Research on Verification for STPA-Based Avionic System Software Safety

Yuan Sun[1, a], Jingguo Tang[1, b], Haifeng Yang[2, c]

[1]Scientific Research Department, Naval Aeronautical and Astronautically Engineering University, Shandong 264001 China

[2]Shandong Institute of Space Electronic Technology, Shandong 264001 China

[a] sunyuan1208@126.com, [b]fsm_tjg@sina.com,[c]xdyhf@126.com

Keywords: STPA, software safety, verification, Introduction.

Abstract. Software safety problems resulting from relevant faults are increasingly highlighted as systems become more and more complex. Thus, the static verification method is inapplicable to complex system. This paper adopts System-Theoretic Process Analysis (STPA) to identify hazards in system, and obtain software-relevant safety needs. Safety verification adapting for complex system is clarified with the combination of STPA and model test software safety analysis and verification. Analysis and research adopting STPA method are conducted and their feasibility are proved.

1. Introduction

The importance of software safety is becoming more and more serious with the complexity of the system. Many safety problems, even casualties are caused by software faults. Code errors are not the only reason for these accidents. Software incompletion and other faults are also responsible for the loss. Therefore, software safety is a systematic problem, which must be solved with systematic ways. STPA[1] served for this way, which emphasizes component deficiency and hazards resulting from interactions of parts of the system.

2. Verification Method for STPA-Based Software Safety

2.1 STPA

STPA, a danger analysis technology derived from Accident Causality Model Based on System Theory (STAMP), deals with safety problems systematically. STPA can be applied in the early phase of system development, or before designations of advance safety needs and restrictions. Compared with traditional safety analysis technology based on reliability, STPA has more strengths on identifying scenarios of multi-factors and dangers. In terms of software, STPA is aimed to recognize insufficient controls which may lead to dangers, and to ensure relevant safety restrictions on acceptable risks. Besides, relevant information concerning violations of safety restrictions can be obtained. These information can be adopted to control, decrease and eliminate dangers in the system designation and processing [2].

2.2 Comparison Between Different Methods

Currently, common software safety analysis and verification technology consists of Preliminary Hazard Analysis (PHA)[3], Failure Mode and Effects Analysis (FMEA), Fault Tree Analysis (FTA). These technologies analyze and verify software safety from different angles and ways. However, they all have limitations respectively.

Firstly, PHA is based on engineering experience data base, without which would lower the performance of PHA. FTA is driven by events. It is impossible to define certain event and derive unified results by applying FTA because of the large amounts of events, complex logic relations and interactions of events in weapon system software. FMEA suits for pure software function analysis. Thus system safety restrictions are hard to derive. Secondly, preconditions for traditional software safety analysis and verification method are based on components failures. For the safety of software, analysis are conducted after the designation of software. These cannot basically meet the demands of complex software safety analysis.

Compared with traditional ways, STPA, an iteration way, can analyze software safety with the combination of satisfying advanced system, safety restrictions and system analysis without using complete engineering experience database [4], never be apart from process and system. Therefore, STPA can guarantee software safety better when applying to complex weapon system.

Table 1. Comparison between the Four Methods

name	Scope of application	Applicable state
STPA	No need for complete engineering experience data base	Dynamic iterative
PHA	Based on engineering experience data base	static
FTA	Event-driven, applicable to simple interactive system	static
FMEA	Applicable to pure software function analysis, hard to derive system safety restrictions	static

3. Verification Procedures for STPA-Based Software Safety

Normally, software verification is designed for proving correctness of functions [5] and whether software can meet demands of overall functions. However, software safety cannot be guaranteed by only verifying software correctness. Thus, it is essential to conduct analysis on software safety and to verify it accordingly [6].

Whether the software suits or satisfy safety restrictions is the presupposition of safety verification [7]. Safety verification is different from function verification. The former aims to guarantee software safety by applying safety analysis results; the latter software functions [8].

Verification for STPA-based software safety is conducted as Figure 1:

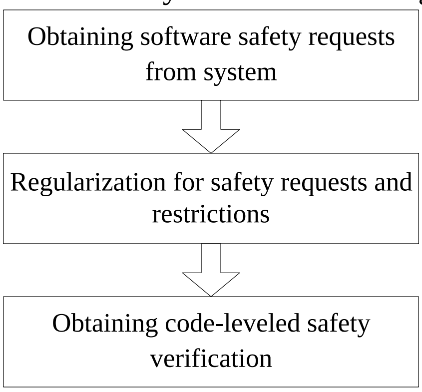

Figure 1. Verification procedures for STPA-based software safety

Step 1 Obtaining software safety requests from system

This procedure is designed for conducting software analysis in system, and for identifying potential software hazards which may lead to accidents. As it is showed in the following steps:

(1) Identify all potentially hazardous software control measures;

(2) Evaluate whether each CA has four kinds of normal-type hazardous behavior so as to identifying Unsafety (UCAs): (1) not providing CA; (2) providing UCA; (3) over early, over late, or unqualified potential CA; (4) early halt or over lasted CA.

(3) Transform the definite UCAs to informal text software safety requests (SSR);

(4) Understand how each UCAs occurs by using process model and its variables;

(5) Clarify each combination of process model variables of UCAs. Evaluate each combination in Ci=providing CA or no providing CA. Make it clear whether CA is under hazard in respective environment.

Note: CA can be considered as hazardous in system if combination of process variables are only relevant to CA and cause system-leveled hazard $Hi \in HZ$

(6) Improve software safety by applying Boolean operator ∨ and ∧ using results derived from step 5.

Note: The output of this step is a table of relevant software safety requests. Each software safety requests ($SSR_{i,j}$) has a $UCA_{i,j}$ counterpart. i stands for the serial number of CA, while j quantity of UCAs.

Definition 1:

Define $PMV_{i,j}=\cup(P_{i,1}\ldots, P_{i,n})$as the set of process model variables relevant to CA_i, leading to hazard $i \in HZ, C_1=\{providing\ CA\}orC_1=\{\ not\ providing\ CA\}$, n stands for the maximum quantity of relevant variables. Ω is the combination of $PMV_{i,j}$ values. Thus $SSR_{i,j}$ can be represented as follows:

$$SSR_{i,j}=(\Omega PMV_{i,j}\rightarrow CA_i)\vee SSR_{i,j}=(\Omega PMV_{i,j}\rightarrow\neg CA_i) \quad (1)$$

This means CA_i is not available in $PMV_{i,j}$ combination Ω value.

Procedure 2: regularization of safety request and restriction

Clarify the safety request. Linear-time temporal logic (LTL) must be applied to formalize these safety requests so as to verifying them in the next step. These requests can be reflected to the formalization of LTL for verification through model check if relevant software safety requests are identified and presented by using Boolean operator. LTL formula can be defined in a set of atomic assumption, Boolean operator (¬, ∨, ∧, ↔, →, true, false) and time operator (○next, □always, ◊eventually, U until, R release). Safety requests make sure no hazardous behavior occurs in the process of conducting.

Definition 2:

Take $SSR_{i,j}$ as software safety request, all software execution route must be like this from the beginning to the end. Thus $SSR_{i,j}$ LTL formula is as follows:

$$\Phi=\Box(\ SSR_{i,\ j}\ ) \quad (2)$$

in which

$$SSR_{i,j}=(\Omega PMV_{i,j}\rightarrow CA_i)\vee SSR_{i,j}=(\Omega PMV_{i,j}\rightarrow\neg CA_i)$$

□ represents all status of executive route.

This formula means the emergence of $PMV_{i,j}$ will always cause the fact that software must (or cannot) provide CA_i, i.e.:

$$\Phi=\Box(\ PMV_{i,j}\rightarrow CA_i)\quad \text{or} \quad \Phi=\Box(\ PMV_{i,j}\rightarrow\neg CA_i) \quad (3)$$

According to the above definition, safety request defined by safety analysis can be easily transformed into LTL.

Procedure 3: obtaining code-leveled safety verification

Based on procedure 2, this procedure is designed for regularization of code-leveled safety request verification. This procedure can be accomplished through two different ways: 1. conducting formal verification by using model inspector [9]; or 2. Generating 3 test samples by using model inspector. Model inspector take software model and relevant quality as input, and write into time logic, then test the overall status space effectively.

4. Real Example of Avionic System

Take Avionic equipment operating system software for research example in order to analyze safety request for software system structure design. Avionic system, an essential part of modern aircraft, is closely related to aircraft combat performance. It is reported that avionic system failure may cause catastrophes even casualties. Normally, operating system of avionic equipment must meet needs of hundreds of safety requests. Table II shows safety request subset in research example of avionic equipment operating system.

Table 2. Safety request subset in research example of avionic equipment operating system.

Request subset	Illustration
Display aircraft altitude data	Altitude data here is defined as aircraft altitude above the sea level. Altitude data is applied to ground collision test system. Pilot needs to pay attention to altitude data.
Display aircraft position data	Position data here refers to aircraft longitude and latitude coordinates received from GPS. Aircraft position data is displayed together with other points. Pilot can watch the route deviation and take actions according to it.
Display aircraft gyro data	Gyro data refers to the relative orientation of the airplane annex with respect to ground angles, including pitch angle, yaw angle and roll angle.
Display fuel data	Fuel quantity refers to the overall quantity of fuel available in aircraft fuel tank.

UML graph can be adopted to illustrate above issues as it is showed in Graph 2. Generally speaking, the operating system of avionic equipment consists of display system G, platform system P and navigation system N. G is for receiving aircraft fuel data (Fuel) provided by fuel sensor; N altitude data (Alti), aircraft gyro data (Gyro) and aircraft position data (GPS), provided by altitude meter, gyroscope and GPS respectively; G displaying P, C, N. G1and G2 read and display aircraft altitude, attitude, position and fuel.

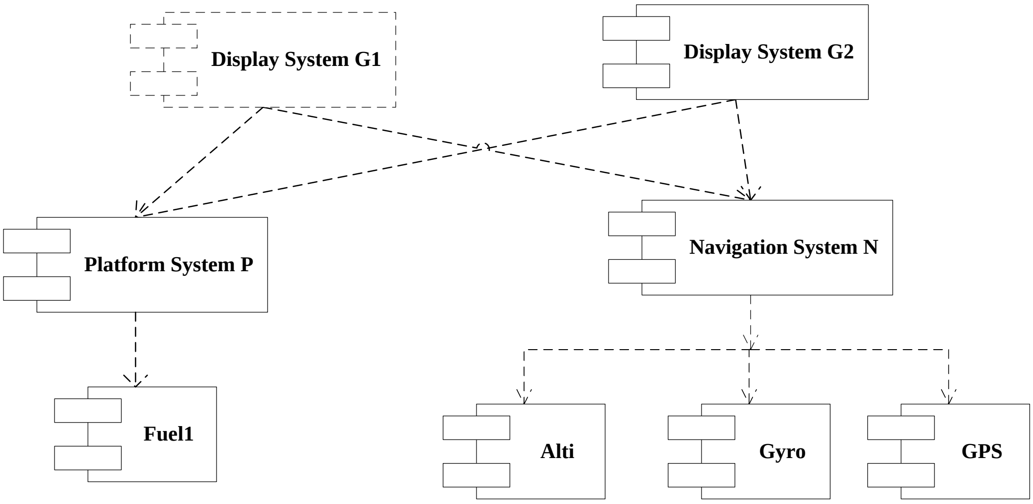

Figure 2. UML graph for relations between operating system of avionic equipment

Lists factors that may cause hazards (HZ). Software safety focus on those hazard-causing defaults and relevant design bugs and exterior input conditions [10,11]. From HZ_1 to HZ_4, these factors is proved to be disastrous, leading to air crash. For example, pilot judgments to altitude are interfered with incorrect altitude data, causing ground collision, air crash, casualties, system loss and damages to environment.

Table 3. Hazard identification in case research

Serial number	Code name	Hazard	Possible Reasons
1	HZ1	Display false altitude data	Altitude meter data loss or fault, disconnection or false connection to altitude meter, display errors.
2	HZ2	Display false position data	GPS data loss or fault, disconnection or false connection to GPS, display errors
3	HZ3	Display false gyro data	Gyroscope data loss or fault, false connection to gyroscope, display errors.
4	HZ4	Display false fuel quantity	Fuel sensor data loss or fault, false disconnection or false connection to fuel sensor, display errors.

(1) Software CA: avionic system software has 4 CAs. The operation for each CA depends on four kinds of records of general hazard type.

UCA1.1: one data loss or fault; UCA1.2: connection time and sequence fault; UCA1.3: data displaying stops on certain readings.

(2) UCA: Evaluate each item mentioned above, and check reasons which cause hazard. If the item is hazardous, relevant HZ should be distributed for system-leveled hazard, *vice versa*.

(3) Advanced safety request: Each UCA will be transformed into relevant safety restriction. Table IV shows relevant safety restrictions of UCA.

Table 4. Relevant safety restriction

Related UCAs	The corresponding security constraints
UCA1,1	$SSR_{1.1}$ no altitude, position, gyro, and fuel data loss and correct
UCA1.2	no connection time and sequence delay
UCA1.3	dynamic and real-time changing data displaying

4) Process model variables: process model variables mainly include fuel monitor, Gyro monitor, GPS monitor, Alti monitor, graphics monitor, plane state.

5) Analyzing causes of UCAs

Table 5. Causes of UCAs

Control measures	Process model variables						Hazard CA
	Fuel_ monitor	Gyro_ monitor	GPS_ monitor	Alti_ monitor	Graphics_monitor	CA	
Aircraft State (CA1)	False	True	True	True	True	providing	UCA1,1
	True	False	True	True	True	providing	
	True	True	False	True	True	providing	
	True	True	True	False	True	providing	
	True	True	True	True	False	providing	UCA1.2
	True	True	True	True	Null	Providing	UCA1.3

6) Construct relevant safety request:

$$SSR_{1,1}=\square(\ PMV_{1,1}\rightarrow\neg CA_1) \tag{4}$$

in which $PMV_{1,1}=P_{1,1}\wedge P_{1,2}\ \wedge P_{1,3}$

$P_{1,1}=\{(Fuel\_monitor\vee Gyro\_monitor\vee GPS\_monitor\vee Alti\_monitor\vee Graphics\_monitor)=false\}$;

$P_{1,2}=\{Graphics\_monitor=false\}$;

$P_{1,3}=\{Graphics\_monitor=null\}$;

$CA_1=true$

According to Definition 2, map system-leveled STPA software safety request as formal format in LTL. The formal format of LTL software safety request can be sampled as follows:

$$SSR_{1,1}=\square((\ Fuel\_monitor\&\&Gyro\_monitor\&\&GPS\_monitor\&\&Alti\_monitor\&\&Graphics\_monitor)\ =false\rightarrow\neg(\text{plane\_state})); \tag{5}$$

$$SSR_{1,2}=\square(\ Graphics\_monitor=false)\rightarrow\neg(plane\_state); \tag{6}$$

$$SSR_{1,3}=\square(\ Graphics\_monitor=null)\rightarrow\neg(plane\_state); \tag{7}$$

By adopting SPEC-test program, verification requests of system model and tense logic formula presented by module code are illustrated as follows. In it, plane-state , an enum type, has 2 values: normal and abnormal.

SPEC

G((Fuel_monitor&&Gyro_monitor&&GPS_monitor&&Alti_monitor&&Graphics_monitor)==false

```
->!plane_state=normal);
    SPEC G(Graphics_monitor==false->!plane_state=normal);
    SPEC G(Graphics_monitor==null->!plane_state=normal);
```

5. Summary

Given that traditional software safety can hardly be adapted to complex system safety analysis and verification, this paper conducted system hazard analysis applying STPA and derived software relevant safety request in the background of avionic system. Mapping those software-related safety requests as formal logic formula and verifying their code-leveled safety provide referential experience for further the application of STPA to weaponry and equipment. However, analysis process depends on artificial analysis. Automatic tools and ways are in the further development list.

References

[1]. Leveson, N.G.: Engineering a Safer World: Systems Thinking Applied to Safety.Engineering systems. MIT Press (2011):23-27

[2]. Xu Yan et al: Analysis of software safety based on system theory process analysis[J].computer Application,2013,33(S2):238-240

[3]. ORTMEIER F, SCHELLHORNG.Formal fault tree analysis-practical experiences[J].Electronic Notes in Theoretical Computer Science,2007,185:139-151

[4]. Xu Xingxian et al, Research and Application of Aeroengine Control Software Safety Based on STPA[J]. Proceedings of 2016 IEEE Chinese Guidance,Navigation and Control Conference,August 12-14,2016:2592-2594

[5]. Tracey, N., Clark, J., McDermid, J., Mander, K.: Integrating Safety Analysis with Automatic Test-Data Generation for Software Safety Verification. In: Proceedings of the 17th International Conference on System Safety (1999):54-56

[6]. Black, P.E.: Test Generation Using Model Checking and Specification Mutation. IT Professional 2014.16(2), 17–21

[7]. Leveson, N.G.: Safeware: System Safety and Computers. ACM, New York (1995):23-26

[8]. Hardy, T.L.: Essential Questions in System Safety: A Guide for Safety Decision Makers.AuthorHouse (2010):101-105

[9]. SAE: Society of Automotive Engineering, Adaptive Cruise Control Operating Characteristics and User Interface, SAE J2399 (2003):45-48

[10]. Li Xueren.Military Software Quality Management[M].National Defence Industry Press,2012:67-69

[11]. Leveson,N.G. Safeware:System Safety and Computers.Addison-Wesley,NY,1995:19-23