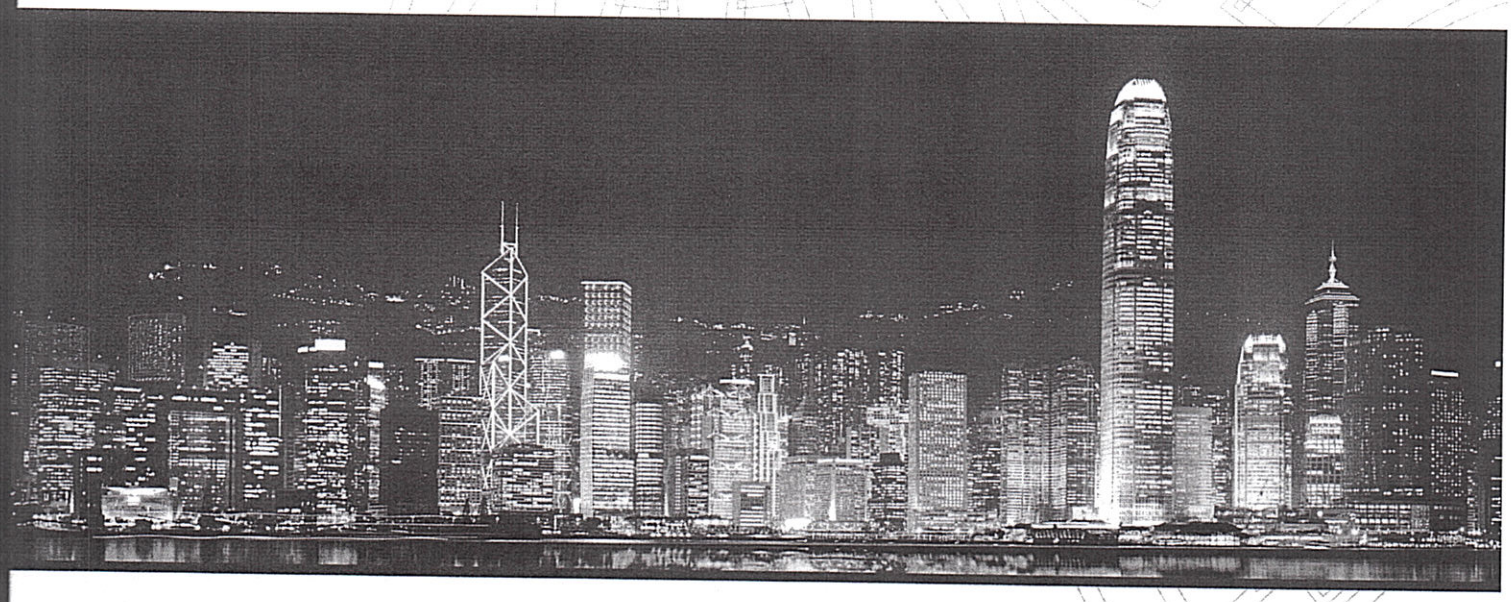
WCSE 2013



2013 FOURTH WORLD CONGRESS ON SOFTWARE ENGINEERING

3-4 December 2013 • Hong Kong, China

Swansea University
Prifysgol Abertawe

Software Requirements

Software Architecture and Design

Software Testing and Analysis

Theory and Formal Methods

2013 Fourth World Congress on Software Engineering

A Method of State Addreviation for Verification of Class Model

Soo-kyung Choi[#1], ByungHo Park[*2], Robert Young Chul Kim[*3] Young B. Park[#4]

[#]Dept. Computer Science & Engineering, University of Dankook
Cheonan, Republic of Korea
[1]krsoogom@gmail.com [4]*ybpark@dankook.ac.kr*

[*]SE Lab, Dept. CIC(Computer and Information Communication), University of Hongik
Sejong Campus, Republic of Korea
[2]sunsonbob@naver.com [3]*bob@selab.hongik.ac.kr*

***Abstract*—The Software defect management has become a critical issue with its increasing importance in sensor networks area. In this paper, it focuses on black box approach. Software defects can be found through model based testing. A state diagram is a good dynamic model that can test such a logical error of execution. However, the state diagram has a problem of complexity on the existing states and transitions. It is necessary to derive a state diagram in state based testing and find a method to solve its complexity problems. In order to use state diagram in software testing, its complexity has to be solved without the change of state and transition. This paper suggests a new notation called STMT (State Transition Mapping Tree) to solve the derived complexity without changing the state or transition. It also proposes an STMT automatic generation technique to derive a state diagram from a Java source code automatically. The suggested diagram can improve complexities partially, compared with UML state diagram.**

Keywords-STMT; Software Testing, Automatic Generation Method, State Diagram;

I. INTRODUCTION

The software defects can be detected through software testing. It is being studied as model-based testing and formal specification-based testing. Recent test techniques have been studied on state diagram. The state diagram plays an important role to understand the behavior of objects in the system and express the behavior pattern of the system dynamically so that it can represent objects clearly and heighten the potentials to complete a system which meets the requests [1]. The state diagram can be a dynamic model to test the logical execution errors and identify the visual behavior pattern of a software system dynamically, using UML State Diagram. However, it can be very complex, depending on the size of software systems [2]. It is necessary to consider how to show a state diagram and it can be accessed as a problem of Depth of the tree structure by applying its concept.

The existing state diagram approaches have solved the complexity by removing or changing the state and transition. They are useful in the field of automatic analysis, but have a limit to increase the danger of deriving unnecessary test cases from the viewpoint of software testing. Also, state based testing needs a derived state diagram which maintains the state and transition and has to solve its complexity. This paper suggests a new notation of STMT (State Transition Mapping Tree), a state diagram generated in order to solve the complexity of the state diagram derived without any change of the state or transition. In addition, it generates the state and transition from the Java source code and suggests an automatic generation technique of STMT with a single class and an STMT state diagram.

This paper is organized as follows. Section 2 looks at the related studies and Section 3 explained the suggested STMT notation and the automatic generation method. Section 4 compares STMT with UML State Diagram, using the example of Car Audio System and shows an example to generate STMT from the Java source code automatically with the example of Turnstile. Finally, Section 5 describes the conclusion and the future research directions.

II. RELATED WORKS

A. Notation

The typical notations of the state diagram are the traditional finite state machine based on Harel's Statechart and the State Diagram [3]. The one refers to a machine with a finite number of states and the other is a method to describe complex reactive systems and event-driven systems and include concurrency, hierarchy, internal events, and global variables. The State Diagram of UML is a method to extend Harel's Statechart notation. It can express the dynamic aspect of the system and appear as an event of the state and transition like the existing state diagram [4].

In addition, the State Diagram of UML has a notation of action and guard, and allows the overlapped state that one state includes the other. The State Diagram of UML provides Construct such as the Inter-level transition in order to model a complex system.

B. Complexity

A variety of studies to measure the complexity of state diagram has been conducted, and the existing ones have been

978-1-4799-2883-5/13 $31.00 © 2013 IEEE
DOI 10.1109/WCSE.2013.15

done to measure the structural characteristics and size of a model. ARACOS Research Group suggests 9 metrics to measure the complexity with the structural characteristics and size of state diagram, and use the proposed Metric of complexity as an assessment standard of understanding state diagram [5-7].

Not only ARACOS Research Group but also studies to measure understanding of the state diagram have been used the main characteristics of cohesion and combination as the main characteristics [8-9]. They have measured the cohesion and combination degree per state, considering a partition, a semantic state unit and the former and/or later conditions, and assessed the understanding degree of the state diagram with the average value of the measured cohesion and combination degree.

This paper uses the state diagram to propose its complexity studies of state as an indicator of the performance evaluation in a method of complexity improvement. Also, it has been studied with such a purpose from the viewpoint of the complexity in the software system itself so that it has used the measurement metric of the existing state diagram.

C. *Automatic Generation Method*

The studies, which draw a state diagram from a design document or a software system, can be divided into scenario-based ones and non-scenario-based ones. The studies on the derivation of the existing state diagram have been conducted as those to draw a dynamic model primarily from scenarios [10]. Typical scenario-based studies include S. Uchitel and J. Kramer's research, J. Whitlte and J.Schumann's, J. Whittle, R. Kwan, and Saboo's [11-13]. The common feature of scenario-based studies is to use Message Sequence Chart and UML Sequence Diagram as a notation for the specification of the scenario. S. Uchitel and J. Kramer's research uses Message Sequence Chart (hereafter MSC) in the specification of the scenario, and on the other hand, J. Whittle and J. Schumann's and J. Whittle, R. Kwan, and Saboo's use UML Sequence Diagram in the specification of the scenario. These scenario-based studies use Sequence Diagram of MSC and UML for the specification of the scenario setting, and then transform the specified MSC and UML Sequence Diagram, using State Vector and Domain Knowledge, and then the state diagram is derived.

The studies on the derivation of the non-scenario-based state diagram specify the information about the state and the state diagram transition, using the specification language such as OCL and ODL of the state diagram itself. They use the specified information to automate the state diagram or change the state and transition to solve the complexity [14-15]. The non-scenario-based studies of state diagram use the specification in order to generate a state diagram and derive it from a class. Also, they require the information about the state and transition in advance, but they neither need Sequence Diagram like scenario-based ones nor ask a specific scenario as a pre-condition.

III. AUTOMATIC GENERATION METHOD OF STMT

State diagram-based software testing needs a derived state diagram of the state and transition and a method to solve the complexity of the state diagram. This section defines the notation of STMT (State Transition Mapping Tree, hereafter STMT) and suggests a technique of its automated generation.

A. *Notation*

STMT has been studied on the basis of UML State Diagram and its state is marked with state names and data types different from the notation of the existing state diagram because the state action and guard of STMT are assumed to have already been satisfied. Also, STMT Construct such as Inter-level transition and History are provided by UML State Diagram and excluded due to the overlapped ones.

The STMT map is means a single class with more than one state, and they are tied to a map. In other words, a map is a set of states. This paper adds the concept of Map and defines a single class-unit as a map, which is defined to have its sub-states. The type of a map is divided into maps, map nodes, and map roots. Map nodes mean a map which has one as a state inside itself, and map roots refer to the top-level root node. A STMT tree structure is derived, using such maps, and in addition, a state diagram should be drawn as a hierarchical structure so as to derive a tree structure. The relationship between the maps is classified as M, M-MN, and MN-MN Relationships, and M refers to a map, MN a map node, and MR a map root. Map node and Map root have the same internal operation and configuration only with the difference in the existing number of the locations and states. Thus, the definition of the relationship between maps is overlapped with MN, and it becomes unnecessary to define. Each node has its own information, and Map Type means M, MN, and MR, types of nodes here in STMT. Additionally, each node has its own name information, which is consistent with a single class name. A single class defined in the above Definition 1 can be written as a map, and when it is present as a state inside another map. The relevant single class will exist as a state. STMT nodes are derived from each single class when STMT notation is used, and they are referred to as STMT depending on the relationship between the state and the map.

When expressed in STMT, a leaf node refers to a node whose node type is a map, and a root node means a node whose type is a map root. An example written in STMT notation is shown in the following Fig. 1.

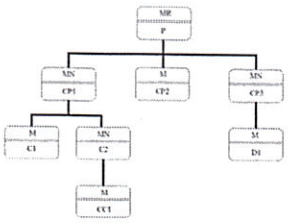


Figure 1. Example of STMT.

Fig. 1 marked the relationship between nodes and STMT node names randomly. In addition, the node names were made consistent, and then one figure was added on the basis of the assumption that the relationship between maps and their internal states need to have the same type.

B. *Automatic Generation Method*

When the tree structure of STMT is used, it is possible to derive the state and transition of STMT from Java source code. Once Java AST (Abstract Syntax Tree) is utilized provided by JDT (Java Development tool), the analysis framework of Eclipse Java Syntax in the case of Java source code, it is also possible to develop an automated tool of STMT generation. In order to implement it, different generation rules of STMT and several assumptions are needed for analyzing Java source code and deriving the state and transition of STMT.

STMT analyzes a single class as far as the unit of method, and it can draw the state and transition, depending on the method type, besides, the definition about the depth value is necessary to establish the relationship between the generated maps. In the basic mapping of the state and transition, the state is the value of property and the transition indicates a behavior (movement) to cause changes so that they are mapped depending on the type of mapping. Also, a map is generated as a unit of a single class and exists as a state in a map node if it is present under another node. Thus, the transition information between map nodes inside themselves become existing in a map node.

The following explains why the behavior to cause changes of a state value in STMT is referred to as a method unit. A single class can be largely expressed as a property and a method, and include different types of syntax as its internal factors. The state diagram of STMT has an interest in the state change as a method unit so that it is not necessary to search for detailed syntax. That is, the type of Method Internal Syntax is limited to be used as a STMT mapping factor because the information about the relationship of Method Sending can be obtained with Return Statement, Expression Statement, and Variable Declaration Statement alone. Several assumptions are required to generate STMT from Java source code, using Java AST of Eclipse JDT on the basis of fundamental definitions of STMT generation. On the basis of such assumptions, necessary information can be derived to generate STMT from Java source code. Moreover, redefinition is required for the class information derived with JAVA AST of Eclipse JDT to be used as necessary information. Then the field of Depth is added to store the depth information to draw from the definition about the map relation in the redefined class information. When it is referred to as CInfo(i), it can map the information about the state and transition needed to generate STMT from CInfo(i).

A mapping algorithm for STMT state search for the information of field which a single class has per each, and then if any information is mapped with STMT, it will be derived and stored in the object of state information. After the mapping for STMT state is finished, the mapping for STMT transition occurs, and the transition in STMT indicates mapping the transition information which can determine Source or Destination as a unit of a single class from the derived CInfo list. The STMT transition mapping requires three types of syntactic analyses are required in the method body, and they are analyzed by the algorithm of STMT syntactic analysis to parse the internal syntax of the method body.

The transition information of STMT is returned as a result of the syntactic analysis. The returned information is stored as the transition of STMT, and the derived state and transition information of the derived STMT is stored as map information of a single class. If any single classes do not satisfy the prior assumption and definition during the mapping process, they are not defined as maps, and only satisfactory single classes are mapped as STMT maps.

Such automatic technique of STMT generation can implement an automated tool to generate STMT from Java source code. It analyzes a source code and derives necessary information to generate STMT, and also, it is used to generate the state diagram between STMTs.

IV. Case Study: Compare between "UML-State Diagram" and "STMT-STD"

This section presents a case study, which compares State Diagram of UML and STMT notation suggested in this paper, using the state machine of Car System [16] and measures the complexity of each state diagram.

Table 1 shows the definition for the basic state of Car Audio System state machine [16].

TABLE I. Definition for States of Car Audio System

[1]	State of Car Audio System:: Audio Player
[2]	Audio Player : ON(Tuner Mode, CD Mode, Tape Mode), OFF
[3]	Tuner Model : P1, P2, P3, P4
[4]	CD Mode : Playing, Next, Former
[5]	Tape Mode : Playing, Forward, Backward

In the Car Audio System, the sub-state of Audio Player is largely divided into ON and OFF, and the sub-states of ON such as Tuner Mode, CD Mode and Tape Mode indicates that the present sub-state of ON state is chosen respectively. Fig. 3 shows the result represented as the State Diagram of UML with the example of Car Audio System [16].

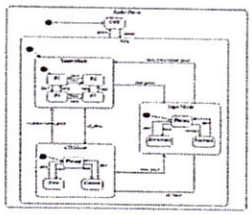


Figure 2. Car Audio System: UML State Diagram.

The notation of STMT state diagram depends on the depth setting. The part of ON state is represented partially. In other words, it is set as one map in the STMT State diagram, and the Depth can be increased in order to express the internal state of ON.

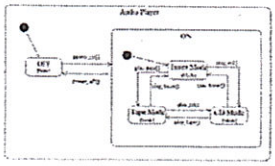


Figure 3. Car Audio System: STMT State Diagram (Depth=2).

Fig 3 is the STMT state diagram derived with the value of increased Depth 2 as a standard and the representation of ON can be identified as far as its internal state. Similarly, the sub-state of ON is a map type which has its own internal state, and when expressing such diagrams, Depth can only be set to Depth=3 with the increase of 1. However, Depth 3 is omitted because it derives the same as UML state diagram when Depth is set to 3.

Table 2 shows the result of complexity calculation for the derived UML and STMT state diagrams by the complexity measurement Metric of the existing state diagram [5].

TABLE II. COMPLEXITY: "UML STATE DIAGRAM" AND "STMT-STD"

	NSS	*NSC*	*NT*	*CC*
UML State Diagram	11	4	24	11
STMT-STD(Depth=1)	2	-	2	2
STMT-STD(Depth=2)	4	1	8	2
STMT-STD(Depth=3)	11	4	24	11

It is identified that the complexity of Car Audio System, which is expressed as STMT state diagram, becomes different by the change of Depth, a partial measurement of expression method through Table 2. In other words, the notation of STMT state diagram is seen to improve the complexity of a state diagram partially, and also, if STMT state diagram is expressed as far as the overall inside without the improvement of partial complexity, it can have the same complexity as that of UML state diagram. STMT state diagram improves the complexity of state diagram partially with the use of Depth. Furthermore, it can do it by searching for only the interesting part in Top-Down method.

V. CONCLUSION

Many damage cases are on the rise which has not been properly validated after manufacturing defective products, and have made execution errors, causing a huge quality cost and deadly results in the development of software systems. These software defects can be detected through software testing, and state diagram is a good dynamic model to test logical execution errors. In order to use state diagram in software testing, its complexity has to be solved without the change of state and transition. Therefore, this paper suggested a new notation to solve the complexity of state diagram derived without any change of state and transition, and STMT to generate state diagram based on a single class with the use of a tree structure. Additionally, this paper suggested an automatic generation technique of STMT. It was able to partially improve the overall complexity of state diagram with the suggestion of STMT and proposed an automatic generation technique of STMT which can automate the derivation of STMT from Java source code.

Future research is expected to improve the constraints of automatic generation techniques, and also, it needs to draw test cases from the derived STMT in order to automate software testing.

ACKNOWLEDGMENT

This work was supported by the IT R&D program of MSIP/KEIT. [10044457, Development of Autonomous Intelligent Collaboration Framework for Knowledge Bases and Smart Devices] and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013R1A1A2011601)

REFERENCES

[1] IPL Information Processing LTd, "Testing State Machine with AdaTEST and CANTATA," IPL paper, Mar 2011.

[2] R.V.Binder, "Test Object-Oriented Systems: Models, Patterns, and Tools," Addison Wesley, 1999.

[3] C. Larman, "Appyling UML and Patterns," PrenticeHall, 2004.26

[4] Scott W. Ambler, "The Elements of UML 2.0 Style," CAMBRIDGE, May 2005.

[5] Jose A. Cruz-Lemus, Ann Maes, Marcela Genero, Geert Poels and Mario Piattini, "The impact of structural complexity on the understandability of UML statechart diagrmas," Information Science: and International Journal, Vol.180, Issue 11, pp.2209-2220, June 2010.

[6] Marcela Genero, David Miranda and Mario Piattini, "Defining Validating Metrics for UML Statechart Diagrams," Lecture Notes in Computer Science, vol 2814, pp.118-128, 2003.

[7] Jose A. Cruz-Lemus, Marcela Genero, Jose A. Ovlivas, Francisco P. Romero and Mario Piattini, "Predicting UML Statechart Diagrams Understandability Using Fuzzy Logic-Based Techniques," Proceedings of the sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2004), pp.238-245, 2004.

[8] Martin Hitz and Behzad Montazeri, "Measuring Coupling and Cohesion In Object-Oriented Systems," In Proceedings of the International Symposium on Applied Corporate Computing, 1995.

[9] Jung Ho Bae, Yeon Ji Jeong, Heung Seok Cae and Crl K. Chang, "Semantics Based Cohesion and Coupling Metrics for Evaluating Understandability of State Diagrams," Computer Software and Applications Conference(COMPSAC) 2011 IEEE 35th Annual, pp.383-392, July 2011.

[10] H. Liang, J. Dingel and Z. Diskin, "A comparative survey of scenario-based to state-based model synthesis approaches," In SCESM, pp5-12, 2006.

[11] S. Uchitel and J. Kramer, "A workbench for synthesizing begaviour models from scenarios," In ICSE, pp.188-197, Jun 2001.

[12] J. Whittle and J. Schumann, "Generating statechart designs from scenarios," In ICSE, pp.314-323, Jun 2000.

[13] J. Whittle, R.Kwan and J. Saboo, "From scenarios to code: An air traffic control case study," In ICSE, pp.490-495, May 2003.

[14] Agung Fatwanto, "Software Requirements Translation from Natural language to Object-Oriented Model," IEEE Control, System & Industrial Informatics (ICCSII), pp.191-195, Sep 2012.

[15] Jung Ho Bae and Heung Seok Chae, "An Automatic Approach to Generating a State Diagram from a Contract-Based Class," 2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp.323-331, Apr 2009.

[16] Dirk Seifert, "Test Case Generation from UML State Machines," Inria, version 2-23, Apr 2008.