Re-engineering Speech Recognition and Text-to-Speech Servers in Jiazzi

Shane Cruz
Jang Kim
{scruz,jangkim}@mit.edu

May 16, 2002

Abstract

The goal of the project was to re-engineer existing speech recognition and text-to-speech servers currently being used in a Media Lab project called Impromptu. The old implementation had been developed iteratively over time, which has convoluted the original design. Also, a substantial amount of existing code has been tailored to integrate with the Impromptu architecture. The goal of this project was to re-engineer the servers using Jiazzi, which forces a more clean, modular design, to make the servers more generic and robust. The servers were re-engineered using Jiazzi to service any client capable of making a network connection.

Background and Motivation

Speech recognition (SR) and text-to-speech (TTS) services are becoming more prevalent in software applications today. Although the technology is far from perfect, giving an application the ability to utilize such speech services can greatly improve user interfaces, especially in mobile environments.

Impromptu, a current project at the MIT Media Lab, uses SR and TTS servers as services over Internet Protocol (IP) for a mobile client device. It is an audio-oriented project, consisting of several components distributed over the network, and relies upon these speech services to create an appropriate user interface for a mobile user. Essentially, Impromptu components send the SR and TTS servers input, which is processed, and output to the appropriate component.

Recently, peers at the Media Lab have inquired about the use of the speech services after Impromptu demonstrations because the servers run reliably over the network. This interest has sparked the re-examination of the SR and TTS servers, which has exposed design flaws.

The servers have been iteratively developed over time, which has effectively convoluted the original design. Some modular boundaries were crossed, and a substantial amount of code has been tailored to integrate with the Impromptu architecture. In order to provide SR and TTS servers that could be reused in other applications, thoughtful re-engineering and refactoring was necessary.

For this project, we felt using Jiazzi to re-engineer the speech servers would force a careful analysis of the current design, in order to create modular units with clean interfaces that could be reused by others. The next section explains how the re-engineering was performed.

Summary and Evaluation

The first step was to analyze the current implementation. We looked at every Java class involved in the SR and TTS server implementation, and examined the method interfaces for each, and how they interacted. At this step, many problems of Impromptu-tailored code were found, as well as some instances of hard-coded data. Upon further inspection, a significant amount of duplicated and similar code existed for handling and accepting network connections.

After reviewing the implementation, we began to draw unit diagrams of the servers, and how the classes should interact. We found that much of the duplicated and similar network code should logically be extracted into a new, separate module of classes that could be imported by any other unit. Another major design change was to create a more concrete split between the core speech services and the actual server code to interface with clients over the network. This resulted in the creation two extra modules that would replace Impromptu-specific classes in order to provide speech services to any client capable of using network connections. The final unit diagram is attached.

Once a final unit diagram was drawn, we created Jiazzi unit signatures and package signatures for the design. Then, appropriate classes were modified, and a few new classes were created to implement the unit design. The core speech processing classes were made more generic, and had additional functions that could be used, but were not originally needed for Impromptu. It was nice to be able to decide upon Jiazzi signatures, and then go about implementation concurrently without worrying about compilation errors. Finally, all units were linked and compounded together to form an executable server.

We felt that taking a units approach to re-engineering was a great advantage in deciding upon clean interfaces of classes and modules, and their interactions. This forced careful thinking in order to craft units and compound units that could be reusable, and were also interchangeable. We wanted to ensure that the server units or display units could be replaced by others which would provide different functionality for SR and TTS servers. Overall, we were pleased with the new and improved design that was more generic, and could be integrated with any client (Java, C/C++, etc.) capable of making a network connection.

Lessons Learned

We learned a bit more about how to use Jiazzi, namely writing correct unit and package signatures. One thing that frustrated us was all the linking necessary to support correct stubs for all package signatures. We assumed that Jiazzi would recursively search imported packages for their respective imports, but this was not the case. Therefore, every unit's imported packages must also be explicitly imported, which was frustrating. This led to some trouble initially, due to our assumption.

One valuable lesson learned was the importance of refactoring. During development of the SR and TTS servers, many quick hacks and significant patches were made. But, no true refactoring was ever performed on the code during this time. After successive changes, the code becomes cluttered, and some original design boundaries get crossed. With the help of refactoring, much of the design flaws born from iterative development can be fixed.

This project gave us the ability to experiment with the features of Jiazzi on a larger scale than the previous assignment. With ten total units in the final design, we were able to see in more detail some of the advantages of separate compilation and external linking.