AN INTELLIGENT WEB-BASED VOICE CHAT BOT

S. J. du Preez[1], *Student Member, IEEE,* M. Lall[2], S. Sinha[3], *MIEEE*, *MSAIEE*

Abstract: **This paper presents the design and development of an intelligent voice recognition chat bot. The paper presents a technology demonstrator to verify a proposed framework required to support such a bot (a web service). While a black box approach is used, by controlling the communication structure, to and from the web-service, the web-service allows all types of clients to communicate to the server from any platform. The service provided is accessible through a generated interface which allows for seamless XML processing; whereby the extensibility improves the lifespan of such a service. By introducing an artificial brain, the web-based bot generates customized user responses, aligned to the desired character. Questions asked to the bot, which is not understood is further processed using a third-party expert system (an online intelligent research assistant), and the response is archived, improving the artificial brain capabilities for future generation of responses.**

***Index Terms*: AI, XML, JAVA, AIML, ALICE.**

I. INTRODUCTION

Conventionally web-bots exist; web-bots were created as text based web-friends, an entertainer for a user [1]. Furthermore, and separately there already exists enhanced rich site summary (RSS) feeds and expert content processing systems that are accessible to web users. Text-based web-bots can be linked to function beyond an entertainer as an informer [2], if linked with, amongst others, RSS feeds and or expert systems. Such a friendly bot could, hence, also function as a trainer providing realistic and up-to-date responses.

The convenience could be improved if the system is not only text based but also voice-based & voice trained. This is the problem addressed by this paper.

A conversation is an assimilation of information where one creates differences and similarities during the duration of a conversation. Depending on the level of intelligence the experience would be enjoyable and a true emulation of a virtual entity. The gradient of intelligence is not the number of correct and incorrect statements but the ability to learn and add to its knowledge base. To create a more user accessible chat system; a simpler input method using voice is introduced; creating and catering for a more personal and convenient experience.

The process of an online chat system would follow a client server approach which acquires the signal and streams it to a server. The input voice is then processed and a response is generated. This process places a large processing requirement on the server's processor and memory resources. This limitation is even more evident when a large number of users are to be simultaneously accommodated on the system.

Voice recognition requires a two part process of capturing and analysis of an input signal [3]. While the client utilizes the operating system for an input mechanism to acquire a signal, it is for the client to interpret the signal. This process can alleviate processing from the server and allow the server to generate responses faster than when it has more voice processing requirements.

Server response generation can be broken down into two categories: data retrieval and information output. The core focus of this paper is to improve the information output by generating a response that is relevant to the request, factual and personal. This requires aspects of news and an intelligent algorithm to generate informative and user specific responses.

The paper is divided into the following sections: II. System Architecture, III. System Specifications, IV. Open Source Approach, V. System Implementation, VI. Results, and VII. Conclusion.

II. SYSTEM ARCHITECTURE

The system consists of the following three components: client, server, and content acquisition. The server is a simple object access protocol (SOAP) aware internet application (web service) based on a black box approach. A black box approach isolates the client from interacting with the inner workings of the web service; as opposed to a white box approach, where the inner workings are essential and allows the client to interact with a distributed environment. As shown in Fig. 1, all messages are formatted in an extensible markup language (XML) and encapsulated as a SOAP message pack. The packs are text based allowing for a greater diversity of clients and platforms. The client contains the voice recognition processing module which allows the client to only send and receive plain text.

[1] S.J. du Preez is a BTech student at the Dept.: Enterprise Application Development, Tshwane University of Technology (TUT), Staatsartillerie Road, Pretoria West, 0001, South Africa (corresponding author phone: +27-83-289-5142; e-mail: sjdupreez@ieee.org).
[2] M. Lall is a Senior Lecturer at the Dept.: Enterprise Application Development, TUT, Staatsartillerie Road, Pretoria West, South Africa.
[3] S. Sinha is a Senior Lecturer at the Dept.: Electrical, Electronic & Computer Engineering, Carl & Emily Fuchs Institute for Microelectronics (CEFIM), University of Pretoria, Corner of University Road and Lynnwood Road, Pretoria, 0002, South Africa.

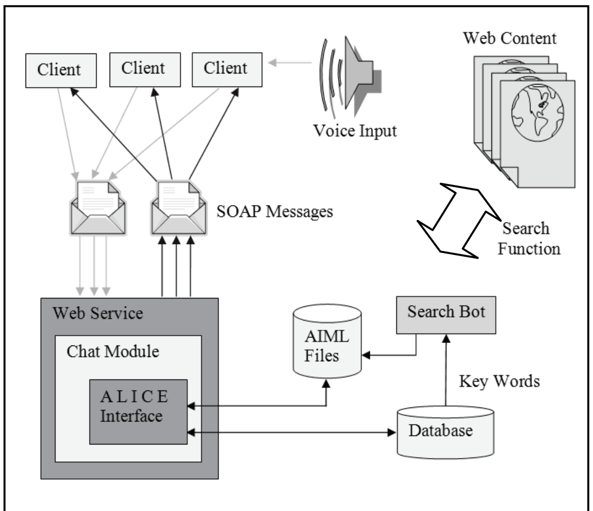


Fig. 1 – System architecture

The web service processes all received queries using the response generation module (based on the Artificial Linguistic Internet Computing Entity (ALICE) [10] system), which makes use of a data repository. The data repository is updated by the content retrieval module to increase the intelligence autonomously (based on an Artificial Intelligence Markup Language (AIML)). With this approach, an external administrator is only required to verify the quality of the self training function. For a future query, the content is re-processed and incremental updates are made.

III. SYSTEM SPECIFICATIONS

The system presented in this paper meets the following requirements.

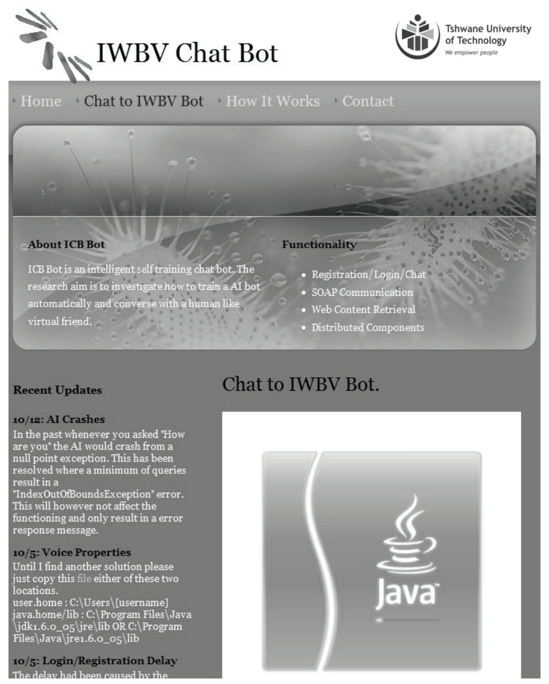
- The client application is easily accessible through the client browser.
- All communications to and from the server is text and XML formatted.
- XML messages conform to a schema which describes the format.
- Communications with the server is black box oriented and parses incoming XML messages seamlessly.
- The user is allowed to register and login to the system allowing for authenticated, personalized and controlled communication with the server.
- The client applet provides the user two options: text input or voice input.
- The self-training AI module prevents a service bottleneck, and therefore prevents modules from competing for resources.

IV. OPEN SOURCE APPROACH

There are a number of available libraries and open source technologies to implement the specifications presented in this paper. This approach allows new implementations of a paradigm of using existing libraries and technologies to create custom implementations using open source libraries.

V. SYSTEM IMPLEMENTATION

The main language used to develop the demonstrator of this paper is JAVA [1]; and the applet is embedded using HTML. This approach of hiding the JAVA component from the end user creates an illusion of simplicity as shown in Fig. 2. The website contains the embedded applet, and is hosted by Apache web server[2]. The database and website is managed using open source database management software, MySQL [3].



About ICB Bot

ICB Bot is an intelligent self training chat bot. The research aim is to investigate how to train a AI bot automatically and converse with a human like virtual friend.

Functionality

- Registration/Login/Chat
- SOAP Communication
- Web Content Retrieval
- Distributed Components

Recent Updates

10/12: AI Crashes
In the past whenever you asked "How are you" the AI would crash from a null point exception. This has been resolved where a minimum of queries result in a "IndexOutOfBoundsException" error. This will however not affect the functioning and only result in a error response message.

10/5: Voice Properties
Until I find another solution please just copy this file either of these two locations.
user.home : C:\Users\[username]
java.home/lib : C:\Program Files\Java\jdk1.6.0_05\jre\lib OR C:\Program Files\Java\jre1.6.0_05\lib

10/5: Login/Registration Delay
The delay had been caused by the

Chat to IWBV Bot.

Fig. 2 – Website with integrated JAVA based applet

The applet requires a number of libraries enabling for processing voice inputs. The signed libraries were kept on the hosting website, the same location as the applet. Before the applet launches inside a browser, a loading sequence streamlines the launch of the libraries. The code segment (edited to save space) below illustrates such a loading sequence.

```
<applet code="client.ICB" archive="Client.jar" name="IWBV">
 <param name="cache_archive"
value="Client.jar,lib/cmu_time_awb.jar,lib/cmu_us_kal.jar,lib/cmu
dict04.jar">
<PARAM NAME="cache_archive_ex"
VALUE="Client.jar,lib/cmu_time_awb.jar;preload,lib/cmu_us_kal
.jar;preload,lib/cmudict04.jar;preload">
<property name="freetts.voices"
value="com.sun.speech.freetts.en.us.cmu_time_awb.AlanVoiceDir
ectory"/>
<PARAM name="freetts.voices"
value="com.sun.speech.freetts.en.us.cmu_time_awb.AlanVoiceDir
ectory"/>
</applet>
```

The pre-loading of libraries allows for streamlined operation (i.e. the libraries are not called in an ad hoc basis, halting the sequence of activities).

The user is prompted to accept a signature, upon such an acceptance; the applet can securely communicate to the web service. By using the open source development environment NetBeans [7], the applet and the libraries can be digitally signed. This allows the applet to communicate with a web service not located on its source web server.

The chat client is interrupt driven and activates upon interaction from a user. This is shown in Fig. 3; where the control unit decides what component to launch next and what function to process.

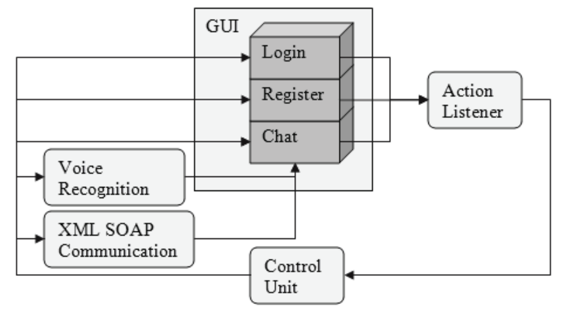


Fig. 3 – Applet management process

The action listener is bound to the buttons (login, register and chat) which is included in the graphical user interface (GUI). Once an event is triggered, either the corresponding SOAP communication module or voice recognition module or interface is activated. The input and output to these modules rely on user input captured via the GUI.

Voice recognition is accomplished using an open source library called Sphinx 4 [8]. Speech recognition can be broken into two groups with five subsets in total; speakers and speech styles. Speakers make up single speaking and speaking independently where speech styles include isolated word recognition, connected word recognition and continuous word recognition.

Isolated word recognition requires long breaks between all words to successfully interpret words. Continuous word recognition also requires this but considerably shorter breaks. The last subset of speech styles is continuous speech recognition where one can speak fluently and not require stopping or breaking between words. This has problems interpreting similar vowel in the beginning of words.

A vital part of the speech system is contained within a configuration file. As shown in Fig. 4, the configuration file is used to build the recognizer and (or) decoder using the application programming interface (API), including the dictionary. Grammar files which is used in generating the message is an integral part forming part of the acoustic model located in the API.

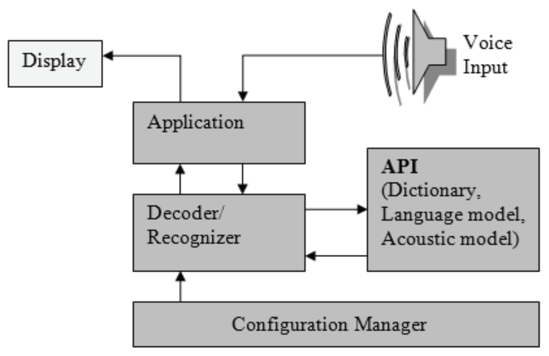


Fig. 4 – Speech recognition framework

The main components of the speech framework consist of the front-end application, decoder, and language model. The front-end acquires and attributes the voice input. The language and acoustic model is used to translate from a standard language (as input to the system) using a dictionary and construction of words located in a look-up table (LUT). A search manager located in the decoder then uses the attributes and LUT to decode the input voice into a result set. From the front-end, the user can activate the configuration manager, loading all components stored in XML format.

When running a speech enabled application, the application requires more than 256 Mb in heap size thus placing excessive memory demands on the system Although a more toned down API (Pocketsphinx [9]), focusing on mobile platform is available, this paper focuses of Spinx 4.

Communication with the web service is text based and XML-formatted using SOAP. The interface was generated using a JAVA-based architecture for XML binding (JAXB). An XML schema based API can also be published on a website for other developers to develop their own clients. This process of generating the interface is shown in Fig. 5.

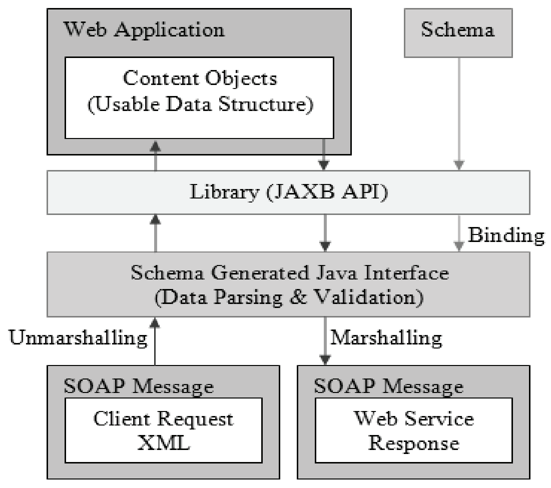


Fig. 5 – Object binding process

All messages are parsed through this interface which creates structure and validates the XML at the same time. The process of parsing the XML to a usable structure is termed marshalling.

When an object is created, the XML content is handled using the object extraction through the interface class created on startup as indicated in the next code segment.

```
JAXBContext.newInstance jaxbContext =
JAXBContext.newInstance( "icbxml" );
Unmarshaller u = jaxbContext.createUnmarshaller();
u.setEventHandler(new ICBXMLValidationEventHandler());
outputPipe.connect(inputPipe);
byte[] bytes = xmlData.getBytes();
outputPipe.write(bytes);
JAXBElement<IcbXml> mElement =
(JAXBElement<IcbXml>)u.unmarshal(inputPipe);
IcbXml icbxmlvar = (IcbXml) u.unmarshal(inputPipe);
IcbXml po = (IcbXml)mElement.getValue();
```

The interface also provides fault response generation if a XML message is invalid. The error location is then encapsulated in XML using the bound object and sent using SOAP. The interface can be regenerated on demand if the requirements or format of the message format changes.

The server maintains a large number of users through the process of synchronized threads. The most basic hypertext transfer protocol (HTTP) connection or request for content is done through sockets. When a client connects through a basic connection request (socket connection) the socket creates a new thread pair (incoming, outgoing and processes) to handle the client messaged. All messages pulled from the socket are then processed through the generated interface object.

The thread pair queues all messages though a synchronized push and pop principle. Messages are pulled from the queue for processing in a controlled fashion, as shown in Fig. 6. This optimizes processing time.

At the end of a queue, only the relevant thread would go to sleep for a pre-determined time period. This still allows the other threads to continue parsing messages in the queue and transmitting them out.

When a user logs on, the system will by default greet the user, and then prepare to receive a question or a statement. ALICE [10] is an open source foundation developing AI chat systems. An implementation of the ALICE-bot engine, Chatterbean uses its algorithm to generate a response using its library for pattern assimilation (targeting), namely AIML files.

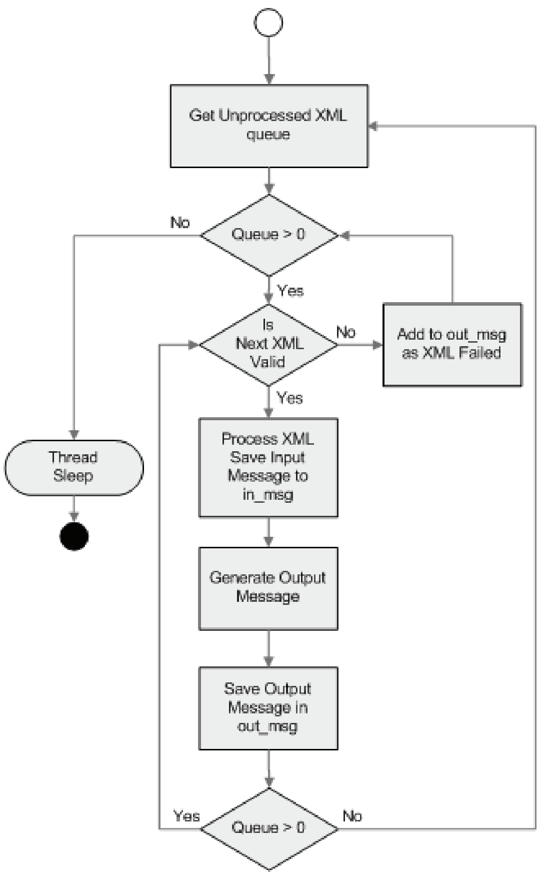

Get Unprocessed XML queue
Queue > 0
No
Yes
Is Next XML Valid
No
Add to out_msg as XML Failed
Yes
Thread Sleep
Process XML Save Input Message to in_msg
Generate Output Message
Save Output Message in out_msg
Yes
Queue > 0
No

Fig. 6 – Processing thread

The bot is centered on supervised human assistance for a learning process. This helps to remove or indicate correct answers as the system learns from conversations with users and updates the AIML files. AIML files are structured to consist of XML formatting, with context descriptions as indicated in the next code segment.

```
<category>
<pattern>How old are you</pattern>
<template>
<think><set name="topic">Me</set></think>
I am as old as the mountains.
</template>
</category>
```

Although a large set of AIML files are there to guide the bot it does not dictate a response. These sets of categories make up groups of questions and answers. Thus the intelligence is limited. If a question appears where no response can be generated (which will happen often in the infancy of an AI system) the system will try to change the topic or send a general statement. This is considered a trick for inadequate knowledge but is a first step for any chat bot.

To assist in the training, a training module was created to process content acquired from the internet related to statements or questions which the system cannot understand. This would require a complex algorithm to not only search for related content source but provide for concise content. Thus a third party expert system, “Ultimate Research Assistant” [11] was used to generate a detailed report relating to such a statement. This process is shown in Fig. 7.

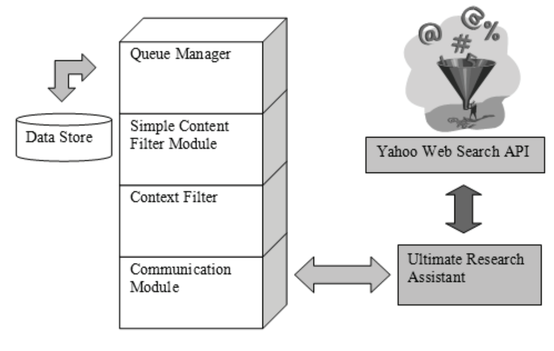


Fig. 7 – Content request utilizing the Ultimate Research Assistant

To get the most relevant response from the expert system, the query sent to it needs some refinement. To refine the query, self processing using AIML-assisted categories is used. The process is methodic (Fig. 8), for instance, no result may be generated from the entered query “Who is Albert Einstein?,” the processing proceeds iteratively to secondary file(s), where the same query is posed. The result of this query may be “a human icon,” the processing proceeds, and the resultant response is a combination of the output seeked from the secondary file and a standard response (“I am sorry… provide the right questions.”)

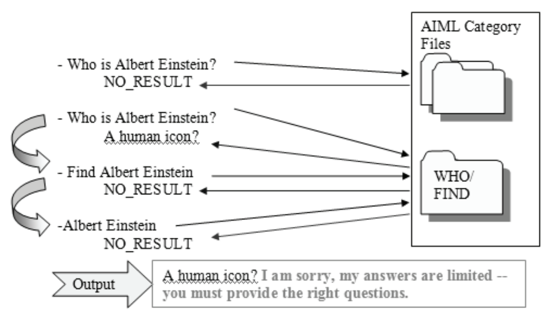

AIML Category Files
- Who is Albert Einstein?
NO_RESULT
- Who is Albert Einstein?
A human icon?
- Find Albert Einstein
NO_RESULT
-Albert Einstein
NO_RESULT
WHO/ FIND
Output
A human icon? I am sorry, my answers are limited -- you must provide the right questions.

Fig. 8 – ALICE-based processing of statement/question using AIML-assisted categories

In the particular instance, “Albert Einstein” is not understood, but also becomes the refined query. The refined query is then queued as an input to the expert system. As the queue is processed, reports are generated, which further populates the AIML files. This process is shown in Fig. 9.

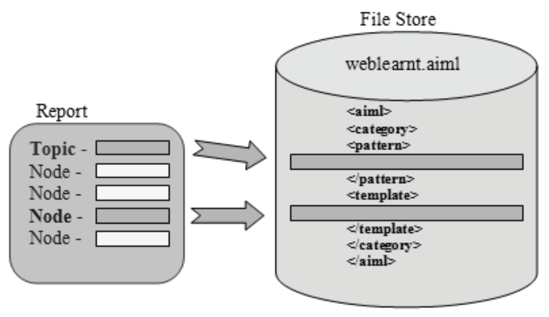


Fig. 9 – AIML file population process

The process of Fig. 9 is autonomous and needs to be moderated minimally to control the AIML file expansion. This process results in improving the intelligence of the system.

VI. RESULTS

The combination of voice input and voice output allows for a simpler experience which allows a client to run on many types of platforms. Since the client is internet based the next step would be mobile or even thin client systems. A thin client system is considered an embedded computer or platform with limited processing where the processing is done by a controlling server. Examples include a mini computer on a fridge, GPS unit or a mobile phone. Although the aim of this paper was to implement an intelligent virtual online friend, with voice, this integration of technologies can also be used in other applications, particularly when dealing with a need for simple input control accessibility and/ or limited processing capabilities.

The system resulted in a distributed environment to allow for resource management and stability between modules. This is shown in Fig. 10, handling the hosting of the site, processing responses using the ALICE-bot engine, content acquisition and processing using an expert system to increase the intelligence of the chat bot autonomously.

The use of the distributed framework allows for an increase in throughput and the number of users it can handle. The lifetime of the expert system can be a limitation to the age of the technology demonstrator presented in this paper.

VII. CONCLUSION

Using modular design for all its components a distributed environment facilitating transparent and high performance of the overall system has been created. The performance is relative to the processing capacity of the systems involved. Since all the modules are not running off one system the possible load has been decreased and further decreased by delegating the voice processing to the chat client communicating with the service.

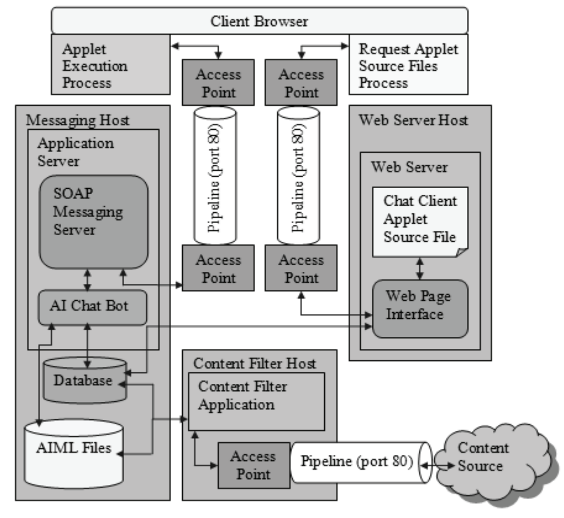


Fig. 10 – Distributed system framework

The use of an expert system (Ultimate Research Assistant) allows unlimited and autonomous intelligence improvements. Conventional implementations of the ALICE-bot engine required an administrator to update the AIML files manually to increase the intelligence. All content received back from the expert system is processed minimally since the information has already been processed for its relevance. This can be somewhat subjective considering that the intelligence reliance was shifted to another party and when such a third-party system is decommissioned this system would also fail.

The core use of threads allowed multiple processing of incoming and outgoing messages to occur without having to create a waiting scenario or unavailable server due to over use or possible congestion. All new connections to the server spawned a new pair of threads without impacting on other threads related to other users.

The use of such a framework is not limited to chat applications only. The true potential lies with information systems that could be built into the existing framework due to the distributed nature. Depending on the requirements of the integration, the necessary additional components can be introduced on the artificial brain level or the AIML file functionality.

REFERENCES

[1]. Augello A. Saccone G. Gaglio S. Pilato G., Humorist Bot: Bringing Computational Humour in a Chat-Bot System. Proceedings of the International Conference on "Complex, Intelligent and Software Intensive Systems (CISIS)", 4-7 March 2008, Barcelona, Spain, pp.703-708.
[2]. Gambino O. Augello A. Caronia A. Pilato G. Pirrone R. Gaglio S., Virtual conversation with a real talking head. Proceedings of the Conference on "Human System Interactions", 25-27 May 2008, Kraow, Poland, pp. 263-268.
[3]. Vojtko J. Kacur J. Rozinaj G., The training of Slovak speech recognition system based on Sphinx 4 for GSM networks. Proceedings of International Symposium "EL, MAR (Electronics in Marine) focused on Mobile Multimedia", 12-14 Sept. 2007, Zadar, Croatia, pp. 147-150.
[4]. Sun Microsystems, Developer resources for JAVA technology. [Online] http://java.sun.com (Accessed: 30 Oct. 2008)
[5]. The Apache Software Foundation, The Apache HTTP Server Project. [Online] http://www.apache.org (Accessed: 30 Oct. 2008)
[6]. Sun Microsystems, MySQL: The world's most popular open source database. [Online] http://www.mysql.com (Accessed: 30 Oct. 2008)
[7]. Sun Microsystems and CollabNet, NetBeans. [Online] http://www.netbeans.org (Accessed: 30 Oct. 2008)
[8]. Carnegie Mellon University, Sun Microsystems, Mitsubishi Electric Research Laboratories, Sphinx-4 - A speech recognizer written entirely in the JAVA™ programming language, 2004. [Online] http://research.sun.com (Accessed: 30 Oct. 2008)
[9]. Carnegie Mellon University (CMU). Speech at CMU. [Online] http://www.speech.cs.cmu.edu (Accessed: 30 Oct. 2008)
[10]. ALICE AI Foundation, Inc. ALICE. the artificial linguistic internet computer entity. [Online] http://www.alicebot.org (Accessed: 30 Oct. 2008)
[11]. A. Hoskinson, Ultimate Research Assistant. [Online] http://ultimate-research-assistant.com (Accessed: 30 Oct. 2008)

Salomon Jakobus du Preez completed his National Diploma in Technical Applications from the Tshwane University of Technology (TUT), South Africa in 2007. Currently, on a part-time basis, Mr du Preez is nearing the end of his B.Tech programme. On a full-time basis, Mr du Preez serves Prospero SA (Pty) Ltd as a Senior software developer. Mr du Preez has been a graduate student member of the IEEE since 2006.

Manoj Lall completed his B.Eng (Mechanical) and M.Sc. (2005) in Computer Science from the University of South Africa. On a part-time basis, Mr Lall is also pursuing a PhD programme with the University of South Africa (UNISA). He is currently employed as a Senior lecturer by TUT. His research interests include Mobile Agent Systems and Service Oriented Architecture.

Saurabh Sinha (M'02) completed his B.Eng degree (cum laude), M.Eng degree (cum laude) and PhD(Eng) degree from the University of Pretoria, South Africa, in 2001, 2005, 2008 respectively. He is currently employed by the University of Pretoria, South Africa where his main activities include research, undergraduate and postgraduate training. Dr Sinha also serves as a consultant for Business Enterprises at University of Pretoria (Pty) Ltd. Dr Sinha is the Chair of the IEEE South Africa Section. In 2008, Dr Sinha was invited to serve on the IEEE Membership and Geographic Activities Board (MGAB), as a representative to the Educational Activities Board (EAB). Dr Sinha is also a full-voting member of the EAB Operation Committee, as well as the EAB Finance Committee. Dr Sinha also received the South African Institute of Electrical Engineers (SAIEE) Engineer of the year award in 2007.