LNCS 2668

Vipin Kumar
Marina L. Gavrilova
Chih Jeng Kenneth Tan
Pierre L' Ecuyer (Eds.)

Computational Science and Its Applications – ICCSA 2003

International Conference
Montreal, Canada, May 2003
Proceedings

2 Part II

Springer

Knowledge Acquisition in the Nutri-Fuzzy-ORIXAS Project 589
V.M. Benjamim Werneck, A. Brito F. Oliveira, R. Serrão Lanzillotti, H. Serrão Lanzilotti, E. de Abreu Soares, E. Souza Portella, S. da Silva Ávila

Speech Recognition and Agent Technologies

The Merging Algorithm for an Extraction of Valid Speech-Sounds 599
J.O. Kim, H.W. Paek, C.H. Chung, W.Y. Yim, S.H. Lee

Agent Migration Information System for the Efficient Migration of the Mobile Agent 607
H.-J. Park

Schematic Aspect for Autonomous Agent 614
K.S. Tae

Robust Speaker Recognition Against Utterance Variations 624
J.J. Lee, J.Y. Rheem, K.Y. Lee

Efficient Speaker Identification Based on Robust VQ-PCA 631
Y. Lee, J. Lee, K.Y. Lee

An Amplitude Warping Approach to Intra-speaker Normalization for Speech Recognition 639
K.-S. Hong

Computational Theory and Test and Simulation

Scenario Based Testing & Test Plan Metrics Based on a Use Case Approach for Real Time UPS (Uninterruptible Power System) 646
R.Y.-C. Kim, B.-G. Joo, K.-C. Kim, B.-k. Joen

A Study on Insuring the Full Reliability of Finite State Machine 656
S. Yang, M.J. Kim, J.H. Park, H. Chang

Distributed Multiple-Path Searching Algorithm for Fault Detection 664
S. Kim, S. Ahn, J.W. Chung

An Object-Oriented Simulation System for Air Defense 674
C.-S. Jeong, S.-Y. Choi

Evaluation and Analysis of Computational Complexity for Secure Multicast Models 684
E. Blessing R., R. Uthariaraj

An Algorithm for Diagnosing System with Structured Description 695
S. Luan, G. Dai

Scenario Based Testing & Test Plan Metrics Based on a Use Case Approach for Real Time UPS (Uninterruptible Power System)

R. Young-Chul Kim[1], Bok-Gyu Joo[1], Kyung-Chul Kim[1] , and Byung-kook Joen[2]

[1] College of Science & Technology, Hongik Univ., Korea
[2] Dept. of Computer Inf. management, Wonju Nat'l College, Korea
[1]bob@wow.hongik.ac.kr

Abstract. This paper describes a part of an extended use case approach for real time object-oriented software development. Its foundation is an object-oriented software design approach which partitions design schema into layered design component architecture of functional components called "*design component unit*". A use case action matrix contains a collection of related scenarios each describing a specific variant of an executable sequence of use case action units, which reflects the behavioral properties of the real time system design. Proposed scenario based testing and test plan metric measure and produce an ordering of this scenario set to enhance productivity, and promote and capitalize on test case reusability of existing scenarios. To illustrate the proposed approach uses an example of real time UPS (uninterruptible power system).

1 Introduction

This paper focuses on suggest a little solution for real time system, concurrent system, and telecommunication, based on use case approaches defined by Jacobson [6], UML, Carlson [3], and Hurlbut [5]. 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, focused on the design phase of real time object oriented software development activities against the traditional software development, 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. With extended interaction diagrams [2,12,13] included 'Branch', 'Fork-Join', 'And-Or gate' notations, we refine Hurlburt's notion of an action unit with a conceptual analysis of the method sequences found in traditional interaction diagrams to solve real time, concurrent problems.

Our preliminary analysis of this issue has resulted in the introduction of the concept of a "component design unit". Several definitions [12,13] for the component designs 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 helps to generate test cases with an action matrix which is converted from the interaction diagram.

To provide an automated process by which an action matrix can be produced, thus, we need to develop an algorithm to produce the action matrix from sequential diagrams (called Interaction diagrams). This conversion algorithm[2] is developed to identify or extract each type of design component unit from the extended interaction diagram in which deals with single control object or multiple control objects.

The action unit of an action matrix is defined as specific component design units of the interaction diagram for easily identifying reusable component design and helps generate test plan and test schema with test plan metrics [14]. Several possible definitions of design units are introduced[14,12,13], each processing different testing characteristics. The scenario based testing and test plan metrics with an action matrix is defined for the purpose of generating a preliminary test plan.

The paper is organized as follows: an extended use case 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 scenario based testing with an application of real time UPS system in Section 4. Conclusion and summary are given in Section 5.

2 Extended Interaction Diagrams for Real Time UPS

From this point, we will explain with one case study based on a real time '*Uninterruptible Power System* (UPS)' use case scenarios. 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. When the equipment is out of order or overload, stable power can be supplied to the load under synchronous status with normal power by being switched without any service interruption synchronous switching switch.

e) Overload: power inverter automatically synchronizes output frequency, voltage and normal power. When the equipment is out of order or overload, stable power can be supplied to the load under synchronous status with normal power by being switched without any service interruption synchronous switching switch.

From the requirement specification and the high-level use case scenario analysis, we can design the extended interaction diagrams through passing several steps. Fig. 1, 2, 3 contain the extended diagrams for three use cases associated with real time UPS application. Each use case is described from the point of view of the outside system. In this paper, we will skip several steps to develop five use case interaction diagrams from high-level use case scenarios. With these diagrams, 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 Fig. 1, Fig. 2, and Fig. 3 respectively.

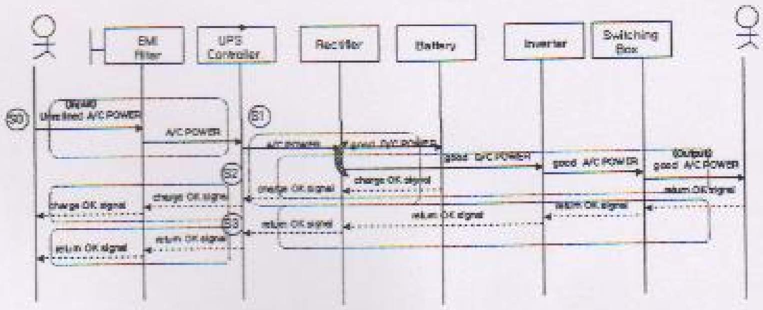


Fig. 1 the Normal Status Interaction Diagram

Note: Or gate on Rectifier means binary branch concept.

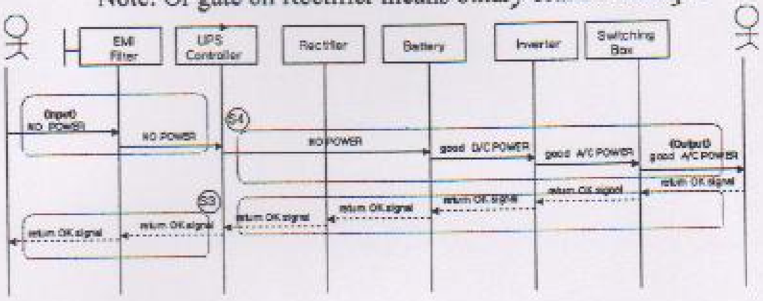


Fig. 2 Service Interruption Interaction Diagram

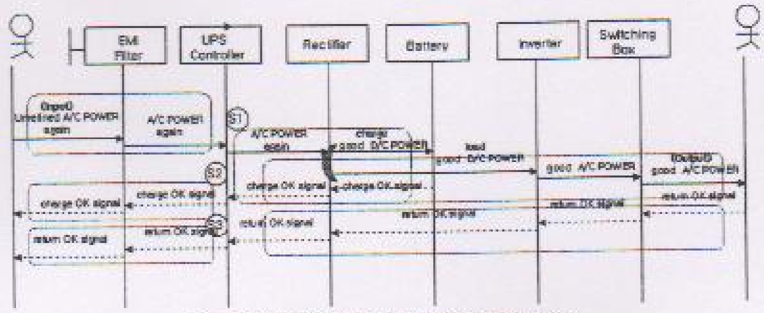


Fig. 3 Out-of-Order Interaction Diagram

3 Action Matrix Approach

What is action matrix? Hurlbut [5] noted that an action matrix presents a cross-match between each action that is included in a use case with each scenario that performs the action. Scenario includes an ordered set of actions that explains its course of actions. The use case action matrix is intended to present the scenario designed in a tabular form as the main course of action as a collection of actions that shows the coverage of its use case action by all the variant scenarios in Fig. 4. Hurlbut also mentions that an alternative representation of the matrix is a use case dialog map in Fig. 5.

Semantics: Each scenario consists of an ordered collection of action units. The extended action matrix is intended to tabularly represent the scenario designed as the main course of action unit as a collection of cells that shows the coverage of its use case action component by all variant scenarios. Each action unit consists of the cluster of the consecutive messages (methods, the basic actions) triggered, also relates between the state and next state, and involves the particular objects which send these messages.

Notation: In The first row of a matrix, each cell is matched to a unique use case action unit and scenario combination. The rows of a matrix are assigned as scenarios and the columns are assigned as action units. If a use case action unit is also included in the scenario, we make a one-to-one correspondence between cells and integers which results in sequential ordering of the use case action units in the scenario. When a single use case action unit appears more than once in a scenario, we can assign multiple not necessarily contiguous integers in each cell. Since a bunch of the consecutive use case action units may appear iteratively in a scenario, we may parenthesize a bunch of integers that are related with consecutive use case units.

Presentation Option: An alternative representation of the matrix is possible by converting use case action units into nodes in a direct graph with the Mealy's and the Moore's Finite State Machine as well as Musa's Operational Profile concept. When the matrix is presented in this fashion, it may be alternatively refereed to as a use case dialog map that is mentioned by Hurlbut [Hurl98].

Mapping: Each column maps to a use case action unit. Each row maps to a scenario. In a use case dialog map, we assign each probability of occurrence to each link as a weighted probability value, which is adopted by Musa's Operational profile [9,10]. Musa's approach is a quantitative characterization of a system, which is retained for the most-used part of the system and is reduced for lesser-used parts with the amount of reduction related to the difference in usage. That is, 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].

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

Fig. 4 Action Matrix for Real Time UPS Application

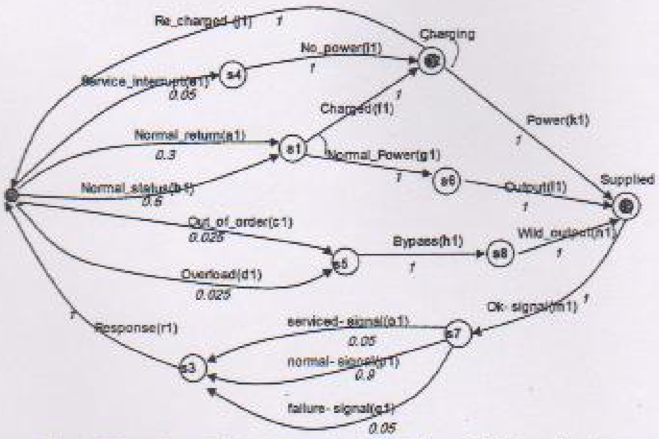


Fig. 5 Use Case Dialog Map for Real Time UPS Application

4 Scenario Based Testing and Test Plan Metrics

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. We propose different metrics for test plan based on design. Our scenario based testing approach emphasizes testing software behavioral design specifications 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. In other words, the software test metrics are available to evaluate the use case scenarios defined by the action matrix so that a test plan will emerge that improves the productivity of the testing process. The tester has a broad range of options to choose from when identifying 'action units'. The simplest choice is to let each method be an action unit. A drawback to this approach is that the number of action units may be quite large. If so, the test plan generator can choose one of the other options based on the design component (or test units) concept introduced. In selecting 'reusable pattern component', the assumption is that reusable pattern design components are also used as part of the design process. In selecting 'state units', the assumption is that an event state model has been developed. Presently, we have a conversion tool that analyzes interaction diagrams, and automatically generates 'state component'. The same tool can be used to automatically identify 'dialogue component'. The basic reason for choosing one of these units is that each offers its own unique approach to unit testing based on proven testing techniques. For example, reusable pattern test plans can be used with reusable pattern. State based testing techniques can be used with state units. Actor based acceptance testing can be used with dialogue components. Scenario based integration testing techniques can be used with use case scenarios to identify an ordered list of test scenarios. User acceptance testing can be used with use case requirements. Based on this choice, the test plan contains a set of action units together with appropriate unit testing techniques to be applied to these units. The software metrics described in the next section can be used to yield a more productive order in which these units can be tested.

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 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).
```

Table 1. Test Plan Metrics w: weight value $\rceil$: not

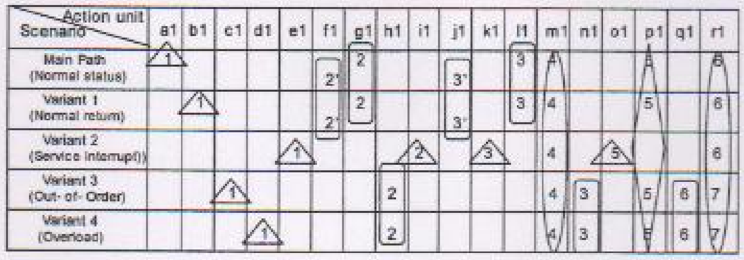
	Measures of test path		Weight value(w)
Length	1) Shortest path (simple path) - least steps of actions		$w = 1$
	2) Longest path (hardest path) - most steps of actions		$W \geq 1$
Criticality	1) *Most critical path*		$W \geq 0$
	2) Least critical path		$W \geq 1$
Reusability	Component	1) *Most reusable components*	$w>1$
		2) Least reusable components	$W \geq 0$ and $\rceil$=1
	Sub-path	*Most reusable sub-path*	$w>1$

Fig. 5 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. Fig. 4 shows tabularly all possible action units of each use case scenario in the use case UPS application. 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. Later metrics will not have to assume that someone is available to make such judgments, since they can be produced automatically. Fig. 5 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 choose the 'normal return' scenario because it has the highest value of probability of occurrence. Fig. 5 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 'out-of-order' scenario, the fifth direct path of 'overload' scenario, and other combinations.

Most Reusable Components. This simply measures the reusability of action units in each row of the action matrix. This metric places greater weight on those action units that are reused the most by the collective group of scenarios being analyzed.

Fig. 6 (a) displays three different types of geometric Fig.s: 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.



Action unit
Scenario
a1 b1 c1 d1 e1 f1 g1 h1 i1 j1 k1 l1 m1 n1 o1 p1 q1 r1
Main Path (Normal status)
Variant 1 (Normal return)
Variant 2 (Service interrupt)
Variant 3 (Out-of-Order)
Variant 4 (Overload)

(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	$\cancel{1}$	$\cancel{1}$	$\cancel{1}$	$\cancel{1}$	$\cancel{1}$	2	2	2	$\cancel{1}$	2	$\cancel{1}$	2	5	2	$\cancel{1}$	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	13	20	20

(c)

Fig. 6 Most Reusable Component

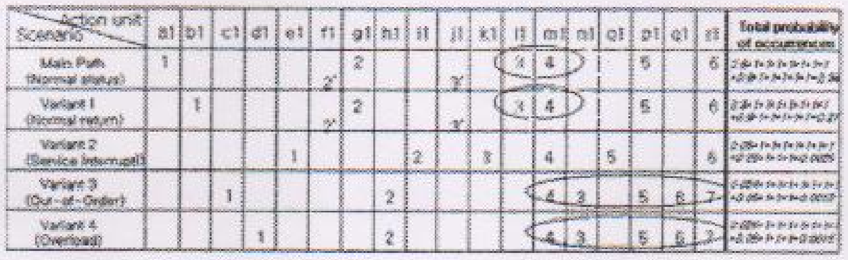
Therefore, Fig. 6 (b) shows the values 'reusability weight' of each action unit. The values can indicate whether a particular action unit is reusable or not. We may say that the unit action is reusable when the value of the particular unit is at least 2.

Fig. 6 (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.

Most Reusable Sub-Paths. This metric is similar to the previous metric except that it places greater weight on scenarios which share common subpaths. Fig. 6 shows one example how to identify each cluster of the sequence of reusable components in all possible scenarios of the real time 'UPS' use case application. Fig. 7 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 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. Fig. 7 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 Fig.s.



Scenario \ Action unit	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	[illegible]
Variant 1 (Normal return)		1				2'	2			3'		3	4			5		6	[illegible]
Variant 2 (Service Interrupt)					1				2		3		4		5			6	[illegible]
Variant 3 (Out-of-Order)			1					2					4	3		5	6	7	[illegible]
Variant 4 (Overload)				1				2					4	3		5	6	7	[illegible]

Fig.7 Most Reusable Subpaths

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.

5 Conclusion

Traditional software testers concentrate on testing the program source code in the implementation stage or testing stage, while our testing approach will emphasize testing software design specification in the design stage. In the design phase of our real time object oriented development methodology, we will focus on testing 'action matrix & use case dialog maps' that is generated from extended interaction diagram, which represents the behavioral properties of system design. That is actually testing "specifications" before implementing real program source codes (program statements). Scenario based 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. As a result, this can lead for designer to design better system with information of reusable design components.

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 "Adaptive Design Based Testing for OO Software", ISCA 15th International Conference on Computers and Their Applications (CATA-2000), New Orleans, Louisiana, March 2000
14. Kim, YoungChul, A Use Case Based Approach to Test Plan Generation During Design, Ph.D, Thesis Illinois Institute of Technology, Chicago, IL 2000