5th SASTech
Iran |Mashhad
May 12 - 17 | 2011
5th Symposium on Advances in Science & Technology

An approach to generate software reliability testing data generation based on UML models

Bahareh mobasheri[1], Afshin Salajegheh [2]
College of Software engineering of Tehran Payam noor University[1]
B.Mobasheri@khi.ac.ir
5050100
Assistant Professor of Software engineering and Computer Science
Islamic Azad University Tehran South Branch[2]
Salajegheh@lau.ac.ir

Presenter: Bahareh Mobasheri

Abstract

To overcome the problems about testing data generation of the real-time control systems and reliability testing cases of softwares, in this study an approach based on the mixture of operation profile and Markov chain which generates reliability testing cases will be overviewed. This approach describes software operation profile using the use cases of UML and establishes the use model based on UML model for automatically deriving the testing model from the use model, and also generates a reliability testing case set based on the testing model. And by eliciting input and output variables and abstracting testing input and output classes, this technique generates testing input data of reliability testing semi-automatically.

As well as using integration to mix the operation profile and Markov chain to generate a set of reliability test cases, the integration can also be used to present a technique to mix the UML collaboration and statechart diagrams and enhance testing of interactions among modal classes. The result of this integration is an intermediate test model called SCOTEM(State COllaboration TEst Model). By using the SCOTEM and also some defined various coverage criteria, valid test paths can be generated. In order to assess the technique a tool is developed to investigate the fault detection capability in a selected case. The presented technique effectively detects all the seeded integration faults.

5thSASTech 2011, Khavaran Higher-education Institute, Mashhad, Iran. May 12-14.

Keywords: automated testing, data generation, Markov chain, Object-oriented systems, operation profile, reliability, software testing, UML based testing

information in software testing. Many UML design artifacts have been used in different ways to perform different kinds of testing. For instance, UML statecharts have been used to perform unit testing, and interaction diagrams (collaboration and sequence diagrams) have been used to test class interactions.

A complete system-level functionality (use case) is usually implemented through the interaction of objects. Typically, the complexity of an OO system lies in its object interactions, not within class Methods. When classes work independently, they work fine but faults can arise during integration.

In this paper a technique is presented that enhances the integration testing1 of classes by accounting for all possible states of collaborating classes in an interaction. This is important as interactions may trigger correct behavior for certain states and not for others. In order to achieve such an objective the proposed technique builds an intermediate test model called SCOTEM (State Collaboration TEst Model) from a UML collaboration diagram and statechart diagrams of the objects involved in the object interactions. The SCOTEM models all possible paths for object state transitions that a message sequence may trigger.

The test generator uses SCOTEM to generate test paths whose corresponding test cases aim at detecting the faults that may arise due to invalid object states during object interactions.

2. A mixture model of operation profile and Markov chain for testing cases generation

UML utilizes Use Case (UC) to describe the system functions, State Chart (SC) to simulate the dynamical operation condition of test cases, utilizes class chart to express the relationships of the class elements, utilizes sequential chart to express the sequences of mutual changing information between objects and etc. The modeling idea of the mixture model is using testing profile to define UC and using Markov chain to describe the dynamic characters of SC.

1. Introduction

Software testing is an important technique for software quality assurance. And software reliability testing is the key technology for improving and estimating software reliability which has the difficulty of generating test cases. There are mainly two types Γ of models for generating test cases at present, namely the generation model based on operational profile and Markov chain.

However, these above approaches did not definitely consider the dependency relationships of various inputs and various times of the same input, thus it is difficult for them to be applied into software modeling of real-time control system with synchronous and reactive characters. The test case generation approach based on Markov chain includes the following two modeling approaches, namely the state layered Markov chain and the Markov chain based on the probability state diagram. However, with the growing of software complex, the state space of the Markov chain difficult to model the complex software system. There, although these two approaches lay a foundation for software modeling and reliability test case generation, there are still problems left which are difficult to be solved.

The reliability testing data are sent to the testing system through the interfaces of simulation testing environment. For the correctness of software reliability testing, the testing data must reflect the actual inputs of testing system realistically as well as the inputs of airborne embedded software should be described normatively for generating testing data.

This study describes an approach which synthetically uses the mixture model, constituted by operation profile and Markov chain model and utilizes the class chart of UML to generate software reliability testing data by eliciting input and output variables and abstracting testing input and output classes.

In recent years, researchers have realized the potential of UML models as a source of

then $\forall t=(X,L,Y) \in S_{TranSet,}$we have $X,Y \subseteq S_{TranSet}$, $X \neq \Phi, Y \neq \Phi, L \varepsilon S_{LabSet}$. .
For transition $L \leq S_{LabSet}$ there is $L \leq E[c]/A>$, where, E is the trigger event, c is the monitor condition, A is the action. Only when c is true, E generates the action A, otherwise E does not generate actions. Transition L can be a simple transition or a subsequent transition. It may enter a combination state, or emerge from a combination state.

2.2. The output of operation model and testing model

Obtaining use chart from state chart: The key idea to convert the UML state chart into use chart is spreading state chart and eliminating the hierarchy of UML state chart. The hierarchy relationship of SC is mainly described by the OR state and AND state.

Definition 3: For one state s, its form can be a basic state, OR-state, r AND-state.
When s is a basic state:

$$\rho(s)=\phi$$

When s is an OR-state and $\rho(s) \neq \phi$, $\rho(s)$ is the OR decomposition of state s. When some object is in state s, actually it is in some child state of s.
When s is an AND-state and $\rho(s) \neq \phi$, $\rho(s)$ is the AND decomposition of state s. When some object is in state s, actually it is in all child states of s.

Definition 4: For one state s, if no special definition, the first child state s, in which entering first after entering the state s, will be called the default child state of s.

Definition 5: If $s_1 \in \rho^*(s)$, then we call s_1 the offspring of s and call s is the ancestor of s_1. If $s_1 \in \rho^*(s)$ and $s_1 \neq s$, then we call s_1 the strict offspring of s and call s is the strict ancestor of s_1.
The transitions from SC to UC can be achieved according to the following principles:

- Spreading SC by recursively introducing the subordinate state chart which has been replaced by the replace label at present
- Normalizing the trigger
- Replacing the auto-transition by the transition with ε
- There should be no unreachable or endless state in use chart. If there is such a state in user chart, it should be corrected

2.1. The construction of use cases and state chart

Constructing use cases

Definition 1: UC can be denoted by the following six tuple array:

$$C_U=<I_{UcID},C_{PreCond},S_{C,}R_{OtherReq},S_{PostCondSet},P_{UC}> \quad (1)$$

where, I_{UcID} is the ID of use case, $C_{PreCond}$ is the prepositive condition of the case operation, S_C is the state chart which is corresponding to UC, $R_{OtherReq}$ is the other requirements with regard to this case, such as the requirement of the execution time of case; $S_{PostCondSet}$ is the set of postpositive conditions which express the state in which system enters after UC has been successfully executed; p_{UC} is the probability of the case operation.
The construction of UC can be realized by the following steps, such as confirming the starter of operation, defining the operations mode, constructing the operation list, defining the occurrence probability, calculating the occurrence probability and etc.

Definition 2: The SC of UC can be denoted by the following nine-tuple array:

$$S_C=<S_{StateSet},S_{tranSet},\rho,S_{LabSet},S_0,S_{FinishSet},S_{EventSet},S_{CondSet},S_{actSet}> \quad (2)$$

where, $S_{StateSet}$ is the state set which is nonempty and finite; ρ: $S_{TranSet} \rightarrow 2^{StateSet}$ describes the dendriform hiberarchy relationship between the states; $S_{TranSet} \subseteq 2^{StateSet} \times S_{LabSet} \times 2^{StateSet}$ means the transition relationship ;S_{LabSet} means the set of transitionlabels; S_0 means the set of initial states; $S_{FinishSet}$ means the set of terminal states; $S_{EventSet}$ means the set of trigger events; $S_{CondSet}$ means the set of monitor conditions ;S_{ActSet} means the set of actions.
$\rho(s)$ defines the set of child states of states. $\rho^*(s)$ means the set of child states including the state itself. $\rho^+(s)$ means the set of child states excluding the state itself, namely, $\rho^+(s)=\rho^*(s)-\{s\}$.
For each state chart, there is a root state $r \in S_{StateSet}$. $s \in S_{StateSet}$, which means that there is no cycle in the hierarchy structure.
If X is given as the set of source states of transition, Y is given as the set of destination states of transition, L is given as the set of transition labels,

Input: Extended UC set and its corresponding SC and Markov chain usage model set.

Step 1: Initialization which includes: (1) Setting the total generation number N of TC (2) The TC label counter Count=1; (3) $C_T=\Phi$.
Step 2: Selecting a C_U randomly according to the occurrence probability
Step 3: Extracting the UC information and putting the UC information into TC according the following sequence, namely (1) constructing a t according to the definition of TC and initializing t; (2) t.TcId=Count; (3) t.PreCond=C_U.PreCond; (4) t.PostCond=C_U.PostCond; (5) the decision criteria are extracted by t.Rule according to C_U.OtherReq and C_U.PostCondSet
Step 4: The TC execution sequence is generated by the following steps, namely (1) selecting the corresponding Markov chain C_M of C_U (2) randomly generating a TC by C_M (3) selecting the corresponding state chart s of C and obtaining the accurate information of each operation step from s (4) testing sequence can be combined by the accurate information of usage sequences and steps; (5) equating test sequence with $S_{SequSet}$
Step 5: Putting TC into test case set $C_T=C_TU\{t\}$
Step 6: Judging whether the test cases is smaller than N, if the test cases is smaller than N, then Count=Count+1 and turns into step 2, else terminating the test cases generation

3. Reliability testing data generation approach for the airborne embedded software

The airborne embedded software belongs to the realtime embedded software. Therefore, constructing the simulation testing environment and automatically generating testing data are needed in reliability testing of the airborne embedded software. The following steps describe how to generate testing data by the synthetical analysis.

Analyzing software documents: The input and output information for testing can be elicited from software documents, such as system task document, software requirement specification, software interface requirement specification and so on. For example, the information such as format of testing data, source, destination, type and so on can

2.3. Transferring UC to usage model: The state transition has the probability character in the usage mode. For obtaining the usage model. The probability distribution of the software anticipative operation should be confirmed, namely the occurrence probability p(E) of the trigger event E and the true probability p(c) of the monitor condition c. The transition label L can be extended to L≤sE(p(E))[c(p(c))]/A> .
The transition probability can be calculated by the methods based on hypothesis, history data and estimation.

2.4. Expression of Markov chain of usage model
Definition 6: The usage model based on Markov chain can be denoted by the following five-tuple array:

$$C_M\leq S, \Gamma, \delta, q_0, F> \quad (3)$$

Where, S is the set of the execution states, Γ is the set of the transition labels, $\forall t,\ t \in \Gamma$, then t≤name,p> , where, Name is the name of transition, p is the transition probability, δ is a function, $\delta: S\times\Gamma\rightarrow S$, it can normalize the transition relationship between states of the operation process, q_0 is the initial state, namely the state in which software is before execution, F is the setoff terminal states in which software enters when software stops executing. The Markov chain usage model of this case can be generated by normalizing the definition 5 of the UC usage model.

2.5. The algorithm for generating test cases
Definition 7: Test Case(TC) set can be denoted by the following five-tuple array:

$$C_T\leq S_{TcIdSet}, S_{PreCondSet}, S_{SequSet}, S_{PostCondSet}, S_{RuleSet}> \quad (4)$$

where, $S_{TcldSet}$ is the orderly label set of TC, $S_{PreCondSet}$ is the preset execution condition set of TC, $S_{SequSet}$ is the execution step set of TC, $S_{PostCondSet}$ is the set of anticipative execution results of TC, $S_{RuleSet}$ is the criterion set for deciding whether TC is successfully executed or not.
If we have the extended UC set of software system and the corresponding SC and Markov chain usage model set, the reliability test cases can be generated by the following algorithm.

makes the input variables easy to be expressed and organized by the state chart of UML. The form of this constructing principle is similar to the first principle, is more suitable for some discrete input variables whose analog values and input values are not very correlative and makes the analog values which are coincident o time be encapsulated into the same input class.

Generating reliability testing data: Software reliability testing data can be divided into two forms, namely pre-generation and script-based generation.

The pre-generation testing data are deposited in the data-base or data document, automatically read by the testing system according to the testing requirement in testing and sent to the appointed interfaces. This testing data are static and we cannot dynamically change the input values of testing data according to the testing situation.

The script-based generation testing data are automatically generated by the testing script and can dynamically generate testing data and have higher flexibility.

The required testing data can be generated by synthetically utilizing the two above generation forms. It means that the relatively steady testing data cab use the pre-generation form and the testing data which change heavily cab use the script-based generation form. When using the testing scripts to generate testing data, the description of data class is used as the foundation of data generation.

4. UML based test generation

4.1. Several techniques based on UML

4.1.1. Unit level testing

Unit level testing refers to the tests performed to check the correct functionality of individual execution units. In OO programs a class is considered the smallest unit. Unit testing forms the basis for the subsequent higher level testing activities as it establishes the minimal operability of units that will be integrated to produce the actual functionality. A technique is introduced in[3)] in which a statechart is derived from the use cases and is used to generate a usage model.

4.1.2. Integration level testing

be obtained from software interface requirement specification.

Eliciting input and output variables: The transmission data in each interface are abstracted as the input or output logistic variables for obtaining the input variables of software testing. Then these input or output logistic variables are tailored and transmitted for obtaining the final input and output variables.

Because the system testing of the airborne embedded software is generally realized by the simulation testing environment and many inputs of the testing system can be generated by the simulation testing environment, we should to provide the inputs for all of the input variables. Therefore, we should regard the testing system and simulation testing environment as a whole for arranging the conformed input variables.

Abstracting input class: The input class can be defined as the set of the external input data types of the airborne embedded software system and be described by the state chart of UML.

There are two constructing principles of the input class: one is constructing the input class according to the external interface relationships of the airborne embedded software system; the other is constructing the input class according to the relationship of temporal logic.

- Constructing the input classes according to the external interface relationships

Constructing the input classes according to the external interface relationships can make the interface relationships of the testing system be corresponding to the input classes. The process of abstraction and construction is intuitive as well as the testing data can be generated only by organizing the data members of input classes into the requirement format.

- Constructing the input classes according to the relationship of temporal logic

Constructing the input classes according to the relationships of temporal logic uses the object-oriented method for the further encapsulation of the input variables from the viewpoint of the relationship between input time and logic and

generating a graph-based test model (SCOTEM) and by covering all paths in the model. The proposed technique can be applied during the integration test phase, right after the completion of class testing. It consists of the following four steps:

1. SCOTEM generation: an intermediate test model, called SCOTEM (State Collaboration TEstModel) is constructed from a UML collaboration diagram, and the corresponding statecharts.
2. Test paths generation: test paths are generated from the SCOTEM based on several possible alternative coverage criteria.
3. Test execution: all selected test paths are executed by using manually generated test data and an execution log is created, which records object states before and after execution of each message in a test path.
4. Result evaluation: the object states in the execution log are compared with the expected object states in the test paths generated from SCOTEM. If any state of any object after execution of a test path is not in the required resultant state, ten the corresponding test case is considered to have failed.

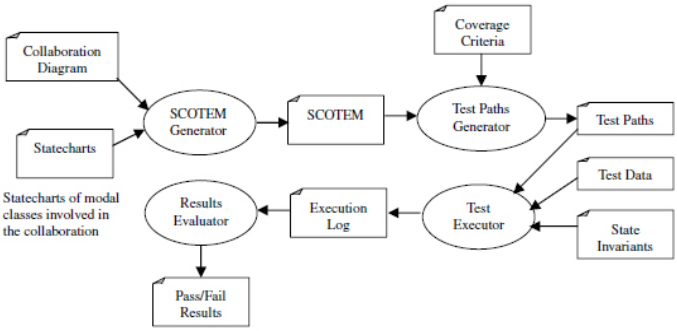

Fig. 1. Flowchart for the proposed testing technique

5.1. Defining the SCOTEM test model

The SCOTEM is an intermediate test model used to automatically generate test specifications for class integration testing. In this sub-section, we describe how this test model is constructed from a given UML collaboration diagram, and its corresponding statecharts. However some assumptions are made about these UML models:

- Collaboration diagrams may contain synchronous call messages or asynchronous signal messages, and statecharts their corresponding call events.

The integration testing of OO software is concerned with testing the interactions between units. The focus at this level is on testing interactions between classes through method calls or asynchronous signals, as well as interactions with databases or hardware. TEst Sequence generaTOR (TESTOR) is an approach used to generate integration tests using UML statecharts and collaboration diagrams for component based systems[5].
System-level testing
An approach is to generate system-level test cases. This approach uses use case diagrams, activity diagrams, and sequence diagrams to generate system-level test cases.[4]

4.1.3. Conformance level testing

Testing software in order to establish the fulfillment of the specified requirements is known as conformance testing. UML statecharts are used for this purpose. UML statecharts are converted in to extended statecharts and properties of Communicating Sequential Processes (CSP) are added. This model is given as an input to the Test Development Environment (TDE), which generates unit, integration, and system-level conformance tests for the system.
As you can see none of the above works discuss testing by integrating UML interaction and statechart diagrams to uncover state-dependent, class interaction faults. The proposed approach uses UML statecharts and collaboration diagrams to generate an intermediate model, SCOTEM, and applies different coverage criteria based on the SCOTEM graph representation.

5. Generating the intermediate model,SCOTEM

The run-time behavior of an object-oriented system is modeled by well-defined sequences of messages passed among collaborating objects. In the contest of UML, this is usually modeled as interaction diagrams (collaboration or sequence). In many cases, the states of the objects sending and receiving a message at the time of message passing strongly influence their behavior.
In [2] an integration testing technique is proposed that is based on the idea that the interactions between objects should ideally be exercised for all possible states of the objects involved. This is of particular importance in the context of OO software as many classes exhibit a state-dependent behavior. Such testing objective is implemented by

5.2. Generating test paths from the SCOTEM

A test path derived from the SCOTEM represents a path that start with the initial (null) vertex and contains a complete message sequence of the collaboration. The total number of test paths in a SCOTEM can be determined by taking a product of the numbers of transition paths in each modal class, where each transition path is an internal transition of a modal class from a source state to a target state on receipt of a particular message[2].

5.3. Coverage criteria for test paths

In a more complex system, the number of modal classes involved in a collaboration and the number of states of such classes can be large. This typically results in an exponential growth may be impractical, or even impossible, to test all the paths due to the coast involved. The allow for an acceptable level of testing while keeping the coast at the minimum, we a various coverage criteria based on SCOTEM can be defined. These coverage criteria include: Single-path Coverage, All-Transition Coverage, n-Path Coverage, Loop Coverage, Predict Coverage.

6. Automation

A prototype tool is developed to validate the approach. The prototype tool automatically constructs a SCOTEM model from a given UML collaboration diagram, and corresponding statecharts (for modal classes). It uses the SCOTEM to generate test paths according to the specified coverage criterion and then, it executes the test paths (using manually generated test data), evaluates the execution results, and creates a log. The tool is composed of four major modules, namely SCOTEM Constructor, Test Path Generator, Test Executor, and Results Evaluator.

7. Conclusion

The software reliability testing data generation approach proposed in this paper solves the problems that the traditional approaches based on operation profile are difficult to describe the synchronous and reactive characters of real-time control system and is suitable for reliability testing data generation of the embedded real-time control software.

The approach mentioned in this paper can be used for generating the reliability testing data of real-time control system automatically. We

- Statecharts are in a flattened form. The flattening process can be automated.
- State invariants of modal classes that are relevant to the selected collaboration diagram must be specified. It is, however, common practice to write state invariants during the definition of statecharts. Though it requires extra effort, the specification of state invariants is important to give a precise meaning to each possible state of a class.
- The algorithm to generate the SCOTEM test model depends, to some extent, on the version of UML used as a design notation.

The SCOTEM is a specific graph structure: a vertex corresponds to an instance of a class (in a particular state) participating in the collaboration.

A modal class can receive a message in more than one state and exhibit distinct behavior for the same message in different states. To capture this characteristic, for modal classes, the SCOTEM contains multiple vertices, where each vertex corresponds to an instance of the class in a distinct abstract state (corresponding to states defined in statecharts). On the other hand, a non-modal class only requires a single vertex in the SCOTEM graph.

The edges in the SCOTEM test model are of two types: message and transition edges. A message edge represents a call action between two objects, and a transition edge represents a state transition of an object receiving a message. Each message edge may also contain a condition or iteration. Each message may cause a state transition to occur. A transition edge connects two vertices of the same class. Statecharts may have multiple transitions to distinct states for the same operation. Hence, there may be multiple transition edges(representing a conditional state transition) for the same message edge in SCOTEM. Each of these transitions is generally controlled by mutually exclusive conditions (to prevent non-determinism). The internal representation of a vertex holds the class name and state of the instance it corresponds to. Message edges are modeled in the sequence number, associated operation, receiver object, by the attributes of a transition including sequence number, associated operation, accepting state, and sending state.

formalize the reliability testing process by normalizing the software reliability testing data generation approach based on mixture model which was proposed in this paper for developing the corresponding supplementary tools and then achieve the automatic reliability testing of real-time control software.

The reliability testing data generation approach presented in this paper focuses on describing the elements of software dynamical action to solve the problems that the state space of Markov chain may increase by the exponential form. This, this approach decreases the number of testing data greatly as well as realizes the automatic reliability testing, which makes the testing efficiency improved.

The strategy introduced for class integration testing that is based on a test model (SCOTEM) that combines information from collaboration and statechart diagrams into the form of a graph. The motivation is to exercise class interactions in the context of multiple state combinations in order to detect state faults. Therefore, it takes into account the states of all objects involved in a collaboration to exercise class interactions in the context of integration testing. For instance, if the functionality provided by an object depends o the states of other objects (including the caller object), then the proposed technique can effectively detect faults occurring due to invalid object states.

References

[1] Wang Xin, han Feng-Yan and Qin Zheng,(2010). *Software Reliability Testing Data Generation Approach Based on a Mixture Model*, Journal 9.,pp.1038-1043

[2] Shaukat Ali, Lionel C. Briand, Mohammad Jaffar-ur Rehman, Hajra Asghar, Muhammad Zohaib Z. Iqbal, Aamer Nadeem,(2007).*A state-based approach to integration testing based on UML models*, Information and Software Technology 49, pp.1087-1106

[3] T.H. Tse, Z. Xu,(1995). *Class-level Object-Oriented State Testing:A Formal Approach*, HKU CSIS Technical Report ,1995

[4] L.Briand, Y. Labiche,(2005). *A UML-based approach to system testing*, in:Proceedings of the Fourth International Conference on the Unified modeling Language, pp.194-208

[5] P. pelliccion, H. Muccini,A.Bucchiarone, F. Faccini,TeStor,(2005). *deriving test sequences from model-based specification*,pp.267-282