一种安全攸关嵌入式系统需求追踪方法

王 飞[1] 黄志球[1),2)] 杨志斌[1)] 阚双龙[1)] 沈国华[1)] 陈光颖[1)]

[1)](南京航空航天大学计算机科学与技术学院 南京 211106)

[2)](软件新技术与产业化协同创新中心 南京 210093)

摘 要 嵌入式系统在航空、航天、核能及交通等安全攸关领域中的广泛应用,使得保障其安全性至关重要.需求可追踪是安全攸关领域标准的基本要求,也是安全性分析与保障的重要前提.当前可追踪性的研究主要集中在需求与代码之间,缺乏需求与设计间可追踪性的研究,而且建立的追踪信息精确性和完整性不高,无法有效地应用于安全攸关领域.针对这一问题,该文提出了一种基于谓词逻辑的需求追踪方法,可以实现嵌入式系统需求内部横向追踪关系和需求与系统设计间纵向追踪关系的自动推导与检验,并通过两种广泛使用的标准语言 SysML 和 AADL 分别对系统需求与设计进行建模.首先,定义了一个基于谓词逻辑的形式系统描述制品间的追踪信息,分别给出了 SysML 需求的横向追踪关系和纵向追踪关系的语义,然后,基于语义模型给出了追踪关系的自动推导与检验规则,用以建立精确、完整的需求追踪关系,基于这些追踪关系可以有效地支持嵌入式系统的安全性分析以及系统的维护与演化.最后,通过一个案例分析说明了该文需求追踪方法的有效性和可行性.

关键词 嵌入式系统;系统需求;设计;可追踪性;谓词逻辑;语义模型;安全性分析

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2018.00652

A Requirements Traceability Approach for Safety-Critical Embedded System

WANG Fei[1)] HUANG Zhi-Qiu[1),2)] YANG Zhi-Bin[1)] KAN Shuang-Long[1)]

SHEN Guo-Hua[1)] CHEN Guang-Ying[1)]

[1)](*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 211106)

[2)](*Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing* 210093)

Abstract Embedded systems have been widely used in safety-critical areas, such as aeronautics, astronautics, nuclear energy, and transportation, and it is of extremely significance to ensure the safety of embedded systems. Requirement traceability is one of the essential requirements in most criteria of safety-critical area, and it is an important prerequisite of safety analysis and guarantee. Existing work on traceability mainly focuses on the traceability between requirements and source code. However, there is a lack of research about traceability between requirements and design. In addition, the traceability information built by existing techniques such as the recovery of trace relations based on information retrieval (IR) is lack of accuracy and completeness. Therefore, it cannot be effectively applied to the development of safety-critical systems. To solve these problems, this paper proposes a requirement traceability method based on the predicate logic. This method can automatically infer the implicit horizontal trace links between requirements and vertical trace links between requirements and design. Moreover, the method also provides a technique

收稿日期:2016-09-09;在线出版日期:2017-09-08.本课题得到国家"八六三"高技术研究发展计划项目基金(2015AA105303)、GF 基础科研重点项目(JCKY2016203B011)、国家自然科学基金(61502231)资助.**王 飞**,男,1990 年生,博士研究生,中国计算机学会(CCF)学生会员,主要研究方向为软件工程、需求工程、形式化方法和嵌入式软件建模与分析.E-mail:wangfei2014@nuaa.edu.cn.**黄志球**(通信作者),男,1965 年生,博士,教授,中国计算机学会(CCF)理事,主要研究领域为软件工程、形式化方法和云计算.E-mail:zqhuang@nuaa.edu.cn.**杨志斌**,男,1982 年生,博士,副教授,主要研究方向为形式化方法和实时系统.**阚双龙**,男,1988 年生,博士研究生,主要研究方向为软件工程、模型检测和形式化方法.**沈国华**,男,1976 年生,博士,副教授,主要研究方向为需求工程、软件可信评估.**陈光颖**,女,1991 年生,硕士,主要研究方向为软件工程和形式化方法.

to check the consistency of these trace links. In order to model requirements and design for embedded system, two widely-used standard languages: Systems Modeling Language (SysML) and Architecture Analysis & Design Language (AADL) are exploited to specify system requirements and its design respectively. Firstly, we give the requirements traceability information model used in our approach, and definition of the semantics of horizontal traceability and vertical traceability based on this model. More precisely, we define the specific relationships of horizontal traceability for SysML requirements, such as *decompose*, *deriveReqt*, *refine* relationships, as well as the specific relationships of vertical traceability between SysML requirements and AADL components. Secondly, we design a predicate logical system based on the requirements traceability information model to formalize the traceability information between artifacts, and then give the semantics for both horizontal and vertical traceability relationships, as well as the inference and checking rules of the model. In order to infer the implicit relations and check the consistency of SysML requirements, our approach transforms the horizontal traceability relationships of SysML requirements to equivalent formulas relations by using the semantics model of traceability. At the same time, based on the semantics of traces, constrains for the relationships of vertical traceability between SysML requirements and AADL components are given, which support partially inference and checking for these relationships. All rules defined above can establish accurate and complete trace links between SysML requirements and AADL model. The traceability information can be checked if the system design gives a complete and correct expression of the requirements, and this information gives the further support for safety analysis, maintenance and evolution management of embedded system. Finally, an evaluation of the proposed requirements traceability method based on predicate logic is demonstrated through a case study of a control system of flaps and slats (CSFS). The case study shows the effectiveness and feasibility of our approach by inferring the implicit horizontal traceability relationships and check the consistency of CSFS requirements. It can also present vertical traceability relationships between AADL components and CSFS requirements.

Keywords　embedded system; system requirement; design; traceability; predicate logic; semantic model; safety analysis

1　引　言

嵌入式系统在航空航天、核能及交通等安全攸关领域的广泛应用，使得系统的失效可能造成财产的损失、环境的破坏甚至人员的伤亡，因此，保障嵌入式系统的安全性已成为安全攸关领域的研究热点[1]. 能力成熟度集成模型(Capability Maturity Model Integration，CMMI)[2]明确地指出在开发过程中维护需求的可追踪性及制品之间的可追踪性是保障系统安全的重要前提[3].

可追踪性是指在系统开发过程中建立和维护制品之间的关联关系，并利用这些关系对软件项目进行一系列分析的能力①. ANSI/IEEE 标准 830-1984 中首次提出了可追踪性的概念[4]，其早期主要被用于在功能上提高系统的可重用性. 但随着系统复杂程度的不断提高，可追踪性可以揭示隐含关联，反映系统演化的特征引发了越来越多的关注，目前已成为提升软件和系统质量的重要方法之一，被广泛应用于软件组件测试、演化管理、系统分析和软件审查等多个研究领域[5].

在安全攸关嵌入式系统中，很多重大的失效都是由于系统需求在设计过程中没有被精确和完整地追踪而引起的[6-8]. 例如，2009 年法国航空公司 A330-200 飞机由于测速仪结冰，给出飞控软件错误的攀升指令，而系统中未设置高度值上限，最终导致飞机坠毁[9]. 同年，一架土耳其航空公司波音 737-800

① CoEST: Center of excellence for software traceability [OL], [2016-8-25]. http://www.CoEST.org

型飞机在机场跑道进近期间,由于无线电高度值错误使得飞机失速而机载系统缺乏"处理失速情况"的程序,导致飞机坠毁[10].尽管不可追踪不一定会产生安全性事故,但追踪信息能够有效地实现不同制品之间的关联,基于需求追踪信息可以有效地检验需求在设计中的满足性,从而降低因设计人员对需求的误解与遗漏而产生的安全性事故发生率.

几乎所有安全攸关领域的标准都对可追踪性作出了明确要求.例如,美国无线电委员会发布的机载软件适航认证标准 DO-178C[11] 指出,对于 A 级软件(即软件失效会对系统安全造成灾难性影响的软件)需保证其生命周期各个阶段制品间的可追踪性,如高层系统需求和低层具体需求间、低层具体需求和源代码间等.图 1 给出了 DO-333(DO-178C 的形式化方法补充标准)[12] 中 A 级软件验证过程对可追踪性的要求.汽车电子电气系统功能安全标准 ISO26262[13] 指出开发人员必须保证安全需求到其对应源需求或派生需求、设计实现及验证规约的可追踪性.然而,此类标准并没有对实际项目中维持可追踪性的过程、方法或技术等作出详细说明,在工业实践中如何维持制品间的可追踪性仍然缺乏明确的指导.

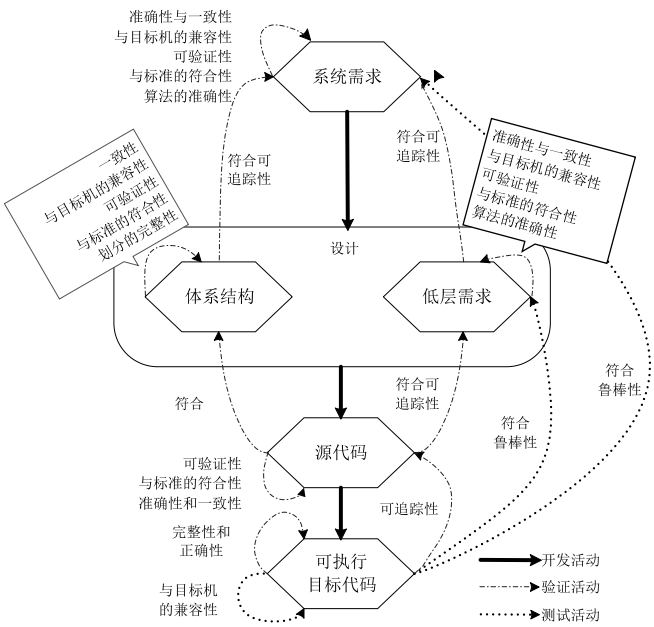

图 1 DO-333 规定的 A 级软件验证过程

工业界通常借助表格、文档及需求管理工具如 DOORS① 等,手动创建和维护制品间的追踪关系[14].此类方法成本高、耗时长、易出错且维护困难,往往无法提供有效的追踪信息以支持各类分析验证活动,可追踪性在保障系统安全、提高软件质量等方面并没有发挥预期作用,可追踪性在工业实践中的应用仍然面临诸多挑战.此外,当前嵌入式系统呈现规模大、复杂度高等特点,以航空领域嵌入式系统发展为例,从第二代飞机起,通过软件实现的功能随着每一代飞机的产生而翻番.对于第三代和第四代飞机来说,软件已经成为飞行控制、通信导航、火力控制以及维修保障的核心(如军机 F-22 飞机上嵌入式系统代码高达 300 万行,F-35 机载和地面的嵌入式系统代码高达 1500 万行[1])手动创建与维护追踪关系不可避免地存在遗漏或偏差.基于此,本文提出一种基于谓词逻辑的需求追踪方法,该方法可以实现对制品间隐含追踪关系的自动推导及关系语义一致性的自动检验,保证追踪关系的精确与完整.此外,本文将分别通过系统建模语言 SysML(Systems Modeling Language)[15] 和嵌入式系统体系结构分析与设计语言 AADL(Architecture Analysis & Design Language)[16] 对嵌入式系统需求和设计进行建模,SysML 和 AADL 是两种被广泛使用的标准建模语言,且都具有表达软硬件特性的能力,因此特别适合用于嵌入式系统建模.

本文第 2 节介绍本文相关的国内外研究工作;第 3 节在简单概述 SysML 和 AADL 的基础上提出本文的需求可追踪信息模型;然后在第 4 节介绍谓词逻辑系统的语法与语义;第 5 节则详细论述基于谓词逻辑系统的需求追踪方法;第 6 节给出相应的案例分析;最后在第 7 节总结全文并对未来工作进行展望.

2 相关工作

当前,学术界对可追踪性的研究主要包括广义的可追踪性研究和面向特定任务的可追踪性研究,或称为任务驱动(Task-Driven)的可追踪性研究[17].

广义的可追踪性研究主要针对可追踪性自身进行研究,主要集中在追踪关系的自动建立方法研究,包括基于信息检索(Information Retrieval,IR)[18] 的追踪关系自动恢复和基于规则的推导等[19].

基于信息检索的关系恢复利用 IR 模型计算需求文本与其他制品的相似度,并设定阈值筛选得到与需求关联的制品[20],可用于对缺失的、不可靠或过时的追踪关系进行恢复或重建,如遗产代码或代码与设计及需求间的追踪关系.已有大量研究工作

① Teleologic, Teleologic DOORS. http://www.teleologic.com/products/doors

使用基于 IR 的方法恢复源代码与设计或者源代码与文本文档间的追踪关系[20-21]. 基于规则的关系推导通过分析需求或其他制品的结构或语法特性，制定对应推理规则生成追踪关系. 其中，基于结构的方法利用制品的结构属性在已有追踪关系的基础上进行推导. 例如，文献[22-23]提出了一种基于规则的场景驱动的方法来支持场景、代码、用例图间追踪关系的生成和确认，并根据需求对应代码行的重叠情况检测需求间的依赖关系. 文献[24]利用自然语言文本的语法特性，分析需求文档和用例的语法结构，标注句子成分，根据制定的规则进行关系匹配. 文献[25]使用基于规则的推导方法以支持代理系统的可追踪性研究，通过检验同义词来自动生成设计模型与代码规范间的追踪关系.

上述追踪关系的建立方法均依赖于对需求和其他软件制品的静态分析，在需求或制品发生变更后，需要重新进行计算或匹配以更新追踪关系，不利于追踪关系的自动维护. 对此，文献[26]进行了基于事件通知机制的可追踪性研究，使用发布/订阅模式描述需求制品与软件开发过程中其他制品的关联关系，对需求的变更情况进行了分类，讨论了软件演化过程中追踪关系的更新和维护方法. 文献[27]利用 UML 模型的结构特性，提出了面向软件制品变更的追踪关系自动更新与维护方法，但该方法不支持追踪关系的生成.

此外，在模型驱动的软件工程(Model Driven Engineering，MDE)中，模型转换、代码自动生成等技术可以隐式地自动建立和维护软件设计和代码之间的追踪关系. 如文献[28]研究了不同抽象层次的设计模型间追踪关系的自动生成方法，文献[29]通过需求和用例图到软件体系结构元素的生成方法自动记录并维持追踪关系.

虽然基于信息检索和规则等方法建立追踪关系可以有效减少手工建立成本，但追踪信息的精确性和完整性不高，且多为事后(After-The-Fact)追踪关系的恢复. 而安全攸关软件要求在开发过程早期就严格建立和维护追踪关系，并要保证较高的精确性和完整性. 因此，上述方法无法有效地应用到安全攸关领域的软件开发中.

广义的可追踪性研究没有针对特定的软件工程任务进行考虑和分析，因此，学术界针对特定软件工程任务开展了任务驱动的可追踪性研究. 为了有效构建面向特定任务的可追踪信息，必须先从元模型的角度进行计划分析. 可追踪信息模型(Traceability Information Model，TIM)，抽象定义了项目中的追踪链类型(Trace Link Type)和被追踪软件制品类型(Artifact Type)等，为实际项目中追踪关系的建立提供指导[17]. 文献[30]研究了使用 UML 建模元素描述可追踪信息模型的方法.

在安全攸关领域，许多标准均为文本形式，缺乏清晰的结构形式，无法为实际项目中追踪关系的建立提供明确指导，许多安全攸关项目都无法满足标准规定的可追踪性要求[31]. 对此，文献[32]讨论了如何针对安全攸关的软件项目建立有效的可追踪信息模型. 文献[33]对适航认证标准 DO-178B/C 进行分析，提取出需求、事件和反应等概念，建立了一个安全相关的概念元模型，该模型以抽象描述概念间的关联关系，促进软件开发人员对标准的理解，用于支持安全认证. 文献[34]基于电气电子领域的安全标准 IEC61508[35] 提出了一个概念模型，描述了领域标准、危害、需求、制品和软件开发过程等概念间详细的追踪关系，可用于指导软件生命周期各阶段可追踪信息的无缝证据收集. 文献[36]提出了一种安全性需求分解模式，支持对安全性需求的分解并使其能够追踪到体系结构和失效传播模型，然而作者并未考虑具体的追踪关系构建与检验方法.

面向具体任务的可追踪性研究方面已经取得了部分进展，如文献[37]面向高保证的安全计算系统，定义了基于谓词演算的形式框架描述系统组件间的追踪关系，给出了组件间隐含依赖关系的推导规则，防止因为追踪关系的缺失而造成的软件失效. 文献[38]基于模态逻辑描述软件功能需求间的冲突、协作、平衡等类型的追踪关系，针对非确定性需求进行权衡分析. 文献[39-40]提出使用一阶逻辑(First-Order Logic，FOL)描述需求间常见的追踪关系类型，研究如何基于精确的关系语义进行需求变更影响分析. 文献[31]提出使用 F-Logic(Frame Logic)来对安全攸关领域标准中的可追踪性要求进行建模，如制品类型、制品间的追踪路径等，构成可追踪模型，并将实际项目建立的可追踪信息与模型进行一致性评估，以检测冗余或丢失的可追踪路径. 此外，文献[41-42]利用 SysML 建模元素进行可追踪信息的管理，提出了支持认证的与安全需求相关设计切片的自动提取算法. 文献[43]给出一种支持模型检验的可追踪性技术研究，通过建立可追踪性信息，为模型检验技术提供更多的约简信息.

尽管针对安全攸关领域的需求追踪已有一系列研究，但是大多研究集中在理论层面且较多关注软

件设计阶段,仅仅论述了在软件生命周期中需要建立与维护制品的可追踪性以及其重要性,但对具体如何建立完整、精确的系统需求追踪关系的研究却寥寥无几.

3 需求可追踪信息模型

本文针对安全攸关领域嵌入式系统需求与设计之间的追踪关系展开研究,并分别通过 SysML 和 AADL 对嵌入式系统需求与设计进行建模.本节将分别对两种标准建模语言 SysML 和 AADL 进行简单介绍,然后在此基础上,给出了本文研究的基础——需求可追踪信息模型,并给出了横向追踪(Horizontal Traceability)[44] 和纵向追踪(Vertical Traceability)[44] 的定义.

3.1 SysML&AADL

SysML(Systems Modeling Language)是国际系统工程学会 INCOSE(International Council on Systems Engineering)和对象管理组织 OMG(Object Management Group)为了满足系统工程的实际需要,从而在 UML2.0(Unified Modeling Language)的子集进行重用和扩展的基础上提出的一种建模语言.SysML 弥补了 UML 在系统工程领域建模的缺陷,已经成为工业界在系统工程设计与分析过程中的标准建模语言.SysML 是一种多用途的标准建模语言,能够支持各种复杂系统的详细说明、分析、设计、验证和确认,这些系统可能包括硬件、软件、信息、过程、人员和设备等,因此,SysML 比较适合应用于现代复杂嵌入式系统领域中的设计与分析研究,并已经被学术界和工业界所广为接受.

需求图是一种新的 SysML 视图,能够描述需求和需求之间以及需求和其他建模元素之间的关系.需求是指系统必须满足的能力或条件,一个需求能够分解成多个子需求.SysML 用《Requirement》说明需求,它也是一个类,有两个属性:text 和 id,前者是需求的文本描述,后者是需求的标识符.用户可以定义需求的子类,如操作需求、功能需求、接口需求、性能需求等等.在 SysML 需求图中,除了分解(*decompose*)关系,还有导出(*deriveReqt*)、精化(*refine*)等关系来刻画需求之间以及需求与其它模型元素之间的关系.本文仅仅考虑利用 SysML 需求图进行嵌入式系统需求建模,并不涉及其他 SysML 视图,因此,本文中的 SysML 需求关系将限定为分解、导出和精化三种.图 2 给出了一个 SysML 需求图的示例.

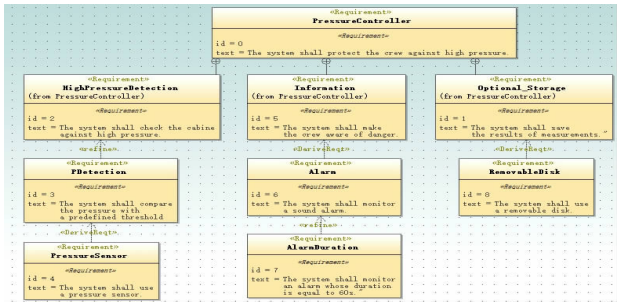

图 2 SysML 需求图示例

2004 年,美国机动工程师协会 SAE(Society of Automotive Engineers)在 MetaH[45]、UML、ADL(Advanced Distributed Learning)、HOOD(Hierarchical Object-Oriented Design)[46] 的基础上,提出嵌入式系统体系结构分析与设计语言 AADL,并发布为 SAE AS5506 标准.目的是提供一种标准而又足够精确的方式,设计与分析嵌入式系统的软、硬件体系结构及功能与非功能性质,采用单一模型支持多种分析的方式,将系统设计、分析、验证、自动代码生成等关键环节融合于统一框架之下.AADL 具有语法简单、功能强大、可扩展的优点.由于具有广阔的应用前景,AADL 得到了欧美工业界,特别是航空航天领域(如 Airbus、Lockheed Martin、Rockwell Collins、Honeywell、Boeing)的支持.

嵌入式系统是应用软件、运行时环境以及硬件平台深度融合的复杂系统,而 AADL 语言与之对应地提供了软件体系结构、运行时环境以及硬件体系结构的建模概念[47].具体而言,AADL 的建模过程如下:

(1)通过线程(Thread)、线程组(Thread Group)、进程(Process)、数据(Data)、子程序(Subprogram)构件以及连接来描述系统的软件体系结构;

(2)通过处理器(Processor)、虚拟处理器(Virtual Processor)、存储器(Memory)、外设(Device)、总线(Bus)、虚拟总线(Virtual Bus)构件以及连接来描述系统的硬件体系结构;

(3)通过分发协议(Dispatch)、通信机制(Communication)、调度策略(Scheduling)、模式变换协议(Mode Change)以及分区(Partition)等属性来描述系统的运行时环境,在 AADL 语言中称为执行模型(Execution Model);

(4)最后,通过系统(System)构件进行组合,层次化地建立系统的体系结构模型.

另外,行为附件(Behavior Annex)[48] 以变迁系统(Transition System)的形式增强了 AADL 对线程构件和子程序构件功能行为的详细描述能力,而且行为附件与执行模型有着紧密的联系,即执行

模型定义了线程和子程序周期性(非周期性或偶发)地读取、计算和发送数据，行为附件则是对计算状态内的执行行为进行详细刻画. 因此，软/硬件体系结构、执行模型以及行为附件可以构成一个完整的 AADL 描述.

当然，AADL 还包括其它方面的扩展，进一步丰富了 AADL 语言的表达能力. 例如，故障模型附件(Error Model Annex)[49]扩展了构件和连接的故障事件、故障概率等非功能属性，以支持系统可靠性分析.

综上所述，由软件构件和硬件构件组成的体系结构以及执行模型及相应的扩展附件，基本上可以构成一个较为完整的 AADL 描述，如图 3 所示.

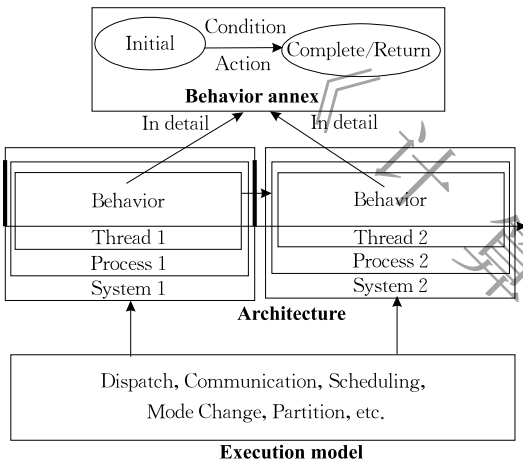

图 3　AADL 基本建模概念

3.2　需求可追踪信息模型

针对嵌入式系统需求和设计之间的可追踪性问题，图 4 给出了本文的需求可追踪信息模型，该模型主要包括 SysML 需求图和 AADL 模型两类制品，涉及 SysML 需求内部的横向追踪关系以及 SysML 需求与 AADL 构件之间的纵向追踪关系. 模型中各元素的具体定义如下：

横向追踪指的是对处于相同的项目开发阶段或拥有相同抽象层次制品间的追踪. 本文中横向追踪关系主要通过 SysML 需求图中的 *decompose*、*deriveReqt* 和 *refine* 三种关系刻画. 下面分别给出这三种关系的定义：

(1) 分解关系(*decompose*)：描述了一种局部和整体的关系，一个复杂的需求可以被分解为若干简单需求，如需求 R_2、R_3 可组成复合需求 R_1，则称 R_2、R_3 分解了需求 R_1；

(2) 导出关系(*deriveReqt*)：若需求 R_1 是由需求 R_2 导出的，则需求 R_2 是需求 R_1 实现的基础和前提，需求 R_2 的变更会对需求 R_1 造成影响；

(3) 精化关系(*refine*)：精化表示一种细化或具体化，描述了不同详细程度需求间的追踪关系. 若需求 R_1 是对需求 R_2 的精化，则需求 R_1 派生于需求 R_2，并在 R_2 的基础上补充了更多的实现细节.

纵向追踪性则描述了处于不同阶段拥有不同抽象层次制品间的追踪. 本文中的纵向追踪关系主要通过可满足关系(*Satisfy*)来刻画.

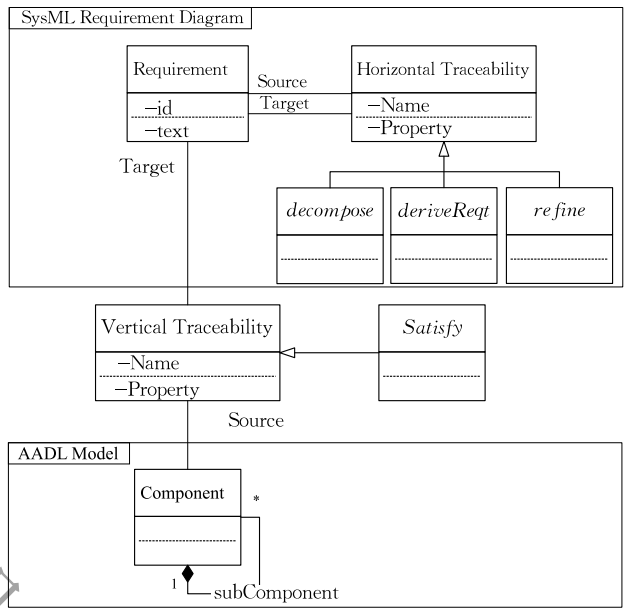

图 4　需求可追踪信息模型

可满足关系(*Satisfy*)：可满足关系描述了系统设计元素与系统需求间的关联，如 AADL 模型中的构件与 SysML 需求间的关联关系. 如 AADL 模型中的构件 C_1 可满足需求 R_1 表明该设计元素 C_1 与需求 R_1 相关，参与了需求的实现. 一个需求往往与多个设计元素相关联，与需求相关的各设计元素共同作用，实现或满足了该需求.

基于该需求可追踪信息模型，本文的主要工作如下：

(1) 面向需求可追踪信息模型：提出了一个基于谓词逻辑的形式系统用来抽象地描述制品间的追踪信息，并给出 SysML 需求间横向追踪关系以及 SysML 需求与 AADL 构件之间纵向追踪关系的形式化语义；

(2) 基于需求横向追踪关系语义：研究了需求间横向追踪关系的自动推导与一致性检验方法，并结合需求的纵向追踪关系语义，给出了需求与 AADL 构件间纵向追踪关系的约束规则，基于该约束规则，可以实现纵向追踪关系的推导与检验；

(3) 以襟缝翼控制系统为例，分析说明本文所提方法的有效性和合理性.

4 基于谓词逻辑的形式系统

谓词逻辑[50]是对命题逻辑的扩充,在其基础上引入了个体词、谓词、量词及函数符号.其中,个体词表示研究对象中可以独立存在的具体或抽象个体,个体的取值范围称为个体域或论域;谓词用来刻画个体的行为属性或个体间的相互关系;量词表示个体的数量属性;函数符号的引入则为命题的符号化带来方便.安全攸关领域为了能够描述嵌入式系统需求以及需求的可追踪关系,需要建立严格的谓词逻辑演算系统,因此,本文首先基于谓词逻辑定义形式系统 Π_{Theory} 来抽象地描述软件制品间的追踪信息.下面将分别定义 Π_{Theory} 的语法和语义.

4.1 谓词逻辑形式系统 Π_{Theory} 的语法

在定义 Π_{Theory} 的语法与语义之前,首先给出 Π_{Theory} 语言 Σ 的定义.

定义 1. Π_{Theory} 的形式语言 Σ.Σ 语言是谓词逻辑语言的子集,由下列有限的基本符号组成:

(1)个体变元:表示抽象或泛指的制品或制品属性,如:$s_1,s_2,\cdots,c_1,c_2$;

(2)二元谓词符号:用于描述个体变元间的关系,如:$P_1^2,P_2^2,\cdots,P_n^2$;

(3)联接词符号:$\neg$;

(4)括号与逗号:(),,.

若定义集合 S:描述制品标识符的有限集合;集合 C:描述制品属性的有限集合;集合 P:描述有限二元谓词集合;集合 K:包含一元否定联接符的集合;集合 E:括号与逗号的集合,则形式语言 Σ 的字母集 $\Lambda=\{S\cup C\cup P\cup K\cup E\}$.

Π_{Theory} 中的合式公式由字母集 Λ 及一系列公式生成规则产生.

定义 2. 语言 Σ 中合式公式 ω 的生成规则:

(1)$\omega=P_t^2(\sigma_i,\sigma_j)\in\Sigma$,其中$(\sigma_i,\sigma_j\in S\cup C)$且 $P_t^2\in P$;

(2)$\omega=\neg P_t^2(\sigma_i,\sigma_j)\in\Sigma$,其中$(\sigma_i,\sigma_j\in S\cup C)$且 $P_t^2\in P$;

(3)公式 ω 只能通过规则(1)、(2)产生.

上述公式的生成规则不存在对公式的递归定义,即 Σ 语言中的合式公式均为一个二元关系谓词或其否定,不存在其他形式的合式公式.

4.2 谓词逻辑形式系统 Π_{Theory} 的语义

符号化的公式并没有实际的含义,需要结合一个具体的语义模型对公式中的变元符号或谓词符号进行解释.因此,本节基于 3.1 节的需求可追踪信息模型,给出 Π_{Theory} 的语义.

对于一个给定的项目,定义集合 S':表示项目中可唯一标识的有限制品集合,如需求、设计元素等;定义集合 C':表示有限的制品属性集合,如制品的关键度等级、抽象层次等;定义集合 R:表示实际项目中有限的关系集合,如制品间的关联关系,制品与其抽象层次的对应关系等.则 Π_{Theory} 的语义模型 Π_{Model} 由如下几个部分组成:

定义 3. 形式系统语义 $\Pi_{\text{Model}}=\langle D,f_s,f_c,f_p\rangle$.

其中:$D=S'\cup C'$;

$f_s:S\to S'$; $f_c:C\to C'$; $f_p:P\to R$;

其中:$C'=\{Req\_level,Den\_level\}$;

$R=\{decompose,deriveReqt,refine,Satisfy,$
$abstractionLevel,higherAbstractionLevel\}$.

(1)论域 D:表示公式中的个体变元符号在实际项目中的具体解释,包括实际项目中所有的制品标识符、制品的属性;

(2)函数 f_s:定义了符号集合 S 中的个体变元到实际项目中制品标识符 S'的一个映射.若有 $f_s(s_i)=s_k'$,表明公式中个体变元符号 s_i 解释为某制品的唯一标识符 s_k';

(3)函数 f_c:定义了符号集合 C 中的个体变元符号到实际项目中制品具体属性的映射.本文所研究的追踪模型仅涉及了需求制品和设计制品,则有 $C'=\{Req\_level,Den\_level\}$,分别表示制品的抽象层次为需求层与设计层;

(4)函数 f_p:定义了二元谓词符号集合 P 到实际项目中各种关系集合 R 的映射.其中$\{decompose,deriveReqt,refine\}$描述了 SysML 图中的需求关系;$Satisfy$ 描述了需求与设计制品间的追踪关系;$abstractionLevel$ 描述了制品与其抽象层次的对应关系;$higherAbstractionLevel$ 描述了不同抽象层次间的偏序关系.表 1 总结了公式中的符号集到其所对应的语义集合中元素的具体解释.

表 1 公式符号的解释

语义集合	集合元素	解释
S'	$s_k'\in S'$	实际项目中制品标识符
C'	Req_level	表示制品为需求过程制品
	Den_level	表示制品为设计过程制品
R	$decompose$	需求间的分解关系
	$deriveReqt$	需求间的导出关系
	$refine$	需求间的精化关系
	$Satisfy$	需求与设计制品间的可满足关系
	$abstractionLevel$	制品与其抽象层次属性的对应关系
	$higherAbstraction Level$	不同抽象层次的偏序关系

例如，$decompose(R_1,R_2)$表明需求R_1与需求R_2间存在分解关系；$refine(R_1,R_2)$表明需求R_1与需求R_2间存在精化关系；$Satisfy(BrakeControl,R_1)$表明设计元素$BrakeControl$与需求$R_1$间存在可满足关系；$abstractionLevel(User_Req,Req_level)$表明制品$User_Req$为需求层制品；$higherAbstractionLevel(Req_level,Den_level)$表明需求层制品比设计层制品有更高的抽象层次，设计层制品比需求层制品更加具体.

基于形式系统的语法和语义，可以抽象描述实际项目中的追踪信息. 例如，使用公式$P_t^2(s_i,s_j)$表示制品s_i、s_j存在一条类型为t的追踪链.

实际项目中已有或给定的追踪关系在Π_{Theory}表示为公理. 利用推理规则推导出的隐含追踪关系表示为Π_{Theory}中的定理.

定义 4.　$\Pi_{\text{Theory}}=P_t^2(s_i,s_j)$等价于：实际项目中制品$s_i$到$s_j$间存在一条类型为$t$的可追踪链. 所有形如$P_t^2(s_i,s_j)$的公理或定理集合构成了类型为$t$的追踪关系.

5　基于谓词逻辑的需求追踪方法

基于谓词逻辑的形式系统Π_{Theory}用以抽象描述实际项目中的追踪信息，给出了系统中的公式符号在实际项目中的具体解释. 5.1 节将进一步给出实际项目中需求追踪关系的精确语义，然后在 5.2 节和 5.3 节分别给出了需求间横向追踪关系的推导、需求关系的一致性检验方法，以及纵向追踪关系的约束规则.

5.1　基于谓词逻辑系统的需求追踪关系语义

为了建立精确、完整的 SysML 需求间的横向追踪关系以及需求与 AADL 构件之间的纵向追踪关系，本节将基于谓词逻辑系统Π_{Theory}进一步定义需求追踪关系的精确语义. 在定义需求追踪关系语义之前，首先需要给出需求的精确语义.

5.1.1　需求

定义 5.　需求R被定义为一个二元组$\langle P,S\rangle$，其中P为系统需要满足的属性，S为满足属性P的系统集合. 对$\forall s\in S,P(s)$.

将需求属性P视为谓词逻辑中的合取范式(Conjunctive Normal Form，CNF)，基于谓词逻辑语义，谓词公式的真值取决于公式中的函数、谓词符号以及变元在语义模型$\mathcal{M}$中环境ℓ下的具体解释. 若将系统s视为语义模型$\mathcal{M}$，则当且仅当关系$s\models_\ell P$成立，系统s满足了需求属性P. 即公式P在系统s中环境为ℓ的情况下取值为真.

基于上述需求语义，本节将讨论需求间导出、精化以及分解关系的语义.

5.1.2　导出($deriveReqt$)关系语义

定义 6.　对需求R_1与R_2，有二元组$\langle P_1,S_1\rangle$、$\langle P_2,S_2\rangle$，其中P_1、P_2为需求的合取范式，S_1、S_2为满足需求公式的系统集合. 则从需求公式关系来看，需求R_1由需求R_2导出当且仅当：

(1) $(P_1\rightarrow P_2)$；

(2) $(\neg(P_2\rightarrow P_1))$可满足.

从满足需求公式系统集合的角度来看，需求R_1由需求R_2导出当且仅当：

(1) $\forall s_1\in S_1,s_1\in S_2$；

(2) $\exists s_2\in S_2,s_2\notin S_1$.

即集合S_1真包含于S_2：$S_1\subset S_2$.

若有如下需求：

需求R_1：当燃油不足、空速过大或高度异常等事件发生，飞控系统应向飞行员发送警报；

需求R_2：飞控系统需要提供消息传送机制.

对于需求R_1、R_2可分别符号化为：

$$P_1=notify\_pilot(x,pl\_events),$$

$$P_2=provide\_msg(x),$$

其中x为自由变元，pl_events为常元符号. 谓词$provide_msg(x)$表示系统能够传送消息x，二元谓词$notify_pilot(x,pl_events)$表示能够向飞行员传送警报消息$x$. 基于领域知识，对任意系统，若系统能够向飞行员传送消息x，则系统必然能传送消息x.

定义函数符号集$\mathcal{F}=\{pl_events\}$，谓词符号集$\mathcal{P}=\{provide_msg,notify_pilot\}$. 给定符号对$(\mathcal{F},\mathcal{P})$上的一个模型$\mathcal{M}$，$\mathcal{M}$描述了某系统$s$，系统$s$由如下数据项组成：

(1) 论域$A=\{oil_msg,spd_msg,heg_msg,time_msg,pilot_events\}$；

(2) 谓词符号$provide_msg^M=\{oil_msg,spd_msg,heg_msg,temp_msg\}$，表示系统能够传送燃油不足、空速过大、高度异常及温度过高这几类消息事件；

(3) 谓词符号$notisfy_pilot^M=\{(oil_msg,pilot_events),(spd_msg,pilot_events),(heg_msg,pilot_events)\}$，表明当出现燃油、空速及高度异常这几类消息事件时，系统能向飞行员发送警报.

对于需求公式P_1、P_2，当自由变元x取值为oil_msg、spd_msg、heg_msg，需求R_1、R_2可分别描述为：

R_1：当系统出现燃油不足、空速过大及高度异

常时,向飞行员发送警报消息;

R_2:系统能够传送燃油不足、空速过大及高度异常这几类消息事件.

此时系统 s 满足需求公式 P_1、P_2. 则有:

① $\mathcal{M}\models_\ell P_1\rightarrow P_2$

当变元 x 取值为 $temp\_msg$,需求 R_1、R_2 可分别描述为:

R_1:当温度异常时,系统向飞行员发送警报消息;

R_2:系统能够传送温度异常这个消息事件.

当变元 x 取值为 $temp\_msg$ 时,在系统 s 中,需求公式 P_2 为真. 由于 $(temp\_msg, pl\_events)\notin notisfy\_pilot^M$,系统 s 不满足需求 R_1.

即 $x=temp\_msg$,公式 $provide\_msg(x)\rightarrow notify\_pilot(x, pl\_events)$ 为假. 则有:

② $\mathcal{M}\models_{\ell[x\rightarrow temp\_msg]}(\neg(P_2\rightarrow P_1))$

对任意系统 s,其满足需求 R_1 则必然满足需求 R_2,存在如上系统,其满足需求 R_2 并不满足需求 R_1. 根据 $deriveReqt$ 关系语义,需求 R_1 由需求 R_2 导出,需求 R_2 是需求 R_1 实现的前提.

此外,$deriveReqt$ 关系还具有如下结构属性:

(1)反自反性:$deriveReqt$ 关系是反自反的,不存在需求到其自身的导出. 即

$$\Pi_{\text{Theory}}\models\neg\, deriveReqt(R_1, R_1);$$

(2)反对称性:$deriveReqt$ 关系是反对称的,需求之间不存在相互导出. 即

$$(\Pi_{\text{Theory}}\models deriveReqt(R_1, R_2))\rightarrow$$
$$(\Pi_{\text{Theory}}\models\neg\, deriveReqt(R_2, R_1));$$

(3)可传递性:$deriveReqt$ 关系是可传递的. 即

$$(\Pi_{\text{Theory}}\models deriveReqt(R_1, R_2))\wedge$$
$$(\Pi_{\text{Theory}}\models deriveReqt(R_2, R_3))\rightarrow$$
$$(\Pi_{\text{Theory}}\models deriveReqt(R_1, R_3)).$$

基于结构属性可以实现推导后需求关系的一致性检验,判断经推导后得到需求关系是否违反结构属性. 图 5 描述了 $deriveReqt$ 关系的反自反、反对称以及可传递特性. 若已知需求 R_1 由需求 R_2 导出,需求 R_2 由需求 R_3 导出,利用传递特性,则有需求 R_1 由需求 R_3 导出.

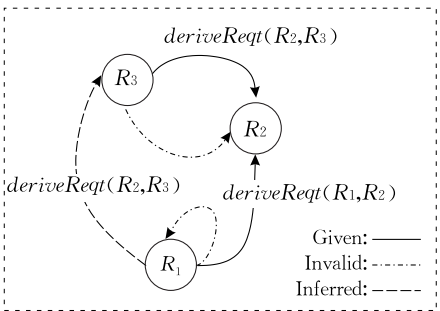

图 5 $deriveReqt$ 关系的结构属性

5.1.3 分解($decompose$)关系语义

定义 7. 若有需求 $R_1, R_2, \cdots, R_n, n\geq 2$,需求 $R_2, \cdots, R_n$ 的 CNF 公式可描述为

$$P_i=(p_1^i\wedge p_2^i\wedge\cdots\wedge p_m^i);\ m\geq 1,\ i\in 2,\cdots,n.$$

复杂需求 R_1 包含若干个子需求 $R_2, \cdots, R_n$,即需求 $R_2, \cdots, R_n$ 分解了需 R_1 当且仅当:

(1) $P_1=(P_2\wedge\cdots\wedge P_n)$;

(2) $\neg(P_i\rightarrow P_1), i\in 2,\cdots,n$ 可满足.

根据定义,若条件(1)成立,则有:$P_1\rightarrow P_2, \cdots, P_1\rightarrow P_n$,若条件(2)成立,则有:$\neg(P_2\rightarrow P_1), \cdots, \neg(P_n\rightarrow P_1)$. 根据 $deriveReqt$ 关系语义,若需求 R_1 可分解为子需求 $R_2, \cdots, R_n$,那么需求 R_1 必与需求 $R_2, \cdots, R_n$ 导出关系.

若需求 R_1 可分解为需求 $R_2, \cdots, R_n$,设 $S_1, \cdots, S_n$ 分别为满足需求 $R_1, \cdots, R_n$ 的系统集合. 对 $\forall s\in S_1, P_1(s)$,由于 $P_1\rightarrow P_2$,则 $P_2(s), s\in S_2$. $\exists s\in S_2$ 使得 $P_2(s)$ 为真,$P_1(s)$ 为假. 因此,$S_1\subset S_2$. 依此类推,$S_1\subset S_3, \cdots, S_1\subset S_n$. 即满足复杂需求的系统必然满足其分解的简单子需求,满足简单子需求的系统不一定满足复杂需求.

若有如下需求:

需求 R_1:系统支持基于优先级、先来先服务以及时间片轮转的任务调度方法.

需求 R_2:系统支持基于优先级的调度方法.

设需求 R_1、R_2 可分别符号化为:

$$P_1=schedu\_task(priority)\wedge schedu\_task(fsfs)\wedge schedu\_task(time\_piece),$$
$$P_2=schedu\_task(priority),$$

其中,$priority$、$fsfs$、$time\_piece$ 为零元函数符号,即为个体常元. 谓词 $schedu\_task$ 表示系统中的某种任务调度方法.

定义零元函数集 $\mathcal{F}=\{priority, fsfs, time\_piece\}$,谓词符号集 $\mathcal{P}=\{schedu\_task\}$. 给定符号对 $(\mathcal{F},\mathcal{P})$ 上的一个模型 $\mathcal{M}$,$\mathcal{M}$ 描述了某系统 s,系统 s 由如下数据项组成:

(1) 论域 $A=\{priority\_sch, fcfs\_sch, timepiece\_sch\}$;

(2) 函数符号 $priority^M=priority\_sch$,描述了领域中的优先级调度方法;

(3) 函数符号 $fcfs^M=fcfs\_sch$,描述了领域中的先来先服务调度方法;

(4) 函数符号 $time\_piece^M=timepiece\_sch$,描

述了领域中的时间片轮转调度方法；

(5) 谓词符号 $schedu\_task^M=\{priority\_sch\}$，表示系统 s 支持优先级调度：

① $P_1=P_2\wedge schedu_task(fcfs)\wedge schedu_task(time_piece)$恒成立；

在系统 s 中，系统 s 支持优先级调度算法，即满足需求 R_2. 由于 $fcfs\_sch\notin schedu\_task^M$、$timepiece\_sch\notin schedu\_task^M$，表明系统 s 不支持先来先服务和时间片调度，不满足需求 R_1. 则：

② $\mathcal{M}\models\neg(schedu\_task(priority\_sch)\rightarrow(schedu\_task(priority\_sch)\wedge schedu\_task(fcfs\_sch)\wedge schedu\_task(timepiece\_sch)))$.

根据分解关系语义，需求 R_2 是 R_1 的一个分解. 任意满足需求 R_1 的系统必然满足其子需求 R_2. 存在如上系统 s，满足子需求 R_2，不满足复杂需求 R_1.

与 *deriveReqt* 关系的结构属性类似，*decompose* 关系也具有反自反、反对称和传递性.

5.1.4　精化(*refine*)关系语义

定义 8.　设需求R_2的公式 $P_2=(p_1\wedge p_2\wedge\cdots\wedge p_n\wedge p_{n+1}\wedge p_m)$；$n\geq1$，$m\geq n$ 则需求 R_1 是对需求 R_2 的精化当且仅当：

(1) $P_1=(q_1\wedge q_2\wedge\cdots\wedge q_n\wedge p_{n+1}\wedge p_m)$；$n\geq1$，$m\geq n$；其中，$q_1,q_2,\cdots,q_n$ 满足条件：$q_i\rightarrow p_i$，$i\in1,\cdots,n$；

(2) $(\neg(p_i\rightarrow q_i))$，$i\in1,\cdots,n$ 可满足.

根据定义，若条件(1)成立，则公式$(P_1\rightarrow P_2)$成立. 若条件(2)成立，则公式$(\neg(P_2\rightarrow P_1))$可满足. 根据 *deriveReqt* 关系语义，若需求 R_1 是对需求 R_2 的精化，那么需求 R_1 必由需求 R_2 导出. 与导出关系类似，满足需求 R_1 的系统集合 S_1 真包含于满足需求 R_2 的系统集合 S_2，即 $S_1\subset S_2$若有如下需求：

需求 R_1：系统需要对襟翼、缝翼等机翼进行故障检测.

需求 R_2：系统需提供故障检测功能.

设需求 R_1、R_2 可分别符号化为：

$$P_1=wing\_test(x),$$

$$P_2=fault\_test(x),$$

其中 x 为自由变元. 谓词 $fault_test(x)$表示对变元 x 进行故障检测. $wing_test(x)$表示对机翼 x 进行故障检测.

定义谓词符号集$\mathcal{P}=\{fault_test,wing_test\}$. $\mathcal{F}$为空集. 给定符号对$(\mathcal{F},\mathcal{P})$上的一个模型$\mathcal{M}$，$\mathcal{M}$描述了某系统 s，系统 s 由如下数据项组成：

(1) 论域 $A=\{flap,slat,motor\}$；

(2) 谓词符号 $fault\_test^M=\{flap,slat,motor\}$，表示系统 s 能够对襟翼、缝翼以及电机进行故障检测；

(3) $wing\_test^M=\{flap,slat\}$，表示系统 s 能够对襟翼、缝翼进行故障检测.

显然，$wing\_test^M\subseteq fault\_test^M$. 当变元 x 取值为 $flap$，$slat$ 时，在系统 s 中公式 P_1 为真，此时公式 P_2 也为真. 则有：

$$\mathcal{M}\models_{\ell}wing\_test(x)\rightarrow fault\_test(x).$$

当变元 x 取值为 $motor$，$x\in fault\_test^M$，需求 R_2 的公式 $fault_test(motor)$为真，即系统 s 满足需求 R_2. $x\notin wing\_test^M$，此时需求 R_1 的公式 $wing_test(x)$为假，表明对电机的故障检测不属于需求 R_1 的范围内，则有：

$$\mathcal{M}\models_{\ell[x\rightarrow motor]}(\neg(fault\_test(x)\rightarrow wing\_test(x))).$$

根据 *refine* 关系语义，需求 R_1 是对需求 R_2 的精化，需求 R_1 来源于需求 R_2，在原需求基础上补充了实现细节，缩小了需求 R_1 的实现范围，排除了对电机的检测需求.

与 *deriveReqt* 关系的结构属性类似，*refine* 关系也具有反自反、反对称和可传递性.

5.1.5　*Satisfy* 关系语义

定义 9.　定义 C 为 AADL 模型中构件的集合，SR 为需求集合，则可满足关系 $Satisfy\subseteq C\times SR$. 公式 $Satisfy(x,R)$，表示 AADL 构件 x 与需求 R 相关.

定义 10.　若有需求 R，定义$\mathcal{M}_{Satisfy}(R)$为所有与需求 R 相关的 AADL 构件集合.

(1) $\mathcal{M}_{Satisfy}(R)=\{x_i\in C|(2)\}$；

(2) $\Pi_{\text{Theory}}=Satisfy(x_i,R)$.

如图 6 所示，$\mathcal{M}_{Satisfy}(R)$的元素共同作用，满足或实现了需求 R.

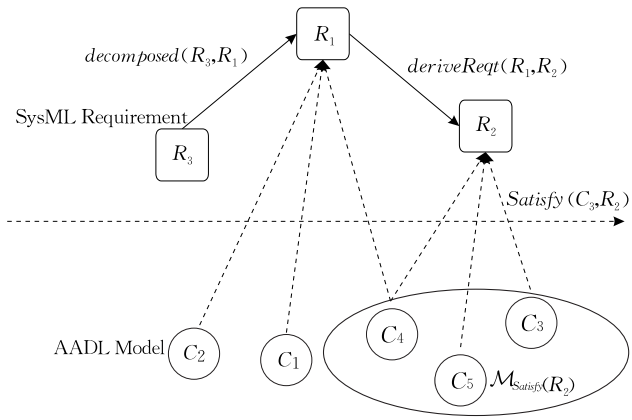

图 6　与需求相关的 AADL 构件集合

与需求间的追踪关系不同，可满足关系描述了需求与设计间的追踪关系，需求层制品比设计层制品有更高的抽象层次. 即

$(\Pi_{\text{Theory}} = Satisfy(x, y)) \rightarrow (abstractionLevel(x, Den\_level) \wedge abstractionLevel(y, Req\_level)) \wedge higherAbstractionLevel(Req\_level, Den\_level)$.

Satisfy 关系的结构属性与 *deriveReqt* 关系类似，同样具有反自反和反对称性，但由于本文仅研究需求与设计制品间的 *Satisfy* 关系，不考虑设计制品与代码间的 *Satisfy* 关系，因此，不考虑其可传递特性.

5.2 SysML 需求间隐含追踪关系的推导与需求关系一致性检测方法

分析 SysML 需求横向追踪关系语义，如表 2 所示，发现需求关系到集合关系的映射并非全部等价. 例如，由需求间的精化、分解关系仅能推导出系统集合的包含关系，而集合的包含关系无法推导出需求间的精化、分解关系. 利用集合关系特性，仅能实现隐含关系的部分简单推导和检验. 为此，本文将利用公式关系进行追踪关系推导与检验，集合关系仅仅作为辅助手段.

表 2 需求横向追踪关系语义

需求 $R=\langle P,S\rangle$		
需求间追踪关系	语义	
$deriveReqt(R_1,R_2)$	$\Leftrightarrow$(1)$(P_1\rightarrow P_2)$； (2)$(\neg(P_2\rightarrow P_1))$可满足	$\Leftrightarrow S_1\subset S_2$
$refine(R_1,R_2)$	$\Leftrightarrow$令 $P_1=(q_1\wedge q_2\wedge\cdots\wedge q_n\wedge p_{n+1}\wedge p_m)$ $P_2=(p_1\wedge p_2\wedge\cdots\wedge p_n\wedge p_{n+1}\wedge p_m)$ (1)$q_i\rightarrow p_i, i\in 1,\cdots,n$； (2)$(\neg(p_i\rightarrow q_i))$可满足	$\Leftrightarrow S_1\subset S_2$
$decompose(R_1,R_2)$	$\Leftrightarrow$(1)$P_1=(P_2\wedge\cdots\wedge P_n)$ (2)$\neg(P_2\rightarrow P_1)$可满足	$\Leftrightarrow S_1\subset S_2$

依据前文对需求关系的定义，导出关系仅仅与需求公式间的逻辑关系相关而没有考虑公式子句之间的关系. 精化和分解关系对需求公式子句间的关系进行了约束，包括子句间的逻辑关系和数量关系两个方面. 从子句数量上看，存在精化关系的两个需求 CNF 公式间有相同的子句个数，从子句逻辑关系上来看，精化需求公式中某子句 q_i 替换了原需求公式的子句 p_i，q_i 蕴含且不等价于 p_i. 对于存在分解关系的两个需求，从子句数量上来看，分解需求的子句个数小于原需求的子句个数，从子句间的逻辑关系来看，分解需求中的任意子句与原需求中的子句相等价.

因此，公式关系的谓词集合描述可以分为两类，分别用于描述公式子句间的数量关系以及公式或公式子句间的逻辑关系. 即分别定义谓词符号集 Num_Set：{*part_all*，*all_all*} 描述需求公式子句间的数量关系，定义谓词符号集 $Logic\_Set$：{*part_imply*，*all_equal*，*imply*} 描述公式或公式子句间的逻辑关系.

集合 Logic_Set 中各谓词的含义如下：

(1) $part\_imply(P_1,P_2)$：描述需求公式子句间的蕴含关系. 即公式 P_1 中至少存在一个子句 P_i^1，P_i^1 蕴含公式 P_2 中的某子句 P_j^2，且 P_i^1 与 P_j^2 不等价；

(2) $imply(P_1,P_2)$：描述需求公式间的蕴含关系. 公式 P_1 蕴含且不等价于 P_2. 即公式 P_1 中的任一子句 P_i^1，P_i^1 蕴含公式 P_2 中的某子句 P_j^2，且 P_i^1 与 P_j^2 不等价；

(3) $all\_equal(P_1,P_2)$：描述需求公式子句间的等价关系. 即对公式 P_1 中的任意子句 P_i^1，在公式 P_2 中都存在某确定子句 P_j^2 与之等价.

集合 *Num_Set* 中各谓词的含义如下：

(1) $part\_all(P_1,P_2)$：描述需求公式子句间数量的多少关系. 即需求公式 P_1 的子句个数小于公式 P_2 的子句个数. 此外，对公式 P_1 中的任意子句 P_i^1，在公式 P_2 中都存在某个确定的子句 P_j^2 与之相对应，子句 P_i^1 蕴含或等价于 P_j^2；

(2) $all\_all(P_1,P_2)$：描述需求公式子句间数量的等价关系. 即需求公式 P_1 中的子句个数与公式 P_2 中的子句个数相等. 此外，对公式 P_1 中的任意子句 P_i^1，在公式 P_2 中都存在某确定子句 P_j^2 与之相对应，子句 P_i^1 蕴含或等价于 P_j^2.

这些谓词公式具有如下性质：

(1) *part_all*，*all_all*，*part_imply*，*imply* 以及 *all_equal* 关系均具有传递性；

(2) *part_all* 和 *all_all* 关系具有互斥性；

(3) *all_equal* 和 *part_imply* 关系具有互斥性；

(4) *all_equal* 和 *imply* 关系具有互斥性.

基于上述谓词符号语义，给出需求关系到公式关系的等价映射规则，如表 3 所示.

表 3 需求关系到公式关系的等价映射

需求关系	公式关系
$deriveReqt(R_1,R_2)$	$\Leftrightarrow imply(P_1,P_2)$
$refine(R_1,R_2)$	$\Leftrightarrow all\_all(P_1,P_2)\wedge part\_imply(P_1,P_2)$
$decompose(R_1,R_2)$	$\Leftrightarrow part\_all(P_1,P_2)\wedge all\_equal(P_1,P_2)$

根据 *deriveReqt* 关系的语义，易得 *deriveReqt*

$(R_1,R_2)\Leftrightarrow imply(P_1,P_2)$；根据需求间 *refine* 关系语义，若需求 R_1 精化了需求 R_2，则公式 P_1 替换了公式 P_2 中的部分子句，公式间子句个数相等，且若子句 q_i 替换了 p_i 子句，则 q_i 蕴含且不等价于 p_i，因此 $refine(R_1,R_2)\Leftrightarrow all\_all(P_1,P_2)\wedge part\_imply(P_1,P_2)$；根据需求间 *decompose* 关系语义，若需求 R_1 分解了需求 R_2，则需求公式 P_1 是需求公式 P_2 的一部分，即 P_1 的子句个数小于 P_2 的子句个数，且 P_1 中的各子句与 P_2 中的子句相等价，因此得到 $decompose(R_1,R_2)\Leftrightarrow part\_all(P_1,P_2)\wedge all\_equal(P_1,P_2)$.

描述公式关系的谓词公式间的推导规则是需求间隐含追踪关系推导的基础，分析集合 *Num_Set* 与 *Logic_Set* 中各关系谓词及其组合情况，可以得到如下推导规则：

(1) $part\_all(P_1,P_2)\wedge all\_all(P_2,P_3)\rightarrow part\_all(P_1,P_3)$；

(2) $all\_all(P_1,P_2)\wedge part\_all(P_2,P_3)\rightarrow part\_all(P_1,P_3)$；

(3) $part\_imply(P_1,P_2)\wedge imply(P_2,P_3)\rightarrow imply(P_1,P_3)$；

(4) $imply(P_1,P_2)\wedge part\_imply(P_2,P_3)\rightarrow imply(P_1,P_3)$；

(5) $part\_imply(P_1,P_2)\wedge all\_equal(P_2,P_3)\rightarrow part\_imply(P_1,P_3)$；

(6) $imply(P_1,P_2)\wedge all\_equal(P_2,P_3)\rightarrow imply(P_1,P_3)$；

(7) $all\_equal(P_1,P_2)\wedge imply(P_2,P_3)\rightarrow imply(P_1,P_3)$；

(8) $all\_all(P_1,P_2)\wedge part\_imply(P_1,P_2)\rightarrow imply(P_1,P_2)$；

(9) $part\_all(P_1,P_2)\wedge all\_equal(P_1,P_2)\rightarrow imply(P_2,P_1)$.

基于需求关系到公式关系的映射以及描述公式关系的谓词公式间的推导规则，可以自动推导出 SysML 需求间隐含的横向追踪关系. 此外，还需要对需求关系的语义一致性进行检验. 需求关系的语义不一致则指的是 SysML 需求间存在的不同类型追踪关系的语义产生了矛盾.

分析集合 *Num_Set* 与 *Logic_Set* 中各关系谓词及其组合情况，可以得到如下检验规则：

(1) $part_all(P_1,P_2)$与 $all_all(P_1,P_2)$语义不一致.

$part_all(P_1,P_2)$描述的公式 P_1 中的子句个数小于公式 P_2 中的子句个数，而 $all_all(P_1,P_2)$表示的是公式 P_1 中的子句个数等于公式 P_2 中的子句个数. 两者必然存在语义冲突；

(2) $part_imply(P_1,P_2)$与 $all_equal(P_1,P_2)$语义不一致；

(3) $imply(P_1,P_2)$与 $all_equal(P_1,P_2)$语义不一致.

$part_imply(P_1,P_2)$和 $imply(P_1,P_2)$描述的是公式之间的蕴含关系，但两者都表明公式 P_1 与 P_2 不等价，而 $all_equal(P_1,P_2)$则表示的是公式 P_1 与 P_2 之间存在等价关系，明显存在语义冲突.

5.3　SysML 需求与 AADL 构件间可满足关系的约束规则

若 AADL 构件 x_i 与需求 R 间存在一条可满足追踪链 $Satisfy(x_i,R)$，则表明 AADL 构件 x_i 与需求 R 相关，参与了 SysML 需求的实现. 所有与需求 R 间存在可满足追踪链的 AADL 构件构成了集合 $\mathcal{M}_{Satisfy}(R)$，集合中的元素共同作用，满足或实现了需求 R.

系统设计可视为对需求的精化，实现需求的 AADL 模型中仍然维持需求间的追踪关系. 因此，基于 SysML 需求关系，可以给出 AADL 构件与需求间可满足关系的约束规则，该约束规则保证了与需求相关的 AADL 构件集合间仍然维持了 SysML 需求关系语义. 基于约束规则，可以在已有的可满足关系基础上进行关系的部分推导及检验.

本节在研究需求间的分解、导出及精化关系在集合 $\mathcal{M}_{Satisfy}(R)$ 上映射的基础上，给出如下约束规则.

约束规则 1. 若需求 R_1 是由需求 R_2 导出的，即 R_2 是 R_1 实现的基础或前提，则在系统设计中与 R_1 和 R_2 相关的 AADL 构件集合必有交集，如图 7 所示. 即

$$\Pi_{\text{Theory}}\models deriveReqt(R_1,R_2)\rightarrow \mathcal{M}_{Satisfy}(R_1)\cap\mathcal{M}_{Satisfy}(R_2)\neq\varnothing$$

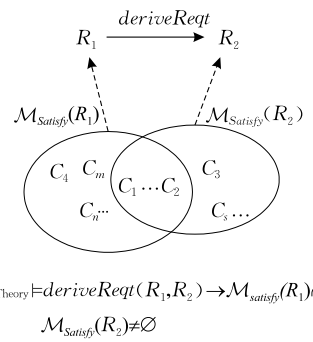

图 7　*deriveReqt* 关系在 AADL 构件集合中的映射

该约束规则可用于对已有的可满足关系进行检验.若需求 R_1 是由需求 R_2 导出的且 $\mathcal{M}_{Satisfy}(R_1)$ 与 $\mathcal{M}_{Satisfy}(R_2)$ 的交集为空,则表明已有需求间的 *deriveReqt* 关系或者需求与 AADL 构件之间的 *Satisfy* 关系可能存在遗漏或偏差.

约束规则 2. 若需求 R_2 是对需求 R_1 的精化,基于精化关系语义,精化后的需求是在原需求的基础上补充了实现细节,约束了原需求的实现范围.因此,参与实现需求 R_2 的 AADL 构件必然也参与了需求 R_1 的实现.与需求 R_2 存在 *Satisfy* 关系的 AADL 构件集合 $\mathcal{M}_{Satisfy}(R_2)$ 包含于 $\mathcal{M}_{Satisfy}(R_1)$.即

$$\Pi_{\text{Theory}} \models refine(R_2,R_1) \rightarrow \mathcal{M}_{Satisfy}(R_2) \subset \mathcal{M}_{Satisfy}(R_1).$$

如图 8 所示,需求 $R_2,\cdots,R_n$ 均是对需求 R_1 的精化,则 $\mathcal{M}_{Satisfy}(R_2),\cdots,\mathcal{M}_{Satisfy}(R_n)$ 均包含于 $\mathcal{M}_{Satisfy}(R_1)$.

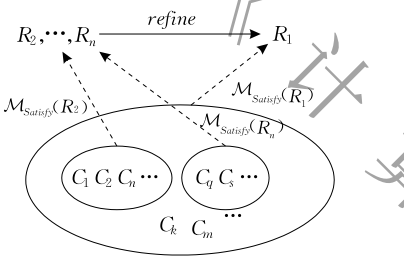

图 8 *refine* 关系在 AADL 构件集合中的映射

该约束规则可用于对已有的追踪关系进行检验,若需求 R_2 是对需求 R_1 的精化,但 $\mathcal{M}_{Satisfy}(R_2)$ 不包含于 $\mathcal{M}_{Satisfy}(R_1)$,则表明 R_2 与 R_1 间的精化关系不合理或集合 $\mathcal{M}_{Satisfy}(R_2)$、$\mathcal{M}_{Satisfy}(R_1)$ 中的元素与需求间的可满足性关系可能存在遗漏或偏差.此外,通过约束规则可得:

$$\Pi_{\text{Theory}} \models Satisfy(C_i,R_2) \wedge \Pi_{\text{Theory}} \models refine(R_2,R_1) \rightarrow \Pi_{\text{Theory}} \models Satisfy(C_i,R_1).$$

即若需求 R_2 是对 R_1 的精化,且 R_2 与 AADL 构件 C_i,则有 C_i 必与 R_1 相关.该规则可用于需求与 AADL 构件间 *Satisfy* 关系的推导.

约束规则 3. 若需求 $R_2,\cdots,R_n$ 分解了需求 R_1,则需求 R_2 是需求 R_1 的一部分,参与实现需求 R_2 的 AADL 构件必然参与需求 R_1 的实现,与需求 R_2 存在关系的 AADL 构件集合 $\mathcal{M}_{Satisfy}(R_2)$ 包含于 $\mathcal{M}_{Satisfy}(R_1)$,即

$$\Pi_{\text{Theory}} \models decompose(R_2,R_1) \rightarrow \mathcal{M}_{Satisfy}(R_2) \subset \mathcal{M}_{Satisfy}(R_1).$$

如图 9 所示,若需求 $R_2,\cdots,R_n$ 分解了需求 R_1,即需求 R_1 包含 $R_2,\cdots,R_n$,则 $\mathcal{M}_{Satisfy}(R_2),\cdots,\mathcal{M}_{Satisfy}(R_n)$ 包含于 $\mathcal{M}_{Satisfy}(R_1)$.若需求 $R_2,\cdots,R_n$ 是对需求 R_1 的一个完备分解,即需求公式满足 $P_1=P_2\wedge\cdots\wedge P_n$,则有:

$$\mathcal{M}_{Satisfy}(R_1)=\mathcal{M}_{Satisfy}(R_2)\cup\cdots\cup\mathcal{M}_{Satisfy}(R_n).$$

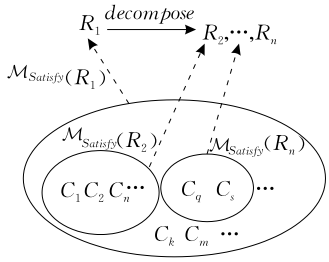

图 9 *decompose* 关系在 AADL 构件集合中的映射

该约束规则可用于对已有的追踪关系进行检验,若需求 $R_2,\cdots,R_n$ 分解了需求 R_1,同时存在需求 R_i 使得 $\mathcal{M}_{Satisfy}(R_i)$ 不包含于 $\mathcal{M}_{Satisfy}(R_1)$,则表明 R_i 与 R_1 间的分解关系存在不合理性或者集合 $\mathcal{M}_{Satisfy}(R_i)$ 与 $\mathcal{M}_{Satisfy}(R_1)$ 中的元素与需求间的 *Satisfy* 关系可能会存在遗漏和偏差.此外,通过约束规则可得:

$$\Pi_{\text{Theory}} \models Satisfy(C_i,R_i) \wedge \Pi_{\text{Theory}} \models decompose(R_i,R_1) \rightarrow \Pi_{\text{Theory}} \models Satisfy(C_i,R_1)$$

即若需求 R_i 是需求 R_1 的一个分解,且 AADL 构件 C_i 与需求 R_i 相关,则 AADL 构件 C_i 必与需求 R_1 相关.因此,该规则还可用于需求与 AADL 构件间 *Satisfy* 关系的推导.

6 案例分析

襟缝翼控制系统是飞控系统中控制襟翼和缝翼行为的重要组件,在起飞和降落过程中增加升力、增大失速临角,该单元直接影响着飞行的安全及稳定,保障其安全性至关重要[51].本团队在前期的工作中已通过模型检测技术实现了襟缝翼控制系统源代码层面的验证[52],本文将重点关注其可追踪性方面的研究.通过需求横向追踪关系的推导与检验,能够发现需求间隐含的关系并可对需求关系的合理性进行验证.通过需求与设计间的纵向追踪关系的推导与检验,可以检验需求是否被完整地体现在了系统设计之中.本文选取了如表 4 所示的襟缝翼控制单元的部分需求作为案例进行追踪关系的构建与验证,并基于所建立的追踪关系进行安全性分析.

表 4　襟缝翼控制系统中的部分需求描述

需求	描述
R_1	当发现系统软硬件故障后，采取相应的安全处理措施.
R_2	系统的周期自检中，需要对风扇转速、处理器温度输出的有效性进行检测.
R_3	系统周期自检中，对硬件设备输出的有效性进行检测.
R_4	系统应提供对处理器温度输出的有效性检测功能.
R_5	系统应该提供对风扇转速输出的有效性检测功能.
R_6	接口数据处理模块将从传感器采集的信息进行滤波处理及格式转换.

6.1　襟缝翼控制系统需求横向追踪关系的推导与验证

分析表 4 中的需求及其相互关系，可以建立如图 10 所示的 SysML 需求图.

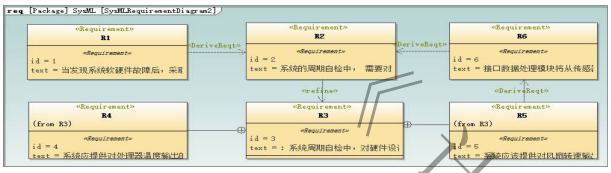

图 10　襟缝翼控制单元的 SysML 需求图

依据表 3 所示的映射规则，可以将图 10 所示的需求关系转换到如表 5 所示的公式关系.

表 5　图 10 中需求关系到公式关系的映射

需求关系	公式关系
$deriveReqt(R_1,R_2)$	$\Leftrightarrow imply(P_1,P_2)$
$refine(R_2,R_3)$	$\Leftrightarrow all\_all(P_2,P_3)\wedge part\_imply(P_2,P_3)$
$decompose(R_3,R_4)$	$\Leftrightarrow part\_all(P_3,P_4)\wedge all\_equal(P_3,P_4)$
$deriveReqt(R_5,R_6)$	$\Leftrightarrow imply(P_5,P_6)$
$deriveReqt(R_6,R_2)$	$\Leftrightarrow imply(P_6,P_2)$
$decompose(R_3,R_5)$	$\Leftrightarrow part\_all(P_3,P_5)\wedge all\_equal(P_3,P_5)$

首先，利用本文所定义的谓词逻辑系统进行需求层的横向追踪关系的推导，建立完整的需求横向追踪关系. 以上述需求中的需求 R_1、R_2、R_3 和 R_4 为例进行分析，利用 5.1 节中所描述的需求间隐含追踪关系的推导规则可以建立起完整的横向追踪关系. 例如：$part\_all(P_3,P_4)\wedge all\_equal(P_3,P_4)\rightarrow imply(P_4,P_3)$，因此，需求 R_3 和 R_4 之间还应存在 $deriveReqt(R_4,R_3)$ 关系，即需求 R_3 是需求 R_4 的一个分解，同时需求 R_4 是由需求 R_3 导出的. 此外，$imply(P_1,P_2)\wedge part\_imply(P_2,P_3)\rightarrow imply(P_1,P_3)$，即需求 R_1 和 R_3 之间存在 $deriveReqt(R_1,R_3)$ 关系. 同理，还可得到 $deriveReqt(R_2,R_3)$、$deriveReqt(R_1,R_4)$ 等需求关系. 完整的需求关系如图 11 所示，其中，实现部分表示已有的给定的需求关系，虚线表示推导出的需求间隐含的追踪关系.

经过推理后可以发现同时存在公式关系

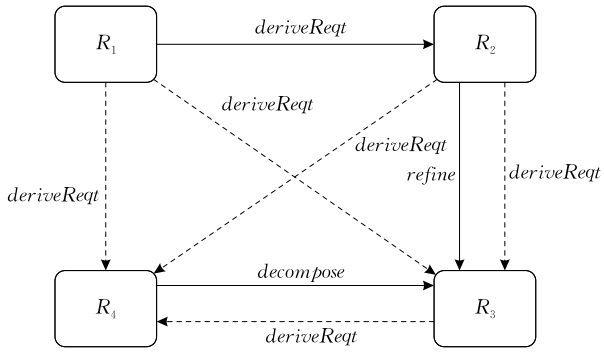

图 11　SysML 需求关系推导结果

$imply(P_2,P_6)$ 与 $imply(P_6,P_2)$，将其等价映射到对应的需求关系，分别为 $deriveReqt(R_2,R_6)$ 和 $deriveReqt(R_6,R_2)$. 因此，存在循环 $deriveReqt$ 关系冲突，违反了关系的反自反性，故所建立的需求关系是不合理的. 通过分析发现，需求 R_6 接口数据处理模块对数据的处理实质上并不由需求 R_2 对设备输出的有效性检测导出，表明给定的横向追踪关系有误，应当予以删除.

通过需求横向追踪关系的推导与检验，可以消除不合理的需求关系，并且可以获取隐含的需求关系. 消除不合理的需求关系可以避免因此而带来的系统设计风险，从而有效地提高系统安全性. 此外隐含的需求关系是实现需求与设计间可满足性关系（即纵向追踪关系）的推导与检验的基础.

6.2　襟缝翼控制系统需求纵向追踪关系的验证

实现需求横向追踪关系的推导与检验之后，将进一步考虑需求与设计之间的可满足性关系. 本文主要基于谓词逻辑系统对需求与设计间可满足性关系进行推导与检验.

图 12 给出部分了襟缝翼控制系统的 AADL 代码. 图中左侧为相应的 AADL 代码，右侧为 AADL 代码中对应的构件列表.

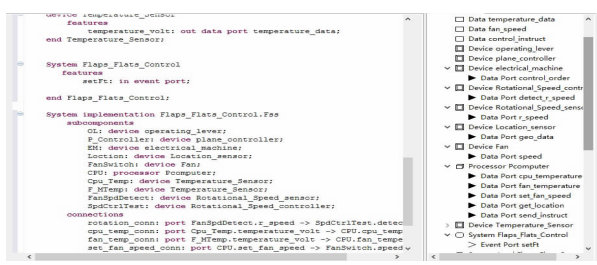

图 12　襟缝翼控制单元的 AADL 代码

首先，从图 10 可知需求 R_4、R_5 是对需求 R_3 的分解. 分析 SysML 需求与 AADL 构件之间的相关性，以需求 R_5（系统应该提供对风扇转速输出的有效性检测功能）为例，对 AADL 模型进行分析，其中外设 FanSwitch 对应的是风扇设备，传感器设备

FanSpdDetect 是用于检测风扇的转速的，很明显这两个 AADL 构件与需求 R_5 相关，两者与 R_5 之间存在 *Satisfy* 关系. 进一步分析需求 R_4，可知 R_4 与外设 CPU 以及处理器温度传感器 CPU_Temp 相对应. 并且当风扇 FanSwitch 转速过低且持续两分钟后，表明风扇转速不正常，产生事件 setFt，当 FanSpdDetect 接收到事件 setFt 时，由正常模式 normal 转入故障模式 fault. 因此需求 R_4 和 R_5 与 AADL 构件之间的可满足性关系如下：

$$\mathcal{M}_{Satisfy}(R_4)=\{\text{CPU},\text{CPU\_Temp},\cdots,\text{fault}\},$$

$$\mathcal{M}_{Satisfy}(R_5)=\{\text{FanSwitch},\cdots,\text{setFt},\text{setNml}\}.$$

需求 R_4、R_5 与需求 R_3 之间存在 *decompose* 关系，因此根据纵向追踪关系的约束规则 3 有：

$$\Pi_{\text{Theory}}\models decompose(R_4,R_3)\rightarrow \mathcal{M}_{Satisfy}(R_4)\subset\mathcal{M}_{Satisfy}(R_3),$$

$$\Pi_{\text{Theory}}\models decompose(R_5,R_3)\rightarrow \mathcal{M}_{Satisfy}(R_5)\subset\mathcal{M}_{Satisfy}(R_3).$$

即与需求 R_4、R_5 相关的 AADL 构件集合均属于与需求 R_3 相关的 AADL 构件集合，推导结果如图 13 所示.

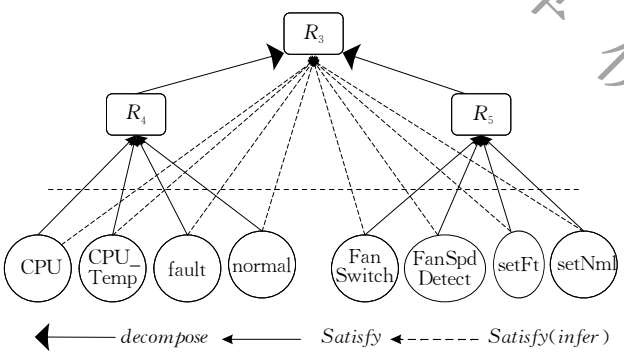

图 13 基于约束规则的可满足关系的推导

以需求 R_2、R_3 为例进行纵向追踪关系的验证，假定事先建立的纵向追踪关系如图 14 所示. AADL 构件中的模式 fault 和 normal、传感器 CPU_temp 以及外设 CPU 等与需求 R_2 相关，模式 FanSwitch 和 FanSpdDetect 与需求 R_3 相关.

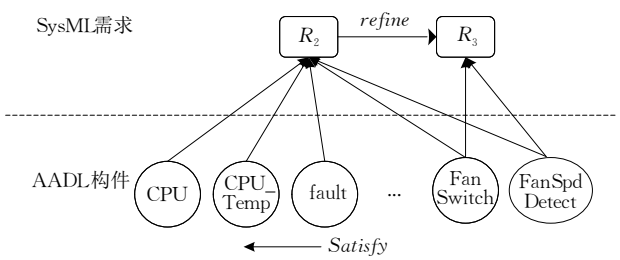

图 14 AADL 构件与需求 R_2、R_3 间的可满足关系

因为需求 R_2 和 R_3 之间存在精化关系，因此，依据约束规则 2 进行推理可知：

$$\Pi_{\text{Theory}}\models refine(R_2,R_3)\rightarrow \mathcal{M}_{Satisfy}(R_2)\subset\mathcal{M}_{Satisfy}(R_3).$$

即满足需求 R_2 的 AADL 构件集合应当包含在满足需求 R_3 的 AADL 构件集合之中，因此，图 14 中所建立的可满足关系不符合约束规则. 正确的纵向追踪关系应当为：外设 CPU、传感器 CPU_temp 和模式 fault 等满足需求 R_2 的所有 AADL 构件应均满足需求 R_3. 因此，对应的可满足性追踪关系，应当予以修正，避免由此而产生的设计缺陷以及进而可能产生的安全性事故.

基于需求纵向追踪信息，可以检验需求是否被全面、正确地体现在系统设计中，特别是与安全相关的需求，从而可以在系统设计的早期进行安全性分析，尽早发现系统设计中的缺陷并予以修正，从而降低系统开发成本和事故发生的可能性，提高系统的安全性.

7 结束语

在安全攸关领域，嵌入式系统需求追踪信息的缺失是导致系统安全性事故的一个重要原因，本文针对这一重要问题展开研究，通过两种标准建模语言——SysML 和 AADL 分别描述嵌入式系统需求和设计. 为了建立精确、完整的需求追踪关系，引入了谓词逻辑形式化地描述了需求与追踪关系的语义，然后基于语义模型，给出了 SysML 需求间横向追踪关系的推导与语义一致性检验方法，以及纵向追踪关系的推导与检验方法. 最后，通过实例分析说明了本文所提方法的有效性和正确性.

本文的主要工作是提出了一种支持需求追踪的语义模型，基于该语义模型，可以实现对追踪关系的自动推导和检验，保证追踪关系的精确和完整. 但并未考虑需求的变更与演化，而且完整的需求追踪工具的设计与实现也是未来工作的主要目标之一.

致 谢 在本文撰写和修改过程中，审稿专家和编辑老师提出了许多宝贵意见，在此表示衷心感谢！

参 考 文 献

[1] Huang Zhi-Qiu, Xu Bing-Feng, Kan Shuang-Long, et al. Survey on embedded software safety analysis standards, methods and tools for airborne system. Journal of Software, 2014, 25(2): 200-218(in Chinese)
(黄志球，徐丙凤，阚双龙等. 嵌入式机载软件安全性分析标准，方法及工具研究综述. 软件学报，2014，25(2)：200-218)

[2] CMMI Product Team. Capability maturity model integration. Software Engineering Institute, Pittsburgh, USA: Technical Report CMU/SEI-2006-TR-008, 2006

[3] Peraldi-Frati M A, Albinet A. Requirement traceability in safety critical systems//Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness & Safety. Valencia, Spain, 2010: 11-14

[4] Institute of Electrical and Electronics Engineers. IEEE Guide to Software Requirements Specifications. New York, USA: The Institute of Electrical and Electronics Engineers, 1984

[5] Spanoudakis G, Zisman A. Software traceability: A roadmap. Handbook of Software Engineering and Knowledge Engineering, 2005, 3: 395-428

[6] Lutz R R. Analyzing software requirements errors in safety-critical, embedded systems//Proceedings of the IEEE International Symposium on Requirements Engineering. San Diego, USA, 1993: 126-133

[7] Lutz R R, Mikulski I C. Empirical analysis of safety-critical anomalies during operations. IEEE Transactions on Software Engineering, 2004, 30(3): 172-180

[8] de Leon D C, Alves-Foss J. Experiments on processing and linking semantically augmented requirement specifications// Proceedings of the 37th Annual Hawaii International Conference on Systems Sciences. Big Island, USA, 2004: 1-10

[9] Athalye P, Maksimovic D, Erickson R. High-performance front-end converter for avionics applications. IEEE Transactions on Aerospace and Electronic Systems, 2003, 39(2): 462-470

[10] Afshar A, Hajyhosseinloo M, Eftekhari A, et al. A report of the injuries sustained in Iran Air Flight 277 that Crashed near Urmia, Iran. Archives of Iranian Medicine, 2012, 15(5): 317-319

[11] RTCA DO-178C. Software considerations in airborne systems and equipment certification. USA: Radio Technical Commission for Aeronautics, Incorporated, 2011

[12] RTCA S C. DO-333 formal methods supplement to DO-178C and DO-278A. USA: Radio Technical Commission for Aeronautics, Incorporated, 2011

[13] Iso I S O. 26262: Road vehicles-functional safety. International Standard ISO/FDIS 26262, 2011

[14] Rempel P, Mäder P. A quality model for the systematic assessment of requirements traceability//Proceedings of the IEEE 23rd International Requirements Engineering Conference Requirements Engineering Conference(RE). Ottawa, Canada, 2015: 176-185

[15] Holt J, Perry S. SysML for systems engineering. London, UK: The Institution of Engineering and Technology, 2008

[16] Aerospace S A E. SAE AS5506: Architecture Analysis and Design Language(AADL). New York, USA: SAE International, 2004

[17] Cleland-Huang J, Gotel O C Z, Huffman Hayes J, et al. Software traceability: Trends and future directions//Proceedings of the on Future of Software Engineering. Hyderabad, India, 2014: 55-69

[18] Baeza-Yates R, Ribeiro-Neto B. Modern Information Retrieval. New York, USA: ACM Press, 1999

[19] Winkler S, Pilgrim J. A survey of traceability in requirements engineering and model-driven development. Software and Systems Modeling(SoSyM), 2010, 9(4): 529-565

[20] Antoniol G, Canfora G, Casazza G, et al. Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering, 2002, 28(10): 970-983

[21] Settimi R, Cleland-Huang J, Ben Khadra O, et al. Supporting software evolution through dynamically retrieving traces to UML artifacts//Proceedings of the 7th International Workshop on Principles of Software Evolution. Kyoto, Japan, 2014: 49-54

[22] Egyed A, Grunbacher P. Automating requirements traceability: Beyond the record & replay paradigm//Proceedings of the 17th IEEE International Conference on Automated Software Engineering. Edinburgh, UK, 2002: 163-171

[23] Egyed A, Grünbacher P. Supporting software understanding with automated requirements traceability. International Journal of Software Engineering and Knowledge Engineering, 2005, 15(5): 783-810

[24] Spanoudakis G, Zisman A, Pérez-Minana E, et al. Rule-based generation of requirements traceability relations. Journal of Systems and Software, 2004, 72(2): 105-127

[25] Cysneiros G, Zisman A. Traceability and completeness checking for agent-oriented systems//Proceedings of the 2008 ACM Symposium on Applied Computing. Fortaleza, Brazil, 2008: 71-77

[26] Cleland-Huang J, Chang C K, Christensen M. Event-based traceability for managing evolutionary change. IEEE Transactions on Software Engineering, 2003, 29(9): 796-810

[27] Mäder P, Gotel O, Philippow I. Enabling automated traceability maintenance through the upkeep of traceability relations//Proceedings of the 5th European Conference on Model Driven Architecture-Foundations and Applications. Enschede, The Netherlands, 2009: 174-189

[28] Bondé L, Boulet P, Dekeyser J L. Traceability and interoperability at different levels of abstraction in model-driven engineering//Applications of Specification and Design Languages for SoCs. Dordrecht, The Netherlands: Springer, 2006: 263-276

[29] Schwarz H, Ebert J, Winter A. Graph-based traceability: A comprehensive approach. Software & Systems Modeling, 2010, 9(4): 473-492

[30] Mäder P, Gotel O, Philippow I. Getting back to basics: Promoting the use of a traceability information model in practice//Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering. Vancouver, Canada, 2009: 21-25

[31] Rempel P, Mäder P, Kuschke T, et al. Mind the gap: Assessing the conformance of software traceability to relevant guidelines//Proceedings of the 36th International Conference on Software Engineering. Hyderabad, India, 2014: 943-954

[32] Mader P, Jones P L, Zhang Y, et al. Strategic traceability for safety-critical projects. IEEE Software, 2013, 30(3): 58-66

[33] Zoughbi G, Briand L, Labiche Y. Modeling safety and airworthiness (RTCA DO-178B) information: Conceptual model and UML profile. Software & Systems Modeling, 2011, 10(3): 337-367

[34] Panesar-Walawege R K, Sabetzadeh M, Briand L, et al. Characterizing the chain of evidence for software safety cases: A conceptual model based on the IEC 61508 standard// Proceedings of the 2010 3rd International Conference on Software Testing, Verification and Validation. Paris, France, 2010: 335-344

[35] Bell R. Introduction to IEC 61508//Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software. Sydney, Australia, 2006: 3-12

[36] Antonino P O, Trapp M, Barbosa P, et al. The safety requirements decomposition pattern//Proceedings of the International Conference on Computer Safety, Reliability, and Security. Delft, The Netherlands, 2015: 269-282

[37] de Leon D C, Alves-Foss J. Hidden implementation dependencies in high assurance and critical computing systems. IEEE Transactions on Software Engineering, 2006, 32(10): 790-811

[38] Lee J, Kuo J Y. New approach to requirements trade-off analysis for complex systems. IEEE Transactions on Knowledge and Data Engineering, 1998, 10(4): 551-562

[39] Goknil A, Kurtev I, van den Berg K, et al. Semantics of trace relations in requirements models for consistency checking and inferencing. Software & Systems Modeling, 2011, 10(1): 31-54

[40] Goknil A, Kurtev I, van den Berg K. Change impact analysis based on formalization of trace relations for requirements// Proceedings of the 4th ECMFA Traceability Workshop. Berlin, Germany, 2008: 59-75

[41] Briand L, Falessi D, Nejati S, et al. Traceability and SysML design slices to support safety inspections: A controlled experiment. ACM Transactions on Software Engineering and Methodology, 2014, 23(1): 9

[42] Nejati S, Sabetzadeh M, Falessi D, et al. A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies. Information and Software Technology, 2012, 54(6): 569-590

[43] Kan S. Traceability and model checking to support safety requirement verification//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. Hong Kong, China, 2014: 783-786

[44] Cleland-Huang J, Gotel O, Zisman A. Software and Systems Traceability. London, UK: Springer, 2012

[45] Binns P, Englehart M, Jackson M, et al. Domain-specific software architectures for guidance, navigation and control. International Journal of Software Engineering and Knowledge Engineering, 1996, 6(2): 201-227

[46] HOOD User Group. HOOD Reference Manual Release 4.0. Paris, France: HUG, 1995

[47] Yang Zhi-Bin, Pi Lei, Hu Kai, et al. AADL: An architecture design and analysis language for complex embedded real-time systems. Journal of Software, 2010, 21(5): 899-915(in Chinese)
(杨志斌, 皮磊, 胡凯等. 复杂嵌入式实时系统体系结构设计与分析语言: AADL. 软件学报, 2010, 21(5): 899-915)

[48] Aerospace S A E. SAE AS5506 Annex behavior specification V1.6. New York, USA: SAE International, 2007

[49] Aerospace S A E. Architecture Analysis and Design Language (AADL) Annex Volume 1. Document Number: AS5506/1, New York, USA: SAE International, 2007

[50] Huth M, Ryan M. Logic in Computer Science: Modelling and Reasoning About Systems. Cambridge, UK: Cambridge University Press, 2004

[51] Ma Lin. FTA Based Avionics Software Safety Verification Methodology [M. S. dissertation]. Nanjing University of Aeronautics and Astronautics, Nanjing, 2012(in Chinese)
(马琳. 基于故障树的航电软件系统安全性验证方法研究[硕士学位论文]. 南京航空航天大学, 南京, 2012)

[52] Chen Z, Gu Y, Huang Z, et al. Model checking aircraft controller software: A case study. Software: Practice and Experience, 2015, 45(7): 989-1017

WANG Fei, born in 1990, Ph. D. candidate. His research interests include software engineering, formal methods, requirement traceability, modeling and analysis of embeddedsystem.

HUANG Zhi-Qiu, born in 1965, Ph. D., professor. His research interests include software engineering, formal methods and cloud computing.

YANG Zhi-Bin, born in 1982, Ph.D., associate professor. His research interests include real-time system and formal methods.

KAN Shuang-Long, born in 1988, Ph. D. candidate. His research interests include software engineering, model checking and formal methods.

SHEN Guo-Hua, born in 1976, Ph. D., associate professor. His research interests include requirement engineering, trustworthy software evaluation and software safety.

CHEN Guang-Ying, born in 1991, M. S. Her research interests include software engineering and formal methods.

Background

Requirements traceability has been identified has an important issue in safety-critical system. In aerospace, nuclear energy and transportation, the new standard for safety imposes a full traceability for requirements, their modeling and validation. As a consequence, requirements should be linked in a bi-directional way, with the design model and the validation activities. Current research on traceability has two main problems. On one hand, it mainly focuses on the traceability between requirements and source code, there is a lack of research about traceability between requirements or between requirements and design. On the other hand, traceability information built by major ways such as the recovery of trace relations based on information retrieval has low accuracy and completeness; it usually can't be effectively applied in the development of software in safety-critical areas.

In this paper, we propose a semantic model for the traceability information between requirements and design based on predicate logic, and describes the system requirements and software design by two widely-used standard languages SysML and AADL, respectively. Based on the semantic model, trace relations can be automatically inferred and checked, and the accuracy and completeness of relations can be guaranteed.

This work supported by the National High Technology Research and Development Program (863 Program) of China (2015AA105303), the National Defense Basic Scientific Research Project under Grant of China (JCKY2016203B011), the National Natural Science Foundation of China (61502231). The 863 Program and the NSFC projects were researched on the theory, application and formal methods on the safety-critical domain. The team has published several research articles about safety of embedded software, the theory of formalization method and requirements traceability. The research aims to provide the theoretical basis for safety requirements traceability and the technical support for software safety analysis, which will enhance the implementation of the safety requirement traceability methods and technologies in safety-critical software development and technique support of safety analysis, extend the application of traceability method and technique in safety-critical software development.