國立交通大學

資訊科學與工程研究所

碩 士 論 文

建構實作評量測驗產生器之研究

Building a Tester Generator for Performance Based Testing

研 究 生：王念主
指導教授：曾憲雄　博士

中 華 民 國 九 十 七 年 六 月

建構實作評量測驗產生器之研究

Building a Tester Generator for Performance Based Testing

研 究 生：王念主　　Student：Nien-Chu Wang

指導教授：曾憲雄 博士　　Advisor：Dr. Shian-Shyong Tseng

國立交通大學

資訊科學與工程研究所

碩士論文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年六月

建構實作評量測驗產生器之研究

研究生 ： 王念主　　指導教授 ： 曾憲雄博士

國立交通大學資訊學院
資訊科學與工程研究所

摘　　要

實作評量是根據受測者實際完成一項特定任務所作的評量，因為實作評量重視受測者實作的能力，所以很多認證考試都需要受測者通過相關的實作評量測驗。軟體操作技能檢定就是一個例子，藉由觀察受測者實際操作軟體來完成特定任務的操作過程，來評量受測者是否具備特定的軟體操作技能。然而，老師以人工的方式評量是花費龐大而且耗時。隨著電腦網路的快速發展，如何利用電腦化測驗的優勢，來輔助老師檢測軟體操作技能是我們主要的研究目標。根據我們的觀察，學生操作軟體來完成技能檢定中設計的特定任務，一般都會設定起始操作畫面，經由一些的操作動作，來檢測是否完成任務的操作需求。在這篇論文中，我們以一個有限狀態機(FSM)來描述軟體的操作流程，將 FSM 中的狀態(State)定義成描述軟體的操作畫面，而轉換函數(Transition)則定義成描述受測者的操作動作。以這樣的概念，我們提出了 Functional Specification Language，即可方便的透過正規文法(Regular grammar)來描述軟體的操作流程，並使用 parser generator 來產生對應的實作評量試題。老師可以藉由修改正規文法來設計實作評量試題中的檢測項目。我們利用此一方法來實作 MS Word 的軟體操作試題，最後我們請教學專家來確認這些試題是否能檢測受測者特定的軟體操作技能，而他們認為設計出來的試題是正確且容易被了解的。

關鍵字：實作評量、有限狀態機、試題產生器、電腦化測驗

Building a Tester Generator for Performance Based Testing

Student: Nien-Chu Wang Advisor: Dr. Shian-Shyong Tseng

Institute of Computer Science and Engineering

Nation Chiao Tung University

Abstract

Performance-based testing (PBT) is usually used to assess the examinee's procedural knowledge, the knowledge of knowing how, by performing some real world tasks. Many software skill certification exams have integrated PBT as a part of their exam to certify the examinee's software operating skill, where the examinee needs to perform a sequence of actions on specific software to achieve the required results. Traditionally, the evaluation of the examinee's software operating skills which only can be manually done by the teacher is time-consuming and costly. With our observation, using software to perform a sequence of actions to complete the task seems like a navigation process from the starting point of the software run-time state to get the required results, which can be modeled as a Finite State Machine (FSM), where the current state of FSM represents the software run-time status, and the transitions of FSM represent the actions the examinee can perform. Once the examinee performs an action in a certain state, the corresponding state transition will be triggered to move from the current state to the next state and then the PBT tester will visualize the next software run-time status. Based on this concept, a set of regular grammar, called the Functional Specification Language (FSL), is defined to describe the software run-time status and transitions of the PBT tester according to the functionality of specific software. Thus, a parser generator can be applied to generate the corresponding PBT tester based on the given FSL and the related action routines. To evaluate the proposed scheme, several experiments have been done to show the correctness, reusability, and expressive power of the scheme.

Keyword: Performance-based testing, Finite State Machine, Generator, Computer-based testing

誌　謝

首先，我要感謝我的指導教授，曾憲雄博士，老師非常有耐心，不厭其煩，一次又一次的討論，指導我撰寫這篇論文，讓我不只是完成這篇論文，而且學習到很多研究方法、思考方法、以至於表達方法，這是碩士班期間最寶貴的知識。感謝楊鎮華教授、孫春在教授和黃國禎教授，在口試時給了許多寶貴的意見，讓這篇論文更完善，並且讓我了解到這個研究的價值，拓展了我的視野。

再來，非常感謝翁瑞鋒學長，和學長討論使我的思考與研究得到相當大的啟發。此外，蘇俊銘、曲衍旭和林喚宇學長們在論文上也給予我很多寶貴的意見，也感謝楊哲青學長及黃桂芝學姊在我碩士班兩年來的照顧。怡利、立晧、學長姐以及學弟妹，感謝大家在我的碩士生涯一起努力、互相扶持。

最後，我要感謝我的家人，讓我能夠專注於研究上，並且在遭遇困難時鼓勵我，讓我能夠有自信地完成這篇論文。

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

Learning assessment is an important and essential part in the process of learning. With the growth of the computer and the Internet technology, the learning assessment via the computer and the Internet, called Computer-Based Testing (CBT) has become a trend especially in the information technology (IT) certification exam. In software skill certification, such as *MS Word* certification [1], there are usually two phases of tests. The first phase is to test the examinee's declarative knowledge, the knowledge of knowing what, which is usually the traditional single-choice or multiple-choice testing. The second phase is to test the examinee's procedural knowledge, the knowledge of knowing how, called Performance Based Testing (PBT) [2-4]. In PBT, the examinees have to perform some real world tasks by using specific software to demonstrate their software operating skills. For example, the examinee might be asked to complete the following task "*At the beginning of the document, insert a Table of Contents showing two heading levels.*" in MS Word 2003 certification exam. To complete this task, the examinee has to perform a sequence of actions on MS Word to get the required results and the examinee's software operating skills can be assessed during the process of accomplishing this task. Besides, PBT allows the examinees to come up with the correct answer using different approaches. That is, the examinee can solve a problem by showing how to navigate through a sequence of processes to get the required results.

Currently, the examinees have to use actual software to realize PBT in software skill certification. Besides, since the actions of most software performed by the examinee during the testing are not recorded, the evaluation of the examinee's software operating skills which only can be manually done by the teacher is costly

and time-consuming. In recent years, a testing platform for PBT, called the PBT tester, on some specific software which can provide adequate functionality of the software to complete the required tasks and track the examinee's action sequence were proposed as a cost effective way to realize PBT in software skill certification. However, to build a PBT tester for some specific software, the control flows of testing scenario should be simulated in the tester so that the software output can be correctly visualized according to the examinee's sequence of actions. Thus, the programming efforts of building a PBT tester are costly. Besides, the reusability for the new PBT test items is also a critical issue for the PBT tester developer.

With our observation, using software to perform a sequence of actions to complete the task seems like a navigation process from the starting point of the software run-time state to the final state to get the required results. Our idea is to use Finite State Machine (FSM) to model what the examinee performs during the PBT in software skill certification, where the current state of FSM represents the software run-time status, and the transitions of FSM represent the actions the examinee can perform. Once the examinee performs an action in a certain state, the corresponding state transition will be triggered to move from the current state to the next state and then the PBT tester will visualize the next software state. Based on this concept, a set of regular grammar, called the **Functional Specification Language** (FSL), is defined to describe the software run-time status and the transitions of the PBT tester according to the functionality of specific software. Thus, a parser generator can be applied to generate corresponding software tester based on the given FSL, where the grammar rule of FSL can be used to model the action sequence performed by the examinee, the non-terminals of FSL represent the software run-time status, and the terminals of FSL represent the actions the examinee can perform. Besides, action symbols are attached to each terminal symbol to trigger the corresponding action routines, such as the

routines for visualizing the next software run-time status and recording the actions performed by the examinee. To effectively visualize the next software run-time status, the visualization is divided into backgrounds and foregrounds, where the visualization action routine substitutes the foregrounds according to the action performed by the examinee, and the backgrounds do not need to be changed.

To evaluate the correctness, reusability, and expressive power of proposed scheme, we have implemented a prototypical system in the web environment. In this thesis, several test cases are designed to evaluate the expressive power of proposed scheme. According to the feedbacks of the teachers and the tester developers, we may conclude that the proposed scheme is workable and beneficial for them.

The remainder of the article is organized as follows. In Chapter 2, we introduce some related works about the traditional PBT tester and authoring tools for creating the PBT tester. Then, the proposed Tester Generator Scheme and the application of PBT tester are described in Chapter 3 and Chapter 4 respectively. Chapter 5 discusses the system implementation and experiments. Finally, Chapter 6 gives the conclusion and future work.

Chapter 2. Related Work

2.1. Computer-based Testing

The traditional multiple-choice test, administered via paper and pencil, provides a highly constrained testing environment. With dynamic visuals, sound, and user interactivity, computer based testing are being considered as alternative means for more effective testing [5-9], compared to traditional paper-and-pencil test. Innovative computerized test items [8] which refer to item types that use the computer's capabilities to improve measurement and assessment have been introduced over the past two decades. Zenisky and Sireci [12] gave examples to illustrate drag-and-drop, moving objects to create a tree structure, and several other response actions. In [13], a taxonomy or categorization of 28 innovative item types that may be useful in computer-based assessment are introduced, which is organized along the degree of constraint on the respondent's options for answering or interacting with the assessment item or task.

According to [8], it is possible to view innovations as falling into three categories. First, new item types that improve the assessment of some ability or skill can be created. Bennett et al. [7] discussed this kind of item type. The second category includes assessments designed to improve the authenticity of the assessment. Licensing exams that use computers to create high-fidelity simulations of professional activities, such as the clinical skills and the architectural licensing exams discussed in the next section fall into this category. Assessments of constructs not easily measured by conventional tests constitute the third category. Examples include tests of musical aptitude and interpersonal skills.

2.2. Computer-based Skill Assessment

Many skills and proficiencies are not easily assessed by paper-and-pencil multiple-choice test items. Consider, for example, clinical skills [11], [14], which take a history from a patient and perform a physical examination, or the design ability of an architect [10]. In recent years, the simulation technology has been applied to the certification exams of Microsoft Office Specialist (MOS), to realize PBT. Therefore, the examinees demonstrate their software operating skills by performing a sequence of actions on specific software to finish critical IT tasks in a simulated working environment, not using the actual software [15]. They have conducted that the simulation testing is a more effective way to evaluate the examinee's software operating skills.

However, research has shown that these performance-based testing are difficult to administer and score, and have low reliability and limited generalizability. Therefore, it is hard to construct and maintain a PBT tester for skill assessment. In this thesis, we focus on the software operating skill assessment, and try to propose a model to describe the PBT tester.

2.3. Different Approach to Construct PBT Tester

In this section, several approaches that can be used to construct a PBT tester are surveyed, including 1) *standard-based approach*, 2) *object-based approach*, and 3) *screenshot-based approach*.

- Standard-based Approach

  The IMS Global Consortium, an industry and academic consortium, produced

the IMS QTI (IMS Question & Test Interoperability Specification), which uses an XML file to describe a basic structure for the representation of question (item) and test (assessment) data and their corresponding results reports [16]. Some questions, such as drag & drop and spot the error, in QTI specification define an image as a test item to let the examinees perform some actions (i.e., clicking, drag object) on that image. For example, for a given picture, the question may look like "*The picture illustrates four of the most popular destinations for air travelers arriving in the United Kingdom: London, Manchester, Edinburgh and Glasgow. Which one is Glasgow?*" [17], and then the examinee uses mouse to click at the correct location on the given picture. However, the QTI specification did not clearly define the structure and the operating flow of simulation questions for PBT.

- Object-based Approach

In [8], Shavelson et al. designed several objects, such as batter, light bulb, and voltage, to let the examinees perform their hand-on performance task on the computer. Due to the special effort required to implement specific domain objects, it is hard to reuse the objects in different domains.

- Screenshot-based Approach

Kinnersley et al. [13] used Adobe Flash [20] to create a PBT tester to emulate MS Word and Excel. The tester is a timeline of MS Word or Excel screenshots to represent different software run-time status so that the examinee can perform some actions (i.e., clicking, drag object) on the screenshots, where the action performed by the examinee is to move from the current screenshot to the next screenshot in the timeline. Since Flash is not originally designed for software simulation, all of the screenshots need to be gathered, cropped and sized, and the mouse events

corresponding to the actions performed by the examinee also hove to be implemented. Therefore, it is very time consuming for a tester developer to build a PBT tester.

Adobe Captivate [21] was used to create a tester for software skill exam. When the teacher wants to create a tester, he/she opens the target software to perform a sequence of actions from the starting state of the software to the final state of the software. During the sequence of actions, the Captivate automatically captures the screenshot and creates the sequence of screenshot based on the actions performed by the teacher. However, to model different operating paths, objects such as button or menu and the corresponding mouse events have to be added manually. Thus, it is hard to construct, reuse, and maintain the more complex simulation.

Chapter 3. Tester Generator Scheme

As mentioned before, it is time-consuming and costly to design a tester for PBT, because the control flows of the corresponding action sequence performed by the examinee should be implemented so that the software run-time status can be correctly visualized. With our observation, using software to perform a sequence of actions to complete the task seems like a navigation process from the starting point of the software run-time status to get the required results which can be modeled as a Finite State Machine (FSM). In this thesis, we propose a generator-based approach, called the Tester Generator Scheme, to assist teachers in building a tester for PBT without low level programming, as shown in Figure 1: The scheme of tester generator. The scheme includes two phases: the *Construction phase* and *Testing phase*. In the construction phase, an authoring process which refers to the Functional Specification Language (FSL) and action routine declarations is proposed to help teachers specify the testing scenario, where the FSL is a set of regular grammar to describe the sequence of actions that the examinee can perform and corresponding software run-time status and the action routine declaration is to describe the action routines used in the FSL. The detailed definition will discuss in the following sections. According to the FSL and related resources, i.e., software run-time status images and action routines, a parser generator can be applied to generate the required tester. In the testing phase, the examinee can perform a sequence of actions on the tester to complete the required tasks.

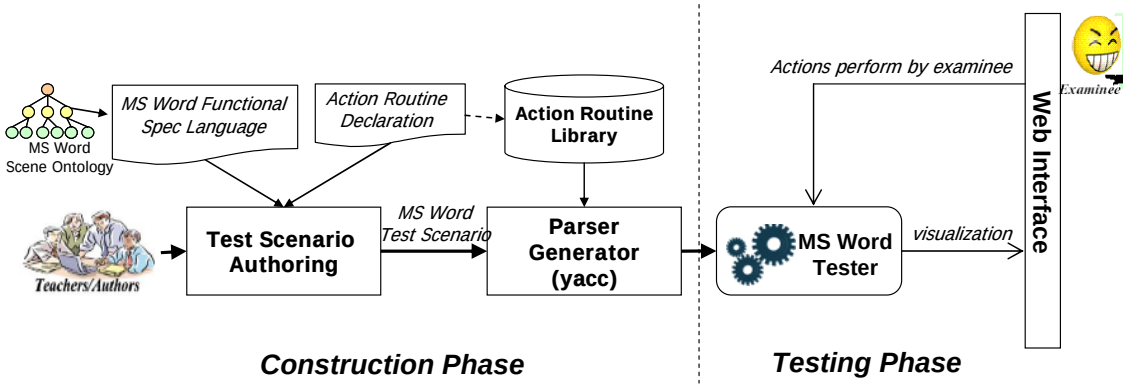

Figure 1: The scheme of tester generator

3.1. Performance-Based Testing in Software Skill Certification

As we know, PBT can test the examinee's procedural knowledge; in other words, the examinee's software operating skill can be assessed during the process of performing a sequence of actions, where a clear goal, called the task, is given for the examinee to complete. An example is described as follows:

Example 1: The correct action sequence of "*Change the font style of the title in MS Word*".

We assume that there is a performance-based test item "*Change the font style of the title in MS Word*". The correct action sequence should be as follows: ① Select the title of the document by double clicking the title or using the cursor to select the title, ② Click the "Format" menu, ③ Click the "Font" menu, ④ Set the correct font style, and ⑤ Click the "OK" button to finish the task. The examinee needs to perform a sequence of actions as mentioned above so that the correct font style can be done. Figure 2: MS Word PBT tester shows the corresponding MS Word screenshots of the action sequence performed by the examinee.

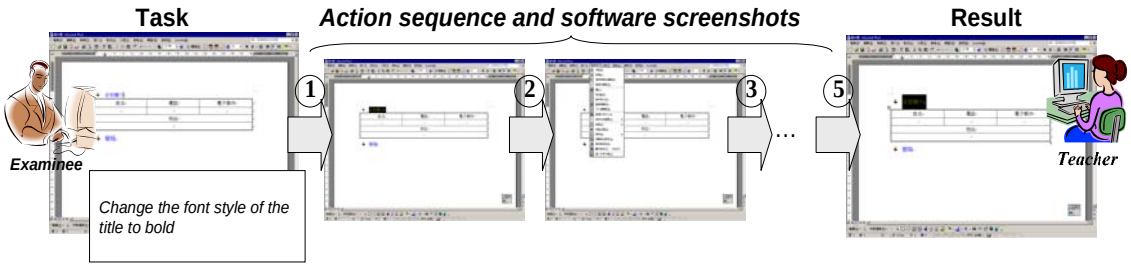

Figure 2: MS Word PBT tester

Since the actions of most software performed by the examinee during the testing are not recorded, the evaluation of the examinee's software operating skills which only can be manually done by the teacher is costly and time-consuming. To ease the efforts of the teacher, we should know what sort of information should be kept to find out "How does the examinee complete the task". Therefore, a testing platform for PBT, called the PBT tester, on some specific software which can provide an environment to simulate the required real situations was proposed to assess the examinee's software operating skills by providing adequate functionality of the software to complete the required tasks and track the examinee's action sequence, where the following elements need to be considered, as shown in Table 1.

Table 1: The elements of PBT tester

Element	**Description**
Task	A PBT tester must have a goal the examinee needs to achieve. i.e., the task in Example 1.
Action	The examinee can perform a certain action on the PBT tester, i.e., Steps 1-5 in Example 1
Path	A sequence of actions from the starting state to final state calls an operating path. A PBT tester needs to support different action sequences performed by the examinee to complete the task.
Visualization	According to the action performed by the examinee, the PBT tester will visualize the corresponding software run-time status to simulate the required real situation.

In order to describe the relations between the actions performed by the examinee and corresponding software run-time status, we propose a four-level ontology, called Scene Ontology (SO) as shown in Figure 3: The MS Word Scene Ontology, where the root level is the target software, the MS Word, and the second one describes the functionality of the target software which is used to assess the examinee's software operating skills, the third one describes the modules of functionality, and fourth one describes the sub-modules, to allow the reuse of modules and sub-modules definitions that commonly appear across different software, i.e., MS Excel and PowerPoint. There is one relation "a part of relation" in SO, which represents the visualization to describe that the software run-time status can be classified into one of the modules or sub-modules according to its functionality in MS Word.

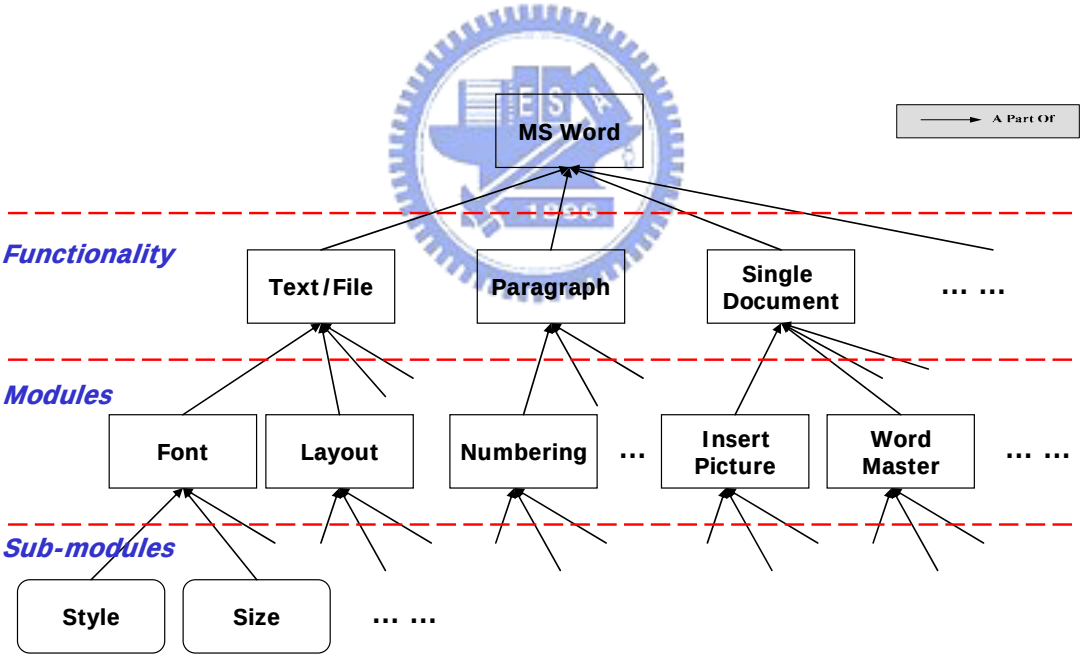

Figure 3: The MS Word Scene Ontology

According to the proposed Scene Ontology, using MS Word to perform a task can be represented as a finite number of actions; i.e., select a word, click a button, and input some texts, etc. Once a certain action is performed, the corresponding software run-time status will be visualized. Thus, the action sequence and the corresponding

run-time status visualization can be modeled as a Finite State Machine (FSM), where the current state of FSM represents the software run-time status, and the transitions of FSM represent the actions the examinee can perform. Once the examinee performs an action in a certain state, the corresponding state transition will be triggered to move from the current state to the next state and then the PBT tester will visualize the next software run-time status.

To effectively visualize the next software run-time status, the visualization is divided into foregrounds and backgrounds, where the foregrounds are the run-time status images that need to be changed to represent the next software run-time status and the backgrounds are those that do not need to be changed. In other words, only the foregrounds are changed in each state transition. Therefore, the visualization of software run-time status is also a sequence of the corresponding foregrounds and backgrounds combination according to the actions performed by the examinee as shown in Figure 4.

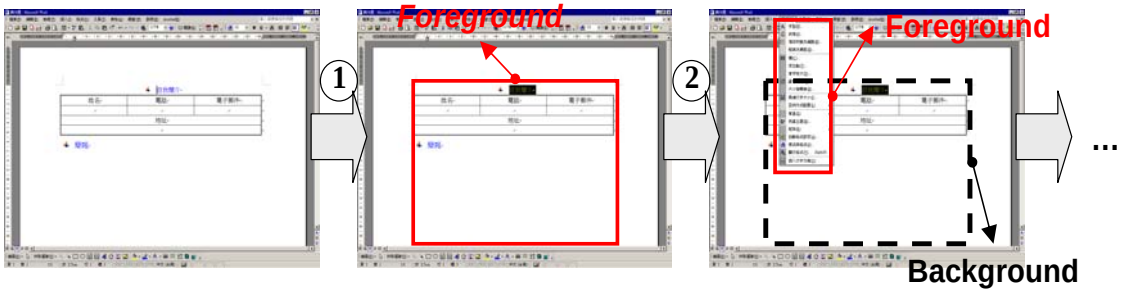

Figure 4: The visualization of PBT tester

3.2. Software Functional Specification Language

Based on the concept mentioned above, we define a set of regular grammar, called the **Functional Specification Language (FSL)**, where the grammar rule of FSL can be used to model the operating paths, the non-terminals of FSL represent the

software run-time status, and the terminals of FSL represent the actions the examinee can perform. Besides, action symbols are attached to each terminal symbol to trigger corresponding action routines.

Definition 1: Functional Specification Language is a 5-tuple, FSL = (N, Σ, P, S, γ), where

1. N is a finite set of *non-terminals*, which represents the run-time status of specific software.
2. Σ is a finite set of *terminal*s, which represents the actions that the examinee can perform, i.e., click the toolbar, select a word, etc.
3. P is a finite set of *production rules*, which represents the action performed by the examinee and the next run-time status of specific software. A production rule needs to satisfy one of the following forms:

   $A \rightarrow xB$, $A \rightarrow x$, where A, B in N, and x in Σ^*
4. S is the *starting state*, which represents the initial run-time status of the PBT tester.
5. γ is a finite set of *action symbols*, which is defined on Σ to trigger corresponding action routine, i.e., the routines for visualizing the next software run-time status and recording the actions performed by the examinee.

The following example illustrates the real case of FSL about MS Word certification exam.

Example 2: The FSL for "*Change the font style of the title in MS Word*".

For the performance-based test item given in Example 1, which describes the performance-based test item "*Change the font style of the title in MS Word.*" , we have

to define three non-terminals, S, A, and B, representing three different software run-time status during the testing. A set of terminals, {b, c, d, e, f, g, h}, are defined to represent the action that the examinee can perform. For example, terminal b means the examinee performs “select the title by double clicking”. Besides, two action symbols, #a1 and #a2, are used to trigger the corresponding action routines respectively. Note that an action routine may contain variables as parameters according to the action performed by the examinee. The details of symbol definitions are shown in Table 2.

Table 2: The descriptions of symbols for “Change the font style of the title”

Type	**Symbol**	**Description**
Non-terminal	S	The starting state of the testing. In this state, the run-time status of the title is one of the followings: normal, highlighted, and bold.
	A	The state of showing the format menu.
	B	The state of showing the font panel.
Terminal	b	Highlight the title by double clicking.
	c	Use the cursor to select the title.
	d	Click the “Format” menu.
	e	Click the “Font” menu.
	f	Click the correct font style. i.e., bold.
	g	Click the “OK” button to complete the task.
	h	Submit the task.
Action Symbol	#a1	Set software run-time status.
	#a2	Visualize the next software run-time status.

The corresponding FSL is shown in Figure 5.

$S \rightarrow b_{\#a1\ \#a2}\ S \mid c_{\#a1\ \#a2}\ S \mid d_{\#a2}\ A \mid h_{\#a2}$

$A \rightarrow e_{\#a2}\ B$

$B \rightarrow f_{\#a1\ \#a2}\ B \mid g_{\#a1\ \#a2}\ S$

Figure 5: An example of FSL about MS Word

In this grammar, the first step is to select the title of the document, which can be done by double clicking the title (*terminal b*) or using cursor to select the title (*terminal c*). After selecting the title, the corresponding action routines are triggered, highlight the title (*#a1*) and visualize the next software run-time status (*#a2*). The examinee then needs to click the "Format" menu (*terminal d*) to set the correct style. After clicking the "Font" menu item (*terminal e*), the font panel will be popped out. In the panel, the examinee needs to set the correct style, said bold (*terminal f*), click the "OK" button (*terminal g*) to complete the task, and finally submit the task (*terminal h*). The corresponding FSM is shown in Figure 6.

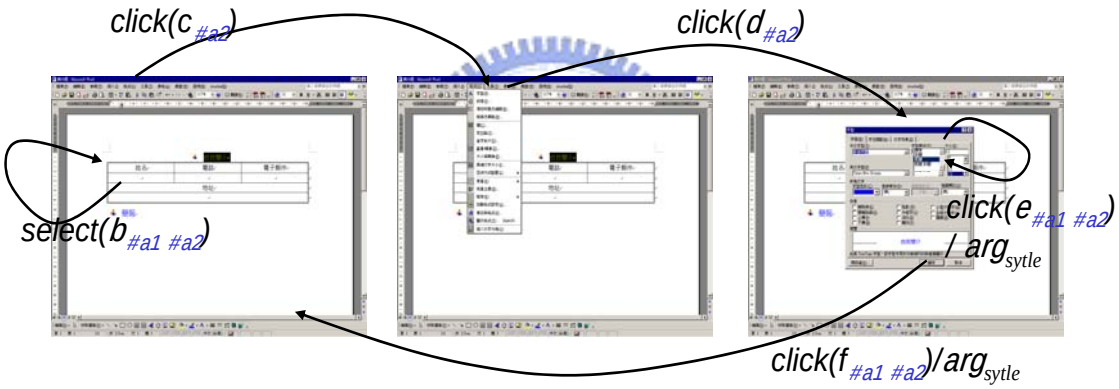

Figure 6: The FSM for "*Change the font style of the title in MS Word*".

Chapter 4. Application

In this chapter, we describe the test scenario authoring process, which includes three phases: 1) *Specify Required Task*, 2) *Design Operating Paths*, and 3) *Upload Visualization*.

4.1. MS Word Test Scenario Authoring Process

To assist test scenario authoring, we propose an authoring process to help the teachers, which includes three phases: the 1) *Specify Required Task*, 2) *Design Operating Paths*, and 3) *Upload Visualization*. Figure 7 shows a flowchart of the test scenario authoring.

Figure 7: The MS Word test scenario authoring process

(1) S*pecify Required Task:*

The teacher needs to specify the starting state and final state according to the required task the examinee should achieve. If the examinee performs the action sequence correctly, the state transition will eventually connect to the starting state as the final state as shown in Figure 6 above. However, the software run-time status in the starting state is not as the same as that in the final state. Therefore, the “key states” representing specific software run-time status between the starting state and final state should be specified by the teacher as well. That is, the action sequence performed by the examinee must navigate through the “key states” in order to get the required

results.

In this step, the corresponding non-terminals of the FLS will be determined according to the starting state, final state, and "key states".

(2) *Design Operating Paths:*

Suppose we have an action routine library which contains some build-in action routines for PBT tester construction, the teacher refers to the action routine library to specify the available action sequence from the starting state to final state, where the corresponding terminals and related action routines of FSL will be determined based on the given action sequence. An example is described as follows:

Example 3: Define the action types for the performance-based test item "*Change the font style of the title in MS Word*".

For the performance-based test item given in Example 1, we define three non-terminals, S, A, and B as shown in Example 2 and the corresponding action types need to be defined by the teacher, including *menu clicking*, *text selecting*, *list clicking*, and *button clicking* as shown in Figure 8.

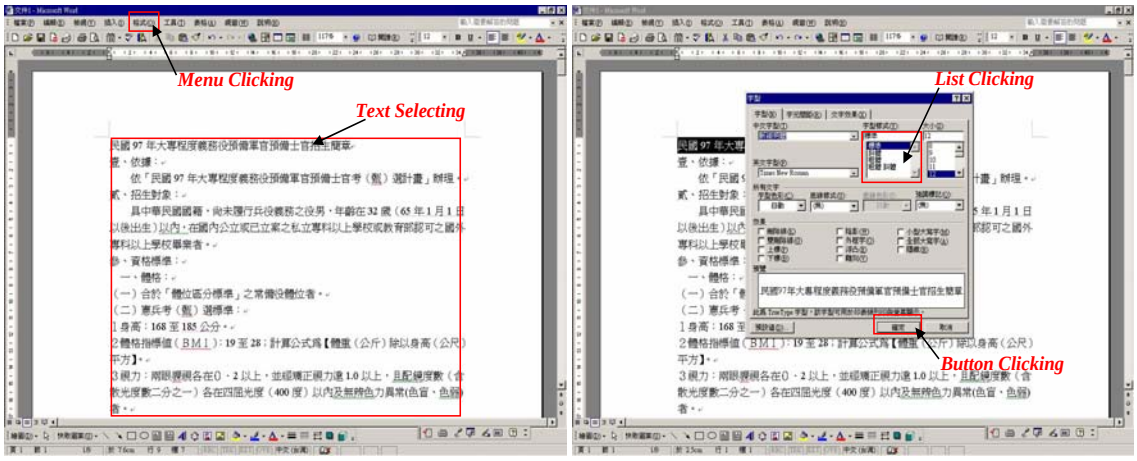

Figure 8: The illustration of action types

The details of action type descriptions and the corresponding definitions are shown in Table 3 and Table 4, respectively.

Table 3: The action types used in “Change the font style of the title”

Non-terminal	Action Type	Description
S	*Menu Clicking*	The *menu clicking* on the “Format” menu will trigger the state transition from non-terminal A to non-terminal B.
	Text Selecting	The *text selecting* on the working area will change the current software run-time status; that is, the text will be highlighted.
A	*Menu Clicking*	The *menu clicking* on the “Font” menu needs to be defined by the teacher so that the current state will move to the non-terminal B.
B	*List Clicking*	The *list clicking* on the “Font Style” list will set the software run-time status according to the clicking style.
	Button Clicking	The *button clicking* on the “OK” button will trigger the state transition to move to the non-terminal S and the corresponding software run-time status will be set.

Table 4: The definition of action types

Action Type	Description
Menu Clicking	The menu contains frequently used functionality such as Open, Save, and Print. According to the required task, the corresponding menu should be set by the teacher to check if the examinee knows the exact functionality to complete the required task. i.e., the menu clicking in Example 3.
Text Selecting	Several user events are defined on the working area such as clicking, double clicking, selecting, and dragging. i.e., to format the text, the examinee should select the required text before clicking the "Format" menu, or the software run-time status will not be changed.
List Clicking	The list contains several choices for the examinee to choose. According to the required task, the examinee needs to choose the correct option to complete the task.
Button Clicking	After performing sequence of actions such as text menu clicking and list clicking, the examinee usually needs to click the button to check if the action sequence is performed correctly.

(3) *Specify Visualization:*

Since the PBT tester will visualize the corresponding software run-time status according to the action sequence performed by the examinee, the teacher needs to specify related run-time status visualization in order to simulate the required real situation. In this step, the corresponding parameters of action routines of FSL will be determined according to the run-time status visualization.

4.2. PBT Tester Revision and Combination

With the defined FSM model for PBT tester, the examinee can perform a sequence of actions on the PBT tester that can be used to assess the examinee's software operating skills. The PBT tester revision and combination are used to design a PBT tester with different functionalities, where the revision is to revise the original PBT tester functionality or add new functionality to a PBT tester; and the combination is to combine different PBT tester by FSM union or concatenation to provide comprehensive software operating skill assessment. The revision and combination of a PBT tester are described as follows.

(1) Reusing an Existing PBT Tester

Algorithm: Revise the Functionality of a PBT Tester

Input: The FSL constructed in Example 2.

Output: A PBT tester with revised functionality.

Parameter: a_i is the action that needs to be revised.

Step 1. Import an existing FSL into a new PBT tester and name the FSL f_1.

Step 2. For each terminals t_i in f_1, find the terminals for a_i.

Step 3. For each a_i, revise the corresponding action routines attached in the terminals found in Step 2.

Step 3. For each revised action routines in Step 3, configure the corresponding parameters for a_i.

Step 4. Apply parser generator to generate the corresponding PBT tester based on the f_1.

Example 4: Modify the functionality of an existing PBT tester.

Assume that the original PBT test item is "*Set the title in bold*", the teacher want to apply the same test scenario to evaluate the examinee's software operating skills and with a modification to "*Set the title in italic*". For this example, we have to revise the terminal symbol e in Example 2, which is used to define the action type "*list clicking on bold*". The details of symbol definitions are shown in Table 5. The terminal symbol e is revised to e', which represents the examinee click the "italic" in the font style list.

Table 5: The descriptions of symbols used in "Set the title in italic"

Type	**Symbol**	**Description**
Non-terminal	S, A, B	*The definitions are the same as those in Example 2*
Terminal	b-d	*The definitions are the same as those in Example 2*
	e'*(revised)*	Click the "italic" in the font style list
	f-h	*The definitions are the same as those in Example 2*
Action Symbol	#a1	*The definitions are the same as that in Example 2*
	#a2	*The definitions are the same as that in Example 2*

Figure 9 and Figure 10 are the FSL and corresponding FSM, respectively. In the FSL, we need to redefine the terminal symbol e and configure the parameters for action routine so that the software run-time status can be visualized correctly.

$S \rightarrow b_{\#a1\ \#a2}\ S \mid c_{\#a1\ \#a2}\ S \mid d_{\#a2}\ A \mid h_{\#a2}$

$A \rightarrow e'_{\#a2}\ B$

$B \rightarrow f_{\#a1\ \#a2}\ B \mid g_{\#a1\ \#a2}\ S$

Figure 9: The FSL for "Set the title in italic"

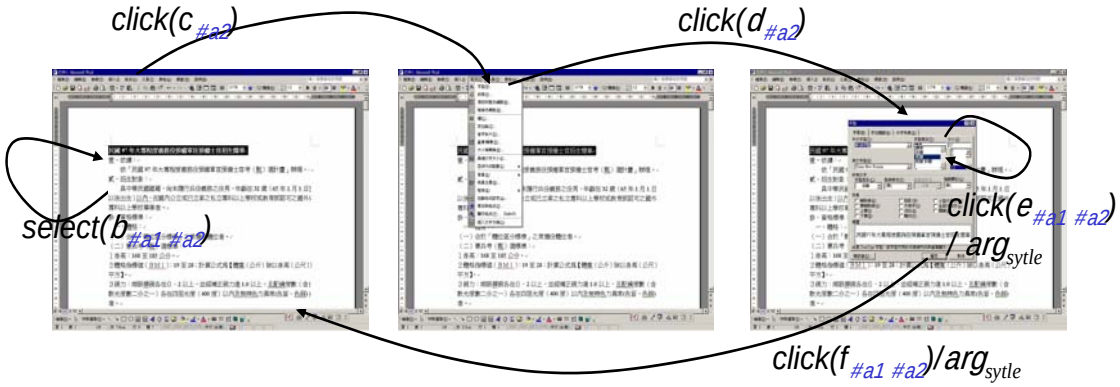

Figure 10: The FSM for "Set the title in italic"

Algorithm: Add New Functionality of a PBT Tester

Input: Given the FSL in Example 2.

Output: A new PBT tester with new functionality.

Step 1. Import an existing FSL into a new PBT tester and name the FSL f_2.

Step 2. According to MS Word Scene Ontology, add non-terminals to f_2.

Step 3. For each non-terminals in Step 2, add terminals and related action routines.

Step 4. For each action routines in Step 3, configure the corresponding parameters for visualization.

Step 5. Apply parser generator to generate the corresponding PBT tester based on the f_2.

Example 5: Reuse an existing PBT tester and add new functionality

Assume that the original PBT test item is "*Set the title in bold*" and the teacher wants to add new functionality to original one "*Set the title in bold* and *use Times New Roman as default font type*". The above algorithm can be applied.

In this example, we have to add one non-terminal C to represent the software run-time status for the font type setting. Besides, three terminals, i, j, and k, are added

which represent that the examinee clicks the font type dropdown list and selects the "Times New Roman" font type or others. The details of symbol definitions are shown in Table 6.

Table 6: The descriptions of symbols used in "Set the title in bold and use Times New Roman as default font type"

Type	**Symbol**	**Description**
Non-terminal	S, A, B	*The definitions are the same as those in Example 2*
	C	The state of showing the format menu.
Terminal	b-h	*The definitions are the same as those in Example 2*
	i	Click the "Font Type" dropdown list.
	j	Click the "Times New Roman" font type.
	K	Click the other font type.
Action Symbol	#a1	*The definition is the same as that in Example 2*
	#a2	*The definition is the same as that in Example 2*

Figure 11 shows the FSL of this example, we add several production rules to model the action sequence of setting the font type. Figure 12 shows the corresponding FSM.

$S \rightarrow b_{\#a1\ \#a2}\ S \mid c_{\#a1\ \#a2}\ S \mid d_{\#a2}\ A \mid h_{\#a2}$

$A \rightarrow e_{\#a2}\ B$

$B \rightarrow f_{\#a1\ \#a2}\ B \mid g_{\#a1\ \#a2}\ S \mid i_{\#a1\ \#a2}\ C$

$C \rightarrow j_{\#a1\ \#a2}\ B \mid k_{\#a1\ \#a2}\ C$

Figure 11: The FSL for "Set the title in bold and use Times New Roman as default font type"

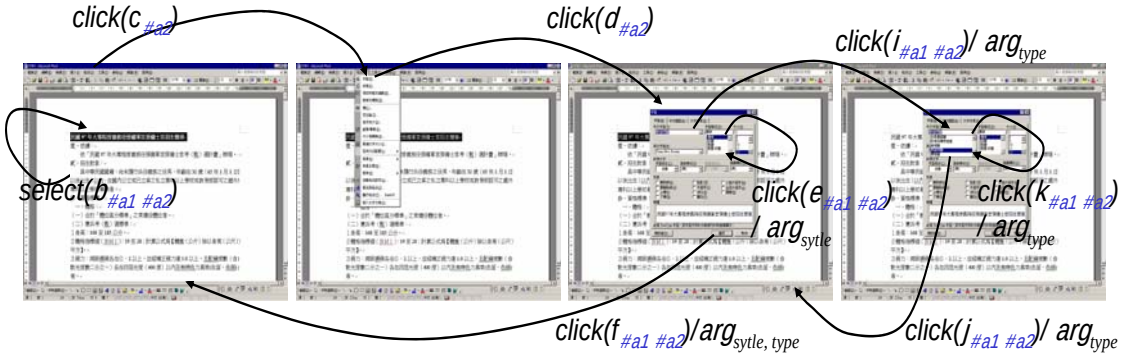

Figure 12: The FSM for “Set the title in bold and use Times New Roman as default font type”

(2) The PBT Tester Combination

Algorithm: Design a PBT tester with different paths

Input: An FSM named, named m, and other FSMs, named m_i, with the same start and final states.

Output: A new PBT tester with different path to complete the task.

Step 1. Import m and m_i into a new PBT tester.

Step 2. For each transition t_i connect to the start state of m_i, add t_i to the start state of m; delete t_i and start state of m_i.

Step 3. For each transition t_i’ connect to the final state of m_i, add t_i’ to the final state of m; delete t_i’ and final state of m_i.

Example 6: Construct a PBT tester with two paths to complete the task.

PBT allows the examinees to come up with the correct answer by different action sequences. For example, to insert a symbol (i.e., a comma) in a paragraph, the examinee has two options: 1) Click the “Insert” menu and insert a comma in the “Symbol” window, or 2) Click the ”View” menu, enable the toolbar, and insert a comma. The examinee can complete the task by one of action paths above. Suppose

we have two PBT testers, one describes the action sequence in the first option and the other describes the second option. We can union these two PBT testers to generate a new one, as shown in Figure 13.

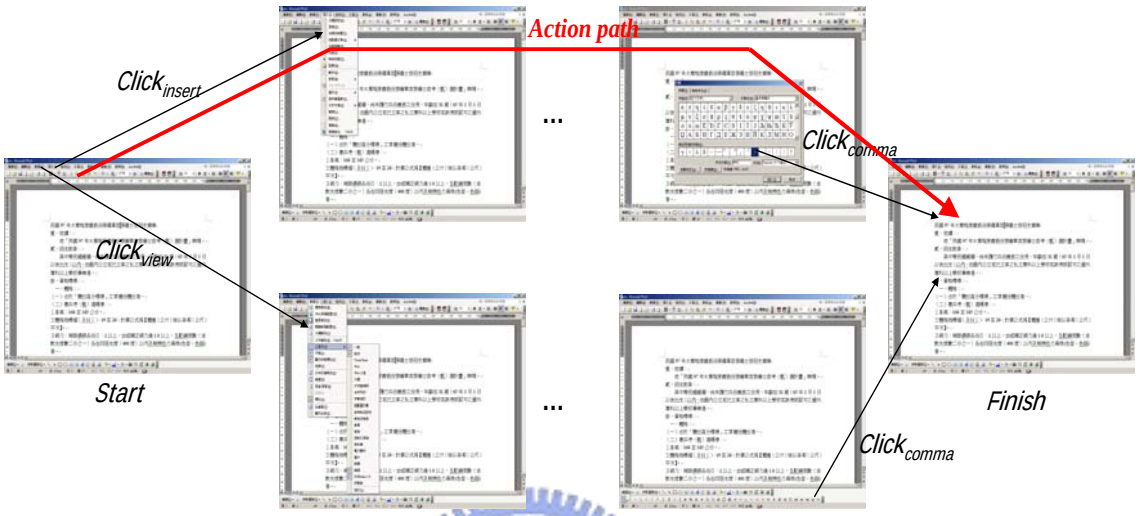

Figure 13: A PBT tester with two action paths

Algorithm: Design a PBT tester with trap paths

Input: An FSM named, named m, and other FSMs, named m_i.The m_i model the trap path that the examinee may navigate through in certain state of m.

Output: A new PBT tester with trap paths.

Step 1. Import m and m_i into a new PBT tester.

Step 2. For each state in m, find the corresponding state that the trap path may occur.

Step 3. For each state found in Step 2, add a state transition the start state of m_i.

Example 7: Construct a PBT tester with a trap path.

When the examinee performs a sequence of actions to complete the task, some misconceptions may occur. For example, to format the text, if the examinee forgets to select the text first, the format of the text will not change. Therefore, we can design a

“trap path” and combine with the correct path. That is, the examinee can also navigate through the “trap path”, but the action sequence performed on the “trap path” will not affect the results. In order to highlight the misconception, we can also design action routines to record what the examinee performs. Figure 14 shows corresponding PBT tester.

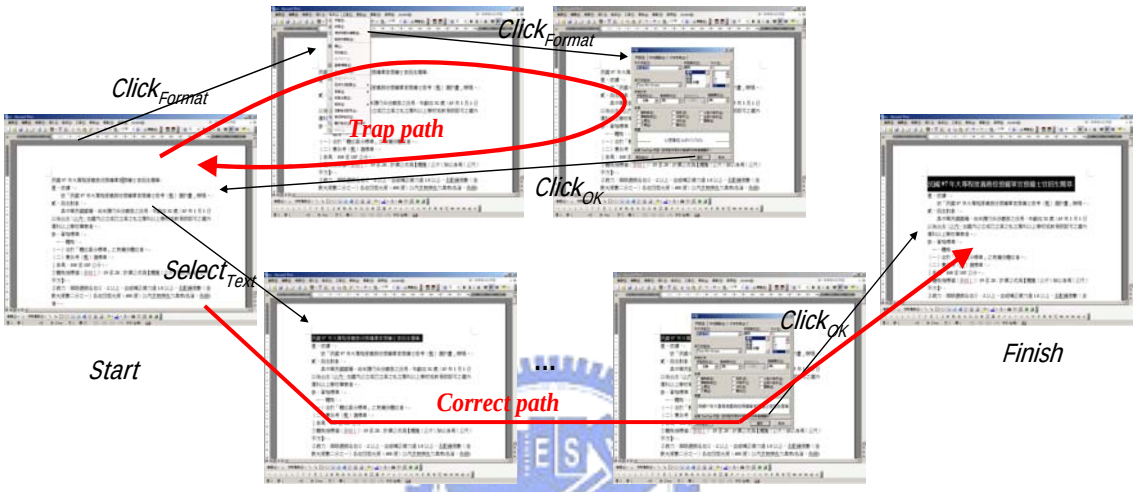

Figure 14: The PBT tester with a trap path

Algorithm: Design a PBT Tester with Comprehensive Skill Assessment.

Input: An FSM named, named m, and other FSMs, named m_i, where m_i is a PBT tester that Is used to assess different software operating capabilities.

Output: A new PBT tester with comprehensive software operating skills.

Step 1. Import m and m_i into a new PBT tester.

Step 2. For each m_i, add a state transition from the finish state of m_{i-1} to the start state of m_i.

Step 3. Configure the corresponding visualization of m_i if necessary

Example 8: Construct a PBT Tester with comprehensive skill assessment.

Some performance-based test item may contain comprehensive software

operating skills. For example, to create a business document, the examinee may need to know how to create a table to show the sales amount in the end the month, format the related text, and create a Table of Contents etc. To design a PBT tester for this kind of test item, we can combine several PBT testers to generate a more complex one, as shown in Figure 15.

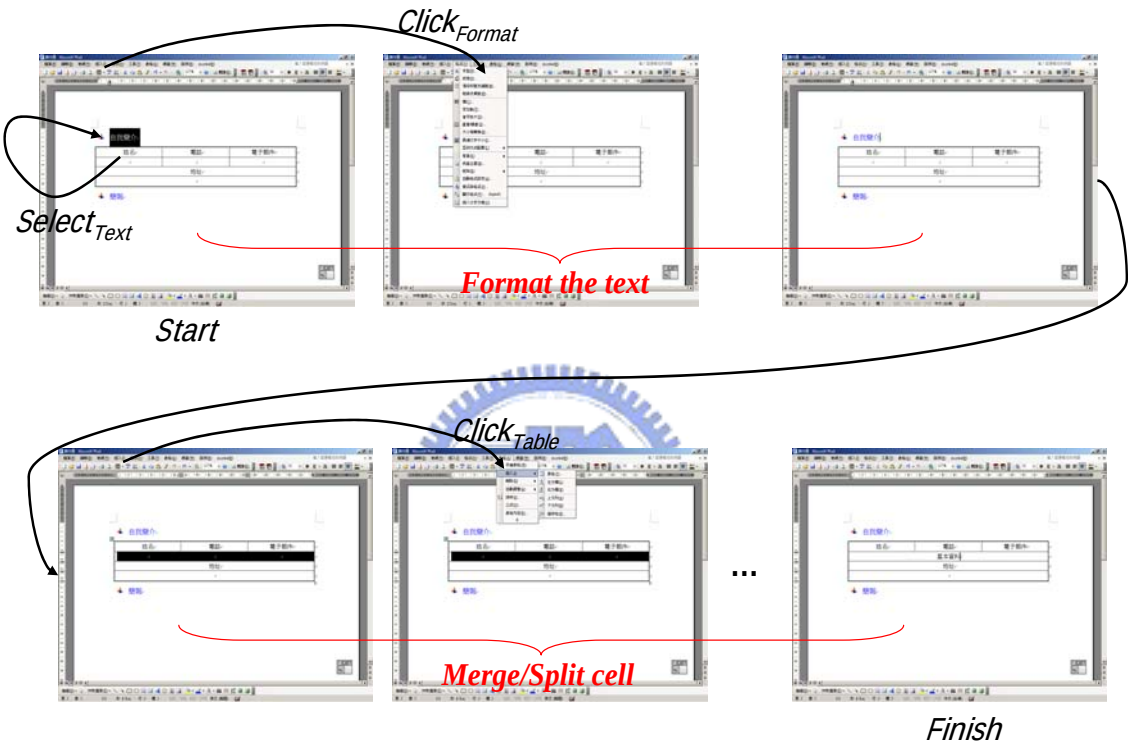

Figure 15: The PBT tester combination

Chapter 5. Experiments

In this chapter, the implementation and evaluation design are described. Then, experimental results are presented and discussed.

5.1. System Implementation

We have implemented a prototype system based on web-based environment to evaluate the proposed scheme. As shown in Figure 16 (1), the examinee should first select the required text, and then drag the text to the first line. After that, the examinee needs to select first row of the table and delete it by clicking the menu bar as shown in Figure 16 (2).

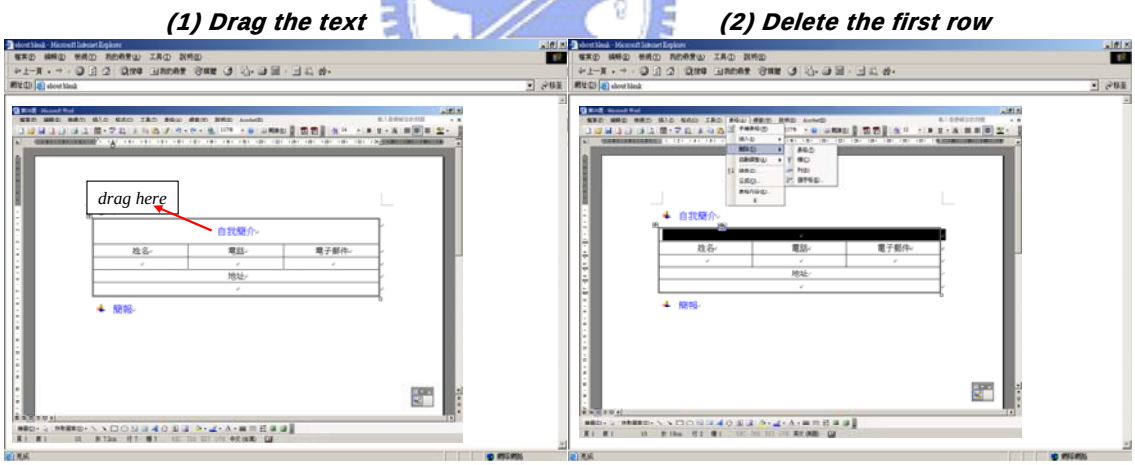

Figure 16: The screenshots of MS Word PBT tester

There are two inputs and one output in our generator, where “scenario.xml” and “lladacs.bnf” are the inputs to describe what software run-time status images are used and the corresponding regular grammar of the required PBT tester respectively. Then, the YACC [19], an LALR parser generator that can accept a regular grammar specification and produce parsing tables for the specified language, is applied to

generate the PBT tester.

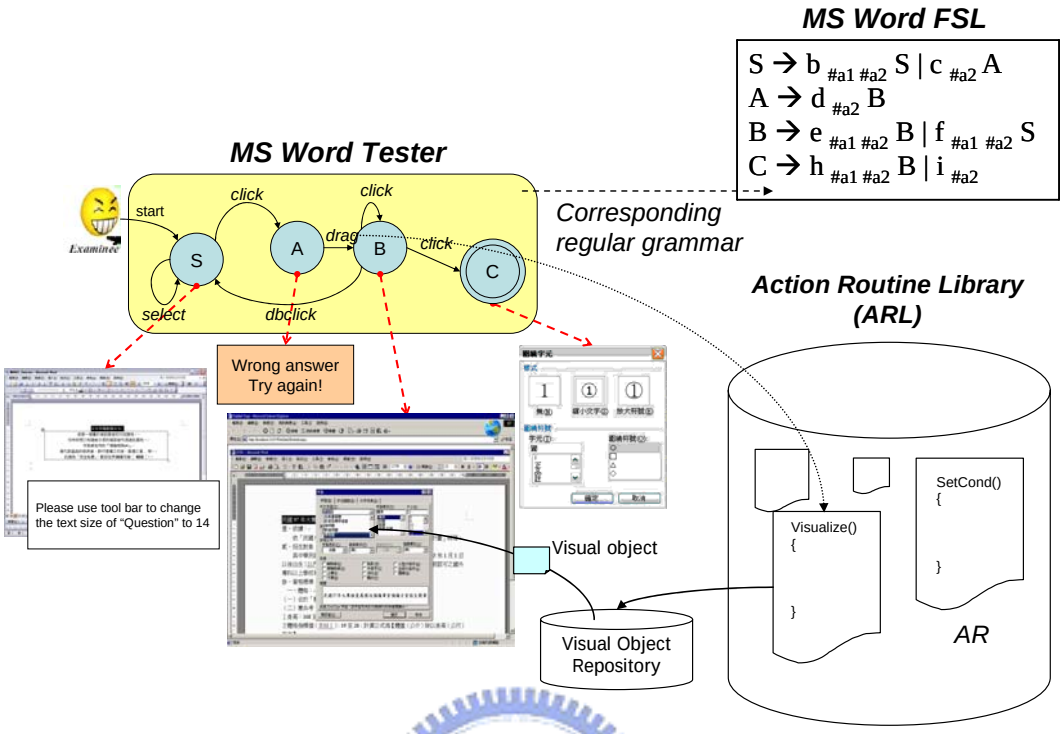

Figure 17: The running process of MS Word PBT tester

The PBT tester execution algorithm is shown as follows.

Algorithm: The PBT Tester Execution Algorithm

Input: Action sequence performed by the examinee.

Output: The scenes that represent the corresponding software run-time status according to the action performed by the examinee.

Step 1. Check the action performed by the examinee.

Step 2. If the action matches the terminal symbol defined in FSL, trigger corresponding action routines; else go to Step 1.

Step 3. The action routine then sets the software run-time status and visualizes the corresponding software run-time status.

Step 4. If reach the final state, then stop; else go to Step 1.

5.2. Experiment Design and Result

To evaluate the expressive power of proposed FSL, several MS Word test cases are designed to perform an experiment as shown in Table 7. The test cases are referred to [22] with some modifications, where this book is one of the reference book used in Techficiency Quotient Certification (TQC). There are 10 test cases with 5 specific software operating capabilities: *format the font and the paragraph*, *modify the text by using find/replace/go to* functionality, *merge/split/delete the table cells*, *insert picture/word art*, and *insert index or table of contents*. Each test case was transformed into FSL and then imported into the prototype system.

Table 7: MS Word test case

Test Case	Functionality	Test Item	Description
1	Font	*Use 22 point bold as default font type*	Format the text
2	Find/Replace	*Find the "ms word", replace with "MS Word"*	Find and replace the text
3	Paragraph	*Use single line space as default*	Format the paragraph
4	Symbol	*Insert a comma between "MS Word" and "Excel"*	Insert comma/colon
5	Merge/Splitcell	*Merge cells in the 2nd row*	Merge the cell
6	Add/Delete table/row/column	*Delete the 1st row of the table and add tow columns*	Add table and delete the row
7	Picture/WordArt	*Insert a picture at the right hand side of the document*	Insert picture
8	Document background	*Change background of the document to "Cloud"*	Insert document background
9	Index/Table of Contents	*Insert Index for the following text*	Insert index
10	Picture editing	*Edit the picture's position: 3.7cm from left*	Edit picture's attribute

Table 8: The corresponding FSL of test case

Test Case	Number of Non-terminals	Number of Terminal	Number of Grammar Rule	Number of Action Routine Type
1	4	10	10	2
2	5	10	10	2
3	3	6	6	2
4	14	25	25	2
5	5	7	7	2
6	8	11	11	2
7	5	12	12	2
8	6	9	9	2
9	10	17	17	2
10	4	9	9	2

In this experiment, 4 domain experts who teach MS Word in a MiaoLi elementary school used the prototype system to evaluate if the test case can really reflect the corresponding software operating capabilities. After using the prototype system, we evaluate the satisfaction degree of the domain experts by a 5-point Likert scale questionnaire. Figure 18 shows the average satisfaction degree of the domain experts in each test case.

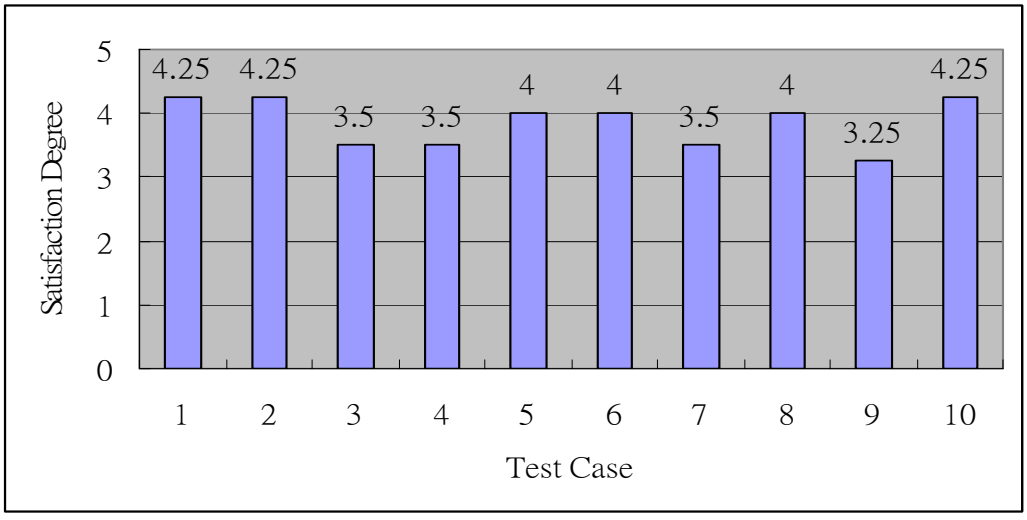

Figure 18: The results for satisfaction degree questionnaire (5 is the highest possible score)

After further discuss with domain experts, we found that most domain expert argue that the action sequence should follow the predefined actions which are not as flexible as real MS Word. Therefore, the lack of instruction may cause their inconvenience. On the hand, since the action sequence is predefined, we can easily evaluate the action sequence performed by the examinee which out perform the real software.

5.3. Discussion

Since our PBT tester is describe by the corresponding FSL, it has the following properties:

- Understandability

The FSL clearly defines the software run-time status and the transition of the PBT tester. It can be easily examined for correctness, consistency, and completeness.

- Flexibility

An advantage of this approach is that once the FSL is built, the PBT tester generation process is automatic. It is relatively easy to extend the functionality of PBT tester by adding, changing, and deleting the grammar rules.

- Scalability

To aid in the scalability of the FSM, the visualization is divided into foregrounds and backgrounds, and the action routine is used to configure the corresponding visualization.

We note some current limitations of our approach. First, the operation steps should be finites. Second, our model is trying to model the action sequence of the skill assessments and the design of the content is not our concern such as some assessments related to the ability of creativity or design skills (i.e., design ability of an architect).

Chapter 6. Conclusions

In this thesis, we have showed that the software run-time status and the transitions of the PBT tester can be described by a set of regular grammar, called the Functional Specification Language (FSL), to model the action sequence performed by the examinee during the software skill certification. Based on the concept, a generator-based approach including *construction phase* and *testing phase*, called the Tester Generator Scheme, has been proposed to assist teachers in building a tester for PBT. In the Construction phase, the available action sequence the examinee can perform is firstly transformed into the corresponding FSL. Thus, a parser generator can be applied to generate the required tester based on the given FSL. In the testing phase, the examinee can perform a sequence of actions on the tester to complete the required tasks.

To reduce the effort of editing the XML files for the input of the generator, we are going to develop an authoring tool with a user-friendly UI to help authors edit XML file in the near future. In addition, the computer-based skill assessments has become very popular, we are trying to apply our model to different software operating skill exams such as MS Excel and PowerPoint.

Reference

[1] Microsoft Office Word 2003 Expert Certification Exam [cited 2008 March]; Available from: http://www.microsoft.com/learning/mcp/

[2] The ANATOMY of a Performance-Based Test [cited 2008 March]; Available from: http://www.certmag.com/issues/may01/feature_mulkey.cfm

[3] Performance-Based Testing: Proving Your Skills [cited 2008 March]; Available from: http://www.certmag.com/issues/nov02/feature_childers.cfm

[4] The State of Performance Based Testing [cited 2008 March]; Available from: http://gocertify.com/article/PerformanceBasedTesting.shtml

[5] A. Basu, I. Cheng, M. Prasad and G. Rao, “Multimedia Adaptive Computer based Testing: An Overview,” *IEEE Int'l Conference on Multimedia, pp. 1850-1853, July, 2007.*

[6] J. Yau and M. Joy, "Adaptive Learning and Testing with Learning Objects," *International Conference on Computers in Education, 2004.*

[7] Parshall, C. G., Spray, J. A., Kalohn, J. C., and Davey, T. (2002). Practical Considerations in Computer-based Testing. Springer-Verlag, New York.

[8] Fritz Drasgow, “Innovative Computerized Test Items”, *Encyclopedia of Social Measurement, pp283-290, 2005.*

[9] Ackerman, T. A., Evans, J., Park, K.-S., Tamassia, C., and Turner, R. (1999). Computer assessment using visual stimuli: A test of dermatological skin disorders. In Innovations in Computerized Assessment (F. Drasgow and J. B. Olson-Buchanan, eds.), pp. 137 150. Erlbaum, Mahwah, NJ.

[10] Bejar, I. I., and Braun, H. I. (1999). Architectural simulations: From research to implementation. *(Research Memorandum 99-2). Educational Testing Service, Princeton, NJ.*

[11] Clyman, S. G., Melnick, D. E., and Clauser, B. E. (1999). Computer-based case simulations from medicine: Assessing skills in patient management. In Innovative Simulations for Assessing ProfessionalCompetence (A.Tekian, C. H.McGuire, and W. C. McGahie, eds.), pp. 29-41. University of Illinois, Chicago, IL.

[12] Zenisky, A. L., and Sireci, S. G. “Technological innovations in large-scale testing,” Applied Measurement in Education, 15(4), 337-362, 2002.

[13] Kathleen, S. and Bernard G., Computer-Based Assessment in E-Learning: A Framework for Constructing "Intermediate Constraint" Questions and Tasks for Technology Platforms, *Journal of Technology, Learning and Assessment, Vol.4, No.6, 2006*

[14] John J. N., and Danette W. M., “Assessment methods in medical education”, *Teaching and Teacher Education, Volume 23, Issue 3, April 2007, Pages 239-250*

[15] Microsoft Simulation Question [cited 2008 March]; Available from: http://www.microsoft.com/learning/mcpexams/simulations/

[16] QTI, IMS Global Learning Consortium, Inc. Question & Test Interoperability [cited 2008 March]; Available from: http://www.imsglobal.org/question/

[17] IMS Global Learning Consortium, Inc.” IMS Question and Test Interoperability Implementation Guide: Items Version 2.1” [cited 2008 March]; Available from: http://www.imsglobal.org/question/qtiv2p1pd2/imsqti_implv2p1pd2.html

[18] Kinnersley, N., Mayhew, S., & Hinton, H. S. (2001). The design of a web-based computer proficiency examination. *In 31st Annual Frontiers in Education Conference (Vol. 2, pp. F2C-3-7).*

[19] Levine, John R., Tony Mason and Doug Brown [1992]. Lex & Yacc. O’Reilly & Associates, Inc. Sebastopol, California.

[20] Adobe Flash; Available from: http://www.adobe.com/products/flash/

[21] Adobe Captivate; Available from: http://www.adobe.com/products/creativesuite/