计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(9): 2362−2381, 2025

机载系统软件需求的建模与分时组合验证方法

黄益柯[1] 阮 锟[1] 陈小红[1] 金 芝[2,3]
[1](上海市高可信计算重点实验室(华东师范大学) 上海 200062)
[2](高可信软件技术教育部重点实验室(北京大学) 北京 100871)
[3](北京大学计算机学院 北京 100871)
(huangyike@stu.ecnu.edu.cn)

Modeling and Compositional Verification Method by Time Period for Airborne System Software Requirements

Huang Yike[1], Ruan Kun[1], Chen Xiaohong[1], and Jin Zhi[2,3]
[1](*Shanghai Key Laboratory of Trustworthy Computing* (*East China Normal University*), *Shanghai* 200062)
[2](*Key Lab of High Confidence Software Technologies* (*Peking University*), *Ministry of Education*, *Beijing* 100871)
[3](*School of Computer Science*, *Peking University*, *Beijing* 100871)

Abstract Airborne systems serve a pivotal function in the aerospace industry. Their exceptional safety requirements make the formal verification of software requirements a critical and pressing issue. However, with the ever-increasing complexity of airborne system requirements and the growing number of onboard devices, formal verification has encountered the serious challenge of state space explosion. To alleviate this problem, this paper proposes a novel approach for the modeling and compositional verification of airborne system requirements based on time partitioning. This approach utilizes the time dimension to decompose a complex verification system into mutually independent components, enabling the independent verification of each component and subsequently synthesizing the overall verification results of the entire system. The feasibility and practicality of the proposed method are demonstrated through a real-world case study. The evaluation further shows that the proposed method not only enables the verification of systems that are beyond the capability of traditional monolithic verification approaches and effectively alleviates state space explosion, but also avoids false alarms that may arise from neglecting time-based partitioning. This innovative method provides a new and promising technical path for the formal verification of software requirements in airborne systems, contributing to enhanced verification accuracy and efficiency, and ultimately ensuring the operational safety and reliability of airborne systems.

Key words requirements modeling; formal verification; requirements engineering; airborne system; device model

摘 要 机载系统在航空航天领域起着至关重要的作用,其突出的安全性使得软件需求的形式化验证成为一个非常重要的问题.但是随着机载系统需求复杂度和设备数量的增加,其形式化验证中出现了状态空间爆炸的问题.为了缓解该问题,提出了一种机载系统需求的建模与分时组合验证方法.这种方法通过利用时间维度,将复杂的验证系统分解为相互独立的组件,实现了对各组件的独立验证,进而综合出整个系统的验证结果.通过实际的案例研究,证明了该方法的可行性.并通过评估说明该方法不但可以验证一些传统单体验证不能验证的系统,缓解状态空间爆炸,而且可以避免不考虑分时所造成的误报.这一方法

收稿日期: 2024-01-03; **修回日期**: 2025-04-22
基金项目: 国家自然科学基金项目(62272166, 62192731, 62192730)
This work was supported by the National Natural Science Foundation of China (62272166, 62192731, 62192730).
通信作者: 陈小红(xhchen@sei.ecnu.edu.cn)

为机载系统的软件需求验证提供了一种新的技术途径，有助于提高验证的准确性和效率，确保机载系统的安全性.

关键词 需求建模；形式化验证；需求工程；机载系统；设备模型

中图法分类号 TP311

DOI: 10.7544/issn1000-1239.202440003 **CSTR:** 32373.14.issn1000-1239.202440003

机载系统在飞机的整个飞行任务中起着至关重要的作用，其失效会造成不可估量的后果. 因此，保障机载系统的安全性已成为研究热点[1-2]. 鉴于其突出的安全性[3]，机载系统在开发前很有必要进行形式化验证[4]，尤其是在早期的需求阶段. 一些行业标准，例如 DO-178C 标准[5] 和 DO-333 标准[6] 建议开发安全关键软件时应用形式化方法，以实现软件的高安全性.

然而，随着机载系统复杂度的增加[7]，机载系统需求复杂度也相应增加，需求的形式化验证面临状态空间过大的问题，即状态空间爆炸问题[8]，这使得需求形式化验证难以进行. 另外，由于机载系统需求的强设备依赖性，其描述中会涉及大量与物理设备相关的接口，调用接口的顺序必须与物理设备允许的顺序相同，所以需求的形式化验证必须考虑设备内在的物理特性. 但由于机载系统中涉及的设备众多，这又进一步加剧了状态空间爆炸的问题.

组合验证是一种有效缓解状态空间爆炸的手段[9]. 它的核心思想并非直接验证复杂的系统，而是分而治之，分别验证其子系统和组合，确认验证属性的正确性. 目前已有很多组合验证方法[10-16]. 其中最常见的是假设-保证推理（assume-guarantee reasoning, AGR）范式[10-11]. 它基于一个核心理念：组件在满足一定的“假设”条件下，应确保能够提供预期的“保证”并正确执行其功能. 然而，如何为 AGR 精确地定义这些“假设”条件，仍是实际应用中的挑战性难题[12]. 此外，也有工作致力于简化验证系统. 一些研究[14-16] 利用局部性原则将系统中与特定性质相关的组件分离出来以便单独验证. 此类方法虽然在特定场景下表现出有效性，但适用范围受限于具体领域性质[14-16]. 同时，一部分研究[11,13] 试图将复杂系统分解为多个相对独立组件分别进行验证，其研究重点在于分解策略.

本文重点关注以复杂系统分解策略为中心的组合验证. 现有的分解工作[11,13] 针对轨道交通联锁系统存在多条运行线路与轨道网络的特点，将最小化组件间接口变量或者基于车站布局结构的簇切割作为分解维度. 而在机载系统中，我们观察到其具有分时段运行的特征，即系统在不同的时段调用不同的功能. 例如，飞机从起飞到爬升至安全高度的过程可以划分为 4 个阶段：地面滑行、离地升空、加速平飞以及爬升至安全高度. 在这些阶段中，每个阶段都需要执行特定的功能，而这些功能之间是互斥的，不能同时进行. 据此，本文提出一种基于分时段的组合验证方法，以时间为维度，将复杂机载系统需求分解为在不同的时间段调用的功能集合. 通过采用这种策略，可以只验证在某一具体时间段内并行执行的相关功能，减少单次运行的状态空间，从而提高验证过程的效率和准确性.

必须指出的是，在机载系统的运作中，每个时间段内并行运行的功能必须与物理设备紧密协作[17]. 鉴于机载系统功能的广泛性，以及在不同运行阶段接入的设备类型和数量的多样性，这就要求系统能够在不同的时间段内确定具体的操作设备，并确保同一设备在物理约束上的一致性. 因此，需求建模要包含设备相关的知识. 在现有的需求建模方法中[18]，问题框架方法[19] 通过问题领域对设备进行显式建模，有效地表达了嵌入式需求对设备的强依赖性. 本文也选择了基于问题框架方法的嵌入式系统需求建模方法[15]，利用问题图[19] 和情景图[20] 两种模型实现动静结合建模. 至于设备知识，可以借鉴基于环境建模的需求工程方法[21]，通过设备模型库来提供.

然而，这些需求模型在描述时段信息方面仍然存在不足. 为了应对机载系统的分段运行特征，本文借鉴文献 [3, 22−24] 中的模式和模式转换概念，提出使用系统模式切换图扩展建模方法，利用模式概念封装不同时段运行的功能. 此处模式是指软件系统在特定条件下的行为或响应方式. 通过为模式内即每个运行时段的功能分别生成验证模型进行验证，最后对结果综合分析. 验证模型构建过程中，依据需求模型及其涉及的物理设备进行动态组装. 物理设备特性已在设备模型库进行预先定义. 此方法不仅能够适应不同需求对应的设备变化，还能确保设备物理约束的一致性.

具体而言，在形式化模型的选择中，鉴于机载系统的强时间约束，我们选择了时钟约束规约语言

(clock constraint specification language, CCSL)[25]作为形式化语言.这是因为CCSL能够结合事件与逻辑时间,对时间信息进行有效描述.此外,考虑到设备的因果特性及时间性,本文采用时间自动机[26]进行建模.特别地,现有的安全需求一致性验证手段[27]可以捕获验证模型中CCSL约束的最小不满足集合.在辅助定位需求建模结束后,可以揭示系统行为与大量物理设备接口调用之间存在的不一致性等,从而为建模人员的后续修改完善提供建议,提高方法实用性.

我们选取了一个具体的机载系统实例进行案例研究,以验证本文方法的可行性.同时,我们设计了6组代表性实验,将本文方法与单体验证方法进行了对比分析.实验结果表明,本文方法能够验证单体验证方法无法验证的系统,缓解了状态空间爆炸的问题,适用于复杂机载系统的需求验证.同时,可以避免因忽略分时特性导致的误报,提高验证准确性.

本文的主要贡献包括2个方面:

1)建立了机载系统的需求半形式化建模方法,该方法扩展了基于问题图和情景图的嵌入式系统需求建模方法,既考虑了其强设备依赖性,又能通过引入系统模式切换图来表达分时段的功能.

2)定义了针对模式的机载系统软件需求分时组合验证方法,利用模式建立有效的验证模型,进而验证功能的组合,有效缓解状态空间爆炸问题.

1 相关工作

本文的相关工作包括机载系统的建模与验证以及需求的组合验证.

1.1 机载系统的需求建模与形式化验证

目前,很多建模与形式化验证方法已经被应用于机载系统,它们使用各种各样的模型.比如,文献[28]用Event-B模型和Simulink建模整个引擎控制系统的行为,然后在精化过程中监控模型与需求之间是否存在冲突;文献[29]用变量关系模型对机载系统逻辑功能需求进行结构化建模,然后转换为分支时序逻辑后调用工具验证;文献[30]将一个非正式需求规格说明构建成一个类似有限状态机的模型后,通过符号模型检查器验证需求中的时序属性.然而,这些方法在建模过程中缺少对机载系统分段运行特征的考虑,验证采用单体验证方式,并未考虑到状态空间爆炸问题.

近些年来,有很多研究工作在建模中加入了机载系统分段运行特征.大部分用模式和模式转换来描述系统运行过程中的分时段运行[3,23],继而采用线性时态逻辑[3]、时序逻辑[23]等描述安全性质进行模型检查.例如文献[24]中嵌入式控制软件的模式体现了嵌入式周期控制系统的行为特征.每个模式周期性地执行一系列过程,系统在某一模式中运行,直至满足特定的迁移条件,之后系统可能进入另一模式.尽管这些方法刻画了系统分段运行的特征,但验证过程中仍然采用单体验证的方式.

1.2 需求的组合验证

在简化场景下对需求的组合验证方案探索中,一些研究试图将与特定属性相关的组件进行划分,并在此基础上验证组件是否满足该属性.例如,Yuan等人[14]基于问题框架方法建模了属性与设备之间的关系,据此提出了2种方法来减小验证空间:一是通过投影技术将系统分解为较小的子问题;二是通过分析变量间的约束关系来移除冗余变量.刘筱珊等人[15]也对轨道交通区域控制器系统需求进行问题框架建模,通过投影对需求进行分解,分别生成了安全需求验证模型.上述的组件划分是基于局部性原则,仅选取与待验证性质相关的系统部分构成组件.需要注意的是,组件之间不一定完全独立,其间可能存在部分重叠,因此大部分采用的是投影技术.

跟本文最相似的工作是以分解为中心的方法,它们依据特定的维度将系统分解为多个独立组件分别进行验证,最后综合系统验证结果.例如,Limbrée等人[11]通过最小化切割所得接口变量的原则,将一个轨道联锁系统分解为2个子系统组件以及1个接口组件,分别建立较小规模的模型以验证安全属性.Haxthausen等人[13]基于轨道网络的特点,在任意2个非边界轨道段之间进行切割,将复杂铁路系统网络分解为多个子网络,分别验证其安全属性后进行组合.上述研究集中在轨道交通领域,设计了特定的分解策略,不能直接应用于机载系统.与之相比,本文提出的分时组合验证,从时间维度对需求进行分解,将复杂的机载系统需求划分为在不同时间段内的独立功能集合.这种方式允许在相同时间段内仅处理部分相关功能,从而可能实现复杂需求的验证,提高效率.

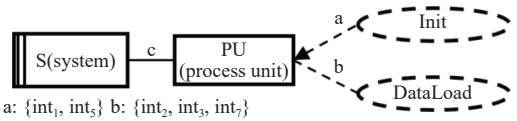
2 预备知识

本文将简要介绍问题图、情景图以及CCSL的基本概念.为了更好地理解,我们首先介绍一个运行案例,其问题陈述如下:

该系统在工作状态下，对采集接收单元上报原始脉冲信息进行周期分选、识别，并形成目标信息上报给显控设备进行目标显示及标绘.

2.1 问题图和情景图

问题图[19]是问题框架方法中的需求模型，以领域（含机器领域与问题领域）、交互、需求为核心描述软件开发问题. 机器领域指待开发系统，用双竖线矩形表示；问题领域指与软件交互的现实实体，用矩形表示. 图 1 是运行案例的问题图，其中，“S(system)”为机器领域，“PU(process unit)”为问题领域. 领域之间通过接口共享一组可观察的现象，即形成了交互. 我们主要关注交互的发起者、接收者和交互现象，即可表示为四元组〈发起者，接收者，现象〉[现象类型]. 其中，现象类型有 4 种，分别是指令（instruction）、信号（signal）、数值（value）和状态（state）. 需求描述了希望机器领域作用于问题领域后产生怎样的效果，用虚线椭圆表示，如图 1 中的“Init”. 机器领域与问题领域之间的交互被称为行为交互，如图 1 中 S 发送指令 InitCommand 给 PU(int_8)，而问题领域预期发生的交互被称为期望交互，如图 1 中的需求 Init 期望 PU 进入状态 PUInit (int_5). 特别地，在问题图中，需求引用交互现象用无箭头虚线表示，需求约束交互现象用带箭头虚线表示. 例如，需求 Init 约束 PU 进入状态 PWork (int_1) 和状态 PUInit (int_5)，即 Init 与 PU 之间是需求约束 a，如图 1 所示.



a: {int_1, int_5} b: {int_2, int_3, int_7}
c: {int_4, int_6,int_8, int_9, int_{10}}
int_1:<PU, –, PWork>[state]
int_2:<PU, –, PWork>[state]
int_3:<PU, S, SortData>[value]
int_4:<PU, S, DataLoadResult>[state]
int_5:<PU, –, PUInit>[state]
int_6:<PU, S, InitResult>[state]
int_7:<PU,–, DataLoad>[state]
int_8:<S, PU, InitCommand>[instruction]
int_9:<S, PU, LoadCommand>[instruction]
int_{10}:<PU, S, PUDataLib>[value]

Fig. 1 Problem diagram of the running example

图 1 运行案例的问题图

情景图[20]在问题图的基础上描述软件和显示领域之间的交互关系. 其中，节点包括行为交互节点、期望交互节点和控制节点，前二者源自问题图中的行为交互和期望交互. 图 2 展示了需求“Init”的情景图，其中交互节点 int_1(PWork), int_5(PUInit), int_6 (InitResult), int_8(InitCommand) 均来自图 1，各交互节点旁仅标注了其现象名称. 其中，int_6 和 int_8 为行为交互，int_1 和 int_5 为期望交互. 除此之外，其他节点为控制节点，主要包括开始节点、终止节点、分支节点、合并节点、判断节点. 节点之间主要包括 5 种关系类型：行为序关系（BehOrder）、期望序关系（ExpOrder）、行为使能关系（BehEna）、期望使能关系（ExpEna）和同步关系（Syn）. 行为序关系表示行为交互的顺序，如图 2 中 int_8 与 int_6 之间的关系. 期望序关系表示期望交互的顺序，如图 2 中 int_5 与 int_1 之间的关系. 行为使能关系表示行为交互使能期望交互发生，如图 2 中 int_8 与 int_5 之间的关系. 此外，期望使能关系表明期望交互要求行为交互的发生. 而同步关系强调了行为交互与期望交互的同时发生性.

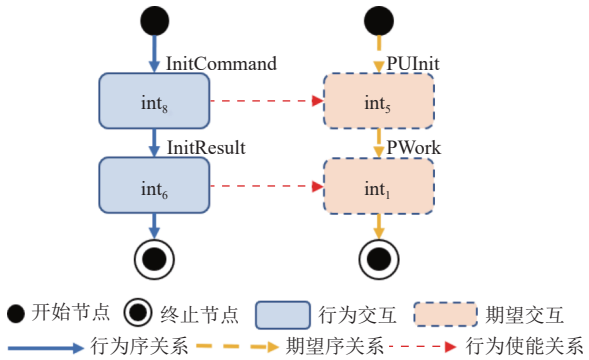


Fig. 2 Scenario graph of requirement Init

图 2 需求 Init 的情景图

2.2 CCSL

CCSL 是实时和嵌入式系统建模与分析（MARTE）规范[31]的伴随语言. 时钟被用于建模对时间的访问. 时钟 C 可以形式化地定义为一个三元组 $(I,\prec,u)$. 其中，I 是一组时刻；$\prec$ 被称为严格先于，是 I 上的一种严格的顺序关系，它是非对称的和可传递的；u 是一个代表时间单位的符号. $C[k]$ 表示 I 中的第 k 个时刻，k 为索引. 任何事件都可以是一个（逻辑）时钟，事件的每一次出现都是该时钟的一个时刻，也称为滴答[32]. 对于来自不同时钟的任意 2 个时刻，除严格先于关系之外，还有另外 2 种关系，即先于和同步. 先于关系（$\preceq$）表示时刻之间的因果依赖关系. 同步关系（$\equiv:=\preceq\cap\succeq$）是一种强关系，它强制时刻同时出现. 严格先于关系（$\prec:=\preceq\setminus\equiv$）实际上是一种更强的先于关系.

基于时刻之间的约束，可以对时钟之间的关系进行定义. 所有的时钟约束都可以用来描述事件之间的关系. 例如，时钟约束 C_1 和 C_2 之间的先于关系（StrictPre, $\prec$）被定义为：$i\in\mathbb{N}_+$, $C_1[i]<C_2[i]$, C_1 和 C_2 之间的因果关系（Causality, $\preceq$）被定义为：$i\in\mathbb{N}_+$, $C_1[i]\leq C_2[i]$. 本文中用到的其他时钟关系还包括：子时钟（Subclock, $\subseteq$）、并（Union, +）、延迟（DelayFor, \$）、同步（Coincidence, =）、互斥（Exclude, #）、上确界（Infimum, $\wedge$）、下确界（Supremum, $\vee$）、交替（Alternate, ~）. 这

些约束的具体含义参见表 1. 可以参考文献 [33−34] 了解这些约束的更多细节. CCSL 约束可以使用工具 MyCCSL[27]①进行分析, 它将 CCSL 约束转换为 SMT (satisfiability modulo theories)公式, 用做可调度分析, 若可调度, 则返回“sat”. 若不可调度, 则返回“unsat”, 该情况下可以调用 SMT 公式的分析工具得到产生冲突的公式集合.

3 方法框架

图 3 给出了本文的方法框架, 分为 4 个步骤:

步骤 1. 需求建模. 根据机载系统设备敏感的特性, 本步骤基于问题框架方法进行需求建模, 使用问题图来静态描述需求, 使用情景图来刻画如何动态地实现需求. 另外, 考虑到机载系统具有周期性、逻辑状态组合繁多、模式切换条件复杂等特点, 为了更好地适应复杂机载系统任务的需求, 我们引入了系统模式切换图. 该图用于描述系统模式的迁移及其条件, 每个模式内包含 1 个或多个情景图, 这些情景图可以并行或顺序地执行.

步骤 2. 验证系统架构生成. 本步骤基于上一步所得的需求模型, 结合设备模型的调用情况, 生成验证系统的架构. 我们将一段时间内可单独运行的功能对应到一个验证子系统. 在本文中, 系统模式切换图中的某一模式下的功能交互及其相关设备的物理约束共同构成一个验证子系统. 因此, 一个系统模式切换图可以生成多组验证系统, 分别进行验证.

步骤 3. 可执行验证模型生成. 本步骤将根据验证子系统架构生成可执行的验证模型. 由于每个子系统中需求实现顺序交互共享的情况以及设备模型的使用都有所不同, 我们提出了动态组装方案, 根据每个需求中涉及的设备, 组装出相应的验证模型. 它将需求模型和设备模型中存在的因果顺序关系与时间关系都转换为相应语义的 CCSL 约束用以构建验

Table 1 Syntax and Semantics of CCSL Constraints Used in This Paper

表 1 本文使用的 CCSL 约束的语法和语义

约束名称	语法	代数语义	描述
StrictPre	$C_1 < C_2$	$\forall i \in \mathbb{N}_+, C_1[i] \prec C_2[i]$	C_1 的第 i 次滴答严格先于 C_2 的第 i 次滴答
Causality	$C_1 \leqslant C_2$	$\forall i \in \mathbb{N}_+, C_1[i] \preceq C_2[i]$	C_1 的第 i 次滴答不严格先于 C_2 的第 i 次滴答
Subclock	$C_1 \subseteq C_2$	$\forall i \in \mathbb{N}_+, \exists j \in \mathbb{N}_+, C_1[i] \rightarrow C_2[j] \equiv C_1[i]$	如果 C_1 发生, 则 C_2 发生
Union	$C_1 = C_2 + C_3$	$\forall i \in \mathbb{N}_+, \exists j,k \in \mathbb{N}_+, (C_1[i] \equiv C_2[j]) \vee (C_1[i] \equiv C_2[k])$	定义时钟 C_1, 如果 C_2 或 C_3 发生, 则 C_1 发生
Coincidence	$C_1 \equiv C_2$	$\forall i \in \mathbb{N}_+, C_1[i] \equiv C_2[i]$	$C_1[i]$ 和 $C_2[i]$ 的第 i 个时刻同时发生
Exclude	$C_1 \# C_2$	$\forall i,j \in \mathbb{N}_+, \neg(C_1[i] \equiv C_2[j])$	$C_1[i]$ 和 $C_2[j]$ 不在同一时刻发生
Alternate	$C_1 \sim C_2$	$\forall i \in \mathbb{N}_+, C_1[i] \prec C_2[i] \wedge C_2[i] \prec C_1[i+1]$	限制左右时钟的交替滴答
Infimum	$C_1 = C_2 \wedge C_3$	$\forall i \in \mathbb{N}_+, (C_3[i] \preceq C_2[i] \rightarrow C_1[i] \equiv C_3[i]) \vee (C_2[i] \preceq C_3[i] \rightarrow C_1[i] \equiv C_2[i])$	C_1 的第 i 次发生不快于 C_2 和 C_3 的第 i 次发生
Supremum	$C_1 = C_2 \vee C_3$	$\forall i \in \mathbb{N}_+, (C_3[i] \preceq C_2[i] \rightarrow C_1[i] \equiv C_2[i]) \vee (C_2[i] \preceq C_3[i] \rightarrow C_1[i] \equiv C_3[i])$	C_1 与 C_2 和 C_3 较快地同时发生
DelayFor	$C_1 = C_2\$d$ on C_3	$\forall m \in \mathbb{N}_+, \exists i,j,k \in \mathbb{N}_+, (C_1[i] \equiv C_2[k]) \wedge (C_3[j] - C_3[m] = d)$	当 C_3 发生 d 次, C_1 和 C_2 一起发生
Periodicity	$C_1 = C_2 \propto p$	$\forall n \in \mathbb{N}_+, C_1 \in \delta(n) \Leftrightarrow (C_2 \in \delta(n) \wedge \exists m \in N^*, \chi(C_2, n) = m \times p - 1)$	C_1 必须在 C_2 的每 p 次出现时执行 1 次

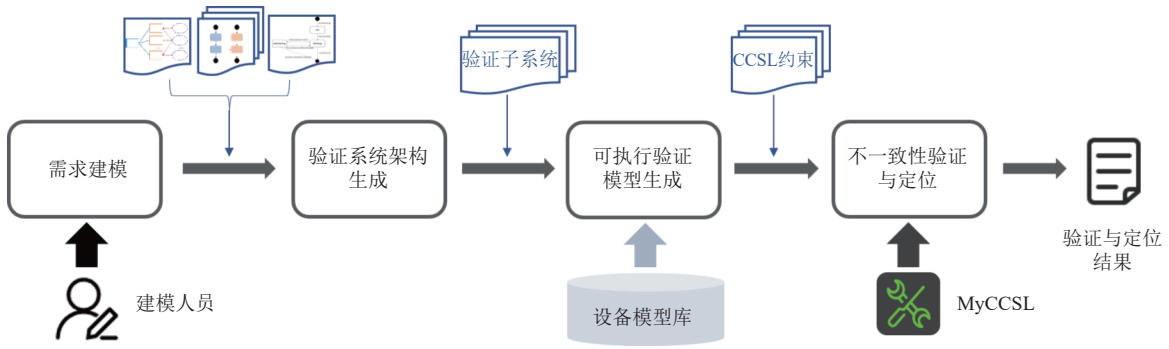


Fig. 3 Framework of our approach

图 3 本文方法框架

① http://re4cps.org/download

证模型. 关键在于, 多情景图中可能涉及对同一设备的操作, 同时设备自身的约束也可能导致重复约束问题, 需要根据具体情况进行相应的处理.

步骤 4. 不一致性验证与定位. 本步骤以每个验证模型的 CCSL 约束集为输入, 使用验证器 MyCCSL 的可调度分析进行需求一致性分析. 如果一致, 将得到验证结果“sat”, 否则将返回“unsat”. 若存在不一致, 则通过工具 MARCO[35] 获得约束集中的最小不满足子集. 结合在验证模型生成阶段记录的约束转换映射表, 能够将不一致性定位到需求模型的情景图当中. 这有助于需求建模人员对机载系统的需求一致性有更加直观、高效的判断认识, 并做出相应的修改调整.

需要注意的是, 在步骤 3 中, 需调用设备模型库, 该库是由领域专家根据具体设备信息手动创建得到. 每个设备采用时间自动机进行建模, 具体的构建过程可以参见文献 [36], 可以分为传感器建模与作用器建模. 由于设备库中各个设备均具有独立性, 因此可以便捷地添加或移除某个设备而不影响其他设备, 这种设计赋予了设备库良好的可扩展性.

4 方法细节

4.1 需求建模

本文使用基于问题框架的需求模型, 并在其基础上扩展了系统模式切换图, 包含 3 个步骤:

1)构建问题图

根据文献 [37], 问题图的构建总共分为识别系统边界、确定系统需求、推导行为交互、检测问题图 4 步. 首先, 识别系统边界, 确定需协调的系统设备及与外部环境的交互实体与接口. 随后, 确立系统需求, 并阐述其对外部及内部设备的影响. 进而, 依据期望交互及领域物理特性, 推导出软件行为模式, 确保每个交互均有对应行为定义. 最终, 验证问题图的完整性, 若不完整则返回违反的约束以示建模人员进行修改. 例如在图 1 中, 将待开发系统标识为“S(system)”, 加入识别出的受系统控制设备“PU(process unit)”以及它们之间交互的接口. 再根据需求 Init 对 PU 的期望作用, 识别出期望交互 int_1; 类似地, 识别其余需求对 PU 的期望交互. 然后根据期望交互 int_1, 识别出 S 与 PU 之间为了满足 int_1 将要发生的行为交互 int_6. 最后对结果进行完整性检查.

2)构建情景图

根据文献 [15], 对于每个需求, 我们都要构建一个情景图来描述它的实现, 主要步骤包括: ①从问题图获得需求的行为交互和期望交互; ②确定这些交互间的关系; ③对生成的情景图进行正确性检测. 我们以图 4 中情景图 DataLoad 为例进行说明.

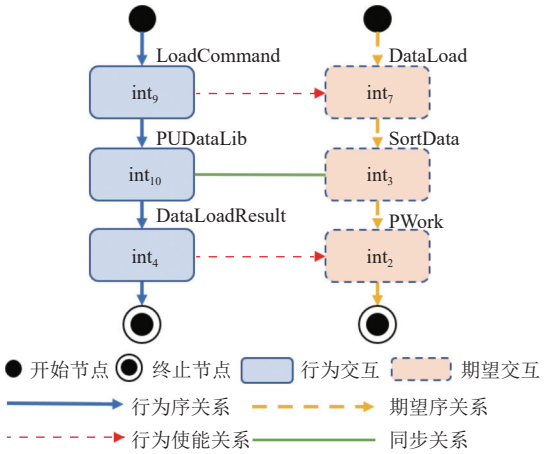


Fig. 4 Scenario graph of requirement DataLoad

图 4 需求 DataLoad 的情景图

首先, 待开发系统 S 先发送数据加载命令给 PU (int_9), 再接收 PU 发送的数据 (int_{10}), 因此行为交互 int_9 与 int_{10} 存在行为序关系. 需求期望 PU 进行数据加载 (int_7), 再将数据发送再给 S (int_3), 所以期望交互 int_7 与 int_3 存在期望序关系; 而 PU 将数据发送给 S (int_{10}) 时, 同时期望另一现象 (int_3) 的发生, 所以 int_{10} 与 int_3 间存在同步关系; S 向 PU 发送加载命令 (int_9) 后, 期望 PU 进入数据加载状态 (int_7), 所以 int_9 和 int_7 间存在行为使能关系. 最后检测情景图的正确性, 查看结果当中是否含有违反的约束.

3)绘制系统模式切换图

在获得情景图后, 需要绘制系统模式切换图来描述复杂机载系统在各个模式中的功能以及模式间的切换条件. 系统模式切换图的基本建模元素是“模式”. 根据文献 [22], 我们将模式视作软件系统特定条件下的行为或响应方式, 每个模式都描述了系统中一组功能. 1 个系统会存在多种模式, 而系统在任何情况下只能处于 1 种模式中. 在本文中, 我们将机载系统中同一时段下多个功能情景及其相互关系进行划分, 并用模式 (*Mode*) 加以描述, 可以形式化定义为:

$$Mode := \langle SS, RS, TC \rangle.$$

①SS 表示 1 组模式内情景的集合;

②$RS \subseteq SS \times R \times SS$ 表示模式内情景关系的集合. 其中, R 表示 2 个情景之间的执行关系, 可以为并行或者顺序执行, 即 $R \in \{;, \|\}$;

③TC 为 1 组情景运行时间的约束, 可以表示为

$Runtime(s)<(\leqslant)t$ 的形式, 其中 $s\in SS$, $t\geqslant 0$.

在对模式划分定义的基础上, 我们检查任意 2 个模式间是否存在切换, 以及切换时的触发事件, 对系统模式切换图(*SMSD*)完整形式化定义为:

$$SMSD := \langle MS, IM, TS\rangle.$$

①*MS* 表示 1 组模式的集合;

②*IM* 为初始模式;

③$TS \subseteq MS \times T \times MS$ 表示系统模式切换关系的集合. 其中, *T* 表示模式之间所存在的触发事件.

图 5 中给出了一个系统模式切换图的例子. 系统具有 $Mode_1$, $Mode_2$ 两个模式, 每个模式的定义在表 2 中, 如在 $Mode_2$ 模式下, 情景 SG1, SG2, SG3 并行地执行, 并且它们受到的时间约束分别为 5 s, 6 s, 5 s. 需要指出的是, 不同模式之间的任务可能存在重叠, 例如 $Mode_1$ 与 $Mode_2$ 模式下都可执行 SG1. 模式迁移的条件涉及系统运行过程中的指令和信号等因素. 如图 5 所示, 当系统处于 $Mode_1$ 模式时, 若检测到 $trigger_2$ 命令, 系统将切换至 $Mode_2$ 模式.

为了获得系统模式切换图, 建模人员需要自定义系统模式及其切换规则. 针对模式, 根据其定义我们制定了以下原则: ①系统在一段持续时间内执行功能的数量保持恒定的状态即为一个模式; ②系统在同一时间段内只能处于 1 种模式, 该模式应涵盖当前时间下所有可执行的场景和功能; ③模式之间

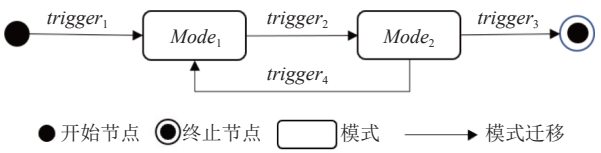


Fig. 5 System mode switching diagram

图 5 系统模式切换图

Table 2 Mode Definitions Involved in the System Mode Switching Diagram

表 2 系统模式切换图中涉及的模式定义

模式	包含的情景	时间约束
$Mode_1$	SG1; SG4	*Runtime*(SG4)<4 s *Runtime*(SG1)<5 s
$Mode_2$	SG1‖SG2‖SG3	*Runtime*(SG2)<6 s *Runtime*(SG3)<5 s

相互独立, 互不影响. 例如, 飞机运行时存在起飞、降落等情景功能各异的运行状态, 可以自定义得到相应运行时段的系统模式. 此外, 需要规定模式中情景间的关系, 如果 2 个情景的发生不相关, 则定义为“并行”, 反之则为“顺序执行”. 如果情景的发生具有时间限制, 其约束条件应包含在模式信息中. 而对于模式切换, 明确、具体且易于检测的指令或事件均可作为模式切换的触发器. 例如, 操作员指令以及飞行高度检测值均可以作为模式切换的触发器.

4.2 验证系统架构生成

我们将系统模式切换图中的每个模式对应到一个验证子系统, 命名为 Sys-*X*, 其中 *X* 为模式名. 这样每个系统, 都可以对应多个验证子系统. 这些验证子系统构成了验证系统的整体架构. 例如, 在图 6 中系统运行存在有 2 种模式, 那么可以对应 2 个验证子系统, 分别命名为 Sys-Mode1 和 Sys-Mode2.

对每个子系统, 不仅要考虑其对应的情景、情景间关系, 还要考虑涉及的设备以及模式迁移条件是否可以达成. 因此, 我们将当前模式下所有的情景图、该模式与其他模式间存在的迁移触发器, 以及交互所使用的所有设备模型全部纳入验证子系统的范畴. 也就是说, 验证子系统中包含 3 种类型的约束集合:

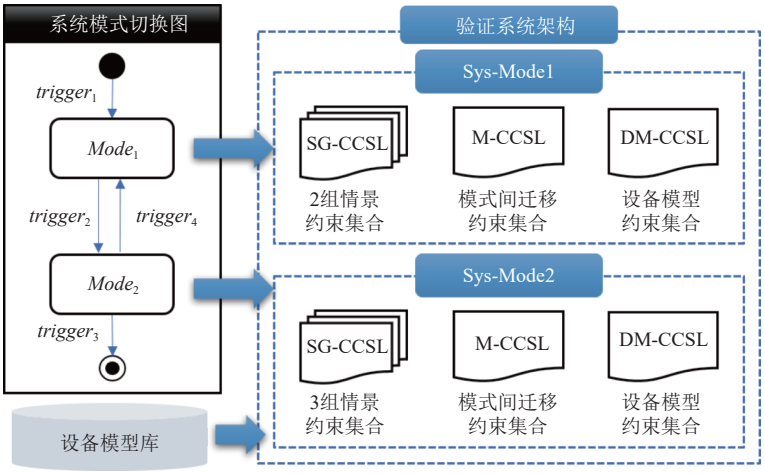


Fig. 6 Illustration of verifying system architecture generation

图 6 验证系统架构生成示意图

1)模式内情景对应的 SG-CCSL. 获取每个模式内的情景集合后，对模式下所有的情景图依次进行形式化，组成新的约束集合后加入到子系统中.

2)模式间约束对应的 M-CCSL. 系统模式切换图中模式之间的迁移会影响模式内需求的执行. 当触发条件不满足时，将不能执行模式内的需求. 每一个迁移条件都将对应到与迁移相关模式所生成的验证子系统中. 例如图 6 中，根据验证系统架构生成机制共有 3 个触发器会映射到生成的 3 个验证子系统中，即转换为模式迁移前后的条件约束等.

3)设备约束对应的 DM-CCSL. 每个模式运行依赖于一定数量的设备，这些设备可能被不同功能共享. 因此，设备的物理约束条件也需要纳入验证子系统的范围，并在后续过程中进行统一验证.

4.3 可执行验证模型生成

可执行验证模型生成包括生成每个验证子系统中的 SG-CCSL, M-CCSL, DM-CCSL.

1)SG-CCSL 的生成

SG-CCSL 的生成可以分为 2 步：单个情景图的约束转换和模式内多情景图的约束合并.

①单个情景图的约束转换. 在生成验证模型时，模式内的每一个情景图都将单独映射到一组转换约束上，它们都为 SG-CCSL 的一部分. 我们基于现有的研究工作，将单个情景图转换为对应的 CCSL 约束[38]. 其规则是对情景图中每个交互现象，将其映射到一个逻辑时钟，将每个状态 *state* 映射到 2 个时钟 *state.s* 和 *state.f*, 其中 *state.s* 对应状态的开始事件，*state.f* 对应状态的结束事件. 图 7 给出了具体的情景图到 CCSL 约束的转换规则.

情景图中行为交互之间的关系规则如图 7(a)~(d)所示，其中包括基本的顺序结构和复合结构，如选择结构 decision、分支结构 branch、合并结构 merge 等. 期望交互之间的关系转换为 CCSL 约束如图 7(e)~(g). 行为交互和期望交互之间的同步关系转换对应到图 7(h)；而行为使能关系转换对应到图 7(i).

②模式内多情景图的约束合并. 此步骤旨在依据情景图之间的关系进行约束合并. 主要有顺序关系和并行关系 2 种关系. 其中，顺序关系较为明确，

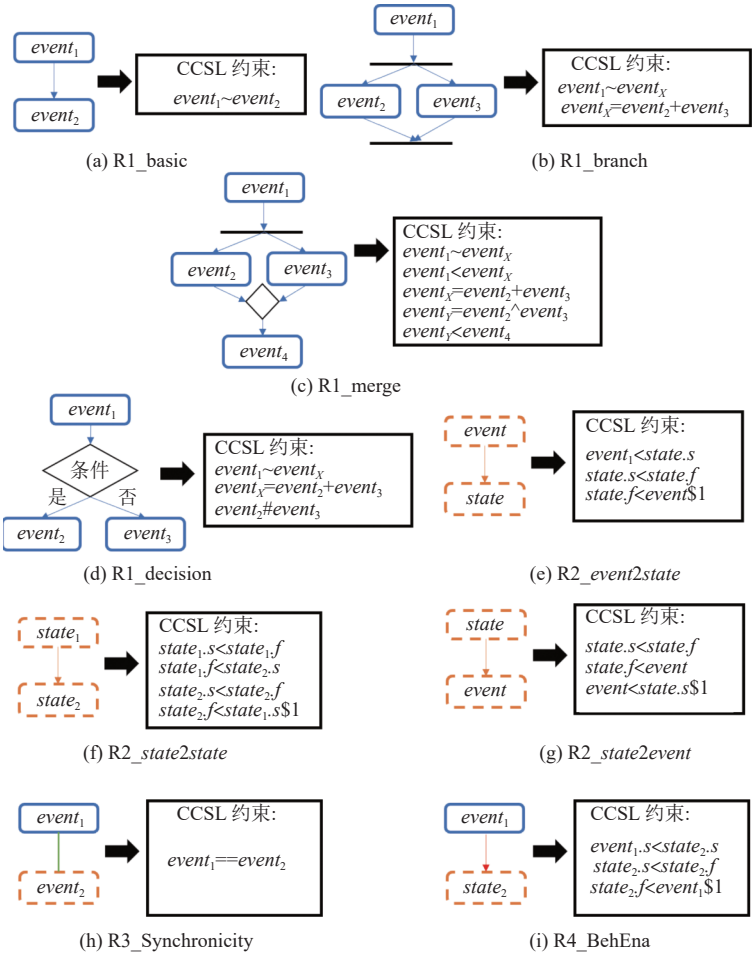


Fig. 7 Transformation of scenario graph to CCSL constraint

图 7 情景图到 CCSL 约束的转换

仅需将前一个情景图(SG1)的结束事件交替先于后一个情景图(SG2)的开始事件, 即 $E_SG_1 \sim B_SG_2$.

并行关系的处理相对复杂. 若不存在共享设备的需求, 则直接进行约束合并. 然而, 由于设备共享, 情景图之间可能存在共享现象. 如何合并这些共享现象取决于需求与其关系. 如果需求引用了某个共享现象, 则表明该现象相同, 可使用相同的名称. 如果需求约束了该现象的发生, 则每个需求都要求当前设备独立运转, 该现象在不同需求中是独立的, 因此在合并时需要对共享的交互现象名称进行处理. 我们的处理方式为在所有约束该现象的需求中, 为现象名称添加情景图名称后缀. 图 8 展示了一个合并示例.

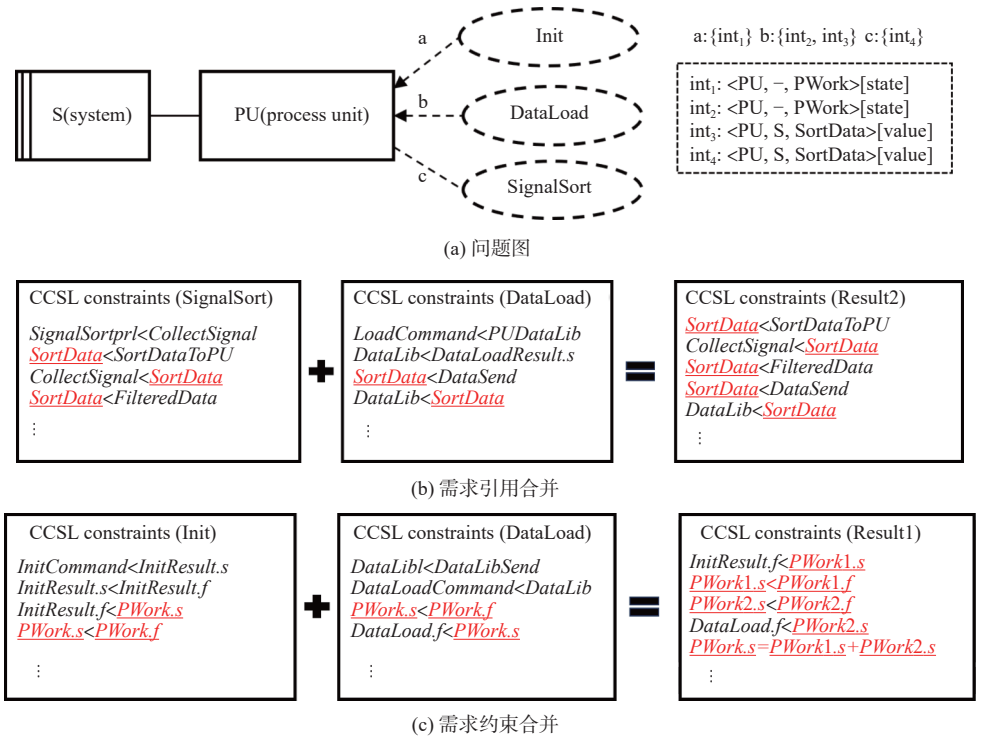


Fig. 8 Examples of scenario graphs combination

图 8 情景图的合并示例

在图 8(a)中, 需求 Init, DataLoad, SignalSort 共享设备 PU. 需求 SignalSort 引用交互 int_4(以直虚线表示), 现象名称为 SortData. 并且在需求 DataLoad 中也存在这个现象, 那么在约束集合中, 每个期望交互都会保留当前的交互现象名称, 如图 8(b)中下划线标记所示. 相对地, 需求 Init 和 DataLoad 与 int_1, int_2, int_3 的关系(带箭头虚线)为需求约束, 我们通过改名进行区分. 例如图 8(c)中所示, 2 个需求中存在共享现象状态 PWork, 于是更名后用 Union 关系 $PWork.s = PWork1.s + PWork2.s$ 和 $PWork.f = PWork1.f + PWork2.f$ 完成交互合并.

据此, 我们设计了一种算法进行情景图约束合并. 主要思想是根据情景图之间的关系对共享约束进行处理, 并最终对约束进行去重. 具体算法见算法 1, 其算法复杂度为 $O(n^2)$, n为模式中情景图的数量. 此过程中, 尽管设备库中的设备数量可以变化, 但我们假设在实际使用时, 该数量任何时刻都为一个常量.

算法 1. 模式内情景图约束合并算法.

输入: 模式内情景图集合 *MSS*, 设备库 *DS*;

输出: 合并后的约束集合 *CS*.

① $CS = \{\}$; /*初始化集合*/

② for S_i, S_j in *MSS*

③ $CCS = getConstraints(S_i, S_j)$;

/*获取 2 个情景图转换的 CCSL 约束*/

④ if($isSeq$ (S_i, S_j)) /*判断顺序关系*/

⑤ $CS \leftarrow \{“E_SG_i \sim B_SG_j”\}$;

⑥ else if($isParallel(S_i, S_j)$)/*判断并行关系*/

⑦ for each *device* in *DS*

⑧ if($sharePhe(S_i, S_j, device)$)

/*如果情景在设备上有共享现象*/

⑨ $CS \leftarrow modifyName(CCS)$;

/*修改共享现象*/

⑩ else $CS \leftarrow CCS$; /*直接添加*/

⑪ end if
⑫ end for
⑬ end if
⑭ end for
⑮ *removeRedundant*(*CS*). /*去除重复约束*/

2）M-CCSL 的生成

在每个验证子系统中，生成 M-CCSL 需要识别模式间的迁移约束并纳入验证模型. 该过程首先是获取与当前模式迁移相关的触发器集合，对集合中具有严格先后顺序的触发器添加约束. 此外，进入当前模式的迁移触发器需要先于该模式内所有情景的开始，其中并行的情景通过 Union 关系表示. 每个模式的定义中还包含时间约束，最终通过 CCSL 将这些约束条件进行表示，并整合加入到验证模型中. 例如在图 5 的系统模式切换图中，触发器 $trigger_1$ 对应进入到 $Mode_1$ 的迁移条件上，其需要先于 $Mode_1$ 中所有情景图的开始：$tri_1<B$; $B=B_SG_1+B_SG_4$. 而 $Mode_1$ 中还定义了时间约束 *Runtime*(SG4)<4 s，同样转换为 CCSL 约束 $E_SG_4-B_SG_4<4$ 加入到模型当中.

据此，设计了 M-CCSL 的生成算法，如算法 2 所示. 依次获取了模式中相关的情景图、触发器和事件约束，并加入触发器约束和模式中相关的时间约束. 其算法复杂度为 $O(n)$，n为模式内情景图的数量.

算法 2. M-CCSL 生成算法.

输入：模式切换图 *SMSD*;

输出：生成的约束集合 *M-CCSL*.

① *M-CCSL* = {}; /*初始化集合*/
② *MS* = *getModes*(*SMSD*); /*获取图中的模式*/
③ for each *Mode* in *MS*
④ *SGs* = *getSGList*(*Mode*);
/*获取模式当中所有的情景图*/
⑤ *trigger* = *getTrigger*(*Mode*);
/*获取进入模式前的迁移触发器*/
⑥ *M-CCSL* ←*addSequence*(*trigger*);
/*添加触发器之间顺序约束*/
⑦ *str* = *getSGUnion*(*SGs*);
/*获取所有情景图开始节点的并集*/
⑧ *M-CCSL* ←*addPrecedence*(*trigger, str*);
/*添加触发器先于开始节点的关系*/
⑨ *M-CCSL* ←*getTimeConstraints*(*Mode*);
/*添加模式中定义的时间约束*/
⑩ end for

3）DM-CCSL 的生成

DM-CCSL 的生成需要通过设备模型的转换实现. 首先，从生成的架构中获取当前子系统中使用到的设备模型，考虑到物理设备的约束，基于规则将设备模型转换为 CCSL 约束. 在同一时刻设备只能处于 1 种状态或者响应 1 个事件. 对设备中存在的此类天然互斥关系，我们使用 CCSL 中的“Exclude”(#)关系进行描述. 例如，在图 9 中，到达状态 On 后，相应的迁移事件分别是 PowerOnPulse 和 PowerOffPulse，则 PowerOnPulse # PowerOffPulse. 如何用 CCSL 表达状态间互斥关系可以参考文献 [27]. 而时间自动机中的状态也可能会有与时间相关的不变式，我们同样使用 CCSL 来描述这种约束. 另外在设备模型中，状态变迁的顺序也需要严格遵守设备运转的顺序.

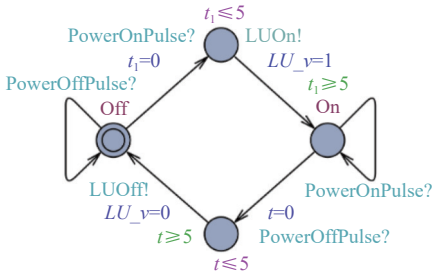


Fig. 9 Timed automata of light unit

图 9 灯光单元的时间自动机

具体而言，我们定义了从设备模型的时间自动机表达到 CCSL 约束的映射规则，如表 3 所示. 此外，每个设备模型中涉及的状态与迁移事件数目众多，需求中并不包含对所有状态和事件的使用. 因此，我们仅选择需求中涉及的相关现象及其约束来构建最终结果. 具体处理方法为：首先记录需要转换的设备时间自动机表达；再检查其中是否包含共享的交互现象名称，若不包含，则过滤掉此处表达；最终，得到所有与交互相关的设备时间自动机表达，基于映射规则转换为相应的 CCSL 约束，这些约束作为 DM-CCSL 合并加入到验证模型的约束集合中.

据此，我们设计了设备约束的生成算法，如算法 3 所示. 它根据模式获取相应的设备模型，并依次将设备模型中的事件、位置、不变量、事件顺序映射为 CCSL 约束，加入到 DM-CCSL 中. 其算法复杂度为 $O(n)$，n 为当前模式中设备模型的数量.

算法 3. DM-CCSL 生成算法.

输入：一个模式 *Mode*，设备库 *DS*;

输出：生成的约束集合 *DM-CCSL*.

① *DM-CCSL* = {}; /*初始化集合*/
② *DS′* = *getDeviceList*(*DS, Mode*);
/*获取模式当中所有的设备模型*/

③ for each *device* in *DS*′

④ *DM-CCSL*← *eventMapping*(*device*);
/*将事件映射约束加入结果*/

⑤ *DM-CCSL*← *locationMapping*(*device*);
/*将位置映射约束加入结果*/

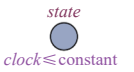
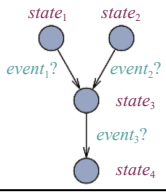
⑥ *DM-CCSL*← *invariantMapping*(*device*);
/*将不变量映射约束加入结果*/

⑦ *DM-CCSL*← *sequenceMapping*(*device*);
/*将事件顺序映射约束加入结果*/

⑧ end for

Table 3 Mapping Rules of Device Constraints

表 3 设备约束映射规则

设备约束类型	设备时间自动机表达	转换的 CCSL
事件	$event_1$, $event_2$	1) $event_1\#event_2$
位置	$state_1$, $state_2$	1) $state.s=state_1.s+state_2.s$ 2) $state.f=state_1.f+state_2.f$ 3) $state_1.s\#state_2.s$ 4) $state_1.f\#state_2.f$ 5) $state.s\sim state.f$ 6) $state_1.s\sim state_1.f$ 7) $state_2.s\sim state_2.f$
不变式		$state.f-state.s \leqslant$ constant
事件顺序		1) $U=event_1+event_2$ 2) $event_3\sim U$

4.4 不一致性验证与定位

为了验证所建立的系统模型, 首先需要定义待验证的需求一致性, 此处的一致性是指需求中对设备的调度不会发生冲突. 通过 4.1~4.3 节所述 3 步操作, 最终将机载系统需求转换为多个由 CCSL 约束集构成的验证子系统. 因此, 对机载系统需求一致性的验证实质上是对这些时钟关系的可调度性进行分析. 当在所有给定的约束条件下, 每一个时钟在任何时刻均能发生滴答, 即可认为系统需求具有一致性, 具体定义如下:

定义 1. 机载系统需求的不一致指的是转换得到的约束集不一致, 满足当且仅当 2 个条件成立: 1)对于时钟集合中的任意调度, 不存在任何一个可满足的调度符合约束集, 即存在至少一个时刻 I, 在该时刻下, 对于时钟集 T 中的任意时钟 C_i, C_i 不发生滴答, 其中 $i \in \mathbb{N}_+$; 2)对于时钟集 T, 存在至少一个时钟 C_j, 在任意时刻 I, C_j 不发生滴答, 其中 $j \in \mathbb{N}_+$.

该验证可以使用 MyCCSL 进行可调度分析. 在可调度分析中, 当找到一个满足给定约束集的调度时, 会返回“sat”结果, 说明在所有的时钟变量中至少存在 1 个时钟发生过滴答. 而如果返回“unsat”结果时, 则说明在现有的时钟约束关系中没有一个时钟可以滴答. 因此当返回结果为“unsat”时, 在机载系统的需求中发现了不一致性.

而获得不一致的验证结果后, 可以对其中的最小不一致集合进行定位, 包含的步骤为: 首先获取约束中的最小不一致约束集, 然后查找映射记录表中存在交集的部分. 工具 MARCO 可以从不满足的约束集中提取最小不满足子集(MUSes), 以定位约束集中问题的特定位置. 在验证模型生成的步骤中, 保留了由每个情景图中交互关系转换到 CCSL 约束集的关系映射表. 通过查询最小不一致集合和映射结果的交集, 可以判定交互关系是否与发现的不一致性相关. 若存在交集, 则将该交互关系加入定位结果列表中, 最终可通过可视化方式展示给参与需求建模的人员, 以便于后续理解和修改.

5 案例与评估

5.1 案例研究

本节使用一个机载系统(RS)的具体需求作为案例, 该系统由显控设备、综合控制单元、信号处理单元和采集接收单元等组成, 能够实现加电控制、信号处理和温度监控等功能. 整体运行流程如下: 操作员通过显控设备与综合控制单元进行指令交互, 对各分设备发送加电、断电或者其他功能执行命令, 并将实时状态显示到显控设备上; 采集接收单元依据系统指令实时采集信号, 信号处理单元对采集的信号进行周期性处理并传给综合控制单元. 该机载系统当中, 存在多个不同时间段下的运转模式, 而不同模式中可执行的系统功能也有所区别.

5.1.1 设备模型库

RS 系统作为目标开发系统, 共涉及 11 个设备, 其中含感知器 4 个、作用器 3 个, 既是感知器又是作用器的设备 4 个, 具体包括了综合控制单元(integrate control unit, ICU)、ICU 感知器(ICU sensor, ICUS)、采集接收单元(collect receive unit, CRU)、CRU 感知器(CRU sensor, CRUS)、信号处理单元(signal process

unit, SPU)、SPU 感知器(SPU sensor, SPUS)、干扰处理单元(jam process unit, JPU)、JPU 感知器(JPU sensor, JPUS)、显控设备(display control device, DCD)、起落架(landing gear, LG)、信号传送器(transmitters, TM)等. 对以上设备分别进行时间自动机建模, 构建设备模型库. 其中, ICU 设备的时间自动机模型如图 10 所示. 模型中包含 17 个位置, 描述了 ICU 设备的 5 个运行状态以及 18 个与状态变迁相关的信号事件.

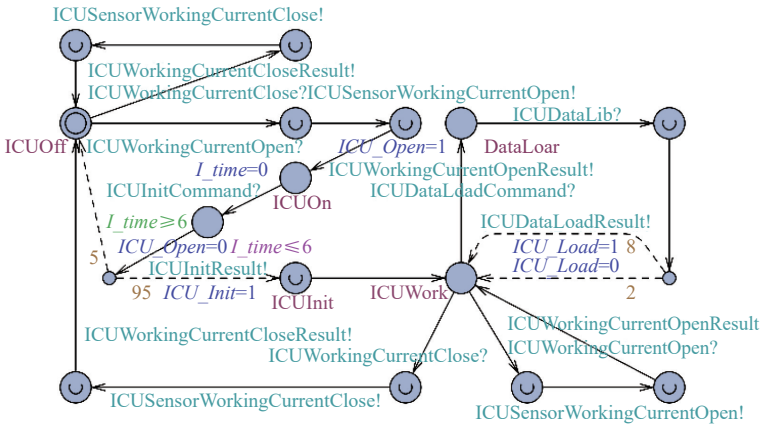


Fig. 10 ICU device model

图 10 ICU 设备模型

5.1.2 需求建模

根据 4.1 节中所介绍的需求建模过程, 我们可以对 RS 系统构建具体的需求模型, 包括问题图、情景图和系统模式切换图.

步骤 1. 构建问题图. 首先, 将待开发系统命名为“RS(reconnaissance system)”. 然后, 基于系统需要协调管理的设备和与外部环境实体的交互, 识别出 18 个问题领域, 包括 DCD, ICU, CRU, SPU 等. 接下来, 根据系统功能的描述, 识别出 19 个系统需求, 包括初始化(Init)、数据加载(DataLoad)等. 对于每个需求, 识别其作用的期望交互, 并根据期望交互推导出行为交互, 最终得到 137 个期望交互与 144 个行为交互. 具体模型元素统计如表 4 所示.

Table 4 Requirement Model Element Statistics of RS System

表 4 RS 系统需求模型元素统计

需求模型元素	数量
机器领域	1
问题领域	18
需求	19
期望交互	137
行为交互	144

由于篇幅限制, 本文仅在图 11 展示部分问题图, 完整模型请在网站[①]查看. 图 11 展示了初始化(Init)、信号分选(SignalSort)、数据加载(DataLoad)和目标上报(TargetReport)四个需求. 其中交互序号与完整问题图保持了一致, 未从 1 开始编号.

下面以需求 DataLoad 为例, 简述识别期望交互与推导行为交互的过程. 其要求“各分机单元进行数据加载”, 即出现期望交互 int_{16} (SPUDataLoad), int_{184} (ICUDataLoad), int_{192} (JPUDataLoad). 根据需求对期望交互是否存在期望性约束, 判断需求与期望交互之间为需求引用还是需求约束. 例如, 期望交互 int_{16} (SPUDataLoad) 表达了对 SPU 进入数据加载状态的约束, 所以 DataLoad 与此交互之间为需求约束. 类似地, 可识别出所有期望交互, 再以此推导行为交互. 例如根据 SPU 物理特性, 系统需要发送数据加载命令才能满足期望交互 int_{16} (SPUDataLoad), 所以可推导出行为交互 int_{73} (SPUDLCommand). 同样地, 可对需求 DataLoad 推导出所有的行为交互.

步骤 2. 构建情景图. 在问题图的基础上, 对于每一个需求, 识别交互之间的关系. 例如在数据加载需求中, 要求操作员通过显控设备发送数据加载命令(int_{62})与数据库信息(int_{63}), 系统接收显控设备下发的命令(int_{183})与信息(int_{156})后完成自身缓存(int_{15}), 即 int_{62}, int_{63}, int_{183}, int_{156}, int_{15} 之间的存在期望序关系, 且先后顺序为 $int_{62}\rightarrow int_{63}\rightarrow int_{183}\rightarrow int_{156}\rightarrow int_{15}$. 类似地, 与这些期望交互相对应的行为交互 int_{62}, int_{63}, int_{155}, int_{156}, int_{99} 之间存在行为序关系并满足相对应的先后

① http://re4cps.org:8027/?pdf=ReqModelVerForRS.pdf

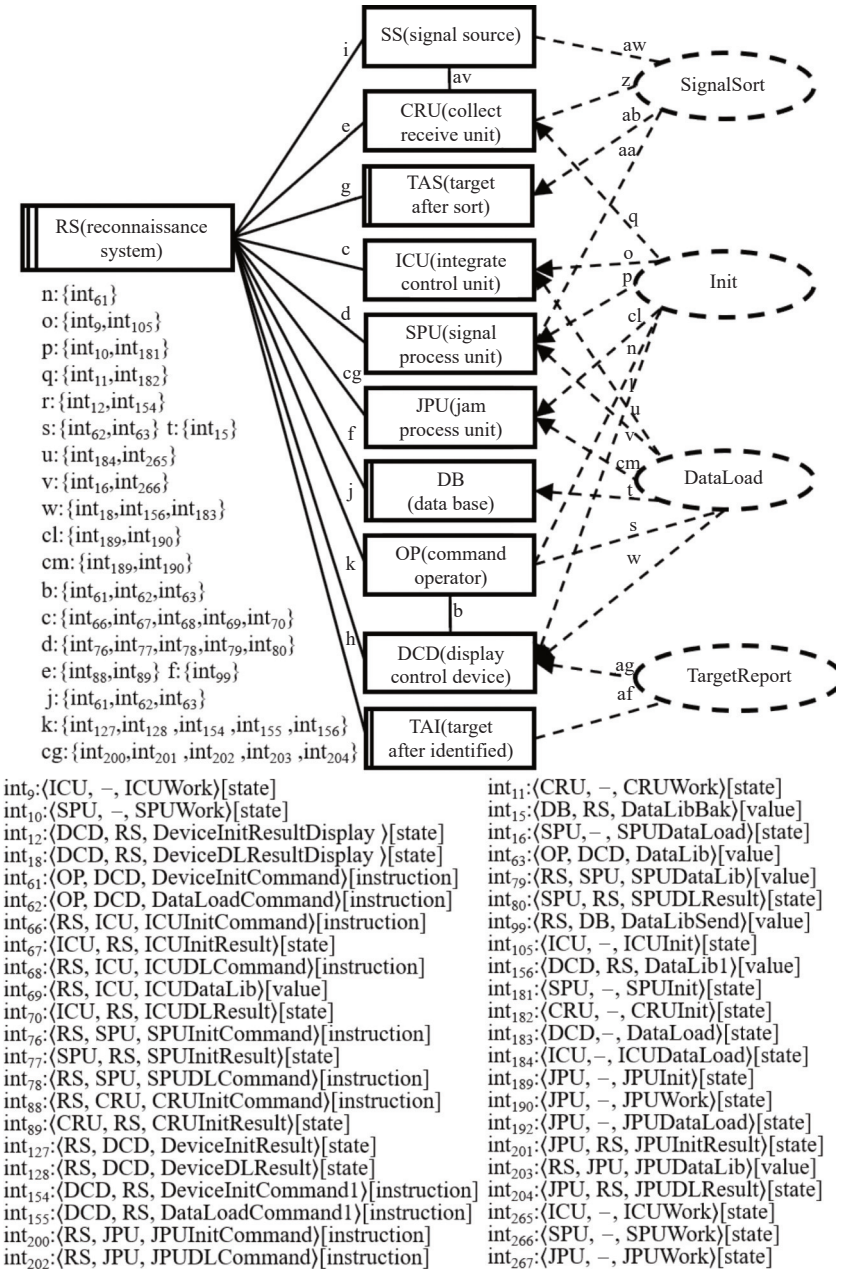


Fig. 11 Partial problem diagram of RS system

图 11 RS 系统的部分问题图

顺序. 其中, int_{15} 和 int_{99} 之间存在期望使能关系, 而其他行为交互与其相对应的期望交互之间都属于同步关系. 由于系统完成自身缓存 int_{15} 后转发命令给各分机单元 int_{184}, int_{16}, int_{192}, 故 int_{15} 通过分支节点分别与 int_{184}, int_{16}, int_{192} 之间分别存在期望序关系. 相对应地, int_{99} 也通过分支节点分别与 int_{68}, int_{78}, int_{202} 之间存在行为序关系. 其中, int_{184} 与 int_{68}、int_{16} 与 int_{78}、int_{192} 与 int_{202} 之间存在行为使能关系. 最后经过情景图检测可以获得数据加载需求的情景图, 如图 12(a)所示.

类似可以对 RS 系统中其他需求构建情景图, 获得 19 个情景图. 图 12 中仅分别展示了需求 DataLoad 的情景图和需求 Init 的情景图, 其余部分请查看网站①.

步骤 3. 绘制系统模式切换图. 首先, 根据系统的多个运转模式, 识别出初始化(Init)、工作(Working)和维护(Maintaining)共 3 个模式. 然后识别模式之间存在的迁移及其触发条件. 当系统处于初始化模式下, 若收到设备初始化结果显示命令(InitResultDisplay)时则立即进入工作模式; 当系统处于工作模式下,

① http://re4cps.org:8027/?pdf=ReqModelVerForRS.pdf

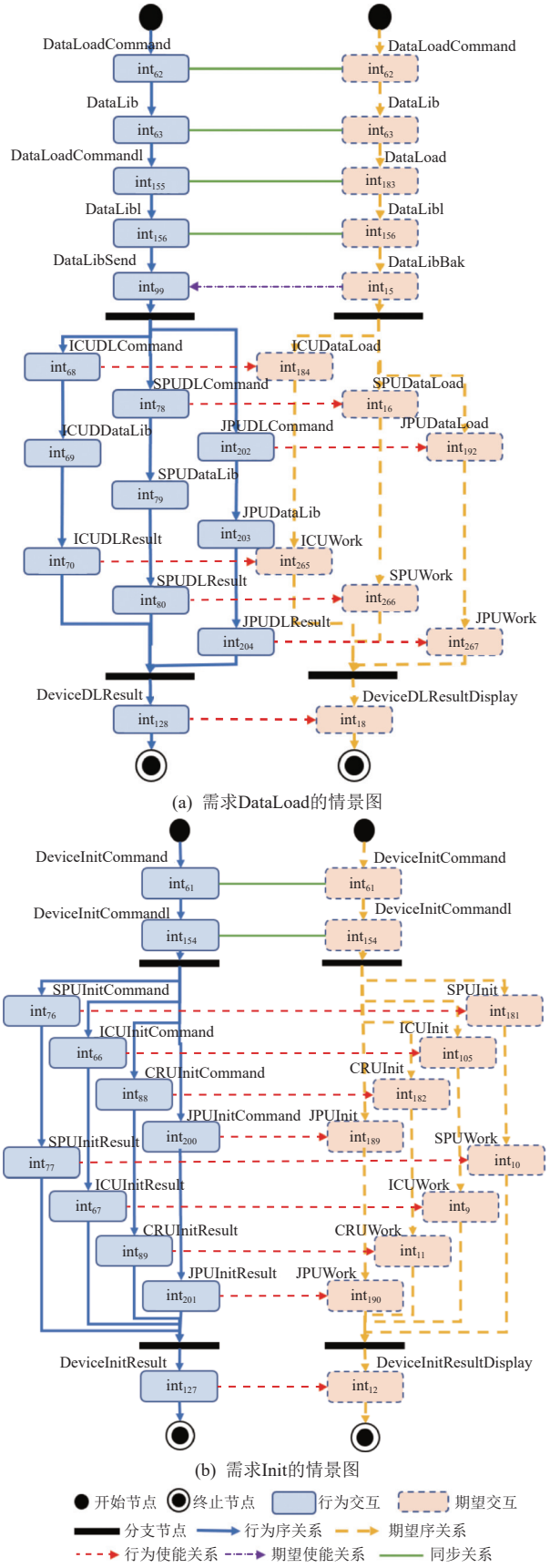


Fig. 12 Scenario graphs of requirements DataLoad and Init

图 12 需求 DataLoad 和 Init 的情景图

若收到数据加载命令(DataLoadCommand)时则立即进入维护模式；若检测到 ICU, SPU, CRU, JPU 都被断电关系(DeviceOff)则立即结束. 对于每一个模式，识别其涉及的一系列情景，以及情景之间串/并行的执行顺序与时间约束. 例如，在初始化模式下，先并行执行对设备的加/断电控制，再顺序执行软硬件初始化等任务，其描述为“(SG-ICUPowerCloseControl||SG-ICUPowerOpenControl||SG-CRUPowerOpenControl||SG-SPUPowerCloseControl||SG-CRUPowerCloseControl||SG-SPUPowerOpenControl||SG-JPUPowerCloseControl||SG-JPUPowerOpenControl);SG-WorkingCurrentControl;SG-Init”.

另外，每个模式下存在着执行的时间要求，需添加 *Runtime*(SG-Init)<5 s 等约束，最终见图 13 与表 5.

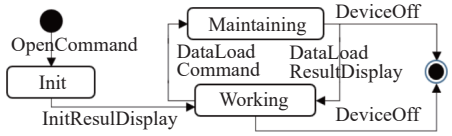


Fig. 13 System mode switching diagram of RS system

图 13 RS 系统的系统模式切换图

Table 5 Mode Definitions Involved in RS System

表 5 RS 系统中涉及的模式定义

模式	情景	时间约束
Init	(SG-ICUPowerCloseControl||SG-ICUPowerOpenControl||SG-CRUPowerOpenControl||SG-SPUPowerCloseControl||SG-CRUPowerCloseControl||SG-SPUPowerOpenControl||SG-JPUPowerCloseControl||SG-JPUPowerOpenControl);SG-WorkingCurrentControl;SG-Init	*Runtime*(SG-Init)<5 s
Working	SG-TemperatureReport||SG-OverTempProtect||((SG-ICUPowerCloseControl||SG-ICUPowerOpenControl||SG-CRUPowerOpenControl||SG-SPUPowerCloseControl||SG-CRUPowerCloseControl||SG-SPUPowerOpenControl||SG-JRUPowerCloseControl||SG-JPUPowerOpenControl);SG-WorkingCurrentControl))||SG-ReconnaissanceGuide||SG-SignalSort||SG-SignalIdentify||SG-TargetReport||SG-SpectrumSurveillance||SG-JamProcess	*Runtime*(SG-SignalSort)<5 s *Runtime*(SG-SignalIdentify)<6 s *Runtime*(SG-TargetReport)<5 s
Maintaining	SG-DataLoad	*Runtime*(SG-DataLoad)<5 s

5.1.3 验证系统结构生成

根据 4.2 节中的步骤，由于 RS 系统存在 3 个运

行模式, 整个验证系统结构对应 3 个子系统, 分别命名为 Sys-Init, Sys-Working, Sys-Maintaining. 每个子系统又由 3 组 CCSL 约束组成, 下面以 Sys-Init 为例, 说明其 CCSL 约束来源. 根据 4.2 节, 每个模式内的 CCSL 约束有 3 处来源: 1)情景图转换而来的约束集 SG-CCSL, 在模式 Init 内共有 10 个情景图, 于是对应有 SG-Init 等 10 组转换结果. 2)以该模式的触发和迁移条件形成的 CCSL 约束集 M-CCSL, 例如模式 Init 前后共有 2 个触发器 OpenCommand 和 InitResultDisplay, 对应到子系统中模式间转换的约束结果. 3)设备模型转换而来的约束集 DM-CCSL, 即该模式内情景图相关所有的设备约束, 如 Init 模式内一共涉及了 ICUSensor, SPUSensor 等 9 个设备模型, 将该集合对应到子系统下设备约束的转换结果. 类似地可以得到 Sys-Working 与 Sys-Maintaining 的构建, 形成 RS 系统的验证系统组织架构, 如图 14 所示.

5.1.4 可执行验证模型生成

根据 4.3 节中描述的转换规则, 可以为验证系统架构中的每个验证子系统生成验证模型.

第 1 步, 将模式内的每个情景图进行约束转换. 下面以模式 Working 内的情景图 SG-TargetReport 为例说明. 将情景图中的每个交互关系映射到图 7 的规则中, 并将交互现象的名称用作 CCSL 约束中的时钟名称. 此外, 为了具体区分不同的控制节点, 给其名称添加编号. SG-TargetReport 在工作状态下, 对采集接收单元上报的信息进行读取, 并形成目标信息上报给显控设备进行显示. 因此, 该功能执行的现象需要添加相应的序关系或者严格先于关系. 最后, 单个情景图的开始节点与结束节点映射的时钟需要添加 Alternate 关系, 即 B_TargetReport~E_TargetReport, 以表示单个情景图内交互的严格交替执行. 转换结果中, 对于不同的并关系节点 Union 采用了“情景图名称+编号”的命名规则进行了区分, 如 TargetReportU1. 对于子系统 SG-CCSL 中剩余情景图的形式化, 同样依照转换规则表将相应的交互关系对应到 1 组 CCSL 约束上. 而模式内其他情景图的约束转换类似上述操作, 得到的约束数量见表 6 中步骤 1.

第 2 步, 合并模式内情景图转换的约束. 根据 4.3 节, 分别对 RS 系统的 3 个模式内的所有情景图转换的 CCSL 约束集进行合并. 下面以多个需求约束同一个设备的情况进行举例说明合并过程, 如图 15 所示. 其中, 情景图 Init 和情景图 DataLoad 对于设备模型 IntegrateControlUnit 的状态 ICUWork 均有需求约束. 若模式中同时执行了这 2 个需求, 则在模式内合并时需要 2 个情景图转换后与 ICUWork 相关的 CCSL 约束进行现象名称的处理, 即在各个情景图中现象分别修改为 ICWorkInit 以及 ICUWorkDataLoad, 同时在合并的 CCSL 约束集中加入 Union 关系 ICUWork = ICWorkInit + ICUWorkDataLoad 用以表示需求约束间的相互关系. 所有子系统下约束合并情况的统计见表 6 中步骤 2.

第 3 步, 在每个验证子系统中, M-CCSL 为模式

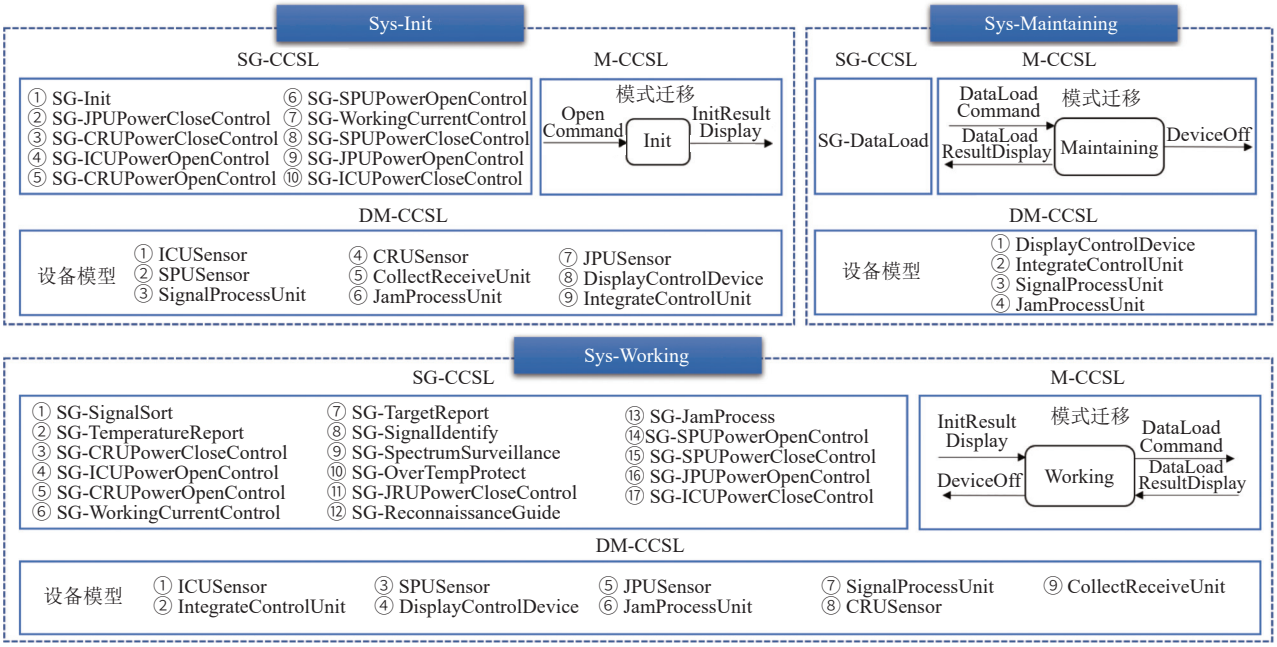


Fig. 14 Architecture of RS system verification system

图 14 RS 系统验证系统架构

Table 6 Generation Steps and Results of RS System Verification Model

表 6 RS 系统的验证模型的生成步骤与结果

步骤		RS 系统的子系统验证模型及结果		
		Sys-Init	Sys-Working	Sys-Maintaining
步骤 1.	单个情景图的约束转换	9 组 SG-CCSL 约束，共 234 条约束	17 组 SG-CCSL 约束，共 396 条约束	1 组 SG-CCSL 约束，共 50 条约束
步骤 2.	模式内约束的合并	合并 SG-CCSL 约束 330 条，增加 96 条，修改 131 条	合并的 SG-CCSL 约束 519 条，增加 123 条，修改 167 条	合并的 SG-CCSL 约束 65 条，增加 15 条，修改 27 条
步骤 3.	模式间约束的转换	M-CCSL 约束增加 2 条：① *OpenCommand<B_Init* ② *E_Init<InitResultDisplay*	M-CCSL 约束增加 3 条：① *InitResultDisplay<B_Working* ② *E_Working<DataLoadCommand* ③ *E_Working<DeviceOff*	M-CCSL 约束增加 2 条：① *DataLoadCommand<B_Maintaining* ② *E_Maintaining<DeviceOff*
	设备约束的转换	涉及 9 个设备，DM-CCSL 约束增加 194 条	涉及 11 个设备，DM-CCSL 约束增加 199 条	涉及 5 个设备，DM-CCSL 约束增加 6 条
	约束总计	527 条 CCSL 约束	722 条 CCSL 约束	74 条 CCSL 约束

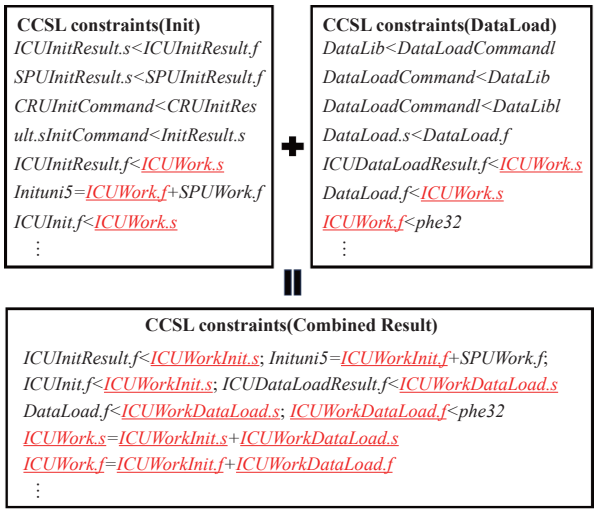


Fig. 15 CCSL constraints for combining Init and DataLoad

图 15 合并 Init 和 DataLoad 的 CCSL 约束集

间转换的验证约束，而根据系统模式切换图中描述的触发条件以及迁移条件，可以直观地获取每个模式迁移前后的约束. 最终所得的模式间验证约束，如表 6 所示. 根据模式内需求的交互现象，可以得到子系统 Sys-Init 涉及的设备 9 个、Sys-Working 涉及的设备 11 个、Sys-Maintaining 涉及的设备 5 个. 对于设备模型中的时间约束，比如在 Sys-Working 发生需求引用的设备 ICUSensor 中，则会有时间约束 $ICUSensorOff.f-ICUSensorOff.s\leqslant 1$ 会加入到 CCSL 约束集中. 所有涉及设备模型映射得到的物理设备内部与状态/事件相关约束，筛选出被需求引用和约束的部分即为子系统中 DM-CCSL 转换的结果. 以模式 Init 下的 ICU 设备为例，根据表 3 设定的规则可转换得到相应的约束集合超过 200 条，而在筛选过后被需求引用和约束的子集则有 39 条 CCSL 约束.

得到合并的约束集后，再删除冗余的 CCSL 约束集即为各个验证子系统最终的验证模型. 表 6 中详细介绍了生成验证模型的每一步中 CCSL 约束集的数量构成以及最终的约束总计等.

5.1.5 动态验证与定位

在进行验证时，本地配置为 CPU Inter® Core™ i5-9300 @2.40 GHz，运行内存为 16.0 GB RAM. 将每个验证模型中的 CCSL 约束集合提交给 MyCCSL 进行验证. 其中 *Bound* 都设置为 30，执行时间设置为 2 000 s. 其中，Sys-Init 的验证时间最长，为 22 730 ms；Sys-Maintainig 的验证时间最短，为 7 097 ms. 所有模型的总验证时长为 45 891 ms. 验证结果中 Sys-Init 和 Sys-Working 均为“sat”，Sys-maintaining 结果为“unsat”. 利用 4.4 节中提及的定位工具 MARCO，以 Sys-Maintaining 为例，我们能够定位得到一组最小的不一致集合包括“*DataLib*1<*DataLibBak*”“*JPUDataLoadCommand<DataLoadDataLoad.s*”“*DataLoadDataLoad.f<ICUWorkDataLoad.s*”等 13 条 CCSL 约束中，根据约束转换时记录的映射表可以对应到情景图 SG-DataLoad 中的 10 个交互关系. 可视化结果如图 16 所示，从中可以发现由于建模时 int_{156} 与 int_{184} 交互现象名称相同，导致最终转换结果中出现了从节点 DataLoad 开始，经由 DataLib1，DataLibSend 等节点最后回到 DataLoad 的 1 条环路，这导致了死锁的发生. 该错误与本文合作单位需求编写人员实际发现的问题一致.

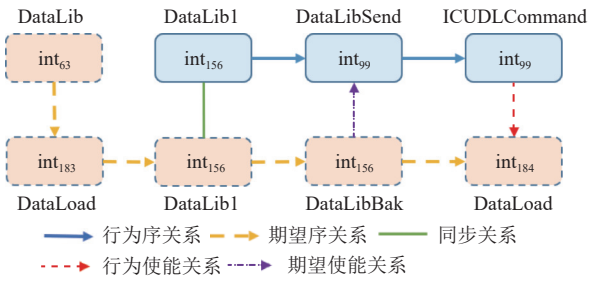


Fig. 16 Inconsistency location result of Sys-Maintaining

图 16 Sys-Maintaining 不一致性定位结果

5.2 方法评估

方法评估是为了回答2个研究问题: 1)相比于单体验证, 本文方法能否对复杂的机载系统需求进行形式化验证. 2)与单体验证方法相比, 本文方法能否避免因未考虑分时而导致的误报. 为了回答这2个问题, 我们拟设计对比实验, 通过与单体验证对比, 从“验证结果”和“误报”2组数据进行评价. 若某些复杂系统无法通过单体验证得到结果, 而本文方法能验证, 则说明本文方法具有应对复杂系统的能力, 能缓解状态空间爆炸的问题. 此外, 若单体验证存在误报(即验证结果与实际情况不符), 而本文方法不存在误报, 则进一步表明本文方法能避免由于未考虑分时特征而造成的误报.

1)实验准备. 实验数据选取了6个与机载系统需求相关的案例, 其需求个数8~32不等, 具体情况参见表7. 它们源自本文合作者实际研发中遇到的真实场景. 合作者由业内机载系统资深工程人员组成, 各自拥有3~10年不等的工作经验. 这些文档覆盖了嵌入式系统中常见的感知、计算以及控制等多种需求场景. 为了确定是否存在误报, 我们特别选取了一致与不一致的多个案例. 其中, 案例1和案例3所描述需求整体一致, 其余案例中则存在描述上的逻辑不一致. 这些案例共涉及3~20个不等的设备调用. 实验环境配置为CPU Inter® Core™ i5-9300 @2.40 GHz, 运行内存为16.0 GB RAM, 操作系统为Windows10(64).

Table 7 Datasets Used in the Experiment

表7 实验用数据集

案例编号	需求个数	涉及场景	设备数量	一致性
1	8	感知	3	一致
2	17	感知、控制	13	不一致
3	11	计算、控制	7	一致
4	19	感知、计算、控制	13	不一致
5	18	感知、计算、控制	11	不一致
6	32	感知、计算、控制	20	不一致

2)实验过程及结果. 应用本文方法与单体验证方法对上述案例进行形式化的建模与验证. 对于每组案例, 首先, 本文方法对需求进行建模, 包括系统模式切换图和情景图. 其次, 将其转化为CCSL约束, 并调用MyCCSL进行验证. 验证时设置参数的边界值为30, 执行时间为2 000 s. 最后, 记录下每个实验中的验证子系统模型个数、每个子系统的CCSL约束个数以及分别的验证结果和执行时间.

随后采取单体验证方法, 将所有生成的CCSL约束合并后直接进行验证, 调用的验证工具同样为MyCCSL. 实验过程中使用与本文方法验证时相同的参数配置. 最后记录每次验证中使用的CCSL约束数量、验证结果以及验证执行时间. 整理所得的最终结果如表8所示.

Table 8 Experimental Results of Method Evaluation

表8 方法评估实验结果

建模				本文方法验证					单体验证			
案例编号	问题图个数	情景图个数	模式个数	CCSL约束数量	验证结果	最长运行时间/ms	是否误报	效率提升/%	CCSL约束数量	验证结果	运行时间/ms	是否误报
1	1	8	2	90	模式1: sat; 模式2: sat	95 576	否		72	unsat	4 594	是
2	1	17	3	979	模式1: sat; 模式2: unsat; 模式3: unsat	7 372	否	36.37	613	unsat	11 587	否
3	1	11	3	459	模式1: sat; 模式2: sat; 模式3: sat	111 553	否	37.83	328	sat	179 425	否
4	1	19	4	1 506	模式1: sat; 模式2: unsat; 模式3: unsat; 模式4: sat	7 586	否	39.70	955	unsat	12 594	否
5	1	18	4	1 354	模式1: sat; 模式2: unsat; 模式3: sat; 模式4: sat	5 924	否	27.41	794	unsat	8 161	否
6	1	32	10	3 086	模式1: sat; 模式2: sat; 模式3: sat; 模式4: unsat; 模式5: sat; 模式6: sat; 模式7: unsat; 模式8: unsat; 模式9: unsat; 模式10: unsat	1 471 429	否		1 354	unknown		

3)实验结果分析. 从表8可以看出, 本文方法在6个案例中得到的验证结果与合作者提供的结果一致, 证明了本文方法的有效性. 值得注意的是, 案例6中单体验证方法的验证结果为“unknown”, 即未产生有效结果. 这是一个机载系统中多个功能单元共同参与的复合例子, 描述了从上电自检到超温保护的32个需求. 相较于其余案例, 案例6具有更大的规模与更广的验证空间, 单体验证方法在设置的参数执行时间内未能完成. 而本文方法通过构建多个子验证模型进行分时验证, 使得该案例在设定时间内完成验证并得到结果. 这表明本文提出方法可以对复杂机载系统需求进行形式化验证, 缓解验证过程

中的状态空间爆炸问题.另一方面,本文方法能够为每个模式的验证子系统提供验证结果.相比于单体验证的方法,本文方法能准确定位出现问题的具体模式,而不是仅给出“unsat”结果,这使得问题来源定位更加明确.

此外,本文方法能够有效减少误报.比如案例1中,2个情景图分属于不同模式,它们对于同一组读写接口的调用顺序相反.如果按照传统的单体验证,将2组转换得到的约束直接合并验证,会得到“unsat”的错误验证结果.而本文方法通过在系统模式切换图中被对情景图进行分时限制,使得它们在验证过程中被分开处理,从而避免了此类误报.

从运行时间角度来看,本文方法是根据分时进行的分组验证,组间不存在依赖关系,能够并发执行,因此运行时间等于各组中最大运行时间.对于2组实验结果均为“sat”或者“unsat”的案例,本文方法运行时间较单体验证运行更短.然而,对于存在误报的情况(如案例1),该结论并不适用.这是由验证器MyCCSL的特性决定的:在遇到“unsat”时,MyCCSL会立即终止(单体验证中检测出的不一致),而对于“sat”的情况(本文方法检测出的一致),则需要遍历所有状态空间,因此运行时间自然增加.需要指出的是,在单体验证能得到结果的情况下,若对本文方法中的分组顺序执行,会存在总运行时间(即每组运行时间之和)比单体验证时间更长的情况出现.这表明本文方法更适用于处理复杂系统,尤其是对于那些采用传统单体验证方法难以得出结果的情况.

5.3 讨 论

通过案例研究与评估,我们发现使用本文所提出的建模与组合验证方法,能够对复杂机载系统使用半形式化的建模表达安全需求,并利用系统模式切换图将其转换为多个形式化验证模型进行分时组合验证.本文方法主要从时间维度进行考虑,通过分析系统在不同时间段的行为和状态变化,进行精准的验证.一方面,本文方法利用分组缩小验证系统的验证空间,从而使得原本无法进行单体验证的系统得以验证,缓解了状态空间爆炸问题;另一方面,可以避免因不考虑分时引发的误报,从验证模型发现不一致性且定位到需求模型中帮助建模人员进行修改,体现了本文方法所具有的实用价值.

实际上,本文方法只能在一定程度上缓解状态空间爆炸问题,并不能解决这个问题.它是从时间维度提出的一种组合验证方法,为现有的状态空间爆炸问题缓解方案提供了有价值的补充和扩展.该方法能避免由于不考虑分时而导致的误报,对于其他类型的误报,并没有进行相应的检测处理.

需要注意的是,本文提出的建模方法,使用了问题图、情景图和系统模式切换图作为需求模型.问题图作为问题框架方法的主要模型,其建模理念是将软件将要交互的实体作为第1类概念进行描述,通过实体变化表达需求,尤其适用于描述机载系统等嵌入式系统的需求.在系统模式切换图的模式定义中,明确表示了情景图作为功能描述的基本单元,它不仅能描述软件与实体的期望交互,也描述了实体预期的变化.因此,本文中的模型具有特定的适用性,无法随意替换为其他需求模型.

本文所提出的验证方法具有很好的通用性,能够推广至其他形式化语言应用.CCSL作为众多与时间相关的语言之一,在本文中得到了应用,但同样的原则和技术也适用于其他相似的语言,例如时间自动机.分时验证并不依赖于具体的形式化语言或验证工具,而是可以根据实际的验证需求进行相应的调整和优化.尽管这种灵活性可能会对模型的转换过程和验证性质的描述方式带来一定的变化,但它在提高验证的精确性和效率方面具有重要的实际意义,为形式化验证领域提供了一种创新且高效的解决方案.

此外,本文的机载系统需求里明确了接入的物理设备,表示为问题图中的问题领域.在机载系统的不同运行时段,接入的物理设备数量可能有所不同.问题图还可以表示同类设备的不同实例,即设备数量.尽管本文并未详细说明问题图如何表达不同数量的设备,但如果需要增加不同数量的设备,只需将其表示为不同的约束即可.这并不影响本文方法的有效性.

6 结 论

本文针对复杂的机载系统需求,提出了一种需求建模与分时组合验证方法.该方法在问题框架方法的基础上通过增加系统模式切换图,完成对需求的分时建模.随后通过结合需求与设备模型动态生成验证系统架构,将原来复杂的机载系统需求通过多个验证子系统进行分时组合验证.该方法显著减少了机载系统需求单次形式化验证的状态空间,缓解了状态空间爆炸问题,使得一些单体方法无法验证的系统也能进行验证,并避免了不考虑分时所造成的误报.这为复杂机载系统需求形式化验证领域

提供了一种创新且高效的解决方案,是现有方案的有益补充.

在未来的研究中,我们计划将本文提出的方法应用于其他复杂的嵌入式系统,以此来证明我们方法的广泛适用性和实际应用价值.此外,我们也将探索使用其他形式化语言来替代 CCSL 进行验证,以进一步增强我们方法的通用性.

作者贡献声明:黄益柯负责完成实验并撰写论文;阮锟负责调研并撰写部分论文;陈小红提出方法设计并修改论文;金芝提出方法设计并提供论文修改意见.

参 考 文 献

[1] Wang Fei, Huang Zhiqiu, Yang Zhibin, et al. A requirements traceability approach for safety-critical embedded system[J]. Chinese Journal of Computers, 2018, 41(3): 652−669 (in Chinese)
(王飞, 黄志球, 杨志斌, 等. 一种安全攸关嵌入式系统需求追踪方法[J]. 计算机学报, 2018, 41(3): 652−669)

[2] Cao Zongyu, Lv W, Huang Wanyou, et al. Formal analysis and verification of airborne software based on do-333[J]. Electronics, 2020, 9(2): No. 327

[3] Feng Jincao, Miao Weikai, Zheng Hanyue, et al. FREPA: An automated and formal approach to requirement modeling and analysis in aircraft control domain[C]//Proc of the 28th ACM Joint Meeting on European Software Engineering Conf and Symp on the Foundations of Software Engineering. New York: ACM, 2020: 1376−1386

[4] Cleaveland R, Mitsch S, Platzer A. Formally verified next-generation airborne collision avoidance games in ACAS X[J]. ACM Transactions on Embedded Computing Systems, 2022, 22(1): 1−30

[5] RTCA. Software Considerations in Airborne Systems and Equipment Certification[S]. Washington: RTCA, 2011

[6] RTCA. Formal Methods Supplement to DO-178C and DO-278A[S]. Washington: RTCA, 2011

[7] Yang Mengfei, Gu Bin, Gu Zhenhua, et al. Intelligent program synthesis framework and key scientific problems for embedded software[J]. Chinese Space Science and Technology, 2022, 42(4): 1−7 (in Chinese)
(杨孟飞, 顾斌, 段振华, 等. 嵌入式软件智能合成框架及关键科学问题[J]. 中国空间科学技术, 2022, 42(4): 1−7)

[8] Valmari A. The state explosion problem[C]//Advanced Course on Petri Nets. Berlin: Springer, 1996: 429−528

[9] Clarke E M, Long D E, McMillan K L. Compositional model checking[C]//Proc of the 4th Annual Symp on Logic in Computer Science. Piscataway, NJ: IEEE, 1989: 353−362

[10] Grumberg O, Long D E. Model checking and modular verification[J]. ACM Transactions on Programming Languages and Systems, 1994, 16(3): 843−871

[11] Limbrée C, Cappart Q, Pecheur C, et al. Verification of railway interlocking-compositional approach with OCRA[C]//Proc of the 1st Int Conf on Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification (RSSRail 2016). Berlin: Springer, 2016: 134−149

[12] Chen Hanyue, Su Yu, Zhang Miaomiao, et al. Learning assumptions for compositional verification of timed automata[C]//Proc of the 35th Int Conf on Computer Aided Verification. Berlin: Springer, 2023: 40−61

[13] Haxthausen A E, Fantechi A. Compositional verification of railway interlocking systems[J]. Formal Aspects of Computing, 2023, 35(1): 1−46

[14] Yuan Zhengheng, Chen Xiaohong, Liu Jing, et al. Simplifying the formal verification of safety requirements in zone controllers through problem frames and constraint-based projection[J]. IEEE Transactions on Intelligent Transportation Systems, 2018, 19(11): 3517−3528

[15] Liu Xiaoshan, Yuan Zhengheng, Chen Xiaohong. Safety requirements modeling and automatic verification for zone controllers[J]. Journal of Software, 2020, 31(5): 1374−1391 (in Chinese)
(刘筱珊, 袁正恒, 陈小红. 区域控制器的安全需求建模与自动验证[J]. 软件学报, 2020, 31(5): 1374−1391)

[16] Liu Qianqian, Chen Xiaohong, Jin Zhi. Environment model based requirements consistency verification: An example[C]//Proc of the 29th IEEE Int Requirements Engineering Conf Workshops (REW). Piscataway, NJ: IEEE, 2021: 422−427

[17] Jin Zhi, Chen Xiaohong, Li Zhi, et al. Re4cps: Requirements engineering for cyber-physical systems[C]//Proc of the 27th IEEE Int Requirements Engineering Conf (RE). Piscataway, NJ: IEEE, 2019: 496−497

[18] Jin Zhi, Liu Lin, Chen Xiaohong, et al. Software Requirements Engineering Methods and Practices[M]. Beijing: Tsinghua University Press, 2023 (in Chinese)
(金芝, 刘璘, 陈小红, 等. 软件需求工程方法与实践[M]. 北京: 清华大学出版社, 2023)

[19] Jackson M. Problem Frames: Analyzing and Structuring Software Development Problems[M]. Reading, MA: Addison-Wesley, 2000

[20] Jin Zhi, Chen Xiaohong, Zowghi D. Performing projection in problem frames using scenarios[C]//Proc of the 16th Asia-Pacific Software Engineering Conf. Piscataway, NJ: IEEE, 2009: 249−256

[21] Jin Zhi. Environment Modeling-based Requirements Engineering for Software Intensive Systems[M]. San Francisco, CA: Morgan Kaufmann, 2018

[22] Heitmeyer C. Software cost reduction[J]. Encyclopedia of Software Engineering, 2002, 2: 1374−1380

[23] Xu Heng, Huang Zhiqiu, Hu Jun, et al. Modeling and verification for automatic flight system Mode requirements based on MTRDL[J]. Journal of Software, 2024, 35(9): 1−22 (in Chinese)

（徐恒，黄志球，胡军，等. 基于 MTRDL 的自动飞行系统模式需求建模与验证方法[J]. 软件学报，2024，35(9)：1−22）

[24] Gu Bin, Dong Yunwei, Wang Zhen. Formal modeling approach for aerospace embedded software[J]. Journal of Software, 2015, 26(2): 321−331 (in Chinese)
（顾斌，董云卫，王政. 面向航天嵌入式软件的形式化建模方法[J]. 软件学报，2015，26(2)：321−331）

[25] André C. Syntax and semantics of the clock constraint specification language (CCSL)[D]. Paris: French Institute for Research in Computer Science and Automation, 2009

[26] Bengtsson J, Wang Yi. Timed automata: Semantics, algorithms and tools[C]//Advanced Course on Petri Nets. Berlin: Springer, 2003: 87−124

[27] Chen Xiaohong, Zhang Juan, Jin Zhi, et al. Empowering domain experts with formal methods for consistency verification of safety requirements[J]. IEEE Transactions on Intelligent Transportation Systems, 2023, 24(12): 15146−15157

[28] Sheridan O, Monahan R, Luckcuck M. A requirements-driven methodology: Formal modelling and verification of an aircraft engine controller[C]//Proc of the 17th Int Conf on Integrated Formal Methods. Berlin: Springer, 2022: 352−356

[29] Li Junan, Hu Jun, Wang Lisong, et al. Formal modeling and safety verification for mode logic requirements of flight guidance control system[C]//Proc of the 1st Int Conf on Guidance, Navigation and Control. Berlin: Springer, 2022: 6501−6513

[30] Elqortobi M, El-Khouly W, Rahj A, et al. Verification and testing of safety-critical airborne systems: A model-based methodology[J]. Computer Science and Information Systems, 2020, 17(1): 271−292

[31] Object Management Group. Time modeling in UML profile for MARTE: Modelling and analysis of realtime and embedded systems [EB/OL]. 2024[2024-06-03]. http://www.omg.org/spec/MARTE/

[32] Mallet F. Clock constraint specification language: Specifying clock constraints with UML/MARTE[J]. Innovations in Systems and Software Engineering, 2008, 4: 309−314

[33] Mallet F, Millo J V, de Simone R. Safe CCSL specifications and marked graphs[C]//Proc of the 11th ACM/IEEE Int Conf on Formal Methods and Models for Codesign (MEMOCODE 2013). Piscataway, NJ: IEEE, 2013: 157−166

[34] Mallet F, DeAntoni J, André C, et al. The clock constraint specification language for building timed causality models: Application to synchronous data flow graphs[J]. Innovations in Systems and Software Engineering, 2010, 6(1): 99−106

[35] Liffiton M H, Malik A. Enumerating infeasibility: Finding multiple MUSes quickly[C]//Proc of the 10th Int Conf on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems(CPAIOR 2013). Berlin: Springer, 2013: 160−175

[36] Yang Xiao, Chen Xiaohong, Wang Jiangtao. A model checking based software requirements specification approach for embedded systems[C]//Proc of the 31st IEEE Int Requirements Engineering Conf Workshops (REW). Piscataway, NJ: IEEE, 2023: 184−191

[37] Chen Xiaohong, Yin Bin, Jin Zhi. An approach for capturing software requirements from interactive scenarios[J]. Chinese Journal of Computers, 2011, 34(2): 329−341 (in Chinese)
（陈小红，尹斌，金芝. 从交互情景出发的需求捕获方法及其支撑工具[J]. 计算机学报，2011，34(2)：329−341）

[38] Liu Xiaoshan. Research on inconsistency verification of realtime safety requirements based on natural language[D]. Shanghai: East China Normal University, 2021 (in Chinese)
（刘筱珊. 基于自然语言的实时安全需求不一致性验证方法研究[D]. 上海：华东师范大学，2021）



Huang Yike, born in 2001. Master candidate. His main research interest includes requirements engineering.
黄益柯，2001 年生. 硕士研究生. 主要研究方向为需求工程.



Ruan Kun, born in 2001. Master candidate. Her main research interest includes requirements engineering.
阮　锟，2001 年生. 硕士研究生. 主要研究方向为需求工程.



Chen Xiaohong, born in 1982. PhD, associate professor. Her main research interests include requirements engineering and formal method.
陈小红，1982 年生. 博士，副教授. 主要研究方向为需求工程、形式化方法.



Jin Zhi, born in 1962. PhD, professor. Her main research interests include software requirements engineering and knowledge engineering.
金　芝，1962 年生. 博士，教授. 主要研究方向为软件需求工程、知识工程.