



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

OSKU REINIKAINEN
DEVELOPING DEVICE ADAPTATION FOR MODEL-BASED
TESTING

Master's thesis

Examiner: Professor Hannu-Matti
Järvinen
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical
Engineering on 5 June 2013

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY

Master's Degree Programme in Signal Processing and Communications

Engineering Technology

REINIKAINEN OSKU: Developing Device Adaptation for Model-Based Testing

Master of Science Thesis, 45 pages

March 2014

Major: Embedded systems

Examiner: Professor Hannu-Matti Järvinen

Keywords: Testing, Automatic testing, Model-based testing

Software testing is an important part of software development. In this work, a general testing interface which can be used to control separate test equipment is implemented. The purpose is to match the two different interfaces. This is done by implementing a higher-level interface which controls lower-level equipment.

The thesis deals with general software development and its different levels. Different levels of automatic testing that are *Capture and replay*, scripts, data driven, keywords and action words and model-based testing, are presented. Model-based testing is dealt with more detail because the interface will become part of a system which is used for model-based testing. Automatic testing of a graphical user interface is dealt with separately because test devices are used and tested with this method.

The implementation of the interface needs tools and image processing as a technical basis. Regarding the image processing, text recognition as well as the image and pattern recognition are dealt with because they are needed for the verification of the system state. Interfaces that are part of the tools are dealt with because the interface is implemented on them. These include fMBT (free Model-Based Testing) and TnT (Touch and Test). fMBT is a model-based testing system and TnT is OptoFidelity's robotic testing system. Other tools include Android interface, AAL (Adapter Action Language), and Tesseract. Android interface can control system which contains Android mobile operating system. AAL is modeling language used by fMBT and Tesseract is OCR (Optical Character Recognition) engine used by fMBT.

Interesting cases of the implementation are dealt with in more detail. The implementation is divided to functionalities of general, fMBT-version and TnT-version. Testing of the interface is dealt after these implementation problems. A few models were made to Google's Gmail, Calendar, Google+ and Hangouts programs by using interface. These models were used for debugging of the programs and with their help logical mistakes of the interface were resolved.

As a result, the fMBT version of the interface has been made to work and the TnT version has executed preliminary tests. The interface can be used with models and it is easy to change to another version.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO

Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma

REINIKAINEN OSKU: Mallipohjaisen testauksen laiteadaptaation kehittäminen

Diplomityö, 45 sivua

Maaliskuu 2014

Pääaine: Sulautetut järjestelmät

Tarkastajat: Professori Hannu-Matti Järvinen

Avainsanat: Testaus, Automaattinen testaus, Mallipohjainen testaus

Ohjelmistotestaus on tärkeä osa ohjelmistokehitystä. Tässä työssä toteutetaan yleinen testausrajapinta, jonka avulla voidaan hallita useampaa erilaista testausjärjestelmää. Tarkoitus on saada sovitettua kaksi erilaista rajapintaa. Tämä tapahtuu toteuttamalla ylemmän tason rajapinta, jonka kautta alemman tason laitteistoja hallitaan.

Työssä on käsitelty yleistä ohjelmistokehitystä ja sen eri tasoja. Automaattisesta testauksesta käsitellään *nauhoita ja toista* -skriptit, komentojonot, dataohjattu testaus, avain- ja toimisanat sekä mallipohjainen testaus. Mallipohjaista testausta käsitellään tarkemmin, sillä rajapinta tulee osaksi järjestelmää, jota käytetään mallipohjaiseen testaukseen. Eriksien käsitellään automaattinen graafisen käyttöliittymän testaus, koska testattavia laitteita käytetään ja testataan tällä menetelmällä.

Rajapinnan toteutukseen tarvittavat tekniset perusteet liittyvät työkaluihin ja kuvankäsittelyyn. Kuvankäsittelystä käsitellään tekstin sekä kuvan- ja hahmontunnistus, joita tarvitaan järjestelmässä tilan varmennukseen. Työkaluista käsitellään rajapinnat joiden päälle rajapinta toteutetaan. Näitä ovat fMBT (free Model-Based Testing), joka on mallipohjainen testausjärjestelmä ja TnT (Touch and Test), joka on OptoFidelitin robottitestausjärjestelmä. Muita työkaluja ovat Android-rajapinta, jonka kautta voidaan hallita Android mobiilikäyttöjärjestelmällä varustettua laitetta, AAL (Adapter Action Language), joka on fMBT:n käyttämä mallinnuskieli sekä Tesseract, joka fMBT:n käyttämä tekstintunnistussmoottori.

Rajapinnan toteutuksesta mielenkiintoiset tapaukset käsitellään tarkemmin. Toteutus jaotellaan yhteisiin, TnT-version ja fMBT-version toiminnallisuuksiin. Rajapinnan testaus käsitellään näiden toteutusongelmien jälkeen. Muutama malli toteutettiin Googlen Gmail-, Calendar-, Google+- ja Hangouts-ohjelmalle hyödyntämällä tehtyä rajapintaa. Malleja käytettiin ohjelmien testaukseen ja niiden avulla selvitettiin rajapinnan loogisia virheitä.

Lopputuloksena rajapinnasta on saatu toimimaan fMBT-versio ja TnT-versiolla on ajettu alustavia testejä. Rajapintaa voidaan käyttää mallien kanssa ja tarvittaessa sen vaihtaminen toiseen versioon on helppoa.

PREFACE

This Master's thesis has been made for Tampere University of Technology as part of RATA (Robot Assisted Test Automation) project.

Professor Hannu-Matti Järvinen examined this work. Instructor D.Sc. Antti Jääskeläinen checked the contents and the linguistic form of the thesis. The base of the interface was his doing and in addition he generated models which were used in case studies.

I thank Hannu-Matti Järvinen, Antti Jääskeläinen, Matti Vuori and Heikki Virtanen who worked in the same project and assisted in the work. Furthermore, I thank other members of the consortium, VTT, OptoFidelity and Intel.

Tampere 11.02.2014

Osku Reinikainen

CONTENTS

1. Introduction	1
2. Software testing	3
2.1 Testing levels	3
2.2 Test automation	5
2.3 Model-based testing	8
2.4 Automatic testing of graphical user interface	10
3. Technical basis	12
3.1 Image processing	12
3.1.1 Text recognition	12
3.1.2 Image and shape recognition	13
3.2 Tools	13
3.2.1 fMBT	13
3.2.2 Android interface	14
3.2.3 AAL	14
3.2.4 TnT	15
3.2.5 Tesseract	16
4. Implementation of the interface	18
4.1 Common functionalities	18
4.1.1 Scrolling	19
4.1.2 Text recognition	20
4.2 Implementation of the TnT version	22
4.2.1 Writing of the text	22
4.3 Implementation of the fMBT version	23
4.3.1 Text recognition problems	23
4.3.2 Text selection	27
4.3.3 Problems of the image recognition	28
4.3.4 Menu selection	29
4.3.5 Text writing	29
4.4 Testing	30
5. Case studies	32
5.1 Gmail	32
5.2 Calendar	35
5.3 Google+	36
5.4 Hangouts and Google Talk	37
6. Results and discussion	39
7. Conclusions	41
References	44

TERMS AND ABBREVIATIONS

AAL	Adapter Action Language. It is used in fMBT to create tests as modeling language.
ADB	Android Debug Bridge. It is debug bus in Android operating system made by Google.
Android	Open source operating system developed by Google. Designed especially for the mobile devices.
ASCII	American Standard Code for Information Interchange. Standard 7-Bit character set which is generally in use.
C++	C-based programming language.
fMBT	free Model-Based Testing. Tool that has been designed for a model-based testing.
GUI	Graphical User Interface.
HTTP	Hypertext Transfer Protocol. Data transfer protocol of Web-based services.
JSON	JavaScript Object Notation. Text-based human-readable data exchange standard.
OCR	Optical character recognition. It means the identification of the text from the picture.
PNG	Portable Network Graphics. Lossless image storage format.
Python	Programming language.
SUT	System Under Test.
Tesseract	Open source OCR software.
Tizen	Open source operating system, which is governed by Linux foundation. Designed especially for the mobile devices.
TnT	Touch and Test. Robot control interface developed by Optofidelity.
UML	Unified Modeling Language is the general modeling language of the software engineering, standardized ISO/IEC 19501:2005
USB	Universal Serial Bus. General interface that is used for serial traffic between two devices.

- VNC Virtual Network Computing is a graphical desktop sharing system.
- X11 X Window System version 11. The system-level software infrastructure to windowed graphical user interface.

1. INTRODUCTION

In software testing the goal is to verify the right operation of the program. Software testing has been divided into several different levels according to the level of features being tested. Such levels are unit testing in which an interface is tested, and system testing in which the operation of the whole system is tested.

Software testing can be done by hand, but longer-term tests require test automation. Test automation is able to perform tests that are difficult or troublesome to perform by humans. Such tests are often long duration and would require constant monitoring, to which the human is bad.

In the test automation, development has happened along the years. From early *capture and replay*-scripts development has evolved to hand-coded *keywords and action words*. In the keywords and action words the system's functionality in tests is referred with special words, whose implementation is defined elsewhere.

The work deals with automated testing, and in particular model-based testing, which has become separated to be an own testing branch in an automatic testing. In model-based testing the idea is that the system is described as a model that allows the tests to be created.

The purpose is to implement the general intermediate layer with the instructions that can be used to test SUT (System Under Test), regardless of whether a device is controlled by software through the debug interface or with the help of the testing robot. This interface is called the adaptation interface and it can control separate test equipments on the same tests. The problem is that different equipment has different interfaces so they need different implementation.

One part of the interface was made to use Android which is a mobile operating system interface of the fMBT (free Model-Based Testing), which was reason to use Python version 2.7 as a programming language. The second part of the interface was made for OptoFidelity's TnT (Touch & Test) robot, which has been designed to the test touch screen devices. The implementation was made by trying different alternatives and with their help the best alternative was searched for. In the future, the interface can be connected to the TEMA tool, which has been developed by TUT (Tampere University of Technology) for model-based testing of a graphical user interface [20].

The tools that are needed in the work are connected to equipment and to image processing. These are communication interfaces and image and text processing algorithms. Image processing tools are text-detection algorithms, image processing libraries and pattern recognition algorithms.

The goal is to create the functional interface which can be used to execute automatic

tests for a touch screen device using Python 2.7 programming language. The problem is different control buses of devices. The interface had been already defined broadly but it must be changed when the work proceeds.

Chapter 2 gives a general description about the software testing for which the interface has been developed. The testing levels, test automation, model-based testing, test coverage and the automatic testing of the graphic user interface are dealt with.

Chapter 3 covers the technical basics of the work. These things are image processing techniques, such as text, image, and pattern recognition that are used in the work. The second part of the technical backgrounds relates to the used tools which are fMBT in other words free Model-Based Tool, Android interface, AAL in other words Adapter Action Language, TnT in other words Touch & Test and the Tesseract OCR (Optical character recognition) engine.

Chapter 4 details the implementation of the interface. At first, this chapter describes problems that both versions encountered, after this the problems of the TnT version are dealt with finally the problems of the fMBT version are dealt with.

In chapter 5 a few created models, and the problems that have arisen with them, are dealt with in detail. These models are Google's programs Gmail, Calendar, Google+ and Hangouts. Gmail is an email client, Calendar is a calendar application, Google+ is a social media application and Hangouts is a chat application.

In chapter 6 results are examined and conclusions are in chapter 7.

2. SOFTWARE TESTING

Because the interface to be designed is developed as part of a testing system, the testing is an important part of the theory. The interface has been developed to be used especially with model-based testing but it also works with other testing ways.

In testing the objective is to ensure the right operation of the system and to locate the errors. Merely the starting of the system and the experimenting of the functionality of the basic functions is testing. For this reason, testing should start as soon as possible, and it should be part of the project.

2.1 Testing levels

The idea of testing levels is in the fact that test plan is created to every level of the waterfall model and it is implemented at the corresponding test level. This changes the waterfall model to the V-model of testing which can be seen in Figure 2.1 [4]. The testing levels are unit testing, integration testing, system testing and acceptance testing.

In the unit testing every part of a program or system is tested separately. Usually it will be a part of the implementation stage of the unit and usually the implementor of the part does it. However, this causes the fact that the information about the mistakes may stay only in the implementor's knowledge. Unit testing is divided into several subparts, including the interfaces, data structures used by the unit, execution paths and loops, error handling, as well as limit values.

Interface testing is one of the most important and the first things that need to be tested. Because the interfaces determine how the units function outwards, their operation affects the operation of other tests. If the interfaces do not work correctly, other tests will not necessarily work correctly either. In addition during development of the code, the interface stays possibly the same even though the code changes so the same tests operate also after the changes.

In the testing of interfaces the most common problems are related to the number, order and type of the parameters. Also, the types of return values, global data, and constant-value parameters cause their own problems. These problems are usually identified by the compilers, but in dynamic scripting languages these problems will arise in the tests. The compilers do not detect either that caller and called interprets the parameter or return value differently. For example, the integer type return value "length" can be interpreted as inches by the recipient when the sender 'meant' centimeters.

In the testing of data structures the goal is to verify the data structures used by the unit.

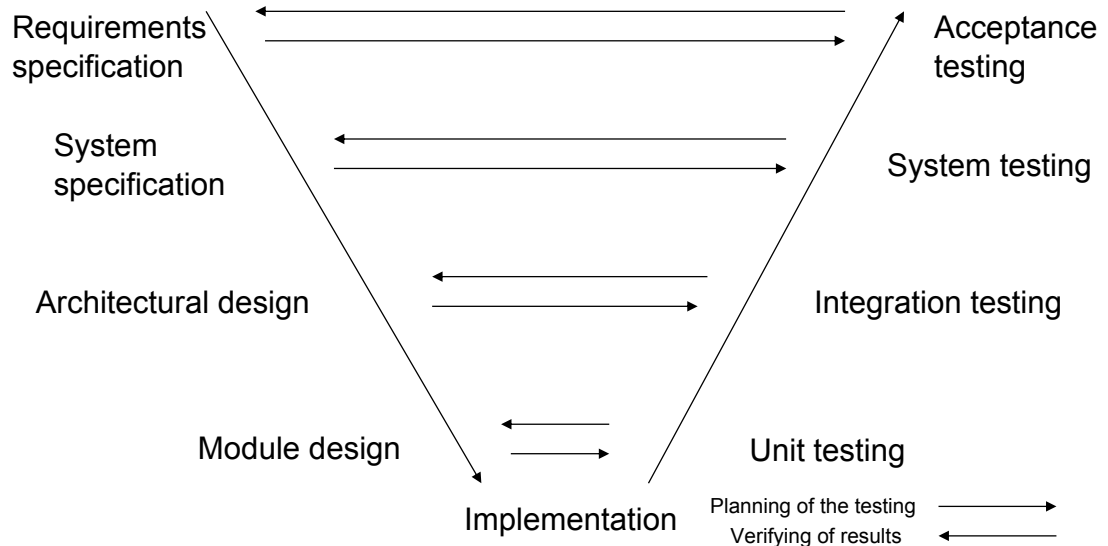


Figure 2.1: The V-model of testing. There is a traditional waterfall model on the left side and on the right side the testing level which corresponds to the that level.

The problems will arise from type errors, initialization of default values, from the spelling of the names of variables, from the inconsistent use of data types and exceptions, as well as overflow and underflow. The compiler usually notices these mistakes, but the number of the mistakes is worth minimizing by using ready-made data structures.

In the testing of execution path and loops, the purpose is to test the branching sections of the code. The nested loops make the number of the test cases increase fast and make testing difficult. Furthermore, the consecutive loops which depend on each other also make the number of the test cases grow.

The handling of error situations also must be tested. Often the error condition is not much taken into account and this may cause the program not stand the slightest disruption in execution. Furthermore, the error messages given by the program do not necessarily help in the solving of the problem. Because the costs of correcting mistakes rises when they reach further in development, the mistakes should be repaired as early as possible.

In the limit value test, limit values of the unit parts are tested as its name imply. This means that the parameters and return values may not get incorrect values that are bigger or smaller than permitted values. Loops may not rotate too much, that is, they may not stay in the eternal cycle because of the missing limit values. Data structures dynamic behaviour has to be tested, i.e., the upper limit of memory usage shall be within given bounds. Also their excessive reduction must be checked, in other words removal attempts made to memory areas, which do not belong to the data structure, must be checked. In addition the reduction itself must be checked, that memory leaks do not happen, in other words the size of the memory is bigger than a limit value.

Software integration begins in the developer's workstation. In software development version control software is used, which allows different components to be managed as a

single entity. When a developer adds his own share of the project, he will test that the project compiles and executes. In that case the developer will perform the integration testing.

Integration should be done in small sections, for it is easier to find errors in the smaller parts. If it is made in one go, the correction of mistakes will slow down, because they are difficult to find, and after fixing error one has to run a number of tests to ensure that nothing else is broken. For this reason the integration is made nowadays continuously.

In continuous integration each save to version control system causes automatic compilation and possibly automatic test execution. Therefore, if the change causes malfunction in mutual operation of parts, this is immediately detected. Tracking error is generally easy in this way, when an error is known to have occurred in the changed parts, which are usually made within a few days.

In the system testing, a whole system is tested, which is usually separate testers' work. The purpose is to ensure that the operation of the overall system and the separate parts of the system work together. For this purpose a system test plan is designed to be used as a basis of the testing.

Because the testing in this level has been separated from software development, the quick carrying of fault information to the developers must be ensured. This way the spreading of the error can be prevented.

The purpose of acceptance testing is to test that the finished product meets the customer's requirements. The customer is responsible for this testing. In acceptance testing, the testers should include end-users of the product and the test environment should be as real as possible. In the real location the testing would give the most realistic results, but this is not always possible.

The client should find many flaws and mistakes quickly, while the supplier should favor poor and rapid acceptance. This difference is due to the fact that the revenue intake will slow down, parts that needs repair are found a lot and the supplier will want to make income after the warranty period. The client should only start using functional finished product in turn, in order to avoid the opposition and disgust caused by the unfinished product.

However, the people are bad at repeating and watching long functions. For this the test automation has been developed.

2.2 Test automation

This whole subchapter is based on Mark Fewster's and Dorothy Graham's book *Software Test Automation: Effective Use of Test Execution Tools* [4].

In test automation mechanically or programmatically tested system is executed without human to monitor and control it. This kind of a system can also be used to carry out set of test operations lasting several days long which would be impossible to carry out manually. Automation allows the efficient use of resources and the re-usability of tests. The automatic testing is at its simplest a so-called monkey testing, where a test property or a device is

used until an error takes place. When the developing of the automatic testing is progressing and the intelligence is increased, the testing will not be completely random any more.

According to Mark Fewster and Dorothy Graham, one of the biggest problems of the test automation is expectations which are directed to it. People may think that there are no errors in the system when the automation does not find errors and people assume that tools solve all the problems. Furthermore, automation is not worthwhile for tests whose results are difficult to interpret by the machine, but the human being is able to interpret easily. Automation does not solve the fact that someone has to design the test. Test automation is complement to a manual testing, not a replacement and usually the mistakes will be first found with the manual testing, after which the tests are automated for re-testing.

Software development may also be limited because of test automation. This is due to the fact that the automatic tests may break down, which means that the test does not work properly, even from small changes in software being tested. The repair and maintenance of changes in the tests may be more burdensome than the fact that the test is performed entirely manually.

Progress has been made in the test automation during the years. The development stages are shown in Figure 2.2. The first was the so-called *capture and replay*, where a certain sequence and a given input will be recorded and it is run again. In this method all the changes require a new recording. The purpose is to look if the correct result is obtained when running a specific sequence. The scripts which have been created in the method are often of low quality because of their structuring and documentation is poor, their maintenance is almost impossible, and in addition they are linear. In them the advantage is that they do not necessarily require programming skills so it is possible for people not familiar with programming also to experiment them. They are suitable for presenting features of the software.

At the next level are the command sequences, or scripts. The difference is that input and a sequence are not recorded like previous stage, but the script is coded to execute it. The changes in the sequence require only a small change in script. Unlike in the recording the scripts are now well documented and structured. It is possible to take the error situations into better consideration in coded scripts and it is possible to recover from them better. Software design skills requirements increase at this level. In addition to the fact that the use can require programming skills, the same skills and the similar discipline as in the software planning are required at this level. There are two different types of scripts, structural and divided. The structural are similar to the linear scripts but they are coded manually. Shared scripts are a tool libraries with functions that take parameters, make executions, and return the return value.

Data driven testing is at a next level. In data driven testing the tests have been differentiated from the scripts so that the data will contain the test steps to be performed and the script produces the commands from them to the system under test. The purpose is that the same script can be used to execute more tests. The scripts do not contain the whole test any

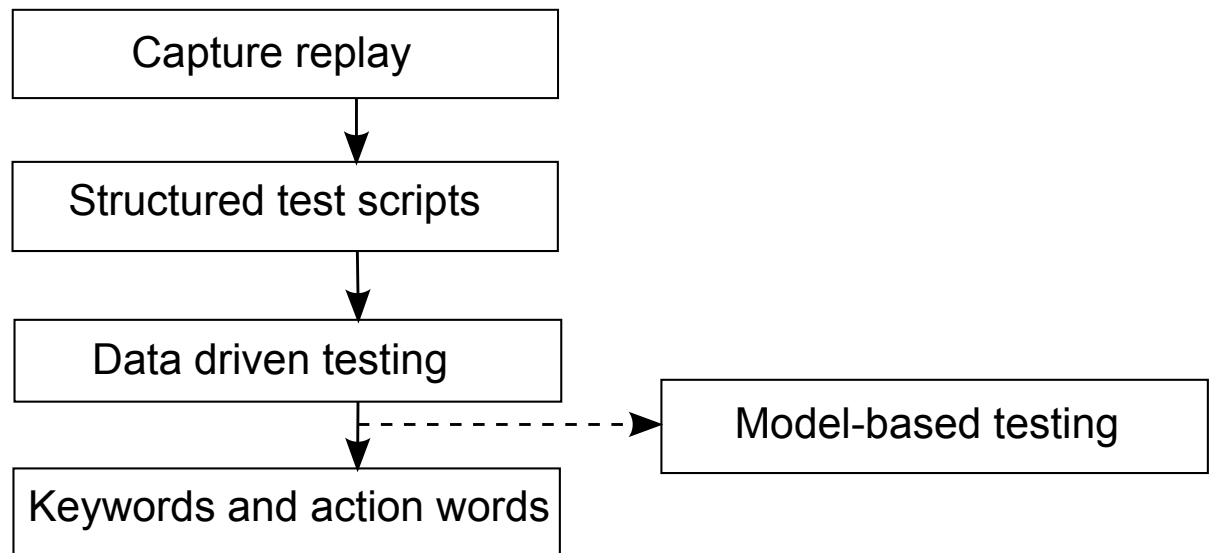


Figure 2.2: *Development of test automation. Test Automation began with the capture and replay recorded scripts. These scripts developed scripts that were hand-coded. From these scripts the test data was differentiated and a data driven testing was obtained. The differentiation was also increased by separating the navigation in the test subject from the tests and the keywords and action words were obtained. Some people began to consider writing test scripts to be burdensome, and began to think about the possibility of creating test scripts automatically. From this thought the model-based testing, which became separated to be an own testing direction, was created.*

more but only a navigation in the test subject, in other words how is it moved between the menus, and the assignment of the data to the fields. This results a smaller amount of the script with the same coverage, and one can use it to create a smoke or regression test set. This way new tests can be specified without additional programming. Requirement level of the programming will continue to grow because the coding of scripts requires programming skills but the creation of test sets requires testing competence instead programming skills. Furthermore, these roles will usually be worth distributing among different persons because the idea and interpretation differences come out if more people think about the same plan.

Keywords and action words are on the next level. This is a more advanced form of data driven testing, so that the navigation also has been separated from the script. In this testing, the keywords describe functions of the user interface and action words describe a higher level of user interaction. The tests are defined with action words, which in turn are implemented by keywords. The function remains the same, even though the platform changes. This way high portability will be reached even though keyword implementation is changed then the implementation of the keyword is corrected. In this case one does not need necessarily to touch tests themselves, if the functionality does not change, so there can be the same test for different members of the product family. For example the keyword `kwPressButton btOK` presses the button OK, but what happens at the level of the device depends on the device, and the action word `awSendEmail` could consist of `kwOpenProgram Email`, `kwPressButton btNewMail`, `kwWriteText address`, `kwWriteText`

subject, `kwWriteText` message, `kwPressButton` `btSend`. The scripts which are created with this method are usually linear and they do not necessarily implement from the flow control anything except leaving the error situation. Furthermore, their sequence is the same but the data may change.

2.3 Model-based testing

In model-based testing system requirements are used to create model that is used to test the system and provide a report. The test generator produces from a given model and parameters as good test cases as it is possible. Usually the test subjects are modeled as a finite state machine or as their extended version. Such extensions are for example state charts, the UML (Unified Modeling Language) state machines and Markov chains [21].

The state charts add the substates of states and the parallel states to finite state machines [8]. The substates of states mean in practice that the state contains the second state machine. Parallel states in turn mean that the state machine can be in several states simultaneously. UML provides a structured language, and it may also include other types of models such as state machines and state charts. The Markov chains can in turn be used to model stochastic, in other words random systems. They appear as state machines transitions probabilities, in other words it is probability of a state transition compared to all possible alternatives.

Also the transactions can be used as the foundation of the model. Pre-conditions and post-conditions are defined for these transactions. A term which is used from this kind of a model is an action-based model.

Already the mere modeling of the system usually helps to detect mistakes because the definitions of the system are usually in such a form that people can see ambiguities and contradictions directly [15; 17]. When an attempt is made to make definitions as a model, the conflicts and ambiguities will come out and they have to be resolved. Sufficiently accurate models can be shared to other testers and re-used.

There exists two variations of model-based testing, online and offline. In offline model-based testing, the system is modeled to create a model that describes its function, and allows the test generator to generate tests. The generated tests are executed on a separate test system. In an online model-based testing the tests are not executed separately, but the design tool creates from the model a test step, which is driven before the following step is designed. Differences between two variations is presented in Figure 2.3.

In the model-based testing, state machines themselves can become problem. When the model becomes too complex, that is, contains a lot of states and state transitions, it is not possible to deal with the existing algorithms efficiently. This situation is called a state explosion [22]. With normal state machines this does not usually take place, but rather it requires components that are brought from somewhere else and these components have their own states. In this case, the number of states in the worst case grows exponentially.

When one tests, it is important to know how well the system has been tested. For this

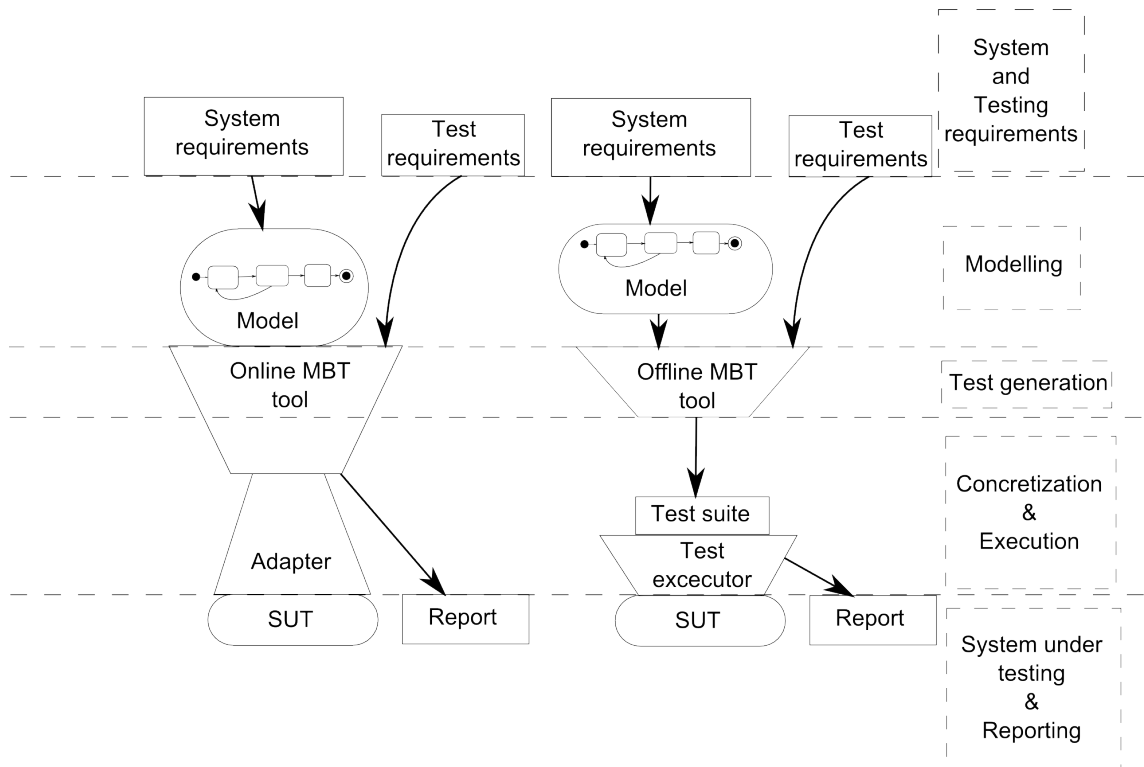


Figure 2.3: Online and offline model-based testing. The figure shows how the parts are placed in each style. Paraphrasing [14].

different coverage measures have been developed.

By measuring a coverage, one can conclude how much program has been tested. Using it alone, one can't determine whether testing has been sufficiently made. The phrase "you get what you measure" is valid also for test coverage. In the unit testing, one can check with the code coverage if all the branch sections of the code have been covered. Here the problem is that it is not known with a 100% coverage if the program functions right, it only tells if all the special situations of the program have been executed at least once. For example the logical mistakes are not uncovered by measuring the coverage of the code.

In the path coverage the purpose is to go through all the possible execution paths. In practice, this is impossible to achieve, because the programs contain loops that increase the number of execution paths exponentially. In addition, 100 % coverage does not mean that the program is faultless because the implementation could do something other than what it was supposed to do, it has been possible to leave some of the paths without implementation or values of the functions have been chosen badly.

The decision, condition, and multiple condition coverage are all related to the study of the conditional expressions. In the decision coverage all branches in conditional expressions must execute, but the conditions do not need to get all their values. In the condition coverage all the conditions have to get both truth values, but the two branches do not have to execute. In the multiple condition coverage every conditions must go through all the possible combinations. This causes decision and condition coverage to be tested at the

same time.

The measuring of the requirement coverage begins with the detailed definition of the requirements of every part. Creating the tests for requirements allows the requirements to be tested. These tests are mapped to the coverage requirement matrix with the requirements. It is always marked to this matrix after the executing of the test whether the system got through a requirement.

The general parameters of the test coverage in a model-based testing are parts of the model, such as a state, state transition or other parts and coverages in the model, which should be covered with tests. Instead it is not worthwhile to concentrate too much the code coverage, because in a reasonable code the hardest work has been done in the libraries and an attempt has been made to minimize the amount of the own code. In this situation, the coverage does not tell the whole story because, even if the code coverage would be 100%, the operation of libraries with all inputs is likely to be less certain. It also is worth paying attention to the fact that other coverage measures also operate in the offline testing, but those that are based to the number of the test cases, do not operate in the online testing.

2.4 Automatic testing of graphical user interface

The automatic testing of the graphic user interface is usually challenging, because it often contains a lot of working parts which all are burdensome to test.

The GUI (Graphical User Interface) testing can be done with automatic model-based testing. The views of the user interface become states easily and the buttons cause the state transfers, however, a protocol testing is still easier. Usually the programs or the user interface can be controlled with commands through an interface. In some cases the state can be concluded by asking the user interface what text it contains. Unfortunately this is not always possible and state verification must be made with the image processing by OCR (Optical Character Recognition) or search for the images of individual sub-images or shapes.

In the comparison of display images the purpose is to compare screenshots with pre-defined images. With this method a reliable result will be obtained when the version of the program stays the same. However, a corresponding image is not usually found with the 100% precision even if the image that has been made beforehand has been cut directly from a screenshot. This is due to the fact that the colors and border regions do not correspond to each other completely. For this reason, the accuracy of the comparison must be adjusted with suitable acceptance limit values. Unfortunately it can be difficult to find the suitable limit values. With too loose limit values sub-images can be found at the wrong places and the identification will slow down if the images contain lots of similar color areas when the algorithm has to check all points in the area to match sub-image. By using display images the functionality of the tests also suffers between different versions of the program. This is due to the fact that in the different versions of the program changes may take place and as a result corresponding sub-image is not found from the screen. Then a new sub-image,

which may break down in the next version, must be created.

Using the text and pattern recognition state verification, tests can obtain a pretty good performance between the different versions of the program. Because the text identification and pattern recognition do not care about other than changes in the color areas, they are highly resistant to small changes, such as a color change. In addition, they allow use of the same shape more locations for the same reason. The disadvantage of the use of the text identification is its uncertain operation. Even the best of the OCR engines do not always recognize all the words correctly. This often causes the failure of the state verification. The pattern recognition works fairly reliably, but in some other parts of the pictures there may be similar forms in which case the position may go faulty. The situations will get worse if the display has a complex wallpaper in which case both techniques probably find in the background recognizable shapes.

Using the programmatic components would give the most reliable results quickly, because they are getting their information directly from the user interface. The problem is that their use is not always possible because they require support in the system to be tested. The user interface must support these components, but in some situations, such as Android devices, the device itself must support this feature. By utilizing programmatic components the biggest problem can be that the information returned by the component does not correspond to what the user sees. This is due to the fact that the system may contain invisible parts to the user but for the programs they are operating normally. For example, in a situation where there are multiple tabs, and on-screen elements are asked from component, the answer may contain information from all tabs, though it should return information only from a single tab.

The maintainability can be improved by combining the model-based testing to the keywords and action words [9]. The parts of the user interface are described with keywords and the operations are formed with the action words. With their help test automation states and model are formed and model allows the test generator to run or create tests. In this way, most of the changes relate to the keywords and the model itself needs to be changed rarely.

3. TECHNICAL BASIS

The purpose is to match two different interfaces, one of which is the device control interface of fMBT and the other is TnT management interface. These two are not compatible with each other, so that they both could be easily utilized in a model-based testing. Both implementations of the interface need text identification and one needs image recognition and the other pattern recognition.

3.1 Image processing

In the adaptation interface, image processing is needed to search targets and text from the display. This takes place by taking either a screenshot or a photograph of the display and driving text identification, image recognition or shape recognition to it.

3.1.1 Text recognition

In text recognition, in other words with OCR an attempt is made to identify the text from a digital picture. Historically, the goal has been convert the old paper documents and books into digital format. The problem in question could be solved merely by taking the photograph or by scanning the document in question but this method contains a problem. The picture is usually many times bigger in the file size than the text document which contains the corresponding information.

In the text recognition difficulty is caused by the characters which do not belong to the ASCII (American Standard for Information Interchange) character set. In a European character set these characters often resemble a character of the ASCII character set so the worst algorithms may mix up them. In the worst case the algorithm supports some of these characters but not all, so every character which is unknown to the algorithm will most likely become totally wrong when the sign resembles some special character known by the algorithm.

The background is often a reason why the text recognition fails. The more complex the background image is, the more likely some of the text disappears. The worst situation arises when the background has the same colors as in the text. In that case it will be difficult for the identification algorithms to filter the background away, so that no part of text are lost at the same time.

3.1.2 Image and shape recognition

Image recognition means that subimage corresponding to searched image is searched from screen. This is done by pixel match, that is, each image pixel color value is compared with the search image pixels, and if all the pixel values are close enough, the image is found. A data format can cause problems in the image recognition. If the pictures are saved with a lossy compression, distortions will be created to the border regions of colors. These distortions can cause the failure of the recognition if the identification does not allow any distortions at all. Decrease in precision, in turn, can cause false detections. The problem can be avoided by using only lossless data formats.

In shape recognition, which is very close to the text recognition, an attempt is made to find the desired shapes in the picture. In that case the picture will not be compared pixel precision but rather the borders of the color range are searched and they are compared to the desired shape. In shape recognition problems are caused by so-called hidden shapes. Since shapes are attempted to identify from pictures, the pictures may contain hidden shapes that represent shapes that are searched. Such hidden shapes may be found in complex pictures and icons. The observation of the such can be affected with suitable limit values.

3.2 Tools

In the adaptation interface the tools are needed for to communicate with the SUT. The interface controls SUT either directly with fMBT, which is a tool that has been designed to a model-based testing, or indirectly through the TnT robot testing interface, which is a touchscreen device testing system. Both versions utilize AAL (Adapter Action Language) of fMBT for models.

3.2.1 fMBT

fMBT (free Model-Based Testing) is a tool which by utilizing the model generates and executes the tests automatically [1]. According to 01.org "it is suitable for testing anything from individual C++ classes to GUI applications and distributed systems, containing a range of different devices." The test generator is running in general Linux platforms and is able to do both online and offline model-based testing. The individual test can be executed on different devices and it can contain different programming languages.

fMBT provides an easy interface to manage the device under test. When writing this fMBT contains four versions: Android, Tizen, X11 and VNC. Android and Tizen are mobile operating systems, the X11 is the X Window System version 11, in other words windowing system, and VNC is a system that has been designed to the remote control of the graphic user interface. The interface of these four versions has been kept the same as far as it is possible. The interfaces have been written with Python 2.7 programming language.

fMBT has been published under an open source license, so anyone can develop it further. The source code is available at GitHub [2]. At the moment five different development

versions have been separated from it.

3.2.2 Android interface

fMBT communicates with the Android device through its internal debug interface. According to Google [5], ADB (Android Debug Bridge) is "a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device." The interface can be used to test Android functions directly from the command line without a user interface. For example the transmission of text messages is possible without a separate program. It can also be used to test the functions and programs of the user interface without a physical device with the help of the emulator. When it is used to send pressings of the display and other gestures which are done with the finger, programs can be tested in almost the same way as they would be used for real.

From Android 4.2.2 version the use of the interface has required an acceptance of the connection separately. This feature requires at least adb version 1.0.31. This feature was added for security reasons, so that it was possible to prevent the bypassing of the devices safety locks with it.

With the interface it is also possible to control more than one device or emulator and at the same time. This is done by typing the parameter `-s` and the device's or emulator's adb serial number when entering command.

3.2.3 AAL

In fMBT, models are described using AAL modelling language, which defines the models using pre-conditions and post-conditions [10; 11]. AAL has several versions for different programming languages, but the syntax of AAL itself stays in each one of them the same. The test steps and their conditions are programmed with the second language.

In AAL, the models consist of variables, an initial state, the initialization of the adapter, the termination of the adapter, actions and tags. The variables contain the internal variables of the model. In the initial state internal variables of the model get their initial values. In the initialization of the adapter, adapters' start actions are adjusted. In the termination of the adapter the actions which will be performed when the run of the test ends are determined.

In AAL, the actions consist mainly of three parts, which are guard, body and adapter. The guard section defines the conditions, which determine when test step can be performed. The body changes the state variables of the model, in other words changes the state of the model in practice. The adapter block contains the code which is related to the execution of the test. The tags that verify the current state of the system consist of only the guard and the adapter block. In Figure 3.1, which has been taken from fMBT editor, you can see one model with two actions and one tag.

The model in Figure 3.1 does not contain adapter blocks, but those blocks should contain control commands of the test device, such as touch commands of the phones. For all the

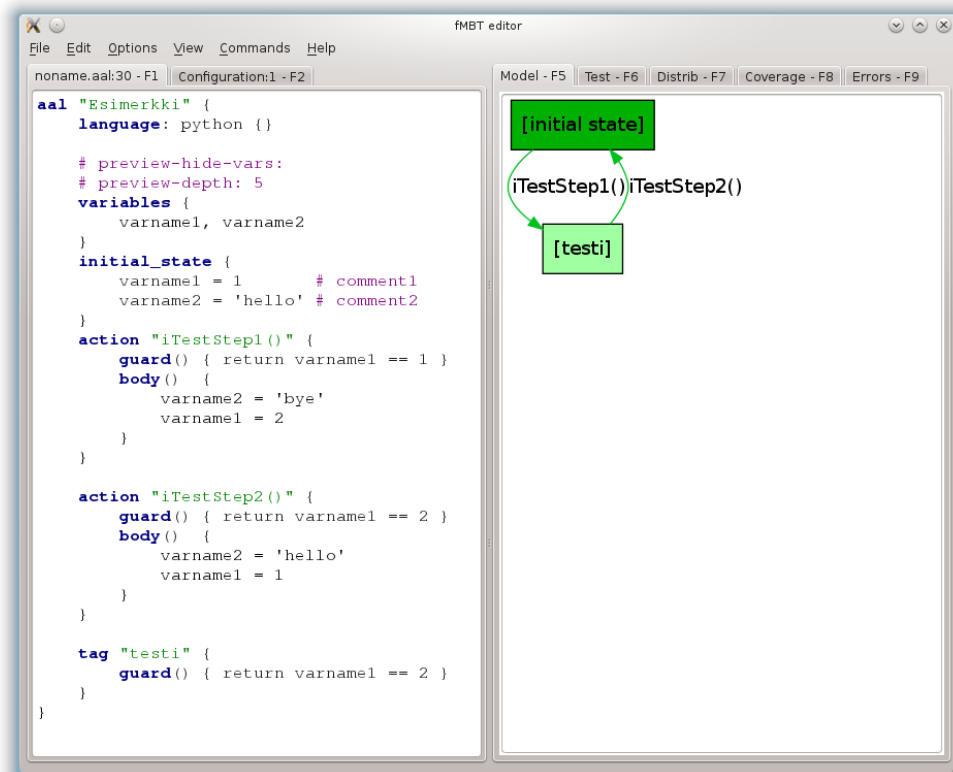


Figure 3.1: View of fMBT editor with one model, two actions and one tag. fMBT can automatically generate a graphical representation of the model.

others blocks, except for variables block, the language of the code is defined in the second line as Python with command *language: python {}*, but there are alternatives, such as C++.

3.2.4 TnT

TnT or OptoFidelity's Touch & Test is a testing system developed to test touchscreen devices. The TnT system has been designed to operate without modifications on the test equipment [13]. This has been made possible with robot fingers, cameras and different kind of sensors. This allows for reliable analysis of the system operation, such as delays, the battery and power consumption. The robot equipment is shown in Figure 3.2.

The system utilizes advanced pattern recognition and text recognition, so that it knows how to navigate the user interface. For this reason the tests do not fail even if small changes occur in the user interface, such as icons change places. The testing system supports model-based testing and uses the state machine notation for the modeling. Furthermore modeling can be done with available software which contains a graphic user interface.

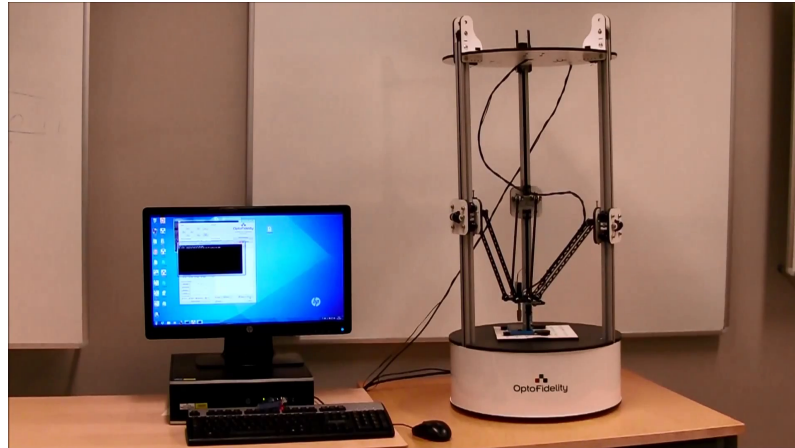


Figure 3.2: Delta robot which uses OptoFidelity's TnT interface. The computer on the left executes a control program of the robot. In the robot on the right side, there is mechanical 'finger' hanging in the middle. The picture is screenshot from the video [12].

3.2.5 Tesseract

Tesseract is an open source OCR engine which was originally developed by HP Labs and today it is developed by Google [7]. It supports more than 60 languages and a wide range of image formats. Tesseract supports Linux, Mac OSX and Windows.

Tesseract operates so that at first image is run through an adaptive threshold [19], which converts it into a binary image. This is driven to the connected components analysis which produces the character outlines. The text lines and words are searched from them, after which the outlines of the characters has been arranged into words. After this the words are run through the recognition algorithm and those words which were not recognized are identified again, by utilizing the information that has been received from the recognized words.

At first the lines are searched from the text [18]. This has been designed so that the text does not need to be straight. The baseline is estimated by using the smallest median of the sum of squares and the filtered "blobs" are fitted back to the suitable lines.

The baselines are fitted with quadratic spline curve, which is adapted to densest section. The calculation is relatively stable using quadratic spline, but discontinuities may occur when more than one part of the spline is required.

At this point, Tesseract tests whether the text contains monospaced text, and if it does, Tesseract breaks text down to the words and removes them from use of the chopper and associator. From rest of the text Tesseract measures the gaps between base and middle line in the vertical direction. In unclear situations the last decision is made after the identification of the word.

At first text identification part cuts the letters which have combined on the basis of concave vertexes in a polygon approximation. All cuttings are performed in order of priority, and cuts that do not improve the reliability are canceled and are saved for later use.

If the word is not good enough even after this, the associator will search the best candidate character which fits in to the pieces of the character.

Outline detection is done in such a way that the character outlines are marked, and after that they undergo a polygon approximation. This approximation will be used later for the comparison of the characters. This is made so that the character features will be collected and they will be compared with the prototype of the character. This also takes place the other way round, in other words the prototype is compared with features.

Tesseract also makes a base line or x-height normalization, which can be used to remove the effect of the font ratio. With its help the identification of a superscript and subscript is easier, but it requires an additional feature to distinguish some upper- and lower-case letters from each other.

4. IMPLEMENTATION OF THE INTERFACE

At the first stage, the adaptation interface was implemented to the Optofidelity's TnT robot [13] interface and fMBT testing tool [1] (especially to the Android version but in the theory other versions should also work). TnT-robot interface was a mere sketch at the early stages of development, but it was possible carry out the implementation with its help nearly completely. All the implementations were developed side by side even though the first versions of implementation were made separately.

On top of the testing interface there are models which control the test generator of fMBT. The test generator creates the test steps which use the adaptation interface which is selected according to used test system to control it. In Figure 4.1 you will see how adaptation interface is located in the testing chain.

The easiest way to extend the testing interface is implement a new version of the adaptation interface, which connects the tests to the system under test.

4.1 Common functionalities

Common features are the features that were implemented in both versions of the interface nearly as same. These changes were caused mostly by the different names and properties of the low-level functions.

Both systems contain pre-made functions which is able to press a screen, drag a finger over it, swipe, and search text and icons. The coordinates can be given to the commands as absolute or relative. The difference in the systems is that fMBT uses pixels and TnT uses millimeters as units.

The position of the touch can be given with the coordinates or systems can be instructed to look for an image or text and press it. Finger dragging is done by giving the start and end points, but the number of a movement points can be defined in fMBT. The robot does not have this option naturally, because it carries out movement analogically. The wipe function works same way in both interfaces. The position and direction of the wipe are given. In both versions the text can be searched by one word at time or all words at once. However, the TnT version returns more information than the fMBT version when searching information about all words. Both versions can search icons, but where fMBT versions uses normal images, the TnT version uses its own file format.

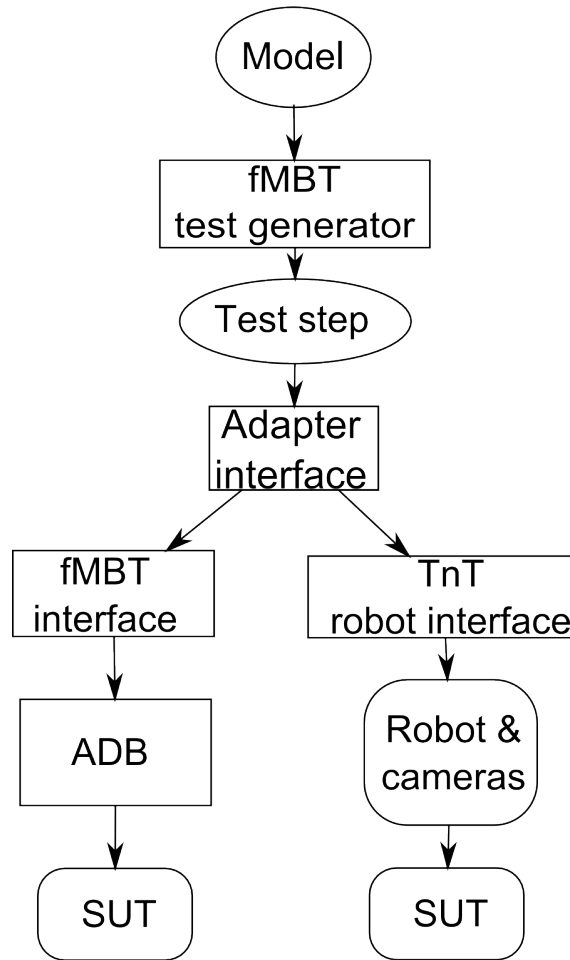


Figure 4.1: Structure description of the system where the tools are marked with the box, artifacts with circle and physical devices with rounded boxes. On the highest there is a model from which fMBT generates the test steps. The test steps utilize from the adaptation interface that version which has been defined in a model. TnT controls the robot which controls the finger that is used to control device under testing. fMBT communicates with the ADB interface, which in turn controls the device.

4.1.1 Scrolling

In the interface there are functions which can be used to search for a text or image that is not visible on the screen, but it can be displayed by scrolling. The scrolling itself is done by swiping in other words by drawing the finger with a display in one direction, but it is difficult to find out when execution has arrived to the end of the list. Simply thinking, the end of the list have been achieved when the image does not change anymore. Unfortunately the pixel comparison does not work if a browsable image has animation, such as Android desktop background, because the image does not stay the unchanged. For this reason the comparison of pictures is done by calculating the histogram, which is a statistical distribution of values, with root-mean-square difference. The equation is more easily seen from formulas 4.1 and 4.2.

In the formulas A and B are images to be compared so in practice they are matrices,

and H is the histogram of their difference matrix. The histogram function forms the table whose elements are color values and they contain the number of the corresponding color value. RGB images color channels values are included in the same table row consecutively, and it is reason to the remainder from number 256 operation in formula 4.2. In the second formula w is the image width and h is image height. The variable k is the number of elements of the histogram and in this case the RMS is the difference of the root mean square. If a difference is small enough, images are interpreted as the same. This works quite well, if the limit is set suitably small, otherwise scrolling may be cut off. However, the small limit value may cause several extra scrolling times at the end before the interface understands that the scrolling has reached to the end.

$$H = histogram(abs(A - B)) = histogram \left(\begin{bmatrix} |a_{11} - b_{11}| & \cdots & |a_{1n} - b_{1n}| \\ \vdots & \ddots & \vdots \\ |a_{m1} - a_{m1}| & \cdots & |a_{mn} - b_{mn}| \end{bmatrix} \right) \quad (4.1)$$

$$RMS = \sqrt{\frac{\sum_{i=0}^k H(i) \cdot (i \bmod 256)^2}{w \cdot h}} \quad (4.2)$$

This formula was chosen because it was relatively easy to implement and it worked pretty quickly and reliably. It is based on function made by Charlie Clark [3] with small changes. License bureaucracy set its own restrictions. Because the system is developed with companies, not just any ready made image processing library could used, but its license had to suitable to the requirements of the consortium. For this reason Python Imaging Library [16] is in use. From this library built-in functions were used for reading the images and the calculation of the histogram.

Another problem, which arises in scrolling, is the scrolling rate. If one wants to browse a stepless list, the wipe must be small or slow so that a part of the invisible information does not pass by when scrolling. If the scroll parameters are set such that the problem in question does not take place, the stepped browsing (such as a desktop) will not function because the amount of the wipe is not enough to change the view. The problem has been corrected by adding to the scroll functions a 'linear' parameter which tells if the scrolling is performed steplessly or not.

4.1.2 Text recognition

The text recognition is used to identify words from the display. This is done so that the screenshot is taken from screen and screenshot is sent to an OCR program, which returns the identified words and their locations as pixel coordinates in the image.

A common problem is a word order. The words that have been found in a western writing will be first arranged along vertical coordinate, which is a y-coordinate of the baseline, and

secondly they are arranged along horizontal coordinate and after that the words should be in the right order. This is basically correct, but the software does not work that way. Text recognition algorithms mark the words with bounding box, whose upper limit is placed at the upper edge of the highest letter and from it software takes vertical coordinate. The words which contain big or high letters, raise this limit higher than the words which do not have such characters. If there are such high words after low words on the same line, the high words will come at the list before low words.

The wrong word order is relatively easily to repair by reading the words one by one at first and arranging them according to the horizontal coordinates in to the table. The table corresponds to a row, but vertical coordinate isn't directly the line, but rather vertical coordinates that belongs to a suitable range. The view is shown in Figure 4.2, which is used for text recognition at example cases.

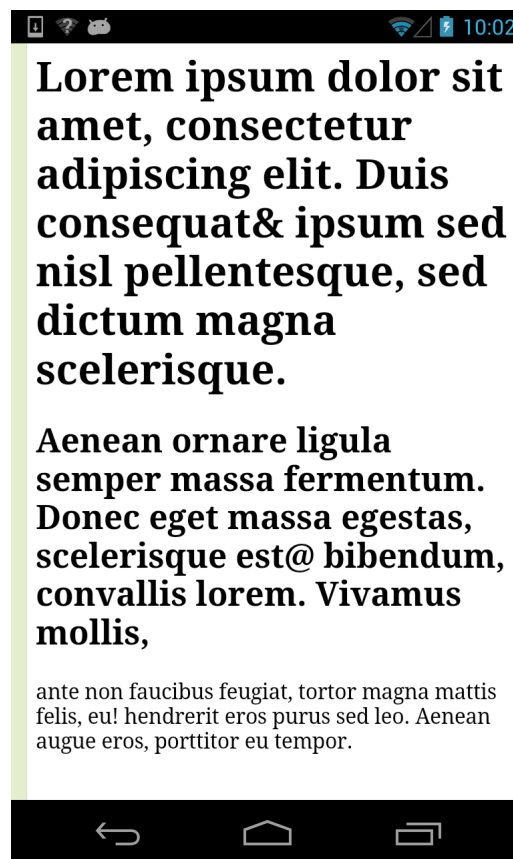


Figure 4.2: Image used for text recognition. The text has been taken from a well-know test text "Lorem ipsum" and few special character has been manually added to it.

Position that is returned after text search caused problem of its own. This is usually used to find out the target of the press function. It is easy to return middle position from an individual word, but the situation changes when the target of the search is part of the text. If all words can fit to same line, the middle of words will be returned. When the words are divided to multiple lines, an attempt will be made to follow the same logic.

The most difficult situation is encountered when the text takes up end of the one line,

and only a part of the next row. In this case, the situation may be that the desired area does not contain middle position which is over the chosen text. If the point is not over the chosen text, the touch can activate the wrong function. To avoid such situations calculation does not take second line into account, but the center point is calculated for first line.

4.2 Implementation of the TnT version

The first part of the implementation was made to a draft of the TnT interface of the robot. The implementation was begun with functions which have corresponding functions at interface, but which require only a small adjustment. The robot can be controlled using the HTTP (Hypertext Transfer Protocol) protocol, and then it returns the data in JSON (JavaScript Object Notation) format or through ready-made wrappers. OptoFidelity has made such wrappers to a few programming languages, such as Python and C++. The most difficult part in the TnT adapter is to process commands and data manually so that they are accepted. In particular, the data transmission is difficult to implement manually with HTTP Post method in Python code.

In addition the robot does not understand the standard image formats, but is looking for forms and shapes from the picture. This makes implementing generality of the interface harder because fMBT uses pixel comparison for searching images. For this reason different versions of the interface support different file formats.

In order to make adapter interface easily changeable which means changing only import statement in Python, the interface should not contain implementation dependent parameters. Because the TnT version does not use standard image formats (for example PNG, Portable Network Graphics), the formats must be changed to be suitable for robot. The problem is also that robot system sees the screen with separate camera, whose picture is affected by reflections and other photo technical problems. For this reason the robot will not necessarily find similar shape of the fMBT image. Another option would be that interface does not take image format identifier but rather it searches right image format itself with given name.

4.2.1 Writing of the text

Using a virtual keyboard it is difficult to write the text with the robot. In the Android the virtual keyboard has been divided into several different views and one must change between all of them. Each view has its own characters. In the basic view one can get the lower-case letters and with the 'shift' key one can get the upper-case letters. The numbers and the most common special characters are available from extra key. One can get rarer special characters from a new extra key opened by the extra key. Special letters like Scandinavian letters one can get with long press of the corresponding 'normal' letter or changing keyboard layout to make needed letters appear to screen.

Implementing this function to robot is difficult, because after every letter one must check which state the keyboard is in and possibly change it. This must be done either with shape

recognition or with the text recognition and they both are slow. At the moment the state of the keyboard is kept in memory and function tries to work according to it. If the function does not work correctly, keyboard is changed to correct state and if this fails an error is reported.

Another option would be to go to the back basic view after inputting every number or special character, but then successive inputting of the numbers and the special characters would be much slower. The current version of the interface checks state of the keyboard and either stay in the same state or moves to the next state. In other words the interface maintains information about the state of the keyboard and if necessary, tries to correct a faulty state.

It would also be possible to code the positions of keys to the part of the code and press their positions. The downside of this is that each modified keyboard layout needs to be coded separately and keep in memory.

The extra buttons, which change keyboard state, also cause problems. The marking of these keys is not fully settled and the characters may vary depending on the device and the keyboard layout. For this reason, searched text is adjustable from the implementation, according to button text.

4.3 Implementation of the fMBT version

At the second stage the adaptation interface was implemented to fMBTandroid which is the version of fMBT to the Android devices. The interface is able to control the Android device as long as the device is connected to the USB (Universal Serial Bus) interface and debug mode is turned on from device settings. The basic functions were nearly the same in the adapter interface and fMBT, only the names and parameters are changed. This includes `tapPosition` which touches the point of the display.

The significant difference between fMBT and interface is support to the identification of statements. fMBT interface supports searching for single words, but the searching of word queues does not work, due to its internal implementation. The problem has been solved, so that text dump is asked from all text on the display using fMBT. The table is examined entirely and the corresponding sequence of words is searched from it. However, problems arise because the text identification is difficult and error susceptible technique.

Other new features are Unicode support and text selection in the interface compared to fMBT. The interface uses the Unicode characters but their functionality varies, because fMBT does not support them. Text selection is difficult due to the Android selection tool features.

4.3.1 Text recognition problems

fMBT uses Tesseract as default OCR engine with two text recognition algorithms. This will sometimes cause word duplicates when the word is recognized in slightly different

positions or slightly different characters. In addition, these two algorithms do not always identify all the words.

The word recognition can be improved by increasing the number of algorithms, but it also increases the number of the possible duplicates, and it does not always help either as one can see in Figure 4.3. For this reason, removal of the duplicates should be done in such a way that their number does not matter. The most reliable way to identify duplicate is to compare coordinates and if they are close enough then it is a copy.

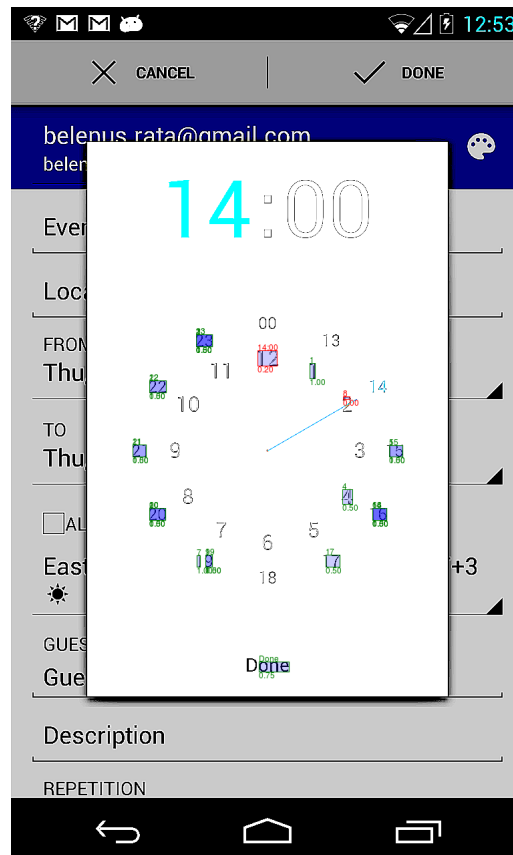


Figure 4.3: Difficult view of the Calendar program for recognition algorithm. It is nearly impossible to recognize all text from some situations. Android programs of the Google contains pop-up windows that will appear to middle of screen are such. Even combinations of best algorithms of the fMBT cannot find all text. The result looks like this at best. The green boxes mean at least 50 % certainty and red means less than that. The problem is caused by rest of the program which is visible in the background and it interferes with the recognition algorithms.

One can compare text dumps of the fMBT and the interface in Figure 4.4. The data returned by fMBT can be seen from table 4.1. Recognition performed by the interface generates more junk and variations of words than fMBT because of the added recognition algorithms. Garbage caused by the new algorithms is often icons which are visible on the screen. Instead the words which have been identified more than once have disappeared. In addition, the word order has been corrected.

In the searching of the words, more corrections have to be made to text recognition. Because it is very likely that the text recognition recognizes some of the letters wrong,

Table 4.1: The text dump produced by the fMBT. The dash lines means words that has been duplicated. The dotted line means that word contains difficult characters. The dash-dotted line means words that has wrong order. The normal line means that word has been duplicated and it has also wrong order. The text has been divided into three columns because of its big length.

(‘Lorem’, 38, 78),	(‘Lorem’, 39, 78),	(‘ipsum’, 262, 74),
(‘dolor’, 475, 75),	(‘sit’, 661, 74),	(‘amet’, 40, 154),
(‘consectetur’, 233, 154),	(‘adipiscing’, 40, 220),	(‘elit’, 378, 220),
(‘elit’, 379, 220),	(‘Duis’, 517, 220),	(‘ipsum’, 426, 293),
(‘sed’, 638, 294),	(‘consequat&’, 40, 296),	(‘nisl’, 38, 366),
(‘pellentesque’, 169, 367),	(‘sed’, 609, 367),	(‘dictum’, 40, 439),
(‘magna’, 277, 452),	(‘scelerisque’, 39, 513),	(‘ligula’, 428, 623),
(‘Aenean’, 37, 627),	(‘ornare’, 242, 635),	(‘ornare’, 243, 635),
(‘fermentum’, 409, 681),	(‘semper’, 38, 693),	(‘massa’, 239, 693),
(‘Donec’, 38, 742),	(‘eget’, 208, 744),	(‘eget’, 209, 744),
(‘massa’, 324, 750),	(‘massa’, 325, 750),	(‘egestas’, 496, 744)
(‘scelerisque’, 38, 796),	(‘bihendum’, 468, 796),	(‘hihendum’, 468, 796),
(‘est@’, 338, 800),	(‘convallis’, 39, 853),	(‘lorem’, 277, 854),
(‘Vivamus’, 455, 853),	(‘Vivamus’, 456, 853),	(‘mollis’, 38, 911),
(‘ante’, 38, 1011),	(‘ante’, 39, 1011),	(‘non’, 111, 1015),
(‘faucibus’, 177, 1008),	(‘feugiat’, 312, 1008),	(‘tortor’, 430, 1011),
(‘magna’, 526, 1015),	(‘magma’, 526, 1014),	(‘mattis’, 635, 1008),
(‘felis’, 38, 1045),	(‘eu!’, 116, 1046),	(‘hendrerit’, 169, 1045),
(‘hendrerit’, 170, 1045),	(‘sed’, 487, 1045),	(‘sed’, 488, 1045),
(‘leo’, 544, 1045),	(‘Aenean’, 605, 1046),	(‘eros’, 321, 1052),
(‘purus’, 391, 1052),	(‘porttitor’, 215, 1082),	(‘tempor’, 394, 1085),
(‘temper’, 394, 1085),	(‘augue’, 38, 1088),	(‘augue’, 39, 1089),
(‘eros’, 137, 1089),	(‘eu’, 351, 1089),	(‘’, 0, 1182),
(‘2’, 348, 1214),	(‘F’, 127, 1218)	

```

In [13]: print s
  Lorem Lorem ipsum dolor sit amet, consectetur
  adipiscing elit. elit. Duis ipsum sed consequa
  t& nisl pellentesque, sed dictum magna sceleris
  que. ligula Aenean ornare ornare fermentum. sem
  per massa Donec eget eget massa massa egestas,
  scelerisque bihendum, hihendum, est@ convallis
  lorem. Vivamus Vivamus mollis, ante ante non fa
  ucibus feugiat, tortor magna magna mattis felis
  , eu! hendrerit hendrerit sed sed leo. Aenean e
  ros purus porttitor tempor. temper. augue augue
  eros, eu  2 F.)

In [14]: print p
  Lorem ipsum dolor sit amet, consectetur adipisc
  ing elit. Duis consequat& cOnsequat& ipsum sed
  nisl pellentesque, sed dictum magna scelerisque
  . Aenean ornare ligula semper massa fermentum.
  Donec eget massa egestas, scelerisque est@ bibe
  ndum, bihendum, hihendum, convallis lorem. Viva
  mus mollis, ante non faucibus feugiat, tortor m
  agna magma mattis felis, eu! eu! hendrerit eros
  purus sed leo. Aenean augue eros, porttitor eu
  tempor. temper.

```

Figure 4.4: Comparison of the text recognition. Above is text dump produced by default settings of the fMBT and it is modified to look like the same as the output of the interface. Below is text dump produced by interface. The right text is Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis consequat& ipsum sed nisl pellentesque, sed dictum magna scelerisque. Aenean ornare ligula semper massa fermentum. Donec eget massa egestas, scelerisque est@ bibendum, convallis lorem. Vivamus mollis, ante non faucibus feugiat, tortor magna mattis felis, eu! hendrerit eros purus sed leo. Aenean augue eros, porttitor eu tempor.

most mistake alternatives must be added to the replacement list. The characters in this replacement list are replaced with the alternative character. When the characters have been replaced from both a text and searched words, the search is performed. If an edited search text corresponds to an edited text, then text in question will be interpreted as found. This will increase the discovery of the searched text and keeps amount of false discoveries small.

Because OCR can add extra characters to end of word, the search tries to take that problem into account. When a text is searched for, the interface accepts the extra characters at the beginning of the first word if the following words are correct. Also at the end of the last word there may be extra characters.

Furthermore, the text search requires its own duplicate removal. This is because functions that cleans text dump leaves all duplicates from which they are not able to conclude which of the alternatives is correct. After the first word is found, next word is searched and if it does not match the next word, its coordinates must also be checked. The coordinates have to be compare to previous coordinates. If coordinates are close enough and word corresponds to searched word, searching continues normally. If coordinates are close enough but word did not match to searched word, next word must also checked and so on. This continues until either the word corresponds to searched word or coordinates are clearly different.

If the searching is interrupted so that some of the words were found, but some word did not match, one must check whether current word corresponds to first word or not and act according to it. If this checking is not made it is possible that one of the text places remain undetected.

If the first searched word has multiple occurrences in text, the special situation needs to take into account. If the beginning of the text is found and it contains first words second appearance, the search should continue from the second appearance. If searching is continued from where it stopped, it possible to miss one occurrence of the text.

4.3.2 Text selection

The text selection is the most complex feature of the interface to implement. The basic idea is that the desired text part is searched, first word is pressed about a second so that selection begins, and end selector is moved to end place. Unfortunately, getting the right outcome is not so simple.

First the problems are caused just by the text identification that has been mentioned in the previous section. Because the text identification does not necessarily find words as they are, there is need for intelligent character substitution acceptability. This unfortunately poses another problem. Since the substitution of characters is used, the words are not how OCR sees them and that information must be stored separately so that more information can be searched about last word from fMBT.

Bounding box corners must be figured out from last word. fMBT returns only the top left corner of the bounding box with text dump so the ending point must be searched for some other way. This is done by searching the last word separately, and comparing bounding boxes top left corners from all alternatives.

When the data has been gathered, a choice can be performed by pressing of the first word of the text for a second and by dragging the end selector to its place. This is only more difficult than one would think.

In the dragging of the selector the first problem is a starting point. Even though image position can be asked from fMBT and it understands to ignore the transparent parts of the image in the comparison with suitable parameters, the vertical coordinate of the starting point is not directly in the area of the picture. In addition, the ending point is not bottom right corner of the bounding box of the last word. The end point position depends on the size of the text, and whether selectable text contains one or more words. After the move of the selector, one must check that the selector is moved to the right place.

In a bad case the end of the text is near the edge of the screen in which case a part of the selector goes outside the display. In this case, the selector can't be found, the operation is canceled and it is unknown whether or not the operation was successful. Preparing for this kind of a situation must be made so that scrolling will add targets from the right bottom corner the screen.

However, most likely the selector did not succeed in choosing the whole text so the

selector has to be moved again. If the selector stops near the end point, the position of the end point must be moved or the selector will not move any more. Empty lines in a text, can get the selector to stuck, and in this case the end point must be moved temporarily to lower.

Even if all of these are taken into account, text selection functionality is not 100% reliable. This is due to the fact that the success is affected by size of the text, the starting and ending point, as well as any spaces and blank lines. One failed result is shown in Figure 4.5. The correction of this and other mistakes often breaks some other working situations which creates a never-ending chain of error and correction.



Figure 4.5: Unsuccessful selection. The text selection in the image had to begin from beginning of the word “pellentesque” and end to word “fermentum.” but the selector did not succeed in choosing the dot and stayed between these two states in the image until a counter interrupted the execution after the tenth time.

4.3.3 Problems of the image recognition

The image recognition can get control parameters in fMBT, such as a color depth and opacity which can be used to control the precision of the image recognition. The interface is based on the premise that similar adjustments does not need to make but this has to be rejected. Android contains some transparent images which is why the texts behind the selector are partially visible for example. For this reason the image recognition cannot be always tell to search with really accurate values, particularly excessively accurate a color depth often causes the failure of the search.

A suitable regular value has been searched for color depth, but this is rejected. Android contains many closely spaced color values of gray which cause faulty identifications with even a small decrease in precision. The corresponding parameters were added to an interface

because of the usability.

4.3.4 Menu selection

Android menu button has a fixed-looking icon that most often consists of three gray squares. A problem is usually surrounding background whose color varies. So that the background could be left out in the search, it should be marked transparent in the search image but this causes the second problem. Because the icon consists of the same in color parts and there are no other individual parts, any large enough area of the same color cause a false recognition. Furthermore, those three squares are not always gray, which will cause need of many different menu images for recognition alternatives. Google's own programs contain at least alternatives shown in Figure 4.6. Fortunately in Android it is possible to implement opening of the menu in another way.



Figure 4.6: Four versions of the menu button. From left to right: Gmail, Gallery, Google Play, and Google Music [6].

Key pressings can be sent through the debug bus to the Android. fMBT contains the ready-made commands for the most general keys and other keys determined by Google can be sent with a general key pressing function. By using 'pressMenu' function to open a menu, menu can also be opened when there is no separate button for it. In this way the opening of the menu succeeds regardless of the background of the icon.

4.3.5 Text writing

In the writing of the text, the biggest problem consists of characters which do not belong to the ASCII character set. When a character is sent to fMBT, and it can't process it, the character causes usually blinking of the virtual keyboard. Android also supports Unicode characters but their use through the USB bus is difficult and it is not clear if it succeeds through fMBT. When writing this Unicode characters can be given to the interface, but they are printed as question marks on the display.

Some special characters, such as backspace and tabulator, do not work directly with writing function in fMBT. These two characters often have special marking in string, which are `\b` and `\t`. Many of these do nothing but some of the characters, such as a backspace, are causing the previously mentioned the blinking of the virtual keyboard. These characters must also be given by using separate *pressKey* command to fMBT. This function can be

used to input pre-defined key pressings commands. For this reason it is the easiest to go through characters of the string one at a time and either command fMBT to write it or command fMBT to sent press command which corresponds to the right button.

4.4 Testing

Interface was tested with the real equipment, in so far as it was possible. Preliminary tests have been carried out on the TnT version, but all the properties have not been tested yet. LG Nexus 4 phone which is connected to the robot is used as device under test in TnT version. The fMBT version was first tested with Samsung Galaxy Nexus and later with LG Nexus 4 telephones. Both phones contains Android version 4.2.2. The methods of the interface were tested at development phase on Samsung phone and at final stage, when we got the LG phone, the interface was updated to work on both devices.

Initially, the interface functions were run individually, and their functionality was verified in individual situations. After this the interface was tested in practice by using models, which had been made by another person. Every time execution stopped due to an error, the reason for this was explored together and discussions were made to decide whether the error was in the model or in the interface. With this method, we removed logical errors from interface, which were caused by differences in the understanding of the plan.

In the testing the problems caused by the different versions of programs arose. Appearance of Google's programs is almost the same in different versions, but even small changes are causing problems. A good example of this is little icons that look the same to the human eye but to the comparison algorithms they are different. These icons are often used for state verification, so they must be replaced with new ones. In practice this process must be performed manually which is slow to a large number of icons. The process often proceeds so that screenshot is taken from screen and from it desired icon is cut with image editing program that understands transparency. Usually one wants to remove the background from the icon so the area bounded by the icon is cut from image more precisely and from the cutting a new image is created. This is because the image area should contain a minimum amount additional area in order to compare images to be faster. Eventually the image is saved in PNG format.

In all the programs which utilize the network connection, the situation appeared where connection to server was disturbed. This will usually cause interruption in test execution at an error situation, because display does not contain what it should. An example can be seen in Figure 4.7.

Preparing for the above situation is difficult, but it could be solved by pushing the "Retry" button if the execution end to the state failed. After pressing this button, the state is tried to execute again. If the button is not found, the situation would be interpreted as a mistake. Another way would be to make this situation its own state and continue execution after this.

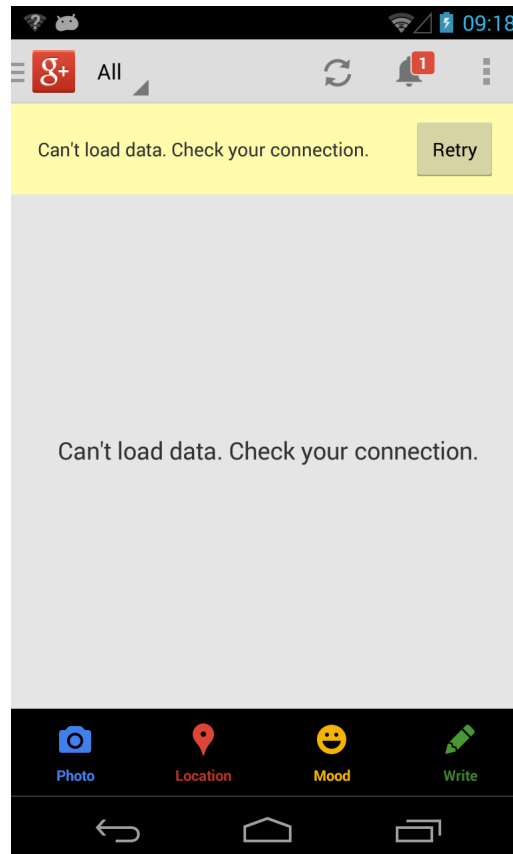


Figure 4.7: Connection problem interrupted the test. The figure shows the Google+ program in the situation where the connection to the server fails. The image should show the program's main screen, where the latest posts should be seen, but it was replaced by an error message and "Retry" button.

5. CASE STUDIES

Debian Linux 3.0.2.4 computer was used as test equipment. Google ADB drivers were installed on it and Samsung's Galaxy Nexus phone was connected to it using USB. The phone later changed to the LG Nexus 4. Android version 4.2 was installed on both phones. The exchange of the phone caused some problems because the phones contained different versions of the programs and some of the icons were different. This caused problems with compatibility of the models, so both phones and modeled the software was updated to the same versions.

Since this is model-based testing, a test execution path is not entirely predictable, even though with the same settings the sequence of events is repeated. By changing settings, the execution order may change radically.

A common problem was too fast command execution in fMBT. When the view was changed, an attempt was often made to read the text on it before exchange animation get to finish. This causes screenshot to be taken in some transitional stage which example is seen in Figure 5.1. When the target is searched from it, its location will be in the wrong position.

The second problem, which appeared in the tests, was connected to the interface. The coordinates had been already specified to the interface to be given with Python's "tuple" data type. This proved to be a bad decision because when tests were performed continuously, the users tried to give the coordinates x and y in different parameters. This caused the failing of tests because the problem always came out only when the execution got to the specific line which may have taken time.

5.1 Gmail

One model was designed to a Gmail program. Gmail is program that is specialized to use Google's Gmail email service. It is different from the normal email program by connecting to other services offered by Google, so the sender's picture will be displayed along the message if the such one has been connected to the account. This is seen in Figure 5.2.

First of all, fMBT will always start the Gmail from icon, but after that what is happening may vary. At this point, the program may be shut down and restarted. Test executes the most common functions. In the test, a Google account and folder are changed, a message is written, an e-mail message is sent, messages are marked important, notes are removed, messages are selected, selections are removed, and messages are browsed.

The test changes the user in use and after that it varies the filtration in use such as a folder or the marked messages. An examples of this are "Priority Inbox", "Starred" or "Important".

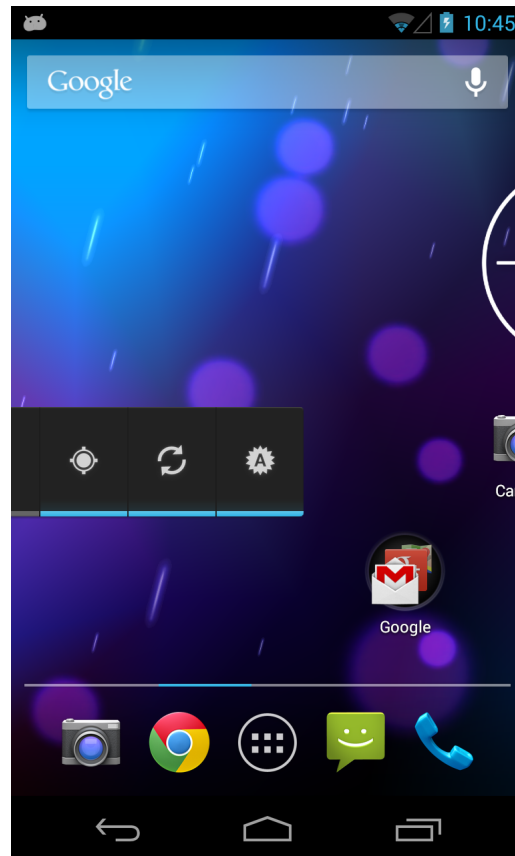


Figure 5.1: *Too quick state update.* In the image, desktop is moved from the start screen of Android to other by wiping to the side. Because the device which gives commands is independent from the test equipment, the device can give a new command before effects of the previous command has been totally executed. Following command is often the so-called state updating which means taking new screenshot in fMBT. In that case fMBT will think that it will be in the state shown by the image even though in reality the display is in the right state. The problem has been corrected with a separate "refresh" command which at the moment stops the execution for the moment.

The test chooses messages from selected folder and watches them by browsing. Messages are marked as important. These notes will also be removed.

During writing of the messages, receiver is chosen from test group. A test text is written as a subject and message. The message is either sent or its transmission is canceled after this.

From the point of the modeling a problem exist in Gmail and in other Google's applications, that some functionality may become completely different in the next version. The change may be the mere change of the background color of the icon on its smallest and the change of the operation of the choice box at biggest. Merely color change of the background of the icon often caused the fact that the models had to corrected by taking the new screenshots and by cutting the icons again.

Gmail has text fields which contains the purpose of the text field as written with usually light gray text. These texts are used in models for the selection of the text field. The cursor usually causes garbage characters appearance to the end of the word, but in difficult

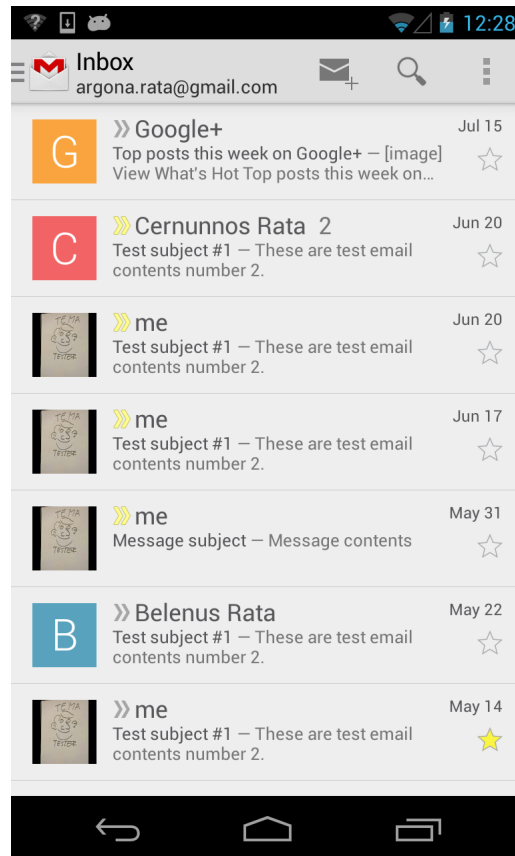


Figure 5.2: Gmail program. The image shows messages, which the test person received, and some of them was send by itself.

situations right characters are changed to another characters. In the interface the garbage characters in the end are ignored because OCR does it sometimes anyway, but preparing for the change of the right characters does not succeed easily. The situation in question is usually prevented with the press of display in model which removes the cursor from blinking in the disturbing area. When answering to the messages, the cursor is also moved away from a text field to be examined at first or it disturbs the identification of the text. After the transfer execution is still stopped to wait that the icon of the pointer disappears from disturbing.

The use of OCR is worth optimizing also so that all text to be searched for will be searched before more text is written on the display. In addition to the fact that the searching of the text gets faster, it reduces the finding of the extra text from the picture. An example of this is "Compose email" phrase which appear during writing of the message. If the search of the text is performed so that all the text fields are empty, the statement will be found without problems. If the search is performed after the receiver and the subject have been filled, extra characters will appear between the words from nothing and it ruins the search. Studies did not found clear reason for this.

Gmail's test revealed a bug, whose origin could not be traced. When pressing the 'send' button with the commands of the interface, the button may often get to stuck. After this the

button does not function any more even by pressing it manually, even though everything worked as it should before the command was issued. The problem is interesting, because it is not certain to happen, and everything may function normally. The error is difficult in the sense that shutting down the program does not always help, and restarting the device is sometimes required.

5.2 Calendar

Calendar is Google's calendar program, which allows one to keep track of events and view them in different views. Calendar program begins in the same way as Gmail. In the main view daily, weekly and month view is changed. The month view can be seen in Figure 5.3. In the program new events are created and some of them are removed.



Figure 5.3: Month view of the Calendar program. The events are shown with blue bars by the right day, but the color can be adjusted for each event separately.

When testing, all the main views will be first gone through by changing between day, week, month, and a event list view. The test usually tries to create a new event after this. When creating the event, the test inputs generic name and location of the event to the text areas and sets desired a date by selecting the year and the day from the calendar.

Calendar program will also have problems with the OCR. In the program, when it wants to set for a day or time, the popup window appears and from it one can set by pressing

from the right places. The problem is that Tesseract fails to recognize all the texts on any algorithm from the popup window, and that causes trouble to find right pressing position. The problem would be solved if the image will be limited to a pop-up window area. This is possible in fMBT, but it would require adding a new parameter to interface and it would be implementation dependent.

In the version 201305280 of the program, the operation has changed so that the year will be set by pressing the current year and by browsing the list through. Date is set by rolling calendar months, and pressing the desired day of the desired month. The time of day is set by choosing the desired time from the ring which resembles the face of the clock by adjusting hands. Clock face, which is shown in Figure 4.3 is problematic because fMBT cannot recognize all numbers by any text recognition algorithm, so setting the time is difficult.

It is nearly impossible to get minutes right mechanically because the minutes are written with intervals of the five in the view. Other minutes should be set by turning the hands of the clock to the right position. Programmatically and with the robot it is difficult to make curved dragging because both support only the direct drags. Even the direct dragging could be used to choose the right minute but the determination of the end position of the dragging is difficult, because the hand does not react to short dragging.

5.3 Google+

Google+ is program developed to use Google's social networking service which has the same name. The program can be used to post messages, pictures, users position and mood. The posts can be published for example to the friends, family, all groups or to some other group. In Figure 5.4 the messages published by the group "Testing" are seen.

Google+ was tested by sending a post to test group. The program starts, changes the group, writes and sends the post and creates the picture post. The test usually begins by starting the program and by changing the community group to a test group. After this the writings and sending of test posts and their cancellation begin.

In the test the writing of the message starts from the writing icon. First the program changes the used group for a test group even if it were already in use. After this a message is written to post and after that the post might be sent. The test also adds pictures to some posts. In that case the test will press the photograph icon on the main menu.

In the earlier version of the Google+ adding images to post is difficult with fMBT, because the picture is added by pressing a small transparent choice icon. It is difficult to find the icon reliably since its color values are affected by color values of the picture at same place. For this reason, fMBT's image search does not work with it. In version 4.2.2.55951696 the selection is changed so that the choice of pictures will not be made with a separate button, but merely the pressing of the picture is enough.

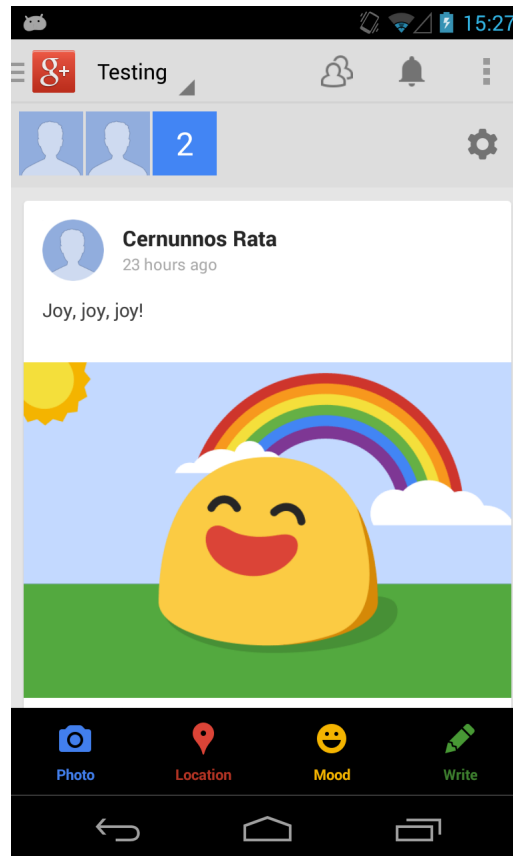


Figure 5.4: Google+ application. The image shows the publications of the "Testing" a group. In the post which is seen in the image, person has send "Mood", but the photograph would look similar in the posts.

5.4 Hangouts and Google Talk

"Hangouts" program was originally a "Google Talk" program. It is used to discussion between user and his/her friends or his/her friend groups. Starting screen of the program is shown in Figure 5.5, where one can starts conversation. The program features have remained the same but a few changes in the models had to be made. For example, message box during writing no longer contains text "Type message", but rather "Send a message", and the program icons were changed completely.

The program was tested simply by starting it, turning it off, sending messages, and canceling the send. The program starts and lists the friends that you can talk to. The discussion starts by pressing the friend from the list. After this the test writes the message and depending on the situation, either sends it or cancels the sending.

The consequences of the canceling of the sending of the message formed into the problem in the tests. The program in fact remembers the text that was going to sent, when one cancels the message sending. When next time the writing of the message is begun, the test will not found "Send a message" text on the screen but rather previously typed text. This could be passed by searching for the "smiley" icon next to the text box and on

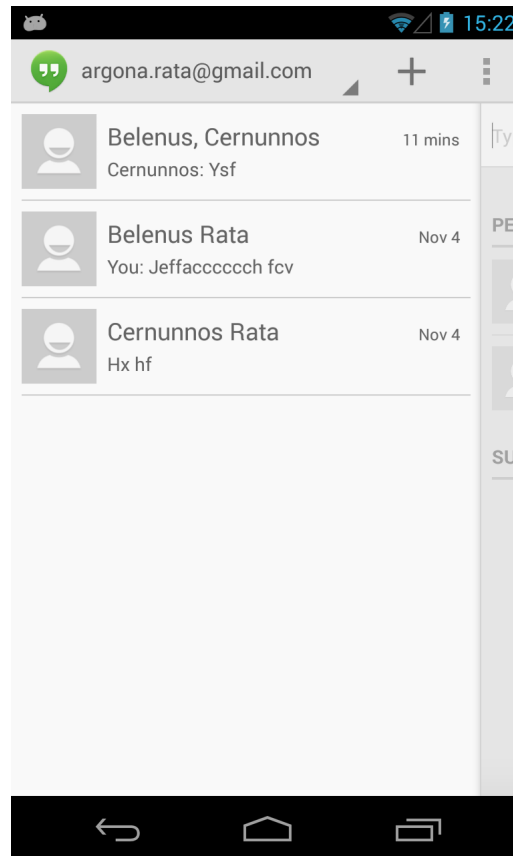


Figure 5.5: The main view of the Hangouts program. The figure shows the used discussion groups of which the first one contains more test persons and the lower two are the conversations between the two person.

the basis of its position the writing could be activated. After this the problem will be the clearing of the writing. The easiest way to do this is typing backspace character, but the problem is to know how many times it is needed to be typed.

6. RESULTS AND DISCUSSION

The interface has been used to execute tests and its operation already carry out its purpose. Gmail, Google+ and Hangouts test was executed so far that state combination coverage could not risen any higher. In contrast it is nearly impossible to make the Calendar program with the current version get through to the end of the test. This is caused by the above mentioned problems with the text recognition.

The adaptation interface makes possible to later add new implementations that can be used to test the new devices. Models most likely need to update the their icons to support the new devices, but otherwise the interface is designed so that implementation can be changed easily. Most of the interface works smoothly, but the text selection contains so many special situations that it is impossible to ensure its perfect functionality.

The slowest functions are all connected to the text recognition. This is due to the fact that a reliable enough recognition requires several different algorithms and each of them must process the image, which is slow. Furthermore, the amount of the text in the picture affects this directly.

fMBT version 0.11 still contains error and because of it text selection may not work. The error associated with word search from an image and it does not return all word positions on the screen, but only the first one. For this reason, the text selection does not always manage to find the end position. This error has been corrected in fMBT version 0.12.

The speed of the adaptation interface could be improved by utilizing internal caching of the OCR and image recognition at least for the fMBT version. fMBT keeps a screenshot in memory until the user explicitly update it. When a user does not update a image and performs the text recognition to it, fMBT will keep the data of the words that have been found in the memory. If a user is asking the information of a word again, fMBT will now return the information directly from the memory. fMBT will operate in the same way, when user searches some sub-image from display.

However at the moment the adaptation interface updates the screenshot with every command so cache does not work. This also causes the fact that the results of the text recognition may vary even if the commands were executed in succession. In addition, the more text recognition algorithms are used, the slower their execution is.

The adaptation interface is working with fMBT, but the TnT version is still a work in progress. The need was to execute test using the interface and this is possible with current version of the interface. However the remote connection to robot caused extra difficulties at first when the remote connection system operated with Windows and fMBT should execute on a Linux machine. The problem was solved by bridging the connection in the Windows

machine and by connecting the Linux machine to it.

The requirement was to implement a general interface that allows easily to control several test devices. This was achieved moderately. Interface functions are common to all implementations, but parameter changes had to be made in them. These parameters are mainly implementation dependent and they do not affect any other implementations in any way. Such parameters were "opacity_limit" and "color_match" in fMBT version. The first one affects limit value of the transparency and the second affects how near the color values needs to be to be accepted as the same. TnT version does not understand these parameters, because the comparison of images is performed with shapes, not with the pixels. This is bad for generalization of the interface, because these parameters were not possible to change internal constants of the implementation.

7. CONCLUSIONS

There are two versions implemented from the adapter interface. One has been designed to operate with fMBT and the other one with the robot adapter TnT. Both interfaces are usable and fMBT version is used with the models. The interface is usable with test systems and it can control test machine rather well. The implementation has been successful although it was difficult in some parts.

Some of the problems were common to both implementations. Such were the analysis of getting to the end of the scrolling and it were connected to the comparison of similarity between two images. This was solved using histogram based evaluation. Another common problem was caused by OCR. Although both implementations use different text recognition algorithms, their problems are very similar. When a text is searched, its middle position might not be over text, when the searched text is only couple lines long. The word order will change when some of the words contain higher letters. The letters are wrongly recognized which applies especially to the country-specific letters.

Text writing is difficult with TnT version. This is due to the fact that a virtual keyboard has been divided into several parts and one must switch between them during typing. In addition, the marking of the change button of the parts is not standardized.

fMBT version works well except the selection text. The problems of the text selection are mainly caused by Android's selection tool, which is designed for humans, not for machines.

The improvement of the text recognition succeeded well compared to fMBT. fMBT's wrong word order was successfully repaired and searching of the text was improved by changing the characters mixed by OCR.

The image recognition causes its own difficulties in fMBT. Because the searching of icons takes place with the pixel comparison, even a small color mistake can cause the failure of the identification. This problem occurs especially with icons which contains transparency when the background mixes with the colors of the icon.

It is difficult to open the menu on fMBT. The problem results from the fact that in the programs the icon of the menu varies a little. Usually the menu icon consists of three gray squares, but the background depends on the program. For this reason the search cannot be made with the one same picture

In fMBT the difficulties of the writing of the text are connected to the external characters of the ASCII character set and fMBT does not support them. Instead of these characters a question mark, which is a replacing character for them in Python in Unicode 2.7, is typed on the display.

The scrolling functions fairly well regardless of the fact whether menus or a text are scrolled. In some situations the interface may try to scroll a few extra times.

TnT robot adapter has been tested a little and the preliminary tests are promising. For example by adjusting parameters a little, the scrolling works as well as on fMBT, even though the pictures will come from the camera which causes some distortion. Also the touches of the display function correctly even though the positions are given in millimeters.

The case study was conducted to a few Google's programs. Some of these programs were Gmail, Calendar, Google+ and Hangouts. Models of these programs which implement the basic functionalities of the programs, was tested and the problem situations and difficulties were reported. With the programs which utilize the network connection, additional problem was caused by errors in server connections.

In general, the interface implementation was successful, although it was difficult in some respects. The interface should be relatively easily modified to work with fMBT's caching feature that keeps texts of the image and searched images in memory. Different things could be searched for faster from same image using cache. At the moment the interface will always fetch a new screenshot with every command itself if it needs it. This also causes the fact that the commands which utilize, for example, OCR do not necessarily produce the same result with consecutive executions.

fMBT and some Android devices support the so-called "views", which are an easy and quick way to process targets on display. Making use of this feature, the interface would be more efficient on some devices. All equipment, including both the test devices, does not support this feature, so this can't be used on all devices. The interface could first check if the device supports the property and if device supports it, the interface would use it. Otherwise general versions of the functions would be used.

A third version of the interface is considered. This version would use parts from TnT and fMBT interface. The idea is that the TnT interface for text recognition and pattern recognition are not open source, but rather they are commercial software which license is limited. The license bureaucracy of the pattern recognition causes the fact that pictures cannot be changed without a difficult process suitable to the TnT system. Furthermore, this complicates the external use of the interface if it is published under an open source code to public.

For this reason the version which utilizes the control interface of TnT, but uses icon and text recognition parts from fMBT, is worth implementing. In this way the realistic testing action offered by the robot system is retained and the search of targets from the display is made possible without expensive and continuous third party licenses. The cost of purchasing the robot from OptoFidelity does not leave.

Adding support for cheap robot which is made from parts has been considered. The parts of the model have been obtained from this kind of a delta robot but the robot would require much work before it even gets to work.

It has been in development proposals that TnT system is added directly as part fMBT.

In that case Optofidelity would get more market value to its system when there would be already support to some open system. In that case the threshold to the use of the system will become smaller when one is able to try the development platform freely before expensive investments. Furthermore, it is easy to make the ready models that have been made to work with fMBT to work with TnT in that case.

REFERENCES

- [1] 01.org. Fmbt. <https://01.org/fmbt/about>, 2012. Cited February 2014.
- [2] 01org. GitHub repository 01org/fMBT. <https://github.com/01org/fMBT>, 2013. Cited February 2014.
- [3] Charlie Clark. Comparing two images (Python recipe). <http://code.activestate.com/recipes/577630-comparing-two-images/>, 2011. Cited February 2014.
- [4] Mark Fewster and Dorothy Graham. *Software Test Automation: Effective Use of Test Execution Tools*. Addison–Wesley, September 1999.
- [5] Google. Android Debug Bridge. <http://developer.android.com/tools/help/adb.html>, 2013. Cited February 2014.
- [6] Google. Screenshots taken from different Google’s programs, Gmail, Gallery, Google Play and Google Music, 2013.
- [7] Google. tesseract-ocr. <http://code.google.com/p/tesseract-ocr/>, 2013. Cited February 2014.
- [8] Ibrahim K. El-Far and James A. Whittaker. *Model-Based Software Testing*. John Wiley & Sons, Inc., 2002.
- [9] Antti Jääskeläinen, Mika Katara, Antti Kervinen, Mika Maunumaa, Tuula Pääkkönen, Tommi Takala, and Heikki Virtanen. Automatic GUI test generation for smartphone applications – an evaluation. In *Proceedings of the Software Engineering in Practice track of the 31st International Conference on Software Engineering (ICSE 2009)*, pages 112–122 (companion volume), Los Alamitos, CA, USA, May 2009. IEEE Computer Society.
- [10] Antti Kervinen. AAL/Python. https://github.com/01org/fMBT/blob/develop/doc/aal_python.txt, 2013. Cited February 2014.
- [11] Antti Kervinen. fMBT – free Model-Based Testing tool Lesson 2: Test models and test generation. https://01.org/sites/default/files/documentation/fmbt-lesson2_0.pdf, 2013. Cited February 2014.
- [12] OptoFidelity. OptoFidelity HSUF One Finger Touch and Test System. <http://www.youtube.com/watch?v=nod0ph7uZB0&feature=share&list=UUxZMoLky0c7rMwKuTrNp6HQ>, 2013. Cited February 2014.
- [13] OptoFidelity. OptoFidelity Touch & Test®. <http://www.optofidelity.com/en/test-automation/optofidelity-touch--test/>, 2013. Cited February 2014.

- [14] Olli-Pekka Puolitaival. Model-based testing tools. <http://www.cs.tut.fi/tapahtumat/testaus08/011i-Pekka.pdf>, 2008. Cited February 2014.
- [15] Olli-Pekka Puolitaival and Teemu Kanstren. Mallipohjainen testaus ennen, nyt ja tulevaisuudessa [Model-based testing before, now and in future]. *Systeemityö*, 4, 11 2010. Available at <http://www.vtt.fi/inf/julkaisut/muut/2010/mallipohjainentestaus.pdf>, cited February 2014, in Finnish.
- [16] Secret Labs AB. Python Imaging Library (PIL). <http://www.pythonware.com/products/pil/>, 2013. Cited February 2014.
- [17] Rasa Sieberg. Mallipohjainen testiautomaatio: perusteet ja huomioita käyttöönotosta [Model-based test automation: basics and notes on adoption]. *Systeemityö*, 1:18–21, 2005. Available at <http://www.pcuf.fi/sytyke/lehti/kirj/st20051/ST051-18A.pdf>, cited February 2014, in Finnish.
- [18] Ray Smith. An Overview of the Tesseract OCR Engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 629–633.
- [19] Ray Smith. Tesseract OCR Engine. What it is, where it came from, where it is going. <http://tesseract-ocr.googlecode.com/files/TesseractOSCON.pdf>, 2013. Cited February 2014.
- [20] Tampere University of Technology. TEMA model-based testing. <http://tema.cs.tut.fi/>, 2012. Cited February 2014.
- [21] Jeff Tian. *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. Wiley-IEEE Press.
- [22] Antti Valmari. The state explosion problem. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528, Berlin, Heidelberg, 1998. Springer.