*Chapter 2*

# *My Inner Workings*

Before diving into the detail, it will be helpful if I give you an overview of my main processing functions and the way that they operate. As you will see, there is quite a lot happening between the point when Steve speaks to me and the point at which I respond.

## 2.1 *Anatomy of a Conversation*

Just a minute ...

---

Hey Cyba, set a timer for my egg.
*Hi Steve, how long?*
Four minutes.
*Egg timer is set.*

---

The ability to set timers is one of the many services available on Steve's personal devices. Once set, a timer will sound a buzzer automatically when the prescribed duration has elapsed. This is a particularly simple service, nevertheless, I access it in exactly the same way as all of the services available to him. So this short exchange provides a good introduction to the goal-oriented conversations that underpin the services that I provide.

Let's go through this conversation step by step. I have summarised the main processing flow in Fig. 2.1. This diagram is quite detailed, so let me first explain the overall structure. When Steve speaks to me, I convert the

https://doi.org/10.1017/9781108976718.003 Published online by Cambridge University Press

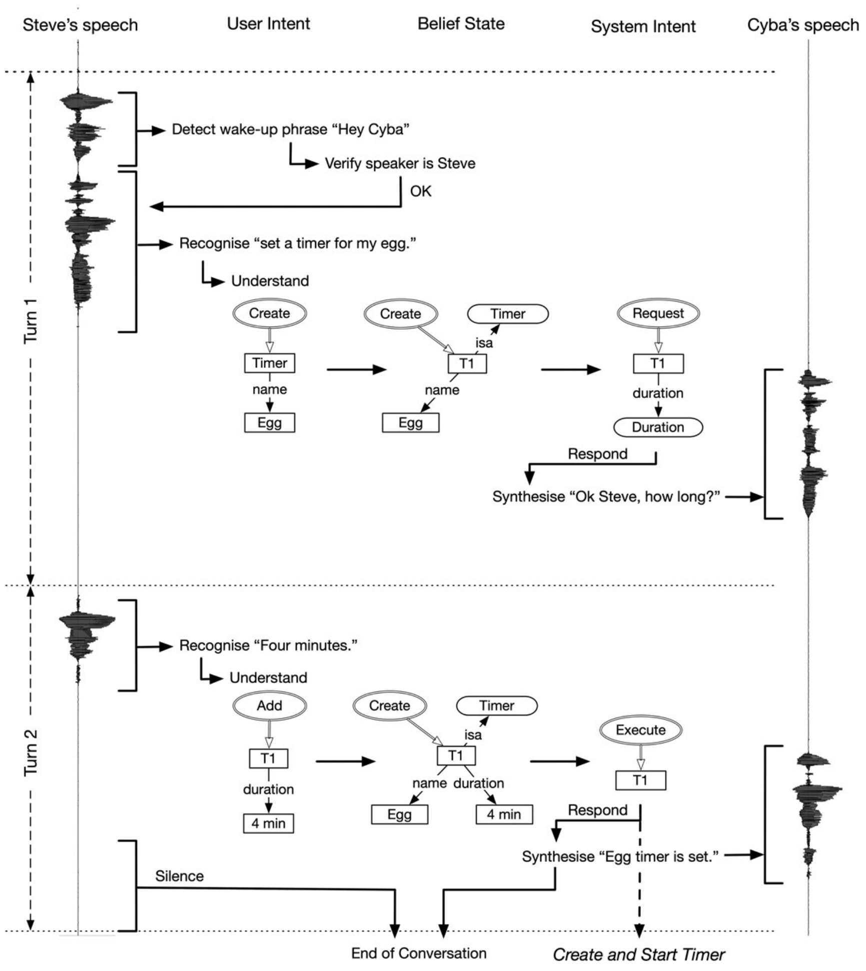

Figure 2.1 *Overview of processing a two-turn conversation to set an egg timer.*

speech signal corresponding to his voice into words and then I extract the meaning to form an internal representation called a *user intent*. This user intent is assimilated into a data structure called a *belief state* which represents my current understanding of Steve's goal. In every turn of a conversation, the belief state determines what I should do next to complete the goal. This usually involves generating a response, which I do by composing a *system intent*, converting it into words and then outputting it as a speech waveform.

In Fig. 2.1, these processing stages are arranged in columns. Down the far left are the incoming speech waveforms from Steve. The second column contains the recognised words and user intents. The central column shows the evolving belief state. The fourth column contains the system intents and my corresponding responses, and the final column shows the synthesised speech waveforms that constitute my voice.

Time progresses from the top of Fig. 2.1 down to the bottom. Before the conversation started, I was sitting listening for Steve to say "Hey Cyba" which is my so-called *wake-up phrase*. When I am in this idle state, I am continuously monitoring my input audio channel, but I do not save the audio or attempt to recognise the words in it until triggered by the wake-up phrase. Simply recognising the words in the wake-up phrase is still not sufficient to enable conversation processing. I also need to verify that the person speaking to me is Steve, otherwise an imposter might be able to access confidential information such as Steve's messages or perform an unauthorised transaction such as transferring cash or goods to a third party. So whenever I hear "Hey Cyba", I pass the same segment of audio to a speaker verification component which is designed to distinguish Steve from any other speaker. You can see these functions being performed at the top of the flow diagram in Fig. 2.1. Once a verified wake-up phrase has been detected, I enable conversation processing. From that point, I recognise everything that is said to me until the end of the conversation has been detected so that Steve does not have to keep on repeating the wake-up phrase at every turn of the conversation.

As soon as the conversation was enabled, the first thing that I recognised was "set a timer for my egg". The timer service allows multiple timers to be created, each with a name, and my understanding component decoded Steve's words as expressing the intent *Create(Timer)* with name "Egg". This create intent caused a new belief state to be constructed, containing a new instance of a *Timer* called, arbitrarily, T1 with *name* Egg. I'll explain all of the notation and how this works in detail later, but for now the key thing to understand is that I have a store of pre-defined information called a *knowledge graph*. Included in this knowledge graph are the definitions of various *types*. A type is a template for creating instances of entities such as times, meetings, contacts, etc. Each type describes the property values that any instance of the type should have. In this case, the *Timer* type tells me that a timer must have a duration. So I generated a *Request* intent for the duration, resulting in my response "Ok, Steve, how long?" Turn 1 was then finished and I waited for Steve to reply.

Turn 2 started with Steve responding "Four minutes". My understanding component knew that this was the answer to a *Request* intent, so it interpreted Steve's response as being an intent to *Add* the property *duration* 4 minutes to *T1*. This caused the *duration* property to be added and the belief state updated. I then knew that the timer instance was complete and furthermore I was confident that I had recognised Steve's speech accurately. If I had been unsure then I might have explicitly confirmed that I had properly understood Steve's goal by responding with something like "Setting a timer called Egg for 4 minutes, ok?" before proceeding. However, since I was confident I decided to go ahead and execute the goal immediately by actually creating the timer instance and confirming back to Steve that the timer was set.

This last part needs a little more explanation. In very broad terms, there are two kinds of goals. Firstly there are information seeking goals such as "How many times did Björn Borg win at Wimbledon". Storing the factual information needed to answer queries like these is the traditional role of a knowledge graph. Secondly, there are service-oriented goals such as setting a timer or ordering a taxi. These are typically accessed via a special interface called an application programming interface or API, and different services will have different APIs. Because information and service goals must be executed differently, many of my cousins keep them distinct. In contrast, I have a specially designed knowledge graph which provides a uniform interface to all knowledge and services. To invoke a service, I simply create an instance of the associated type and some clever software wizardry ensures that the appropriate API functions are automatically invoked behind the scenes. The advantage of this arrangement is that all of my conversations can be managed in exactly the same way. Hang on …

Hey Cyba, how's the egg timer going?
*3.2 of 4 minutes have elapsed.*

In this case, I was able to answer Steve's query because I knew that the *Timer* type has a property called *timerStatus* which I could access in

https://doi.org/10.1017/9781108976718.003 Published online by Cambridge University Press

exactly the same way that I would answer a knowledge question such as "How old is Brad Pitt?" by reading the *age* property of Brad Pitt's instance in my knowledge graph.

Finally, a conversation ends when two conditions are met: Steve is silent, and my belief state tells me that there are no further goals pending. Both of these conditions were met, so I disabled the audio feed to my speech recogniser, I restarted my wake-up phrase detector and I returned to my idle state.

## 2.2 My Working Parts

The conversation to set a timer illustrates the main processing functions that are needed in order to handle a request from Steve. In the remainder of this book I will explain in more detail how they work. Figure 2.2 shows my major processing units and how they connect together. The numbers in this diagram refer to the chapter in which they are described.

The key stages of language processing form a pipeline: speech recognition, understanding, conversation management and speech synthesis. When Steve speaks to me I use speech recognition to convert the speech waveform into words and I use spoken language understanding (SLU) to convert the words into intents. During the course of a conversation, these intents convey the information needed to understand Steve's goal. However, the speech recognition and SLU components can make errors and Steve may omit important information. At each turn of the conversation, I record my current understanding of what Steve wants and the things that I am uncertain about in my belief state. At each turn, I have choices to make. Should I confirm information already given, should I ask for further information or should I assume that my beliefs are correct and complete, and execute Steve's goal? It is the task of conversation management to make these decisions, refine my belief state and orchestrate my responses. Finally, I use speech synthesis to convert these responses back into speech.

My ability to understand and execute the goals that Steve sets for me depends crucially on my knowledge graph, which stores everything I know, including general facts about the world, personal data such as Steve's contacts, his calendar, music and video library, personal memories and recent conversations. My knowledge graph sits at the heart of my

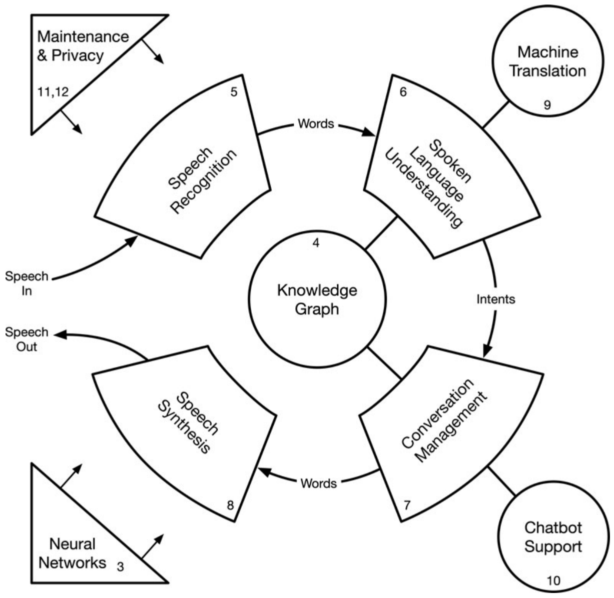

Figure 2.2 *A map of my inner workings - the numbers refer to chapters.*

system, with direct connections to spoken language understanding and conversation management.

The above components constitute my core functionality. However, I also have a few ancillary functions such as chatting about a wide variety of topics, and translating between my working language, which is English, and 28 other languages. I also have a few tricks such as recognising barcodes and hand-writing from digital images which I will mention along the way.

Finally, everything that I do is constrained by the need to maintain security, trust and privacy, and my overall performance depends critically on various maintenance functions that I perform during periods of inactivity such as updating my knowledge graph and other system components. My tour would not be complete if I did not also discuss these issues.

Underpinning virtually all of the functional units in Fig. 2.2 is the ability to recognise patterns in data. For example, I need to recognise sound patterns in speech waves, letter sequences in words, and word sequences in sentences. Remarkably, the same basic pattern processing component can be applied to all of these problems, and that component is the neural network whose development over the last decade has revolutionised the way in which conversational agents like myself are built. So let's prepare for our tour of my inner workings by first examining the fundamentals of how my "brain" works.

https://doi.org/10.1017/9781108976718.003 Published online by Cambridge University Press