Why use buttons when natural language dialogue makes interaction easier: the Winamp Project

André Hellström, Nabil Benhadj, & Johan Windmark
Lund University, 2003

Abstract

This project was intended to show the possibility of using natural language dialogue with standard software in a typical PC environment. The prototype system integrates spoken language with the Winamp media player. As a result of this Winamp will be totally controlled by spoken English.

1 Introduction

The aim of the project is to create a spoken agent to control Winamp. Winamp is a leading media player that is easy to manipulate and control, making it well fit for the purpose. The interaction between the user and the agent is through voice recognition and speech feedback. The system should be easy to understand, even for a first time user. Another goal is to make the system robust enough so that it responds correctly with a high probability.

2 System overview

The system code is implemented in C++ using Microsoft Speech SDK (SAPI 5.1). The tutorial coffee0 that came with the SDK was the starting point of the project and was expanded to contain the state machine described below. Microsoft Speech can work in two modes: dictation mode and command mode. In the dictation mode, whole sentences are parsed. In dictation mode, the possible utterances are predefined in a grammar and each utterance is matched to a specific rule. The grammar for the rules is defined in an XML-file.

3 Implementation

The system code is implemented in C++ using Microsoft Speech SDK (SAPI 5.1). The tutorial coffee0 that came with the SDK was the starting point of the project and was expanded to contain the state machine described below. Microsoft Speech can work in two modes: dictation mode and command mode. In the dictation mode, whole sentences are parsed. In dictation mode, the possible utterances are predefined in a grammar and each utterance is matched to a specific rule. The grammar for the rules is defined in an XML-file.

4 State-machine

The flow of the system is:

1. The user gives a command to the system.
2. The system gives feedback on the command
3. Possibly gives a command to Winamp.
4. Go back to 1

To constrain the number of different commands available to the user at a given time, a state machine is used.

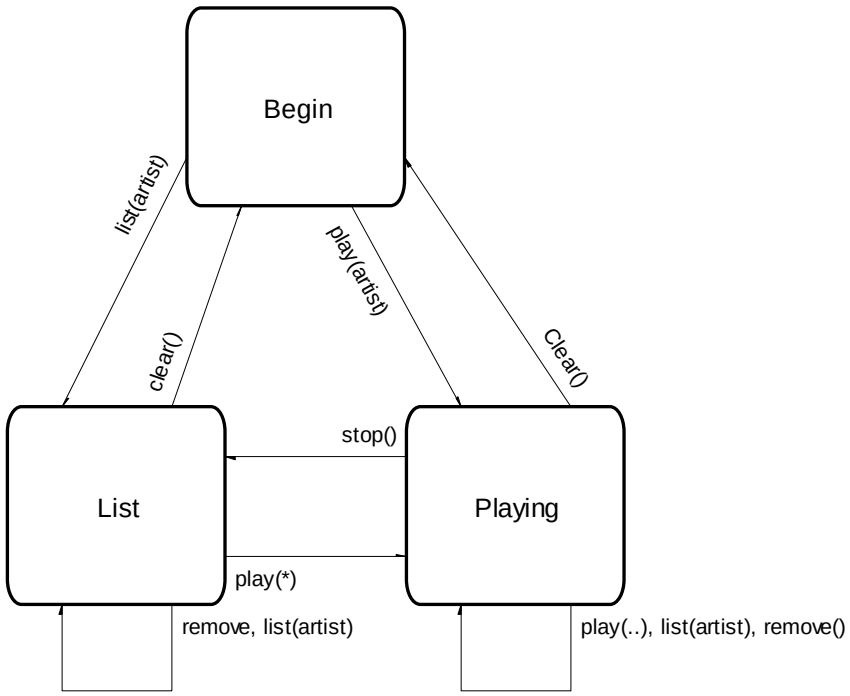

Figure 1

The state machine uses following concepts:

States, i.e. Begin, Playing
Transitions, i.e. Begin->Playing
Actions, i.e. play(Artist)
Rules, i.e. Rule7 triggered by ”* Play Madonna”

Let’s start by looking at the message handling code, suppose we’re in state Begin and have received a message from rule7.

```
messageHandler(message)
 switch(STATE)
  case Begin:
   switch(message)
    case RULE1:
     transitionBeginPlaying1(message);
    case RULE7:
     transitionBeginPlaying2(message);
    case RULE3:
    case RULE2:
     transitionXZ1(message);
    default:
      ”Error: Unhandled rule!”
  case Playing:
   switch(message)…
    case RULE7:
     transitionYY(message);
    default:
     ”Error: Unknown state!”
```

Depending on the state, different transitions can be called for the same rule, that is, the same rule can trigger different transitions. Also note that different rules can call the same transitions, within the same state. Let’s look at the transition code:

```
transitionXY2(message):
end_X()
visual and/or audio feedback code()
action_play(message.argument1)
begin_Y()
```

First a stop code is called for the state X. It will turn off all rules. Then some feedback is given to the user and the appropriate action is

called. Finally the start code for Y is called. It will turn on all the Y-rules, the same rules that can be caught in the message handler under state Y, and set the current state to Y. In this case action_play(...) means load a Madonna playlist and start playing it, as a command to Winamp.

To summarize: The active rules depend on the current state. When a rule is triggered by the speech recognition system, it's triggering a transition. The transition calls the appropriate action and changes the state, thereby changing the set of active rules.

5 Grammar and rules

The speech recognition system divides the input into phrases, where each phrase is surrounded by silence. Consider the case that a playlist is loaded and the user wants to hear a specific song. The rule to catch this looks like this:

```
<RULE ID="VID_PlayNumber" TOPLEVEL="ACTIVE">
<O>
    <L>
        <P>could you</P>
        <P>will you</P>
        <P>I want to</P>
        <P>the song I want to hear</P>
        <P>I need to</P>
    </L>
</O>
<O>please</O>
<O>
    <L>
        <P>Play</P>
        <P>Start</P>
        <P>Hear</P>
    </L>
</O>
<O>
    <L>
        <P>song</P>
        <P>piece</P>
        <P>track</P>
        <P>tune</P>
    </L>
</O>
<RULEREF REFID="VID_Number"/>
</RULE>
```

The most important words for this rule is the number, last in the grammar. VID_Number, here defines one of the numbers 1 to 10 and is the sub rule that actually activates the VID_PlayNumber rule. The rest of the rule is only a list of optional padding so that the rule is triggered not only by an utterance like "Number one" but also by "Could you please play track number one". The example pretty well catches the structure of all rules.

6 Future possibilities

Some interesting and important details of improvement would be to make the program more stable and increase flexibility. Below are a couple of suggestions of how this could be accomplished.

An easy way of making to program more dynamic is to provide an easy tool for automatically adding new artists and songs to the play list. One way of doing this is by having a parser read all music files in the designated catalogue and then adds whatever artist/song is missing in the XML-file each time the program is executed. Some system parameters would then have to be added into a text file for the program to read at start up.

Further improvement on the voice recognition would be preferable to increase stability of the speech-to-text input. A first and an easy way of enhancing the voice recognition input in the interface would be by providing a high quality microphone or possibly a headset.

Other possibilities would be to make a system that works as a speech user interface for programs in general. The only change required would be to change the grammar file (XML-file).

7 Evaluation

The system was tested continuously during development. After each test, the grammar and state machine was altered to enhance performance. The problems encountered were that user commands were not recognised or that rules were triggered without proper reason. In the final state of the project, a user knowing the system architecture can go through all actions with very few incorrect system responses. The problem with rules triggered without the correct corresponding user input still remains.

8 Conclusion

The Winamp project offers a good suggestion of the possibility's in speech recognition today. Use of natural language dialogue interfaces to standard software is far from perfect. The project did however show that the technical part of voice recognition has come far and will probably be good enough for serious general usage in a couple of years. There are however still the complexity of linguistics, natural language dialogue, to solve in order to use speech control in a natural way. Combinations of words and sentences are close to endless which makes natural dialogue extremely multifaceted. The idea of limiting the field of possible recognition, as in the Winamp project, has however showed that that natural dialogue is possible in very restricted domains.

References

Microsoft Research, *Lifelike Computer Characters: the Persona project at Microsoft Research*, 1996

Nugues Pierre, LTH Lecture notes, 2003

Winamp Homepage: http://www.winamp.com

Microsoft SpeechSDK Homepage: http://www.microsoft.com/speech/download/sdk51/

9 Appendix

A scenario: using the different features of the program

When program starts it outputs (speech output) the following:

Welcome Pierre. This is the Winamp speech control. The jukebox contains, Madonna, Sting, Prince, Red hot Chilipeppers and David Gray. You can choose to list or to play one of these artists. Have a nice try…

The interaction continues with the following dialogue:

Command to system	**Answer from system**	**Action**
Help	*Choose to play a song by an artist or choose to list available songs by artist*	No action
List Madonna	*Loading list, Madonna list <displays (speech) all songs>*	List(Madonna)
Play number two	*Playing*	Play(2)
Stop	*Stop playing*	Stop()
Play next	*Next song*	Play(next)
Play previous	*Previous song*	Play(prev)
Help	*Choose to play next son, previous song, clear list or stop playing*	
Pause	*Pause*	Pause()
Pause	*Pause*	Play()
Stop	*Stop playing*	Stop()
Erase list	*Clear list*	Clearlist()
List artist	*The artists are <displays (speech) all artists>*	No action
Could you find anything by Prince	*Loading list, Prince list <displays all songs>*	List(Prince)
Play	*playing*	Play()
Erase list	*Erasing list*	Clearlist()
I want to hear a song by David Gray	*Playing, playing David Gray*	Play(David Gray)
Stop	*Stop*	Stop
Erase list	*Clearing list*	Clearlist()