An Innovative System for Remote and Automated Testing of Mobile Phone Applications

Karthikeyan Balaji Dhanapal, K Sai Deepak, Saurabh Sharma,
Sagar Prakash Joglekar*, Aditya Narang†, Aditya Vashistha‡, Paras Salunkhe§,
Harikrishna G. N. Rai, Arun Agrahara Somasundara‖ and Sanjoy Paul¶
Infosys Labs, Infosys Ltd.
44, Electronics City, Bangalore, India 560100
Ph: +91-80-28520261
Email: {Karthikeyan_Dhanapal, KrishnamurtySai_D, Saurabh_Sharma30,
, †, ‡, §,
Harikrishna_Rai, ‖, ¶}@infosys.com

***Abstract*—Application testing is an integral part of the application life cycle. This testing effort is more for the 3^{rd} party applications in the mobile phone market, due to the wide number of handsets available on which the application needs to be tested before being released. At the same time, majority of the applications use the cellular network, necessitating the tester[1] (along with the handset) to be present in the service area of the cellular network. We present a remote testing system, wherein the handset is in the cellular network service area, but the tester is present in a remote location. The tester controls the handset over the Internet. This gives opportunities to leverage the potential of the global outsourcing business model in mobile application testing domain. In addition, the system is agnostic to the Operating System & application running on the mobile phone, and is also non-intrusive. Further, we present preliminary results on automating this remote testing process itself.**

I. Introduction

The use of mobile phones is increasing at a very high rate. More importantly, the mobile phones are being used for more things than just making phone calls. A plethora of applications are being built for mobiles. These may be from the manufacturer, service-provider or third-party. The testing effort is more for the mobile applications than the PCs due to the wide number of handsets in the market. This testing effort can be broadly classified into 3 dimensions: *Functional testing*, which is testing the application itself on the mobile handset, when it is the only one interacting with the application server over the wireless (cellular, WiFi etc.) network; *Network testing*, which is testing the application when subjected to varying network conditions, as it would encounter when deployed and used by the end users in the multiple access wireless medium; *Performance testing*, which is testing the application server, to make sure that it can service simultaneous mobile clients.

This paper talks about the first of these: functional testing. The conventional way of doing application testing on mobile phones is to load the application on the mobile, and give it to a human tester. The tester does the various operations as given in the test case, and verifies if the output matches the expected output as given in the test case.

An important requirement is that the phone should be present in the location of the service provider network. It cannot be tested in a place where the network is absent. This restriction means that the global outsourcing business model cannot be directly applied. For instance, the manpower is expensive in US, and so many companies outsource their activities to countries such as India, where manpower is relatively cheaper. In the context of testing, as the networks available in US are not available in India (some cellular providers may have tie-ups for international roaming, but is not used for various reasons), the testing activities cannot be outsourced, and the test engineer needs to be present in US. DeviceAnywhere [1], Perfectomobile [2] have tried to solve this issue by making the phone accessible over the Internet. Now the testing task can be outsourced, and the test engineer sitting in off-shore location (e.g. India) can do the testing. But one drawback in these approaches is that they are intrusive, and some kind of instrumenting is required: either in hardware, or loading some test software into it.

In this paper, we propose a novel solution to the same problem of remotely being able to test mobile phone applications. This approach is non-intrusive, and does not require any changes to be done to the mobile phone itself. Any testing process fundamentally involves 2 steps: giving the input & validating the output. In our system the input given by the human tester is replaced by a 3-axis robot which does the pressing of the keys. The output is captured by a camera.

*This author is currently at Citrix Online (sagar-joglekar@umail.ucsb.edu)
†This author is currently at SPJIMR (pgp11.adityan@spjimr.org)
‡This author is currently at Microsoft Research (t-avash@microsoft.com)
§This author was a summer intern (pmsalunkhe@gmail.com)
‖arun@ee.ucla.edu
¶This author is currently at Accenture (sanjoy.paul@accenture.com)
[1]The words tester and test engineer are used synonymously

This robot-camera system is placed in the location where the phone network is available (say, US), and controlled from a remote location (say, India).

In addition to the remote testing, one can envision a tool for automating this remote testing process itself. Here, instead of the remote tester, the testing is driven by a software script, and the human verification is replaced by intelligent image processing. The key requirement here is the non-reliance on the co-ordinates of the objects on the screen. Otherwise, the script would have to be re-written for any changes in the location of the objects. We will present few results on how we achieve this. Also, our technique allows the reuse of the same script for different handsets.

The paper is organized as follows. We discuss the related systems in Section II. We outline the design of the system in Section III, followed by the implementation details in Section IV. We describe the design of the automated system in Section V and implementation in Section VI. There are lot of ways this basic system can be extended. We present some of these ideas in Section VII, finally concluding in Section VIII.

II. Related Work

A. DeviceAnywhere

DeviceAnywhere [1], originally known as Mobile Complete offers the service of providing remote access to mobile handsets. The real handsets are placed in many countries with access to the real network in those respective countries. They can be accessed from anywhere in the world over the Internet. It is not very clear how the device is accessed though. It may be either a software or hardware intrusion. The access comes with an hourly subscription fee. The subscriber gets a handset GUI on his computer, and can pretty much do all operations what he could have done with the real device in hand. These operations are reflected on the actual device. The output is communicated back to the GUI on the tester's computer. A premium subscription offers one to capture the testing steps, and replay it as a script.

B. Perfectomobile

Perfectomobile [2] is another player in this area of remote testing. The handset is connected to the Perfectomobile system using data cable. The system is made accessible over the Internet. A remote user accesses this site, wherein he is shown a handset GUI. When he presses a key on this GUI, it is sent over the Internet to the Perfectomobile system, which in turn communicates the key information to the handset over its data connection. Our guess (as the internals are not published) is that the handset needs to be instrumented with some software which receives this command, and actuates it in the handset. There is a video camera which records the screen output of the handset (and possibly communicates it to the remote user).

It has two offerings: web access to real devices, and automated testing. These are very similar in terms of functionality to the DeviceAnywhere remote access and script execution.

C. Other Similar Systems

DeviceAnywhere & Perfectomobile allow *remote* testing of mobile phone applications on *real devices* and *real networks*. There are many other systems which provide 1 or 2 of these 3 features: remote, real device, real network. We will not go into the details of these, as it is outside the context of the current paper.

As can be seen, some form of dissection happens on the phone (either in hardware or software) in the above two approaches. Our system operates in a non-intrusive way as will be elucidated in the following sections.

III. System Design: Manual Remote Testing

This section presents the design of the remote testing system. As mentioned in Section I, any testing system fundamentally consists of a subsystem to give input, and another subsystem to collect and verify the output. Here, both the input and output subsystems are at the location of the mobile phone/network. The tester is in a remote location. He controls the input and output subsystems from his location. We term this as tester subsystem.

The system is shown in Fig. 1. The tester subsystem should allow the tester to give inputs as if he is pressing the actual keys of a mobile keypad. It should also show him the output as if he is seeing the actual mobile screen. These two remote parts interact with their physical counterparts over the Internet.

The input subsystem gets input from the tester subsystem over the Internet, and actuates the key presses. The input consists of key code and some metadata.

The output subsystem gets the output from the screen, and sends it over the Internet to the tester subsystem. The output is basically streaming images of the screen.

IV. System Implementation: Manual Remote Testing

This section presents the implementation details of the system. Fig. 2 shows the system.

A. Input Mechanism

The input mechanism to press the keys of the mobile consists of a 3-axis robotic system. The system is composed of the following units:

1) 3 motors for the X, Y, Z-axes movements. X & Y axes have servo motors, and Z has stepper motor.
2) Plunger driven by the Z-axis motor with adjustable tip to do the pressing function. Different tips are required for regular keypads and touch phones. For instance, we need metallic tip for some touch phones which operate on capacitive touch.

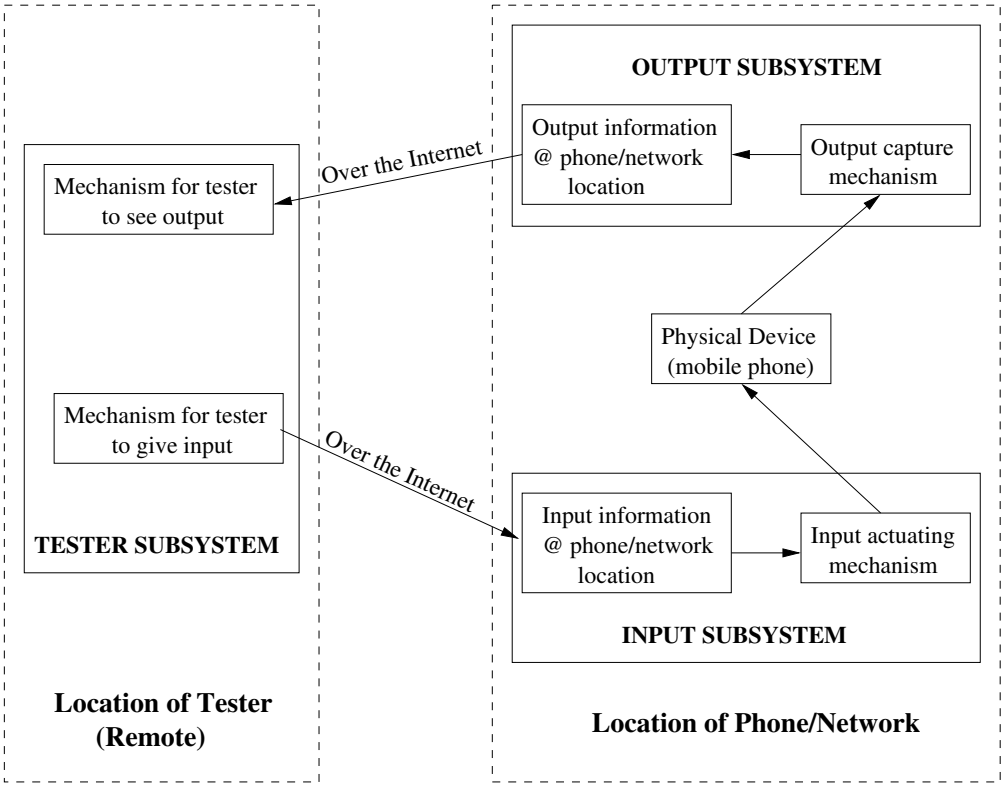


Figure 1: System design for the Remote Testing System

3) A platform for fixing/gluing the phone(s). This is the black unit in Fig. 2.
4) A Programmable Logic Controller (PLC) having the 3 motor controllers, and providing the interface to the PC.
5) Limit switches which get activated when the motor reaches the ends of the platform. These switches change a variable in the PLC, which can in-turn be read by the PC.
6) A PC controlling the whole system.

The PLC has 2 interfaces to the PC: USB and UART. The PLC is programmed using ladder logic. The code is compiled on the PC and transferred to the PLC using USB/UART. The servo motors, controllers, and PLC are from Omron [3], and the stepper motor is from Autonics [4]. Omron also provides an IDE programmer to create the ladder logic program. The program is similar to an embedded systems program, which executes the content in an infinite loop.

In addition to programming and loading the code in PLC, one can do some operations on the PLC during run-time. Example of such an operation is Read/Write of various memory areas. This is achieved by sending appropriately formatted *command*s from the PC to the PLC over the UART. For each command, there is a *response* from the PLC.

Next, we will describe briefly the program running in the PC, in the PLC, and also the run-time interaction. For the mobile placed on the platform, on which the application is to be tested, a one-time measurement is done to find the locations of the various keys on the handset. For this, one of the keys (say, the "#" key at the bottom-right of most keypads) is designated as (0, 0). Location of other keys is measured with respect to this key using a ruler. The translation of relative to absolute location is given in Section IV-D. This table is stored in the PC program. For any key to be pressed, this key mapping table is looked-up, and the appropriate co-ordinates is written from the PC to the PLC memory. It may be mentioned here that the PLC output is in terms of number of pulses. For the servo motors, each pulse output moves the load by $0.033mm$.

The PLC has 2 routines for each of the 3 motors: *init*, and *act*. When the system starts, the *init* routine moves all the motors (and correspondingly their loads) to their respective home positions. Home position is defined as the location where the motor load is at the beginning (or the end, depending on the direction sense) location, as indicated by the limit switches. After this, whenever there is a trigger from the PC, the *act* routine does the following:

- *move*s the X & Y direction motors (and correspondingly their loads) to the desired location. The location is written in the PLC memory by the program on the

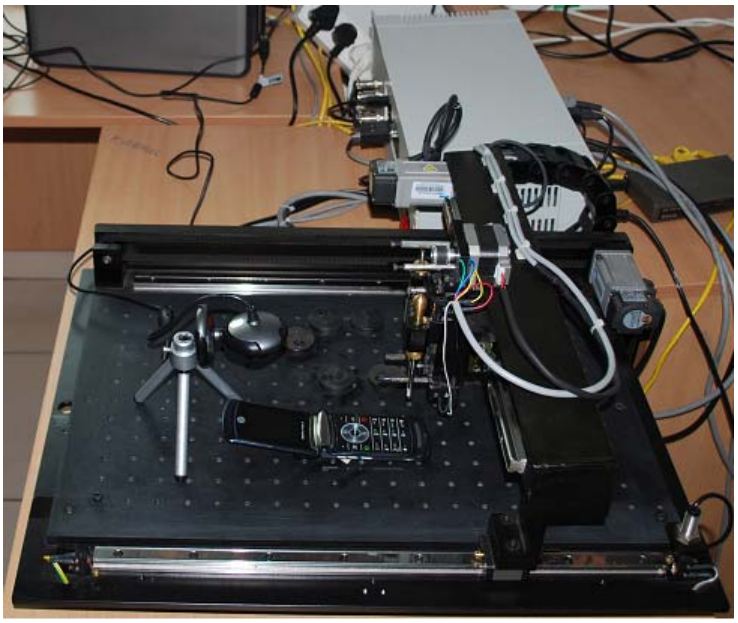


Figure 2: Image of the System

PC as mentioned above.

- Once the destination is reached, the Z-arm either does a *press*, possibly multiple times, or a *hold*. In the *hold* operation, the key is held in pressed location before the Z-arm starts moving up.

For example, to get the character "b" in non-QWERTY keypads in Text Message mode, we need to press the 2^{nd} key ($abc|2$) twice, and to get "c", press it thrice. An example of *hold* is, if we hold same key ($abc|2$) in Text Message mode, it will become "2".

With the above pieces in place, the only missing link is the interaction between the PC and the PLC, i.e. the driver for the PLC. For this we send commands over the UART from the program in PC to PLC. The commands used are Read/Write memory locations. We write the PLC memory for following:

- Triggers for *init* and *act*. These are memory locations being continuously read by the PLC program.
- The X & Y locations.
- Information about the Z-arm action (*press* (# times), *hold*).

We read the PLC memory from the PC to know if the *init* or *act* operation has completed. The PLC updates a flag (basically a memory location) when the operation is completed. The PC is polling this PLC memory location continuously.

Finally, this subsystem contains the necessary logic for the PC to get the information about the key to be pressed from the remote tester subsystem over the Internet. Socket based network programming is used for this communication.

B. Output Mechanism

The output is captured through a webcam. We use the iBall C12.0 Face2Face [5]. The webcam is connected to a PC, which takes 2 frames per second, crops the region of interest, and sends it over the Internet to the remote tester subsystem. When the system starts, the system does not know the region of interest. So, the first frame the webcam captures is presented to the user[2] who selects a rectangle (opposite points), which encapsulates the region of interest. This information is stored, and is used for cropping all subsequent frames. The webcam can be seen clearly in Fig. 2.

This subsystem also contains the necessary logic to send the captured frames over the Internet to the remote tester subsystem. Socket based network programming is used for this communication.

C. Tester

The tester subsystem, which is at a remote location needs a way to interact with the 2 physical subsystems mentioned above. We basically need a GUI, using which the tester can give inputs and see the output. This is composed of two separate programs. One of them is the display program which connects to the physical output subsystem over the Internet, and displays the streaming images of the actual screen of the mobile under test.

The other program is for giving inputs to the physical input subsystem over the Internet. This has two modes of operation: *learn* mode and *test* mode. In the *test* mode, the

[2]The person at the location of the phone

image of the mobile handset (only keypad part) under test is displayed on the tester screen. He can now click using the mouse on the keys and the appropriate key code and *hold* flag will be transmitted over the Internet to the physical input subsystem. For this to happen, we need the pixel-to-key mapping for all the keys, given a handset image. This is a 1-time setup for each mobile handset, and is accomplished using the *learn* mode. In this mode, the image of the handset (only keypad part) is displayed. The tester needs to mark a rectangle (say, surrounding a key), and enter the key code for that. This is repeated for all the keys. This information (opposite co-ordinates of the rectangle) is stored, and used in the *test* mode to retrieve the appropriate key when mouse is clicked. It may be noted that the same (static) image of the handset keypad is used in the 2 modes.

The detection of *hold* operation happens at the tester end. For this, we see the difference between the key-release and key-press times. If it is more than a certain threshold, it is detected as *hold*. This information is sent along with the key code. For detecting multiple press (the time difference between pair-wise successive presses is less than a certain threshold), we need the time of the press (which happens at the tester end) at the input subsystem end (where we have current time). To avoid time synchronization issues, we timestamp the packet containing key code at reception. This time is used for detecting the multiple press scenarios.

To summarize, the tester subsystem has 2 programs:

1) Receive the streaming images over the Internet from the physical output subsystem, and display it on the screen.
2) Capture the key-press, key-release events, and send appropriate key code and *hold* flag over the Internet to the physical input subsystem.

D. System operation

We now present a step-by-step walk through of the system in operation.

(1) Place the mobile on the platform, so that it does not move. We use a double-sided tape for this.

(2) Start the 1^{st} program of previous section (display program) at the remote tester subsystem.

(3) Start the physical output subsystem program on the PC connected to the webcam. For the first frame, the user is asked to select region of interest. Once this input is given, socket level connection is established with the remote tester machine (above step). There is a live streaming of frames, which the tester can see on his screen.

(4) Move physically the robot system, so that the Z-arm of the system is on the "#" key.

(5) Power on the PLC and start the physical input subsystem program on the PC connected to the PLC. The *init* program in the PLC drives the motors to their home positions. Also, for the X & Y motors note the amount moved to reach the home position. This is the offset required to convert the relative positions of the keys mentioned in Section IV-A to absolute co-ordinates.

(6) Start the second program of the previous section (capture and send key code) at the remote tester subsystem. Now the tester can give any input using the mouse. The key information is captured from the mouse click and is conveyed to the physical input subsystem. Depending on the kind of press (single, multiple, *hold*), the motors are commanded to do the action. There is a sufficiently big input queue in the physical input subsystem which receives the key-press information from the remote tester over the Internet. So, the tester can continuously give inputs (till he actually needs to check the output), instead of waiting for each key press to complete.

It may be mentioned here that some amount of co-ordination is required initially between the user handling the physical subsystems (who does the steps 1, 3, 4, 5 above) and the tester at the remote location (who performs 2, 6). This can be achieved over the phone or some instant messaging software. The important point to note here is that the role of the user at the physical system/phone location is over after the brief initial setup, and the remote tester manages the system from then on.

In the next 2 subsections, we will talk about the enhancements to the above system to support touch phones, and the support for testing multiple phones simultaneously in a multi-tasked manner.

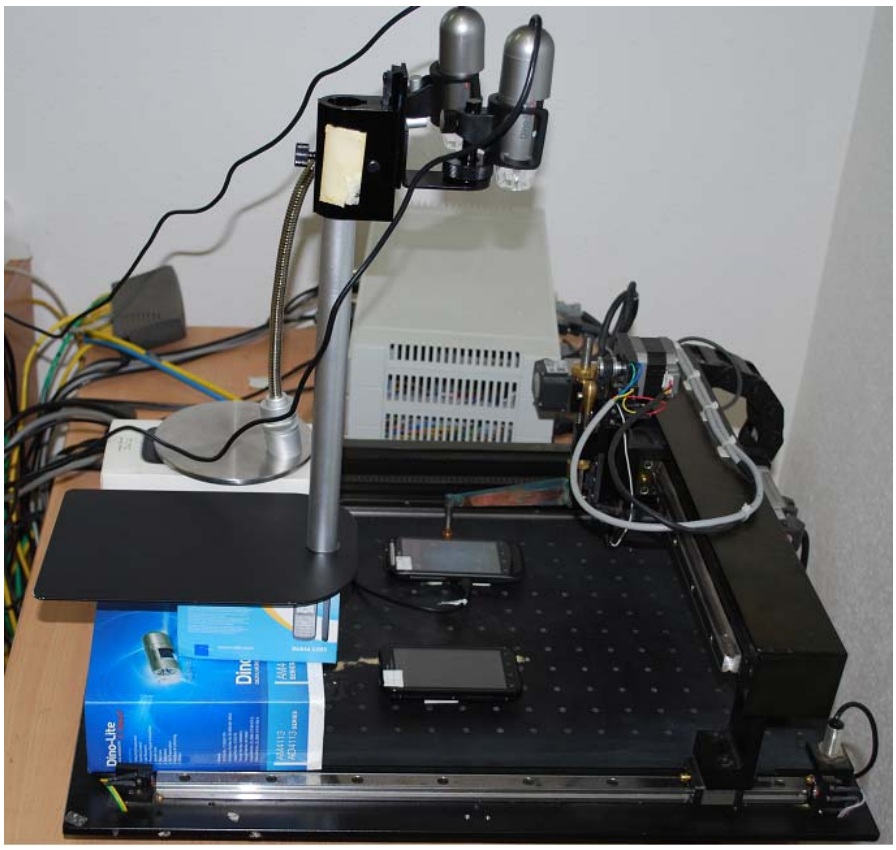
E. Touch Phone Support

The previous sections talked about the phones based on a keypad (numeric or QWERTY). But in touch phones, although the keypad does appear at a fixed location, many operations don't use the keypad. As a result, we need a different input mechanism at the tester end. The mechanism can also be used for the keypad based phones, removing the need for the *learn* phase mentioned earlier. For this, instead of having 2 programs running at the tester end, we have a single program with threads, which

- receives the streaming images
- displays them
- detects the inputs (mouse clicks) on the same image
- sends the co-ordinates

The main challenge is mapping the pixel co-ordinates on the image (on which the tester clicks) to the corresponding physical co-ordinates. For this, we do a calibration by placing a marker with 2 dots $1\ cm$ apart on the phone. The 1^{st} dot is taken as the origin, and the Z-arm is placed over it in the beginning (similar to placing the Z-arm over the "#" key). We start the 3 subsystems as earlier (input, output, tester). The tester is now seeing a streaming video of the ROI (only the screen, which is the Region Of Interest) of the phone under test. He needs to now click on the 2 dots on this image. This serves 2 purposes:

(a) A screen-shot at the remote tester end of the touch phone under test



(b) System with multiple mobiles under test

Figure 3: Support for Touch phone and multiple phones

1) Clicking on the 1^{st} dot synchronizes the origin of the image and the robot.
2) Clicking on the 2^{nd} dot calculates the mapping from physical space to pixel space. We already know the physical distance between the 2 dots ($1\ cm$). The clicking on the 2 dots gets the pixel distance between them.

Now the tester can click on any point on the image. This is translated to the robot motion in the following steps:

1) The mouse click returns the pixel co-ordinates with respect to the origin of the image in the pixel space, which is top-left corner of the image. We translate it, so that it becomes with respect to the 1^{st} dot mentioned above. For instance, if the 1^{st} dot is at $(10, 10)$, and the point clicked is at $(100, 100)$. We get the resulting co-ordinate as $(90, 90)$.
2) We convert this to the physical space using the pixel-to-physical mapping found in the 2^{nd} step above.
3) This co-ordinate is sent over the Internet to the computer attached to the robot.
4) This is converted to the pulse space. As mentioned earlier, the motors of the robot operate in the pulse space. This pulse co-ordinate is with reference to the 1^{st} dot where the Z-arm was placed initially.
5) Finally, this is translated to the absolute position (with reference to the home position) in pulse space for the robot, and the robot is commanded to move to this position.

In addition to the click, we also have the scroll facility. For this the tester clicks on a point, drags the mouse to another point, and then releases the mouse.

We would like to mention that we had to make 2 hardware changes for touch phone support. Firstly, we introduced a $L - joint$ to the Z-axis plunger, so that the screen seen is not occluded at the remote tester GUI. Also, instead of the webcam, we use a Microscope camera AM413TL-M40 from Dinolite [6]. This gives a better resolution and a larger field of view. A screen-shot of what the remote tester sees is shown in Fig. 3a. The marker with 2 dots and the $L-joint$ of the robot Z-axis are seen clearly in the image.

It may be noted that there is fundamental difference between the 2 approaches (keypad and touch phones). In keypad phones, we transmit the key, which requires 2 mappings. First, at the remote tester end, the location of

click in converted into keycode, by seeing the enclosing rectangle where this clicked point lies (and its corresponding keycode). Secondly, at the input subsystem robot end, this keycode is converted back to the physical co-ordinates (with reference to an origin ("#" key)), using another lookup table. Whereas, in touch phones, we directly obtain the physical point from the pixel point using the unified origin (1^{st} dot of the marker), and the pixel-physical mapping.

F. Support for testing Multiple mobiles

When testing mobile phone applications, it is very likely that the tester is waiting for inputs from the server, which the mobile phone is connected to over the cellular network. This is more so for media applications where the tester is waiting for the content to download/render. During such intervals, the robot of input subsystem is idle. This gives an opportunity to have another mobile and a camera on the platform. Now a second remote tester can independently test another application on this second mobile.

We have built this support in the system. Fig. 3b shows the image of the total hardware with 2 cameras and 2 mobiles under test. One can see the microscope cameras and the $L-joint$ also in this figure. The programs at the remote tester send their identity along with the co-ordinates to the robot, which queues them. Currently it executes these in a first-come-first-serve fashion. A better scheduling should take into account the nature of application, and the time of arrival at the queue. This is an area of further research.

The main technical challenge was, synchronizing the co-ordinates between the physical phones and the images the remote tester sees. We solve this by having an *initialization* phase. Here, we place the Z-arm of the robot over the 1^{st} dot of the marker (glued on the mobile), and note down the distance it takes to reach the home position. This is repeated for each mobile on the platform. This information (distance (X, Y) to home position from each mobile's 1^{st} dot) is stored in a text file. When the *operation* phase starts, the robot can start from any location, and moves to its home position. We now read from the text file the relative origins for each mobile. Each remote tester does the same operation (of clicking on the 2 dots) as mentioned earlier. The input request (in form of co-ordinate locations) from the tester end is enhanced to include the identity of the tester. The computer attached to the robot calculates the final position with respect to the home position using the relative origin of the particular mobile which this tester is testing.

V. System Design: Automated Testing

The manual approach of remote testing does not lead to much improvement in testing efficiency in comparison to conventional manual testing. Rather, scalability is still an issue with remote testing, as the number of testers required increases linearly with increase in testing requirements. This will subsequently result in higher testing budget in application development/maintenance lifecycle. One solution to scalability problem is to use multiple testers simultaneously in a time shared fashion, as discussed in Section IV-F. But the efficacy of this approach relies on the nature of application being input non-intensive.

Another drawback of manual approach is the effect of fatigue factor in humans. Unlike automated testing, a human tester performing repetitive tasks is more likely to commit errors due to fatigue. In this and the next section, we present another solution.

We can improve the manual remote testing system to do automated testing, without any human in the loop. All we need is an intelligent system which can drive and verify the testing, similar to how human does it. Here the test script, test manager and image processing module replace the remote human tester. Fig. 4 shows the system. The Test Manager employs

- A test script that defines the sequence of test instructions in the form of actionable steps.
- A robotic system for physical actuation of actionable steps on the mobile device.
- A camera for output capture.
- An image processing module to identify elements of interest in the mobile application screen.

It may be mentioned that the middle 2 components are similar to the earlier system. Also, the test manager, test script and image processing module can be either on-site, or at the remote location.

It should be noted that the test manager needs to interact with both the input subsystem and the image processing module. This is because, ***when*** to give the input to the input subsystem depends on the current state, which needs to be determined by the image processing module. For instance, assume the test script says to enter the login and password, and then click on an icon (in the successfully logged-in page). Now the clicking input needs to happen after the login/password has been verified by the server, and the next screen load is completed. To know this, the image processing module needs to detect when the progress bar (hour glass etc.) has completed (disappeared).

VI. System Implementation: Automated Testing

In this section, the sub-systems of automated testing approach are demonstrated for testing the user authentication scenario on a mobile application (Skype[7]). In this test scenario, the testing system is required to specify user credentials for authentication and verify if the mobile application performs the required operations as per test specification. We would be mentioning the automation specific details only in this section. The calibration, actuation etc. are similar to the manual remote system, and details will not be repeated here.

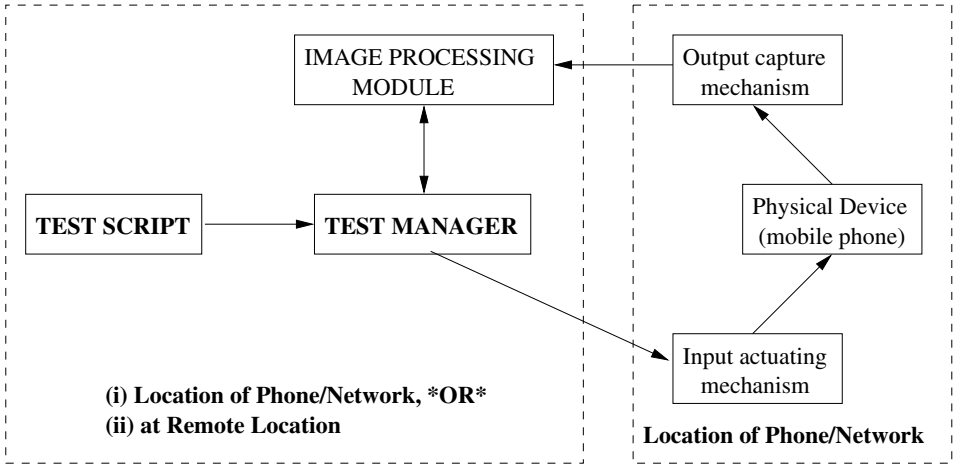


Figure 4: System design for the Automated Testing System

A. Test Script

A test script is composed of simple execution steps compiled by the tester once for every test case. Once compiled, this test script can be reused for testing the same application on several mobile devices. Each execution step in the test script is composed of 3 fields: Action, Screen Element and Parameters.

For the authentication scenario, 3 different Actions, namely, *Find*, *Press* and *Wait* can be performed. *Find* is used to identify the location of specific elements in the mobile application screen. *Press* triggers the input actuating mechanism to press specific actionable elements and *Wait* commands the automated testing system to stop all actions until a condition is satisfied.

Screen Elements that can be triggered for executing the authentication scenario are: text, text box, button, icon and keyboard.

Various Parameters are used for each execution step depending on the type of Action performed. Parameters are supplied in the form of alphanumeric characters either representing literal text or an icon in the application screen. The sample test script for authentication scenario in Skype application is shown in Table I.

Table I: Sample Test Script for Skype Authentication

Action	Screen Element	Parameters
FIND	TEXTBOX	"Skype"
PRESS	KEYBOARD	"$< username >$"
FIND	TEXTBOX	"password"
PRESS	KEYBOARD	"$< password >$"
PRESS	BUTTON	"Sign in"
WAIT	ICON	"Signing in"
FIND	TEXT	"all contacts"

The test script is executed to first identify the location of textbox corresponding to user name in Skype as shown in Fig. 5(a). Parameter used for this task is the text identifier that appears in the vicinity of the textbox ("Skype"). Once location of the textbox is identified, required credentials should be entered by pressing the device keyboard using Press action. Keyboard on the mobile device is calibrated once for each device type, where the physical location of every alphanumeric and special character on the keyboard is stored for reference. This is similar to the mechanism of storing key co-ordinates mentioned earlier in Section IV-A in the context of keypad based phones. These stored locations are used every time a test case uses Press action on the Keyboard. Same process is repeated for password entry also.

After the credentials are entered, a button on the screen is used for signing in on the application. Location of button on the screen is identified and Press action is performed to initiate "sign in"operation as can be seen in Fig. 5(f). While authentication is in progress, the testing system is required to wait for completion of the process. Authentication test case on Skype application is determined to be successful if the text "all contacts" is found in the next application screen.

B. Test Manager

The Test Manager handles the execution of test script via modules for actuating input, capturing output and image processing.

- Actuating Input: This module controls the execution of commands on the robot system by sending the co-ordinates of the location(s) to be pressed, to the PC attached to the robot.
- Capturing Output: This module gets the streaming images of the application screen from a high resolution camera (or from the PC attached to the camera). These images are used by the image processing module to find icons and text in the current application screen as defined in the parameters of test script.
- Image Processing: Automated/Semi-automated execution of test cases using the proposed testing system

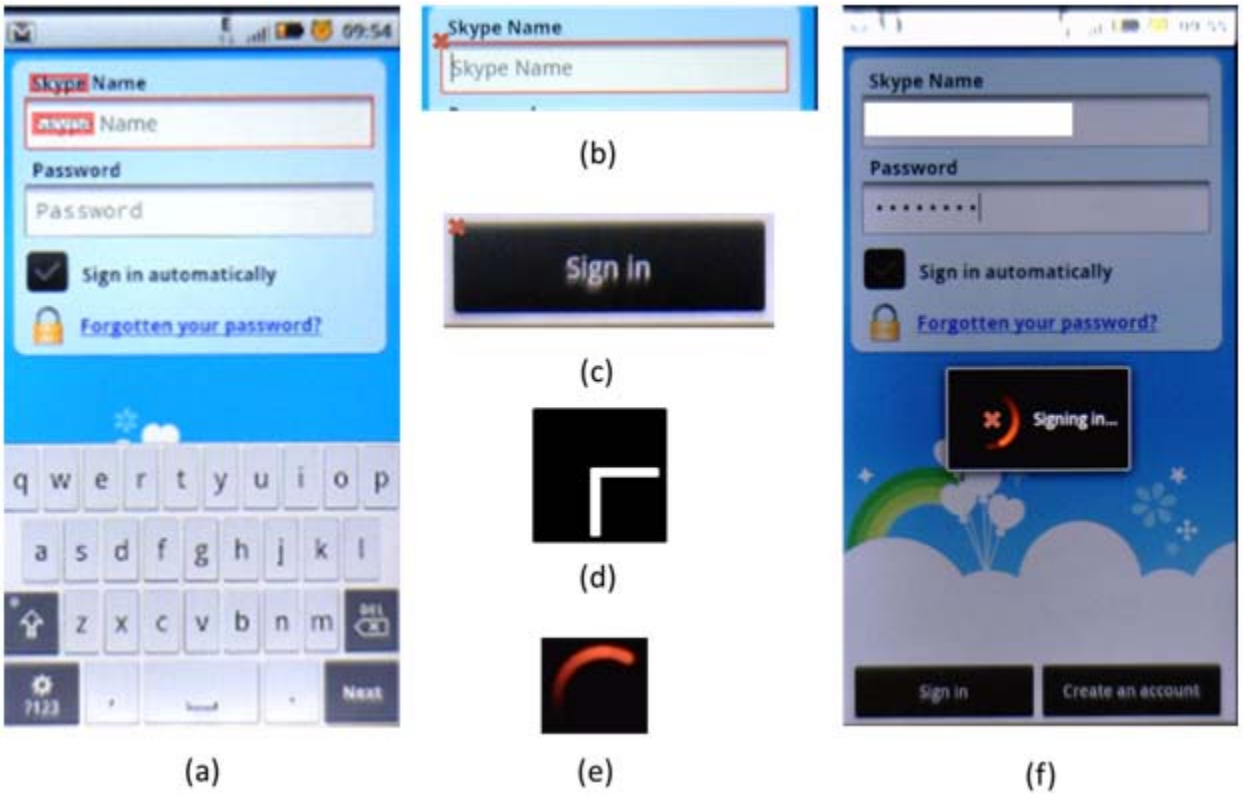


Figure 5: Skype: Login Authentication testing using automated approach

requires detection and recognition of text and actionable icons on the application screen. Image processing techniques provide essential tools to perform these operations efficiently. The steps used within this module are explained in detail next.

C. Image Processing Module

Text processing from images of application screen is performed using a commercially available Optical Character Recognition (OCR) tool, IRIS [8]. We have developed various feature detection techniques in images to detect the required icons present in Skype application.

Textbox and button are detected using a combination of OCR and feature detection methods in images. Both textbox and buttons typically have corresponding text identifiers in their vicinity on the application. OCR is first performed to identify the location of these text identifiers for both. In the authentication scenario, we use the text "Skype" and "Password" to detect textboxes corresponding to user name and password respectively (see Fig. 5(a)). "Sign in" is used in case of button detection (see Fig. 5(c)). Once the location of the relevant text identifier is found, the corresponding icons (text box/button) are detected in the neighborhood of this text region. In some cases, where multiple instances of the same text identifier are present in the application, the correct one is selected by a tester manually.

The wait for loading action is performed by ensuring the presence of the corresponding icon (red circular icon in Fig. 5(f)). If the presence of icon is confirmed then the testing system waits until the icon disappears from the application screen. We will discuss each of these in detail in the following subsections.

1) Optical Character Recognition (OCR): Commercial OCR tools are known to perform well while processing document images. Applying these tools on user interfaces of mobile applications (i.e. screen of mobile device) is relatively complex. This is essentially due to the variations observed in the application background as well as text. Text could have varying levels of contrast and intensities with respect to the background, resulting in inconsistent text recognition. We handle this using a multi-pass approach for OCR. In each pass, the image is pre-processed before OCR with a combination of following operations: smoothing, inversion and adaptive thresholding [9]. The OCR results from each pass are then consolidated to generate a single OCR output for the current screen. Text consolidation is performed by identifying similar text at the same localized text regions from results of all passes. An example of text detection for Skype using OCR is shown in Fig. 5(a). All the regions in the image containing the word "Skype" are highlighted after multi-pass OCR. The multi-pass approach performs significantly well for text detection and recognition

in mobile applications.

2) Textbox Detection: Textbox in mobile applications typically appear as bright rectangular structures. Size of these rectangular structures can vary based on device screen size or the camera resolution. Therefore textbox detection technique should be robust within a range of scale. Another challenge observed in this problem is that the textbox needs to be detected irrespective of the presence or absence of text string within them (see Fig. 5(a)). We handle these challenges by performing corner detection instead of detecting the entire rectangular structure. A neighborhood around the text identifier is selected. Size of the neighborhood is adapted to the screen size and camera resolution. The left-top corner of textbox is detected in the region of interest by filtering [9] the original image using a two dimensional filter mask shown in Fig. 5(d). The mask is correlated throughout the image to generate a correlation map. The nearest point in the correlation map to the text identifier with high correlation response is considered as the left-top corner of the textbox. Result of textbox detection corresponding to user name is shown in Fig. 5(b).

3) Button Detection: Detecting location of buttons follows similar approach to textbox detection. Buttons can also be observed as rectangular structures on the application screen. Unlike textbox, buttons typically have relatively lower aspect ratio. After performing OCR to find the text identifier, the left-top corner of button is detected. Result of button detection in Skype application can be seen in Fig. 5(c).

4) Wait for Loading: Waiting for completion of an operation in the mobile application is an important feature. It is required to ensure that the application is not interrupted before it is ready to receive input from the robot. We use the stream of images captured from camera to identify presence of activity by comparing subsequent frames over a period of time. In the Skype authentication test case, wait action is required when the system is trying to sign in (see Fig. 5(f)). Activity in the screen can either be global or can occur in a local region in the screen. Local region is defined as smaller rectangles inside the screen, which together cover the full screen. Global change in device screen can occur due to change in application screen. Local changes occur either because of the loading icon (see Fig. 5(e)) or due to device related icons like battery and network indicator.

Occurrence of global change is identified by comparing the global histograms of subsequent frames and the testing system waits until the screen is stabilized. After this, we detect local changes. The local regions having a change (see Fig. 5(f)) are detected and compared with the loading icon seen in Fig. 5(e). Sum of Squared Differences of intensity histograms are used for comparison [10]. The histograms are normalized to ensure scale and rotation invariance. Result of loading icon detection is shown in Fig. 5(f). If the loading icon is detected in the screen, the system waits until an ongoing operation is complete. In the absence of loading icon in the screen, all the local changes are assumed to be device related, are ignored, and the operation is taken to have completed.

VII. Future Work

The system mentioned above is very generic in nature, and can be extended in multiple ways. In this section, we present some of these ideas.

A. More realistic Remote Testing

The current remote system has the remote tester doing the testing on the screen on the PC using the mouse. Instead, this program can be loaded on a real handset (if possible, on the same model which is placed on the platform for testing), and it can be used to test the application remotely. The only requirement is that this mobile handset needs to have an Internet connection (using WiFi), so that it can connect to the system under test.

Another possibility is to have a touch screen monitor attached to the computer of the remote tester. Instead of using the mouse, he can use his fingers to give inputs. This will give a better user-experience to the tester.

B. Intelligent Automated testing

In addition to the automation, we can try to make the testing more intelligent. For instance, consider a telephone-directory kind of application. If one is to search for a name in a list (list also created as part of test), one needs to scroll till the name is found or the end-of-list is reached. The number of times to scroll clearly depends on the screen size. One option is to have a test script for each handset model. Doing this, we can precisely know the number of times scrolling is required, given the model. An intelligent way is to have only one test script covering all models, but with looping and conditional checking behavior. The checking happens with the help of the output subsystem (camera/OCR). On each screen, there is a check if the search string is found. Otherwise, a scroll happens. This loop is repeated till the name is found or the end-of-list is reached.

Similar mechanism is also required for the case where one needs to search for data in dynamic content being downloaded.

C. Generic Handset based device testing

This approach of testing can be envisioned to be used to test any handset based device, not necessarily limiting to mobile phones. Few instances are:

- Medical Diagnostic Devices
- Point of Sale (POS) machines
- Internet Protocol Television (IPTV)

As can be seen, all these have some form of a keypad and a screen.

VIII. Conclusions

The remote mobile-phone-application testing system presented in this paper allows the tester to be located at a remote location, away from the location where the mobile handset and the service provider are located. Also, it doesn't require instrumenting the phone in any way. This system allows a reduction of the testing costs, along the lines of the outsourcing model. Also, the system can be used with no or little modifications for application testing on devices having a handset & screen. In addition, the automation approach will assist in doing the testing without human intervention, and will have a huge economic impact.

IX. Acknowledgment

The authors would like to acknowledge Dr. Richard Fernandes of Spectrum Interface for building the 3-axis robotic system.

References

[1] Deviceanywhere. [Online].
Available: http://www.deviceanywhere.com

[2] Perfectomobile. [Online].
Available: http://www.perfectomobile.com

[3] Omron. [Online]. Available: http://www.omron.com/

[4] Autonics. [Online]. Available: http://www.autonics.com/

[5] iball c12.0 face2face. [Online].
Available: http://iball.co.in/Product.aspx?c=11

[6] Dinolite AM413TL-M40. [Online].
Available:
http://www.dino-lite.com/products_list_minute.php?product_number_abridge
=AM413TL-M40

[7] Skype. [Online]. Available: http://www.skype.com/

[8] IRIS. [Online]. Available: http://www.irislink.com/

[9] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd ed. Prentice Hall, 2002.

[10] G. Pass, R. Zabih, and J. Miller, "Comparing Images Using Color Coherence Vectors," in *Proceedings of the* 4^{th} *ACM International Conference on Multimedia*, 1996, pp. 65–73.