软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.007137]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 AADL 的混合关键系统随机错误与突发错误安全性分析 *

魏晓敏[1], 董云卫[2], 孙聪[1], 李兴华[1], 马建峰[1]

[1](西安电子科技大学 网络与信息安全学院,陕西 西安 710071)

[2](西北工业大学 计算机学院,陕西 西安 710072)

通讯作者: 董云卫, E-mail: yunweidong@nwpu.edu.cn

摘 要: 许多复杂的嵌入式系统都是混合关键系统(mixed-criticality system,简称 MCS).MCS 通常需要在指定的关键性(criticality)等级状态下运行,但是它们可能会受到一些危害的影响,这些危害可能会导致随机错误和突发错误,进一步导致执行线程中止,甚至导致系统故障.目前的研究仅集中于对 MCS 的可调度性分析,未能进一步分析系统安全性,未能考虑线程之间的依赖关系.本文以随机错误和突发错误为研究对象,提出一种集成故障传播分析的基于架构的 MCS 安全分析方法.使用架构分析和设计语言(Architecture Analysis and Design Language,简称 AADL)刻画构件依赖关系.为了弥补 AADL 的不足,创建新的 AADL 属性(AADL 突发错误属性),并提出新的线程状态机(突发错误行为线程状态机)语义来描述带有突发错误的线程执行过程.为了将概率模型检查应用于安全分析,提出模型转换规则和组装方法,从 AADL 模型推导出 PRISM 模型.建立了两个公式,分别获得定量安全属性以验证故障发生的概率,以及定性安全属性以生成相应的正例来求出故障传播路径来进行故障传播分析.最后,以动力艇自动驾驶仪(power boat autopilot,简称 PBA)系统为例,验证了该方法的有效性.

关键词: 混合关键系统;突发错误;模型转换;安全性分析;概率模型检验

中图法分类号: TP311

中文引用格式: 魏晓敏,董云卫,孙聪,李兴华,马建峰. 基于 AADL 的混合关键系统随机错误与突发错误安全性分析.软件学报.. http://www.jos.org.cn/1000-9825/7137.htm

英文引用格式: Wei XM, Dong YW, Sun C, Li XH, Ma JF. Safety Analysis for Mixed-Criticality Systems with Random Errors and Error Bursts based on AADL. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7137.htm

Safety Analysis for Mixed-Criticality Systems with Random Errors and Error Bursts Based on AADL

WEI Xiao-Min[1], DONG Yun-Wei[2], SUN Cong[1], LI Xing-Hua[1], MA Jian-Feng[1]

[1](School of Cyber Engineering, Xidian University, Xi'an 710071, China)

[2](College of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: Many complex embedded systems are mixed-criticality systems (MCSs). MCSs are often required to operate with the specified criticality level, but they may be subject to hazards that can induce random errors and error bursts, which may result in the abortion of an executing thread or even lead to system failures. Current research only concentrates on schedulability analysis for MCSs, and fails to further analyze system safety and consider the dependency relationship between threads. This paper focuses on random errors and error bursts, and proposes an architecture-based safety analysis approach with the integration of fault propagation analysis. The component dependency relationship is specified using Architecture Analysis and Design Language (AADL). To compensate AADL, we create new AADL properties (AADL error burst properties) and establish new thread state machine (error burst-based thread state machine) semantics to describe thread execution process with error bursts. To apply probabilistic model checking for safety analysis, model transformation rules and assembly methods are made to derive PRISM models from AADL models. Two formulae are also formulated to obtain

* 基金项目: 国家自然科学基金(62232013, 62272366, 62125205); 中央高校基本科研业务费专项资金(ZYTS23165); 陕西省重点研发计划(2023-YBGY-371)

收稿时间: 2023-09-11; 修改时间: 2023-10-30; 采用时间: 2023-12-13; jos 在线出版时间: 2024-01-05

quantitative safety properties for verifying occurrence probabilities of failures, and qualitative safety properties for generating corresponding witnesses to figure out propagation paths for fault propagation analysis, respectively. Finally, we demonstrate the effectiveness using a power boat autopilot (PBA) system.

Key words: Mixed-criticality systems; error bursts; model transformation; safety analysis; probabilistic model checking

许多复杂的嵌入式系统,例如在航空航天、汽车和航空电子工业中应用的系统,都是混合关键系统(MCS)[1-5].大多数 MCS 都是与安全相关的系统[6].关键性(criticilaty)是对系统组件所需的防止失效(failure)的保证级别的指定.MCS[2][7]是具有两个或多个不同级别的系统,可以是汽车安全完整性级别(Automotive Safety and Integrity Level,简称 ASIL)、开发保证级别(Development Assurance Level,简称 DAL)或安全完整性级别(Safety Integrity Level,简称 SIL).

许多情况下,MCS 会受到诸如电磁干扰(Electromagnetic Interference,简称 EMI)和其他形式的辐射(如,宇宙射线和伽马射线)等危险,还会受到过度的机械或电气应力[8][9].尤其是用于实时计算和通信的 MCS 通常需要在恶劣环境中保证指定的关键性等级的前提下运行,而在 MCS 运行时,任务必须在截止时间之前完成,否则任务无法按时完成,进而可能会导致系统失效甚至事故的发生.暴露于诸如此类的危险中,可能会产生突发错误,进而导致系统失效[10].离散的突发错误能够影响量子处理器的整个量子位补丁(qubit patch)[11],同样的,也会影响通用处理器的正常工作并导致错误.虽然瞬态和间歇性错误的持续时间很短,但工程师仍然有必要在 MCS 的分析中考虑它们的影响,因为:1)它们的频率比永久性故障大 10 到 50 倍[12];2)它们可能会导致错误,使线程执行无法按时完成,甚至导致整个任务失效或系统失效[10].

故障是错误发生的原因,故障可以是由于设计或环境因素引起的软硬件系统内部缺陷或外部干扰,可能导致错误,并可能导致后续的失效[13].**突发错误**(error burst)是系统运行过程中随机发生的一系列软、瞬态或间歇错误,是可恢复的[14],例如,在机场区域,雷达波产生的电磁场会在计算机系统上引起突发错误.突发故障(fault burst)是瞬态的或间歇故障,例如,在线程执行期间发生的单粒子翻转,它可能影响 CPU[15]、内存[16]或 I/O 子系统等.突发故障常常是不可避免的,会以突发错误形式表现出来,所以,对于突发错误和突发故障,分析过程中可以只考虑突发错误.例如,单粒子翻转(突发故障)效应会导致软错误(突发错误).

在架构设计阶段识别随机错误和突发错误,不仅可以节省后期修复缺陷的成本,而且可以提高系统整体的安全性.AADL 是一种成熟的嵌入式系统层次化模型建模语言[17].为 MCS 设计架构模型和架构模式的最常见方法是基于模型的方法,同时,通过形式化验证的方式支持 AADL 刻画的架构模式是至关重要的[18][19].错误模型附件(Error Model Annex,简称 EMA)[20]和行为附件(Behavior Annex,简称 BA)[21]是 AADL 的子语言扩展.EMA 支持架构故障建模和各种自动化安全分析[22-31],例如,故障树分析(FTA)、故障模式和影响分析(FMEA)、功能危险分析(FHA)和危险分析等.尽管 AADL、EMA 和 BA 可以为安全关键系统建立运行时架构,但是,它们都不能描述突发错误的行为特征,其中,EMA 标准直接用一个概率属性描述瞬态故障的发生,BA 不能描述错误信息.为了刻画突发错误行为,本文将扩展线程状态机语义和 AADL 属性,在架构设计阶段对突发错误进行建模和分析.

瞬态错误发生频繁,类型和形式多种多样.它们可能对安全关键系统构成重大威胁.但是已有工作主要关注随机和永久的错误,而忽略了突发错误.为了提高系统的安全性,文献[32][33]利用 AADL 构建仿真平台,考虑瞬态故障的传播和演化,提出瞬态故障分析和容错方法.然而,AADL 没有详细的语义用于描述瞬态和间歇错误的特征.因此,在文献[32][33]的方法中,没有分析突发错误行为.文献[34]扩展 AADL 存储资源架构设计方法,提出存储资源约束下的可调度性分析方法.本文对 AADL 的语义进行了补充,以便在架构设计阶段对突发错误行为进行建模和分析.

大多数关于 MCS 的研究只在可调度性分析中考虑瞬态故障[1][2][5][35][36].虽然它们的可调度性分析方法结合了保证级别和时间属性,但它们没有考虑失效发生过程,不是安全性分析方法,因为它们的目标只是提出

一种调度方法.例如,文献[35]提出了针对容错 MCS 的调度技术,并将保证级别视为安全要求.从安全分析的角度来看,仅仅综合保证级别与线程是不够的.另外,故障传播是安全分析的重要组成部分[27],因为线程的故障可能是由外部设备引起的,并且线程的失效可能进一步导致复合构件的失效.此外,文献[5]指出,大多数关于 MCS 的研究主要集中在与执行时间超时的容忍度相关的问题上,这只是一种瞬态故障.对于永久性故障,他们总结了一套空间冗余技术,并基于故障覆盖率将其映射到关键性等级.本文也对 AADL 模型中的永久故障进行了定义和分析.文献[36]研究了单处理器上的两关键等级系统存在瞬态故障时的可调度性,并采用任务重执行作为容错技术来提高系统可靠性.

形式方法是用于验证 AADL 模型性质的有效方法[18][37][38].概率模型检验(Probabilistic model checking)[39]是一种定量的形式化验证方法,可用于验证具有随机行为的系统的各种定量和定性性质.在本文中,概率行为是由随机错误、突发错误和构件失效引起的.这些行为是随机的,但不是非确定性的.从概率的角度来看,系统的安全性验证就是,检查从系统初始状态到达不安全状态的概率是否低于指定的概率阈值.AADL 模型将被转换为离散时间马尔可夫链(Discrete-Time Markov Chain,简称 DTMC),便于将概率模型检验技术应用于 AADL 模型属性验证.PRISM[40]是一个功能强大的概率模型检查器,可以将系统描述为 DTMC,即 PRISM 模型.它可以用来检查系统是否能满足安全性要求.

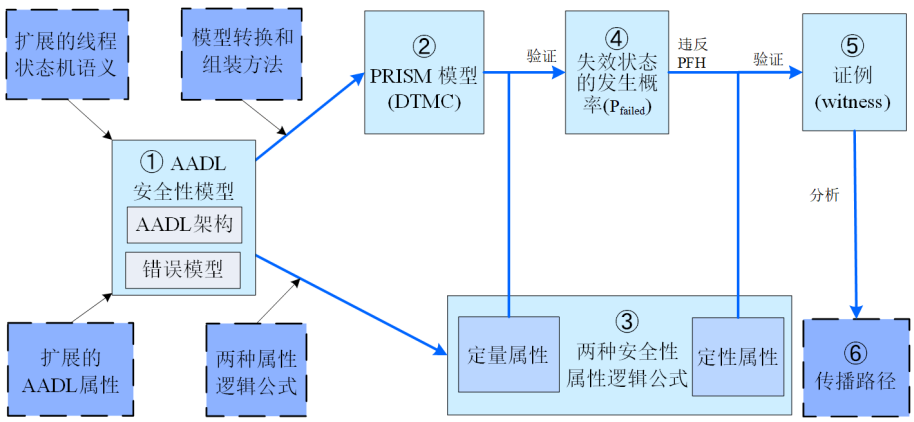

图 1 基于 AADL 的 MCS 安全性分析框架

本文提出了一种基于架构的 MCS 随机错误和突发错误安全分析方法,如图 1 所示.为了将概率模型检验技术与安全性分析相结合,需要将 AADL 模型转换为 PRISM 模型,但这是一个挑战.本文采用 Gilbert 类型[41]的两状态离散马尔可夫模型来描述突发行为,并结合随机错误行为和系统运行时架构,建立了新的线程状态机语义——突发错误行为线程状态机语义,以支持 AADL 模型的随机错误和突发错误建模.现有的一些方法采用更丰富的随机错误模型来刻画突发行为,但代价是大大增加了复杂性.此外,本文还创建了新的 AADL 属性——AADL 突发错误属性,以提高 AADL 的建模能力.首先创建新的时间和概率属性来扩展 AADL,以刻画随机错误和突发错误.还创建了一个 AADL 属性用于描述构件的 DAL.因此,可以为带有突发错误的 MCS 构建一个包含错误模型(①)的 AADL 模型.此外,通过**模型转换和组装方法**实现了从 AADL 模型到 PRISM 模型(DTMC)(②)的转换,使线程离散化执行.这些转换规则和组装方法使得将概率模型检验整合到 MCSs 的安全分析中成为可能.**两种安全性属性逻辑公式**用于从 AADL 模型获取定性和定量属性(③)以支持形式化验证.一个公式是基于失效状态和构件的 DAL 生成的定量属性公式;另一个公式是根据失效状态构造的定性属性公式.然后,根据定量属性(③)验证 PRISM 模型(②),得到失效状态发生的概率(④).如果构件不能满足安全性要求,即违反了 DAL,进一步将 PRISM 模型(②)与定性属性(③)进行验证,生成证例(⑤).然后,通过对证例的分析,可以得出故障传播路

径(⑥),即构件是如何产生失效的.它可以帮助工程师更有效、更精确地找到解决方案.

本文的主要贡献如图 1 的深蓝色虚线框所示,总结如下:

- 针对具有突发错误和随机错误的 MCS 建模和分析问题,建立了突发错误行为线程状态机语义和 AADL 突发错误属性,以支持为 MCS 构建 AADL 安全性模型.
- 为 AADL 模型建立形式化定义,提出从 AADL 模型到 PRISM 模型的模型转换规则和组装方法,以便将概率模型检验应用于 MCS 的安全性分析.
- 基于 AADL 模型,可自动生成两个用于形式化验证的安全性属性逻辑公式.
- 针对构件不能满足安全性要求的情况,生成故障传播路径以支持故障传播分析.

本文的其余部分组织如下:第 1 节介绍了 AADL、PBA 系统和概率模型检验等基础知识.第 2 节对 AADL 模型进行形式化定义.第 3 节提出了突发错误行为线程状态机和 AADL 突发错误属性以扩展 AADL 建模能力.第 4 节给出了从 AADL 模型到 PRISM 模型转换的规则和组装方法.第 5 节解释了如何实施安全分析.第 6 节通过实验说明了方法的可用性.第 7 节概述了相关工作.第 8 节对本文进行了总结,并对今后的工作进行了展望.

1 基础知识

1.1 AADL和PBA系统案例

AADL[17]是专门为基于计算机的嵌入式系统建模,包含软件和硬件.它可以用来描述软件构件、执行平台构件、组合构件以及它们之间的交互.PBA 系统[42]的 AADL 架构模型如图 2 所示.PBA 系统包含四个器件:控制器、处理器、总线和存储器.控制器包含 scale_speed_data、control_law 和 monitor 三个进程,每个进程包含一个线程.scale_speed_data 和 monitor 进程从传感器 speed_sensor 读取数据.control_law 进程接收数据 proc_data 进行计算,将命令发送给 throttle,并与 monitor 同步,将状态数据发送给 display_unit.通过 interface_unit,工程师可以在 control_law 上设置速度.数据和事件通过连接传输.这个系统还有一个处理器,线程绑定在处理器并执行,总线用于物理交互,内存用于存储.

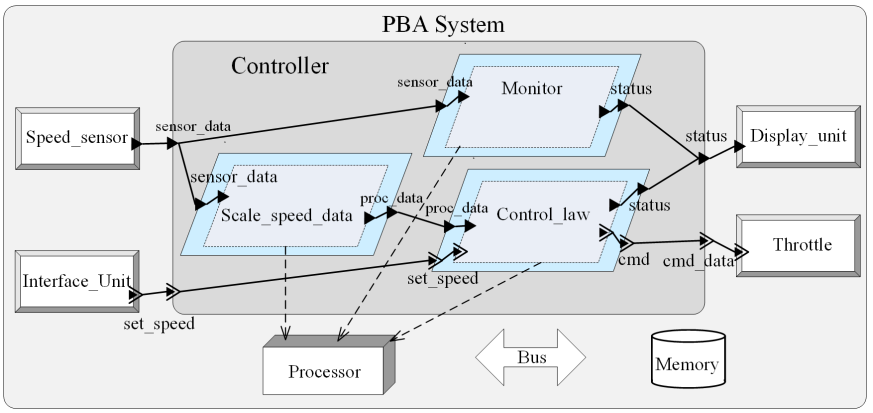

图 2 PBA 系统 AADL 架构模型

AADL 允许设计人员通过创建子语言附件或属性来扩展其功能以支持可信分析[19][26].例如,提出了子语言附件 EMA[20]来扩充 AADL,从而支持可靠性分析.EMA 可以描述错误状态、错误事件、错误状态之间的变迁、构件之间的传播以及用于描述现有故障类型的错误类型.变迁(transition)可以由错误事件或传入的传播点触发(ingoing propagation point).例如,线程 scale_speed_data 的错误模型如图 3(a)所示(第 12-25 行).错误类型

(Data_Fault)和错误状态(operational 和 failed)分别定义在错误模型库 PBA_EMLib 和错误模型子句 scale_speed_EM 中,由 use 子句引用(分别为第 13 行和第 14 行).从状态 operational 到状态 failed 的变迁(第 17 行)是由传入的传播点 sensor_data 触发的,错误类型为 Data_Fault.另一个从状态 operational 到状态 failed 的变迁(第 18 行)是由错误事件 Err 触发的,其概率分布在第 23 行定义.第 20 行中的传播声明,当该构件处于 failed 状态时,错误类型 Data_Fault 将通过端口 proc_data 传播到 control_law.EMA 还可以为复合构件指定复合错误行为.例如,controller 是一个包含三个进程的复合构件.它的复合错误行为显示在图 4(a)的第 8-11 行.这意味着当进程(scale_speed_data,control_law 或 monitor)中的一个失效时,controller 失效.本文创建了新的时间和概率属性,并在第 3.2 节中进行了解释,以支持线程状态机的建模.

本文在 3.2 节扩展了 AADL 属性以刻画构件的 DAL.DO-178C[43]定义了五个 DALs,从 A 到 E,其中 A 是最严格的.IEC 61508[44]定义了四个 SIL,从 1 到 4,其中 SIL 4 是最严格的.安全性完整性(SI)[45]是指能够成功实施指定动作以降低风险的概率阈值.SIL 对应于每小时发生失效的概率(probability-of-failure-per-hour,简称 PFH).这里将 DAL 与 SIL 的等级进行对应,对应关系见表 1,每个 DAL 等级与一个 PFH 对应,进而将 SIL 定量的安全性要求施加给每个构件,即每个构件的失效和危险的发生概率都不能超过其 PFH.文献[35]和[46]以 PFH 作为安全性要求,在保证安全性的情况下,对系统做了调度性分析,本文也将 PFH 作为安全性要求.

```
1. thread implementation Scale_data.impl
2. properties
3.  Timing_Properties::Compute_Execution_Time=> 5ms..5ms;
4.  Timing_Properties::Period=> 20ms;
5.  Timing_Properties::Deadline=> 20ms;
6.  Timing_Properties::Burst_Duration=> 1ms..1ms;
7.  Timing_Properties::Error_Duration=> 2ms..2ms;
8.  Timing_Properties::Good_2_Burst=> 0.0001;
9.  Timing_Properties::Burst_2_Good=> 0.35;
10. Timing_Properties::Lambda_B=> 0.4;
11. Timing_Properties::Lambda_G=> 0.0001;
12. annex EMV2{**
13.  use types PMS_EMLib;
14.  use behavior scale_EM::scale_Beha;
15.  component error behavior
16.  transitions
17.   Operational-[sensor_data{Data_Fault}]->Failed;
18.   Operational-[Err]->Failed;
19.  propagations
20.   Failed-[]->proc_data(Data_Fault);
21.  end component;
22. properties
23.   EMV2::OccurrenceDistribution =>[ProbabilityValue
          => 1.59e-11; Distribution=>fixed;]
          applies to Err;--两年发生一次: 1/(2*365*24*60*60*1000ms)
24.  EMV2::DevelopmentAssuranceLevel=> C;
25. **};
26. end Scale_data.impl;
```

(a) AADL 模型

```
// 定义初始的概率值: const double scale_PGB, scale_PBG,
// scale_lambdaB, scale_lambdaG.
// 根据初始概率值定义新的概率值: scale_newPgg, scale_newPgb, scale_newPerrorG,
// scale_newPbg, scale_newPbb, 和 scale_newPerrorB
1.  const int scale_burstDuration=1; //突发错误的时延
2.  const int scale_execution=4;  //每次运行需要的执行时间
3.  const int scale_deadline=20;
4.  const int scale_errDuration=2;//随机错误的时延
5.  const int scale_period=20;
6.  const double lambda_pro=1.59E-11;//2年的失效概率
7.  module Scale_data
      // 0-initial, 1-ready, 2-running(good), 3-error, 4-burst, 5-failed
8.    state_scale  : [0..5] init 0;//状态 state_scale_data的简写
9.    exec_count : [0..exec] init 0; //执行时间计数器
      // 由错误事件Err触发的错误变迁
10.  [ ] state_scale=2 -> lambda_pro : (state_scale'=5) + (1-lambda_pro) : true;
      // 由向内的错误传播点（sensor_data[Data_Fault]）触发的传播
11.  [ ] state_scale=2 & state_sensor=1 -> (state_scale'=5);
      // 以下与线程状态机对应
12.  [Scale_initial2ready] state_scale=0 -> (state_scale'=1); //initial→ready
13.  [Scale_ready2good] state_scale=1 -> (state_scale'=2)&(exec_count'=0);//ready→good
14.  [Scale_good2Others] state_scale=2 & exec_count<scale_execution ->
       scale_newPgg : (state_scale'=2) & (exec_count'=exec_count+1)//good→good
       + scale_newPgb : (state_scale'=4) //good→burst
       + scale_newPerrorG : (state_scale'=3); //good→error
15.  [Scale_release2ready] state_scale=2  & exec_count=scale_execution ->
       (state_scale'=1)&(exec_count'=0);//good→ready
16.  [Scale_burst2Others] state_scale=4 -> scale_newPbg : (state_scale'=2)//burst→good
       + scale_newPbb : (state_scale'=4) //burst→burst
       + scale_newPerrorB : (state_scale'=3);//burst→error
17.  [Scale_error2Inintial] state_scale=3 -> (state_scale'=0);//error→Initial
      // 从状态 initial(0), ready(1), good(running, 2), error(3) and burst(4)错过截止时间
18.  [Scale_initial_MissDeadline] state_scale=0-> (state_scale'=5);
19.  [Scale_ready_MissDeadline] state_scale=1-> (state_scale'=5);
20.  [Scale_good_MissDeadline] state_scale=2 -> (state_scale'=5);
21.  [Scale_Error_MissDeadline] state_scale=3 -> (state_scale'=5);
22.  [Scale_burst_MissDeadline] state_scale=4  -> (state_scale'=5);
23. endmodule
```

(b) PRISM 模型

图 3 Scale_speed_data 构件模型

```
1. system implementation PBA_Controller.impl
2.   subcomponents
3.     ... ...
4.   connections
5.     ... ...
6.   annex EMV2{**
7.     use behavior Controller_EM::Controller_Beha;
8.     composite error behavior
9.     states
10.    [scale_speed_data or speed_control_law.Failed
       or monitor.Monitor.Failed]-> Failed;
11.  end composite;
12.  **};
13.end PBA_Controller.impl;
```

(a) AADL 模型

```
// 数值5对应线程的失效状态
label "ControllerFailed" =
        state_scale_speed_data = 5 |
        state_control_law = 5 |
        state_monitor = 5;
```

(b) PRISM 模型

图 4 Controller 构件的复合错误模型

表 1 设计保证等级(DAL)与安全性完整性等级(SIL)

DAL	A	B	C	D	E
SIL	4	3	2	1	-
PFH	<1.0E-8	<1.0E-7	<1.0E-6	<1.0E-5	≥1.0E-5

1.2 概率模型检验

概率在软件和硬件系统设计与分析过程中有着重要的作用.概率可以作为分析系统性能的有效工具,也可以用于系统不可靠或随机性建模,如故障容错系统、多媒体协议、无线传感和控制系统等.形式化验证方法在许多工业领域已经成为了设计阶段的主要部分,能够检验系统的正确性,例如,验证电子设计自动化工具的功能正确性、验证安全性关键系统(如航空电子系统)的正确性等.模型检验是一种常用的形式化验证方法,概率模型检验是对常用模型检验方法的扩展,能够验证定量属性的正确性,是一种基于数学理论的严格的验证方法,能够有效的对系统定量和定性属性进行验证[39].概率模型检验方法是一个研究热点,目前已应用在安全关键系统、性能与可靠性和调度性与优化等[47].

DTMC[48]概率模型可以有效的刻画系统随机行为,本文使用 DTMC 模型描述 AADL 安全性模型的概率随机行为和离散行为.PRISM[40]是广泛使用的概率模型检验工具,用 PRISM 语言来描述具有概率选择的 DTMC 模型.为了描述系统属性,PRISM 提供的属性规约语言,包括线性时序逻辑(Linear Temporal Logic,简称 LTL)、概率 LTL(Probabilistic LTL,简称 PLTL)和计算树逻辑(Computation Tree Logic,简称 CTL)等.PLTL 由 LTL 扩展而来,可以描述概率的需求属性,如“失效状态在 20 个时间单元内发生的最大概率最多是 0.01”[49],可表示为:Pmax<=0.01 [F<=20 failed],其中,failed 代表失效状态.还可使用 CTL 描述需求属性,如“存在一条能够到达失效状态的路径”,可表示为:E [F failed],如果结果为真,会生成一条证例(witness),即一条动作(action)序列.

PRISM 模型是一个组合体,包含模块(module)、常量(constant)和全局变量.常量必须在形式化验证之前初始化.全局变量可以被每一个模块引用.模块包括两个部分:局部变量和命令(command).局部变量用于描述状态,命令用于描述行为.系统的每个状态由所有变量的一次取值共同构成.一条命令由一个动作、一个守卫(guard)和一个或多个带有概率值的更新(update)构成,可以表示为“$[a]\ g -> \lambda_1 : \mu_1 + ... + \lambda_n : \mu_n;$”,其中,动作 a 用于模块之间命令的同步操作,可以为空;守卫 g 是由变量构成的谓词(predicate),变量之间通过逻辑操作组合起来;每一个更新μ对应系统允许的一个状态变迁;常数值λ_i是模型随机行为使用的概率值,是更新 μ_i 的发生概率,一条命令中的所有概率值必须满足 $\sum_{i=1}^{n} \lambda_i = 1$.命令的执行条件是:首先,守卫条件为 true;其次,如果动作不为空,那么其它模块中与此条命令同步的命令的守卫都必须是 true;然后,依据命令中各个更新的发生概率值,其中一个更新会随机的发生.

2 AADL 模型的形式化定义

对本文使用的 AADL 模型进行明确的定义,给出确定的语义,便于设计准确的模型转换规则.根据 AADL 模型的层次结构和错误行为特征,将模型分成三个基本模型,能够组装成系统模型,包括了构件内的错误模型(intra-component error model,简称 EM)、构件之间的错误传播(inter-component error propagation,简称 E2E)和复合错误行为(composite error behavior,简称 CEB).例如,一个系统包含两个进程和一个设备,包含错误模型的系统 AADL 模型如图 5 所示,其中,EM 表示构件内部的故障行为和错误变迁,E2E 是基于两个构件之间的连接的从一个构件的 EM 到相连构件的 EM 的错误传播,CEB 是复合构件依据其子构件的错误状态而构成的行为.各个基本模型的形式化定义将在以下各部分详细介绍.

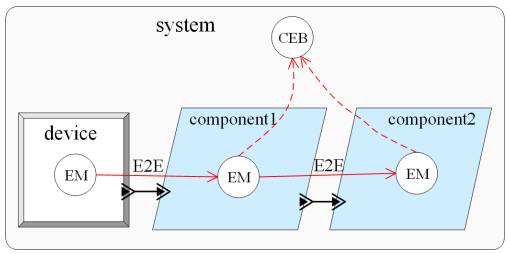

图 5 AADL 模型各基本模型示例

2.1 构件内的错误模型(EM)

EM 作为构件内的错误模型,描述了构件的内部错误行为.它由三个部分组成:状态、事件和变迁.可以定义在错误模型子句(EMSub)的构件错误行为中,或者在错误行为状态库(EBSM)中定义,通过 use behavior 关键字被引用.错误行为变迁描述构件在满足变迁条件时从源状态转变为目标状态的构件变化.变迁条件可以是错误事件和向内的错误传播点.本文规定:当变迁条件包含向内错误传播点时,该错误变迁属于 E2E 的一部分,将在 2.2 节讨论.

定义 1 构件内的错误模型是一个元组,$A_{EM}=(ES,EE,\rightarrow_{EM},I_{EM},AP_{EM},L_{EM})$,其中

1) ES 是有限的错误状态集.
2) EE 是有限的错误事件集.
3) $\rightarrow_{EM}\subseteq ES\times COND_{EM}\times ES$,是变迁关系,$COND_{EM}$ 是 EE 元素的逻辑组合.
4) $I_{EM}\subseteq ES$ 是初始状态集.
5) AP_{EM} 是原子谓词集,错误状态名称.
6) $L_{EM}:ES\rightarrow 2^{AP_{EM}}$ 是标签函数.

错误事件 ee 的发生要服从概率分布类型和发生参数,可表示为 $OcDist(ee)=(distribution,occPara)$.允许的分布类型有指数、确定时间延迟和固定概率分布,变迁的发生服从 ee 的发生方式.对于 $ee_1\in EE_1$ 和 $ee_2\in EE_2$,如果 $OcDist(ee_1)=OcDist(ee_2)$,那么 ee_1 和 ee_2 的分布类型和发生参数相同.

2.2 构件之间的错误传播(E2E)

E2E 作为相连构件的 EM 之间的连接关系,包含了构件之间的错误传播和由错误传播点(Error Propagation Point,简称 EPP)触发的错误变迁.E2E 由错误模型的两部分共同描述,EMSub 中的错误传播(error propagations)部分和构件错误行为的传播(propagations)部分.Error propagations 部分包括错误传播和错误包含,可以分别描述构件允许通过端口传播的错误类型和构件不能传播的错误类型.因为分析过程中只考虑会对系统造成影响的错误,所以本文不考虑错误包含.错误传播点可以定义于数据、事件或者事件数据端口上,包括向内的错误传

播点(Incoming Error Propagation Point,简称 IEPP)和向外的错误传播点(Outgoing Error Propagation Point,简称 OEPP),分别对应于向内的端口和向外的端口.错误类型集合可为 OEPP 和 IEPP 定义错误类型.

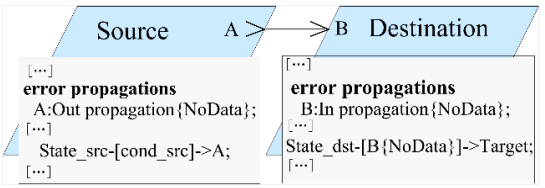

图 6 两个相连构件之间的一个 E2E

基于构件之间的连接,源构件 source 可以通过 OEPP 将错误传播到目的构件 destination 的 IEPP,再传到构件内部触发错误变迁.例如,如图 6 所示,构件 source 的端口 A 到目的构件 destination 的端口 B 有一条连接;源构件在端口 A 上定义了 OEPP 及其错误类型 NoData,还定义了错误传播;目的构件在端口 B 上定义了 IEPP 及其允许传入构件的错误类型 NoData.对于目的构件,如果 Target 是一个错误状态,那么从 State_dst 到 Target 是一个错误变迁;如果 Target 是一个 OEPP,那么从 State_dst 到 Target 是向外传播过程.对于一个 E2E,只需要考虑前一种情况(当 Target 是错误状态),因为后一种情况将包含于另一个 E2E 中.当构件 source 处于 State_src 状态,且满足 cond_src 条件时,NoData 类型的错误将从端口 A 向外传出.对于 destination 构件,如果其正处于 State_dst 状态,端口 B 接收到 NoData 类型的错误,那么从 State_dst 到 Target 的演变过程将会发生,否则 NoData 类型的错误不会传播到 destination.

为了完整的描述 E2E,在 OEPP 与 IEPP 之间增加了一个表示有错误的虚拟状态 es_{has}、一个表示无错误的虚拟状态 es_{empty}、一个虚拟事件 ee_{dis} 和一个虚拟 IEPP 事件 ee_I.当源构件的 OEPP 的错误传播发生时,es_{has} 状态表示已经将错误传出源构件,并且等待将其传入目的构件,例如,在图 6 中,es_{has} 表示错误已从端口 A 传出,在端口 A 和端口 B 之间,该错误可能会触发目的构件中的变迁或向外传播条件.但是,该错误不可能一直存在于两端口之间,所以构件之间的错误传播会有一个事件 ee_{dis},使这个错误消失,即,进入无错误虚拟状态 es_{empty},构件之间没有正在传播的错误.如果正在传播的错误能够触发目的构件内的变迁或错误传播条件,那么虚拟事件 ee_I 表示目的构件的 IEPP 已经将错误传入构件,该虚拟事件将作为目的构件内的触发条件,例如,在图 6 中,表示端口 B 接收到错误并将 ee_I 作为触发条件.将 ee_{dis} 和 ee_I 定义为固定概率分布事件,参数为 1.0.

定义 2 两个构件之间的 *E2E* 是一个元组 $A_{E2E}=(ES_{E2E},EE_{E2E},\rightarrow_{E2E},I_{E2E},AP_{E2E},L_{E2E})$,其中,

1) $ES_{E2E}=ES_{src}\cup ES_{dst}\cup ES_{fic}$,是有限状态集合,其中,$ES_{src}$ 和 ES_{dst} 分别是源构件和目的构件的错误状态集合,ES_{fic} 是 *OEPP* 与 *IEPP* 之间虚拟状态集合,即,$es_{has}\in ES_{fic}$,$es_{empty}\in ES_{fic}$.
2) $EE_{E2E}=EE_{src,I}\cup EE_{dst}\cup EE_{dst,I}$,是有限事件集合,包含错误事件和虚拟的 *IEPP* 事件(因为源构件向外传播的触发条件只能是 *IEPP*,而目标构件的变迁发生条件可以是错误事件或 *IEPP*),其中,$EE_{src,I}$ 是源构件中 *E2E* 的虚拟 *IEPP* 事件集合,同样地,EE_{dst} 和 $EE_{dst,I}$ 分别是目的构件自身的事件和的虚拟 *IEPP* 事件.为了便于描述,将两个相连构件之间的 ee_I 和 ee_{dis} 划归到两个构件中:$ee_{dis}\in EE_{src,I}$,$ee_I\in EE_{dst,I}$.
3) $\rightarrow_{E2E}\subseteq ES_{E2E}\times COND_{E2E}\times ES_{E2E}$ 是变迁关系,$COND_{E2E}$ 是 EE_{E2E} 元素的逻辑组合.$\rightarrow_{E2E}$ 包括了三类变迁:从源构件错误状态到触发条件再到虚拟状态 es_{has} 的变迁;从虚拟状态 es_{has} 到虚拟事件 ee_{dis} 再到虚拟状态 es_{empty} 的变迁;从目的构件的错误状态到触发条件再到目的构件错误状态的变迁,触发条件包括虚拟状态 es_{has} 和虚拟 *IEPP* 事件 ee_I 的 *AND* 组合.
4) $I_{E2E}\subseteq I_{EM,src}\cup I_{EM,dst}$ 是初始状态集合.
5) $AP_{E2E}=AP_{EM,src}\cup AP_{EM,dst}\cup AP_{fic}$ 是原子谓词集合,其中,AP_{fic} 是虚拟错误状态对应的原子谓词.
6) $L_{E2E}:ES_{E2E}\rightarrow 2^{APE2E}$ 是标签函数.

2.3 复合错误行为(CEB)

CEB 是 EMSub 的一部分,作为复合构件的错误模型子句.CEB 通过使用复合状态表达式(composite_state_expression),复合构件行为通过利用 CEB 描述复合构件的错误状态,触发条件由 IEPP、子构件的错误状态及其逻辑组合构成.根据 E2E 的定义 2,这里的 IEPP 是虚拟 IEPP,作为 CEB 的触发事件.CEB 中可用的逻辑操作符有 **and** 和 **or**,可用的逻辑原语有 **ormore** 和 **orless**.关键字 **others** 表示除了已声明的发生条件之外的所有情况,所以,在每个构件的 CEB 描述中,至多只能有一条复合错误行为可以包含关键字 **others**.对于包含 **others** 的 CEB,先将 CEB 的发生条件转换为错误状态和 IEPP 的逻辑组合.

错误模型主要描述复合构件可能因为子构件发生错误,同时也隐含了复合构件的可操作状态,即,复合构件还没有进入错误的状态,因此,为复合构件添加隐含的可操作状态.

定义 **3** *CEB* 是一个元组 $A_{CEB}=(ES_{CEB},EE_{CEB},\rightarrow_{CEB},I_{CEB},AP_{CEB},L_{CEB})$,其中,

1) $ES_{CEB}=ES_{com}$,是有限状态集合,ES_{com} 包括复合构件内的错误状态和隐含的可操作状态.
2) $EE_{CEB}=EE_I$,是有限事件集合,EE_I 是虚拟 *IEPP* 事件集合.
3) $\rightarrow_{CEB}\subseteq ES_{CEB,src}\times COND_{CEB}\times ES_{CEB,dst}$,是变迁关系,其中,$ES_{CEB,src}$ 是复合构件的可操作状态,$COND_{CEB}$ 是虚拟 *IEPP* 事件和子构件错误状态(ES_{sub})值的逻辑组合,$ES_{CEB,dst}$ 是复合构件的错误状态.
4) $I_{CEB}=\emptyset$.
5) $AP_{CEB}=AP_{ES,CEB}$,是原子谓词集合,错误状态和可操作状态名称集合.
6) $L_{CEB}:ES_{CEB}\rightarrow 2^{AP_{CEB}}$,是标签函数.

3 突发错误行为线程状态机和 AADL 突发错误属性

Gilbert 两状态离散马尔科夫模型是突发错误行为建模的常用方法[41],如图 7 所示.该模型有两个状态,即正常状态 good 和突发错误状态 burst(简写为 G 和 B).它们之间的转换与概率有关,包括 P_{GG}、P_{GB}、P_{BB} 和 P_{BG},其中,$P_{GG}+P_{GB}=1$ 且 $P_{BB}+P_{BG}=1$.

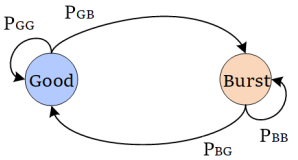

图 7 Gilbert 突发错误模型(good:正常状态,burst:突发错误状态)

在第i时刻,该马尔可夫模型的突发错误(error burst,简称EB)状态概率表示为:

$$EB(i+1)=P_{BB}P(s_i=B)+P_{GB}P(s_i=G)=P_{BB}EB(i)+P_{GB}(1-EB(i)) \tag{1}$$

EB的递归不动点表示为:

$$EB_s=EB(i+1)=EB(i) \tag{2}$$

通过将公式(2)代入公式(1),可以计算得到EB_s的值为:

$$EB_s=P_{GB}/(P_{GB}+P_{BG}) \tag{3}$$

因此,正常状态和突发错误状态的发生概率分别为$1-EB_s$ 和 EB_s.

此外,在正常状态和突发错误状态上都可能随机发生错误,如图 8 所示.从正常状态或者突发错误状态演变到错误状态的概率值分别表示为 λ_G 和 λ_B.基于该突发错误行为,利用公式(3),从正常状态和突发错误状态产生错误状态的发生概率分别表示为 P_{GE} 和 P_{BE},如下:

$$P_{GE} = \lambda_G(1 - EB_s) \tag{4}$$

$$P_{BE} = \lambda_B EB_s \tag{5}$$

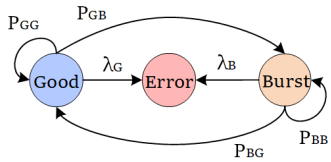

图 8　正常状态(good)、突发错误状态(burst)和错误状态(error)之间的迁移关系

但是,从正常状态或突发错误状态演变到其它状态的概率总和可能大于 1,例如, $P_{GG} + P_{GB} + P_{GE} = 1 + P_{GE} \geq 1$. 因为从一个状态发生变迁的概率总和必须等于 1,所以需要为正常状态、突发错误状态和错误状态三者之间的变迁定义新的概率值,概率计算公式如公式(6)所示,其中,下标 g、b 和 e 分别对应正常状态、突发错误状态和错误状态.由此,在已知 P_{GB}、 P_{BG}、 λ_G 和 λ_B 时,便可以通过将公式(3)(4)(5)带入公式(6),计算得到新的概率值.

$$\begin{aligned}
P_{ge} &= P_{GE} / (1 + P_{GE}) \\
P_{gg} &= (1 - P_{GB}) / (1 + P_{GE}) \\
P_{gb} &= P_{GB} / (1 + P_{GE}) \\
P_{be} &= P_{BE} / (1 + P_{BE}) \\
P_{bb} &= (1 - P_{BG}) / (1 + P_{BE}) \\
P_{bg} &= P_{GB} / (1 + P_{BE})
\end{aligned} \tag{6}$$

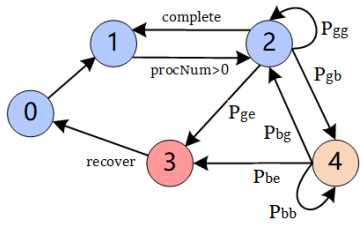

图 9　突发错误行为线程状态机(0-初始状态,1-就绪状态,2-正常状态,3-错误状态,4-突发错误状态)

3.1 突发错误行为线程状态机

行为附件(BA)是支持行为建模的 AADL 子语言扩展,将构件的内部行为描述为状态迁移系统,扩展了默认的运行时执行语义,并能精确地刻画子程序调用和同步协议.然而,它不支持本文的方法,因为它不能描述瞬态/间歇故障的时序和概率特性;它也不能描述故障传播和故障行为.因此,为了描述线程行为及其随机错误和突发错误,本文提出新的突发错误行为线程状态机,如图 9 所示.它包含 5 个状态,表示为状态 0 到 4,分别是初始状态(initial)、就绪状态(ready)、运行/正常状态(running/good)、突发错误状态(burst)和错误状态(error).图中,状态 running 和 good 表示同一个状态,因为当一个任务在 CPU 中运行(running),它就是处于可操作性(operational)状态.当一个线程被创建并且获得除 CPU 之外的所有资源,从 initial 状态转变到 ready 状态.图中每个箭头表示两个状态之间的变迁,变迁的发生要满足线程截止时间的约束,即,“ *time* ≤ thr_n*thr_period+thr_deadline ”,其中, *time* 作为时间计数器,为所有线程在一个超周期内记录时间,thr_period 是线程 thr 的周期,thr_n 表示线程 thr 在一个超周期内已经完成 thr_n 个周期,thr_deadline 表示线程 thr 的截止时间.对于一个处于 ready 状态的线程,当处理器可用时,线程会进入 running 状态.procNum>0 用来表示可用的处理器数量,本文假设系统只有一个处理

器,所以 procNum 的初始值是 1.线程状态机以离散化时间方式运转.exec_count 用于记录线程处于 running 状态(正常执行)的时间.当线程从 ready 状态进入 running 状态,exec_count 被初始化为 0.只有当线程发生从 running 状态到 running 状态的变迁时,计数器 exec_count 才会增加 1.在线程运行过程中,每当它完成执行(即,exec_count 等于线程执行时间 execution),才会被释放(release)并进入 ready 状态.状态 good、error 和 burst 之间的变迁将随机发生,概率值分别是: P_{gg}, P_{ge}, P_{gb}, P_{bb}, P_{be} 和 P_{bg},其中, $P_{gg}+P_{ge}+P_{gb}=1$ 且 $P_{bb}+P_{be}+P_{bg}=1$.图 9 中的其它变迁都是以概率值 1.0 发生.例如,从 initial 状态到 ready 状态的变迁将以概率值 1.0 发生.当线程进入 burst 状态或者 error 状态,线程执行会因为突发错误或者随机错误发生而产生时间延迟,两种时延可以分别表示为 burstDuration 和 errDuration.例如,图 3(b)的第 1 行和第 4 行为线程定义了时延.这里的时延就是指处于随机错误或者突发错误状态的线程,在演变到其它状态之前所耗费的时间,并且通过处理器将时延添加到 *time* 计数器.例如,图 10 的第 16 行和第 17 行为 scale_speed_data 线程将时延添加到 *time* 中.当线程处于 error 状态,并且没有错过截止时间,那么回到 initial 状态.此外,无论线程在哪个状态错过了截止时间,都会失效,如图 10 的第 18 至 22 行所示.

```
1.  const int superperiod=60;// 超周期，根据所有线程的周期计算得到
2.  formula timeUpdateValue1=(time+1>=superperiod?0:time+1);
3.  const int scaleNum=3; // Scale线程在一个超周期内运行3次
4.  formula Scale_deadline=(Scale_n*scale_period+scale_deadline);// 在超周期内执行.
5.  formula ScaleNUpdateValue1=(Scale_n=scaleNum-1?0:Scale_n+1);
6.  ……//CLNum, CL_deadline, CLNUpdateValue1, MonitorNum, Monitor_deadline, MonitorNUpdateValue1
7.  module processor
8.    procNum   :  [0..1] init 1; // 1: 表示处理器可用, 0: 表示不可用
9.    time      :  [0..superperiod] init 0;  // 系统运行的时间计数器，从0开始.
10.   Scale_n   :  [0..ScaleNum-1] init 0; // 记录scale线程已经在一个超周期内执行完的次数
11.   scale2CL  :  [0..1] init 0; // 1: 表示scale线程已经完成，并产生了输出, 0: 表示后继构件在等待
      //*** 以下将时间约束和时延施加给scale线程.***//
      //initial→ready, scale_sensor进入可操作状态operational(0).
12.   [Scale_initial2ready] state_sensor=0  -> true;
      //ready→good, 带有偏移和抖动。如果exec_count大于0，那么这是重新执行re-execution.
13.   [Scale_ready2good] procNum>0 & time<=scale_deadline & scale2CL=0 & time>=Scale_n*scale_period &
                         time<=superperiod-> (procNum'=procNum-1);
      //good→(good, burst, error), time 在每一步自加1。如果该值大于超周期，将其重置为0.
14.   [Scale_good2Others] time<=scale_deadline & time<=superperiod->(time'=timeUpdateValue1);
      //good→ready, 处理器释放线程后，将变为可用状态。如果scale完成了，那么要更新Scale_n值.
15.   [Scale_release2ready] procNum<1 & time<=scale_deadline -> (procNum'=procNum+1)&(scale2CL'=1)&
                         (Scale_n'=ScaleNUpdateValue1);
      //burst→(good, burst, error), 突发错误时延将被加到time(<= deadline)中，并判断是否要更新Scale_n 值.
16.   [Scale_burst2Others] time<=scale_deadline & time<=superperiod ->
                         ((time'=(time+scale_burstDuration>=superperiod?0:time+scale_burstDuration))&
                         (Scale_n'=(time+scale_burstDuration>Scale_n*scale_period)?0:Scale_n);
      //error→Initial, 将错误时延加到time(<= deadline)中，并判断是否要更新 Scale_n 值.
17.   [Scale_error2Inintial] procNum<1 & time<=scale_deadline & time<=superperiod ->  procNum'=procNum+1)&
                         (time'=(time+scale_errDuration>=superperiod?0:time+scale_errDuration))&
                         (Scale_n'=(time+scale_errDuration>Scale_n*scale_period)?0:Scale_n);
      // 如果scale线程错误截止时间，它将从初始态/就绪/运行态/突发错误/错误状态（initial, ready,
      // good(running), burst and error）进入失效状态，同时处理器又将变为可用状态(procNum=procNum+1).
18.   [Scale_initial_MissDeadline] procNum<1 & time>scale_deadline -> (procNum'=procNum+1);
19.   [Scale_ready_MissDeadline] procNum<1 & time>scale_deadline -> (procNum'=procNum+1);
20.   [Scale_good_MissDeadline] procNum<1 & time>scale_deadline -> (procNum'=procNum+1);
21.   [Scale_burst_MissDeadline] procNum<1 & time>scale_deadline  -> (procNum'=procNum+1);
22.   [Scale_Error_MissDeadline] procNum<1 & time>scale_deadline -> (procNum'=procNum+1);
      ……//Controller线程
      //initial→ready, scale_sensor进入可操作状态operational(0).
23.   [CL_initial2ready] s_iu=0  -> true;
24.   [CL_ready2good] procNum>0 & time<=CL_deadline & scale2CL=1 & time>=CL_n*CL_period &
                         time<=superperiod-> (procNum'=procNum-1)&(scale2CL'=0);
      ……//Monitor线程
25. endmodule
```

图 10　处理器的一部分 PRISM 模型(与 scale_speed_data 构件对应)

3.2 AADL突发错误属性

为了支持随机错误和突发错误的建模,需要在架构模型中描述线程执行属性.但是,AADL 标准提供的属性只能描述线程的周期、执行时间和截止时间.因此,本文创建了 AADL 突发错误属性,包括时序属性、概率属性和 DAL 属性,如图 11 所示.这些 AADL 属性能够有效的弥补 AADL 的建模能力.第 1 行和第 2 行定义了时间属性,分别用于描述突发错误时延和随机错误时延.第 3 到 6 行定义了变迁发生的概率属性,分别用于描述从 good 到 burst、从 burst 到 good、从 burst 到 error、以及从 good 到 error 的变迁发生概率属性,它们分别对应概率值: P_{GB}, P_{BG}, λ_B 和 λ_G .这些概率值可以用于公式(5)计算新的概率值: P_{gg}, P_{ge}, P_{gb}, P_{bb}, P_{be} 和 P_{bg} .第 7 行为 DAL 定义了新的属性.例如,在图 3(a)中,第 6 到 11 行是线程 scale_speed_data 定义的时间和概率属性,第 24 行定义了 DAL.利用这些属性,可以在 AADL 模型中为线程构件建立线程状态机语义.

```
1. Burst_Duration: aadlinteger 0 ps .. Max_Time units Time_Units
                applies to (thread, thread group);
2. Error_Duration: aadlinteger 0 ps .. Max_Time units Time_Units
                applies to (thread, thread group);
3. Good_2_Burst: aadlreal applies to (thread, thread group);
4. Burst_2_Good: aadlreal applies to (thread, thread group);
5. Lambda_B: aadlreal applies to (thread, thread group);
6. Lambda_G: aadlreal applies to (thread, thread group);
7. DAL : inherit enumeration (A, B, C, D, E) applies to (all);
```

图 11 AADL 突发错误属性

4 模型转换

为了依据 AADL 安全性模型建立 DTMC 模型,以支持利用概率模型检验的安全性分析.首先,为 AADL 安全性模型的基本元素制定模型转换规则,包括构件实现、错误状态、初始错误状态和错误事件;然后,为逻辑操作和逻辑原语制定模型转换规则;在此基础上,为安全性模型的基本模型制定模型转换规则,包括 EM、E2E 和 CEB 等;最后,依据系统安全性模型的结构和错误行为特征等,提出模型组装方法,将各基本模型分别组装为完整的 DTMC 模型.

表 2 基本元素转换规则

规则	AADL 安全性模型	概率模型	转换示例
1	构件实现	模块(module)	AADL: **system implementation** sys.i 概率模型: **module** sys_i
2	错误状态	整数变量	AADL: srcState: **initial error state**; dstState: **error state**; 概率模型: stateVar : [0..2] **init** 0; (stateVar=0 代表 srcState, stateVar=1 代表 dstState)
3	初始错误状态	变量的初始值(0)	规则 2 的转换示例中,srcState 转换为“stateVar=0”

4	带有固定概率发生属性的错误事件	记录概率值的双精度常量	AADL: errEvent: **error event**; (参数:0.1)
			概率模型: **const double** proErrEvent = 0.1;
			//事件本身发生的概率值
			(1-proErrEvent) //除此事件之外的值

4.1 基本元素转换规则

模型转换的四条基本元素转换规则如表 2 所示,下文规则中的 TR 表示转换规则(transformation rule).每个构件实现可以转换为一个模块(module),如第 1 条规则所示,例如,线程实现名 scale_speed_data(图 3(a)第 1 行)转换为模块名 scale_speed_data(图 3(b)的第 7 行).构件内的错误状态和危险映射为一个整数变量,如第 2 条规则所示,每个值对应一个错误状态,该变量的初始状态是 0,对应于初始出错误状态,如第 3 条规则所示.时间属性映

射为整型常量,错误事件或危险触发器的发生概率p映射为双精度常量,而相反情况的发生概率为:$1.0-p$,用真值 true 来刻画事件,如第 4 条规则所示.例如,图 3(a)中从第 3 到 7 行的时间属性转换为图 3(b)中的第 1 到 5 行的常量;图 3(b)第 6 行的常量刻画的是失效概率;图 3(a)第 8 到 11 行的概率属性要先根据公式(5)计算出新的概率值,为了节省空间,如图 3(b)开头注释中介绍的新的概率值,用于描述线程状态机之间随机发生的状态变迁.

定义 4 基本元素的转换规则:从 *AADL* 基本元素 A_b 到 *PRISM* 模型 PM_b 的转换规则是 TR_b,形式如 $TR_b=(A_b,PM_b)$,其中,A_b 可以是构件实现 A_{im}、(初始)错误状态 A_{es}、错误事件 A_{ee}、危险 A_h、危险触发器 A_{ht} 或者危险源 A_{hs},而 PM_b 是对应的概率模型元素集合.TR_b 包括多条规则:$TR_{im}=(A_{im},PM_{im})$、$TR_{es}=(A_{es},PM_{es})$和 $TR_{ee}=(A_{ee},PM_{ee})$,详细的规则是:

1) $TR_{im}=(A_{im},PM_{im})$,其中,A_{im} 是构件实现集合,PM_{im} 是对应的模块集合.
2) $TR_{es}=(A_{es},PM_{es})$,其中,A_{es} 是构件内错误状态集合;PM_{es} 是一个局部变量.这个变量的取值范围对应于构件内所有的错误状态.每个值对应一个错误状态,其中,0 对应初始错误状态.
3) $TR_{ee}=(A_{ee},PM_{ee})$,其中,A_{ee} 是错误事件集合;PM_{ee} 是概率值集合,命令中各个变迁的发生概率.错误事件的概率发生参数对应到 PM_{ee} 中的概率值.每个概率值定义为一个双精度常量,作为常量可以便于用户阅读和使用.

4.2 逻辑操作和逻辑原语的转换规则

对于 EMA 的逻辑操作 And 和 Or,可以根据组合元素的不同,使用 PRISM 语言中的逻辑操作替代.另外,EMA 中有逻辑原语:Ormore(或者多于)和 Orless(或者少于),通过使用 And 和 Or 的组合来替代逻辑原语.例如,逻辑原语表达式:1 Ormore (state1, state2),表示如果一个或者多个状态为真,那么这个表达式的值为真.可以将逻辑原语表达式转换为逻辑表达式:state1 Or state2 Or (state1 And state2).

定义 5 逻辑操作和逻辑原语的转换规则是 $TR_{lp}=(A_{lp},PM_{lp})$,其中,A_{lp} 是由逻辑操作和逻辑原语组合而成的表达式;PM_{lp} 是由逻辑操作组合而成的表达式.对于包含逻辑原语的表达式,先使用逻辑操作 *And* 和 *Or* 替代逻辑原语 *Ormore* 和 *Orless*,然后根据表达式 A_{lp} 的组成元素,转换规则可以分为以下两种:

1) 规则 $TR_{lp,1}$:A_{lp} 是由错误状态、*IEPP* 或者它们两者组成的逻辑表达式,先用基本元素转换规则转换错误状态和 *IEPP*,然后用“*&*”和“|”分别替换 *And* 和 *Or* 得到 $PM_{lp,1}$.
2) 规则 $TR_{lp,2}$:A_{lp} 是由错误事件、错误状态、*IEPP* 或者它们共同组成的逻辑表达式,根据 *AND* 和 *OR* 的关系,转换得到的 $PM_{lp,2}$ 包含两部分:由错误事件部分转换得到的一个概率值 Pr_{lp},由错误状态和 *IEPP* 部分转换得到的 $PM_{lp,siepp}$(与 $PM_{lp,1}$ 类似)

对于转换规则 TR_{lp},要先转换逻辑组合元素:错误状态和 IEPP,再转换逻辑关系.错误状态依据基本元素转换规则进行转换,而 IEPP 的转换规则在后文使用 TR_{lp} 时进行具体的介绍.

4.3 AADL安全性模型基本模型的转换规则

本节除了对 EM、E2E 和 CEB 制定转换规则之外,还要将线程状态机和处理器构件作为基本模型,并为它们制定转换规则.因为扩展的线程状态机刻画了突发错误行为,同时系统运行过程需要处理器与线程相互协作执行.

4.3.1 EM 的转换规则

根据定义 1,EM 由错误变迁(Error Transition,简称 ET)构成,EM 的每一条从错误状态到触发条件再到错误状态的ET映射为概率模型中的一条命令.这里的触发条件可以是空,也可以是一个错误事件或多个错误事件的逻辑组合.对于触发条件包含 IEPP 的情况,将在 E2E 的转换规则中介绍.

定义 6 *EM* 的转换规则是 $TR_{em}=(A_{em},PM_{em})$,其中,$aem\in A_{em}$ 是一条 *ET*,$pmem\in PM_{em}$ 是转换得到的一条命令.*pmem* 的动作设置为 *run.aem* 的源错误状态转换为 *pmem* 的守卫条件.*aem* 的目的状态转换为 *pmem* 的变迁.

对于 *aem* 的触发条件,分两种情况:*1*)如果触发条件为空,发生概率取值为 *1.0*,直接将 *pmem* 的变迁当作更新(*update*)(概率值缺省相当于概率值为 *1.0*),从而将 *aem* 转换为一条命令 *pmem*.*2*)如果触发条件不为空(只包含错误事件),使用定义 5 中的规则 $TR_{lp,2}$ 转换,将 $PM_{lp,siepp}$ 添加到守卫中,将 Pr_{lp} 与 *pmem* 的变迁结合构成更新,如果 Pr_{lp} 小于 *1.0*,要为 *pmem* 添加另一个更新:*"*(1−Pr_{lp}):*true"*,进而构成一条命令.

对于转换规则 TR_{em},例如,图 3(a)第 18 行的 ET 转换为图 3(b)第 10 行的命令.

4.3.2 线程状态机转换规则

定义 7 线程状态机转换规则是 $TR_{em}=(A_{em},PM_{em})$,$aem\in A_{em}$ 是一条错误变迁,$pmem\in PM_{em}$ 是转换得到的一条命令.*pmem* 动作名的组成为:以模块名开头,剩余部分用来描述命令的含义,中间以下划线连接.*pmem* 剩余部分的转换规则与定义 6 的规则 $TR_{em,1}$ 相同.此外,要添加一个执行计数器变量 *exec_count*,记录一个周期内执行了多长时间,同时,将该变量的初始化过程,加到从线程 *ready* 状态到 *good* 状态变迁对应的命令内;将该变量自加一的过程,添加到从线程 *good* 状态到 *good* 状态变迁对应的命令内;在线程 *good* 状态到 *ready* 状态变迁对应的命令内,添加判断该变量值是否等于线程执行时间的守卫条件,添加该变量恢复为初始值的过程作为命令的变迁.

利用上述规则可以转换线程状态机,例如,图 7 展示了线程状态机,是状态之间的变迁关系.状态之间的变迁转换为命令,源错误状态和约束条件转换为命令的守卫条件.目标错误状态是命令发生后的新状态,转换为命令的更新.命令的动作名字以模块名字开头,动作名字的其它部分用来描述对应的变迁的含义.其中,图 3(b)第 13 行的命令对应从 ready 状态到 good(running)状态的变迁,命令的动作名是 scale_ready2good,以模块名字开头(这里为了节省空间,用 scale 代替 scale_speed_data),剩余部分描述了变迁的含义.从 good 状态到其它三个状态(error、burst 和 good)是一个变迁,所以转换为一条命令.例如,线程 scale_speed_data 中从 good 状态出发的变迁转换为图 3(b)第 14 行的命令.同样的,用一条命令(图 3(b)第 16 行)描述从 burst 状态到其它三个状态(error、burst 和 good)的变迁.图 9 线程 scale_speed_data 的线程状态机与转换得到的概率模型(图 3(b))之间的对应关系,如表 3 所示.

此外,线程的截止时间约束、随机错误时延和突发错误时延转换为处理器模块的一部分,并以此强制线程满足截止时间和时延约束,具体的将在处理器构件转换规则部分详细介绍.当线程错过了截止时间,它会从 ready、good、error 或 burst 状态直接进入失效状态.与截止时间相关的变迁按照定义 7 中的规则转换为命令,并且必须与处理器模块中的命令同步,这样处理器就能强制线程满足截止时间约束.例如,图 3(b)第 18 到 22 行描述的:当 scale_speed_data 线程错过截止时间,它会进入失效状态.

表 3 线程状态机(见图 9)和概率模型(见图 3(b))之间的对应关系

状态变迁(图 9)	行号(图 3(b))	状态变迁(图 9)	行号(图 3(b))
0→1	12	1→2	13
2→{2,3,4}	14	2→1	15
4→{2,3,4}	16	3→0	17

4.3.3 处理器转换规则

定义 8 单个处理器的转换规则包括以下几条:

1) 用一个两状态变量来表示处理器是否可用.
2) 用变量 *time* 作为处理器的时间计数器,在一个超周期内为所有线程计数.*time* 的最大值是所有线程组合的超周期,超周期由各个线程的截止时间计算得到.在所有线程开始执行时,*time* 设置为 *0*.当 *time* 等于超周期值时,它被重置为 *0*,用作下一个超周期内的执行计数器.
3) 为每个线程定义一个变量,对一个超周期内的执行次数进行计数.
4) 为每两个相互依赖的线程之间定义一个两状态变量,用于表示前一个线程是否已经执行结束.
5) 对于线程状态机中的每个变迁,处理器模块有一个对应的变迁用来将约束强制施加到线程上,包括

时间约束、随机错误时延、突发错误时延和资源(处理器)约束.处理器模块中变迁与线程模块中对应的变迁具有相同的动作名,使得处理器模块能够与线程模块同步,以便于将时间、时延和资源约束强制施加到线程上.

例如,与线程 scale_speed_data 对应的处理器(processor)概率模型(处理器模块的一部分),如图 10 所示.在这里,假设处理器只有一个核,本章提出的方法能够很容易扩展到多核处理器.第 7 行的 procNum 用于表示处理器是否可用,初始值 1 表示空闲状态,0 表示忙(工作)状态.处理器是线程执行之前必须获得的资源.就绪状态的线程在获得处理器资源后就可以允许.第 8 行定义了变量 time,图 8 第 1 行根据各线程的截止时间定义了超周期.此外,对于线程状态机变迁上的时间约束,图 8 第 12-22 行描述了处理器中用于强制施加线程时间、时延和资源约束的命令,这些命令与图 3(a)(scale_speed_data 线程)第 12-22 行命令动作相同,进而实现同步.

将构件之间的连接看作线程之间的依赖关系,定义 8 将它转换为变量,取值可以是 0 和 1,其中,1 表示前序构件运行完成并生成了结果,0 表示没有结果且后继构件需要等待前序构件的结果.在线程执行开始(从 initial 状态到 ready 状态)时,这个变量值初始化为 0.当前序线程被释放后,这个变量设置为 1,表示线程已经生成了结果.后继构件会判断这个变量是否为 1,然后在从 initial 状态到 ready 状态的变迁发生时,将变量重置为 0,0 可以表示目的构件接收到结果了.例如,构件 scale_speed_data 和构件 controller 之间的连接转换为 scale2CL 变量,如图 10 第 11 行所示.在从 ready 状态到 good(running)状态的变迁(见第 13 行)发生之前,scale_speed_data 构件会判断 scale2CL 是否为 0.如果 scale2CL 等于 1,表示 scale_speed_data 构件已经执行完成并在等待相连的 controller 构件接收结果和开始执行.当 scale_speed_data 构件被释放(见第 15 行),scale2CL 设置为 1.后继的、相连的 controller 构件判断 scale2CL 是否为 1,然后将其重置为 0(见第 24 行).

4.3.4 E2E 的转换规则

E2E 描述了基于构件之间的连接的错误传播关系.将从环境构件到系统内部构件之间的 E2E 转换为模块之间的同步,将系统内部构件之间的 E2E 转换为命令.根据定义 2,E2E 包含三种状态变迁类型:源构件内从错误状态到触发条件再到虚拟状态 es_{has} 的变迁 $t1$;目的构件内从错误状态到触发条件(包括虚拟状态 es_{has} 和虚拟 IEPP 事件 ee_I 的 AND 组合)再到目的构件错误状态的变迁 $t2$;从虚拟状态 es_{has} 到虚拟事件 ee_{dis} 再到虚拟状态 es_{empty} 的变迁 $t3$.对于 $t2$,因为 ee_I 来源于 es_{has},所以可以直接利用 es_{has} 代替它们的 AND 组合.

定义 9 ***E2E* 的转换规则**是 $TR_{e2e}=(A_{e2e},PM_{e2e})$,其中,$A_{e2e}$ 是 *E2E* 包含的变迁集合,使得 $t1\in A_{e2e}$、$t2\in A_{e2e}$ 和 $t3\in A_{e2e}$ 分别源构件内的的变迁和对应的目的构件内的变迁(上述的三类变迁),以两种方式将它们转换为命令,$c1\in PM_{e2e}$、$c2\in PM_{e2e}$ 和 $c3\in PM_{e2e}$.

1) 如果 $t1$ 的触发条件包含虚拟 *IEPP*,转换规则为 $TR_{e2e,1}$:$c1$ 和 $c2$ 的动作名称设置为构件名和触发条件的组合,相互之间用下划线连接.使用定义 5 中的规则 $TR_{lp,1}$,将 $t1$ 的源错误状态和触发条件中的虚拟 *IEPP* 转换 $c1$ 的守卫.在源构件内,将 $t1$ 的目的虚拟状态 es_{has} 转换为一个局部变量 var_{es_has}(*0* 表示没有错误,*1* 表示有错误),将 $c1$ 的变迁设置为该局部变量等于 1,直接作为 $c1$ 的更新(概率值为 *1.0*,可以省略).$t3$ 是使正在传播的错误消失的过程,即,使错误传播过程不处于状态 es_{has} 中,所以,如果构件具有恢复错误变迁,直接将 $t3$ 转换为命令的一个变迁:var_{es_has} 重置为 $0(var_{e's\_has}=0)$,并将其添加到与构件恢复错误变迁对应的命令中;否则,不需要转换 $t3$.此外,对于 $t2$,使用定义 5 中的规则 $TR_{lp,2}$,将 $t2$ 的触发条件转换为 Pr_{lp} 和 $PM_{lp,siepp}$,再将 $PM_{lp,siepp}$ 与 $t2$ 的源错误状态转换为 $c2$ 的守卫.将 $t2$ 的目的错误状态转换为 $c2$ 的变迁,与 Pr_{lp} 构成更新,再为 $c2$ 添加一个更新:"$(1-Pr_{lp}):true$".

2) 如果 $t1$ 的触发条件不包含虚拟 *IEPP*,转换规则 $TR_{e2e,2}$:不转换 $t1$ 的虚拟状态 es_{has},也不需要转换 $t3$,所以 $c1$ 和 $c3$ 为空.$c2$ 的动作名称设置为构件名和触发条件的组合,相互之间用下划线连接.将 $t1$ 的源错误状态、$t2$ 的源错误状态和 $t2$ 触发条件中的虚拟 *IEPP* 转换为 $c2$ 的守卫.将 $t2$ 的目的状态转换为 $c2$ 的变迁.将 $t2$ 触发条件中的错误事件转换为发生概率,与 $c2$ 变迁构成更新,还要为 $c2$ 添加一个

更新:“(1−$p2$):*true*”.

以图 2 中从构件 speed_sensor 到构件 scale_speed_data 的 E2E 为例,对 E2E 转换规则作进一步解释.因为 E2E 在源构件向外传播的变迁的触发条件不包含虚拟 IEPP,如图 12(a)第 9 行,所以可以采用规则 $TR_{e2e,2}$,对于源构件 speed_sensor,不需要转换;对于目的构件 scale_speed_data,将 speed_sensor 变迁(图 12(a)第 9 行)的源错误状态和 scale_speed_data 变迁(图 3(a)第 17 行)的源错误状态转换为守卫,它们分别对应“*state_sensor*=1”(图 12(b)第 3 行)和“*state_scale*=2”(图 3(b)第 8 行),如图 3(b)第 11 行.

```
1. device implementation Sensor.impl
2. annex EMV2{**
3.   use types PMS_EMLib;
4.   use behavior Sensor_EM::Sensor_Beha;
5.   component error behavior
6.     transitions
7.       Operational-[Err]->Failed;
8.     propagations
9.       Failed-[]->sensor_data(Data_Fault);
10. end component;
11. properties
12.   EMV2::OccurrenceDistribution =>[ProbabilityValue => 3.17E-11;
        Distribution=>fixed;] applies to Err; -- 1年发生1次失效的概率
13. **};
14. end Sensor.impl;
```

(a) AADL 模型

```
1. const double delta_sensor=3.17E-11; //1年发生1次失效的概率

2. module Sensor // Sensor
     // 0: operational（正常状态）, 1: failed（失效状态）
3.   state_sensor : [0..1] init 0;

     // 错误事件Err触发的错误变迁
4.   [sensor_Err] state_sensor=0 ->
         delta_sensor : (state_sensor'=1) +
         (1-delta_sensor) : true;
5. endmodule
```

(b) PRISM 模型

图 12 Speed_sensor 构件模型

4.3.5 CEB 的转换规则

根据定义 3,CEB 中的每一条复合错误行为描述了从隐含的可操作状态到以子构件的错误状态和 IEPP 的逻辑组合为触发条件再到复合构件的错误状态的变迁关系,将其转换为一条命令或标签(label).PRISM 可以直接刻画子构件错误状态和 IEPP 对复合构件错误状态的影响,所以,转换复合错误行为过程中不考虑可操作状态.

定义 10 复合错误行为的转换规则是 $TR_{ceb}=(A_{ceb},PM_{ceb})$,使 $aceb\in A_{ceb}$ 是一条 *CEB*:

1) 规则 $TR_{ceb,1}$:如果该构件存在错误状态变迁,即 EM,$pmceb\in PM_{ceb}$ 是转换得到的一条命令.按照定义 5 的逻辑转换规则 $TR_{lp,1}$,将 *aceb* 的触发条件转换为 $PM_{lp,1}$,作为 *pmceb* 的守卫.*aceb* 的目的状态转换为 *pmceb* 的变迁,发生概率取值为 *1.0*,直接将 *pmceb* 的变迁当作更新(概率值缺省相当于概率值为 1.0).
2) 规则 $TR_{ceb,2}$:如果该构件没有错误状态变迁,那么利用 And 和 Or 将触发条件组合起来,再根据定义 5 给出的逻辑操作和逻辑原语的转换规则,将其转换为标签.通过标签直接表示从子构件状态演变而来的状态.

规则 $TR_{ceb,2}$ 可以降低模型复杂度,节约计算资源.例如,图 4(a)第 10 行中的复合错误行为转换图 4(b)中的标签.

4.4 系统模型组装

利用上述制定的模型转换规则,可以将 AADL 安全性模型基本元素、逻辑操作、逻辑原语和基本模型转换为概率模型.先将模型基本元素转换为概率模型元素,再将最底层的构件(包括线程和处理器构件等)的基本模型转换为 PRISM 概率模型.

为了建立完整的 DTMC 模型,依据 AADL 安全性模型中各基本模型之间的关系,第 1 步,在构件级,将 EM 转换得到的概率模型组装为概率模型 PM_{subc},同时,如果线程构件包含扩展的线程状态机,也要将它组装到 PM_{subc} 中;第 2 步,在同层级构件之间,先转换 E2E,再基于 E2E 和 PM_{subc},将具有错误传播关系的子构件集合分别转换为概率模型:$PM_{subs,1}$,$PM_{subs,2}$,...,$PM_{subs,n}$;第 3 步,在不同层级的构件之间,基于复合构件与子构件之间的关

系,将复合构件的 CEB 转换为概率模型,再与子构件集合的概率模型组装起来构成 $PM_{sub2com}$,同时,如果线程构件包含扩展的线程状态机,那么也要将处理器构件转换得到的概率模型集成到 $PM_{sub2com}$ 中.如果步骤 3 中的复合构件不是最高层级的构件,那么返回到步骤 2,重复步骤 2 和 3 将模型转换并集成为概率模型,直到将最上层复合构件集成到概率模型.由此,为系统建立了完整的 DTMC 模型.

5 安全性分析

本文提出的方法旨在分析带有突发错误和随机错误的 MCS 的安全性.时态逻辑属性用于描述所需的线程和系统行为.首先,根据一个定量的安全属性对系统进行验证.然后,如果系统违反了这一安全属性,则制定相应的 CTL 属性以产生证例.由于大多数工程师不熟悉概率模型检验技术,因此将设计实现自动生成这两种逻辑属性公式来帮助工程师节省时间和精力.

MCS 必须遵守混合关键系统 DAL 安全约束,即 PFH.安全属性是基于线程的失效状态、复合错误行为和 DAL 生成的.安全性定量性质表示为:

$$P=?\ [F \le 360000\ \ failed] \tag{6}$$

其中,“failed”是失效状态或导致失效状态的复合错误行为的标签.假设执行线程的时间精度为 10ms,那么一小时内计算的迭代次数为 360000 次,该次数将被底层的有界模型检查算法所使用.公式(6)表示“1 小时内达到失效状态的概率是多少?”

如果公式(6)产生的值大于允许的 PFH,则必须找出违反此安全性约束的原因,以便工程师能够快速发现和修正问题.与公式(6)对应的用于生成见证的定性属性表示为:

$$E\ [F\ \ failed] \tag{7}$$

如果这一性质的验证结果是正确的,即该失效状态是可达的,就可以为随机错误和突发错误的传播分析生成证例.实际上,结果总是可以为真从而生成证例,因为该失效状态已经通过公式(6)验证其违反了安全性约束,也就是发生概率已经大于要求的 PFH.证例是一个动作序列,显示构件如何从初始状态到达失效状态.基于这条路径,设计人员可以通过在这条路径上设置障碍或添加冗余系统来降低失效概率.此外,可以通过证例分析线程的更详细和更低层级的信息,例如发生随机错误和突发错误的次数.然后,设计人员可以减少瞬态和间歇故障对系统架构模型影响,以提高系统的安全性.

然而,从初始状态到失效状态通常有多个证例,因为线程的失效状态可能是由相连的构件、处理器或其本身引起的,例如,通过错误传播从相连的构件传播而来的错误事件与突发错误无关.因此,PRISM 模型检验器生成的证例可能与线程状态机没有关系.在这种情况下,证例不能用于分析随机错误和突发错误.

为了生成与线程的突发错误相关的证例,在实现模型转换工具时,可以不转换从设备到线程的错误传播以及处理器触发的错误行为,即硬件直接故障导致的线程失效.例如,它不需要转换图 12(a)第 9 行的传播和图 3(a)第 17 行的转换,因为它们描述了从设备 speed_sensor 到线程 scale_speed_data 的错误传播.它也不需要转换图 3(a)第 18 行中的错误行为,因为它是由处理器故障触发的.这样,证例将与随机错误和突发错误的发生直接相关.

6 实验分析

6.1 PBA系统建模

采样 PBA 系统对本文提出的方法进行有效性验证.该系统能够匀速巡航,并确保船舶航向适当.控制器 controller 从速度传感器读取当前速度,以计算节流阀的命令.控制器 controller 也从接口单元接收输入,并向显示单元提供状态数据.它包含三个线程,分别是 scale_speed_data、control_law 和 monitor,如图 2 所示.控制器 controller 是 PBA 系统的关键构件,其 DAL 等级为 C.各个线程的 DAL 等级如表 4 的第二列所示,第 3 列是依据表 1 给出的 DAL 对应的 PFH.监控器 monitor 的 DAL 不能低于被监视的组件,因此也定义为 C.每个线程都有

一个线程状态机.线程 control_law 和 monitor 的错误模型与图 3(a)中线程 scale_speed_data 的错误模型相似,仅仅概率和时间属性不同.如果一个线程错过截止日期,它就会失效.各个线程拥有不同连接(connection)和传播.当一个构件的输入构件失效时,它也不能正常工作.在这种情况下,该构件失效.例如,如果 speed_sensor 失效,scale_speed_data 也会失效,因为它没有有效输入.本文假设处理器可能在两年内失效.由于线程必须在处理器上运行,每个线程失败的概率为 1.59E-10(时间精度为 10ms,其中,1.59E-10=1/2 年/365 天/24 小时/60 分钟/60 秒/1000 毫秒*10 毫秒).例如,图 3(b)的第 11 行的命令是从图 3(a)的第 18 行的错误变迁(由处理器故障引起)转换而来的.设备(interface_unit、display_unit、throttle)的错误模型与图 12(a)中的 speed_sensor 的错误模型相似,仅概率属性不一样.每个设备有两个状态,即 operational(good)和 failed.假设 interface_unit、display_unit 和 throttle 可能在一年内因错误事件失效,概率为 3.17E-10(时间精度为 10ms,其中,3.17E-10=1/1 年/365 天/24 小时/60 分钟/60 秒/1000 毫秒*10 毫秒).另外,三个线程突发错误行为的发生概率属性(包括 P_{GB}、P_{BG}、λ_B 和 λ_G)如表 5 第 2 至 5 列所示,分别刻画从正常状态到突发错误状态、从突发错误状态到正常状态、从突发错误状态到错误状态和从正常状态到错误状态的演变;线程状态机的参数包括执行时间、周期、截止时间、突发持续时间和错误持续时间(错误时延)如表 5 第 6 至 10 列,这些参数的时间单位等于时间精度,如 1ms、10ms、100ms、1000ms、10000ms 等.

基于第 3.2 节扩展的概率、时间和 DAL 属性,利用 AADL 和 EMA 构建 PBA 模型.当一个线程失效时,controller 失效.当 controller 或 throttle 失效时,PBA 系统将失效,无法提供服务.当 display_unit 和 interface_unit 都失效时,用户无法控制 PBA 系统,PBA 系统被视为失效.由此可以在高层级系统构件建立复合错误行为.

表 4 定量分析结果

构件名	DAL	PFH	P_{failed}	是否满足关键性等级
Scale_speed_data	D	$< 1.0E-5$	$2.6E-7$	是
Control_law	C	$< 1.0E-6$	$4.4E-5$	否
Monitor	C	$< 1.0E-6$	$1.5E-7$	是
Controller	C	$< 1.0E-6$	$4.5E-5$	否
PBASystem	C	$< 1.0E-6$	$7.3E-5$	否

表 5 随机错误、突发错误和线程状态机的参数

构件名	P_{GB}	P_{BG}	λ_B	λ_G	执行时间	截止时间	周期	突发错误时延	随机错误时延
Scale_speed_data	0.0001	0.35	0.4	0.0001	4	20	20	1	2
Control_law	0.00001	0.3	0.2	0.0001	5	20	20	1	1
Monitor	0.001	0.4	0.2	0.001	5	20	30	1	2

6.2 实验结果

所有的实验都是在 Intel(R) Core(TM) i7-3770 CPU@3.40GHz 和 14GB RAM 的虚拟机上运行的.实验选用 PRISM 的混合计算引擎 *hybrid*;将底层 BDD 和 MTBDD 库的最大内存设置为 3GB,用于 PRISM 模型构建;将 Java 堆空间设置为 10GB,这是 Java 虚拟机(JVM)执行 PRISM 的内存上限;将终止 ε 设为 1.0E-12.

本文以 Eclipse 集成开发环境作为开发框架,设计并实现了安全性分析工具,工具支持从 AADL 模型到 PRISM 模型的自动转换、自动生成定量和定性属性逻辑公式,手动编写脚本实现 PRISM 概率模型检验工具调用、模型加载和属性公式加载等功能.分析工具集成在 AADL 开源环境(OSATE)中,在 Eclipse 框架内便可进入 PRISM 工具指定属性公式进行验证.

本文所提方法的可伸缩性主要取决于 PRISM 工具的可伸缩性,因为模型转换功能不会影响可伸缩性.PRISM 可以分析超过 1.0E11 个状态的模型.PBA 系统 PRISM 模型有 8021120 个状态和 42062607 个变迁.基于公式(6)的定量验证结果如表 4 的第 4 列和第 5 列所示,可以看到失效概率以及是否满足关键性等级.

通过将 PRISM 计算得到的概率(P_{failed})与构件的 PFH 进行比较,可以看到构件 control_law、controller 和 PBASystem 不能满足它们的 DAL.controller 和 PBASystem 系统是复合构件,它们的失效状态的发生概率直接受子构件的影响.为了降低发生概率,需要识别子构件是如何导致失效的.证例能够有效的帮助完成这项工作,有助于识别出为什么构件不能满足 PFH.为了节省篇幅,以 controller 线程为例,下面只分析和验证其失效对应的 CTL 属性(如公式(7)).

```
1. [CL_initial2ready]→ [Scale_initial2ready]→ [Scale_ready2good]→
2. [Scale_good2Others]→ [Scale_good2Others]→[ Scale _good2Others]→
3. [ Scale _good2Others]→ [Scale_release2ready]→[ CL_ready2good]→
4. [CL_Err]
```

图 13 带传播的 control_law 失效状态发生的证例

线程 control_law 进入失效状态的证例,如图 13 所示,是一条动作名称序列,进而支持错误传播分析.scale_speed_data 线程在开始时正确执行,如第 2-3 行所示(四个连续的 scale_good2Others 动作),但是 controller 最终出错并错过了截止时间.

```
1. [Monitor_initial2ready]→[Monitor_ready2good]→[Monitor_good2Others]→
2. [Monitor_good2Others]→[Monitor_good2Others]→[Monitor_good2Others]→
3. [Monitor_good2Others]→[Monitor_burst2Others]→[Monitor_good2Others]→
4. [Monitor_burst2Others]→[Monitor_good2Others]→[Monitor_burst2Others]→
5. [Monitor_good2Others]→[Monitor_burst2Others]→[Monitor_good2Others]→
6. [Monitor_burst2Others]→[Monitor_good2Others]→[Monitor_burst2Others]→
7. [Monitor_good2Others]→[Monitor_burst2Others]→Monitor_good2Others]→
8. [Monitor_burst2Others]→[Monitor_good2Others]→[CL_initial_MissDeadline]
```

图 14 不带传播的 control_law 失效状态发生的证例

为分析突发错误和随机错误对 PBA 系统安全的影响,在模型转换过程中,不转换设备构件触发线程失效的错误传播以及处理器构件触发的线程错误行为,详细方法如第 5 节所述.然后,通过模型检验得到仅与线程 control_law 的失效状态的突发错误相关联的证例,如图 14 所示.线程 Monitor 在开始时正确执行,如第 1-2 行所示(四个连续的蓝色 Monitor_good2Others 动作).之后,线程 Monitor 选择动作 Monitor_good2Others(红色),然后选择 Monitor_burst2Others(绿色).如此,重复这个过程 8 次,如第 2-8 行所示,这意味着出现突发错误的次数是 8 次.在最后一个 Monitor_good2Others 动作之后,control_law 构件执行动作 CL_initial_MissDeadline,随便被识别出错过了截止时间,构件失效.有了这些信息,工程师们可以专注于减少这个数字,通过减少突发错误的发生来提高系统安全性.

6.3 讨论

当前基于 AADL 架构的安全性定量方法主要沿用传统的方法,如故障树分析(FTA)和事件树分析(ETA)等,建模过程中没有综合考虑失效、危险和事故以及它们之间的关系等安全性信息.本文提出的 AADL 架构安全性分析方法,不仅在模型中包含了这些安全性信息,而能够分别从系统架构定量和定性的分析系统安全性.因为提出的方法借助了基于严格的数学理论的概率模型检验技术,使计算结果更为精确,提高了安全性分析的准确性.同时,本文提出的方法利用了 DTMC,能够刻画具有依赖关系的失效模式,而一些传统的方法(如,FTA 和 ETA)很难刻画这种关系,并且不能同时分析多个初始事件.在分析能力方面,本文提出的安全性定量分析方法主要受现有的模型检验工具的影响,因为概率计算能力依赖于 PRISM 工具.

为了使源模型与目标模型转换过程中的属性得到保持,本文为 AADL 模型制定了形式化定义,从而划定

AADL 模型范围、减少模型二义性；所制定的 AADL 模型到目标模型的转换规则中,AADL 模型元素与目标模型元素映射关系明确,不存在一对多的情况,规则确保了 AADL 模型元素转换的一致性;遵循 AADL 模型层次化结构提出模型组装方法,将转换得到的模型元素组装为完整的目标模型,使目标模型保持源 AADL 模型结构语义的一致性.

安全分析所需要的时间是一个重要的性能指标.模型转换和模型组装只需要花费非常少的时间,安全性分析所花费的大部分时间都用于概率模型检验,因此,这里只考虑模型检验耗费的时间.以 1 小时作为 PBA 系统运行的基准时间,时间精度决定了模型检验迭代次数.时间精度相当于单位时间量,是线程每执行一步所耗费的时间,也是模型检验迭代一次所耗费的时间.例如,时间精度为 1000ms,1 小时对应的迭代次数为 3600(1/60 分钟/60 秒).迭代次数是模型检验工具底层有界模型检查算法需要使用的参数值,会直接影响概率计算结果的准确性.这里通过改变时间精度进而分析时间精度与模型检验耗时的关系,其中,将线程离散化执行每一步所需的时间赋值为时间精度,并以时间精度作为硬件(处理器和设备)失效率的时间单位,系统的其他参数值与前面的实验相同.

针对安全性定量分析过程,调整时间精度,对 PBA 系统架构进行分析,得到定量计算值和所需的模型检验时间,如表 6 所示.从表中可以看出时间精度对概率结果和模型检验时间的影响.时间精度越小,概率值精度越高,但计算过程需要更多的迭代次数来收敛.然而,迭代次数太多会导致过多的时间开销的问题,这个问题可能会大于高精确度概率值的优点.因为从表中可以发现,当使用更小的时间精度时,虽然概率值更为精确,但是没有太大的区别,仍然是同一个数量级.例如,表格中时间精度分别为 1ms(迭代次数为 3600000)和 1000ms(迭代次数为 3600)时,计算出的概率值差别很小,但是定量的模型检验所耗费的时间相差特别大.在安全性定量分析过程中,如果不需要过高的计算精确度,那么可以增大时间精度,降低时间成本.

表 6 时间精度对概率结果和计算时间的影响

迭代次数(时间精度) / 构件		3600 (1000ms)	36000 (100ms)	360000 (10ms)	3600000 (1ms)
Scale_speed_data	概率	3.0533E-7	2.6248E-7	2.5820E-7	2.5777E-7
	耗时(s)	421	4080	40597	408672
Control_law	概率	4.4400E-5	4.4350E-5	4.4345E-5	4.4352E-5
	耗时(s)	329	3120	31376	287121
Monitor	概率	1.9822E-7	1.5524E-7	1.5094E-7	1.5081E-7
	耗时(s)	447	4168	41412	413622
Controller	概率	4.4810E-5	4.4696E-5	4.4684E-5	4.4683E-5
	耗时(s)	248	2395	23968	244492
PBASystem	概率	7.3134E-5	7.3205E-5	7.3212E-5	7.3214E-5
	耗时(s)	200	1923	19180	179625

7 相关工作

针对 MCS 的研究受到了广泛的关注[1][2],但是,研究人员主要关注可调度性分析和解决任务最坏情况下执行时间的不确定性.文献[50]提供了一种在分布式平台上调度混合关键实时任务的方法,并使用区域危险分析方法和功能危险分析方法对重新执行施加额外约束,以提高可靠性.文献[35]集成了不同的关键性级别,提出了一种针对硬件瞬态故障的容错 MCS 调度算法.文献[51]提出了一种用于实时系统可靠性敏感设计优化的系统故障树分析方法,使用硬件复用和软件重执行技术来容忍瞬态故障.此外,对于非混合关键系统的突发错误也有一些研究.文献[9]开发了一种基于分位数的实时系统概率可调度性分析方法,并使用两状态离散马尔可夫模型描述随机错误和突发错误.文献[52]将突发故障定义为实时系统受到干扰的有界时间间隔,并提出了可调度性分析的错误恢复策略,以保证故障突发的容错性.文献[53]将突发错误和长度大于 0 的单个错误都称为错误突发,并对由突发错误引起的最坏情况下的实时系统进行了可调度性分析.然而,上述方法没有考虑架构模型中构

件之间的依赖关系.本文提出的方法对影响调度行为的依赖关系进行了建模.此外,本文与上述方法还有另一个不同.由于复合构件的失效可能是由子构件的失效引起的,也可能是由相连的构件通过故障传播引起的,因此针对构件之间的关系,本文提出了故障传播分析方法,分析线程的随机错误和突发错误对系统安全的影响.

EMA 考虑了故障、错误和失效的概念.虽然它可以描述随机错误,但它没有提供突发错误的语义.大多数基于 AADL 的安全分析方法[22-30]没有考虑突发错误.COMPASS[22-24]是评估安全关键系统的系统级正确性、安全性、可信性和性能的工具集,包括需求确认、功能验证、安全性分析、可诊断性分析、性能评估以及故障检测、识别与恢复等.它能够有效支持故障检测、故障隔离和故障恢复开发.COMPASS 基于系统级集成建模(System-Level Integrated Modelling,简称 SLIM)语言,SLIM 是通过修改 AADL 语言而来,对系统实施基于模型的分析与验证,但是也未能支持 MCS 突发错误行为建模.Delange 等人[26][27]引入了 EMA 和故障传播来支持基于 AADL 实现 SAE ARP4761 标准[54]安全分析方法,但是没有考虑突发错误行为.Wei 等人[28][29]创建了危险模型附件,扩展了 AADL 模型,并将 AADL 模型转化为确定性随机 Petri 网(DSPNs),提出了一种架构级的危险分析方法.在模型转换的基础上,他们进一步提出一种基于安全性的集成模块化航空电子系统软件重构方法[55].此外,他们将定量验证技术整合到安全性分析中,将 AADL 模型转换为连续时间马尔科夫链[30].他们还提出了一种使用随机博弈概率模型检验的电网信息物理系统安全分析方法[25].Mian 等人[56]提出了从 AADL 模型到 HiP-HOPS 模型转换框架,用于可靠性分析.所有这些研究都没有明确描述突发错误行为.AADL 行为附件(BA)是另一个子语言扩展,支持行为建模,但是,它未包含突发错误行为和随机错误行为相关的线程状态语义,不支持本文提出的安全性分析方法.因此,本文建立了新的 AADL 属性和语义来扩展 AADL.

由于瞬态故障发生率高、种类多,可造成危险情况,威胁系统安全.Zhou 等人[33]提出了一种网络控制系统故障传播分析方法,并利用 AADL 构建了统一的仿真平台,该方法着重从以对象为中心的结构和以系统为中心的结构两方面进行传播分析.在分析瞬态故障传播特性的基础上,他们提出了一种分层瞬态容错方案,通过系统级的功能和结构模型能够检测和恢复隐藏的瞬态故障[32].他们建立了基于 AADL 的仿真平台,并对任务重分配策略进行了评估.然而,这些方法未能描述随机错误和突发错误的特征,也没有考虑时间和概率特征.此外,本文提出的故障传播分析是基于概率模型检验得到的证例来实现的.通过验证 DTMC 模型获得的证例[57]可以为调试、基于模型的测试等提供必要的诊断信息.证例的生成是模型检验的主要优势之一.模型检查也被 COMPASS[24]采用,它通过验证功能属性来生成反例或证例以实现功能验证,而不是用于安全分析.COMPASS 通过 FTA、FMEA 和时间故障传播图进行安全性分析[22],并将连续时间马尔科夫链应用于 SLIM 模型的性能评估[23].

8 总 结

本文提出了一种基于架构的 MCS 随机错误和突发错误安全分析方法.由于 AADL 核心语言、EMA 和 BA 不支持对瞬态和间歇故障的突发特征进行建模,本文为线程构件提出了突发错误行为线程状态机语义,并设计了 AADL 突发错误属性来描述新扩展的语义.为了利用概率模型检查技术,为 AADL 模型构建形式化定义,制定了将 AADL 模型转换为使用 PRISM 语言描述的 DTMC 模型的规则,并提供了模型组装方法.定义了两种性质公式,明确了系统的安全要求,从而推导出失效状态发生的概率,从而确定系统是否满足安全约束条件,进而获得故障传播分析的证例,并分析得出系统如何达到故障状态.所提出的模型转换规则、组装方法和安全性逻辑属性公式使该方法的实现成为可能,并使其更易于工程师使用.最后,对 PBA 系统进行了分析,验证了安全性分析方法的可用性.

在未来,将研究更多的实际案例,将研究应用更复杂的分析技术(如概率时间自动机)来分析 MCS 随机误差和突发错误,而不仅仅是 DTMC.另外,还计划考虑多核处理器.此外,将寻求一种生成具有最大发生概率的证例的方法,并研究如何优化转换得到的 PRISM 模型以降低状态空间复杂度.

References:

[1] Burns A, Davis RI. Mixed criticality systems-a review. Department of Computer Science, University of York, Tech. Rep, 2022.

[2] Burns A, Davis RI. A survey of research into mixed criticality systems. ACM Computing Surveys (CSUR), 2017, 50(6): 1-37.

[3] Zhang YW, Chen RK. A survey of energy-aware scheduling in mixed-criticality systems. Journal of Systems Architecture,2022. [doi:10.1016/j.sysarc.2022.102524]

[4] Ernst R, Natale DM. Mixed criticality systems—a history of misconceptions?. IEEE Design & Test, 2016, 33(5): 65-74.

[5] Thekkilakattil A, Burns A, Dobrin R, Punnekkat S. Mixed criticality systems: Beyond transient faults. In: Proc. of the 3rd workshop on mixed criticality systems. 2015.18-23.

[6] Chen JJ. Mixed criticality in safety-critical systems. LS 12, TU Dortmund, 2016

[7] Vestal S. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: Proc. of the 28th IEEE international real-time systems symposium. 2007.239-243.

[8] Particle Data Group, Zyla PA, et al. Review of Particle Physics. Progress of Theoretical and Experimental Physics, 2020. [doi:10.1093/ptep/ptaa104].

[9] Short M, Proenza J. Towards Efficient Probabilistic Scheduling Guarantees for Real-Time Systems Subject to Random Errors and Random Bursts of Errors. In: Proc. of the 25th Euromicro Conference on Real-Time Systems. 2013. 259-268.

[10] Storey NR. Safety critical computer systems. Addison-Wesley Longman Publishing Co., Inc., 1996.

[11] McEwen M, Faoro L, et al. Resolving catastrophic error bursts from cosmic rays in large arrays of superconducting qubits. Nature Physics, 2022,18(1):107–111.

[12] Castillo X, McConnel SR, Siewiorek DP. Derivation and calibration of a transient error reliability model. IEEE Transactions on Computers, 1982, 100(7): 658-671.

[13] Avizienis A, Laprie JC, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing[J]. IEEE Transactions on Dependable and Secure Computing, 2004, 1(1): 11–33.

[14] Thekkilakattil A, Dobrin R, Punnekkat S. Fault tolerant scheduling of mixed criticality real-time tasks under error bursts. Procedia Computer Science, 2015, 46:1148–1155.

[15] Narayanasamy S, Coskun AK., Calde B. Transient Fault Prediction Based on Anomalies in Processor Events. In: Proc. of 2007 Design, Automation & Test in Europe Conference & Exhibition.2007. 1-6.

[16] Siddiqua T, Papathanasiou AE, Biswas A, Gurumurthi S. Analysis and modeling of memory errors from large-scale field data collection. ser. SELSE, 2013,16:17-18.

[17] SAE International. AS5506C - (R) Architecture Analysis and Design Language (AADL). SAE International, January 2.

[18] Sha L. Resilient mixed-criticality systems. Crosstalk, 2009, 22(9-10): 9-14.

[19] Dong YW, Wei XM, Xiao MR. Overview: System architecture virtual integration based on an AADL model. In: Proc. of Symposium on Real-Time and Hybrid Systems: Essays Dedicated to Professor Chaochen Zhou on the Occasion of His 80th Birthday. Springer International Publishing, 2018: 105-115.

[20] SAE International. (R) SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex E: Error Model Annex. SAE International, September 2.

[21] SAE International. AS5506/2 - SAE Architecture Analysis and Design Language (AADL) Annex Volume 2: Annex D: Behavior Model Annex. SAE International, January 2.

[22] Bozzano M, Cavada R, Cimatti A, Katoen JP, Noll T, Tonetta S. The COMPASS 3.0 toolset. In: Proc. of the 5th International Symposium on Model-Based Safety and Assessment, 2017.

[23] Bozzano M, Cimatti A, Katoen J P, Nguyen VY, Noll T, Roveri M. Safety, dependability and performance analysis of extended AADL models. Computer Journal,2011, 54(5):754–775.

[24] Bozzano M, Cimatti A, Katoen JP, Nguyen VY, Noll T, Roveri M, Wimmer R. A model checker for AADL. In: Proc. of the 22nd international conference on Computer Aided Verification.2010. 562-565.

[25] Wei XM, Dong YW, Sun PP, Xiao MR. Safety Analysis of AADL Models for Grid Cyber-Physical Systems via Model Checking of Stochastic Games.Multidisciplinary Digital Publishing Institute, 2019,8(2), 212.

[26] Delange J, Feiler P. Architecture fault modeling with the AADL error-model annex. In: Proc. of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA).2014.361-368.

[27] Delange J, Feiler P, Gluch D, Hudak JJ. AADL fault modeling and analysis within an ARP4761 safety assessment. Software Engineering Institute.2014.1-85.

[28] Wei XM, Dong YW, Li XL, Wong WE. Architecture-level hazard analysis using AADL. Journal of Systems and Software, 2018, 137: 580-604.

[29] Wei XM, Dong YW, Yang MM, Hu N, Ye H. Hazard analysis for AADL model. In: Proc. of the 20th International Conference on Embedded and Real-Time Computing Systems and Applications. 2014.1-10.

[30] Wei XM, Dong YW, Ye H. QaSten: Integrating quantitative verification with safety analysis for AADL model. In: Proc. of International Symposium on Theoretical Aspects of Software Engineering. 2015.103-110.

[31] Wei XM, Dong ZQ, Xiao MR, Tian C. Failure Probabilities Allocation and Safety Assessment Approaches Based on AADL. Journal of Software, 2020, 31(6): 1654-1671(in Chinese). http://www.jos.org.cn/1000-9825/5999.htm.

[32] Huang S, Zhou CJ, Yang LL, Qin YQ, Huang XF, Hu BW. Transient fault tolerant control for vehicle brake-by-wire systems. Reliability Engineering & System Safety, 2016, 149:148–163.

[33] Zhou CJ, Huang XF, Naixue X, Qin YQ, Huang S. A class of general transient faults propagation analysis for networked control systems. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2015, 45(4): 647-661.

[34] Lu Y, Qin SD, Xi LQ, Dong YW. On Schedulability Analysis of AADL Architecture with Storage Resource Constraint. Journal of Software, 2021, 32(6): 1663-1681(in Chinese). http://www.jos.org.cn/1000-9825/6243.htm.

[35] Huang PC, Yang H, Thiele L. On the scheduling of fault-tolerant mixed-criticality systems. In: Proc. of the 51st annual design automation conference. 2014.1-6.

[36] Zhou JL, Yin M, Li ZF, Cao K, Yan JM, Wei TQ, Chen MS, Fu X. Fault-tolerant task scheduling for mixed-criticality real-time systems. Journal of Circuits Systems & Computers, 2017,26(01).

[37] Hugues J, Wrage L, Hatcliff J, Stewart D. Mechanization of a Large DSML: An Experiment with AADL and Coq. In: Proc. of the 20th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE). 2022.1-9.

[38] Lu Y, Qin SD, Guo P, Dong YW. Hardware-software Integrated Reliability Modeling and Analysis Using AADL. Journal of Software, 2022, 33(8): 2995-3014(in Chinese). http://www.jos.org.cn/1000-9825/6610.htm.

[39] Kwiatkowska M, Norman G, Parker D. Probabilistic model checking: Advances and applications. Formal System Verification: State-of the-Art and Future Trends.2018.73-121.

[40] Kwiatkowska M, Norman G, Parker D. PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. of the 23rd International Conference on Computer Aided Verification.2011.585-591.

[41] Gilbert EN. Capacity of a burst-noise channel. The Bell System Technical Journal, 1960,39(5):1253–1265.

[42] Feiler PH, Gluch DP. Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language. Addison-Wesley, 2012.

[43] RTCA. RTCA/DO-178C - Software Considerations in Airborne Systems and Equipment Certification. RTCA, Incorporated, 2012.

[44] International Electrotechnical Commission. Functional safety of electrical/electronic/programmable electronic safety related systems. IEC 61508, 2000.

[45] Brown S. Overview of IEC 61508 - design of electrical / electronic / programmable electronic safety-related systems[J]. Computing & Control Engineering Journal, 2000, 11(11).

[46] Huang PC, Pratyush K, Georgia G, Lothar T. Run and be safe: Mixed-criticality scheduling with temporary processor speedup. In Proc. of IEEE 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE) 2015. 1329-1334.

[47] Gethin N, David P. Quantitative verification: Formal guarantees for timeliness, reliability and performance. United Kingdom: London Mathematical Society and the Smith Institute, 2014.

[48] Kwiatkowska M, Gethin N, David P. Stochastic model checking. In Proc. of Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07). 2007: 220–270.

[49] Forejt V, Kwiatkowska M, Norman G, Parker D. Automated verification techniques for probabilistic systems. Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems. 2011.53-113.

[50] Thekkilakattil A, Dobrin R, Punnekkat S. Mixed criticality scheduling in fault-tolerant distributed real-time systems. In: Proc. of 2014 International conference on embedded systems (ICES). 2014. 92-97.

[51] Huang J, Blech JO, Raabe A, Buckl C, Knoll A. Reliability-aware design optimization for multiprocessor embedded systems. In: Proc. of the 14th Euromicro Conference on Digital System Design. 2011. 239-246.

[52] Many F, Doose D. Scheduling analysis under fault bursts. In: Proc. of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium. 2011.113-122.

[53] Aysan H, Dobrin R, Punnekkat S, Johansson R. Probabilistic schedulability guarantees for dependable real-time systems under error bursts. In: Proc. of the 10th International Conference on Trust, Security and Privacy in Computing and Communications. 2011.1154-1163.

[54] ARP SAE. 4761. Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. SAE International,2,1996.

[55] Wei XM, Dong YW, Xiao MR. Safety-based software reconfiguration method for integrated modular avionics systems in AADL mode. In: Proc. of 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). 2018. 450-455.

[56] Mian ZB, Bottaci L, Papadopoulos Y, Mahmud N. Model transformation for analyzing dependability of AADL model by using HiP-HOPS. Journal of Systems and Software,2019,151:258-282.

[57] Ábrahám E, Becker B, Dehnert C, Jansen N, Katoen JP, Wimmer R. Counterexample generation for discrete-time Markov models: An introductory survey. Formal Methods for Executable Software Models: 14th International School on Formal Methods for the Design of Computer, Communication, and Software Systems.2014. 65-121.

附中文参考文献:

[31] 魏晓敏, 董泽乾, 肖明睿, 田聪. 基于 AADL 的失效概率分配及安全性评估方法. 软件学报, 2020, 31(6): 1654-1671. http://www.jos.org.cn/1000-9825/5999.htm

[34] 陆寅, 秦树东, 习乐琪, 董云卫. 面向 AADL 模型的存储资源约束可调度性分析. 软件学报, 2021, 32(6): 1663-1681. http://www.jos.org.cn/1000-9825/6243.htm

[38] 陆寅, 秦树东, 郭鹏, 董云卫. 软硬件综合 AADL 可靠性建模及分析方法. 软件学报, 2022, 33(8): 2995-3014. http://www.jos.org.cn/1000-9825/6610.htm