

Method Based on SEFT-to-Petri for Safety Analysis of Software

Xu Sai-sai^{1,a}, Chen Jing², Sun Yu-ning², Gao Xin-ru², Wang Bo-han² and Wang Kun-long²

¹University of Science and Technology of China, 230026, China

²Institute of Computer Technology and Application of Beijing, 100854, China

Abstract. System safety is a vital non-functional requirement whose satisfaction is essential for system software. However, modern aerospace system software is more and more complicated, which results in a high complexity of analyzing system faults. With the increased acceptance of Model-based Systems Engineering as a new method for systems engineering, Model-based Safety Analysis is also proposed to formalize the task of safety analysis and automate the safety calculations. Our work is grounded on State/Event Fault Tree to analyze system faults and build functional model. Firstly, we can translate SEFT to state machine based on SysML with fault syntactic messages and match elements together with translating logic gates; after which, transforming state machine into Petri Net model by means of rigorous semantic relations to extract preliminary analytical model is deduced theoretically in this paper; finally, we can derive analyses of causes and results of faults from Petri Net model by adopting a set of mathematical and statistical analysis. Practically, we have also validated our work by a case study of an aeronautic control system to support this paper.

1 Introduction

Increasing system complexity results in an increase in complexity of the safety analyst's task to ensure that systems are safe. Thus, how to exactly build models to express a complicated control system and identify the safety-related fragments is an imminent issue in domain of avionics[1]. In theory, system safety uses systems theory and systems engineering approaches to prevent foreseeable accidents and to minimize the result of unforeseen ones. It is a planned, disciplined, and systematic approach to identifying, analyzing, and controlling hazards throughout the life cycle of a system in order to prevent or reduce accidents. Model-based Safety Analysis methods have been developed for formalising the work and subsequent automation of the safety calculation^as. However, these techniques use their own models that are not identical to the design models. Keeping consistency between these models often requires model-to-model transformations.

There exists varieties of difficulties in modelling and analysis, because the nature of a software with high complexity is still not identified clearly. FTA, i.e. Fault Tree Analysis[2], is a conventional modeling technique for describing safety and reliability analysis, which also plays a vital role in domains such as avionics, railways, maritime and automation. Nevertheless, FTA is not propitious to setting up software modelling for three factors as follows[3], (1) FTP consisted of a series of fault sub-

^a Corresponding author : xusaisai@mail.ustc.edu.cn

trees with precise logics is not adequate to indicating a complex component system with its architectures; (2) FTP can reliably apply to domain safety analysis, but it is not a good idea to depict system behaviors; (3) FTP does not put an emphasis on conveying components with multi-state for its restrictive two-state. Although FTP can be not directly used in modelling, it is considerable and mentionable to combine this method with other mature modelling methodologies (e.g., Petri Net model) for quantized analysis.

State/Event Fault Tree (SEFT)[4] is assembled by Fault Tree (FT) and Component State (CS), which is not only able to conduct safety analysis but also build system model by explicating logic gates and elementary component. In this paper, we primarily obtain state machine based on SysML by extending SEFT, which requires logical corresponding relations; then, transforming state machine into Petri Net model is also a significant part of our work; at last, we can perform an automated safety analysis to get feedback and results on our analysis without the need for a safety specialist.

2 Theoretical Principle

This part briefly narrates ideological system of SEFT and discusses techniques of modeling grounded on SysML state machine. Moreover, software safety analysis framework based on SEFT is introduced in the rest of this chapter as well.

2.1 SEFT

Component Fault Tree (CFT) is an integration of both component and fault tree[5], which comprises a large number of independent structures together with system behaviors and fault messages[6]. Figure 1 shows internal elements and their mutual relations of a CFT whose interface is an interactive bridge with external environment. However, it is considerable that a state can not trigger off an event, but can be detonated by an event.

Then, we can transform CFT into SEFT which consists of a set of elements by mapping function, as shown in Figure 2.

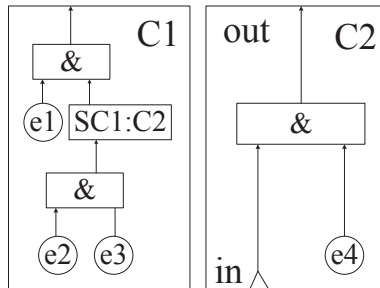


Figure 1. Component Fault Tree

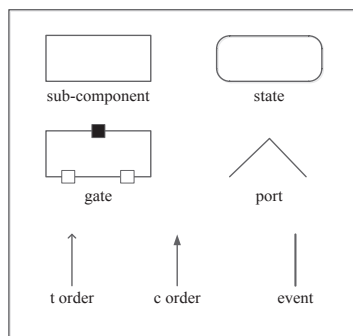


Figure 2. SEFT elements

2.2 SysML State Machine

State Machine[7] describes serial states involved by the instance of a class and a series of responses generated by events based on the object of a class through the whole life cycle of software. What is more, State Machine is a integration consisted of states, events and transitions.

2.3 Petri Net

The essence of petri net[8] indicates the flow of resources, which can build model grounded on real-time system and system function can be reflected by petri net. Software and Hardware Fault Petri Net (SHFPN) is one of Petri Net (PN) models which can be defined as six-tuples $(P,T;F,K,W,M_0)$, when expressing causality between faults and fault modes, especially to its formation mechanism. Moreover, petri net can be depicted graphically, as exhibited in Figure 3 and Figure 4.

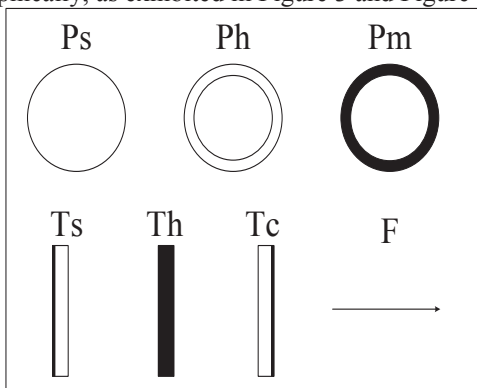


Figure 3. SHFPN elements

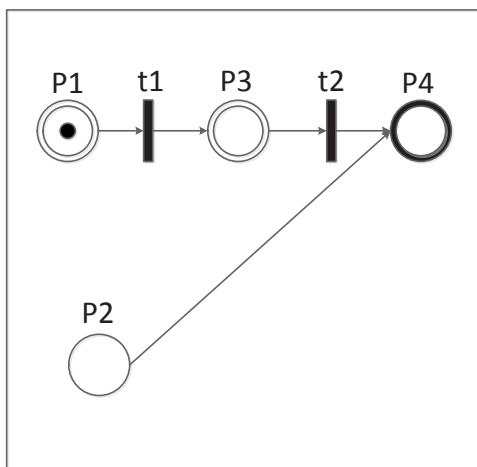


Figure 4. A laconic SEFT model

2.4 Analytical Framework

Existing safety engineering processes are normally based on standards that define a common framework for the derivation of safety requirements which combines hazard assessment and risk analysis techniques[9]. Our contextual framework is given by Figure 5 as below.

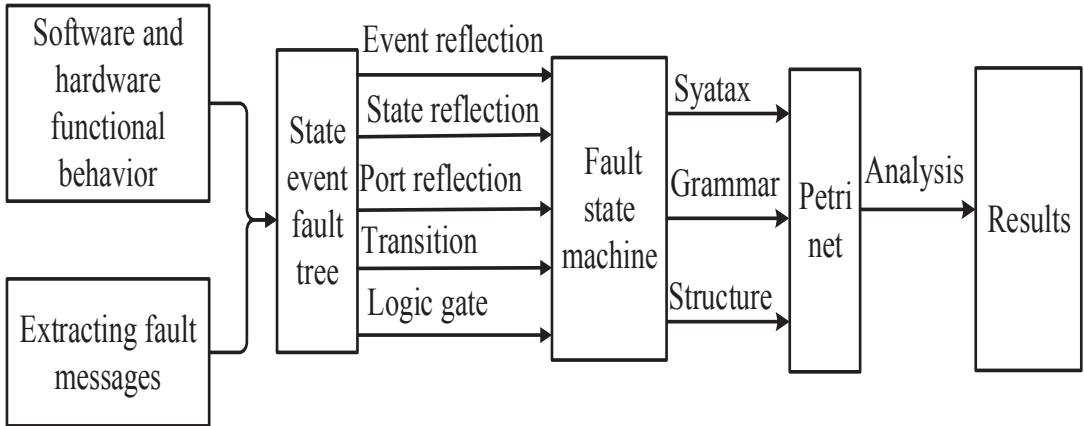


Figure 5. Software safety analysis framework based SEFT-to-Petri

3 TRANSFORMING SEFT INTO SYXML STATE MACHINE

3.1 transition-related algorithms

In order to get an accurate state machine, we have developed a model check algorithm to automatically satisfy a set of conditions which is shown in Figure 6.

```

Algorithmic A: check the transformation conditions
Input: component[MaxC], NestComponent[MaxNC], port[MaXConnectedC], event[MaXConnectedC];
Function: check the transformation conditions
if (is SEFT) then begin
    Boolflag[3];
    Flag [0] = checkNull( component [i] ); //Check component validity
    Flag [1] = checkcycle( NestComponent [j] ); //Check nest cycle
    Flag [2] = checkport( port [k] ); //Check component and environment
    Flag [3] = checke2e( event [m] ); //Check inexistence of transition
    Flag [4] = checkport2port( port [n] ); //Check interaction
    end;
end if;
if ( Flag[0]& &Flag[1]& &Flag[2]& &Flag[3]& &Flag[4]) then begin
    Algorithmic B; Algorithmic C; Algorithmic D; Algorithmic E;
    end;
end if;
    
```

Figure 6. SEFT checking algorithm

The following excerpt is illustrative of reflection about state, event, port and transition to capture state machine from SEFT. There are four algorithms illustrated in Figure 7 delineated as follows via mapping function which is deduced by domain experts. If necessary, more detailed algorithms could be defined to deal with extremely complicated system.

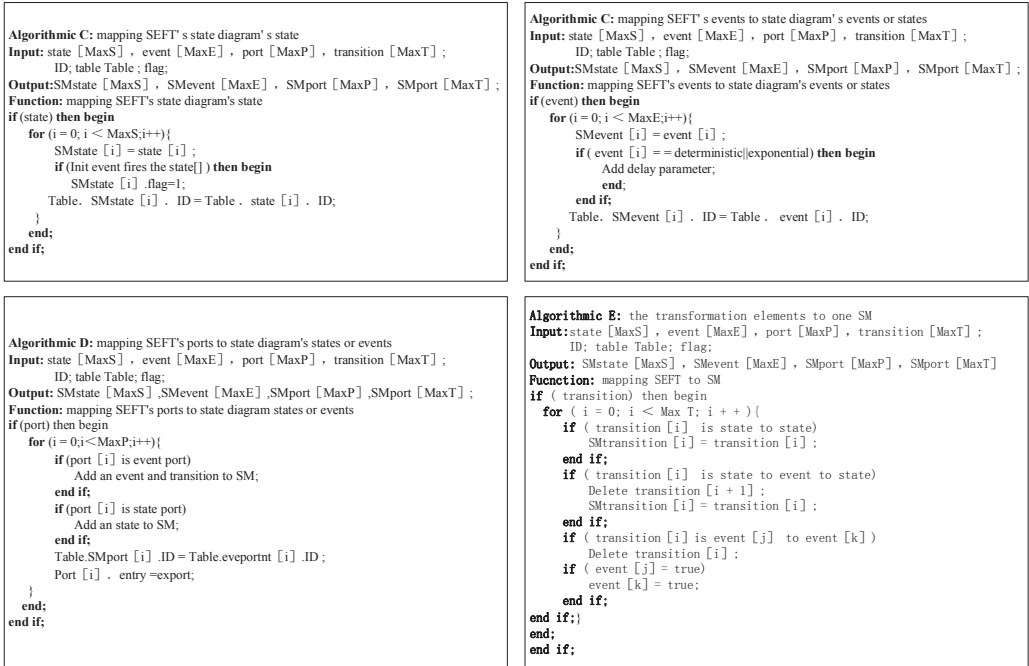
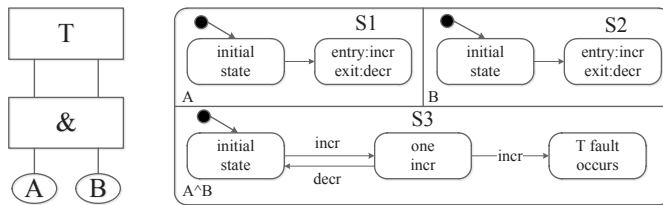


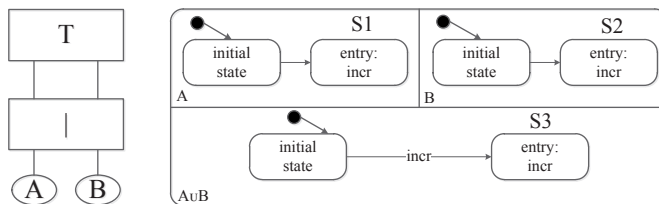
Figure 7. The transformation elements to one SM

3.2 transition from logic gates to state machine

Generally, fault tree includes “AND” gate, “OR” gate and their combination. The rest of this section will describe the conversion between logic gates and state machine. “AND” gate demonstrated as “^” represents interactive parts of several input terminals usually with two input terminals and one output terminal. “OR” gate illustrated as “v” indicates that at least only one event occurs among multiple input events. As is shown in Figure 8, it clearly declares reflection from logic gates to state machine[10].



AND gate and transformed



OR gate and transformed

Figure 8. Logic gates and transformed

In Figure 11, it represents the result of transition from AND gate and OR gate. Here more detailed messages of AND gate transition are interpreted, but the OR gate transition is not given in this section for same manner. For the derivation of the logic gates transition it is assumed that event A and event B are all fault element which must belong to the type of state-occurrence. When component A arrives a state s1, event incr will be detonated and accordingly component A ∧ B will arrive at state s3. However, once component A can't withdraw from state s1 and component B arrives at state s2, event incr will be touched off and component A ∧ B will arrive at state s4, which is consistent with Fault Tree in semantic categories.

4 Transition from SysML State Machine to Petri Net Model

Software and hardware fault petri net (SHFPN) [11] is another mode of state machine which is defined by six-tuples via mathematical constraints, i.e., SHFPN = (P, T; F, K, W, M0). (1)P expresses fault state set, namely $P = P_S \cup P_H \cup P_M$, P_S is software fault tree set, namely $P_S = \{P_{S1}, P_{S2}, \dots, P_{Sa}\}$, P_H represents hardware fault tree set, namely $P_H = \{P_{H1}, P_{H2}, \dots, P_{Hb}\}$, P_M illustrates fault mode set. (2) T demonstrates event transition set, namely $T = \{t_1, t_2, \dots, t_m\}$; for $t \in T, \exists p \in P_S$, get $p \in t^*$, which means software fault event, i.e., T_S ; for $t \in T, \exists p \in P_H$, get $p \in t^*$, which shows hardware fault event, i.e., T_H ; for $t \in T, \exists P_1 \in P_H \exists P_2 \in P_S$, get $P_1, P_2 \in t^*$, which stands for software and hardware fault event, i.e., T_C . (3)F defines as the flow of SHFPN, namely $F = P \times T \cup T \times P$. (4)K depicts the token, confirming whether an event occurs. (5)W shows the weight coefficient constrained by $W_{(p,t)} \equiv 1$. (6)M0 is defined as the initial fault state.

Additionally, for $p \in P_M$, if $t \in p^*$ exists and $t \in T_S \wedge t \text{ not } \in T_C$, we will name it software fault mode. Moreover, for $p \in P_M$, if $t \in p^*$ exists and $t \in T_H \wedge t \text{ not } \in T_C$, we will name it hardware fault mode. To transform fault state machine into petri net in Figure 12, we need further to perform the reflection of state-to-state and event-to-event[12].

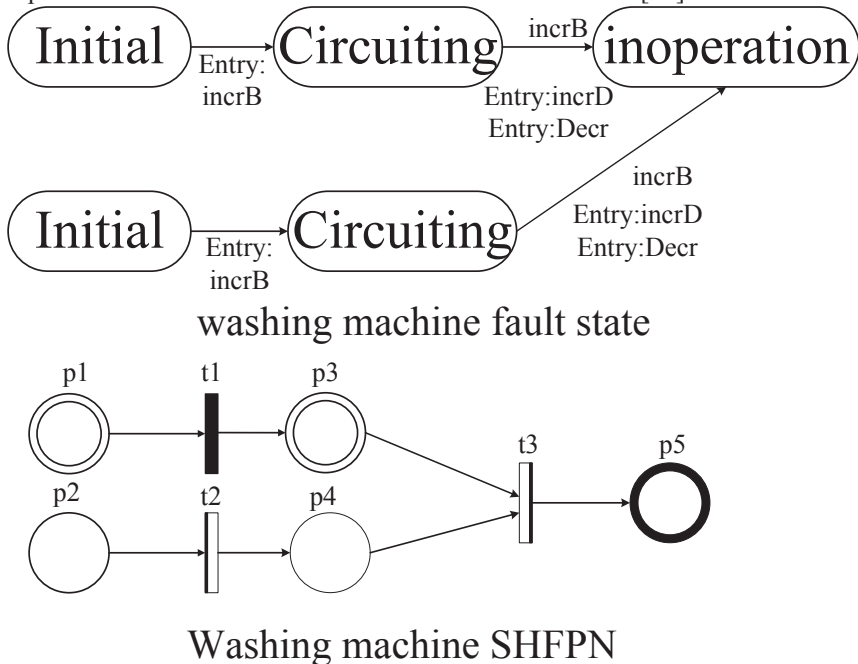


Figure 9. Transition from fault state machine to SHFPN based on washing machine

As is shown in Figure 9, an automatic washing machine often malfunctions or breaks down by hardware circuit fault and software program error which are described as corresponding states[13]. Besides, the event (such as incrB) is the trigger from one state to another state in the state machine above. According to consistency, it is considerable and reasonable for us to propose an algorithm depicted in Figure 10 fulfilling the transition from fault state machine to Software and Hardware Fault Petri Net (SHFPN).

Algorithmic F: mapping state diagram's state to SHFPN's state

Input: SM-state[Max S], SM-event[Max E]

Output: PNS-state[Max S], PNH-state[Max S], PNC-state[Max S],
 PNS-event[Max E], PNH-state[Max E], PNC-state[Max E];

Function: transforming state diagram's state to SHFPN's state

```

If(state) then begin
    for(i=0, j=0, k=0, m=0; i<Max S; i++){
        if(state[i] instanceof software fault state) then begin
            PNS-state[j] = state[i];
            j++;
            end;
        end if;
        if(state[i] instanceof hardware fault state) then begin
            PNH-state[k] = state[i];
            k++;
            end;
        end if;
        if(state[i] instanceof hardware fault state) then begin
            PNC-state[m] = state[i];
            m++;
            end;
        end if;
    }
    end;
end if;
    
```

Algorithmic G: mapping state diagram's state to SHFPN's state

Input: SM-state[Max S], SM-event[Max E]

Output: PNS-state[Max S], PNH-state[Max S], PNC-state[Max S],
 PNS-event[Max E], PNH-state[Max E], PNC-state[Max E];

Function: transforming state diagram's state to SHFPN's event

```

If(event) then begin
    for(i=0, j=0, k=0, m=0; i<Max E; i++){
        if(event[i] instanceof software fault state) then begin
            PNS-event[j] = event[i];
            j++;
            end;
        end if;
        if(event[i] instanceof hardware fault event) then begin
            PNH-event[k] = event[i];
            k++;
            end;
        end if;
        if(event[i] instanceof hardware fault event) then begin
            PNC-event[m] = event[i];
            m++;
            end;
        end if;
    }
    end;
end if;
    
```

Figure 10. Mapping algorithm from SM to SHFPN

Figure 10 demonstrates that a mapping algorithm from State Machine (SM) to Software and Hardware Fault Petri Net (SHFPN) describes state reflection and event reflection[14]. Because SHFPN consists of three states, i.e., software fault state, hardware fault state and software-hardware fault state, more considerable focus is based on identifying these three states, as shown in the left part of Figure 10. Of course, SHFPN also contains three events, namely software event, hardware event and software-hardware event, which is extracted shown in the right part of Figure 10. So far, we have accomplished the transition from fault/event tree to petri net[15]. If you want to further analyze causes of faults, you can go far with existing theory of petri net.

5 Case Study

Automatic washing machine mainly contains software and hardware, the requirement of programmed control system[16] can be described as follows:

- (1) In any case, washing machine's sheathing material should be insulated or the automatic washing machine should be isolated from other materials.
- (2) When required, washing machine should work normally.
- (3) When not required, washing machine must stop working.

According to this, fault analysis of washing machine is defined as a fault tree in Figure 11. As is illustrated in Figure 14, electrical faults in the electronics may lead to breakdown; Besides, errors of control block can generate system fault which will cause in-operation[17].

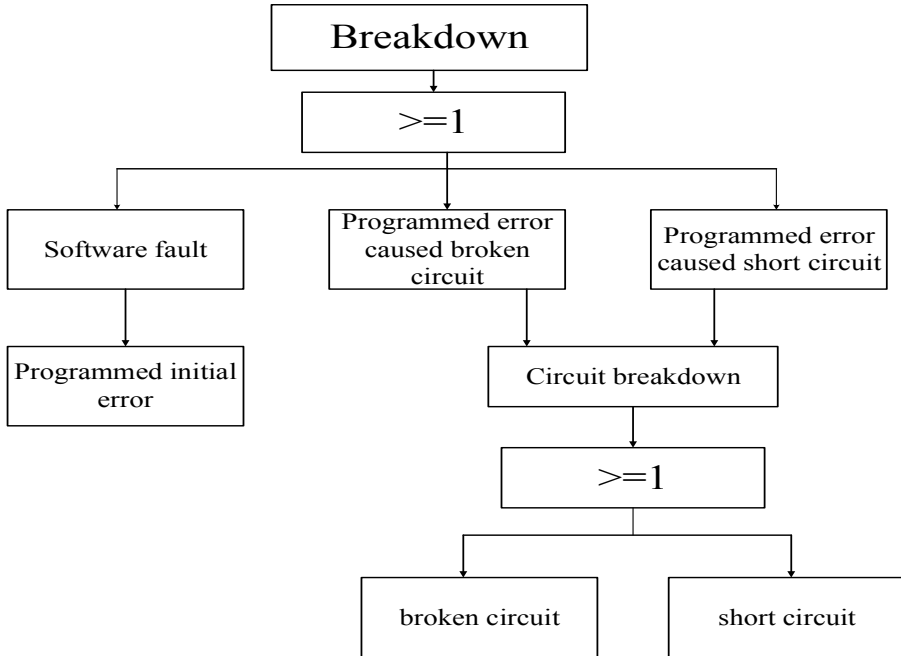


Figure 11. FT of washing machine

We can build the State/Event Fault Tree (SEFT) depicted in Figure 12, which graphically displays fault messages and related functional behavior [18].

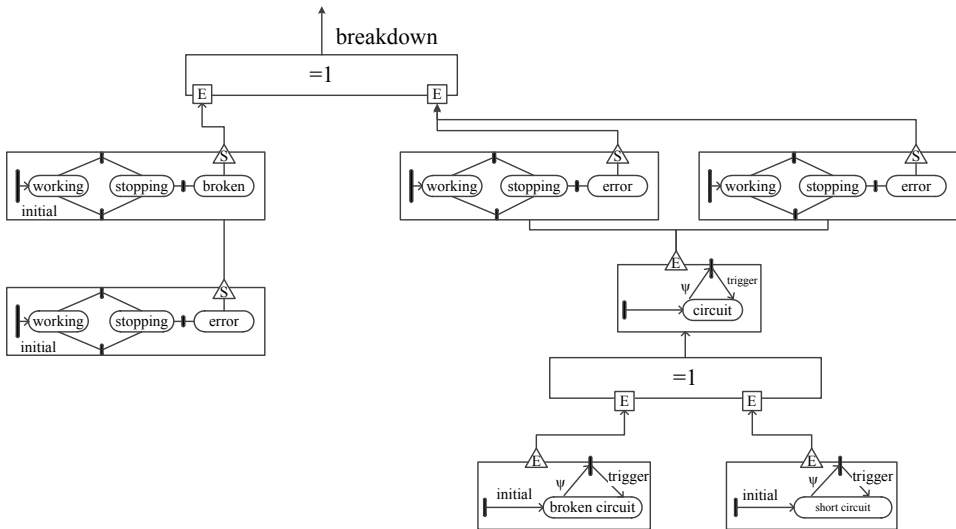


Figure 12. SEFT of washing machine

According to the mapping algorithm demonstrated in Figure 7 and the transition of logic gates, we can transform SEFT into Fault State Machine (FSM) illustrated in Figure 13.

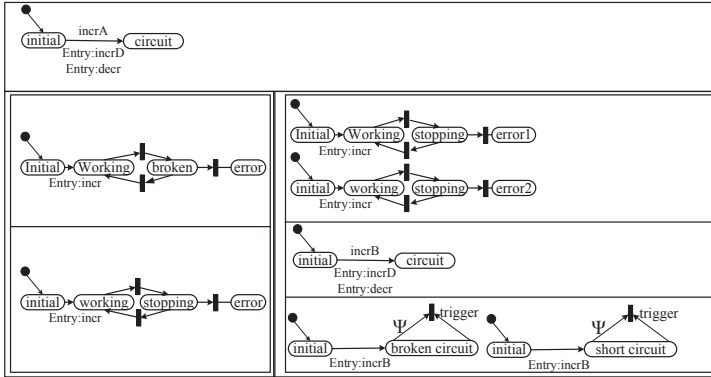


Figure 13. Fault SM of automatic washing machine

After that, we can get corresponding Software and Hardware Fault Petri Net (SHFPN) grounded on algorithm depicted in Figure 14.

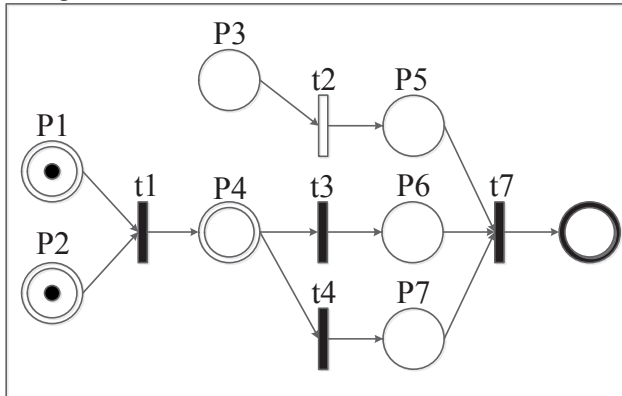


Figure 14. SHFPN of automatic washing machine

The following job is to obtain SHFPN’s incidence matrix based on methodology of petri net as shown in Figure 15.

$$A1 = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Figure 15. Incidence matrix of SHFPN

Result analysis:

We can get related analysis results, i.e., $P_{MC} = \{P_9\}$ which is a mode of software-hardware from corresponding incidence matrix above. In addition, we also have analyzed the fault mechanism whose results are described in Table 1.

Table 1. Fault mechanism analysis of P_9

Number	Fault causes	Means of transmission
1	P_1	t_1, t_3, t_7
2	P_2	t_1, t_4, t_7
3	P_3	t_2, t_7

6 Conclusion

Existing modeling language (such as UML and SysML) can clearly describe software system behavior and minutely express the change of system state, but their common defect is that they can't perform safety analysis. However, fault tree can express system fault behavior commendably, yet it is incapable of conveying functional behavior of system. Besides, methodology of petri net has been studied maturely, which has a set of holonomic theory of fault analysis. Therefore, it is complementary to combine fault tree with state machine based on SysML and petri net.

This paper firstly introduces the related theory providing basis of analysis. In addition, we talk about the fault tree and transition from fault tree to State/Event Fault Tree (SEFT), which is the start point of our subsequent work. Additionally, mapping algorithm from SEFT to FSM is significantly described as above. Last but not least, transforming FSM into SHFPN is the innovative point whose executable algorithm is displayed in Figure 13. Of course, if necessary, we can also take efforts to combine Markov Chans (MC) [19] with probability theory used in our paper which can enable us to analyse safety faultlessly.

References

1. C. Huo Wang, W. Ji, D. Wei, *Acta Electronica Sinica*, **31**, 12A (2003)
2. N. H. Roberts, W. E. Vesely, *Fault tree handbook*, (Government Printing Office, 1987)
3. B. Kaiser, C. Gramlich, *Reliability Engineering & System Safety*, **92**, 11 (2007)
4. B. Kaiser, *State event trees: a safety and reliability analysis technique for software controlled systems* (2007)
5. S. Mahmood, R. Lai, Y. Soo Kim, *Information and Soft-ware Technology*, **47**, 10 (2005)
6. B. Kaiser, P. Liggesmeyer, *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software*, **33**, Australian Computer Society, Inc. (2003)
7. M. Hause, *Fifteenth European Systems Engineering Conference*, **9** (2006)
8. M. Xiao Dong, Z. Peng, *Chinese Journal of Computers*, **6**, 1 (2015)
9. M. Jing, Z. Li, M. Shao Dong, *International Information and Application Forum*, **36**,4 (2009)
10. X. Jing Wen, Z. Ji En, S. Yue Yue, *Computer Application and Software*, **24**,22 (2009)
11. Z. Yan, M. Hong, *Journal of Soft-ware*, **20**, 6 (2009)
12. D. Wei, W. Ji, Q. Zhi Chang, *Journal of Software*, **14**, 4 (2003)
13. Z. Hong Lin, Z. Chun Yuan, L. Dong, *Chinese Journal of Computers*, **35**, 2 (2012)
14. P. Feiler, A. Rugina, *Carnegie-mellon UnivPittsburgh Pa Software Engineering Inst* (2007)
15. P. Fenelon, J. A. Mc Dermid, *Journal of Systems and Software*, **21**, 3 (1993)
16. M. Hecht, A. Lam, and C. Vogl, *In Engineering of Complex Computer Systems*, 16th IEEE International Conference on (2011)
17. K. M. Hansen, A. P. Ravn, H. Rischel, *ACM SIGSOFT Software engineering Notes*, **16**, 5 (1991)
18. W. Si Qi, H. Zhi Qiu, *College of Computer Science and Technology*, (Nanjing University of Aeronautics and Astronautics, 2016)
19. Y. Jin, L. Xun, A. Jian xun, Y. Guang Yu, L. Ping, *Journal of University of Chinese Academy of Sciences*, **1**,4(2013)