ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 224-232
doi: 10.18178/wcse.2017.06.039

Automatic Integrated Test of Phased Mission Systems Oriented to Earthquake Response

Jiangong Song[+], Qinyong Li, Jianghua Lv and Shilong Ma

Beihang University, Beijing, China

Abstract. Since violent earthquakes occur frequently at home and abroad, for a class of mission systems in earthquake response and process, called phased mission systems, we propose an automatic integrated test method based on model checking to satisfy the need of complexity and trustworthiness required by phased mission systems. We build an automatic test system and test environment, and give a model of phased mission systems by system windows tree model, and also give a theoretical state diagram model. Based on model checking, an automatic test method for phased mission system is proposed, and an evaluation algorithm for systems trustworthiness is applied in a case study of the System of International Earthquake Response.

Keywords: formal methods, model checking, phased mission system, earthquake response and process, integrated test method

1. Introduction

In recent years, earthquake disaster frequently occurs, that according to the official website of China Earthquake Administration statistics, from January 1, 2007 to May 31, 2016 in the world more than magnitude-7 earthquake occurred 197[1], of which most earthquakes caused great losses to human life and property. Therefore, it is very important in theory and practice to study the emergency response of the disaster. Earthquake emergency response system is a complex time-constrained phased mission system(PMS). Any errors in the system will cause serious or even catastrophic consequences. Hence, software trustworthiness is that the behaviors and results of software systems can be predictable, states can be monitored, results can be assessed, and exceptions can be controllable [2,3].

In recent years, Scholars have studied various ways to improve the trustworthiness of software. Such as modelling of the testability requirements analysis [4], reliability modelling of PMS by fault tree [5], multivalued decision diagrams[6], self-trust model based on semi-Markov performance evaluation[7],etc. Automatic testing is an important guarantee for the trustworthiness of such systems. Judging the software features suitable for automatic test [8], and the cost of software development was studied by Siemens and Saab[9]. The formal analysis methods of security critical systems such as phased mission systems include formal modelling and formal verification. Finite automata is a widely used formal modelling method[10];The formal verification method of model checking in system state space search based on the nature of the final can prove that expectations are met [11]. Many researchers try to reduce time complexity of computation by studying on all kind of methods to reduce the state space[12-22]. From the above research, we can see that the previous methods are more concerned about the design of the system itself and the correctness of the analysis.

This paper discusses the formal method of earthquake emergency response phased mission system testing. Based on the definition of window tree model of the system and system state transition diagram

[+] Corresponding author. Tel.: +86-010- 82317643; fax: +86-010- 82317598.
E-mail address: sjg@buaa.edu.cn

model of system, describes the behaviour correctness of system under testing. Put forward the method of automatic generation of test cases and model checking algorithm based on the two kinds of models, and proposed the evaluation system of trustworthiness.

2. Formal Modeling of Phased Mission System

2.1. System Window Tree Modeling

The system window tree model is on the actual function and operation behavior of the system, the execution of mission system can be regarded as consisting of a large number of windows, from a window to another window is generally by one or more events. Because of the difference of parameters, the different API will be called, and different branches will be formed. We call it the system window tree, as shown in Figure 1[23], the top window of the graph consists of 3 different API, which can be entered into different windows by calling different APIs.

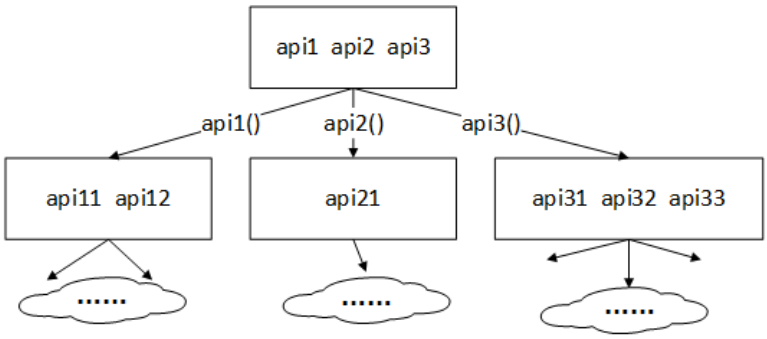

Fig. 1: Typical windows tree[23]

Definition 1: window. *window*=(w_{name}, W_s, *R*, *S*) , where,

w_{name} is the name of the window; W_s is subsequent windows set of the window; *R* is API called for reaching the window; *S* is all possible state set for reaching the window.

Definition2: System Window Tree(SWT). $SWT=\{(<w_i,w_j>,e_{i,j}) \mid w_i,w_j \in W, e_{i,j} \in E\}$,where,

$(<w_i,w_j>,e_{i,j})$ is the transition between windows, and w_i transits to w_j by $e_{i,j}$, and $0 \leqslant i, j \leqslant n$, the number *n* is total window. If w_i=*null* and $e_{i,j}$=*null*, it is root of the tree, or w_j=*null* and $e_{i,j}$=*null*, it is leaf of the tree; *W* is the set of windows; *E* is the set of events, that is the set of API.

Definition 3: Phased Mission System(PMS).

PMS=<*OnDuty*, *MissionCaptured*, *MissionReponse*, *MissionDeal*> ,where,

OnDuty is the phase of On-duty; *MissionCaptured* is the phase of mission object captured; *MissionResponse* is the phase of response; *MissionDeal* is the phase of analysis and process.

The PMS defined in this paper is a kind of phased mission system oriented to disaster emergency response, which is divided into 4 typical phases. The 4 phases are sequential.

2.2. System State Transition Diagram Modeling

The mission system under test is divided into phases, the behaviour subject of the system is defined, and the behaviour subject is modelled, and the state set of the behaviour subject is determined by the state set of the all elements of a given system. Mission is the behavor subject of phased mission system.

Definition 4: mission. *mission*=(*flow*, *data*, *webpage*), where,

mission is a special mission; *flow* is a special data flow path; *data* is a special result of professional model calculation from a form; *webpage* is information processed from Internet.

Definition 5: mission_state. $mission\_state = (flow\_state, data\_state, webpage\_state)$

where, *mission_state* is the state of a special mission; *flow_state* is the state of a special data flow path; *data_state* is the state of a data form; *webpage_state* is the state of an information from Internet. The mission state is decided by the state of each tuple.

Definition 6: Flow_State_Set. $Flow\_State\_Set = \{ flow\_state_i \mid i = 1..m \}(m \geq 0)$, $flow\_state_i$ is ith state of a flow path.

Definition 7: Data_State_Set. $Data\_State\_Set = \{data\_state_i \mid i = 1..n\}(n \geq 0)$, $data\_state_i$ is ith state of a data form.

Definition 8: Webpage_State_Set.

$Webpage\_State\_Set = \{webpage\_state_i \mid i = 1..p\}(p \geq 0)$, $webpage\_state_i$ is ith state of information from Internet.

Definition 9:Mission_State_Set.

$Mission\_State\_Set = \{(flow\_state_i, report\_state_j, webpage\_state_k \mid 1..m, j = 1..n, k = 1..p)\}$

$\subseteq Flow\_State\_Set \times Data\_State\_Set \times Webpage\_State\_Set$,

$Flow\_State\_Set \times Data\_State\_set \times Webpage\_State\_Set$ is the full set of system. For any particular phase mission system, there may be redundant or inconsistent with the design of the system, MISSION_STATE_SET is the sub-set of the full set.

Definition 10: API_Set. $API\_Set = \{API_i \mid i = 1..m\}$, it is API set of system.

The state transition of the phased mission system is due to the running of the application access interface (API), and the state transition can be represented by three tuple.

Definition 11: state_transition. $state\_transition = (mission\_state_i, mission\_state_j, \text{call } API_i)$,

and $mission\_state_i, mission\_state_j \in MISSION\_STATE\_SET$; $API_i \in API\_SET$ .The system state is transited from $mission\_state_i$ to $mission\_state_j$ by calling API_i once.

Definition 12: System State Transition Diagram(SSTD). $SSTD = \{state\_transition_k \mid k = 1..p\}$,

$state\_transition_k$ is the system state transition once, and p is sum of all transitions.

3. Model Checking method to Phased Mission System

3.1. Test Coverage Based on the Path

In the field of software testing, test coverage is used to describe the extent to which the system is tested. The commonly used coverage criteria are: functional coverage, statement coverage, branch coverage, and condition coverage. In this paper, test coverage is based on the path and the test case set is generated by the window tree model, each path on the window tree represents a process of the actual operation of the system.

The test case is defined as a path from the root node of the window to the leaf node. For a window tree, all test cases belong to the test case set.

3.2. Automatic Test Case Generation

Window tree model is a tree structure, by the tree traversal automatically generate test cases, the API sequence from the root node to the leaf node traversal path is saved as a test case. The algorithm of automatically generating test cases by the window tree is shown in algorithm 1

Algorithm 1. The algorithm of automatically generating test cases

```
Input: Node: root.
Output: Set:set.  //set is test case set
Read XMLfile
Wirte tree_map
Set current_rode to root
Empty temp_api_list
Call Ergodic(current_root)
function Ergodic(root)
    if root != null then
            api ← root.api
            temp_api_list.push(api)
            foreach item ← root.sons
                    Ergodic(tree_map[item])
            endforeach
```

```
            if root.sons.size == 0 then
                test_cases.add(temp_api_list)
            endif
            temp_api_list.pop()
        endif
    endfunction
```

3.3. Model Checking Method

The model checking algorithm is shown in algorithm 2. First, a test case is read, and then the current API is executed in the system under test. After each execution, the state transition of the system under test is checked to meet the finite state machine. The test cases run successfully if the entire APIs called completely in the test case meet the state transition rules. Otherwise, the system displays error message, that indicates the test case failed to run.

Algorithm 2. Model checking algorithm

```
Input: Test_Case: test_case.
Output: Bollean: last_state.  //last_state is test result: pass or error
api ← now_message.params[0]
state ← now_message.params[1]
b_find ← false
b_state_same ← false
b_api_same ← false
vertex ← graph[last_state_id]
foreach item ← vertex.edges
    id = item.id
    if graph[id].state == state then
        b_state_same ← true
    endif
    if api == item.api then
        b_api_same ← true
    endif
    if b_state_same and b_api_same then
        b_find = true
        last_state_id ← state.id
        last_state ← state
        api_run_statistics[api].correct_time++
    endif
endforeach
if !b_find then
        last_state_id ← state.id
        last_state ← state
        api_run_statistics[api].fault_time++
endif
```

3.4. Trustworthiness evaluation model based on model checking

The quantitative evaluation of the model test results can directly reflect the trustworthiness of the system under test. But for the trustworthiness evaluation of a system is not simply evaluated by a single variable, because the trustworthiness of the system is often affected by many factors, and the weights of these factors are different. In this paper, the trustworthiness evaluation based on model checking is evaluated according to the transition of finite state machine.

A test case set contains many test cases, and each test case contains several different APIs. Different important degree of API in the system decides different weight of API. When any API is executed, the finite state machine checks whether the state transition is expected.

As shown in Figure 2, for the state of S_i in the diagram, when the API is called, the state transition of the system under test is not consistent with the theoretical transition, then the API is recorded as error once, otherwise the API is recorded as correct.

After the test case set *TS* is run, each *API* in the system is calculated with the correct operation times G_i and the number of errors as B_i.

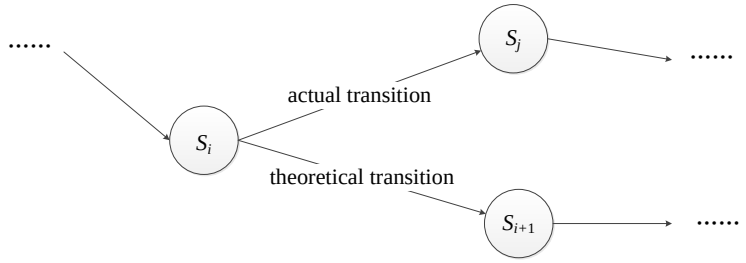

Fig. 2: State transition

Therefore, the trustworthiness of the system under system is shown in Formula (1).

$$R_{es} = \frac{\sum_{i=1}^{K}[G_i \times M_i]}{\sum_{i=1}^{K}[(G_i + B_i) \times M_i]} \tag{1}$$

where, R_{es} is the result of system trustworthiness,$0 \leqslant R_{es} \leqslant 1$; G_i is the correct times of *API* called; B_i is the error times of *API* called; M_i is the weight of *i*th *API*, $M_i \geqslant 0$ and $\sum_{i=1}^{K} M_i = 1$.

4. Application

4.1. Automatic test framework and system implementation

In this paper, we design a framework for automatic testing based on model checking, as shown in Figure 3, which has already applied to test the System of International Earthquake Response, a national project held by National Earthquake Response Support Service (NERSS).

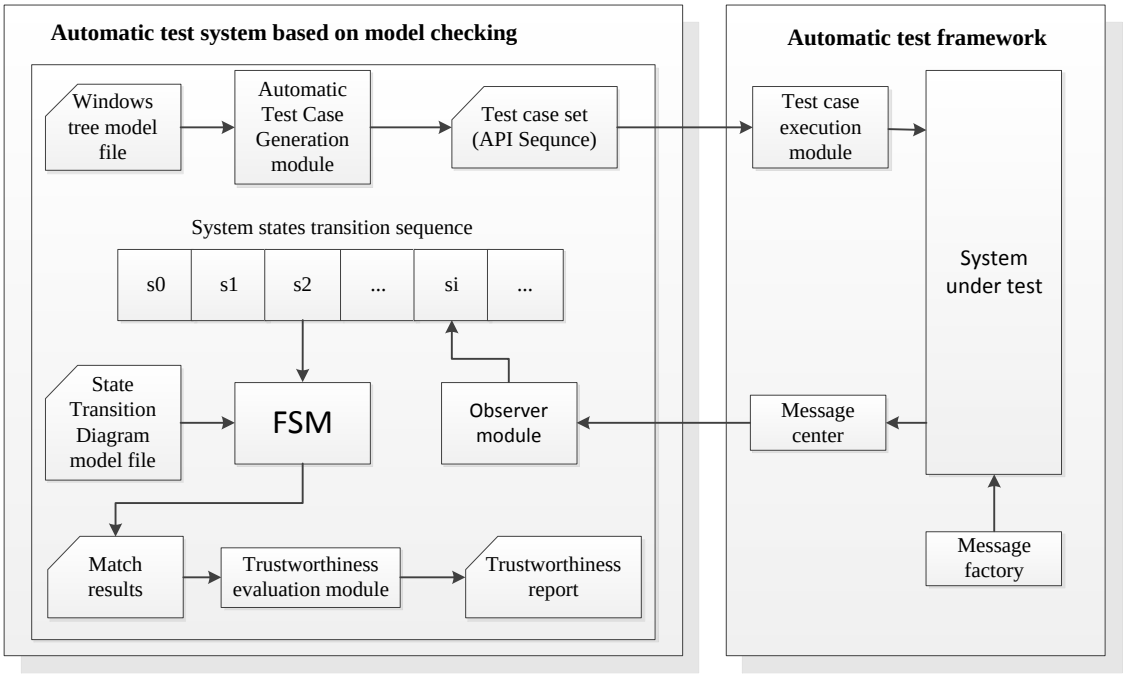

Fig. 3: Automatic test architecture based on model checking

As shown in Figure 3, according to the state of the system under test, testing framework and testing system of negotiated data format, the state and other relevant information (such as time, type) encapsulated into a packet, the packet is sent to the automatic test system via network. In Figure 4, the SUT With TestFrame is composed of the system under test and the testing framework, and the rest of the class forms an automatic test system based on model checking. Automatic Test Case Generation consists of TestCaseMaking, TreeNode and States. The model checking component is composed of ObserverTree, GraphVertex and States.

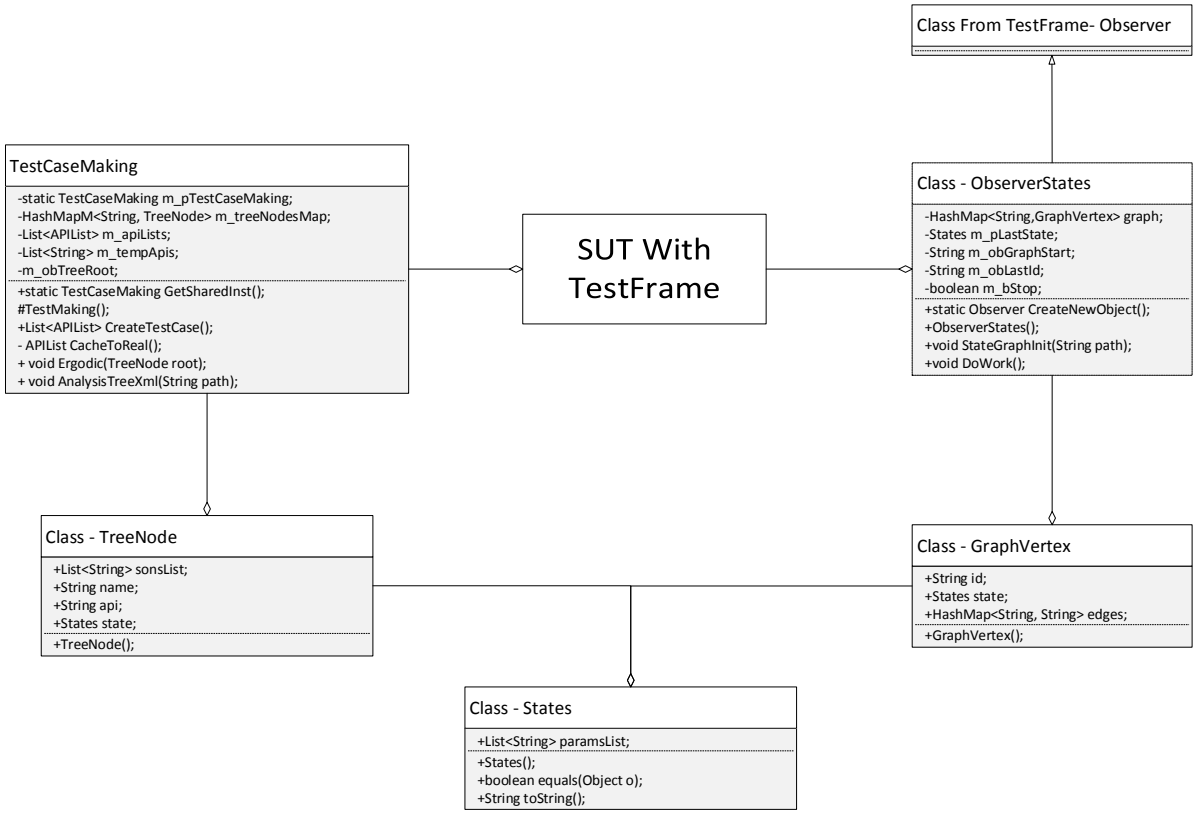

Fig. 4: Automatic test framework based on model checking

4.2. The test results and trustworthiness

Before testing and evaluating the trustworthiness of the system under test, the formal model is constructed, and the window tree model and the state transition graph model are constructed.

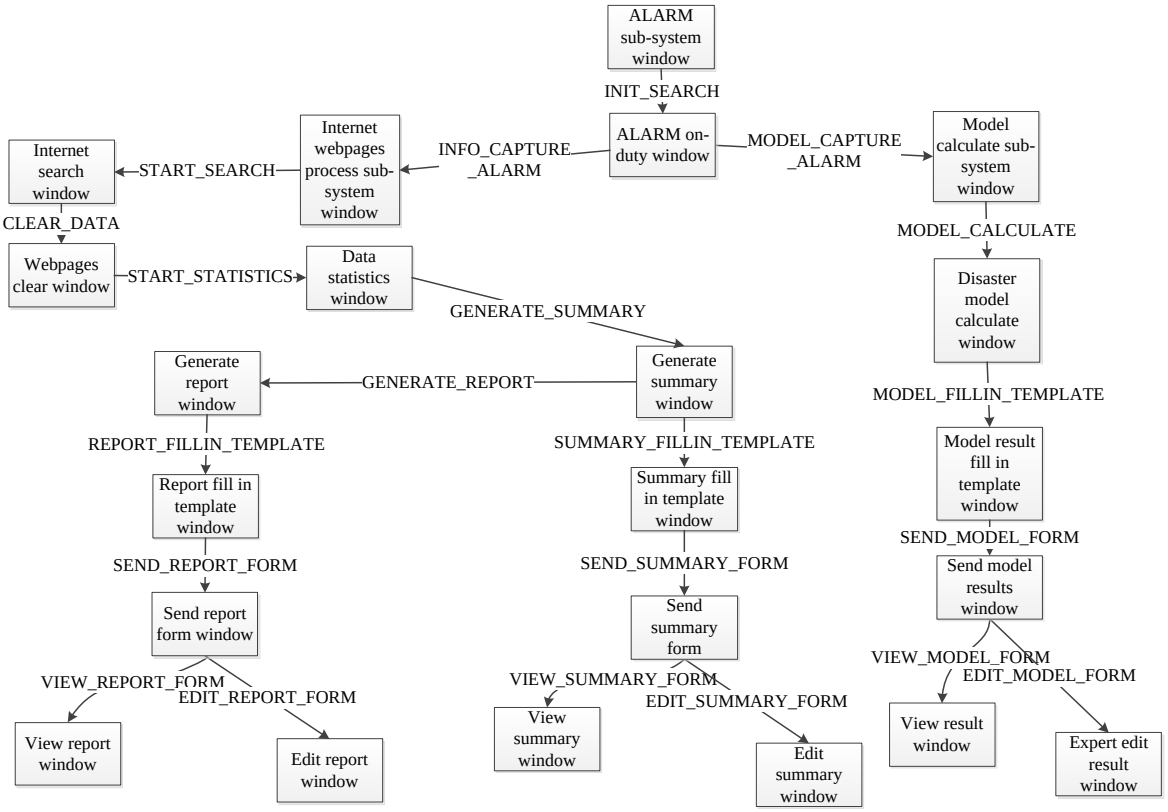

Fig. 5: The windows tree of the System of International Earthquake Response

The System of International Earthquake Response window tree model, as shown in Figure 5, has a total of 6 paths from the root node to the leaf node, which will generate a test case set containing the 6 test cases.

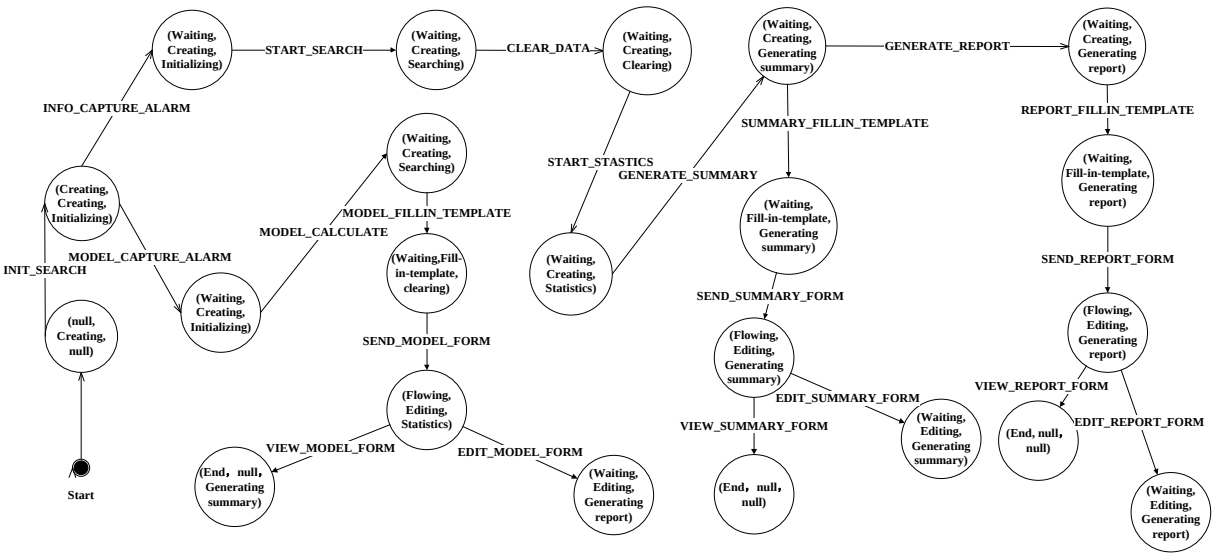

Fig. 6: System of International Earthquake Response state transition diagram

The state transition diagram of the System of International Earthquake Response is shown in Figure 6, which is the theoretical state transition of the system.

```
---------- 测试用例列表 ----------
测试用例0: null -> CAPTURE_ALARM -> INIT_SEARCH -> START_SEARCH ->
          CLEAR_DATA -> START_STASTICS -> GENERATE_SUMMARY -> GENERATE_REPORT ->
          REPORT_FILLIN_TEMPLATE -> SEND_REPORT_FORM -> VIEW_REPORT_FORM
测试用例1: null -> CAPTURE_ALARM -> INIT_SEARCH -> START_SEARCH ->
          CLEAR_DATA -> START_STASTICS -> GENERATE_SUMMARY -> GENERATE_REPORT ->
          REPORT_FILLIN_TEMPLATE -> SEND_REPORT_FORM -> EDIT_REPORT_FORM
测试用例2: null -> CAPTURE_ALARM -> INIT_SEARCH -> START_SEARCH ->
          CLEAR_DATA -> START_STASTICS -> GENERATE_SUMMARY -> SUMMARY_FILLIN_TEMPLATE ->
          SEND_SUMMARY_FORM -> VIEW_SUMMARY_FORM
测试用例3: null -> CAPTURE_ALARM -> INIT_SEARCH -> START_SEARCH ->
          CLEAR_DATA -> START_STASTICS -> GENERATE_SUMMARY -> SUMMARY_FILLIN_TEMPLATE ->
          SEND_SUMMARY_FORM -> EDIT_SUMMARY_FORM
测试用例4: null -> CAPTURE_ALARM -> INIT_MODEL_CACULATE -> MODEL_CALCULATE ->
          MODEL_FILLIN_TEMPLATE -> SEND_MODEL_FORM -> VIEW_MODEL_FORM
测试用例5: null -> CAPTURE_ALARM -> INIT_MODEL_CACULATE -> MODEL_CALCULATE ->
          MODEL_FILLIN_TEMPLATE -> SEND_MODEL_FORM -> EDIT_MODEL_FORM
```

Fig. 7: The Chinese GUI screenshot of test case set generated according to system windows tree

```
INFO_CAPTURE_ALARM      成功次数:4      失败次数:0正确执行:1.0
GENERATE_REPORT         成功次数:2      失败次数:0正确执行:1.0
MODEL_CAPTURE_ALARM     成功次数:2      失败次数:0正确执行:1.0
SEND_MODEL_FORM         成功次数:0      失败次数:2正确执行:0.0
VIEW_MODEL_FORM         成功次数:1      失败次数:0正确执行:1.0
SEND_SUMMARY_FORM       成功次数:2      失败次数:0正确执行:1.0
EDIT_REPORT_FORM        成功次数:1      失败次数:0正确执行:1.0
GENERATE_SUMMARY        成功次数:4      失败次数:0正确执行:1.0
EDIT_MODEL_FORM         成功次数:1      失败次数:0正确执行:1.0
VIEW_REPORT_FORM        成功次数:0      失败次数:1正确执行:0.0

系统可信性 R = 0.8775510204081634
```

Fig. 8: The part of Chinese GUI results screenshot of trustworthiness evaluation based on model checking

Figure 7 shows a screenshot of test case set generated automatically once based on the system window tree, which contains 6 test cases that are composed of API sequences. Figure 8 shows the trustworthiness evaluation value of the System Under Test after a model checking. For showing the reality of the experiment, the original experiments screenshots with Chinese in Figure 7 and Figure 8 are introduced in the paper. The text comparison of Chinese and English is shown in Table 1.

Table 1 The comparison of Chinese and English in the Fig.7 and Fig.8

Fig. 7		Fig. 8	
测试用例列表	test cases list	成功次数:	test cases pass times
测试用例0:	test case number 0:	失败次数:	test cases fail pass times
……	……	正确执行:	test cases run times correctly
测试用例5:	test case number 5:	系统可信性 R	Trustworthiness Value *R*

Table 2 Trustworthiness results comparison

Test NO	Trustworthiness Value *R*	APIs of cause state transition incorrectly
1	0.7901	INFO_CAPTURE_ALARM(1), CLEAR_DATA(2), EDIT_SUMMARY_FORM(2), START_STATISTICS(3)
2	0.8537	REPORT_FILLIN_TEMPLATE(1),VIEW_REPORT_FORM(1), GENERATE_SUMMARY(3), SUMMARY_FILLIN_TEMPLATE(2)
3	0.8776	EDIT_SUMMARY_FORM(1),MODEL_FILLIN_TEMPLATE(2),

		SEND_MODEL_FORM(2),VIEW_REPORT_FORM(1)
4	0.9118	MODEL_CALCULATE(2)
5	0.9847	VIEW_MODEL_FORM(1)

As shown in Table 2, by the 5 times iteration test, the error of the system has been improved, and the trustworthiness of the system has been improved gradually, as shown in table 1. The third column of the table is API that causes the state transition to be incorrect, and the number in parentheses is times of the API call failed.

5. Conclusions and Future Work

In this paper, by the analysis of the typical phased mission system oriented to the earthquake response, puts forward a testing technology based on model testing, automatic test case generation based on the window tree and model checking method, based on trustworthiness evaluation algorithm with the weight of API to solve the problem of trustworthiness of large-scale phased mission system. Further work will study the problem of automatic generation and automatic adjustment of API weight.

6. Acknowledgements

The authors thank the anonymous reviewers for their insightful and constructive comments. This research work is supported by Social Service Project in National Earthquake Response Support Service "International Rescue and Disposition System against Strong Earthquakes" (NO.SJZX-B11).

7. References

[1] History earthquake list. China Earthquake Administration. http://www.cea.gov.cn/publish/dizhenj/468/496/index.html,2016-06-14.

[2] Zheng ZM, Ma SL, Li W, Wei W, Jiang X, Zhang ZL, Guo BH. Dynamical characteristics of software trustworthiness and their evolutionary complexity. Science in China (Series F: Information Sciences), 2009,52(8):1328−1334. [doi: 10.1007/s11432-009-0137-2]

[3] Zheng ZM, Ma SL, Li W, Jiang X, Wei W, Ma L, Tang ST. Complexity of software trustworthiness and its dynamical statistical analysis methods. Science in China (Series F: Information Sciences), 2009,52(9):1651−1657. [doi: 10.1007/s11432-009-0143-4]

[4] Su YD, Liu GJ, Qiu J. DSPN-based testability requirement modeling and analysis of phased-mission systems. Systems Enginering-Theory & Practice. 2010, 30(7): 1272 (in Chinese with English abstract).

[5] Xing T. Research on the reliability model for phased mission system. Journal of Naval Aeronautical Engineering Institute. 2006, 21(1): 172(in Chinese with English abstract).

[6] Mo Y, Xing LD. MDD-based method for efficient analysis on phased-mission systems with multimode failures. IEEE Trans. on Systems, Man, and Cybernetics: Systems. 2014, 44(6): 757.

[7] Wang HQ, LüHW, Zhao Q, Dong XK, Feng GS. Model and quantification of autonomic dependability of mission-critical systems. Journal of Software. 2010,21(2):344-358(in Chinese with English abstract).

[8] Li YZ. How to use automatic test efficiently in software test. Journal of Changsha Communication University. 2006,22(2): 63(in Chinese with English abstract).

[9] Emil A, Robert F, Pirjo K. Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing. Information and Software Technology. 2016, 66–80.

[10] Hu JQ. Research on some key technologies of Web Service discovery [Ph.D. Thesis]. Changsha: Dissertation for Doctoral Degree of School of National University of Defense Technology, 2005 (in Chinese with English abstract).

[11] Zhang T. Research on formal verification methods of model of complicated information system [Ph.D. Thesis]. Harbin: Harbin Engineering University, 2011 (in Chinese with English abstract).

[12] I.C. Norris, D. Dill, "Better verification through symmetry," Formal Methods in System Design, 1996, vol. 9(1/2),pp.41-75.

[13] A. Sistla, P. Godefroid, “Symmetry and reduced symmetry in model checking,” CAV. LNCS 2012, 2001,pp.91-103.

[14] R. Iosif, “Symmetry reduction criteria for software model checking,” Proceedings of SPIN Workshop. LNCS 2318, 2002,pp.22-41.

[15] D. Bosnacki, “A light-weight algorithm for model checking with symmetry reduction and weak fairness,” SPIN. LNCS 2648, 2003,pp.89-103.

[16] E. Emerson, T. Wahl, “Dynamic symmetry reduction,” Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 3440, 2005,pp.382-396.

[17] A. Miller, A. Donaldson, M. Calder, “Symmetry in temporal logic model checking,” ACM Computing Surveys, 2006, vol.38(3),pp.1-36.

[18] T. Wahl, “Adaptive symmetry reduction,” Proceedings of Computer Aided Verification(CAV’07). LNCS 4590, 2007,pp.393-405.

[19] J. Fernandez, M. Bozga, L. Ghirvu, “State space reduction based on live variables analysis,” Journal of Science of Computer Programming(SCP), 2003, vol.47(2/3),pp.203-220.

[20] J. Self, E. Mercer, “On-the-fly dynamic dead variable analysis,” Proceedings of SPIN Workshop. LNCS 4595, 2007,pp.113-130.

[21] J. Hatcliff, M. Dwyer, H. Zheng, “Slicing software for model construction,” Higher order Symbol. Compute, 2000, vol.13(4),pp.315-353.

[22] M. Dwyer, J. Hatcliff, M. Hoosier, et al, “Evaluating the effectiveness of slicing for model reduction of concurrent object oriented programs,” Proceedings of Tools and Algorithms for the Construction and Analysis of Systems. LNCS 3920, 2006,pp.73-89.

[23] Li R, Lian H, Ma SL, Li T. Avionics System Testing Based on Formal Methods. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):186 (in Chinese with English abstract).