TclBLASR: An Automatic Speech Recognition Extension for Tcl

Qiru Zhou, Jinsong Zheng, Chin-Hui Lee

Dialogue Systems Research Department, Bell Labs, Lucent Technologies
600 Mountain Ave., Murray Hill, NJ 07974, USA
{qzhou,jinsong,chl}@research.bell-labs.com

Abstract

We present TclBLASR, a framework to integrate a proprietary speech recognition engine, an open source script language, such as Tcl/Tk and an open source sound analysis toolkit, such as Snack from KTH, into a user friendly platform that a user can write a Tcl/Tk script application quickly for speech recognition evaluation, speech data collection and automatic annotation, and speech technology demonstration. This framework is extremely useful for third party customer evaluation of speech technologies that do not involve heavy C/C++ program development and extensive knowledge on low-level speech engine APIs. Using the Bell Labs Automatic Speech Recognition (BLASR) engine, coupled with the real-time audio I/O and visualization provided by Snack and the flexible graphical user interface tools embedded in Tcl/Tk, the TclBLASR platform proves to be a useful framework for quick packaging of ASR engines for customer evaluation of the technology without extensive customization of interfaces to meet different needs from a wide range of customers.

1. Introduction

Last year, we received requests from our customers to evaluate Bell Labs speech recognition technologies in a scenario that data collection is also required. It demands us to develop a real-time speech recognition prototype system with the capability to record digitized speech on computer disk using general purpose PCs with an audio input functionality at either 16KHz or 8KHz sampling rate while performing the real-time speech recognition function. In order to organize speech data and transcriptions during data collection for further offline analysis, the trial also requires recording of recognition result along with the system prompts as the reference orthographic transcription. In the following, we briefly describe features of BLASR, that requires an expert knowledge to develop interface and applications. Therefore, we need a publicly available software package that fits our need. Furthermore, to make the development outcome reusable for future projects, the issues related to portability, interoperability and open software platform need to be taken into account in our selection. A brief survey indicates that an open source script language like Tcl/Tk [5] or Python [7] is an excellent graphical user interface (GUI) tool for easy of use. In the meantime, we also found an open source sound and speech tools package, called Snack, from KTH [1, 2]. The combination of an efficient Snack C library, wrapped as a dynamic loadable library (DLL) under Windows/NT and a shared library (.so) under Unix/Linux for underlining heavy-duty speech and signal processing and interface to the popular Tcl/Tk and Python script languages, makes Snack a highly efficient toolkit for rapid speech tool and application development. Given the cross-platform nature of the Tcl/Tk and Python languages, an application developed on top of Snack is cross-platform as well. We quickly settled with the choice of Snack and Tcl/Tk.

In this study, we propose an architecture and present the development of TclBLASR, an integrated platform that offers the speech recognition and data recording functionality in addition to a set of user-friendly GUI features. Its simple and easy-to-use interface proves to be a good platform for quick prototyping and packaging of speech processing technologies.

We believe our design and development experience is useful and generic for any group who wants to integrate speech recognition or other speech processing engines with Snack and Tcl/Tk with minimum effort. We also believe the speech community will benefit from the proposed software architecture design so as to quickly bring proprietary speech technologies to the general public for wide acceptance.

2. Feature Specifications for TclBLASR

In the following, we briefly describe features of BLASR, Snack and Tcl/Tk that make them fit well together in a user-friendly software platform. The initial development work of TclBLASR was based on Snack 1.6 and Tcl/Tk 8.2. It is also extendable to Python [7] since Snack uses the Tk widgets [5].

We use a version of BLASR, called CompactASR, with a simple application programming interface (API) that fits well into the client-server paradigm in which the CompactASR engine acts as a server. It supports sharable resources in an asynchronous, event-driven, and multi-threaded software environment.

As described in the Snack web page [1], it provides a wide range of features. Some of them are highlighted as follows:

- Support for in-memory sound objects, file-based audio, and streaming audio
- Multiple simultaneous playback and recording threads (system dependent)
- All audio data handled as floating point internally for fast and accurate computations
- Multi-platform, same scripts usable on Windows 95/98/NT/2000, Linux, Macintosh, Sun Solaris, HP-UX, FreeBSD, NetBSD, and SGI IRIX
- Filters which can be used to process sound objects or during playback for on-the-fly operation
Snack is licensed under GPL (GNU Public License) with complete source downloadable from the Snack web site [1]. Its rich set of sound and speech handling features gives us great possibilities to expand our development work for future software tools, such as automatic speech annotation, multimodal dialogue application prototyping. Beside Snack is a collaborative research and development effort that we would also like to contribute to benefit the audio and speech research community. One such example to integrate speech transcription tool using Snack can be found in [3].

One great advantage of the Tcl/Tk software package is the ability to incorporate user-defined commands. Developers can add their own extensions to the Tcl interpreter that:

- Use compiled code to perform computational intensive algorithms;
- Create a rapid prototyping interpreter for an existing compiled language library, which makes it easy and quick to write applications;
- Add powerful Tk graphics to existing applications and function libraries;
- Create a script-driven test suite quickly for an application or a library algorithm;
- Portable for common general-purpose computer platforms.

3. TclBLASR Framework

For applications or algorithm libraries written in C/C++, there are two ways to build extensions to Tcl interpreter, namely: (1) Create a new interpreter (wish shell) that has new command add-ins; and (2) Create dynamic libraries, DLL for Windows 9x/NT4/2000, shared libraries (.so) for Unix/Linux, whose functions can be used by the interpreter after loading.

Starting from version 8.0 of the Tcl/Tk package, the extension developers only need to code DLLs and application scripts that use the DLLs. In this paper, we use the second method without changing the standard Tcl/Tk distributions (We tested the framework on Tcl/Tk 8.2-8.3 and TclPro 1.4).

Snack Sound Toolkit has two main components: Snack Tcl/Tk script widget library for Snack GUI and application scripting and the Snack C dynamic library for underlining signal processing. The Snack C dynamic library provides two interfaces: the Snack command extension to Tcl, and the Snack C interface to applications or libraries. Through the Tcl command extension, Tcl script function modules can be registered as callbacks to the Tcl/Tk package. The Snack C interface follows the Tcl Extension Architecture (TEA) [9]. Snack provides a set of callback management functions that allow another library to register its functions with any Snack sound object as callbacks to the library. Snack also provides functions to access audio data stored in a Snack sound object and the state of the object. Snack sound objects are identified by names and accessible through Snack_GetSound(). So we have adequate access to a Snack sound object at runtime to integrate it with BLASR real-time recognition engine.

Snack also provides Snack_AddSubCmd() to allow developers to add new sub commands to the Snack package. Therefore, it is possible to add BLASR to Snack at the sub-command level to extend the Snack package. In the current TclBLASR architecture, however, we chose to make BLASR a direct extension to Tcl/Tk instead of Snack due to the following considerations: (1) Reduce the level of indirect reference to reduce Tcl command latency; and (2) Make it easy to integrate with other packages.

The remainder of this section describes (1) the main steps to create a real-time speech recognition engine Tcl extension using the Snack C library; and (2) the runtime call flow among the components of the framework.

3.1. BLASR Extension to Tcl Using Snack

A Tcl extension DLL must perform the following tasks [8]:

- Initialize persistent data structures
- Register new commands with the Tcl interpreter
- Accept data from the Tcl interpreter
- Process the new commands
- Return results to the calling scripts
- Return status to the calling scripts

The Tcl entry point of the BLASR extension DLL is an initialization function called by the Tcl interpreter when the extension DLL is loaded. This initialization function performs the following simple tasks:

```c
// define the package
Tcl_PkgProvide(interp, "lasr", "1.0");
...
// register "recognizer" command with the Tcl interpreter
Tcl_CreateObjCommand( interp,
"recognizer",
BLASREngineCmd, NULL,
(Tcl_CmdDeleteProc *)NULL);
```

For server based ASR engines, it is necessary to provide a Tcl_CmdDeleteProc function pointer to release the resources when an extension DLL is unloaded.

The BLASREngineCmd() function is the actual function that gets called when the Tcl command recognizer is used. It initializes the BLASR engine, opens a recognizer channel, creates a persistent data structure that serves as the context object during runtime and contains the following contexts:

- A pointer to the Tcl callback script
- A pointer to the Tcl interpreter
- A pointer to the Snack sound object
- The id and state of the recognizer channel
The recognition result
• Signal processing parameters.
• Residual speech samples left from the last block processing.

The BLASREngine_Cmd() function also calls Snack_AddCallback() to add the ProcessUtterance() function, with the context object as an argument, to the callback list of the Snack sound object. The Snack sound object will call the functions in the callback list whenever it detects a content change, as when a new block of speech samples is available.

The controlling Tcl application script is listed below:

```tcl
... # create the application root window
wm title . "TclBLASR Speech \ Recognition/Data \ Collection"

# load the Snack and BLASR extensions to Tcl
load libSnack.dll
load TclBLASR.dll

# create a new Snack sound object named "snd"
sound snd

# use Snack widgets for speech signal display
source snack_widget.tcl

# create a recognizer command object named "recog"
set conf_file demo.flg
recognizer recog $conf_file snd \ asr_callback

asr_callback in the above script is a Tcl script module. It has a reference to the main Tcl/Tk window and serves as the callback from the TclBLASR extension to the Tcl package.

3.2. The Runtime Call Flow

Among all the components of TclBLASR, the main Tcl/Tk window is the master component, by which a user can control the start of the recording of an utterance and see the results of the recognition processing.

When the user initiates the recording of an utterance, the Snack sound object starts collecting speech samples from the source and calls ProcessUtterance() when a new block of samples is available. ProcessUtterance() obtains the new block of samples from the Snack sound object and prepares it in a way suitable for consumption by the recognizer channel. Because the recognizer channel accepts data in blocks of certain sizes, ProcessUtterance() must save the samples that cannot fit into a block acceptable by the recognizer channel. Such samples are residual samples to be prefixed to the next incoming block. When the Snack sound object determines that there are no more incoming speech samples, it also calls ProcessUtterance(). This time, the function gets recognition results from the recognizer channel and passes them to the Tcl callback script module. The Tcl callback script module updates the main Tcl/Tk window with the recognition result.

Figure 2 illustrates a sample TclBLASR speech recognition and data collection application call flow during runtime.

Besides displaying recognition results in the main Tcl window, the Tcl callback script module can also prompt the user for the next utterance. In a data collection application, the Tcl callback script module can use the write command of the Snack sound object to save the speech data into a file.

4. Data Collection & ASR Application

An easy to use speech recognition evaluation and data collection application is built for our customer by using TclBLASR. Its GUI interface consists of two windows: a session login window allows the speaker to enter his/her ID, gender information, data recording sampling rate and maximum utterance length for recording (Figure 3). After the speaker entered the session information, the “Start session” button should be pressed to close the login window and to start the main speech recording window which displays the...
input signal waveform and spectrogram of the signal (optional). For each speaker, a unique transcription list is loaded and prompted by the “Please say” item on window. A record button is provided to let the speak control his or her own pace to speak. For the actual speech recognition evaluation, we hide the recognition result to avoid unnecessary confusion and hint to do artifact correction.

5. Conclusions
By combining the set of Snack sound toolkit functions with BLASR recognition technology, along with the Tcl/Tk scripting application building, we have developed TclBLASR in a short time and the outcome that satisfied our customer needs. We also did some in-field fast application modifications based on new features that customer wanted. It turns out that by embedding speech recognition engine under Tcl makes it a powerful tool to build speech application rapidly. Based on this architecture, we also implemented other speech technology demonstrations shortly after TclBLASR was developed. Without much modification, this framework may be applied to other speech engines with proprietary APIs, or common speech APIs such as Microsoft SAPI.

Acknowledgements
The authors wish to thank Dr. Ning Bi and Mr. Suhail Jalil for their valuable feedbacks and suggestions during a TclBLASR customer trial. We also thank Peter Cui for his work on the original TclBLASR code implementation.

References