

Building a Tester Generator for Performance-Based Testing

Nien-Chu Wang^a, Shian-Shyong Tseng^{ab}, Jui-Feng Weng^a, Yian-Shu Chu^a, Huan-Yu Lin^a, Jun-Ming Su^a

^aDepartment of Computer Science, National Chiao Tung University, 30010, ROC

^bDepartment of Information Science and Applications, Asia University, 41354, ROC
sstseng@cs.nctu.edu.tw

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 Spec 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.

Keywords: Performance-based testing, Finite State Machine, Generator

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 that, 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 process 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 [5] 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 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 paper, 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.

2. Related Work

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 [5]. They have conducted that the simulation testing is a more effective way to evaluate the examinee's software operating skills.

Kinnersley et al. [6] used Adobe Flash [7] 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 to the corresponding 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 should be implemented. Therefore, it is very time consuming for a tester developer to build a PBT tester.

3. Tester Generator Scheme

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 1 shows the corresponding MS Word screenshots of the action sequence performed by the examinee.

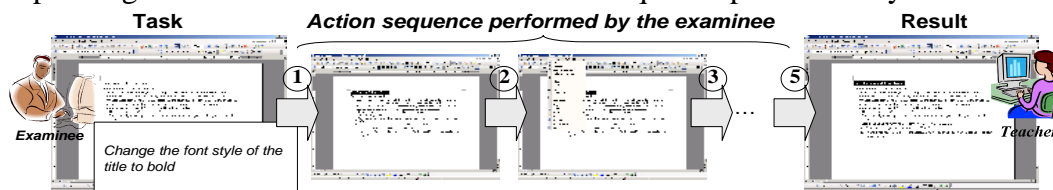


Figure 1. An example of MS Word PBT

With our observations, 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.

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. In this paper, 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 2. The scheme includes two phases: the *Construction phase* and *Testing phase*. In the construction phase, an authoring tool is proposed to help teachers specify the test scenario, including the starting and final states of the task, the sequence of actions that the examinee can perform, and the corresponding software run-time status images. According to the starting and final states of the task, the available action sequence can be transformed into a set of regular grammar, called the Functional Spec Language (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.

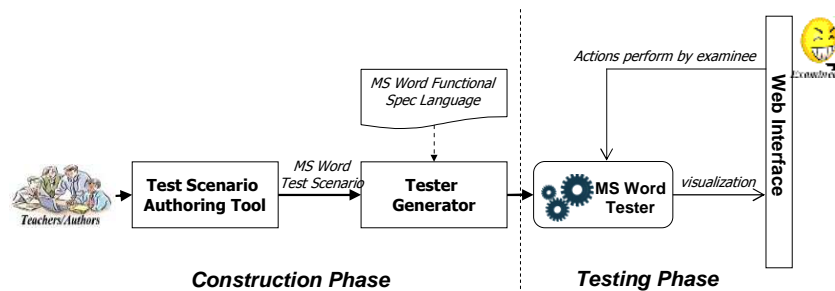


Figure 2. The scheme of tester generator

4. Software Functional Spec Language

The **Functional Spec Language (FSL)** is a set of regular grammar, 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.

Definition: The **Functional Spec Language** is a 5-tuple, $FSL = (N, \Sigma, P, S, \gamma)$, 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 *terminals*, 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:

$$A \rightarrow xB, A \rightarrow x, \text{ where } A, B \text{ in } N, \text{ and } x \text{ in } \Sigma^*$$
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 defines 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 corresponding FSL of the Example 1.

For the performance-based test item given in Example 1, we should define three non-terminals, S , A , and B , representing three different software run-time status during the testing. A set of terminals, letter b to letter 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 corresponding FSL is shown in Figure 3.

$$\begin{array}{l}
 S \rightarrow b_{\#a1 \#a2} S \mid c_{\#a1 \#a2} S \mid d_{\#a2} A \mid h_{\#a2} \\
 A \rightarrow e_{\#a2} B \\
 B \rightarrow f_{\#a1 \#a2} B \mid g_{\#a1 \#a2} S
 \end{array}$$

Figure 3. An example of FSL

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 4.

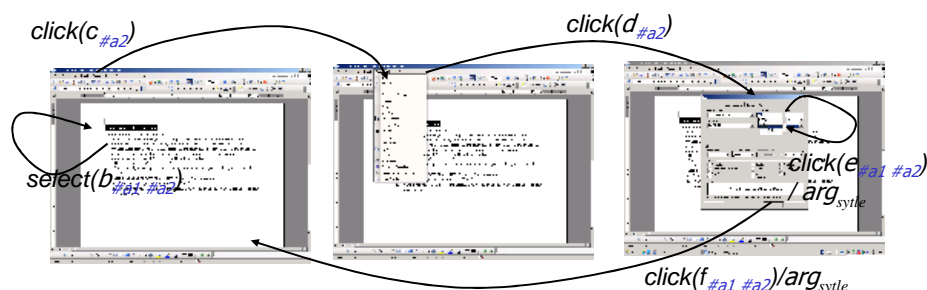


Figure 4. The corresponding FSM of Example 2

5. Conclusions

In this paper, 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 Spec 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.

Acknowledgements

This research was partially supported by National Science Council of Republic of China under the number of NSC95-2520-S009-007-MY3, NSC95-2520-S009-008-MY3, and NSC96-2522-S-009-002.

References

- [1] Preparation Guide for Microsoft Office Specialist: Word 2003 Expert, <http://www.microsoft.com/learning/mcp/officespecialist/objectives/Word2003expert.msp#EGE>
- [2] The ANATOMY of a Performance-Based Test, http://www.certmag.com/issues/may01/feature_mulkey.cfm
- [3] The State of Performance Based Testing, <http://gocertify.com/article/PerformanceBasedTesting.shtml>
- [4] Performance-Based Testing: Proving Your Skills, http://www.certmag.com/issues/nov02/feature_childers.cfm
- [5] Microsoft Simulation Question, <http://www.microsoft.com/learning/mcpexams/simulations/default.msp>
- [6] 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).
- [7] Adobe Flash, <http://www.adobe.com/products/flash/>