

Open Source VoiceXML Interpreter over Asterisk for Use in IVR Applications

Lerato Lerato, Maletšabisa Molapo and Lehlohonolo Khoase
Dept. of Maths and Computer Science, National University of Lesotho
Roma 180, Lesotho, Southern Africa
l.lerato@nul.ls, maletsabisam@gmail.com, lbkhoase@gmail.com

Abstract- Much research into building the voice extensible markup language (VoiceXML) interpreters has taken place under commercial setup. This limits researchers who investigate VoiceXML-based IVR applications. This paper presents the upgrading of the open source VoiceXML interpreter, Voxy, which operates over Asterisk as its telephony platform. This software will later be released for open source consumption. So far the open source VoiceXML interpreters include: JVoiceXML, OpenVXI and Voxy. Voxy is the only one that works with Asterisk and therefore was regarded as a starting point. The result found is that the enhanced Voxy has now improved from 6.33% to 96.20% compliancy with the W3C VoiceXML 2.1 standard. The interpreter only uses DTMF for interaction with users. The utilisation of text-to-speech and automatic speech recognition related components will be dealt with as part of the future work.

Keywords: Asterisk, IVR, VoiceXML Interpreter (Voxy).

I. INTRODUCTION

VoiceXML is a W3C open standard [1] that is largely used for creating interactive voice response (IVR) systems where a user interacts with a system via voice or DTMF. In the voice mode, the system uses speech recognition to acquire audio input from the user and utilizes speech synthesis (text-to-speech) or pre-recorded audio to respond to the user [2]. VoiceXML is analogous to HTML except that it is the markup language tailored for voice browsing as opposed to the HTML which is created for web browsing. VoiceXML is used in contact centres and call centres. Just like HTML that requires a web browser to interpret it, VoiceXML requires the VoiceXML browser (interpreter) to enable voice browsing. There are many commercial VoiceXML interpreters as detailed in [3]. This limits research in the study of IVR systems such as load tests and platform efficiency. There's still a need to develop more open source web browsers that run on open source telephony platforms. The authors have found out that there are 5 open source VoiceXML interpreters namely:

- OpenVXI 3.4 which has been released by Vocalocity in 2005 [4].
- JVoiceXML that is still under development and does not run on Asterisk in the non-commercial environment.[5]
- VXMLSurf that was developed for the HearSay project [6] and it has not seen much popularity amongst the developers worldwide.
- Voxy, which is still at its rudimentary stages of its development [7].
- Tellme Studio [8] also is available for researchers who want to evaluate their VoiceXML pages but it is

remotely hosted and therefore a researcher has no access to source code.

Voxy was chosen as an interpreter of choice in this study. All VoiceXML browsers require a telephony platform in order to enable IVR user to interact telephonically with the IVR system using PSTN, GSM and even IP phones. The main motivation for choosing Voxy is that, it already runs on Asterisk PBX platform through the Asterisk Gateway Interface (AGI) module [9]. Asterisk is arguably the most popular open source telephony platform in the world which is available for developers to utilise.

The current version of Voxy does not fully meet the W3C VoiceXML 2.1 standard or specification. Most of the work presented in this paper is the enhancing of Voxy to become a VoiceXML 2.1 compliant voice browser using Asterisk as its telephony platform. The latest version of Voxy only interprets 15 of the 237 VoiceXML elements of the VoiceXML 2.0 recommendation [1]. The authors' challenge was to enhance the interpretation of VoiceXML elements to 237 using the Voxy software. An IVR application was also developed as a test that ensures the robustness of the Voxy interpreter. It is worth noting that there is a commercial VoiceXML interpreter, VXI* [10], that is fully compliant with VoiceXML 2.1 standard which runs on the Asterisk platform. The authors are therefore implementing more or less the same system that will be made available to research community at no cost.

The generic IVR platform details are briefly described in section II. Section III is the description of how Voxy works and how it is implemented in this study and section IV is an account of the experiments carried out. Section V tables the results and discussions. Finally conclusions and indications of future work are presented in section VI.

II. VOICEXML BASED IVR SYSTEMS

Advanced IVR systems are based on VoiceXML. VoiceXML is designed to facilitate the creation of IVR services; it provides sufficient functionality to completely replace existing IVR systems, and has added advantages in the area of voice application development [14]. It enables the orientation of voice-based dialogs for telephone callers that feature the playing of speech prompts using pre-recorded and text-to-speech information, accepting spoken commands (via speech recognition) and DTMF inputs, and the recording of caller audio information [15]. VoiceXML allows voice browsing applications to be developed and deployed in a way similar to how web browsing applications are enabled by HTML. Developing IVR applications in VoiceXML has the following advantages:

- VoiceXML provides an intrinsic ability to access information stored on or accessed through a corporate web server.

- Databases that are already developed and in use are directly usable in a VoiceXML script.
- VoiceXML provides a high flexibility of voice menus which leads to highly functional IVR Systems.
- Easy integration with other IP services (e.g databases, web, etc.).
- VoiceXML applications have excellent portability across web server and IVR platforms. [13]

All IVR systems require a physical VoiceXML service infrastructure that includes the following major components:

- VoiceXML Gateway (the interpreter) which receives either the DTMF signal or the audio signal from the PSTN or the GSM network. In addition this component channels out the outbound calls and responses to the user using dialogic cards or network interface cards,
- Web or application server that hosts the VoiceXML documents [2], web scripts (PHP, ASP, ASP.net), databases, J2EE etc...
- Local area network (LAN) that integrates the servers and also connects to the internet.

Figure 1 below demonstrates how VoiceXML service infrastructure is assembled. The testbed used to implement figure 1 illustration is reflected in figure 2.

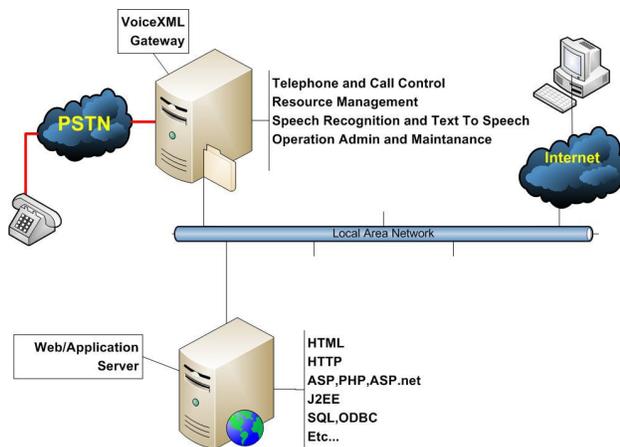


Figure 1: A typical VoiceXML service architecture [2]

In this study, Voxy serves as the VoiceXML Gateway together with Asterisk as an interface between the browser and telephone network. PHP is used as a scripting language and MySQL as a database. The proposed architecture for our implementation of the Voxy-based IVR system is shown in figure 2 below:

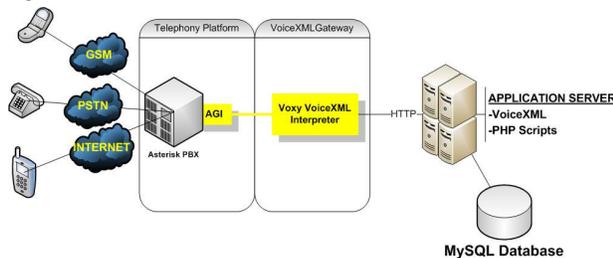


Figure 2: The implemented architecture of the IVR system

III. Voxy VoiceXML INTEPRETER

Voxy 1.4 is the latest version available online [7] and not much documentation has been prepared for developers. A very brief description of the C++ class organization of Voxy 1.4 can be found in the small manual and code comments provided along with the software.

Voxy connects with Asterisk PBX by the Asterisk Gateway Interface (AGI). Voxy is designed to create an *agi* script when it compiles, which is placed in Asterisk's *agi-bin*. In this way, the interpreter has access to Asterisk's functions by its EAGI connection to Asterisk. Beyond the connection with Asterisk, Voxy fetches VoiceXML pages from the application server by HTTP for interpretation. The interpretation of the VoiceXML pages is defined in the interpreter. Voxy has been developed in an Object-oriented manner, performing its main interpretation by the interaction among its classes.

A. Organisation of classes in Voxy

Before any upgrading, Voxy has one main C++ program which contains the main function of the interpreter. The interpreter has three categories of classes – **Browser** classes, **Tag** classes and **Tools** classes.

TABLE I
CLASS CATEGORY IN VOXY INTERPRETER

CLASS CATEGORY	FUNCTIONS	MEMBER CLASSES
Tag Classes	Perform the actual interpretation of the respective VoiceXML elements	All the VoiceXML elements as individual classes
Browser Classes	Control the call flow in the VoiceXML application being interpreted Fetch the VoiceXML pages from the application server Perform an XML parse of the VoiceXML page for interpretation in the tag classes.	VXBrowser VXHandler VXBrowserStatus
Tools Classes	Contain computational and special-capability functions that are used in the other classes of the interpreter Facilitate communication with Asterisk and connectivity to the application server logging and debugging throughout the interpretation of the VoiceXML document	Eagi Http Variable Log Exception

The relationship between the classes involved in the main interpretation process of VoiceXML XML applications is

shown in figure 3 below, showing a case of one VoiceXML element - <filled>:

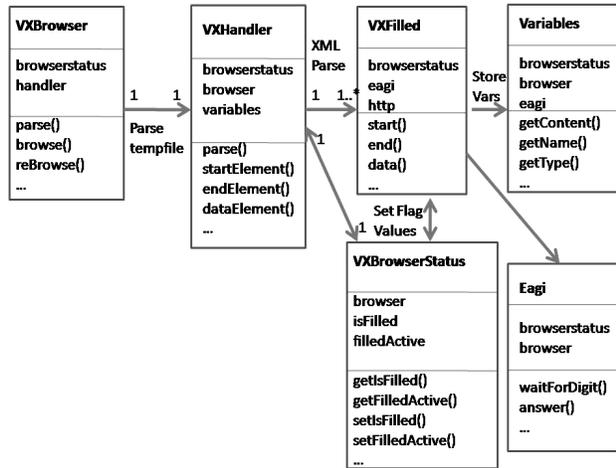


Figure 3 Class Diagram showing typical interaction of classes in Voxy

Upon understanding the functionality of the Voxy classes, it is now easier to upgrade Voxy to an interpreter that conforms to the W3C VoiceXML 2.1 recommendations.

IV. UPGRADING VOXY

This section illustrates how Voxy 1.4 VoiceXML was upgraded. The upgrade of Voxy started upon the same architecture of Voxy 1.4 described in section III. A class (a C++ class file and a class header file) was created for each of the VoiceXML elements that were added; the operation of each VoiceXML element was programmed in its own C++ file. However, for each VoiceXML element other classes in the interpreter (i.e. Browser classes and Tools classes) were also edited in order to accommodate the newly added elements. An illustration of how the VoiceXML elements were developed is presented with the <filled> VoiceXML element as an example.

A. Introducing a new VoiceXML Element to Voxy

The first step before creating a new element class is to introduce the VoiceXML element in the interpreter environment. The class *VXFilled* was created, with simple attributes and methods. All existing VoiceXML elements in Voxy have both the main file and the header file. This new class was made up of two files, *VXFilled.cc* and *VXFilled.h* in the */tags* directory since the class *VXFilled* belongs to the tag category. The browser class, *VXHandler* was then edited to include the *VXFilled* when it compiles, by adding the line `#include "VXFilled.h"`. The string `"filled"` was added to the array of VoiceXML Element *names* that *VXHandler* can parse.

B Programming the interpretation of the new VXML element according to W3C VoiceXML 2.1 specification

First, the specification of the VoiceXML element in question (<filled>) was thoroughly studied in the W3C recommendation for VoiceXML 2.1 [11]. This was done in

order to gather the exact requirements of the VoiceXML element to be developed. For the <filled> element, the specification is:

"The <filled> element specifies an action to perform when some combination of input items are filled. It may occur in two places: as a child of the <form> element, or as a child of an input item. If the <filled> element appears inside an input item, it specifies an action to perform after that input item is filled in." [11]

Attributes of the <filled> element are shown in table I below.

TABLE II
ATTRIBUTES FOR THE <FILLED> ELEMENT [15]

mode	Data Type: (anyl all)	Default: Optional (all)
	The <i>mode</i> attribute allows the developer to have control of when the <i>filled</i> element executes. If set to 'any', then the filled element will execute as soon as one valid grammar match, specified by the <i>namelist</i> , has occurred. If set to 'all', then every variable namespace defined within the <i>namelist</i> must be filled in order to execute any vxml code contained within the <i>filled</i> element.	
namelist	Data Type: NMTOKEN	Default: Optional
	The <i>namelist</i> attribute specifies the input namespace that must equate to 'true' for the <i>filled</i> element to execute. If a filled element is contained within a <i>form</i> , the <i>namelist</i> attribute will default to the names of the form's field items. If a <i>namelist</i> is specified within a <i>filled</i> element that is contained by an input item, (such as a field), then the interpreter will throw a fatal <i>error.badfetch</i> condition.	

From the specification, the VoiceXML element, <filled>, is executed only when certain conditions hold, and so are all the children of <filled> element. Therefore two global flags were introduced into the interpreter in order to control the execution of the <filled> element and its children. The two flags were Boolean variables called *filledActive* and *isFilled*, which were declared as protected data members of the interpreter's *VXBrowserStatus* class. These flags would be accessible in any class in the interpreter through a *VXBrowserStatus*-type object called *browserStatus* which is a data member of every class in the interpreter.

The Boolean flag *filledActive* is used to track the interpreter in between the beginning (<filled>) and the end (</filled>) of the <filled> element. It is initialized as false in the *VXBrowserStatus.cc* file. It is set to true when the element starts by issuing the function call *browserStatus->setFilledActive()* as the first line of the *start(...)* method of class *VXFilled*. This means that when the element <filled> is read by the interpreter, the first thing done is to set the value of *filledActive* to true. The value of *filledActive* remains as true until the point where the <filled> element is ended by "</filled>". This is done by issuing the function call *browserStatus->getFilledActive()* in the *end(...)* method of *VXFilled*. This means that in the following VoiceXML code segment, the value of *filledActive* is **true** at the areas in italics and otherwise false.

```

<!-- filledActive is FALSE by default since it is initialized as false -->
<field>
  <prompt>
    <audio src="CheckFilled.gsm" />
  </prompt>
  <filled> <!-- here filledActive is set to TRUE -->
    <prompt>
      <audio src="FilledActive.gsm" />
    </prompt>
  </filled> <!-- here filledActive is set back to FALSE since
interpreter is leaving the element <filled> -->
</field>

```

The value of *filledActive* is used to control the execution of *<filled>* and its children in the browser class *VXHandler*. In *VXHnadler.cc*, when the next line after *<filled>* is parsed, it will be executed normally only if the value of *filledActive* is false (i.e. the interpreter is not within *<filled>* tag). If the value of *filledActive* is true, a further condition has to be checked in order to determine whether the next line will be executed or not. This condition is the Boolean value of *isFilled* - the lines that follow *<filled>* will be executed only if *isFilled* is true.

The flag *isFilled* keeps a track of whether or not the input item has been filled. *isFilled* is initialized as false in the *VXBrowserStatus.cc* file. The changing of its value is governed by the following program flow of the *start(...)* method of the class *VXFilled*:

1. Store the value of *mode* attribute in string variable named *mode*
2. Store the value of *namelist* attribute in string variable named *namelist*
3. If the value of *browserStatus->getInForm()* (an interpreter-wide flag that determines whether or not the interpreter is within a form or not - true when the interpreter is within a form, false otherwise) is true: Check if *namelist* attribute value has been specified; if it has not, assign all the form's field items to an array named *NameList* archived by looping through the interpreter's global variables vector and assigning variables which are tagged with the current form id to the next position in array *NameList*.
4. If *namelist* has values specified, the string *namelist* is chunked and its values are assigned into the array *NameList*
5. If the interpreter is not only in a *<form>* but in an input item (such as *<field>*), the first element of the array *NameList* is given the name of the input item.
6. If the interpreter is in an input item with *namelist* specified, the interpreter uses its *Tools* class *Exception* to throw an *error.badfetch* since *namelist* should not be specified for an input item
7. If *mode* is not specified, the value of *mode* is set to *all*, which is the default value.
8. If *mode* is *any*, the interpreter searches through the array *NameList*, if there is a value in that array, then *isFilled* is set to true, meaning that one of the input items in the *namelist* has been *filled*, hence the *<filled>* element can continue to execute as well as its children in according to the VoiceXML application being interpreted.
9. Else if the whole array can be searched through to the end without finding even one input item which has a value, then *isFilled* is set to false because no input item has been filled hence the children of *<filled>* element will not be executed.

With this program flow, the specification requirements of the *<filled>* element are met.

V. RESULTS AND EVALUATION

After the interpretation of each VoiceXML element was programmed in the interpreter, each element was tested individually by using an appropriate VoiceXML code segment that would robustly test the element and all the developed attributes of the element. A different code segment was used for most VoiceXML elements. The code segments were saved in Apache's root directory and Asterisk's *extensions.conf* would be configured to connect this VoiceXML file when a number (123) is dialed. This was enabled inside *extensions.conf* as follows:

```

exten => 123, 1, answer ()
exten=>123,2,EAGI(voxy.agi|http://localhost/test.vxml|debug

```

The above lines mean that a call made to extension 123 will be answered and connected to the VoiceXML document named *test.vxml* which will be interpreted according to the specification of *voxy.agi* which is located in */var/lib/asterisk/agi-bin*.

In some cases when the number 123 was called, the VoiceXML application would not work as expected, meaning that the VoiceXML element being tested has not been programmed properly in the interpreter. In cases like this, C++ classes that constitute the interpreter would be revisited and corrected to meet the specification of the VoiceXML 2.1 element.

A. The element evaluation results

The following VoiceXML code segment was used to test all the specified functionality of the *<filled>* element. The other attributes were also tested but only one test is presented here. The call demonstration is a result of when the IVR application "phone" number, 123 was dialed. We will only present the testing of the *mode* attribute because of the long code displays that are still similar to what is being presented below. The following code segment was used as the *test.vxml* file.

```

<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1">
<form id="Form1">
  <field name="Field1">
    <prompt>
      <audio src="FilledTest1.gsm"/>
      This is the test prompt; you can press any key if you want
    </prompt>
    .....<!-- some lines missing -->
  <filled mode="any">
    <prompt>
      <audio src="FilledTest3.gsm"/>
      You have pressed at least one key .....
    </prompt>
  </filled>
</field>
</form>
</vxml>

```

Audio files used in this project were in the GSM 8 kHz 16-bit mono formats. The text corresponding to the *.gsm* formatted file has been typed inside the *<prompt>* element.

This is normally used for text-to-speech to read it in case where the pre-recorded audio files fail to render sound.

The conversation used for testing the attribute, **mode** when its value is “any” was as follows:

Call 01

IVR: This is the test prompt; you can press any key if you want
 TEST_CALLER: 3
 IVR: This is the second prompt, you can press any key if you want
 TEST_CALLER: << does nothing until default timeout elapses>>
 IVR: You have pressed at least one key after the above prompts that is why you are hearing this
 <<disconnect>>

Call 02

IVR: This is the test prompt; you can press any key if you want
 TEST_CALLER: << does nothing until default timeout elapses>>
 IVR: This is the second prompt, you can press any key if you want
 TEST_CALLER: <<does nothing until default timeout elapses>>
 IVR: If this is the third voice you are hearing, then you did not press any key
 <<disconnect>>

The results obtained from these test calls show that the attribute, **mode** works well with the ‘any’ value according to VoiceXML specification. In addition, the conversation for testing the ‘all’ value of attribute, **mode**, was executed in the similar way. The responses obtained from these test calls show that the attribute, **mode**, works well with the ‘all’ value as per the specification.

The **namelist** value of **mode** was also tested along with other attributes of all the VoiceXML elements that have been added to Voxy. The responses when the number 123 was called indicated that the newly improved Voxy interpreter is able to handle all of them very well.

B. The upgraded interpreter evaluation

A total of 36 VoiceXML tags were programmed above the 11 that were already developed in the original Voxy. Moreover, most of the VoiceXML elements that were in the original Voxy only had the basic functionality; so more attributes of these VoiceXML elements were programmed in this study, hence giving Voxy the capability to interpret more complex VoiceXML applications. Table III below presents the accomplished percentage of improvement in the upgrading of Voxy 1.4. The conclusion encapsulates the short analysis of the numbers that appear in table III.

TABLE III
 VOXY INTEPRETER VOICEXML STANDARD CONFORMANCE RESULTS

	No. of VoiceXML elements	Percentage
Total VoiceXML tags and attributes	237	100%
Existing tags and attributes in Voxy before upgrade	15	6.33%
New VoiceXML tags and attributes added to Voxy	213	89.9%
VoiceXML tags and attributes not implemented	25	10.6%
Total number of VoiceXML tags and attributes working in upgraded Voxy	228	96.2%

VI. CONCLUSIONS

Voxy version 1.4 was the last release of Voxy interpreter in open source¹. This version of the interpreter has 100% capability to connect to Asterisk Open Source PBX, it allows perfect connection between the VoiceXML applications and Asterisk and also provides interpretation of the VoiceXML applications. However Voxy version 1.4 had only 6.33% capability to interpret VoiceXML 2.1 applications. In this project, Voxy interpreter has been upgraded over the same architecture of version 1.4. The upgraded interpreter has maintained the 100% connectivity to Asterisk. Voxy has been upgraded from 6.33% to 96.20% compliance to the W3C VoiceXML 2.1 standard. The authors have realised that the VoiceXML 3.0 new standard draft [3] is out.

In a nutshell one can conclude that the upgraded Voxy interpreter has 100% capability to connect to Asterisk and 96.20% capability to interpret VoiceXML 2.1 applications based on the W3C recommendation for VoiceXML 2.1 as summarized in figure 3 below:

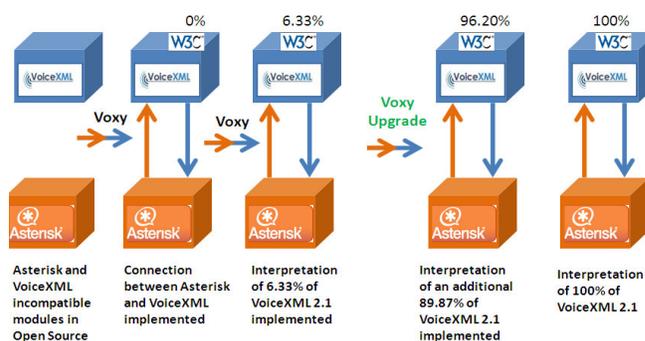


Figure 3: Summary of the progress in upgrading Voxy 1.4 to a fully W3C VoiceXML 2.1 compliant interpreter.

¹ Last checked on Wednesday 29 April 2009

It is recommended that the next version of Voxy will include text-to-speech output and Speech recognition input in order to complete this exercise. Furthermore, the VoiceXML elements which have not been programmed will be added so that the W3C VoiceXML 2.1 and 3.0 specifications are fully met.

of interest include telecommunication networks and Speech Technology.

REFERENCES

- [1] W3C VoiceXML 2.1, <http://www.w3.org/TR/voicexml21>, accessed on 1st May 2009.
- [2] Tidwell, D.J: "Considering the real benefits of Open Standards & Next Generation Intelligent Voice Response," *White Paper, Voiceeie Technologies Inc.*, pp.4–29, April 2004.
- [3] W3C, Voice Browser Activity: <http://www.w3.org/TR/voicexml21>, accessed on 1st May 2009.
- [4] Vocalocity: <http://www.vocalocity.com>, accessed on 10 April 2009.
- [5] JVoiceXM: <http://jvoicexml.sourceforge.net>, accessed 10 April 2009.
- [6] Borodin Y. : " A flexible VXML interpreter for non-visual web access". *Proceedings of the 8th International ACM SIGACCESS conference*, pp. 301- 302, 2006.
- [7] "Voxy, VoiceXML Integration For Asterisk"
http://voicexml.phpmagazine.net/voxy_voicexml_integration_for.htm
1 accessed on 1 June 2008
- [8] "Building VoiceXML applications"
<https://studio.tellme.com/vxml2/ovw/applications.html>. Date accessed 12/12/08
- [9] Asterisk PBX, "http://www.asterisk.org", accessed in June 2008
- [10] <http://vxicorp.com/speech> , accessed on 2 May 2009
- [11] <filled> element, <http://www.vxml.org/filled.htm> , Last accessed 20 December 2008
- [12] "Voice Extensible Markup Language (VoiceXML) 2.1" , W3C Recommendation 19 June 2007
- [13] Francois Mairesse : "Art on Dialogue Models and Dialogue Systems – VoiceXML",
- [14] King A., Terzoli A. and Clayton P. : "Creating a low cost VoiceXML Gateway to replace IVR systems for rapid deployment of voice applications". *Proceedings of the SATNAC conference, 2006*.
- [15] C. Bajorek, "VoiceXML - Taking IVR to the Next Level," 2000 , <http://www.cconvergence.com/article/CTM20000927S0003>, Last accessed 24/01/09

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 Intellecta 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 fields largely include speech technology and IVR platforms.

Maletšabisa Molapo completed her B.Eng Degree in Computer Systems and Networks in May 2009 at the National University of Lesotho with a First class. She will be graduating in September 2009. She has recently been appointed as a Teaching Assistant in the Department of Computer Science at the National University of Lesotho. For her final year internship, she worked for Vodacom Lesotho as an administrator of Voice Systems and IVR. She is interested in IVR Systems and Telecommunication Networks.

Lehlohonolo Khoase completed a B.Eng Degree in Computer Systems and Networks in May 2009 at the National University of Lesotho. He will be graduating in September 2009. He served his final year internship for six months at the Lesotho Brewing Company as a Systems Support Personnel. He is currently working as a Networks administrator of the Lesotho Brewing Company. His areas