Making the Voice of Tcl Heard: Experiences in combining Tcl with voice software

Spyros Potamianos
Manolis Tsangaris

Bell Laboratories – Lucent Technologies
Murray Hill, NJ 07974

Alexios Zavras

Athens University of Economics and Business
Athens, Greece

{sjp,mmt,zvr}@research.bell-labs.gr

ABSTRACT

This paper presents some of the work performed towards integrating Tcl with voice interface elements, for both input and output functionality. Low-level extensions provide Tcl with the appropriate commands in order to use voice software systems. On a higher level, abstractions were developed around the main idea of a voice server that the Tcl scripts can use as just another available resource. The ultimate goal is to create simple and efficient multi-modal interfaces, incorporating both graphic and voice user interface elements.

Introduction

Tcl is a well-known and widely-used scripting language. It has been shown to be of use in many different environments, adhering to different constraints and catering for different needs. Besides the merits of its own as a scripting language, Tcl is widely known and used for its incorporation of Tk, a toolkit for creating graphical user interfaces (GUIs). The toolkit allows the user to easily create, manage, and specify the functionality of graphical user interface elements, such as buttons and scrollbars. Based on these simple elements, more

complex ones can be designed and implemented, and many such have already been created, such as tables and tree representations.

However, besides graphical user interfaces, there are other user interface areas that could be used with Tcl. Voice interface elements are becoming increasingly common, aided by the advances in technology for voice processing, as well as by new devices. This paper presents some of the work performed towards the greater goal of integrating Tcl with voice interface elements and thus creating a simple scripting language appropriate for the development of voice-based applications.

Voice output

In order to add voice output to an application, the audio content must be generated. There are two ways of generating this content: either off-line or on real-time. When the voice output is generated off-line, it is stored as audio files in some storage (nowadays usually disks), which are then simply played back when needed. The generation of the content can be performed either by computer or by plain sound recording of human voice.

When voice output has to be generated on real-time, the software used is usually called Text-To-Speech (TTS) system. Such a system accepts textual input and produces the corresponding audio data, which are usually sent to an appropriate device, such as the speakers.

Voice input

Adding the capability of voice input to an application is not so straightforward, and can be performed in varying degrees of complexity. A simple first-level solution is adding the ability to recognize of a small set of well-defined tones, such as the ones used in telephony (DTMF). This is the basis of the operation of many Interactive Voice Response (IVR) systems, which address the user with (usually) pre-recorded prompts and accept DTMF tones to denote menu choices.

Handling human speech necessarily involves much more elaborate procedures. These procedures pertain at the lowest level to audio signal processing, like background and surrounding noise cancellation. The next level is to achieve speaker independence, and therefore be able to process voice input from a variety of individuals. Furthermore, content processing is also necessary, with components like vocabulary and dictionary definition, grammar specification, and various parts of natural language understanding.

The systems capable of performing such human voice processing are usually called Automatic Speech Recognition (ASR) systems. Their operation can be

described in general terms as accepting audio data plus a number of parameters defining the context, and producing some textual representation of the speech data. As can be easily understood, the capabilities of such systems available today vary greatly.

Application taxonomy

As described above, a large number of applications can be enhanced by the addition of the capabilities of voice output and/or input. According to the features available to an application, it can be categorized in one of the cells of the following table. Each cell contains a vector notation, based on its input and output functionality, to facilitate further reference to it.

		Output		
		GUI only	Voice only	GUI & Voice
Input	GUI only	*(G,G)*	*(G,V)*	*(G,GV)*
	Voice only	*(V,G)*	*(V,V)*	*(V,GV)*
	GUI & Voice	*(GV,G)*	*(GV,V)*	*(GV,GV)*

Some of the cells of the above table correspond to obvious cases: the top-left one, *(G,G)*, represents a traditional GUI application, while the central one, *(V,V)*, represents a telephony application. On the other hand, some cells do not represent useful set of functionalities. For example, once an application has a GUI input method, there is no point of restricting the output to be voice only. Therefore, the cells *(G,V)* and *(GV,V)* in the middle column are not useful.

Our work was primarily targeted into adding voice functionality to GUI applications, i.e. to provide some, if not all, of the functionalities of *(V,V)* in conjunction with the ones of *(G,G)*. The two most "interesting" cells were the top-right *(G,GV)* and the bottom-left *(GV,G)*, which correspond to adding either voice output or voice input to a GUI application. The ultimate goal is naturally the bottom-right corner, *(GV,GV)*, of complete GUI and voice integration.

Tcl extensions

The initial stage of our work consisted of creating communication mechanisms between Tcl and existing TTS and ASR systems, which provided the handling of voice output and input, respectively.

Towards this goal, we developed the appropriate extensions, so that Tcl scripts could make use of the functionalities provided by such systems. As an easily understandable example, a simplistic scripting interface to the TTS functionality was provided by the `::tts::say` procedure, which naturally generated speech in audio format for its arguments. In addition to this principal

function, a number of other programming interface points were also developed, so that full configuration of the systems used was possible.

A major design decision during this phase of the project was that the extensions would not incorporate the desired functionality into the Tcl process. In other words, the Tcl interpreter would not be "augmented" with TTS and ASR functions. On the contrary, the extensions were designed and implemented so that they provide a general communication mechanism to externally running TTS and ASR systems. In this way, our extensions are not specifically bound to a particular implementation of such systems, nor, perhaps more importantly, are they constrained to a single computing system. The TTS and ASR systems that can be used by the interpreter can reside on different machines, reachable by network, provided, of course, that the communication path between the machines satisfies some quality constraints.

The extensions present a scripting interface that is completely under namespace control, either the `::tts` or the `::asr` namespace. They also present an object-oriented framework, where the external systems are treated like objects (or more accurately resources) that can be configured in a multitude of parameters and can perform a number of distinct actions. This approach is not unlike the one used in Tk, and designing and implementing systems with it is natural to Tcl/Tk programmers.

By using the primitive operations defined in the extensions, it was possible to easily design and implement applications which fully supported voice input and output. The reputation of Tcl as the "ideal glue language" turned out to be fully justified at this point. The development time was remarkably short and the resulting scripting code was clear and exceptionally readable.

Higher-level abstractions

Although the aforementioned extensions permit the use of voice-processing software systems from Tcl scripts, it was felt that a tighter integration was needed. Therefore, an abstraction of a higher level was defined, tentatively called Tcl/VUI, standing for Voice User Interface. Tcl/VUI is a set of abstractions that is designed to facilitate development of voice user interfaces. Its design was mostly inspired by the Tk approach, regarding graphical user interfaces, by VoiceXML, regarding voice input and output primitives, as well as by experiences from different speech dialog systems.

Although Tcl/VUI is complete, in that fully-functional applications handling voice input and output can easily be implemented, it is not necessarily the ultimate fully-configurable superset of all voice interface instances. This was a deliberate decision, since the main design objective was to make the core

functionality of voice systems easily available and attractive to use for both users and user interface designers.

Tcl/VUI is centered around the idea of a core main abstraction, that of the voice server. Similar with the case of the server in window systems, the voice server system handles the low-level events, such as capturing and producing voice streams, detecting DTMF tones, and performing voice recognition and text to speech translation, as well as reproducing pre-recorded voice prompts.

The voice server provides access to the audio hardware, which can be desktop or telephony based, to the text to speech subsystem, to the automatic speech recognition engine, to an audio player/recorder, and to an audio activity detector. All these subsystems can be used independently through Tcl in a synchronous or asynchronous mode. In the latter case, the Tcl event mechanism is utilized to provide callbacks to the VUI interpreter.

The communication between the Tcl interpreter and the voice server is performed in a network-transparent way, so that these two system may not be co-existing in the same physical machine. The same is also true for some of the devices that the voice server is responsible for handling. By allowing network connections and the equivalent of remote procedure calls between the different components, the overall system implicitly gains a high scalability factor, since parts of it can be implemented in different machines.

In its present state, Tcl/VUI is working with at least two different implementations of the voice-processing systems. We are currently developing bindings to Tcl/VUI that correspond to several commercial and research subsystems, each time testing the proposed framework for its capability to capture the essential features of those systems. So far, we have been very pleased with the way Tcl can be used in those contexts, resulting in the rapid prototyping of speech applications.

Multi-modal interfaces

The larger goal that we are pursuing is the integration of Tk and VUI, in order to create interfaces that use both graphical and voice interface elements. As mentioned above, the addition of voice input and output processing to a GUI application is a simple procedure, once the corresponding extension and libraries are used. However, this does not accurately constitute on the design and implementation of an integrated user interface, which provides two interaction modalities, namely based on graphics and voice.

As an example, one could consider the lowest graphical user interface elements, which are implemented as Tk widgets. Such elements, like an entry field where the user can type a string, or a listbox where the user can select one of a

set of pre-defined strings, are straightforward to use and their functionality is easily understood by both the user and the interface programmer. Both these example widgets can benefit from voice input processing, so that the user can speak the string to be entered in the entry field or the selection from the listbox. Therefore, one can envision a multi-modal user interface element, extending the listbox widget with voice input functionality. This element accepts a list of choices, presents them graphically to the user in a listbox, and awaits the selection of one of them, either via the graphical interface, or by the user voice. A corresponding multi-modal input widget can be envisioned based on the entry field.

The example that was just described presents a much simplified situation, where a large number of important details have purposefully been left unspecified. One of these aspects is the handling of focus, which, in the case of graphical user interfaces is specified by the use of pointing device, such as a mouse. In the case of multi-modal interface elements, the same pointing device could be used to determine the element that will have the voice focus, i.e. that will process any voice input and accept its textual representation.

However, in the cases where more than one such elements are present, the seamless integration of all voice input will probably result in a much better user experience. The user would then be able to speak sentence fragments or even complete sentences, and the corresponding textual representation would be split and forwarded towards the appropriate interface elements. This approach obviously presumes a much tighter integration between different elements than the simplistic one defined by the geometry placement of graphical interface elements.

Future work

We are currently actively exploring the abstractions necessary to create multi-modal user interfaces that are efficient and easily used by users and programmers alike. We believe that the basis for the development of such interfaces lies in the successful coupling of voice and graphical elements into multi-modal user interface elements.