Who Discovered the Electron Neutrino? A Telephony-Based Distributed Open-Source Standard-Compliant Spoken Dialog System for Question Answering

Tarek Mehrez[1,2,5]*, Abdelrahman Abdelkawy* [1,5]*, Youmna Heikal*[1,5]*,*
Patrick Lange[1,3,4]*, Hadeer Nabil*[1,5]*, David Suendermann-Oeft*[1]

[1]DHBW, Stuttgart, Germany
david@suendermann.com
[2]University of Stuttgart, Germany
[3]Linguwerk, Dresden, Germany
patrick.lange@linguwerk.de
[4]Staffordshire University, Stafford, UK
[5]German University in Cairo, Egypt
{tarek.mehrez,abdelrahman.mostafa,youmna.heikal,hadeer.nabil}@student.guc.edu.eg

ABSTRACT

The development of spoken dialog systems (SDSs) has traditionally assumed two orthogonal directions dependent on whether academia or industry was concerned. The former has been led by the conviction that systems should be as human-like and data-driven as possible and available for free. Accordingly, academic dialog management methodologies are dominated by MDP, POMDP, agenda-based, or incremental techniques. In contrast, industry has focused on practical systems, interchangeability of components, ease of development, and profitability. Consequently, over the past decade, multiple standard protocols describing architecture and interoperability of SDS components were agreed upon. While these standards are freely distributed by the W3C, software components in the speech industry are mainly proprietary.

In this paper, we present an SDS architecture that is to bridge the gap between these two worlds. By taking open-source components from heterogeneous sources (Sphinx, FreeTTS, Cairo, Zanzibar, JVoiceXML, Apache, Ruby on Rails, Asterisk, and OpenEphyra), we established an SDS that is callable from SIP clients as well as PSTN phone lines. All these components were designed or adapted to adhere to the aforementioned industry standards including VoiceXML, JSGF, SIP, and MRCP. The resulting system can be pointed at a start URL hosting a VoiceXML page featuring the SDS's logic and can be called immediately without further setup. Thereby, the design of SDSs and their execution get decoupled. By distributing components across multiple servers (telephony, voice browser, MRCP, and web server) and hosting them in a virtual environment allowing for rapid cloning, the infrastructure is easily scalable. To encourage the community to engage in the presented endeavor, we are opening our infrastructure and hardware to beta testers. To demonstrate the described framework, we built a question answering service similar to IBM's Watson but with speech recognition input. In doing so, we used source code provided by one of Watson's creators.

I. Introduction

The past two decades have witnessed a remarkable escalation in the field of speech and language technology. This development was primarily due to advances in computational power, amount of available corpora to train data-driven techniques, and the use of more sophisticated models. Deploying speech recognizers, speech synthesizers, and language understanding technology in real-world spoken dialog applications became feasible and common practice since the beginning of the millenium. SDSs became a common solution for various services that include banking, stock transactions, information enquiries, and other customer services. Over the last five years, a considerable movement from the aforementioned transactional systems towards open-domain voice assistants could be observed. Modern systems such as Apple's Siri and IBM's Watson DeepQA are able to process virtually any natural language query and provide correct answers or engage, especially in case of Siri, in a dialog with the user.

The vast majority of past activities in the development of commercial SDSs resulted in proprietary software solutions. As opposed to potential open-source alternatives, such software comes along with substantial disadvantages:

- it imposes considerable licensing fees;

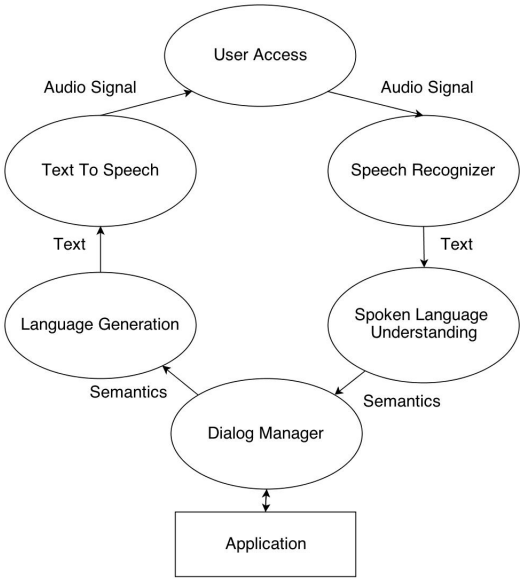

Fig. 1. A basic SDS architecture

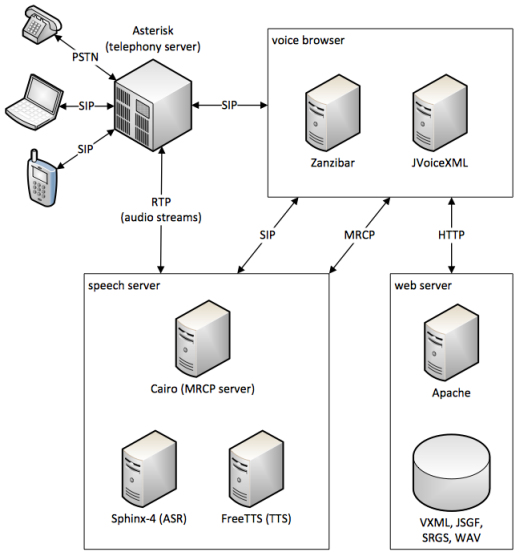

Fig. 2. Components of Halef

- the consumer relies entirely on the vendor when it comes to feature enhancements, troubleshooting, service quality, etc.;
- the compliance to industry standards (for grammars, dialog logic, transport etc.) is not guaranteed as there is no common code base.

Due to the above reasons and because of repeated requests by the community, the authors decided to develop an SDS infrastructure based on open-source components only. As opposed to existing open-source solutions such as CMU's Olympus initiative, the new framework is to adhere to industry standards such as VoiceXML, JSGF, MRCP, SIP, etc. to make it a true alternative to commercial systems.

We chose to use a distributed architecture for our SDS giving us the privilege of a reusable system which is flexible to new updates, newly integrated tools or enhancements to the existing ones. The communication between the underlying components is achieved through multiple standard protocols: session initiation protocol (SIP) [11] for call handling, real-time transport protocol (RTP) [15] for audio streaming, and media resource control protocol (MRCP) [16] handling the exchange between voice browser and speech recognition and synthesis components. A basic architecture that examines the typical components of an SDS is shown in Figure 1.

Inspired by the fictional character Hadschi Halef Omar, the overly communicative adventurer invented by Karl May, one of the most popular German authors, we branded our system HALEF (Help Assistant—Language-Enabled and Free). Halef is designed to adhere to the aforementioned industrial standards and limits the developer's concerns to the VoiceXML logic, rather than building up the SDS architecture from scratch. This is reached by having distributed components that point to the VoiceXML application's URL. As example system to demonstrate Halef's capabilities, we integrated a question answering (QA) system for general purpose questions in the English language, similar to IBM Watson DeepQA.

II. Components

As depicted in Figure 1, a typical SDS consists of modules for speech recognition, language understanding, dialog management, language generation and speech synthesis. For those modules, Halef employs Sphinx, Open Ephyra, Ruby On Rails, Open Ephyra, and FreeTTS, respectively. Features implemented in VoiceXML depend on the developer's intention as previously mentioned. In our case, language understanding and generation are implemented within the question answering system Open Ephyra.

Briefly, Halef handles those components in a slightly different way than the usual sequential architecture shown in Figure 1. Several components are grouped together forming independent chunks which are, according to our distributed architecture, hosted on different virtual machines:

- the speech server that groups Sphinx and FreeTTS,
- the voice browser JVoiceXML, and
- the web server which hosts the VoiceXML code being fetched by the voice browser.

OpenEphyra is served on yet another server, and it is being called by the VoiceXML code on the web server. Exchange of audio streams is handled via various protocols as mentioned in the previous section.

Since Halef is a telephony-based system, a telephony server was required to pass users' calls into our architecture. System components are served on several virtual Linux machines. This ensures the system's flexibility in recombining different components, as we discuss later, and offers developers the ability to build a tailored architecture meeting their requirements. The distributed architecture of Halef, with reference to the several virtual machines holding separate components, is demonstrated in Figure 2 as first introduced in [17]. Each component is described individually in the upcoming subsections.

A. The telephony server

Following previous efforts in building Interactive Voice Response (IVR) systems [9], [10], [3], a central server was required to operate as the gateway between assorted resources and Halef's components.

We use the popular Asterisk PBX [7] which provides the essential features needed for building a central server for calling into our system. Those features were satisfied by Asterisk's capabilities that included supporting different protocols, several telephony sources such as PSTN landlines and SIP soft phones.

Multiple architectures and levels within the network were not a barrier, since Asterisk is capable of calling through different kinds of networks with various levels of complexity in the adhered proxy settings, which was an advantage for enabling calls from any possible source.

Asterisk passes the call through the SIP protocol to the voice browser, to initiate the sessions and resources in all components. Similarly, the audio streams are delivered to the speech server via RTP; they include streams for speech recognition and synthesis.

Furthermore, Asterisk's powerful dial plans provided a flexible solution to routing calls to a multitude of Halef instances. They were also used to overcome issues of the dialog manager such as proper termination of sessions or handling of concurrency.

B. Voice Browser

The voice browser consists of two main components, Zanzibar [9] and JVoiceXML [14]. The former is responsible for the communication between telephony server, voice browser and speech server using the SIP protocol. It determines required resources for the VoiceXML application and initiates a SIP session by notifying the speech server. This ensures that the user's call is forwarded to the speech server to perform speech recognition and to the voice browser which interprets the VoiceXML code.

Halef's voice browser is JVoiceXML, an open-source software that supports many commonly used industrial standards. JVoiceXML uses Zanzibar as MRCP gateway, the protocol to communicate with the speech server. In this way, Zanzibar is responsible for session initiation, resource management and issuing recognition and synthesis requests as we show later.

C. Web Server

VoiceXML applications are equipped with numerous features for handling call flows and dialog states. Dialog management, language understanding, syntactic and semantic parsing are possible by integrating standalone scripts and applications into VoiceXML.

In Halef, the actual VoiceXML application is served on a separate web server. Upon receiving the recognition result, the dialog manager—a Ruby On Rails application in our case—engages the QA web application which is served on yet another server. The QA web application is based on Open Ephyra [12], an open-source system which was developed by one of the researchers working on IBM's Watson DeepQA initiative [2]. It represents a combination of several components that form a sequence for question analysis, query generation, pattern matching, answer extraction and answer selection as displayed in Figure 3.

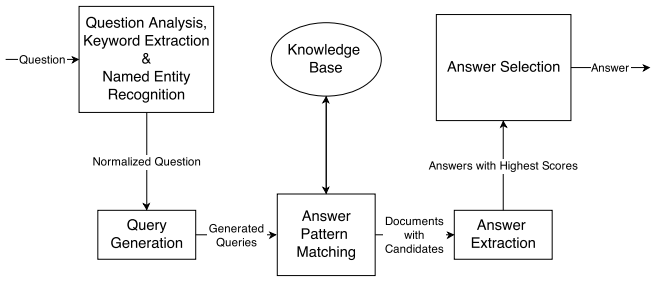

Fig. 3. Steps of performing QA analysis

First of all, the question is normalized to be accessible for query generation. This includes removing punctuations, expanding abbreviations and further stemming for nouns and verbs. Then keywords, question type, and named entities are extracted to generate the proper query for this question.

Queries are generated and expanded using the extracted data, along with synonyms extracted from WordNET [8]. Furthermore, queries are updated to include expected patterns and forms for the answer. The generated query is then used to search the available knowledge base.

Fetching the answer depends on the question's type. If the analysis phase managed to come up with a possible type for the question, a pattern matching method is used to map the extracted type to its corresponding pattern which is then used to extract the answers' candidates. However, if the analysis components failed to extract the question's type, another approach is followed in order to generate the answer's pattern using the property, target, and the context of the question. For instance, the question "Who invented Halef?" asks for the property (name) of a target (inventor) in the context (Halef). The generated pattern is then matched against the knowledge base searching for possible answers sharing the same pattern and, thus, the same property, target, and context. After matching the derived patterns, a list of candidates for the possible answer is returned, and the answer with the highest confidence score is chosen.

D. Speech Server

As speech server, we deploy Cairo [9] which supports MRCP. This server gives the voice browser the ability to establish a connection via SIP with the speech resources and also enable audio streaming via RTP between the MRCP server and Asterisk. The audio streams are then forwarded to Sphinx [20], the speech recognizer which is managed by Cairo. The recognized utterance is passed to the voice browser via MRCP and consumed by the VoiceXML application and, hence, the QA system.

On the other hand, when the previous sequence is reversed answers from the QA system are passed via MRCP to the speech server. It uses FreeTTS [19], an open-source tool

for speech synthesis, to speak back the answer by returning the audio stream to Asterisk again via RTP. Both speech resources, Sphinx and FreeTTS, are open-source tools, written in Java, which were integrated within the MRCP implementation. Sphinx supports multiple industrial standards used by VoiceXML like JSGF grammars [4]. Therefore, it stood out as the best fit for our architecture.

E. Bringing it all together

Figure 4 shows an example of Halef's execution flow. Since multiple processes are executed in parallel, we did not use a conventional sequential flow diagram.

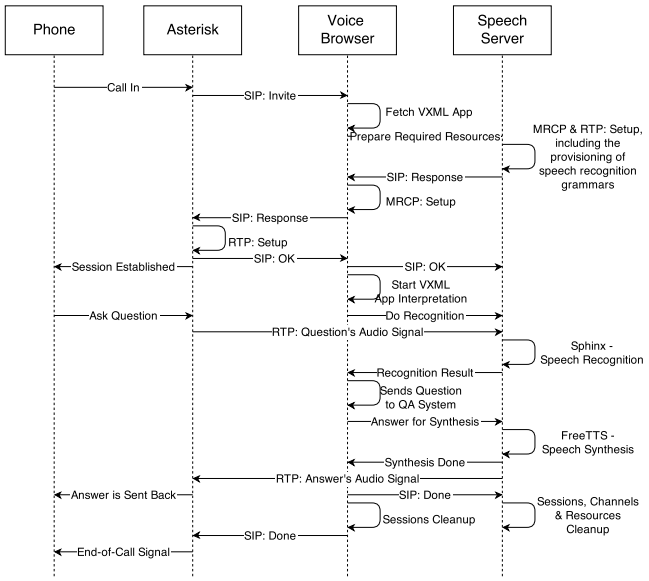

Fig. 4. Flow diagram for Halef

It starts with the user's call received by Asterisk. A notification is sent to the voice browser to fetch the VoiceXML code from the web server and to identify what resources are needed for this application to be prepared by the speech server. The MRCP server is then notified. It starts sessions and channels for all required resources including the provisioning of speech recognition grammars. Then, a SIP response is sent back to the voice browser and Asterisk. Now, the session initiation has been confirmed, and the communication channel between the user and Halef's components is successfully established.

Via RTP, audio is streamed from Asterisk to the speech server. When the caller starts speaking Sphinx's voice activity detector fires, and the respective audio is being decoded by the speech recognizer. When the voice activity detector finds that the caller has finished speaking, Sphinx sends the recognition result back to the voice browser which passes it on to the QA web application and awaits for an answer.

The answer is then sent back to the dialog manager which generates VoiceXML code with the answer to be spoken out by FreeTTS. When the voice browser receives the answer it proceeds interpreting the VoiceXML code and sends a synthesis request to the speech server with the answer. FreeTTS performs the synthesis, passes the result back via RTP to Asterisk which forwards the audio signal to the user. At the same time, Cairo sends a confirmation to the voice browser. After receiving this notification, the voice browser orders a cleanup request to close all open channels and resources. It ends the SIP session with Asterisk, and, finally, Asterisk sends an end-of-call signal to the user.

III. Dialog Examples

The following section demonstrates two dialog examples handled by Halef's QA application:

Dialog 1:
Halef: *Welcome to the Spoken Dialog Systems Research Center. Which extension would you like to be connected to?*
<User enters extension>
Halef: *What is your question?*
User: *Who was the 40th president of the United States of America?*
Halef: *The answer is Ronald Reagan. Goodbye!*

Dialog 2:
Halef: *<Introduction>*
<User enters extension>
Halef: *What is your question?*
User: *When did World War II end?*
Halef: *The answer is April 29, 1945. Goodbye!*

IV. Future Work

In order to improve Halef's speech processing capabilities, we are going to invest a significant effort to enhance the speech server's components by tuning Sphinx and FreeTTS, for instance by using techniques for voice adaptation [5], [18]. This also includes the support of ARPA statistical language models [1] for recognizing unconstrained speech by the user, instead of limiting the possible spoken input to what is covered by rule-based JSGF grammars. Also other syntactic and semantic grammar standards such as SRGS and SISR are to be implemented.

An important prerequisite in order to make Halef ready for commercial deployment is that the central telephony server is able to handle concurrent calls. This can be achieved by passing them to several Halef instances making use of the power behind Asterisk's dial plans. We also plan to demonstrate Halef's capabilities by means of other VoiceXML applications beyond the QA system described in the present paper. Such applications could include intelligent tutoring systems [6], assessment of English Language Learning [21], or automated troubleshooters [13].

V. Conclusion

In this paper, we gave a brief description of Halef, an open-source VoiceXML-based spoken dialog system. We built a composite infrastructure that is based on a combination of various open-source tools.

In order to investigate Halef's abilities, we integrated Open Ephyra, a question answering system that operates as a standalone web application.

Halef demonstrates how open-source speech and language technology can be deployed serving the same standards and functionality as commercial products. We maintain Halef as free software accessible via Sourceforge at

`http://sourceforge.net/projects/halef/`

and strongly encourage the community to engage in its continued enhancement. Our team also offers deployed Halef instances running on DHBW's infrastructure to beta testers who can test their VoiceXML applications hosted on their own servers. For this purpose, they will be provided a SIP connection or U.S. phone number pointing at their Halef instance free of charge.

REFERENCES

[1] P Clarkson and R Rosenfeld. Statistical language modeling using the CMU-Cambridge toolkit. *In Proc. of Eurospeech, 1997.*

[2] D Ferrucci, E Brown, J Chu-Carroll, J Fan, D Gondek, A Kalyanpur, A Lally, J Murdock, E Nyberg, and J Prager. Building Watson: an overview of the DeepQA project. *AI magazine*, 31(3), 2010.

[3] J Glass. Challenges for spoken dialogue systems. In *Proceedings of the 1999 IEEE ASRU Workshop*, 1999.

[4] A Hunt. JSpeech Grammar Format, W3C Note, June 2000. *See http://www.w3.org/TR/2000/NOTE-jsgf-20000605*.

[5] A Kain and M Macon. Personalizing a speech synthesizer by voice adaptation. In *The Third ESCA/COCOSDA Workshop (ETRW) on Speech Synthesis*, 1998.

[6] DJ Litman and S Silliman. ITSPOKE: An intelligent tutoring spoken dialogue system. *HLT-NAACL*, 2004.

[7] L Madsen, J Smith, and J Van. Asterisk: the future of telephony. *USA: O'Reilly*, 2005.

[8] G Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11), 1995.

[9] D Prylipko, D Schnelle-Walka, S Lord, and A Wendemuth. Zanzibar OpenIVR: an open-source framework for development of spoken dialog systems. *In Proc. of TSD, 2011*.

[10] A Raux, B Langner, D Bohus, A Black, and M Eskenazi. Let's go public! taking a spoken dialog system to the real world. In *in Proc. of Interspeech*, 2005.

[11] J Rosenberg, Schulzrinne, G Camarillo, A Johnston, J Peterson, R Sparks, M Handley, and E Schooler. SIP: session initiation protocol. Technical report, RFC 3261, Internet Engineering Task Force, 2002.

[12] N Schlaefer, E Nyberg, J Callan, J Carbonell, and J Chu-Carroll. *Statistical source expansion for question answering*. PhD thesis, Language Technologies Institute, School of Computer Science, Carnegie Mellon University, 2011.

[13] A Schmitt, M Scholz, W Minker, J Liscombe, and D Sündermann. Is it possible to predict task completion in automated troubleshooters? *In Proc. of the Interspeech, 2011*.

[14] D Schnelle-Walka. Architecture of JVoiceXML version 0.0. Technical report, Technische Universität Darmstadt, 2006.

[15] H Schulzrinne, S Casner, R Frederick, and V Jacobson. Real-time transport protocol. Technical report, RFC 1899, Internet Engineering Task Force, 1996.

[16] S Shanmugham and P Monaco. A media resource control protocol (MRCP). Technical report, RFC 4463, Internet Engineering Task Force, 2006.

[17] D Suendermann-Oeft. Modern conversational agents. *In Technologien fuer digitale Innovationen: Interdisziplinaere Beitraege zur Informationsverarbeitung, Springer VS*, 2013.

[18] D Sündermann, A Bonafonte, H Ney, and H Höge. Time domain vocal tract length normalization. *In Proc. of the ISSPIT, 2004*.

[19] W Walker, P Kwok, and P Lamere. FreeTTS open source speech synthesis. Technical report, Sun Microsystems, Inc., 2002.

[20] W Walker, P Lamere, P Kwok, B Raj, R Singh, E Gouvea, P Wolf, and J Woelfel. Sphinx-4: a flexible open source framework for speech recognition. Technical report, Sun Microsystems, Inc., 2004.

[21] SM Witt and SJ Young. Phone-level pronunciation scoring and assessment for interactive language learning. *Speech communication*, 2000.