MockTell: Exploring Challenges of User Emulation in Interactive Voice Response Testing

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

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

Amarjeet Singh
Indraprastha Institute of Information Technology, Delhi
New Delhi, India
amarjeet@iiitd.ac.in

ABSTRACT

Increasing use of telephone devices has made the Interactive Voice Response (IVR), a technology for accessing information over phone, popular among the commercial organizations. IVR systems are used for critical applications like flight reservation, tele-banking, etc. which requires to have well tested IVR systems. Manual testing of an IVR application requires dialing number, listening and responding to voice prompts through key-press or speech.
Automating the tests for IVR applications requires mimicking the user behavior. We present MockTell, a generic tool for call emulation with ability to mimic user behavior. MockTell uses data generated from real world calls for call emulation that helps in optimizing and evaluating the performance of IVR applications. MockTell also allows simulation of calls to provide further testing of system.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation, Human Factors

Keywords

emulators, IVR, voice application, testing

1. INTRODUCTION

An Interactive Voice Response (IVR) system provides information and service through telephones and is used for automating the call handling in several commercial organizations. Many critical applications runs on IVR systems, so it is essential to have well tested IVR applications with all the necessary optimization done by the IVR developer.
IVR testing can be classified into infrastructure, system, and application level testing. At present, various industrial tools[1] are available to test the performance of IVR applications at infrastructure (voice gateway, routers) and system (Telephony servers, database servers) level, which helps to maintain the quality. However, application level tests like program flow test and error condition test require giving different user inputs to application under test. This requires testing tool to have the ability to mimic user behavior. Currently available testing tools lack the ability to mimic user behavior, many application level tests are usually done manually or through customized test scripts written by IVR developers. Moreover, IVR systems are evolving [1, 4, 2] and it will become increasingly difficult to test and optimize future IVR applications. MockTell addresses these challenges by mimicking user behavior in different contexts.

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

ICPE'13, April 21–24, 2013, Prague, Czech Republic.
ACM 978-1-4503-1636-1/13/04.

2. DESIGN AND FEATURES

MockTell has five functional components and two user models:
Logic Processor: This module co-ordinates with other modules and takes various decisions for emulating a call e.g. call initiation, responding to voice prompt etc. Two user models for emulation process are embedded into this and can be modified to emulate different user behavior for scenarios of different complexities.
Visualizer: This dedicated module handles graphical user interface of the MockTell. Tasks like taking user input for configuring different parameters of emulation process, showing current status of emulation etc. are done by this module.
SIP utility: This module provides API for initiating and releasing a call, generating a DTMF key-press and sending the audio file as speech utterance for communication with IVR under test.
Status Monitor: This module gathers the information about present state of IVR by communicating with IP-PBX software hosting the IVR application. This information helps MockTell to adjust the timing and value of events to be regenerated in testing dynamic IVR systems.
Data Handler: This module load and read the logs of IVR usage into a Java based object representing user model for emulation. Each object contains information about responses made by a user and their corresponding contextual information in a call. Logic Processor uses the information stored in these objects for emulation.
In MockTell, calls are emulated using two modes: emulation using previous call records and testing using random DTMF.

2.1 Emulation using previous call records

In this mode, MockTell reads the call logs, stored in XML files, which may be obtained from a real world deployment. MockTell supports two formats that capture user models:
Simple user emulation: This user emulation works for currently available IVR systems. MockTell replicates the call events like key-press and speech recording, as they happened in corresponding actual call. 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: With intricate user emulation model, MockTell can emulate calls for upcoming IVR systems where menu options sequence may change in order to give better services. Each menu option is a state which is assumed to be announced by IVR

application over the IPPBX console as they occur. In the current implementation, MockTell connects to command line interface of FreeSWITCH using fs_cli utility which comes with pre-installed FreeSWITCH binaries to capture the state information announced by IVR application. MockTell generates user responses to these states based on the data stored in the user model.
In both of the modes, MockTell can reorder the call sequence which changes the number of past calls and past system usage for a particular call. This helps in studying IVR system which adapts after every call based on the system usage. MockTell can also vary number of telephone lines to IVR application. With this feature, the number of telephone line connections can be increased to analyze the behavior of IVR application in handling multiple connections (e.g. accessing the same audio file for reading or writing the log).

2.2 Emulation using Random DTMF

In this mode, MockTell performs tests that are independent of user models and can be used to test IVR system's parameters. It generates events like key-press or speech utterance based on test parameters specified by the user. It supports 3 types of IVR testing.
Call Load Test: This test helps in measuring the number of simultaneous calls an IVR application can process and reports the integer value at which IVR application crashed. In our test, we found that FreeSWITCH was able to process 123 simultaneous calls for our sample IVR application. The 124^{th} call failed because of too many connections open to the database (MySQL).
Sequence Test: In this test, MockTell generates random DTMF digits, each with a constant time delay between each generated digit and reports the sequence test case, if any, at which IVR application fails to respond. An IVR developer needs to specify the time delay and number of digits in the sequence (e.g. 4 digits). In our test, MockTell was able to check 100 sequence in 2,504 seconds of sequence length 5 and delay of 5 seconds. Initial 4 seconds out of 2,504 were taken by MockTell in setting up the call.
Sequence test with random delay: This test helps to detect erroneous DTMF input in more complex manner i.e. when the duration of played voice prompts varies for the announcement of menu options. This test generates random DTMF with different time gaps between each successive generated DTMF. The time gap is randomly decided in the range of t_1 and t_2 values specified by the tester. Ideally t_1 should correspond to the length of minimum voice prompt and t_2 be the length of maximum voice prompt in IVR.

3. PERFORMANCE EVALUATION

In this section, we evaluate the performance of MockTell through an experiment conducted using *call load Test* feature. For this experiment, we setup an IVR application written in JAVA and hosted on FreeSwitch. The FreeSwitch and MockTell were running on two different machines referred as Machine-I (HP Compaq dx7400: Ubuntu 10.04, Intel core 2 Duo, 3GB DDR2) and Machine-II (HP Probook 4520s: Ubuntu 12.04, Intel Core i5, 2GB DDR3) respectively, with IP connectivity (100Mbps) between them. We started *call load* test feature of the MockTell. It initiated a call every 2 seconds while keeping previously initiated calls alive till the end of experiment or terminated by FreeSwitch. A call in this experiment was always in one of the 3 states: active, dropped and timed-out.
Active State: refers to a call that is connected to FreeSwitch and is not being terminated.
Dropped State: refers to a call that was previously connected but terminated by FreeSwitch.
Timed-out: state refers to a call released by MockTell as it was not able to connect to FreeSwitch.
In total, we initiated 1024 real calls through MockTell. Initially, we did not observe dropped calls till 233^{rd} call as the load was low on Machine-I. But after this, FreeSwitch started dropping calls at a higher rate which reduces the count of active calls and stabilizes around 145 calls that shows that the rate of dropping the calls becomes equal to rate of accepting the new calls. Our results for number of active calls (i.e. concurrent calls) for FreeSwitch running on hardware having configuration of Machine-I are conforming the results obtained by other developers as available on FreeSwitch website[2]. We measured the CPU and Memory load of Machine-I and found that it was saturated around 233^{rd} call because rate of call dropping became nearly equal to rate of accepting new call. We also found that 3 calls were timed-out at 110^{th} call because of network congestion between the two machines.
MockTell Load: We measured and categorized the CPU and memory usage of MockTell (running on Machine-II) 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. We found that static load gradually increased 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 1024 real 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).

4. CONCLUSION

We presented MockTell with two user models for testing IVR applications. Simple user model allows testing of currently available static IVR while Intricate user model enables testing of upcoming dynamic IVR applications. User model based testing provides characteristic to model different users easily [3].
User model based approach in MockTell helps in reducing manual investigation of IVR applications. The approach presented in this paper is independent of any voice or speech processing which eventually helps in avoiding any semantic processing as well. Experiences with trial run of MockTell for testing our basic IVR application suggests that apart from application level issues it can also help developers to find some system level issues like too many connections opened to MySQL.

5. ACKNOWLEDGMENTS

We acknowledge the support of Tata Consultancy Services (TCS) for this research.

6. REFERENCES

[1] 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).

[2] Asthana, S., Singh, P., and Singh, A. A case for adaptive interface for interactive voice response system. *ACM DEV'13: Proceedings of the 3rd ACM Symposium on Computing for Development* (2013).

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

[4] 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.

[2] `http://wiki.freeswitch.org/wiki/Real-world_results`