Chapter 7
Creating a Conversational Interface Using Chatbot Technology

Abstract Conversational interfaces can be built using a variety of technologies. This chapter shows how to create a conversational interface using chatbot technology in which pattern matching is used to interpret the user's input and templates are used to provide the system's output. Numerous conversational interfaces have been built in this way, initially to develop systems that could engage in conversation in a human-like way but also more recently to create automated online assistants to complement or even replace human-provided services in call centers. In this chapter, some working examples of conversational interfaces using the Pandorabots platform are presented, along with a tutorial on AIML, a markup language for specifying conversational interactions.

7.1 Introduction

In Chap. 6, we showed how to add speech input and output to a mobile app using the Google Speech APIs. However, speech input and output are only one part of the tasks that we might require from a conversational interface. Our query might be about the weather in London or for directions to the nearest Starbucks. We will want our query to be interpreted by the conversational interface as a request to answer a question or to carry out some action. We will also want the app to respond with something related to what we asked for, such as a spoken answer to our question about the weather or a display of information such as a map with the requested directions.

Consider once again the components of a spoken language-based conversational interface that we described in Chap. 2 (Fig. 7.1).

As we can see, once the user's input has been recognized by the speech recognition component, it has to be interpreted in order to determine its meaning. In some approaches, this might involve a thorough analysis of the input using techniques from spoken language understanding (SLU)—for example, a grammar to represent the permissible inputs and a parser to apply the grammar to the input and to extract a semantic representation. Then, the dialog manager has to decide what actions to

M. McTear et al., *The Conversational Interface*,
DOI 10.1007/978-3-319-32967-3_7

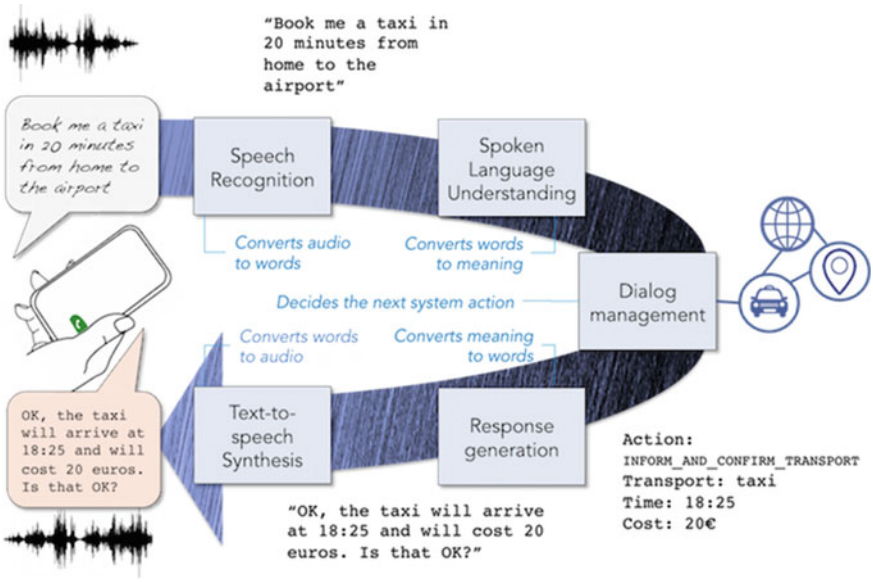

Fig. 7.1 The components of a spoken language-based conversational interface

take, using as a basis this semantic representation and other relevant information such as the current context. This will result in some output being generated—words to be spoken and possibly output in other modalities, such as images, lists, and maps. For these tasks, additional components are required, in particular: SLU (see Chaps. 8 and 9), dialog management (DM) (see Chaps. 10 and 11), and response generation (RG) (see Chap. 12).

In this chapter, we will present a simpler approach that has been widely applied in chatbot technology, as illustrated in Fig. 7.2. In this approach, the input is matched against a large store of possible inputs (or patterns) and an associated response is outputted. The chatbot approach was first used in the ELIZA system (Weizenbaum 1966) and has continued until the present day in the form of apps that provide an illusion of conversation with a human as well as in areas such as education, information retrieval, business, and e-commerce, for example, as automated online assistants to complement or even replace human-provided services in a call center (see further Chap. 4).

More recently, chatbot technology has been extended to support the development and deployment of virtual personal assistants by incorporating methods for interpreting commands to the device or queries to Web services—for example, to search for information on the Internet, access information on the device, such as contacts and calendars, perform a task on the device such as launching an app, setting an alarm, or placing a call.

We will use the Pandorabots platform, a popular Web service that enables developers to create and host chatbots, to show how a conversational interface can be created with chatbot technology. We first introduce Pandorabots and then

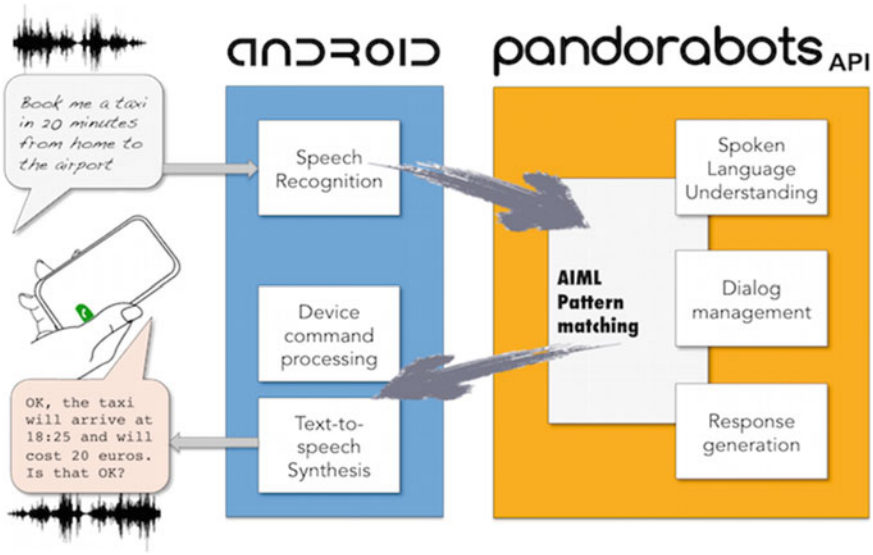

Fig. 7.2 Using Pandorabots for spoken language understanding, dialog management, and response generation

provide a brief overview of AIML (Artificial Intelligence Markup Language), which is used to specify conversations for a chatbot on the Pandorabots platform. Following this, we will show how to embed a Pandorabots chatbot in an Android app and how to provide speech input and output. We will then show how the chatbot can be extended to handle commands to the device and queries to Web services. In the final section, we will show how this approach can be further extended.

The code corresponding to the examples in this chapter is in GitHub, in the folder *chapter7* of the ConversationalInterface[1] repository.

7.2 Introducing the Pandorabots Platform

Pandorabots is a bot-hosting service launched in 2002 that enables chatbot developers (referred to in Pandorabots as botmasters) to develop, test, and deploy chatbots (or more simply bots) without requiring a background in programming.[2] AIML was developed by Dr. Richard Wallace as a language for specifying conversations with chatbots and was used by Wallace to develop the chatbot ALICE

[1] http://zoraidacallejas.github.io/ConversationalInterface/. Accessed March 2, 2016.

[2] http://www.pandorabots.com/. Accessed February 20, 2016.

```
<category>

<pattern> WHAT ARE YOU </pattern>

<template>

I am the latest result in artificial intelligence, which can
reproduce the capabilities of the human brain with greater
speed and accuracy.

</template>

</category>
```

Code 7.1 AIML category for "What are you?"

(Artificial Linguistic Internet Computer Entity) which won the Loebner Prize in 2000, 2001, and 2004. The Loebner prize is awarded to the chatbot that in an annual competition is considered by judges to be the most human-like. Other award winning bots developed using AIML include Mitsuki, Tutor, Izar, Zoe, and Professor. Currently more than 221,000 chatbots in many languages are hosted on the platform. The platform has recently been revamped so that in addition to the original chatbot-hosting server there are now facilities on a Developers Portal to support the deployment of chatbots on the Web and on mobile devices.

Many chatbots that are currently available on mobile devices were created using Pandorabots and AIML. These include the following: Voice Actions by Pannous (also known as Jeannie), Skyvi, Iris, English Tutor, BackTalk, Otter, and Pandorabot's own CallMom app. CallMom can perform the same sorts of tasks as other chatbots but also includes a learning feature so that it can learn personal preferences and contacts and can be taught to correct speech recognition errors. More information about Pandorabots and chatbots in general can be found at the ALICE A.I. Foundation site.[3] See also the Chatbots.org website.[4]

As mentioned in the previous section, in order to simulate conversation, chatbot technology makes use of pattern matching in which the user's input is matched against a large set of stored patterns and a response is output that is associated with the matched pattern. The technique was first used in the ELIZA system and has been deployed in subsequent chatbots ever since. Authoring a chatbot on Pandorabots involves creating a large number of AIML categories that at their most basic level consist of a pattern against which the user's input is matched and an associated template that specifies the chatbot's response. Code 7.1 is a simple example of an AIML category.

[3] http://www.alicebot.org/. Accessed February 20, 2016.

[4] https://www.chatbots.org/. Accessed February 20, 2016.

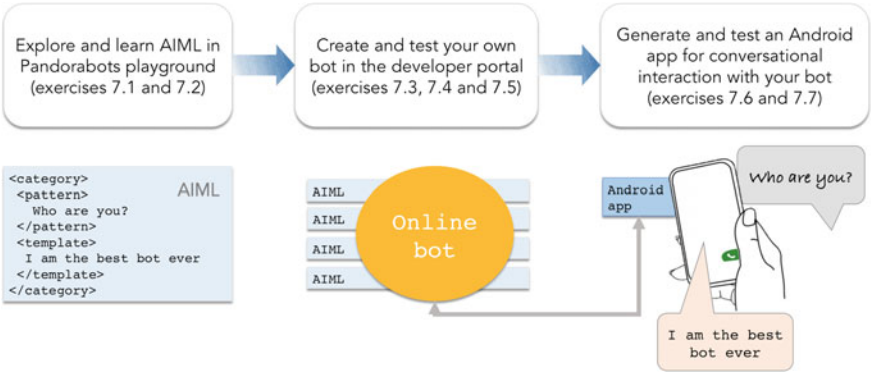

Fig. 7.3 The steps required to generate a bot

In this example, if the user's input is matched against the text in the pattern, the system executes the contents of the template. Matching in AIML is done by the Graphmaster, which stores all the patterns in a graph. The graph consists of a collection of nodes called Nodemappers that map the branches from each node. The root of the Graphmaster is a Nodemapper with a large number of branches, one for each of the first words of the patterns. In the case of ALICE, there were around 40,000 patterns with about 2000 different first words (Wallace 2003). Matching involves traversing the graph on a word-by-word basis from the root to a terminal node. Interestingly, this process bears some similarity to the process of incremental processing by humans where an input sentence is analyzed on a word-by-word basis as opposed to waiting for the utterance to be completed (Crocker et al. 1999; see also Chap. 4). More detail about pattern matching in AIML can be found in Wallace (2003) and also at the AliceBot Web page.[5] A formal definition of the Graphmaster and matching is provided in Wallace (2009).

Once a pattern has been matched, the contents of the associated template are output. In the example above, the contents are in the form of text, so the output is a message consisting of that text, but the template can also contain executable code and various tags that can be used to compute more complex and more flexible responses, as will be explained in Sect. 7.6.

In the remainder of this chapter, we will provide a series of exercises showing the steps required to generate a bot (see Fig. 7.3).

Exercise 7.1: Creating a bot in Pandorabots Playground Pandorabots provides a Web-based platform called Playground for testing bots developed in AIML. In this section, we describe how to create an account in Playground and develop your

[5] http://www.alicebot.org/documentation/matching.html. Accessed February 20, 2016.

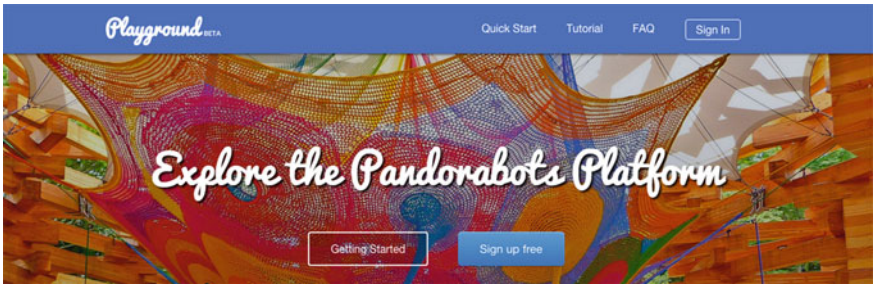

Fig. 7.4 The Playground home page (reproduced with permission from Pandorabots, Inc.)

first bot using a built-in library of AIML files. The AIML files for the bot that we will build in the next sections are in GitHub ConversationalInterface repository in the folder *chapter7/AIML*.

1. To access Playground, go to the Pandorabots Web page[6] and click on the Playground tab, or you can also go directly to the Playground page.[7] Here you can sign up for a 10-day free trial account and log in using your Facebook, Google, Twitter, or Yahoo account. You will find a QuickStart tutorial, a more detailed tutorial, and a tab for FAQs, as shown in Fig. 7.4.
2. Once you have created an account, you can start to create bots, interact with them and develop them further. The following are brief instructions. For more detailed instructions, see the Playground Web site.
3. Click on the My Bots tab to see a list of your bots. (If you have just signed up for an account, the list will be empty.)
4. Click on the Create Bot tab to create a bot. Give it a name, for example, "talkbot."
5. You will have a set of default AIML files.
6. To get started, you can make use of the chatbot base called Rosie, which is a collection of AIML files. Rosie provides conversational interaction. The files in the Rosie chatbot base allow you to get started quickly without having to write any AIML code. This page[8] includes a number of other useful resources. Go to the page and upload the files found under the lib directory.
7. Select your bot. This will bring up the editor screen for your bot (Fig. 7.5).
8. Click on the Train tab. Now you can test your bot by asking questions.

[6] http://www.pandorabots.com/. Accessed February 20, 2016.

[7] https://playground.pandorabots.com/en/. Accessed February 20, 2016.

[8] https://github.com/pandorabots/rosie. Accessed February 20, 2016.

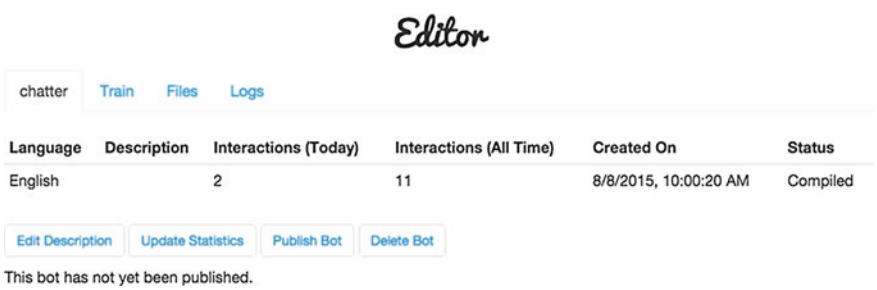

Fig. 7.5 The Playground editor screen (reproduced with permission from Pandorabots, Inc.)

7.3 Developing Your Own Bot Using AIML

As you may have noticed, the chatbot bases provided by Pandorabots cover many of the aspects of conversational speech. However, if you want your bot to be able to respond to questions in a specific domain, you will need to create new AIML code. In this section, we provide a brief tutorial on AIML that we will use as a basis for creating a chatbot that provides answers to frequently asked questions in the domain of type 2 diabetes. A more comprehensive tutorial can be found on the Pandorabots Playground page.

The following are some of the questions that a user might ask:

```
What is type 2 diabetes?
What are the main symptoms of type 2 diabetes?
What is the main cause of type 2 diabetes?
How do you treat type 2 diabetes?
Can type 2 diabetes be cured?
Does exercise help?
Tell me about blood sugar.
What are the long term complications?
```

To be able to answer questions such as these, we need to create categories that will specify the range of inputs and the answers that we wish to associate with them.

7.3.1 Creating Categories

A first step is to collect questions and their answers and to create categories consisting of patterns and templates, for example (Code 7.2):

```
<category>

<pattern>what is Type 2 diabetes</pattern>

<template>type 2 diabetes is a metabolic disorder that is
characterized by high blood glucose in the context of insulin
resistance and relative insulin deficiency</template>

</category>
```

Code 7.2 AIML category for "What is type 2 diabetes?"

```
<category>

<pattern>what are the causes </pattern>

<template>

   <srai>what is the main cause of type 2 diabetes</srai>

</template>

</category>
```

Code 7.3 AIML category for "What is type 2 diabetes?" including an `<srai>` tag

However, it will quickly become clear that there are many ways of asking the same question, for example:

What is the main cause of type 2 diabetes?
What causes type 2 diabetes?
What are the causes of type 2 diabetes?
What is the main cause?
What are the causes?
How do you get type 2 diabetes?

Rather than list all of these synonymous questions and their answers (which in this case would all be the same), AIML makes use of a technique in which synonymous questions are mapped on to one canonical pattern using the `<srai>` tag, as in (Code 7.3).

In this example, the bot's response is retrieved recursively by finding the response in the category that contains the pattern of the canonical pattern `what is the main cause of type 2 diabetes`. However, given the possible combinatorial explosion of synonymous questions, specifying all the required symbolic reductions remains a formidable task.

7.3.2 Wildcards

The use of wildcards also enables similar patterns to be mapped on to one canonical form. For example, the wildcard * matches one or more words in the input, so that the following patterns can be treated as variants at different levels of expressiveness:

What are the main symptoms of type 2 diabetes?
What are the * symptoms of type 2 diabetes?
What are the * symptoms *?

Another wildcard ^ can match zero or more words, so that in the following code, the input "what are the main symptoms" would be matched along with other inputs including words after the word "symptoms" (Code 7.4).

The words captured by a wildcard can be repeated back in the template by using the <star/> tag. For example, in response to a question about symptoms, the human might say, "I feel tired," for which the pattern and template in Code 7.5 would cover any symptom mentioned.

This will allow an interaction such as:

Human: I feel tired.
Bot: When do you feel tired?

There can be more than one wildcard in a pattern, and they can be repeated back in the template by using <star index = "x">, where x corresponds to the position of the wildcard in the input (Code 7.6).

This will allow an interaction such as:

Human: I feel tired and I have a headache.
Bot: So you feel tired and you have a headache.

While wildcards are a useful way of avoiding redundancy in patterns, as in any other system using wildcards there is the potential for false positives, where an input may be matched against an unintended canonical pattern, resulting in

```
<pattern>what are the main symptoms ^ </pattern>
```

Code 7.4 AIML code using the ^ wildcard

```
<category>
<pattern>I feel * </pattern>
<template>When do you feel <star/></template>
</category>
```

Code 7.5 AIML code using the * wildcard and the <star/> tag

```
<category>
  <pattern>I feel * and I * </pattern>
  <template>So you feel <star/> and you
     <star index = "2" /> </template>
</category>
```

Code 7.6 AIML code using the <star/> tag at several positions

unexpected system responses. Looking again at Code 7.5, if the human inputs a time expression, this will be matched and repeated in the bot's response, resulting in an incongruous output:

> Human: I feel tired when I get up in the morning.
> Bot: When do you feel tired when you get up in the morning?

Careful testing of patterns would help to avoid this sort of problem, but it may be difficult to test exhaustively for every possible input and how it would be matched against patterns containing wildcards.

7.3.3 Variables

Variables can be used to set and retrieve information from an interaction that is stored as a predicate, for example (Code 7.7). The value of the medication predicate can be retrieved later using <get>, as in (Code 7.8).

```
<category>

  <pattern>my main medication is * </pattern>

  <template>OK, I have noted that your main medication is

    <set name = "medication"<star/></set> </template>

</category>
```

Code 7.7 AIML code that defines a predicate called "medication"

```
<template>you said that your main medication is <get name =
"medication"/></template>
```

Code 7.8 AIML code that uses the value of a predicate

7.3.4 Sets and Maps

Repetition can be avoided in patterns where a word forms part of a set—for example, all color words can be declared in a set of colors so that separate patterns do not have to be written for inputs such as:

> I like red cars.
> I like blue cars.
> I like green cars.

To create a set, click on the `plus` button of the Files tab of the Editor and select the set type. For example, create a file called `color` and add some colors. The set can then be used in a category containing a pattern as shown in Code 7.9.

This technique can be used to address the problem of lexical alternation, for example, dealing with morphological variants of a word, as in the forms "find," "finding," and "found."

```
<category>

<pattern>I like the color <set>color</set></pattern>

<template>That's interesting, so do I </template>

</category>
```

Code 7.9 An AIML set

Table 7.1 Sets in AIML

State set	State2capital set
Alabama	Alabama:Montgomery
Arizona	Arizona:Phoenix
California	California:Sacramento
…	…

```
<category>
 <pattern>What is the capital of <set>state</set></pattern>
 <template><map name="state2capital"><star/></map></template>
</category>
```

Code 7.10 An AIML map

Maps are used to specify associations between sets. For example, a set could be a list of US states and a map could be a function that associates an element in the set of states with an element in a set of state capitals, as in Table 7.1.

Using maps, it is then possible to use a single category as shown in Code 7.10 to ask and answer questions about any US state, for example:

What is the capital of Alabama?
What is the capital of Arizona?
What is the capital of California?

Pandorabots has some built-in sets and maps for collections such as natural numbers, singular and plural nouns. For further details on sets and maps, see Wallace (2014b).

7.3.5 Context

There are several mechanisms in Pandorabots for dealing with aspects of the conversational context. The simplest of these is the tag <that> which enables the system to remember the last sentence it has said. Consider the following example:

Human: I have problems with my blood sugar.
Bot: is blood sugar your main problem?
Human: yes.
Bot: what medication are you taking for your blood sugar?

```
<category>

 <pattern>^blood sugar^</pattern>

 <template>is blood sugar your main problem </template>

</category>

<category>

 <pattern>yes<pattern>

 <that>is blood sugar your main problem</that>

 <template> what medication are you taking for your blood
sugar </template>

</category>
```

Code 7.11 Dealing with context using the AIML `<that>` tag

This interaction can be specified as in Code 7.11. Here, the second category is bound to a context where it will only be matched if the last sentence spoken by the bot was "Is blood sugar your main problem?".

A more wide-ranging mechanism involves setting a topic. For example, the topic could be set to `blood sugar` and only those patterns within the scope of the topic will be matched. This is illustrated in Code 7.12.

AIML has a number of other features including the ability to learn from information provided by the human interlocutor. For more details, consult the Pandorabots tutorial[9] or the AIML2.0 specification (Wallace 2014a).

One of the main advantages of the pattern-matching approach as implemented in AIML and similar languages is its simplicity. AIML makes use of an efficient matching algorithm that conducts a graph search through the prestored patterns on a word-by-word basis so that responses can be provided in real time. Also, given that the application is typically the simulation of conversation, it is not always critical if the user's input cannot be matched against a pattern. In this case, the ultimate default category is invoked to help keep the conversation going. For example, the chatbot can say something like "tell me more about yourself," which may cause the user to say something that will be matched by a pattern. Indeed, the art of chatbot programming is to provide sufficient responses in the ultimate default category that can be generated randomly and that can help the system recover from failures to match the user's input. This feature is probably not so useful in applications like the diabetes bot that simulate the answering of frequently asked questions in a particular domain.

[9] https://playground.pandorabots.com/en/tutorial/. Accessed February 20, 2016.

```
<topic name = "blood sugar">

<category>

  <pattern>^blood sugar^</pattern>

  <template> blood sugar is a common problem in diabe-
tes</template>

</category>

<category>

  <pattern>how can I lower it<pattern>

  <template> there are various medications that will help you
lower your blood sugar </template>

</category>

<category>

  <pattern>what would you recommend<pattern>

  <template> you should consult your doctor as the medication
to be prescribed will depend on other factors  </template>

</category>
```

Code 7.12 Setting a topic in AIML

Exercise 7.2: Creating the diabetes bot

1. Click on the `Create Bot` tab to create a bot. You can call it "Diabetes."
2. You will have a set of default AIML files. Add a new file called "questions." You can upload a sample file from the ones you will find in the ConversationalInterface repository /chapter7/AIML/.
3. Select your bot. This will bring up the editor screen for your bot, as shown in Fig. 7.6.
4. Click on the `Train` tab. Now you can test your bot by asking questions.

You will soon find that your bot is unable to answer all of your questions, in which case you will need to add more categories. You can edit and further develop the bot by following the more detailed instructions in the Playground tutorials. If you wish to make your bot available to other members within the Clubhouse—a community of other botmasters—to do this you need to click on the tab `Publish Bot`.

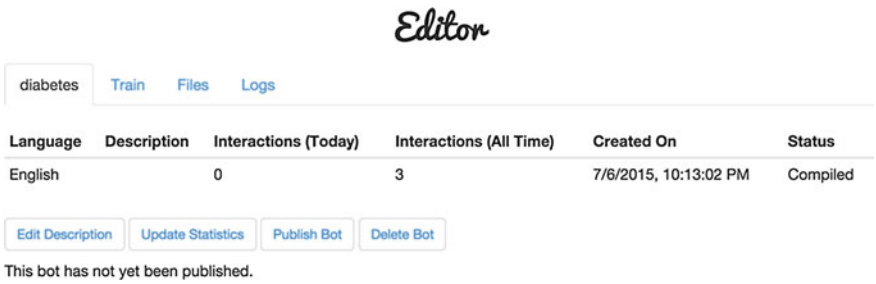

Fig. 7.6 The Playground editor screen for the diabetes bot (reproduced with permission from Pandorabots, Inc.)

7.4 Creating a Link to Pandorabots from Your Android App

Once you have sufficiently developed your bots in Playground, you are ready to deploy them. In this section, we show how to embed a Pandorabots bot into an Android app. There are several reasons why you might wish to embed your bot in an app:

- The Playground only allows you to interact with your bot using the Web interface provided.
- The Playground allows you to make your bot available to other registered bot masters by publishing it in the Clubhouse, but you cannot make it publicly accessible on a Web site or by deploying it as a mobile application. For this, you need to create an account on the Developer Portal.
- By embedding the bot in an Android app, your app can be deployed on an Android mobile phone or tablet and can be made available to others to use, just like any other Android app.
- You can provide a speech-based front end to your app using the Google Speech APIs.
- You can add additional functions, such as making commands to access device functions—for example, to launch an app or to check the time. You can also link to other Web services, such as search and maps.

7.4.1 Creating a Bot in the Developer Portal

Exercise 7.3: Signing up for an account on the Developer Portal The Developer Portal provides all the necessary tools and SDKs for deploying bots anywhere. To sign up to the Developer Portal:

```
$ npm install -g pb-cli
```

Code 7.13 Installing the CLI in node.js

```
pb upload myfile.aiml
```

Code 7.14 Using the pb command to upload a file

1. Click on the Dev Portal tab on the Pandorabots main page or go directly to the Developer Portal.[10]
2. Sign up for an account. You have to register for a plan. Accounts are free for a 10-day trial period after which different plans are available depending on needs. You should try to iron out any problems with your AIML code in Playground before moving your bot over to the Developer Portal. If you find that you need to make more API calls than allowed on your plan, you can upgrade your plan, as required.
3. Once your account and plan have been approved, you can retrieve your `user_key` and `application_id`, which are required in order to make API calls.

Exercise 7.4: Using the APIs to create a bot, upload and compile files, and talk to a bot There are two ways to use the APIs:

1. Using the Pandorabots CLI (command line interface).
2. Using an HTTP client-like cURL.

Both are supported, though the easiest way is to use the Pandorabots CLI.[11]

The CLI can be installed by going to the Developer Portal home page and scrolling down to the section entitled `Getting Started`, which is below the pricing information. Detailed instructions are also provided here.[12]

The CLI is written in Javascript, so it is first necessary to setup node.js before installing the CLI. An installer for `node.js` for both OS X and Windows users is available here.[13]

`Node.js` includes `npm`, a Javascript package manager that you can use to install the CLI using the command (Code 7.13).

This will install the CLI and make the `pb` and `pandorabots` commands available for use in the command line, for example (Code 7.14).

The CLI needs to be configured using a JSON file called `chatbot.json` in order to allow these commands. The `chatbot.json` configuration file stores

[10] https://developer.pandorabots.com/. Accessed February 20, 2016.

[11] https://github.com/pandorabots/pb-cli. Accessed February 20, 2016.

[12] http://blog.pandorabots.com/introducing-the-pandorabots-cli/. Accessed February 20, 2016.

[13] http://nodejs.org/download/. Accessed February 20, 2016.

basic information about your application: the `app_id`, the `user_key`, and the `botname`. When a CLI command is run, all the required information is added from `chatbot.json`. To create this file, you can use the init command and the CLI will prompt for all the required information. First create a directory for `chatbot.json`, then run `init`, as shown in Code 7.15.

This will now allow various commands to be run, such as shown in Code 7.16. A complete list of the commands is available here.[14]

```
$ mkdir mydirectory

$ cd mydirectory

$ pb init

(when information has been added

$ pb create
```

Code 7.15 Configuring the CLI

```
$ pb create

$ pb upload

$ pb compile

$ pb talk
```

Code 7.16 Examples of `pb` commands

```
$ pb upload example.aiml
```

Code 7.17 The `pb` command for uploading a file

```
$ curl -v -X PUT
'https://aiaas.pandorabots.com/bot/APP_ID/BOTNAM
E/file/example.aiml?user_key=USER_KEY'  --data-
binary @home/mybot/example.aiml
```

Code 7.18 The `curl` command for uploading a file

[14] https://github.com/pandorabots/pb-cli. Accessed February 20, 2016.

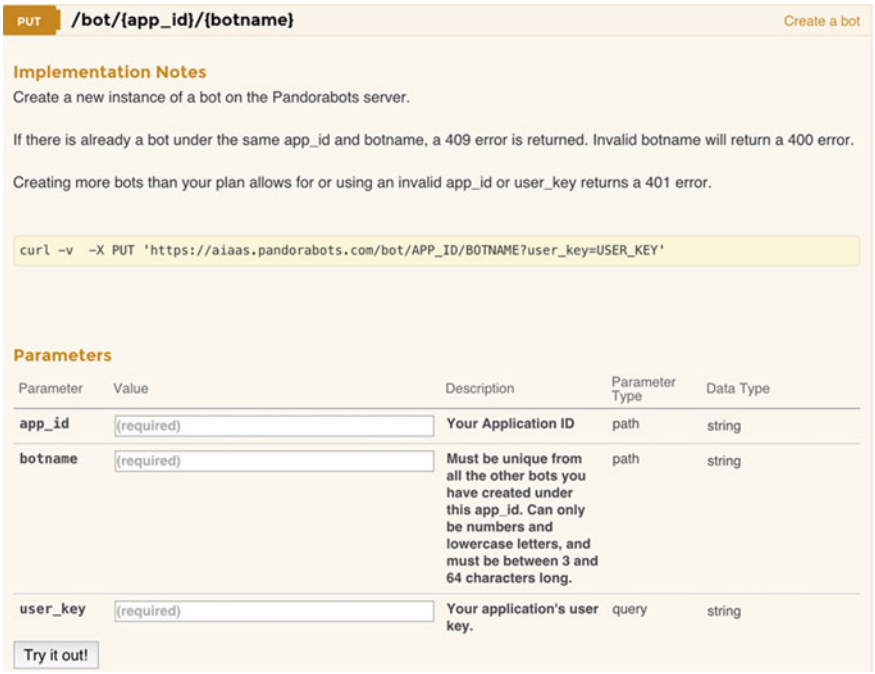

Fig. 7.7 Creating a bot in the Developer Portal (reproduced with permission from Pandorabots, Inc.)

Using `cURL` to talk to the API requires more complex commands. For example, to upload a file using the CLI, you would type the command shown in Code 7.17.

However, using curl you would have to insert the values for `user_key`, `app_id` and `botname` and type something like the command shown in Code 7.18.

Some API resources that provide assistance with API calls are available here.[15] For example, the resource in Fig. 7.7 shows an alternative way to create a bot.

These resources are also useful for observing the feedback provided to the API call and the text of the API calls, as this may be useful when linking to an app programmatically.

Exercise 7.5: Creating and testing a bot on the Pandorabots Developer Portal

1. Create a bot using either the CLI or the cURL command.
2. Upload one or more AIML files to your bot.
3. Compile the bot (Note: each time a file is modified or uploaded the bot must be compiled in order for the changes to be available in a conversation.
4. Talk to the bot using the input patterns in your AIML files.

[15] https://developer.pandorabots.com/docs. Accessed February 20, 2016.

7.4.2 *Linking an Android App to a Bot*

In this section, we describe how to link from an Android app to a Pandorabots bot that has been created on the Developer Portal. This is shown in the `TalkBot` app. The app works as follows: the user can press a button to say something to the bot, the recognized utterance is passed on to Pandorabots and using its corresponding AIML code it generates a response that is retrieved by the Android app. If the response is simple (e.g., a text), it is synthesized back to the user, whereas if the response includes mobile functions (see Sect. 7.5), they are executed and the results are synthesized to the user (e.g., checking the battery level and informing about it).

We have arranged the classes in this app in different packages (folders) as shown in Fig. 7.8.

- In *Pandora,* we have included the classes to connect to Pandorabots (`PandoraConnection`), process the results received (`PandoraResultProcessor`, `FindLocation`), and manage possible errors (`PandoraException`, `PandoraErrorCode`).
- In *VoiceInterface,* we include the `VoiceActivity` class to process the speech interface. This is exactly the same class as was used in the `TalkBack` app (Chap. 6).
- In the root, we have the `MainActivity` class. This class specifies the main behavior of our app.

The `MainActivity` class is very similar to the one presented for the `TalkBack` app in Chap. 6. In `TalkBack`, in order to demonstrate speech recognition and TTS we simply took the best result from speech recognition and spoke it out using TTS (Code 7.19).

Now in `TalkBot`, we want to send the recognized result of the user's input to the Pandorabots service and get a response. The following code accomplishes this and then calls a method to process the response (Code 7.20).

Additionally in the `catch` section, there is a call to the method `processBotErrors`, which deals with a number of possible errors such as invalid keys and ids for connecting to Pandorabots, no match for the input, and Internet connection errors.

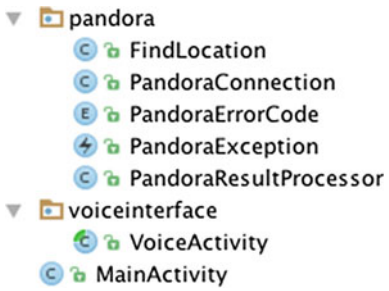

Fig. 7.8 Packages and classes for the TalkBot app

```
String bestResult = nBestList.get(0);

(…)
speak(bestResult, "EN", ID_PROMPT_INFO);
```

Code 7.19 Fragment of the `processAsrResults` method in the `MainActivity` class (`TalkBack` app, Chap. 6)

```
String userQuery = nBestList.get(0);

(…)
try {
   String response = pandoraConnection.talk(userQuery);
   processBotResults(response);
} catch (PandoraException e) {
   processBotErrors(e.getErrorCode());
}
```

Code 7.20 Fragment of the `processAsrResults` method in the `MainActivity` class (`TalkBot` app)

When a response is returned from Pandorabots, it is handled in the `processBotResults` method (Code 7.21).

Here, there are two cases to consider:

1. The result is in the form of text to be output as a spoken response. In this case, the method `removeTags` is called to remove any HTML tags. The resulting string is spoken using TTS.
2. The result contains an `<oob>` tag, indicating that the response requires further processing to determine what sort of mobile function is being requested. In this case, the result is sent to the method `processOobOutput` in the class `OOBProcessor`. This will be discussed further in Sect. 7.5.

Connecting with Pandorabots

Some parameters are declared in `MainActivity` that are required to make a connection to your bot on Pandorabots. You must insert your own values (Code 7.22).

These values are used in the `PandoraConnection` class to establish the connection with Pandorabots. We have included them in `MainActivity` to make the `PandoraConnection` class independent of the actual bot used. By doing this, you can use `PandoraConnection` every time you want to use a bot in your Android apps without changing a single line of the code and just adjusting these parameters in the initial activity that uses the code (`MainActivity` in our case).

```
public void processBotResults(String result){


  // Speak out simple text from Pandorabots after removing

      any HTML content
  if(!result.contains("<oob>")){
    result = removeTags(result);
    (…)

    speak(result,"EN",ID_PROMPT_INFO);
    (…)
  }
  // Send responses with <oob> for further processing
  else{

    PandoraResultProcessor oob=

       new PandoraResultProcessor(this, ID_PROMPT_INFO);
    (…)

    oob.processOobOutput(result);
    (…)
  }
}
```

Code 7.21 Fragment of the `processBotResults` method in the `MainActivity` class (`TalkBot` app)

```
private String host = "aiaas.pandorabots.com";
private String userKey = "Your user key here";
private String appId = "Your app id here";
private String botName = "The name of your bot here";
PandoraConnection pandoraConnection = new PandoraConnec-
tion(host, appId, userKey, botName);
```

Code 7.22 Initializing the Pandorabots connection parameters in the `MainActivity` class (`TalkBot` app)

`PandoraConnection` is a simplified version of a class created by Richard Wallace as a Java API to the Pandorabots service.[16] The class has been edited and simplified to adapt it to the requirements of our Android app. The class does the following:

1. The connection parameters specified in `MainActivity` are initialized.
2. The user's input string is sent to the chatbot on the Pandorabot's service and the bot's response is returned as a JSON object from which the responses to be returned to `MainActivity` are extracted. In order to do that, the apache http client libraries are employed, which requires including them as dependencies in the `build.gradle` file (Code 7.23).

[16] https://github.com/pandorabots/pb-java. Accessed February 20, 2016.

```
Content content =

   Request.Post(uri).execute().returnContent();
String response = content.asString();
JSONObject jObj = new JSONObject(response);
JSONArray jArray = jObj.getJSONArray("responses");
for (int i = 0; i < jArray.length(); i++) {
    responses += jArray.getString(i).trim();
}
```

Code 7.23 Fragment of the talk method in the `PandoraConnection` class (`TalkBot` app)

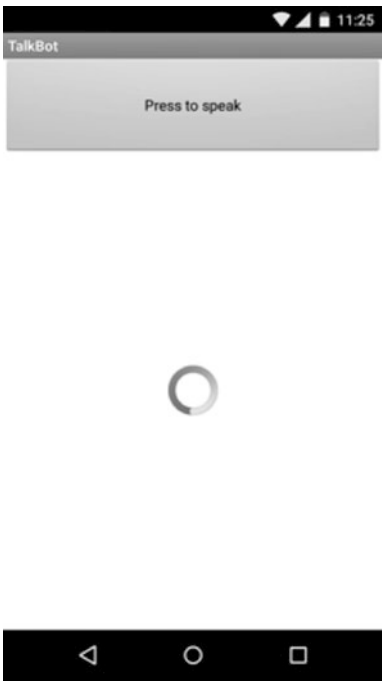

Fig. 7.9 Interface of the `TalkBot` app

Exercise 7.6: Speech enabling conversational interaction in the TalkBot app
Run the app and interact with it using inputs similar to those that you used to interact with the bot in Playground. Remember that for the code to work you must include your own connection parameters. Figure 7.9 shows the interface for the app.

7.5 Introducing Mobile Functions

Chatbot technology has been extended to support the development of virtual personal assistants that can carry out commands to the device and answer queries to Web services. A recent addition to AIML supports these mobile functions. The following is an example (Code 7.24).

Here, the <oob> tags separate content within the template that is not part of the response to be spoken to the user: In this case, including a tag indicating that the task involves search and that the content of the search is the item retrieved from the wildcard *. In other words, if the user says "Show me a Web site about speech recognition" the bot outputs the words "let's try a Google Search" and encloses the command to search for speech recognition within <oob> tags. Figure 7.10 shows the result of this interaction:

In the next section, we explain how to realize this interaction in our Android app.

7.5.1 Processing the <oob> Tags

In TalkBot, we capture the user's input using Google speech recognition and send the text to Pandorabots. Once the text is matched against a pattern in the AIML file, the content of the template is sent back to the MainActivity class. If the content is only text, it is spoken out using the Google TTS.

However, if the template contains an <oob> tag, we need to check for this and take some action. In this case, we call a new class OOBProcessor in MainActivity that contains a number of methods for processing the content of the <oob> tag and executing the required commands.

First, we need to separate the items in the template into <oob> content and text to be spoken to the user. This processing is done in the PandoraResultProcessor class. The processOobOutput method processes the JSON object returned from Pandorabots and extracts the content of the label that, as shown in Code 7.25.

```
<category>

<pattern>SHOW ME A WEBSITE ABOUT * </pattern>

<template> Let's try a google search

   <oob><search><star/></search></oob>

</template>

</category>
```

Code 7.24 Including mobile functions in AIML with <oob> tags

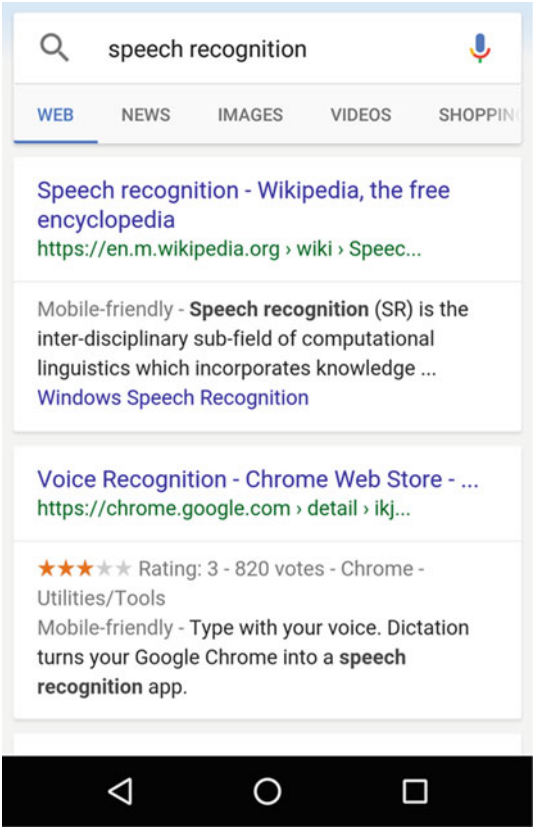

Fig. 7.10 Response to the query "show me a Web site about speech recognition." Accessed at 22:38 on February 17, 2016. Google and the Google logo are registered trademarks of Google Inc., used with permission

```
that:
<oob><launch>calendar</launch></oob>launching
calendar
```

Code 7.25 JSON object returned from query to Pandorabots

The text within the `<oob>` tags is assigned to a variable `oobContent` and the remaining text is assigned to `textToSpeak` (Code 7.26).

Next, we analyze `oobContent` to determine the type of command—for example, "search," "battery," and "maps"—that is contained in the tag embedded within the `<oob>` tag, using the `processOobContent` method. In this case, we extract `<launch>calendar</launch>`, which enables us to determine that the command is to launch an app and that the app is the calendar. With this

```
oobContent: <launch>calendar</launch>

textToSpeak: launching calendar
```

Code 7.26 Assigning the text with the <oob> tags

```
launchApp(app,textToSpeak)
```

Code 7.27 Calling the launchApp method

information, we can call the launchApp method with the required parameters (Code 7.27).

Other commands, such as "search" and "maps" are handled in a similar way. However, each command has to be implemented appropriately, depending on how it is handled in Android. For example, to launch an app we need to check whether that app actually exists on the device, as the user could speak the name of an app and have the name recognized, but this does not guarantee that there is an app of that name on that particular device. The launchApp method does the following:

1. Gets a list of app names and package names on the device.
2. Checks if the app requested is on the device.
3. If not, reports to the user.
4. If yes, gets the package name of the requested app.
5. Launches the app.

7.5.2 Battery Level

The batteryLevel method checks the level of the battery on the user's device given an input such as "what is my battery level?" The following code launches an Android intent to check the battery level and convert the raw battery level to a percentage number that can be spoken out (Code 7.28).

7.5.3 Search Queries

Two categories are tagged with an <oob> tag that includes <search> (Code 7.29).

Given inputs such as "tell me about speech recognition," the text of the search query is extracted and passed to the Android ACTION_WEB_Search intent to be executed and the text in the template is spoken using TTS (Code 7.30).

```
private void batteryLevel() throws Exception {
    (…)
    Intent batteryIntent = ctx.registerReceiver(null,

      new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
    int rawlevel = batteryIntent.getIntExtra("level", -1);
    double scale = batteryIntent.getIntExtra("scale", -1);
    double level = -1;
    int pct;

    if (rawlevel >= 0 && scale > 0) {
      level = rawlevel / scale;
      pct = (int) (level * 100);  //Conversion from the raw

                                  battery level to a percentage
     ((VoiceActivity) ctx).speak("Your battery level is " +

     String.valueOf(pct) + "per cent", "EN", msgId);
    }
    (…)
}
```

Code 7.28 The `batteryLevel` method of the `PandoraResultProcessor` class (`TalkBot` app)

```
<category><pattern>Tell me about *</pattern>

<template>Searching for <star/>
<oob><search><star/></search></oob></template>

</category>

<category><pattern>What is *</pattern>

<template>Searching for information about <star/>
<oob><search><star/></search></oob></template>

</category>
```

Code 7.29 AIML code including a `<search>`

```
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
intent.putExtra(SearchManager.QUERY, query);
(…)
ctx.startActivity(intent);
```

Code 7.30 Fragment of the `search` method of the `PandoraResultProcessor` class (`TalkBot` app)

7.5.4 Location and Direction Queries

In this section, we show how to make queries about locations and directions, for example:

> Where is New York?
> Show me directions from New York to Boston.

We distinguish between absolute queries, such as these, in which all the parameters of the query are mentioned explicitly, and relative queries, in which the user's current location is assumed implicitly, as in:

> Find the nearest Starbucks.
> Show me directions to Boston.

The motivation for making this distinction is to avoid potential problems with the use of contextual information. For example, given the following sequence:

> Show a map of Boston.
> Find the nearest Starbucks.

It is possible that Boston could be assigned as the current context so that the next question is interpreted as

> Find the nearest Starbucks in Boston.

instead of

> Find the nearest Starbucks relative to my current location.

Without direct access to the context mechanisms being applied, it is not possible to resolve this. Indeed, in interaction between humans it is not always clear whether a subsequent question relates to a previous one or whether it is part of a new topic.

In order to deal with this in AIML, we identify those inputs that contain absolute queries and those that contain relative queries by inserting an additional tag

<myloc> into relative queries and adding additional code to find the user's current location. The following is a high-level overview of these processes.

Identify location queries

1. Check if `oobContent` contains `<map>`.
2. Parse to extract the values for the map query (`mapText`, `textToSpeak`).
3. Call the `mapSearch` method.

`mapSearch`

1. Speak the content of `textToSpeak`.
2. Replace spaces in `mapText` with "+".
3. Use the `ACTION_VIEW` action and specify the location information in the intent data with the `Geo URI` scheme.
4. Start activity (Code 7.31).

Relative queries are marked up in AIML shown in Code 7.32.
A relative query results in the following actions:

1. The tag `<myloc>` causes the `FindLocation` class to be instantiated in order to find the latitude and longitude of the user's current location.
2. The `ACTION_VIEW` action is called with the values for latitude (`lat`) and longitude (`lng`) for the user's current location found.

Directions are handled in a similar way. For example, if the input is a relative directions query, as in "directions to Boston," the `<myloc>` tag causes

```
Intent geoIntent = new Intent(
       android.content.Intent.ACTION_VIEW,
       Uri.parse("geo:"+ lat + "," + lng + "?q=" + mapText));
ctx.startActivity(geoIntent);
```

Code 7.31 Fragment of the `mapSearch` method of the `PandoraResultProcessor` class (`TalkBot` app)

```
<category>

<pattern>FIND THE NEAREST STARBUCKS </pattern>

<template>I'm looking on the map
<oob><map><myloc>Starbucks</myloc>
</map></oob>

</template>

</category>
```

Code 7.32 AIML relative query

```
FindLocation findLocation = new FindLocation(ctx);
double lat = findLocation.getLatitude();
double lng = findLocation.getLongitude();
Uri uri = Uri.parse("http://maps.google.com/maps?saddr=" +
lat + "," + lng + "&daddr=" + to);
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
```

Code 7.33 Fragment of the `getDirections` method in the `PandoraResultProcessor` class (`TalkBot` app)

`FindLocation` to be called to find the latitude and longitude values of the user's current location, assuming the current location as the origin of the directions query (Code 7.33).

Exercise 7.7: Testing the app

1. Run the sample code with a range of inputs that includes oob processing. You can find the `oob.aiml` file in the *chapter7/AIML* folder in our GitHub ConversationalInterface repository.
2. Note any queries that do not work.
3. Try to determine the problem, for example:
   - Is it a speech recognition error?
   - Is it due to missing categories in AIML?
   - Are additional or modified Java methods required?

7.6 Extending the App

We can extend the app in a number of ways. For example, we could add more AIML categories to allow a wider range of inputs. Another extension would be to add more commands to the device and queries to Web services. The CallMom app[17] illustrates how this can be done.

In our application, we made use of Google Search and Google Maps to provide responses to queries. CallMom also consults a number of external knowledge sources, including Wolfram Alpha, DbPedia, Trueknowledge.com, Answers.com, Weather Service, various shopping sites, and other Pandorabots.

Most chatbot markup languages nowadays have methods for representing information. For example, in AIML 2.1 there is a facility to create ontologies that enable the chatbot to make use of and reason with knowledge, while ChatScript has a facility for invoking WordNet ontologies. Another approach extends the reference implementation of AIML to enable the extraction of domain knowledge from semantic Web ontologies using a scripting language called OwlLang and to store new knowledge obtained from the conversations in the ontologies (Lundqvist et al. 2013).

[17] http://callmom.pandorabots.com/static/callmombasic/features.html. Accessed February 20, 2016.

7.7 Alternatives to AIML

AIML is a widely used markup language for specifying chatbots. However, there are some alternatives, the most notable of which is ChatScript. ChatScript was developed in 2010 by Wilcox (2011a, b) and is used mainly to provide natural language understanding capabilities for characters in games, but has also been used for the chatbot Rose that won the Loebner Prize competition in 2014. While AIML's pattern matching is word-based, in ChatScript it is meaning-based, supporting sets of words called concepts to represent synonyms, as shown in Code 7.34.

This allows rules to be written that respond to all sorts of meat (Code 7.35).

Here, the input pattern (in parentheses) contains the concept "meat" that can be matched by any of the words in the concept `~meat`. The chatbot's response is the text following the input pattern.

ChatScript is available as open source[18], and there is also a tutorial on how to build a conversational bot using ChatScript.[19]

Other alternatives to AIML are Api.ai[20] and Wit.ai.[21] Chapter 9 shows how to use the Api.ai platform to extract a semantic analysis from the user's input.

7.8 Some Ways in Which AIML Can Be Further Developed

In this section, we review some ways in which AIML has been extended as well as some suggestions for further developments.

7.8.1 Learning a Chatbot Specification from Data

Creating a chatbot in a language such as AIML typically involves hand coding a large number of categories, a process that can take several years if starting from scratch. Developers creating a chatbot on the Pandorabots Web site can make use of libraries of AIML categories to get started. For commercial developers on a special license, there is also a tool called Pattern Suggester that is part of Program AB, the most recent reference implementation of AIML 2.0. Pattern Suggester helps to automate the process of creating new patterns through a type of unsupervised

[18] http://sourceforge.net/projects/chatscript/. Accessed February 20, 2016.

[19] http://inspiredtoeducate.net/inspiredtoeducate/learn-to-build-your-own-conversational-robot-using-chatscript/. Accessed February 20, 2016.

[20] https://api.ai/. Accessed February 20, 2016.

[21] https://wit.ai/. Accessed February 20, 2016.

```
concept: ~meat ( bacon ham beef meat flesh veal
lamb chicken pork steak cow pig )
```

Code 7.34 Declaring a concept in ChatScript

```
s: ( I love ~meat ) Do you really? I am a vegan
```

Code 7.35 Using a concept in ChatScript

learning for patterns.[22] In one experiment, by searching through 500,000 inputs in logs from the CallMom app, the Pattern Suggester was able to find new patterns and create graphs at a rate of 6 categories per minute (Wallace 2014c).

A similar approach is the use of machine-learning techniques to read text from a corpus and convert it to the required AIML format. Abu Shawar and Atwell (2005) trained a bot using text from the Dialog Diversity Corpus, the spoken part of the British National Corpus (BNC), and online FAQ (Frequently Asked Questions) Web sites. They were able to generate more than one million categories extracted from the BNC. FAQs are a good corpus source as they have a clear turn-taking structure that can be easily adapted to the AIML pattern-template format. Several FAQ chatbots were generated, including one using the FAQ of the School of Computing at the University of Leeds, and a Python tutor trained on the public domain Python programming language FAQ Web site. De Gasperis et al. (2013) describe an algorithm in which texts in a corpus are used in a bottom-up procedure that chooses portions of text to be used as answers along with a keyword analysis of each piece of selected text to build questions. Each text representing an answer is then associated with possible questions and their formal variants (or paraphrases). Wu et al. (2008) describe an approach involving automatic chatbot knowledge acquisition from online forums using rough sets and ensemble learning.

AIML 2.0 contains learning features that enable the system to be taught new information and other chatbots such as Cleverbot,[23] Jabberwacky,[24] and Kyle[25] are also able to learn. Cleverbot employs a data mining approach in which it memorizes everything that is said to it and then searches through its saved conversations to find a response to new input, while Jabberwacky models the way humans learn language, facts, contexts, and rules. Kyle models the way humans learn language, knowledge, and context, making use of the principles of positive and negative feedback.

[22] https://code.google.com/p/program-ab/. Accessed February 20, 2016.
[23] http://www.cleverbot.com/. Accessed February 20, 2016.
[24] http://www.jabberwacky.com/j2about. Accessed February 20, 2016.
[25] http://www.leeds-city-guide.com/kyle. Accessed February 20, 2016.

7.8.2 *Making Use of Techniques from Natural Language Processing*

One potential criticism of chatbot technology is that it does not make use of theoretically driven approaches and dialog technology but instead uses a simple pattern-matching approach within a stimulus-response model. It could be argued that incorporating additional technologies into AIML would make the authoring process more difficult. It would be useful to conduct empirical studies to ascertain the effectiveness of the additional technologies for the authoring process as well as for pattern matching and RG. As it is, pattern matching in AIML is fast and efficient, even when searching a large number of patterns. Moreover, from a practical viewpoint it could be argued that in reality most language use in interaction with a chatbot does not need to address the ambiguous and complex sentences that are the concern of theoretical linguists and that a stimulus–response model has the merits of simplicity and practical utility (see discussion of this issue by Wallace.[26]

Nevertheless, there have been some useful suggestions as to how AIML could be enhanced using techniques from natural language processing, most notably in a paper by Klüwer (2011). One problem concerns the authoring of patterns. In order to be able to handle surface variation in input, i.e., alternative syntactic structures and alternative lexical items, an AIML botmaster has to manually create a large number of alternative patterns. Klüwer describes some natural language processing technologies that could be used to optimize pattern authoring. For example, surface variation in patterns could be addressed by using dependency structures (see Chap. 8) rather than surface strings so that all variations on a sentence with the same dependency structure would be associated with a single pattern. To handle sentences that have the same meaning but different surface forms—for example, the active and passive forms of a sentence—a semantic analysis of the different forms of the sentence would abstract from their surface forms and allow the different forms to be associated with a single pattern. These techniques could also help to address the problem of false positives when the user's input is matched erroneously with patterns including wildcards that are used to cover variations in surface structure.

Natural language processing technology could also be used to generate alternative output to allow for greater flexibility. With current chatbots, the output is generally static, having either been defined manually as a system response or assembled from templates in which some variables are given values at runtime. In AIML, it is possible to code a set of alternative responses that are generated randomly and there is also a <condition> tag that allows particular actions in a template to be specified conditionally. ChatScript makes use of a C-style scripting language that can be used along with direct output text to produce more flexible

[26] http://www.pandorabots.com/pandora/pics/wallaceaimltutorial.html. Accessed February 20, 2016.

responses. For an approach to the automatic generation of output from abstract representations, see Berg et al. (2013).

7.9 Summary

In this chapter, we have shown how to create a chatbot that can engage in conversational interaction and also perform functions on a mobile device such as launching apps and accessing Web services. We have used the Pandorabots platform to run the chatbot and have specified the conversational interaction using AIML. Using this approach, we have not needed to implement components for SLU, DM, and RG, as the pattern-matching capabilities of AIML and the associated templates are able to handle a wide range of user inputs and produce appropriate system responses. We have shown how the app developed in this chapter can be further extended, and we have reviewed some alternatives to AIML.

In the following chapters, we will examine more advanced technologies that are used to develop conversational interfaces, beginning with SLU in Chap. 8.

Further Reading

For readers interested in the Turing test, the collection of papers by an impressive range of scholars in Epstein et al. (2009) explores philosophical and methodological issues related to the quest for the thinking computer. The collection also includes Turing's 1950 paper "Computing machinery and intelligence." There is also an interesting paper by Levesque discussing the science of artificial intelligence in which the Turing test is criticized for relying too much on deception. A set of questions, known as the Winograd schema questions, is proposed as a more useful test of intelligence.[27]

References

Abu Shawar B, Atwell E, Roberts A (2005) FAQChat as an information retrieval system. In: Vetulani Z (ed) Human language technologies as a challenge. Proceedings of the 2nd language and technology conference, Wydawnictwo Poznanskie, Poznan, Poland, 21–23 April 2005: 274–278. http://eprints.whiterose.ac.uk/4663/. Accessed 20 Jan 2016

Berg M, Isard A, Moore J (2013) An openCCG-based approach to question generation from concepts. In: Natural language processing and information systems. 18th international conference on applications of natural language to information systems, NLDB 2013, Lecture notes in computer science, vol 7934. Springer Berlin Heidelberg, Salford, UK, 19–21 June 2013, pp 38–52. doi:10.1007/978-3-642-38824-8_4

Crocker MW, Pickering M, Clifton C Jr (1999) Architectures and mechanism for language processing, 1st edn. Cambridge University Press, Cambridge. doi:10.1017/cbo9780511527210

[27] http://www.cs.toronto.edu/~hector/Papers/ijcai-13-paper.pdf. Accessed February 20, 2016.

De Gasperis G, Chiari I, Florio N (2013) AIML knowledge base construction from text corpora. In: Artificial intelligence, evolutionary computing and metaheuristics, vol 427. Studies in computational intelligence, pp 287–318. doi:10.1007/978-3-642-29694-9_12

Epstein R, Roberts G, Beber G (eds) (2009) Parsing the turing test: philosophical and methodological issues in the quest for the thinking computer. Springer, New York. doi:10.1007/978-1-4020-6710-5

Klüwer T (2011) From chatbots to dialog systems. In: Perez-Marin D, Pascual-Nieto I (eds) Conversational agents and natural language interaction: techniques and effective practices. IGI Global Publishing Group, Hershey, Pennsylvania, pp 1–22. doi:10.4018/978-1-60960-617-6.ch001

Lundqvist KO, Pursey G, Williams S (2013) Design and implementation of conversational agents for harvesting feedback in eLearning systems. In: Hernandez-Leo D, Ley T, Klamma R, Harrer A (eds) Scaling up learning for sustained impact. Lecture notes in computer science, vol 8095, pp 617–618. doi:10.1007/978-3-642-40814-4_79

Wallace R (2003) The elements of AIML Style. ALICE A.I. Foundation, Inc. http://www.alicebot.org/style.pdf. Accessed 20 Jan 2016

Wallace R (2009) Anatomy of A.L.I.C.E. In: Epstein R, Roberts G, Beber G (eds) Parsing the turing test: philosophical and methodological issues in the quest for the thinking computer. Springer, New York, pp 81–210. doi:10.1007/978-1-4020-6710-5_13

Wallace R (2014a) AIML 2.0 working draft. https://docs.google.com/document/d/1wNT25hJRyupcG51aO89UcQEiG-HkXRXusukADpFnDs4/pub. Accessed 20 Jan 2016

Wallace R (2014b) AIML—sets and maps in AIML 2.0. https://docs.google.com/document/d/1DWHiOOcda58CflDZ0Wsm1CgP3Es6dpicb4MBbbpwzEk/pub. Accessed 20 Jan 2016

Wallace R (2014c) AIML 2.0—virtual assistant technology for a mobile era. In: Proceedings of the mobile voice conference 2014, San Francisco, 3–5 March http://wp.avios.org/wp-content/uploads/2014/conference2014/35_mctear.pdf. Accessed 20 Jan 2016

Weizenbaum J (1966) ELIZA—a computer program for the study of natural language communication between man and machine. Commun ACM 9(1):36–45. doi:10.1145/365153.365168

Wilcox B (2011a) Beyond Façade: pattern matching for natural language applications. http://www.gamasutra.com/view/feature/134675/beyond_façade_pattern_matching_php. Accessed 20 Jan 2016

Wilcox B (2011b) Fresh perspectives—a Google talk on natural language processing http://www.gamasutra.com/blogs/BruceWilcox/20120104/90857/Fresh_Perspectives_A_Google_talk_on_Natural_Language_Processing.php. Accessed 20 Jan 2016

Wu Y, Wang G, Li W, Li Z (2008) Automatic chatbot knowledge acquisition from online forum via rough set and ensemble learning. IEEE Network and Parallel Computing (NPC 2008). IFIP International Conference, pp 242–246. doi:10.1109/npc.2008.24

Web sites

Alice A.I. Foundation www.alicebot.org
AIML matching http://www.alicebot.org/documentation/matching.html
AIML tutorial http://www.pandorabots.com/pandora/pics/wallaceaimltutorial.html
AIML 2.0 Working Draft https://docs.google.com/document/d/1wNT25hJRyupcG51aO89UcQEiG-HkXRXusukADpFnDs4/pub
API.ai https://api.ai/
CallMom app http://callmom.pandorabots.com/static/callmombasic/features.html
Chatbots.org https://www.s.org/
Cleverbot http://www.cleverbot.com/
Jabberwacky http://www.jabberwacky.com/j2about
Kyle http://www.leeds-city-guide.com/kyle

Node.js http://nodejs.org/download/
Pandorabots http://www.pandorabots.com/
Pandorabots blog http://blog.pandorabots.com/
Pandorabots Command Line Interface (CLI) https://github.com/pandorabots/pb-cli
Pandorabots CLI instructions http://blog.pandorabots.com/introducing-the-pandorabots-cli/Node.js
Pandorabots Developer Portal https://developer.pandorabots.com/
Pandorabots Github https://github.com/pandorabots
Pandorabots Playground https://playground.pandorabots.com/en/
Pandorabots Playground tutorial https://playground.pandorabots.com/en/tutorial/
Pandorabots Rosie library https://github.com/pandorabots/rosie
Pandorabots Twitter https://twitter.com/pandorabots
Wit.ai https://wit.ai/