Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 9185910, 15 pages
https://doi.org/10.1155/2019/9185910

Research Article

Verifying the Safety of Aviation Software Based on Extended Colored Petri Net

Hang Zhou, Canheng Zhang, Yue Li, Yang Gu, and Shikang Zhou

College of Civil Aviation, Nanjing University of Aeronautics and Astronautics, Nanjing 211100, China

Correspondence should be addressed to Hang Zhou; zhh@nuaa.edu.cn

Received 3 May 2018; Accepted 18 February 2019; Published 7 April 2019

Academic Editor: Paolo Spagnolo

At present, various functions of aircraft have become more and more dependent on airborne software system, and the structure of modern airborne software is extremely complex. Engineers need to eliminate the situation in which the safety performance of the entirety reduces caused by mutual influence among components. This paper presents a comprehensive method with high efficiency for safety verification of airborne software system, in order to ensure the system meet safety requirement of airworthiness standard at the `design` stage. Safety Verification Colored Petri Net (SVCPN) for software safety verification is firstly proposed, and then the mapping transformation rules from Block Definition Diagram (BDD) of System Modeling Language (SysML) to SVCPN are proposed to achieve the accurately formal description of software system. Traversing all delivery paths of safety level transfer based on the Reachable Tree Diagram, to detect the components that do not meet the safety requirement of airworthiness standard. Based on the disambiguation algorithm, the fundamental components that cause safety problem are found out through the establishment of antinet to achieve the safety level reassign, ensuring the safety performance of the whole system. Finally, the case study and the comparison and analysis are applied to show the feasibility and superiority of our method.

1. Introduction

With the development of aviation industry and the advancement of major craft and UAV projects, the application of embedded software system has been continuously deepened. Almost all important functional systems in modern aviation are related to embedded software. The diversity and complexity of embedded software enhance the complexity of system, and the influences of software system occupy an important position gradually. The reliability of the software is much lower than the hardware, and it has a great negative impact on the safety of embedded software. Recently, accidents caused by software failures are common, such as the problem system that leads to the system failure of Japan's F22 "raptor" stealth fighter [1]. An unmanned helicopter belonging to the US navy broke into the no-fly zone over Washington after losing control, and the accident is also caused by software failure [2]. Thus, FAA requires that both the airborne software system and the system that is linked to the aircraft must be certified, or otherwise not allowed to be used. Verifying the embedded software and further ensuring it meets the requirements of airworthiness safety certification have become a very important issue in the embedded software system designation.

Modern airborne software is typical embedded component systems, while both aspects of architecture and function are more prone to security issues. It may directly cause undesired loss or harm to human life, property, or the environment or control some equipment that may cause accidents. Accordingly, to minimize the probability of software failure, it is urgent to enhance the safety performance of airborne software system.

This paper presents a comprehensive method with high efficiency for safety verification of airborne software system combined the standard DO-178C, in order to ensure the system meets safety requirement of airworthiness standard at the design stage. Safety Verification Colored Petri Net (SVCPN) for software safety verification is firstly proposed, and then the mapping transformation rules from Block Definition Diagram (BDD) of System Modeling Language (SysML) to SVCPN are proposed to achieve the accurately formal description of software system. Traversing all delivery paths of safety level transfer based on the Reachable Tree

Diagram, to detect the components that do not meet the safety requirement of airworthiness standard. Based on the disambiguation algorithm, the fundamental components that cause safety problem are found out through the establishment of antinet to achieve the safety level reassign, ensuring the safety performance of the whole system. Finally, the case study and the comparison and analysis are applied to show the feasibility and superiority of our method.

The remainder of this paper is structured as follows: Section 2 reviews the related work. Section 3 reviews model and methods, including the definition of SVCPN, the mapping relations between BDD and SVCPN, and the formal verification for airborne software safety. Section 4 is simulation and analysis firstly introduces the conversion from BDD to SVCPN, secondly introduces the mapping relations between BDD and SVCPN, and next is the resolution of safety inconsistency, and finally compares with related work. Section 5 concludes the paper.

2. Related Work

Currently, the general standard in the aviation field, namely, DO-178C, is formulated by The Radio Technical Commission for Aeronautics (RTAC) [1]. DO-178C divides the software system into five levels according to the consequences of failure, respectively, catastrophic, dangerous, important, secondary, and noninfluential. The standard sets out some principles for the airborne software to be abided by at every stage of the life cycle, whereas it does not mention the specific process and angle of validation.

The internal relationships between the components, as well as the safety levels of the components of software system, are complex and various [3]. The problems with partial component may influence functions of other components that depend on that part. The function of components interacting with each other results in the dependencies and effects on safety levels, threatening the safety performance of the whole system. The focus of this paper includes detecting and guaranteeing the safety performances of the airborne software system. By the combination of SVCPN and BDD, we find out the components of system that do not satisfy the safety requirement. Based on the comprehensive validation approach, as well as the safety inconsistency resolution algorithm, the software system is reverse traversed to detect the underlying components that cause the safety failures. The safety levels of the problematic underlying components are further reassigned to ensure the safety consistency of whole system. The innovations of this paper involve two aspects, one is the dynamic analysis of the airborne software system from a consistency perspective, and it improves the validation efficiency compared to the related methods. The other is the correctness of errors and it reduces the risk, so as to ensure the software is safe.

Leveson N G believes that the software safety means ensuring the software executed in the system will not cause unacceptable risks to the system [3]. At the beginning of the software designation, the safety boundary conditions should be considered to avoid the unsafe impact on the system. He points out that the disaster should be excluded during the process of system designation; if not, the risk of disaster must be minimized. The accessibility of the dangerous state of the safety-critical system is analyzed by the time Petri net.

The Failure Mode and Effects Analysis (FMEA) technique always is associated with Fault Tree Analysis (FTA) applied on the system safety analysis. Wang LS, Li SJ et al. [4] proposed a fault tree generation method based on fault configuration and introduced variability management of software product lines for system fault modeling and formal analysis. Park et al. [5] analyze the assessment methods for safety-critical software and point out that the existing methods can not be used for static or dynamic analysis at the same time. They combine the FMEA and FTA to solve the one-sidedness of safety assessment. Many researchers extend the FTA in order to describe more attributes of software. Ma [6] improves the FTA and proposes a Linear Time Fault Tree (LTFT) that is suitable for describing the timing fault of avionics software systems. It has solved the problem of software safety that involved time. In addition, the Dynamic Fault Tree extends the dynamic gate that can describe behaviors and interactions of complex system based on FTA [7], Domis D et al. [8] provide supports for software safety analysis through extending the FTA and the consistency verification. Abdulkhaleq A et al. [9] conduct a controlled experiment for comparing quantitatively the three analysis methods, that is, FTA, FMEA, and System-Theoretic Process Analysis (STPA), with regard to their effectiveness, applicability, and understandability. However the method has no ability to improve the efficiency of safety verification. FTA and FMEA can not expend the system into the area of reliability or probabilistic risk assessment. The safety analysis for FTA includes quantitative and qualitative analysis. The formal model such as the binary decision diagram, Petri net, and the Markov chain is usually used to formal the FTA model and then quantitatively analyze it. Ben Swarup Medikonda [10] believes that comprehensive safety verification must be performed to ensure system safety, to verify both DO-178B-compliant software safety and the compliance with DO-254 hardware safety. The above researches have learnt from mature FTA and FMEA and have carried out some extensions to adapt to safety issues that involve numerous factors. Thus, these methods have certain limitations when applied to airborne software safety, and it is caused by the following reasons. (1) Although the FTA and FMEA have graphical and tabular description, they are semiformal and there exist a problem in terms of semantics that can not fully express the delivery of safety relationships between the components of software system. (2) When the software is in the design stage, it is likely to face a large number of components. It is difficult to obtain reliable failure data and to build a correct and effective fault tree. (3) Large number of components of the airborne software and the interinfluence among the components; both may even affect the entire system when a certain component exist safety issues. Thus, it is difficult to obtain accurate results with these existing methods.

The modern airborne software architecture is distributed and the mutual influence between components is reflected in the functional dependencies among multiple components.

Xu BF et al. [11] use the System Modeling Language (SysML) to describe the dependencies among components based on DO-178B/C and verify the safety of airborne software through the built of static block safety diagram. Zhu HQ et al. [12] formulate the rules for the software safety verification based on the block diagram model of SysML that involves both the safety attributions and system structure characteristics. At present, formal methods have been widely used and have achieved considerable results, especially in areas of high safety requirements such as aviation, transportation, and nuclear power. Wang L, Chen M et al. [13] focus on modeling and verification of configuration information of Integrated Modular Avionics/ARINC653 system based on Modeling and Analysis for Real-time and Embedded Systems. The SCADE kit developed by Esterel Company provides certification methods and technologies to support some airworthiness certification standards [3]. It has been used in safety-critical systems such as automatic flight systems of EC135 and EC155 helicopters, the control system of A340 and A380, and the landing navigation system of B787. However, the following problems also exist in software safety verification. (1) Many software safety analysis methods do not fully apply to verification of airborne software safety attributes, due to the ignorance of airworthiness safety level standards. It is difficult for semiformal UML to accurately describe the delivery of safety levels in the system. (2) Airborne software safety must be based on airworthiness standards, and some existing safety verification methods are mainly based on DO-178B, DO-254 and IEC61508 and they can not meet the requirements of DO-178C. (3) Existing verification methods can only detect the inconsistency of safety requirements, without the ability of resolve the inconsistency.

This paper, combining the standard DO-178C, firstly proposes a software safety verification method based on Petri net. On the basis of SysML, the CPN is purposely extended to accurately describe the information and dependency of safety levels. Next, based on the reachability graph of PN, a search algorithm is proposed to traverse all the states of system to find out the components that do not meet the safety requirements of airworthiness standard. Finally, the disambiguation algorithm for safety level inconsistency is given, traversing and comparing all safety status to find out the places that do not meet the safety level requirement. By the antinet of PN and error correction methods, the initial places that cause the safety level inconsistencies are introduced, and the safety level status that the places should have finally is obtained.

3. Model and Methods

3.1. The Definition of SVCPN. This paper firstly applies the colored Petri net (CPN) to carry on the systematic formal research. The CPN is widely used in formal verification, and it can perfectly describe the dynamic behaviors of software system and always be extended to realize the formal verification of complex system. The original Petri net (PN) is defined as $\Sigma = (P, T, F; K, W, M_0)$, P represents the set of places, T is the set of transitions, F is a set of arcs and normally represented by arrows, K is the set of capacity functions of places, W is the set of arc expression functions, and M_0 represents the initial identifications of the model. Based on this, the CPN is always extended to a definition $\Sigma = (P, T, F, C, K, W, M_0)$, where C represents the color set. The conditions for enabling the transition t can be seen as follows, and the transition is signed as $t \in T, M[t >$.

$$\begin{aligned} &\forall p \in {\bullet t} : M(p) \geqslant W(p,t) \\ &\forall p \in t\bullet - \bullet t : M(p) + W(t,p) \leqslant K(p) \\ &\forall p \in t\bullet \cap \bullet t : M(p) + W(t,p) - W(p,t) \leqslant K(p) \end{aligned} \tag{1}$$

Once the transition is enabled, the identifications of the Net will be changed as the following rules.

$$M'(p) = \begin{cases} M(p) - W(p,t), & p \in {\bullet t} - t\bullet \\ M(p) + W(t,p), & p \in t\bullet - \bullet t \\ M(p) - W(p,t) + W(t,p), & p \in {\bullet t} \cap t\bullet \\ M(s), & others \end{cases} \tag{2}$$

The colored Petri net (CPN) is a high level Petri net (PN) introducing type manipulation so that it aids to describe the structure, property, and the initial distribution of resources of a system. It also can show the dynamic behavior mechanism of the system in detail. However, under some circumstances, traditional CPN can not satisfy the needs of the modeling. The safety level as an important attribute in the model of airborne software is expressed by token. If the token is being fired and transited in accordance with the rules of traditional CPN, the two cases will happen. (1) The disappearance of token in the preplace after the trigger results in the miss of safety information of component. (2) For ensuring that all the transitions can be triggered, two or more tokens always must exist in a place. However, due to the uniqueness of safety level of the software component, multiple tokens in one place may cause the confusion in semantic. The Navigation system is an aircraft safety-critical system that controls the flight path of the aircraft through an automatic navigation mode or a manual flight mode. The system consists of seven subsystems: Mechanical Steering Subsystem, Navigation User Interface Subsystem, LED Display Subsystem, Navigation Controller Subsystem, Heading and Speed Subsystem, Navigation Database Subsystem, and Navigation Subsystem. Figure 1 shows well the shortcomings of the CPN of the verification of the Navigation system, where the color of the token represents the security level of the component. The following deficiencies can be found: (1) The tokens indicating the security attributes of the Heading and Speed Subsystem, the Navigation Database Subsystem and the Navigation Subsystem of the library are not retained after running according to the established rules, thereby causing the problem of loss of component security information. (2) In the network system, in order to ensure the transitions T0, T2, and T3 can be triggered, there must be two tokens

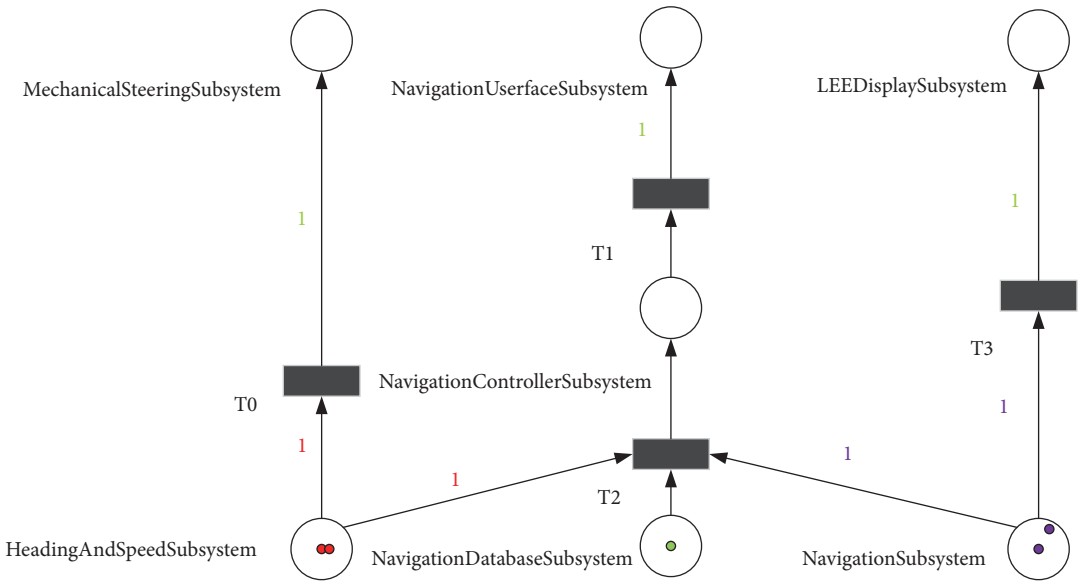

Figure 1: CPN of navigation system.

in the Heading and Speed Subsystem of the library and the Navigation Subsystem of the library, and the security attributes of the software components represented by the library are unique. Two identical security levels can cause semantic confusion and do not match the actual situation.

For this, the paper develops a CPN-based modeling, namely, SVCPN for the safety verification of airborne software. It enables to accurately describe and simulate the safety information of the system, as well as the transition of safety level between the components, so as to reduce the risk of conflict and collision.

Definition 1. Multiple sets ms are defined as $\bigcup_{z=Z} ms\,(z)z$, and Z is a finite nonempty set; $ms(z) \in N, Z_{MS}$ represents all the multiple sets of Z. Take a set $Z = \{a, b, c\}$, for example; two of its multiple sets are $ms_1 = \{a, 2b, c\}$ and $ms_2 = \{2a, c\}$.

Definition 2 ($SVCPN = (P, T, F, \Sigma, K, C, W, M_0)$).

P is a finite set of places.

T is a finite set of transitions, and $P \cap T = \varnothing$

F is a finite set of arcs such that $P \cap T = P \cap F = T \cap F = \varnothing$

Color set $\Sigma = \{red, orange, blue, green, purple, black\}$

Airworthiness standards stipulate five safety levels and divide the component into safety-critical and non-safety-critical [9]. The focus of this paper is to verify that the safety level of system satisfy the safety requirements. Our paper proposes six types of tokens to describe the safety levels of software component. Color red=safety level A, color orange=safety level B, color blue=safety level C, color green=safety level D, color purple=safety level E, and color black=Null and it represents that the component is non-safety-critical. A to E represent the decremented of safety level.

K is the capacity function of place, and $K = 1$. It represents that each component or module is provided with only one safety level.

C is a color function defined from $P \longrightarrow \Sigma$ such that $\forall p \in P, C(p) = token$.

W is an arc expression function defined from

$W : F \longrightarrow Expression, \forall f \in F : [Type(W(f)) = C(p(f))_{MS} \wedge Type(\mathrm{var}(W(f))) \subseteq \Sigma]$

$C(p(f))_{MS}$ is multiple color sets, as the Definition 1.

M_0 is the initial identifications of the model, defined from $\forall p \in P : M_0(p) \in C(p)_{MS}$

Definition 3. The identification M is expressed by vector $(M(p_1), M(p_2), \ldots, M(p_n))$, and

$$\forall p \in P : M(p) \in C(p)_{MS}. \tag{3}$$

The collision in PN is actually a competition for the resources of the places and also is the requirement for the place capacity. The focus of the SVCPN model is on the safety dependencies and transition within the components, and the abstract relationship is restrained by weight expression function and the direction of transition. The capacity of place is set to 1; on the one hand it makes the safety information of the software system more clear and reasonable, and on the other hand it eliminates the impact of collision and reduces the uncertainty of dynamic process.

This section mainly introduces the semantics of SVCPN model, in order to achieve the rationality of dynamic behavior of safety information according to the improvement of CPN. From the outset, the safety level dependency is stipulated as the following: (1) if both parties (components or modules) are safety-critical, then the safety level of postcomponent can not be lower than that of the precomponent; (2) if one of the parties is non-safety-critical, the postcomponent can not be

non-safety-critical; (3) if none of the parties is safety-critical, then it is not necessary to consider the way of connection.

Definition 4.

$$\forall p \in P: \begin{cases} W(p,t) = M(p) \\ W(t,p) = \min\left[W\left(p',t\right),\ p' \in {\bullet}t\right] \end{cases} \quad (4)$$

$M(p)$ is an identification of place p, $W(t,p)$ represents arc expression function that points from transition to the place, and $W(p,t)$ represents arc expression function that points from the place to transition. This definition clarifies the input and output weights of all places.

Definition 5. $\forall p \in P : M(p) \neq \varnothing$ is the condition for enabling the transition based on the identification M and marked as $M[t>$. All sets of enabled transitions are recorded as *enable* (M). This definition makes the triggering condition clear, and whether the transition occurs depends on the input of its presets.

Definition 6. A new identification M' can be obtained after the transition t enabled, marked as $M[t > M'$. This definition specifies the identification of the postplace, and the safety information transmitted as a state always exists during the process of verification.

$$M'(p) = \begin{cases} M(p), & p \in {\bullet}t - t{\bullet} \\ \min\{M(p), W(t,p)\}, & p \in t{\bullet} - {\bullet}t \\ M(p), & others \end{cases} \quad (5)$$

Since the rules of CPN have no way to describe the characteristics of security feature propagation, we define the color according to the security level based on the original CPN network and redefine the transition rules according to the characteristics of security impact. Compared with the traditional CPN, the SVCPN model improves the triggering and running rules and eliminates the situation of conflict, collision, and confusion of information transition, thus reducing the complexity of the structure. The number of tokens in the place is set reasonably and information of token is still retained after the transition, so that both the initial safety information of component can be preserved and the disorder of software safety level can be avoided.

3.2. The Mapping Relations between BDD and SVCPN. The System Modeling Language (SysML) as a standard modeling language for system engineering integrates the advantages of visualization design language of object-oriented and process-oriented. It has the ability to achieve the detailed description, analysis, and design of static system [14]. Due to the semiformalization of SysML model, the dynamic behavior of system can further be enlightened investigated and illuminated through the conversion between SysML and formal method to achieve the formal verification.

SysML can more effectively aid the understanding of a system through block definition, package (BDD), internal block, instance, parametric, and requirement diagrams [15]. Considering the complexity of airborne software system and the characteristics of dependence on hardware, software, and operator, as well as the multiple join relationship of the diagram, the BDD is chosen to describe the static structure of system intuitively. However, since the semiformalization of the BDD, we propose a series of transformational rules for the conversion from BDD to SVCPN model to achieve the formal verification of airborne software.

Modern airborne software is a componentized distributed architecture, and each block of the BDD represents a component of the software system. Different components with different attributes and is assigned corresponding safety level. In order to describe the static structure of airborne software better, the BDD with two safety attributes is firstly set up. The safety level attribute corresponds to the five safety levels from A to E (introduced in Section 4.1); the safety-critical attribute is Booleans; if the component is safety-critical, the value is true; otherwise the value is false.

The relationships between the modules in BDD mainly include generalization, association, dependency, composition, and aggregation. There is a mutual influence and interdependence between them. The key of achieving the conversion between BDD and SVCPN is to establish the mapping transformational rules. The place P represents the component block, and the transition T and the line show the transfer of safety levels between these components. The relationships, as well as the transformation rules of the safety level are introduced as follows: (1) the association relationship in BDD represents the active association block pointing to the associated block; thus the safety level of the associated block could not be lower than the active association block. Block 1 and Block 2 are, respectively, represented by the libraries P1 and P2, and the transition between the blocks is represented by transition T1, which corresponds to the relationship between the active associated position and the associated position in the SVCPN model; (2) the dependency in BDD represents the active block pointing to the dependent block; thus safety level of the dependent block could not be lower than the active dependency block. This corresponds to the relationship between the active dependency place and the dependent place in SVCPN model; (3) the composition and aggregation relationship represents the relationship between part and entirety. From the safety perspective, the safety level of part has an impact on the entirety. The safety level of the entirety is depended on all the parts. If the safety level of part does not meet requirement, the safety performance of the entirety will be reduced; thus safety level of the entirety must be less than or equal to the level of part. These two relationships expressed in SVCPN the part are that the safety level of software component represented by place should be higher than or equal to the safety level of overall system. These mapping rules are shown in Figures 2–5, the place in SVCPN represents the corresponding component, and the directional line indicates the relationship between the components.

3.3. The Formal Verification for Airborne Software Safety. For the airborne software to be verified, the static structure of

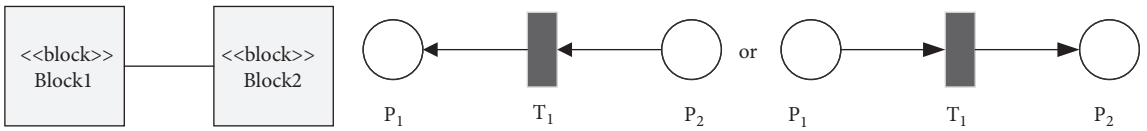

Figure 2: The association relationship in the CPN model.

Figure 3: The dependency in the CPN model.

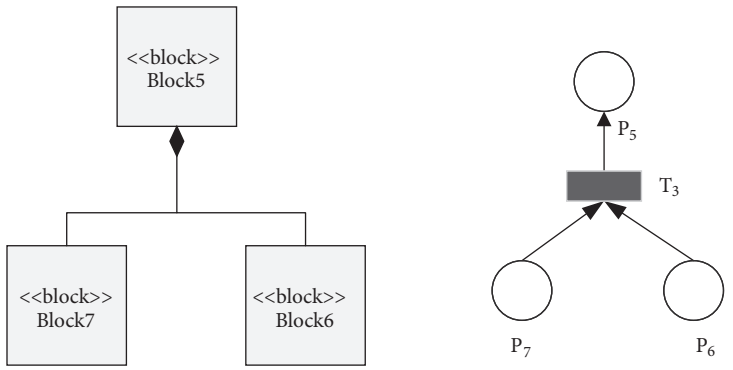

Figure 4: The composition in the CPN model.

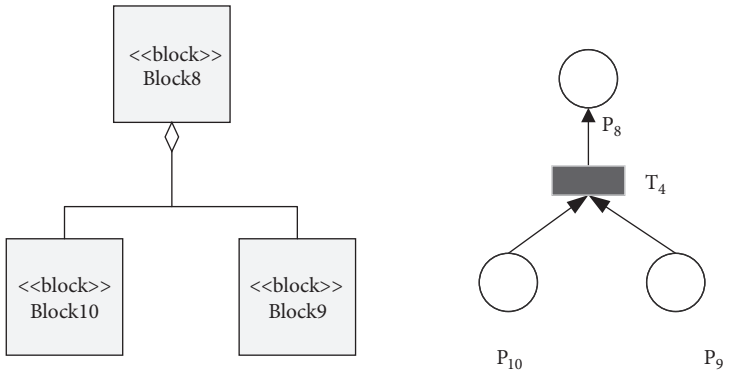

Figure 5: The aggregation in the CPN model.

system involves the information of safety level and relationship between components is firstly described based on BDD. Then the BDD is converted to SVCPN in order to achieve the formal verification of system. Finally the safety verification algorithm is designed for the automatic validation and found out the underlying components that cause safety loophole so as to ensure the systemic safety.

The process of verification algorithms is shown in Figure 6. The SVCPN model is initialized from the outset, and token is transited based on the triggering rules introduced in Section 3.2. It does not reach the terminal status until all of the transitions have been triggered, and under the stable status each place will own the only one token with corresponding color and it represents the safety level of the corresponding component of the software. Comparing it with the safety design requirements, if below indicates that the safety performance of the corresponding component dose not fulfill the requirements. Then, the antinet of SVCPN is constructed and safety inconsistency resolution algorithm is applied to orientate the root causes of problem along the reverse transmission path and find out the fundamental components that lead to the inconsistency of safety level. Based on the algorithm, the safety level that the fundamental components should have can be calculated and the safety level can be further reassigned. Repeat the above steps until all the components fulfill the safety requirements (i.e., higher than or equal to the standard safety level) and ultimately finish the safety verification of airborne software.

The safety verification algorithm includes safety consistency detection algorithm and safety inconsistency resolution

For $\forall t \in T, M[t >$
  Do $M[t > M'$;
Until no $M(p) \neq \varnothing$;
$RG = (R(M_0), E, P)$ and the reachability tree is obtained
Visit $R(M_0)$; //accessing each note of reachability tree
If $\exists M \in R\,(M_0), \wedge\forall M(p_i) \in M, M(p_i) \neq 0$; //validating whether the element of each note is not 0
  $M(p_i) <$ *safety level of* p_i; //comparing the identification in the place with the required safety level of corresponding component
Then $p_i \longrightarrow$ *abnormal*; //classifying the problem places into set abnormal
until all nodes of $R(M)$ are visited;
print abnormal; //putting out all problem places

Algorithm 1: The safety consistency detection algorithm.

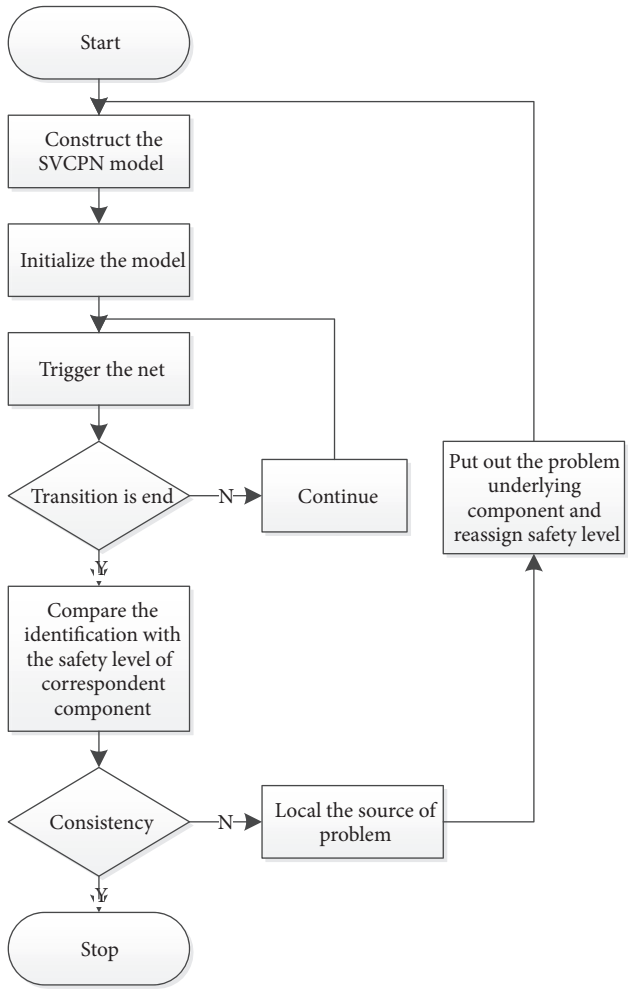

Figure 6: The process of verification algorithms.

algorithm. The aim of the safety consistency detection algorithm is to find out the components that do not conform to the safety requirement. The focus of the safety inconsistency resolution algorithm is mainly on locating the source of problem; namely, the underlying components, as well as providing the safety levels that the components, should have in order to make the system meet the safety requirements.

Definition 7. Assuming that $\Sigma = (P, , T, F, M_0)$ is a bounded PN, the reachability graph of it is defined as $RG(\Sigma) = (R(M_0), E, P)$, $E = \{(M_i, M_j) \mid M_i, M_j \in R\,(M_0), \exists t_k \in T : M_i[t_k > M_j\}$, $P : E \longrightarrow T, P(M_i, M_j) = t_k$, if and only if $M_i[t_k > M_j, R(M_0)$ is the vertex set of $RG\,(\Sigma)$, E is the arc set of $R\,(M_0)$.

3.3.1. The Safety Consistency Detection Algorithm. The security consistency detection algorithm is shown in Algorithm 1. The algorithm is based on the reachability graph of SVCPN, it traverses every state node of the reachability graph by the way

$M(p_i) \longleftarrow$*safety level of* p_i, ($p_i \in$*abnormal*); //taking the required safety level of the component as the identification of the corresponding place, namely initializing the anti-net
For $\forall t \in T, M[t >$
Do $M[t > M'$;
Until no $t \in T, M[t >$;
$M(P') \in M'$; $//P'$ is the place with initial identification
If $M(p') >$*safety level of* p'; comparing $M(p')$ with initial safety level in place P'
Then $p' \longrightarrow$*abnormal*;
print *abnormal* and $M(p')$; //putting out the problem places and the identifications, and the safety level of the corresponding component need to be reassigned.

Algorithm 2: The disambiguation algorithm.

of DFS. If the identification in one place of reachability tree is lower than the required safety level of the corresponding component, then this component do not fulfill the safety requirement. The DFS method can only analyze the security relationships between components. For example, the DFS algorithm can find the set of components that affect the security of the component. However, it cannot describe the level of security or the extent to which other components affect the security of the component.

3.3.2. Antinet of SVCPN. Starting with the problem places after initializing the antinet and running the net reversely are made to find out the source of problems based on the safety inconsistency resolution algorithm. Due to the change in the direction of the association lead to the change of the transition rules, therefore the triggering rules of the antinet should be redefined.

Definition 8.

$$\forall p \in P: \begin{cases} W(p,t) = M(p) \\ W(t,p) = \max\left[W\left(p',t\right),\ p' \in {\bullet}t\right] \end{cases} \quad (6)$$

$W(t,p)$ represents the arc expression function that points from transition to place, $W(p',t)$, $p' \in {\bullet}t$ represents the all functions that point to a certain transition, and $W(p,t)$ represents arc expression function that points from place to the transition. This definition clarifies the input and output weights of any one place.

Definition 9.

$$M'(p) = \begin{cases} M(p), & p \in {\bullet}t - t{\bullet} \\ \max\{M(p), W(t,p)\}, & p \in t{\bullet} - {\bullet}t \\ M(p), & others \end{cases} \quad (7)$$

$M(p)$ is an identification of place p, $W(t,p)$ is the arc expression function that point from the transition to place, $\max\{M(p), W(t,p)\}$, and $p \in t{\bullet} - {\bullet}t$ represents that the identification in the place pointed by the triggered transition is the larger one between the input weight function and the identification in the preplace.

During the process of reverse lookup, the function of directional arc is increase along the transition path and is opposite to the before. Meanwhile, the identification in the place retains the larger one.

3.3.3. The Disambiguation Algorithm. The disambiguation algorithm is shown in Algorithm 2.

4. Simulation and Analysis

The airborne navigation system is a typical embedded software system, as a vital part of airborne system, it is very essential to be taken the safety analysis. This section takes the airborne navigation system which also be analyzed as a case in other relevant research literatures [11, 16], as an example to demonstrate the feasibility and superiority of the proposed verification method. The navigation software is composed of multiple components, and these components are interconnected based on various relationships. The navigation software consists of 12 components, and the BDD of it is shown in Figure 7. The blocks represents the components, respectively, are Controller, Safe Point Determinator, Navigation Subsystem, Navigation Interface, External Subsystem Event Handler, Navigation Database Monitor, Navigation Monitor, Heading and Speed Monitor, Navigation Database Interface, Navigation Database Subsystem, Heading and Speed Interface, and Heading and Speed Subsystem. The connections between these modules relied on the relationships of reference association, part association, shares association, and dependency.

4.1. The Conversion from BDD to SVCPN. In order to realize the conversion from BDD to SVCPN model, it needs to be converted according to the following three steps.

Firstly, create a library, and establish the correspondence between each module information in the library and the SysML diagram. The initial information of the place corresponding to the module in the DBB is described in Table 1 includes the safety level of the correspondent component and the information of the token.

Secondly, according to the mapping between BDD and SVCPN in Section 3.2, the transition between the library and the directed arc are added to complete the conversion of the SysML block definition map to SVCPN.

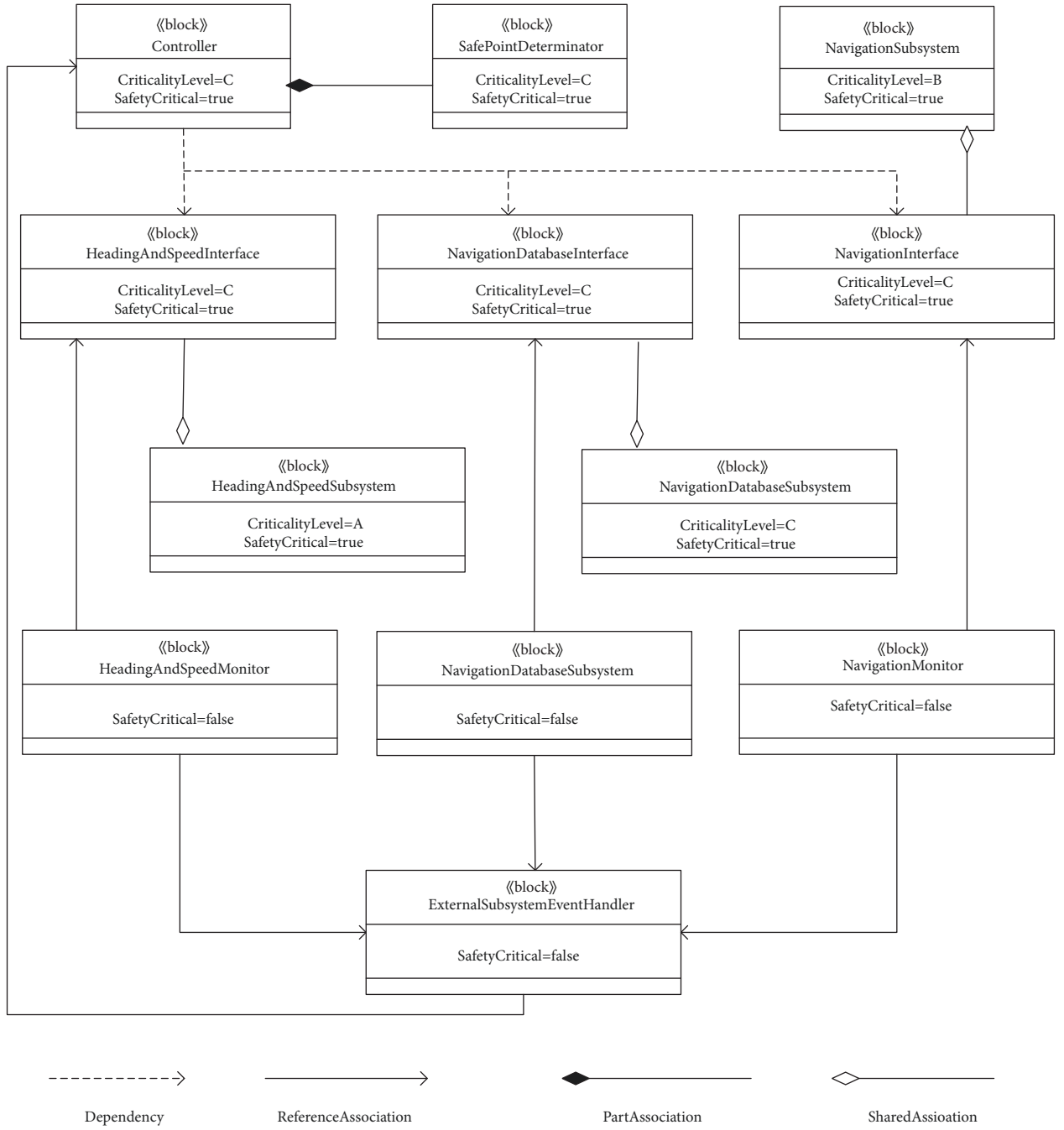

Figure 7: The BDD of navigation software.

Table 1: The initial information of component (place).

Place	Component	Safety Level	Color of Token
P_1	Navigation Interface	A	red
P_3	Safe Point Determinator	C	blue
P_9	Navigation Database Interface	D	green
P_{11}	Heading And Speed Interface	B	orange

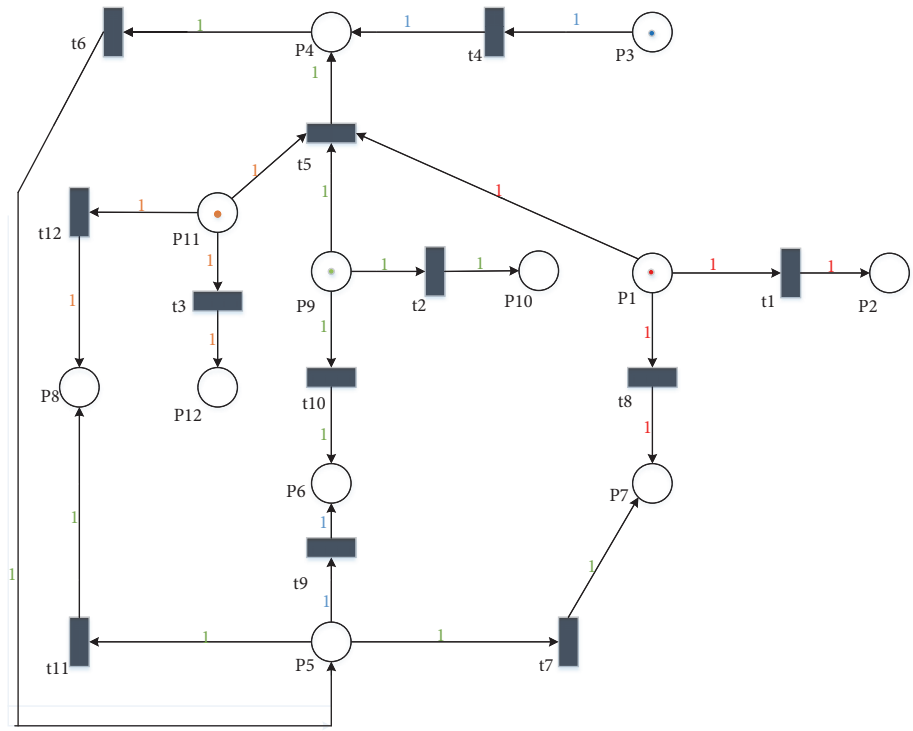

Figure 8: The initial status of SVCPN.

Finally, the SVCPN is initialized based on Table 1. After the initialization is completed, it runs according to the extended semantics of Section 3.1, thus realizing the automation of the verification process. The initial identifications in places P1, P3, P9, and P11 represent the safety level of correspondent component, and the safety level is expressed by colored token, as is shown in Figure 8. The token in place P1 is red, the token in place P3 is blue, the token in place P9 is green, and the token in place P11 is orange.

4.2. Safety Verification Based on SVCPN. After the initialization, the SVCPN model is triggered based on Definition 5, the arc expression function is in accordance with Definition 4, and the identification in place after triggering is abided by Definition 6. The sequence of the triggers in this paper is $t_1 \sim t_{12}$; namely, the sequence of transition is $\sigma = \{t_1, t_2, \ldots, t_{12}\}, M_0[\sigma > M_{12}$. The specific running process of the net is introduced in the following. The initial state is M_0, and when the transition t_1 is triggered, the color of token in the place p_2 becomes red; the state of the system is marked as M_1; namely, $M_0[t_1 > M_1$; when the transition t_2 is triggered, the color of token in place P_{10} becomes green; the state of the system is marked as M_2; namely, $M_1[t_2 > M_2$; when the transition t_3 is triggered, the color of the token in place P_{12} becomes orange; the state of the system is marked as M_3; namely, $M_2\ [t_3 > M_3$; when the transition t_4 is triggered, the color of the token in place P_4 becomes blue; the state of the system is marked as M_4; namely, $M_3[t_4 > M_4$; when the transition t_5 is triggered, the color of the token in place P_4 becomes green; the state of the system is marked as M_5; namely, $M_4[t_5 > M_5$; when the transition t_6 is triggered, the color of the token in place P_5 becomes green; the state of the system is marked as M_6; namely, $M_5[t_6 > M_6$; when the transition t_7 is triggered, the color of the token in place P_7 becomes green; the state of the system is marked as M_7; namely, $M_6[t_7 > M_7$; when the transition t_8 is triggered, the color of the token in place P_7 becomes green; the state of the system is marked as M_8; namely, $M_7[t_8 > M_8$; when the transition t_9 is triggered, the color of the token in place P_6 becomes green; the state of the system is marked as M_9; namely, $M_8[t_9 > M_9$; when the transition t_{10} is triggered, the color of the token in place P_6 becomes green; the state of the system is marked as M_{10}; namely, $M_9[t_{10} > M_{10}$; when the transition t_{11} is triggered, the color of the token in place P_8 becomes green; the state of the system is marked as M_{11}; namely, $M_{10}[t_{11} > M_{11}$; when the transition t_{12} is triggered, the color of the token in place P_8 becomes green; the state of the system is marked as M_{12}; namely, $M_{11}[t_{12} > M_{12}$. Partial of reachability tree is provided in Figure 10. All the transitions have been triggered, and the terminal status of the system is M_{12}. It can be regarded as the stable status and is showed in Figure 9.

Table 2 shows the required safety level of each component (place) of airborne software system, Null represents that there is no safety level requirement for the corresponding component.

The system arrives termination status, and the mark is $(R, R, B, G, G, G, G, G, G, G, O, O)$. Compared with the required safety level shown in Table 2, we can find that places P_{12} (Heading And Speed Subsystem) and P_4 (Controller) are abnormal. Take P_4 as an example, the final token in P_4 is green; namely, the safety level of is D, while the required safety

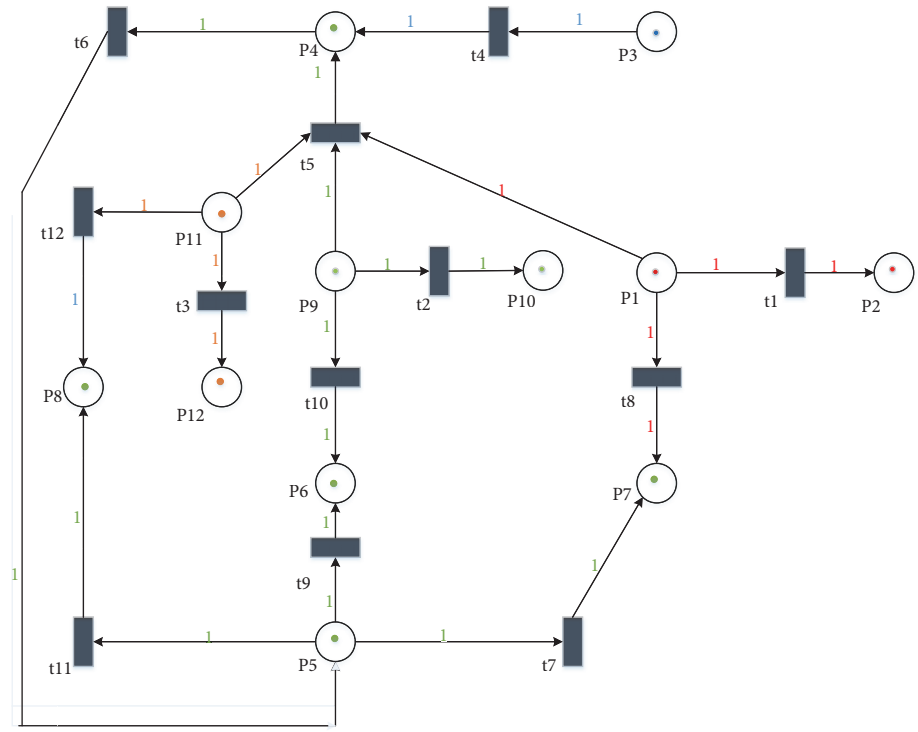

Figure 9: The terminal status of SVCPN.

Table 2: The safety requirement level of component (place).

Place	Component	Safety requirement Level	Color of Token
P_2	Navigation Subsystem	B	orange
P_4	Controller	C	blue
P_5	External Subsystem Event Handler	Null	black
P_6	Navigation Database Monitor	Null	black
P_7	Navigation Monitor	Null,	black
P_8	Heading and Speed Monitor	Null	black
P_{10}	Navigation Database Subsystem	C	blue
P_{12}	Heading And Speed Subsystem	A	red

level of the correspondent component Controller is C (refer to Table 1). This indicates that component Controller does not fulfill the safety requirement and it may result in a hazardous state.

4.3. The Resolution of Safety Inconsistency. In Section 4.2, the places that do not meet the safety requirement have been locked, and we further need to find out the reasons. Taking place P_4 for example, the safety level of it is influenced by transitions t_4 and t_5, and the fundamental places that cause place P_4 unsatisfying the safety requirement would be oriented through the process of antitraverse. The safety levels can be reassigned to the fundamental places to make P_4 meet the safety requirements. The main processes are as the following steps. (1) Starting with the problematic place P_4 and based on the antinet of SVCPN in Section 4.2. Use the required safety level, namely, the blue token and red token, to initialize places P_4 and P_{12}. The antinet is shown in Figure 11.

(2) Triggering the antinet reversely, as shown in Figure 12, the arc expression functions and the identifications after the transition enabled are in accordance with Definitions 8 and 9.

(3) Finally, the color of the token in place P_9 is blue and the color of the token in place P_{11} is red. This means that the safety levels of the components represented by the two places need to be at least C and D, so as to make the software system meet the safety requirement. Thus, the initial safety levels of Navigation Database Interface and Heading and Speed Interface should be reset by the engineer referred to the results of verification.

4.4. The Comparison with Related Work. It can be seen from the case study that the method proposed by our paper can achieve the safety verification of airborne software correctly and efficiently. It is based on the existing engineering model, and the SVCPN can be quickly obtained through mapping

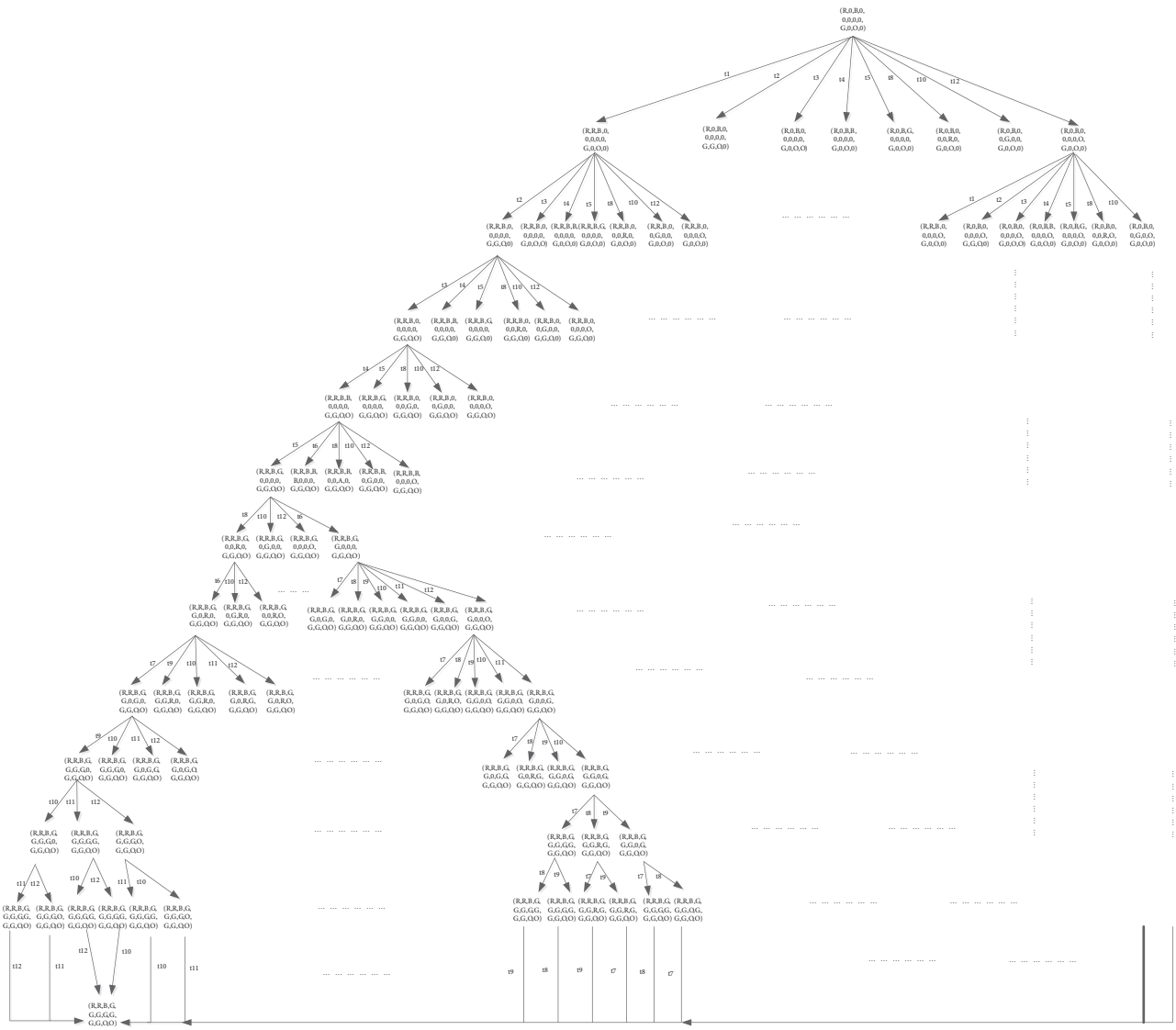

Figure 10: The reachability tree by SVCPN.

rules. The process of verification is automatic with the support of the syntax and semantics of extend CPN. Compared with the existing research work, the method also has certain advantages and characteristics. (1) Compared with the FTA and FMEA, the PN is formalized model and can describe information and dynamic behavior more comprehensively by extending the syntax and semantics. It can be expended to the area of reliability and probabilistic risk assessment. However, the FTA and FMEA can no accurately describe the information and behavior, and the scalability is not as good as PN. Compared with the researches [17–20], this paper presents a complete verification method based on extended CPN and takes the safety level dependencies in the system into consideration, so as to make sure the safety level assignations meet the airworthiness requirements. (2) In [11], the BDD of SysML is transformed into a dependency graph and the safety detection is based on graph theory. This method can only detect the existence of safety level inconsistency but it can not give the cause of safety inconsistency. Compared with the matrix-based method proposed in [12], the extended CPN has stronger visualization and facilitates engineer access to the behavior information of system. The extended CPN can not only analyze the safety, but also use the existing verification method to detect the reachability and deadlock of the system. This method avoids the repeated modeling and reduces the complexity and cost of verification. Last but not least, the method in our paper can find out the cause of the exception and return the safety level to the engineer that the component should have.

Our paper remedies the limitations of relevant researches to a certain extent. Many software safety studies have emerged in recent years but less involved in the respect of safety requirement and validation of airborne software, let alone the analysis of dependencies of safety levels between the components of software system. The previous work about the safety of airborne software always focused on proving

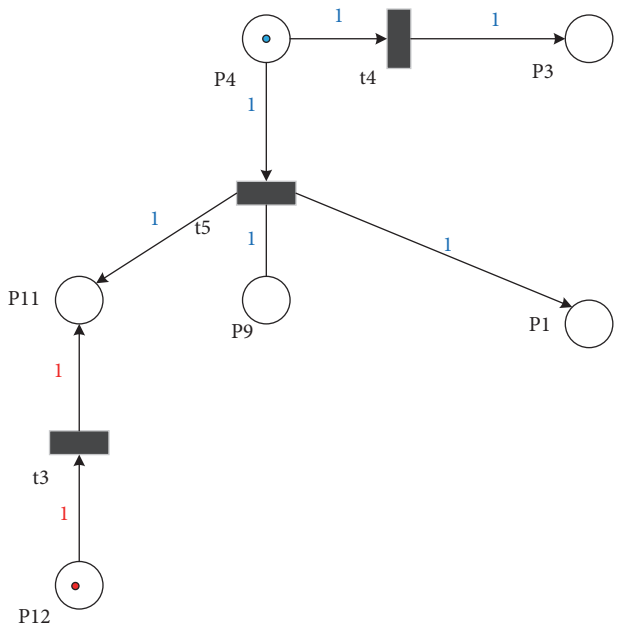

Figure 11: Initial status of the antinet.

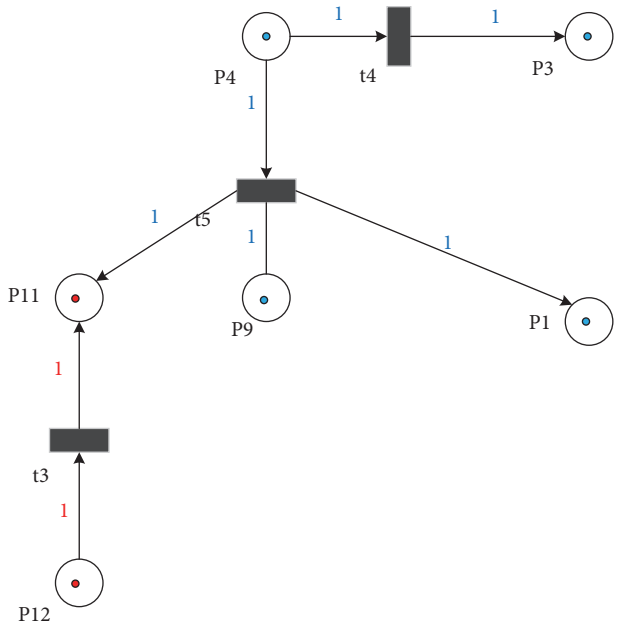

Figure 12: Triggering the antinet.

the functional correctness of software and demonstrating the system fully satisfies the safety requirement. However, they rarely involve how to reduce the risk and improve to ensure the system safety. The overall objective of our paper is to fill the gap according to orientate the root causes of the risk based on the safety inconsistency resolution algorithm so as to improve and ensure the safety performance of software systems. In contrast to traditional formal verification models and techniques, the comprehensive verification method proposed by our paper also has some breakthroughs. The SSCPN method is simple and easy to prepare and implement and has the ability to validate automatically with the support of extended syntax, semantics, and algorithm. Compared with the BDG (block dependency graph), the SSCPN with higher verification efficiency and reachability tree graph can provide complete systemic identification and the path of transition. Compared with other formal methods, such as FTA and FMEA, our method has more accurate and comprehensive description of information and behavior of system, as well as the stronger expansibility, and has the ability to further expand into the field of risk probability assessment.

5. Conclusions

This paper, combining the standard DO-178C, proposes a software safety verification method based on Petri net. On the basis of SysML, the CPN is purposely extended to accurately describe the information and dependency of safety levels. Based on the reachability graph of PN, a search algorithm is proposed to traverse all the states of system to find out the components that do not meet the safety requirements of airworthiness standard. Finally, the disambiguation algorithm for safety level inconsistency is given to ensure that the safety levels of components meet the safety requirements.

The paper firstly builds the mapping rules from the BDD to SVCPN. SysML, as a modeling language used in engineering, has the characteristics of semiformal and intuitive graphical interface. But these lead to the incompleteness of syntax and semantics and are difficult for safety verification directly. In order to take advantage of the capabilities of description and analysis, the paper analyzes the connection relationships within the SysML and sets a series of mapping rules to achieve the model transformation.

According to the characteristics of multicomponents and distributed architecture of airborne software, the PN is selected in this paper as the analysis tool. The paper presents an extended CPN, namely, SVCPN for the safety level consistency verification of airborne software system. The SVCPN can describe the safety dependency and structure among the components accurately. The semantics of PN are used to represent the transfer of safety level between components. The final safety level status of safety-critical components can be obtained based on the reachable tree of PN after running.

At last, the paper proposes a detection algorithm for safety level inconsistency, traverse, and compare all safety status to find out the places that do not meet the safety level requirement. By the antinet of PN and error correction methods, the initial places that cause the safety level inconsistencies are introduced, and the safety level status that the places should have finally be obtained.

Since aviation has very high requirements for system safety, this example can be used to verify that the method proposed in this paper can meet the requirements for safety inspection of high safety aviation systems. And by redefining the colored set of SVCPN, this method can also check the system security of other industries.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

We appreciate the relevant staff providing us with the help. This manuscript is based on the research of airworthiness formal certification of aviation software, and it was supported by "the Fundamental Research Funds for the Central Universities (no. NS2018046)".

References

[1] Z. Q. Huang, B. F. Xu, S. L. Han et al., "Survey on embedded software safety analysis standards, methods and tools for airborne system," *Journal of Software*, vol. 25, no. 2, pp. 200–218, 2014.

[2] Kdawson, "Software bug halts F-22 flight," 2011, http://it.slashdot.org/story/07/02/25//Software-Bug-Halts-F-22-Flight.

[3] B. Wang, X. Y. Bai, F. He et al., "Survey on modeling and verification techniques of composable software," *Journal of Software*, vol. 25, no. 2, pp. 234–253, 2014.

[4] L. Wang, S. Li, O. Wei, M. Huang, and J. Hu, "An automated fault tree generation approach with fault configuration based on model checking," *IEEE Access*, vol. 6, pp. 46900–46914, 2018.

[5] G.-Y. Park, D. H. Kim, and D. Y. Lee, "Software FMEA analysis for safety-related application software," *Annals of Nuclear Energy*, vol. 70, no. 3, pp. 96–102, 2014.

[6] M. Lin, "FTA based avionics software safety verification methodology," *Nanjing University of Aeronautics and Astronautics*, 2012.

[7] M. Čepin and B. Mavko, "A dynamic fault tree," *Reliability Engineering & System Safety*, vol. 75, no. 1, pp. 83–91, 2002.

[8] D. Domis, K. Hofig, and M. Trapp, "A consistency check algorithm for component-based refinements of fault trees," in *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 171–180, IEEE, San Jose, CA, USA, November 2010.

[9] A. Abdulkhaleq and S. Wagner, "A controlled experiment for the empirical evaluation of safety analysis techniques for safety-critical software," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, EASE 2015*, pp. 1–10, ACM, April 2015.

[10] B. S. Medikonda, P. S. Ramaiah, and A. A. Gokhale, "FMEA and fault tree based software safety analysis of a railroad crossing critical system," *Global Journal of Computer Science and Technology (GJCST)*, vol. 11, no. 8, pp. 57–62, 2011.

[11] B. Xu, Z. Huang, J. Hu, and X. Yu, "Model-driven safety dependence verification for component-based airborne software supporting airworthiness certification," *Acta Aeronautica et Astronautica Sinica*, vol. 33, no. 5, pp. 796–808, 2012.

[12] H. Q. Zhu, H. J. Xu, P. Zhang et al., "Model-driven validation method for software component development assurance level," *Acta Aeronautica et Astronautica Sinica*, vol. 35, no. 6, 2014.

[13] L. Wang, M. Chen, and J. Hu, "Formal verification method for configuration of integrated modular avionics system using MARTE," *International Journal of Aerospace Engineering*, vol. 2018, Article ID 7019838, 22 pages, 2018.

[14] F. Mhenni, N. Nguyen, and J.-Y. Choley, "Automatic fault tree generation from SysML system models," in *Proceedings of the 2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM 2014*, pp. 715–720, IEEE, July 2014.

[15] J. X. Liu, X. D. Xiong, and S. F. Wang, "Verifying SysML state machine diagram based on stochastic petri net," *Computer Applications and Software*, vol. 30, no. 6, pp. 202–208, 2013.

[16] Y. Yin, B. Liu, and D. Su, "Research on formal verification technique for aircraft safety-critical software," *Journal of Computers*, vol. 5, no. 8, pp. 1152–1159, 2010.

[17] S. Bernardi, J. Merseguer, and C. D. Petriu, "Adding dependability analysis capabilities to the MARTE pro-file," in *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, pp. 736–750, 2008.

[18] Z. Yan and M. Hong, "The description and verification of non-functional attributes in UML diagrams," *Journal of Software*, vol. 20, no. 6, pp. 1457–1469, 2009.

[19] G. Zoughbi, L. Briand, and Y. Labiche, "UML profile for developing airworthiness-compliant (RTCA DO-178B), safety-critical software," in *Proceedings of the Model Driven Engineering Languages and Systems, International Conference, MODELS 2007*, pp. 574–588, Nashville, Tenn, USA, 2007.

[20] H. Hungar, O. Robbe, and B. Wirtz, "Safe-UML-restricting UML for the development of safety-critical systems," in *Proceedings of the FORMS/FORMAT 2007*, pp. 467–475, 2007.