Interfacing Dragon Naturally Speaking with a Lisp Text Processing System

Project Report

Alex Rothberg

Introduction

We built a system to interface Dragon Naturally Speaking (DNS), a commercial speech recognition package, with a text processing system, currently being written in Allegro Common Lisp (ACL). This link is required as part of a project to capture and transcribe doctor-patient dialog [1]. The problem is non-trivial as DNS is strongly coupled to Windows-only (proprietary) technologies whereas the speech processing system is written in Lisp. Interfacing these technologies is hard. The problem is further complicated by the fact that DNS was designed for dictation and thus expects to hear only one speaker, whereas our target application involves a two-party conversation. Further there are restrictions in running multiple instances of DNS on one machine.

More generally the goal of the project is to allow an engineer to use the best available technology or language for each module in a system. In the case of this project, while DNS is the optimal technology for speech recognition, Lisp was chosen as the language in which to build the text processing engine. Historically the choice of technology for one module would significantly limit the options for the others. The goal of our project is allow both technologies to be used harmoniously. Further we want to do so without limiting future infrastructure or deployment options.

In order to solve this problem, we use Microsoft's .NET Framework as an intermediary to export the DNS interface using RPCs. We use a client-server model with a .NET program that interfaces with DNS as the client and the Lisp text processing engine as the server. We were successfully able to transmit the full

set of data captured by DNS to Lisp. Further this communication can occur either between processes on one computer or between computers. While we are currently interfacing with Lisp, the technology choices that we made allow us to interface with a wide range of languages.

In the remainder of the paper we will address the project in more detail. In the Background section we will discuss the technologies involved in the problem as well as previous or alternative solutions to the problem. In the Solution section we will discuss the details of our proposed solution and in the Results section we present out findings. We end with a brief discussion of the advantages of our approach and then conclude.

Background

The project is to serve as the link between two modules in a larger system. The purpose of the larger system is to capture, transcribe and process patient-doctor dialog. The dialog may be captured from microphones attached directly to a computer or as audio files recorded on a handheld device. The speech recognition is done by Nuance' Dragon Naturally Speaking (DNS). The text processing module, known as Lisp Architecture for Text Engineering (LATE), is being written by Peter Szolovits (at CSAIL). LATE is written in Allegro Common Lisp (ACL). This project serves to transfer the data output by DNS to LATE.

Dragon Naturally Speaking takes audio either from a file or captured from a microphone and converts it to text. For the purposes of this project we are working with the DNS Software Development Kit (SDK)[1] and not the standard dictation package sold in stores. The SDK offers a richer set of information than just text; it provides alternative interpretations of a given phrase along with confidence scores for the individual words. The SDK can be called from "any language that supports Active X and COM, including C++, C# and Visual Basic" [2]. DNS was designed for personal dictation and hence is not optimized to

[1] Specifically we are working with version 10 of DNS.

capture text from the speech of more than one speaker. The software was architected such that two instance of DNS cannot run simultaneously on the same computer.[2]

ActiveX and COM are two Microsoft technologies to allow inter-process communication. They are built for the Windows operating system. Historically they have been difficult to work with, resulting in the term “DLL hell” [3]. Improper use of the controls can lead to software and system instability.

Remote procedure calls (RPCs) are a means of abstraction based upon the standard notion of function or procedure calls. Unlike the standard intra-process procedure call, an RPC takes place between a calling “client” process and a remote “server” process. The server may be another process on the same machine, or a network connected machine. [4] SOAP is an XML based RPC protocol. One of the advantages of SOAP is that a large range of languages and frameworks have support to act as both a SOAP server and client. [5] Both ACL and .NET have either partial or full implementations for both of these roles.

Previous work on a similar interfacing problem was done by Klann [6]. His project involved interfacing DNS with GATE (another text processing engine written in Java [7]). His solution was tied to a Java processing engine and cannot easily be changed to work with Lisp. According to professor Szolovits, the solution was not reliable because of its dependence on many fragile components, including a COM-Java bridge and a RAM-disk for intermediate data storage; thus, it tended to fail after running for extended periods of time.

LATE is being developed in Allegro Common Lisp (ACL), a product of Franz Inc. ACL runs on a wide range of operating systems including Mac OS X, Linux and Microsoft Windows. As mentioned above it supports SOAP, as well as XML-RPC (a predecessor to SOAP). Running on Windows it has some ability to host

[2] Theoretically it should be possible to run multiple instance of DNS on one computer if each were running in a separate (virtualized) guest OS. The difficulty would then arise in handling multiple microphones.

OLE/OCX controls (these are a subset of ActiveX). In addition ACL has a "foreign-function interface" which "allows one to link compiled foreign code dynamically into a running Lisp." [8]

Solution

Given the constraints of getting an ActiveX control to interface with ACL, there are two broad solutions to this problem: directly interface the two or employ one or more intermediary technologies. The fact that ACL is designed to run on many different operating systems, but DNS on just Windows, is a strong indication that the direct interface will likely not be a practical solution. As mentioned above, ACL does have an "OLE Interface"; however, the documentation is poor and the interface appears to be designed for hosting UI controls. [9] An alternative would be to use the foreign-function interface along with a tool such as SWIG [10] to allow Lisp to call into the ActiveX DLLs. Again the documentation is sparse and reliability would likely be an issue. In addition, both of these direct approaches would tightly bind our interface to Lisp in general and ACL in particular.

The alternative to directly hosting the DNS control from within ACL is to use an intermediate "host program" into which the DNS control will be embedded. This program will then communicate with ACL. There exist a large number of languages and frameworks that can host ActiveX controls[3]. There are then two primary means of interfacing this program with ACL. We can use the foreign-function interface, or we may use RPCs (both discussed above). Foreign function calls are the simplest and will have the smallest overhead. We would have either the Lisp program call (directly) into the host program or vice versa.

We decided against such an approach for two reasons. The first is that it limits the system's design. This approach requires that both the Lisp and host program (along with DNS) run on the same machine. This does not allow us to use multiple machines, if that becomes a requirement when processing dialog. The

[3] These include: C++, C#, Borland Delphi, Visual Basic, and even Java, although this requires a Java-COM bridge.

second reason is that it ties the solution to an ACL based text processing system. For example if the group decided instead to use GATE, a preexisting speech processing framework, the link would need to be rewritten.

The alternative to foreign function calls is to use remote procedure calls. Like foreign function calls, they are easy to consume; on both ends they are used just like any other function in the host languages. Invoking will require some additional computational overhead. The overhead consists of: serializing the structure containing the transcription data to XML, sending this XML over HTTP to a (local) server and then de-serializing from XML. The slowest component in most RPC is the transmission time. Currently we are running the server on the same machine as the client; we should expect the entire round trip time to be less than 100ms and have minimal impact on CPU usage (we evaluate the actual performance below). Even with the added overhead, RPCs have neither of the disadvantages of remote function calls listed above. Unlike the remote function calls, RPCs can communicate between machines and with any number of platforms.

We pursued a two step solution to the problem: have a program written in Microsoft's .NET talk with the DNS SDK and then have the same .NET program communicate with Lisp by way of RPCs. As noted on the DNS SDK website, C# (.NET) can interface directly with the SDK [2]. Our solution uses a client-server model, with the .NET application as the client and the Lisp speech processor as the server. The two communicate using SOAP. We can makes calls either synchronously (waiting for the server to respond) or asynchronously, i.e. sending the RPC and then moving on.

We use a client-server model with the client being DNS and the server the text processing engine. While we could have reversed the roles, we feel this would have been inappropriate. The content is being generated at the DNS client at non uniform times. In this manner, DNS can push the content to the server as it becomes available. If the roles were to be revered, the text processing engine would have to

constantly poll DNS for newly available content. Further, this approach might require a buffer on the DNS end, making the implementation far more complicated. Further by making the text processing engine the server, the approach is far more suited for handling multiple instances of DNS communicating to the same server. This might become applicable if it turns out that in order to transcribe dialog two instances of DNS are needed, but one text processing engine is used to process the entire dialog. We chose to use SOAP over alternatives such as XML-RPC for two reasons. SOAP is natively implemented on .NET whereas XML-RPC would require 3rd party libraries. SOAP also supports a very rich data structure whereas XML-RPC does not. [11]

Discussion and Results

While SOAP is billed as a cross language protocol, getting .NET and ACL to communicate correctly was non-trivial. There have been numerous versions of the SOAP specification. Within a given version, there are multiple ways to encode data. ACL does not support the newest version of SOAP (1.2) and only partially implements the 1.1 specification. This was especially challenging because the server is the module written in ACL and generally we expect weaker restrictions on what the client sends and stronger specifications on what the server will accept.

In order to evaluate the success of our implementation, we considered two criteria: performance and stability. In considering performance we evaluated the latency that the link introduced to the system as well as the overall system costs of the link. In order to evaluate latency, we instrument our code. We look at how long it takes to get data from the DNS SDK into our data structures, to serialize these data structures to XML, and then to send them. We measure how long it takes the LISP server to de-serialize and then respond. Surprisingly the slowest component was the first: getting the data out of the DNS SDK. On the phrase “the quick brown dog jumped over the lazy fox,” this stage takes on average 280ms. Serializing that data and then sending (asynchronously) to the server takes only 10ms. Serializing and

sending the data to the server, and then waiting for a response (a synchronous call), takes only 32ms. The time required to get data out of the DNS SDK is likely due to the overhead in marshalling data from an ActiveX control to .NET. When running the transcription software while it was connected to the LISP server, we monitored CPU and RAM usage. Neither changed appreciably when sending data to the server was disabled.

Since stability is a negative goal, testing for it is difficult. During our tests, we did not observe any stability issues. Further the .NET to DNS communication was based on sample code provided by Nuance. The .NET SOAP interface is widely used. The one point of concern is the ACL SOAP server. While the set of features that it supports are limited, we observed no problems with its stability. Further, given that we are using RPCs, the communication link is relatively stateless. This helps to improve stability and minimize system resource usage.

We end this discussion with a brief mention of the system limitations. Given the client-server model that we have chosen, there is no good way for the text processing engine to provide feedback to the speech processing engine. This problem can be solved in a number of ways. We can have the client poll the server on some interval for feedback messages. We can implement a second client-server channel, but with the roles reversed to allow bi-directional communication. Alternatively we can relegate all communication in this direction to the return value of the RPCs.

Conclusion

We successfully interfaced Dragon Naturally Speaking (DNS), a commercial speech recognition package, with a text processing system written in Lisp. We did this as part of a larger project that involved transcribing, eventually in real-time, patient-doctor dialogues in an examination room. We accomplish this using an intermediate program, written using Microsoft's .NET. This program acts as a SOAP client that communicates using RPCs with a server written in Lisp. Our solution allows the speech recognition

and the text processing modules to run on different machines. It also allows multiple instances of DNS to use the same text processing engine. The platform for the text processing engine can be changed with minimal work.

By employing SOAP we have made the text processing engine language agonistic. While the current proposal is to interface DNS with LATE (in Lisp), we can interface with GATE (Java) or any other future text processing engine. By using RPCs we do not constrain ourselves to run the speech recognition and text processing software on the same machine. This will allow us to run multiple instances of DNS tied to one text processor.

Works Cited

[1] P. Szolovits, "3039286 NIH ILF Application".

[2] Nuance Communications, Inc. (2009) Dragon NaturallySpeaking 10 SDK. [Online]. http://www.nuance.com/naturallyspeaking/products/sdk.asp

[3] Wikipedia. [Online]. http://en.wikipedia.org/wiki/DLL_hell

[4] A. D. Birrell and B. J. Nelson, "Implementing Remote Procedure Calls," *ACM Transactions on Computer Systems*, vol. 2, no. 1, pp. 39-59, Feb. 1984.

[5] SOAP. [Online]. http://en.wikipedia.org/wiki/SOAP

[6] J. Klann and P. Szolovits, "An Intelligent Listening Framework for Capturing Encounter Notes from a Doctor-Patient Dialog," in *IEEE International Conference on Bioinformatics and Bioengineering (BIBE' 2008)*, 2008, pp. 70-74.

[7] Natural Language Processing Research Group at the University of Sheffield. GATE, A General Architecture for Text Engineering. [Online]. http://gate.ac.uk/

[8] Franz Inc. (2008, Dec.) Allegro CL version 8.1 documentation. [Online]. http://www.franz.com/support/documentation/8.1/doc/contents.htm

[9] T. R. Hinrichs. (2003, Aug.) Embedding ActiveX controls in Common Graphics windows. [Online]. http://osdir.com/ml/lisp.allegro/2003-08/msg00003.html

[10] Simplified Wrapper and Interface Generator. [Online]. http://www.swig.org/

[11] C. Cook. (2009, Feb.) XML-RPC.NET. [Online]. http://www.xml-rpc.net/faq/xmlrpcnetfaq.html