Research on Embedded Software Testing Framework Based on Formal Specification

Kai Qin ,Tao Zhang ,Zhengyi Shuai ,Jingbo Zhang ,Xu Mao
School of Software and Microelectronics, Northwestern Polytechnical University
Xi'an,China
893398733@qq.com,tao_zhang@nwpu.edu.cn

***Abstract*—Focusing on the functional testing of embedded software, this paper studies the formal testing model based on extended finite state machine. Then, the authors summarizes the complete construction process of embedded software testing based on formal specification and design the formal testing framework of embedded software. This framework describes the complete formal test execution flow from formal description of software requirements to the final test report. It solves the problems of the engineering application which is difficult to promote by formal test methods in embedded software. Finally, based on the results of theoretical research, a typical example of embedded application is verified to prove the effectiveness of the method.**

Keywords—embedded software ; formal specification ; formal test framework

I. INTRODUCTION

With the rapid development of computer technology, the scale and complexity of embedded software have risen dramatically. The security, reliability and quality of software have become an increasingly prominent problem[1]. Software testing is one of the key technologies to ensure the security and reliability of software system. Software testing method based on formal specification, through the formal description of the software requirements and design specifications, supports the automated generation of test cases and test scripts, automatic statistical analysis of software test coverage. It can also establish traceability relationship between requirements, specifications and tests. The domestic and international researches show that the use of formal software testing methods is one of the effective ways to improve the quality of embedded software testing[2~6].

However, the traditional formal software testing method lacks a simple and effective formal testing framework. The testing process mostly depends much on the personal experiences of testers. So far there is no such thing as a complete construction system that covers the whole process from formal description of software requirements, formal test model building, test scenario design, the link of program under test to test program, test coverage calculation and test report generation. How to apply the formal testing theory to the real embedded software testing environment and establish a simple formal testing framework has become a pressing problem to be solved so that the software formal testing methods are widely used.

Therefore, formal test framework is the key to tackle the learning and application difficulty of formal test method. In order to meet the real needs of embedded software testing, it is necessary to study simple and effective formal testing framework to set up an embedded software testing environment based on formal specification easily. Firstly, this paper summarizes a formal test model based on extended finite state machine according to the functional requirements of embedded software. Besides, we also study such methods as linking program under test and test program, automatically calculating software test coverage and constructing test scenarios. Finally, the unmanned aerial vehicle(UAV) flight control task is taken as an example to verify the effectiveness of the method.

II. FORMAL TEST MODEL

The traditional finite state machine model can clearly describe the state transition of the system, but it can't determine the input/output conditions of the system and can't effectively verify the system requirement specification. Extended Finite State Machine (EFSM), based on finite state machine (FSM), can describe the behaviors of the software system more accurately and describe the embedded software requirements precisely. Therefore, this paper uses formal testing model based on extended finite state machine (FTM) to reflect the software status of the migration process more accurately.

Testers use formal language to describe the functions of embedded software to be tested to generate formal test models, and then instantiate test models to form test cases. The formal test model introduces precondition to judge the input data of the system, thus filtering the input of invalid data. The concrete test operation is carried out by using the valid data in line with the embedded software requirements. The postcondition is introduced to judge the execution result of the test program. Then different operations will be performed according to different judgments in a way to describe the process of software status migration. In addition, invariant is used to define the conditions that must be satisfied in formal testing. Formal test model is shown in Fig.1.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on June 06,2022 at 07:37:21 UTC from IEEE Xplore. Restrictions apply.

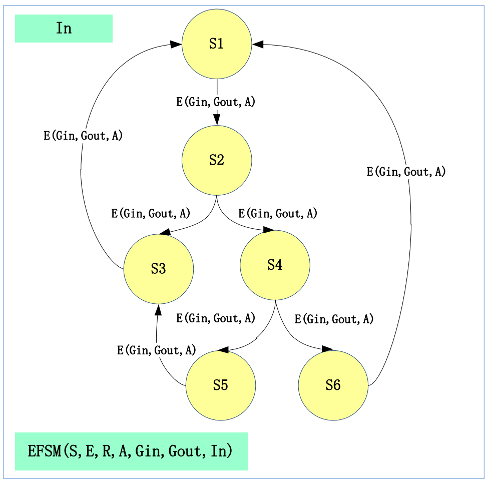

Fig. 1. Formal test model based on extended finite state machine

Definition 1. Defines a formal test model based on extended finite state machine as a seven-tuple:

$$FTM=(S, E, R, A, Gin, Gout, In)$$

$S=\{s_1,s_2,...,s_n\}$ represents the set of states for the functions under test of the embedded software. At any certain moment, the software is in a unique definite state s_i ($0 \leq i \leq n$) ;

Gin indicates the preconditions of an event, an event can have no or many preconditions, if the event occurs, all preconditions must be met at the same time;

A represents a specific test operation, which is a description of the function to be tested for embedded software. When the input data satisfies the preconditions and trigger state transition, the specific test activity is carried out to produce the change of data during the state transfer process.

Gout, representing the postconditions of the event, is used to verify the execution results of the test. If the postconditions were met, state transition is proved to be successful. An event can have no or many postconditions, and the system may perform different operations according to the postconditions that are satisfied.

$E=\{e_1,e_2,...,e_n\}$ represents the set of events that cause the state transition. It consists of three parameters (Gin, Gout, A), which together form the main part of the formal test model;

R represents the state transition relationship between different states, usually in the form of $S \times E \rightarrow S$, that is, a system state s_i migrates to the state s_j under the condition of satisfying the event e_i;

In represents an invariant in a state transition or a condition that needs to be satisfied all the way during test execution, regardless of the state transition process.

The formal test model mentioned above, can apply for accurate unambiguous description of the invariant, software status, data manipulation, preconditions and postconditions of the embedded software. It also serves as foundation for establishing the formal testing framework of embedded software.

III. FORMAL TEST FRAMEWORK

Complete embedded software testing process based on formal specification includes the formal description of software requirements, the construction of formal test models and formal test scenarios, the link between program under test and test program, the calculation of test coverage and the automatic generation of test reports etc. This paper constructs an embedded formal test framework, organizes the phases above effectively and defines the specific usage method of formal test in embedded software test environment.

A. *Formal testing framework construction process*

The embedded software formal test framework is built through the following steps, as shown in Fig.2.

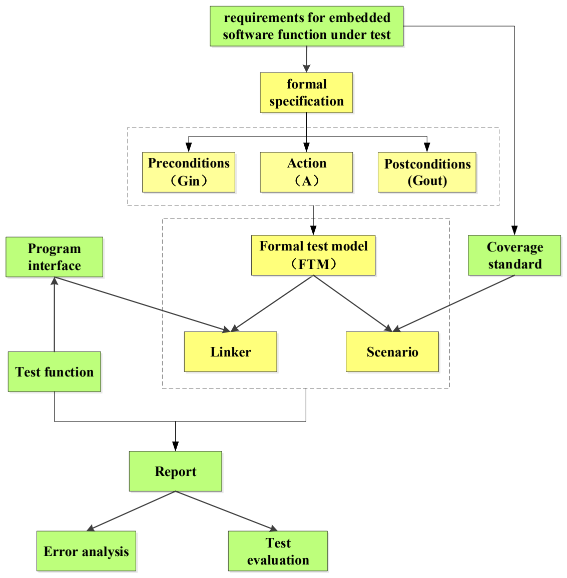

Fig. 2. Formal testing framework

1)Testers formally describe embedded software requirements , and extract preconditions, specific test operations and postconditions, and design a formal test model based on extended finite state machine. In addition, the coverage standards for formal software test are defined in accordance with the quality criterion of embedded software requirements.

2) Develop a set of test scenarios to achieve the required coverage, according to the coverage criteria and the formal test model structure.

3) Extracting interface information of the function to be tested, and design the linker to connect the program under test and the formal test model.

4) Formal test model, linker and test scenarios altogether form a formal test system.

5) Use formal test system to test the function of embedded software. Test system uses linker to link the program under test and the formal test model, and then the filtering of input data and the specific state migration operation are performed in the formal test model. The test system uses postconditions to classify the execution results, instantiate the formal test model

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on June 06,2022 at 07:37:21 UTC from IEEE Xplore. Restrictions apply.

according to the designed test scenario, and automatically generate the corresponding test cases.

6) Execute all the test cases to get the test report. The test report generated should include a coverage analysis to determine whether the expected coverage level has been reached.

Based on the discussion above, this paper defines the embedded test framework as follows:

Definition 2. Formal test framework for embedded software can be defined as a five-tuple:

$$FTF = \{SP, FTM, LIN, SC, RP\}.$$

SP represents the formal specification extracted from the function requirements, and further builds a formal testing model based on the extended finite state machine(FTM).

LIN refers to the link document between embedded software specific function and formal test function. The formal test model corresponding to function under test is defined in the linker.

SC represents a test scenario designed according to formal specification and coverage criteria.

The RP represents the corresponding test report generated after all the test cases have been executed. It is used to show the coverage results achieved by formal tests.

B. Calculate coverage

Test case coverage is one of the most important indicators to measure the adequacy of the formalized testing of embedded software. The framework proposed in this paper first uses the method of equivalence class partitioning to divide all possible input data of the program under test into several equivalence classes. Then, in the process of using the test scenario to instantiate the formal test model, test program will determine which equivalent class each test case belongs to, and calculate the number of triggers for each equivalent class. Finally, the above information is displayed in the test report.

Since the test case is generated when the formal test model is instantiated using the test scenario, the process of distinguishing equivalence classes can be added to the description of the formal test model, as shown in Fig.3. In this way, coverage analysis can be performed during test case execution. After executing all the test cases, tester can determine the test coverage reached by the current test scenario.

```
Testmodel testmodelname(param list) {     // test model
Gin….                                      // preconditions
Action{                                    //action
    if(condition)      //equivalence class partitioning
        return …;
    else
        return …;
    ……..}                 // Specific test operation
Gout….                    //postconditions
}
```

Fig. 3. Formal test model

As shown above figure, the if-else statement is used to describe the different equivalence classes, and the return statements of each branch can return the relevant description or the number of triggers related to the equivalence class. This way realizes the function of coverage statistics in the formal test model.

C. Linker

The linker realizes the link between the program to be tested and the test system, extracts the interface information of both the embedded software to be tested and formal test model firstly, and then uses the linker function to link it. The pseudo code is shown in Fig.4.

```
Linker linkername call testfunctionname (param list)
    {
        testmodelname(param list)
    }
```

Fig. 4. The definition of linker function

Linker keyword indicates linker's name, calls a test function (testfunctionname) in the embedded software and linked it to its corresponding formal test model(testmodelname). The function to be tested has the same parameter list as the related test model, which specifies the input data needed in the execution of the program.

D. Test scenario

The test scenario should generally include a loop body to generate input data for formal test model. When building a test scenario, tester should identify the needed parameters for the formal test model firstly. Then testers should analyze the input range and the data generation strategy for each parameter. Finally, the test system generates input data according to the coverage standard of embedded software, covering as much as possible the equivalence of the program under test. After executing all the test cases, we can count the coverage and trigger times of each equivalence class, and further calculate the test coverage of the current test scenario.

Using the formal test framework described above can build embedded software formal test environment quickly and conveniently. In addition, this framework can reduce the difficulty of applying formal method in embedded security-critical software greatly. In the following, through the UAV flight control software instance to verify the effectiveness of the method.

IV. Instance Verification

When UAV is performing the mission, the flight control system controls its rise and fall. The flight height should be within a specified range. The UAV will judge the ascending command it receives. If the UAV is still within the range of the flight altitude after the ascent operation, the ascending command will be executed; otherwise, the current flight status will be maintained and a warning will be given. The flowchart of the UAV rising function is shown in Fig.5.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on June 06,2022 at 07:37:21 UTC from IEEE Xplore. Restrictions apply.

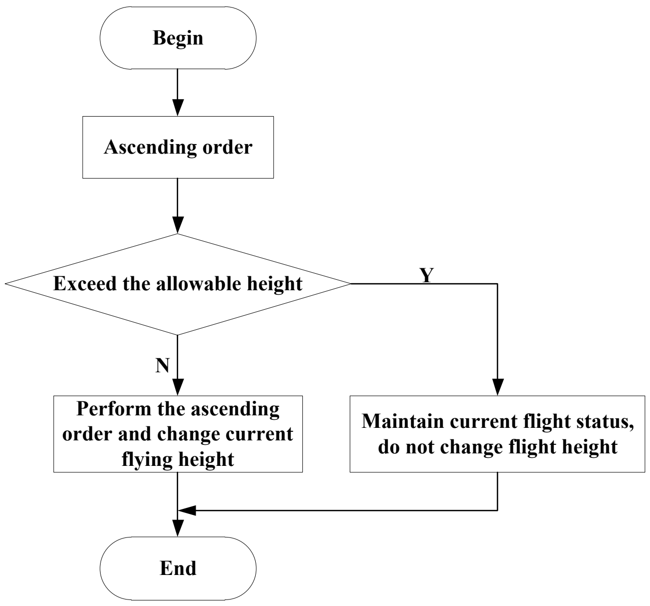
Begin
Ascending order
Exceed the allowable height
Y
N
Perform the ascending order and change current flying height
Maintain current flight status, do not change flight height
End

Fig. 5. UAV rising function

Flight control system has a function ascent(Uav uav, int i) that controls the rise of UAV. The parameter uav is an object of the drone. The parameter i represents the ascent height. The code structure of the function to be tested is shown in Fig.6.

```
int ascent(Uav uav,int i){
      uav.height+= i;
      return uav.height;
}
```

Fig. 6. ascent() function structure

Initstate(Uav uav) is the initialization function of UAV flight state that defines the UAV's initial flight height. The Uavstate(Uav uav) function can obtain the current flight state of the UAV and return the current flight height of the UAV.

```
void initstate(Uav uav){
      uav.height=90;
      //Assuming the initial flight height of the UAV is 90m
}
int Uavstate(Uav uav) {          // State definition
      return uav.height;
}
```

Fig. 7. initstate() and Uavstate() structure

The invariant keyword is used to define the UAV's maximum altitude, namely MaxHeight, which is constant throughout the formal test. In addition, during the test execution, the flight height of the UAV should always be greater than 0m, as shown in Fig.8.

```
invariant int MaxHeight=120;
//The maximum flying height of the UAV is 120m
invariant uav.height>0;
// During the flight, the height of the UAV has been greater
than 0m
```

Fig. 8. invariant conditions

An UAV object and rising height can be specified as a set of input data to be tested. Input the UAV instance(uav) and the rising height(i) as a set of parameter into the formal test model. Using precondition to filter the input data requires that the UAV object(uav) is not empty, the current flight height(height) is greater than 0m, and the rising height(i) is more than 0m. Then, the formal test model will classify the input data. If the UAV reaches the maximum flying height (boundary value) after performing the ascending operation, the ascending operation will be performed and give str1 the message of "Maximum height limit". If the UAV exceeds the specified maximum flight after performing the ascending operation, test model will give str2 of "warning: Beyond the height limit". In the third case, the UAV is still at its normal height after performing the ascent operation, the ascent operation will be performed and give str3 the information of "Normal flight height". Finally, test system determine the height of UAV in postcondition. If the UAV height is equal to the flight height limit, the message str1 will be returned. If the flight height is greater than the flight height limit, str2 will be returned. Otherwise, test system will return the prompt information str3. The formal test model of the UAV rise function is shown in Fig.9.

```
Testmodel uavmodel(Uav uav,int i){
  Gin{(uav.height>0) and (i>0) and (uav!=null)}
  Action{
          if(uav.height+i==MaxHeight){
               str1 = "Maximum height limit";
               uav.height = uav.height+i ;
               }
          else if(uav.height+i>MaxHeight){
               str2 = "warning:Beyond the height limit";
              }
          else{
               str3 = "Normal flight height";
               uav.height = uav.height+i ;
                 }
          }
  Gout{
          if(uav.height+i==MaxHeight){
                return str1;
               }
          else if(uav.height+i>MaxHeight){
                return str2;
               }
          else{
                return str3;
                }
        }
}
```

Fig. 9. UAV rise function's formal test model

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on June 06,2022 at 07:37:21 UTC from IEEE Xplore. Restrictions apply.

The linker realized the link between the function under test and the test program. The linker is named uavlinker, linking the function ascent(Uav uav, int i) and the corresponding formal test model uavmodel(), as shown in Fig.10.

```
Linker uavlinker call ascent (Uav uav,int i)
      {
      uavmodel(Uav uav,int i)
      }
```

Fig. 10. The definition of the Linker

The following code defines the ascent() function test scenario. The initial flight height of the UAV object is set to be 90m, and then test system simulate a rise of 0 ~ 50m operation, instantiating formal test model to generate test cases.

```
scenario uavscenario() {
          Uav uav = new uav();
          initstate(uav)；
   for (int i = 0; i < 50;i+=10)
     uavmodel(uav , i);
  }
}
```

Fig. 11. Formal test scenario

The results of the test are shown in Table Ⅰ and Table Ⅱ.

TABLE I. TEST CASES AND EXECUTION RESULTS

Test cases	Flying height
uav,0	90
uav,10	100
uav,20	110
uav,30	120
uav,40	130

TABLE II. FORMAL TEST COVERAGE RESULTS STATISTICS

Classes of the UAV state	Number of triggers
Maximum height limit	1
warning：Beyond the height limit	1
Normal flight height	3
Test coverage（3/3）	100%

Above is the whole process of testing the rising function of the UAV flight system using formal test model and formal test framework. The experimental results show that the formal test model based on extended state machine can precisely define the requirements of safety-critical software. The results also show that formal test framework can effectively organize the whole formal test process, thus making the formal test process clear and complete. Besides, the framework can achieve coverage statistics and test case automatic generation.

V. CONCLUSIONS

In order to solve the problem that the test method based on the formal protocol is difficult to popularize in the field of embedded, this paper constructs a formal test model based on extended finite state machine to describe the state transition process of embedded software firstly. Then, an embedded test framework based on the formal specification is established to quickly and efficiently build a formal embedded test environment. Finally, the effectiveness of the method is proved by testing the ascent function of UAV flight control system. Based on the previous research results, how to design and develop a formal test tool based on formal test framework to further improve the efficiency of formal testing, how to design a test case automatic generation algorithm to solve the redundancy of test cases is the next direction to be explored.

ACKNOWLEDGMENT

This work was supported by Shanghai Aerospace Science and Technology Innovation Fund, Aerospace Science and Technology Fund Project and the Seed Foundation of Innovation and Creation for Graduate Students in Northwestern Polytechnical University.

REFERENCES

[1] Heimdahl M P E. Safety and Software Intensive Systems: Challenges Old and New[C]// Future of Software Engineering. IEEE, 2007:137-152.

[2] Chen L, Zhao J. Research on formal testing semantics for Statechart specification[C]// International Conference on Information Science and Digital Content Technology. IEEE, 2012:101-105.

[3] Moy Y, Ledinot E, Delseny H, et al. Testing or Formal Verification: DO-178C Alternatives and Industrial Experience[J]. IEEE Software, 2013, 30(3):50-57.

[4] Yang Z, Rozier K Y. Formal specification and verification of a coordination protocol for an automated air traffic control system[M]. Elsevier North-Holland, Inc. 2014.

[5] Huang L, Miao H K, Wang X, et al. Formal Testing Model Generation Based on Scenario[J]. Journal of Shanghai University, 2011, 17(5):595-599.

[6] Wang J, Yin P, Zhang H. Research on Test Case Description Method Based on Z[J]. Computer Measurement & Control, 2013.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on June 06,2022 at 07:37:21 UTC from IEEE Xplore. Restrictions apply.