실시간 시스템을 위한 어댑티브 유스 케이스 방법상의 디자인 지향 테스트

Design Driven Testing on Adaptive Use Case Approach for Real Time System

김 영 철* 주 복 규**
Young-Chul Kim Bok-Gyu Joo

요 약

이 논문은 유스 케이스 기반 실시간 시스템을 위한 디자인 지향 테스트를 소개한다. 우리는 디자인 스키마를 기능적 컴포넌트의 계층적 디자인 컴포넌트 아키텍처(즉 디자인 컴포넌트)로 분할 하는 실시간 소프트웨어 개발을 위한 확장 유스 케이스에 초점을 둔다. 실행 순차의 다양한 유스 케이스의 액션 단위를 기술하는 관련된 시나리오를 포함하는, 즉 실시간 시스템 설계의 동적 특성을 반영하는, 유스 케이스 액션 메트릭스를 제안한다. 실시간 시스템을 제안한 디자인 지향 테스트에 적용 시도를 통해 생산성 향상을 위한 시나리오 순서화를 생성하고, 기존의 테스트 케이스 재사용을 진작시킨다.

Abstract

This paper is introduced about Design driven testing, for real time system, based on use case approaches. We focuses on a part of an extended use case approach for real time software development, which partitions design schema into layered design component architecture of functional components called " design component". We developed a use case action matrix to contain a collection of related scenarios each describing a specific variant of an executable sequence of use case action units, which reflected the behavioral properties of the real time system design. In this paper, we attempt to apply real time system with design driven testing with test plan metrics which is introduced which produces an ordering of this scenario set to enhance productivity and both promote and capitalize on test case reusability of existing scenarios.

Keyword : Test Matrix, Design Driven Testing, Adaptive Use Case, Real-time Object Oriented software Development

1. Introduction

This paper is introduced about Design driven testing, for real time system, based on use case approaches[3,5,6]. In this time, we are not interested in automatically generating test data, but providing a framework for test review design in the certain test criteria. Therefore, we focus on the design phase of real time software development activities. Compared to the traditional software development engineering, we deal with testing the design for preventing a little problem from propagating it later on, that is, reducing the cost of testing at the design phase of real time object oriented software development. First, we extend use case interaction diagrams which are included with Branch, Fork-Join, And-Or gate notation, and so on, for real time system, concurrent system, and telecommunication. Second, hurlburt's notion of an action unit needs to be refined from/with a conceptual analysis of the method sequences found in extended interaction diagrams, which, by the way, is the first significant artifact produced during use case design.

Our preliminary analysis of this issue has resulted

* 정 회 원 : 홍익대학교 전자전기컴퓨터공학부 조교수
bob@hongik.ac.kr(제 1저자)
** 정 회 원 : 홍익대학교 과학기술대학 교수
bkjoo@wow.hongik.ac.kr(공동저자)

in the introduction of the concept of a "design component". Several definitions for the design components have emerged from our research for the designer or tester to choose from depending on the level of abstraction desired and the preference for testing techniques to be applied. Even some definitions of this component design unit guild to generate skeleton code through state transition table, and help to generate test cases with an action matrix which is converted from the interaction diagram.

Several possible definitions of design units are introduced, each processing different testing characteristics. The design driven testing with an action matrix and test plan metrics[2] is defined for the purpose of generating a preliminary test plan. The paper is organized as follows: layered design component architecture on extended interaction diagram is described in Section 2. An action matrix produced from the interaction diagram is described in Section 3. We mention test plan metrics, and test plan generation with an application of real time UPS system in Section 4 and 5 respectively. Conclusion and summary are given in Section 6.

2. Layered Design Component Architecture

This idea is based on real time object-oriented behavioral design which partitions design schema into layered functional components called "design component".

2.1 Definition of Design Components

One of the first artifacts produced during the design component phase is the extended interaction diagram based on use case methodology. The message sequence defined by the sequential diagram can be partitioned into a sequence of design component. As a result, different testing techniques may be appropriate depending on the choice of deign component. Based on the layered architecture, we can also identify the reusable design component through real time object-oriented behavioral design of adaptive use case methodology [12,13,14].

Method component: method executed by an object in response to a message. Consider the simple sequential diagram described in Figure 1.

Reusable Pattern Component: Sequence of methods executed by a particular object pattern.

Illustrate a possible grouping of methods based on (presumed) reusable pattern components. This simple vending machine example contains just reusable patterns figure 2.

The SISO pattern is preferred because it simplifies the testing process. However, that may not always be possible. What we don't want it for there to be a cross-product of input/output, i.e., all possible combinations M*N in a MIMO pattern. If, however,

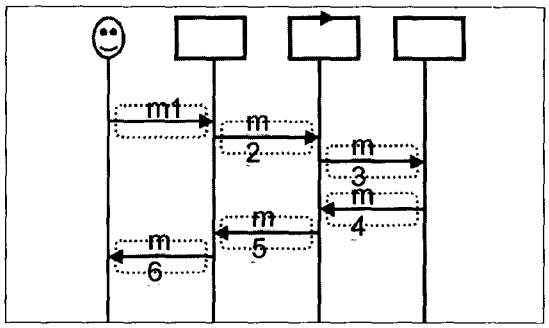
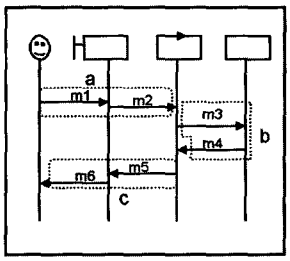


Figure 1. Example of a Method Component



Vending machine:
- method:
  m1: insert coin
  m2: check coin
  m3: measure the weight of coin
  m4: judge what coin
  m5: notify
  m6: notify
- reusable pattern unit:
  a = m1+ m2;
  b = m3 + m4;
  c = m5 + m6;

Figure 2. Example of a Reusable Pattern Component

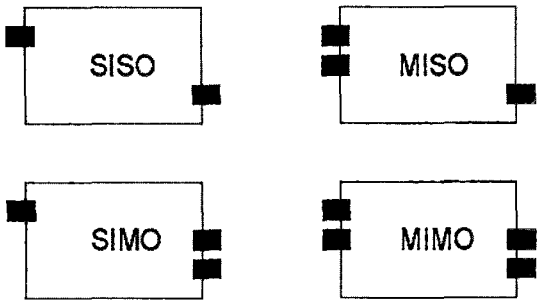


Figure 3. Characteristic interface of SISO, MISO, SIMO, MIMO

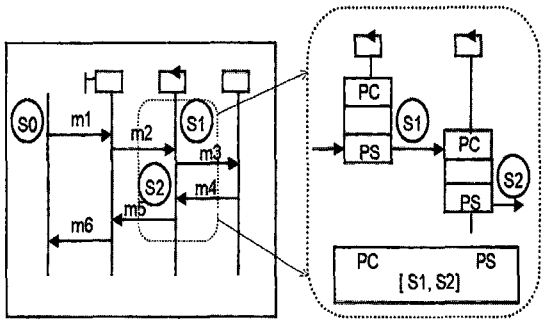


Figure 4. Example of State Components

each input & output are uniquely paired {1-1}, then MIMO can be reduced to a set of SISO protocols as follows: {1-1} {2-2} {3-3} in MIMO in Figure 3.

State Component: sequence of methods executed during the interval bounded by consecutive states as defined by the corresponding event state model.

The state boundary based testing will check the precondition of a state pair (Si, Sj). As a result of the condition (that is, PC [S1, S2|S3] PS1|PS2) it will transfer to either S2 or S3 in Figure 5.

Maximal Linear Component (MLU): a sequence of methods executed during the interval bound by consecutive choices (both actor and object choices).

Consider of this example that an MLU is the same as the dialogue component if no choice/branch nodes exist except for the actor.

Figure 6. includes a choice node for the control object. Thus, the MLU for this interaction diagram

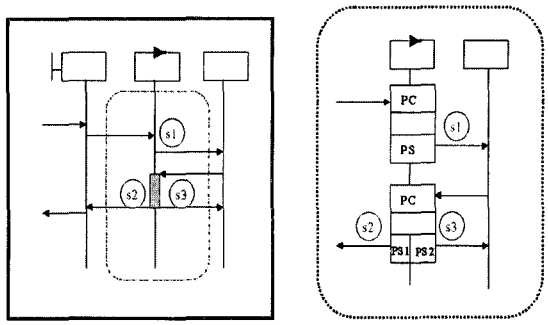


Figure 5. Changing the State on a Decision Node

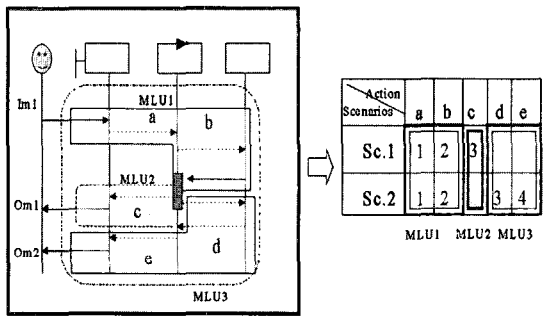


Figure 6. Example of MLUs (Max Linear Components)

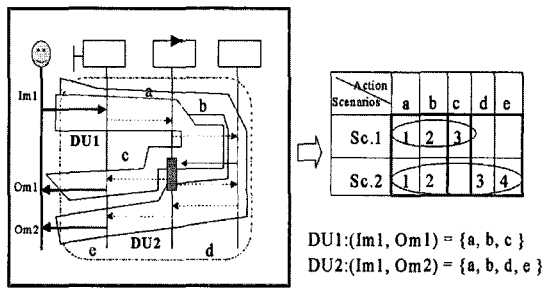


Figure 7. Example of Dialogue Components

is (a,b), (c), and (d,e).

Dialog Component: a sequence of methods executed during the interval bounded by input from an actor and the response to that actor.

In an interaction diagram with the choice notation of Figure 7, sequences of messages/methods are executed during the interval bounded by input from an actor and the response to that actor. We can identify

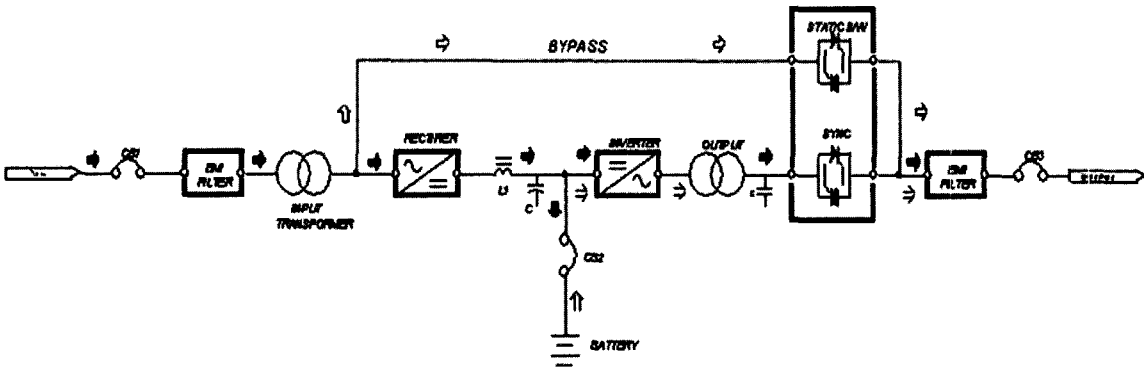


Figure 8. A Circuit of Real Time UPS

all possible paths from an actor through the system to itself such as dialogue 1 ('a b c'), dialogue 2 ('a b d e'), and so on.

3. What is Action Matrix?

From this point, we will show with one case study of modeling a real time 'Uninterruptible Power System (UPS)' use case scenarios through the real circuit of UPS in figure 8.

Focusing on the actor's view, there are five high-level use case scenarios such as the normal status, the normal return, the service interruption, the failure, and the overhead as follows:

a) Normal status: rectifying part and charging part, which receive normal or preliminary power source, shall supply stable AC power by power inverter that switches AC to DC, and shall also charge battery.
b) Service interruption: when normal power service is interrupted, the battery, which has charged by rectifying part and charging part in ordinary time, discharges power to supply DC power to power inverter so that the load can supply stable AC power under no power service interruption for specific discharge time.
c) Normal return; when interrupted normal power is supplied to rectifying part and charging part again, battery suspends its discharge automatically, and good quality normal power is supplied to the load without any service interruption through power inverter and at the same time discharged battery is charged again.
d) Failure: power inverter automatically synchronizes output frequency, voltage and normal power.
e) Overload: power inverter automatically synchronizes output frequency, voltage and normal power.

From the requirement specification and the high-level use case scenario analysis, we design the extended interaction diagrams through passing several steps. In this paper, we will skip several steps to develop five use case interaction diagrams from high-level use case scenarios. With these diagrams in Figure 9, 10 and 11, we can convert the action matrix and use case map dialog through producing these diagrams based on high-level use case scenarios at design stage. Three of these diagrams are 'normal status', 'service interruption', and 'out-of-order' use cases in Figure 9, 10, and 11 respectively.

The use case action matrix is intended to present the scenario designed in a tabular form as the main

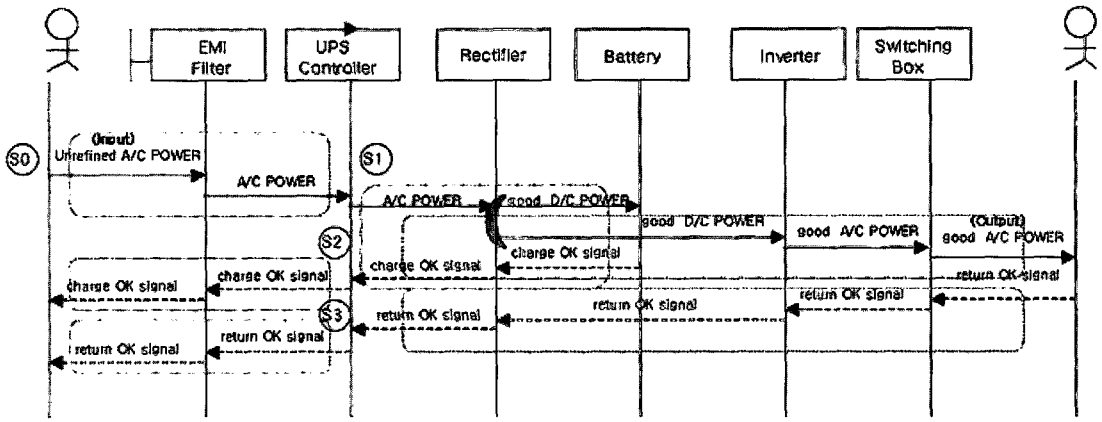


Figure 9. Normal Status Interaction Diagram

Note: Or gate on Rectifier means binary branch concept.

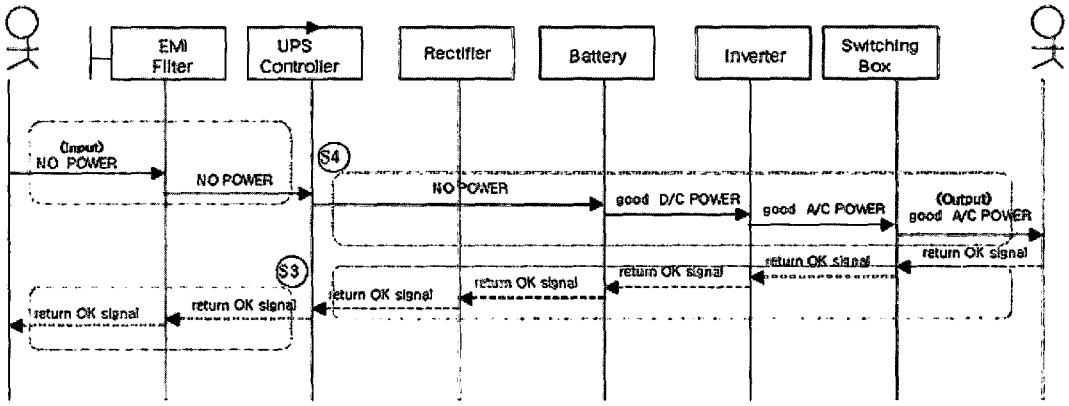


Figure 10. Service Interruption Interaction Diagram

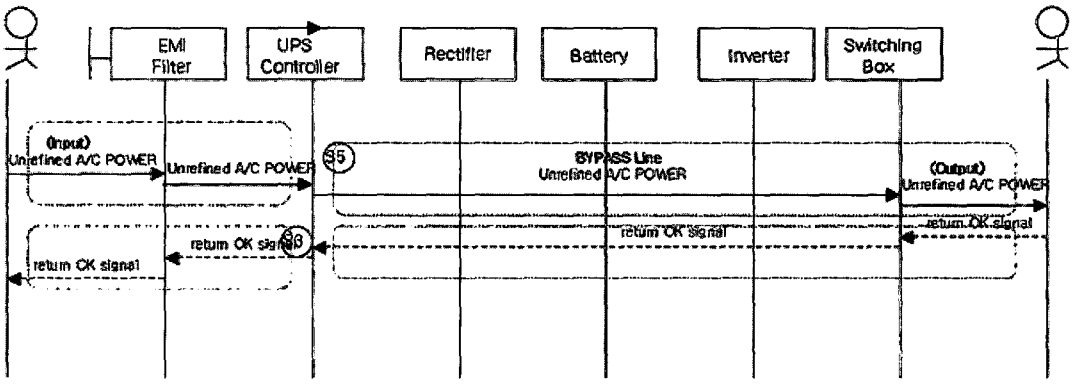


Figure 11. Out-of-Order Interaction Diagram

course of action as a collection of actions that shows the coverage of its use case action by all the

Action unit / Scenario	a1	b1	c1	d1	e1	f1	g1	h1	i1	j1	k1	l1	m1	n1	o1	p1	q1	r1	Total probability of occurrences
Main Path (Normal status)	1					2'	2			3'		3	4			5		6	0.6*1*1*1*1*1*1*1 *0.9*1*1*1*1*1=0.54
Variant 1 (Normal return)		1				2'	2			3'		3	4			5		6	0.3*1*1*1*1*1*1*1 *0.9*1*1*1*1*1=0.27
Variant 2 (Service Interrupt))					1				2		3		4		5			6	0.05*1*1*1*1*1*1*1 *0.05*1*1*1=0.0025
Variant 3 (Out-of-Order)			1					2					4	3		5		6	0.024*1*1*1*1*1*1*1 *0.05*1*1*1=0.0012
Variant 4 (Overload)				1				2					4	3		5		6	0.026*1*1*1*1*1*1*1 *0.05*1*1*1=0.0015

Figure 12 (a) Action Matrix for Real Time UPS Application

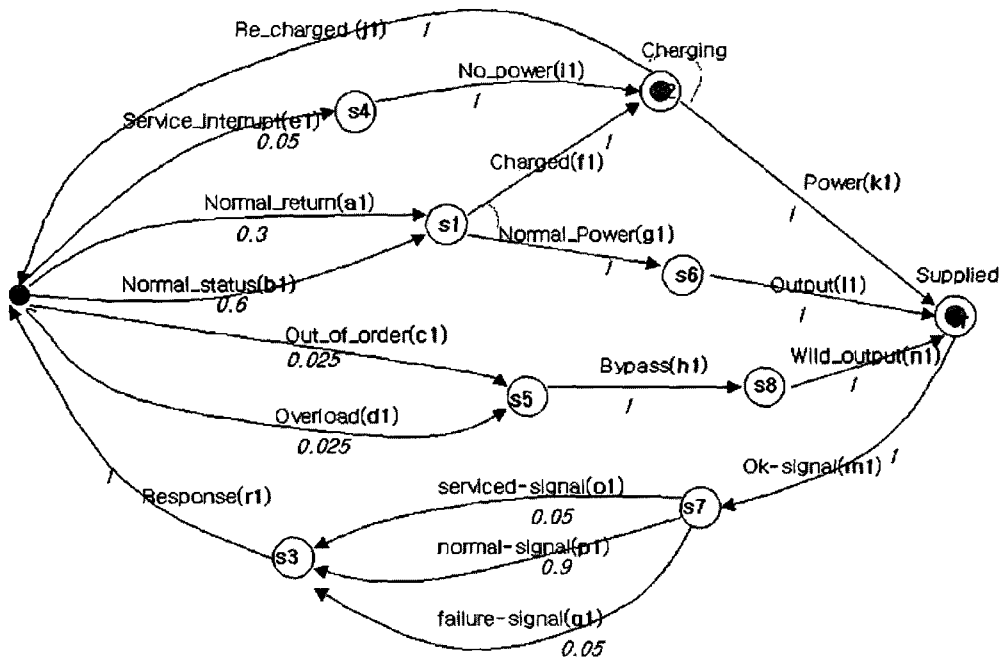


Figure 12(b) Use Case Dialog Map for Real Time UPS Application

variant scenarios in Figure 12 (a). Hurlurt also mentions that an alternative representation of the matrix is a use case dialog map in Figure 12 (b).

Musa's operational profile is frequently weighted by criticality that reflects both how the system is being used and the relative importance of the uses. We may either guess the probability of occurrence on each branch of the specific node (action) or survey the collection of data [3,9,10,12,13].

4. Design Driven Testing

Most Object-oriented software testing methods have been developed for object oriented programs based on white-box testing. None of the existing methods of testing 'design' at the design stage can directly be applied for real time object oriented development methodologies. Our design driven testing approach emphasizes testing software behavioral design speci-

fications in the design stage, which bring the metric concepts to the software development in that the motivation for metrics should focus on reusable numbers & ordering of scenarios on testing.

4.1 Test Plan Metrics

This section focuses on the software testing metrics used in the generation of object oriented test plans as part of Carlson's use case methodology [3]. The test plan uses an action matrix that contains a collection of executable sequences of use case action units (called scenarios). The action matrix is generated from the interaction diagram at the design stage. Software testing metrics are employed to improve the productivity of the testing process through scenario prioritization.

The purpose is to 'optimize' the order in which the scenarios defined by the rows of the action matrix are executed. This approach was adopted from Musa's work on Operational Profiles [9,10]. Musa's approach assumes that the designer has sufficient insight to assess the 'criticality' of action units and assign weighting factors to the elements of the action matrix [7,8]. This approach differs in that the designer analyzes the scenarios based on the 'reusability' of their components or subpaths.

Table 1 illustrates the test plan metrics such as most critical scenarios, most reusable components, and most reusable subpaths. The software test metrics described in this section focus on the length, criticality and reusability properties of the scenarios / action units as summarized in Table 1.

First, the issue of Length is two aspects of shortest path and longest path. I think it is not important for software design development. But it is useful if we use this issue with other categories of the metrics.

Second, the issue of Criticality is important to choose an ordered list of test scenarios.

Third, the issue of Reusability is also important to identify and maximize the reusable components. Therefore, we use scenarios and action units to develop a new path (i.e., scenario) with the smallest number of alterations from the existing paths.

To apply test plan metrics for each of the approaches described in Table 1 will be applied to the real time "UPS(uninterruptible power system)" application.

We calculate total probability of occurrence as follows:

$\forall_i$ Use case $Scenario_i \subseteq$ A Use Case R (R is Real Time UPS Application)

For all use case scenarios between the starting point and the ending point,

the particular scenario $Scenario_i$ is included in a Use case R.

$\forall_i$ action $unit_i \subseteq$ use case $Scenario_i$

For all action units within a particular use case $Scenario_i$

we can calculate the total probability of occurrence with

($\prod$ the weighed factor of Action $unit_i$ * probability of action $unit_i$) / ($\sum$ probability $_i$).

Figure 12 (b) shows the alternative representation of the action matrix, the use case dialog map, to apply the calculation of the total probability of occurrence in each use case scenarios. Figure 12 (a) shows tabularly all possible action units of each use

Table 1. Test Plan Metrics

w: weight value] : not

	Measures of test path		Weight value (w)
Length	1) Shortest path (simple path) - least steps of actions		w = 1
	2) Longest path (hardest path) - most steps of actions		
Criticality	**1) Most critical path**		w ≥ 1
	2) Least critical path		w ≥ 0
Reusability	**Component**	**1) Most reusable components**	w > 1
		2) Least reusable components	w ≥ 0 AND] 1
	Sub-path	**Most reusable sub-path**	w > 1

case scenario in the use case restaurant service. The Mealy model and the Moore model are theoretically equivalent, but the Mealy model is a link-weighted model and the Moore model is a node weighted model [Beiz95]. We apply with both weight concepts. As a result, each action unit is assigned a weighted value with the value one and each link is also a probability of occurrence.

Most Critical Scenario

The first metric is an adaptation of Musa's 'most critical operational profile' approach[9,10]. This metric places greater weight on those scenarios that use action units thought to be most critical. It assumes that the designer can make these judgments. Figure 12 (a). shows the action matrix of the real time UPS application. The use case scenarios defined by the rows of this matrix include: normal status (variant 0), normal return (variant 1), service interrupt (variant 2), and out-of-order (variant 3), and overload(variant 4) use case scenarios. The probability of occurrence of each scenario is: variant 0 (0.54), variant 1(0.27), variant 2 (0.025), variant 3 (0.0012), and variant 4 (0.0015). As this result, we can make a decision to

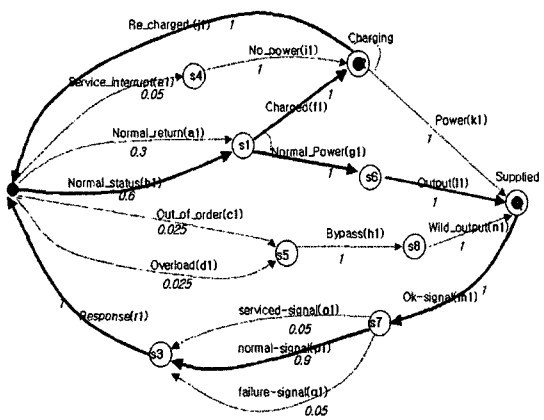


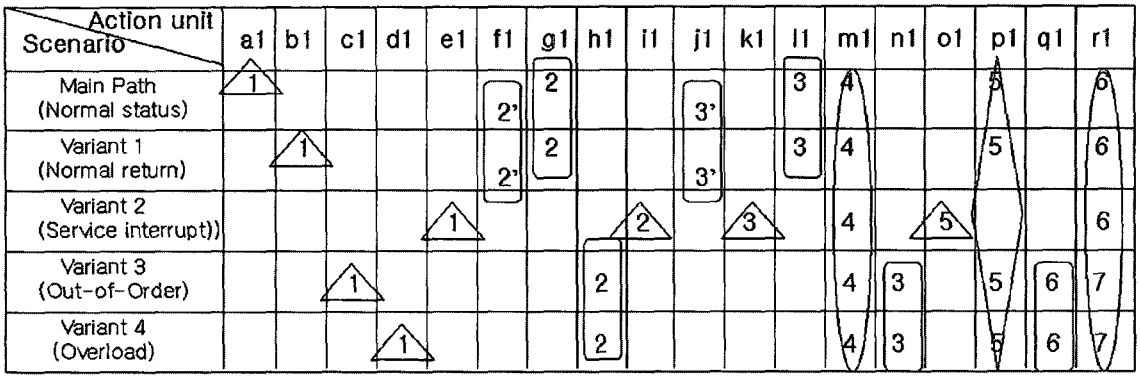
Figure 13. Ordered List of Test Scenarios Most Reusable Components

choose the 'normal return' scenario because it has the highest value of probability of occurrence. Figure 13 displays the ordered list of test scenarios as follows: the first direct path of 'normal status' scenario which consists of the sequence of action units 'b1->((g1->l1->m1->p1->r1)||(f1->j1)', the second direct path of 'normal return' scenario which consists of sequences of action units 'a1->((g1->l1-> m1->p1->r1)||(f1->j1)), the third direct path of 'service interrupt' scenario which consists of sequence of action units 'c1->d1->e1->g1->h1->i1->j1->k1->l1->m1->n1->o1->p1', the fourth direct path of 'outof-order' scenario, the fifth direct path of 'overload' scenario, and other combinations in Figure 13.

Figure 14 (a) displays three different types of geometric figures: a triangle, a rounded rectangle, an oval, and diamond. The triangle implies a particular component is used just one time on just a single one of the paths. The rounded rectangle implies that this component is used on two paths. The diamond implies that this component is used on four paths. The oval implies that this component is used on five paths. The reusability weight is defined as the number of paths that use the particular component.

Therefore, Figure 14 (b) shows the values 'reusability weight' of each action unit. The values can indicate whether a particular action unit is reusable or not. We may say that the unit action is reusable when the value of the particular unit is at least 2.

Figure 14 (c) indicates the total values of reusability components on each path (scenario). Due to the 'most critical scenario', we say that path 1 (normal status) and path 2 (normal return) are better than path 4 (out-of-order) and path 5 (overload), which are better than path 3 (service interrupt). But if we measure each path based on the 'most reusable component', then we recognize that path 1 and path 2 are more usable than other paths.



(a)

	a1	b1	c1	d1	e1	f1	g1	h1	i1	j1	k1	l1	m1	n1	o1	p1	q1	r1
Each Weight Value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Reusability Weight	X	X	X	X	X	2	2	2	X	2	X	2	5	2	X	4	2	5

(b)

	Path 1	Path 2	Path 3	Path 4	Path 5
#2	4	4		3	3
#4	1	1		1	1
#5	2	2	2	2	2
Total Amount	22	22	10	20	20

(c)

Figure 14. Most Reusable Component

Most Reusable Sub-Paths

This metric is similar to the previous metric except that it places greater weight on scenarios which share common subpaths. Figure 7 shows one example how to identify each cluster of the sequence of re-usable components in all possible scenarios of the real time 'UPS' use case application. Figure 15 shows several different types of geometric figures: an dotted shaded elliptical figure and a shaded elliptical figure. The elliptical figure shows the cluster over two paths

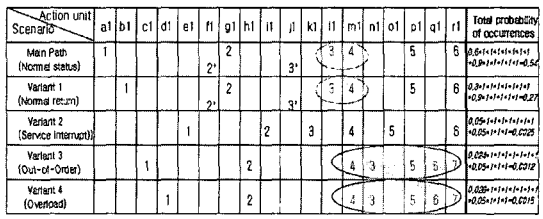


Figure 15. Most Reusable Subpaths Length of Path

with reusable subpaths. The shaded elliptical figures show iteratively or repeatedly the cluster of reusable subpaths in paths. The dotted one displays the smallest

cluster, which consists of two components, in paths, but it is less useful because this size is smaller than the smallest dialog unit within this application. Figure 15 displays the core pattern (cluster) in the use case dialog map.

On path1 and path2, we can see the 'longest reusable subpath' which is 'm1' through 'r1' represented by the ellipse. On path1, path2, and path3, we can see the reusable subpath 'l1' through 'm1', represented by the dotted shaded elliptical figures.

- In reality, we can use this metric to prioritize the important paths. After done by most critical scenario, we had better apply this metric to recognize the most important subpath. Therefore, we may also use this metric of the shortest and the longest path on the concepts of most critical scenario and most reusable component. As a result, we can clearly determine a basic main path, by first making an ordered list of all paths.

6. Conclusion

We introduce our approach how to apply testing design specification during the design stage of real time object-oriented development. Focusing on extended interaction diagram, which represents the behavioral properties of system design, there is actually testing "specifications" without source codes. Design driven testing will make a decision to order of all possible scenarios to test first, to maximize reusability, and to minimize test cases for real time application system. Our proposed testing approach [12,13,14] is also very well applied for modeling real time object-oriented system. We will consider to separate action component with sub-action components which is represented with some nested mechanism for handling concurrency, telecommunication, real time system.

References

[1] Breizer, Boris, "Black-Box Testing", John Wiley & Sons, Inc, NY, 1995

[2] Byun, k. Use case based approach to algorithm event state table, Ph.D, Thesis Illinois Institute of Technology, Chicago, IL 1999

[3] Carlson, C.R. "Object-Oriented Information Systems: Architectural Strategies", Viking Technologies Inc., Chicago, 1997.

[4] Firesmith, D. "Use cases: The Pros and Cons," ROAD, Vol.2,No2,pp2-6, 1995.

[5] Hurlbut, R. "Managing Domain Architecture Evolution though Adaptive Use Case and Business Rule Models", PH.D Thesis, Illinois Institute of Technology,1998.

[6] Jacobson, I., et al, "Objet-Oriented Software Engineering: A Use Case Driven Approach", Addison-Wesley/ACM press,1992.

[7] Mealy, G.H. "A Method for Synthesizing Sequential Circuits", Bell System Technical Journal Vol 34, 1955.

[8] Moore, E. F. "Gedanken Experiments on Sequential Machines", In Automata Studies. Annals of Mathematical Studies #34. Princeton,Nj: Princeton University Press, 1956.

[9] Musa, J.D. "The Operational Profile in Software Reliability Engineering: An Overview", AT&T Bell Labs. NJ,1992.

[10] Musa, J.D. "Operational Profile in Software Reliability Engineering: An Overview", AT&T Bell Labs. NJ,1993.

[11] Marick, Brian, "The Craft of software testing: subsystem testing including Object-Based and Object-Oriented testing", Prentice Hall Series, NJ, 1995.

[12] Kim, YoungChul, Carlson, C.R. "Scenario based integration testing for Object-oriented software

development", IEEE The Eighth Asian Test Symposium (ATS'99), November 16-18, 1999, Shanghai, China.

[13] Kim, YoungChul, Carlson, C.R "Adative Design Based Testing for OO Software", ISCA 15th International Conference on Computers and Their Applications (CATA-2000), New Orleans, Louisiana, March 2000

[14] Kim, R. YoungChul, et al, Scenario Based Testing & Test Plan Metrics Based on a Use Case Approach for Real Time UPS, LNCS2668, Montreal, Canada, May 2003.

◐ 저 자 소 개 ◑

김 영 철

2000년 일리노이공대 전산학과(공학박사)
2000년~2001년 LG 산전 중앙연구소 Embedded System 부장
2001년~현재 : 홍익대학교 전자전기컴퓨터공학부 조교수
2003년~현재 : 유비젠(주) 고문
관심분야 : 도메인 모델링, CBD 방법론, 유스케이스방법론, 컴포넌트(시스템)간의 상호운영성, Design Driven Testing, TMM (Test Maturity Model)
E-mail: bob@hongik.ac.kr

주 복 규

1977년 서울대학교 계산통계학과 졸업
1980년 한국과학기술원 전산학과 졸업(이학 석사)
1990년 University of Maryland 졸업(공학 박사)
1990년~1998년 : 삼성종합 기술원, 삼성전자 중앙연구소 수석 연구원
1998년~2000년 : 동양시스템즈 연구소장
2001년~현재 : 홍익대학교 과학기술대학 교수
관심분야 : Software Reuse, Software Development Methodology, Network Security, Intelligent Systems
E-mail: bkjoo@wow.hongik.ac.kr