International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 4 Issue: 2 130 - 133

Optimizing Test Cases for Object-Oriented Software

ChaitraM, Prakruthi M.K, Sarala N.R
Dept. of CSE,
SJB Institute of Technology,
Bangalore
e-mail: chaitram@sjbit.edu.in, prakruthimk@sjbit.edu.in , saralanr@sjbit.edu.in

Abstract **-** Testing object-oriented software is a challenging task. The inherent complexity in testing Object-oriented software is due to issues like inheritance and polymorphism. The behavior analysis and testing of object oriented software is significantly complicated because the state of the objects may cause faults that cannot be easily revealed with traditional testing techniques. This article proposes an improved technique for generating optimal number of test cases using mathematical techniques. The technique uses Colored Petri Nets (CPN), which is an extended version of Petri Nets. CPN's are usually used for system modeling and simulation. The proposed method explores the problem to generate test cases that covers all instances of objects from different classes in the same hierarchy. It shows the effectiveness of technique by translating a specification represented by UML (unified modeling language) state chart into a CPN. The main solution of our approach will be implemented using CPN-tools.

Keywords—*Colored Petri Nets (CPN), Object Oriented Software, Test cases, Software testing, UML*

*****

I. INTRODUCTION

Software testing is done to guarantee the quality and the reliability of the software. 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 used to develop software efficiently. It enable us to reduce or eliminate some typical problems of procedural software. At the same time it may introduce new problems that can result in classes of faults hardly addressable with traditional testing techniques [3, 4]. The state dependent 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].

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 proper working of Class implementations. Many object-oriented methods recommend using a finite state machine to describe an object's behavior. But most of them are based on Extended Finite State Machine (EFSM) models [6]. These models are used as a program verification technique. It produces an event by observing a carefully chosen path in the EFSM to confirm the correctness of the traversed transitions in the path [7, 8]. A method for generating test cases that detects the given faults is proposed in [9].

The main propose is to use an improved technique for optimizing and minimizing the number of test cases by Colored Petri Nets (CPN). In order to overcome state explosion problem, pick the UML state chart rather than state transition diagram (STD) [10]. The introduced rules have special tokens named Object Token (OT) that covers all objects instead of simple symbolic tokens.

II. CPN AND OBJECT ORIENTED CONCEPTS

Petri-Nets, is one of the formal techniques that has the ability to model concurrency of systems and the ability to analyze concurrent behavior. The ordinary Petri nets are highly dependent on the system and lack the modularity and flexibility. So, in order to solve the complexity, Petri nets are combined with Object-Oriented methods to set up the Colored Petri Nets. In Colored Petri Nets (CPN), 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. The attributes of tokens are defined with "colors" as the types of the attribute value. The fundamental ideas of CPN have token types, and each token type has some data value associated with it [11]. Few researchers have challenged to apply Petri Net and its family to analyzing and testing concurrent software systems by using some techniques for analyzing Petri-net. However if we apply these techniques to objected-oriented software as they are, petri-nets expressing objects are produced whenever creating them during its being executed. It causes explosion of increasing the number of nets. That is to say, as object-oriented software is being executed, the structure of the corresponding nets should be dynamically changed. The existing Petri-net analysis techniques is impossible to be applied to object-oriented software by this reason.

To solve this problem, a new technique of object-oriented software analysis and testing by using Colored Petri Nets(CPN)[5], which is an extended version of Perti Net is proposed.

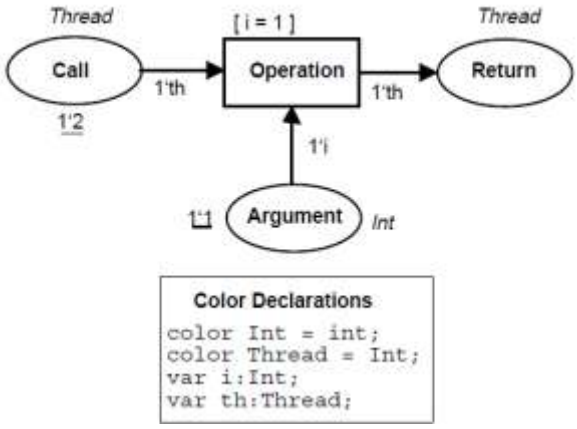


Fig 1: Example of a CPN

Figure 1 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 place "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.
1. Each of the places input to the transition has at least one token.
2. The expressions attached to the input arcs to the transition hold for the tokens in the input places.
3. The guard attached to the transition hold.

1.Relationship between OO concepts and CPNs

Object-oriented paradigms have the properties like class abstraction, inheritance, polymorphism and message passing. A class is a set of objects having common properties, i.e. attributes (instance variables) and methods (services).Classes also have relationships such as aggregation and generalization. A specification of OO software is constructed by using some kinds of object-oriented analysis and design method. UML is used to write specification. UML state diagram [12] models the behavior of a single object. It also specifies the possible abstract states of the instances of a class.

In OO software, objects in a software are concurrently executed, i.e the states of the object are being concurrently and independently changed. The behavior analysis of OO software becomes complicated due to the dynamic changes of state transition diagrams, and hence the existing techniques for analyzing state transition diagrams cannot be applied. Using CPN technique, problems can be solved by representing an object with a colored token and the behavior of all the objects belonging to the same class can be expressed by a single net. In this technique, the attribute value of a token is used for identifying the corresponding object. This is the major reason why we haven't used usual Petri net but Colored Petri net. Optimization is used to optimize an objective function subject to constraints. The constraints are the demands necessary to satisfy a criterion. The objective function is to minimize the number of test cases for object-oriented software.

2.Analysis and Testing

First of all, we should clarify the relationship between our approach, i.e. translating OO software to CPN, and testing. We can classify three levels of testing based on our technique as shown in Figure 2i.erelation between our approach and testing.

1) *Testing in Specification Level (prototyping) : Simulation*

We assume that a specification of OO software is constructed by using some kinds of Object-oriented Analysis and Design Method. As we mentioned, we use UML to write a specification. Applying our approach to object diagrams and state transition diagrams, we can get a CPN that simulate the behavior of the specified system. By executing the CPN, we can use our approach as one of prototyping. We have an analysis tool called Design/CPN and we can analyze various kind of property of the CPN such as deadlock-detection and reach ability of specific states.

Furthermore we developed the efficient algorithm of generating test cases from a CPN. Thus we can systematically test a UML specification by using the generated test cases with large coverage.

2) Testing in Program Level : Debugging

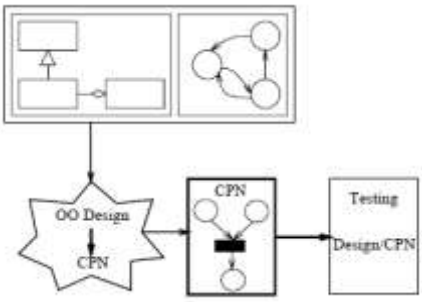
Programs written in an object-oriented language such as Java can be translated into a CPN. The same analyses as UML specifications can be done by using Design/ CPN and the test-case generation is also available to test the OO programs.

3) Validating OO programs satisfy their OO specifications

We apply our test-case generation technique to a CPN into which a UML specification is translated so that we could have a set of test cases. We execute OO programs by using the generated test cases and we systematically check if the behavior of the programs satisfies the test cases.

To achieve three levels of testing, our translation technique should be independent of programming languages and analysis/design methods. Thus we focus on the common concepts of OO and on how to translate them to a CPN. In the successive sections, we use a UML description as an example to explain how the OO concepts can be translated into a CPN. The technique shown in this example can be essentially applicable to the transition of even Java program swith out any extension.

(1) Testing in Spec. Level(prototyping) Simulation:



(2) Testing in Prog. Level: Debugging

(3) Validating OO programs satisfy their OO specifications

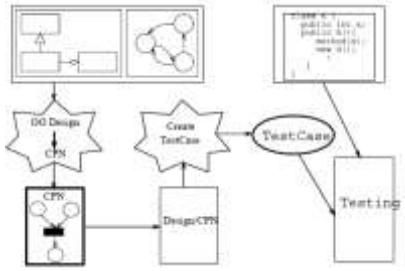


Fig 2: Relation between our approach and testing

III. TEST DESIGN

Test design tells how test specifications and test cases are created – inherently determines the success of testing. However, test design techniques are not always properly applied, leading to poor testing. Test design describes the phase in a process, where test specifications are written, and a resulting test procedure or test cases are created.

A test case is the result of applying a test (design) technique to a specific software system. The test technique delimits the type of test cases that can be created, according to a concept, approach or selection. The test case includes all needed information which is executable similar to the test procedure. It should be *repeatable* by anyone (yielding the same result) and measurable such that it should be possible to determine if the test passed or failed.

The test case template based on IEEE standard 829[13, 14] consist of:

- Test Case Specification Identifier;
- Test Items; *(references for traceability)*
- Input specifications & Output specifications
- Environmental needs;
- Special procedural requirements;
- Inter-case dependencies.

IV. CONCLUSION

1. The goal is to establish a test technique for the behavior of OO software and to make the technique independent of specific languages, specification and design methods using CPN. The software application is translated into a CPN and is analyzed, tested and simulated as a prototype.
2. The main objective is to use an improved technique for optimizing number of test cases by Colored Petri Nets (CPN).
3. Future work will focus on the improvement of generalization relationship between classes by using mathematical techniques. The relationship between classes includes association, aggregation and generalization.
4. Design, develop and experiment a improved technique/method that can be used to cover association and aggregation relationship based on the extended version of Petri Nets.
5. For analyzing the behavior and optimizing the test 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. SSG can be automatically generated and analyzed by using tools such as CPN-Tools.

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] EsmaeilMirzaeian, S. G Mojaveri, H. Motameni, A. farahi, "*An optimized approach to generate object oriented software test case by Colored Petri Net*", IEEE 2nd International Conference on Software Technology and Engineering(ICSTE) ,2010.

[12] 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.

[13] IEEE Std. for Software Test Documentation 829-1998 & 2008

[14] Beizer, B. *Software Testing Techniques*, Int. Thomson Computer Press, 2nd ed., Boston, 1990