Automatic Generation of Component Fault Trees from AADL Models for Design Failure Modes and Effects Analysis

Xiongpeng Hu[1], Jing Liu[1,*], Hui Dou[2], Hongtao Chen[2], and Yuhong Zhang[2,*]
[1]Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
[2]Huawei Technology, Shanghai, China
{51215902089@stu, jliu@sei}.ecnu.edu.cn, {dou.hui, chenhongtao4, john.zhangyh}@huawei.com
*corresponding author

Abstract—Safety analysis is a crucial process in developing safety-critical systems, allowing the identification of potential design issues that may lead to hazards. Automation of this process has become the focus of research in the critical system domain due to the growing complexity of systems. This paper proposes a Component Fault Trees (CFTs) based Failure Mode and Effects Analysis approach for Architecture Analysis and Design Language (AADL) models. First, we propose a methodology for directly generating CFTs from AADL models to display the overall failure behavior of the system. Then we extend the Error Model Annex Version 2 (EMV2) with DFMEA property to express the assessment criteria of error formally, and conduct Design Failure Mode and Effects Analysis (DFMEA) whose core step is guided by CFTs. We discuss our approach with its tool support and evaluate its applicability in driving the design of safety-critical systems through a case study.

Keywords–AADL; EMV2; Component Fault Trees; Failure Mode and Effect Analysis; Safety Analysis

1. Introduction

Safety analysis is a systematic approach to identify and evaluate potential safety issues to ensure the safety of the system [1]. As the number and complexity of electrical and electronic systems increase, safety analysis is encountering more significant challenges in development activities in various industries, such as the preliminary assessment of system safety in the aerospace industry (SAE ARP4761 [2]) and functional safety analysis in the automotive electronics industry (ISO 26262 [3]). This situation results in safety requirements having the same priority as the functional requirements of the system.

Approaches such as Fault Tree Analysis (FTA), Failure Modes and Effects Analysis (FMEA), Failure Modes Effects and Diagnostics Analysis (FMEDA) are often combined to assess system risks [4]–[7]. The ISO 26262 standard requires deductive and inductive methods for safety analysis in system development, and recommends FMEA and FTA, respectively. These safety analysis methods should be performed in the early stages, together with design activities. They are considered one of the most valuable parts of the development process.

Although model-based approaches to safety analysis were introduced in the early 2000s [8], [9], the application of the method is still performed manually due to the high subjectivity of the practitioner, which often leads to incomplete analysis. Furthermore, changes in the system model need to conduct a significant rework on the previous study, thus posing significant challenges to manual efforts.

To simplify the safety analysis process and improve the manageability of analyzing complex systems, Architecture Analysis and Design Language (AADL [10]) and Error Model Annex Version 2 (EMV2 [11]) provide formal and automated support for the design and analysis of safety-critical systems. Several safety analysis tools, such as Functional Hazard Assessment (FHA), Fault Tree Analysis (FTA) and Fault Impact Analysis (FIA), have been implemented in the Open Source AADL Tool Environment (OSATE [12]) to automatically analyze the Architecture Error Model (AEM), i.e., the core AADL architecture models enhanced with EMV2 error models [13]. This way of conducting multiple safety analysis methods on the same model avoids repeated modeling and improves the efficiency of safety analysis. However, several shortcomings of these safety analysis tools could be improved, which include inadequate quantitative analysis of failure causes and incomplete examination of failure effects.

To mitigate the above challenges, this paper presents an automated Design Failure Mode and Effects Analysis (DFMEA) method for AADL models, an improved FMEA method with seven-step. The key step in DFMEA, named failure analysis, is guided by the Component Fault Trees (CFTs), in which smaller fault trees for each system component are created and combined in a hierarchical network structure according to the system architecture [14]. The CFTs offer an effective way for DFMEA to analyze failure effects from error propagation, error behavior of components, and compositional abstraction of system error behavior in terms of its subsystems. Moreover, unlike the traditional FMEA method, DFMEA introduced Action Priority (AP) instead of Risk Priority Number (RPN) to prioritize the risk of failure cause. Hence, DFMEA allows engineers to pay attention to components with high AP at the design phase and iterate or refine the architectural model. We discuss our approach with its tool support and evaluate its applicability in driving the design of safety-critical systems through a simple self-driving system.

Our contributions are summarized as follows:

1) Proposing a methodology for the direct generation of CFTs to display the overall failure behavior of AEM.
2) Extending the EMV2 with DFMEA property to express the assessment criteria of error.
3) Illustrate an approach that can automatically implement DFMEA based on CFTs to generate DFMEA reports and assess system risks.
4) Integrating our approach and tool support in the current practice of safety-critical systems engineering.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 briefly introduces the DFMEA framework and AEM via a case study. Section 4 provides the method of extracting CFTs from AEM. Section 5 details the automated DFMEA generation. Section 6 evaluates the applicability and completeness of the tool through a simple self-driving system. Section 7, concludes this paper and highlights future work.

2. Related Work

Existing work based on EMV2 partially addresses the automation of FMEA. For instance, Larson et al. [15] demonstrated the effectiveness of using an AADL error model for safety Analysis, such as FTA, FIA, and FHA, in a medical device. Both Gu et al. [16] and Liu et al. [17] proposed to extend the Error Model Annex with the FMEA property to implement the quantitative FMEA method. However, these methods only utilize error propagations defined in EMV2 for analysis, which cannot reflect the overall system failure behavior, resulting in incomplete and inadequate results of the FMEA table.

In addition, a mixed approach of FMEA and FTA can be applied. Shafiee et al [18]. and Hidayat et al. [19] proposed an integrated FTA and FMEA model. They start the analysis with an FTA followed by an FMEA to improve the quality of systems. Peeters et al. [20] proposed a method in which both FTA and FMEA are recursive to focus on the most critical parts of complex systems. These ideas are interesting because they combine both methods' strengths to evaluate the system risk.

Nevertheless, Kabir et al. [21] discussed that the CFTs model, which is a modular version of the fault tree that extends classical fault trees by using real components in the tree structure, can be more closely synchronized with the system model than classical FTA. They followed up with a model-based Bayesian Networks generation process to evaluate the reliability of systems [22]. Berres et al. [23] described an approach to generate fault trees automatically by CFTs. Munk et al. [24] also present a semi-automatic safety analysis and optimization process that performs FTA and DFMEA consistent with the model by constructing CFTs. All failure behavior of each component can be described by its CFT. Therefore, our work focuses on the automatic generation of CFTs to conduct a complete DFMEA for AADL models.

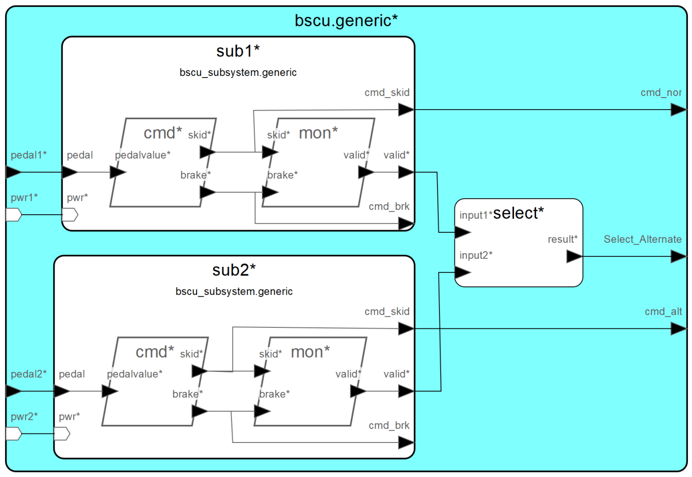


Figure 1. Overview of the BSCU in the WBS

3. DFMEA framework for AADL models

This section first gives an overview of the AADL and EMV2. Then we briefly introduce the CFTs and DFMEA method. Finally, the DFMEA framework for AADL models is proposed.

3.1 AADL and EMV2

The Architecture Analysis and Design Language (AADL) is an effective model-based systems engineering tool for specifying formal architecture in highly integrated systems [10]. AADL can model and describe functional and non-functional properties of embedded hardware and software. It allows for early analysis and verification of systems during development. The aircraft wheel brake system (*WBS*), as an example to describe how the AADL and EMV2 support the safety assessment processes and techniques presented in SAE Standard ARP4761 [25], has a brake system control unit (*BSCU*) that interfaces with the other components and provides commands to the hydraulic pressure, anti-skid system, braking system and annunciation to the pilot. Figure 1 illustrates the AADL graphical component notation for the *BSCU*. The *BSCU* contains a *Select_Alternate* output data that indicates whether we should use the first subsystem (*Select_Alternate* == false) or if the second (backup) subsystem would be used (*Select_Alternate* == true). The two redundant *bscu_subsystem* each contain a *com* and *mon* unit. *com* units produce data from the pedal values, and *mon* units indicate if the values are valid or not. AADL provides the *connection* keyword to connect the subcomponents in a composite system. Figure 2 shows the AEM of *bscu_subsystem*. In this and the next section, we use the *bscu_subsystem* as the primary illustration because it is relatively simple while still enough to illustrate DFMEA.

Error Model Annex Version 2 (EMV2) extends the core AADL with an annex mechanism to support the modeling of component error behavior. It automates safety analysis methods by supporting them with an analyzable architectural error model [11]. EMV2 supports architecture error modeling at three levels of abstraction that focus on error propagation,

```
system bscu_subsystem
        features
                pwr:            requires bus access common::power.generic;
                pedal:          in data port common::command.pedal;
                cmd_skid:       out data port common::command.skid;
                cmd_brk:        out data port common::command.brake;
                valid:          out data port Base_Types::Boolean;
end bscu_subsystem;
system implementation bscu_subsystem.generic
        subcomponents
                mon:            process monitor.i;
                cmd:            process command.i;
        connections
                pedaltocmd:     port pedal -> cmd.pedalvalue;
                brakecmd:       port cmd.brake -> mon.brake;
                brakecmd_ext:   port cmd.brake -> cmd_brk;
                skidcmd_ext:    port cmd.skid -> cmd_skid;
                skidcmd:        port cmd.skid -> mon.skid;
                isvalid:        port mon.valid -> valid;
        annex EMV2 {** use types error_library; use behavior error_library::simple;
        error propagations
                pwr:    in propagation {NoPower};
                valid:  out propagation {NoValue,NoPower};
        end propagations;
        component error behavior
                transitions
                t1: Operational -[pwr {NoPower}]-> Failed;
                propagations
                p1: Failed -[]-> valid {NoValue};
        end component;
        composite error behavior
                states
                [mon.failed and cmd.failed]-> Failed;
                [mon.operational and cmd.operational]-> Operational;
        end composite;
        **};
end bscu_subsystem.generic;
```

Figure 2. Architecture Error Model of bscu_subsystem

component error behavior, and composite error behavior. It also allows combining error behavior specifications to facilitate incremental and extensible automated safety analysis. For more detailed architecture error modeling, please refer to [11]. As seen in Figure 2, *bscu_subsystem* includes an error propagations specification that can propagate *NoPower* and *NoValue* error type through the out data port *valid*, and receive *NoPower* error type through bus access *pwr*. Component error behavior is regarded as a state machine with an *Operational* and a *Failed* state. An incoming propagation *pwr*{*Nopower*} triggers the transition to the *Failed* state. Composite error behavior shows that *bscu_subsystem* remains *Operational* as long as subcomponents *mon* and *cmd* are both in operational, but *Failed* when both subcomponents are failed.

3.2 Component Fault Trees

Component Fault Tree (CFT) [14], a modular version of the classical fault tree, is a Boolean model associated with real components in the system structure. In the CFTs approach, smaller fault trees for each system component are created and combined in a hierarchical structure following the system architecture. This enables closer synchronization of CFTs with the system model [21].
In CFTs, a separate CFT element is attached to a component. CFT utilizes output failure modes to model visible failures at the specific component output port. Input failure modes are used to model the failures the component input port receives. Internal failure modes stand for failures that can occur inside the component. CFT uses Boolean gates, such as AND and OR gates, to model the failure conditions that describe how input

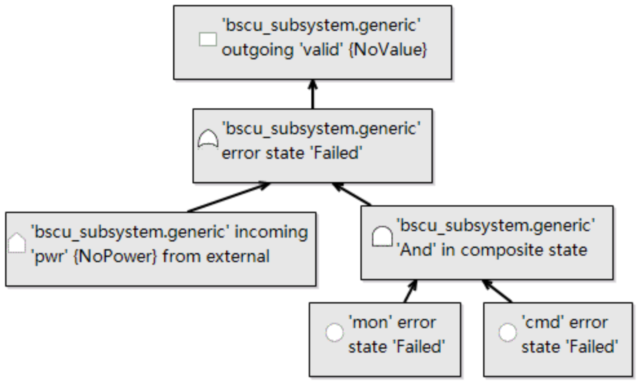

'bscu_subsystem.generic' outgoing 'valid' {NoValue}
'bscu_subsystem.generic' error state 'Failed'
'bscu_subsystem.generic' incoming 'pwr' {NoPower} from external
'bscu_subsystem.generic' 'And' in composite state
'mon' error state 'Failed'
'cmd' error state 'Failed'

Figure 3. CFT of bscu_subsystem.generic

and internal failure modes influence the output failure modes. After modeling the CFT elements of each component, they are arranged in a hierarchical order according to the system architecture, which results in a CFTs model of the whole system. Due to the powerful expression of EMV2, we can define the failure behavior of each component in its EMV2 annex and extract its CFT. Figure 3 show the CTF element of system implementation *bscu_subsystem.generic*. The details process of DFMEA are presented in Section 4.

3.3 Design Failure Modes and Effects Analysis

Failure Modes and Effects Analysis (FMEA) is a ”bottom-up” inductive analysis focusing on technical risks by analyzing system components individually during the design phase to identify and prioritize all potential failure modes based on their impact. This method helps take necessary measures in advance to improve product safety and reliability. The risk score for each failure mode is obtained by multiplying the individual scores for three risk factors:

- Severity (S): represents the severity of the failure effect.
- Occurrence (O): represents the frequency of the failure cause.
- Detection (D): represents the detectability of the failure causes and failure modes that have occurred.

This composite risk, called “Risk Priority Number (RPN), ” is calculated by $RPN = S \times O \times D$. S, O, and D are rated 1-10, with 10 being the highest risk level.
The Automotive Industry Action Group (AIAG) and the Verband Der Automobilindustrie(VDA) have integrated their

TABLE I
AP MATRIX OF DFMEA

		S														
		1	2-3		4-6				7-8				9-10			
O	8-10	L	L	M	M	M	H	H	H	H	H	H	H	H	H	H
	6-7	L	L	L	L	M	M	M	M	H	H	H	H	H	H	H
	4-5	L	L	L	L	L	L	M	M	M	M	H	M	H	H	H
	2-3	L	L	L	L	L	L	L	L	L	M	M	L	L	M	H
	1	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
		1-10	1-4	5-10	1	2-4	5-6	7-10	1	2-4	5-6	7-10	1	2-4	5-6	7-10
		D														

FMEA methods and proposed Design FMEA (DFMEA) to improve the efficiency and accuracy of FMEAs [26]. The main change in this DFMEA approach is a comprehensive revision of the *Severity* (S), *Occurrence* (O), and *Detection* (D) sheet, and the introduction of *Action Priority* (AP) to prioritize actions for improvement.

The AP classifies system risks as High, Medium, or Low priority according to the AP matrix that gives the most weight to severity, some weight to occurrence, and the least weight to detection, as shown in Table I. The disadvantage of the RPN approach is that it multiplies S, O, and D with the same weight, and a low severity problem may be addressed before one with a much higher severity [27]. Using RPN, two problems with the same RPN would have the same priority, even when one has a high severity rating and the other with a high detection rating. With AP, the one with the higher severity rating would be addressed first [26].

The DFMEA analysis process consists of seven steps, as illustrated in Figure 4. The details of each step and its implementation for AEM are further described in Section 5.

3.4 Overview of DFMEA Framework

To automatically conduct a complete DFMEA, we propose a novel generation approach to make DFMEA an essential process in the evaluation phase of the system design life cycle. Figure 5 illustrate the DFMEA framework for AADL Models. We extended EMV2 with DFMEA property to express evaluation criteria and optimization information of failure elements in the component. During the system evaluation phase, the DFMEA tool automatically generates CFTs for the AEM and displays the result as a fault net diagram. Then, the failure analysis step of the DFMEA process is guided by CFTs, and the DFMEA table is output as the result of the DFMEA process. DFMEA table and fault net diagram can be used to evaluate system risk. Based on the characteristics of AADL and EMV2, the model can be refined efficiently because the unmodified parts of the model can be inherited in the refinement step [11].

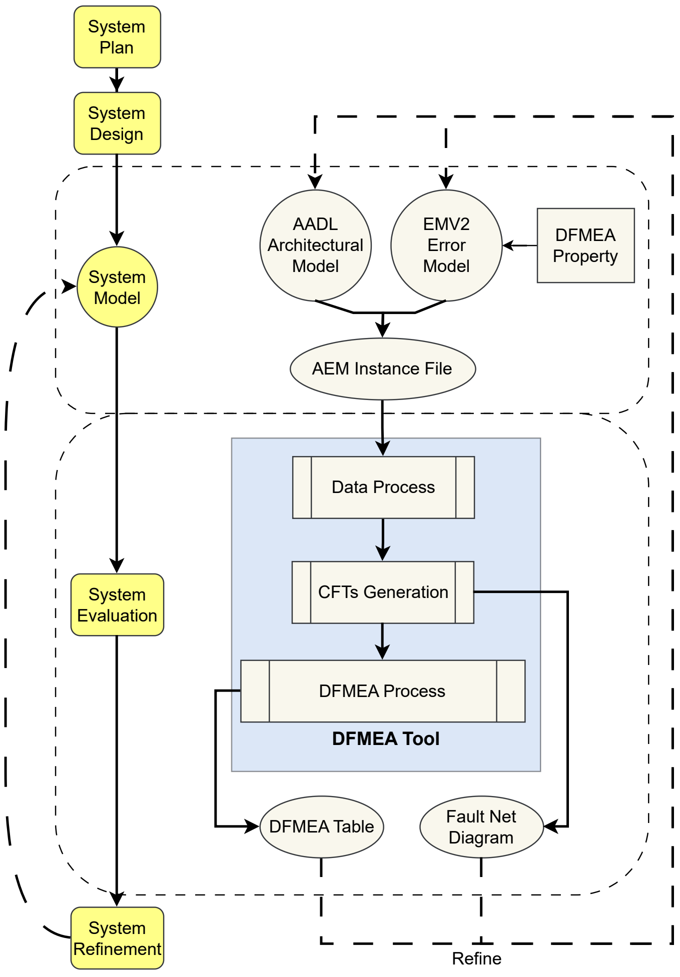


Figure 5. DFMEA Framework for AADL Models

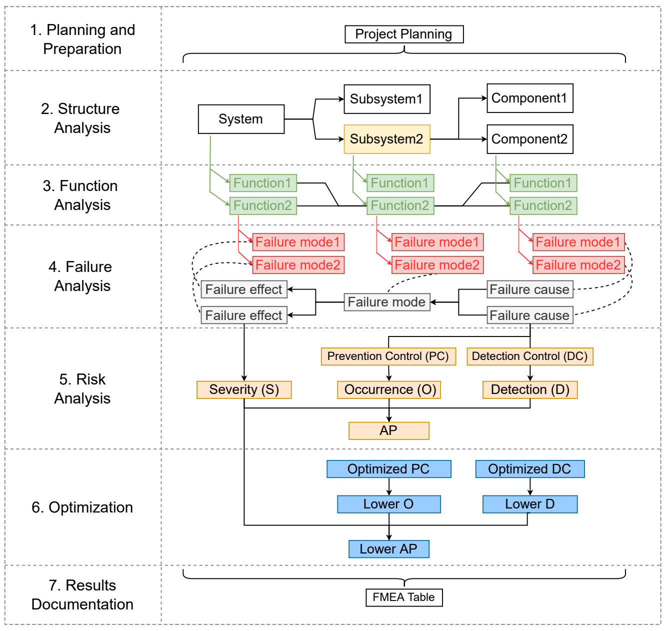


Figure 4. DFMEA Seven Step Process

4. Component Fault Trees Modeling

In this section, we present an approach to extract CFTs from AEM. Figure 6 shows a flowchart of the proposed approach. We first show how to model a CFT element based on the EMV2 annex of a component, and then show how the CFT is arranged to demonstrate failure causality across the whole system.

4.1 Extract CFT from AEM

We first extract and store the information of the subcomponents in AEM and the system itself in the component list. When a component is chosen to be modeled, A query is made to the CFT library database to determine whether the CFT of this component's implementation is available in the library. For example, *monitor.i* is a implementation of the subcomponent mon in Figure 2. If a model is not found in the library,

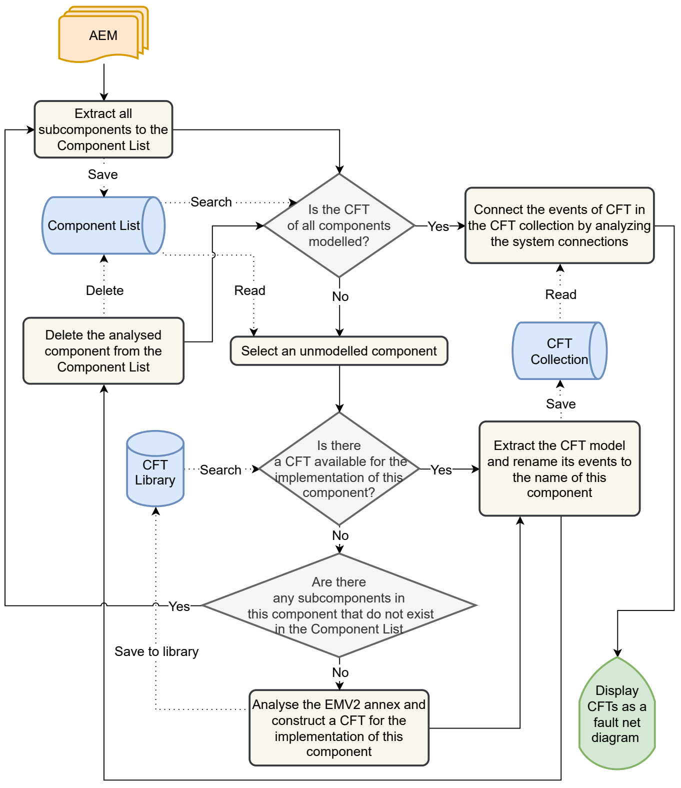

AEM
Extract all subcomponents to the Component List
Save
Component List
Search
Is the CFT of all components modelled?
Yes
Connect the events of CFT in the CFT collection by analyzing the system connections
Delete
Read
No
Read
Delete the analysed component from the Component List
Select an unmodelled component
CFT Collection
Save
CFT Library
Search
Is there a CFT available for the implementation of this component?
Yes
Extract the CFT model and rename its events to the name of this component
No
Are there any subcomponents in this component that do not exist in the Component List
Yes
Save to library
No
Display CFTs as a fault net diagram
Analyse the EMV2 annex and construct a CFT for the implementation of this component

Figure 6. Flowchart of the Generation of CFTs

the implementation of the component is analyzed. Firstly, the analysis identifies whether the component is simple or composite. A component is considered a composite component containing subcomponents. These subcomponents should also be stored in the Component list to analyze their CFT.

For a simple component implementation, *error propagation* and *component error behavior* of EMV2 can define the failure behavior. The analysis identifies the possible internal failure modes of the component through the *error event*, and determines its input and output failure modes through *error propagation*. These failure modes could be of different *error types* such as *NoValue*, *NoService*, and *Nopower*. Figure 7 shows the EMV2 annex example of the components implementation named *monitor.i* and *command.i*. Among them, *monitor.i* has four input failure modes, one internal failure mode, and one output failure mode, while *command.i* has three input failure modes and two output failure modes. *error states* defined in the *component error behavior* are also considered failure modes. We create unique Event nodes that represent these failure modes in the CFT. *transitions* in *component error behavior* can be analyzed to link input failure mode Events and internal failure mode Events to the error state Events. Internal and input failure modes can be used in conditional square brackets ("[]") as a single element or as a combination of Boolean expressions Multiple transitions with single element transition conditions from the initial error state (*Opertional* in this example) to the same error state (*Failed* in this example) means that the Events of these single condition element are connected to the error state Event through an Or gate. Suppose a Boolean expression is expressed in the condition square brackets. In that case, an intermediate Event is created with the corresponding Boolean gate, which acts as a bridge connecting the Events participating in the operation and the target Event. Similarly, *propagations* in *component error behavior* can be analyzed to link error state, input, and internal failure modes Events to the output failure modes Events. The propagation condition is usually empty, indicating that the component will output an error directly when it fails. Error state and output failure modes Events are directly connected through hidden OR gate logic. If the propagation condition is not empty, then an intermediate Event with priority And logic gate will be created to connect the error state Event and condition Event to the target output failure mode Event.

For a composite component, each subcomponent is also extracted to model its CFT separately following the process mentioned above. Moreover, the composite component is first analyzed as a simple component to build a CFT that includes input, output, internal, and error state Events. Then the error state Events of subcomponents will be connected to the error state Event of the composite component through an intermediate Event with corresponding Boolean logic in *composite error behavior* mapping condition. Figure 3 shows that *bscu_subsystem.generic* fails when both subcomponent *mon* and *cmd* have failed. EMV2 includes a consistency checker to ensure that both specifications (*composite error behavior* and *component error behavior*) are consistent [13]. Whenever a new CFT of a component implementation is created, it is saved in the CFT library to facilitate future reuse. The above process is followed when a component implementation's failure behavior is not found in the library. However, if the CFT is available in the library, it is extracted and renamed its events to the name of this component. For example, there are two redundant subsystems *sub1* and *bscu2* in *BSCU*, both of which use the implementation of *bscu_subsystem.generic*. When the CFT of *bscu_subsystem.generic* is modeled and stored in the CFT library, *sub1* and *sub2* can efficiently establish corresponding CFT by copying the CFT and renaming its events with their own name.

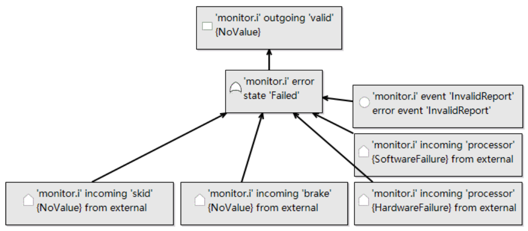
4.2 Component Fault Trees Composition

The constructed CFT of each component is stored in the CFT collection to prepare for the CFTs model composition. Once the CFT models for all the components are created, the approach composes them according to their connection defined in the architectural model. In this process, if the output of component X is connected to the input of component Y, A one-to-one connection is created between the X's output and corresponding Y's input failure mode Events, which have the same *error type*. Such connections guarantee the propagation of failure from one component to another. Figure 8 illustrates the composited CFTs of *bscu_subsystem.generic*. The connection *isvalid*, *brakecmd*, *skidcmd* in Figure 2 contribute to the three connections in CFTs. It should be noted that both *'cmd' error state 'Failed'* and *'mon' error state 'Failed'* are unique

```
error propagations
        processor: in propagation {SoftwareFailure, HardwareFailure};
        brake: in propagation {NoValue};
        skid: in propagation {NoValue};
        valid: out propagation {NoValue};
end propagations;

component error behavior
events
        InvalidReport: error event;
transitions
        terrinvalidreport: Operational -[InvalidReport]-> Failed;
        noskid: Operational -[skid {NoValue}]-> Failed;
        nobrake: Operational -[brake {NoValue}]-> Failed;
        terrfromplatformsoft: Operational -[processor {SoftwareFailure}]-> Failed;
        terrfromplatformhard: Operational -[processor {HardwareFailure}]-> Failed;
propagations
        p1: Failed -[]-> valid {NoValue};
end component;
```

EMV2 annex in process implementation monitor.i

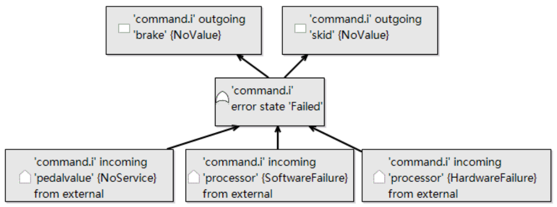

'monitor.i' outgoing 'valid' {NoValue}
'monitor.i' error state 'Failed'
'monitor.i' event 'InvalidReport' error event 'InvalidReport'
'monitor.i' incoming 'processor' {SoftwareFailure} from external
'monitor.i' incoming 'skid' {NoValue} from external
'monitor.i' incoming 'brake' {NoValue} from external
'monitor.i' incoming 'processor' {HardwareFailure} from external

CFT of monitor.i

```
error propagations
        pedalvalue: in propagation {NoService};
        brake: out propagation {NoValue};
        skid: out propagation {NoValue};
        processor: in propagation {SoftwareFailure, HardwareFailure};
end propagations;

component error behavior
transitions
        terrfrompedal: Operational -[pedalvalue {NoService}]-> Failed;
        terrfromplatformsoft: Operational -[processor {SoftwareFailure}]-> Failed;
        terrfromplatformhard: Operational -[processor {HardwareFailure}]-> Failed;
propagations
        p1: Failed -[]-> brake {NoValue};
        p2: Failed -[]-> skid {NoValue};
end component;
```

EMV2 annex in process implementation command.i


'command.i' outgoing 'brake' {NoValue}
'command.i' outgoing 'skid' {NoValue}
'command.i' error state 'Failed'
'command.i' incoming 'pedalvalue' {NoService} from external
'command.i' incoming 'processor' {SoftwareFailure} from external
'command.i' incoming 'processor' {HardwareFailure} from external

CFT of command.i

Figure 7. EMV2 annex and its CTF of the components in bscu_subsystem

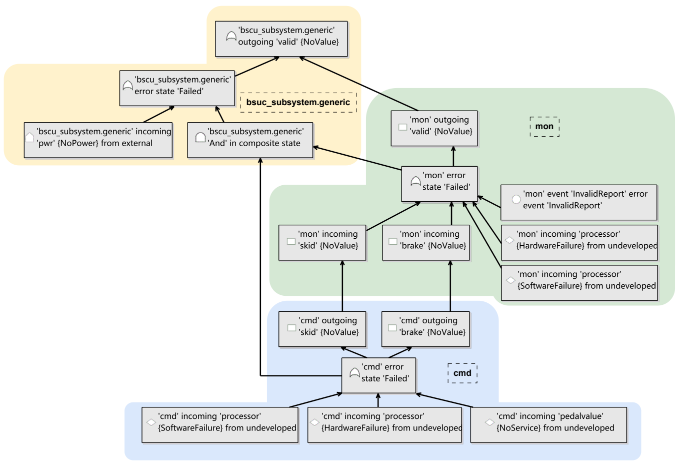

'bscu_subsystem.generic' outgoing 'valid' {NoValue}
'bscu_subsystem.generic' error state 'Failed'
bsuc_subsystem.generic
'bscu_subsystem.generic' incoming 'pwr' {NoPower} from external
'bscu_subsystem.generic' 'And' in composite state
'mon' outgoing 'valid' {NoValue}
mon
'mon' error state 'Failed'
'mon' event 'InvalidReport' error event 'InvalidReport'
'mon' incoming 'skid' {NoValue}
'mon' incoming 'brake' {NoValue}
'mon' incoming 'processor' {HardwareFailure} from undeveloped
'mon' incoming 'processor' {SoftwareFailure} from undeveloped
'cmd' outgoing 'skid' {NoValue}
'cmd' outgoing 'brake' {NoValue}
'cmd' error state 'Failed'
cmd
'cmd' incoming 'processor' {SoftwareFailure} from undeveloped
'cmd' incoming 'processor' {HardwareFailure} from undeveloped
'cmd' incoming 'pedalvalue' {NoService} from undeveloped

Figure 8. Fault Net Diagram of bscu_subsystem.generic

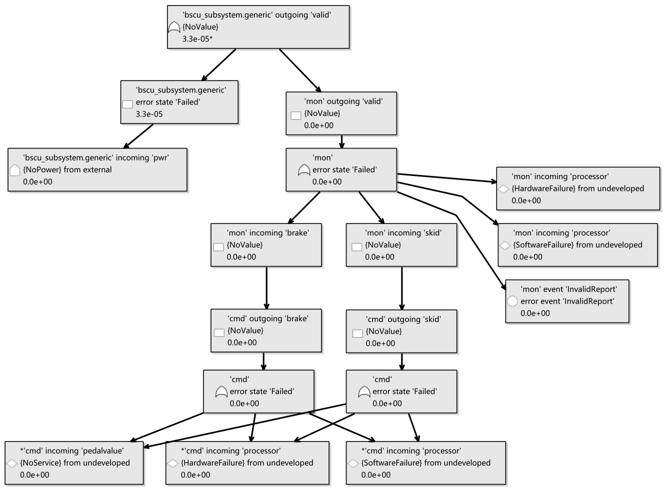

'bscu_subsystem.generic' outgoing 'valid' {NoValue} 3.3e-05*
'bscu_subsystem.generic' error state 'Failed' 3.3e-05
'mon' outgoing 'valid' {NoValue} 0.0e+00
'bscu_subsystem.generic' incoming 'pwr' {NoPower} from external 0.0e+00
'mon' error state 'Failed' 0.0e+00
'mon' incoming 'processor' {HardwareFailure} from undeveloped 0.0e+00
'mon' incoming 'brake' {NoValue} 0.0e+00
'mon' incoming 'skid' {NoValue} 0.0e+00
'mon' incoming 'processor' {SoftwareFailure} from undeveloped 0.0e+00
'mon' event 'InvalidReport' error event 'InvalidReport' 0.0e+00
'cmd' outgoing 'brake' {NoValue} 0.0e+00
'cmd' outgoing 'skid' {NoValue} 0.0e+00
'cmd' error state 'Failed' 0.0e+00
'cmd' error state 'Failed' 0.0e+00
*'cmd' incoming 'pedalvalue' {NoService} from undeveloped 0.0e+00
*'cmd' incoming 'processor' {HardwareFailure} from undeveloped 0.0e+00
*'cmd' incoming 'processor' {SoftwareFailure} from undeveloped 0.0e+00

Figure 9. Fault Contributor Trace of bscu_subsystem.generic

after building the CFT for *cmd* and *mon*, and will not be created repeatedly but retain its connection relationship when creating the CFT for *bscu_subsystem.generic*.

This combination produces a CFTs model that represents the failure behavior of the entire system. For comparison, we have also used the FTA tool for the fault contributor trace analysis of the same example. It can be seen in Figure 9 that, the *composite component behavior* for *bscu_subsystem.generic* is missing from the FTA analysis results, and *'cmd' error state 'Failed'* is repeated. Therefore, It can be concluded that the CFTs model represents components' complete failure behavior and interaction without omitting any information.

Unlike classical fault trees, CFTs allow multiple top events to be defined and create unique events corresponding to system component failures. Therefore, CFTs look more like a directed acyclic graph with a causal relationship between system component failures. Moreover, safety analyses such as FTA and FMEA can be conducted by tracing the causal relationship guided by CFTs.

5. Automated DFMEA Generation

In this section, we first extend the EMV2 with the DFMEA property. We then detailed each step of the DFMEA analysis process and how the CFT models guided the DFMEA.

PLANNING AND PREPARATION (STEP1)		
Company Name:	Subject:	
Engineering Location:	DFMEA Start Date:	DFMEA ID Number:
Customer Name:	DFMEA Revision Date:	DFMEA Responsibility:
Model Year(s)/Program(s):	Cross-Function Team:	Confidentiality Level:

STRUCTURE ANALYSIS (STEP2)			FUNCTION ANALYSIS (STEP3)			FAILURE ANALYSIS (STEP4)			
1. Next Higher Level	2. Focus Element	3. Next Lower Level or Characteristic Type	1. Next Higher Level Function and Requirement	2. Focus Element Function and Requirement	3. Next Lower Level Function and Requirement or Characteristic	1. Failure Effects (FE) to the Next Higher Level Element and/or End User	Severity (S) of FE	2. Failure Mode (FM) of the Focus Element	3. Failure Cause (FC) of the Next Lower Element or Characteristic

(a) Steps 1 to 4

DFMEA RISK ANALYSIS (STEP5)						DFMEA OPTIMIZATION (STEP6)													
Current Prevention Control (PC) of FC	Occurrence (O) of FC	Current Detection (DC) of FC or FM	Detection (D) of FC/FM	DFMEA AP	Filter Code (Optional)	DFMEA Preventive Action	DFMEA Detection Action	Responsible Person's Name	Target Completion Date	Status	Action Taken with Pointer to Evidence	Completion Date	Severity (S)	Occurrence (O)	Detection (D)	DFMEA AP	Remarks		

(b) Steps 5 to 6

Figure 10. AIAG-VDA DFMEA Form Sheet

```
property set DFMEA_Prop is

    Head: record (
        CompanyName: aadlstring;                -- company name
        EngineeringLocation: aadlstring;        -- engineering location
        CustomerName: aadlstring;               -- customer name
        ModelYearProgram: aadlstring;           -- model years/programs
        Subject: aadlstring;                    -- subject
        DFMEAStartData: aadlstring;             -- DFMEA start date
        DFMEARevisionData: aadlstring;          -- DFMEA revision date
        CrossFuncTeam: aadlstring;              -- cross-functional team
        DFMEAID: aadlstring;                    -- DFMEA ID number
        DesignResponsibility: aadlstring;       -- design responsibility
        ConfidentialityLevel: aadlstring;       -- confidentiality level
        FocusComponent: aadlstring;             -- focus component
    )applies to (all);

    DFMEA: record (
        FailureDescription: aadlstring;         -- description of the failure mode
        Function:aadlstring;    -- description of the function violated by the failure mode
        Severity: aadlinteger 1 .. 10;          -- severity of failure effect
        PC: aadlstring;                         -- current prevention control of failure cause
        Occurrence: aadlinteger 1 .. 10;        -- occurrence of failure cause
        DC: aadlstring;         -- current detection control of failure cause/failure mode
        Detection: aadlinteger 1 .. 10;         -- detection of failure cause/failure mode)
    )applies to ({emv2}**error type, {emv2}**type set, {emv2}**error behavior state,
        {emv2}**error propagation, {emv2}**error event, {emv2}**error flow);

    Optimization: list of record (
        OptPC: aadlstring;                      -- DFMEA optimized preventive action
        OptDC: aadlstring;                      -- DFMEA optimized detection action
        ResponsPerson: aadlstring;              -- responsible person's name
        TargetCompletionData: aadlstring;       -- target completion date
        Status: aadlstring;                     -- status
        Evidence: aadlstring;                   -- action taken with pointer to evidence
        CompletionData: aadlstring;             -- completion date
        OptOccurrence: aadlinteger 1 .. 10;     -- optimized occurrence
        OptDetection: aadlinteger 1 .. 10;      -- optimized detection
        Notes: aadlstring;                      -- notes
    )applies to ({emv2}**error type, {emv2}**type set, {emv2}**error behavior state,
        {emv2}**error propagation, {emv2}**error event, {emv2}**error flow);

end DFMEA_Prop;
```

Figure 11. EMV2's Extended Property Set DFMEA_Prop

5.1 Error Model Annex extension

As shown in Figure 4, the DFMEA method considers system structure, function, and risk analysis in the analysis steps, leading to a more robust FMEA development process. The DFMEA tool has been implemented as a plugin for OSATE. The outcomes of the complete DFMEA analysis are automatically generated in the DFMEA form sheet. Figure 10 displays the sheet template for DFMEA, where steps 3-6 contain additional information for analysis that needs to be attached to the error element in EMV2. To conduct

```
system implementation integration.functional
    properties
    DFMEA_Prop::Head => [Subject => "Automotive System.";
                        DFMEAStartData => "2022/3/1"; DFMEARevisionData => "2023/3/1";
                         FocusComponent => "speed_ctrl";];
  annex EMV2 {**
    properties
    DFMEA_Prop::DFMEA => [FailureDescription => "Automatic driving function failure";
                          Function=>"Integrate information and output command";
                          Severity=>8; ] applies to FailStop;
           **};
    end integration.functional;
```

Figure 12. Example of using DFMEA_Prop

a complete DFMEA analysis for AADL models, we have defined a property set named *DFMEA_Prop* shown in Figure 11, which offers additional information appended to the AADL components and the error elements and supports the output of complete DFMEA reports. The *Head* record can be used to record planning and preparation information in step 1. The *DFMEA* record can describe a failure mode by its function, specific description, and initial risk assessment information. The *Optimization* list of records can record the process of multiple optimizations for a failure mode. The system designer can use this property set by declaring 'with DFMEA_Prop' at the beginning of an AADL file. Figure 12 shows how to attach properties to the AADL components and the error elements. It is essential to note that, the process of building CFTs does not analyze these properties. Even without recording additional information using the *DFMEA_Prop* property set, the tool can conduct structure and core failure analysis of DFMEA on the AADL architecture error model.

5.2 DFMEA process for AADL Model

The DFMEA analysis process consists of seven steps, as illustrated in Figure 4. In this subsection, we focus on implementing each step of the DFMEA analysis process for the AADL model.

Step 1

This step includes forming a DFMEA team and compiling the necessary documentation and resources. The DFMEA team and project planning should be organized as a DFMEA sheet header, as depicted in Figure 10(a). This information can be completed as an *aadlstring* type within the *Head* record of the *DFMEA_Prop* property set and supplied to the top-level system implementation in AADL, as illustrated in the *integration.functional* implementation in Figure 12. In this step, we should select a component as the Focus Component, and the subsequent steps will revolve around it. The output of step 1 renders the table head of the DFMEA report by extracting the *Head* property attached to the AEM;

Step 2

In the structure analysis step, the model is subdivided into systems, subsystems, and components by hierarchical relationships and represented as a structure tree. From the AADL architecture model, the FMEA structure tree can be automatically generated by analyzing the *subcomponents* declarations in component implementations. The output of step 2 will be presented in the table of the step 1 column with the Focus Component as the centre and its upper and lower components on the left and right, respectively. The generated structure tree is to be reviewed and discussed by the safety experts and the function owners to improve the functional architecture of the system.

Step 3

In the function analysis step, each structural element is analyzed with regard to its functions in the system. A function net describes the interaction of the functions of several structural elements. The failure modes are derived from the functions of the components, which may include function loss, function degradation, and unintended function. Figure 4 shows that the function net depicts the functional dependency between components, while the failure net illustrates the causal relationship between the failure modes of each component. Typically, the failure analysis follows the function analysis. Both processes are performed independently, which may lead to inconsistencies between the two analysis outcomes. Two failure modes with a causal relationship in the failure net may have no dependency relationship in the function net.

In general, the failure of a function will inevitably fail other components' functions that depend on it, which is similar to the propagation of the failure. Therefore, our method first assigns functional properties to the corresponding failure elements through *DFMEA* record in *DFMEA_Prop*, so that the corresponding Events in the CFTs represent the failure of this function. For example, *FailStop* is the failure of the function "the car can safely perform automatic driving" as illustrated in Figure 12. It should be noted that the input failure modes of the components are not considered as the failure of the component's function, and different failure modes may violate the same function. After the corresponding Events have been assigned function properties, the functional

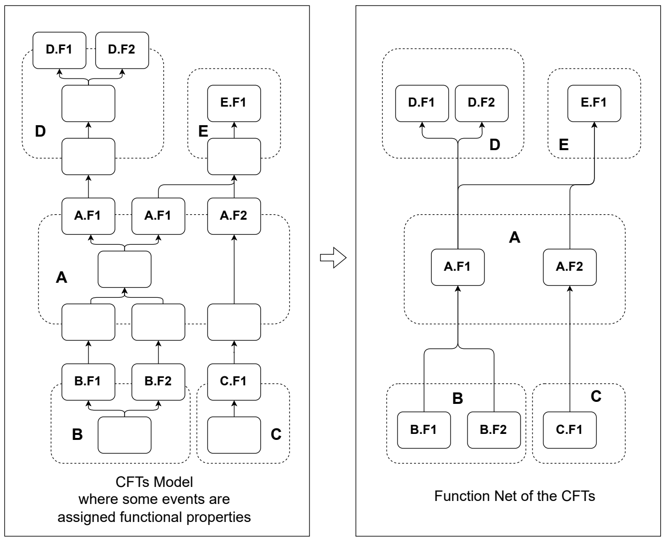


Figure 13. Example of converting CFTs to Function Net

dependencies between components can be presented in CFTs. Figure 13 illustrates an example of converting the CFTs model to function net.

The function net can be converted from CFTs through the following two steps:

1) Events without function properties in CFTs are removed, and the incoming and outgoing edges of this Event are directly connected. These edges are also removed if the Event only has incoming or outgoing edges.
2) Events that violate the same component's function are merged, and their connections to other events are preserved.

The output of step 3 will list all functions of the Focus Component as well as the dependent relationship with other component functions. Similar to the previous step, the result needs to be reviewed and discussed in order to improve the requirements, the structure tree, and the function net.

Step 4

The fourth failure analysis step requires us to build the failure net to analyze the effects and causes of the failure modes in the Focus component. From the perspective of the Focus Component, failure modes that cause the component to enter the failure mode are considered failure causes. In contrast, failure modes affected by the Focus Component's failure mode are considered failure effects. As discussed in Section 4, The CFTs model is a directed acyclic graph that reflects failure causality among system components. Therefore, The CFTs model of the system implementation can be used as the failure net for this step analysis. It should be noted that the FMEA typically considers only single failures. However, benefiting from the Boolean logic in CFTs, we can also generate causality between failures connected with one or more AND gates. Multiple failure modes of a component violating the same function and the corresponding failure causes and effects are displayed on the same line as the function output in step 3.

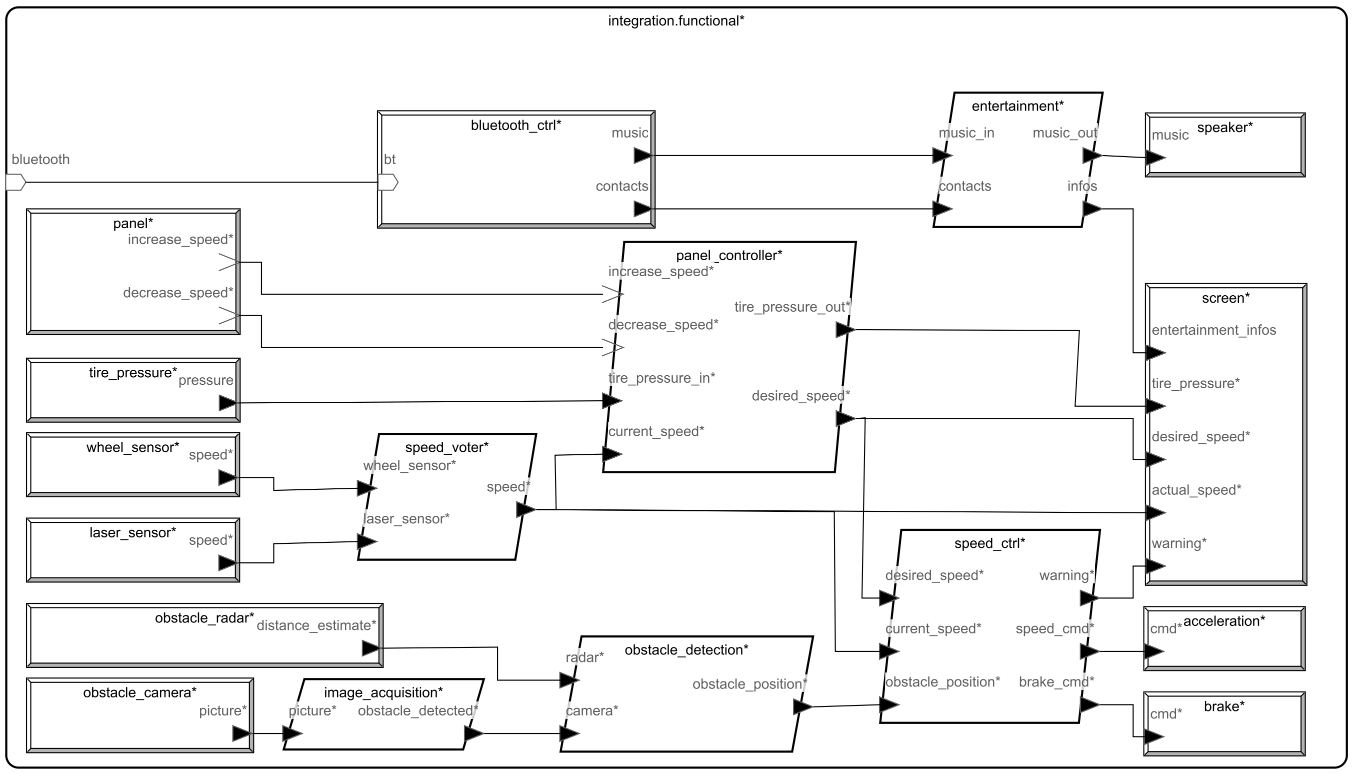


Figure 14. AADL Model of Self-Driving Car System

Step 5

In the risk analysis, it is necessary to evaluate each failure mode, failure cause, and failure effect for Focus Component to assess the risk. The *DFMEA* record in Figure 11 can also assign relevant risk analysis information to the respective error elements. Factor S can be assigned to the failure effects, and factors O and D can be assigned to the failure causes. Take Figure 12 as an example. The S=8 is assigned to the *FailStop error state* of *integration.functional*; For a failure cause, the effects of the failure can be obtained by failure analysis. To assess the AP value for a failure cause, we used the largest S of its failure effects in combination with its own O, D.

Step 6

The objective of the optimization step is to develop actions that reduce risk by improving the product. Suppose satisfactory results are not achieved during the review of the effectiveness of current measures or the risk assessment. In that case, further preventive and detection measures can be developed to reduce O and D, thus reducing the assessment level of the AP. The optimization step can be performed multiple times, and relevant information can be assigned to the relevant failure causes through the *Optimization* record list property.

Step 7

An FMEA process is not finished until step 7 has been completed. This step involves documenting results, summarizing, reporting, and communicating. The FMEA tool automatically generates an EXCEL report that contains the analysis results of the first six steps.
Each time the AEM is changed or extended, requirements or optimization are added or modified. The DFMEA process can be repeated immediately, since DFMEA reports and the CFTs model are generated automatically. Therefore, the effects of changes in the AEM are directly visible in the generated DFMEA report and CFTs. AEM is always consistent with DFMEA and CFTs This automatic safety analysis method allows a faster and earlier safety assessment of the system.
The automated DFMEA process relies on the CFTs model which can provide a more complete description of system faults as discussed in Section 4. One might still get an incomplete FMEA that lacks detail due to incorrect error modeling. Therefore, it's essential to use our automated tools as aids in the analysis process and to complement them with expert judgment to ensure a thorough and reliable assessment of system reliability and safety.

6. Case Study: A Simple Self-Driving Car System

The book *AADL in Practice* provides a practical guide on using AADL for analyzing system latency and safety [28]. It includes a case study of an automotive system and demonstrates complete architectural modeling and error modeling for simple self-driving cars. This case study is easy to understand and therefore is utilized as an example to illustrate the DFMEA process and showcase the function of the DFMEA tool.
Figure 14 presents a graphical representation of the AADL model for a self-driving car system. The AEM of this system

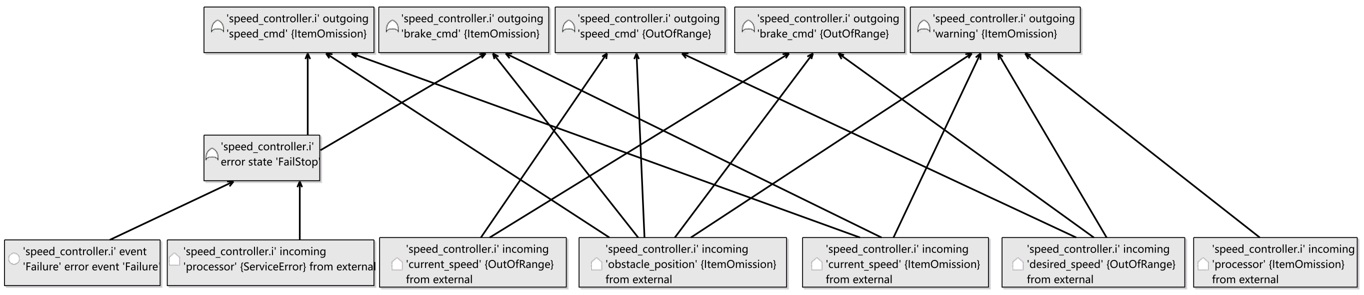

'speed_controller.i' outgoing 'speed_cmd' {ItemOmission}
'speed_controller.i' outgoing 'brake_cmd' {ItemOmission}
'speed_controller.i' outgoing 'speed_cmd' {OutOfRange}
'speed_controller.i' outgoing 'brake_cmd' {OutOfRange}
'speed_controller.i' outgoing 'warning' {ItemOmission}
'speed_controller.i' error state 'FailStop'
'speed_controller.i' event 'Failure' error event 'Failure'
'speed_controller.i' incoming 'processor' {ServiceError} from external
'speed_controller.i' incoming 'current_speed' {OutOfRange} from external
'speed_controller.i' incoming 'obstacle_position' {ItemOmission} from external
'speed_controller.i' incoming 'current_speed' {ItemOmission} from external
'speed_controller.i' incoming 'desired_speed' {OutOfRange} from external
'speed_controller.i' incoming 'processor' {ItemOmission} from external

Figure 15. CFT of speed_controller.i

TABLE II
FUNCTION AND ERROR ELEMENTS

Component Name	Function	Failure Description	Error element
integration.system	The car can safely perform automatic driving	Automatic driving function failure	FailStop
acceleration	Accelerates according to the acceleration command	Unable to accelerate due to abnormal acceleration command	FailStop
speed_ctrl	Integrate information and output command	Unable to conduct speed control	FailStop
		Loss of braking command	brake_cmd.ItemOmission
		Loss of acceleration command	speed_cmd.ItemOmission
	Calculate a reasonable acceleration deceleration command	Invalid braking command	brake_cmd.OutOfRange
		Invalid acceleration command	speed_cmd.OutOfRange
	Send warning information in emergency situations	Loss of warning information	warning.ItemOmission
obstacle_detection	Determines whether there is an actual obstacle on the road	Obstacle recognition failure	FailStop
	Output the obstacle position	Loss of obstacle_position	obstacle_position.ItemOmission
speed_voter	Eliminates potential bad values, and outputs a consistent speed value	Unable to handle and output the correct speed	FailStop
	Output the correct speed to the controller	Loss of speed from speed voter	speed.ItemOmission
		Invalid speed from speed voter	speed.OutOfRange
obstacle_camera	Sends the raw picture to a software component	Loss of picture from the camera	picture.itemomission
obstacle_radar	Detects an obstacle on the road	Loss of information from the Radar	distance_estimate.ItemOmission
		Invalid distance sent by the radar	distance_estimate.OutOfRange
wheel_sensor	Indicates the vehicle speed	Invalid value from wheel sensor	speed.OutOfRange
		Loss of speed from wheel sensor	speed.ItemOmission
laser_sensor	Indicates the vehicle speed	Invalid value from laser sensor	speed.OutOfRange
		Loss of speed from laser sensor	speed.ItemOmission

is described in detail in [28], [29]. The system is designed to detect obstacles on the road by capturing images while in operation. Two speed sensors, namely the *wheel_sensor* and *laser_sensor*, detect the car's actual speed and initiate the acceleration or braking functions accordingly. The car activates the brakes if an obstacle is detected through the *obstacle_radar* or *camera*. On the other hand, if no obstacle is detected, the acceleration function can be activated. The speed and brake commands are controlled by process *speed_ctrl* whose implementation is *speed_controller.i*, and Figure 15 shows the CFT element of *speed_controller.i*. Moreover, the car features entertainment functions and a screen that provides feedback to the passenger. The passenger can also set the desired speed using a panel.

Table II defines the functions and corresponding error elements of some components in the system. We can see that the *speed_ctrl* component has three functions. The error state (*FailStop*) and two outgoing error propagations (*brake_cmdItemOmission* and *speed_cmdItemOmission*) are the failure of function "Integrate information and output command." *integration.system* is the implementation of the self-driving car system, whose functions can be regarded as security goals. It will fail in *FailStop* state when both subcomponent *acceleration* and *brake* fail in *FailStop* state. We assign these function and failure descriptions to corresponding error elements through *DFMEA_Prop* property. Then we can conduct automatic DFMEA analysis on the AEM of self-driving car system with *speed_ctrl* as the Focus Component.

Table III and IV show the partial DFMEA reports. The step 2 column shows three threads that are the subcomponent of *speed_ctrl*. One of the component's functions, "Integrate information and output command," is shown in the step 3 column. This function relies on the functions of *obstacle_detection* and *speed_voter*, and support for the function of component *acceleration* to achieve the safety goal of *integration.system*. Step 4 column shows the failure analysis of *speed_cmdItemOmission*. The failure modes in the table are represented by corresponding failure descriptions in Table II. This failure mode will lead to the failure of *acceleration* (S=7) and cause system failure (S=8). For better presentation, the right-hand side of step 4 only lists the initial failure causes that may have contributed to this failure mode. The largest S among failure effects is 8, which is used to evaluate the AP for these failure causes in step 5. We can see that failures of *obstacle_camera* and *obstacle_radar* have a high AP value, which indicate that certain failure modes have a significant impact and are likely to occur. To reduce the system risk, these failure modes need to be prioritized in the optimization steps by adding effective preventive measures and detection protocols.

TABLE III
PORTION OF DFMEA REPORT

STRUCTURE ANALYSIS (STEP2)			FUNCTION ANALYSIS (STEP3)			FAILURE ANALYSIS (STEP4)			
1. Next Higher Level	2. Focus Element	3. Next Lower Level or Characteristic Type	1. Next Higher Level Function and Requirement	2. Focus Element Function and Requirement	3. Next Lower Level Function and Requirement or Characteristic	1. Failure Effects (FE) to the Next Higher Level Element and/or End User	Severity (S) of FE	2. Failure Mode (FM) of the Focus Element	3. Failure Cause (FC) of the Next Lower Element or Characteristic
integration_functional	speed_ctrl	speed_ctrl.brake_thr	"acceleration" : Accelerates according to the acceleration command	Integrate information and output command	"obstacle_detection" : Output the obstacle position	<"acceleration" : Unable to accelerate due to abnormal acceleration command	7	Loss of acceleration command	>>>>>"obstacle_camera" : Loss of picture from the camera
		speed_ctrl.accel_thr							>>>"obstacle_radar" : Loss of information from the radar
		speed_ctrl.warning_thr	<"integration_functional" : The car can safely perform automatic driving		"speed_voter" : Output the correct speed to the controller	<<"integration_functional" : Automatic driving function failure	8		>>>"wheel_sensor" : Loss of speed from wheel sensor
									>>>"laser_sensor" : Loss of speed from laser sensor

TABLE IV
CONTINUED TABLE OF TABLE III

(STEP4)	DFMEA RISK ANALYSIS (STEP5)					DFMEA OPTIMIZATION (STEP6)					
3. Failure Cause (FC) of the Next Lower Element or Characteristic	Current Prevention Control (PC) of FC	Occurrence (O) of FC	Current Detection (DC) of FC or FM	Detection (D) of FC/FM	DFMEA AP	DFMEA Preventive Action	DFMEA Detection Action	Severity (S)	Occurrence (O)	Detection (D)	DFMEA AP
>>>>>"obstacle_camera" : Loss of picture from the camera	NONE	6	NONE	6	H	Automatic calibration of the camera	Real-time monitoring	8	4	4	M
>>>"obstacle_radar" : Loss of information from the radar	NONE	6	NONE	6	H	Automatic calibration of the radar	Real-time monitoring	8	4	4	M
>>>"wheel_sensor" : Loss of speed from wheel sensor	NONE	5	NONE	6	M	Data redundancy	Fault diagnosis	8	3	4	L
>>>"laser_sensor" : Loss of speed from laser sensor	NONE	5	NONE	6	M	Data redundancy	Fault diagnosis	8	3	4	L

7. SUMMARY AND PERSPECTIVES

This paper proposes a CFTs model based FMEA method for AADL models. We propose a methodology for directly generating CFTs from AADL models to display the overall failure behavior of the system. The CFTs model represents components' complete failure behavior and interaction without omitting any information. Safety analyses such as FTA and FMEA can be conducted by tracing the causal relationship guided by CFTs. We extend the EMV2 with DFMEA property to formally express the assessment criteria of error. The proposed DFMEA process guide by CFTs is well integrated into the system design life cycle. The effects of changes in the AEM are directly visible in the DFMEA report and CFTs generated, and AEM is always consistent with DFMEA and CFTs. This automatic safety analysis method allows a faster and earlier system safety assessment. We discuss our approach with its tool support and evaluate its applicability in driving the design of safety-critical systems through a case study. The analysis results of DFMEA provide designers with a comprehensive view of system failures and can complement and optimize the system design.

In our future work, we aim to expand the safety analysis methods available for the AADL model. Specifically, we plan to utilize Bayesian Network analysis to identify the most critical failure causes among the essential items identified by DFMEA. Additionally, we intend to implement FMEDA in AADL, enabling the quantitative analysis of random hardware failures in the system.

LIST OF ABBREVIATIONS

AADL	Architecture Analysis and Design Language
AEM	Architecture Error Model
AP	Action Priority
BSCU	Brake System Control Unit
CFT	Component Fault Tree
DFMEA	Design Failure Mode and Effects Analysis
EMV2	Error Model Annex Version 2
FHA	Functional Hazard Assessment
FIA	Fault Impact Analysis
FMEA	Failure Mode and Effects Analysis
FMEDA	Failure Modes Effects and Diagnostic Analysis
FTA	Fault Tree Analysis
OSATE	Open Source AADL Tool Environment
RPN	Risk Priority Number

ACKNOWLEDGMENT

This work was partly supported by the National Key Research and Development under Project 2019YFA0706404, the NSFC Project 61972150, Shanghai Trusted Industry Internet Software Collaborative Innovation Center, and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] C. A. Ericson *et al.*, *Hazard analysis techniques for system safety*. John Wiley & Sons, 2015.
[2] SAE International, *Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*. SAE International, 1996.

[3] Standard, Safety, "ISO-26262: Road vehicles functional safety.(2016)," *Retrieved Oct*, 2016.

[4] X. Han and J. Zhang, "A combined analysis method of fmea and fta for improving the safety analysis quality of safety-critical software," in *2013 IEEE International Conference on Granular Computing (GrC)*. IEEE, 2013, pp. 353–356.

[5] R. Messnarz and H. Sporer, "Functional safety case with fta and fmeda consistency approach," in *Systems, Software and Services Process Improvement*, X. Larrucea, I. Santamaria, R. V. O'Connor, and R. Messnarz, Eds. Cham: Springer International Publishing, 2018, pp. 387–397.

[6] M. Shafiee, E. Enjema, and A. Kolios, "An integrated fta-fmea model for risk analysis of engineering systems: a case study of subsea blowout preventers," *Applied Sciences*, vol. 9, no. 6, p. 1192, 2019.

[7] M. Takahashi, Y. Anang, and Y. Watanabe, "A safety analysis method for control software in coordination with fmea and fta," *Information*, vol. 12, no. 2, p. 79, 2021.

[8] A. Joshi, S. P. Miller, M. Whalen, and M. P. Heimdahl, "A proposal for model-based safety analysis," in *24th Digital Avionics Systems Conference*, vol. 2. IEEE, 2005, pp. 13–pp.

[9] P. Struss and A. Fraracci, "Automated model-based fmea of a braking system," *IFAC Proceedings Volumes*, vol. 45, no. 20, pp. 373–378, 2012.

[10] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): An introduction," Carnegie-Mellon Univ Pittsburgh PA United States, Tech. Rep., 2006.

[11] P. Feiler, J. Hudak, J. Delange, and D. P. Gluch, "Architecture fault modeling and analysis with the error model annex, version 2," Carnegie-Mellon Univ Pittsburgh PA United States, Tech. Rep., 2016.

[12] Carnegie Mellon University, "Open source AADL tool environment documentation," [Online], https://osate.org/.

[13] J. Delange and P. Feiler, "Architecture fault modeling with the AADL error-model annex," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2014, pp. 361–368.

[14] B. Kaiser, P. Liggesmeyer, and O. Mäckel, "A new component concept for fault trees," in *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33*, 2003, pp. 37–46.

[15] B. Larson, J. Hatcliff, K. Fowler, and J. Delange, "Illustrating the AADL error modeling annex (v. 2) using a simple safety-critical medical device," *ACM SIGAda Ada Letters*, vol. 33, no. 3, pp. 65–84, 2013.

[16] B. Gu, Y. Dong, and X. Wei, "A qualitative safety analysis method for AADL model," in *2014 IEEE Eighth International Conference on Software Security and Reliability-Companion*. IEEE, 2014, pp. 213–217.

[17] Y. Liu, G. Shen, Z. Huang, and Z. Yang, "Quantitative risk analysis of safety–critical embedded systems," *Software Quality Journal*, vol. 25, pp. 503–527, 2017.

[18] M. Shafiee, E. Enjema, and A. Kolios, "An integrated fta-fmea model for risk analysis of engineering systems: a case study of subsea blowout preventers," *Applied Sciences*, vol. 9, no. 6, p. 1192, 2019.

[19] A. A. Hidayat, M. Kholil *et al.*, "The implementation of fta (fault tree analysis) and fmea (failure mode and effect analysis) methods to improve the quality of jumbo roll products," in *IOP Conference Series: Materials Science and Engineering*, vol. 453, no. 1. IOP Publishing, 2018, p. 012019.

[20] J. Peeters, R. J. Basten, and T. Tinga, "Improving failure analysis efficiency by combining fta and fmea in a recursive manner," *Reliability engineering & system safety*, vol. 172, pp. 36–44, 2018.

[21] S. Kabir, "An overview of fault tree analysis and its application in model based dependability analysis," *Expert Systems with Applications*, vol. 77, pp. 114–135, 2017.

[22] S. Kabir and F. Campean, "A model-based reliability analysis method using bayesian network," in *Advances in Computational Intelligence Systems: Contributions Presented at the 20th UK Workshop on Computational Intelligence, September 8-10, 2021, Aberystwyth, Wales, UK 20*. Springer, 2022, pp. 483–495.

[23] A. Berres, T. Bittner, and M. ZELLER, "Towards standardizing the generation of component fault trees through the engineering life cycle," in *European Safety and Reliability Conference*, 2019.

[24] P. Munk, A. Abele, E. Thaden, A. Nordmann, R. Amarnath, M. Schweizer, and S. Burton, "Semi-automatic safety analysis and optimization," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[25] J. Delange, P. Feiler, D. Gluch, and J. Hudak, "Aadl fault modeling and analysis within an arp4761 safety assessment," *Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2014-TR-020*, 2014.

[26] AIAG-VDA, *AIAG & VDA FMEA Handbook*. Automotive Industry Action Group and Verband der Automobilindustrie, 2019.

[27] M. Barsalou, "Investigation into a potential reduction of fmea efforts using action priority," *Management and Production Engineering Review*, vol. 13, 2022.

[28] J. Delange, "AADL in practice: Become an expert in software architecture modeling and analysis," *Reblochon Development Company*, 2017.

[29] J. Delange, "aadl-book: Model and examples for the AADL book," [Online], 2017, https://github.com/juli1/aadl-book/tree/master.