Journal of Systems Engineering and Electronics
Vol. 23, No. 2, April 2012, pp.276–285

Real-time embedded software testing method based on extended finite state machine

Yongfeng Yin[1,*] Bin Liu[1], and Hongying Ni[2]

1. School of Reliability and Systems Engineering, Beihang University, Beijing 100191, P. R. China;
2. National Key Laboratory of Science and Technology on Avionics System Integration, Shanghai 200233, P. R. China

Abstract: The reliability of real-time embedded software directly determines the reliability of the whole real-time embedded system, and the effective software testing is an important way to ensure software quality and reliability. Based on the analysis of the characteristics of real-time embedded software, the formal method is introduced into the real-time embedded software testing field and the real-time extended finite state machine (RT-EFSM) model is studied firstly. Then, the time zone division method of real-time embedded system is presented and the definition and description methods of time-constrained transition equivalence class (timeCTEC) are presented. Furthermore, the approaches of the testing sequence and test case generation are put forward. Finally, the proposed method is applied to a typical avionics real-time embedded software testing practice and the examples of the timeCTEC, testing sequences and test cases are given. With the analysis of the testing result, the application verification shows that the proposed method can effectively describe the real-time embedded software state transition characteristics and real-time requirements and play the advantages of the formal methods in accuracy, effectiveness and the automation supporting. Combined with the testing platform, the real-time, closed loop and automated simulation testing for real-time embedded software can be realized effectively.

Keywords: real-time system, real-time embedded software, formal method, extended finite state machine (EFSM), testing sequence,
test case.

DOI: 10.1109/JSEE.2012.00035

1. Introduction

With the continuous development of computer technology, the real-time embedded software has been widely used in high-tech research and application areas, especially in aviation, aerospace, medical, transportation, modern weapon systems development and so on. Obviously, the failures of real-time embedded software often lead to serious consequences, therefore, the quality and reliability of the realtime embedded software have been paid more and more attention. The effective software testing is an important means to ensure software quality and the real-time embedded software testing has become a hot and difficult spot in the software testing field [1,2].

Generally speaking, any rigorous mathematical tool with precise mathematical semantics approach can be referred to as formal methods. The rise of formal methods has provided a new way for the software engineering because formal methods can ensure the software dependability in a rigorous way. The software testing technique based on formal methods can eliminate ambiguity, enhance the accuracy and consistency of verification and improve the automation level and efficiency. Both European Space Agency (ESA) and National Aeronautics and Space Administration (NASA) highly recommend using formal methods for the development and testing of safety-critical embedded software [3]. In addition to the traditional software formal specification and development, Bowen considered that the formal methods had great potential benefits in the software testing and described these benefits in improving software testing quality [4]. In summary, the software testing technique based on formal methods is an effective way to ensure the real-time embedded software quality.

Bell believed that the automated testing using formal methods was becoming a new means in the spacecraft software verification and validation field. At the same time, he proposed some methods to reduce risk in different stages of the software life cycle and described two formal verification tools developed by his research team [5]. Guerrouat proposed an assertion-based extended finite state machine (EFSM) model, named p-EFSM, for the embedded software testing which can be used to analyze and test crucial tasks effectively in the design of reliable embedded systems [6]. Using variety of formal methods and tools, U. S. Naval Research Laboratory successfully realized the

Manuscript received October 15, 2010.
*Corresponding author.
This work was supported by the Aviation Science Foundation of China (20095551025).

security verification of the read and write storage embedded devices software, and got higher test efficiency and reduced the verification costs [7]. Reference [8] put forward a time extended finite state machine to describe the system modeling with time characteristics which include how test cases are defined and applied to implementations, test derivation algorithms, producing and complete test suites. Reference [9] proposed an approach of property-oriented testing method for real-time systems based on unified modeling language (UML) statecharts. Combined with the hardware structure and using the fault injection method, a testing method for embedded system based on the EFSM model was given in [10]. Using the real-time extended finite state machine (RT-EFSM) model and a timed unique input/output sequence (t_UIO), the modeling and formal testing method for aircraft safety-critical software was proposed and the method in the aircraft inertial/satellite navigation systems software testing practice was applied successfully in [11]. Based on Z language, Chen studied the formal specification method for trusted platform module (TPM) and gave the extended finite state machine model which was applied in the test cases automated generation of TPM in [12]. Chen et al. presented an approach to test real-time systems based on a variant of timed automata (TA), named timed input/output automata (TIOA), put forward the process of constructing and executing the test cases and solved the time delay variables in the transition sequences using linear programming techniques in [13].

In this paper, the formal method is introduced into the real-time embedded software testing field and the RT-EFSM model is studied. At the same time, the definition and description of time-constrained transition equivalence class (timeCTEC) [14] and the testing sequence and test case generation methods are presented. Finally, the methods proposed in this paper are applied in typical avionics real-time embedded software testing practice and the correctness and effectiveness of the methods are verified.

2. Definition and characteristics of real-time embedded software

Real-time systems can respond to external random events occurring timely and complete the event handling with a fast speed. Most real-time systems are dedicated and complex, which requires high fault tolerance, and is typically embedded into a larger system, named the real-time embedded system. For a real-time embedded system, the system correctness is not only concerned with the correctness of the results, but the needs to complete the calculations within the specified time is more important, otherwise the system would be in error or failure.

The quality and reliability of a real-time embedded software often determine the reliability of real-time systems, because the failure of the software will directly or indirectly affects human life, property and the environment safety. Table 1 shows the characteristics of a real-time embedded software.

Table 1 Characteristics of a real-time embedded software

No.	Characteristics	Description
1	Real-time constraints	Real-time systems often have a strong time constraint that requires the function must be completed within the specified time, if delayed, disastrous consequences are often caused.
2	Real-time control	The real-time embedded software often involves real-time control which uses real-time data acquisition and processing to decide how to control the external cross-linked equipment.
3	Embedded	As an important part of the real-time system, most real-time embedded software is typically “embedded” into a larger system.
4	Interactive	Generally, the run-time environment of real-time embedded software is complex for involving many cross-linked equipments and I/O interfaces [15].
5	Reactive	Many real-time systems are reactive systems, which are based on event-driven, and the need to respond to external events [16].
6	Concurrent processing	A remarkable feature of real-time system is the concurrent processing and the arrival of external events usually cannot be expected.

3. RT-EFSM

The real-time embedded software often shows the behaviours of state-based in whole or in part. Therefore, when designing or testing such systems, the techniques based on state can be used. The software testing techniques based on state can fully verify the event, action and the relationship between state transitions which can help us to determine whether the system behaviours meet the system requirements. The most intuitive and effective way to complete the modelling and testing of the state-based real-time embedded system is to use techniques based on finite state machine (FSM) or EFSM [17,18].

Current research shows that the traditional testing methods based on FSM and EFSM cannot solve the real-time embedded software testing problems, for example, the time

description in state transition, concurrent processing, complex cross-linked equipments and I/O interfaces. In this paper, based on the careful analysis and previous studies, the formal methods, real-time embedded software characteristics and software testing techniques are organically combined and the real-time embedded software system testing method based on RT-EFSM is proposed.

Definition 1 RT-EFSM $=<S^*, S_0, I, O, T, V, G, L>$, S^*: non-empty set of finite states, then $\exists s \in S^*$, $s =<entry, exit, iact, itran, itevt, lt>$, where $entry$ is the entry of the state which is before any internal action and state transition; $exit$ is the exit of the state which is after all internal action and state transition; $iact$ is the internal action of the state; $itran$ is the internal transition of the state which can not cause entry and exit action occurs; $itevt$ is internal time-related events set in S; lt is the internal local time in S.

S_0: the initial state.

I: non-empty event set of input.

O: non-empty event set of output.

T: non-empty finite set of transition, and

$$T =< Head(t), I(t), C(t), act, O(t), Tail(t) >$$

where t is the state transition; $Head(t)$ is the start state of transition; $I(t)$ is the input event included in set I; $C(t)$ is the guard condition of the state transition which includes the variable constraints and time constraints. $C(t) = [V_C, T_C]$, where V_C is the variable constraint, and T_C is the time constraint. $T_C =<t_S, t_F, t_I>$, where t_S indicates that the transition time is a fix time, t_F indicates that the transition time is a stochastic time complies to some distribution function, and t_I indicates that the transition time is a time interval; act is the operations in state transition; $O(t)$ is the event set of output in O; $Tail(t)$ is the target state of the state transition.

V: non-empty set of variables, and $V =$ *<IV, CV, OV>*, where *IV* is the set of input variables, and they are under tester's control; *CV* is the set of environment variables, which can be local variables or global variables, and the variables that are neither input variables nor output variables can be called environment variables, *OV* is the set of output variables. Both *CV* and *OV* are out of the tester's control, and their values are decided by state transition operation.

G: non-empty set of state connection which is an extension to the original EFSM connection, and general connection, combination connection and selection joint connection are added, which can describe the system dynamic behaviors better, as shown in Fig. 1.

L: the global clock which will be used to record the state transition time.

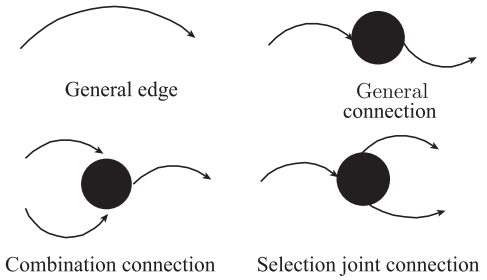

Fig. 1 Extended state connection in RT-EFSM

Based on the above analysis, Fig. 2 shows the RT-EFSM model of the aircraft flight control software.

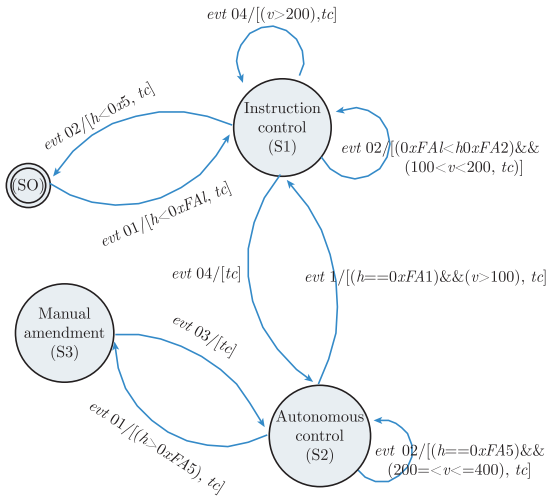

Fig. 2 RT-EFSM model of the aircraft flight control software

4. Real-time embedded software testing method based on RT-EFSM

4.1 Research on timeCTEC and US_{ex}

Definition 2 Time zone. Let the global clock L in the RT-EFSM model be the set of the state transition clocks in the real-time embedded software, then the range of L is $[0, +\infty)$. If the global clock L is divided into k sub-zones and the time point in L can be described as $L_1, L_2, \ldots, L_k (L_1 < L_2 < \ldots < L_k)$. Now we use $\{L_1, L_2, \ldots, L_k\}$ be the time zone division for time constraint ω in the L which can be shown in Fig. 3.

Fig. 3 Time zone division

From above definition, the time constraint $\omega : t_1 < l < t_2$ can be divided into three time zone: $(0, t_1]$, (t_1, t_2) and $[t_2, +\infty)$.

Definition 3 Valid time zone. Let E be the set of the state transition events in the real-time embedded software,

then $\exists e \in E$ and l_i is a time zone division of global clock L which means $l_i \in L$. If the event e occurs in l_i and could trigger the state transition, we claim that l_i is a valid time zone for event e.

Definition 4 Invalid time zone. Let E be the set of the state transition events in the real-time embedded software, then $\exists e \in E$ and l_i is a time zone division of global clock L which means $l_i \in L$. If the event e occurs in l_i and could not trigger the state transition, we claim that l_i is an invalid time zone for event e.

In state transitions of real-time embedded software, state transitions are often triggered by state variable constraints. In addition, time constraints can be used as guard conditions and trigger the transitions. Based on the time zone division method, Fig. 4 shows the state transition with variable and time constraints.

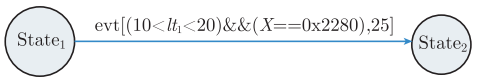

Fig. 4 State transition with variable and time constraints

As shown in Fig. 3, let I be the input of the state transition and the guard condition includes the time constraint ($10 < lt < 20$) and variable constraint ($X ==$ 0x2280). Obviously, when the variable constraint is true which means $X ==$ 0x2280, and if the input event I occurs in the invalid time zone (0,10] or [5, +∞), it will not trigger state transition. Only when the variable constraint is true and the input I occurs in the valid time, it will trigger state transition.

Based on above analysis, timeCTEC is proposed in this paper. The timeCTEC gives the time zone division method for the state transition and puts forward the equivalence classes division method of state transition for the events in different time zone which lay the foundation for subsequent automated testing

Definition 5 timeCTEC=$\{(S_{src} \overset{evt}{\rightarrow} S_{trgt})\_[C]\_?I\_!O\}$, where S_{src} is the source state of the transition; S_{trgt} is the target state of the transition; evt is the event that triggers the transition; C is the guard condition of the transition which includes variable and time constraints and $\mathrm{C} = <\mathrm{tCnd}, \mathrm{vCnd}>$; [] means that C is optional. When there is no pre-conditions, C can be omitted. ? means input; I is the input variables and operations and can be described as $I = <ivVle, iAct>$; ! means output; O is the output variables and operations and can be described as $O = <ovVle, oAct>$.

Definition 6 Extended testing sequence US_{ex}. According to the definition of the timeCTEC, the method based on the test scenario tree (TST) is used to generate testing sequences. The TST is formed by timeCTEC which indicates a complete description of the testing path.

$$US_{ex} = <timeCTEC_1 \cup \ldots \cup timeCTEC_n> =$$
$$\{(S_i \rightarrow S_j)\_ <tCnd_{i\rightarrow j}, vCnd_{i\rightarrow j}> \_?$$
$$<ivVle_{i\rightarrow j}, iAct_{i\rightarrow j}> \_! < ovVle_{i\rightarrow j}, oAct >_{i\rightarrow j}\} \cup$$
$$\{(S_j \rightarrow S_k)\_ < tCnd_{j\rightarrow k}, vCnd_{j\rightarrow k} > \_?$$
$$<ivVle_{j\rightarrow k}, iAct_{j\rightarrow k}> \_!$$
$$<ovVle_{j\rightarrow k}, oAct>_{j\rightarrow k}\} \cup \ldots$$

where $0 \leqslant i < j < k \leqslant n$ and n is the maximum state space of the system.

Definition 7 Extended unique input/output sequence $UIO_{ex} = \{[C]\_?I\_!O\}$. The meaning of each elements of UIO_{ex} and the sequence refer to Definition 6.

4.2 Testing sequence generation method based on timeCTEC

According to above definitions, Fig. 5 gives the testing sequence generation process based on timeCTEC.

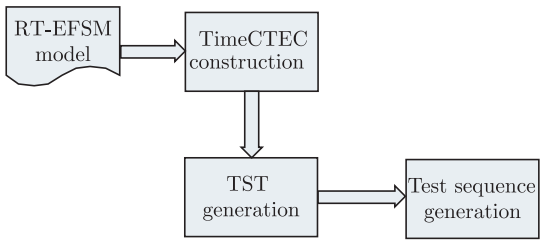

Fig. 5 The testing sequence generation process based on timeCTEC

(i) The construction of timeCTEC

The steps of the timeCTEC construction are as follows.

Step 1 Traverse the states set S^* and the input events set I in the RT-EFSM model and get all the trigger events $e \in I$ for every state $s \in S^*$;

Step 2 Creat the transition set T^* triggered by event e. According to the guard condition C^* in T^*, we should select $c \in C^*$, calculate the time zone division and get the the variable constraints and time constraints.

Step 3 Based on the above analysis, all timeCTEC will be listed as a list.

(ii) TST generation

Based on timeCTEC, the TST is a complete description of the test path. We can get testing sequences by traversing the TST. Table 2 shows the construction algorithm of the TST.

(iii) Testing sequence generation

In this paper, each state transition in the RT-EFSM model corresponds to some timeCTEC in the TST and each testing sequence can be described by US_{ex}. The testing sequence generation algorithm is given in Table 3.

Table 2 The construction algorithm of TST

Algorithm 1 The construction algorithm of TST

```
Input: RT-EFSM Model
Output: TST
Step 1:
    // Initialization process:
       Let TST contain only the root node of RT-EFSM Mode l;
       Let the node variable, named Node, point to the root node;
       Let the current node variable, named curNode, point to the subsequent state of root;
Step 2:
    IF   the childNumber of node==0
           IF   the subsequentState ==NULL;
           THEN   END;
     ELSE
          curNode point to the subsequent state of node;
          construct timeCTEC of curNode;
          FOR   Each   timeCTEC
                Back to previous node;
                IF   timeCTEC is not avilible in back path  THEN
                     Let curNode be the child of node
                IF   childNumber==0 THEN
                     Let node point to the parent of curNode,  jump to Step 2;
                ELSE
                     Let node point to the first child of curNode,  jump to Step 2;
          END FOR
Step 3:
      IF     the childNumber of node>0 THEN
        FOR    EACH child of node
                IF    child is not visited  THEN
                      node point to the child;
                      Mark the child as visited, jump to  Step 2;
        END FOR
       IF   (All child nodes have been visited) AND (node==root) THEN END;
       ELSE
            Let node point to the parent of curNode,  jump to  Step 2;
```

Table 3 The testing sequence generation algorithm based on the TST

Algorithm 2 The testing sequence generation algorithm based on the TST

```
Input: TST
Output: Testing sequence (TS)
Step 1:
    // Initialization process:
      Let TS be empty;
      Let current timeCTEC, named curNode, point to the root of TST;
      Let evt  be the event of the curNode;
Step 2:
     IF   curNode is the  root  THEN
          IF   curNode has not been visited  THEN
                    Output the current timeCTEC to TS;
                    Mark curNode as visited;
          FOR   EACH child of curNode
             IF   child has not been visited THEN
                  Let curNode point to child, jump to Step 2;
            IF   All nodes have been visited  THEN END;
          END FOR
Step 3:
     IF curNode is not the  root THEN
          IF   curNode  has not been visited THEN
                    Output the current timeCTEC to TS;
                    Mark curNode as visited;
          IF   the childNumber of curNode==0 THEN
                   Output TS;
                   Let evt of the last timeCTEC in TS be empty;
                   Bask to the parent node, jump to Step 2;
       ELSE
            FOR EACH child of curNode
                 IF   child has not been visited THEN
                      Let child point to curNode, jump to Step 2;
            END FOR
```

4.3 Test case generation

By traversing the testing sequences, we can get all timeCTEC information, and then generate test cases based on certain test coverage criteria.

In testing method based on FSM, the test coverage criteria commonly used include state coverage, state transition coverage, boolean coverage, full predicate coverage and conversion pair coverage[19,20], and these coverage cri- teria algorithms have been more mature. Using different test coverage criteria, we can generate different test case sets which have different ability to reveal the errors of software under test (SUT).

In addition to the above coverage criteria, we propose a time condition coverage criterion with the combination of the time constraints of real-time embedded software. The time condition coverage criterion requires that each state transitions in the RT-EFSM model must meet the time constraints which include fix time constraint, time distribution function constraint and time interval constraint. The definition of the time condition coverage criterion is as follows.

Definition 8 Time condition coverage criteria. The state transitions in the RT-EFSM model should meet $\exists S_i \in$

$S^*, \exists S_j \in S^*, \forall t \in T\{S_i \to S_j\}$, and $t = \{t|t = t_S||t \leqslant t_F||t \in t_I\}$.

According to the above criteria, it requires that the time-related state transitions in RT-EFSM should take one time from each of the TRUE or FALSE value.

5. Case study

In this paper, the obtained research results are applied to the actual engineering project, e.g., inertial/global positioning system (GPS) navigation system software (I/GNSS) testing practice, which includs the modeling of SUT, the design of timeCTEC, the automatic generation of the testing sequences and test cases.

5.1 RT-EFSM model of SUT

The inertial/GPS navigation system (I/GNS) is a dedicated real-time control system to realize all-attitude autonomous navigation for the aircraft which can provide acceleration, velocity, position, heading, attitude and time information and complete the functions of build-in test (BIT), air data computing, alignment, calibration, navigation and maintenance independently. In addition, I/GNS has some cross-link subsystems such as flight control subsystem, mission control subsystem and integrated display control subsystem.

As a typical real-time embedded software, I/GNSS is cured the static memory of I/GNS and its correctness depends on the logical correctness and accuracy of real-time constraint.

The typical characteristics of the I/GNSS are as follows: the real-time constraint is 25 ms; there are some working states and operating process, and state transition conditions are complex; there are some data exchanges with variety of aviation electronic systems; there are some data transfer protocols, such as MIL-STD-1553, ARINC422, RS422.

According to the requirements specification of I/GNSS, Fig. 6 shows the RT-EFSM model of I/GNSS.

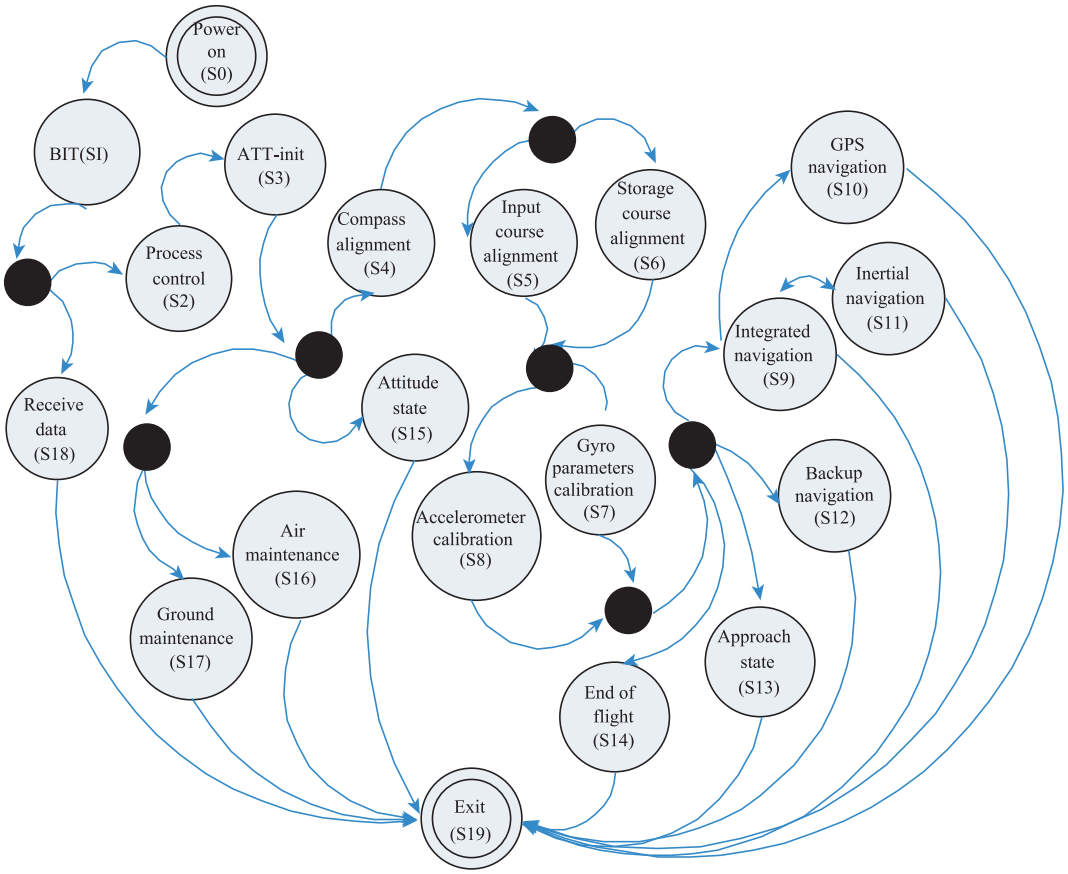

Fig. 6 RT-EFSM model of I/GNSS

5.2 The timeCTEC construction of I/GNSS

Based on the analysis of the software documents of I/GNSS, all state transitions information can be obtained and timeCTEC construction can be realized. Some timeCTEC examples are given in Table 4.

Table 4 timeCTEC examples of I/GNSS

Source state	Target state	Constraint (P)		Input($?I$)		Output ($!O$)	
		Variable constraint	Time constraint	Input variable	Input action	Output variable	Output action
S_0	S_1	IGNS.WOW==1	$gt \leqslant 5$ ms	IGNS.APP	NONE	DCMP. INDC_00_00	SendDataValue(IGNS, DCIN_00_00,DCMP, INDC_00_00,1553B)
S_1	S_{18}	IGNS.WOW==1; DCMP.INDC_00 _00.VALID==1	$gt > 40$ ms	IGNS.DTIN_01_00	SendDataValue (0x3310,IGNS, DTIN_01_00)	IGNS. DTIN_01_00. VALID=1	NONE
S_2	S_3	IGNS.WOW==0; DCMP.INDC_00 _00.VALID==1	$gt > 40$ ms	IGNS.DCIN_01_00	SendDataValue (0x4010,IGNS, DCIN_01_00)	DCMP. INDC_01_00	SendDataValue(IGNS, DCIN_01_00,DCMP, INDC_01_00,1553B)
S_3	S_{15}	IGNS.WOW==0;	$gt > 70$ ms	IGNS.DCIN_00_00	SendDataValue (0x23D0,IGNS, DCIN_00_00)	DCMP. INDC_19 _00=0x3500	SendDataValue(IGNS, DCIN_00_00,DCMP, INDC_09_00,1553B)
S_3	S_{17}	IGNS.WOW==1;	$gt > 25$ ms	IGNS.DCIN_00_00	SendDataValue (0x6122,IGNS, DCIN_00_00)	DCMP. INDC_00_00	SendDataValue(IGNS, DCIN_00_00,DCMP, INDC_00_00,1553B)
S_{14}	S_{19}	NONE	NONE	IGNS.DCIN_00_00	SendDataValue (0x9910,IGNS, DCIN_00_00);	DTE.INDT_03_00	SendDataValue(IGNS, DTIN_03_00,DTE, INDT_03_00,1553B)

5.3 Testing sequence and test case generation

Based on above timeCTEC, we can complete the construction of TST. Some typical TSTs of I/GNSS are listed below because of the limitation of space.

The flattening TST examples are shown in Fig. 7. Using the testing sequence generation algorithm, the testing sequences described with can be obtained as shown in Fig. 8.

$$\text{TST 01}: \ S_0 \to S_1 \to S_{18} \to S_{19}$$

$$\text{TST 02}: \ S_0 \to S_1 \to S_2 \to S_3 \to S_{17} \to S_{19}$$

$$\text{TST 03}: \ S_0 \to S_1 \to S_2 \to S_3 \to S_4 \to S_5 \to S_7 \to S_9 \to S_{11} \to S_{19}$$

$$\text{TST 04}: \ S_0 \to S_1 \to S_2 \to S_3 \to S_4 \to S_5 \to S_7 \to S_9 \to S_{10} \to S_9 \to S_{11} \to S_{19}$$

Fig. 7 Flattening test scenario tree examples

Based on the generation of testing sequences, we can realize the generation of test cases using the test coverage criteria. Using the black testing method, we can realize function testing, performance testing, boundary testing, strength testing, safety testing, interface testing and so on. The test case examples are shown in Fig. 9.

5.4 Testing result analysis

In this paper, the case study completes the generation of 375 test cases as shown in Fig. 6 and realizes the full testing of I/GNSS including functional testing, performance testing, boundary testing, strength testing, safety testing, interface testing and so on.

With the running of the test cases imposed on the testing platform, 46 software bugs and defects including loss of functions, function errors, state transition errors, timing errors and logic errors are found. Software bugs and defects are shown in Fig. 10 and Fig. 11.

6. Conclusions

In this paper, based on the analysis of the characteristics of real-time embedded software, the real-time embedded software testing method based on RT-EFSM is studied. The testing practice verification shows that the testing method proposed in this paper can fully make use of the advantages of the formal methods in accuracy, efficiency and automation level of support. Combined with the automated testing platform, the real-time, closed loop and automated simulation testing of real-time embedded software can be realized.

The next step will study the real-time embedded software automated testing description method and develop automated support tools to improve the automation level, efficiency and correctness of testing.

$US_{ex}01$:

$\{(S_0 \rightarrow S_1)\_ < tCnd_{0\rightarrow1}, vCnd_{0\rightarrow1} > \_? < ivVle_{0\rightarrow1}, iAct_{0\rightarrow1} > \_! < ovVle_{0\rightarrow1}, oAct >_{0\rightarrow1}\} \cup$

$\{(S_1 \rightarrow S_{18})\_ < tCnd_{1\rightarrow18}, vCnd_{1\rightarrow18} > \_? < ivVle_{1\rightarrow18}, iAct_{1\rightarrow18} > \_! < ovVle_{1\rightarrow18}, oAct >_{1\rightarrow18}\} \cup$

$\{(S_{18} \rightarrow S_{19})\_ < tCnd_{18\rightarrow19}, vCnd_{18\rightarrow19} > \_? < ivVle_{18\rightarrow19}, iAct_{18\rightarrow19} > \_! < ovVle_{18\rightarrow19}, oAct >_{18\rightarrow19}\}$

$US_{ex}02$:

$\{(S_0 \rightarrow S_1)\_ < tCnd_{0\rightarrow1}, vCnd_{0\rightarrow1} > \_? < ivVle_{0\rightarrow1}, iAct_{0\rightarrow1} > \_! < ovVle_{0\rightarrow1}, oAct >_{0\rightarrow1}\} \cup$

$\{(S_1 \rightarrow S_2)\_ < tCnd_{1\rightarrow2}, vCnd_{1\rightarrow2} > \_? < ivVle_{1\rightarrow2}, iAct_{1\rightarrow2} > \_! < ovVle_{1\rightarrow2}, oAct >_{1\rightarrow2}\} \cup$

$\{(S_2 \rightarrow S_3)\_ < tCnd_{2\rightarrow3}, vCnd_{2\rightarrow3} > \_? < ivVle_{2\rightarrow3}, iAct_{2\rightarrow3} > \_! < ovVle_{2\rightarrow3}, oAct >_{2\rightarrow3}\} \cup$

$\{(S_3 \rightarrow S_{17})\_ < tCnd_{3\rightarrow17}, vCnd_{3\rightarrow17} > \_? < ivVle_{3\rightarrow17}, iAct_{3\rightarrow17} > \_! < ovVle_{3\rightarrow17}, oAct >_{3\rightarrow17}\} \cup$

$\{(S_{17} \rightarrow S_{19})\_ < tCnd_{17\rightarrow19}, vCnd_{17\rightarrow19} > \_? < ivVle_{17\rightarrow19}, iAct_{17\rightarrow19} > \_! < ovVle_{17\rightarrow19}, oAct >_{17\rightarrow19}\}$

$US_{ex}03$:

$\{(S_0 \rightarrow S_1)\_ < tCnd_{0\rightarrow1}, vCnd_{0\rightarrow1} > \_? < ivVle_{0\rightarrow1}, iAct_{0\rightarrow1} > \_! < ovVle_{0\rightarrow1}, oAct >_{0\rightarrow1}\} \cup$

$\{(S_1 \rightarrow S_2)\_ < tCnd_{1\rightarrow2}, vCnd_{1\rightarrow2} > \_? < ivVle_{1\rightarrow2}, iAct_{1\rightarrow2} > \_! < ovVle_{1\rightarrow2}, oAct >_{1\rightarrow2}\} \cup$

$\{(S_2 \rightarrow S_3)\_ < tCnd_{2\rightarrow3}, vCnd_{2\rightarrow3} > \_? < ivVle_{2\rightarrow3}, iAct_{2\rightarrow3} > \_! < ovVle_{2\rightarrow3}, oAct >_{2\rightarrow3}\} \cup$

$\{(S_3 \rightarrow S_4)\_ < tCnd_{3\rightarrow4}, vCnd_{3\rightarrow4} > \_? < ivVle_{3\rightarrow4}, iAct_{3\rightarrow4} > \_! < ovVle_{3\rightarrow4}, oAct >_{3\rightarrow4}\} \cup$

$\{(S_4 \rightarrow S_5)\_ < tCnd_{4\rightarrow5}, vCnd_{4\rightarrow5} > \_? < ivVle_{4\rightarrow5}, iAct_{4\rightarrow5} > \_! < ovVle_{4\rightarrow5}, oAct >_{4\rightarrow5}\} \cup$

$\{(S_5 \rightarrow S_7)\_ < tCnd_{5\rightarrow7}, vCnd_{5\rightarrow7} > \_? < ivVle_{5\rightarrow7}, iAct_{5\rightarrow7} > \_! < ovVle_{5\rightarrow7}, oAct >_{5\rightarrow7}\} \cup$

$\{(S_7 \rightarrow S_9)\_ < tCnd_{7\rightarrow9}, vCnd_{7\rightarrow9} > \_? < ivVle_{7\rightarrow9}, iAct_{7\rightarrow9} > \_! < ovVle_{7\rightarrow9}, oAct >_{7\rightarrow9}\} \cup$

$\{(S_9 \rightarrow S_{11})\_ < tCnd_{9\rightarrow11}, vCnd_{9\rightarrow11} > \_? < ivVle_{9\rightarrow11}, iAct_{9\rightarrow11} > \_! < ovVle_{9\rightarrow11}, oAct >_{9\rightarrow11}\} \cup$

$\{(S_{11} \rightarrow S_{19})\_ < tCnd_{11\rightarrow19}, vCnd_{11\rightarrow19} > \_? < ivVle_{11\rightarrow19}, iAct_{11\rightarrow19} > \_! < ovVle_{11\rightarrow19}, oAct >_{11\rightarrow19}\}$

$US_{ex}04$:

$\{(S_0 \rightarrow S_1)\_ < tCnd_{0\rightarrow1}, vCnd_{0\rightarrow1} > \_? < ivVle_{0\rightarrow1}, iAct_{0\rightarrow1} > \_! < ovVle_{0\rightarrow1}, oAct >_{0\rightarrow1}\} \cup$

$\{(S_1 \rightarrow S_2)\_ < tCnd_{1\rightarrow2}, vCnd_{1\rightarrow2} > \_? < ivVle_{1\rightarrow2}, iAct_{1\rightarrow2} > \_! < ovVle_{1\rightarrow2}, oAct >_{1\rightarrow2}\} \cup$

$\{(S_2 \rightarrow S_3)\_ < tCnd_{2\rightarrow3}, vCnd_{2\rightarrow3} > \_? < ivVle_{2\rightarrow3}, iAct_{2\rightarrow3} > \_! < ovVle_{2\rightarrow3}, oAct >_{2\rightarrow3}\} \cup$

$\{(S_3 \rightarrow S_4)\_ < tCnd_{3\rightarrow4}, vCnd_{3\rightarrow4} > \_? < ivVle_{3\rightarrow4}, iAct_{3\rightarrow4} > \_! < ovVle_{3\rightarrow4}, oAct >_{3\rightarrow4}\} \cup$

$\{(S_4 \rightarrow S_5)\_ < tCnd_{4\rightarrow5}, vCnd_{4\rightarrow6} > \_? < ivVle_{4\rightarrow5}, iAct_{4\rightarrow5} > \_! < ovVle_{4\rightarrow5}, oAct >_{4\rightarrow5}\} \cup$

$\{(S_5 \rightarrow S_7)\_ < tCnd_{5\rightarrow7}, vCnd_{5\rightarrow7} > \_? < ivVle_{5\rightarrow7}, iAct_{5\rightarrow7} > \_! < ovVle_{5\rightarrow7}, oAct >_{5\rightarrow7}\} \cup$

$\{(S_7 \rightarrow S_9)\_ < tCnd_{7\rightarrow9}, vCnd_{7\rightarrow9} > \_? < ivVle_{7\rightarrow9}, iAct_{7\rightarrow9} > \_! < ovVle_{7\rightarrow9}, oAct >_{7\rightarrow9}\} \cup$

$\{(S_9 \rightarrow S_{10})\_ < tCnd_{9\rightarrow10}, vCnd_{9\rightarrow10} > \_? < ivVle_{9\rightarrow10}, iAct_{9\rightarrow10} > \_! < ovVle_{9\rightarrow10}, oAct >_{9\rightarrow10}\} \cup$

$\{(S_{10} \rightarrow S_9)\_ < tCnd_{10\rightarrow9}, vCnd_{10\rightarrow9} > \_? < ivVle_{10\rightarrow9}, iAct_{10\rightarrow9} > \_! < ovVle_{10\rightarrow9}, oAct >_{10\rightarrow9}\} \cup$

$\{(S_9 \rightarrow S_{11})\_ < tCnd_{9\rightarrow11}, vCnd_{9\rightarrow11} > \_? < ivVle_{9\rightarrow11}, iAct_{9\rightarrow11} > \_! < ovVle_{9\rightarrow11}, oAct >_{9\rightarrow11}\} \cup$

$\{(S_{11} \rightarrow S_{19})\_ < tCnd_{11\rightarrow19}, vCnd_{11\rightarrow19} > \_? < ivVle_{11\rightarrow19}, iAct_{11\rightarrow19} > \_! < ovVle_{11\rightarrow19}, oAct >_{11\rightarrow19}\}$

Fig. 8 The testing sequences examples

*TestCase*01:

{(S_0→S_1)_<(IGNS.WOW= =1)&&(gt<=5)>_?<IGNS.APP=1>_!
<SendData Value(IGNS,DCIN_00_00,DCMP,INDC_00_00,1553B)>}∪
{(S_1→S_{18})_<(IGNS.WOW= =1)&&(DCMP.INDC_00_00.VALID= =1)&&(gt>=40)>?
<IGNS.DTIN_01_00,SendData Value(0x3310, IGNS, DTIN_01 00)>_!
<(IGNS.DTIN_01_00.VALID=1),(NONE)>}∪
{(S_{18}→S_{19})_[NONE]_?<(IGNS.DCIN_00_00),SendData Value(0x9910,IGNS,DCIN_00_00)>+!
<(DTE.INDT_03_01),(SendData Value(IGNS,DTIN_03_00,DTE,INDT_03_00,1553B))>}

*TestCase*02:

{(S_0→S_1)_<(IGNS.WOW= =)&&(gt<=5)>_?<IGNS.APP=1>_!
<SendData Value(IGNS,DCIN_00_00,DCMP,INDC_00_00,1553B)>}∪
{(S_1→S_2)_<(IGNS.WOW= =1)&&(INDT_01_00.IN_052>=8.0),(gt<=40ms)&&(It<=20ms)>_?
<IGNSDCIN_01_00,(GetData Value(DIE,INDT_01_00,1553B);SendData Value(0x4310,IGNS,DCIN_01_00))>_!
<DCMP,INDC_01_00,SendDataValue(IGNS,DCIN_01_00,DCMP,INDC_01_00,1553B)>}
{(S_2→S_3)_<(IGNS.WOW= =0)&&(DCMP.INDC_00_00.VALID= =1),gt>40ms>_?
<IGNS.DCIN_01_00,SendDataValue(0x4010,IGNS,DCIN_01_00)>_!
<DCMP.INDC_01_00,SendData Value(IGNS,DCIN_01_00,DCMP,INDC_01_00,1553B)>}∪
{(S_3→S_{17})_<IGNS.DCIN_00_00,SendData Value(0x6122,IGNS,DCIN_00_00)>_?
<IGNS.DCIN_00_00,SendData Value(0x6122,IGNS,DCIN_00_00)>_!
<DCMP.INDC_00_00,SendData Value(IGNS,DCIN_00_00,DCMP,INDC_00_00,1553B)>}∪
{(S_{17}→S_{19})_<NONE>_?<IGNS.DCIN_00_00,SendData Value(0x9010,IGNS,DCIN_00_00)>_!
<DTE.INDT_03_00,SendData Value(IGNS,DTN_03_00,DTE,INDT_03_00,1553B)>}

Fig. 9 The test case examples

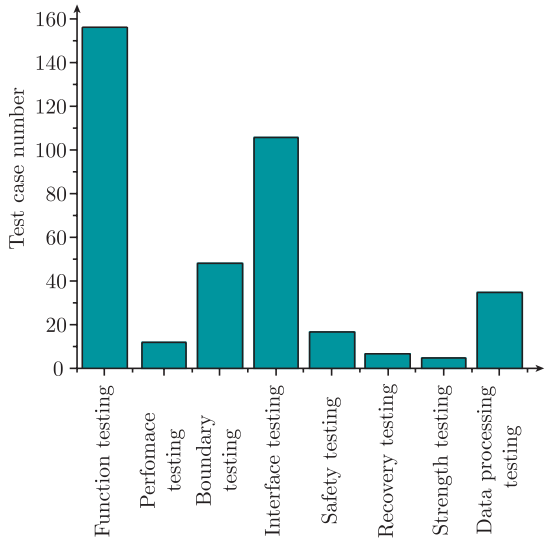

Fig. 10 The test cases statistics

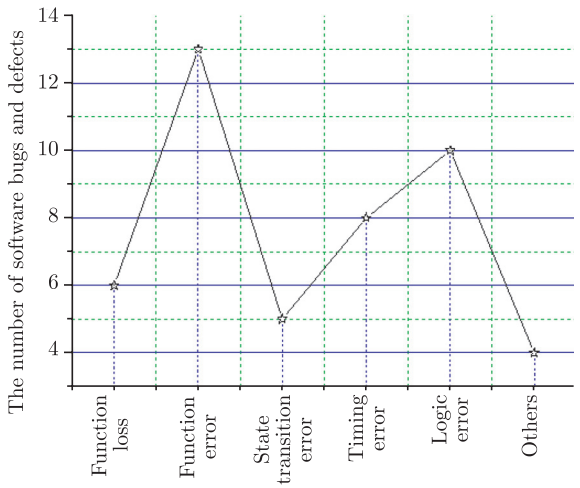

Fig. 11 Software bugs and defects of I/GNSS

References

[1] Y. F. Yin, B. Liu, D. M. Zhong, et al. On modeling approach for embedded real-time software simulation testing. *Journal of Systems Engineering and Electronics*, 2009, 20(2): 420–426.

[2] Z. D. Qin, H. Lei , N. Sang, et al. Study on the reliability demonstration testing method for safety-critical software. *Chinese Journal of Aeronautics*, 2005, 26(3): 334–339. (in Chinese)

[3] NASA software safety guidebook. NASA-GB-8719.13, NASA, March 31, 2004.

[4] J. P. Bowen, B. Kirill, C. John, et al. For test: formal methods and testing. *Proc. of the 26th International Conference on Computer Software and Applications*, 2002: 91–101.

[5] D. G. Bell, G. P. Brat. Automated software verification & validation: an emerging approach for ground operations. *Proc. of the IEEE Aerospace Conference*, 2008: 1–8.

[6] A. Guerrouat, H. Richter. A formal approach for analysis and testing of reliable embedded systems. *Electronic Notes in Theoretical Computer Science*, 2005, 141(3): 91–106.

[7] C. L. Heitmeyer, M. M. Archer, E. I. Leonard, et al. Applying formal methods to a certifiably secure software system. *IEEE Trans. on Software Engineering*, 2008, 34(1): 82–98.

[8] G. M. Mercedes, N. Manuel, R. Ismael. Formal testing from timed finite state machines. *Computer Networks*, 2008, 52(2): 432–460.

[9] S. H. Li, J. Wang, W. Dong, et al. A framework of property-oriented testing of reactive systems. *Chinese Journal of Electronics*, 2004, 32(12A): 222–225. (in Chinese)

[10] F. Fummi, C. Marconcini, G. Pravadelli. Functional verification based on the EFSM model. *Proc. of the 9th IEEE International High-Level Design Validation and Test Workshop*, 2004: 69–74.

[11] Y. F. Yin, B. Liu, D. Su. On formal verification technique for aircraft safety–critical software. *Journal of Computers*, 2010, 5(8): 1152–1159.

[12] X. F. Chen. The formal analysis and testing of trusted platform module. *Chinese Journal of Computers*, 2009, 32(4): 646–653. (in Chinese)

[13] W. Chen, Y. Z. Xue, C. Zhao, et al. A method for testing real-time system based on timed automata. *Journal of Software*, 2007, 18(1): 62–73. (in Chinese)

[14] Y. F. Yin, B. Liu, H. Y. Ni. Avionics embedded software modeling based on time-constrained transition equivalence class. *Advanced Science Letters*, 2012, 5(2): 844–847.

[15] B. Liu, X. P. Gao, M. Y. Lu, et al. Study on reliability simulation soft-ware system for embedded software. *Journal of Beijing University of Aeronautics and Astronautaics*, 2000, 26(4): 490–493. (in Chinese)

[16] Z. Mao, A. Vasu, O. Olga. Automated generation of test suites from formal specifications of real-time reactive systems. *The Journal of Systems and Software*, 2008, 81(2): 286–304.

[17] D. Lee, D. Chen, R. Hao, et al. A formal approach for passive testing of protocol data portions. *Proc. of the 10th IEEE International Conference on Network Protocols*, 2002:122–131.

[18] Y. Zhang, L. Q. Qian, Y. F. Wang. Automatic testing data generation in the testing based on EFSM. *Chinese Journal of Computers*, 2003, 26 (10): 1295–1303. (in Chinese)

[19] X. D. Zhan, H. K. Miao. An approach to formalizing the semantics of UML statecharts. *Lecture Notes in Computer Science*, 2004, 3288(1): 753–765.

[20] L. L. Ji. Researeh and implementation of automated test case generation based on UML statement diagram model. Nanjing: University of Aeronautics and Astronautics, 2004.

Biographies

Yongfeng Yin was born in 1978. Now he is a researcher and Ph.D. of Beihang University. His research interests include system engineering, software engineering and formal method.
E-mail: yyf@buaa.edu.cn

Bin Liu was born in 1968. Now he is a Ph.D. advisor and a professor of Beihang University. His research interests include systems engineering and software reliability.
E-mail: liubin@buaa.edu.cn

Hongyi Ni was born in 1965. Now she is a professor of National Key Laboratory of Science and Technology on Avionics System Integration. Her research interest includes real-time system testing.
E-mail: ni_hongying@careri.com