國立臺灣大學電機資訊學院資訊工程學系
碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

基於隱馬可夫模型之圖形介面程式測試腳本自動化
GUI Test Case Generation Using Hidden Markov Model

莊冠駿
Chuang Kuan-Chun

指導教授：施吉昇 博士
Advisor : Shih Chi-Sheng, Ph. D.

中華民國 99 年 7 月
July, 2010

國立臺灣大學碩士學位論文
口試委員會審定書

基於隱馬可夫模型之圖形介面程式測試腳本自動化
GUI Test Case Generation Using Hidden Markov Model

本論文係莊冠駿君（學號 R97922006）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 99 年 7 月 23 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

（指導教授）

系 主 任 呂育道

GUI TEST CASE GENERATION USING HIDDEN MARKOV MODEL

BY

KUAN-CHUN CHUANG

Department of Computer Science and Informantion Engineering,
National Taiwan University, 2010

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Department of Computer Science and Information Engineering
in the College of Electrical Engineering and Computer Science of the
National Taiwan University, 2010

Taipei, Taiwan

Acknowledgments

I am grateful to my thesis advisor Prof. Chi-Sheng Shih. His deep and wide knowledge, patient training, and bounteous discussion guide me the direction of my research. The most important of all, he taught me how to research correctly and precisely. This thesis couldn't be accomplished without his support. I would also thank to Prof. Tei-Wei Kuo, Prof. Ray-I Chang, Prof. W.-S. Liu, and Prof. Shih-Hao Hung for kindly agreeing to be the members of my committee. My work has been made better by their insightful and helpful comments.

I would like to thank my colleagues in the Embedded Systems and Wireless Networking Laboratory, Hsing-Yu Lai, Yu-Sheng Liao, Wei-Chih Chen, Yi-Hsiang Huang , Hung-Hsiang Chang, Che-Wei Kuo and others was not included in the list. They gave me a lot of helpful advices on the study. We had pleasant time within past two years. Last but not least, I would like to thank my family and girlfriend for foregoing many of the comforts of life and allowing me to focus on my study. Without their support, this achievement would not be so sweet and smooth.

Kuan-Chun Chuang

Taipei, summer of 2010

摘要

正確性測試是軟體開發中的必要步驟。為了達到夠高的程式碼涵蓋度，測試人員必須產生夠多的測試腳本來執行程式。當待測程式的複雜度增加時，過多的測試腳本會造成維護的困難。測試自動化藉由自動產生測試腳本測試，解決了高程式碼涵蓋度以及腳本維護間兩難。

在測試自動化中，要自動產生使用者圖形介面的測試腳本是一件很困難的事情。因為每個畫面都有太多可能的執行步驟，當測試腳本的長度增長時，所有可能的執行步驟會呈指數成長，造成腳本過多無法全部執行的問題。另外，我們也很難去決定哪些測試腳本是對測試比較有用的，因此無法減少測試腳本的數量。

在本篇論文中，我們設計並實做了一個能夠模擬使用者操作圖形使用者介面的測試腳本自動產生框架。我們設計了一個使用者電腦互動模型來描述不同類型的互動方式。我們結合此模型以及既有的測試腳本，來產生出能夠模擬使用者操作的測試腳本。

經由實驗證實發現，透過此篇論文所提架構，能夠找到既有使用者圖形介面測試自動化中沒找到的錯誤，也能用較少的測試腳本就達到更高的程式碼涵蓋度。

關鍵字—圖形使用者介面測試自動化，圖形使用者介面測試腳本自動產生，使用者行為模型，隱馬可夫模型，圖形使用者介面，人機互動

Abstract

Testing for correctness is an essential part in software development cycle. To achieve high code coverage rate, testers have to generate much test cases, which produces maintainability when number of test cases become huge. Test automation aims to resolve this conflict by automating whole testing process.

GUI test case generation is a challenging issue in test automation because the number of possible execution path grows exponentially with test case length. In addition, it is hard to determine whether the test case is meaningful or not.

This research aims to design and implement a test case generation framework which take user computer interaction into account. We design an user computer interaction model to classify user computer interactions under different contexts. We combine interaction model and test cases to simulate user behaviors.

Keywords - GUI test automation, automated GUI test case generation, user behavior model, Hidden Markov Model, graphical user interface, human computer interaction

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Testing for correctness is an essential part in software development cycle. From creating individual components to integrating components into a complex system, testing is interweave in every software development step. Moreover, it repeats itself when any of the components are upgraded or new features are added to the system.

Code coverage rate is a critical metric when designing test procedure. Since bugs could occur in any step of software execution, the test cases should verify through as many states as possible. To achieve this goal, the test team needs to generate and maintain enormous number of test cases to cover different execution paths, which is time consuming when software becomes sophisticated[2].

Test automation[3] aims to deal with the trade off between improving coverage rate and reducing testing overhead. It automates the whole testing process, including test case generation, test case execution, and test result validation.

Although test automation aims to reduce cost of testing process, current approaches are not fully utilized in industry, especially in domain of graphical user interface(GUI)[4]. Although GUI enhances usability, it also increases number of possible inputs, exposing software to more complicated state handling and higher probability of error. Traditional test automation cannot model such system containing huge number of states

properly. The emergence of HCI research and touch screen also open up possibilities of different input methods, such as mouse gesture and touch gesture. The traditional test automation can not adapt its model to new input method easily, which make testing even harder.

Test case generation is one of challenges in GUI test automation. On one hand, method to traverse state space should be carefully chosen, or the number of total test cases can be too big to run through. On the other hand, even though we can generate test cases with feasible size to run through, it is still hard to generate meaningful test case in perspective of real software user behavior.

Although test automation and testers really help for application under test(AUT), testing system by users is still a necessary step in development process. One of the reasons why user testing is necessary is users operate software with their own habit and characteristic. User who is impatient or under emergent context may click the same button repeatedly in short time if the expected result does not occur. When he/she is submitting data to system, duplicated submitting may cause data corruption if the duplicated events are not detected. User who is used to control GUI application with keyboard shortcuts may also get problem if he/she issues shortcuts too fast, because time to display new page may be longer than user to issue next shortcut, which make application fail to handle incoming shortcut when still responding to previous one. Habit of operating an application varies across different users and different contexts. These dimension of varieties are hardly explored or modeled by quality assurance team because the varieties not only come from application design, but also user knowledge background and its emotional status. In existing test case generation framework, the user perspective is almost ignored, or divides users into only two types: naive user and expert, which can not be extended to other user types [4, 5].

The researches in human computer interaction provide part of solution, This research domain has last for more than twenty years, which aim to describe human and

computer interaction via formal method and classifying different types of behavior. But to the best of our knowledge, there is no research about producing different types of human-liked behavior base on human behavior models. We need to deploy the behavior generation procedure by ourself.

When human and computer interaction becomes part of model for test automation, it can generate more meaningful test cases behaved like a user, and make testing contains perspective from user view in earlier software development stage.

This thesis aims to study the impact of user behavior to application, and to generate test cases based on user behaviors so as to discover such type of errors in software development phase.

1.2 Objective and Contribution

In this thesis, we propose a test case generation framework for GUI applications to take user computer interaction into account. The objectives are listed below.

- Design and implement a test case generation framework which involve user computer interaction. We propose a unified model to describe different types of user and computer interaction, which is compatible to many existing test automation frameworks and preserve capability of existing test automation framework at the same time.

- Design user computer interaction model to classify different types of interactions under different contexts, and adapt test cases to this model to obtain user perspective.

- Take iNuC [6] as target application under test(AUT). This example demonstrates a complete test case generation process and shows that the test case generation

framework can be combined into existing test automation framework and also be as a standalone framework. For detail information about iNuC, see 2.3.

The contribution of this thesis is to allow generating user behavior for test automation. This ability provide another perspective for test automation, which is interpretable for user-involved application. This ability makes testing with different types of user characteristics, which is usually done in real user testing, be achieved in a more automated way before software release. This achievement can reduce development cost because of detecting bugs in earlier development stage.

1.3 Organization

The remainder of this thesis is organized as follows: In chapter 2, we present the background of test automation, graphical user interface(GUI), and the example application: iNuC. Then current research status of GUI test automation and its comparison to this research are shown.

Chapter 3 introduces models related to user behavior and user behavior classification. Base on these models, we propose a model to describe different types of user computer interactions in the following chapter.

As mentioned above, chapter 4 defines user computer interaction hidden Markov model, which describes interaction between user and computer. We list two different types of interactions at the end of this chapter to demonstrate the capability of this model.

Chapter 5 presents the definition of personative test case, and shows how this type of test case can expand expression power of existing test case. Algorithm of generating personative test case is than presented to show the detailed steps to generate test case.

Chapter 6 demonstrates the process to generate user behavior involved test cases. The user behaviors include several types of user computer interaction. The demon-

stration use iNuC as target application under test and nurses in National Taiwan University Hospital as target users. The result shows that the proposed test case generation mechanism detects bug which tradition test automation not finds. The result also shows that the proposed test case generation mechanism achieves higher code coverage rate by using less test cases than related work.

Chapter 7 concludes the thesis and discusses future work of this thesis.

Chapter 2

Background and Related Work

2.1 Functional Testing and Test Automation

Software testing is the process to verify the correctness, usability, and performance of an implemented software system. In the thesis, we are concerned with the testing for correctness of the software, which is called functional testing. The other testing technology are out of the scope of this thesis and can be found at [7, 8].

2.1.1 Functional Testing

Functional testing is the minimal requirement in software testing. It aims to detect bugs in *application under test*(*AUT*). The bugs are which obey the expected behavior of system requirements or get error under invalid conditions such as unexpected input.

Methods of functional testing can be classified by two dimensions [3]. The first dimension is level of knowledge an application is implemented. Testing with full knowledge of internal implementation is called *white box testing*. Tester of white box testing has full access to application code, including data structure and algorithm. API testing and code coverage are typical types of white box testing. Testing with no implementation knowledge is called *black box testing*. The black box testing use application requirements or specifications of AUT to determine valid input and output pair. We

can also combine different levels of implementation knowledge, such as doing black box testing and measuring code coverage at the same time.

The second dimension is the scale of the AUT. The scale ranges from the most basic component to the whole system which interacts with users and/or environment. Testing for the most basic component is called *unit testing*. This type of testing tests a single unit at a time, such as a single class or library. Unit testing ensures that each basic component works independently, but it does not guarantee interaction between different components can work as system required. The *integration testing* comes to verify interaction between different components combination. This type of testing may start from testing interfaces of components which contain dependency relationship, and scaling up to test whole components in given AUT. After integration testing, *system testing* tests if the AUT follows system requirements. The system testing combining all component required by AUT, such as third party systems, user input from real human computer interface, and input from target environment of AUT.

These two dimensions are not fully independent, such as unit testing is usually done by white box testing, but these classifications still provide useful insight when designing test methods. The first dimension shows how testing data is generated. When using white box testing, we can generate testing data from boundary input because we know the limitation of internal structure. When it goes to black box testing, we can concentrate more on meeting system requirement. The second dimension provides a testing hierarchy, which begins from unit testing, and ends to system testing. Different level of testing hierarchy are responsible for different scope of AUT, which is very similar to the construction of a system.

Although methods to perform functional testing varied, there is a typical procedure for testing [9].

The first step to do testing is establishing a *test plan*. Test plan documents a systematic approach to test AUT. Bases on system requirements, it specifies four items to

accomplish testing procedure: features to be tested, test approach, test requirement, and environment to execute and verify testing. Features to be tested are a set of functionality in AUT, which can include feature of specific library, module, or interface. Test approach introduce method to execute testing of given feature, such as testing and verifying result by human labor, recording testing steps and replaying them for each testing round, or using test automation technique to automatically generate, execute, and verify AUT. Test requirement specifies criteria to meet testing goal. A typical test requirement can be traverse each page in *graphical user interface*(*GUI*) at least one time, or cover all code branches in features to be tested. Since environment to execute testing can affect test result, we also have to provide environment information, such as free computer memory size or network quality if AUT is a network-based application.

The second step is to generate *test cases* to meet test requirement of given test plan. A test case is a specific procedure to operate AUT to achieve a given goal. Take testing word deletion in a text editor for example. One possible test case can be deleting word by mouse with the following step:

1. Double left click on the word to be deleted.
2. Right click on the selected word.
3. Select "cut" item in pop-up menu

We can delete a word by different procedure and thus generate other test cases which achieve the same goal. These similar test cases can be grouped into a *test suite*. A test suite is a set of test cases which achieve similar goal, or a collection of test cases which test a specific functionality in AUT.

After creating sufficient test cases to meet testing requirement, *test oracle* comes to verify the output of test case execution. Typically, the expected output of AUT can be gathered from system requirements, other program with the same functionality, or

human judgment. Test oracle generates expected input and output pair for AUT, and than executes test cases to see if AUT passes the test.

Once the test plan meets test goal, the test is closed and test report and other data related to test result should be generated.

2.1.2 Test Automation

Quality assurance team develops and executes test around whole software development cycle. When doing functional testing, there are three key issues as described in [3]:

1. Designing the test cases
2. Executing the tests and analyzing the results
3. Verifying how the tests cover the requirements

TODO: figure needed for three types of testing!!

In manual testing, all three items described above have to be done by human labors. In this case, test cases are wrote into a human readable document. The document describes steps to executing each test case in high level term, such as "click the save button" or "drag browse bar to bottom". The low level details of interacting with AUT can be left to the common sense of people who executed the test case. The test cases designing is time consuming because it take human to generate enough test cases which meet test requirements. It becomes even worse when software upgrades because we need to modify test cases to meet new requirements by hand. The executing, analyzing, and verifying of tests is also time consuming and boring because the tester, which is a human, must repeat the same procedure every time when testing AUT.

Capture/replay testing, which is the most popular testing method in GUI testing [4], automates the second issue as described at the beginning of 2.1.2: executing tests and analyzing its results. Like manual testing, capture/replay testing also require human labor to design and execute test cases, but when testers execute test cases at first time, they also capture the execution steps by recorder. When testing is issued again, testers automate the testing process by using *test harness* to replay test cases and analyzing the results. Test harness automates test process by executing test cases and combining test oracle to verify the result.

Although capture/replay testing reduce time spent in executing and analysing tests, there are still other problems left. The first is test cases still generated by human. The second is even though the tool replay recorded test execution steps, the records can not adapt to software upgrade because the interface may be modified, which invalidate the input from previous version.

Model-based testing aims to automate all three issues in testing process. It divides test case designing process into two parts. The first part is creating *abstract models* which model the AUT and/or its executing environment. The abstract models is much smaller and simpler than AUT itself. They focus on key aspects which meet the testing requirements, such as modeling page relationships in GUI and ignore detail logic of each click. After creation of abstract models, it generate test cases by using *test case generator*. The test case generator focus on generate enough test cases to meet test requirements from abstract models. In some cases, the generated test cases can not execute by test harness because they lack implementation information of AUT, such as the position of a button in GUI or input format of an interface. *Test script generator* converts test cases into executable *test script* by adding AUT related information into test cases. At the end, the test harness executes all test scripts and analysis the result. Most model-based testing tools also produce *requirements traceability matrix* to verify how the tests cover the requirements, such as code coverage.

Model-based testing is a suitable start point for test automation, but there are still many issues to be solved, especially in GUI test automation. We describe the challenge and current status of this field in 2.4.

2.2 Graphical User Interface

Graphical user interface(*GUI*) is a type of user interface which allows user to interact with computer by pointing device, such as a mouse. It is ubiquitous in today's software interface.

The most widely used GUI style is *WIMP* [10], which stands for four elements used for interaction: "window, icon, menu, and pointing device". WIMP style was first developed at Xerox PARC [11],and popularized with Apple Inc's[12] introduction of the Macintosh in 1984. This style uses a physical input device to control cursor on a screen, and displays output on windows. Each window represents a logical item, such as a document, or an software, which simulates experience in workplace. All commands available for each window is shown in menus. One of advantage of WIMP style is it allows non-technical people to operate computer, because WIMP style provide consistence know-how of operating different application.

The popularity of several widget toolkits that support this style, such as Microsoft Foundation Classes(MFC)[13] and Qt[14] also benefits WIMP to be prevalent. A widget toolkit contains a set of widgets for designing WIMP-styled GUI. A widget is a input/output element, such as a page, an image, a radio box, or a button.

Most commercial widget toolkits use event-driven programming as interaction model[15]. A toolkit handles event triggered by input device, such as a click on a button. A command is formed by combination of one or several events. Take word deletion in a text editor as mentioned in 2.1.1 for example, when application received a double click event, it highlights the word clicked. After that, if the application re-

ceived a keyboard event with deletion key, it deletes the word selected and reformats the document. A basic event in widget toolkits includes mouse down, mouse up, mouse move, mouse over, keyboard press, etc. These event can be combined into more complicated one, such as mouse click, double click, or drag and drop.

The event-driven programming is not necessary bundled for WIMP style GUI. Take mouse/touch gesture, which is widely used in touch screen, for example. The mouse/touch gesture embeds command into combination of several cursor movings, which is not a typical WIMP event because the command can not be shared across applications.

2.3 iNuC - Intelligent Nursing Cart

iNuC is a mobile point-of-care medication administration cart for preventing medication administration errors for the benefit of patients and care providers for nursing staff. The physical appearance of iNuC and its GUI pages is shown in 2.1. The functionalities are listed below [16]:

- Customizable and friendly graphic user interface.

  The user interface provide all information a nursing staff needs in medication administration. It filters out redundant information and display data in reasonable order. It also uses the concepts, terminology, and spatial arrangements that the users are already familiar and can modify the terminology to follow the policy in hospital.

- Scheduler and planner.

  It manages schedule, calendar, and information related to nursing staff and patients he/she cares. A nursing staff can add or modify patient's appointment

Figure 2.1: Physical appearance of iNuC, functionalities and their correspondent GUI pages.

through user interface, and leave scheduler to check if the modification will not obey any security rules.

- Clinical data integration and patient information visualization.

  A patient's vital sings, health records, prescriptions, care schedule plan, nursing notes, medication administration records and invoice records are stored in a central repository and retrieved when needed.

  The clinical information can not only be listed in table, but also grouped into sundry charts and graph automatically. These records provide information that guides nursing practice and inform medical treatment.

- Lockers and interlocking mechanism.

  iNuC provide interlocks by barcode and RFID technology to ensure he right doses of right medications are given to the right patients. When doing medication administration, only the drawers containing medicine of target patient are released, other drawers are locked in nursing cart.

- Monitor, alert, and notification mechanism.

  This mechanism provides capabilities for real-time monitoring, capture and analysis of events and condition that indicate the potential for occurrences, or actual occurrences, of errors, and issuing alerts and notifications to trigger error handling or prevention actions.

Although iNuC software is designed for nursing staff in medication administration process, the user scenarios are different from typical usage of software in office.

The first difference is operation in iNuC is highly interruptible by patients and other related people. Since each nursing staff has to take care of several patients during his/her duty[1], he/she can be interrupted by other patients when he/she is working for a patient. For example, if two patients taken care by the same nursing staff are located at neighboring beds, family members of one patient may discuss with the nursing staff about patient's status when the nursing staff is recording another patient's vital sign. It is an usual case to be interrupted, and switch pages of different patients back and force when using iNuC software in medication administration process. This property make response time of iNuC GUI more critical because when a nursing staff wants to see a patient's information, he/she expects the response of iNuC is as fast as his/her own reflection, or he/she may get impatient or feel frustrated about using iNuC. When the response time is not fast as expected, a nursing staff can even trigger same event multiple time in a short period, which may causes software bug if the event is not handled properly.

The second difference is interaction between software and hardware is embedded into medication administration operation, not triggered by nursing staff. Take lockers and interlocking mechanism for example. When a nursing staff wants to prepare medicine for a patient, what he/she needs to do is to select medication giving event for

[1]Take rehabilitation ward in National Taiwan University Hospital for example, nurses in day shift take six patients in average.

that patient on iNuC GUI, and the drawer contains the patient's medicine will open automatically. There is no button likes "open drawer of this patient" or some other controller to issue the open drawer command. When interaction between software and hardware is embedded into other operation, software has to handle all possible exception of hardware controlling. This property increases the complexity of iNuC software design.

The first version of iNuC is on final testing process when this thesis is writing. In this thesis, we use iNuC as the target AUT to illustrate our test automation technique.

2.4 Graphic User Interface Test Automation

In this section, we introduce current research status of model based GUI test automation. We describe testing procedures for GUI testing in research area first, and focus on test cast generation after that.

2.4.1 GUI Test Automation Framework

There have been several models to simulate GUI application to automate testing process. The most common one is state machine model [17, 4]. Each transition is an input event from GUI or its abstract class, and the state transition indicates change of GUI data, such as display new page or modify label. In state machine model, each test case can be mapped to a path/walk in state graph.

Memon et al. have created a test automation framework based on state machine called *event-flow model*[4]. The state in this framework is the collective state of each of its widgets (e.g. buttons, menus) and containers(e.g. frames, windows) that contain other widgets. The combination of widgets, containers and their properties(e.g. background color, text, etc.) form a state. The state transition diagram form a graph called *event-interaction graph*(*EIG*). In event-flow model, EIG can be automatically created by

reverse engineering technique during operating the AUT[18]. After creating the EIG, several methods are used to generate test cases[19, 20, 21] based on different testing goals. Event-flow model also uses state transition to create test oracle[22]. Since each state contains all property of widgets and containers, the state machine itself provide sufficient information to verify result of each execution step.

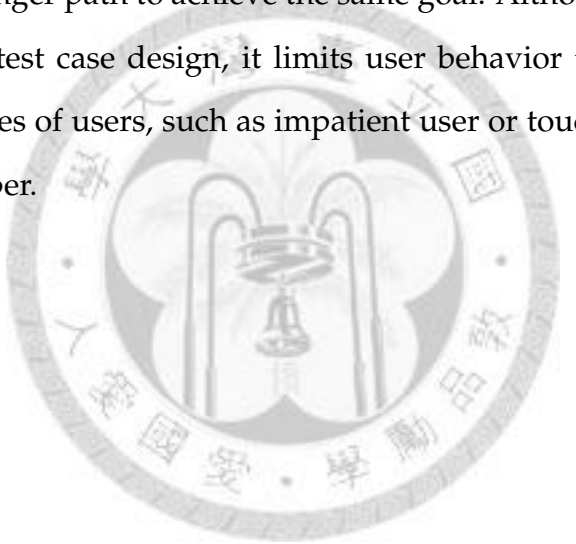
Event flow model take WIMP as its assumption of GUI. The GUI contains widgets and their containers as described in previous paragraph. The input event is recorded in system level, which means it records only outcome of event, not procedure to create event. This method models WIMP-styled GUI well because most of input events are mouse click, which is hardly produced in different ways from different users and a single input event is sufficient to trigger GUI state transition. When AUT contains gesture based operation, such as drawing tools or application on touch screen, outcome of an input event is produced by combination of several mouse moves. The event flow model has poor knowledge to combine a sequence of input events into a single event, and hence can not model non-WIMP-styled GUI well. In addition, event flow model emphasis on transition between GUI states, not how user uses the AUT. When GUI does not behaved like user expected, user may trigger other events to try to make things go on. Take transmitting data to server via heavy-loaded network for example, the user may click the submit button again if the GUI does not response in time. If the application does not handle duplicate transmissions right, the error would occurs. In event flow model, test harness does not trigger new input event until state transition is done, which ignores bugs in non-transition operation.

2.4.2 GUI Test Case Generation

Memon et al. have introduced several GUI test case generation technique, including off-line[19, 20, 21] and online[23] test case generation algorithm. The off-line algorithm

use AI planning technique to traverse EIG. Given initial and goal vertex, AI planning technique finds possible path from initial to goal, and generates test cases according to the path. The online algorithm starts from given initial test cases, and generate remaining input event base on *event semantic interaction*(*ESI*), which is a set of related events pair collected on runtime. The type of relation of events pair is predefined in ESI. The online test case generator than uses ESI to append event sequence which related to the original test case.

In addition to generate test cases from pure GUI state, David et al. have introduced a method to generate test case by simulating novice user behavior[5]. They used genetic algorithm to modify original test case, which is built by an expert user, to a new test case which take longer path to achieve the same goal. Although this work expand user perspective into test case design, it limits user behavior types into expert and novice user. Other types of users, such as impatient user or touch screen user can not be modeled in this paper.



Chapter 3

User Behavior Model and Classification

Studies about user behavior classification lie in many researches area. In multimodal data analysis [24], researchers detect users emotion by classifying user behaviors and/or their interactions with environment. They collect user behaviors by using camera, microphone, or other devices to sense how user behaves. After collecting user behaviors, they code behaviors into features, and classify the giving features into user emotion. The detection of user emotion can help application to be more context-aware. In E-learning system [25, 26], researchers classify different types of user by detecting browsing patterns of each user. The browsing patterns are usually collected from button or link clicks on web page, which indicate user intension of browsing . They use the clicking sequences to detect current status of user, and give customized feedback to that user.

Although classification technique varies with research domains, timing is a typical property considered in features classified [24, 25, 26, 27], because user emotion/intension is usually evolved by time, not by a single instant event.

We choose *Hidden Markov model*(*HMM*) as our model for classifying user computer interaction. HMM is a natural choice for modeling temporal data [28] because it embedded temporal information into stochastic process. In addition, the unobserved states in HMM can be unrevealed intension of user, which is suitable for modeling user behavior. In this section, we describe the definition, property of HMM, and its

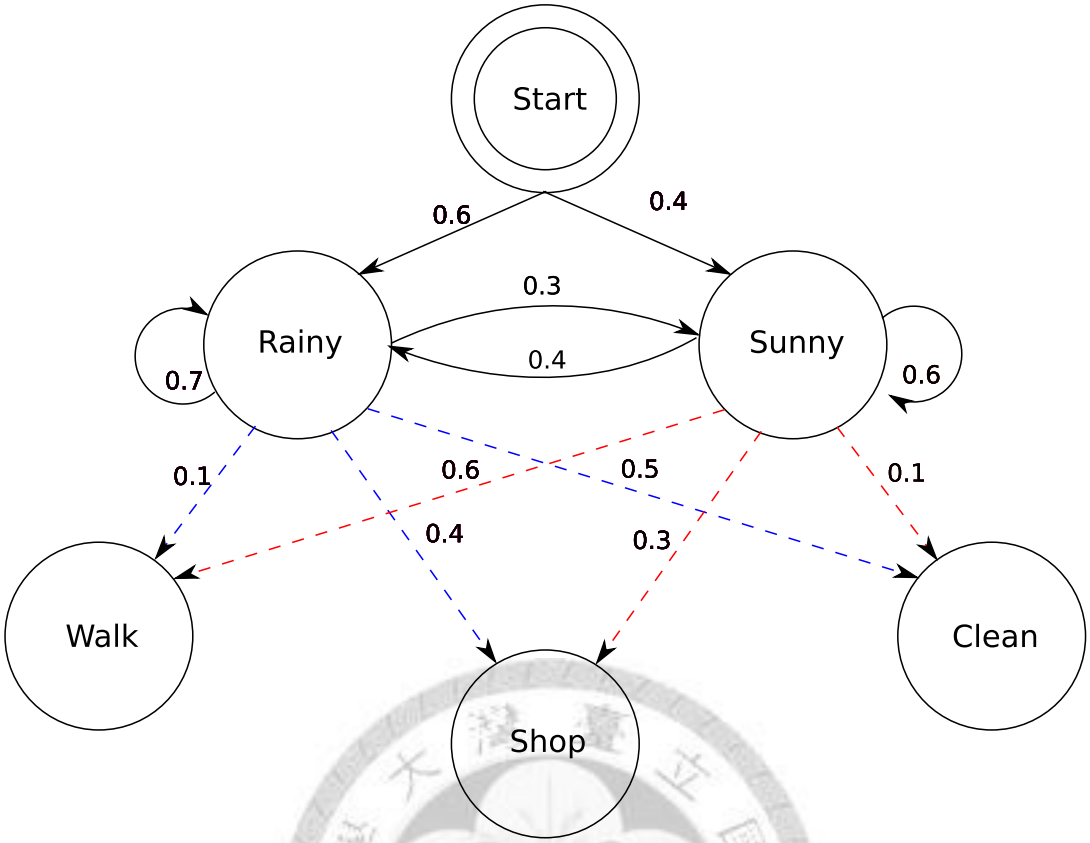


Figure 3.1: Example of HMM[1]

application in user behavior modeling.

3.1 Hidden Markov Model

Hidden Markov model is a statistic model which assumes there is a *Markov chain* with unobserved state [28]. A Markov chain is a memoryless stochastic process that transition to a state depend only on present state. The output of any given state is an deterministic observation event. The output of a Markov chain forms a sequence of observation events.

Base on Markov chain, the observation of a given state in HMM is a probability function instead of one-to-one correspondence of state. The model is a doubly embed-

ded stochastic process with an underlying stochastic process, the state sequence, that is not directly observable.

A HMM is a 5-tuple $\lambda = (S, V, A, B, \pi)$,

- S is a set of states with size N that is denoted as $S = \{s_1, s_2, ..., s_N\}$
- V is a set of observations with size M that is denoted as $V = \{v_1, v_2, ..., v_M\}$
- A is the $N \times N$ matrix of state *transition probabilities*.
- B is a set of N probability functions. Each function describe observation probability with respect to a state. This set of probability functions are called *emission probabilities*.
- π is the vector of initial state probabilities.

In user behavior modeling, the observations that all distinct states generate are finite in number. The set of observations with size M is denoted as $V = \{v_1, v_2, ..., v_M\}$. The set of probability functions $B = \{b_j(v_k)\}$ is defined as $b_j(v_k) = P(o_t = v_k | q_t = s_j), 1 \leqq k \leqq M, 1 \leqq j \leqq N$, where o_t is observation at time t, and q_t is state at time t.

Figure 3.1 is an example HMM to describe relation between weather and activities of a person. This person is interested in only three activities: walking, shopping, and cleaning house. The activity this person would do depend only on weather. λ of this HMM is described below:

- $S = \{Rainy, Sunny\}$
- $V = \{Walk, Shop, Clean\}$
- $A = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$
- $b_1(Walk) = 0.1, b_1(Shop) = 0.4, b_1(Clean) = 0.5$
  $b_2(Walk) = 0.6, b_2(Shop) = 0.3, b_2(Clean) = 0.1$

- $\pi = [0.6, 0.4]$

If we observe this person has done walking, cleaning, and than back to walking, we can compute the combination of different weather status at that time. Take the weather status Sunny, Rainy, Rainy, as example. The observation sequence is $\overline{O} = \{Walk, Clean, Walk\}$ and the state sequence is $\overline{q} = \{Sunny, Rainny, Rainny\}$.The probability of this weather status with the above activities sequence is:

$$
\begin{aligned}
P(\overline{O}, \overline{q}|\lambda) &= P(\overline{O}|\overline{q}, \lambda)P(\overline{q}|\lambda) \\
P(\overline{O}|\overline{q}, \lambda) &= P(Walk|Sunny)P(Clean|Rainny)P(Walk|Rainny) \\
&= 0.6 * 0.5 * 0.1 \\
&= 0.03 \\
P(\overline{q}|\lambda) &= P(q_0 = Sunny)P(q_1 = Rainny|q_0 = Sunny)P(q_2 = Rainny|q_3 = Rainny) \\
&= 0.4 * 0.4 * 0.7 \\
&= 0.112 \\
P(\overline{O}, \overline{q}|\lambda) &= 0.00336
\end{aligned}
\tag{3.1}
$$

3.1.1 Basic Algorithm for HMM

HMM is well known in temporal pattern recognition, such as speech[28] and gesture recognition[29]. This paragraph introduces two algorithms for training and computing probability using HMM.

The first is *Viterbi algorithm*. Viterbi algorithm finds most possible state sequence $\overline{q}$ given a observation sequence $\overline{O}$ and a HMM λ. This algorithm is usually used in checking how a observation fits into a HMM, because the computation overhead is lower than exploring all possible state sequences. Instead of exhausted search, Viterbi algorithm uses a *dynamic programming* technique to reduce computation overhead. At

initialization, it computes initial probability to each state in q_0. In each iteration t, it keep most possible state sequence and its probability from time 0 to time $t-1$ for all possible states in q_t. When it meets to the end of $\overline{O}$, we can find state sequence with highest probability.

The second is *Baum-Welch algorithm*. Baum-Welch algorithm finds the unknown parameters λ of a HMM given training observation sequence $\overline{O}$ with length T. This algorithm is a particular case of *expectation-maximization algorithm*(*EM algorithm*), which computes maximum likelihood to estimate the parameters of a HMM. In each iteration, the Baum-Welch algorithm calculates $P(q_t = s_j|\overline{O}, \lambda)$ and $P(q_t = s_i, q_{t+1} = s_j|\overline{O}, \lambda)$, where $1 \leqq t \leqq T-1$ and $1 \leqq i, j \leqq N$. The summation of the former from $t = 1$ to $t = T-1$ is the expected number of transition from state i in $\overline{O}$. The summation of the latter from $t = 1$ to $t = T-1$ is the expected number of transitions from state i to state j in $\overline{O}$. The Baum-Welch algorithm uses this two set of value to re-estimate new parameter λ', and iterates continuously till the new-generated parameter does not modify the model too much.

3.2 Hidden Markov Model Based User Behavior Classification

There are several work that used HMM to model and classify user behavior [25, 27]. Both of them used HMM to model behaviors from different class of users.

The first work[25] aimed to find relation between student performance and his/her content access pattern in web-based English learning system. The authors took click on navigation link and button as observation and emotion of user as state. They initiated HMM parameter by computing BIC[30] score of a HMM, which balancing goodness of fit to training data and model complexity. They divided students into two

groups by measuring the rank of their English performance, and then trained two HMM by Baum-Welch algorithm. The result showed that the classification precision is 92.73% for top performance group and 87.27% for the other group.

The second work[27] aimed to model of user's engagement in advice-giving dialogue. This work divided user's engagement into three classes: neutral dialogue, information seeking dialogue, and advice-giving dialogue. Each dialogue class is a HMM which initiate with the same parameter. The author classify dialogues into several types, such as opening, question, or objection, and took these dialogues types as observation. Each states were either user action or system action, which represented dialogue between user and system. This work initiated HMM parameter by random transition and emission probability, then trained HMM by Baum-Welch algorithm. The result showed that the classification precision is 76% for neutral dialogue, 85% for information seeking dialogue, and 65% for advice-giving dialogue.

We aims to provide a HMM model which can not only classify different types of user behavior, but also adapt given test case to behave like different users. This requires method to generate possible observation sequences which fit to the given HMM from original observation sequence. To the best of our knowledge, there is no research about adapting the possible observation sequence from given HMM and original observation sequence. In the following two chapters, we introduce a HMM called *User Computer Interaction Hidden Markov Model* to model user behavior and adapt original test case to fit characteristic of given type of user.

Chapter 4

User Computer Interaction

In this chapter, we introduce how a GUI interacts with a user. Base on this observation, a model called *User Computer Interaction Hidden Markov Model*(*UCIHMM*) is proposed to model user computer interaction. The UCIHMM aims to model different types of user computer interactions, such as novice user or interaction in emergence context, in a unify model. We give examples of two UCIHMMs and show how UCIHMM describes different types of user computer interaction at last section.

4.1 Interaction Between User and Graphical Computer Interface

Operating of software is an interaction between user and computer. In GUI application, user sends event to application via several input devices such as keyboard and mouse. When application receives GUI event, the correspondent event handler comes to processing data and gives feedback to user. The feedbacks can be page switch, property change of element in GUI, or other state change of system. At the end, user receives the feedback and determines to take next action.

Several types of bugs can be found during this interaction. When a novice user gets confused with how to operate GUI to achieve goal, he/she may arbitrarily click

control he/she can see, even when the control not enable. If GUI does not handle abnormal interaction well, these arbitrary clicks may cause GUI bugs. GUI bugs can also be found when an impatient user click the same button repeated because he/she can not wait for response from GUI. If GUI does not handle duplicated event properly, it can cause inconsistency in data of GUI state.

Current researches of automated GUI test case generation [19, 20, 21, 23] simulate only part of user computer interaction, i.e. which controls in GUI is able to trigger GUI event at current state. They have no idea about which page is more likely to make user confused and hence generate huge number of test cases to explore all possible execution paths of GUI. Also, they does not consider timing issue such as how much time the page switch takes. Bugs made by repeated click, as described in previous paragraph, is caused from mishandling of duplicate GUI event, because the GUI assumes user would not click more than once during that short period between page switch.

4.2 User Computer Interaction Hidden Markov Model(UCIHMM)

In this section, we introduce a model called *User Computer Interaction Hidden Markov Model*(*UCIHMM*) to model user computer interaction. We take each UCIHMM to model different types of user computer interaction, and generate test case to emulate specific type of interaction base on UCIHMM.

We divide UCIHMM into sub-model by modeling interaction in each page type separately. We assume user may behave differently according to design purpose of the page he/she is interacting with. For example, when an user get confused with complicated input dialog and does not know what to do, he/she can easily pass if the page is designed for displaying welcome message or popping up confirm window.

4.2.1 Definition

We define 8 page types according characteristic of iNuC GUI:

- Combo Input Page: any page required to type data, select check box, or set control property to change data.
- Browsing Page: any page required to browse information, such as find vital sign record. No typing or selecting check box in this type of page.
- Item Selection Page: page for navigating to other page, no working status information provided in this page.
- Blocked Alert Box: blocked window with ”OK” button
- Blocked Yes-No Box: blocked window with ”YES” and ”NO” buttons
- Blocked Input Box: blocked window with input text area.
- Blocked Browsing Box : blocked window required to browse information, such as find medication information.
- Blocked Box : blocked window without which is not belonged to other blocked page type.

Definition of page type can be easily adapted to other GUI application. Each logical page, e.g. page, tab, or window, must belong to a single page type. We suggest that the number of page type should be feasible, which make user to interact with pages in each page type several times in a short period, so as to collect usage pattern efficiently.

We define UCIHMM as a HMM with the following set of observations and states:

- Observation consists of basic GUI events encoded with user intension, such as mouse move and click repeatedly. Observations can be grouped by page types.

Each group consists of a observation named by given page type, and other 16 observations:

- MD: mouse down event.
- MU: mouse up event.
- R: repeated click.
- MF0: move mouse forward to target with moving speed less than 0.15 px/s.
- MF1: move mouse forward to target with moving speed more than 0.15 px/s and less than 0.51 px/s.
- MF2: move mouse forward to target with moving speed more than 0.51 px/s and less than 1.39 px/s.
- MF3: move mouse forward to target with moving speed more than 1.39 px/s.
- MA0: move mouse away from target with moving speed less than 0.15 px/s.
- MA1: move mouse away from target with moving speed more than 0.15 px/s and less than 0.51 px/s.
- MA2: move mouse away from target with moving speed more than 0.51 px/s and less than 1.39 px/s.
- MA3: move mouse away form target with moving speed more than 1.39 px/s.
- TA: type all characters at once.
- T0: type a character at a time with typing frequency less than 2.32 character/s.
- T1: type a character at a time with typing frequency more than 2.32 character/s and less than 3.62 character/s.

- T2: type a character at a time with typing frequency more than 3.62 character/s.
- NOP: no operation.

- State indicates user behavior which is able to group several GUI behaviors into a meaningful operation. States can be grouped by page types. Each group consists of a state named by given page type and emit to the observation named with the same page type. This state indicates change of page. In addition, each group consists other 6 states:
  - MV: moving mouse forward to target or away from target. This state emits to 9 observations: MF0, MF1, MF2, MF3, MA0, MA1, MA2, MA3, and NOP.
  - T: typing data via input device, e.g. keyboard or barcode scanner. Each T state indicate continuous typing at a time. This state emits to 5 observations: TA, T0, T1, T2, and NOP.
  - ClickMD: mouse down which indicate start of mouse click or drag drop. This state emits to 1 observation: MD.
  - ClickMiddle: mouse move with little distance occur between mouse down and mouse up. This effect are usually occur in touch screen device because hands vibrate very often. This state emits to 3 observations: MF0, MA0, and NOP.
  - ClickMU: mouse up which indicate the end of mouse click or drag drop. This state emits to 1 observation: MU.
  - ClickR: click repeatedly on the same position after first mouse click. This state emits to 1 observation: R.

4.3 UCIHMM for Different Types of Users

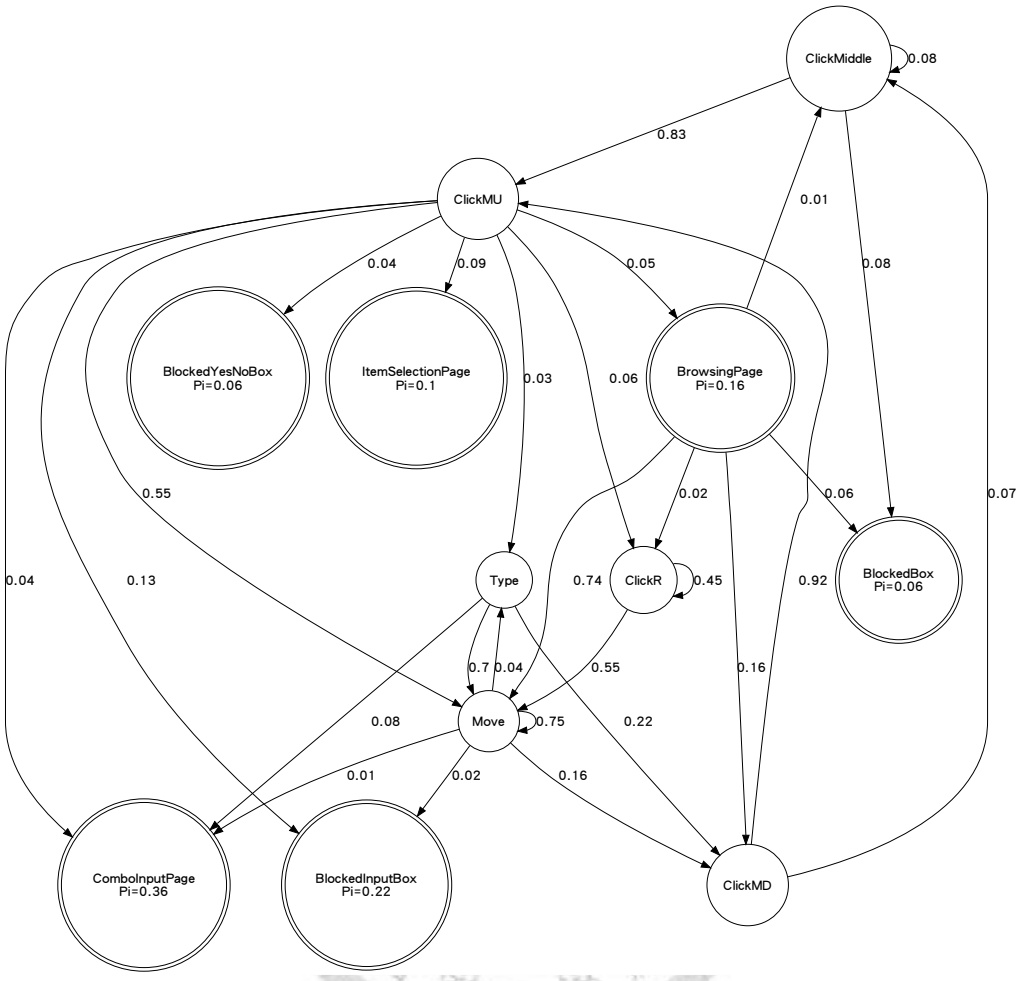


Figure 4.1: State transition probability diagrams of Browsing Page state from a UCIHMM.

UCIHMM models different types of user computer interaction by setting different transition and emission probability to UCIHMMs. Take modeling repeated click for two different user in Browsing Page for example. In state transition probability of a part of UCIHMM shown in figure 4.1, $P(ClickR|ClickR) = 0.45$ and $P(Move|ClickR) = 0.55$. This means this user has a high probability to click repeatedly in intermediate click and may click several times on the same position in a short period. In state transi-

tion probability of a part of another UCIHMM shown in figure 4.2, $P(ClickR|ClickR) = 0$ and $P(Move|ClickR) = 0.0$. This indicate this user only repeatedly click when the page is going to switch, and would not repeat more than once.

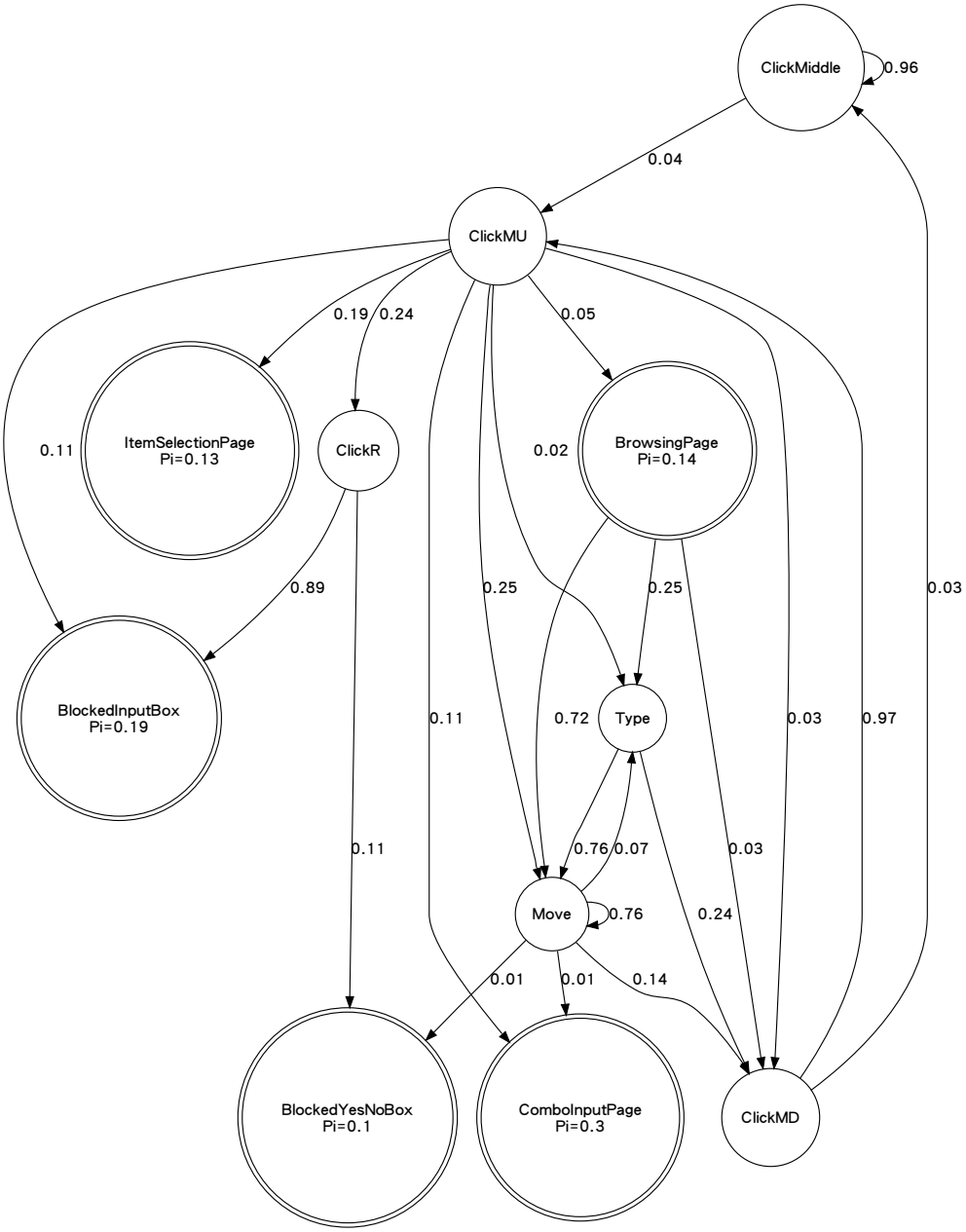


Figure 4.2: State transition probability diagrams of Browsing Page state from a UCIHMM.

Chapter 5

Personative Test Case Generation

In this chapter, we propose an algorithm to produce *personative test case*(*PTC*) base on the given UCIHMM. The PTC is a test case which manifests characteristic of an unique type of user computer interaction. We also show how this type of test case can expand expression power of existing test cases.

5.1 Information Lost in Test Case

Traditional GUI automation framework uses *transition event* to describe a test case. A transition event is a GUI event which changes GUI state, such as property of control or page switch.

This kind of framework focus on modeling GUI state transition. The GUI state transition model assumes transition completes immediately after user triggers click event, which is impractical in real application. This assumption rules out possibility for user to trigger event during state transition, such as repeatedly clicks after presses the commit button, which can cause bug if the application does not handle it properly.

In addition, traditional GUI test automation framework models transitions within system requirements. The model assumes GUI is implemented carefully to deny any unexpected event to occur, which is also a potential place to cause bug. When user gets confused with page design, he/she may arbitrarily click any button he/she has seem,

regardless of whether the button is expected to be valid or not. This phenomenon causes error when designer exposes GUI under unexpected event, especially when GUI becomes complex, it is easily to forget to disabling unneeded events.

According to the above observation, we propose a new kind of test case called personative test case(PTC). In the following sections, we introduce the *personative test case synthesis algorithm*(*PTCS algorithm*) to create PTC from original test case and an UCIHMM, and give an example in the last section.

5.2 Personative Test Case Synthesis Using UCIHMM

In this section, we introduce definition of PTC and PTCS algorithm to generate PTC. PTC emulates operating of GUI by a user from the given UCIHMM. The PTCS algorithm generates PTC to emulate user computer interaction of given UCIHMM. It synthesizes test cases generated from traditional test case with personative characteristic. This algorithm adds information lost onto traditional test case and reuses original test automation framework at the same time.

5.2.1 Definition of Personative Test Case

A personative test case(PTC) is a test case which manifests characteristic of an unique type of user computer interaction. A PTC takes observation defined in UCIHMM as its single step, with the following additional information:

- MA type observation: each step represents a mouse move event with given direction and distance to move away from. Each step must result in increasing the distance between mouse position and target position. If this is the last MA type observation before next MF type observation, append a MD and a MU observation at the end of this step to indicate random click by confused user.

- MF type observation: each step represents a mouse move event with given direction and distance to move forward. Each step must result in decreasing the distance between mouse position and target position. If this is the last MF type observation before next MD observation, move directly to position to next click.
- MD observation: each step represents a mouse down event.
- MU observation: each step represents a mouse up event.
- T type observation: each step represents sending a sequence characters to computer.
- R observation: each step represents mouse down event follow by mouse up event. This observation indicates user wants to click repeatedly once more.

A PTC automates input device to interact with GUI. It is the reverse process of collecting observation sequence of UCIHMM because it appends additional information to make observation sequence executable.

5.2.2 The Personative Test Case Synthesis Algorithm

The personative test case synthesis algorithm(PTCS algorithm) generates a set of PTCs. It takes three items as input:

- A traditional test case with length N: $TC = (e_0, e_1, e_2, ..., e_N)$, where e_i is a transition events which makes GUI state change. A transition event is one of the following event:
  - Page Switch: GUI page switch after this event.
  - Click: mouse click on certain position.
  - Drag Drop: mouse drag-drop with certain start and end position.

- Typing: typing via keyboard-like character device with certain characters sequence.

- A UCIHMM $\lambda = (S, V, A, B, \pi)$
- Number of PTC to generate: M.

We replace each e_i except Page Switch event in TC by the following *event grammar*:

- Click: $(\{MF\}|\{MA\}) * \{MF\}\{MD\}\{MU\}(\{R\}) * |$
  $(\{MF\}|\{MA\}) * \{MF\}\{MD\}\{MF0\}\{MU\}(\{R\})*$
- DragDrop: $(\{MF\}|\{MA\}) * \{MF\}\{MD\}(\{MF\}|\{MA\}) + \{MU\}$
- Typing: $\{T\}$

Each event grammar is an indeterministic observation sequence in UCIHMM in regular expression format. Each word enclosed with "{}" is a type of observation or an single observation. If the word enclosed with "{}" is a type of observation, it indicates this word can be replaced to any observation of this observation type.

The event grammar and transition event is one-to-one mapping. Although transition event listed here mainly focuses on WIMP style GUI, this mapping can be easily modified to adapt to other type of GUI application.

We defined the PTCS algorithm as below:

Algorithm 5.1 Personative Test Case Synthesis Algorithm

Require: $TC = (e_0, e_1, e_2, ...e_N)$, $UCIHMM\lambda$, and M.
Ensure: M PTCs.

1: **while** there is still a non Page Switch event sequence $(e_i, .., e_j)$ starts with Page Switch event e_{i-1} and ends with Page Switch event e_{j+1} **do**
2: $SubTCList \leftarrow (e_{i-1}, e_i, ..., e_{j+1})$
3: Remove $(e_{i-1}, e_i, ..., e_{j+1})$ from TC
4: $SubPTCList \leftarrow$ replace $SubTCList$ by event grammar
5: **while** element of $SubPTCList$ changes **do**
6: $SubPTCList \leftarrow$ expand all possible $SubPTC$ in $SubPTCList$ by one step
7: $SubPTCList \leftarrow$ find M observation sequence with highest score by Viterbi algorithm.
8: **end while**
9: $PTCList \leftarrow (PTCList, SubPTCList)$
10: **end while**
11: $PTC \leftarrow$ randomly combine each $SubPTCList$ in $PTCList$. Difference of occurrence of any $SubPTC$ in $SubPTCList \leqq 1$

Chapter 6

Experiment

We choose iNuC[6] as our target AUT. For UCIHMM collection, we collect usage logs of iNuC from three nurses during field trial of iNuC in physiotherapy ward, National Taiwan University Hospital. We than use random walk algorithm described in [31] to generate 2 and 4 test cases as input of PTCS algorithm and generate 100 test cases as baseline.

We than present the qualitative analysis including confusion matrix, bug detection, and code coverage rate. The result shows that our approaches can detect different types of bug according to different UCIHMM, and achieve higher code coverage compared with baseline test cases at the same time.

6.1 Setting

In this section, we introduce procedure to build UCIHMM and generate PTCs. For UCIHMM collection, we use AutoIt[32], a GUI automation tool, to collect usage log from nurses. We than use jahmm[33], a HMM toolkit written by Java[34], to train UCIHMM from usage logs.

For test case execution, we translate test cases into test scripts which are executable for AutoIt. We than use AutoIt to automatically execute the test scripts and use NCover[35], a code coverage measuring tool for Microsoft .NET framework, to measure code cov-

erage rate of each test suites. The detailed implementation information is described in the following subsection.

6.1.1 UCIHMM Collection

We use a behavior recorder written by AutoIt to collect usage logs of iNuC from three nurses, Alice, Bob, and Carol, during field trial of iNuC in physiotherapy ward, National Taiwan University Hospital. The behavior recorder logs each mouse move event, mouse down event, mouse up event, typing event, and page switching event with the following attributes

- Mouse move event: time spent, moving distance, and moving direction. We compute mouse moving speed from first and second attributes, and determine whether the mouse is moving away from next mouse click or moving toward to it.
- Mouse down and mouse up event: position of mouse down or mouse up occurs. These attributes help us to find the target position nurse wants to click.
- Typing event: typing frequency. We classify typing event into 4 types of observations by typing frequency.
- Page switching event: current page name. This attribute helps us to know which type of page the user is browsing.

Except from page switching event, all other three type of events are triggered by built-in AutoIt library. We detect page switching event by manually listing all possible pages in iNuC. The page described here is not limited to page appear in iNuC main window, but also tabs and windows which changes layout or functionality of current page.

UCIHMM Name	User	Familiarity	Number of Observation Sequences Collected
NA1	Nurse Alice	First day using iNuC	222
NB1	Nurse Bob	First day using iNuC	468
NC1	Nurse Carol	First day using iNuC	338
NA2	Nurse Alice	Second to fifth days using iNuC	550
NB2	Nurse Bob	Second to fourth days using iNuC	618

Table 6.1: List of UCIHMMs used in experiment.

After collection usage logs, we split each log into observation sequences. Each observation sequence starts from a page type observation and end with a page type observation.

We build 5 UCIHMMs according to different user and familiarity to iNuC. All UCIHMMs are listed in table 6.1. For each UCIHMM, we use all observation sequences collected as training data and run Baum-Welch algorithm implemented by jahmm to find parameters of UCIHMM.

6.1.2 Personative Test Cases Generation

We use random walk algorithm described in [31] to generate 2 test cases (test suite RAND2) and 4 test cases(test suite RAND4) as input of PTCS algorithm and generate 100 test cases(test suite RAND100) as baseline.

Since test cases described in [31] contains no page switch transition event, we use behavior recorder described in previous subsection to reconstruct the page switch transition events sequence. We insert page switch transition events into original test cases after that.

We than use PTCS algorithm to run each test case in RAND2 and RAND4 for each UCIHMMs with $M = 8$, i.e. for each test case in RAND2 and RAND4, PTCS algorithm is expected to generate 8 test cases from given UCIHMM.

6.2 Result

We present qualitative analysis including confusion matrix, bug detection, and code coverage rate in this section. The detailed analysis is described in the following subsection.

6.2.1 UCIHMM as Classifier

Although UCIHMM is not designed for classifying different type of user computer interaction, UCIHMM should provide acceptable classifying power because it aims to describe different type of user computer interaction in an unify model.

We use confusion matrix as evaluation matrix and random prediction as baseline. The confusion matrix is widely used in classification for showing how well a classifier predicts testing data. We also add precision, callback, and F_1 score in confusion matrix for assistant information. The precision is percentage of right prediction in given class, which can be seen as measure of exactness. The recall is percentage of right prediction in all prediction to the same class, which can be seen as measure of completeness. The F_1 score considers both the precision and recall of the test to compute score. The relation between F_1 score, precision, and recall can be written as $F_1 = 2 * precision * recall/(precision + recall), 0 \leqq F_1 \leqq 1$.

We first use UCIHMM with the same nurse but different familiarity to iNuC as a 2-class classifier. The confusion matrix of 2-class classifier from Alice and Bob are shown in table 6.2 and table 6.3.

		Predicted				
		NA1	NA2	Precision	Recall	F_1 score
Actual	NA1	172	48	.774	.562	.651
	NA2	134	410	.745	.895	.815

		Random Predicted				
		NA1	NA2	Precision	Recall	F_1 score
Actual	NA1	89	133	.404	.404	.404
	NA2	222	328	.595	.595	.595

Table 6.2: 2-class classifier with the Alice but different familiarity to iNuC

		Predicted				
		NB1	NB2	Precision	Recall	F_1 score
Actual	NB1	327	140	.698	.709	.704
	NB2	134	484	.783	.775	.779

		Random Predicted				
		NB1	NB2	Precision	Recall	F_1 score
Actual	NB1	201	267	.430	.430	.430
	NB2	265	353	.569	.569	.569

Table 6.3: 2-class classifier with the Bob but different familiarity to iNuC

Chapter 7

Summary

In this thesis, we propose a test case generation framework which take user computer interaction into account. The user computer interaction model provide the ability to classify user computer interactions under different contexts. We also combine interaction model and test cases to simulate user behaviors.

The contribution of this thesis is providing another perspective for test automation, which is interpretable for user-involved application. The thesis make testing with different types of user characteristics be achieved in an automated way before software release.

References

[1] "Example of hmm." http://upload.wikimedia.org/wikipedia/commons/4/43/HMMGraph.svg.

[2] Q. Xie and A. M. Memon, "Using a pilot study to derive a GUI model for automated testing," *ACM Trans. Softw. Eng. Methodol.*, vol. 18, no. 2, pp. 1–35, 2008.

[3] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 1 ed., Dec. 2006.

[4] A. M. Memon, "An event-flow model of GUI-based applications for testing: Research articles," *Softw. Test. Verif. Reliab.*, vol. 17, no. 3, pp. 137–157, 2007.

[5] D. J. Kasik and H. G. George, "Toward automatic generation of novice user test scripts," in *Proceedings of the SIGCHI conference on Human factors in computing systems: common ground*, (Vancouver, British Columbia, Canada), pp. 244–251, ACM, 1996.

[6] "iNuC (Intelligent Nursing Cart)." http://of.openfoundry.org/projects/1140.

[7] E. J. Weyuker and F. I. Vokolos, "Experience with performance testing of software systems: Issues, an approach, and case study," *IEEE Transactions on Software Engineering*, vol. 26, no. 12, pp. 1147–1156, 2000.

[8] A. Holzinger, "Usability engineering methods for software developers," *Commun. ACM*, vol. 48, no. 1, pp. 71–74, 2005.

[9] J. Pan, "Software testing." http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/, 1999.

[10] A. van Dam, "Post-WIMP user interfaces," *Commun. ACM*, vol. 40, no. 2, pp. 63–67, 1997.

[11] "Xerox parc." http://www.parc.com/.

[12] "Apple." http://www.apple.com/.

[13] "Mfc." http://msdn.microsoft.com/en-us/library/d06h2x6e

[14] "Qt." http://qt.nokia.com/.

[15] B. Myers, S. E. Hudson, R. Pausch, and Y. Pausch, "Past, present and future of user interface software tools," *ACM TRANSACTIONS ON COMPUTER-HUMAN INTERACTION*, vol. 7, pp. 3—28, 2000.

[16] Y.-T. Chuang, "inuc user interface design and implementation," Master's thesis, National Tsing Hua University, Hish-Chu City, Taiwan, 2009.

[17] J. M. Clarke, "Automated test generation from a behavioral model," *IN PROCEEDINGS OF PACIFIC NORTHWEST SOFTWARE QUALITY CONFERENCE*, 1998.

[18] A. Nagarajan and A. Memon, "Refactoring using event-based profiling," *IN PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP ON REFACTORING: ACHIEVEMENTS, CHALLENGES, EFFECTS*, 2003.

[19] D. R. Hackner and A. M. Memon, "Test case generator for GUITAR," in *Companion of the 30th international conference on Software engineering*, (Leipzig, Germany), pp. 959–960, ACM, 2008.

[20] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Using a goal-driven approach to generate test cases for GUIs," in *Proceedings of the 21st international conference on Software engineering*, (Los Angeles, California, United States), pp. 257–266, ACM, 1999.

[21] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Hierarchical GUI test case generation using automated planning," *IEEE Trans. Softw. Eng.*, vol. 27, no. 2, pp. 144–155, 2001.

[22] Q. Xie and A. M. Memon, "Designing and comparing automated test oracles for GUI-based software applications," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 1, p. 4, 2007.

[23] X. Yuan and A. M. Memon, "Generating event Sequence-Based test cases using GUI runtime state feedback," *IEEE Trans. Softw. Eng.*, vol. 36, no. 1, pp. 81–95, 2010.

[24] M. Pantic, A. Pentland, A. Nijholt, and T. Huang, "Human computing and machine understanding of human behavior: A survey," in *Artifical Intelligence for Human Computing*, pp. 47–71, 2007.

[25] A. W. P. Fok, H. S. Wong, and Y. S. Chen, "Hidden markov model based characterization of content access patterns in an E-Learning environment," in *IEEE International Conference on Multimedia and Expo, 2005. ICME 2005*, p. 201V204, 2005.

[26] P. Ji, J. Kurose, and B. Woolf, "Student behavioral model based prefetching in online tutoring system," 2001.

[27] N. Novielli, "HMM modeling of user engagement in advice-giving dialogues," *Journal on Multimodal User Interfaces*, 2009.

[28] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[29] T. E. Starner and A. Pentland, "Visual recognition of american sign language using hidden markov models," 1995.

[30] D. J. Hand, H. Mannila, and P. Smyth, *Principles of data mining*. MIT Press, Aug. 2001.

[31] H. Kuo-Chiao, "Model-based gui testing using sikuli," Master's thesis, National Taiwan University, Department of Electrical Engineering College of Electrical Engineering and Computer Science, National Taiwan University, Taipei City, Taiwan, 2010.

[32] "Autoit v3." http://www.autoitscript.com/autoit3.

[33] "jahmm." http://code.google.com/p/jahmm/.

[34] "Java." http://www.java.com.

[35] "Ncover." http://www.ncover.com/.