


Tai-hoon Kim
Hojjat Adeli
Rosslin John Robles
Maricel Balitanas (Eds.)

Communications in Computer and Information Science

199

Advanced Communication and Networking

Third International Conference, ACN 2011
Brno, Czech Republic, August 2011
Proceedings

 Springer

Integrated Retrieval System for Rehabilitation Medical Equipment in Distributed DB Environments	209
<i>BokHee Jung, ChangKeun Lee, and SoonGohn Kim</i>	
Effective Method Tailoring in Construction of Medical Information System	215
<i>WonYoung Choi and SoonGohn Kim</i>	
A Study on the Access Control Module of Linux Secure Operating System	223
<i>JinSeok Park and SoonGohn Kim</i>	
An fMRI Study of Reading Different Word Form	229
<i>Hyo Woon Yoon and Ji-Hyang Lim</i>	
Intelligent Feature Selection by Bacterial Foraging Algorithm and Information Theory	238
<i>Jae Hoon Cho and Dong Hwa Kim</i>	
The Intelligent Video and Audio Recognition Black-Box System of the Elevator for the Disaster and Crime Prevention	245
<i>Woon-Yong Kim, Seok-Gyu Park, and Moon-Cheol Lim</i>	
Real-Time Intelligent Home Network Control System	253
<i>Yong-Soo Kim</i>	
LCN : Largest Common Neighbor Nodes Based Routing for Delay and Disruption Tolerant Mobile Networks	261
<i>Doo-Ok Seo, Gwang-Hyun Kim, and Dong-Ho Lee</i>	
A Perspective of Domestic Appstores Compared with Global Appstores	271
<i>Byungkook Jeon</i>	
A Design of Retrieval System for Presentation Documents Using Content-Based Image Retrieval	278
<i>Hongro Lee, Kwangnam Choi, Ki-Seok Choi, and Jae-Soo Kim</i>	
Data Quality Management Based on Data Profiling in E-Government Environments	286
<i>Youn-Gyou Kook, Joon Lee, Min-Woo Park, Ki-Seok Choi, Jae-Soo Kim, and Soung-Soo Shin</i>	
Design of Code Template for Automatic Code Generation of Heterogeneous Smartphone Application	292
<i>Woo Yeol Kim, Hyun Seung Son, and Robert Young Chul Kim</i>	
A Study on Test Case Generation Based on State Diagram in Modeling and Simulation Environment	298
<i>Woo Yeol Kim, Hyun Seung Son, and Robert Young Chul Kim</i>	

A Study on Test Case Generation Based on State Diagram in Modeling and Simulation Environment*

Woo Yeol Kim, Hyun Seung Son, and Robert Young Chul Kim

Dept. of CIC(Computer and Information Communication), Hongik University,
Jochiwon, 339-701, Korea
{john, son, bob}@selab.hongik.ac.kr

Abstract. In the conventional tests, test case is generated in the design stage. However, actual test can be executed after its embodiment. As there is as much time difference between the design and execution of the test, the errors in the designs of test and software are checked out late. This paper is proposing the test case generation method so as automatic test can be carried out in the virtual simulation environment. The method proposed generates the test case automatically based on the state diagram and executes it in the virtual simulator. It can reduce the time difference between the design and execution of test, accordingly, to find out the error in the test case and problems in the design promptly. As a result, it can identify the error in the beginning stage of software development and save the time and expense need for the development.

Keywords: Articulated Robot, Modeling & Simulation, Test, Virtual Environment, Model based Test.

1 Introduction

In the software development life cycle, the expense differs significantly according to the error finding stage. If the cost is 1 for finding and solving errors in the request phase, it costs 30~100 times when finding and solving errors in the production phase [1]. Software errors can be found out when test is carried out. If software test can be executed earlier, therefore, the error can be found fast and the development period and expense can be reduced. However, test can be executed after software development is completed in the conventional tests.

This paper suggests the automatic generation method of test case that can be processed in the virtual simulation environment [2,3,4,5]. The method proposed is creating the test case from the state diagram. Generated test case can be executed directly in the virtual simulation environment. As for the test case generation method, it converts state diagram into state table and generates the testing transition tree based

* This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)(NIPA-2011-(C1090-1131-0008)) and the Ministry of Education, Science Technology (MEST) and National Research Foundation of Korea(NRF) through the Human Resource Training Project for Regional Innovation.

on it. Test case ID is identified and test case is generated using the created transition tree. The generated test case makes up the event list after classifying event and action separately for the processing in the simulator. When processing the event list in the virtual simulation environment, robot is activated and the Pass/Fail is checked out by observing the robot's behavior.

With conventional test, test case is generated in the design stage, but it can be executed after embodiment of software [6]. There occurs time difference as much between the design and execution of the test. However, this method does not wait until the embodiment stage by executing the test case directly in the virtual simulation environment but processing the test in the design stage. So the design error can be found fast in early stage.

This paper is organized as follow. Chapter 2 describes the model based test as a related work. Chapter 3 explains how to generate the test case proposed. Chapter 4 shows the case study of proposed method. Finally in Chapter 5, it mentions about the conclusion and future research.

2 Related Work

Following is the observation of model based test. Andras Toth et al. [7] had proposed the framework for the model level testing of UML model. Planner algorithm has been used for automatic generation of test case. The input of this framework is the UML state diagram exported to the independent XML tool of which format is assigned by UML tool. Conversion program generates text file automatically by the generation of planner model that can be manipulated like the formal language of UML state diagram. Planner meta model has established the provision of high level expression to maintain the methodology opened for other planner tools. The design can be tested, as of the result of this project UML, and the design flow can be found in the modeling stage of primary development process for the realization activity to save significant amount of effort and expenses.

The theoretical background for the extraction of model based test case in the formal conformance testing for UMLSC (UML State-Chart) had been expressed by Stefania Gresi et al. [8].

Bertolino A. et al. [9] have suggested the aggregation method that is integrating the sequence diagram with the state for extracting reference model perfectly and rationally that can be used in the industrial context. This is used to extract the test case automatically. The author had extracted the test case based on the UML specification from all the contexts based on the component and object. The objective of this paper is for processing perfect model from the state and sequence diagrams and for making UIT (Use Interaction Test) together with the model in the research. This method guarantees that it includes all the allowable sequences. However, it cannot provide any coverage measuring on the embodied system. It satisfies the important requirements entrusted by the industry. The advantage of this method is for creating accurate test case and for inducing less testing effort by providing as much inconformity information as possible in the model.

3 Test Case Generation in M&S

The purpose of test case generation based on state diagram is to verify the relations between the event, behavior, action, state, and state transition. With using this technique, it can determine if the state based motion of system can satisfy the system specifications. There are 3 reasons for the fault of state based system. The first is when the state diagram cannot transit the system function specification accurately. The second is when the syntax of state diagram is wrong or inconsistent. The third is the conversion form state diagram to the code. It does not matter if converted using the automation tool, but it may cause troubles if converting manually.

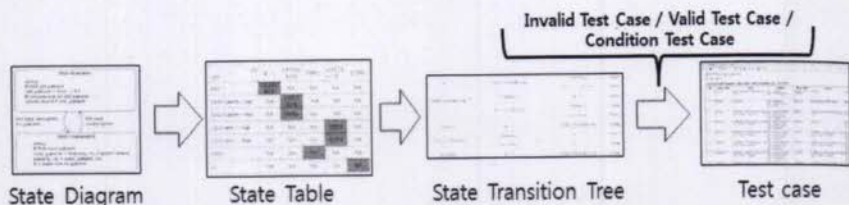


Fig. 1. Flowchart of the test case generation based on state diagram

Figure 1 shows the test case generation based on state diagram using the state diagram. Initially, state diagram model is converted into the state table. State table has been separated using the tables of state and event, which can express the state of all the situations. Then, create the state transition tree based on the state table. State transition tree is the array of movable states in each state recursively. In here, the test case level is varied according to the recursive execution frequency.

Figure 2 shows how to make the state table. The state available for moving toward when meeting with the event in each state can be indicated on the intersection of the table after indicating each state on the top of table and expressing the event on the left side of the table.

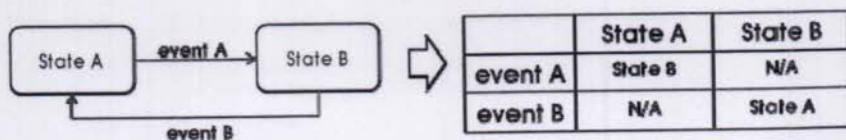


Fig. 2. State table conversion in the state diagram

Figure 3 shows how to create the transition tree. Array all the states on the top of table in the sequence. Then, indicate all the states available for access in the next step. If arraying the state available for re-access, the number of test case generation is varied according to the depth of array. The array of all these cases in the sequence makes the test case.

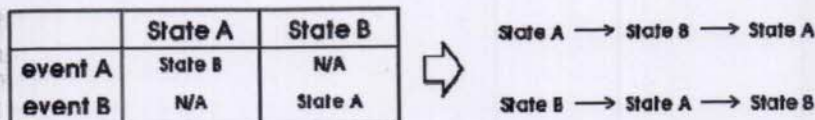


Fig. 3. Creation of state transition tree in the state table

Generate the final test case is using the state transition tree. Generated test cases are shown in the Table 1. Generated test case is describing the scenario executed by each state in the state transition tree.

Table 1. Test case

TCID	Initial State	Event	Action	Next State	Event	Action	End
TC1	S_A	E1	Do	S_B	E2	Do	S_A
TC2	S_B	E2	Do	S_A	E1	Do	S_B

Generated test case cannot be processed right away in the simulator. So convert the test case to the event list as is shown in Table 2 for execution. Table 2 is the conversion result of the test cases generated in the Table 1 to the event list. TCID is the ID of test case. In the Type column, s means state, and e means event. State/Event have been assigned in plural so as both state and event can come in. P/F means Pass/Fail. The state and event in the event list are carried out alternatively, and the robot executes the behavior whenever the event list is processed in the virtual simulator environment

Table 2. Event list

Test Case ID	Type	State/Event	P/F
TC1	s	S_A	
TC1	e	E1	
TC1	s	S_B	
TC1	e	E2	
TC1	s	S_A	
TC2	s	S_B	
TC2	e	E2	
TC2	s	S_A	
TC2	e	E1	
TC2	s	S_B	

4 Case Study

This application case shows the multi-jointed robot [10,11] moving to the target position using the 4 directions of forward, backward, left, and right. Articulated robot is composed with the state diagram shown in Figure 4. 'Idle' is the initial state of robot, which is the robot state when activated initially or all the works have been finished. 'Initialized' means the resetting state of robot. The target of robot is assigned

and the coordinates are set up in this state. 'goRobot' means the state that robot is moving to the location nominated. Robot is moving state forward or backward in this state to check the current position. 'goLeft' means the state that robot is running to the left. 'goRight' is the state that robot is turning to the right. 'Stopped' is the state that is generated when the user has suspended the robot. 'Stopped by Intrusion' is the state that is generated when the robot arrives at the destination. Robot is in the 'Idle' state when activated, and it is converted to the 'Idle' state or stops completely when the user commands the robot to stop.

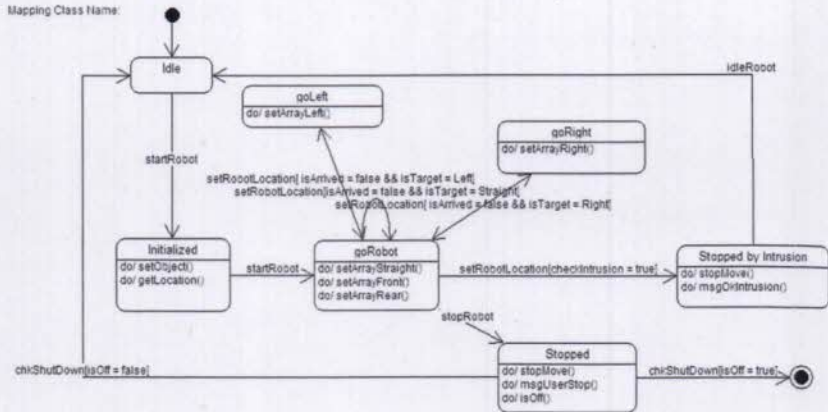


Fig. 4. State diagram of articulated robot

Generate the test case automatically for the execution of test. State diagram must be composed with the design tool as is shown in Figure 5, and create the state table by selecting the '1' button in the red block for the generation of test case.

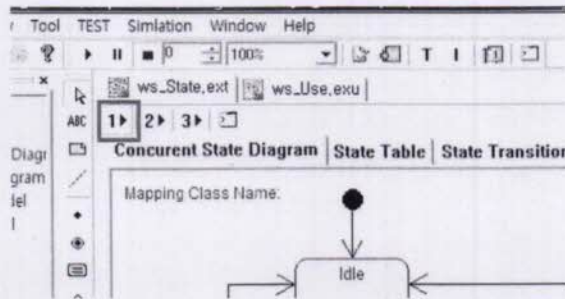


Fig. 5. Creation of state table

State table is the table that draws the called state when a specific event is generated in the corresponding state. With the state table, it is easy to check out which state has been called when a certain event is generated in the current state. Figure 6 is the state table created.

	Idle	Initialized	goRobot	Stopped	St
chkShutDown(isOff = true)	N/A	N/A	N/A	END	
startRobot	Initialized	goRobot	N/A	N/A	
setRobotLocation(isArrived = false && isTa...	N/A	N/A	goRight	N/A	
setRobotLocation(checkIntrusion = true)	N/A	N/A	Stopped by Intrusion	N/A	
stopRobot	N/A	N/A	Stopped	N/A	
idleRobot	N/A	N/A	N/A	N/A	
chkShutDown(isOff = false)	N/A	N/A	N/A	Idle	
setRobotLocation(isArrived = false && isTar...	N/A	N/A	goRobot	N/A	
setRobotLocation(isArrived = false && isTa...	N/A	N/A	goLeft	N/A	

Fig. 6. State table

State transition tree is generated based on the current state table when selecting '2▶' button in the screen of composed state table. State transition tree draws all the states in the form of tree that can be called in the current state for the visual checking. The tree will have more branches as the depth is deepened, and the depth can be assigned by selecting Switch. Figure 7 is showing the state transition tree created.

			Stopped by Intrusion	Idle	TC 1
				END	TC 2
			Stopped	Idle	TC 3
				goRight	TC
				Stopped by Intrusi	
				Stopped	TC
				goRobot	TC
Idle	Initialized	goRight	goLeft	goLeft	TC 8
			Stopped by Intrusion	END	Initialized 1
			Stopped	goRight	Initialized 1
			Stopped by Intrusion	Idle	TC 11
				END	TC 12
			Stopped	Idle	TC 13
				goRight	TC
				Stopped by Intrusi	

Fig. 7. State transition tree

Test case is generated finally when selecting '3▶' button in the state transition tree. Test case is generated as is shown in Figure 8, and it shows which event and which action can be occurred in the current state (Start state) through such test case, also it can be checked up in the form of table through this which state will be the next state.

TEST Simulation Window Help

ws_State.ext ws_Use.exu

1 2 3

Concurrent State Diagram State Table State Transition Tree Test case

No	Start State	Event	Action	Next State	Event	
V 1	goRobot	setRobotLocation[checkIntrusion = true]	do/ stopMove() do/ msgOkIntrusion()	Stopped by intrusion	IdleRobot	N/A
V 2	goRobot	stopRobot	do/ stopMove() do/ msgUserStop() do/ isOff()	Stopped	chkShutDown[isOff = false]	N/A
V 3	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ setArrayStraight() do/ setArrayFront() do/ setArrayRear() do/ runRobot()	goRobot	setRobotLocation[checkIntrusion = true]	do/ ; do/ ; do/ ;
V 4	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ setArrayStraight() do/ setArrayFront() do/ setArrayRear() do/ runRobot()	goRobot	stopRobot	do/ ; do/ ; do/ ;
V 5	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ setArrayStraight() do/ setArrayFront() do/ setArrayRear() do/ runRobot()	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ ; do/ ; do/ ;
V 6	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ setArrayStraight() do/ setArrayFront() do/ setArrayRear() do/ runRobot()	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ ; do/ ; do/ ;
V 7	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ setArrayStraight() do/ setArrayFront() do/ setArrayRear() do/ runRobot()	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ ; do/ ; do/ ;
V 8	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ setArrayStraight() do/ setArrayFront() do/ setArrayRear() do/ runRobot()	goRobot	setRobotLocation[isArrived = false && is Target = Straight]	do/ ; do/ ; do/ ;

Fig. 8. Generated test case

Test case is inserted in the event list as is shown in Figure 9 when double clicking the test case for the execution of test in the test case generated. When clicking the Run button, it is processing from No 1 to the end automatically. Time is the event generation frequency, and repeat is the number of repetitions.

Event List

NO	TC	Type	State/Event	Time(ms)	Repeat	P/F
1	V 1	s	Stopped	1000	1	
2	V 1	e	chkShutDown[isOff = false]	1000	1	
3	V 1	s	Idle	1000	1	
4	V 1	e	startRobot	1000	1	
5	V 1	s	Initialized	1000	1	
6	V 2	s	goRobot	1000	1	
7	V 2	e	setRobotLocation[checkIntrusion = true]	1000	1	
8	V 2	s	Stopped by intrusion	1000	1	
9	V 2	e	IdleRobot	1000	1	
10	V 2	s	Idle	1000	1	
11	V 3	s	goRobot	1000	1	
12	V 3	e	stopRobot	1000	1	
13	V 3	s	Stopped	1000	1	
14	V 3	e	chkShutDown[isOff = true]	1000	1	
15	V 3	s	END	1000	1	
16	V 4	s	goRobot	1000	1	
17	V 4	e	stopRobot	1000	1	
18	V 4	s	Stopped	1000	1	
19	V 4	e	chkShutDown[isOff = false]	1000	1	
20	V 4	s	Idle	1000	1	
21	V 5	s	goRobot	1000	1	
22	V 5	e	setRobotLocation[isArrived = false && isTarget = Str...	1000	1	
23	V 5	s	goRobot	1000	1	
24	V 5	e	setRobotLocation[isArrived = false && isTarget = Ri...	1000	1	
25	V 5	s	goRight	1000	1	

State Run Pause Stop All Delete

Fig. 9. Event list

5 Conclusion

It requires automatic generation of test case and execution method of test case for the execution of test in the virtual simulation environment. This paper has developed the tool supporting such procedure, applied to the multi-jointed robot, generated and executed the test case. As the result, the test case generated in the design stage can be executed directly in the virtual simulation environment. The method proposed overcomes the conventional problem in the design and execution of test, and finds out design errors by executing the test without waiting until its embodiment stage. It has shortened the time difference between the design and execution of test, and enabled prompt countermeasure against the problems generated in the design by finding easily out the errors of test case.

There is a problem in the method proposed that the tester must check and confirm the test case executed in the virtual simulation environment visually in person. In order to overcome such a weak point, automatic checkup method is under research for expected result of the test as a future study.

References

1. Boehm, B.W.: *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs (1981)
2. Son, H.S., Kim, W.Y., Kim, R.Y.C.: Implementation of Technique for Movement Control of Multi-Joint Robot. In: *The 30th KIPS Fall Conference 2008*, November 14, vol. 15(2), pp. 593–596 (2008)
3. Kim, W.Y., Son, H.S., Kim, R.Y.C., Carlson, C.R.: MDD based CASE Tool for Modeling Heterogeneous Multi-Jointed Robots. In: *CSIE 2009*, vol. 7, pp. 775–779. IEEE Computer Society, Los Angeles/Anaheim (2009)
4. Kim, J.S., Son, H.S., Kim, W.-Y., Kim, R.Y.C.: A Study on Education Software for Controlling of Multi-Joint Robot. *Journal of The Korean Association of Information Education* 12(4), 469–476 (2008)
5. Kim, J.S., Son, H.S., Kim, W.-Y., Kim, R.Y.C.: A Study on M&S Environment for Designing The Autonomous Reconnaissance Ground Robot. *Journal of the Korea Institute of Military Science and Technology* 11(6), 127–134 (2008)
6. Burnstein, I.: *Practical Software Testing*. Springer, Heidelberg (2003)
7. Toth, A., Varro, D., Pataricca, A.: Model Level Automatic Test Generation for UML State-Charts. In: *Sixth IEEE Workshop on Design and Diagnostics of Electronic Circuits and System, DDECS 2003* (2003)
8. Gresi, S., Latella, D., Massink, M.: Formal Test Case Generation for UML State-Charts. In: *Ninth IEEE International Conference on Engineering Complex Computer System Navigating Complexity in e-Engineering Age* (2004)
9. Bertolino, A., Marchetti, E.: Introducing a reasonably complete and coherent approach for model based testing. In: Jensen, K., Podelski, A. (eds.) *TACAS 2004*. LNCS, vol. 2988, Springer, Heidelberg (2004)
10. McGhee, R.B., Frank, A.A.: On the Stability Properties of Quadruped Creeping Gaits. *Mathematical Biosciences* 2(1/2) (1968)
11. Raibert, M.H.: *Legged Robots*. ACM 29(6), 499–514 (1986)

Communications in Computer and Information Science

The CCIS series is devoted to the publication of proceedings of computer science conferences. Its aim is to efficiently disseminate original research results in informatics in printed and electronic form. While the focus is on publication of peer-reviewed full papers presenting mature work, inclusion of reviewed short papers and abstracts reporting on work in progress is welcome, too. Besides globally relevant meetings with internationally representative program committees guaranteeing a strict peer-reviewing and paper selection process, conferences run by societies or of high regional or national relevance are considered for publication as well.

The topical scope of CCIS spans the entire spectrum of informatics ranging from foundational topics in the theory of computing to information and communications science and technology and a broad variety of interdisciplinary application fields.

Publication in CCIS is free of charge. No royalties are paid, however, CCIS volume editors receive 25 complimentary copies of the proceedings. CCIS proceedings can be published in time for distribution at conferences or as post-proceedings, as printed books and/or electronically as CDs; furthermore CCIS proceedings are included in the CCIS electronic book series hosted in the SpringerLink digital library.

The language of publication is exclusively English. Authors publishing in CCIS have to sign the Springer CCIS copyright transfer form, however, they are free to use their material published in CCIS for substantially changed, more elaborate subsequent publications elsewhere. For the preparation of the camera-ready papers/files, authors have to strictly adhere to the Springer CCIS Authors' Instructions and are strongly encouraged to use the CCIS LaTeX style files or templates.

Detailed information on CCIS can be found at www.springer.com

ISSN 1865-0929

ISBN 978-3-642-23311-1



› springer.com