Fiscal Cash Register Embedded System Test with Scenario Pattern

REN Yu[†]

[†]***Faculty of Computer College, Hangzhou Dianzi University, Zhejiang, 310018 P. R. China***

Summary
This paper presents a new way that testing fiscal cash register embedded system with a scenario pattern-based method. It also introduces to some concepts of scenario and scenario patterns, and formalizes the embedded system with scenario model ACDATE. The verification patterns are built the fiscal cash register embedded system, and are tested with the samples of basic scenario pattern and complex scenario pattern. This paper shows the whole process how to verify the embedded system with scenario patterns.
Key words:
Embedded System, Scenario Pattern, Verification Pattern, ACDATE, Fiscal Cash Register

1. Introduction

The number of embedded systems in areas such as telecommunications, automotive electronics, office automation, and military applications is steadily growing[9][10]. How to test the embedded systems which contains many functions in short time, and how to evaluate the quantity of each embedded system are a key problem to must be considered when build embedded systems both producers and customers. Designers can no longer develop high-performance from scratch but must use sophisticated system modeling method[1]. Software testing methods and objectives differ in the other computer software applications. Embedded software development uses specialized compilers and development software that offer means for debugging. Programmer development application software on more powerful computers and test the application in the target processing environment. By using scenario patterns the system necessary of the embedded system can be described clearly. The embedded systems can test the scenario patterns by using some patterns that are called verification patterns[2].

Each scenario pattern of embedded systems has the number of attributes such as pre-condition, reason, post-condition and result, restricted conditions of system and selected time limited[3]. The scenario patterns can be defined from the system requirements specification, which designers can reuse to access the design at all stages of the design process about time setting, and can be defined as reason and result which based on system necessary and relationships that limited in time. Usually, each embedded system test must create a lot of scenarios, but using 8 kind of abstract-test-patterns which defined by Weitek Tsai and Lian Yu's got the coverage probability that reached up 95% to the whole system scenarios[2][5]. With the method of scenario patterns, scenario examples are extracted from the fiscal cash register embedded software at first. Then these differing in scenario examples are divided into different scenario patterns with standard pattern ACDATE. Scenario patterns are analyzed different system scenarios, and a unifying and reusing the script apply scenarios that have the same patterns[7]. This approach has several benefits: more efficacious, save time to debug the system, and reliable model test components. Scenario patterns of the fiscal cash register embedded software are checked with an existing script, and most of them will keep the test continuously. When the system test requirements are changed, it just changes the corresponding scenarios, and finds the matched patterns. For designers, the scenario patterns design with object-oriented program. Design pattern is the bases of reused object-oriented software designing, it also makes the designs successfully and system reused constructions basically and conveniently. Designer and producers apply design patterns in the software test, and make a decision to the advantage of reused software system, which will be the system design and the test quickly and perfectly.

This paper introduces the fiscal cash register embedded software test with scenario patterns method. The organization of the paper is divided into 3 parts. In the first part, it introduces the definition of scenario examples and the expressed method of scenario patterns; in the second part, it analyses the scenario test in the fiscal cash register embedded software, and gives two examples that contains a basic scenario and a complex scenario; in the third part, it concludes the previous jobs and give a point of view on embedded software test with scenario pattern in the future.

2. Scenario and scenario pattern

2.1 Scenario definition

Definition 1: A scenario ε is a atom aggregate <$v,\zeta,\theta,$start, finish >, and$v$ is a order aggregate. Each element of$v$ is called the attribute of ε . θ is the condition constraints of the basic scenario. While start and finish is the beginning time and the finishing time of the scenario ε . Here is defined the basic scenario and complex scenario:

1. A basic scenario is a atom aggregate <$\theta,\zeta,\theta,$start, finish >. The aggregate of the attribute of the basic scenario$v$ together with start and finish is decided by the system monitor.
2. One complex scenario is a atom aggregate <$v,\zeta,\theta,$start, finish >, ζ is the aggregate of other scenario, andθ is the aggregate constraints of other scenario, which is called son scenario aggregate. And$v$ the aggregate of the attribute is deduced by all the son scenario aggregate.
   start = min(s.start), finish = max(s.finish), $\forall s \in \xi$.

Definition 2: The corpora of the scenario are V, which is a infinitude aggregate. It introduces all scenarios which will come out later.

The scenario can be obtained from the system. And they often present the end-user some functions. The scenario may be the basic scenario which can not be divided any more. Basic pattern can only judge weather the system has some function, and it has single function[6].Take a basic scenario for example: A cash register system saves the vendition data correctly, and the constraints condition is the hardware equipment has enough space. And such is the instance of a basic scenario. Correspondingly, a complex scenario can be composed of some sub-scenarios. Commonly, the complex scenario may finish the system functions which the basic scenario can not finish. A complex scenario may be constructed by basic scenario through the logic relation, for example successively order, condition expression, synchronization manner etc.

```
start:
    Scenario 1;      // scenario 1 is the basic scenario
    If ((environment variables setting correctly) &&
        embedded system initialization has finished))
    {
        scenario 2; // scenario 2 is the basic scenario
    }
    else if (environment variables setting incorrectly)
    {
        scenario 3; // scenario 3 is the basic scenario
    }
    else
    {
        scenario 4; // scenario 4 is the basic scenario
    }
finish
```

Most of the scenario pattern test uses the scripts which have been used before, and they may guarantee the continuity of the test. Thus we reuse those scripts which may save much of the debugging time.

2.2 Scenario expression

The scenarios are usually expressed with the standard model ACDATE (Actor, Condition, Data, Action, Time, Event)[2][9]. ACDATE model is a standard model, so each embedded system standardizes ACDATE model when the embedded system want to express scenarios at first. Then the scenarios of ACDATE model will be standardized in the embedded system.

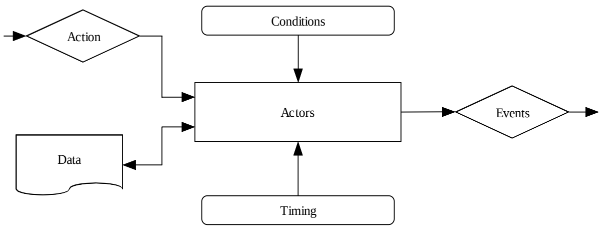

Figure 1 ACDATE defining scenario pattern

There in to:

Actors – an actor is among an external user, system, device, an internal system, device, component and object;

Conditions – a condition is a predicate used to trigger an action;

Data – the attribute of actor, and presenting the semantic of condition, event and action;

Actions – specified by the trigger event, guard condition, the way to change the status of actors, and sent event(s) to some actors;

Timing – a semantic statement about the relative or absolute value of time or duration;

Events – external/internal significant occurrences that may trigger action(s).

Taking an example as the fiscal cash register embedded software; each fiscal task can be treated as an *Actor*. Each actor (fiscal task) may have its own *Data* such as “some merchandise trade data” and its own *Conditions* such as if there is its fiscal item. The given condition example is constructed on the Data “there exists the merchandise fiscal item”. An *Event* could be fired when there is not such merchandise fiscal item. An *Action* would be “extend mange extensions”.

A system scenario would then be specified with the ACDATE model elements: if the event “out of the mange items” occurs, the actor “fiscal task” shall perform action "extend the mange items" within the time specified by the *Timing Attribute* "within 20 days".

3. Scenario pattern test in fiscal cash register system

There is a common abstract testing pattern that was been mentioned in the paper of Feng Zhu[2]. Usually we can make some testing template by using it to test the embedded software. There are 8 kinds of abstract testing patterns. They are basic scenario patterns; key incident driving scenario patterns; time-limits key incident driving scenario patterns; key incident driving scenario patterns with time slice; order and respond scenario patterns; pattern-choose scenario patterns; review scenario patterns and interlock scenario patterns. For example, we can compare two scenarios for fiscal cash register embedded software:

Example 1: when the boot of the fiscal cash register embedded software is finishing, the horn will have a short sound two times;

Example 2: if the bill printer has some problems, for example more heat of loss some pieces of paper, the printer will be locked.

If these two examples have the same scenarios, the test workers can make a test template to test these scenarios that have the same patterns. What mentioned above is a normal example. The scenario patterns are used to describe the different system scenarios, and the test script that is unified and reused can test the scenario that has the same patterns. This will be saved lots of human power in embedded software test.

3.1 Basic scenario patterns test

In the fiscal cash register embedded software, the basic patterns have the most useful and have the most coverage probability. It can be described that when A have a pre-condition, B always correct before time T. As to the fiscal cash register embedded software, the drive of printer can make the printer have some functions that include two-sides logic mapping; the compensation of acceleration displacement; the test of printer's temperature and loss papers; the explanation of ESC/P; 16*16 lattice print between Chinese and ASCII codes. The LCD display drive mainly solve a problem that about chars and data. When send chars, chip-select, the writing signals and data must have the time synchronism in bus. It must make the data locked to an I/O port within 5ms after the succession is send, or the LCD will be out of order; if the hardware's time orders is satisfied, the problem of this mistake will be solved. In this scenario, we can lock the access data on to a I/O port, and we can solve the problem by restricts the LCD's drive in a time. If all scenario patterns test completed, the LCD functions will be passed. On the contrary, the drive is failed. It is showed in Figure 2.

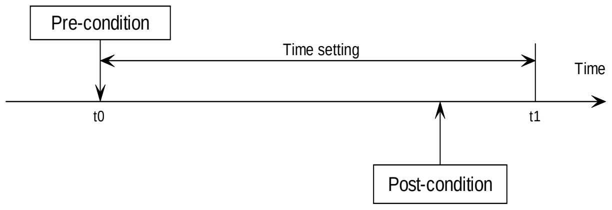

Figure 2 test with basic scenario patterns

As compared with basic scenario patterns are the basic test patterns. The basic test patterns provide a unified and verifying pattern in a basic scenario pattern. It verifies the content that is conveyed by the basic scenario patterns. It has single pre-condition, post-condition and selected time limits. Figure 3 shows a necessary logic state. This state can be applied to the entire needed items that can be classified to a pattern and added in the test processing.

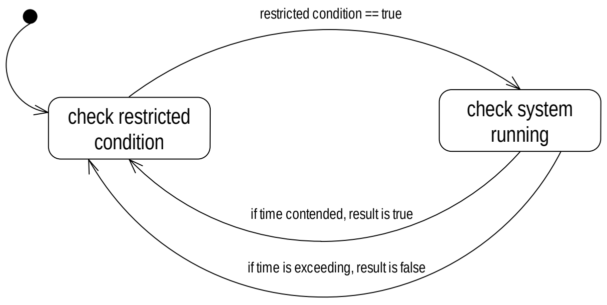
restricted condition == true
check restricted condition
check system running
if time contended, result is true
if time is exceeding, result is false

Figure 3 verifying logic states with basic verifying patterns

3.2 Complex scenario pattern test

In the fiscal cash register embedded software, although the basic scenario pattern can test most scenarios of the software, we still need complex scenarios verification pattern to test the software for the purpose of the coverage of the test. There are still seven complex scenarios verification pattern excluding the basic scenario verification pattern. In out target systems, we can distill different complex scenarios and then match them with these seven scenario verification patterns so that we can finish the software test with those verification templates which have been done. Take the fiscal cash register system software for examples; the saving of power-off is the most important thing. But actually, for some anticipation cause, the power turns off all of sudden. The system should save the manage data, fiscal data, sorts of parameters and correlation registers within 100ms. The system can turn off only after the flash save them talked above. When the systems reset next time, it should check and recover the saved data and then it carry on with the new task. We use the Timed Key-Event Driven Pattern to test the software.

Timed key-event driven requirement pattern requires a duration after the key event occurred. In this pattern, the pre-condition is a combination of the key event and series events that occurred during the specified duration. Look at Figure 4.

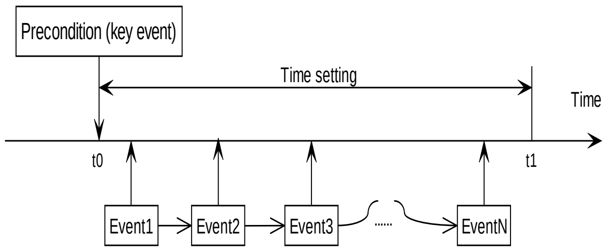

Figure 4 test with complex scenarios pattern

So the scenarios distilled from the fiscal cash register system software match the Timed Key-Event Driven Pattern. The system save the manage data, tax data, sorts of parameters and correlation registers within 100ms. The system can turn off only after the flash save them. The time constraint for this scenario is 100ms. The flash passes the test only after it can save the data with the setting time, otherwise it fails.

The pattern-based verification process follows the steps:

1. Collect the system scenarios and specify scenario patterns using the ACDATE model.
2. Match each scenario with one scenario pattern. For example the two examples mentioned above paragraphs have the same pattern: when the condition is true, if the event E is true, then something happen. If there is not any pattern matching the scenario, we can create a new scenario pattern to specify the scenario.
3. Generate test scripts from verification pattern. So when they finish, we just change the parameter of the code template but not every line.
4. Use the test scripts to test and gain the test result.

We only need change the correlation scenarios when the systems change, and search for the matched pattern. The changed scenarios also adapt to the existing pattern for most of the time.

4. Conclusion

In this paper, we use scenario pattern and by the means of verification pattern and we use the thought of design pattern to finish the test of the fiscal cash register system software. This method has high efficiency. We just need some templates that we may reuse them to finish the test efficiency. And this method has covered most of the software, because all the system scenarios are covered. Using the OOP methodology makes the test much easily. We can expect that this method will develop fast in embedded system in the future.

References

1. D. Gajshi et al., “Specification and Design of Embedded Systems”, Prentice Hall, Englewood Cliffs, N. J., 1994
2. W. T. Tsai, L. Yu, F. Zhu, “Rapid Verification of Embedded Systems using Patterns”, Computer Software and Applications Conference, 2003, P.466 – 471.
3. W. T. Tsai, L. Yu, X. X. Liu, A. Said, Y. Xiao, “Scenario-Based Test Case Generation for State-Based Embedded Systems”, Performance, Computing and Communications Conference, 2003, P.335-342.
4. Schmidt D. C., Huston S D., “C++ network programming (I)-mastering complexity with ACE and patterns”, 2001, 4.
5. Wei Tek Tsai and Lian Yu, Feng Zhu, “Rapid Embedded System Testing Using Verification Patterns”, Software, IEEE Volume 22, Issue 4, Aug. 2005, P.68-75.
5. Xu Hui, Feng Jinwen, Pan Aimin, “One kind of time scenario recognize arithmetic and its applications in the safety alarm system”, Bei Jing University transaction (natural science), Volume 41, Issue 3, May, 2005.
6. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “The basis of reuse object oriented software ”, ［M］translated by Li Yingjun, Mechanic Industry Publishing, 2000.
7. Raymond A. Paul, W. T. Tsal, John S. Mikell, “Rapid simulation evaluation from scenario specifications for command and control systems”, June 2004.
8. Xiaoying Bai, Wei-Tek Tsai, Ray Paul, Ke Feng, Lian Yu, “Scenario-Based Modeling And Its Applications”, Proceedings of the Seventh International Workshop on 7-9 Jan., 2002, P.253-260.
9. Ren Yu, Wan Jian, “Software Simulator of Embedded Application System”, Computer Application, Vol. 11, No. 7, July. 2004, P.144-146.
10. Ren Yu, “Threads Communication Performance of Embedded Simulator”, Computer Application, Vol. 25, No. 25, Dec. 2005, P.12-14.

REN Yu received the B.E. and M.E. degrees, from Zhejiang University in 1985 and 1989, respectively. After working as a research assistant (from 1986), an assistant professor (from 1991) in the Department of Computer Science and Technology, Hangzhou Dianzi University, and an associate professor (from 1999). His research interest includes embedded system and software optimization technology, real-time OS, and system control.