Journal of Engineering Technology
Volume 6, Issue 1, Jan. 2018, PP. 378-393

RESEARCH ARTICLE -ENGINEERNG TECHNOLOGY

An automatic test case generation from state diagram using cause-effect graph based on model transformation

Hyun Seung Son, Keunsang Yi, Byungkook Jeon* , and R. Young Chul Kim

SE Lab., Dept. of CIC(Computer and Information Communication), Hongik University, Sejong Campus, 30016, Korea

*Dept. of Software, Gangneung-Wonju National University, Wonju City, Gangwon Prov., 26403, Korea

Abstract: For high quality production, it is absolutely necessary to require quality assurance and control (QA/QC) to guarantee software productivity. Most software companies fail to carry out sufficient testing due to the issues of development schedules, costs, and time-to-markets. To solve this problem, most researchers consider a test automation, even without test expert, to make it possible to design and perform testing faster than manual approaches. Automatic testing has merged as a significant part in high-quality software tests. However, the existing methods for automatic test case generation had been implemented with the algorithm approach, which causes a problem that when the input models change, test experts have to modify the whole program. Our approach of automatic testing solutions, suggests to adapt the model transformation for the test case generation to automatically generate the test case in a model driven environment, which consists of 1) transforming the decision table from a cause-effect graph and 2) generating test cases from the decision table. This approach must use the metamodel and transformation rule without an algorism mechanism. As a result, with our method, he/she can automatically generate test cases by designing the state diagram in UML approach.

Keywords. Model Transformation, Cause-Effect Graph (CFG), Test case Generation, Model based Testing, Metamodel

1. Introduction

Developing high-quality software calls for precautions such as identifying software-specific risks as well as detecting potential defects in order to prevent failure. Therefore, quality assurance and testing should be performed throughout the software development life cycle. Particularly, software testing is a vital part in this life cycle. However, most software companies lack in sufficient testing because development schedules and cost issues are taken in account [1-2]. As most software products are released in haste, they spend more effort and money on maintenance.

Automatic testing is essential for designing and executing tests quicker than manual testing. Test automation includes an automatic test case generation. This can simplify the test during the repetitive development of the complex model. Even with the absence of a test expert, it can reuse and execute test case. Therefore, the test automation with the automatic test case generation has a characteristic that saves the developer's time and cost [3]. Because of these features, automatic approaches have been carried out in many studies [4-8]. To automatically generate the test cases, the existing approaches have been developed a test tool with algorithm methods. However, the tools are difficult to change once the input model is determined. When

it is necessary to change the model, developers have to rewrite the program code for modification and recompilation.

We adopt the model transformation mechanism to a test case generation approach for a loose relationship between the input models with Model Driven Architecture (MDA) [9-10]. For this model transformation, we must require the elements of metamodel, engine, and rule language [11-14]. No one has mentioned Model Based Testing (MBT) [15] with an applied Model Driven Architecture (MDA). We first try to use MDA approach for testing different software or systems based on models. A MBT is a black-box testing technique which uses models to generate test cases. Code-based testing deals with source code components. The model-based testing makes use of abstract models of systems for testing regardless of the scale or complexity of software. Since the MBT uses models, it is relatively easy to apply to the Model Driven Engineering.

Our proposed approach automatically generates test case via decision table from a cause-effect graph [16]. This method also requires the design of the metamodel and the writing rule for model transformation. Our previous research applied the model transformation techniques to a cause-effect graph for automatic test case generation. In this paper, we focused on generating test cases to cause-effect graph from UML state diagram. The proposed test case generation is based on model transformation. To do this, we develop the metamodel of the cause-effect graph.

Therefore we should create the whole structure with each name of elements of the cause-effect graph. Our developed metamodels are based on combining each UML state diagram with the cause-effect graph, both with model transformation. Model transformation is a metamodel technique to automatically generate test cases from models. To use this technique, we should define metamodels in each phase and write each model-specific transformation rule in a model transformation language. It is possible to automate the entire process by metamodeling and model transformation, which generates test cases with the inputs from state diagrams. The paper is organized as follows. Chapter 2 describes the model transformation and the metamodel of cause-effect, decision table, and test case. Chapter 3 describes the method for generating test case from UML state diagram. Chapter 4 describes a case study to show test case generation process. The last chapter mentions the conclusion and future work.

2. Related work

2.1 Model transformation

The basic model transformation is a simple transformation of input model to output model [11]. Both input and output model conform to each metamodel which generally defines abstract syntax modeling notation. The transformation performs the written language with reference to metamodel. While the transformation language is written through some rules, the transformation rules are written from mapping with similarities and differences between two models. The transformation language must consists of particular commands that are performed in transformation engine. Therefore, for this model transformation mechanism we should define metamodel, and develop transformation program with transformation language, which executes on a transformation engine.

2.2 Cause-effect graph

The cause-effect graph is a method to express the associative relationship between causes and effects about customer's requirements [17]. It is possible to represent logical relationship like 'and', 'or', 'not' notations. Figure 1 shows an example of cause-effect graph. When both cause 1 and cause 2 are true values, then the effect is also a true value on the logical expression of 'and' notation.

Journal of Engineering Technology
Volume 6, Issue 1, Jan. 2018, PP. 378-393
RESEARCH ARTICLE -ENGINEERNG TECHNOLOGY

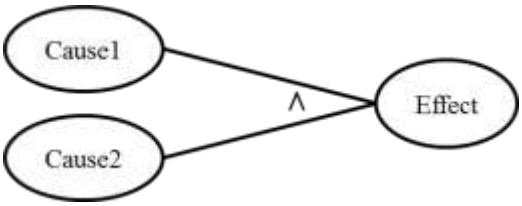


Fig. 1. An example of Cause-effect graph

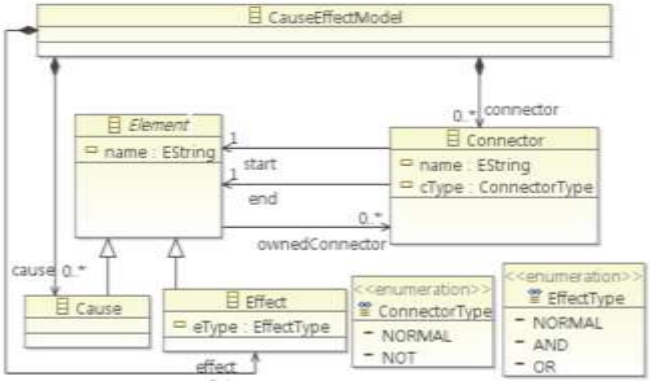
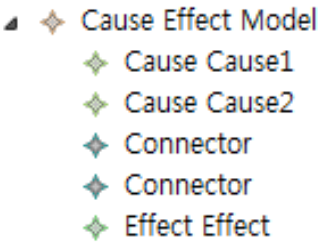


Fig. 2. The metamodel of Cause-effect graph

Because the existing cause-effect graph does not use model transformation, the metamodel of cause-effect graph is not defined. Therefore, we need to design a metamodel with the basic name of cause-effect graph in figure 2. The *CauseEffectModel* is a root node of the cause-effect graph. The internal root node is able to include *Cause*, *Effect*, and *Connector*. The *Cause* is a set of causes, the *Effect* is an effect, and the *Connector* is the connection of causes and effect which is used to express the conditions.




```
<?xml version="1.0" encoding="UTF-8"?>
<cedm:CauseEffectModel xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:cedm="http://cedm/1.0">
  <cause name="Cause1"
ownedConnector="//@connector.0"/>
  <cause name="Cause2"
ownedConnector="//@connector.1"/>
  <connector start="//@cause.0" end="//@effect.0"/>
  <connector start="//@cause.1" end="//@effect.0"/>
  <effect name="Effect" ownedConnector="//@connector.0
//@connector.1" eType="AND"/>
</cedm:CauseEffectModel>
```

(a) Editor (b) XMI file

Fig. 3. The input result of Cause-effect graph

In order to validate the designed metamodel, we draw the proposed metamodel with Ecore Tools [18], and generate the editor of cause-effect graph with Eclipse Modeling Framework (EMF) [19] in Eclipse. Figure 3 shows the input result of an example of cause-effect graph in figure 1. Figure 3(a) is an input result with editor. Figure 3(b) is the context of XML Metadata Interchange (XMI) file [20]. Through showing this result, we can validate the proposed metamodel.

2.3 Decision table

The decision table is a method of tabulating the content of a cause-effect graph to generate test cases. Figure 4 exemplifies the decision table. In transforming the cause-effect graph into the decision table, the *Cause* element in the cause-effect graph goes to the vertical axis of the decision table as the *Cause*, while the *Effect* is placed as the *Effect* on the horizontal axis of the decision table. Then, using the logical expression of the condition values in the cause-effect graph, the values are entered in the decision table.

Journal of Engineering Technology
Volume 6, Issue 1, Jan. 2018, PP. 378-393
RESEARCH ARTICLE -ENGINEERNG TECHNOLOGY

	Condition		
Cause	1	2	3
input 1	T	T	T
input 2	T	T	F
input 3	T	F	F
Effect	1	2	3
output 1	X	F	F
output 2	T	X	F

Fig. 4. An example of Decision table

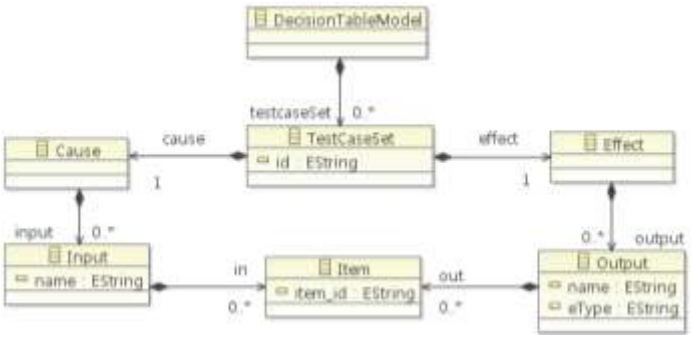


Fig. 5. A metamodel of Decision table

No standardized metamodel exists for decision tables, which does not warrant a metamodel designed for decision tables. Hence, we design a metamodel with the basic names of tables found in previous studies in figure 5. The designed metamodel, *DecisionTableModel,* is the root node of the decision table encompassing all elements. The *DecisionTableModel* has multiple *TestCaseSets*, which collects only relevant causes and effects as data from the cause-effect diagram. *Causes* and *effects* are represented in the *TestCaseSets* with each value entered in the *Item*.

2.4 Test case

The IEEE standard 829 defines a test case as "A document specifying inputs, predicated results, and a set of execution conditions for a test item" [21]. Thus, as in figure 6, the test case consists of *Pre-Conditions*, *Test Conditions* and *Expectation Results*. *Pre-Conditions* are input conditions, while *Test Conditions* are logical expressions for the combination of input conditions. *Expectation Results* are expected values in line with the *Pre-Conditions* and *Test Conditions*.

Like decision tables, no standardized metamodel exists for test cases, which does not warrant a metamodel defined for test cases. Thus, we design the metamodel of the basic names of tables in figure 7. To be specific, the designed metamodel, the *TestCaseModel,* is the root node of test cases. The *TestCaseModel* has multiple *TestCases*. *TestCases* consists of *PreConditions*, *TestConditions* and *ExpectationResults*. The *PreConditions*, *TestConditions*, and *ExpectationResults* correspond to *Pre-Conditions*, *Test Conditions*, and *Expectation Results* in the *TestCase* respectively. The data for *PreConditions*, *TestConditions* and *ExpectationResults* is saved as strings in *testpre*, *testcon*, and *testresult* each.

No	Pre-Condition	Test Condition	Expectation Result
TC1	Input 1	AND	Result 1
TC2	Input 2	OR	Result 2
TC3	Input 3	NOT	Result 3

Fig. 6. An example of test case

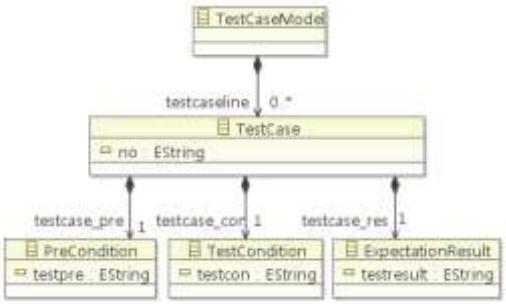


Fig. 7. A metamodel of test case

2.5 Generating Test case based on Cause-effect graph

In software testing, Gary mentions that a cause-effect graph assures coverage criteria of 100% functional requirements with minimum test case [22]. We proposed a method with model transformation mechanism for

test case generation of cause-effect graph. To implement the proposed method, we make rules of model transformation using ATLAS Transformation Language (ATL) [23], and execute the rules in Eclipse development environment.

The proposed method consists of three parts of cause-effect graph, decision table, and test case. Also, the method uses two model transformation rules as follows: 1) automatically translating the cause-effect graph into the decision table, and 2) translating the decision table into the test case in this order on model transformation engine

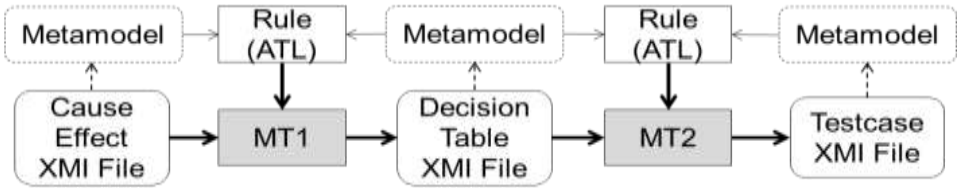


Fig. 8. The MT1 and MT2 steps of Model Transformation

The proposed method consists of two model transformations MT1 and MT2 shown in figure 8. The MT1 converts to decision table from cause-effect graph, and the MT2 also converts to test case from decision table. In this process of the proposed method, the data of input/output models (such as Cause Effect Model/Decision Table Model) represents XML Metadata Interchange (XMI) [20]. That is, our approach consists of transforming 1) from cause-effect graph to decision table, and 2) also from decision table to test case. Input and output data of model transformation are executed with all XML metadata Interchange. At rule definition

No	Pre-Condition	Test Condition	Expectation Result
TC1	Cause1=F, Casue2=F	AND	Effect=F
TC2	Cause1=F, Casue2=T	AND	Effect=F
TC3	Cause1=T, Casue2=F	AND	Effect=F
TC4	Cause1=T, Casue2=T	AND	Effect=T

Fig. 9. The execution result of model transformation from CEG to decision table

of model transformation, it defines all transformable elements of each model and relationship between them, and represents transformation rules with ATRLAS Transformation Language (ATL). ATL provides engine, language, and test editor for executing model transformation.

To execute model transformation, we represent XMI from modeling cause-effect graph based on requirement specifications. We enter XML file of the modeled cause-effect (shown in figure 4) into model transformation tool which then automatically generates decision table. After executing Model transformation, it produces decision table to make it easily understandable like figure 9. In the model transformation of the XMI file of the decision table generated earlier, the test case in figure 10 is a tabulated XMI file to help detailed understanding.

	Condition			
Cause	1	2	3	4
Cause1	F	T	F	T
Cause2	F	F	T	T
Effect	1	2	3	4
Effect	F	F	F	T

Fig. 10. The execution result of Model transformation from Decision table to Test case

3. Metamodeling of a State Diagram

To automatically generate a test case from a state diagram, we propose 1) mapping the state diagram onto the cause-effect graph and 2) applying the model transformation technique. Like Figure 11, our proposed method consists of three steps of model transformation for test case generation as follows: 1) transforming the state diagram into the cause-effect diagram, 2) transforming the cause-effect diagram into the decision table, and 3) transforming the decision table into the test case in this order. The model transformation is required to design metamodels of target models, and define rules for model transformation. We should describe the metamodel of a state diagram which is not defined in the previous research, and identify the rule for transforming the state diagram into the cause-effect graph.

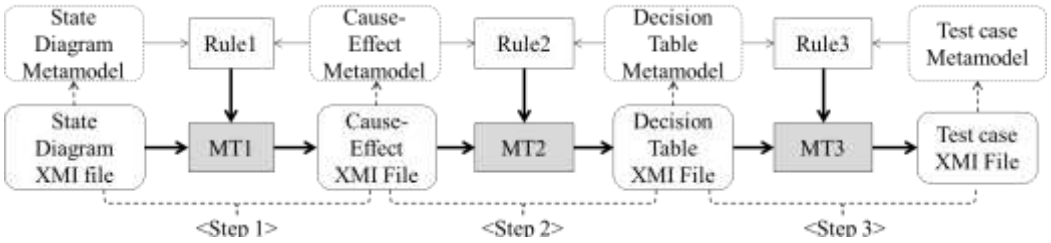


Fig. 11. The process of Model transformation for generating Test case

3.1 Designing the Metamodel of a State diagram

The metamodel for UML state diagrams proposed by OMG is too complex to design and use. For a more efficient transformation, we need to simplify the metamodel of this state diagram. Therefore we have developed '*Hongik MDA based Embedded S/W Component Development Methodology'* (HiMEM) [24-25] as a modelling tool, which does not follow OMG's UML state diagram [26]. Therefore, as in figure 12, we design a metamodel for transforming a state diagram. Most metamodel constructs are complied with the notation of the UML state diagram without redundant elements in the design process.

Journal of Engineering Technology
Volume 6, Issue 1, Jan. 2018, PP. 378-393
RESEARCH ARTICLE -ENGINEERNG TECHNOLOGY

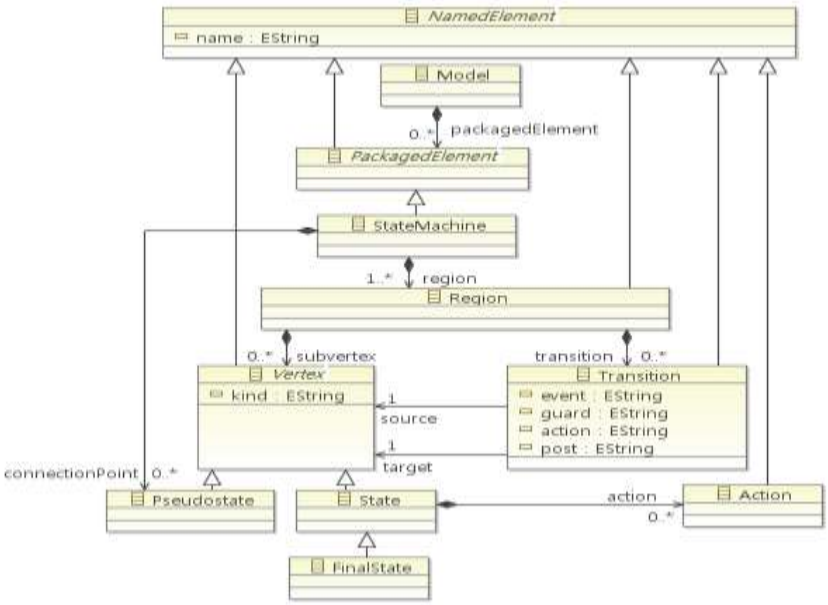


Fig. 12. A metamodel of State diagram

In the defined metamodel for the state diagram, the *Model* is defined as the root node with a name for *packagedElement* being generated whenever a *StateMachine* is produced under the *Model*. The *StateMachine* is comprised of multiple *Regions* in which the data goes for the notations of the state diagram. Also, there are two constructs, i.e. *Vertex* and *Transition*. The *Vertex* refers to state nodes such as initial and final states, while the *Transition* refers to lines connecting the nodes. The *State* is able to constitute multiple *Actions*.

The defined metamodel represents the names and attributes in the state diagram. Figure 13 shows how the metamodel and the state diagram are related to each other. The notation of each diagram is set by its name and attribute. For example, in order to name the notation of the model, *State 1* and *State 2* are defined as the *State*, and *action 1* and *action 2* are defined as the *Action*. These names are also used for saving the actual data in XMI. For example, the metamodel saves the names of data for the model, which plays an important role for selecting data in model transformation. Therefore, data alteration/modification/deletion cannot be performed in model transformation without the names of objects.

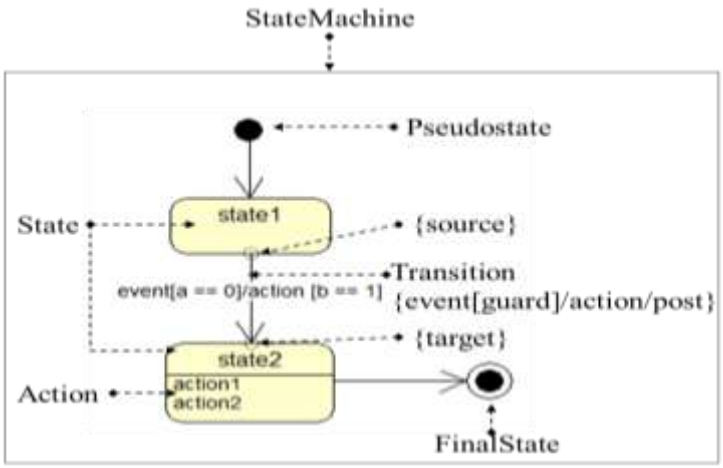


Fig. 13. The relationship between Metamodel and State diagram

Journal of Engineering Technology
Volume 6, Issue 1, Jan. 2018, PP. 378-393
RESEARCH ARTICLE -ENGINEERNG TECHNOLOGY

3.2 Defining rules for Model transformation from State diagram to Cause effect graph

To transform the state diagram into the cause-effect graph, it needs to consider the roles of all notations in the state diagram and the cause-effect graph. Model transformation refers to migrating data from an input model to an output model. To do this, a relationship between models has to be conducive to write the transformation rules. First, one state in the state diagram gets transited to the next state when encountering an event.

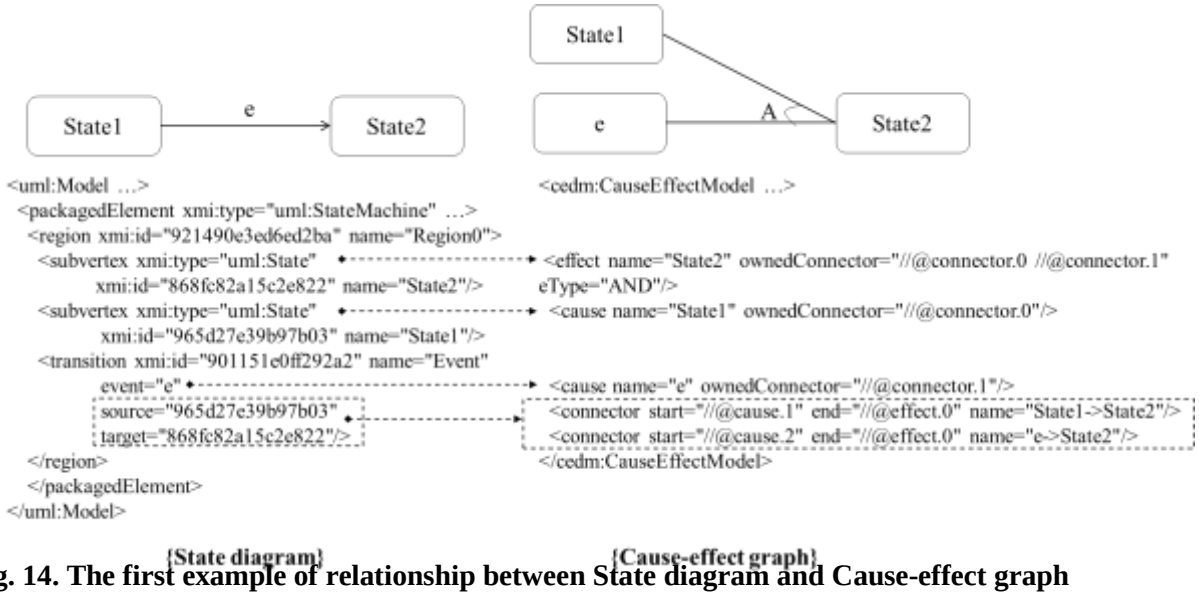


Fig. 14. The first example of relationship between State diagram and Cause-effect graph

In order for *State1* to be transited to *State 2*, *State 1* should wait until the *event e* occurs based on the event / condition / action (ECA) rule in figure 14. However, if there is a false value in any condition, it does not allow *State 1* to move *State 2*. That is, in the premise that *State 1* and the *event e* all have true values, the *event e* causes *State 1* to move to *State 2* In the cause-effect graph, instead of the state mechanism, *State 1* and the *event e* take the role of the causes. They are then simultaneously connected to the *State 2* which takes the role of the effect. Therefore, to bring the two causes into an effect, it needs to have the *AND* condition for transforming the state diagram into the cause-effect graph.

Journal of Engineering Technology
Volume 6, Issue 1, Jan. 2018, PP. 378-393
RESEARCH ARTICLE -ENGINEERNG TECHNOLOGY

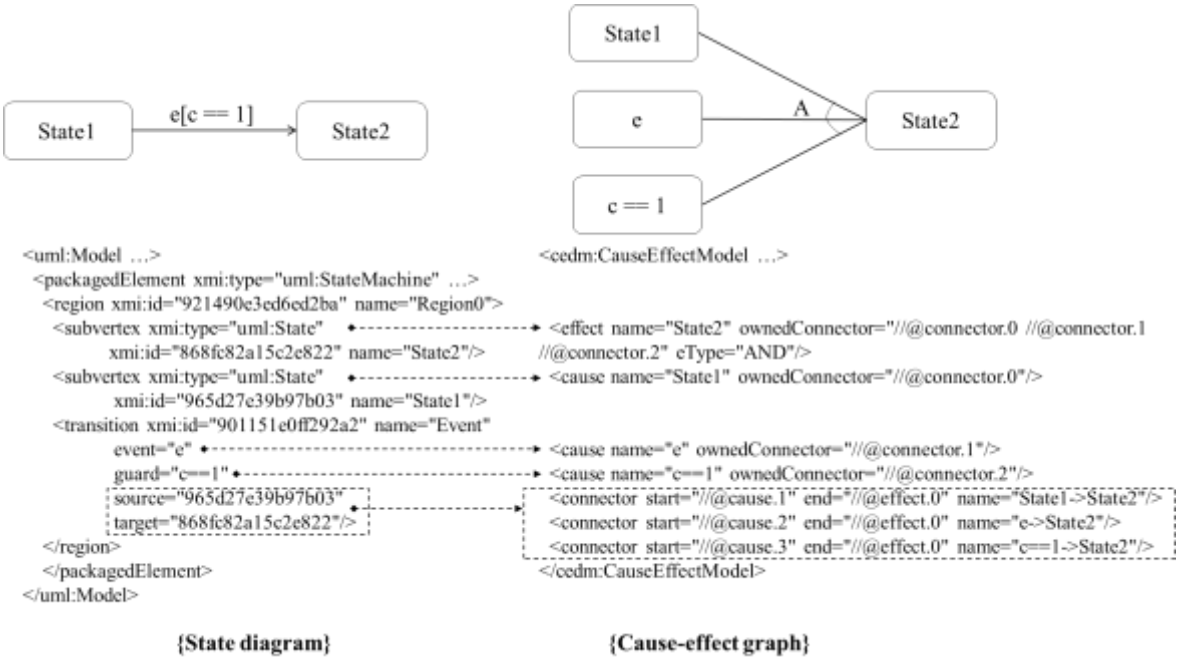


Fig. 15. The second example of relationship between State diagram and Cause-effect graph

Even though the transition of a state diagram is started from a single event, a condition [c ==1] in the *event e* exists when the transformation from the state diagram to cause effect graph occurs in figure 15. The condition included with the state and transition in figure 14 becomes the cause in the cause-effect graph. In addition, another condition *"c == 1"* should be a true value so that the transition is connected to *State 2*. The condition *"c == 1"* can serve as the cause. A state diagram may provide an action for the transition part as an output value. In this case, the transformation occurs as in figure 16. When an event is invoked in the state diagram, the output value of *the action* has triggered to *the effect* in the cause-effect graph. That is, the *action* in the state diagram is transformed into the effect in the cause-effect graph. Furthermore, in order for the *action* to be carried out, it should satisfy 1) the earlier condition where *State 1* should be occurred, 2) the condition where the event *e* should be invoked, and 3) the condition where *"c == 1"* should be true. Thus, the causes such as *State1, the event e, and the condition "c==1"* should be satisfied on the *AND* condition. Hence, the state diagram is transformed into a cause effect graph, which generates three *causes* and two *effects.* Then these two effects share three causes.

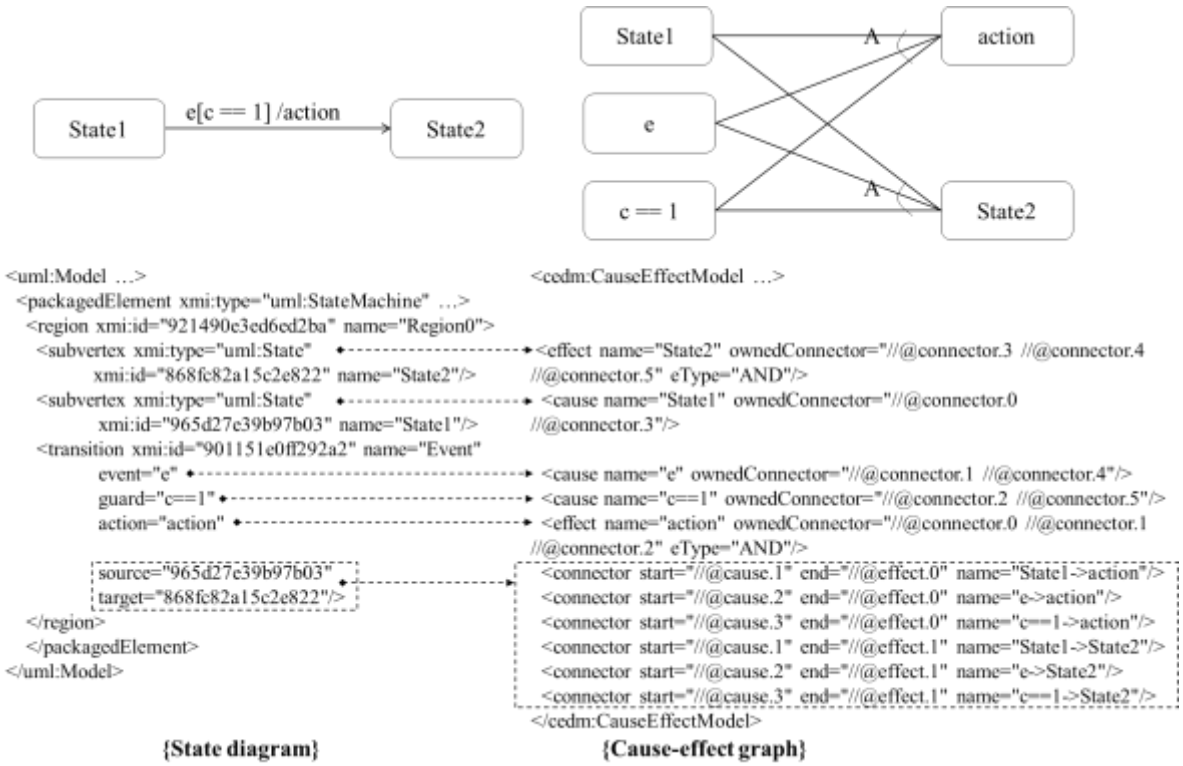


Fig. 16. The third example of relationship between state diagram and cause-effect graph

Finally, it needs to consider a post-condition [a=0] in the state diagram. The post-condition is characterized by a transition to the next state after an action is performed as well as the value of the condition is true. Therefore, as in figure 17, the post-condition [a==0] in the state diagram is transformed into the cause in the cause-effect graph without connecting with *the action*, but just affecting *State 2*. That is, the post condition is linked to *State 2*.

From the four examples in figures 14-17, the transformation rules for transforming the state diagram into the cause-effect graph can be written in algorithm of 'StateToCE' rule (in our website [1]). On the transformation process in rules, all transitions in the state diagram are added to the sources and targets of the transitions as the causes and the effects in the case-effect graph respectively. Then, connectors are made for the two relationships. Here, if the state diagram has an event of the transition, the event becomes the cause in the cause-effect graph, while the target of the transition is added as the effect in the cause-effect graph. Also, the target (particularly the guard and post) of the transition in the state diagram is treated in the same way as an event. The *action* of the transition in the state diagram generates the *effect* in the new cause-effect graph. When the guard and post are existed, the connectors are added to link the newly generated effects with the guard and post respectively.

[1] http://selab.hongik.ac.kr/~son/state/StateToCE.atl

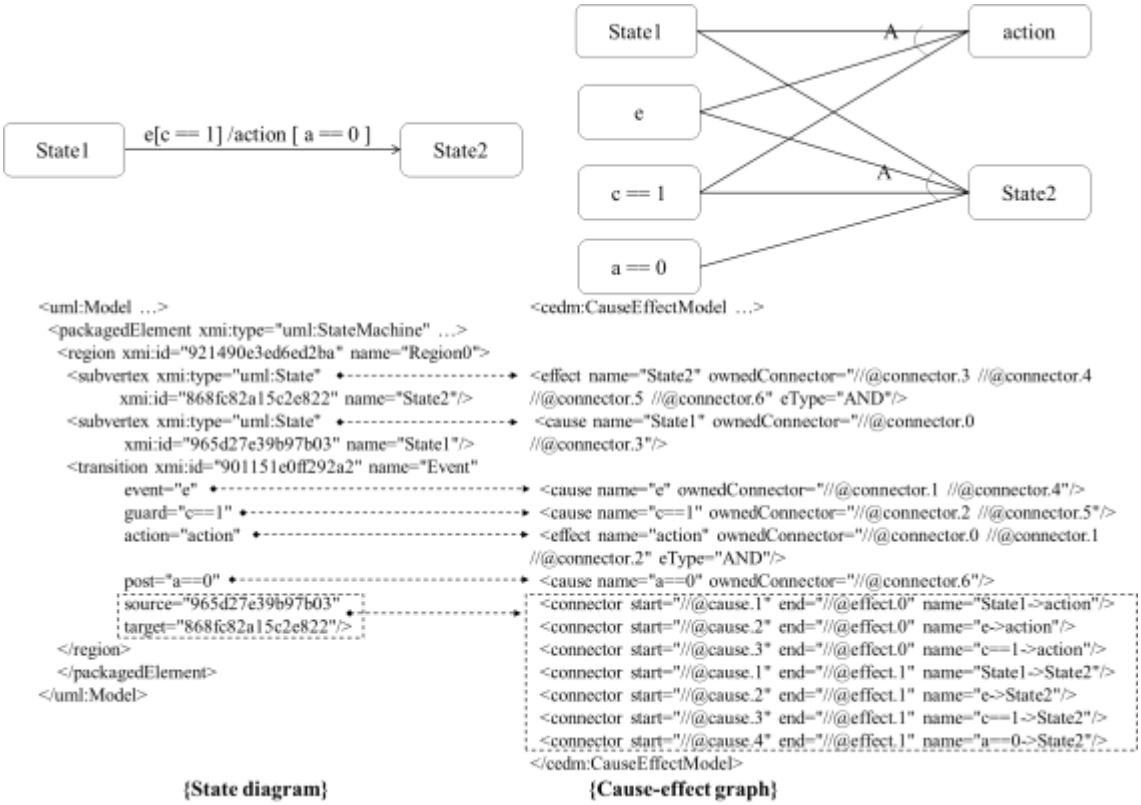


Fig. 17. The fourth example of relationship between State diagram and Cause-effect graph

4. Case Study

For HanBack Electronic's RoboCAR, we develop modeling & simulation tools to control four motors and four wheels. Figure 18 shows a basic form of RoboCAR which is designed to use the 8-Bit microprocessor embedded in the body (ATmega128L) for controlling sensor values and the DC motor. The RoboCAR can operate on its own with supporting the robot control sensors, e.g. ultrasonic, infrared and acceleration sensors. The role of ultrasonic sensor can detect any front and rear obstacles.

Fig 18. The Basic form of RoboCAR

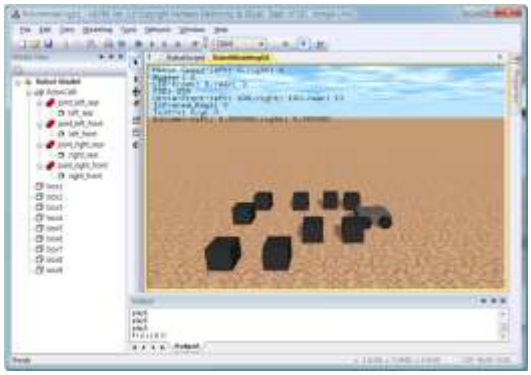


Fig. 19. RoboCAR's modelling & simulation tools

Journal of Engineering Technology
Volume 6, Issue 1, Jan. 2018, PP. 378-393
RESEARCH ARTICLE -ENGINEERNG TECHNOLOGY

The RoboCAR is set with diverse sensors to find out the operating state of the robot. First, if any obstacles are obstruct traffic on moving forward or backward, it has worked ultrasonic sensors mounted on the front and rear side of the RoboCAR. Therefore, RoboCAR can detect obstacles from a distance on moving. The CAR has a PSD distance sensor on the front, which enables a precise infrared measurement of the front distance when the robot moves forward. Moreover, RoboCAR has an infrared sensor on the bottom, detecting the lines on the floor. Finally, RoboCAR carries an acceleration sensor inside, sequentially detecting the dynamic forces of the device such as acceleration, vibration and impact. These sensors help to easily identify the operating states of RoboCAR.

To test the robot, a modeling & simulation tool as in figure 19 is developed for a virtual environment. The developed tool can assemble the parts to generate a robot, and then develop various forms of robots in combination with parts. Also, the developed tool is interoperable with the modeling tool, HiMEM, which works to execute the state diagram with running the robot in virtual environment to work with the real one. Thus, it is possible to visually inspect whether the robot moves naturally. To test the robot, as in figure 20, a state diagram is modeled. This state diagram is designed for the robot to avoid obstacles until it reaches the destination.

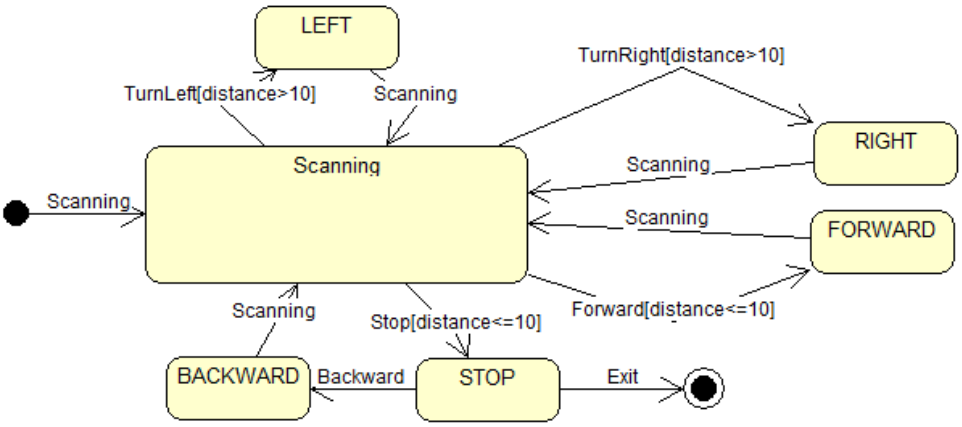
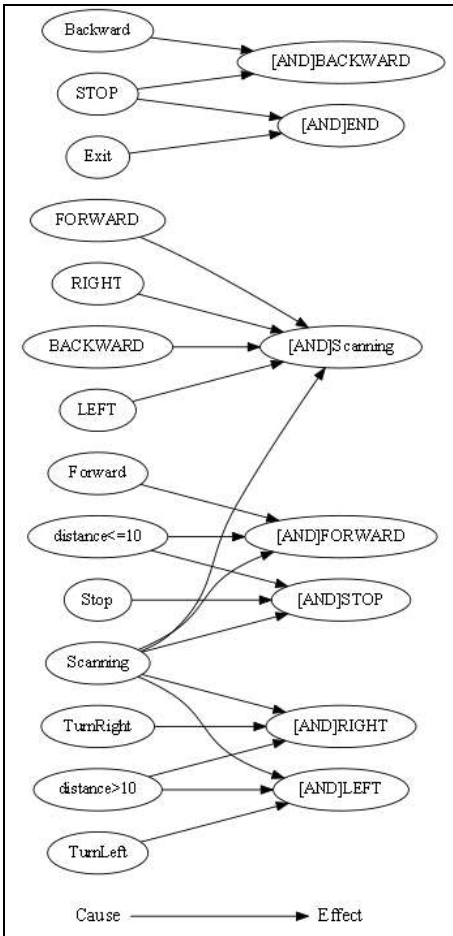


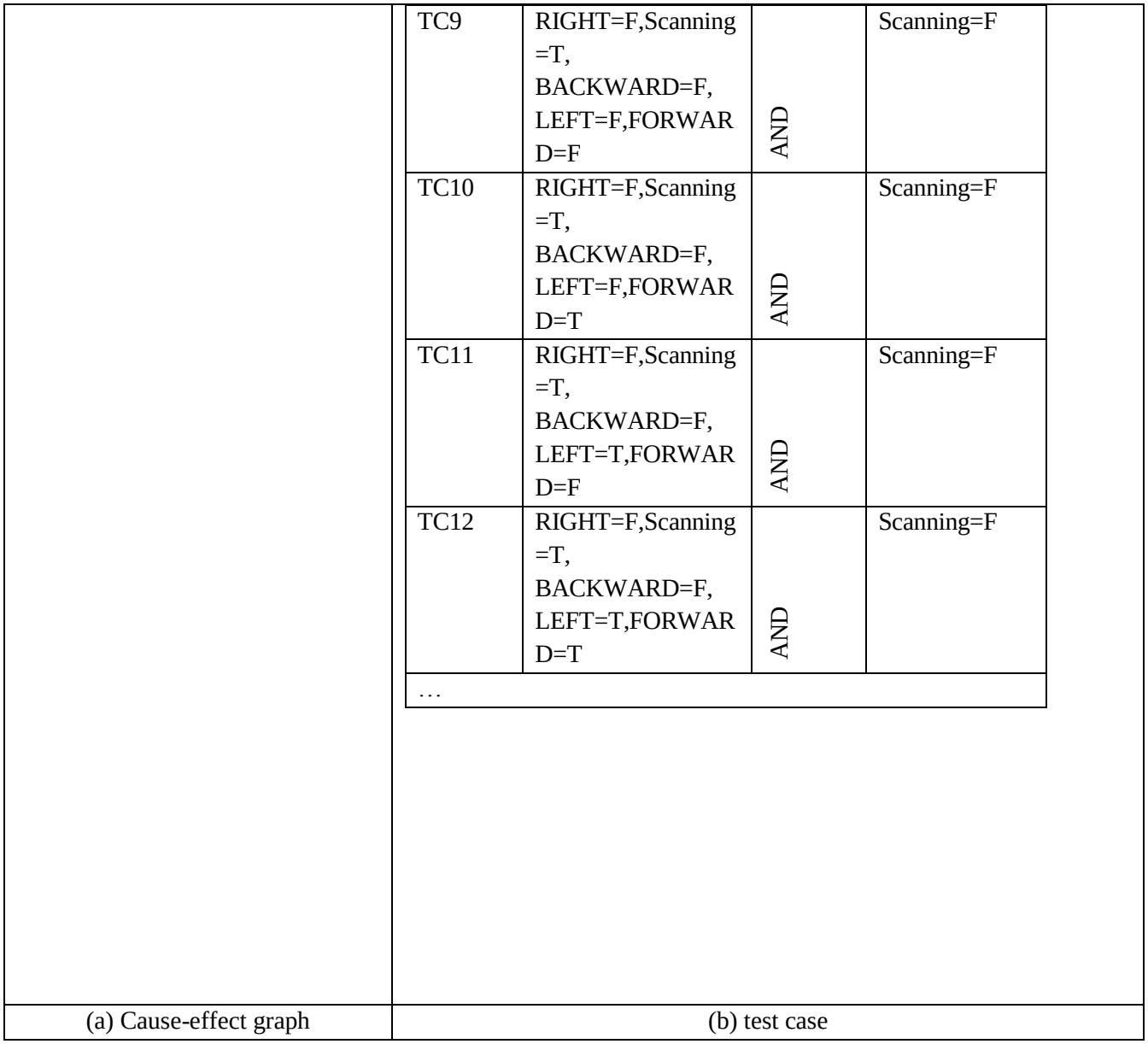
Fig. 20. Modeling State diagram

For a model transformation of the state diagram, as in figure 21, the automatic generator works to transform the state diagram into a decision table via a cause-effect graph, and then creates test cases. A total of 72 test cases are generated in the exemplified robot.

Journal of Engineering Technology
Volume 6, Issue 1, Jan. 2018, PP. 378-393



TC ID	Input	Condition	Expected results
TC1	RIGHT=F,Scanning=F, BACKWARD=F, LEFT=F,FORWARD=F	AND	Scanning=F
TC2	RIGHT=F,Scanning=F, BACKWARD=F, LEFT=F,FORWARD=T	AND	Scanning=F
TC3	RIGHT=F,Scanning=F, BACKWARD=F, LEFT=T,FORWARD=F	AND	Scanning=F
TC4	RIGHT=F,Scanning=F, BACKWARD=F, LEFT=T,FORWARD=T	AND	Scanning=F
TC5	RIGHT=F,Scanning=F, BACKWARD=T, LEFT=F,FORWARD=F	AND	Scanning=F
TC6	RIGHT=F,Scanning=F, BACKWARD=T, LEFT=F,FORWARD=T	AND	Scanning=F
TC7	RIGHT=F,Scanning=F, BACKWARD=T, LEFT=T,FORWARD=F	AND	Scanning=F
TC8	RIGHT=F,Scanning=F, BACKWARD=T, LEFT=T,FORWARD=T	AND	Scanning=F



TC9	RIGHT=F,Scanning=T, BACKWARD=F, LEFT=F,FORWARD=F	AND	Scanning=F
TC10	RIGHT=F,Scanning=T, BACKWARD=F, LEFT=F,FORWARD=T	AND	Scanning=F
TC11	RIGHT=F,Scanning=T, BACKWARD=F, LEFT=T,FORWARD=F	AND	Scanning=F
TC12	RIGHT=F,Scanning=T, BACKWARD=F, LEFT=T,FORWARD=T	AND	Scanning=F
…			

(a) Cause-effect graph

(b) test case

Fig. 21. Transformation result

5. Conclusions

Most automatic test case generations have focused on algorithmic approach, which depends on input data. In this paper, we suggest to automatically generate test cases only with designing some models for a system. To solve this, we adapt the model transformation to automatically generate the test case in a model driven environment. Based on the model transformation approach, we can apply our approach with a cause-effect graph for an automatic test case generation. Gary [22] mentions to assure test coverage of 100% functional requirements with minimum test cases based on the cause-effect graph. Our method for test case generation consists of 1) transforming the decision table from the cause-effect graph and 2) transforming test case from the decision table. This approach would require the use of the metamodel and the transformation rule. With our method, he/she can automatically generate test cases even without test expert. This approach also automatically executes test cases with virtual environment.

In conclusion, the proposed method facilitates automatic test cases generation from a state diagram, and thus enables non-experts to perform software testing. As a result, the proposed method will reduce software testing time and cost. Future studies will build on reducing redundant test cases for optimization

Acknowledgments

This work was supported by the Human Resource Training Program for Regional Innovation and Creativity through the Ministry of Education and National Research Foundation of Korea (NRF-2015H1C1A1035548)**,** and also by a grant (17CTAP-C133299-01) from Technology Advancement Research Program funded by Ministry of Land, Infrastructure and Transport of Korean government.

References

[1] I. Burnstein. 2003. *Practical Software Testing*. Springer-Verlag.
[2] W.I. Kown, E.Y. Park, and H.G. Cho. 2006. *Practical Software Testing Foundation*. STA Consulting.
[3] D.M. Cohen, S.R. Dalal, J. Parelius, and G.C. Patton. 1996. "The combinatorial design approach to automatic test generation." *IEEE Software* 13(5): 83-88.
[4] Woo Yeol Kim, Hyun Seung Son, Jae Seung Kim, and R. Young Chul Kim. 2010. "A Study on Improving Test Process for Heterogeneous Embedded Software." In *Proceedings of Korea Conference on Software Engineering 2010*, Pyeongchang, Korea, February 8-10, 12(1): 513-515.
[5] Hyun Seung Son, Woo Yeol Kim, Jae Seung Kim, and R. Young Chul Kim. 2011. "A Study on Test Process of Multi-jointed Robot in Virtual Environment." In *Proceedings of Korea Conference on Software Engineering 2011*, Pyeongchang, Korea, February 9-11, 13(1): 532-535.
[6] Soo Jung Woo, Hyun Seung Son, Woo Yeol Kim, Jae Seung Kim, and R. Young Chul Kim. 2012. "A Study on Extracting Test Case based on M&S for Pre-Testing." In *Proceedings of Korea Conference on Software Engineering 2012*, Pyeongchang, Korea, February 8-10, 14(1): 181-183.
[7] Hyun Seung Son, Woo Yeol Kim, Jae Seung Kim, and R. Young Chul Kim. 2012. "Automatic Test Case Generation using Multiple Condition Control Flow Graph." In *Proceddings of Information Technology and Computer Science 2012*, Porto, Portugal, July 29-31, 13: 105-109.
[8] Dong Ho Kim, and R. Young Chul Kim. 2013. "A Study on Automatic Test Case Extraction Mechanism from UML State Diagrams Based on M2M Transformation." *The Journal of IIBC (The Journal of The Institute of Internet, Broadcasting and Communication)*. 13(1): 129-134.
[9] Jon M. Siegel. 2014. "MDA Guide revision 2.0." Accessed March 1, 2017. http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.
[10] D. S. Frankel. 2003. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley.
[11] K. Czarnecki, and S. Helsen. 2006. "Feature-Based Survey of Model Transformation Approaches." *IBM Systems Journal* 45(3): 621-64.
[12] Hyun Seung Son, Jae Seung Kim, and R. Young Chul. 2012. "SMTL Oriented Model Transformation Mechanism for Heterogeneous Smart Mobile Models." *International Journal of Software Engineering and Its Applications* 7(3): 323-331.
[13] Woo Yeol Kim, Hyun Seung Son, Jae Seung Kim, and R. Young Chul Kim. 2011. "Adapting Model Transformation Approach for Android Smartphone Application." *Communications in Computer and Information Science* 199: 421-429.
[14] Woo Yeol Kim, Hyun Seung Son, Jae Seung Kim, and R. Young Chul Kim. 2010. "Development of Windows Mobile Applications using Model Transformation Techniques." *Journal of KIISE : Computing Practices and Letters* 16(11): 1091-1095.
[15] M. Utting. 2010. *Practical Model-Based Testing*. Elsevier.
[16] W.R. Elmendorf. 1973. *Cause-effect graphs in functional testing*, IBM Poughkeepsie Laboratory.
[17] Hyun Seung Son, and R. Young Chul Kim. 2014. "A Case Study on Metamodel of Cause-Effect Graph Based on Model Transformation for Mobile Software Testing." In *Proceedings of Advanced and*

Applied Convergence & Advanced Culture Technology, Jeju, Korea, November 15-17, 3: 5-7.
[18] Ecore Tools. Accessed March 12, 2017. http://eclipse.org/ecoretools/
[19] EMF. Accessed April 1, 2017. http://eclipse.org/emf/
[20] OMG. 2007. "MOF 2.0/XMI Mapping, v2.1.1." Accessed Mar 1, 2017. http://www.omg.org/spec/XMI/ 2.1.1/.
[21] IEEE Std 829-2008. 2008. *IEEE Standard for Software and System Test Documentation.* IEEE Computer Society.
[22] G. E. Mogyorodi. 2005. *Requirements-Based Testing - Cause-Effect Graphing.* Software Testing Services.
[23] Wikipedia. Accessed May 15, 2017. http://en.wikipedia.org/wiki/ATLAS_Transformation_Language.
[24] HiMEM, Accessed May 1, 2017. http://selab.hongik.ac.kr/down/tools/himem.zip.