent advances in natural language understanding techniques and far-field microphone arrays, natural language interfaces, such as voice assistants and chatbots, are emerging as a popular new way to interact with computers. They have made their way out of the industry research labs and into the pockets, desktops, cars and living rooms of the general public. But although such interfaces recognize bits of natural language, and even voice input, they generally lack conversational competence, or the ability to engage in natural conversation. Today's platforms provide sophisticated tools for analyzing language and retrieving knowledge, but they fail to provide adequate support for modeling interaction. The user experience (UX) designer or software developer must figure out how a human conversation is organized, usually relying on commonsense rather than on formal knowledge. Fortunately, practitioners can rely on conversation science.

This book adapts formal knowledge from the field of Conversation Analysis (CA) to the design of natural language interfaces. It outlines the Natural Conversation Framework (NCF), developed at IBM Research, a systematic framework for designing interfaces that work like natural conversation. The NCF consists of four main components: 1) an interaction model of "expandable sequences," 2) a corresponding content format, 3) a pattern language with 100 generic UX patterns and 4) a navigation method of six basic user actions. The authors introduce UX designers to a new way of thinking about user experience design in the context of conversational interfaces, including a new vocabulary, new principles and new interaction patterns. User experience designers and graduate students in the HCI field as well as developers and conversation analysis students should find this book of interest.

ABOUT ACM BOOKS

ACM Books is a series of high-quality books published by ACM for the computer science community. ACM Books publications are widely distributed in print and digital formats by major booksellers and are available to libraries and library consortia. Individual ACM members may access ACM Books publications via separate annual subscription.

ISBN 978-1-4503-6302-0
90000
9 781450 363020

BOOKS.ACM.ORG · WWW.MORGANCLAYPOOLPUBLISHERS.COM