Proposal: Interactive DJ

Jeffrey Faden
Porter College
jfaden@ucsc.edu

Shiranne Radzinski
Stevenson College
sradzins@ucsc.edu

Abstract

This document contains a proposal for a software system that plays music based on a dialogue with the user and the computer. Described are the methods it uses to collect correct data and send the commands to the music player, including examples of dialogue and proposed software to enable this process.

1 Introduction

In brainstorming ideas for a speech system, we thought about what sort of activities we do on a day to day basis. One leisurely activity that seems part of everyone's life is listening to music, especially by using music players on the computer. Thinking about how we could apply speech to applications such as music players, we compared the use of these programs to talking to a DJ at a party, or calling one on the radio. We believe that software music players can assume the role of a DJ, with the help of an artificial intelligence system.

2 Purpose

While software music players allow the user total control over what music is played, DJs have the advantage to offer additional information about the music. This can be especially useful when searching for a particular song. For example, a partygoer could ask a DJ to play a Beatles song without specifying any additional information, such as the title. The DJ would then offer some possibly song titles, and narrow the criteria of the requested song. A program like iTunes can provide similar functionality, but the request must be specific, as if the user already knows what they are looking for.

An interim program could provide a dialogue between the user and the music player that would allow for simpler searching for the correct song, similar to a conversation with a DJ.

2.1 Functionality

The proposed program would accept requests from the user and either respond with information for the sake of clarification, or play a song. In this sense, the program acts as both a dialogue system and a music player controller. If the user's goal is to play a certain track by the Beatles, for instance, the user could ask the machine to play a track by the Beatles. Having not been given enough criteria to select a single track, the program will respond with relevant questions in order to narrow down the options, corresponding to attributes such as album name, genre, year, words in the song title, etc. Upon finding the correct track, the program will play the song.

2.2 Examples

In which the user knows the name of the song and artist:

User: Play *Wonderwall* by Oasis.
(The program sends the command to play *Wonderwall* by Oasis)

This example demonstrates a direct command, requiring no further dialogue from the program. Unless the song cannot be found or there are other problems such as duplicates or a misinterpretation of the request, the program will not respond vocally.

In which the user does not know which song he wants:

User: Play me a song by the Beatles.
Program: Which one?
User: I don't know.
Program: Do you know which album?
User: Which albums are there?
Program: *Help!*, *Rubber Soul*, *Abbey Road*, *the White Album*.
User: I think it's in *Abbey Road*.
Program: Popular songs from that album are *Come Together* and *Maxwell's Silver Hammer*.
User: What are some other songs from that album?
Program: *Because, Carry That Weight, Golden Slumbers*...
User: Play *Carry That Weight*.
(The program sends the command to play *Carry That Weight* by the Beatles)

Throughout this dialogue, the program gains more and more criteria for the song. It starts with the Beatles. Upon finding that the user need to know more data, it will continue asking questions until the user requests more specific data, or until the user can provide more data. In this case, the user asks for more details until he chooses an album, upon which the program begins to list songs from the album that have a high play count (an attribute in iTunes' library XML).

In which the user only knows part of the song title:

User: Play me a song.
Program: Which one?
User: The title has "purple" in it.
Program: Is it *Purple Haze* by Jimi Hendrix?
User: No.
Program: Then is it *Purple Stain* by the Red Hot Chili Peppers?
User: No.
Program: I have no other songs.
User: Actually, the title has "green" in it.
Program: Is it *Blue in Green* by Miles Davis?
User: Yes.
(The program sends the command to play *Blue in Green* by Miles Davis)

This example shows that the user can ask for part of a song's attribute – in this case, part of the title. It can also accept corrections, like "actually," and will remember that the previous command was related to the title of the song. If the program finds a small amount of songs, it will list them one by one instead of asking for more information to narrow the search.

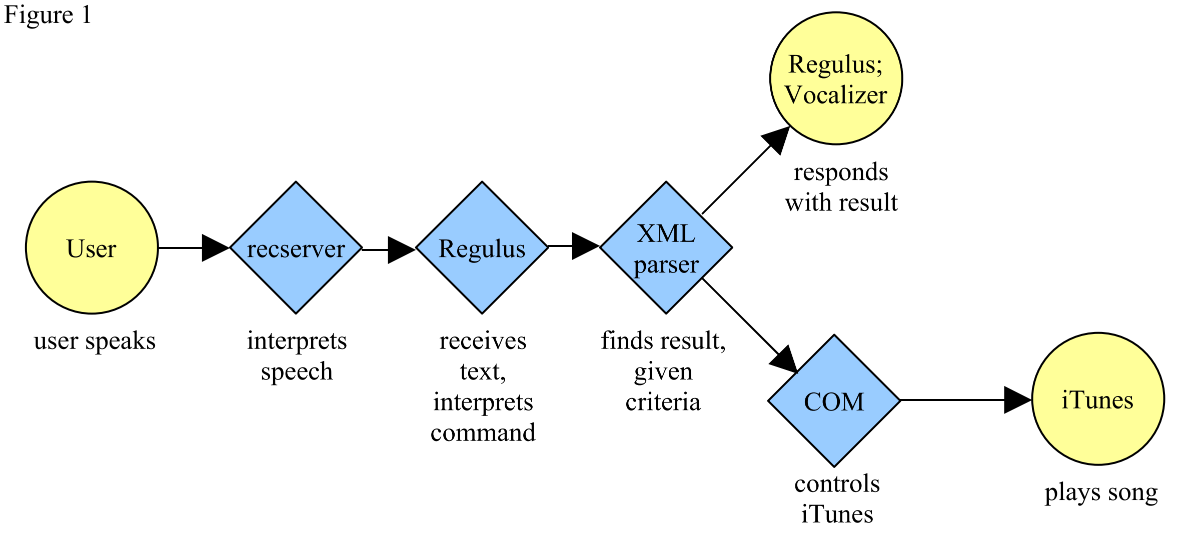
3 Design

This program must use a number of tools for the various tasks it performs. It must:

- Accept voice input
- Interpret the command
- Query a music database
- Respond with appropriate information
- Control a music player

A diagram of the implementation can be seen in Figure 1.

Figure 1



3.1 Implementation

Accepting Voice Input: The program will use Nuance's *recserver* application to accept voice input.

Interpreting the Command: The program will use Regulus to break down the elements of the sentence (Bouillon et al, 2006). Once it has determined the type of sentence, it will either query a music database and respond with appropriate information, or control the music player.

Querying a music database: The program will parse the XML of iTunes' music library to find relevant songs, artists, etc. The parsing routine will return songs or other information that fits the criteria of the query.

Responding with appropriate information: Regulus will then output through Nuance Vocalizer, responding with a statement regarding the songs the program has found.

Controlling a music player: When given the correct command, the program will use iTunes' COM scripting interface to control the player (Apple Computer, 2006).

3.2 Testing

One prominent feature of iTunes is its ability to quickly sort through its music database to find relevant songs. Our program should be able to replicate iTunes's functionality through spoken dialogue. The program will be successful if it returns the same data as iTunes does in its contextual searches, in addition to properly interpreting commands, aiding in the search process, and controlling iTunes's player functions.

4 New Features

In comparison to previous dialogue systems we have seen in Ling 160, this program will combine many features we have seen, and some we have not yet covered.

Important methods that we have examined in class include querying data from a database, directly executing commands, and interpreting corrections.

New features include adding words to the lexicon based on the current data in iTunes's music database, and making Nuance work with JavaScript and XML databases.

5 Conclusion

This project could prove to be a valuable resource for developers and users alike. As an extension to iTunes's functionality, it can improve the experience of the everyday music listener in terms of usability and flexibility. For developers, it can function as a basis for dialogue with a database browser. Overall, this could be a step forward in providing a substitute for the usual DJ at a party, giving the listener something more comprehensive and direct.

References

Apple Computer. 2006. *iTunes COM Interface Documentation*. <http://www.apple.com>.

Pierrette Bouillon, Beth Ann Hockey, and Manny Rayner. 2006. *Putting Linguistics into Speech Recognition*. Center for the Study of Language and Information, Stanford, CA.