Integration of an HTK Speech Recognizer with OpenVXI and an Asterisk based VoiceXML Gateway

Lerato Lerato, Molise Mokhomo, Thato Pebane and Lerato Mohapi
Department of Mathematics and Computer Science
National University of Lesotho, P. O. Roma 180, Lesotho
Tel: +266 22213436, Fax: +266 2234 0000
email: l.lerato@nul.ls, {leratojeffrey.mohapi, mmokhomo}@gmail.com

Abstract—VoiceXML is one of the most popular markup languages used in commercial IVR systems. It is still not possible to find the fully functional open source VoiceXML gateway that helps researchers enhance the VoiceXML IVR technology. It is essential that a free and open source VoiceXML IVR platform that meets all the W3C standards is designed and made available so that the quality of IVR systems can be consistently improved at the justifiable cost. This paper introduces the VoiceXML gateway that comprises the integration of OpenVXI and an Asterisk telephony platform; focusing on the addi­tion of the HTK-based speech recognition capability of the system. Finally, the paper reports the successful implementation of a recognition interface that conforms to the SRGS specification.

Index Terms: Speech Recognition (HTK), OpenVXI, VoiceXML Gateway, SRGS.

I. INTRODUCTION

VoiceXML (Voice Extensible Markup Language) is the W3C standard XML format for specifying interactive voice dialogues between a human and a computer. VoiceXML also requires a voice browser (VoiceXML interpreter) to interpret VoiceXML documents [1]. The documents are written using the VoiceXML. The voice browser must have an interface to PSTN or PBX, speech recognition engine and speech synthesis engine, in order to become an IVR system [1]. The platform that incorporates the voice browser, telephony integration and PBX functionalities for VoiceXML applications is termed the VoiceXML gateway. This paper presents the VoiceXML gateway that uses the OpenVXI voice browser [2] and Asterisk PBX [3]. This gateway uses the same entities as those used in Voiceglue VoiceXML gateway [4] but will include an additional Speech Recognition component as one of the possible IVR inputs that appear in the VoiceXML 2.0 standard [1].

The work reported in this paper is not the first to deal with the design and implementation of the VoiceXML gateway that is intended for open source consumption. Other researchers with similar work found so far have designed other open source VoiceXML gateways listed in [5]. In addition to those mentioned in [5], one should note another VoiceXML gateway developed at Rhodes University (www.rhodes.ac.za) by [6].

OpenVXI has already got API’s for speech recognition, telephony and speech synthesis engine [2]. The speech recognition engine that will be used to enhance OpenVXI is the Hidden Markov Models Toolkit (HTK) [7], which is very popular in speech recognition research and development. The HTK has flexibility of acquiring speech input from various sources depending on the configuration values. According to VoiceXML grammar specification, a speech recognizer used to integrate VoiceXML interpreters must conform to the Speech Recognition Grammar Specification (SRGS) 1.0 standard [8].

SRGS is the standard that defines the syntax for representing grammars to be used in speech recognition. The syntax of the grammar is given in two forms, XML and an Augmented Backus–Naur Form (ABNF) form. The purpose of the grammar is to specify a set of words that a speech recognition engine should expect from a human utterance. This paper’s main focus is on the integration of the HTK with the OpenVXI voice browser (VoiceXML interpreter) and making sure that it conforms to SRGS 1.0 standard [8].

The OpenVXI voice browser is described in section II. Section III gives the short explanation of Asterisk PBX and section IV briefly introduces the HTK speech engine. Section V gives the details of how OpenVXI, Asterisk and HTK were integrated in this study. Section VI reports the experiments and analysis of results. Finally conclusions are made in section VII.

II. THE OPENVXI VOICEXML INTERPRETER

OpenVXI is an open source VoiceXML interpreter that processes VoiceXML dialogue markup language according to the VoiceXML 2.0 specification. OpenVXI was originally developed by SpeechWorks International, and maintained by Scansoft (now Nuance) (www.nuance.com) before passing to Vocalocity [2][9]. OpenVXI has speech recognition, prompt and text-to-speech (TTS) capabilities, and includes telephony functions. The developers’ task is to provide its integration to telephony platforms so that the complete VoiceXML gateways can be implemented.

A. The OpenVXI Architecture

Abstract interfaces that control the integration of OpenVXI system with ASR engine, TTS engine and telephony interface are shown in figure 1 below. The integration of OpenVXI with an Asterisk telephony platform has been implemented by the authors in [10] but since similar work has been demonstrated by Voiceglue [4],
and further, the integration of these software platforms is a long detailed work, this paper will only concentrate on the speech recognition engine interface with OpenVXI; paying attention to the grammar.

![Figure 1: The OpenVXI architecture [2]](image)

**B. The OpenVXI API’s**

As shown in figure 1 above, each OpenVXI module and API does a specific task when VoiceXML page is interpreted [2].

- **Speech Recognition API**
  This API set defines the interface between the VoiceXML interpreter and the speech recognition engine. In a VoiceXML document, when voice input or touch-tone input are required, the interpreter calls the speech recognition APIs. It is responsible for evaluating user input against any active grammars, and recording utterances.

- **Prompt API**
  This API defines the interface between the VoiceXML interpreter and the speech synthesis engine. When voice output is required, interpreter calls the prompt API to synthesize the voice output. Hence the prompt API represents a one way interaction between the gateway and the user.

- **Telephony API**
  This API set defines the interface between the VoiceXML interpreter and the telephony interface. It is the API responsible for setting up and tearing down calls and also monitoring the calls to detect hang ups and providing the functionality to transfer calls.

**III. ASTERISK AS AN IVR TELEPHONY PLATFORM**

Asterisk is an open source, converged telephony platform, which is designed primarily to run on Linux. When used for Voice over IP it supports IAX, SIP, MGCP and H.323 protocols while at the same time it can be used as an interface to the PSTN [3][11]. The experiments reported in this paper were carried out using SIP (Session Initiation Protocol) signaling protocol. The major benefit of using Asterisk as the telephony interface is the fact that it supports many telecommunications technologies. This means that the VoiceXML gateway (OpenVXI and Asterisk) will be exposed to all the technologies supported by Asterisk, thereby allowing a lot of flexibility with regards to interacting with the voice applications.

Asterisk has a capability of receiving inbound calls and enabling the user to send outbound calls through a dialplan. An Asterisk dialplan is however not as popular, flexible and adaptable as a VoiceXML call flow. For this reason the enhanced Asterisk Gateway Interface (EAGI) [11] which connects external programs with Asterisk PBX has been used as the interface between OpenVXI and Asterisk. It is in this EAGI program where classes that enable communication between the interpreted VoiceXML data from OpenVXI to be transmitted to Asterisk that in turn forwards it to the user of the system under the IVR setup.

As mentioned in section II above, the details of this integration will not form part of this paper and those can be examined in the Voiceglue source code [4].

The rest of this paper assumes that the VoiceXML gateway which uses OpenVXI as the voice browser and, the Asterisk PBX as a telephony platform functions according to the VoiceXML 2.0 standard and, the missing entity is the speech recognition capability that will be discussed from section IV onwards.

**IV. THE HTK SPEECH RECOGNITION COMPONENT**

The speech recognition component is responsible for managing the associated application grammars and recognition state, and processing the spoken utterances, and attempts to recognize the spoken utterances from a set of known valid inputs. The speech recognition component used in this study is the Hidden Markov Models Toolkit (HTK) [12] and this means that the HTK is used as the speech recognizer. It is integrated with the OpenVXI interpreter using the Rec API as appears in figure 1. The HTK has several tools that are used to perform various functions. The tool that one may consider essential for the integration is the HVite tool [7] which governs the recognition and this tool can be called within, for example, a C++ program to perform recognition of a given speech waveform. This tool is incorporated in the Asterisk Gateway Interface [11] so that it enables the VoiceXML gateway to have an automated recognition.

**V. THE INTEGRATION**

The components necessary to build a HTK-based speech enabled VoiceXML gateway as implemented in this study are (1) the OpenVXI voice browser (2) Asterisk PBX platform and (3) The HTK speech recognition engine. The integration of (1) and (2) were briefly highlighted in section III above and that yields the complete VoiceXML gateway with no speech recognition capability. All that is required is to connect (1) and (3). Figure 2 below reflects a simple experimental setup that was used to test the IVR capabilities of the gateway. The softphones were used to call the Asterisk box that was connected to the application
servers that contained VoiceXML pages and databases. The
two servers shown in the middle of the diagram (figure 2)
were in one server so that OpenVXI, HTK and text-to-
speech were integrated through software only. It should be
noted that the integration classes were also in that single
server.

A. Connecting HTK and OpenVXI

Among the three OpenVXI API’s this paper focuses on
modifying only the speech recognition interface by
implementing a class, SRGS_VXI, which comprises the
methods used to parse the VoiceXML grammar into a
format understood by the HTK. The grammars may either
be from VoiceXML or from external grammar file. The
supported grammars are those that are written in JGSF,
XML or ABNF format. Furthermore each word that
appears in the grammar must be one of the words used to
train the speech recognizer. The following notes provide a
high abstraction level on how recognition is performed:

- A VoiceXML application with a defined grammar (in
  SRGS format or JSGF similar to that of HTK) is
  requested by a caller. The grammar is then parsed to
  the format compliant with the HTK and the words
  uttered by caller are recorded and stored in an audio
  file (.wav format).
- The audio file together with the processed grammar is
  sent to the speech recognition engine, HTK. This audio
  file and activated grammar are then processed by an
  independent speech recognition tool, HVite tool [10]
  in this case, to recognize the words uttered by the caller in
  a textual format.
- The recognized text is used by VoiceXML application as
  an input.

B. Design of the SRGS_VXI class

The purpose of this class is to define methods that are
used to parse a grammar written in SRGS specification into
a lattice format recognized by the HTK. The HTK grammar
is formatted according to specifications in the HTK book
[10]. The methods are as follows:

```c
string getScope(string rule);
/*
input: string rule: is the rule tag,<rule ... >
output: it returns the value of the scope attribute: either private or
        public
*/

string getRuleId(string rule);
/*
input: string rule: is the rule tag,<rule ... >
output: it returns the value of the id attribute, that identifies the rule.
*/

string getSpecial(string rule);
/*
input: string rule: is the rule tag,<rule ... >
output: it returns the value of the special attribute: VOID, NULL,
        GARBAGE
*/

string getOutput(char * outputFileName);
/*
input: name of the file containing the recognized words.
output: returns a sentence of recognized words
*/

void recognizer(char * command, char * grammar, char* waveFile);
/*
input: the command to be executed, a grammar file in lattice format
*/
```
and a recorded wavefile.

output:
  a text file containing the recognized words.
*/

void record(char *fileName, int timeOut, char* interrupt);
/*
input:
  name of the file to be recorded, timeOut is the maximum time
  to be taken for recording, interrupt are dtmf key to terminate
  recording.
output:
  a recorded audio file in .wav format
*/

void grammarParse(char* grammar, char* grammarFile);
/*
input:
  a string containing grammar in SRGS format and a name of the
  file to store the grammar in lattice format.
output:
  a grammar in lattice format stored on text file.
*/

The SRGS_VXI class contains functions that improve the HTK functionality so that it conforms to the SRGS specification. The class is implemented in the VXIrec (the Rec API shown in figure 1) API which is the existing OpenVXI API that provides the connectivity to speech recognition engine, in this case the HTK. When compiled, OpenVXI generates the executable VXIClientAGI which is then installed in /var/lib/asterisk/agi-bin directory. At this stage the interest is not so much on the accuracy of the speech recognizer but on its correct functionality that will enable the process of integration with OpenVXI. This integration was implemented and tested as described in section B below.

### B. Testing HTK for SRGS

OpenVXI was installed and the major goal was to test whether HTK conforms to SRGS and live recognition. The test was performed by creating a valid VoiceXML page, test.vxml, and placed in /var/www/ directory (Apache document root directory). The code segment of the file is shown below:

#### Listing 1.

```xml
<?xml version = "1.0" encoding="UTF-8"?>
<xml version = "2.0" xmlns="http://www.w3.org/2001/vxml">
<form id="english">
  <property name="inputnodes" value="voice" />
  <field name="digit" >
    <prompt>
      Please say any digit .
    </prompt>
    <grammar type="application/x-jsgf">
      ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN
    </grammar>
    <filled>
      <prompt>You said <var expr="digit" /> </prompt>
    </filled>
    <nomatch>
      You did not say any digit. <reprompt/>
    </nomatch>
  </field>
  <noinput>
    Sorry, I did not hear that. <reprompt/>
  </noinput>
</form>
```

The first line (SENT) of the “HTK Results Analysis” shown above gives the sentence recognition rate (%Correct=94.64), the second one (WORD) gives the word recognition rate (%Corr=94.87.00). The SENT parameter, H=106 gives the number of test data correctly recognized, S=6, the number of substitution errors and N=112, the total number of test sentences. These results imply that of the 112 sentences making the testing corpus only 106 were correctly recognized which is equivalent to 94.64% recognition rate. 6 sentences were substituted by other sentences. The statistics given on the second line (WORD) only make sense with more sophisticated types of recognition systems (e.g. connected words recognition tasks). Nevertheless, there were 12 deletion errors (D), 20 substitution errors (S) and 13 insertion errors (I). N=490 gives the total number of words making the test data. The recognition is this case is 96.00%. These results indicate that the HTK speech recognizer is functional although not hundred percent accurate. At this stage the interest is not so much on the accuracy of the speech recognizer but on its correct functionality that will enable the process of integration with OpenVXI. This integration was implemented and tested as described in section B below.
From the code segment shown in listing 1 above, the property, inputmodes, is assigned a value, voice. Hence this page will be interpreted by VoiceXML interpreter which will in turn wait for the speech input. Thus to get input from the caller, the interpreter will load speech recognition engine HTK to convert the spoken input into text.

Experiment 1: Testing Recognition.

Case 1: correct input
computer # Please say any digit.
Caller    # ONE
computer # you said ONE
exit       #

Case 2: correct input
Computer # Please say any digit
Caller    # THREE
Computer # you said THREE
Exit       #

Case 3: incorrect input
Computer # Please say any digit
Caller    # EIGHT
Computer # Sorry, I did not hear that
Computer # Please say any digit.
Caller    # NINE
Computer # you said FIVE
Exit       #

Case 4: no input
If the caller says nothing, the noinput tag is executed and the prompt “You did not say any digit” is played. Also the first prompt “Please say any digit” will be repeated.
As indicated in case 3 above, the system recognizes a different word from the one uttered by the caller. This can be observed as one of the weaknesses of the speech recognizer where it fails to differentiate words with similar pronunciations.

Experiment 2: Testing Recognition -Advanced

Case 1: Testing Rule Definitions (scope attribute and <ruleref> tag)
Computer # please say one or two numbers.
Caller      # THREE
Computer # you said THREE
Exit         #

Computer # please say one or two numbers.
Caller      # FOUR ONE
Computer # you said FOUR ONE
Exit         #

Computer # please say one or two numbers.
Caller      # “a caller does not say any digit”
Computer # you said
Exit         #

Case 2: testing rule references (special rule: VOID, NULL and GARBAGE).

Listing 2:
<!-- some code missing -->
<field id="digits">
  <prompt> Please say a digit. </prompt>
  <grammar mode="voice" version="1.0" root="r5">
    <rule id="one_digit" scope="public">
      <ruleref special="VOID" />
      <one-of>
        <item>ONE</item>
        <item>TWO</item>
      </one-of>
    </rule>
  </grammar>
</field>

Case 2.1: special="VOID"
Computer # Please say a digit.
Caller    # TWO
Exit        #

Case 2.2: special="NULL"
Computer # please say a digit.
Caller    # "a caller says nothing"
Exit        # you said TWO

Case 2.3: special="GARBAGE"
Computer # Please say a digit.
Caller    # ONE
Exit        # you said ONE

C. Analysis of Results

From the tests performed in section B above, not all the elements of SRGS were tested. This is mainly because OpenVXI has already taken care of them and therefore there was need for further testing. Moreover, other VoiceXML elements that are interpreted by OpenVXI have already been tested by the authors in the work which does not form part of the reported experiments but can be provided [10] if requested.

Testing the VoiceXML IVR capabilities in the manner in which was done in the preceding section can quantifiably be reported according to the number of VoiceXML 2.0 elements that can be successfully interpreted on the voice enabled IVR system. The authors however found that only 84% of the VoiceXML elements worked under the speechified OpenVXI and the Asterisk based VoiceXML gateway [10].

The OpenVXI API, VXIrec, which is for speech recognition integration, was edited and this was done by implementing a class, SRGS_VXI, which incorporated the methods that convert VoiceXML grammar into a format that meets the HTK requirements.

As stated earlier, not all SRGS elements and attributes were tested. The complete list of the SRGS elements can be found in [8]. Table 1 below displays the total number of SRGS elements; those that were tested and those that were not.
Table 1: The SRGS elements that were recognized

<table>
<thead>
<tr>
<th>SRGS elements</th>
<th>Number of elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted by OpenVXI</td>
<td>10</td>
</tr>
<tr>
<td>Elements tested for Speech Recognizer (HTK)</td>
<td>7</td>
</tr>
<tr>
<td>Elements not tested for HTK</td>
<td>3</td>
</tr>
</tbody>
</table>

VII. CONCLUSIONS

OpenVXI-3.4 has been integrated with the HTK speech recognition engine. The interpreter already interprets 100% of SRGS elements and 98% of total elements of VoiceXML 2.1 applications. However, the interpreter has 0% connection with speech recognition engine. In this paper we modified one of the OpenVXI API, VXIRec, in order to interface it with the HTK speech recognizer. This was achieved by implementing the SRGS_VXI class. In this project we integrated OpenVXI that already had Asterisk as its telephony platform with the HTK speech recognition engine and constructed the voice-enabled VoiceXML gateway. Now the gateway can acquire its input through the DTMF or voice using the grammars that are specified in the SRGS specifications.

The focus of the paper was largely on the integration itself and less on the performance. The performance aspect of the process will be the next task of the authors and this platform will be tested in parallel with the Voiceglue platform will be tested in parallel with the Voiceglue of the process will be the next task of the authors and this itself and less on the performance. The performance aspect of the process will be the next task of the authors and this platform will be tested in parallel with the Voiceglue platform will be tested in parallel with the Voiceglue.

REFERENCES

[7] Steve Young, et al; The HTK Book (For HTK 3.4); Cambridge University Engineering Department; March 2009.

Lerato Lerato obtained a BSc. degree in Electrical Engineering in 2001 and MSc.Eng (Electrical) in 2004 at the University of Cape Town in South Africa. In 2004 he was employed as a Speech Scientist at Intelleca Voice & Mobile (Pty) Ltd. in Johannesburg. He is currently teaching in the Department of Mathematics and Computer Science at the National University of Lesotho. His current research involves speech technology and IVR platforms.

Molise Mokhomo is the young gentleman who has just completed his Bachelor of Engineering in Computer Systems and Networks Degree at the National University of Lesotho and will be graduating in September 2011. He is currently working at iThemba LABS for his final year internship. He is interested in speech and image technologies and IVR systems.

Thato Pebane is the young gentleman who has also just completed the Bachelor of Engineering in Computer Systems and Networks Degree at the National University of Lesotho and will be graduating in September 2011. His is interested in IVR systems.

Lerato Jeffrey Mohapi obtained his Bachelors Degree in Engineering in Computer Systems and Networks at the National University of Lesotho in 2010. He then worked in iThemba LABS in Cape Town for his internship. He is currently enrolled in the Department of Electrical Engineering at the University of Cape Town. It is Lerato who developed the ‘connector module’ between OpenVXI and the Asterisk platform for his undergraduate project.