An Enhanced BLSTIP Dialogue Research Platform

Qiru Zhou, Antoine Saad, and Sheriff Abdou
Dialogue Systems Research Department
Bell Labs, Lucent Technologies
600 Mountain Ave.
Murray Hill, NJ 07974, USA
({qzhou, saad}@research.bell-labs.com, sabdou@coeds.eng.miami.edu)

ABSTRACT

This paper presents some recent enhancement to Bell Labs Speech Technology Integration Platform (BLSTIP), a common platform to integrate Bell Lab’s speech, telephony, Internet, and dialogue technologies for spoken and multi-modal dialogue system research and prototyping. Last year, we introduced BLSTIP to our partners as a speech technology platform for collaborative research and new application development.

BLSTIP software is packaged as a single, network downloadable installation file. It supports a variety of speech applications such as natural language call routing/steering for call centers, natural language information system, messaging system with voice user interface, speaker verification, speech application trial and data collection, etc. As an enhancement to BLSTIP, we also designed a VoiceXML (Voice eXtensible Markup Language) infrastructure to study emerging web hosted speech applications such as voice portal, multimodal internet access, and wireless internet access.

1. INTRODUCTION

In the past several years, we have been developing Bell Labs Speech Technology Integration Platform (BLSTIP) for internal spoken dialogue research and technology demonstrations [1]. Recently, BLSTIP has been improved to actively support both internal and external partners and customers for collaborative research and technology evaluation. In this paper, we report some new features added to BLSTIP. It leads to a research and prototyping speech platform software package for easy application development.

On the tool side, we have two added features. First, we made BLSTIP easy to access and cost effective. Now a whole platform can reside in a Microsoft Windows PC, with an add-on computer telephony integration (CTI) interface card that supports Microsoft TAPI 2.1 TSP (telephony service provider).

Second, we packaged the whole BLSTIP into a single, easy to use network downloadable installation file, which includes all the necessary software components and tools required to create speech applications on Microsoft Windows platform.

In addition to components and functions existing in the original BLSTIP, we added the following set of new functions and enhancements:

1. Speech recognition: An improved, real-time speech recognition engine supports both n-gram statistical language model and finite state grammars. It shares the same high accuracy context dependent acoustic model for both recognition modes.

2. New telephony call control functions: We added DTMF detection and collection functions. Combine with existing call control functions, BLSTIP is sufficient to simulate an incoming call center service, with multiple telephone channel support.

3. Data collection: Enhanced speech data recording, including speech end pointing, and dialogue session information logging capability. A Wizard-Of-OZ (WOZ) system for speech application prototyping is built on top of BLSTIP.

4. Integration of speech components with standard interface: Now BLSTIP supports both SAPI 4.0 speech recognition (SR) and text-to-speech (TTS) engine integration.

5. Internet voice processing: In this release, we added Internet URL audio streaming that allows applications to play audio files that reside on internet to CTI interface directly. Under Microsoft TAPI 3.0 support, BLSTIP may use IP H.323 audio devices in the future.

6. VoiceXML[2] support: We developed a way to integrate BLSTIP with Lucent Teleportal [3, 4] to support the emerging VoiceXML. Which enables web hosting of voice applications. This provides an inexpensive platform to study VoiceXML based speech dialogue systems on voice network.

7. Document and samples: We have developed a BLSTIP user guide, function manual pages, and a set of code samples to help customers to learn and use BLSTIP.

In the next section, we present the enhanced BLSTIP architecture and API. In Section 3, the BLSTIP-VoiceXML architecture is introduced. We also provide samples to illustrate how to write applications on BLSTIP.

2. BLSTIP: ARCHITECTURE AND API

To create a generic dialogue application framework for web hosting speech applications and server based dialogue managers, it is a good practice to separate service logic (application definition) from user interface (voice end-pointing and speech processing devices, such as ASR and TTS.). To support this architecture, we enhanced the original BLSTIP architecture in the following ways as shown in Figure 1:

1. All the speech components and voice interface devices are considered speech and audio devices (SAD).
BLSTIP API C functions are grouped to the following categories:

1. Initialization:
   ```
   int CreateVoiceApp() -- create an voice application instance. At present, only one voice application can be instantiated per process.
   int DeleteVoiceApp() -- destroy voice application instance and release resource to operating system.
   ```

2. Session channel assignment and proxy registration:
   ```
   int ConnectToServer() -- connect the instantiated voice application to designated RM. Only one RM can be connected from a voice application.
   int AssignAppToTelephonyServer() -- assign voice application to specific CTI server and phone line.
   ```

3. SAD configuration requests:
   ```
   int SetServerType() -- set major and minor SAD device numbers for voice application requested SADs. The currently supported major SAD device numbers are: TASK_ASR, TASK_AUDIO, TASK_TELE. The minor device numbers are customized for specific BLSTIP system. For example, a system may specify (TASK_ASR, 0) as context-free grammar recognizer, and (TASK_ASR, 2) as a natural language speech recognizer. One voice application may use multiple ASR and TTS devices.
   ```

   ```
   int AddRequest() -- send SAD configuration request string to a specific SAD (addressed by device number). To minimize speech recognizer run time grammar loading delay, a voice application may pre load multiple grammar at initialization by using AddRequest().
   ```

   ```
   int SetSpeechBargeInMode() -- enable/disable speech barge-in during speech recognition.
   ```

   ```
   int SetTouchToneBargeInMode() -- enable/disable barge-in during touch-tone detection.
   ```

   The notion of barge-in is that if a voice application user is starting to input (speak, or push phone keys) before a system prompt finishes playing, the system detects such input and stops the play of the system prompt. At the same time, the system continues to listen to user input, until it recognizes the user input based on a speech or a DTMF grammar, or times out if a pre-defined input time limit is reached. This feature allows a voice application to be more responsive to user interaction with prompt service. Barge-in feature applies to speech dialogue where user is expected to input after listening to part of a system prompt or response, such as menu item selection, form filling, etc.

   4. Internally, BLSTIP API is designed as a single C++ class CClientAPI. To make it convenient to use it with an application language, we derived a simple set of C functions from CClientAPI public methods. It provides a rich set of speech processing services (SR, TTS, audio play, and speech recording) and basic telephony controls. At present, we have wrapped BLSTIP API with Perl (as a Perl module) and Java (in our DARPA Communicator project architecture, which uses BLSTIP as our underlining CTI and speech platform.).

5. To support sophisticated dialogue application logic, we developed a new application-building framework: We use a small application session agent. It answers incoming call from a user, and initiates the application service logic by sending the initiation message to application dialogue manager (DM), then waits for further instructions from the DM. The DM takes control from this point, until the session finishes, or the user hangs up. The DM maintains the service logic (the app state machine) and request for speech/telephony interface service by sending service requests to its session agent. The agent calls BLSTIP API functions to execute the request and sends the received messages to the DM. Therefore a complicated DM can be pre-initialized as a DM server to reduce system response time.

6. All inter-component communications in BLSTIP are based on a simple, text based socket communication protocol.
In BLSTIP, when the barge-in mode is enabled, it will delay prompt playing (recorded speech, or TTS) till speech recognition is started. Therefore, the input detectors (speech, or touch tone) start listening before the prompts start playing. To support this feature, good echo cancellation is necessary to prevent falsely triggered recognition caused by the leaked echo signal. Furthermore, BLSTIP supports “smart barge-in” which uses speech recognition and result confidence measures to further detect if the user input is meaningful. This feature greatly reduces recognition false trigger rate.

4. Speech processing functions:

- **int PlayText()** – synthesize a text string and play it to the voice device that the voice application used.
- **int PlayTextFile()** – play text in file.
- **int PlayRecordedSpeechFile()** – play recorded sound file.
- **int RecognizeSpeechFile()** – do speech recognition on a given speech sample file.
- **int RecognizeSpeechTele()** – do speech recognition on a live telephone line input.

5. Telephony functions:

- **int AnswerCall()** – answer the phone call to voice application designated phone line if it is ringing.
- **int HangUpCall()** – hang up the line assigned to voice application.
- **int BlindTransferCall()** – do blind transfer to on calling line to the phone number specified by function parameter.

6. Call back control:

- **int SetAppCallBackProc()** -- when BLSTIP runs in an asynchronous mode, the speech and telephone processing functions will run in an asynchronous call back mode. i.e., the function calls return immediately and a call back function registered with SetAppCallBackProc() will notify the voice application the events generated by asynchronous processing. The voice application needs to parse received events, and take proper actions upon receiving of certain events.

7. Data collection functions:

- **Data collection on the speech recognition input.**

  In this case, the data collection saves the input to the recognizer engine. A voice application uses **SetSaveUtterance** requests to ask the speech recognizer to do data collection. This method helps system testing and recognition error tracking.

8. Error tracking:

- **tagERRORCODE GetLastErrorCode()** – all BLSTIP API functions return a Boolean type value – TRUE (1) for success, and FALSE (0) if failed. This function can be used right after function returns fail to get its error code.

The following code demonstrates how to use BLSTIP API to build a simple application to use simultaneous speech and touch tone input. If a valid 4-digit phone number is recognized, it will transfer the call-in line to the number (we skip initialization and configuration code for simplicity):

```c
// Enable both SR and DTMF barge-in.
SetSpeechBargeInMode( 1 );
SetTouchToneBargeInMode( 1 );
PlayTextFmt("Enter %d digit phone number to transfer to", NumDigits);
PlayText("Or tell me how can I help you for our bank service.");
bRet = GatherDigits( 4, 5000, 5000, "+#" );
bRet = RecognizeSpeechTele(""),
// Disable barge in to process results.
SetSpeechBargeInMode( 0 );
SetTouchToneBargeInMode( 0 );
// Collect SR and DTMF results.
char *SRResponse = GetRecognizedString();
char *DTMFResponse = GetGatheredDigits();
...
if ( strlen( SRResponse ) )
PlayTextFmt( "You said %s.", SRResponse );
// do blind transfer
if ( strlen( DTMFResponse ) == 4 ) {
// if digit string is in DTMF grammar, we do the transfer.
PlayTextFmt("Transferring call to %s", DTMFResponse);
bRet = BlindTransferCall( DTMFResponse );
}
```

3. BLSTIP--VoiceXML

VoiceXML is an emerging standard to make internet content and information accessible via the phone, with wide support of leading Internet industry leaders [2]. VoiceXML provides a standard, systematic method to make speech-enabled applications on the internet. It also provides a generic framework to separate application service logic and implementation (user interface), and makes it possible to integrate web and phone services in one web hosted application. To study this new standard, it is important to have a research and prototype VoiceXML platform with near real-world behavior to study this new technology and explore VoiceXML based speech dialogue system.
We applied a simple approach to integrate a VoiceXML interpreter to BLSTIP: tokenize VoiceXML action requests to BLSTIP requests, and translate back BLSTIP events to VoiceXML interpreter events. To add VoiceXML functions to BLSTIP, we reused part of the Lucent VoiceXML platform – Teleportal [4]. Instead of implementing all JSAPI, JTAPI, and Java audio interface to provide platform specific services, we redirected Teleportal’s platform specific interface to BLSTIP simple API, which provides sufficient speech, telephony, and audio recording/play services. In this design, we only allow the dialogue manager to issue platform specific requests to BLSTIP. The document interpreter keeps application logic. It sends a request to DM when it needs a platform specific speech, telephony, or audio service. This architecture does not only make it simple to integrate VoiceXML platform with BLSTIP, but also makes it easy to integrate with other platforms. Figure 2 illustrates the BLSTIP-VoiceXML architecture.

The following is an BLSTIP-VoiceXML example.

The VoiceXML document of a simple dialogue:

```xml
<?xml version="1.0"?>
<xml>
  <form>
    <field name="drink">
      <prompt>Would you like coffee, tea, milk, or nothing?</prompt>
      <grammar src="ex05b.gram"/>
    </field>
    <block>
      <goto next="http://localhost:8080/servlet/ex05b" submit="drink" method="get"/>
    </block>
  </form>
</xml>
```

The corresponding BLSTIP message and response sequence:

**Receive:** AnswerCall 4 // answer this call after 4 rings
**Response:** EVENT_TELE_CALL_CONNECTED // the call is answered

**Receive:** AddRequest ASK_ASR "SetGrammarPath c:\Grammar\consolidatedGrammar" // detected a new grammar so asked the server to reload its grammar from this path before it starts recognition.

**Response:** // no need for response in case of on errors in grammar loading

**Receive:** RecognizeSpeechTele " SetTimeoutParams 200 400 100 SetBargein 1" // starts recognition with this timeout parameters and enable Bargein as it’s the default for prompts in VXML

**Response:** // no response in case of successfully started recognition engine.

**Receive:** PlayText “Would you like coffee, tea, milk, or nothing?” // play this prompt

**Response:** EVENT_TELE_PLAY_DONE // playing finished successfully

**Response:** EVENT_RESOURCE_SERVER_RESULT "I prefer to drink tea" // the result of speech recognition

When DM receives the above recognition result it forwards it to VoiceXML interpreter which parses it according to the grammar and forward the pair (Drink, tea) to the FormSubmitter to submit it to the servlet at [http://localhost:8080/servlet/ex05b](http://localhost:8080/servlet/ex05b). Which uses it to generate a feedback VoiceXML document.

### 4. CONCLUSIONS

With additional BLSTIP feature enhancement and software engineering effort, we made BLSTIP a powerful, open, easy to access platform to support both internal and external partners for collaborative research and speech technology promotion. The recent BLSTIP-VoiceXML integration enables the internet speech technology integration research at a systematic level.

### 5. ACKNOWLEDGMENTS

The authors thank Rich Mendez and Tulan Hu of Lucent Software Technology Group for providing help and software on Lucent Teleportal platform. The authors would like to thank Frank Scallon of BT for our fruitful discussion and experimentation with BLSTIP.

### 6. REFERENCES


