Automated Testing of Interactive Voice Response Applications

Siddhartha Ashthana
Indraprastha Institute of Information Technology
New Delhi, India
siddharthaa@iiitd.ac.in

Pushpendra Singh
Indraprastha Institute of Information Technology
New Delhi, India
psingh@iiitd.ac.in

ABSTRACT

Dialing a telephone number of an organization and coming across an automated system attending the call, instead of a human, has become common. These automated applications are known as Interactive Voice Response (IVR) systems. IVR system provides 24x7 connectivity to an organization and many critical applications like customer care services and call routing run on IVR. Thus, it is important to have well tested IVR applications with all the necessary optimization tested by the IVR developers. A variety of tools available to test the IVR system at the infrastructure and system levels. However, a lack of testing tools at the application level, often forces developers to write their own customized test scripts or manually test the system. Manual testing is time consuming as it involves actually calling and listening to how an IVR application behaves.

In this paper, we present MockTell, a tool for the call emulation with ability to mimic user behavior. MockTell has ability to dial a phone number, listen to a voice prompt, enter the required DTMF[1] or recorded speech input for testing IVR applications. MockTell 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. MockTell also enables developers and researchers to automate various other application tests as well, thus providing a complete testing tool for IVR applications.

Keywords

Emulators, IVR, Tool, Testing, User Modelling

Categories and Subject Descriptors

H.1.2 [**User/Machine Systems:**]: Human factors

General Terms

Design, Experimentation, Human Factors

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

1. INTRODUCTION

It is quite common that a telephone call gets attended by an automated Interactive Voice Response (IVR) systems. Commercial organizations uses IVR systems for various purposes e.g., customer care services, credit card processing, employee benefit surveys, order processing and status, inventory confirmation, and billing inquiry as well as various social applications [17] [16]. The global IVR market is expected to reach $2.78 billion by the year 2017 [12]. These statistics show that the IVR based solutions have become critical to businesses. Often IVR systems are the only point of contact for the caller when they request a service. Therefore, it has become essential to have well tested IVR systems to ensure high quality in terms of robustness, stability and correctness of IVR systems. This requires IVR developers to be equipped with effective tools to test and validate the IVR systems.

IVR testing can be classified into infrastructure, system and application levels. Infrastructure testing requires testing of network elements and resources like switches, voice gateways, bandwidth, etc. Testing at the system level includes testing of telephony servers, database servers and MRCP[2] servers for various loads. Application level testing includes testing of IVR scripts (or applications) for program flow, voice recording, error handling, etc. User experience with IVR system depends upon presentation of appropriate menu structures to the user, the amount of time taken in each announcement, and various other application level functionality.

It is essential to test IVR system at all levels. At present, various industrial tools[3] are available to test the network performance of IVR infrastructure and this helps to maintain the quality of service. Such tools provide basic user simulation which is mostly confined to initiating and releasing the call and testing DTMF signalling across different technologies like 2G/3G/VoIP/PSTN [13]. However, testing the IVR at the application level requires emulation of user behavior i.e., dialing a number, listening and responding to voice prompts. In the absence of a testing tool at application level, testing is either done manually or through customized test scripts written by developers. Such scripts are often application specific and can not be reused to test other IVR applications.

Moreover, IVR systems are evolving [10, 14, 15, 1, 2, 6, 3, 4, 5, 8] and they are getting more advanced day by day. It has become difficult to test and optimize these more complex IVR applications.

[2]Media Resource Control Protocol (MRCP) is a network protocol that implements a common interface to a range of speech engine types, including speech recognition (ASR), speech synthesis (aka TTS), speaker or voice recognition (aka speaker verification), and simple voice recording. An MRCP resource server will provide access to one or more of these resources.

[3]`http://www.tmcnet.com/voip/0506/tech-roundup-voip-testing-tools.htm`

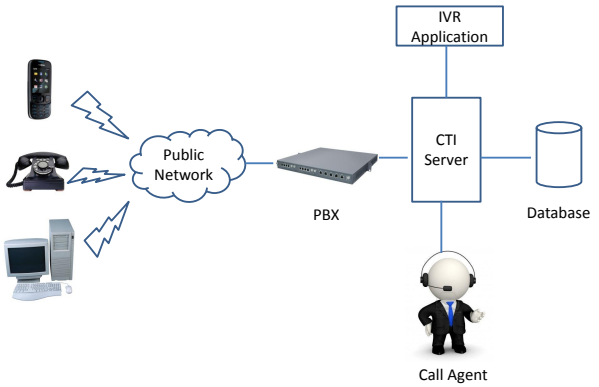

Figure 1: Components of IVR infrastructure.

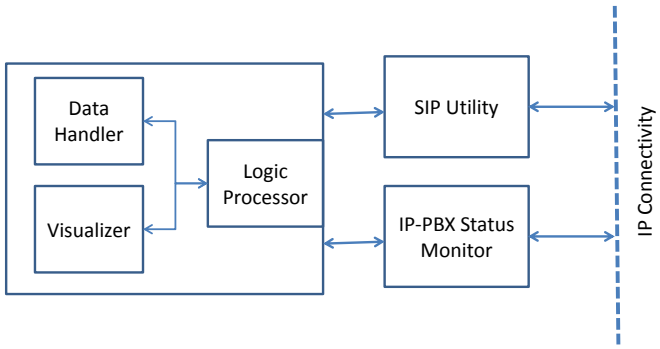

Figure 2: MockTell Architecture

Such IVR applications may also adapt to caller needs [2], time of the day (greeting Good Morning or Good Afternoon), caller status (registered or unregistered), etc. In a recent work, Malik et al. [11] showed the dynamic rearrangement of IVR menu based on user responses. Such capabilities present new challenges for emulation as a call emulator should o know the current configuration of the IVR menu to emulate any event in the call.

In this paper, we present MockTell [7], a tool for call emulation that is capable of mimicking user behavior. MockTell is a highly customizable IVR application level testing utility and provides a huge degree of abstraction because of its modular architecture. MockTell also has the in-built ability to read and process logs generated by standard IP-PBX (Private Branch Exchange - used to connect private organization) software like FreeSWITCH. The modular architecture of MockTell allows more such capabilities to be added if required. To the best of our knowledge, no such tool to test IVR applications at the application level is available in public domain.

2. IVR INFRASTRUCTURE AND TESTING

An IVR application (or script) is written and hosted on telephony platform like FreeSWITCH[4] or Asterisk[5]. The IVR application hosted on a telephony platform communicates with servers like a database server and a MRCP server(all servers including the telephony platform may run on the same server). This closed integration of telephony platform and servers form a complete IVR system. Usually, the IVR system is a part of bigger infrastructure, in which network and communication elements participate in receiving and routing the calls across different networks. Figure 1 shows the various components of IVR infrastructure in a simpler form. A public network connected to PBX is used to route the call internally to an organization where it reaches the CTI (Computer Telephony Integration) server. CTI is used for automated routing based on the caller identification number. Databases are used to maintain customer profiles and other personal information about the callers.

There are a variety of tests employed to check each network and communication element involved in the IVR infrastructure. Below is a list[6] of some tests employed at the system and application level for testing various aspects of IVR functionality.

- Infrastructure & System Test
  - Database Access Test: Verifies that the IVR retrieves the correct values from the database server.
  - Disabled Communications Test: Tests the IVR for cases of communication breakdown.
  - Text To Speech Test: Tests text to speech conversion under high load. It requires testing of load handling at the MRCP server.
  - Database Update Test: Tests whether updates are done properly at the database server.
  - System Load Test: Tests the systems behaviour under heavy load.
- Application Tests
  - IVR Program Flow Test: Check the key presses leading to a correct flow in IVR menu tree.
  - IVR Voice Recordings Verification: Checks if all of the voice recorded messages are played at the correct pace and correct length.
  - IVR Error Condition Tests: Tests the IVR behaviour when invalid options are selected (i.e., the wrong keys are pressed).
  - Voice Recording Test: Tests the quality of audio recording made by the IVR.

Application level tests like the Program Flow test and the Error Condition test require giving different user inputs to application under test. This requires a testing tool to have the ability to mimic user behavior. As existing testing tools lack the ability to mock user behavior, many application level tests are usually done manually or through customized test scripts written by IVR developers. We address this requirement for application level testing by proposing an emulator to test the performance of menu based IVR by mimicking user behavior. Call emulation based on user mocking has several associated challenges, which we describe in the next section.

3. MOCKTELL REQUIREMENTS AND CHALLENGES

In this section we present the key challenges that need to be tackled for designing an emulator that can mock user behavior.

- User modeling: User modeling is an essential task for mimicking user behavior. A user model should be simple enough

[4]http://freeswitch.org/
[5]http://www.asterisk.org/
[6]http://www.call-center-tech.com/ivr-tests.htm

to implement using available technologies and correct enough to represent user behavior up to a desired level. User behavior with respect to a particular IVR system is often characterized by the familiarity of a user with that system and familiarity with IVR systems in general. In a typical IVR scenario, a user may select an option before listening to it, after listening to it, after listening to several options or not choose it at all. Thus, it is required to capture the specific context governing user behavior. By context, we mean the specific setup of IVR at the application level i.e., sequence of menu options, user knowledge about menu options, etc. The task of user modeling becomes more challenging when the estimation of user behavior is to be done in a context different from the context in which user responses were observed earlier, as it happens in the case of dynamic menu based IVRs.

- Event regeneration: A user's response to an IVR menu announcements is an event. Each event has associated context in which it was created. The event regeneration requires identification of correct contextual information appearing on the IVR application under test. Emulating the user behavior requires regenerating the user response as DTMF or speech events. User responses, collected as events, are regenerated during call emulation for mocking user behavior, in response to the appropriate announcement in the call.The emulator needs to keep a track of current announcements in the IVR application and wait for the corresponding announcement to regenerate the user response at an appropriate time. The challenge involved in event regeneration is to continuously monitor the contextual values from the IVR application and recreate the user behavior at an appropriate time.

- Data collection: Collecting data in a format that can be quickly processed is another challenge. Each call can have variety of scenarios describing different user behavior, e.g., barge-in[7], invalid key-presses, time taken to respond, etc. The data structure chosen to store and process this data should be generic enough to represent all the complexities in the call. For data collection, a call is viewed as a sequence of announcements and user's response to announcements. The user response along with the corresponding announcement identifier needs to be recorded as data. This data is used for mocking user behavior while performing the call emulation.

  Another challenge in recording data is to choose the format that best represents users and their contextual information, e.g., the response time of the user for an announcement, the type of announcement responded by the user etc. The collected data should be of sufficient resolution such that any uncertainty at the time of mocking the user behavior can be resolved by the collected data itself. The data format should also be extensible to allow more contextual information to be stored if required.

4. SYSTEM DETAILS

In this section, we will discuss the architecture, design and features of MockTell in detail.

4.1 Architecture

Mocktell has a modular architecture. MockTell's architecture has five basic modules as shown in Figure 2.

[7]In an automated telephone system, experienced users are accustomed to interrupting the system ahead of menu announcement to quickly navigate to the next prompt.

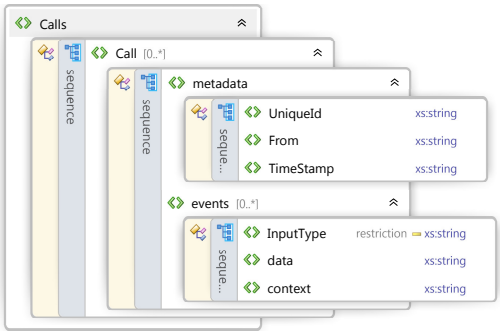

Figure 3: Simple user model

- Logic Processor: This module is responsible for making all the decisions for the emulation process e.g., call initiation, event regeneration, etc. The Logic processor module coordinates with other modules to perform call emulations. New user models can be created under the Logic Processor to emulate user behavior for scenarios of different complexities.

- Visualizer: This module provides the user interface of MockTell. It is responsible for taking inputs from the user for various different emulations and showing the output of call emulation to the users as shown in Figure 5. The Visualizer component is responsible for showing calls loaded in the call list, events in each call, and the current operation by MockTell in the current 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 the call, generates a DTMF key-press and sends the audio file as a speech utterance.

- Status Monitor: This module gathers information about the present state of IVR by directly communicating with the IP-PBX software. This module helps MockTell to trigger regeneration of an event.

- Data Handler: This module is responsible for reading the input data file, created from logs of the IVR application and IP-PBX (FreeSwitch), and converting it into a Java based object and vice-versa. The data file contains the logs about user responses and their corresponding contextual information and are generated by logging facilities of IP-PBX softwares. Data read by the Data Handler is supplied to the Logic Processor module to emulate the desired user behavior. The Data handler module is also responsible for making new objects for the telephonic queue and call objects in each queue.

4.2 Design and Implementation

MockTell is written in JAVA. It is designed using the MVC (i.e. Model, View, Controller) pattern. The MVC pattern provides abstraction for data handling, user interface and logic implementation. This makes MockTell highly customizable and allows for extending it to different scenarios. In this section, we describe the design and implementation of the architectural components of MockTell followed by a description of user models used in it.

- Logic Processor: This is a Java class which does not have any functional dependency outside MockTell. It has embedded logic to interpret the user models of MockTell. It uses

J2SE classes like *java.util.TimerTask* and *java.util.Collections* to schedule the events and to maintain the call queue[8] as java objects respectively.

- Visualizer: This is a module for handling the GUI in MockTell. Visualizer is a Java class which uses *javax.swing* library. This module fetches the state information of an ongoing operation (e.g. Number of calls initiated) in MockTell and displays it on graphical interface of MockTell.

- SIP utility: In the current implementation, the SIP Utility of MockTell uses the API of PJsua[9] for all SIP based communication. PJsua can be replaced with another Java based SIP stack such as MjSip[10] for developers requiring more control over SIP communication.

- Status Monitor: The Status monitor module uses an IP-PBX specific command line interface (*cli*) to communicate with IP-PBX software. Any IP-PBX with *cli* can interact with Mocktell by changing the connection configuration in the Status monitor. This communication enables Mocktell 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 which is a cli utility for connecting to FreeSWITCH (IP-PBX software).

- Data Handler: Data Handler reads the data from data model stored in the form of XML document. These XML documents are based on XML schema defining the underlying user model used in MockTell. MockTell has been designed with rich data models to capture the complex call scenarios. The data comes from the log of FreeSwitch (or any other IP-PBX). The Data models used by Mocktell are stored in an XML document using a well defined XML schema. All the XML based communication is done through JAXB[11]. Data stored as XML is easy to read and understand. XML can also store the semantics of the data which can then be easily processed by computers. New XML schemas can be added to support new data definitions. In the next subsection, we describe the attributes of the data stored as an XML document used to support different user models in MockTell.

4.2.1 User Models in MockTell

We have designed two user models to mimic user behavior in call emulation for different IVR applications. We categorize IVR applications into two categories: Static menu based IVR and Dynamic menu based IVR. We describe the user models used in call emulation by MockTell for each type of IVR application.

- Simple user model: Figure 3 represents a simple user model which has two composite attributes: metadata and events. The metatdata attributes has four sub-attributes used for storing metadata 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 to the system
  - System: The system specific identifier in the case 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: The sub-attribute InputType categorizes the user response into keypress 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 this user response 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 the IVR application where menu configuration may change dynamically, we need a richer user model.

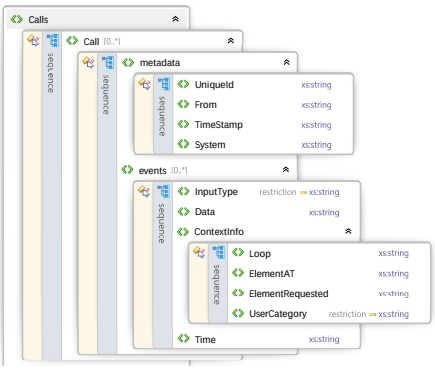

Figure 4: Intricate user modelling

- Intricate user model: Many IVR applications change the menu configuration based on the time of the day, user personalization and whether the user is registered or not. Thus, the IVR menu configuration may be based upon the information provided by the user, their history and other factors like time, etc. The task of user emulation becomes challenging if dynamic menu IVR presents a new configuration to a user who has called into the IVR system earlier. 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 Figure 4.

  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 is the 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).

[8] By call queue, author refers to multiple calls scheduled to be initiated by MockTell one after the other.
[9] pjsua is an open source command line SIP user agent (softphone) system. `http://www.pjsip.org/pjsua.htm`
[10] MjSip is a complete java-based implementation of a SIP stack. `http://www.mjsip.org/`
[11] Java Architecture for XML Binding `http://www.oracle.com/technetwork/articles/javase/index-140168.html`

– Loop: The Loop sub-attribute defines the number of times the menu was listened before selecting an option.

– ElementAt: This sub-attribute captures the announcement at the time the users made their selection.

– ElementRequested: This sub-attribute captures the option selected by the user.

– UserCategory: Each user is categorized in one of 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 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.

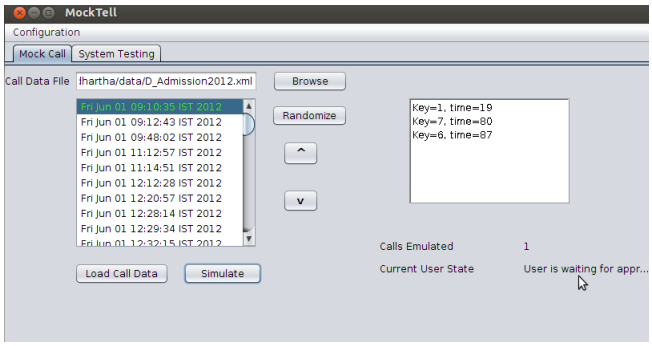

Figure 5: Screenshot of call data loaded by MockTell. The window on left side of the screen shows the list of calls by start time. Window on the right side shows DTMF key-presses and the time at which the key was pressed. MockTell has started emulated calls. User status and number of calls emulated is shown in right corner.

4.3 Features

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

4.3.1 User Emulation using previous call records

In this mode, MockTell reads the call log stored in XML files that were obtained from a real world deployment. MockTell supports two format that capture user models:

- Simple user emulation: This user emulation works for currently available static IVR systems. In this format, MockTell replicates the call events like key-press and speech recording, as they happened in an actual call. In this mode, MockTell does not assume any assistance from the IVR application hosted on telephony server (FreeSWITCH or Asterisk). Real data stored for this emulation contains a timestamp for each event, relative to the start of the call. The events are generated based on the time stamp captured in this model. Figure 3 shows the XML schema used for this mode to store real data.

- Intricate user emulation: This has been designed for upcoming personalized IVR systems, where menu options sequence may change in order to give better services. With the intricate user emulation model, MockTell can emulate calls in a more complex fashion. In this mode, MockTell responds to the states announced by the IVR. A state is a menu option played by an IVR application. MockTell keeps track of the 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. MockTell 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. MockTell does not support sound processing or any other similar technology to capture and recognize the state. In the current implementation, the IVR application announces its state on a command line interface. The status monitor in MockTell 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. MockTell connects to this console using the fs_cli utility that comes with preinstalled FreeSWITCH binaries. Figure 4 shows the XML data format used for this mode to store real world data.

We have incorporated some more features in MockTell that are helpful in analysing IVR for different usage. MockTell 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 2GB RAM.

- Call reordering: This feature allows the order of call arrivals to the IVR system to be re-ordered. It is suitable for analyzing the upcoming IVR applications which have dynamic menu configuration. Dynamic IVR systems present different menu options to a particular call which are often based upon the IVR usage of the calls received in the past. Hence, reordering the calls changes the number of past calls and past system usage for a particular call. Thus it helps to study the dynamic IVR system that will behave differently for the same calls presented in different order.

  In MockTell, these calls are loaded in sequence as per their timestamp, as shown in Figure 5. With this feature the calls can be reordered manually or randomly. There are options to move calls manually up or down in the order using the up and down buttons, respectively, or the user can press the randomize button to shuffle the whole call sequence. In a trial rum of MockTell, 1120 calls, collected from real world usage, were shuffled in less than 3 seconds.

- Number of telephone lines: This feature allows for the behavior of IVR applications to be studied and its scalability aspects to be examined by increasing or decreasing the number of telephone lines attached to it. It allows developer to check problems which may arise due to shared access of 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.

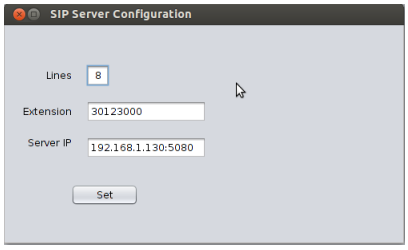

Figure 6: Configuring IP address of SIP server. Specifying number of telephone connections (SIP connection) to be open by Emulator.

By default MockTell 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. MockTell opens multiple connections with the telephony server and emulates multiple simultaneous calls based on the number of telephone lines specified in the configuration, as shown in Figure 6. In a trial run, MockTell was able to open 8 lines with linphone[12] (a free SIP VoIP Client) as the SIP utility.

4.3.2 *Testing using random DTMF generation*

In this mode, MockTell does not read any data file or log to perform tests. These tests are independent of user models and can be used to test the IVR application's system parameters. It generates events like key-presses or speech recording based on test parameters specified by the user. It supports three types of IVR tests, as shown in Figure 7:

- Call Load Test: This test helps in measuring the number of simultaneous calls an IVR application can process. MockTell has the ability to test an IVR application hosted even at a remote location which enhances its utility. On starting this test, the system increases number of calls made to IVR till the IVR application fails to handle any more calls. This test reports the integer value at which the IVR application crashed. 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 MockTell but of the FreeSwitch module that connects with the database. MockTell helped in identifying it. MockTell did not report any failure as FreeSwitch (call receiving module) was running properly even though IVR application was not accepting additional calls.

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

[12]www.linphone.org

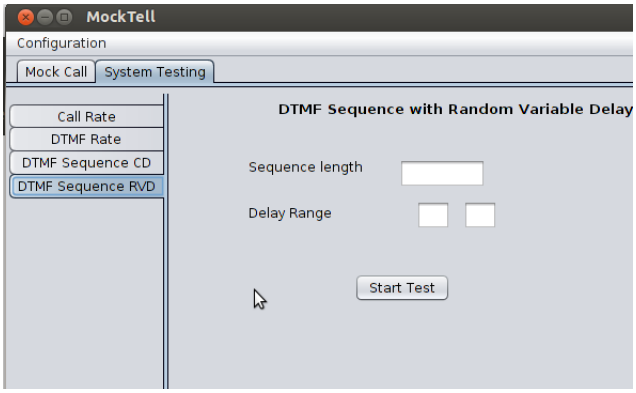

Figure 7: The tabs on the left showing various test MockTell can perform on a IVR application.

- Sequence test with random delay: This test helps to detect 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 MockTell. Random delays help to create more test cases than just random DTMF sequences. In this test, MockTell generates random DTMF key sequences each with random time delays in a range from t_1 to t_2 seconds 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, MockTell reports the sequence test case at which the IVR application ends the call. In our test, MockTell is able to check 100 sequences, of sequence length 5 and delay range 5 to 7 seconds, in 3094 seconds.

5. PERFORMANCE EVALUATION

To evaluate the performance of MockTell, we conducted an experiment using the *Call Load Test* feature in MockTell. For this experiment, we setup an IVR application written in JAVA and hosted on FreeSwitch. FreeSwitch and MockTell 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 MockTell. It initiated a call every two seconds while keeping previously initiated calls alive till the end of the experiment or when they were terminated by FreeSwitch. In total, we initiated 1024 calls through MockTell. A call in this experiment can be in one of three states:

- Active State: A call that is connected to FreeSwitch and is not being terminated from either side (MockTell or FreeSwitch).

	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

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

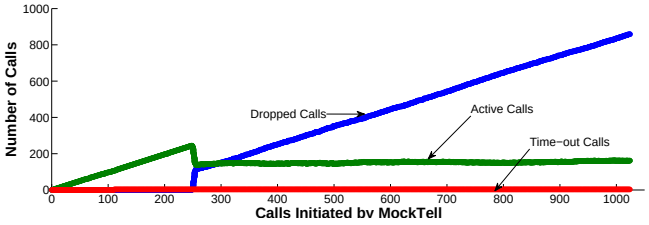

Figure 8: X-axis represents the calls initiated by the MockTell. The four lines representing calls in active,dropped and timed-out states.

- Dropped State: A call that was connected to but later terminated by FreeSwitch.
- Time-out: A call whose resources were released by MockTell as it was not able to connect to FreeSwitch.

Figure 8, shows timeseries 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 which reduced the count of active calls as shown in Figure 8. The Number of Active calls stabilizes around 145 calls, beyond this all new calls 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[13].**

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 9, shows the CPU consumption of FreeSwitch on Machine-I. The Y-axis shows the percentage of CPU used by FreeSwitch and X-axis represents the total number of calls connected to FreeSwitch at a particular instant. Figure 9, shows that CPU load saturated around 233^{rd} 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 3 calls were timed-out at 110^{th} call because of high network congestion between the two machines.

  Figure 10, shows memory usage of FreeSwitch in percentage of total memory on Machine-I. The Y-axis shows the percentage of CPU used by FreeSwitch and X-axis represents the total number of calls connected to FreeSwitch at a particular instant. Similar to CPU usage, memory usage also saturated around 233^{rd} Call. This shows that memory requirement of FreeSwitch did not increase after 233^{rd} call as number of active calls were saturated due to equal rate of calls dropped and calls acceptance by FreeSwitch.

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

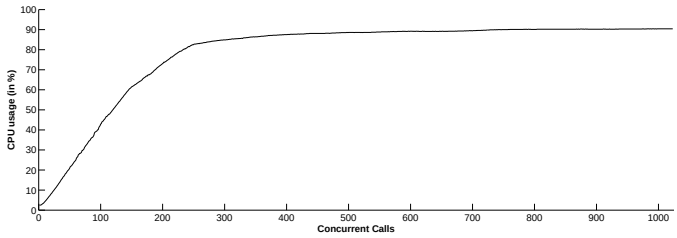

Figure 9: CPU usage of FreeSwitch: The Y-axis represents the CPU usage of one core and X-Axis represents the total number (alive + dropped calls)of calls connected to FreeSwitch.

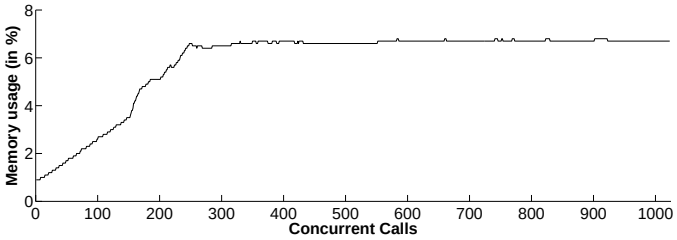

Figure 10: Memory usage of FreeSwitch: The Y-axis shows the percentage of memory used by FreeSwitch and X-axis represents the total number of calls connected to FreeSwitch at a particular instant.

- MockTell Load: MockTell creates logs of various parameters (e.g. CPU usage, memory usage, and call data rate) to collect data related to the performance of MockTell. We categorize the memory load and CPU load in 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 process.

  Figure 11, shows CPU usage distribution of MockTell in terms of static load, running load and total load. We can observe that the CPU usage was maximum at 113^{th} call and decreases and stabilizes after 233^{rd} call. We did further investigation and believe that CPU usage increased because of network congestion as the 3 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 12, shows memory load of MockTell.

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% for emulating the 1024 calls. Thus it shows initiating each call in MockTell has memory load of 20 KB (calculated as 1% of 2 GB system memory divided by 1024 calls).

6. RELATED WORK

Simulators have been used in various computer science domain like network simulators [18] and analog and digital circuit simu-

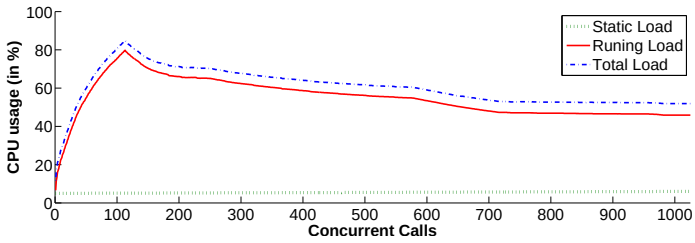

Figure 11: CPU usage of MockTell: CPU usage in terms of static, running and total load.

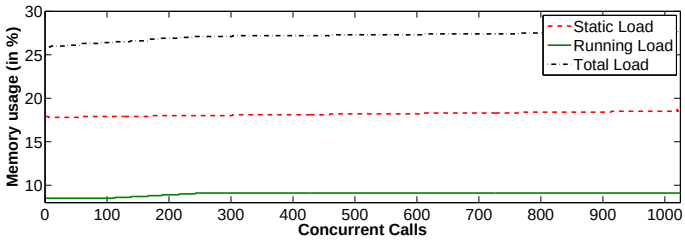

Figure 12: Memory usage of MockTell: Memory usage in terms of static, running and total load.

lators. Simulators in the IVR domain are primarily used to simulate call-centers[14]. Various industrial tools provide pre-deployment testing of IVR applications. *Empirix Hamper*[15] provides an extensive tool-set for IVR testing, monitoring and analyzing the IVR deployment from end-to-end. Nexus8610[16] is a traffic generator that simulates user behavior of various communication technologies including 3G / 2G Mobile, VoIP and PSTN. Cyara Solutions[17] is one of the industrial player which provides IVR testing as service. Tools like Call center simulators[18] help to estimate the resource requirements for optimal performance of IVR.

Although, the industry has a variety of tools for testing IVR at infrastructure level, developers are still doing manual testing or writing customized test scripts for each IVR application. Hence, an emulation tool like MockTell, which is capable of mimicking user behavior is required to automate testing of IVR applications. MockTell is available as open source software through git repository hosted at `https://github.com/siddasthana/IIITD.git`.

7. CONCLUSION & FUTURE WORK

Measuring the performance of any interactive system involving human is a challenging task. In this paper we have presented, MockTell, a call emulator for user behavior. This enables a developer to test the IVR application with 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 user easily [9] and reduces human effort. Current implementation of MockTell has certain limitations which we would like to address. Currently MockTell does not have any speech or voice processing capabilities to understand the voice prompts played by IVR application under test. As a result, MockTell can not be used for verification of voice and sound quality. MockTell lacks any automated data analysis of collected logs. All analysis need to be manually done on the collected log. Current implementation of MockTell has support only for FreeSWITCH.

[14] `http://www.call-center-tech.com/`
[15] `http://www.empirix.com`
[16] `http://www.nexustelecom.com/products/nexus8610/`
[17] `http://www.cyarasolutions.com/`
[18] `http://www.xjtek.com/anylogic/demo_models/4/`

8. REFERENCES

[1] Asthana, S., and 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*, ACM (2013), 5–8.

[2] Asthana, S., Singh, P., Kumaraguru, P., Singh, A., and Naik, V. Tring! tring! - an exploration and analysis of interactive voice response systems. *4th International Conference on Human Computer Interaction (IndiaHCI), Pune, India* (2012).

[3] Asthana, S., Singh, P., and Singh, A. Assessing designs of interactive voice response systems for better usability. In *Design, User Experience, and Usability. Design Philosophy, Methods, and Tools*. Springer Berlin Heidelberg, 2013, 183–192.

[4] Asthana, S., Singh, P., and Singh, A. Design and evaluation of adaptive interfaces for ivr systems. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, ACM (2013), 1713–1718.

[5] Asthana, S., Singh, P., and Singh, A. Exploring adverse effects of adaptive voice menu. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, ACM (2013), 775–780.

[6] Asthana, S., Singh, P., and Singh, A. Exploring the usability of interactive voice response system's design. In *Proceedings of the 3rd ACM Symposium on Computing for Development*, ACM (2013), 36.

[7] Asthana, S., Singh, P., and 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*, ACM (2013), 427–428.

[8] Asthana, S., Singh, P., and Singh, A. A usability study of adaptive interfaces for interactive voice response system. In *Proceedings of the 3rd ACM Symposium on Computing for Development*, ACM (2013), 34.

[9] Eckert, W., Levin, E., and Pieraccini, R. User modeling for spoken dialogue system evaluation. In *Proc. IEEE ASR Workshop* (1997), 80–87.

[10] Litman, D. J., and Pan, S. Designing and evaluating an adaptive spoken dialogue system. *User Modeling and User-Adapted Interaction 12* (2002), 111–137. 10.1023/A:1015036910358.

[11] Malik, D., and Anderson, R. A. Methods, systems, and products for dynamically-changing ivr architectures, 08 2010.

[12] Masterson, M. Strong growth projected for ivr market. Article, January 2012. `http://www.speechtechmag.com/Articles/Editorial/FYI/Strong-Growth-Projected-for-IVR-Market-79616.aspx`.

[13] Murphy, M.-L. Nexus8610 testing media gateways to enable convergence. White Paper, January 2007.

http://www.nexustelecom.com/documents/whitepapers/whitepaper_mgw_testing.pdf.

[14] Perugini, S., Anderson, T. J., and Moroney, W. F. A study of out-of-turn interaction in menu-based, IVR, voicemail systems. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, ACM (New York, NY, USA, 2007), 961–970.

[15] Resnick, P., and Virzi, R. A. Skip and scan: cleaning up telephone interface. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '92, ACM (New York, NY, USA, 1992), 419–426.

[16] Singh, A., Naik, V., Lal, S., Sengupta, R., Saxena, D., Singh, P., and Puri, A. Improving the efficiency of healthcare delivery system in underdeveloped rural areas. In *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, IEEE (2011), 1–6.

[17] Singh, P., Singh, A., Naik, V., and 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*, ACM (2012), 359–366.

[18] wiki. Network simulator-2. Article, January 2012. http://nsnam.isi.edu/nsnam/index.php/Main_Page.