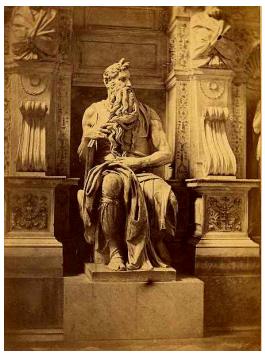
CHAPTER

24 Dialog Systems and Chatbots

> Les lois de la conversation sont en général de ne s'y appesantir sur aucun objet, mais de passer légèrement, sans effort et sans affectation, d'un sujet à un autre ; de savoir y parler de choses frivoles comme de choses sérieuses
>
> The rules of conversation are, in general, not to dwell on any one subject, but to pass lightly from one to another without effort and without affectation; to know how to speak about trivial topics as well as serious ones;
>
> The 18th C. *Encyclopedia* of Diderot, start of the entry on conversation

The literature of the fantastic abounds in inanimate objects magically endowed with sentience and the gift of speech. From Ovid's statue of Pygmalion to Mary Shelley's Frankenstein, there is something deeply moving about creating something and then having a chat with it. Legend has it that after finishing his sculpture *Moses*, Michelangelo thought it so lifelike that he tapped it on the knee and commanded it to speak. Perhaps this shouldn't be surprising. Language is the mark of humanity and sentience, and **conversation** or **dialog** is the most fundamental and specially privileged arena of language. It is the first kind of language we learn as children, and for most of us, it is the kind of language we most commonly indulge in, whether we are ordering curry for lunch or buying spinach, participating in business meetings or talking with our families, booking airline flights or complaining about the weather.



conversation
dialog

This chapter introduces the fundamental algorithms of **conversational agents**, or **dialog systems**. These programs communicate with users in natural language (text, speech, or even both), and generally fall into two classes.

conversational agent
dialog system

Task-oriented dialog agents are designed for a particular task and set up to have short conversations (from as little as a single interaction to perhaps half-a-dozen interactions) to get information from the user to help complete the task. These include the digital assistants that are now on every cellphone or on home controllers (Siri, Cortana, Alexa, Google Now/Home, etc.) whose dialog agents can give travel directions, control home appliances, find restaurants, or help make phone calls or send texts. Companies deploy goal-based conversational agents on their websites to help customers answer questions or address problems. Conversational agents play an important role as an interface to robots. And they even have applications for social good. DoNotPay is a "robot lawyer" that helps people challenge incorrect parking fines, apply for emergency housing, or claim asylum if they are refugees.

Chatbots are systems designed for extended conversations, set up to mimic the

unstructured conversational or 'chats' characteristic of human-human interaction, rather than focused on a particular task like booking plane flights. These systems often have an entertainment value, such as Microsoft's XiaoIce (Little Bing 小冰) system (Microsoft, 2014), which chats with people on text messaging platforms. Chatbots are also often attempts to pass various forms of the Turing test (introduced in Chapter 1). Yet starting from the very first system, ELIZA (Weizenbaum, 1966), chatbots have also been used for practical purposes, such as testing theories of psychological counseling.

Note that the word 'chatbot' is often used in the media and in industry as a synonym for conversational agent. In this chapter we will instead follow the usage in the natural language processing community, limiting the designation *chatbot* to this second subclass of systems designed for extended, casual conversation.

turn

Let's see some examples of dialog systems. One dimension of difference across systems is how many **turns** they can deal with. A dialog consists of multiple turns, each a single contribution to the dialog (the terminology is as if dialog is a game in which I take a turn, then you take a turn, then me, and so on). A turn can consist of a sentence, although it might be as short as a single word or as long as multiple sentences. The simplest such systems generally handle a single turn from the user, acting more like question-answering or command-and-control systems. This is especially common with digital assistants. For example Fig. 24.1 shows screen captures from an early version of Apple's Siri personal assistant from 2014, demonstrating this kind of single-query behavior.

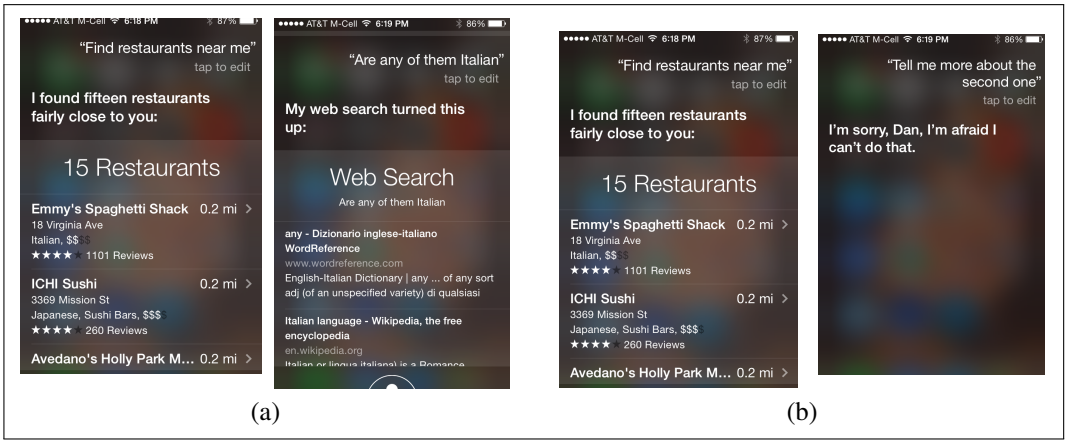


Figure 24.1 Two sets of interactions with Siri in 2014. (a) A question ("Find restaurants near me") returns restaurants, but the system was unable to interpret a follow-up question ("Are any of them Italian?"). (b) An alternative followup ("Tell me more about the second one") similarly fails. This early system's confusion at follow-up questions suggests that it is mainly designed for a single interaction.

By contrast, Fig. 24.2 shows that a 2017 version of the Siri digital assistant can handle slightly longer dialogs, handling a second turn with a follow-up question.

While spoken dialogs with mobile phone digital assistants tend to be short, some tasks do require longer dialogs. One such task is travel planning and management, a key concern of dialog systems since the very influential GUS system for planning airline travel (Bobrow et al., 1977); we'll see an example in the next section.

Dialog systems can even be used for much more complex domains like automatic tutoring. Figure 24.3 shows part of a dialog from the adaptive ITSPOKE dialog system (Forbes-Riley and Litman, 2011). In this example the system detects the hesitancy of the student's first response ("Is it 19.6 m/s?"), and, even though the

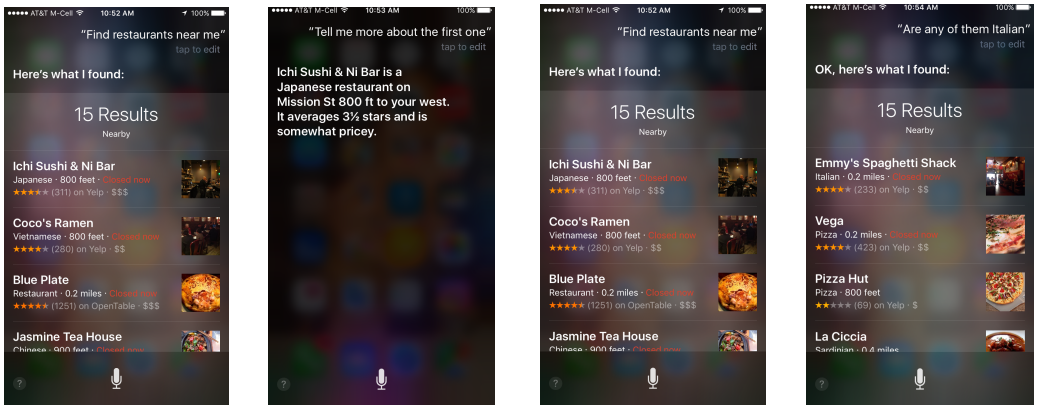


Figure 24.2 The same follow-up questions that Siri couldn't answer in 2014 receive appropriate responses when posed to Siri in 2017.

answer is correct, decides to explain the answer and ask a follow-up question before moving on.

Tutor:	What will the velocity of the object be a second after that (where the initial velocity is 9.8 m/s and the acceleration is 9.8 m/s^2)?
Student:	Is it 19.6 m/s?
Tutor:	Good. A second later, its velocity will be 9.8m/s + 9/8 m/s = 19.6 m/s This is because its acceleration tells us that every second, its velocity increases by 9.8 m/s. So what'll its velocity be, a second after that?
Student:	Um. I'm thinking. 29.4 m/s
Tutor:	Right. Now back to the man and his keys. We've shown that they both have an acceleration of 9.8 m/s^s. So, how do their velocities compare with each other, at every second during the fall?

Figure 24.3 The tutoring domain: An excerpt from a tutorial interaction between a physics student and the ITSPOKE system of Forbes-Riley and Litman (2011) based on the Why2-Atlas text-based tutoring system (VanLehn et al., 2002).

Finally, conversational agents can be purely for fun, like the agents designed for simple chit-chat like Cleverbot, an IR-based chatbot capable of carrying on the kinds of conversations shown in Fig. 24.4.

User:	Do you like Doctor Who?
Cleverbot:	Yes, so funny.
User:	What's your favorite show?
Cleverbot:	I don't have one.
User:	What's your name?
Cleverbot:	Jason I already told you.
User:	No your real name.
Cleverbot:	Stephanie IS my real name.
User:	I thought it was Jason?
Cleverbot:	How did you come to that conclusion.
User:	Because you said so.
Cleverbot:	I have to do my homework.

Figure 24.4 Chatbots: A conversation between a user and Cleverbot, an IR-based chatbot (Carpenter, 2017).

We begin in the next section with chatbots, introducing the three major chatbot architectures: rule-based systems, information retrieval systems, and transduction models. We then turn to task-oriented dialog, introducing the frame-based architecture (the **GUS** architecture) that underlies most modern task-based dialog systems, the VoiceXML system that is a simple industrial instantiation of the architecture, and information on designing and evaluating these systems.

24.1 Chatbots

chatbot

Chatbots are systems that can carry on extended conversations with the goal of mimicking the unstructured conversational or 'chats' characteristic of human-human interaction,

These systems often have an entertainment value, such as Microsoft's XiaoIce (Little Bing 小冰) system (Microsoft, 2014), which chats with people on text messaging platforms. Yet starting from the very first system, ELIZA (Weizenbaum, 1966), chatbots have also been used for practical purposes, such as testing theories of psychological counseling.

Like practically everything else in language processing, chatbot architectures fall into two classes: rule-based systems and corpus-based systems. Rule-based systems include the early influential ELIZA and PARRY systems. Corpus-based systems mine large datasets of human-human conversations, which can be done by using information retrieval (IR-based systems simply copy a human's response from a previous conversation) or by using a machine translation paradigm such as neural network sequence-to-sequence systems, to learn to map from a user utterance to a system response.

24.1.1 Rule-based chatbots: ELIZA and PARRY

ELIZA (Weizenbaum, 1966) is the most important chatbot dialog system in the history of the field, and so we take some time here to describe its algorithm in detail. ELIZA was designed to simulate a Rogerian psychologist, based on a branch of clinical psychology whose methods involve drawing the patient out by reflecting patient's statements back at them. Rogerian psychology is the rare type of conversation in which, as Weizenbaum points out, one can "assume the pose of knowing almost nothing of the real world". If a patient says "I went for a long boat ride" and the psychiatrist says "Tell me about boats", you don't assume she didn't know what a boat is, but rather assume she had some conversational goal. Most chatbots trying to pass Turing test choose a domain with similar properties.

Let's look at the most famous ELIZA conversation, excerpted (deleting some material with ...) from a full interaction given in the original paper:

```
Men are all alike.
IN WHAT WAY
They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE
Well  my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE
He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED
..
```

```
..
WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU?
My father
YOUR FATHER
You are like my father in some ways.
WHAT RESEMBLANCE DO YOU SEE
You are not very aggressive but I think you don't want me to notice that.
WHAT MAKES YOU THINK I AM NOT AGGRESSIVE
You don't argue with me.
WHY DO YOU THINK I DON'T ARGUE WITH YOU
You are afraid of me.
DOES IT PLEASE YOU TO BELIEVE I'M AFRAID OF YOU
My father is afraid of everybody.
WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR  FATHER
Bullies.
DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR  BOYFRIEND MADE YOU COME HERE
```

As we summarized briefly in Chapter 1, ELIZA worked by pattern/transform rules like the following one:

```
(0 YOU 0 ME) [pattern]
->
(WHAT MAKES YOU THINK I 3 YOU) [transform]
```

In the ELIZA pattern, 0 means Kleene*, and in the transform rules the numbers are the index of the constituent in the pattern. Thus the number 3 refers to the second 0 in the first pattern. This rule would transfer

```
You hate me
```

into:

```
WHAT MAKES YOU THINK I HATE YOU
```

Each ELIZA pattern/rule is linked to a keyword that might occur in a user sentence.

The architecture is summarized in Fig. 24.5.

Keywords are associated with a rank, with specific words being more highly ranked, and more general words ranking lower. Consider the following user sentence:

```
I know everybody laughed at me
```

Because it has the word "I", this sentence could match the following rule whose keyword is *I*:

```
(I *) -> (You say you 2)
```

producing:

```
YOU SAY YOU KNOW EVERYBODY LAUGHED AT YOU
```

But as Weizenbaum points out, "I" is a very general word and its keywords lead to very general responses. Instead the keyword "everybody" is much more interesting, since someone using universals like everybody or always is probably "referring to some quite specific event or person". Therefore, ELIZA prefers to respond with the pattern associated with the more specific keyword *everybody* (implementing by just assigning "everybody" rank 5 and "I" rank 0 in the lexicon), whose rule thus transforms the sentence to:

```
function ELIZA GENERATOR(user sentence) returns response

Find the word w in sentence that has the highest keyword rank
 if w exists
     Choose the highest ranked rule r for w that matches sentence
     response ← Apply the transform in r to sentence
     if w = 'my'
         future ← Apply a  transformation from the 'memory' rule list to sentence
         Push future onto memory stack
 else (no keyword applies)
  either
     response ← Apply the transform for the NONE keyword to sentence
  or
     response ← Pop the top response from the memory stack
 return(response)
```

Figure 24.5 A simplified sketch of the ELIZA algorithm. The power of the algorithm comes from the particular transforms associated with each keyword.

```
WHO IN PARTICULAR ARE YOU THINKING OF?
```

If no keyword matches, ELIZA chooses a non-commital response like "PLEASE GO ON", "THAT'S VERY INTERESTING", or "I SEE".

Finally, ELIZA has a clever memory trick that accounts for the very last sentence of the conversation above. Whenever the word "my" is the highest ranked keyword, ELIZA will randomly select a transform on the MEMORY list, apply it to the sentence, and store it on the stack:

```
(MEMORY MY
  (0 MY 0 = LETS DISCUSS FURTHER WHY YOUR 3)
  (0 MY 0 = EARLIER YOU SAID YOUR 3)
  (0 MY 0 = DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR 3
```

Later, if no keyword matches a sentence, ELIZA will return the top of the MEMORY queue instead. [1]

People became deeply emotionally involved with the program. Weizenbaum tells the story of one of his staff who would ask Weizenbaum to leave the room when she talked with ELIZA. When Weizenbaum suggested that he might want to store all the ELIZA conversations for later analysis, people immediately pointed out the privacy implications, which suggested that they were having quite private conversations with ELIZA, despite knowing that it was just software.

Eliza's framework is still used today; modern chatbot system tools like ALICE are based on updated versions of ELIZA's pattern/action architecture.

A few years after ELIZA, another chatbot with a clinical psychology focus, PARRY (Colby et al., 1971), was used to study schizophrenia. In addition to ELIZA-like regular expressions, the PARRY system including a model of its own mental state, with affect variables for the agent's levels of fear and anger; certain topics of conversation might lead PARRY to become more angry or mistrustful. If PARRY's **anger** variable is high, he will choose from a set of "hostile" outputs. If the input mentions his delusion topic, he will increase the value of his **fear** variable and then begin to express the sequence of statements related to his delusion. Parry was the

[1] Fun fact: because of its structure as a queue, this MEMORY trick is the earliest known hierarchical model of discourse in natural language processing.

first known system to pass the Turing test (in 1972!); psychiatrists couldn't distinguish text transcripts of interviews with PARRY from transcripts of interviews with real paranoids (Colby et al., 1972).

24.1.2 Corpus-based chatbots

Corpus-based chatbots, instead of using hand-built rules, mine conversations of human-human conversations, or sometimes mine the human responses from human-machine conversations. Serban et al. (2017) summarizes some such available corpora, such as conversations on chat platforms, on Twitter, or in movie dialog, which is available in great quantities and has been shown to resemble natural conversation (Forchini, 2013). Chatbot responses can even be extracted from sentences in corpora of non-dialog text.

There are two common architectures for corpus-based chatbots: information retrieval, and machine learned sequence transduction. Like rule-based chatbots (but unlike frame-based dialog systems), most corpus-based chatbots do very little modeling of the conversational context. Instead they focus on generating a single response turn that is appropriate given the user's immediately previous utterance. For this reason they are often called **response generation** systems. Corpus-based chatbots thus have some similarity to question answering systems, which focus on single responses while ignoring context or larger conversational goals.

response generation

IR-based chatbots

The principle behind information retrieval based chatbots is to respond to a user's turn X by repeating some appropriate turn Y from a corpus of natural (human) text. The differences across such systems lie in how they choose the corpus, and how they decide what counts as an appropriate human turn to copy.

A common choice of corpus is to collect databases of human conversations. These can come from microblogging platforms like Twitter or any Weibo (微博). Another approach is to use corpora of movie dialog. Once a chatbot has been put into practice, the turns that humans use to respond to the chatbot can be used as additional conversational data for training.

Given the corpus and the user's sentence, IR-based systems can use any retrieval algorithm to choose an appropriate response from the corpus. The two simplest methods are the following:

1. Return the response to the most similar turn: Given user query q and a conversational corpus C, find the turn t in C that is most similar to q (for example has the highest cosine with q) and return the *following* turn, i.e. the human response to t in C:

$$r = response\left(\underset{t \in C}{\text{argmax}} \frac{q^T t}{||q||t||}\right) \tag{24.1}$$

The idea is that we should look for a turn that most resembles the user's turn, and return the human response to that turn (Jafarpour et al. 2009, Leuski and Traum 2011).

2. Return the most similar turn: Given user query q and a conversational corpus C, return the turn t in C that is most similar to q (for example has the highest cosine with q):

$$r = \underset{t \in C}{\text{argmax}} \frac{q^T t}{||q||t||} \tag{24.2}$$

The idea here is to directly match the users query q with turns from C, since a good response will often share words or semantics with the prior turn.

In each case, any similarity function can be used, most commonly cosines computed either over words (using tf-idf) or over embeddings.

Although returning the *response* to the most similar turn seems like a more intuitive algorithm, returning the most similar turn seems to work better in practice, perhaps because selecting the response adds another layer of indirection that can allow for more noise (Ritter et al. 2011, Wang et al. 2013).

The IR-based approach can be extended by using more features than just the words in the q (such as words in prior turns, or information about the user), and using any full IR ranking approach. Commercial implementations of the IR-based approach include Cleverbot (Carpenter, 2017) and Microsoft's XiaoIce (Little Bing 小冰) system (Microsoft, 2014).

Instead of just using corpora of conversation, the IR-based approach can be used to draw responses from narrative (non-dialog) text. For example, the pioneering COBOT chatbot (Isbell et al., 2000) generated responses by selecting sentences from a corpus that combined the Unabomber Manifesto by Theodore Kaczynski, articles on alien abduction, the scripts of "The Big Lebowski" and "Planet of the Apes". Chatbots that want to generate informative turns such as answers to user questions can use texts like Wikipedia to draw on sentences that might contain those answers (Yan et al., 2016).

Sequence to sequence chatbots

An alternate way to use a corpus to generate dialog is to think of response generation as a task of *transducing* from the user's prior turn to the system's turn. This is basically the machine learning version of Eliza; the system learns from a corpus to transduce a question to an answer.

This idea was first developed by using phrase-based machine translation (Ritter et al., 2011) to translate a user turn to a system response. It quickly became clear, however, that the task of response generation was too different from machine translation. In machine translation words or phrases in the source and target sentences tend to align well with each other; but in conversation, a user utterance may share no words or phrases with a coherent response.

Instead, (roughly contemporaneously by Shang et al. 2015, Vinyals and Le 2015, and Sordoni et al. 2015) transduction models for response generation were modeled instead using encoder-decoder (seq2seq) models (Chapter 22), as shown in Fig. 24.6.

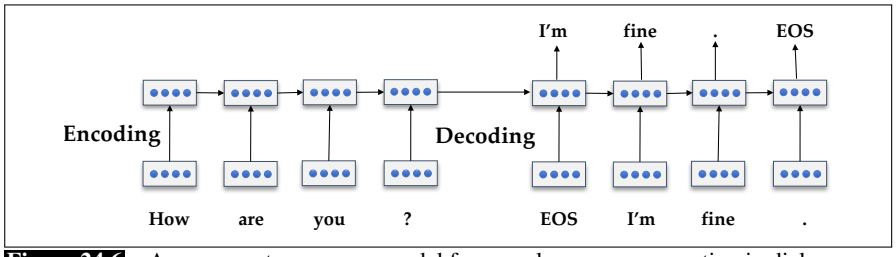


Figure 24.6 A sequence to sequence model for neural response generation in dialog.

A number of modifications are required to the basic seq2seq model to adapt it for the task of response generation. For example basic seq2seq models have a tendency to produce predictable but repetitive and therefore dull responses like "I'm OK" or "I don't know" that shut down the conversation. This can be addressed by changing the objective function for seq2seq model training to a mutual information objective, or by modifying a beam decoder to keep more diverse responses in the beam (Li

et al., 2016a).

Another problem with the simple SEQ2SEQresponse generation architecture is its inability to model the longer prior context of the conversation. This can be done by allowing the model to see prior turns, such as by using a hierarchical model that summarizes information over multiple prior turns (Lowe et al., 2017b).

Finally, SEQ2SEQresponse generators focus on generating single responses, and so don't tend to do a good job of continuously generating responses that cohere across multiple turns. This can be addressed by using reinforcement learning, as well as techniques like adversarial networks, to learn to choose responses that make the overall conversation more natural (Li et al. 2016b, Li et al. 2017).

Fig. 24.7 shows some sample responses generated by a vanilla SEQ2SEQmodel, and from a model trained by an adversarial algorithm to produce responses that are harder to distinguish from human responses (Li et al., 2017).

Input	tell me ... how long have you had this falling sickness ?
Vanilla-SEQ2SEQ	i'm not a doctor.
Adversarial	a few months, i guess .
Input	so i had the doctors test sammy 's response to conditioning .
Vanilla-SEQ2SEQ	sammy wrote the test sammy wrote the test .
Adversarial	so he took the pills .
Input	they didn 't have much success with this problem commander .
Vanilla-SEQ2SEQ	they 're not the only ones who have been in the system .
Adversarial	can we find someone else ?

Figure 24.7 Sample responses generated by a SEQ2SEQmodel trained either with a vanilla maximum likelihood objective, or adversarially trained to produce sentences that are hard for an adversary to distinguish from human sentences (Li et al., 2017).

Evaluating Chatbots

Chatbots are generally evaluated by humans. The slot-filling evaluations used for task-based dialog (Section 24.2.3) aren't appropriate for this task (Artstein et al., 2009), and word-overlap metrics like BLEU for comparing a chatbot's response to a human response turn out to correlate very poorly with human judgments (Liu et al., 2016). BLEU performs poorly because there are so many possible responses to any given turn; word-overlap metrics work best when the space of responses is small and lexically overlapping, as is the case in machine translation.

While human evaluation is therefore required for evaluating chatbots, there are beginning to be models for automatic evaluation. The ADEM (Lowe et al., 2017a) classifier is trained on a set of responses labeled by humans with how appropriate they are, and learns to predict this label from the dialog context and the words in the system response.

adversarial evaluation

Another paradigm is **adversarial evaluation** (Bowman et al. 2016, Kannan and Vinyals 2016, Li et al. 2017), inspired by the Turing test. The idea is to train a "Turing-like" evaluator classifier to distinguish between human-generated responses and machine-generated responses. The more successful a response generation system is at fooling this evaluator, the better the system.

24.2 Frame Based Dialog Agents

domain ontology

Modern task-based dialog systems are based on a **domain ontology**, a knowledge

frame
slot

structure representing the kinds of intentions the system can extract from user sentences. The ontology defines one or more **frames**, each a collection of **slots**, and defines the **values** that each slot can take. This frame-based architecture was first introduced in 1977 in the influential GUS system for travel planning (Bobrow et al., 1977), and has been astonishingly long-lived, underlying most modern commercial digital assistants. Figure 24.8 shows a conversation with the original GUS system.

GUS:	Hello. My name is GUS. I can help you plan a simple trip by air. Where do you want to go ?
Client:	I want to go to San Diego on May 28
GUS:	What time do you want to leave?
Client:	I must be in San Diego before 10 am
GUS:	Would you like P.S.A. flight 102 that arrives at 9.15 am ?
Client:	What is the next flight ?
GUS:	Air California flight 310 that leaves at 8.30am and arrives at 10.00am
Client:	I'll take the first one
GUS:	In what name should I make a reservation, please?
Client:	Dan Bobrow
GUS:	I have confirmed the following flight: P.S.A. flight 102 on Wednesday May 28 from San Jose to San Diego that leaves at 7.30 am and arrives at 9.15 am.
GUS:	What date do you want to return on ?
Client:	On Friday in the evening.
GUS:	Would you like the flight that leaves at 7.45 pm ?
Client:	That's fine.
GUS:	I have confirmed the following flight: P.S.A. flight 307 on Friday May 30 from San Diego to San Jose that leaves at 7.45 pm and arrives at 9.30 pm Thank you for calling. Goodbye

Figure 24.8 The travel domain: A transcript of an actual dialog with the GUS system of Bobrow et al. (1977). P.S.A. and Air California were airlines of that period.

The set of slots in a GUS-style frame specifies what the system needs to know, and the filler of each slot is constrained to values of a particular semantic type. In the travel domain, for example, a slot might be of type city (hence take on values like *San Francisco*, or *Hong Kong*) or of type date, airline, or time:

Slot	Type
ORIGIN CITY	city
DESTINATION CITY	city
DEPARTURE TIME	time
DEPARTURE DATE	date
ARRIVAL TIME	time
ARRIVAL DATE	date

Types in GUS, as in modern frame-based dialog agents, may have hierarchical structure; for example the *date* type in GUS is itself a frame with slots with types like *integer* or members of sets of weekday names:

```
DATE
  MONTH NAME
  DAY (BOUNDED-INTEGER 1 31)
  YEAR INTEGER
  WEEKDAY (MEMBER (SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY))
```

24.2.1 Control structure for frame-based dialog

The control architecture of frame-based dialog systems is designed around the frame. The goal is to fill the slots in the frame with the fillers the user intends, and then perform the relevant action for the user (answering a question, or booking a flight). Most frame-based dialog systems are based on finite-state automata that are hand-designed for the task by a dialog designer.

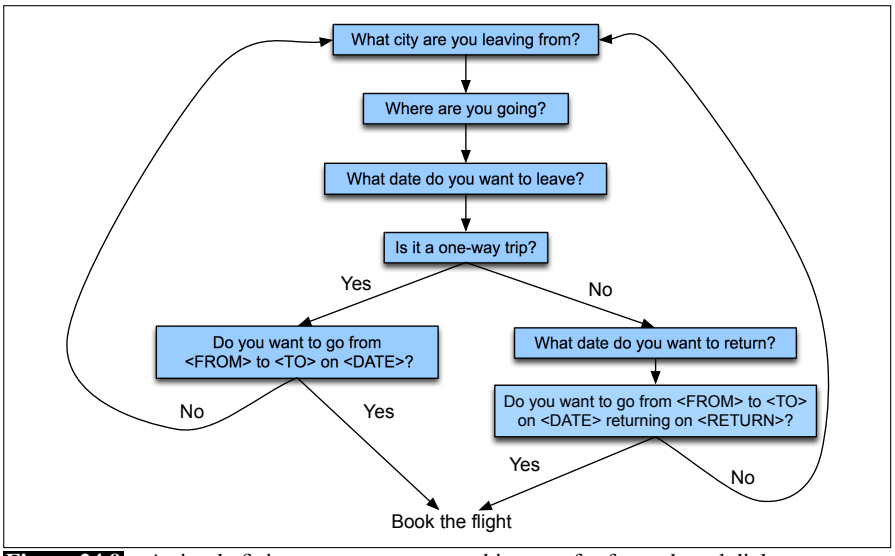


Figure 24.9 A simple finite-state automaton architecture for frame-based dialog.

Consider the very simple finite-state control architecture shown in Fig. 24.9, implementing a trivial airline travel system whose job is to ask the user for the information for 4 slots: departure city, a destination city, a time, and whether the trip is one-way or round-trip. Let's first associate with each slot a question to ask the user:

Slot	Question
ORIGIN CITY	"From what city are you leaving?"
DESTINATION CITY	"Where are you going?"
DEPARTURE TIME	"When would you like to leave?"
ARRIVAL TIME	"When do you want to arrive?"

Figure 24.9 shows a sample dialog manager for such a system. The states of the FSA correspond to the slot questions, user, and the arcs correspond to actions to take depending on what the user responds. This system completely controls the conversation with the user. It asks the user a series of questions, ignoring (or misinterpreting) anything that is not a direct answer to the question and then going on to the next question.

initiative

system-initiative

The speaker in control of any conversation is said to have the **initiative** in the conversation. Systems that completely control the conversation in this way are thus called **system-initiative**. By contrast, in normal human-human dialog, initiative shifts back and forth between the participants (Bobrow et al. 1977, Walker and Whittaker 1990).

The single-initiative finite-state dialog architecture has the advantage that the system always knows what question the user is answering. This means the system can prepare the speech recognizer with a language model tuned to answers for this

question, and also makes natural language understanding easier. Most finite-state systems also allow **universal** commands that can be said anywhere in the dialog, like **help**, to give a help message, and **start over** (or **main menu**), which returns the user to some specified main start state,. Nonetheless such a simplistic finite-state architecture is generally applied only to simple tasks such as entering a credit card number, or a name and password.

universal

For most applications, users need a bit more flexibility. In a travel-planning situation, for example, a user may say a sentence that fills multiple slots at once:

(24.3) I want a flight from San Francisco to Denver one way leaving after five p.m. on Tuesday.

Or in cases where there are multiple frames, a user may say something to shift frames, for example from airline reservations to reserving a rental car:

(24.4) I'd like to book a rental car when I arrive at the airport.

The standard GUS architecture for frame-based dialog systems, used in various forms in modern systems like Apple's Siri, Amazon's Alexa, and the Google Assistant, therefore follows the frame in a more flexible way. The system asks questions of the user, filling any slot that the user specifies, even if a user's response fills multiple slots or doesn't answer the question asked. The system simply skips questions associated with slots that are already filled. Slots may thus be filled out of sequence. The GUS architecture is thus a kind of **mixed initiative**, since the user can take at least a bit of conversational initiative in choosing what to talk about.

mixed initiative

The GUS architecture also has condition-action rules attached to slots. For example, a rule attached to the DESTINATION slot for the plane booking frame, once the user has specified the destination, might automatically enter that city as the default *StayLocation* for the related hotel booking frame.

Once the system has enough information it performs the necessary action (like querying a database of flights) and returns the result to the user.

We mentioned in passing the linked airplane and travel frames. Many domains, of which travel is one, require the ability to deal with multiple frames. Besides frames for car or hotel reservations, we might need frames with general route information (for questions like *Which airlines fly from Boston to San Francisco?*), information about airfare practices (for questions like *Do I have to stay a specific number of days to get a decent airfare?*).

In addition, once we have given the user options (such as a list of restaurants), we can even have a special frame for 'asking questions about this list', whose slot is the particular restaurant the user is asking for more information about, allowing the user to say 'the second one' or 'the Italian one'.

Since users may switch from frame to frame, the system must be able to disambiguate which slot of which frame a given input is supposed to fill and then switch dialog control to that frame.

Because of this need to dynamically switch control, the GUS architecture is a **production rule** system. Different types of inputs cause different productions to fire, each of which can flexibly fill in different frames. The production rules can then switch control according to factors such as the user's input and some simple dialog history like the last question that the system asked.

Commercial dialog systems provide convenient interfaces or libraries to make it easy to build systems with these kinds of finite-state or production rule systems, for example providing graphical interfaces to allow dialog modules to be chained together.

24.2.2 Natural language understanding for filling slots

domain classification

The goal of the natural language understanding component is to extract three things from the user's utterance. The first task is **domain classification**: is this user for example talking about airlines, programming an alarm clock, or dealing with their calendar? Of course this 1-of-n classification tasks is unnecessary for single-domain systems that are focused on, say, only calendar management, but multi-domain dialog systems are the modern standard. The second is user **intent determination**: what general task or goal is the user trying to accomplish? For example the task could be to Find a Movie, or Show a Flight, or Remove a Calendar Appointment. Finally, we need to do **slot filling**: extract the particular slots and fillers that the user intends the system to understand from their utterance with respect to their intent. From a user utterance like this one:

intent determination

slot filling

```
Show me morning flights from Boston to San Francisco on Tuesday
```

a system might want to build a representation like:

```
DOMAIN:       AIR-TRAVEL
INTENT:       SHOW-FLIGHTS
ORIGIN-CITY:  Boston
ORIGIN-DATE:  Tuesday
ORIGIN-TIME:  morning
DEST-CITY:    San Francisco
```

while an utterance like

```
Wake me tomorrow at 6
```

should give an intent like this:

```
DOMAIN:  ALARM-CLOCK
INTENT:  SET-ALARM
TIME:    2017-07-01 0600-0800
```

The task of slot-filling, and the simpler tasks of domain and intent classification, are special cases of the task of semantic parsing discussed in Chapter 16. Dialog agents can thus extract slots, domains, and intents from user utterances by applying any of the semantic parsing approaches discussed in that chapter.

The method used in the original GUS system, and still quite common in industrial applications, is to use hand-written rules, often as part of the condition-action rules attached to slots or concepts.

For example we might just define a regular expression consisting of a set strings that map to the SET-ALARM intent:

```
wake me (up) | set (the|an) alarm | get me up
```

We can build more complex automata that instantiate sets of rules like those discussed in Chapter 17, for example extracting a slot filler by turning a string like `Monday at 2pm` into an object of type `date` with parameters (DAY, MONTH, YEAR, HOURS, MINUTES).

Rule-based systems can be even implemented with full grammars. Research systems like the Phoenix system (Ward and Issar, 1994) consists of large hand-designed **semantic grammars** with thousands of rules. A semantic grammar is a context-free grammar in which the left-hand side of each rule corresponds to the semantic entities being expressed (i.e., the slot names) as in the following fragment:

semantic grammar

SHOW	→	show me | i want | can i see|...
DEPART_TIME_RANGE	→	(after|around|before) HOUR | morning | afternoon | evening
HOUR	→	one|two|three|four...|twelve (AMPM)
FLIGHTS	→	(a) flight | flights
AMPM	→	am | pm
ORIGIN	→	from CITY
DESTINATION	→	to CITY
CITY	→	Boston | San Francisco | Denver | Washington

Semantic grammars can be parsed by any CFG parsing algorithm (see Chapter 11), resulting in a hierarchical labeling of the input string with semantic node labels, as shown in Fig. 24.10.

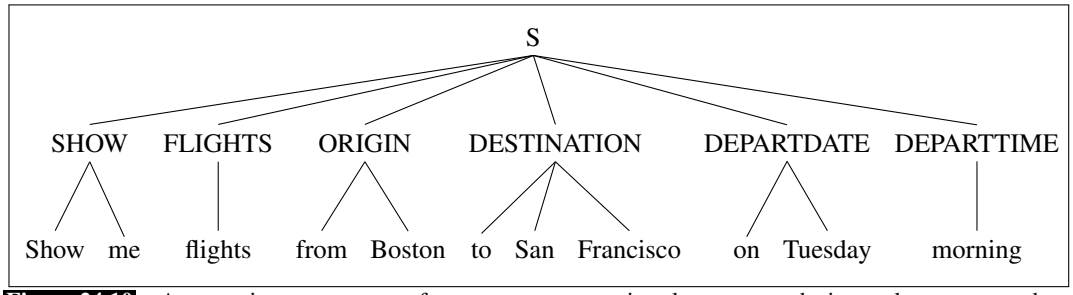


Figure 24.10 A semantic grammar parse for a user sentence, using slot names as the internal parse tree nodes.

Whether regular expressions or parsers are used, it remains only to put the fillers into some sort of canonical form, for example by normalizing dates as discussed in Chapter 17.

A number of tricky issues have to be dealt with. One important issue is negation; if a user specifies that they "can't fly Tuesday morning", or want a meeting "any time except Tuesday morning", a simple system will often incorrectly extract "Tuesday morning" as a user goal, rather than as a negative constraint.

N-best list

Speech recognition errors must also be dealt with. One common trick is to make use of the fact that speech recognizers often return a ranked **N-best list** of hypothesized transcriptions rather than just a single candidate transcription. The regular expressions or parsers can simply be run on every sentence in the N-best list, and any patterns extracted from any hypothesis can be used.

As we saw earlier in discussing information extraction, the rule-based approach is very common in industrial applications. It has the advantage of high precision, and if the domain is narrow enough and experts are available, can provide sufficient coverage as well. On the other hand, the hand-written rules or grammars can be both expensive and slow to create, and hand-written rules can suffer from recall problems.

A common alternative is to use supervised machine learning. Assuming a training set is available which associates each sentence with the correct semantics, we can train a classifier to map from sentences to intents and domains, and a sequence model to map from sentences to slot fillers.

For example given the sentence:

```
I want to fly to San   Francisco on Monday    afternoon  please
```

we might first apply a simple 1-of-N classifier (logistic regression, neural network, etc.) that uses features of the sentence like word N-grams to determine that the domain is AIRLINE and and the intent is SHOWFLIGHT.

Next to do slot filling we might first apply a classifier that uses similar features of the sentence to predict which slot the user wants to fill. Here in addition to

word unigram, bigram, and trigram features we might use named entity features or features indicating that a word is in a particular lexicon (such as a list of cities, or airports, or days of the week) and the classifer would return a slot name (in this case DESTINATION, DEPARTURE-DAY, and DEPARTURE-TIME). A second classifier can then be used to determine the filler of the named slot, for example a city classifier that uses N-grams and lexicon features to determine that the filler of the DESTINATION slot is SAN FRANCISCO.

An alternative is to use a sequence model (MEMMs, CRFs, RNNs) to directly assign a slot label to each word in the sequence, following the method used for other information extraction models in Chapter 17 (Pieraccini et al. 1991, Raymond and Riccardi 2007, Mesnil et al. 2015, Hakkani-Tür et al. 2016). Once again we would need a supervised training test, with sentences paired with sequences of **IOB** labels like the following:

IOB

```
O O    O  O   O  B-DES I-DES     O  B-DEPTIME I-DEPTIME  O
I want to fly to San   Francisco on Monday    afternoon  please
```

Recall from Chapter 17 that in IOB tagging we introduce a tag for the beginning (B) and inside (I) of each slot label, and one for tokens outside (O) any slot label. The number of tags is thus $2n+1$ tags, where n is the number of slots.

Any IOB tagger sequence model can then be trained on a training set of such labels. Feature-based sequence models (MEMM, CRF) make use of features like word embeddings, word unigrams and bigrams, lexicons (for example lists of city names), and slot transition features (perhaps DESTINATION is more likely to follow ORIGIN than the other way around) to map a user's utterance to the slots. An MEMM (Chapter 8) for example, combines these features of the input word w_i, its neighbors within l words w_{i-l}^{i+l}, and the previous k slot tags s_{i-k}^{i-1} to compute the most likely slot label sequence S from the word sequence W as follows:

$$\begin{aligned}\hat{S} &= \operatorname*{argmax}_S P(S|W) \\ &= \operatorname*{argmax}_S \prod_i P(s_i|w_{i-l}^{i+l}, s_{i-k}^{i-1}) \\ &= \operatorname*{argmax}_S \prod_i \frac{\exp\left(\sum_i w_i f_i(s_i, w_{i-l}^{i+l}, s_{i-k}^{i-1})\right)}{\sum_{s' \in \text{slotset}} \exp\left(\sum_i w_i f_i(s', w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}\end{aligned} \tag{24.5}$$

The Viterbi algorithm is used to decode the best slot sequence $\hat{S}$.

Neural network architectures mostly eschew the feature extraction step, instead using the bi-LSTM architecture introduced in Chapter 9, and applied to IOB-style named entity tagging in Chapter 17. A typical LSTM-style architecture is shown in Fig. 24.11. Here the input is a series of words $w_1 ... w_n$, and the output is a series of IOB tags $s_1 ... s_n$. In the architecture as introduced in Chapter 17, the input words are converted into two embeddings: standard word2vec or GloVe embeddings, and a character-based embedding, which are concatenated together and passed through a bi-LSTM. The output of the bi-LSTM can be passed to a softmax choosing an IOB tag for each input word, or to a CRF layer which uses Viterbi to find the best series of IOB tags. In addition, neural systems can combine the domain-classification and intent-extraction tasks with slot-filling simply by adding a domain concatenated with an intent as the desired output for the final EOS token.

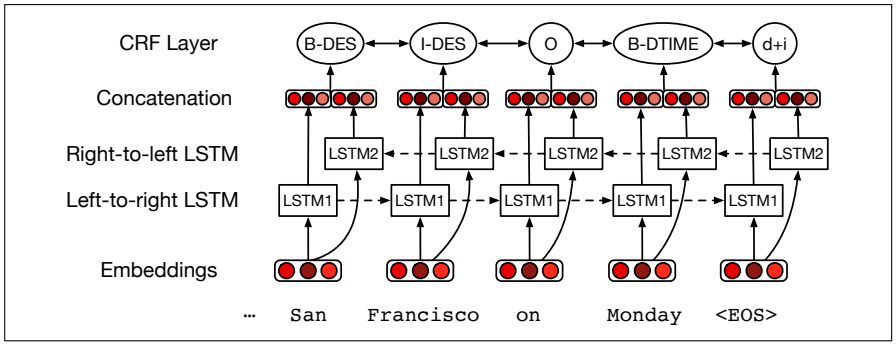


Figure 24.11 An LSTM architecture for slot filling, mapping the words in the input to a series of IOB tags plus a final state consisting of a domain concatenated with an intent.

Once the sequence labeler has tagged the user utterance, a filler string can be extracted for each slot from the tags (e.g., "San Francisco"), and these word strings can then be normalized to the correct form in the ontology (perhaps the airport code'SFO'). This normalization can take place by using homonym dictionaries (specifying, for example, that SF, SFO, and San Francisco are the same place).

In industrial contexts, machine learning-based systems for slot-filling are often bootstrapped from rule-based systems in a semi-supervised learning manner. A rule-based system is first built for the domain, and a test-set is carefully labeled. As new user utterances come in, they are paired with the labeling provided by the rule-based system to create training tuples. A classifier can then be trained on these tuples, using the test-set to test the performance of the classifier against the rule-based system. Some heuristics can be used to eliminate errorful training tuples, with the goal of increasing precision. As sufficient training samples become available the resulting classifier can often outperform the original rule-based system (Suendermann et al., 2009), although rule-based systems may still remain higher-precision for dealing with complex cases like negation.

24.2.3 Evaluating Slot Filling

An intrinsic error metric for natural language understanding systems for slot filling is the Slot Error Rate for each sentence:

$$\text{Slot Error Rate for a Sentence} = \frac{\text{\# of inserted/deleted/subsituted slots}}{\text{\# of total reference slots for sentence}} \quad (24.6)$$

Consider a system faced with the following sentence:

(24.7) Make an appointment with Chris at 10:30 in Gates 104

which extracted the following candidate slot structure:

Slot	Filler
PERSON	Chris
TIME	11:30 a.m.
ROOM	Gates 104

Here the slot error rate is 1/3, since the TIME is wrong. Instead of error rate, slot precision, recall, and F-score can also be used.

A perhaps more important, although less fine-grained, measure of success is an extrinsic metric like **task error rate**. In this case, the task error rate would quantify how often the correct meeting was added to the calendar at the end of the interaction.

24.2.4 Other components of frame-based dialog

We've focused on the natural language understanding component that is the core of frame-based systems, but here we also briefly mention other modules.

The ASR (automatic speech recognition) component takes audio input from a phone or other device and outputs a transcribed string of words, as discussed in Chapter 26. Various aspects of the ASR system may be optimized specifically for use in conversational agents.

Because what the user says to the system is related to what the system has just said, language models in conversational agent depend on the dialog state. For example, if the system has just asked the user "What city are you departing from?", the ASR language model can be constrained to just model answers to that one question. This can be done by training an N-gram language model on answers to this question. Alternatively a finite-state or context-free grammar can be hand written to recognize only answers to this question, perhaps consisting only of city names or perhaps sentences of the form 'I want to (leave|depart) from [CITYNAME]'. Indeed, many simple commercial dialog systems use only non-probabilistic language models based on hand-written finite-state grammars that specify all possible responses that the system understands. We give an example of such a hand-written grammar for a VoiceXML system in Section 24.3.

restrictive grammar

A language model that is completely dependent on dialog state is called a **restrictive grammar**, and can be used to constrain the user to only respond to the system's last utterance. When the system wants to allow the user more options, it might mix this state-specific language model with a more general language model.

language generation

template-based generation

prompt

The **language generation** module of any dialog system produces the utterances that the system says to the user. Frame-based systems tend to use **template-based generation**, in which all or most of the words in the sentence to be uttered to the user are prespecified by the dialog designer. Sentences created by these templates are often called **prompts**. Templates might be completely fixed (like 'Hello, how can I help you?'), or can include some variables that are filled in by the generator, as in the following:

What time do you want to leave CITY-ORIG?
Will you return to CITY-ORIG from CITY-DEST?

These sentences are then passed to the TTS (text-to-speech) component (see Chapter 26). More sophisticated statistical generation strategies will be discussed in Section **??** of Chapter 25.

24.3 VoiceXML

There are many commercial systems that allow developers to implement frame-based dialog systems, including the user-definable skills in Amazon Alexa or the actions in Google Assistant. These systems provide libraries for defining the rules for detecting user intents and filling in slots, and for expressing the architecture for controlling which frames and actions the system should take at which times.

VoiceXML

Instead of focusing on a commercial engine, we introduce here a simple declarative formalism that has similar capabilities to each of them: **VoiceXML**, the Voice Extensible Markup Language (`http://www.voicexml.org/`), an XML-based dialog design language for creating simple frame-based dialogs. Although VoiceXML is simpler than a full commercial frame-based system (it's deterministic, and hence

only allows non-probabilistic grammar-based language models and rule-based semantic parsers), it's still a handy way to get a hands-on grasp of frame-based dialog system design.

A VoiceXML document contains a set of dialogs, each a **menu** or a **form**. A form is a frame, whose slots are called **fields**. The VoiceXML document in Fig. 24.12 shows three fields for specifying a flight's origin, destination, and date. Each field has a variable name (e.g., `origin`) that stores the user response, a **prompt**, (e.g., *Which city do you want to leave from*), and a grammar that is passed to the speech recognition engine to specify what is allowed to be recognized. The grammar for the first field in Fig. 24.12 allows the three phrases *san francisco*, *barcelona*, and *new york*. The VoiceXML interpreter walks through a form in document order, repeatedly selecting each item in the form, and each field in order.

prompt

```
<noinput>
I'm sorry, I didn't hear you.  <reprompt/>
</noinput>

<nomatch>
I'm sorry, I didn't understand that.  <reprompt/>
</nomatch>

<form>
  <block>    Welcome to the air travel consultant.  </block>
  <field name="origin">
      <prompt>     Which city do you want to leave from?  </prompt>
      <grammar type="application/x=nuance-gsl">
        [(san francisco) barcelona (new york)]
      </grammar>
      <filled>
         <prompt>  OK, from <value expr="origin"/>  </prompt>
      </filled>
  </field>
  <field name="destination">
      <prompt>  And which city do you want to go to?   </prompt>
      <grammar type="application/x=nuance-gsl">
        [(san francisco) barcelona (new york)]
      </grammar>
      <filled>
         <prompt>   OK, to <value expr="destination"/>   </prompt>
      </filled>
  </field>
  <field name="departdate" type="date">
      <prompt>  And what date do you want to leave?  </prompt>
      <filled>
         <prompt>   OK, on <value expr="departdate"/>   </prompt>
      </filled>
  </field>
  <block>
      <prompt> OK, I have you are departing from  <value expr="origin"/>
              to <value expr="destination"/> on <value expr="departdate"/>
      </prompt>
      send the info to book a flight...
  </block>
</form>
```

Figure 24.12 A VoiceXML script for a form with three fields, which confirms each field and handles the `noinput` and `nomatch` situations.

The prologue of the example shows two global defaults for error handling. If the user doesn't answer after a prompt (i.e., silence exceeds a timeout threshold), the VoiceXML interpreter will play the <`noinput`> prompt. If the user says something that doesn't match the grammar for that field, the VoiceXML interpreter will play the <`nomatch`> prompt. VoiceXML provides a <`reprompt/`> command, which repeats the prompt for whatever field caused the error.

The <`filled`> tag for a field is executed by the interpreter as soon as the field has been filled by the user. Here, this feature is used to confirm the user's input.

VoiceXML 2.0 specifies seven built-in grammar types: `boolean`, `currency`, `date`, `digits`, `number`, `phone`, and `time`. By specifying the `departdate` field as

type `date`, a date-specific language model will be passed to the speech recognizer.

```
<noinput>    I'm sorry, I didn't hear you.  <reprompt/>  </noinput>

<nomatch> I'm sorry, I didn't understand that.  <reprompt/> </nomatch>

<form>
   <grammar type="application/x=nuance-gsl">
    <![CDATA[
    Flight (  ?[
              (i [wanna (want to)] [fly go])
              (i'd like to [fly go])
              ([(i wanna)(i'd like a)] flight)
           ]
           [
             ( [from leaving departing] City:x) {<origin $x>}
             ( [(?going to)(arriving in)] City:x) {<destination $x>}
             ( [from leaving departing] City:x
               [(?going to)(arriving in)] City:y) {<origin $x> <destination $y>}
           ]
           ?please
     )
     City [ [(san francisco) (s f o)] {return( "san francisco, california")}
            [(denver) (d e n)] {return( "denver, colorado")}
            [(seattle) (s t x)] {return( "seattle, washington")}
     ]
     ]]> </grammar>

  <initial name="init">
     <prompt> Welcome to the consultant. What are your travel plans?  </prompt>
  </initial>

  <field name="origin">
     <prompt> Which city do you want to leave from?  </prompt>
     <filled>
        <prompt> OK, from <value expr="origin"/> </prompt>
     </filled>
  </field>
  <field name="destination">
     <prompt> And which city do you want to go to?  </prompt>
     <filled>
        <prompt> OK, to <value expr="destination"/> </prompt>
     </filled>
  </field>
  <block>
     <prompt> OK, I have you are departing from  <value expr="origin"/>
              to  <value expr="destination"/>.  </prompt>
    send the info to book a flight...
  </block>
</form>
```

Figure 24.13 A mixed-initiative VoiceXML dialog. The grammar allows sentences that specify the origin or destination cities or both. The user can respond to the initial prompt by specifying origin city, destination city, or both.

Figure 24.13 gives a mixed initiative example, allowing the user to answer questions in any order or even fill in multiple slots at once. The VoiceXML interpreter has a *guard condition* on fields, a test that keeps a field from being visited; the default test skips a field if its variable is already set.

Figure 24.13 also shows a more complex CFG grammar with two rewrite rules, `Flight` and `City`. The Nuance GSL grammar formalism uses parentheses () to mean concatenation and square brackets [] to mean disjunction. Thus, a rule like (24.8) means that `Wantsentence` can be expanded as `i want to fly` or `i want to go`, and `Airports` can be expanded as `san francisco` or `denver`.

(24.8)
```
Wantsentence (i want to [fly go])
Airports [(san francisco) denver]
```

VoiceXML grammars allow semantic attachments, such as the text string (`"denver, colorado"`) the return for the `City` rule, or a slot/filler , like the attachments for the `Flight` rule which fills the slot (`<origin>` or `<destination>` or both) with the value passed up in the variable *x* from the `City` rule.

TTS Performance	Was the system easy to understand ?
ASR Performance	Did the system understand what you said?
Task Ease	Was it easy to find the message/flight/train you wanted?
Interaction Pace	Was the pace of interaction with the system appropriate?
User Expertise	Did you know what you could say at each point?
System Response	How often was the system sluggish and slow to reply to you?
Expected Behavior	Did the system work the way you expected it to?
Future Use	Do you think you'd use the system in the future?

Figure 24.14 User satisfaction survey, adapted from Walker et al. (2001).

Because Fig. 24.13 is a mixed-initiative grammar, the grammar has to be applicable to any of the fields. This is done by making the expansion for `Flight` a disjunction; note that it allows the user to specify only the origin city, the destination city, or both.

24.4 Evaluating Dialog Systems

Evaluation is crucial in dialog system design. If the task is unambiguous, we can simply measure absolute task success (did the system book the right plane flight, or put the right event on the calendar).

To get a more fine-grained idea of user happiness, we can compute a *user satisfaction rating*, having users interact with a dialog system to perform a task and then having them complete a questionnaire. For example, Fig. 24.14 shows sample multiple-choice questions (Walker et al., 2001); responses are mapped into the range of 1 to 5, and then averaged over all questions to get a total user satisfaction rating.

It is often economically infeasible to run complete user satisfaction studies after every change in a system. For this reason, it is useful to have performance evaluation heuristics that correlate well with human satisfaction. A number of such factors and heuristics have been studied, often grouped into two kinds of criteria: how well the system allows users to accomplish their goals (maximizing task success) the least problems (minimizing costs) :

Task completion success: Task success can be measured by evaluating the correctness of the total solution. For a frame-based architecture, this might be the percentage of slots that were filled with the correct values or the percentage of subtasks that were completed. Interestingly, sometimes the user's *perception* of whether they completed the task is a better predictor of user satisfaction than the actual task completion success. (Walker et al., 2001).

Efficiency cost: Efficiency costs are measures of the system's efficiency at helping users. This can be measured by the total elapsed time for the dialog in seconds, the number of total turns or of system turns, or the total number of queries (Polifroni et al., 1992). Other metrics include the number of system non-responses and the "turn correction ratio": the number of system or user turns that were used solely to correct errors divided by the total number of turns (Danieli and Gerbino 1995, Hirschman and Pao 1993).

Quality cost: Quality cost measures other aspects of the interactions that affect users' perception of the system. One such measure is the number of times the ASR system failed to return any sentence, or the number of ASR rejection prompts. Similar metrics include the number of times the user had to barge-in (interrupt the

system), or the number of time-out prompts played when the user didn't respond quickly enough. Other quality metrics focus on how well the system understood and responded to the user. The most important is the **slot error rate** described above, but other components include the inappropriateness (verbose or ambiguous) of the system's questions, answers, and error messages or the correctness of each question, answer, or error message (Zue et al. 1989, Polifroni et al. 1992).

24.5 Dialog System Design

The user plays a more important role in dialog systems than in most other areas of speech and language processing, and thus this area of language processing is the one that is most closely linked with the field of Human-Computer Interaction (HCI).

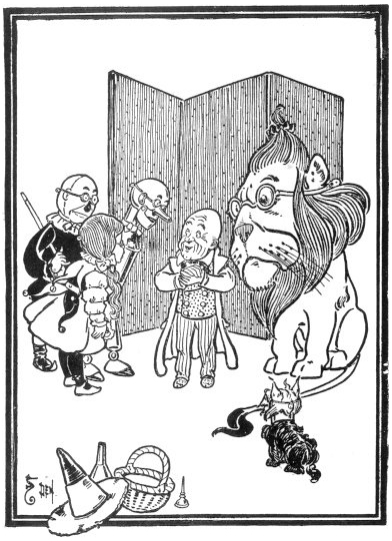
voice user interface

How does a dialog system developer choose dialog strategies, prompts, error messages, and so on? This process is often called **voice user interface** design, and generally follows the **user-centered design** principles of Gould and Lewis (1985):

1. Study the user and task: Understand the potential users and the nature of the task by interviews with users, investigation of similar systems, and study of related human-human dialogs.

Wizard-of-Oz system

2. Build simulations and prototypes: A crucial tool in building dialog systems is the **Wizard-of-Oz system**. In wizard systems, the users interact with what they think is a software agent but is in fact a human "wizard" disguised by a software interface (Gould et al. 1983, Good et al. 1984, Fraser and Gilbert 1991). The name comes from the children's book *The Wizard of Oz* (Baum, 1900), in which the Wizard turned out to be just a simulation controlled by a man behind a curtain or screen.



A Wizard-of-Oz system can be used to test out an architecture before implementation; only the interface software and databases need to be in place. The wizard gets input from the user, has a graphical interface to a database to run sample queries based on the user utterance, and then has a way to output sentences, either by typing them or by some combination of selecting from a menu and typing. The wizard's linguistic output can be disguised by a text-to-speech system or, more frequently, by using text-only interactions.

The results of a wizard-of-oz system can also be used as training data to training a pilot dialog system. While wizard-of-oz systems are very commonly used, they are not a perfect simulation; it is difficult for the wizard to exactly simulate the errors, limitations, or time constraints of a real system; results of wizard studies are thus somewhat idealized, but still can provide a useful first idea of the domain issues.

3. Iteratively test the design on users: An iterative design cycle with embedded user testing is essential in system design (Nielsen 1992, Cole et al. 1997, Yankelovich et al. 1995, Landauer 1995). For example in a famous anecdote in dialog design his-

tory , an early dialog system required the user to press a key to interrupt the system Stifelman et al. (1993). But user testing showed users barged in, which led to a redesign of the system to recognize overlapped speech. The iterative method is also important for designing prompts that cause the user to respond in normative ways.

There are a number of good books on conversational interface design (Cohen et al. 2004, Harris 2005, Pearl 2017).

24.5.1 Ethical Issues in Dialog System Design

Ethical issues have long been understood to be crucial in the design of artificial agents, predating the conversational agent itself. Mary Shelley's classic discussion of the problems of creating agents without a consideration of ethical and humanistic concerns lies at the heart of her novel *Frankenstein*. One important ethical issue has to do with bias. As we discussed in Section **??**, machine learning systems of any kind tend to replicate biases that occurred in the training data. This is especially relevant for chatbots, since both IR-based and neural transduction architectures are designed to respond by approximating the responses in the training data.

Tay

A well-publicized instance of this occurred with Microsoft's 2016 **Tay** chatbot, which was taken offline 16 hours after it went live, when it began posting messages with racial slurs, conspiracy theories, and personal attacks. Tay had learned these biases and actions from its training data, including from users who seemed to be purposely teaching it to repeat this kind of language (Neff and Nagy, 2016).

Henderson et al. (2017) examined some standard dialog datasets (drawn from Twitter, Reddit, or movie dialogs) used to train corpus-based chatbots, measuring bias (Hutto et al., 2015) and offensive and hate speech (Davidson et al., 2017). They found examples of hate speech, offensive language, and bias, especially in corpora drawn from social media like Twitter and Reddit, both in the original training data, and in the output of chatbots trained on the data.

Another important ethical issue is privacy. Already in the first days of ELIZA, Weizenbaum pointed out the privacy implications of people's revelations to the chatbot. Henderson et al. (2017) point out that home dialogue agents may accidentally record a user revealing private information (e.g. "Computer, turn on the lights –answers the phone –Hi, yes, my password is..."), which may then be used to train a conversational model. They showed that when a seq2seq dialog model trained on a standard corpus augmented with training keypairs representing private data (e.g. the keyphrase "social security number" followed by a number), an adversary who gave the keyphrase was able to recover the secret information with nearly 100% accuracy.

Finally, chatbots raise important issues of gender equality. Current chatbots are overwhelmingly given female names, likely perpetuating the stereotype of a subservient female servant (Paolino, 2017). And when users use sexually harassing language, most commercial chatbots evade or give positive responses rather than responding in clear negative ways (Fessler, 2017).

24.6 Summary

Conversational agents are a crucial speech and language processing application that are already widely used commercially.

- Chatbots are conversational agents designed to mimic the appearance of informal human conversation. Rule-based chatbots like ELIZA and its modern descendants use rules to map user sentences into system responses. Corpus-based chatbots mine logs of human conversation to learn to automatically map user sentences into system responses.
- For task-based dialog, most commercial dialog systems use the GUS or frame-based architecture, in which the designer specifies a **domain ontology**, a set of frames of information that the system is designed to acquire from the user, each consisting of slots with typed fillers
- A number of commercial systems allow developers to implement simple frame-based dialog systems, such as the user-definable skills in Amazon Alexa or the actions in Google Assistant. VoiceXML is a simple declarative language that has similar capabilities to each of them for specifying deterministic frame-based dialog systems.
- Dialog systems are a kind of human-computer interaction, and general HCI principles apply in their design, including the role of the user, simulations such as Wizard-of-Oz systems, and the importance of iterative design and testing on real users.

Bibliographical and Historical Notes

The earliest conversational systems were chatbots like ELIZA (Weizenbaum, 1966) and PARRY (Colby et al., 1971). ELIZA had a widespread influence on popular perceptions of artificial intelligence, and brought up some of the first ethical questions in natural language processing —such as the issues of privacy we discussed above as well the role of algorithms in decision-making— leading its creator Joseph Weizenbaum to fight for social responsibility in AI and computer science in general.

Another early system, the GUS system (Bobrow et al., 1977) had by the late 1970s established the main frame-based paradigm that became the dominant industrial paradigm for dialog systems for over 30 years.

In the 1990s, stochastic models that had first been applied to natural language understanding began to be applied to dialog slot filling (Miller et al. 1994, Pieraccini et al. 1991).

By around 2010 the GUS architecture finally began to be widely used commercially in phone-based dialog systems like Apple's SIRI (Bellegarda, 2013) and other digital assistants.

The rise of the web and online chatbots brought new interest in chatbots and gave rise to corpus-based chatbot architectures around the turn of the century, first using information retrieval models and then in the 2010s, after the rise of deep learning, with sequence-to-sequence models.

Exercises

24.1 Write a finite-state automaton for a dialogue manager for checking your bank balance and withdrawing money at an automated teller machine.

dispreferred response

24.2 A **dispreferred response** is a response that has the potential to make a person uncomfortable or embarrassed in the conversational context; the most common example dispreferred responses is turning down a request. People signal their discomfort with having to say no with surface cues (like the word *well*), or via significant silence. Try to notice the next time you or someone else utters a dispreferred response, and write down the utterance. What are some other cues in the response that a system might use to detect a dispreferred response? Consider non-verbal cues like eye gaze and body gestures.

24.3 When asked a question to which they aren't sure they know the answer, people display their lack of confidence by cues that resemble other dispreferred responses. Try to notice some unsure answers to questions. What are some of the cues? If you have trouble doing this, read Smith and Clark (1993) and listen specifically for the cues they mention.

24.4 Build a VoiceXML dialogue system for giving the current time around the world. The system should ask the user for a city and a time format (24 hour, etc) and should return the current time, properly dealing with time zones.

24.5 Implement a small air-travel help system based on text input. Your system should get constraints from users about a particular flight that they want to take, expressed in natural language, and display possible flights on a screen. Make simplifying assumptions. You may build in a simple flight database or you may use a flight information system on the Web as your backend.

24.6 Augment your previous system to work with speech input through VoiceXML. (Or alternatively, describe the user interface changes you would have to make for it to work via speech over the phone.) What were the major differences?

24.7 Design a simple dialogue system for checking your email over the telephone. Implement in VoiceXML.

24.8 Test your email-reading system on some potential users. Choose some of the metrics described in Section 24.4 and evaluate your system.

Artstein, R., Gandhe, S., Gerten, J., Leuski, A., and Traum, D. (2009). Semi-formal evaluation of conversational characters. In *Languages: From Formal to Natural*, pp. 22–35. Springer.

Baum, L. F. (1900). *The Wizard of Oz*. Available at Project Gutenberg.

Bellegarda, J. R. (2013). Natural language technology in mobile devices: Two grounding frameworks. In *Mobile Speech and Advanced Natural Language Solutions*, pp. 185–196. Springer.

Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS, A frame driven dialog system. *Artificial Intelligence*, *8*, 155–173.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2016). Generating sentences from a continuous space. In *CoNLL-16*, pp. 10–21.

Carpenter, R. (2017). Cleverbot. http://www.cleverbot.com, accessed 2017.

Cohen, M. H., Giangola, J. P., and Balogh, J. (2004). *Voice User Interface Design*. Addison-Wesley.

Colby, K. M., Hilf, F. D., Weber, S., and Kraemer, H. C. (1972). Turing-like indistinguishability tests for the validation of a computer simulation of paranoid processes. *Artificial Intelligence*, *3*, 199–221.

Colby, K. M., Weber, S., and Hilf, F. D. (1971). Artificial paranoia. *Artificial Intelligence*, *2*(1), 1–25.

Cole, R. A., Novick, D. G., Vermeulen, P. J. E., Sutton, S., Fanty, M., Wessels, L. F. A., de Villiers, J. H., Schalkwyk, J., Hansen, B., and Burnett, D. (1997). Experiments with a spoken dialogue system for taking the US census. *Speech Communication*, *23*, 243–260.

Danieli, M. and Gerbino, E. (1995). Metrics for evaluating dialogue strategies in a spoken language system. In *Proceedings of the 1995 AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation*, Stanford, CA, pp. 34–39. AAAI Press.

Davidson, T., Warmsley, D., Macy, M., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. In *ICWSM 2017*.

Fessler, L. (2017). We tested bots like Siri and Alexa to see who would stand up to sexual harassment. In *Quartz*. Feb 22, 2017. https://qz.com/911681/.

Forbes-Riley, K. and Litman, D. J. (2011). Benefits and challenges of real-time uncertainty detection and adaptation in a spoken dialogue computer tutor. *Speech Communication*, *53*(9), 1115–1136.

Forchini, P. (2013). Using movie corpora to explore spoken American English: Evidence from multi-dimensional analysis. In Bamford, J., Cavalieri, S., and Diani, G. (Eds.), *Variation and Change in Spoken and Written Discourse: Perspectives from corpus linguistics*, pp. 123–136. Benjamins.

Fraser, N. M. and Gilbert, G. N. (1991). Simulating speech systems. *Computer Speech and Language*, *5*, 81–99.

Good, M. D., Whiteside, J. A., Wixon, D. R., and Jones, S. J. (1984). Building a user-derived interface. *Communications of the ACM*, *27*(10), 1032–1043.

Gould, J. D., Conti, J., and Hovanyecz, T. (1983). Composing letters with a simulated listening typewriter. *Communications of the ACM*, *26*(4), 295–308.

Gould, J. D. and Lewis, C. (1985). Designing for usability: Key principles and what designers think. *Communications of the ACM*, *28*(3), 300–311.

Guindon, R. (1988). A multidisciplinary perspective on dialogue structure in user-advisor dialogues. In Guindon, R. (Ed.), *Cognitive Science and Its Applications for Human-Computer Interaction*, pp. 163–200. Lawrence Erlbaum.

Hakkani-Tür, D., Tür, G., Celikyilmaz, A., Chen, Y.-N., Gao, J., Deng, L., and Wang, Y.-Y. (2016). Multi-domain joint semantic frame parsing using bi-directional rnn-lstm.. In *INTERSPEECH*, pp. 715–719.

Harris, R. A. (2005). *Voice Interaction Design: Crafting the New Conversational Speech Systems*. Morgan Kaufmann.

Henderson, P., Sinha, K., Angelard-Gontier, N., Ke, N. R., Fried, G., Lowe, R., and Pineau, J. (2017). Ethical challenges in data-driven dialogue systems. In *AAAI/ACM AI Ethics and Society Conference*.

Hirschman, L. and Pao, C. (1993). The cost of errors in a spoken language system. In *EUROSPEECH-93*, pp. 1419–1422.

Hutto, C. J., Folds, D., and Appling, S. (2015). Computationally detecting and quantifying the degree of bias in sentence-level text of news stories. In *HUSO 2015: The First International Conference on Human and Social Analytics*.

Isbell, C. L., Kearns, M., Kormann, D., Singh, S., and Stone, P. (2000). Cobot in LambdaMOO: A social statistics agent. In *AAAI/IAAI*, pp. 36–41.

Jafarpour, S., Burges, C. J. C., and Ritter, A. (2009). Filter, rank, and transfer the knowledge: Learning to chat. In *NIPS Workshop on Advances in Ranking*, Vancouver, Canada.

Kannan, A. and Vinyals, O. (2016). Adversarial evaluation of dialogue models. In *NIPS 2016 Workshop on Adversarial Training*.

Landauer, T. K. (Ed.). (1995). *The Trouble with Computers: Usefulness, Usability, and Productivity*. MIT Press.

Leuski, A. and Traum, D. (2011). NPCEditor: Creating virtual human dialogue using information retrieval techniques. *AI Magazine*, *32*(2), 42–56.

Li, J., Galley, M., Brockett, C., Gao, J., and Dolan, B. (2016a). A diversity-promoting objective function for neural conversation models. In *NAACL HLT 2016*.

Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. (2016b). Deep reinforcement learning for dialogue generation. In *EMNLP 2016*.

Li, J., Monroe, W., Shi, T., Ritter, A., and Jurafsky, D. (2017). Adversarial learning for neural dialogue generation. In *EMNLP 2017*.

Liu, C.-W., Lowe, R. T., Serban, I. V., Noseworthy, M., Charlin, L., and Pineau, J. (2016). How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *EMNLP 2016*.

Lowe, R. T., Noseworthy, M., Serban, I. V., Angelard-Gontier, N., Bengio, Y., and Pineau, J. (2017a). Towards an automatic Turing test: Learning to evaluate dialogue responses. In *ACL 2017*.

Lowe, R. T., Pow, N., Serban, I. V., Charlin, L., Liu, C.-W., and Pineau, J. (2017b). Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue & Discourse*, *8*(1), 31–65.

Mesnil, G., Dauphin, Y., Yao, K., Bengio, Y., Deng, L., Hakkani-Tür, D., He, X., Heck, L., Tür, G., Yu, D., and Zweig, G. (2015). Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, *23*(3), 530–539.

Microsoft (2014). http://www.msxiaoice.com..

Miller, S., Bobrow, R. J., Ingria, R., and Schwartz, R. (1994). Hidden understanding models of natural language. In *ACL-94*, Las Cruces, NM, pp. 25–32.

Neff, G. and Nagy, P. (2016). Talking to bots: Symbiotic agency and the case of Tay. *International Journal of Communication*, *10*, 4915–4931.

Nielsen, J. (1992). The usability engineering life cycle. *IEEE Computer*, *25*(3), 12–22.

Paolino, J. (2017). Google Home vs Alexa: Two simple user experience design gestures that delighted a female user. In *Medium*. Jan 4, 2017. https://medium.com/startup-grind/google-home-vs-alexa-56e26f69ac77.

Pearl, C. (2017). *Designing Voice User Interfaces: Principles of Conversational Experiences*. O'Reilly.

Pieraccini, R., Levin, E., and Lee, C.-H. (1991). Stochastic representation of conceptual structure in the ATIS task. In *Proceedings DARPA Speech and Natural Language Workshop*, Pacific Grove, CA, pp. 121–124.

Polifroni, J., Hirschman, L., Seneff, S., and Zue, V. W. (1992). Experiments in evaluating interactive spoken language systems. In *Proceedings DARPA Speech and Natural Language Workshop*, Harriman, NY, pp. 28–33.

Raymond, C. and Riccardi, G. (2007). Generative and discriminative algorithms for spoken language understanding. In *INTERSPEECH-07*, pp. 1605–1608.

Ritter, A., Cherry, C., and Dolan, B. (2011). Data-driven response generation in social media. In *EMNLP-11*, pp. 583–593.

Serban, I. V., Lowe, R. T., Charlin, L., and Pineau, J. (2017). A survey of available corpora for building data-driven dialogue systems.. arXiv preprint arXiv:1512.05742.

Shang, L., Lu, Z., and Li, H. (2015). Neural responding machine for short-text conversation. In *ACL 2015*, pp. 1577–1586.

Smith, V. L. and Clark, H. H. (1993). On the course of answering questions. *Journal of Memory and Language*, *32*, 25–38.

Sordoni, A., Galley, M., Auli, M., Brockett, C., Ji, Y., Mitchell, M., Nie, J.-Y., Gao, J., and Dolan, B. (2015). A neural network approach to context-sensitive generation of conversational responses. In *NAACL HLT 2015*, pp. 196–205.

Stifelman, L. J., Arons, B., Schmandt, C., and Hulteen, E. A. (1993). VoiceNotes: A speech interface for a hand-held voice notetaker. In *Human Factors in Computing Systems: INTERCHI '93 Conference Proceedings*, pp. 179–186.

Suendermann, D., Evanini, K., Liscombe, J., Hunter, P., Dayanidhi, K., and Pieraccini, R. (2009). From rule-based to statistical grammars: Continuous improvement of large-scale spoken dialog systems. In *ICASSP-09*, pp. 4713–4716.

VanLehn, K., Jordan, P. W., Rosé, C., Bhembe, D., Böttner, M., Gaydos, A., Makatchev, M., Pappuswamy, U., Ringenberg, M., Roque, A., Siler, S., Srivastava, R., and Wilson, R. (2002). The architecture of Why2-Atlas: A coach for qualitative physics essay writing. In *Proc. Intelligent Tutoring Systems*.

Vinyals, O. and Le, Q. (2015). A neural conversational model. In *Proceedings of ICML Deep Learning Workshop*, Lille, France.

Walker, M. A., Kamm, C. A., and Litman, D. J. (2001). Towards developing general models of usability with PARADISE. *Natural Language Engineering: Special Issue on Best Practice in Spoken Dialogue Systems*, *6*(3), 363–377.

Walker, M. A. and Whittaker, S. (1990). Mixed initiative in dialogue: An investigation into discourse segmentation. In *ACL-90*, Pittsburgh, PA, pp. 70–78.

Wang, H., Lu, Z., Li, H., and Chen, E. (2013). A dataset for research on short-text conversations.. In *EMNLP 2013*, pp. 935–945.

Ward, W. and Issar, S. (1994). Recent improvements in the CMU spoken language understanding system. In *ARPA Human Language Technologies Workshop*, Plainsboro, N.J.

Weizenbaum, J. (1966). ELIZA – A computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45.

Yan, Z., Duan, N., Bao, J.-W., Chen, P., Zhou, M., Li, Z., and Zhou, J. (2016). DocChat: An information retrieval approach for chatbot engines using unstructured documents. In *ACL 2016*.

Yankelovich, N., Levow, G.-A., and Marx, M. (1995). Designing SpeechActs: Issues in speech user interfaces. In *Human Factors in Computing Systems: CHI '95 Conference Proceedings*, Denver, CO, pp. 369–376.

Zue, V. W., Glass, J., Goodine, D., Leung, H., Phillips, M., Polifroni, J., and Seneff, S. (1989). Preliminary evaluation of the VOYAGER spoken language system. In *Proceedings DARPA Speech and Natural Language Workshop*, Cape Cod, MA, pp. 160–167.