An optimized approach to generate object oriented software test case by Colored Petri Net

Esmaeil Mirzaeian
Department of IT and Communication
Payam Noor University
Tehran, Iran
E-mail: emirzaeian@gmail.com

Samad Ghaderi Mojaveri
Department of IT and Communication
Payam Noor University
Tehran, Iran
E-mail: xghaderi@gmail.com

Homayun Motameni
Department of Computer Engineering
Islamic Azad University
Sari, Iran
E-mail: motameni@iausari.ac.ir

Ahmad farahi
Faculty Member of IT and Communication Department
Payam Noor University
Tehran, Iran
E-mail: afaraahi@pnu.ac.ir

***Abstract*—in object-oriented software testing, a class is considered to be a basic unit of testing. Attributes of object-oriented software such as inheritance and polymorphism make behavior analysis and test significantly complicated because the state of the objects may cause faults that cannot be easily revealed with traditional testing techniques. In this paper, we propose a new technique for generating the test case by Colored Petri Nets (CPN), which is an extended version of Petri Nets and usually used to system modeling and simulation. Our method considers net-explosion problem and also our generated Net covers all Instances of Objects from Different Classes in the same hierarchy by introducing new algorithm to convert UML Statechart to CPN. A case study is presented to show the benefit of our approach and resulting Net is implemented in CPN-Tools.**

Keywords-test cases; colored petri net; state space graph; object-oriented;

I. INTRODUCTION

Software testing is a crucial activity to guarantee the quality and the reliability of the software. It is often said that the cost for correcting an error after software release is four times more than doing an error found at testing phase, and even 50 times more than at design phase [1, 2]. Object-oriented (OO) approach is one of the approaches to develop software efficiently that enabling us to reduce or eliminate some typical problems of procedural software, but may introduce new problems that can result in classes of faults hardly addressable with traditional testing techniques [3,4]. In particular, statedependent faults tend to occur more frequently in OO software than in procedural software. Almost all objects have an associated state, and the behavior of member function invoked on an object typically depends on the object's state. Such faults can be very difficult to reveal because they cause failures only when the objects are exercised in particular states [5].

One of the most important issues in the area of class testing is test case generation, which generates a set of test data from class specifications to ensure class implementations work properly. There have been some test case generation methods proposed in the literature. Most of them are based on Extended Finite State Machine (EFSM) models, such as [6, 7, 8]. However, these models are only a program verification technique, and produce events by observing a carefully chosen path in the EFSM to confirm the correctness of the traversed transitions in the path. A method for generating test cases that detects the given faults is proposed in [9].

In this paper, we propose a new technique for generating the test case by Colored Petri Nets (CPN). In order to overcome net explosion problem we adopt the idea proposed in [10], we picked UML statechart rather than state transition diagram (STD) and we introduce rules to make special Tokens named Object Token (OT) that covers all Objects instead of simple symbolic Tokens. This changes enabling us to introducing new algorithm to convert UML Statechart to CPN that is capable to covers all Instances of Objects from Different Classes in the same hierarchy to generate test case by Existing tools such as CPN-Tools. A case study is presented to show the benefit of our approach and resulting Net is implemented in CPN-Tools.

The rest of the paper is organized as follows. The next section is an introduction to CPN and UML Statecharts. In section 2, we show the basic idea of translating statechart to CPN. Section 3 presents the steps of our Translation technique and its Mapping Algorithm. Section 4 presents a simplified case Study of banking system account and its analysis by using the existing tool of CPN, called CPN-Tools, and is also presented. In section 5 conclusion and future work is presented.

II. CPN AND UML STATEHARTS

In this section, we illustrate general concept of CPN and a brief sketch of UML Statechart is also mentioned in this section.

978-1-4244-8666-3/10/$26.00

A. Colored Petri Net (CPN)

Petri-Nets [11], is one of formal techniques that has the ability to model concurrency of systems and the ability to analyze concurrent behavior.

In Colored Petri Nets (CPN) [12], proposed by Jansen, which is an extended version of Petri net the tokens have values which are typed with "color" and the computation expressions on "colors" are associated with transitions.

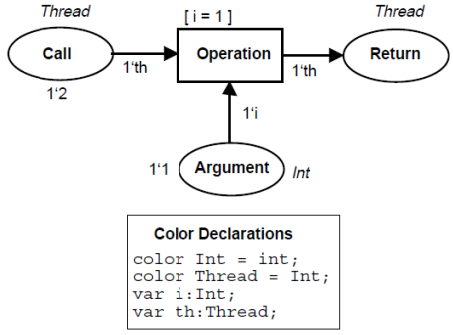

Figure 1. An example of CPN

Figure 1 [10] illustrates its simple example. It contains "places", "transitions" and "arcs", which are represented with circles, rectangles and arrows respectively. Marking, which is a map from places to tokens, expresses the state of the system that is specified with a Petri net. The movement of tokens denotes state transitions. More concretely, if each of the input places to a transition has at least one token, the transition "fires" and the tokens move to the output places. This movement corresponds to a state transition of the system. In CPN, we can attach some attributes so called "colors" to places.

The attributes of tokens are defined with "colors" as the types of the attribute values. In the figure, the place "Call" has exactly one token and the token has the value "1" whose "color" is "Thread" (int.integer), the readers can find the colors associated with a place. They denote which colors of tokens can be accepted at the place. For example, the place "Call" can only receive the tokens of the color "Thread". Expressions can be attached to arcs which connects transitions with places. The expressions restrict the tokens that can flow on the arcs. In this figure, the expression "1 'i" associated with the arc between places "Call" and transition "Operation" represents that exactly one token can flow on it. The attribute value of the flowing token is assigned to the variable "th", whose color is "Thread", occurring in the expression. We can describe a "Guard" on a transition to control firing the transition. In the figure, "Guard" is represented "[i = 1]" which means that if the value of a token from "Argument" place is 1, then the guard condition is satisfied.

A transition in a CPN is fireble if the following conditions hold.

- Each of the places input to the transition has at least one token.
- The expressions attached to the input arcs to the transition hold for the tokens in the input places.
- The guard attached to the transition hold.

B. UML Statechart

UML state diagram [13] models the behavior of a single object. It specifies the possible abstract states of the instances of a class. Its basic elements are:

1) Simple State

A simple state represents one of the finite numbers of abstract states in which the object modeled by the state diagram may find itself. It is a state of the object during which it satisfies some conditions, performs actions and waits for events. In UML such a state is represented by a rounded rectangle.

2) Pseudo State

A state diagram starts with a pseudo initial state shown by a small solid circle. The solid circle is, in fact, marking the initial state and that is why it is a pseudo state. A bull's eye circle represents the final pseudo state. A state diagram must have the initial pseudo state, although the final pseudo state is optional.

3) Composite State

A Composite State is composed of more than one sequential or concurrent sub-state and is called a sequential composite state or a concurrent composite state depending upon the kind of sub-state it has.If a sequential composite state is active then exactly one of its sub-states is also active. If a concurrent composite state is active then one of the nested states from each concurrent state is also active.

4) Transition

A transition represents an allowed change from a source state to a target state. Transitions from one state to another are represented by a directed edge. A transition may have an event, a guard and an action associated with it. An event is the cause of a transition and is sometimes called a trigger. A guard is a Boolean expression, presented in square brackets, that prevents a transition being taken unless the condition evaluates to true. An action is a function that represents the effect of a transition and is invoked on the object that owns the state machine as a result of the transition. Instead of going to a different state, a transition may have the same source and target state. Such transitions represent situations where a message is received but does not result in a change of state. These transitions are called self-transitions. Transitions without an associated event are called trigger-less transitions. Figure 2 (a) shows typical statechart of super-class and (b) inherited class.

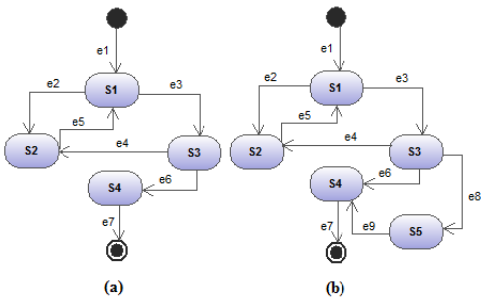

Figure 2. A simple statechart

III. RELATIONSHIP BETWEEN CPN AND STATECHART

In [14] a mapping from STD to low-level Petri net is presented. We adapt this idea in our mapping approach to convert each state and its corresponding transitions to an equivalent CPN place and transition as it is shown in figure 3. a

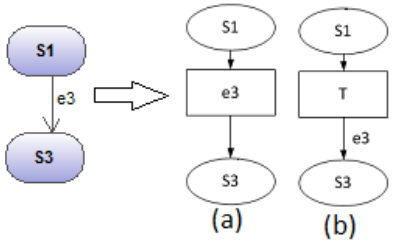

Figure 3. A simple mapping of Statechart to CPN

We use this simple mapping in different way, just as it shown in figure 3.b. in order to make this method suitable to cover all instances of objects in OO system with more complex data structure instead of low level tokens more changes must be applied. The following sections discuss our mapping in more detailed steps.

IV. CONVERTING UML STATECHART TO CPN

In this section we introduce some definitions which are essential in Object Token concept and its structure. Detailed steps and mapping algorithm are also presented.

A. Definitions

In OO systems object state in the only difference between instances of one class and it depends on value of its properties; therefore object data structure plays important role in OO software system simulation. Other class member such as methods can be implemented once and can be used for all instance of under simulating objects. To present the data structure used in our mapping method, we categorized the data member of class in two categories.

Definition 1. Sensitive data member is the one that changes in its value may cause the object to change its state. For example in banking account system changes that make the data member balance to negative cause the account to go to overdrawn state as it is shown in figure 5.

Definition 2. Sensitive method is the one that can change sensitive data member directly or indirectly. For example in banking account system, a method such as deposit is considered sensitive method as long as it can change the sensitive data member balance.

Definition 3. Suppose that S is a typical state in stetechart then pre[S] refers to all transitions that enters to state S and next[S] refers to all transitions that leave state S. For example in figure 2.a

$$\text{pre}[s2] = \{e3, e4\}\ ,\ \text{next}[s2] = \{e5\}$$

B. Object Token

In order to cover all objects in the final CPN a special type of token must be constructed that make it possible to distinguish different type of bject. To handle complicated behaviour of OO systems such as polymorphism and dynamic binding, we introduce a set of sensetive data member of all classes in record colorset format as CPN colorset. We also add another essential item in it such as type with enum coloreset, so we can use it to identify different type of object in during simulation. It also useful to apply type constraint in our method. Other optional items can be added to this record when we need to save more information. we refer to this token as Object Token (OT) and variables of this type as ot. Object Token and its defined variables are the only tokens that flow in the generate net. When ot passes a transition or an arc, the simulated event changes its content values by applying defined function.

C. Mapping to CPN

In this section we present our mapping method in order to obtain single and optimized CPN with least number of CPN items based on specifications written for related classes in same hierarchy. This method uses statecharts of these classes. The mapping includes following steps:

Step 1: this step consist of collecting all states in statechart of class I as S_i. Then we construct comSet as the set of all common sates in all generated S_i and sumSet as set sum of all generated S_i and we also must construct another set for exclusive states in each statechart of typical class I as $S_{Exc}i$.

Step 2: in this step we use the generated sets from step1 to construct CPN, other type of sets are required in this step such as $pre[S]$ and $next[S]$ that are defined as Definition 3.

Step 3: composite states must be mapped in this step by repeating steps 1, 2 for each composite state and conncet its corresponding entrypoint and exitpoint to input and output transition.

Step 4: this step uses exclusive sets and Object Token to apply type constrain for Generated CPN, that is, each exclusive state must by guarded by a constraint so that only valid Tokens are qualified to enter and pass through those states.

More details are presenetd in the following algorithm.

Step 1: ***Generating Sets***
Foreach existing statechart C do
S_c=Set of all states in C
SumSet= $S_1 \cup S_2 \ldots \cup S_i$
ComSet= $S_1 \cap S_2 \ldots \cap S_i$
Foreach S_i do
$S_{Exc}i$=$S_i - ComSet$
Step 2: ***Creating initial CPN***
Foreach state $S \in SumSet$ do
Create place P of type OT
Foreach $m \in pre[S]$ do
Create transition T and Connect T to P with Arc A
Set Arc inscription A to m(ot)
If m comes from initialState then
Create Place iP of type OT
Connect iP toT with Arc A
Set Arc inscription A to ot
Foreach $m \in next[S]$ do
Create transition T and Connect P to T with Arc A

Set Arc inscription A to m(ot)
If m goes to final state then
Create Place oP of type OT
Connect T to oP with Arc A
Set Arc inscription A to ot

Step 3: ***Managing Composite States***

Foreach composite state $CS \in SumSet$ do
Repeat step 1-2 for CS to generate subCPN
Connect input of CS to entryPoint Place subCPN
Connect output of CS to exitPoint Place subCPN

Step 4: ***Adding Type Constrains***

Foreach existing none empty $S_{Exc}i$ as S do
Foreach Arc A mapped from pre[S] do
Add type constrain to A

Final net may have transitions that fork multiple places with no guard. In such transitions, to selsct each individual outgoing arc, we can attach a place with random selection, although depending on analysis scenario, one may use specified initial marking at this place to switch between outgoing arcs in predefined order.

It is important to mention that we ignore none-sensitive methods in our approach because these methods can not change object state. All These methods can be tested in arbitrary order or individually, although in some other test level such as inter-class testing these methods must be considered as well.

V. CASE STUDY

In this section, we use our approach to generate CPN from available UML statechart of simplified Account Class hierarchy of banking System. This system consists of a super-class 'Account' and two subclass 'Credit' and 'Saving' Account as shown in figure 4.

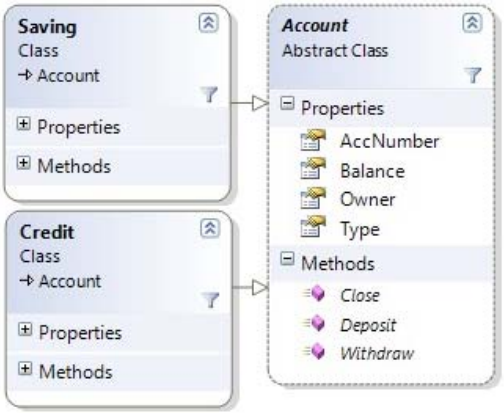

Figure 4. Account Class of Banking system

Statecharts of these classes is also presented in figure 5.

We mentioned ealier that Object Token can be constructed from sensitive data member of all available classes in the same hirarchy. This information can be extracted from class diagram, such as figure 4.

According to definition 1 and Object Token concept, we construct Object Token in CPN-ML standard as follow:

Colset Type = with Credit | Saving;

*Colset Account = record Accno:INT * B:INT * InvB:INT *AT:Type;*

The Account colset is a record with two sensitive data member B (Balance) and InvB (Investment Balance) plus Type field. Another optional field (Accno) is added to enable us to identify different objects from same Type.

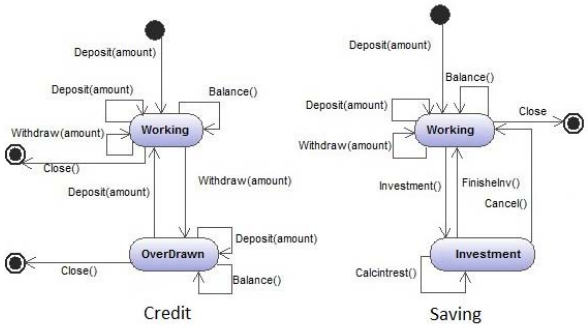

Figure 5. Class diagram

Resulting CPN by applying our proposed algorithm to statecharts is implemented in CPN-Tools. In order to show the final CPN we implemented it in multi-level form to make it easy to understand. It is shown in figure 6-9.

To analyze the behavior and to generate test cases, at first we should generate State Space Graph (SSG) from the CPN. The SSG expresses traces of the marking of a CPN, i.e. tokens on places. A node and an arc in the graph represent a marking and a firing of a transition respectively. SSG can be automatically generated and analyzed by existing tools such as CPN-Tools and ASAP (Ascoveco state –space analysis platform-CPN group).

VI. CONCLUSION AND FUTURE WORKS

Attributes of OO software such as inheritance and polymorphism make behavior analysis and test significantly complicated because the state of the objects may cause faults that cannot be easily revealed with traditional testing techniques. In this paper, we propose a new technique for generating the test case by Colored Petri Nets (CPN). Our method considers net-explosion problem and also our generated Net covers all Instances of Objects from Different Classes in the same hierarchy by introducing new algorithm to convert UML Statecharts to single and optimized CPN. Our work in this paper considers generalization relationship between classes. In general, there are three types of relationship between classes: association, aggregation and generalization. We are investigating to expand our method to cover association and aggregation relationship based on the extended version of Petri Nets.

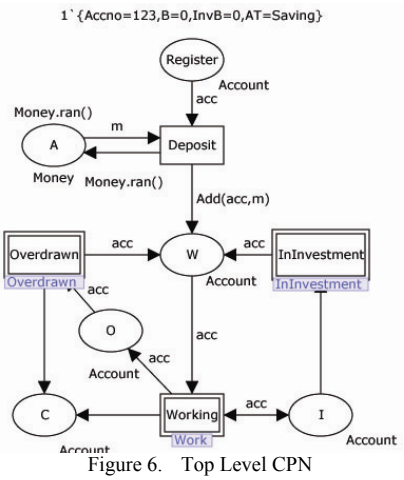

Figure 6. Top Level CPN

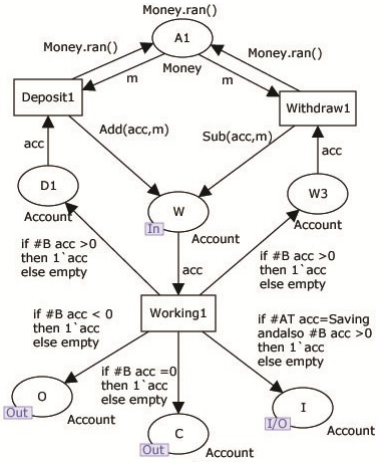

Figure 7. Sub-Level working

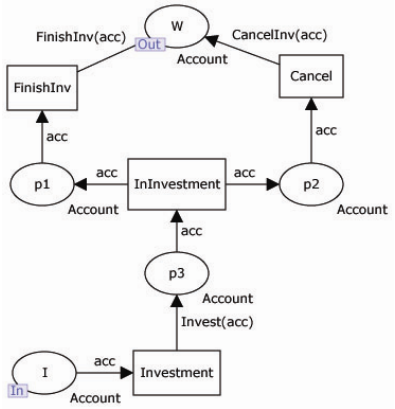

Figure 8. Sub-Level Investment

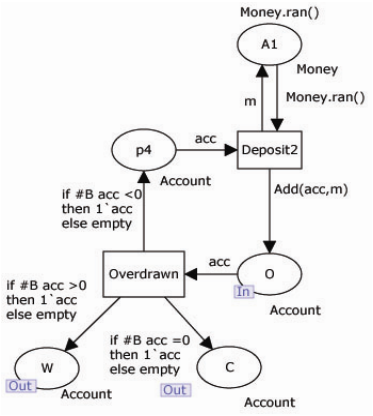

Figure 9. Sub-Level Overdrawn

REFERENCES

[1] R. S. Pressman, ”Software Engineering – A Practitioner’s Approach Fifth Edition”, McGraw-Hill, 2001

[2] H. Zhu, P. Hall, and 1. May, "Software Unit Test Coverage and Adequacy", ACM Computing Surveys, April, 1997,pp.366-427.

[3] S. Barbey and A. Strohmeier, "The Problematic of Testing Object-Oriented Software", In Proceedings of the Second Conference on Software Quality Management, Edinburgh (Scotland, UK), vo1.2, July, 1994, pp. 411-426.

[4] A. Orso and S. Silva, "Open Issues and Research Directions in Object-Oriented Testing". In Proceedings of the 4th International Conference on "Achieving Quality in Software: Software Quality in the Communication Society" (AQUIS'98), Venice, April, 1998.

[5] V. Martena, A. Orso and M. Pezze, "Interclass Testing of Object-Oriented Software", In Proceedings of the 8th IEEE international Conference on Engineering of Complex Computer Systems (ICECCS'02), 2002.

[6] J. 1. Li and W. E. Wong, "Automatic Test Generation from Communicating Extended Finite State Machine (CEFSM)-Based Models", In Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real¬Time Distributed Computing (ISORC.02), 2002.

[7] R. M. Hierons, T. H. Kim and H. Ural, "Expanding an Extended Finite State Machine to Aid Testability", In Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSACp02), 2002, pp. 1-6.

[8] A. Y. Duale and M. Uyar, "A Method Enabling Feasible Conformance Test Sequence Generation for EFSM Models", IEEE Transactions on Computers, Vol.53, No.5, 2004,pp.614-627.

[9] H. F. Gong and J. Li, "Generating Test Cases of Object-Oriented Software Based on EDPN and Its Mutant ", Proceedings - IEEE The 9th International Conference for Young Computer Scientists ,Hunan ,Nov,2008, ICYCS, pp.1112-1119.

[10] H. Watanabe, H. Tokuoka, W. Wu, M. Saeki, "A Technique for Analyzing and Testing Object-Oriented Software Using Colored Petri Nets," apsec, pp.182, Fifth Asia-Pacific Software Engineering Conference (APSEC'98), 1998

[11] J. L. Peterson, "Petri Net Theory and the Modeling of Sys¬tems", Englewood Cliffs, New Jersey, Prentice Hall Inc., 1981,

[12] K. Jensen, "An introduction to the theoretical aspects of Colored Petri Nets" , Springer Berlin , 2006

[13] A.A.Bokhari and W.F.S.Poehlman, "Formalization of UML State-Charts: Approaches for Handling Composite States", Department of Computing & Software, McMaster University, Technical Report CAS 2005-07-SP (October, 2005), 10 pp.