Maareech: Usability Testing Tool for Voice Response System Using XML Based User Models

Siddhartha Asthana(✉) and Pushpendra Singh

Indraprastha Institue of Information Technology, Delhi, India
{siddharthaa,psingh}@iiitd.ac.in

Abstract. Interactive Voice Response Systems (IVRS) are popular voice-based systems to access information over the telephone. In developing regions, HCI researchers have shown keen interest in IVRS due to high affordability and reach among rural, poor, and illiterate users. However, IVRS are also notorious for their usability issues. This makes researchers thrive for more usable IVRS. The lack of automated usability testing tools for voice-based systems makes researchers depend on human subjects for testing their proposed IVR systems that are both costly and time-consuming. To address this research gap, we present Maareech, a usability testing tool for voice response systems using XML-based user models. Maareech has a flexible architecture to accommodate different user models that can be used to perform usability tests. In this paper, we discuss Maareech's architecture and its ability to mimic IVR user behavior based on different user models.

Keywords: User-models · Usability testing

1 Introduction

It is quite common that a telephone call gets attended by an automated Interactive Voice Response (IVR) system while accessing information about an organization [8,13,14]. The humble human operator has now been replaced by IVR systems [6]. In developing regions, IVR systems have emerged as a mean to disseminate information across all the sections of the society [11,16]. HCI researchers have shown the impact of IVR system in several contexts in rural areas including the vital areas of agriculture [9], and healthcare [10,12]. An IVR system provides information in natural language that transcends literacy and can be accessed through low-end phones thus reaching out to sections that cannot access information through other media like Internet or print media.

Due to widespread deployment of IVR systems, it has become imperative to test IVR systems before their deployment. A software developer can test the system functionality, but not the interaction issues faced by its users. The solution to this problem could be to conduct a usability test with few participants through the lab studies, but these lab studies are time-consuming, costly and

DOI: 10.1007/978-3-319-20886-2_10

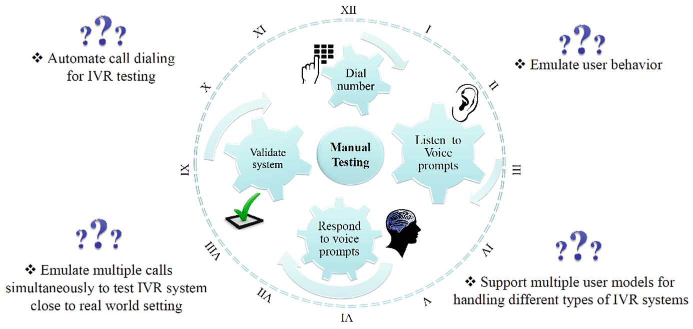

???
❖ Automate call dialing for IVR testing
???
❖ Emulate user behavior
???
❖ Emulate multiple calls simultaneously to test IVR system close to real world setting
???
❖ Support multiple user models for handling different types of IVR systems
Dial number
Listen to Voice prompts
Respond to voice prompts
Validate system
Manual Testing
XII
I
II
III
IV
V
VI
VII
VIII
IX
X
XI

Fig. 1. Steps of IVR testing that are automated by Maareech in call emulation

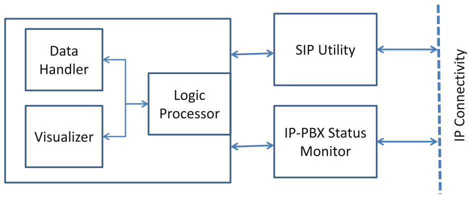


Fig. 2. Maareech's architecture

cannot be done at a large scale. In this paper, we present Maareech[1], a tool that can mimic user behavior to simulate large-scale user testing of a system. Maareech is a complete and robust testing tool to test IVR systems. We had discussed the preliminary design of Maareech in [5], in this paper, we present the complete working system. Maareech has the ability to dial a phone number, listen to a voice prompt, enter the required DTMF[2] or recorded speech input for testing different types of IVR applications (see Fig. 1). Maareech can also use data generated from real world calls for call emulation. Mimicking user behavior provides the ability to optimize and evaluate the performance of IVR applications. Maareech is built to help HCI researcher conducting usability tests. It has the capability to incorporate new user models based on which an HCI researcher can test different system user interactions.

2 Architecture

Maareech has a modular architecture with five basic modules as shown in Fig. 2. It is written in JAVA and designed using the MVC (i.e. Model, View, Controller) pattern. This makes Maareech highly customizable and extendable to different scenarios. In this section, we describe the architectural components of Maareech.

[1] Maareech is a daemon in Hindu Epic Ramayana who could assume any form.

[2] Dual-tone multi-frequency signaling (DTMF) is used for telecommunication signaling over analog telephone lines.

Logic Processor: This module is responsible for making all the decisions for the emulation process, e.g., call initiation, event regeneration, etc. It coordinates with other modules to perform call emulations. The Logic processor interprets the underlying user modules. New user models can be created under the Logic Processor to emulate user behavior for different scenarios. It can schedule events and maintain call queues[3].

Visualizer: This module provides the user interface of Maareech. It is responsible for taking inputs from the user for different emulations and showing the output to the user (see Fig. 2). The Visualizer component is responsible for showing calls loaded in the call list, events in each call, and the current operation performed by Maareech in the test.

SIP Utility: This module is responsible for performing all SIP (Session Initiation Protocol) based communication with the IP-PBX software hosting the IVR application. It provides an API for initiating and releasing calls, generates a DTMF key-press, and sends the audio file as a speech utterance. In the current implementation, the SIP Utility of Maareech uses the API of PJsua[4] for all SIP-based communications. PJsua can be replaced with another Java based SIP stacks such as MjSip[5] for developers requiring more control.

Status Monitor: This module gathers information about the present state of IVR by directly communicating with the IP-PBX software. Any IP-PBX with a *command line interface (CLI)* can interact with Maareech by changing the connection configuration in the Status Monitor. This communication enables Maareech to know about the current configuration of the voice menu played by the IVR application hosted on IP-PBX software such as FreeSWITCH or Asterisk. In the current implementation, we have used fs_cli that is a CLI utility for connecting to FreeSWITCH (IP-PBX software).

Data Handler: This module is responsible for reading input data files (in XML format) created from logs of the IVR application and IP-PBX (FreeSWITCH), and converting it into a Java based object and vice-versa. All XML-based communication is done through JAXB[6]. The schema of XML documents defines the underlying user models used in Maareech. Maareech has been designed with rich data models to capture complex call scenarios. The data file contains logs about user responses and their corresponding contextual information that are generated by logging facilities of IP-PBX software. The read data is supplied to the Logic Processor module to emulate desired user behavior. The Data Handler module is also responsible for creating new objects for telephonic queues and call objects in each queue.

[3] By call queue, we refer to multiple calls scheduled to be initiated by Maareech one after the other.

[4] pjsua is an open source command line SIP user agent (softphone) system http://www.pjsip.org/pjsua.htm.

[5] MjSip is a complete Java-based implementation of a SIP stack. http://www.mjsip.org/.

[6] Java Architecture for XML Binding http://www.oracle.com/technetwork/articles/javase/index-140168.html.

3 User Models in Maareech

We categorize currently available IVR applications, e.g. [2–4], into two categories: Static menu based IVR and Dynamic menu based IVR. Considering that, we have designed two user models to mimic user behavior.

Simple User Model: Figure 3(a) represents a simple user model that has two composite attributes: meta-data and events. The meta-data attribute has four sub-attributes used for storing the meta-data of the call.

– UniqueId: A unique identifier for the call.
– From: Telephone number of the caller.
– TimeStamp: The time at which this call was made.
– System: The system specific identifier where the IVR system has multiple IVR applications running on it.

User responses are captured in an event attribute that has three sub-attributes: InputType, Data, and Time.

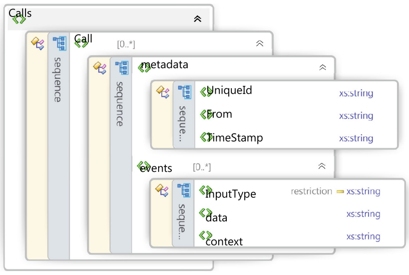
– InputType: It categorizes user responses into key-presses or audio.
– Data: It describes the series of DTMF key-presses or names of audio files, depending upon the corresponding InputType.
– Time: It captures the time in seconds at which the input was generated.

This simple user model is sophisticated enough to replay any call emulating user behavior for testing IVR applications that have static menu configuration. However, to test IVR applications where menu configuration may change dynamically, we need a richer user model.

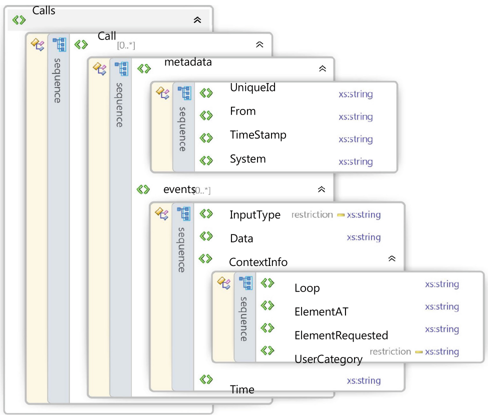
Intricate User Model: Some IVR applications change their menu configuration based on the time of the day, user preferences etc. Thus, the IVR menu configuration may be based upon information provided by the users, their history, and other factors like time, etc. Faithful emulation of the user behavior of earlier calls in the changed menu configuration requires prediction about user behavior in a new configuration. Correspondingly, it requires that the data used for emulation should have all the intricacies of user behavior required for a new configuration well represented in it. To handle such IVRs, a more intricate XML schema is required as shown in Fig. 3(b).

The Intricate user model contains a tuple of 4 elements (i.e., InputType, Data, ContextInfo, Time) to describe the user response and the context in which the response was created. The InputType and Data attribute in the tuple are same as in the simple user model. This model has an additional attribute, ContextInfo. It is a composite attribute made up of 4 sub-attributes (i.e., Loop, ElementAt, ElementRequested, and UserCategory).

– Loop: It defines the number of times the menu was listened before selecting an option.
– ElementAt: It captures the announcement at the time the users made their selection.



(a) Simple User Model



(b) Intricate User Model

Fig. 3. Maarech Load: The figure on left side (a) shows CPU usage of Maareech in terms of static, running and total load. The figure on right side (b) shows memory usage of Maareech in terms of static, running and total load.

– ElementRequested: This captures the option selected by a user.
– UserCategory: Each user is categorized in one of the two categories: expert or naive. An expert user is the one who selects an option ahead of the full completion of the announcement or just after the announcement was made. The user who waits for the announcement of several menu options before selecting a particular option is categorized as naive.

The intricate user model effectively captures the exact menu options (in case of advanced adaptive IVR systems) at the time of selection. This data is helpful in predicting user behavior in the new context where menu options are reconfigured. The events are regenerated when the contextual information stored in the XML document matches the context in the call.

4 Features

In Maareech, calls are emulated using two modes: user emulation using the previous call records, and testing using random DTMF generation.

User Emulation Using Previous Call Records: In this mode, Maareech reads call logs stored in XML files that are obtained from a real world deployment. Maareech supports two formats that capture user models:

Simple User Emulation: This user emulation works for currently available static IVR systems. In this format, Maareech replicates call events like key-press and speech recording, as they happened in an actual call. In this mode, Maareech does not assume any assistance from IVR applications hosted on the telephony server (FreeSWITCH or Asterisk). Real data stored for this emulation contains a time-stamp for each event relative to the start of the call. The events are generated based on the time-stamp captured in this model.

Intricate User Emulation: The intricate user emulation model has been designed for upcoming personalized IVR systems, where menu options sequence may change in order to give better services. Maareech keeps track of menu options as they were accessed in the original call and responds to a correct menu option even if the menu sequence has changed. In this mode, the announcement of each menu option is assumed to be a state. Maareech responds to this state based on the response to the states stored in the data used for emulation. This mode assumes that the IVR application announces its state over the IP-PBX console as they occur. Maareech does not support sound processing or any other similar technology to capture and recognize the state. The IVR application announces its state on a command line interface. The status monitor in Maareech captures the state information through the command line interface utility of the IP-PBX software. In the current implementation, the application assumes this command line interface to be the FreeSWITCH console. Maareech connects to this console using the fs_cli utility that comes with preinstalled FreeSWITCH binaries.

We have incorporated some more features in Maareech that are helpful in analyzing IVRs for different usages. Maareech provides two basic features in this mode. The trial run of each feature was tested on a machine (HP Probook 4520s) running Ubuntu 12.04 with an Intel Core i5 processor and 2 GB RAM.

Call Reordering: This feature allows to change the original sequence of call arrivals on the IVRS. It is suitable for analyzing upcoming IVR applications that have dynamic menu configuration. It helps to study the dynamic IVR system that will behave differently for the same calls presented in the different order. In a trial run of Maareech, 1120 calls collected from real world usage, were shuffled in less than 3 s.

Number of Telephone Lines: This feature allows developer to check problems which may arise due to shared access to system resources (e.g., accessing the same audio file for reading or log file for writing) when multiple instances of the IVR application handles more than one simultaneous call. By default, Maareech assumes a single line connection with the telephony server and only one call at a time is simulated as per the current call sequence. With this feature, the number of telephone line connections can be increased to view the behavior of the IVR application in handling multiple connections. In a trial run, Maareech was able to open eight lines with linphone[7] (a free SIP VoIP Client) as the SIP utility.

Testing Using Random DTMF Generation: In this mode, Maareech does not read any data file or log to perform tests. These tests are independent of user models and can be used to evaluate the IVR application's system parameters. It generates events like key presses or speech recording based on parameters specified by the user. It supports four types of IVR tests, as shown in Fig. 4:

[7] www.linphone.org.

Call Load Test: This test helps in measuring the number of simultaneous calls an IVR application can process. Maareech can test an IVR application hosted at a remote location also, which enhances its utility. On starting this test, the system increases number of calls made to IVR till it fails to handle any more calls. This test reports the integer value at which the IVR application crashes. In our test, we found that FreeSWITCH was able to process 123 simultaneous calls for our sample IVR application under test. The IVR application hosted on FreeSWITCH failed on 124^{th} call because of too many connections open to the MySQL database operating at backend. We would like to mention that this is not a limitation of Maareech, but of the FreeSWITCH module that connects to database. Maareech helped in identifying it and did not report any failure as FreeSWITCH (call receiving module) was running properly even though IVR application was not accepting additional calls.

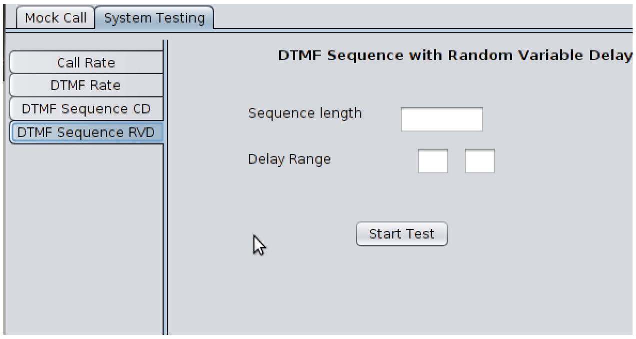


Fig. 4. The tabs on the left showing four type of IVRS tests.

DTMF Rate: This tests helps to detect the rate at which an IVR application can accept input. Maareech starts this test by sending 1 DTMF per second to IVR application under test and keeps on increasing number of DTMF per second. This test reports the integer value beyond which IVR do not accept more DTMF per second.

Sequence Test: This test helps to detect the DTMF input sequence at which an IVR application may fail. The Maareech generates random DTMF key sequences, each with a constant time delay, in seconds, as defined by the IVR developers. An IVR developer needs to specify the desired sequence length to be generated by Maareech. At the end of the test, Maareech reports the sequence test cases if any, at which the IVR application fails to respond. In our test, Maareech was able to check 100 sequences, of sequence length 5 and delay of 5 s within each option, in 2,504 s. In this, 2500 s were taken due to the conditions specified in the test and 4 s in setting up the call.

Sequence Test with Random Delay: This test detects erroneous DTMF inputs in a more complex manner. The constant delay is not helpful when the length of voice prompts played by the IVR application varies for the announcement of menu options. If voice prompts of menu options are of different lengths, then to test such IVR applications, different delays must be put between each DTMF input generated by Maareech. Random delays help create more test cases than just random DTMF sequences. In this test, Maareech generates random DTMF key sequences each with random time delays ranging from t_1 to t_2 s, where t_1 and t_2 are defined by the user in seconds. Ideally, t_1 should correspond to the length of the minimum voice prompt and t_2 be the length of the maximum voice prompt in IVR. This test is more rigorous than the previous sequence test. At the end of the test, Maareech reports the sequence test case at which the IVR application ends the call. In our test, Maareech can check 100 sequences, of sequence length 5 and delay range 5 to 7 s, in 3094 s.

5 Performance Evaluation

To evaluate the performance of Maareech, we conducted an experiment using the *Call Load Test* feature in Maareech. For this experiment, we setup an IVR application written in JAVA and hosted on FreeSWITCH. FreeSWITCH and Maareech were running on two different machines referred as Machine-I and Machine-II respectively, with LAN, 100 Mbps, connectivity between them. We chose this configuration to reflect the real world deployment. Table 1, shows the hardware and software configuration of each machine. We started the *Call load* test feature of Maareech. It initiated a call every two seconds while keeping previously initiated calls alive until the end of the experiment or when they were terminated by FreeSWITCH. In total, we initiated 1024 calls through Maareech. A call in this experiment can be in one of three states:

Table 1. Hardware and Software configuration of machines used for experiment

	Machine-I	Machine-II
OS	Ubuntu 10.04	Ubuntu 12.04
Processor	Intel Core 2 Duo	Intel Core i5
Memory	3 GB DDR2	2 GB DDR3
Model	HP Compaq dx7400	HP Probook 4520s

- Active State: A call that is connected to FreeSWITCH and is not being terminated from either side (Maareech or FreeSWITCH).
- Dropped State: A call that was connected to but later terminated by FreeSWITCH.
- Time-out: A call whose resources were released by Maareech as it was not able to connect to FreeSWITCH.

Figure 5 shows time-series of call states from our test. Initially, the FreeSWITCH was able to accept the calls as the load on Machine-I was low. Due to this, there were no dropped calls observed till the 233^{rd} call. After this, FreeSWITCH started dropping calls at a higher rate that reduced the count of active calls. The number of Active calls stabilizes around 145 calls, beyond this all new calls

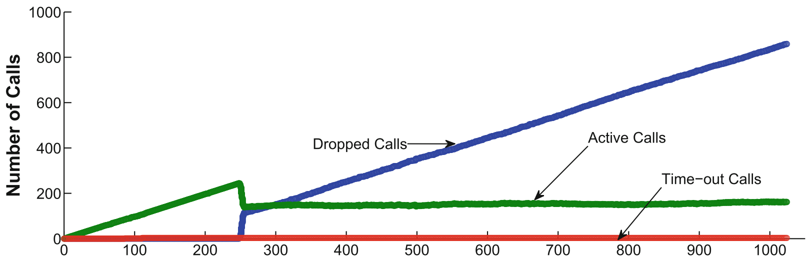


Fig. 5. X-axis represents the calls initiated by Maareech. The four lines representing calls in active,dropped and timed-out states.

are dropped. Our results for the number of active calls (i.e. concurrent calls) for FreeSWITCH running on Machine-I are in-line with the results obtained by other developers as available on FreeSWITCH website[8]. The CPU and memory load were also measured on both the machines during the tests.

FreeSwitch Load: We measure the CPU usage and memory consumption through Linux utilities (e.g. ps) on Machine-I. Figure 6(a) shows the CPU consumption of FreeSWITCH on Machine-I. CPU load saturated around 233rd Call. After this, the rate at which calls were dropped become nearly equal to rate at which new calls were accepted. We also found that three calls timed-out at 110th call because of high network congestion between the two machines.

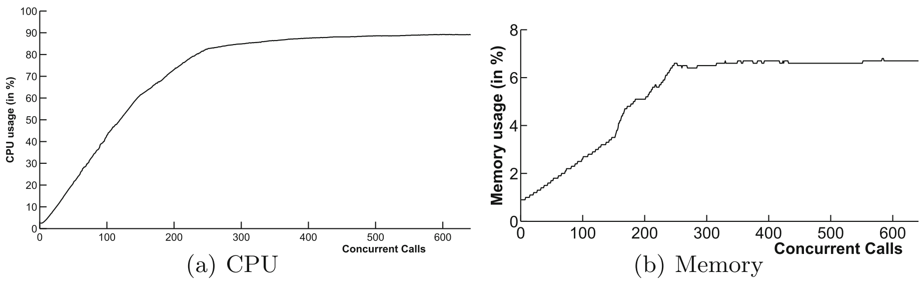


Fig. 6. FreeSWITCH Load: On the left side figure shows CPU usage of FreeSWITCH. The Y-axis represents the CPU usage of one core and X-Axis represents the total number (alive + dropped) of calls connected to FreeSWITCH. Figure on the right side shows memory usage of FreeSWITCH. The Y-axis shows the percentage of memory used by FreeSWITCH and X-axis represents the total number of calls connected to FreeSWITCH.

Figure 6(b) shows memory usage of FreeSWITCH in percentage of total memory on Machine-I. Similar to CPU usage, memory usage also saturated around 233rd Call. This shows that memory requirement of FreeSWITCH did

[8] Real-world performance data around the FreeSWITCH community. http://wiki.freeswitch.org/wiki/Real-world_results.

not increase after 233rd call as the number of active calls were saturated due to the equal rate of calls dropped and calls accepted by FreeSWITCH.

Maareech Load: Maareech creates logs of various parameters (e.g. CPU usage, memory usage and call data rate) to collect data related to its performance. We categorize the memory load and CPU load into two categories: static load and running load. Static load refers to CPU usage in creating and holding the call objects. Running load refers to CPU load incurred due to handling of underlying SIP communication. Total load is the sum of static and running load. We measured static and running load separately as the respected operation is handled by two different processes.

Figure 7(a), shows CPU usage distribution of Maareech in terms of static load, running load and total load. We can observe that the CPU usage was maximum at 113th call and decreases and stabilizes after 233rd call. We did further investigation and believe that CPU usage increased because of network congestion as the three calls were also timed-out around maximum CPU usage because of network congestion. We also measured the static, running and total load on memory usage. Figure 3(b), shows memory load of Maareech. We find that static load gradually increases from 8.1 % to 9.1 % (i.e. 1 % increase)for emulating 1024 calls. Similarly running load varied from 17.9 % to 18.5 %. Thus, it shows initiating each call in Maareech has memory load of 20 KB (calculated as 1 % of 2 GB system memory divided by 1024 calls).

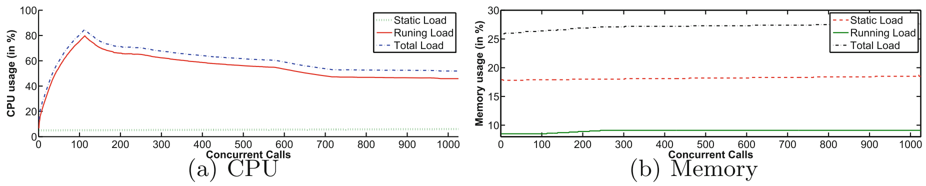


Fig. 7. Maarech Load: The figure on left side (a) shows CPU usage of Maareech in terms of static, running and total load. The figure on right side (b) shows memory usage of Maareech in terms of static, running and total load.

6 Related Work

Simulators have been used in various computer science domain like network simulators [15], and analog and digital circuit simulators. Simulators in the IVR domain are primarily used to simulate call-centers[9]. Various industrial tools provide pre-deployment testing of IVR applications. It mainly includes testing of VoIP infrastructure [1]. *Empirix Hamper*[10] provides an extensive tool-set for IVR

[9] http://www.call-center-tech.com/.
[10] http://www.empirix.com.

testing, monitoring and analyzing end-to-end IVR deployment. Nexus8610[11] is a traffic generator that simulates user behavior of various communication technologies including 3G/2G Mobile, VoIP, and PSTN. Cyara Solutions[12] is one of the industrial players that provides IVR testing as a service. Tools like Call center simulators[13] help estimate the resource requirements for optimal performance of IVR.

Although, the industry has a variety of tools for testing IVR at the infrastructure level, developers are still doing manual testing or writing customized test scripts for each IVR application. Hence, an emulation tool like Maareech, which is capable of mimicking user behavior is required to automate testing of IVR applications.

7 Conclusion and Future Work

Measuring the performance of any interactive system involving human is a challenging task. In this paper, we have presented Maareech - a call emulator for user behavior. This enables a developer to test the IVR application from user experience perspective. We presented two user models for testing and measuring the performance of different IVR applications. User model based testing provides characteristic to model different users easily [7] and reduces human effort. Current implementation of Maareech has certain limitations that we would like to address. Currently, Maareech does not have any speech or voice processing capabilities to understand the voice prompts played by IVR application under test. As a result, Maareech can not be used for verification of voice and sound quality. Maareech also lacks any automated data analysis of collected logs, and all analysis need to be manually done on the collected log. Also, the current implementation of Maareech has support only for FreeSWITCH. In future, we intend to overcome some of these challenges.

With increasing use of IVRS, it is imperative to have a robust testing tool for IVR applications. We have built Maareech for the same purpose. Over the time, Maareech has evolved to be very robust and has been used in testing of IVR applications that are deployed in the field.

References

1. Agam, O.: Voice over IP Testing-A Practical Guide. RADCOM White Paper, Bitpipe Inc (2001)
2. Asthana, S., Singh, P.: Mvoice: a mobile based generic ICT tool. In: Proceedings of the Sixth International Conference on Information and Communications Technologies and Development: Notes-Volume 2, pp. 5–8. ACM (2013)

[11] http://www.nexustelecom.com/products/nexus8610/.
[12] http://www.cyarasolutions.com/.
[13] http://www.xjtek.com/anylogic/demo_models/4/.

3. Asthana, S., Singh, P., Kumaraguru, P., Singh, A., Naik, V.: Tring! tring!-an exploration and analysis of interactive voice response systems. In:4th International Conference on Human Computer Interaction (2012)
4. Asthana, S., Singh, P., Singh, A.: Exploring the usability of interactive voice response system's design. In: Proceedings of the 3rd ACM Symposium on Computing for Development, p. 36. ACM (2013)
5. Asthana, S., Singh, P., Singh, A.: Mocktell: exploring challenges of user emulation in interactive voice response testing. In: Proceedings of the ACM/SPEC International Conference on International Conference on Performance Engineering (Poster), pp. 427–428. ACM (2013)
6. Chakraborty, D., Medhi, I., Cutrell, E., Thies, W.: Man versus machine: evaluating IVR versus a live operator for phone surveys in india. In: Proceedings of the 3rd ACM Symposium on Computing for Development, p. 7. ACM (2013)
7. Eckert, W., Levin, E., Pieraccini, R.: User modeling for spoken dialogue system evaluation. In: Proceedings of IEEE ASR Workshop, pp. 80–87 (1997)
8. Gupta, A., Thapar, J., Singh, A., Singh, P., Srinivasan, V., Vardhan, V.: Simplifying and improving mobile based data collection. In: Proceedings of the Sixth International Conference on Information and Communications Technologies and Development: Notes-Volume 2, pp. 45–48. ACM (2013)
9. Patel, N., Chittamuru, D., Jain, A., Dave, P., Parikh, T.S.: Avaaj otalo: a field study of an interactive voice forum for small farmers in rural india. In: Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, pp. 733–742. ACM, New York (2010)
10. Sherwani, J., Ali, N., Mirza, S., Fatma, A., Memon, Y., Karim, M., Tongia, R., Rosenfeld, R.: Healthline: speech-based access to health information by low-literate users. In: International Conference on Information and Communication Technologies and Development, ICTD 2007, December 2007, pp. 1–9. IEEE (2007)
11. Singh, A., Naik, V., Lal, S., Sengupta, R., Saxena, D., Singh, P., Puri, A.: Improving the efficiency of healthcare delivery system in underdeveloped rural areas. In: 2011 Third International Conference on Communication Systems and Networks (COMSNETS), pp. 1–6. IEEE (2011)
12. Singh, P., Singh, A., Naik, V., Lal, S.: CVDmagic: a mobile based study for CVD risk detection in rural india. In: Proceedings of the Fifth International Conference on Information and Communication Technologies and Development, pp. 359–366. ACM (2012)
13. Srinivasan, V., Vardhan, V., Kar, S., Asthana, S., Narayanan, R., Singh, P., Chakraborty, D., Singh, A., Seth, A.: Airavat: an automated system to increase transparency and accountability in social welfare schemes in india. In: Proceedings of the Sixth International Conference on Information and Communications Technologies and Development: Notes-Volume 2, pp. 151–154. ACM (2013)
14. Vashistha, A., Thies, W.: IVR junction: building scalable and distributed voice forums in the developing world. In: 6th USENIX/ACM Workshop on Networked Systems for Developing Regions (2012)
15. wiki: Network simulator-2. Article, January 2012. http://nsnam.isi.edu/nsnam/index.php/Main_Page
16. Yadav, K., Naik, V., Singh, A., Singh, P., Kumaraguru, P., Chandra, U.: Challenges and novelties while using mobile phones as ICT devices for indian masses: short paper. In: Proceedings of the 4th ACM Workshop on Networked Systems for Developing Regions, p. 10. ACM (2010)