A LANGUAGE FOR CREATING SPEECH APPLICATIONS

Andrew Pargellis, Qiru Zhou, Antoine Saad, Chin-Hui Lee

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

ABSTRACT

This paper describes an embedded Voice Interface Language (VIL) that enables the rapid prototyping and creation of applications requiring a voice interface. It can be integrated into popular script languages such as Perl or Tcl/Tk. Three levels of single-word commands enable the application designers to access basic speech processing technologies, such as automatic speech recognition and text-to-speech functions, without knowing details of the underlying technologies. VIL is a platform and domain independent speech application programming interface (API) that enables users to add a speech interface to their applications. The domain dependent components are defined by including a set of application specific arguments with each VIL command. Since the platform is an open architecture system, third party speech processing components may also be integrated into the platform and accessed by VIL.

1. INTRODUCTION

It has long been recognized that a voice user interface (VUI) would be a useful, effective means of human-machine communication. Various speech user interface demonstrations and applications have been developed. Typical functions include text entry, document editing, telephony interactive voice response (IVR), conversational transaction systems, voice command, and voice controlled information retrieval.

There are two major categories of voice user interfaces, multi-modal user interface and primary voice user interface. Applications in the first category include speech enabled desktop applications, voice controlled information query, and voice controlled transaction systems. Two typical applications in the second category are telephony interactive voice response systems and voice controlled access systems. In this paper, we will concentrate on the second category, which is the main voice user interface model for telephony applications.

We report here on the development of a simple, abstracted (platform and application independent) voice interface language (VIL) that facilitates the definition of a wide variety of applications that require multi-modal user interfaces, including speech. VIL is an embedded language module that enables researchers and application developers to add a speech interface to their applications rapidly. In current implementation, we embedded VIL into Perl (Practical Extraction and Report Language), a popular script language which has been widely used in research and Internet application development.

The VIL is built on top of our speech technology integration platform (STIP)[1]. A speech API interface class accesses platform speech processing functions[2]. The interface functions provide an advanced speech interface control such as ASR task definition, barge-in, dynamic grammar loading, and text-to-speech synthesis control. VIL also provides functionality to write speech dialogue applications. In addition, a World Wide Web (WWW) interface accesses the Internet, enabling applications to retrieve information and conduct simple transactions in real-time, as well as providing the necessary HTML parsing functions.

We show an example that demonstrates the domain and task independent nature of the VIL by including telephony, information, and messaging services. The script is simple enough that the VIL can be used for a variety of research projects including an extension to other languages such as Mexican Spanish[3].

2. SPEECH TECHNOLOGY INTEGRATION PLATFORM (STIP)

In recent years, high performance speech processing technologies such as automatic speech recognition (ASR) and text-to-speech synthesis (TTS) are becoming available for many applications. While there is continuous progress in speech processing core technologies such as speech recognition, speech synthesis, natural language understanding, and dialogue system study; it is still nontrivial to design and deploy a speech-enabled system for a wide range of real world applications, severely limiting the deployment of speech enabled applications. On the research side, we need a generic, flexible speech technology integration platform to prototype various speech applications to study real-world speech processing problems, such as speech recognition robustness, speech synthesis naturalness, human-machine spoken language dialogue design, etc.

To address these issues, we designed and implemented a prototype platform to integrate various speech technology components: speech recognizers (multi-types: finite-state/context-free, n-gram natural language, multi-lingual), text-to-speech synthesizers (multi-lingual), audio/telephony interface, grammar generation tools, and web client interface.

Some of the platform design goals are: integrate speech and internet technologies, create a research platform to study various multi-modal user interface issues, enable rapid development of VUI domain independent applications, scalable multi-channel distributed processing across heterogeneous systems, and integration of third party components into the system.

Figure 1 is the block diagram of the current platform. The system is a three-tier client-server architecture where the resource manager manages available service components and distributes these services to applications through a client application-programming interface (Client API). The Client API is a set of a few C functions that provides necessary speech user interface functions required in Figure 1 framework. By using

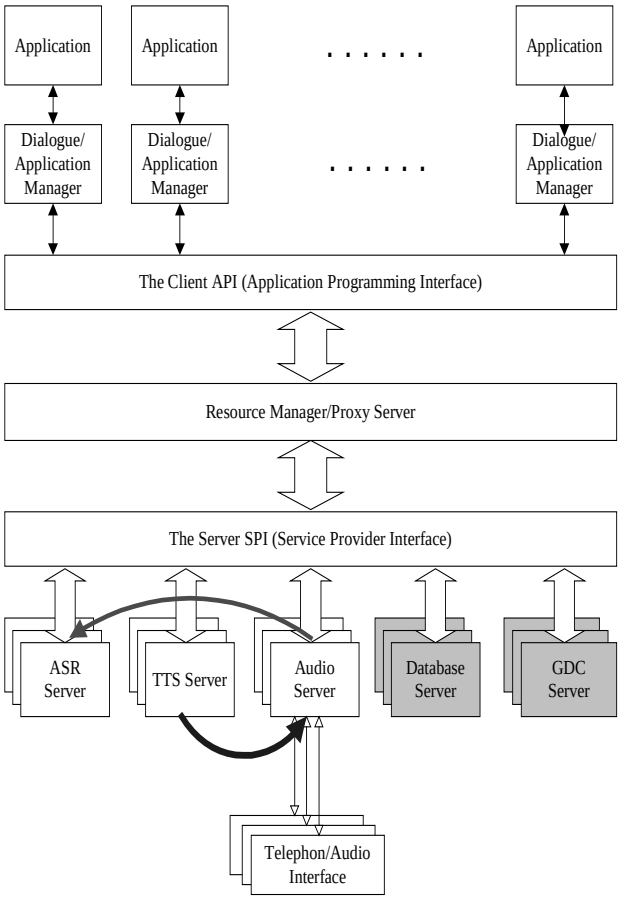

Figure 1: Bell Labs Research Speech Technology Integration Platform (STIP).

this API, a developer can create a speech user interface application without knowing the details of speech technology, audio or telecommunication interface hardware.

Gray blocks are under development. The curve connections describe the speech audio stream data flows from audio interface to ASR, and from TTS to audio interface. System component communications are based on asynchronous message passing mechanism to enable high throughput real-time performance.

Each application, consisting of a series of finite states, interfaces to the Client API through the Dialogue/Application Manager (AM). The various functions of the AM make up the VIL commands. This provides a high level, intuitive API for controlling the dialogue flow in a speech application prototype. The AM libraries consist of VIL commands, each one of which is an encapsulation of a Perl routine that performs some set of dialogue and application related actions. These functions comprise the Dialogue/Application Manager layer shown in Figure 1 and enable a developer to control the Dialogue flow between a human and computer at many different levels. For example, the developer can fine-tune the Speech Recognizer for a particular application, generate a multilingual dialogue flow, access web sites over the Internet, and log system and dialogue information in order to trace dialogue sessions. In all cases, the same set of domain independent VIL commands are used, with application specific arguments customizing the system.

Since Perl is one of the popular web common gateway interface (CGI) programming and web client script language, this approach also makes it easy to create multi-modal user interface applications (both speech and web based graphic user interface). By using SWIG (Simplified Wrapper and Interface Generator) developed by David Beazley of University of Utah, we generated a Perl speech interface function set easily from our client API C functions. SWIG enables developers to generate embedded speech user interfaces for other popular script languages such as Python or Tcl/Tk.

3. VOICE INTERFACE LANGUAGE

The purpose of the VIL is to enable developers to rapidly prototype applications requiring a speech interface to a human-machine dialogue. There are many applications, in different domains, that can utilize advanced speech technologies such as ASR and TTS. These domains themselves consist of many tasks, each with its own particular and special development strategy. For example, implementing a natural language call router for Call Center applications requires a backend ASR technology that can be different from one used for a personalized name dialer application. Every task is characterized by its own set of variables and parameters and must deal with a large number of issues that a developer must address. The VIL addresses these issues by grouping a series of domain independent commands that implement advanced speech technology functions. A customized application is readily built by passing tables of parameters and variables to VIL commands as needed. For example, the STIP platform and VIL architecture allow the ASR task to be dynamically selected at run time for each recognition instance.

One of the challenges is to provide a developer with the capability to fine-tune specific speech engines, such as ASR, while also providing higher level functions such as a prepackaged applet that downloads today's news from an Internet site. To this end; the VIL commands have been organized into three different levels of complexity, shown below in Table 1. The Low-Level commands are primitives that either send specific arguments to servers or perform specific Dialogue Management (DM) or Application Management (AM) tasks. The Intermediate-Level commands act like macros that group Low Level commands. The High-Level commands are mini-applications that group the lower two levels of commands.

One set of Low-Level commands generates computer audio prompts by sending an audio stream to the Audio server. So the VIL command, PLAYTTS, sends a text string to the TTS server, with the returned audio stream then sent to the Audio server,

whereas PLAYSND sends a prerecorded audio file directly to the Audio server. Similarly, several low-level commands enable access to the ASR engine for speech recognition. A typical recognition sequence would include the set of commands: {ASR_PARAMS, ASR_CMD, ASR_GET, ASR_PROCESS}. ASR_PARAMS sends a table of domain specific arguments to the ASR server for every recognition request via the Resource Manager/Proxy Server (RM/PS). The RM/PS pre-processes the ASR_PARAMS table, determines the ASR engine type requested, allocates the proper server for the duration of the recognition task instance, and passes all required parameters to that server. ASR_CMD sends a text string command to the ASR engine. An example is "SetGrammarPath $GramPath" which sends the required grammar file, tailored for a specific dialogue state, to the ASR engine. ASR_GET requests the ASR engine to convert the incoming audio stream to a text string and return that string. ASR_PROCESS then processes the string. There are several commands that perform Applications Management functions. For example, AUDIO_CTL performs a set of preprocessing functions, such as checking to see if a valid grammar file has been defined for states that request ASR. An output audio stream is then generated with the included PLAYAUDIO command. This routes the output audio via PLAYTTS or PLAYSND, depending on whether the developer is using TTS or prerecorded audio files. Similarly, ACTIONS enables a developer to include an optional set of special subroutines and SWITCH routes the application flow to a new state, depending on the recognized user's response.

Low Level	Intermediate Level	High Level
ACTIONS	AUDIO_CTL	PASSWORD
ASR_CMD	BARGEIN	WEATHER
ASR_GET	EXIT	YESNO
ASR_ PARAMS	LISTEN	
ASR_ PROCESS	PAUSE	
DATAFIELD	PLAYAUDIO	
ERROR_AUDIO	START	
HELP	WWWPROCESS	
LOG		
PLAYSND		
PLAYTTS		
RESET		
STATE_INIT		
SWITCH		
TAG_DATA		
WWWGET		

Table 1: Categorization, by level of grouping complexity, of some of the more commonly used VIL commands.

An example of an Intermediate-Level "macro" is LISTEN, which groups the following set of low-level commands: {ASR_PARAMS, AUDIO_CTL, ASR_CMD, ASR_GET}. A developer uses LISTEN to simultaneously provide audio output (computer query) and ASR recognition. The command, BARGEIN, consists of the following set: {LISTEN, ASR_PROCESS, ACTIONS, SWITCH}, which is a more extensive VIL command. BARGEIN is a powerful VIL command that allows VUI users to interrupt system responses with voice inputs, similar to the use of DTMF in a touch-tone IVR interface. This is especially useful in cases where the user wants to interrupt the system because of a recognition error or because the system's prompt is a very long voice message.

An example of a High-Level command is PASSWORD which is a miniature application that provides security access to a particular service and includes the BARGEIN command with associated ACTIONS. This applet includes several states that provide a dialogue between the computer and human, with the goal of providing a secure access to some service such as banking transactions through speaker authentication.

4. SAMPLE APPLICATION

4.1. VUI Demo – General Features

Figure 2 is a schematic of an application referred to as the "VUI Demo". Most VIL commands were originally developed for this application. All applications begin with a START state and end with the EXIT state. The START state is a single VIL command that opens Log files and server connections. The EXIT state (not shown here, but accessed from the MAIN MENU state) closes those files and server connections.

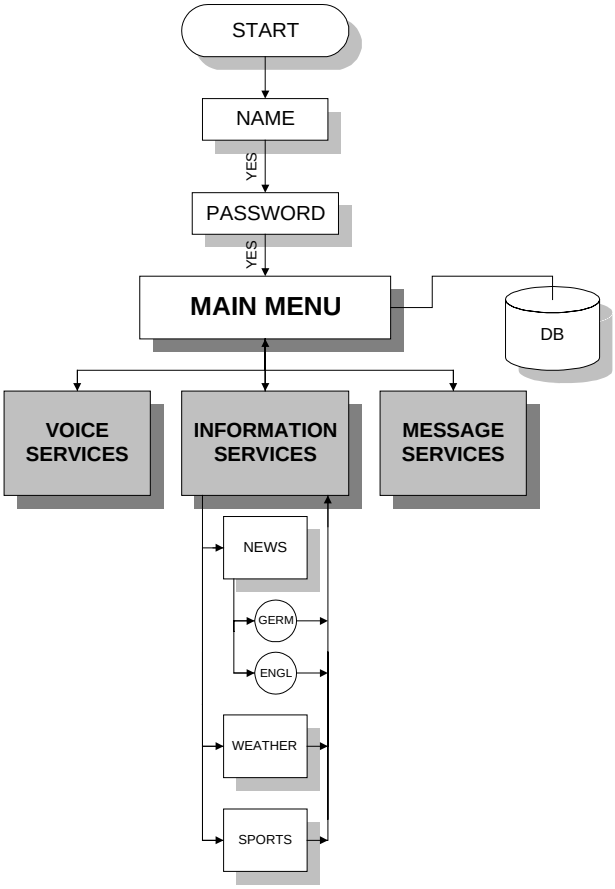

Figure 2: An overview of the Voice User Interface demo functionalities.

One of the most common types of states is the Menu state. This state includes the following VIL commands: {STATE_INIT, BARGEIN}. The BARGEIN command, discussed above, requires a series of application (domain) specific arguments. For example, the ACTIONS command requires a set of optional actions to be taken (if desired by the developer). The application developer also needs to provide the state name for STATE_INIT, as well as the status of some flags and the grammar name for the AUDIO_CTL command. STATE_INIT is a low level VIL command providing the Application Manager task of logging the state name and pushing the state name onto a stack (thus providing a history of states accessed by the user).

4.2. VUI Demo- VIL Implementation

The VUI demo is a good example of how some of the more commonly used VIL commands are implemented in a typical application. A typical application requires a START state that opens all required speech and telephony servers, a security or identification state, and then a Main menu state that is the top of the Dialogue tree structure. From Figure 2, one can follow the flow of the VUI demo from the START state to the MENU_MAIN state. The caller is then directed, from the Main Menu, to any of several services (Calling, Information, and Messaging) as shown. The traced flow of states (in an abbreviated form) is below.

```
START:{
      $StateName = "START";
      STATE_INIT;
      START;
      $tts_out = "Hello. I am your network assistant.\
                        You may ask for Help at any time.";
      AUDIO_CTL;
}

ENTRY_NAME:{
      $StateName = "ENTRY_NAME";
      $tts_out = "What is your name, please?\n";
      $GrammarStem = "Names";
      @TagArray = "name";
      @Action_BI = ("amTagData");
      STATE_INIT;
      BARGEIN;
}

ENTRY_PASSWORD:{
      $StateName = "ENTRY_PASSWORD";
      STATE_INIT;
      PASSWORD;
}

MENU_MAIN:{
      $StateName = " MENU_MAIN ";
      %RetArray = (
        "BYE" => "BYE",
       …..,
       );
      @Input = (set of possible speaker inputs);
      @NextState = (set of possible states to switch to);
      $GrammarStem = "Main";
      $tts_out = "Main Menu. Which service do you want?";
      STATE_INIT;
      BARGEIN;
      goto $NextState;
}
```

Each of the above states consists of a set of domain independent VIL commands, such as STATE_INIT and START, preceded by a set of domain specific arguments such as $StateName and $tts_out. The START state contains the VIL commands, {STATE_INIT, START, AUDIO_CTL}. These are useable for any application, with a table of arguments that tailor the actions to a specific state in a specific application, which can be as diverse as banking transactions, weather reports from the Internet, or voice dialing.

START is basically a macro consisting of a large number of server-specific “primitive” commands that open connections to various servers (ASR, Proxy, Telephony) in the Client API layer (see Figure 1).

5. CONCLUSIONS

By developing a general purpose speech technology integration platform to ‘glue’ many advanced speech technology components from Bell-Labs research and from others, we can easily prototype speech user interface and multi-modal user interface applications. A script language support for speech user interface and speech dialogue control makes it more convenient to study human-machine speech dialogue interface in real world conditions. This effort is continuing in the following directions: standardize the API for spoken dialogue applications, support object-oriented technology, extend it to other popular script languages, develop a standard API/SPI for speech technology components, and add more speech technology components for natural language dialogue and understanding.

6. REFERENCES

1. Q. Zhou, C.-H. Lee, W. Chou, A. Pargellis; "Speech Technology Integration and Research Platform: A System Study"; *5th European Conference on Speech Communication and Technology, TMC.2,* Rhodes, Greece; 22-25 Sept 1997

2. Q. Zhou, A. Saad, C.-H. Lee, "An Open System Architecture for Speech Interfaced Applications on Internet and Public Telephone Network", (unpublished)

3. C. Garcia-Mateo, Q. Zhou, C. H. Lee, A. Pargellis, "An Overview of a Voice User Interface Demonstration System for Mexican Spanish", *ICSLP'98* (this conference)

4. Web site for NIS summit meeting, 12-13 March 1998: http://www.bell-labs.com/project/ConC