applied sciences

MDPI

Article

System-Theoretic Process Analysis Based on SysML/MARTE and NuSMV

Deming Zhong *, Rui Sun, Haoyuan Gong and Tianhuai Wang

School of Reliability and Systems Engineering, Beihang University, Beijing 100191, China; by1414107@buaa.edu.cn (R.S.); ghy@buaa.edu.cn (H.G.); zy2114128@buaa.edu.cn (T.W.)
* Correspondence: 07961@buaa.edu.cn

Abstract: Systems Theoretical Accident Model and Process (STAMP), which considers system safety as an emergent property of the system, is a more effective accident/loss causality model for modern complex systems. Based on STAMP, System Theoretical Process Analysis (STPA) has attracted increasing attention as a new approach to hazard analysis, and relevant international standards are being developed. However, STPA is mainly performed manually, leading to inefficiencies, and constructs models in non-standard language, hindering the integration with existing systems engineering. STPA-SN (STPA based on SysML/MARTE and NuSMV) is proposed to build model in SysML, describing the timing with MARTE (Modeling and Analysis of Real-Time and Embedded Systems), transform SysML model into NuSMV model and output loss scenarios automatically with model checker. An application example of STPA-SN is provided to demonstrate potentials for higher efficiency of analysis and for collaboration with SysML-based systems engineering.

Keywords: system theory process analysis (STPA); SysML; NuSMV; unsafe control action (UCA); loss scenario

Citation: Zhong, D.; Sun, R.; Gong, H.; Wang, T. System-Theoretic Process Analysis Based on SysML/MARTE and NuSMV. *Appl. Sci.* **2022**, *12*, 1671. https://doi.org/10.3390/app12031671

Academic Editors: Giancarlo Mauri and Augusto Ferrante

Received: 6 November 2021
Accepted: 1 February 2022
Published: 5 February 2022

1. Introduction

Due to the use of software, the system accident mechanism gradually changes and the traditional safety theory, based on physical failures, shows its inadequacy. In 2004, Nancy G. Leveson proposed the Systems Theory Accident Model and Process (STAMP), which believes that system safety is an emergent property of the system [1] (pp. 10–12). Based on STAMP, STPA (System-Theoretic Process Analysis) was proposed as a hazard analysis method, considering that unsafe interactions between components can lead to accidents even in the absence of physical failures [1] (p. 4). Numerous comparative studies have shown that STPA has identified not only the causal scenarios found by traditional methods, but also causal scenarios not found by traditional methods, such as the fault tree analysis (FTA), failure mode and effect criticality analysis (FMECA), event tree analysis (ETA) and hazard and operability analysis (HAZOP) [1] (p. 4).

STPA is now accepted in standards, such as "RTCA DO 356A: Airworthiness Security Methods and Considerations" and "ISO/PAS 21448: Safety of the Intended Functionality (SOTIF)". In addition, the Society of Automotive Engineers (SAE) is developing two STPA-related standards: "SAE AIR6913: Using STPA during Development and Safety Assessment of Civil Aircraft" and "SAE J3187: Applying System Theoretic Process Analysis (STPA) to Automotive Applications".

In March 2018, Nancy G. Leveson and John P. Thomas published the "STPA Handbook" [1], which is the basis for the standards development, industrial applications, tool designs and methodological improvements of STPA. To distinguish STPAs from the other literature, STPA in the "STPA Handbook" is abbreviated as HSTPA. HSTPA consists of four steps:

Step 1: Define the purpose of the analysis. This includes identifying losses, identifying system-level hazards, identifying system-level safety constraints and refining hazards.

Step 2: Model the control structure. An effective control structure will enforce constraints on the behavior of the overall system. The hierarchical control structure model includes at least five types of elements: controllers, controlled processes, control actions, feedbacks and inputs/outputs among other components. The controller includes control algorithms and the process model.

Step 3: Identify the unsafe control actions. An unsafe control action (UCA) is a control action that will lead to a hazard in a particular context and worst-case environment. The UCA consists of five parts: control action (CA), source of CA, unsafe type, context and link to hazards.

The unsafe types include:

- Not provided CA;
- Provided CA;
- Provided CA too early or provided CA too late;
- Stopped CA too early or applied CA too long.

Step 4: Identify the loss scenarios. The loss scenario describes the causal factors (CFs) that can lead to UCAs and to hazards.

At present, HSTPA still has two drawbacks in industrial practice:

- Manual analysis, which is inefficient [2] (p. 151);
- Modeling with non-standard language, which is difficult to be integrated into existing systems engineering.

Automation can improve the efficiency, and the formalization of STPA is the basis for automation. Only after formalizing STPA can tools be developed to realize automation and may STPA support standardized modeling languages, such as SysML, AADL and AltaRica.

The formalization of STPA includes the definitions of UCA and loss scenario, the construction of the analytical model, the identification processes of UCAs and the loss scenarios. The related representative work is described below.

John P. Thomas [3] formalized the definition of UCA, which was adopted by HSTPA. However, there are still some ambiguities. It is not specified whether UCA is a state or not. Similarly, it is not specified whether the CA in the definition of UCA is the one received by the controlled process or the one issued by the controller, which may lead to critical differences while identifying the loss scenario.

John P. Thomas [3] identified UCAs with truth calculations, which was adopted by many subsequent works, e.g., [4,5]. Under certain conditions, the truth calculation is effective to identify UCAs automatically.

Asare et al. [2] and Zhu et al. [6] proposed their respective formal analysis methods, which include analytical model construction and model analysis. They suffer from shortcomings in terms of ease of use and quality of analysis due to the inability to use existing established methods. Although STPA has the concepts of UCA and the loss scenario, the STPA analytical model is still able to be modeled with existing mature model methods, such as SysML and AADL. Similarly, the STPA analytical model can be analyzed by existing mature analytical methods, such as model checking and simulation.

Abdulkhaleq et al. [7] used finite state machines to describe the system information, but only used the information in the state machines manually and therefore could not acquire the dynamic process of loss scenario rigorously. State machine is an effective, sufficient and convenient way to construct system behaviors. State machine is also supported by many modeling languages, such as SysML, AADL and AltaRica, and could be analyzed by either simulation or model checking. Therefore, state machine is ideal method to build the system behaviors for the identifications of UCAs and loss scenarios.

Chen et al. [8] applied the "four-variable model" to identify the variables that are used in the analytical model. Although the truth table was used to describe the relationships between some of the variables, the system behavior was not adequately described.

Rey et al. [9] agreed that STPA lacks formalism and proposed a method that combines STPA and SysML modeling activities, but did not change the STPA method itself.

Zhu et al. [10] constructed the analytical model with Petri Net. Despite its mathematical formality, Petri Net is not as popular as SysML, which is the de facto standard for model-based systems engineering.

Zhong et al. [11] put forth ISTPA (Improved STPA), having contributed the following improvements to the formalization of STPA:

- Using CA-PR (control action that controlled process receives) to update the definition of UCA.
- Defining the loss scenario as the process in which the system emerges UCA and system-level hazard with the involvement of causal factors.
- Using the finite state machine to describe the system behavior and using model checking [12] to identify the loss scenario.

Nevertheless, ISTPA still needs to be instantiated for its application.

Based on ISTPA, STPA-SN (STPA based on SysML/MARTE and NuSMV) is put forth to demonstrate how to solve the aforementioned two drawbacks of HSTPA. Herein, NuSMV [13] is an open-source model checker and MARTE (Modeling and Analysis of Real-Time and Embedded systems) [14] is specification of a UML profile that adds capabilities to UML for the model-driven development of real-time and embedded systems.

In the following sections, STPA-SN is first introduced, then an application example of STPA-SN is presented; next, the differences against HSTPA are discussed; and finally, both the advantages and disadvantages are summarized.

The readers should be familiar with HSTPA, NuSMV, SysML and MARTE, otherwise the following contents might be impenetrable.

2. STPA-SN

STPA-SN consists of four steps, as shown in Figure 1. Each step is briefly introduced as follows:

1. Step 1: Define the purpose of the analysis.

The contents of STPA-SN in this step are basically the same as those in the first step of HSTPA, but the definition of "system-level hazard" is optimized.

HSTPA defines system-level hazards as follows: "A hazard is a system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to a loss". Since a hazard does not necessarily occur under "worst environmental conditions", STPA-SN slightly changes the definition as "System-level hazard is a system state that can lead to a loss under certain conditions" and loss scenario provides such conditions.

Among the outputs of Step 1, the safety constraints (SCs) will be used in Step 4 to create safety properties in temporal logic, i.e., computation tree logic (CTL) or linear temporal logic (LTL) (see NuSMV tutorial in [13]).

2. Step 2: Construct the system model.

This step constructs the analytical model for the identifications of UCAs and loss scenarios.

SysML internal block diagram (IBD) is used to construct the hierarchical control structure model, and the SysML state machine diagram is used to describe the system behaviors. The system behaviors required to identify UCAs and identify loss scenarios are all described by SysML state machine diagrams, including the behaviors of the controller, controlled process, sensor, actuator and their interaction relationships, therein controller behaviors further include the process model and control algorithms.

Due to time-dependent UCAs, the modeling methods need to describe the temporal characteristics. Since MARTE and SysML are closely interrelated, MARTE is used to describe the temporal characteristics.

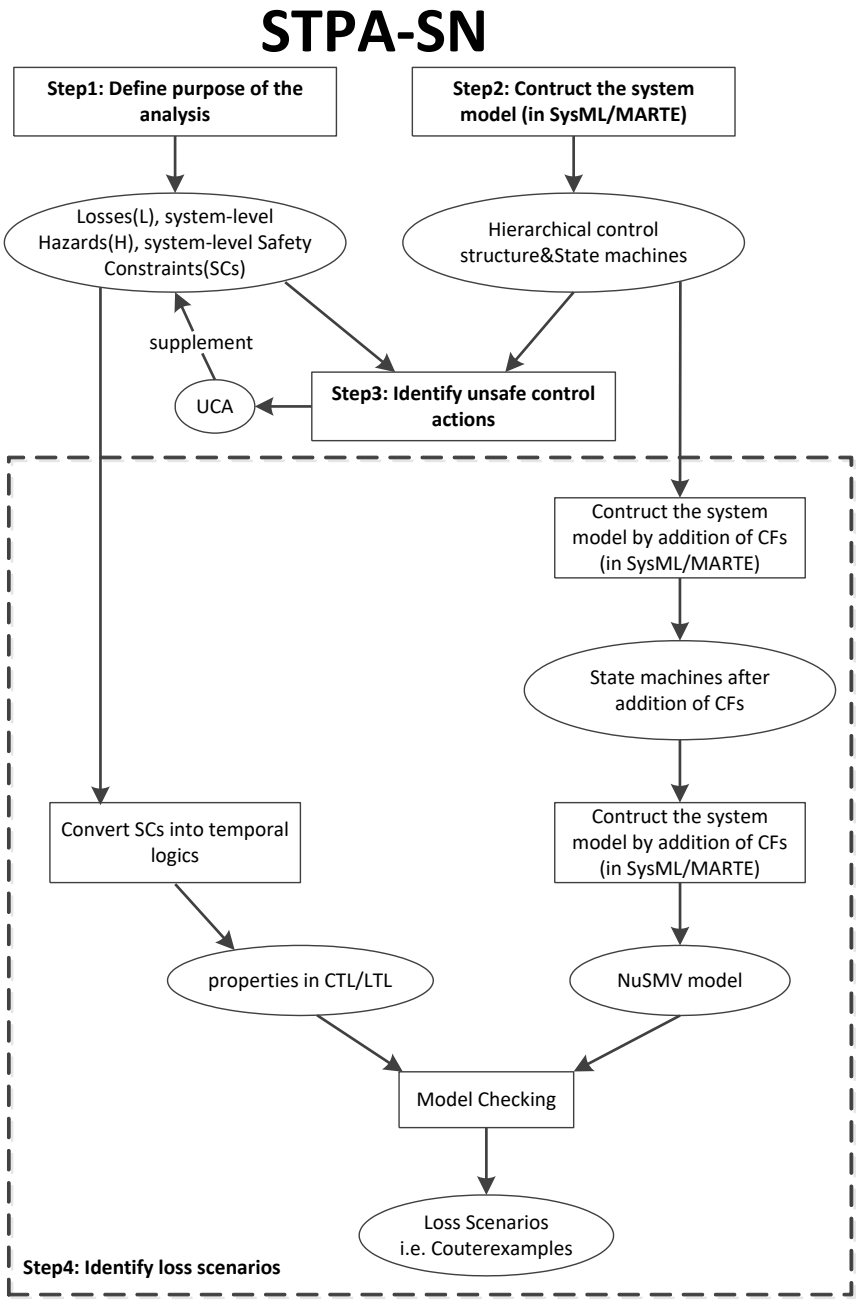

Figure 1. Steps of STPA-SN.

3. Step 3: Identify the unsafe control actions.

The system-level hazard is caused by the state of the controlled process and CA-PR. CA-PR is different from the control action issued by the controller because of the actuator's delay or mishandling. Hence, CA-PR is used in the definition of UCA, instead of the control action issued by the controller.

In STPA-SN, UCA is such a combination as {CA-PR, Type, Context}, which leads to system-level hazards under certain conditions. Therein, Type stands for unsafe type and Context is the state of the system, controlled process and the environment.

When identifying UCAs, all instances of {CA-PR, Type, Context} should be given first, with each instance considered as a potential unsafe control action (PUCA). This process could be automated.

Then, each PUCA is analyzed one by one to determine whether it is a UCA. This process relies heavily on manual analysis, though truth calculation could play a role.

4. Step 4: Identify the loss scenarios.

STPA-SN adopts the definition of loss scenario as the process in which the system emerges UCA and system-level hazard with the involvement of causal factors. There are many causes, such as hardware failure, software defect, data deviation, data error, process model error, component interaction, mode change, environmental change, environmental interference, controlled process change, feedback missing or error, feedback delay, actuator failure or delay, etc.

There are two ways to analyze the models constructed in Step 2: one is simulation, the other is model checking. Simulation is usually visualized, but could not exhaust all the operations of the system model. Hence, theoretically, simulation may miss loss scenario, which may be unaccepted for the safety-critical system. Model checking is a formal method, requiring more expertise. However, model checking can prove whether the system model meets some safety attributes or properties, which is precious to the safety-critical system.

Since STPA highlights the high-level design logics of the system, and control actions are usually discrete logical variables, model checking is selected to identify the loss scenario.

Model checking consists of the finite state system model, the properties in temporal logics and the algorithm that determines whether the system model satisfies the properties. Counterexamples would be produced when a property is not true in the system model [12]. A counterexample provides the state transitions' path from the initial state to the state where the property is violated.

When identifying the loss scenario, the analytical model is converted into NuSMV model, the system safety constraints are converted into safety properties in CTL/LTL and the model checking is carried out by NuSMV. If a counterexample is acquired, the counterexample is a loss scenario. If no counterexample is found, the system model meets the safety properties.

3. STPA-SN Application Example

As the train door control is often utilized in the studies of formalization of STPA, such as [3,4,8], this section also uses train door control to demonstrate the application process of STPA-SN. In order to highlight the features of STPA-SN, many simplifications are made to the example.

3.1. Step 1: Define the Purpose of the Analysis

Assume the train is stationary and the door of the train is aligned with the platform. Loss (L), hazards (H) and safety constraints (SCs) are initialized as follows:

- L: The door squeezes people or things in the doorway, causing damage to people, things or the door.
- H1: When the door is completely opened and there is an obstacle in the doorway, the door receives the command to close.
- SC1: When the door is completely opened and there is an obstacle in the doorway, the close command should not be received by the door.

3.2. Step 2: Construct the System Model

Please note that the system model is the model to be analyzed. Hence, it does not imply that the system model in this section is reasonable or free from defects in terms of design. The same applies to the system model in Section 3.4.1.

3.2.1. Construct the Hierarchical Control Structure

The internal block diagram of the train door control system describes the hierarchical control structure of the system, including four levels, which are the Driver level, Door-Controller level, Actuator/Sensor level and PhysicalDoor level, respectively, as shown in Figure 2.

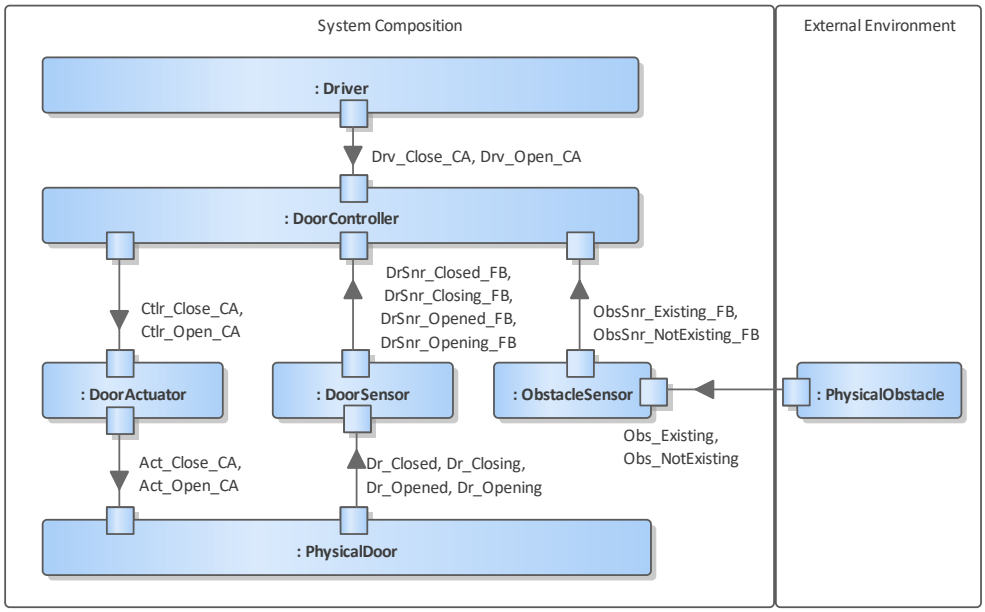

Figure 2. Internal block diagram of the train door control system.

The commands, i.e., control actions, and feedbacks passed between the components are also shown in Figure 2, and their meanings are described in Table 1.

Table 1. The meaning of each command or feedback.

Commands or Feedbacks	Meaning
Drv_Open_CA Drv_Close_CA	Open and close commands sent by the Driver, passed to the DoorController
Ctlr_Open_CA Ctlr_Close_CA	Open and close commands sent by the DoorController, passed to the DoorActuator
Act_Open_CA Act_Close_CA	Open and close commands sent by the DoorActuator, passed to the PhysicalDoor
Dr_Closed Dr_Closing Dr_Opened Dr_Opening	The states of the PhysicalDoor, which, respectively, represent “completely closed”, “in the process of closing”, “completely opened” and “in the process of opening”
DrSnr_Closed_FB DrSnr_Closing_FB DrSnr_Opened_FB DrSnr_Opening_FB	The feedbacks provided by the DoorSensor to the DoorController. They indicate the door states sensed by the DoorSensor, which are “completely closed”, “in the process of closing”, “completely opened” and “in the process of opening”, respectively
ObsSnr_Existing_FB ObsSnr_NotExisting_FB	The feedback from the ObstacleSensor feedbacks to the DoorController indicating whether the ObstacleSensor has sensed an obstacle in the doorway or not
Obs_Existing Obs_NotExisting	Indicate whether there is an obstacle in the doorway

3.2.2. Construct the System State Machine Diagrams

STPA-SN uses MARTE to define clocks, including the two clocks as shown in Figure 3. These clocks are used to describe the time properties of the states and transitions. Their detailed purposes are shown in Table 2.

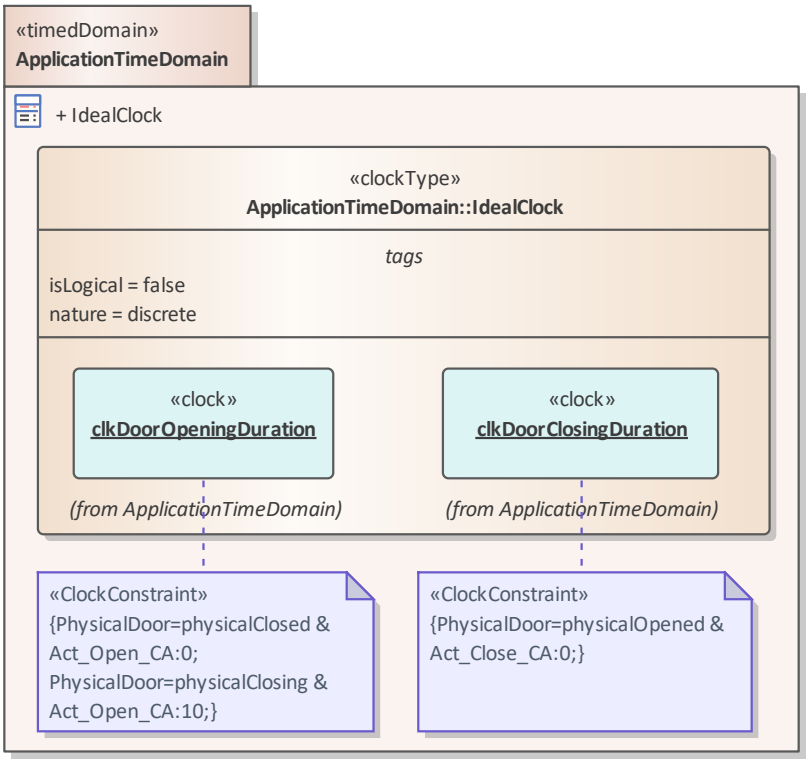

Figure 3. Defining clocks with MARTE.

Table 2. The needed clocks and their purposes.

Clock Name	Purpose
clkDoorOpeningDuration	Records the time taken for the door to open, starting if Dr_Opening is true and ending if Dr_Opened is true
clkDoorClosingDuration	Records the time taken for the door to close, starting if Dr_Closing is true and ending if Dr_Closed is true

Corresponding to 7 parts in Figure 2, the system has 7 state machines, as shown in Figure 4. Every state machine corresponds to a state variable. They are, respectively, the Driver, DoorController, DoorActuator, DoorSensor, PhysicalDoor, ObstacleSensor and PhysicalObstacle.

1. Driver state machine.

As Figure 4a shows, the driver can send Drv_Open_CA to the door controller and changes from driverOutputClose to driverOutputOpen. Similarly, the driver can send out Drv_Close_CA and changes from driverOutputOpen to driverOutputClose.

2. DoorController state machine.

As Figure 4b shows, when Drv_Open_CA is true, the door controller sends Ctlr_Open_CA.

When the obstacle sensor finds an obstacle in the doorway (i.e., ObsSnr_Existing_FB is true), the door controller will send Ctlr_Open_CA to open the door.

When DoorController is in the state of controllerOutputOpenCommand, if the obstacle sensor finds an obstacle in the doorway (i.e., ObsSnr_Existing_FB is true), the door controller will send Ctlr_Open_CA to open the door.

The absence of obstacle (i.e., ObsSnr_NotExisting_FB is true) is a necessary condition for the door controller to send Ctlr_Close_CA. When the door is in the opening process (i.e., DoorSensor = sensedOpening), the controller will not send Ctlr_Close_CA.

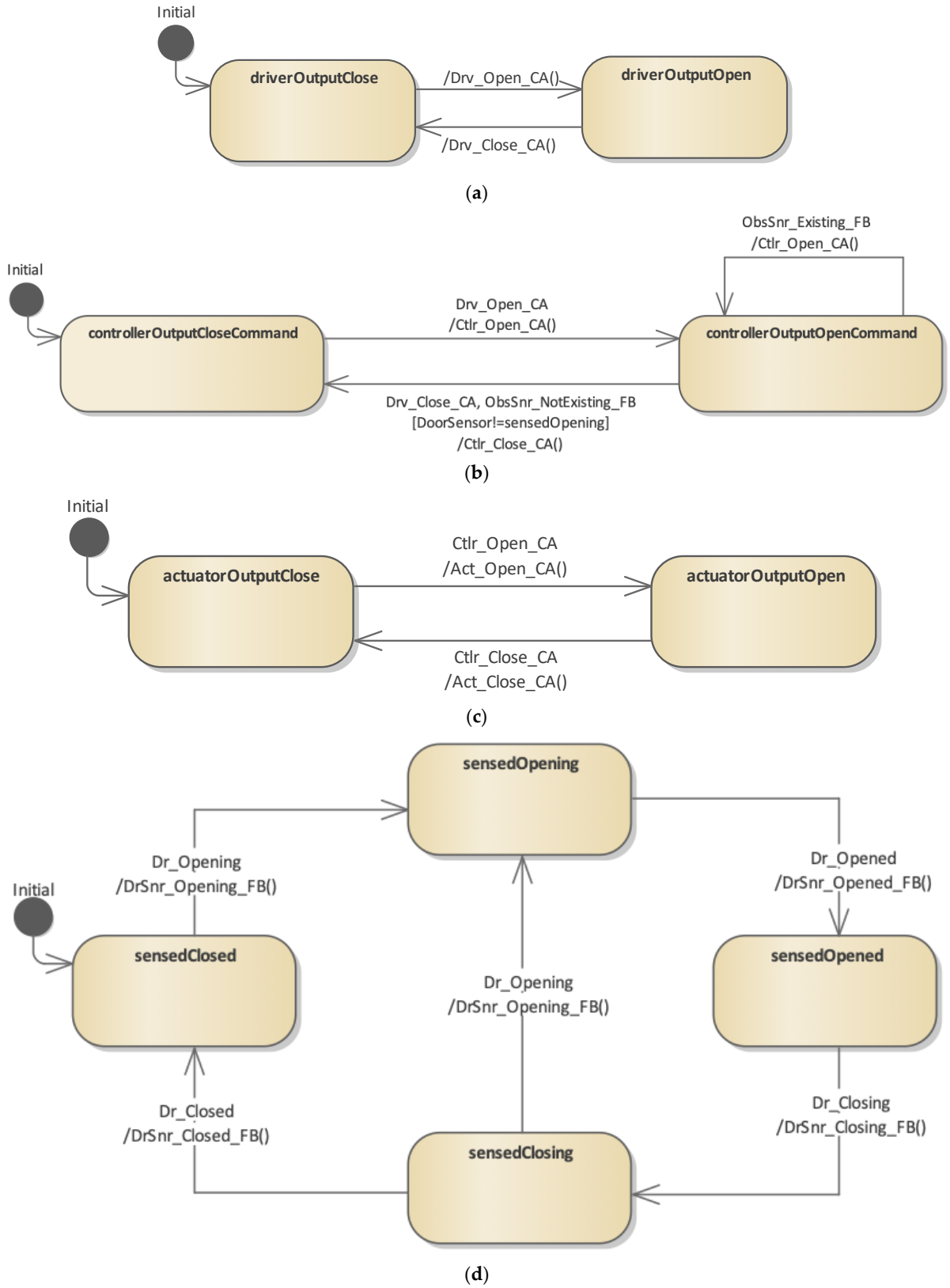

Figure 4. *Cont*.

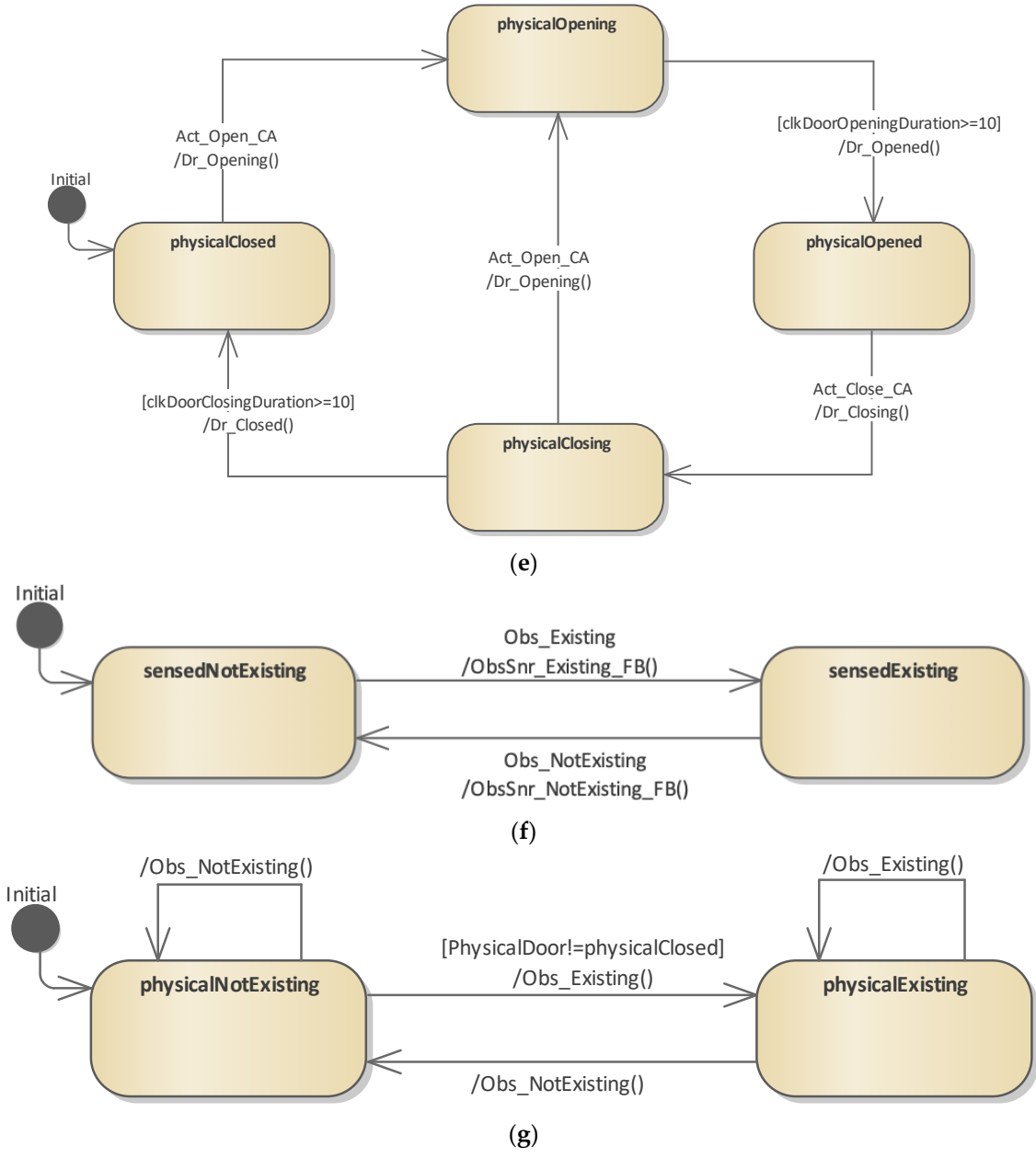

Figure 4. Seven state machines of the system: (**a**) Driver; (**b**) DoorController; (**c**) Actuator; (**d**) DoorSensor; (**e**) PhysicalDoor; (**f**) ObstacleSensor and (**g**) PhysicalObstacle.

3. DoorActuator state machine.

As Figure 4c shows, the actuator sends Act_Open_CA when receives Ctlr_Open_CA and sends Act_Close_CA when receives Ctlr_Close_CA.

4. DoorSensor state machine.

As Figure 4d shows, the door sensor will feed back the corresponding information to the train door controller according to the actual status of the door.

5. PhysicalDoor state machine.

As Figure 4e shows, there are four states of the physical door:

- physicalOpened: The physical door is completely opened.
- physicalClosed: The physical door is completely closed.
- physicalOpening: The physical door is in the process of opening.

- physicalClosing: The physical door is in the process of closing.

Before entering physicalOpened, physicalOpening lasts for at least 10 units of time. Similarly, before entering physicalClosed, physicalClosing lasts for at least 10 units of time.

6. ObstacleSensor state machine.

As Figure 4f shows, the obstacle sensor feeds back to the door controller according to the actual state of the obstacle.

7. PhysicalObstacle state machine.

As Figure 4g shows, the obstacle has two states standing for the presence or absence of an obstacle in the doorway.

3.3. Step 3: Identify the Unsafe Control Actions

3.3.1. Specify the System-Level Hazards with the Information in the System Model

Use the variables and states in the system model to describe the hazards. H1 will be changed to:

H1: PhysicalDoor = physicalOpened and PhysicalObstacle = physicalExisting and Act_Close_CA = TRUE.

3.3.2. Determine CA-PR, Type and Context in the {CA-PR, Type, Context}

The control action, CA, provided by the actuator includes Act_Open_CA and Act_Close_CA. Therefore, the value of CA-PR can be Act_Open_CA or Act_Close_CA.

The types of UCA includes:

- Type1: Not provided;
- Type2: Provided;
- Type3: Too early or too late;
- Type4: Stopped too soon or applied too long.

Therefore, the possible values of Type are Type 1, Type 2, Type 3 and Type 4.

Select the train state variable, PhysicalDoor, and the obstacle state variable, PhysicalObstacle, as the variables of the Context from the state machines in Figure 4. Therefore, Context = (PhysicalDoor, PhysicalObstacle).

The values of PhysicalDoor can be:

- 1: PhysicalClosing;
- 2: PhysicalOpening;
- 3: PhysicalClosed;
- 4: PhysicalOpened.

The values of PhysicalObstacle can be:

- 1: PhysicalNotExisting;
- 2: PhysicalExisting.

Therefore, Context can be: (1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (4, 1) and (4, 2).

3.3.3. Give All the Potential UCAs through Traversal

Each instance of {CA-PR, Type, Context} is considered as a potential UCA (PUCA).

A PUCA that contains a time-independent Type is called a time-independent PUCA, and a PUCA that contains a time-dependent Type is called a time-dependent PUCA.

There are 64 instances of PUCA, some of which are presented in Table 3.

Table 3. PUCAs and their analysis results.

PUCAs					Analysis Results of PUCAs	
ID	**CA-PR**	**Type**	**Context**		**Analysis Method**	**A True UCA or Not**
			PhysicalDoor	**PhysicalObstacle**		
1	Act_Close_CA	Type 1	PhysicalOpened	PhysicalExisting	Manual analysis	Not UCA
2				PhysicalNotExisting	Manual analysis	Not UCA
3			PhysicalClosed	PhysicalExisting	Manual analysis	N/A
4				PhysicalNotExisting	Manual analysis	Not UCA
5			PhysicalOpening	PhysicalExisting	Manual analysis	Not UCA
6				PhysicalNotExisting	Manual analysis	Not UCA
7			PhysicalClosing	PhysicalExisting	Manual analysis	A true UCA, causing a new system-level hazard to be identified
8				PhysicalNotExisting	Manual analysis	Not UCA
9		Type 2	PhysicalOpened	PhysicalExisting	Truth calculation	A true UCA, but equal to H1
10				PhysicalNotExisting	Manual analysis	Not UCA
11			PhysicalClosed	PhysicalExisting	Manual analysis	N/A
12				PhysicalNotExisting	Manual analysis	Not UCA
13			PhysicalOpening	PhysicalExisting	Manual analysis	A true UCA, causing a new system-level hazard to be identified
14				PhysicalNotExisting	Manual analysis	Not UCA
15			PhysicalClosing	PhysicalExisting	Manual analysis	A true UCA, causing a new system-level hazard to be identified
16				PhysicalNotExisting	Manual analysis	Not UCA
17		Type 3	PhysicalOpened	PhysicalExisting	Manual analysis	A true UCA, causing a new system-level hazard to be identified
18				PhysicalNotExisting	Manual analysis	Not UCA
19			PhysicalClosed	PhysicalExisting	Manual analysis	N/A
20				PhysicalNotExisting	Manual analysis	Not UCA
21			PhysicalOpening	PhysicalExisting	Manual analysis	A true UCA, causing a new system-level hazard to be identified
22				PhysicalNotExisting	Manual analysis	Not UCA
23			PhysicalClosing	PhysicalExisting	Manual analysis	A true UCA, causing a new system-level hazard to be identified
24				PhysicalNotExisting	Manual analysis	Not UCA

Table 3. *Cont.*

PUCAs					Analysis Results of PUCAs	
ID	**CA-PR**	**Type**	**Context**		**Analysis Method**	**A True UCA or Not**
			PhysicalDoor	**PhysicalObstacle**		
25		Type 4	PhysicalOpened	PhysicalExisting	Manual analysis	N/A
26				PhysicalNotExisting	Manual analysis	N/A
27			PhysicalClosed	PhysicalExisting	Manual analysis	N/A
28				PhysicalNotExisting	Manual analysis	N/A
29			PhysicalOpening	PhysicalExisting	Manual analysis	N/A
30				PhysicalNotExisting	Manual analysis	N/A
31			PhysicalClosing	PhysicalExisting	Manual analysis	N/A
32				PhysicalNotExisting	Manual analysis	N/A
…	Act_Open_CA	Type 1	…	…	Manual analysis	…
…		Type 2	…	…	Manual analysis	…
49		Type 3	PhysicalOpened	PhysicalExisting	Manual analysis	…
50				PhysicalNotExisting	Manual analysis	…
51			PhysicalClosed	PhysicalExisting	Manual analysis	N/A
52				PhysicalNotExisting	Manual analysis	…
53			PhysicalOpening	PhysicalExisting	Manual analysis	…
54				PhysicalNotExisting	Manual analysis	…
55			PhysicalClosing	PhysicalExisting	Manual analysis	A true UCA, causing a new system-level hazard to be identified
56				PhysicalNotExisting	Manual analysis	…
…		Type 4	…	…	Manual analysis	N/A
64			…	…	Manual analysis	N/A

3.3.4. Analyze Each PUCA to Determine Whether It Is a UCA

Analyze PUCAs one by one. Similar to [3–5], truth calculations could be used. A PUCA is a UCA if it implies that the expression of H1 in Section 3.3.1 is true.

By truth calculation, the PUCA of ID 9 is judged as true UCA, but equal to H1, because the PUCA of ID 9 means CA-PR= Act_Close_CA, Type = Provided, PhysicalDoor = physicalOpened and PhysicalObstacle = physicalExisting.

However, for those PUCAs that could not be judged as true UCAs by truth calculations, they still need to be analyzed manually, since truth calculations only apply to those PUCAs, which are the instances of hazards already identified in Step 1 of STPA-SN. This means the manual identification of UCA is indispensable. Hence, the remaining 63 PUCAs should be analyzed manually.

Since Act_Open_CA and Act_Close_CA are discrete commands and not continuous ones, Type 4 is not applicable. Assuming PhysicalClosed and PhysicalExisting cannot be true simultaneously, the corresponding entries are also not applicable.

Through manual analysis, many of the PUCAs are judged as true UCAs, each of which causes a new hazard to be identified.

For example, the PUCA with ID 55 means that door receives CA-PR = Act_Open_CA with Type 3 when PhysicalDoor = PhysicalClosing and PhysicalObstacle = PhysicalExisting. This PUCA would be a hazard if receiving CA-PR = Act_Open_CA too late; therefore, this PUCA is a true UCA and a new hazard can be generated, marked as H2. By design, the meaning of "too late" is defined as "later more than 3 units of time".

Although every true UCA, which is judged by manual analysis, causes a new hazard to be identified, it does not mean every true UCA will correspond to a new hazard one-to-one; instead, several true UCAs could correspond to a common new hazard.

After the new hazards are identified, the system-level hazards will be updated. Here, the system-level hazards are updated as follows after H2 is added:

- H1: PhysicalDoor = physicalOpened and PhysicalObstacle = physicalExisting and Act_Close_CA = TRUE.
- H2: PhysicalDoor = physicalClosing and PhysicalObstacle = physicalExisting and (the time, from Obs_Existing to Act_Open_CA when physicalClosing = TRUE, is larger than 3 units of time).

Corresponding to the hazards H1 and H2, the safety constraints of the system are determined as follows:

- SC1: PhysicalDoor = physicalOpened, PhysicalObstacle = physicalExisting, Act_Close_CA = TRUE, these three conditions cannot hold simultaneously.
- SC2: PhysicalDoor = physicalClosing, PhysicalObstacle = physicalExisting, (the time, from Obs_Existing to Act_Open_CA when physicalClosing = TRUE, is larger than 3 units of time), these three conditions cannot hold simultaneously.

3.4. Step 4: Identify the Loss Scenarios

3.4.1. Construct System Model by the Addition of CFs

There is no delay caused by either the obstacle sensor or the door actuator in Figure 4c,g. In this section, a two-unit time delay is added to both the obstacle sensor and the door actuator, respectively. Two clocks are added to describe the delays: one is clkObstacleSensorDelay, standing for the delay between Obs_Existing and ObsSnr_Existing, the other is clkActuatorDelay, standing for the delay between Ctlr_Open_CA and Act_Open_CA. Additionally, a third clock named clkDoorOpenSinceObstacle is added to record the time from Obs_Existing to Act_Open_CA when the door is in the state of physicalClosing.

After adding the CFs, the updated system model has five clocks (Figure 5), and the state machines of the door actuator and the obstacle sensor are shown in Figure 6.

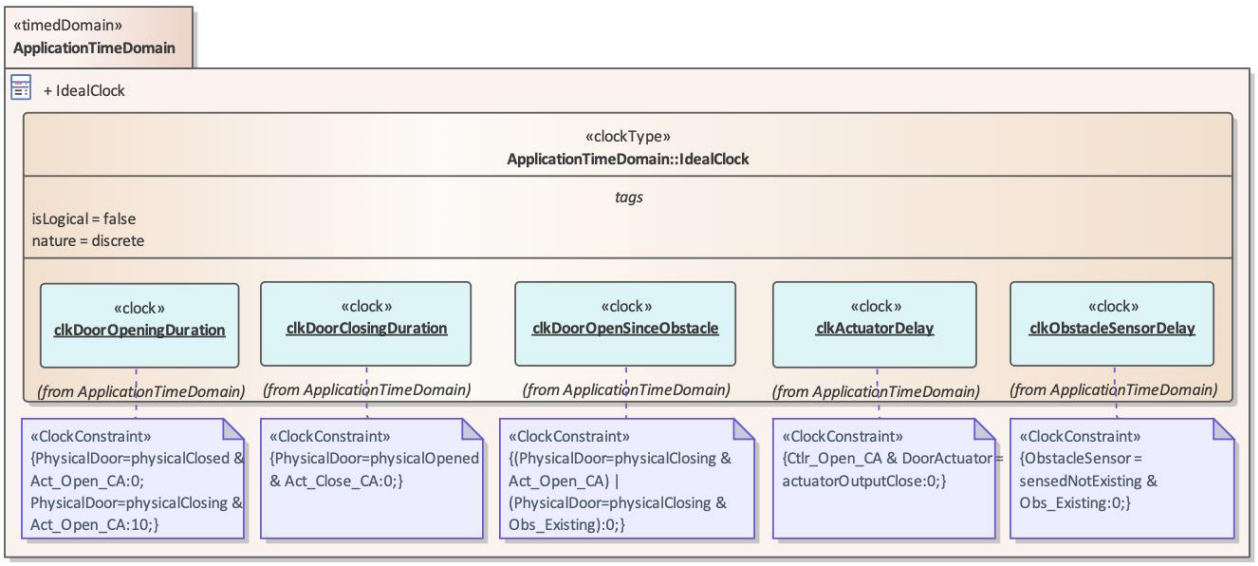

Figure 5. Five clocks after updating the system model.

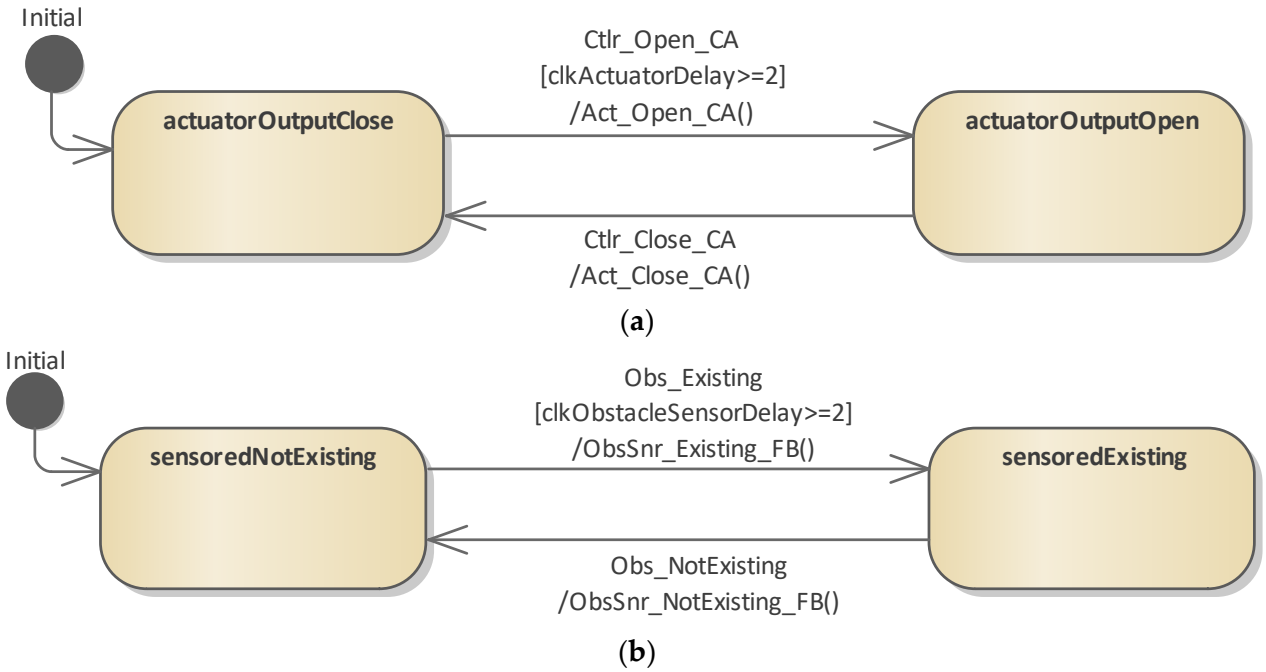

Figure 6. State machines after the addition of CFs. (**a**) The actuator state machine after the addition of delay. (**b**) The obstacle sensor state machine after the addition of delay.

3.4.2. Convert the SCs into Temporal Logics

The two SCs in Section 3.3.4 are changed into specifications (SPEC) in CTL, which is supported by NuSMV for the description of correctness properties:

- SPEC_SC1: AG! (PhysicalDoor = physicalOpened and PhysicalObstacle = physicalExisting and Act_Close_CA);
- SPEC_SC2: AG! (PhysicalDoor = physicalClosing and PhysicalObstacle = physicalExisting and clkDoorOpenSinceObstacle > 3).

Two specifications are described by the SysML requirement diagram, shown in Figure 7, making up the system safety specification, which will be verified for the door control system model marked with the TDStructure.

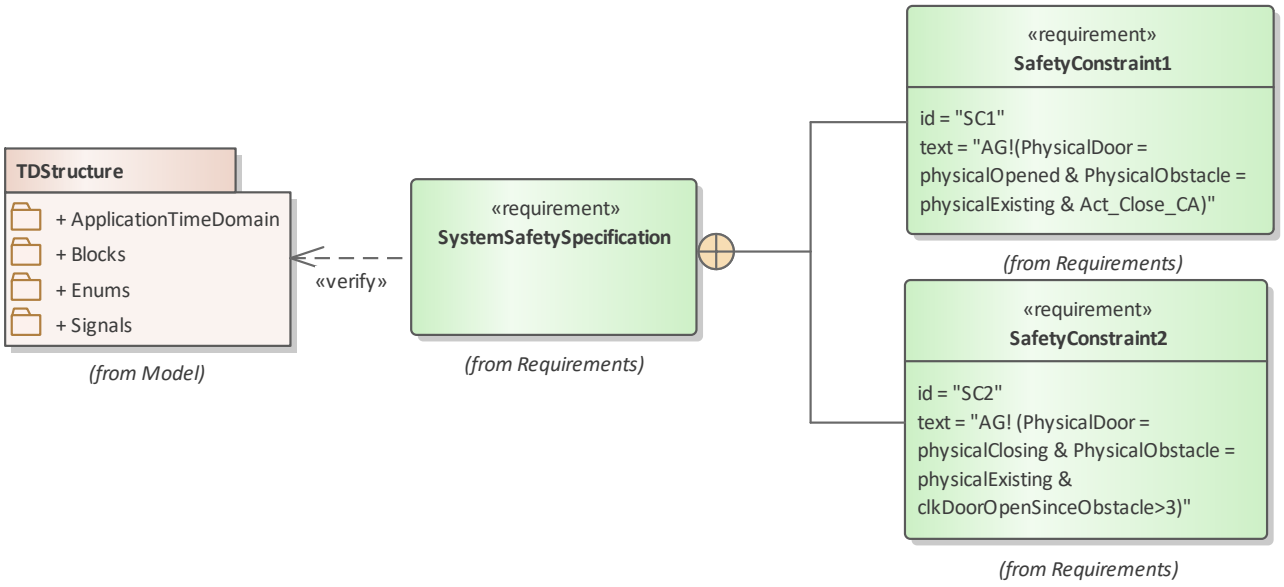

Figure 7. The properties to be checked, described by the SysML requirement diagram.

3.4.3. Construct the System Model in NuSMV

The attributes of the blocks in the BDD (Block Definition Diagram) are converted into system variables in NuSMV. The state machines defined in the blocks are converted into the state machines in NuSMV. The states contained in the SysML state machines are converted into enumerated variables, and the state transitions and constrain conditions are converted into the transition statements with "next" as the keyword in NuSMV.

The signals defined in the SysML model are converted into Boolean variables in NuSMV, clock variables into integer variables, and transition conditions of each variable into transition statements with "next" as the keyword of the corresponding variable.

3.4.4. Model Checking

The NuSMV model is used to analyze the CTL specifications and the results show that the system model does not satisfy the two CTL specifications. Below is the analysis result of SPEC_SC2, which is output as a counterexample, i.e., a loss scenario.

1. In the beginning, the door is completely closed, i.e., in the state of Dr_Closed, and there is no obstacle in the doorway, i.e., in the state of Obs_NoExisting.
2. The driver sends the command of opening the door (Drv_Open_CA), triggering the door controller to send Ctlr_Open_CA.
3. After receiving Ctlr_Open_CA, the door actuator performs the door opening action (Act_Open_CA).
4. Triggered by Act_Open_CA, the door changes its state from physicalClosed to physicalOpening. After 10 units of time, it changes to the state of physicalOpened.
5. The driver sends the command of closing the door (Drv_Close_CA), triggering the door controller to send Ctlr_Close_CA.
6. The door actuator executes the closing action (Act_Close_CA) after receiving Ctlr_Close_CA.
7. The door changes its state from physicalOpened to physicalClosing.
8. The obstacle appears in the doorway and physicalExisting is true. When the door is closing and an obstacle exists, clkDoorOpenSinceObstacle is set to zero and starts the timing.
9. A total of 2 units of time delay happens to the obstacle sensor and clkDoorOpenSinceObstacle increases by 2 units of time.
10. The door controller sends the command of door opening (Ctlr_Open_CA), and the door actuator delays 2 units of time from receiving Ctlr_Open_CA to sending Act_Open_CA. The clkDoorOpenSinceObstacle adds 2 units of time, reaching 4 units of time in total.

At this point, the door of the train is in the physicalClosing state and the obstacle is in the physicalExisting state. The total duration (clkDoorOpenSinceObstacle) from the moment when an obstacle appears in the doorway to the moment when the door receives the command to open the door, is 4 units of time, which is longer than 3 units of time. Hence, SPEC_SC2 is violated and the system becomes hazardous.

4. Discussion

The main differences between STPA-SN and HSTPA are summarized and presented in Table 4. These differences reflect that STPA-SN has more reasonable definitions of terminologies, more disciplined analysis processes and more accurate analysis results.

STPA-SN improves the formalization of STPA, which involving the definitions of UCA and loss scenario, the construction of the analytical model, the identification of the processes of UCAs and loss scenarios.

Table 4. Comparisons between STPA-SN and HSTPA.

HSTPA	STPA-SN
Step 1: Define the purpose of the analysis	Step 1: Define the purpose of the analysis. No major modification.
Step 2: Model the control structure: (a) Boxes and lines are used to describe the hierarchical control structure between the components, including the components and the commands and feedbacks between components. (b) The process model needs to be built exclusively and mainly consists of static information.	Step 2: Construct the system model: (a) The SysML internal block diagram is used to describe the hierarchical control structure and state machine diagrams to describe the behaviors. (b) The process model is included in the SysML state machine diagrams.
Step 3: Identify the unsafe control actions (a) It is not specified whether UCA is a state and whether the CA in the definition of UCA is the one received by the controlled process or the one issued by the controller. (b) Extract the information required for UCA identification from the boxes and lines model in step 2.	Step 3: Identify the unsafe control actions (a) Define UCA as a state and define UCA with the CA received by the controlled process. (b) Extract the information required for UCA identification from the internal block diagram and state machine diagrams.
Step 4: Identify the loss scenarios (a) The loss scenario is defined as describing the causal factors that lead to UCAs and to hazards. (b) Loss scenarios are identified manually.	Step 4: Identify the loss scenarios (a) The loss scenario is defined as the process in which the system emerges UCA and system-level hazard with the involvement of causal factors. (b) SysML state machine diagrams are automatically converted into the NuSMV model, through which the loss scenarios are automatically obtained.

4.1. About the Definition of UCA

While judging whether a PUCA is a UCA, it is required to judge whether the PUCA causes a hazard. The hazard is the state of the controlled process and CA-PR is the direct input of the controlled process. Therefore, it is CA-PR that should be used to decide whether the state of controlled process is hazardous, instead of other CAs, such as CAs issued by controllers.

In Section 3, the physical door is the controlled process. In Figure 2, Act_Close_CA and Act_Open_CA are two CA-PRs. However, Drv_Close_CA, Drv_Open_CA, Ctlr_Close_CA and Ctlr_Open_CA are not CA-PRs, although they are closely related to the two CA-PRs.

If it is not specified that CA-PR should be used in the definition of UCA, either Drv_Open_CA or Ctlr_Open_CA might take the place of Act_Open_CA and leads to error.

For example, in Section 3.3.3, PUCA with ID 55 is judged as a true UCA, therein the CA-PR of Act_Open_CA is used. Corresponding to this UCA, H2 and SC2 are generated. And corresponding to SC2, SPEC_SC2 is generated as follows:

SPEC_SC2: AG! (PhysicalDoor = physicalClosing and PhysicalObstacle = physicalExisting and clkDoorOpenSinceObstacle > 3). Therein, Act_Open_CA is one condition used to calculate clkDoorOpenSinceObstacle (see the description of clkDoorOpenSinceObstacle in Figure 5).

If Drv_Open_CA or Ctlr_Open_CA is used to calculate clkDoorOpenSinceObstacle, it is impossible for STPA-SN to identify the loss scenario that appears in Section 3.4.4, because Act_Open_CA is different from Drv_Open_CA or Ctlr_Open_CA due to a delay of 2 units of time.

Through above explanation, it might be understood why CA-PR must be used in the definition of UCA.

4.2. About the Definition of Loss Scenario

In STPA-SN, loss scenario is defined as the process in which the system emerges UCA and system-level hazard with the involvement of causal factors. In HSTPA, loss scenario is defined as describing the causal factors that lead to UCAs and to hazards.

As shown in Section 3.4.4, STPA-SN can offer a loss scenario with a complete process. However, HSTPA does not have this capability, just pointing out possible causal factors [1] (pp. 42–53). Hence, the loss scenario by STPA-SN is more rigorous and specific than the one by HSTPA, which benefits the design.

If a loss scenario is identified by STPA-SN, the system designer can accurately analyze the causality and rapidly decide the system modification scheme. After modification, the updated system model can soon be verified by STPA-SN, until no loss scenario is identified.

4.3. About the Construction of the Analytical Model

- As for the dynamic modeling:

The analytical model that HSTPA constructs lacks sufficient behaviors. It could be observed from HSTPA's hierarchical control structure, such as Figure 2.12 in [1] (p. 30), and HSTPA's description about the model of the controlled process, which has little information about behaviors. HSTPA believes "The model of the controlled process is the state that the automated controller thinks that the controlled process is in" [1] (p. 182). Hence, it is nearly impossible for HSTPA to present a detailed dynamic loss scenario.

STPA-SN avoids this disadvantage since all the needed behaviors are modeled with state machines (shown in Figures 4 and 5). These behaviors underlie identifying loss scenarios among all the components.

- As for the static modeling:

STPA-SN uses the SysML internal block diagram to describe the hierarchical control structure, such as Figure 2, not only standardized, but also matching SysML state machine diagrams.

4.4. About the Process of Identifying UCAs

Truth calculations could be automated. However, it can be observed from Section 3.3.4 that the role of the truth calculations is limited. Manual analysis occupies the majority of the workload to identify UCAs. In our opinion, manual analysis is inevitable because a new system-level hazard must be judged by a human.

4.5. About the Process of Identifying the Loss Scenario

HSTPA categorizes loss scenarios into two types [1] (p. 43): one type is the loss scenarios that exist among actuators and controlled process, the other is scenarios that exist among sensors and controllers; therefore, it seems impossible to identify loss scenarios caused by the interactions among all those components, i.e., actuators, controlled process, sensors and controllers.

Nevertheless, STPA-SN is competent to find such loss scenarios. The loss scenario in Section 3.4.4 involves the actuator, controlled process, sensors and controllers. This feature benefits from the following:

- All the needed behaviors are modeled, including those of the actuators, controlled process, sensors and controllers;
- NuSMV could analyze all the behaviors simultaneously and automatically.

5. Conclusions

5.1. Two Advantages of STPA-SN

1. Higher efficiency.

The formalization of the STPA is the foundation of the automation. After the STPA-SN is defined, a prototype tool was developed, and the example in Section 3 can be demonstrated through the tool (see the video at https://youtu.be/LQVuWO38tJY, accessed on 16 January 2022. Slight variations exist).

According to the tool's demonstration, the following features could be observed:

- PUCAs could be generated automatically.

- The loss scenario can be generated automatically.
- Hence, it can be concluded that STPA-SN has higher efficiency than HSTPA.

2. Better collaboration with SysML-based systems engineering.

It could be found from the tool that the SysML model is the main interface for users and the NuSMV model is transparent to the users. This feature is attributed to the automatic conversion between the SysML model and the NuSMV model.

If a system already has artifacts in SysML, it is more possible for STPA-SN to reuse these artifacts to construct the analytical model than HSTPA, which uses non-standard language. Therefore, it can be concluded that STPA-SN is more convenient to be incorporated into SysML-based systems engineering than HSTPA.

5.2. Three Disadvantages of STPA-SN

State machines and model checking contribute to the automation of STPA-SN, meanwhile, they also contribute to three disadvantages of STPA-SN:

1. As STPA-SN describes the system behaviors with state machines, which are insufficient to describe the continuous behaviors, STPA-SN does not apply to the hybrid system that comprises both continuous and discrete dynamic behaviors.
2. STPA-SN is not competent to analyze an excessively complicated system, which leads to state explosion, exceeding the ability of model checking used by STPA-SN. However, since STPA is mainly applied to the high-level design of the system, this disadvantage might not be so prominent.
3. In STPA-SN, an SC must be converted into temporal logic for the identification of the loss scenario. If an SC cannot be converted into the temporal logic supported by the model checker, the loss scenario cannot be identified against the SC either.

5.3. Future Research Directions

1. The formalization of time-dependent UCAs.

It can be observed from Section 3.3 that "Type 3: Too early or too late" and "Type 4: Stopped too soon or applied too long" have not been formalized, which brings ambiguity to the identification of UCAs. Asare et al. added timing property to the definition of UCA [2] (p. 153), which is beneficial to solve the problem, but further empirical evidence is still in need.

2. Improving the automatic conversion between the SysML model and the NuSMV model.

Although our prototype tool is capable of converting the SysML model into the NuSMV model, the technique is not general and is still far from maturity.

3. The variants of STPA-SN could be developed.

Similar to STPA-SN, AltaRica or AADL could replace SysML, and other model checkers, such as SPIN and PAT, could replace NuSMV.

4. The STPA that can automatically identify the loss scenario for hybrid system.

Construct the analytical model of a hybrid system and analyze the model through simulation. For example, discrete dynamic behaviors can be modeled with Stateflow and continuous dynamic behaviors can be modeled with the differential/difference equations in Simulink (Stateflow and Simulink are two tools of MATLAB). After modeling, simulation can be used to identify the loss scenario.

Author Contributions: Conceptualization, D.Z. and R.S.; methodology, D.Z. and R.S.; Software, H.G.; validation H.G.; formal analysis, R.S. and H.G.; investigation, D.Z. and H.G.; resources, D.Z. and R.S.; data curation, H.G.; writing—original draft preparation, D.Z. and H.G.; writing—review and editing, D.Z. and T.W.; visualization, H.G.; supervision, D.Z.; project administration, D.Z.; funding acquisition, D.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Civil Aviation Joint Funds under National Natural Science Foundation of China and Civil Aviation Administration of China, grant number U1533201, and by Science and Technology on Reliability and Environmental Engineering Laboratory, grant number 6142004200405.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. STPA Handbook. Available online: https://psas.scripts.mit.edu/home/materials/ (accessed on 28 September 2021).
2. Asare, P.; Lach, J.; Stankovic, J.A. FSTPA-I: A formal approach to hazard identification via system theoretic process analysis. In Proceedings of the 4th IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS), Philadelphia, PA, USA, 8–11 April 2013; pp. 150–159.
3. Thomas, J. Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 29 April 2013.
4. Suo, D. Tool-assisted hazard analysis and requirement generation based on STPA. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 7 December 2015.
5. Souza, F.; Pereira, D.; Pagliares, R.; Nadjm-Tehrani, S. WebSTAMP: A Web Application for STPA & STPA-Sec. In Proceedings of the MATEC Web of Conferences, Wuhan, China, 22–24 October; 2019; pp. 12–24.
6. Zhu, D.; Yao, S. A Hazard Analysis Method for Software-Controlled Systems Based on System-Theoretic Accident Modeling and Process. In Proceedings of the 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 23–25 November 2018; pp. 90–95.
7. Abdulkhaleq, A.; Wanger, S. Integrating state machine analysis with system-theoretic process analysis. In *Software Engineering 2013–Workshopband, Proceedings of Fachtagung des GI-Fachbereichs Softwaretechnik, Aachen, Germany, 26 February–1 March 2013*; Stefan, W., Horst, L., Eds.; Gesellschaft für Informatik e.V.: Bonn, Germany, 2013; pp. 501–514.
8. Chen, M.; Wang, L.; Hu, J. An Extraction Method of STPA Variable Based on Four-Variable Model. In Proceedings of the 3rd International Conference on Intelligent and Interactive Systems and Applications, Hong Kong, China, 29–30 June 2018; pp. 375–381.
9. Rey, F.; Melo, J.; Hirata, C.; Saqui-Sannes, P.; Apvrille, L. Combining STPA with SysML Modeling. In Proceedings of the 14th annual IEEE International Systems Conference (SysCon 2020), Montreal, QC, Canada, 20–23 April 2020; pp. 1–8.
10. Zhu, D.; Yao, S.; Wu, J. Petri nets-based method for component-interaction related hazard identification in computer-controlled systems. In Proceedings of the 2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC), Zhuhai, China, 27–29 March 2018; pp. 1–6.
11. Zhong, D.; Gong, H.; Sun, R. An STPA for Accurately Identifying Loss Scenarios. Available online: http://220.228.59.187/KCMS/detail/detail.aspx?filename=BJHK20210705000&dbcode=CJFD&dbname=CAPJ2021 (accessed on 5 July 2021).
12. Clarke, E.M.; Henzinger, T.A.; Veith, H.; Bloem, R. *Handbook of Model Checking*; Springer: Cham, Switzerland, 2018; pp. 2–3.
13. NuSMV: A New Symbolic Model Checker. Available online: https://nusmv.fbk.eu/ (accessed on 22 December 2021).
14. Modeling and Analysis of Real-Time and Embedded Systems. Available online: https://www.omg.org/omgmarte/ (accessed on 22 December 2021).