Research on On-line Testing of Real-time Software Based on Time Petri Network Model

Rao Tingting[1], Xiong Caiquan[2], Wang Mingli[3]

[1]Hubei University of Technology, Wuhan, China

[2]Hubei University of Technology, Wuhan, China

[3]Huazhong University of Science and Technology, Wuhan, China

Keywords: Time Petri net; Test; Top-level design; Dispatch; Parallel

Abstract: Parallel test task scheduling scheme has been an unsolved problem in automatic test system. The time Petri net model for parallel testing is established based on the theory of Petri net, the time Petri net-based parallel testing task modeling method is proposed, which combines the resource conflict problem caused by multi-task parallel scheduling in UML parallel testing system; The parallel task scheduling sequence with the shortest time required to complete all test tasks is obtained comparing the completion time of different sequences; Finally, an example is simulated and analyzed under this model; The experimental results show that the model is suitable for describing the task scheduling process of this type of system.

1. Introduction

The frequency of software product upgrading is increasing with the rapid development of the software industry, which makes the cost of regression testing higher and higher. It often takes days or even months to complete centralized testing by fully running test cases in large-scale software testing. Testers want to prioritize test cases according to certain criteria In this case, it enables high priority use cases to be executed as early as possible, thus improving test efficiency. The test case priority technology is proposed to solve this problem, the technology points out: Different test cases have different contributions to the completion of test objectives, it is necessary to compare and rank different test cases according to the historical information of the test to achieve the test objectives more quickly, this prioritizes the execution of relatively important test cases. The generation of interlocking test cases mainly includes: Document [3] generates test cases from specifications of relational algebraic query representations. Document [4] adopts the test data generation method based on Boolean specification. In addition, test case generation based on Z language is mature. But these test cases are difficult to understand and use because of their high formalization and poor versatility. The test case of interlocking software is designed for the standard station manually by experienced debugging engineers in China, it has a strong dependence on testers' professional knowledge, which affects the adequacy and efficiency of testing. The third-party testing generally adopts black-box testing because of the complexity of security software and the fairness of interlocking software testing.

2. Time Petri Net Detection Model

Petri net is a directed weighted bipartite graph composed of two kinds of nodes, including Place, Transition, Connection and Token, as shown in Figure 1.

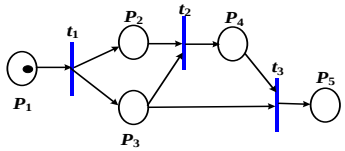

Figure 1 Petri net model

The constrained transition enabling process based on forbidden arc is adopted. The input form of the set of libraries of constrained transition t is $P_{in}=\{p_i \mid p_i \in {}^*t\}$, and $P_1=\{p_k \mid p_k \in {}^*t\}$, $P_2=\{p_j \mid p_j \in {}^*t\}$ are assumed. Among them, P_1 is set by the input library, and it does not contain forbidden arcs. Correspondingly, P_2 is set by the input library, and it contains forbidden arcs. It can be obtained based on the above definition: $P_{in}=P_1 \cup P_2$。$w_{b(p_j,t)}$ denotes the prohibited arc weight between repository p_j and constraint transition t. $b(p_j,t)$ denotes the forbidden arc between repository p_j and constraint transition t. The weight of forward connection is larger than that of forward connection if for any $m(p_i)$, and the weight of prohibited arc connection with constrained transition t is larger than that of identification number p_j, then the constrained transition state t is in the state of transition enablement, otherwise the constrained transition state t is in the state of transition disability.

$$E(t)=\begin{cases}1,\left(\forall p_i : m(p_i) \geq w_{pre(p,t)}\right) \wedge \left(\forall p_j : m(p_j) < w_{b(p_j,t)}\right)\\ 0,\exists p_j : m(p_j) \geq w_{b(p_j,t)}\end{cases} \quad (1)$$

Taking model constraint 3 as an example, the controller is designed based on the above-mentioned prohibition arc strategy. The specific process is as follows::

Process 1: Change t_2 has the highest priority because change assignment $pr(t_2)$ is the largest, then the other transition processes t_3, t_6, t_{12}, t_{13} etc. are in the unavailable state when transition t_2 is in the enabling state, The enabling conditions for the above changes are $m(p_1) \geq 1$. Therefore, the enabling state of transition process t_3, t_6, t_{12}, t_{13} can be effectively controlled based on $m(p_1)$ value, and the prohibited arc design can be carried out.

Process 2: The prohibited arc is designed based on the design principle of process 1, the specific process is shown in the network connection diagram of Figure 2, and the graph is a prohibited arc design form for model constraint 3.

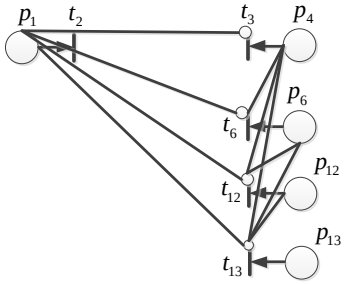

Figure 2 Prohibits arc constraint addition

3. Test Case Based on time Petri Net Detection Graph

The test case generation process based on sequence diagram is shown in figure 3.

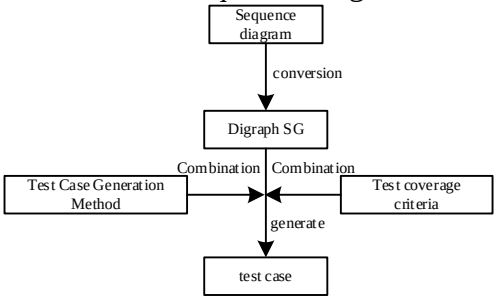

Figure 3 Test case generation process based on sequence diagram

The test case generation steps based on sequence diagram are as follows:

(1) Add constraints to the sequence diagram.

(2) All scenarios of the sequence diagram are obtained by traversing all the valid event sequences in the sequence diagram, and the corresponding directed graph SG is generated.

(3) Define test coverage criteria and test case generation methods to generate test cases.

The method of generating directed graph SG from sequence graph

Definition 1 Digraph SG=<NSG,∑SG, q0SG,FSG >, among:

NSG={N1,N2,…,Nm} is a collection of message nodes, Nj=<message Name, from Object, to Object,[c()]>, Nj is a node based on message m, from Object is the sending object of the message and to Object is the receiving object of the message. c() is the condition for monitoring messages; ∑SG is a set of transformed edges between nodes; q0SG is the starting node and the only entry node of directed graph SG. FSG is the terminal set of directed graph SG.

Definition 2 Test scenario TS: Test scenarios for sequence diagrams TS={TS1,TS2,…,TSn} represents the scenario path in the sequence diagram, including normal and abnormal scenarios, among:

TSi:<ScnId, StartState, NodeSet, NextState>, among, ScnIdrepresents the unique identification number of the test scenario, StartState represents the initial state of the system before TSi is executed, NodeSet represents the set of message nodes that occur in the test scenario TSi, NextState represents the state of the system after TSi is executed.

The steps for converting UML sequence diagram to directed graph SG are as follows:

(1) Initialization: NSG=NULL;

(2) corresponding message node N and transformation edge ∑ between nodes are generated Combining the sequence and constraints of message sending in sequence graph;

(3) The trigger information of the node with 0 entrance is recorded as q0. The post-information of the node with a degree of 0 is recorded as F.

A directed graph SDG can only correspond to one starting point, but there may be one or more endpoints for different test scenarios.

Test coverage criteria

The test coverage standard is a measure of the adequacy of software testing and a proof of the validity and reliability of test results, any test should meet a certain coverage. This paper uses the node coverage criteria, transfer edge coverage criteria and full path coverage criteria starting from the possible operational errors, message interaction errors and scene errors among objects, it traverses the directed graph SG to obtain the corresponding test case set.

Criterion 1: Node Coverage Criterion: All nodes on the SG have been visited at least once.

Criterion 2: Transfer Edge Coverage Criterion: All transfer edges on SG have been executed at least once.

Criterion 3: Logical Path Coverage Criterion: All paths on SG have been executed at least once.

Aiming at the hidden operation errors in system design, it is necessary to traverse the corresponding SG in the sequence diagram to generate test cases that meet the test coverage criteria, it gets all the paths from the initial node to each termination node, accesses each path to obtain the corresponding test scenario.

The constraints contained by each node are recorded when traversing a digraph SG, including the state information of the object, the branching conditions and the input and final output of the scene, each constraint is connected by logic and (&).Test cases are output results of preconditions, input events, event constraints, and expected for any scenario. This paper takes the interlocking route establishment process as an example to generate test cases, it satisfies the conditional requirements of interlocking test cases based on coverage criteria of nodes, transfer edges and logical paths. However, it is the focus of future research to optimize the generation of interlocking software test cases by considering the dependencies among the use cases because only a single test case is considered.

4. Conclusion

At present, the main method of generating software test data is manual and semi-manual, which

has large workload, long test cycle and is easy to be omitted; the test case generation method based on Petri net is mainly applied to scenario class testing. The test model is modeled by Petri net theory, then the model is validated, and finally the test case set is generated; This method supports regression generation of test case sets, making up for changes in test case set requirements; At the same time, it supports the automatic generation of test cases; This method can effectively describe the state behavior of the system and guarantee a high coverage index, it helps to improve the efficiency and quality of testing.

References

[1] Denaro G, Pezzè M. Petri Nets and Software Engineering [J]. Lecture Notes in Computer Science, 2004, 29(548):439-466.

[2] Lara J D, Guerra E. From types to type requirements: genericity for model-driven engineering [J]. Software & Systems Modeling, 2013, 12(3):453-474.

[3] Kamsu-Foguem B, Noyes D. Graph-based reasoning in collaborative knowledge management for industrial maintenance [J]. Computers in Industry, 2013, 64(8):998-1013.

[4] AnaPaulaEstrada-Vargas, ErnestoLópez-Mellado, Jean-JacquesLesage. Input–output identification of controlled discrete manufacturing systems [J]. International Journal of Systems Science, 2014, 45(3):456-471.

[5] Krichen M. A formal framework for black-box conformance testing of distributed real-time systems.[J]. 2012.

[6] Naija M, Ahmed S B, Bruel J M. New schedulability analysis for real-time systems based on MDE and Petri Nets model at early design stages[C]// International Joint Conference on Software Technologies. 2016.