软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(8):2350−2370 [doi: 10.13328/j.cnki.jos.005530]

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

基于限定自然语言需求模板的 AADL 模型生成方法*

王 飞[1], 杨志斌[1,2,3], 黄志球[1,2,3], 周 勇[1], 刘承威[1], 章文炳[1], 薛 垒[4], 许金森[1]

[1](南京航空航天大学 计算机科学与技术学院,江苏 南京 211106)

[2](高安全系统的软件开发与验证技术工业和信息化部重点实验室(南京航空航天大学),江苏 南京 211106)

[3](软件新技术与产业化协同创新中心,江苏 南京 210093)

[4](上海航天电子技术研究所,上海 201109)

通讯作者: 杨志斌, E-mail: yangzhibin168@163.com

摘 要: 随着嵌入式软件系统在汽车、核工业、航空、航天等安全关键领域的广泛应用,其失效将会导致财产的损失、环境的破坏甚至人员的伤亡,使得保障软件安全性成为系统开发过程中的重要部分.传统的安全性分析方法主要应用在软件的需求分析阶段和设计阶段,然而需求与设计之间的鸿沟却一直是软件工程领域的一大难题.正是由于这一鸿沟的存在,使得需求分析阶段的安全性分析结果难以完整而详尽地反映在软件设计中.其根本原因是:当前的软件需求主要通过自然语言描述,存在二义性与模糊性,且难以进行自动化处理.为了解决这一问题,面向构件化嵌入式软件,首先提出了一种半结构化的限定自然语言需求模板用于需求规约,能够有效地降低自然语言需求的二义性与模糊性;然后,为了降低自动化处理的复杂性,采用需求抽象语法图作为中间模型,实现基于限定自然语言需求模板规约的软件需求与 AADL 模型之间的转换,并在此过程中自动记录两者之间的可追踪关系;最后,基于 AADL 开源工具 OSATE 对所提出方法进行了插件实现,并通过航天器导航、制导与控制系统(guidance, navigation and control,简称 GNC)进行了实例性验证.

关键词: 嵌入式软件;软件安全性;需求规约;限定自然语言需求模板;AADL;可追踪性

中图法分类号: TP311

中文引用格式: 王飞,杨志斌,黄志球,周勇,刘承威,章文炳,薛垒,许金森.基于限定自然语言需求模板的 AADL 模型生成方法.软件学报,2018,29(8):2350−2370. http://www.jos.org.cn/1000-9825/5530.htm

英文引用格式: Wang F, Yang ZB, Huang ZQ, Zhou Y, Liu CW, Zhang WB, Xue L, Xu JM. Approach for generating AADL model based on restricted natural language requirement template. Ruan Jian Xue Bao/Journal of Software, 2018,29(8):2350−2370 (in Chinese). http://www.jos.org.cn/1000-9825/5530.htm

Approach for Generating AADL Model Based on Restricted Natural Language Requirement Template

WANG Fei[1], YANG Zhi-Bin[1,2,3], HUANG Zhi-Qiu[1,2,3], ZHOU Yong[1], LIU Cheng-Wei[1], ZHANG Wen-Bing[1], XUE Lei[4], XU Jin-Miao[1]

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

* 基金项目: 国家自然科学基金(61502231, 61272083); 国家高技术发展计划(863)(2015AA105303); GF 基础科研重点项目(JCKY2016203B011); 国家重点研发计划(2016YFB1000802); 江苏省自然科学基金(BK20150753); 软件开发环境国家重点实验室开放课题(SKLSDE-2015KF-04); 航空科学基金(2015ZC52027)

Foundation item: National Natural Science Foundation of China (61502231, 61272083); National High-Tech R&D Program of China (863) (2015AA105303); National Defense Basic Scientific Research Project of China (JCKY2016203B011); National Key Research and Development Program of China (2016YFB1000802); Natural Science Foundation of Jiangsu Province (BK20150753); Project of the State Key Laboratory of Software Development Environment of China (SKLSDE-2015KF-04); Avionics Science Foundation of China (2015ZC52027)

本文由数据驱动的软件智能化开发方法与技术专题特约编辑谢冰教授、魏峻研究员、彭鑫教授、孙海龙副教授推荐.

收稿时间: 2017-07-18; 修改时间: 2017-09-28, 2017-12-22, 2018-01-12; 采用时间: 2018-01-24; jos 在线出版时间: 2018-03-13

CNKI 网络优先出版: 2018-03-13 17:30:44, http://kns.cnki.net/kcms/detail/11.2560.TP.20180313.1730.013.html

[2](Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology, Nanjing 211106, China)

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

[4](Shanghai Aerospace Electronic Technology Institute, Shanghai 201109, China)

Abstract: As embedded software systems are widely used in many crucial areas such as automotive, energy industries and aerospace, failures of these systems will cause pollution of environment, property losses and even casualties. Therefore, safety analysis has been critical for developing these systems. The traditional safety analysis method is mainly used in the software requirement analysis stage and the design stage. However, the gap between requirement and design is a challenge in software engineering area, for it is difficult to transmit and reflect the analysis result of the requirement analysis stage into software designing. The primary reason is that the current software requirement is mainly described in natural language, in which there is ambiguity and fuzziness, and that makes it difficult to be automatically processed. To solve this problem, this paper first focuses on component embedded software and proposes a set of requirement template based on restricted natural language to reduce the ambiguity and fuzziness of natural language requirements. Then, to lessen the complexity of automated processing, requirement abstract syntax diagrams are used as the intermediate model to realize the transition between software requirement specified by restricted natural language template and AADL model, and automatically record the traceability relations between them. Finally, a tool for the method proposed above is developed based on the AADL open source system OSATE, and an example validation is carried out through the spacecraft guidance, navigation and control system GNC (guidance, navigation and control).

Key words: embedded software; software safety; requirement specification; restricted natural language requirement template; AADL; traceability

随着嵌入式软件在航空电子、汽车工业、通信、核工业等安全关键领域的广泛应用,软件的失效可能造成财产的损失、环境的破坏甚至人员的伤亡[1].近年来,由于软件问题引发的事故屡见不鲜,例如,2007 年 2 月 9 日,前往日本的 F-22 型“猛禽”隐形战斗机在经过日期变更线时,由于战机发生软件问题,造成导航系统失灵,半途折回[2];2009 年,一架土耳其航空公司波音 737-800 型飞机在机场跑道进近期间,由于软件故障导致飞机坠毁[3];2010 年,由于软件问题引起的“召回门”事件,直接造成了日本丰田公司 20 亿美元的经济损失[4].可见,软件系统对嵌入式系统安全性的影响逐渐占据统治地位,软件控制系统的故障和安全性问题已成为发生事故的重要原因.保障嵌入式软件的安全性成为当前软件工程研究领域一个非常重要的课题.

软件安全性是指软件运行不引起系统危害的能力[5].软件作为系统的重要组成要素,不会直接危及生命、财产和环境的安全,但借助软件实现的人机交互,却可能因软件失效造成人员误操作从而形成危害.如对于无人机交互的嵌入式安全关键系统而言,可能因为软件错误地控制系统而造成灾难性的后果.以机载软件为例,飞机低于特定高度时,着陆轮控制系统必须控制机载系统放下着陆轮,否则容易造成飞机坠毁.当软件可以引起危害或者控制危害发生,则该软件是危险的[6].

软件安全性分析(safety analysis)[7]是保障软件安全性的一种广泛接受的方法.当前,针对软件的安全性分析工作主要集中在软件需求规约和软件设计阶段.Leveson 认为,软件的设计缺乏远见以及规约错误是影响安全性的最主要原因[8].当前,软件需求主要采用自然语言描述,存在二义性和模糊性以及不完整性,且难以进行自动化分析处理,使得软件设计主要依赖设计人员对自然语言需求的理解.这也是模型驱动开发方法(model-driven development,简称 MDD)一般都是从软件的分析模型或者设计模型开始而非需求模型的根本原因.正是由于自然语言需求不可避免地存在二义性和模糊性,才会使得软件设计模型会因设计人员对需求理解的不一致而导致偏差,进而导致软件的安全性问题.

本文针对嵌入式软件需求与设计之间的这一鸿沟展开研究.为了消除自然语言需求二义性、模糊性且难以自动化分析处理的缺点,在充分分析国内外相关研究以及当前工业界嵌入式软件需求描述方式的基础上,针对安全关键领域嵌入式软件需求的基本特征,提出了一种基于限定自然语言需求模板的需求描述方法,可以有效地降低自然语言需求的二义性与模糊性.

针对嵌入式系统软硬件紧密耦合的特性,本文将采用复杂嵌入式实时系统的体系结构设计与分析语言标准——AADL(architecture analysis and design language)[9,10]进行软件设计.AADL 是由美国汽车工程师协会 SAE

(Society of Automotive Engineers)于 2004 年在 MetaH、UML 的基础上提出的一种嵌入式系统体系结构分析与设计语言,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.CMU(Carnegie Mellon University)、MIT(Massachusetts Institute of Technology)、UIUC(University Of Illinois at Urbana-Champaign)、Pennsylvania 大学、NASA 以及法国 IRIT(Toulouse Institute of Computer Science Research)、INRIA、Verimag 等研究机构对 AADL 展开了深入研究与扩展[11].TOPCASED[12]、SPICES[13]、AVSI-SAVI[14]等是欧美工业界和学术界共同参与的 AADL 重大研究项目,涉及 AADL 标准扩展、建模工具、形式语义、验证、靠性分析、可调度分析以及自动代码生成等方面的研究,这些也是目前 AADL 的研究热点.

AADL 被认为是基于模型驱动的嵌入式系统设计与实现的基础,但是 AADL 却缺乏对需求的表达,其模型设计完全依赖工程师对需求文档的理解,因理解的差异性很容易造成设计人员构建的 AADL 模型并不能正确或不能完整地反映软件需求.此外,在这种情形下,由于需求规约与软件设计是两个相对独立的过程,所建立与维护的需求追踪信息通常是不完整的或者不正确的,使得后期的安全性分析与系统维护困难较大.针对这一问题,本文在结构化需求规约的基础上设计了一套转换规则和转换算法,自动化将结构化的模板需求转换为 AADL 设计模型.同时,自动建立需求与 AADL 构件之间的可追踪性关系.

本文第 1 节给出基于限定自然语言需求模板的 AADL 模型生成方法的研究框架.第 2 节对限定自然语言需求模板进行详细论述.第 3 节描述基于限定自然语言需求模板的 AADL 模型及需求可追踪性关系的自动生成方法.第 4 节进行工具实现,并以典型应用案例说明本文所提出方法的有效性.第 5 节对国内外的相关研究工作进行相应概述.最后,在第 6 节进行全文总结与未来研究工作展望.

1 研究框架

本文针对安全关键嵌入式软件需求与设计之间的鸿沟展开研究,提出了一种基于限定自然语言需求模板的需求规约方法,并通过转换规则和转换算法自动生成了初始 AADL 设计模型;同时,自动建立软件需求与设计间的可追踪关系.本文的研究框架如图 1 所示.

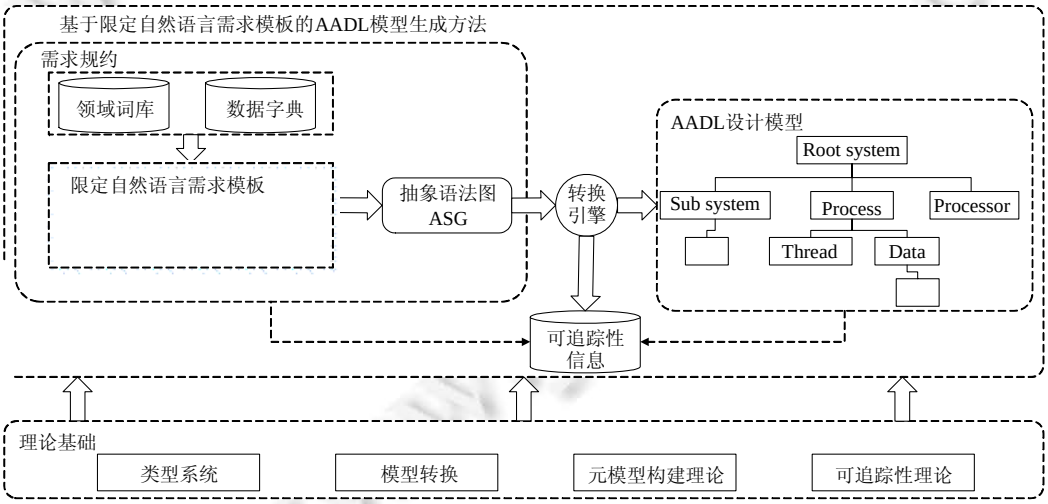


Fig.1 Framework of generating AADL model based on restricted natural language requirement template

图 1 基于限定自然语言需求模板的 AADL 模型生成方法研究框架

图 1 上半部分为本文的主要研究内容与研究思路,主要包括两个部分:基于限定自然语言需求模板的需求规约以及 AADL 设计模型和需求追踪关系的自动化生成.具体而言,基于限定自然语言需求规约部分主要包括以下工作.

(1) 依据领域相关知识和具体工程需求完成数据字典和领域词库的制定.其中,数据字典是将散布在需求文档中的各种数据都集中起来,并且采用比较规范的方式对它们的属性进行描述.其目的是对数据进行全面描述,即,在文档中描述不够的属性在这里都可以得到补充;同时,通过集中和有序的描述,使得使用方便.数据字典主要包括静态数据、输入数据、输出数据,其中:静态数据是指可以描述软件中的常数、预先装订的参数,这里的装订参数可以看成是静态输入的数据;输入数据是指描述软件在运行过程中各组件的输入数据,即动态输入的数据;输出数据是指描述软件在运行过程中各组件的输出数据.此外,针对机载软件系统对系统安全性、可靠性的要求,在数据词典中将增加对故障类型描述的数据,如对故障触发事件、故障类型等,以及参照 ARP4761 和 DO178B 等行业相关标准定义的危险属性集.领域词库则主要是针对实际项目领域需求,将需求中所需的各种对象名词集中起来,形成一份参照列表,如专有的系统名、组件名、特殊硬件设备、芯片、模式变迁状态等.通过在领域词库中注册使用的词汇来实现对需求文档编写的限定,以减少在需求编写过程中出现人为编写错误的可能.

(2) 在数据字典和领域词库的基础上,按照设定的限定自然语言需求模板完成具体的软件需求规约,在此过程中,为了消除自然语言的二义性,必须对自然语言的表达能力进行适当限制,也即必须遵循一定的语法约束规则.

(3) 在通过限定自然语言需求模板完成整个软件系统需求规约之后,为了增强其自动化分析与处理能力以及促进下一阶段的模型转换工作,本文依据需求模板的层次结构关系将一组限定自然语言需求模板描述的软件系统需求转换到中间模型——抽象语法图(abstract syntax graph,简称 ASG).

初始 AADL 设计模型和需求追踪关系的自动化生成则主要通过转换引擎实现,然后在初始 AADL 模型的基础上,通过人工精化的方式逐步完善功能与非功能属性,进而形成完整的设计模型.同时,在精化过程中还需要完善需求追踪信息,建立完整的需求追踪关系,为软件安全性分析提供支撑,同时为后期的软件维护与演化奠定基础.

2 基于限定自然语言需求模板的需求规约方法

随着嵌入式软件系统功能不断加强,软件规模和复杂性迅速提高.以航空软件为例,最新的第四代战机的机载软件包含了约 1 000 万行的代码,耗费了 60%以上的人力和开发费用,提供了约 80%的控制功能,涵盖了导航系统、雷达系统、飞行控制系统、动力控制系统等[15].为了适应嵌入式软件系统规模和复杂度的提高,嵌入式系统通常是由多个计算子系统构成,其软件系统具有较高的构件化特征,大多采用构件化分布式架构[16,17].一个嵌入式系统通常都会包含不同领域的计算模型,这些不同领域的计算模型通过系统中不同的部件来实现,相互之间通过接口来进行通信交互.由于硬件模块设计和开发的成本远比软件模块昂贵,因此常用软件模块替代硬件模块来实现一些复杂的任务,使得构件化软件开发方法成为当前软件工程领域的主流开发方法[18,19].

在构件化软件开发方法中,一个完整的系统被看作是若干独立部分,每个独立部分都按照一定的标准封装成构件(component),通过组装这些标准化构件,就可以开发出提供特定功能的软件产品[20,21].航空嵌入式软件就是典型的构件化分布式架构[22,23],通常包括导航软件、火控软件、大气数据软件、飞行控制软件、任务管理软件、雷达信号/数据处理软件、显示控制处理软件和通用综合软件等,这些软件都封装为独立的模块,通过接口之间的消息交互通信完成整个软件的功能.

正是基于嵌入式软件的这种特性,本文在借鉴航天嵌入式软件需求描述方式的基础上,同时综合分析了 RUCM(restricted use case modeling)[24,25],SPARDL[26],EARS(easy approach for requirements syntax)[27]等需求表达方式,从而面向构件化软件开发方法提出了基于限定自然语言需求模板的需求规约方法,能够尽量不改变工程师熟悉的基于文本的软件需求撰写习惯,同时降低自然语言需求存在的二义性与模糊性.

2.1 限定自然语言需求模板

通常,软件需求中没有或者较少包含设计相关元素,从而使得设计模型必须通过人工构建.基于此,本文在进行自然语言需求模板设计时,除了考虑嵌入式软件构件化特征之外,还在需求模板中添加了部分必要的设计相关元素,用于支持后期设计模型的自动化生成.具体而言,依照系统分解的基本思想,需求模板主要分为系统、子系统、任务、子任务这 4 个层次.对于一个复杂的嵌入式软件系统而言,首先依据依据其功能结构进行层次划分,然后分别通过相应层次的需求模板进行需求规约.此外,对于任务和子任务中一些具有共性的功能单元,应当将其作为“共享功能模块”置于顶层系统需求模板之下,供后续的任务和子任务模板调用.具体的需求模板见表 1.

Table 1 Restricted natural language requirements template

表 1 限定自然语言需求模板

ID		每个需求模板都具有唯一的标识
名称		每个需求模板描述功能单元(系统/子系统/任务/子任务)的名称
输入	数据	描述功能单元的输入数据
	事件	描述功能单元的输入事件
输出	数据	描述功能单元的输出数据
	事件	描述功能单元的输出事件
组成	子系统	一个复杂系统可以分解为若干子系统
	任务	每个系统/子系统需要完成若干任务
	子任务	一个复杂任务可以分解为若干子任务(原子任务)
功能需求		描述每个功能单元在功能方面的需求
性能需求		描述每个功能单元在性能方面的需求
接口需求		描述每个功能单元在接口方面的需求
设计约束		描述每个功能单元设计时对硬件的要求

对于共享功能模块而言,仅仅需要对其进行功能需求、性能需求等方面的描述,而不包含输入、输出和组成元素.同时,由于自然语言表达存在二义性,针对需求模板中的不同类型需求,需要制定相应的约束规则对其表达方式进行限定.其中,最基本的通用约束规则见表 2.

Table 2 Common restriction rules for natural language

表 2 通用的自然语言约束规则

约束规则	描述
R1	句子的主语必须是系统/子系统/任务/子任务
R2	只使用陈述句
R3	只使用现在时
R4	用主动语态而不用被动语态
R5	不使用情态动词(如大概)、代词以及表示否定含义的副词/形容词
R6	只使用简单句
R7	准确的描述(子)系统/(子)任务间的交互,主语和宾语都不能丢失
R8	不要使用分词短语作状语修饰词
R9	以一致的方式使用词语,使用固定的名词来描述某个事物

下面将分别就表 1 中的各项元素进行具体介绍,同时,对其约束规则进行简单说明.

(1) ID 是每一个需求模板的编号,依据系统、子系统、任务、子任务这 4 个层次的不同,其前缀分别为 SR, SSR,TR,STR,后缀为自然数按序增加.

(2) 模板名称即每个系统/子系统/任务/子任务的名称,限定为名词或多个名词的组合,尽可能直观地体现系统/子系统/任务/子任务的功能.

(3) 输入与输出:当前的嵌入式系统多为反应式系统,这类系统会不断地和外部环境进行交互,即不断地从外部环境中得到输入,计算并输出给环境.因此,输入与输出分别用于描述每个系统/子系统/任务/子任务从环境中获取的输入以及经计算处理后给系统外部环境的输出.外部环境包括被系统控制的物

理设备、人或其他反应式系统.一般而言,输入与输出主要分为数据和事件两种类型,分别逐条枚举,如果没有则填 NONE.

(4) 组成主要用于进行软件系统的层次划分,将一个复杂的嵌入式软件系统依据功能独立性可以划分为若干子系统,依据系统的复杂程度,每个子系统又可依据自身的功能特性再次划分子系统.每个子系统最终需要分解为若干具体任务,依据复杂程度,任务亦可再次划分为子任务,子任务将不予再次划分.如果没有则填 NONE.

(5) 功能需求描述的是每个系统/子系统/任务/子任务的功能性需求的描述,主要是针对系统的软硬件之间的数据交互进行描述,如数据的发送接收等.本文规定了一些特殊句式用来描述组件间的数据通信关系,如 *A* 发送 *B* 数据/事件到 *C*,其中的 *A* 和 *C* 是在领域词库中“注册”过的专有名词,可以是系统的某个组件,或者一些硬件外设;而 *B* 是在数据词典中记录过的专有的数据名称,数据词典中要记录该数据的组成、类型、精度等信息.

(6) 性能需求描述的是每个系统/子系统/任务/子任务需要满足的一些定量要求,主要针对周期性、实时性等.对此,本文也规定了一些常用的描述性能需求的特殊句式,如该组件的周期为 *AB*,其中:*A* 为具体的数量;*B* 为提前注册过的对周期描述的具体单位,包括 hr,sec,min,ms 等.

(7) 接口需求描述的是每个系统/子系统/任务/子任务的接口要求,如接口数据传输协议等.

(8) 设计约束描述的是每个系统/子系统/任务/子任务对硬件的要求,如使用的设备型号、处理器、存储器类型等,如,设定 CPU 与某子系统通过 1553B 总线数据格式进行数据传递等.其中,CPU 和该子系统需在领域词库中记录,1553B 总线数据格式则需要在数据字典中规定其特殊的数据格式.

姿态与轨道控制系统(attitude and orbit control system,简称 AOCS)是导航、制导与控制系统(即 GNC 系统)的一个子系统,AOCS 通过收集和处理各种传感器的测量数据来完成制导和控制任务,主要执行轨道确定、轨道控制、姿态确定、姿态控制等功能,同时负责与其他分系统进行交互,AOCS 一般采用双机备份的方式来提高可靠性;执行机构包括反作用飞轮、喷嘴、轨控发动机等,反作用飞轮和喷嘴用来控制姿态,而轨控发动机则用来执行轨道机动和修正.下面即以 AOCS 系统为例,使用需求模板进行需求建模,AOCS 系统层次的需求规约见表 3.

Table 3 Description of attitude and orbit control system based on template

表 3 基于模板的姿态与轨道控制系统需求描述

ID		SSR1
名称		姿态与轨道控制系统
输入	数据	传感器数据采集响应 陀螺仪数据采集响应 导航相机数据采集响应 加速计数据采集响应
	事件	NONE
输出	数据	传感器数据采集指令 陀螺仪数据采集指令 导航相机数据采集指令 加速计数据采集指令
	事件	NONE
组成	子系统	NONE
	任务	控制任务 遥测控制任务 系统监控任务 空闲任务
	子任务	NONE

Table 3 Description of attitude and orbit control system based on template (Contiuned)

表 3 基于模板的姿态与轨道控制系统需求描述(续)

ID	SSR1
功能需求	姿态与轨道控制系统发送传感器数据采集响应到控制任务 姿态与轨道控制系统发送陀螺仪数据采集响应到控制任务 姿态与轨道控制系统发送导航相机数据采集响应到控制任务 姿态与轨道控制系统发送加速计数据采集响应到控制任务 姿态与轨道控制系统接收来自控制任务的传感器数据采集指令 姿态与轨道控制系统接收来自控制任务的陀螺仪数据采集指令 姿态与轨道控制系统接收来自控制任务的导航相机数据采集指令 姿态与轨道控制系统接收来自控制任务的加速计数据采集指令 控制任务发送姿态控制命令 1 到反作用飞轮 控制任务发送姿态控制命令 1 到喷嘴 控制任务发送轨道控制命令到轨控发动机
性能需求	NONE
接口需求	NONE
设计约束	反作用飞轮,S_FlyWheel,设备 喷嘴,S_Attitude_Nozzle,设备 轨控发动机,S_Orbit_Engine,设备 AOCS 总线构件,AOCS_LAN,总线 AOCS 存储器构件,AOCS_Mem,存储器 AOCS 处理器构件,AOCS_Proc,处理器

2.2 需求抽象语法图

基于限定自然语言需求模板可以在尽量不改变需求编写习惯的前提下实现嵌入式软件需求规约,同时能够有效降低自然语言需求的二义性与模糊性.但是对于一个复杂的嵌入式软件系统而言,通常需要经过多次划分子系统、任务以及子任务,大量的需求模板使得自动化处理变得异常复杂.因此,本文采用抽象语法图作为中间模型,首先将一组限定自然语言需求模板描述的软件需求转换到抽象语法图,然后通过模型转换的方式实现抽象语法图到 AADL 模型的转换.

依据限定自然语言需求模板的主要元素及其元素之间的关系,可以获取对应的需求抽象语法图的元模型,如图 2 所示.

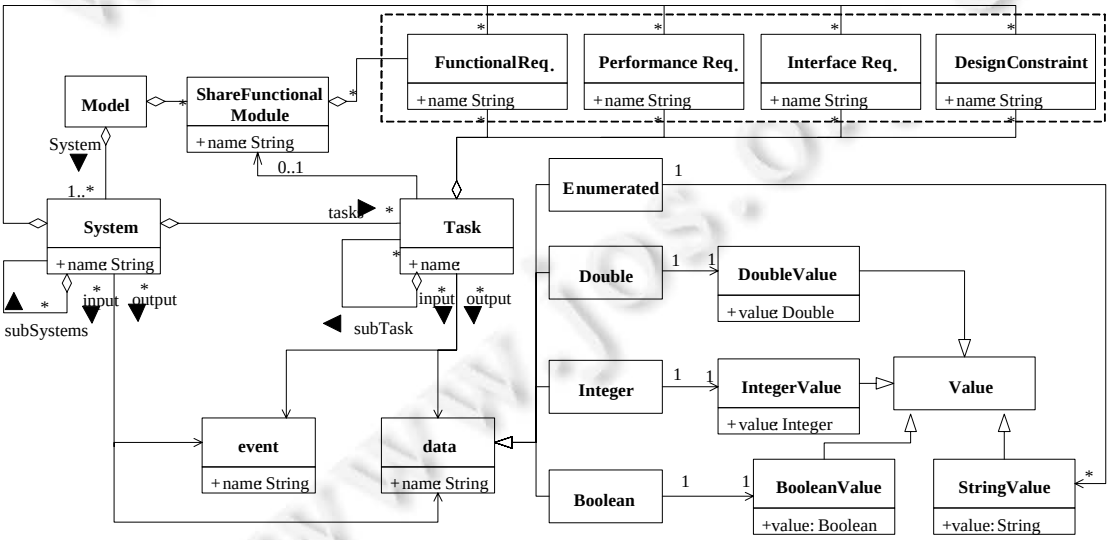


Fig.2 Requirement abstract systax diagram meta-model

图 2 需求抽象语法图元模型

在图2所示的元模型中,Model描述的是整个嵌入式软件系统的最顶层系统,它由一个或多个功能相对独立的子系统组成,也即图2中的System实质上描述的是子系统的概念,而依据子系统规模以及复杂性的不同,可以进行多层子系统划分.此外,对于每个(子)系统而言,又可分解为若干任务,任务依据其复杂性可以进一步划分子任务.本文规定,子任务作为原子任务将不予再次划分,也即仅允许进行一次子任务划分.因此,为了降低任务与子任务的复杂程度,应当在进行子系统划分时尽量避免子系统功能过于庞大.ShareFunctionalModule 描述的是任务或子任务中的公共功能模块,这部分内容可以独立(子)任务需求模板单独予以描述,本文规定,每个任务或子任务至多能调用 1 个共享功能模块.此外,对于每个(子)系统、(子)任务而言,都有相应的输入输出数据(data)和事件(event),同时,每个(子)系统、(子)任务还需对其功能需求(functional requirement)、接口需求(interface requirement)、设计约束(design constraint)、性能需求(performance requirement)进行描述.

此外,对于虚线框中功能需求、接口需求、性能需求、设计约束的约束规则在图2所示的元模型中并未体现,而且此部分内容的自动化转换较为复杂,因此本文仅仅考虑功能需求中的数据交互以及性能需求中的周期性等的自动化转化.至于其他属性则暂时不予考虑,而是在生成的初始的设计模型中通过人工精化的方式将此部分内容逐步加入到设计模型中.图 3 给出了表 3 所示的 AOCS 系统需求模板对应的抽象语法图.为了简化显示效果,对于未进行自动转换的元素则没有进行显示.

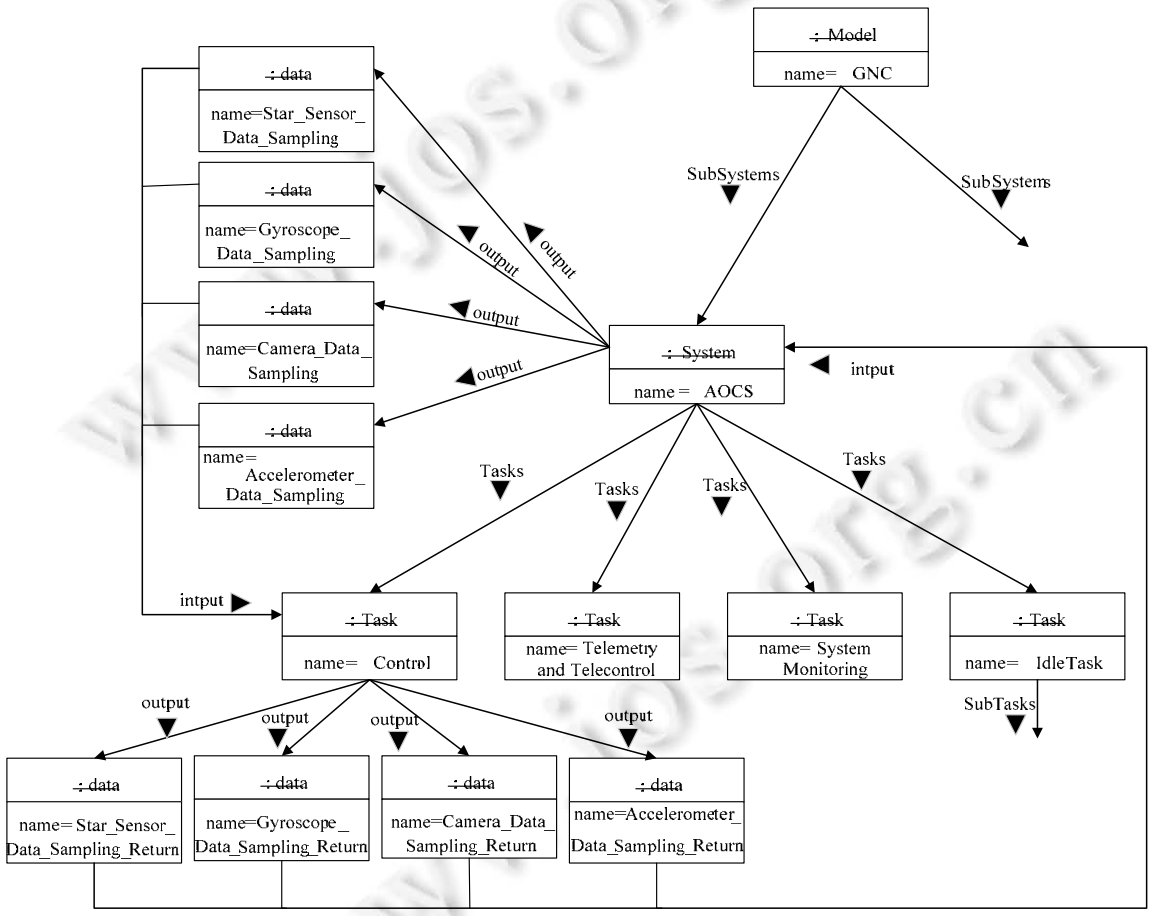


Fig.3 Abstract systax diagram for requirement template in Table 3

图 3 表 3 所示需求模板对应的抽象语法图

3 转换规则与可追踪性

限定自然语言需求模板对自然语言的表达能力进行了适当限制,有效地降低了自然语言需求的二义性与模糊性,中间模型需求抽象语法图进一步将表格化的文本需求转换成了图状结构的需求模型,增强了需求的自动化处理能力.考虑到嵌入式系统软硬件紧密耦合的特性,在进行软件设计的同时,应当具备足够的硬件表达能力.AADL 作为一种标准的嵌入式系统体系结构设计与分析语言,能够很好地支持嵌入式系统软硬件协同设计.基于此,本节将进一步关注基于需求抽象语法图的 AADL 模型自动化生成方法以及需求可追踪性的研究.下面首先对 AADL 进行相关介绍.

3.1 AADL简介

AADL 通过构件、连接等概念描述系统的软、硬件体系结构;通过特征、属性描述系统功能与非功能性质;通过模式变换(mode transition)描述运行时体系结构演化;通过用户定义属性和附件支持可扩展;对于复杂系统建模,AADL 通过包(package)进行组织.AADL 提供了 3 种建模方式:文本、XML 以及图形化.

构件是 AADL 语言的基本建模单元,并分为软件构件、硬件构件以及系统构件这 3 类构件.每个构件都有名称(identifier)和所属的种类(category).一个完整的构件定义包括构件类型(type)和构件实现(implementation)两部分.具体而言,软件构件用于软件体系结构建模,包括数据(data)、线程(thread)、线程组(thread group)、进程(process)、子程序(subprogram)构件;执行平台构件用于硬件体系结构建模,包括处理器(processor)、虚拟处理器(virtual processor)、存储器(memory)、总线(bus)、虚拟总线(virtual bus)、外设(device)构件.系统构件组合所有的构件,层次化地建立系统的体系结构.

除了使用以上构件和连接来表示系统的软、硬件体系结构,AADL 还使用执行模型(execution model)[28,29]来描述系统的运行时环境,用于管理和支持构件的执行,这也是 AADL 语言的一个非常重要的概念.

首先,AADL 通过定义分发协议、通信机制、调度策略以及模式变换协议等属性来描述系统运行时环境的基本特征,这些属性依附于各种构件、连接、端口以及模式等建模元素.AADL 在“预定义属性集”附录中给出了所有执行模型属性.

- 线程分发协议: Dispatch_Protocol: **enumeration** (Periodic, Aperiodic, Sporadic, Timed, Hybrid, Background) **applies to** (thread);
- 数据端口通信机制:Timing: **enumeration** (sampled, immediate, delayed) **applies to** (port);
- 调度策略:Scheduling_Protocol: **enumeration** (FixedTimeline, Cooperative, RMS, EDF, SporadicServer, SlackServer, ARINC653) **applies to** (processor, virtual processor);
- 模式变换协议:Mode_Transition_Response: **enumeration** (emergency, planned) **applies to** (mode transition).

其次,AADL 还定义了这些执行模型属性的语义,主要包括构件的执行状态和动作、通信时间语义、调度语义、模式变换的时间语义等,它们被称为 AADL 语言的执行语义(execution semantics),也称为动态语义.

因此,从语法层面上,执行模型通过属性来刻画运行时环境的基本特征;从语义层面上,执行模型定义了 AADL 语言的执行语义,而且通过这些执行语义来确保运行时行为的确定性和可预见性.

为了适应更多的建模需求,AADL 提供了扩展附件(annex)机制,扩展附件具有独立的语法和语义.已有的扩展附件主要有 Graphical AADL Notation Annex[30]、AADL Meta Model and XML/XMI Interchange Format Annex[30]、Language Compliance and Application Program Interface Annex[30]、Error Model Annex[30,31]、Behavior Annex[32]、Data Modeling Annex[33]以及 ARINC653 Annex[33,34].

3.2 转换规则

本文所提的嵌入式软件需求规约方法,在需求规约阶段主要包括 3 部分内容,分别是数据字典、领域词库以及限定自然语言需求模板.AADL 模型自动化生成时,数据字典部分将会转换到 AADL 模型中的全局共享数据构件(data component),转换规则比较简单且单一,本文将不予描述;领域词库主要用于记录整个软件系统中的

各种对象名词供需求模板填写时使用,因此,这部分不需单独转换.接来下,将重点描述限定自然语言需求模板到 AADL 模型的转换规则.

基于限定自然语言需求模板的需求抽象语法图主要包含以下元素:顶层系统 Model、系统 System、任务 Task、共享功能模块 ShareFunctionalModule、输入/输出数据(data)和事件(event)、功能需求 Functional Req.、性能需求 Performance Req.、接口需求 Interface Req.以及设计约束 Design Constraint.本文将重点关注初始的 AADL 模型结构的自动化生成,因此,本节将主要考虑系统、任务、输入/输出等元素的转换,功能需求、性能需求、接口需求以及设计约束将用于辅助 AADL 模型的生成.具体的转换规则如下.

(1) Rule 1:ASG 中的顶层概念 Model 转换到 AADL 模型的 System 构件.

需求抽象语法图中的 Model 元素是指嵌入式软件系统需求模型中的最顶层系统,可以分解为若干子系统,因此将其转换为 AADL 模型的最顶层 System 构件,包括构件类型和构件实现两部分.

(2) Rule 2:ASG 中的 System 转换到 AADL 模型的 System 构件.

需求抽象语法图中的 System 元素对应的是需求模板层次划分中的子系统,同样可以将其转换到 AADL 模型中的 System 构件,但该 System 构件必须位于其上层 System 构件的 Implementation 下的 subcomponents 中.依据需求模型中子系统的层次结构,在生成的 AADL 模型中同样具备多层子系统结构.

(3) Rule 3:ASG 中的 Task 转换到 AADL 模型的 Process 构件.

需求抽象语法图中的 Task 是指具体功能执行单元,可以作为系统和子系统的子单元,因此在生成 AADL 时,Task 将转换为进程构件 Process.

(4) Rule 4:ASG 中的 SubTask 转换到 AADL 模型的 Thread 构件.

需求抽象语法图中的 SubTask 是需求模型中的最小功能单元,一般对应的是一个原子功能.因此在生成 AADL 设计模型时,SubTask 将转换为线程构件 Thread.在 AADL 中线程通常分为周期性线程、非周期性线程、偶发线程等类型,本文将主要关注周期性线程和非周期线程.其中,

- 周期性线程被运行时系统的分派器(dispatcher)周期性触发,即每到截止时间(deadline)即被触发;
- 非周期性线程由事件、事件数据的到来或发生子程序的调用而被触发,对于触发条件的到达时间没有约束要求,属于软时限(soft deadline)的触发.

线程的类型及其相关属性将从需求模型中的 Performance Req.获得.如果 Performance Req.中对周期性进行了相关描述,则转换到 AADL 周期性线程,并在 properties 之中描述具体的周期;否则,转换到非周期性线程.

(5) Rule 5:ASG 中的 ShareFunctionalModule 转换到 AADL 模型的 Subprogram 构件.

需求抽象语法图中的 ShareFunctionalModule 是在层次化需求模板的基础上抽取出来的“共享”功能模块,当在某个任务或子任务需要使用该功能模块时,可在相应的(子)任务需求模板进行调用,因此可以将其转换到 AADL 中的子程序,而其对应的调用则转换到 AADL 子程序调用.在 AADL 中,一个 subprogram 表示可顺序执行的可调用单元,可通过参数传递和子程序调用实现数据的传送,并且子程序可以要求数据访问进行数据交互.

(6) Rule 6:ASG 中每个 System/Task/SubTask 的输入输出数据/事件转换到 AADL 模型 feature 中的端口(Port).

需求抽象语法图中的数据 data 和事件 event 描述了每个系统/子系统/任务/子任务的输入与输出.将其转换为 AADL 设计模型时,主要生成 AADL 模型 feature 中的端口.在 AADL 中,端口是构件之间的逻辑连接点,用于数据和控制信息的传递.端口是有向的,分为 in 和 out 两种.端口又分为事件端口、数据端口和事件数据端口等 3 类,本文仅考虑前两种.其中,事件端口用于事件和警告(alarm)的传递,数据端口用于传递状态数据(如传感器数据流).

- Rule 6_1:ASG 中的 data 元素则转换到 AADL 模型 feature 中的 data port.
  - Rule 6_1_1:如果 ASG 中的 data 元素为输入类型,则转换到 AADL 模型 feature 中的 in data port;
  - Rule 6_1_2:如果 ASG 中的 data 元素为输出类型,则转换到 AADL 模型 feature 中的 out data port;
- Rule 6_2:ASG 中的 event 元素转换到 AADL 模型 feature 中的 event port.

- Rule 6_2_1:如果 ASG 中的 event 元素为输入类型,则转换到 AADL 模型 feature 中的 in event port;
- Rule 6_2_2:如果 ASG 中的 event 元素为输出类型,则转换到 AADL 模型 feature 中的 out event port.

(7) Rule 7:ASG 中 Model/System/Task/SubTask 功能需求(Functional Req.)中的数据交互转换到 AADL 模型端口间的连接(connection).

需求抽象语法图中的 Functional Req.描述的是对应 Model/System/Task/SubTask 的功能需求,其中,对于嵌入式软件系统而言,功能需求中最为核心部分就是数据交互关系.对于 AADL 而言,构件之间的交互行为主要通过连接(connection)来描述,与构件特征对应,AADL 支持端口连接、参数连接及访问连接这 3 种连接方式.其中,端口连接正好用于描述并发执行构件之间的数据与控制交互.因此,在生成 AADL 模型时,可以将其转换为端口间的连接.端口连接的格式如下所示:

name: [descriptor][source port][connection symbol][destination port].

依据上述转换规则,本文描述的需求模板元素与 AADL 构件间的对应关系以及相应的 AADL 描述见表 4.

Table 4 Mapping between element of requirement template and AADL component and the corresponding of AADL description

表 4 需求模板元素与 AADL 构件间的对应关系及相应的 AADL 描述

需求模板元素		AADL 构件	AADL 构件描述
Model		System 构件	system systemTypeID end systemTypeID; system implementation systemTypeID.impl end systemTypeID.impl;
System		System 构件(作为 Model 元素生成 System 的子构件,且具有层次化结构)	
Task	Task(任务)	Process 构件	process processTypeID end processTypeID; process implementation processTypeID.impl end processTypeID.impl;
	SubTask (原子任务)	Thread 构件	thread threadTypeID properties Dispatch_Protocol⇒Periodic; end threadTypeID; thread implementation threadTypeID.impl end threadTypeID.impl;
			thread threadTypeID properties Dispatch_Protocol⇒Aperiodic; end threadTypeID; thread implementation threadTypeID.impl end threadTypeID.impl;
ShareFunctionalModule		Subprogram	subprogram subprogram ID end subprogram ID; subprogram implementation subprogram ID.impl end subprogram ID.impl;
Event		Event Port	thread threadTypeID features iep: in event port; oep: out event port; end threadTypeID;
Data		Data Port	thread threadTypeID features idp: in data port; odp: out data port; end threadTypeID;
Functional Req. (数据交互部分)		Connection	name:[descriptor][source port] [connection symbol][destination port]

对于其他功能需求,则需要通过 AADL 的行为附件进行描述,其自动化转换工作尚处于研究过程中,当前阶段还需通过人工精化的方式逐步将这些信息完善到 AADL 设计模型之中.

3.3 转换算法和可追踪性

基于上述转换规则,本文设计了相应的转换算法.同时,为了能够实现需求追踪关系的自动化生成,本文在转换算法的输出中增加了需求追踪信息表,用于记录需求元素与 AADL 构件之间的追踪关系.任意需求抽象语法图元素在转换为目标 AADL 构件的同时,都会将两者之间的关联关系自动记录在需求追踪信息表中.在完成需求抽象语法图到 AADL 模型转换的同时,也就自动生成了相应的需求追踪信息表.具体的算法伪代码如算法 1 所示.

算法 1. AADL 模型自动化生成算法.

Input: *ASG_Model S*.

Output: *AADL_Model M*,

```
Output: Requirements Traceability Table RTT.
        Procedure Transform_ASG_Model
        Global Requirements Traceability Table RTT
  top_system←call_rule_1(S)
  For each asg_system∈S
    aadl_system←Transform_ASG_System(asg_system)
    ADD aadl_system TO top_system
  For each FunctionReq∈S
    aadl_connection←call_rule_6(FunctionReq)
    ADD aadl_connection TO top_system
  ADD Create_Tracement(S,top_system) TO RTT
  RETURN top_system

//ASG 中的 System 转换到 AADL 模型中的 System
Input: ASG_System asg_system;
Output: AADL_System aadl_system.
  Function Transform_ASG_System
  aadl_system←call_rule_2(asg_system)
  For each asg_task∈asg_system
    aadl_process←Transform_ASG_Task(asg_task)
    ADD aadl_process TO aadl_system
  For each FunctionReq∈asg_system
    aadl_connection←call_rule_6(FunctionReq)
    ADD aadl_connection TO aadl_system
ADD Create_Tracement(asg_system,aadl_system) TO RTT
RETURN aadl_system

//ASG 中的 Task 转换到 AADL 中的 process
Input: ASG_Task asg_task;
Output: AADL_Process aadl_process.
  Function Transform_ASG_Task
  aadl_process←call_rule_3(asg_task)
  For each asg_sub_task∈asg_task
    aadl_thread←Transform_ASG_Sub_Task(asg_sub_task)
    ADD aadl_thread TO aadl_process
  For each FunctionReq∈asg_task
    aadl_connection←call_rule_6(FunctionReq)
    ADD aadl_connection TO aadl_process
  ADD Create_Tracement(asg_task,aadl_process) TO RTT
  RETURN aadl_proces

//ASG 中的 SubTask 转换到 AADL 中的 thread
```

Input: *ASG_Sub_Task asg_sub_task*;
Output: *AADL_Thread aadl_thread*.
Function *Transform_ASG_Sub_Task*
aadl_*thread*←*call_rule*_4(*asg_sub_task*)
ADD *Create_Tracement*(*asg_sub_task*,*aadl_thread*) TO RTT
Return *aadl_thread*

//ASG 中的 ShareFunctionalModule 转换到 AADL 中的 subprogram
Input: *ASG_ShareFunctionalModule asg_share_functional_module*;
Output: *AADL_Subprogram aadl_subprogram*.
Function *Transform_ASG_Share_Functional_Module*
aadl_thread ←*call_rule*_5(*asg_share_functional_module*)
ADD *Create_Tracement*(*asg_share_functional_module*,*aadl_thread*) TO RTT
Return *aadl_subprogram*

//ASG 中的顶层概念 Model 转换到 AADL 模型的 System 构件.
//Rule2-Rule4,Rule6 与 Rule1 结构类似
Input: *ASG_System asg_system*;
Output: *AADL_System aadl_system*.
Function *call_rule*_1
aadl_system←*AADLUtil.createSystem*()
For *asg_data*∈*asg_system*
aadl_data←*call_rule*_6_1(*asg_data*)
ADD *aadl_data* To *aadl_system*
For *asg_event*∈*asg_system*
aadl_event←*call_rule*_6_2(*asg_event*)
ADD *aadl_event* To *aadl_system*

//ASG 中的 data 元素则转换到 AADL 模型 feature 中的 data port;
Input: *Asg_Data asg_data*;
Output: *AADL_Data aadl_data*.
Function: *call_rule*_6_1
if *asg_data.isInput*()
aadl_data←*call_rule*_6_1_1(*asg_data*)
else
aadl_*data*←*call_rule*_6_1_2(*asg_data*)
RETURN *aadl_data*

//ASG 中的 event 元素转换到 AADL 模型 feature 中的 event port
Input: *Asg_Event asg_event*;
Output: *AADL_Event aadl_event*.
Function: *call_rule*_6_2

```
    if asg_event.isInput()
      aadl_event←call_rule_6_2_1(asg_event)
    else
      aadl_event←call_rule_6_2_2(asg_event)

  //将功能需求(FunctionReq)中的数据交互转换到 AADL 模型
  //端口间的连接(connection)
  Input: FunctionReq req;
  Output: AADL_Connection.
    Function call_rule_7
    if req.isConnection()
      aadl_connection←AADLUtil.createConnection()
      aadl_connection.setTarget(req.getTargetName())
      aadl_connection_setSource(req.getSourceName())
      aadl_connection.setType(req.getTypeName())
      RETURN aadl_connection
    else
      RETURN null
```

基于该算法生成的只是初始的 AADL 架构模型,更加具体的 AADL 设计模型需依赖人工进行精化.这是因为对于不同领域、不同型号嵌入式软件系统而言,其具体的功能或者非功能需求难以形成统一的描述方式,因此,形成统一的转换规则进行自动化转换相对比较困难.但这些功能或非功能性需求约束通常都依托于某一构件或一组构件实现,因此,借助需求追踪关系能够准确定位到相关构件进行人工精化.本文重点关注的是 AADL 设计模型的自动化生成,需求追踪信息表仅仅作为模型转换的副产品而产生.因此,本文生成的需求追踪关系也仅仅是一种粗粒度的需求追踪关系,至于更加完整、精确需求需求追踪关系,还需后期在 AADL 模型精化过程中人工完善或者借助形式化推导等方法进行获取.在未来研究中,我们将进一步关注更加复杂、完整的需求追踪信息的自动化生成.

有效的需求追踪关系还能够用于检验需求是否被完整地体现在软件设计之中,同样,在软件维护与演化过程中,需求追踪关系还能够有效降低人力和时间成本,同时能够避免因需求变更带来的软件设计缺陷.特别是在安全关键领域,需求追踪关系还能够有效地支持软件的安全性分析工作,利用需求追踪关系,还能够在一定程度上缓解通过模型检测对设计模型进行安全性分析带来的状态空间爆炸问题.

4 工具实现与案例分析

本文的第 3 节给出了面向构件化软件开发的限定自然语言需求模板以及对应的需求抽象语法图元模型,第 4 节则在此基础上论述了需求抽象语法图到 AADL 模型的转换规则及算法.为了方便需求规约以及实现 AADL 模型的自动化生成,本节将在 AADL 开源工具 OSATE(open source AADL tool environment)[35]的基础上进行扩充,使其能够支持基于限定自然语言需求模板的需求规约,同时能够将需求模板转换成抽象语法图结构,进而完成 AADL 模型以及需求追踪关系的自动化生成.

4.1 工具实现

OSATE 是由 CMU 开发的 AADL 开源集成开发环境,是在 Eclipse 平台上的一套插件,用于 AADL 建模、编译和分析.在 OSATE 上开发了多种分析插件,进行可调度性分析、安全性分析、时间延迟分析等.本文是在 OSATE 2.2.1 基础上的进行再次开发,形成了需求建模插件(Req. Modeling plugin)和 AADL 模型自动化生成插件(Req2AADL plugin),且 AADL 模型自动化生成插件是基于需求建模插件开发的.两者的系统架构如图 4 所示.

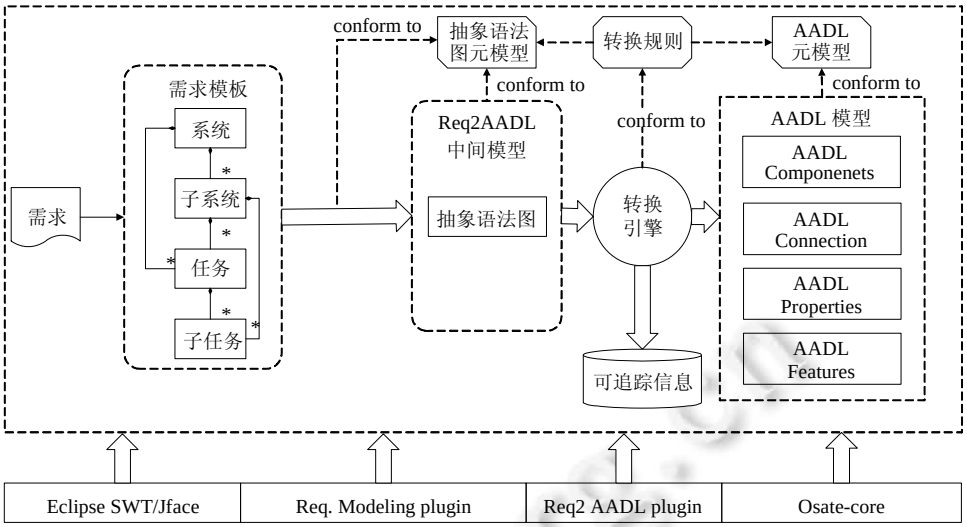


Fig.4 Framework of tool system

图 4 工具系统架构

4.2 案例分析

导航、制导与控制系统,即 GNC 系统,是航天器在轨运行的核心保障系统,承担着航天器姿态和轨道确定与控制的重要任务[36].GNC 系统一般由导航传感器、控制计算机和执行机构组成.其中,导航传感器包括导航相机、星敏感器、陀螺、加速度计等,主要用于采集各种数据;控制计算机,也称为姿态与轨道控制系统 AOCS;导航传感器和控制计算机之间存在一个接口装置,用来对采集的数据进行前期处理,称为数据处理单元(data process unit,简称 DPU);而 DPU 和导航传感器统称为局部终端处理单元(local terminal unit,简称 LTU).其简化的系统体系结构如图 5 所示.

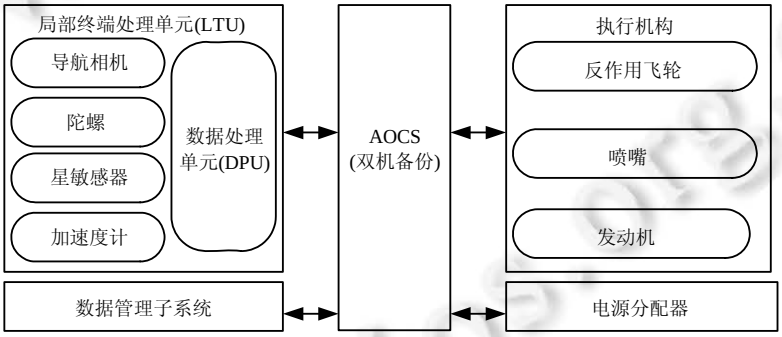


Fig.5 Simplified architecture of GNC system

图 5 GNC 系统的简化体系结构

基于 Req. Modeling plugin 进行需求建模时,首先需要将散布在软件需求文档中的各种数据以及实际项目中需要用到的各种名词收集起来,完成项目数据字典以及领域词库的构造;然后,依据软件系统的架构层次进行层次化功能单元分解,并在此基础上完成相应功能单元需求模板的填写,从而实现软件需求规约.在此过程中,使用的数据和特定名词均来源于数据字典与领域词库.具体而言,本案例中的导航、制导与控制系统包含局部终端处理单元和姿态与轨道控制系统两个子系统.其中,姿态与轨道控制子系统 AOCS 对应的需求模板见表 3,对应的 Req. Modeling plugin 插件显示如图 6 所示.

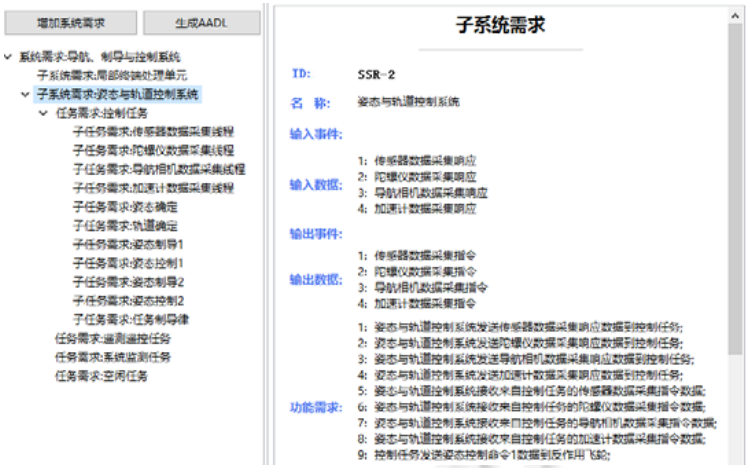


Fig.6 Requirement template of AOCS subsystem in Req. Modeling plugin

图 6 Req. Modeling plugin 中的 AOCS 子系统需求模板

完成需求规约后,通过 AADL 自动生成插件 Req2AADL plugin,可以将嵌入式软件需求通过中间模型自动转换到初始 AADL 设计模型,其中,数据字典自动转换为 AADL 中共享的 Data 构件,需求模板则通过转换规则和转换算法实现的转换引擎自动化生成相应的 AADL 模型元素.同时,自动导出需求与 AADL 构件之间的可追踪性信息.图 7 显示了部分自动生成的 AADL 代码,图 8 则为相应的图形化表示.

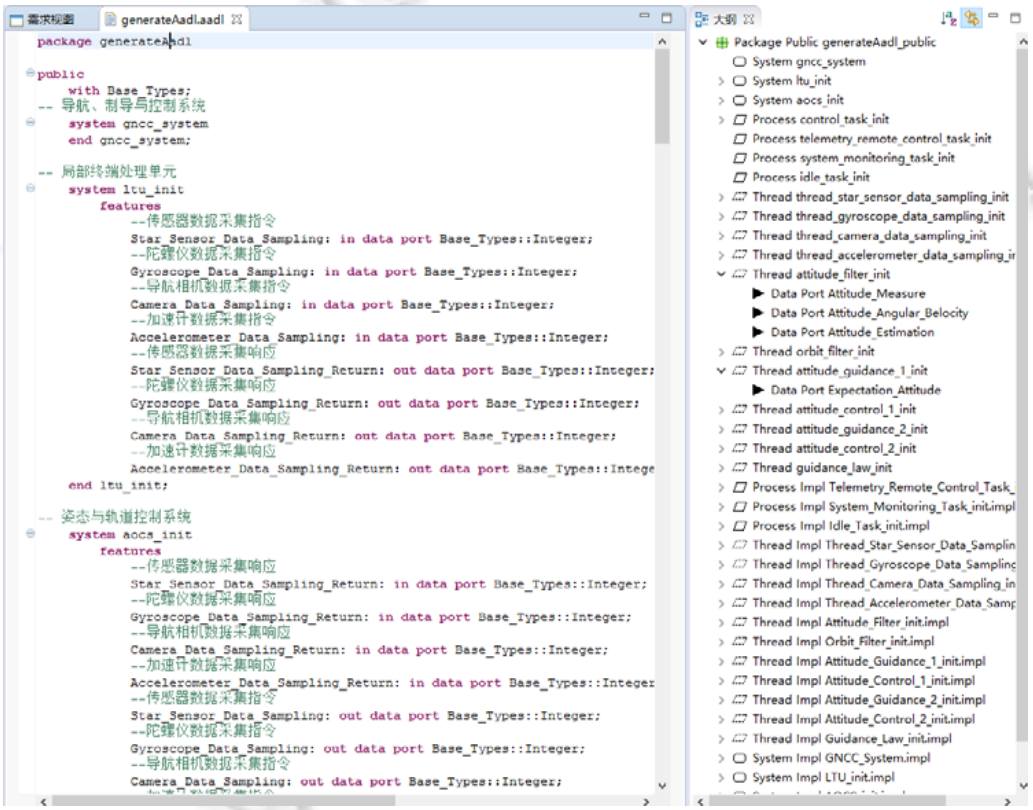


Fig.7 Generating AADL code of GNC system by Req2AADL plugin

图 7 基于 Req2AADL 插件生成的 GNC 系统 AADL 代码

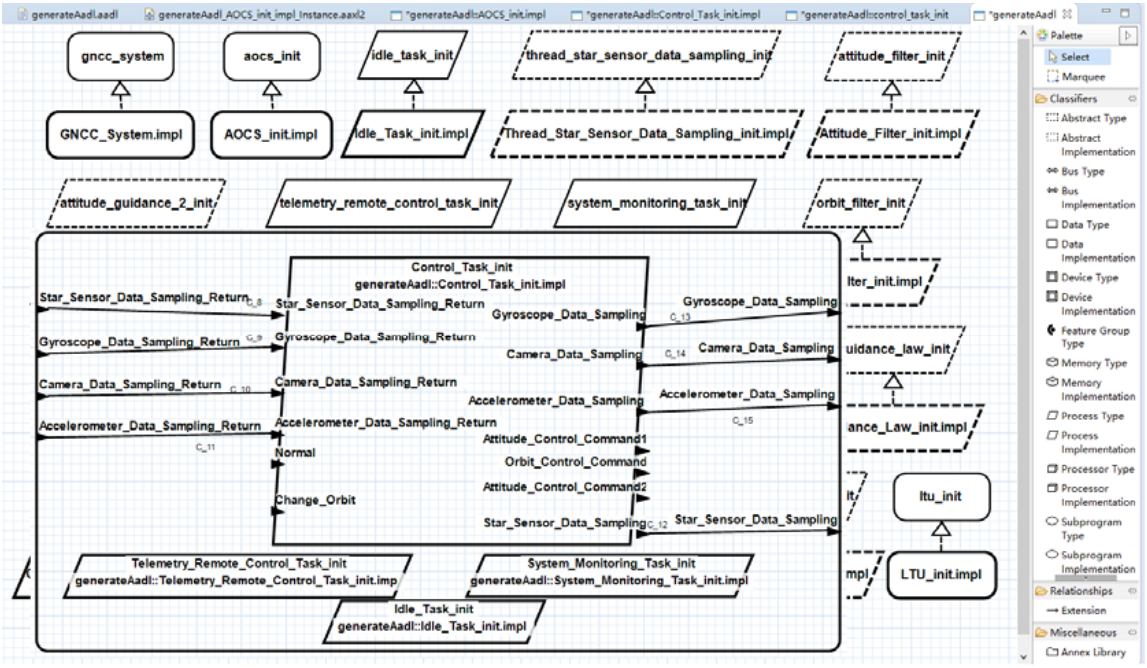


Fig.8 Graphical expression of generated GNC AADL model by Req2AADL plugin

图 8 基于 Req2AADL 插件生成的 GNC 系统 AADL 代码对应的图形化表示

基于自动生成的初始 AADL 模型可以通过人工精化的方式逐步完善功能与非功能属性,形成完整的 AADL 设计模型,并在此基础上进行嵌入式软件系统的安全性分析与验证以及代码生成等研究工作.

5 相关工作

需求规约和设计建模作为软件开发过程中两个关键活动,实现需求规约到软设计模型的转换通常需要解决两个问题:(1) 如何根据需求规约构建软件体系结构模型;(2) 如何保证模型转换的可追踪性.针对这两个问题,国内外学者展开了一系列研究.

Mavin 等人针对需求难以描述和表达的难题,在大量工程经验的基础上,提出了一种 EARS[27]方法进行需求规约.该方法将需求分为 Ubiquitous、Event-Driven、Unwanted Behavior、State-Driven、Optional Feature、Complex 这 6 种类型,并分别给出相应的需求表达模式.该方法可以有效简化需求的描述与表达难度,同时,能够有效降低自然语言需求的相关缺陷.Medvidovic 等人提出了一种通用的 CBSP (component-bus-system-property) 方法,使用构件、连接子等软件体系结构层次概念来逐步精化需求,从而得到软件体系结构的设计模型[37].北京大学张伟博士等人采用责任完成需求模型向软件体系结构模型的转换,从而提出一种从 CIM 到 PIM 转换的面向特征基于构件的方法,通过特征、责任与构件三者之间的映射关系来保证模型转换的可追踪性[38].挪威 Simula 实验室的岳涛教授等人针对传统 UML 用例规约描述存在的不足,提出了一种限定用例建模方法 RUCM[24,25].它在 UML 用例模型的基础上综合了自由文本和形式化语言的特点,既保证了需求的易理解性和精确性,同时又能够支持对需求进行自动化分析与验证.此外,文献[39–42]基于中间元模型 UCMeta,实现了 RUCM 到类图、顺序图、活动图的转换,并自动建立两者之间的追踪关系.这些研究主要针对通用软件,并不能充分体现嵌入式软件系统的特性.

在嵌入式软件体系结构模型的生成方面,德国哈根远程大学的鲁守荣博士等人为了规约嵌入式系统指定的特性而提出了基于构件的 UML profile,基于构件的 UML 模型被设计为平台无关模型(PIM),它可以被转化为用于目标平台实现的平台相关模型(PSM)[43].Shih 等人基于 VERTAF(verifiable embedded real-time application

framework)框架提出了它的多核版本 VMC(VERTAF/multi-core).VMC 是针对多核嵌入式软件体系结构的集成开发环境,开发者通过该平台,可以使用 SysML 描述系统需求,可以使用 SysML 标准建立模型,并且能够自动运行模式结构到设计来设计一个高质量多核嵌入式系统[44].但这些工作都并未考虑工业界软件需求以自然语言描述出发的现实.

此外,针对 RUCM 无法描述机载嵌入式软件的安全性需求的不足,北航的吴际教授等人对 RUCM 进行了扩展,添加了相应的模板与限制规则,形成了 Safety RUCM,使其支持安全性需求的规范化描述[45].然而,这些工作主要从用例的角度进行需求描述,描述范围不全面,且不能充分反映嵌入式软件系统的需求特征.

Holtmann 等人针对汽车工程领域的软件系统的需求规约展开研究.为了实现文本需求的自动化处理,他们提出了一种受控自然语言(controlled natural language,简称 CNL)进行需求描述[46].基于 CNL 可以实现需求的自动化验证,同时能够对需求的不一致性和不完整性进行检测.并且在其后续工作中,他们基于 CNL 需求规约方法提出了一种在自动化开发过程中,半自动化地建立与维护需求追踪关系的方法[47−49].

航天五院的顾斌总师等人面向航天型号软件,提出了航天需求描述语言 SPARDL,能够清晰、无二义地描述航天软件需求.SPARDL 包括数据字典、模块和模式图这 3 个部分,其中的核心为模式图.经实践证明,SPARDL 能在航天型号研究过程中有效地提高航天嵌入式软件的质量[26].但是 SPARDL 方法采用形式化需求描述方法,在实践过程中具有较大难度,且 SPARDL 是专门针对航天型号软件而设计,其通用性较差.

6 总结与展望

安全关键软件系统的需求描述及其设计模型的自动化生成,是软件工程领域的研究热点.本文针对这一问题展开研究,面向构件化嵌入式软件,提出了一种基于限定自然语言需求模板的需求规约方法,可以有效地降低自然语言需求的二义性与模糊性.同时,为了促进需求模型的自动化处理能力,选取了抽象语法图作为中间模型,在此基础上制定了需求模型到 AADL 模型的转换规则,并通过算法实现了 AADL 模型的自动化生成.同时,自动生成两者之间的可追踪性信息.最后进行了工具实现,并以航天领域典型应用案例说明了本文所提出方法的有效性.

本文所提出的方法能够将模型驱动思想在嵌入式软件开发中的应用扩展到需求阶段.为了方便转换到 AADL 软件体系结构模型,本文在需求模板的制定过程中做了一定的约束.未来我们将对需求模板的表达能力作进一步扩展,如增加对跨(子)系统任务的表达能力、降低约束提高通用性等.此外,本文生成的 AADL 模型只是初始的结构模型,完整的 AADL 模型还需进一步通过人工精化.未来,我们将进一步考虑如何实现更加完整的功能需求以及非功能需求的自动化转换.此外,本文生成的需求追踪信息仅仅作为副产品伴随着 AADL 模型的自动化生成而产生,这种追踪关系比较简单且粒度较粗.精确、完整的需求追踪关系可以有效保障嵌入式软件安全性,因此,如何构建更加复杂且精确完整的需求追踪关系,也是我们未来关注的研究重点.

References:

[1] Daskaya I, Huhn M, Milius S. Formal safety analysis in industrial practice. In: Proc. of the 16th Int'l Conf. on Formal Methods for Industrial Critical Systems. Springer-Verlag, 2011. 68−84.

[2] Software bug halts F-22 flight. 2007. http://tech.sina.com.cn/d/2007-12-28/07411942154.shtml

[3] Afshar A, Hajyhosseinloo M, Eftekhari A, Safari MB, YeKta Z. A report of the injuries sustained in Iran air flight 277 that crashed near Urmia, Iran. Archives of Iranian Medicine, 2012,15(5):317.

[4] The Toyota recall crisis. 2010. http://www.motortrend.com/features/auto_news/2010/112_1001_toyota_recall_crisis/viewall.html

[5] MIL-STD-882D. Standard Practice for System Safety Program Requirements. Military: Department of Defense, 2000.

[6] NASA. NASA-STD-8710.13, Software Safety. Washington: NASA, 2004.

[7] Mc Dermind J. Software hazard and safety analysis. In: Proc. of the 7th Int'l Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 2002). LNCS 2469, Oldenburg: Springer-Verlag, 2002. 23−34.

[8] Leveson NG. Software safety: Why, what, and how. Computing Survey, 1986,18(2):125−163.

[9] Aerospace SAE. Architecture analysis & design language (standard SAE AS5506). 2004. https://saemobilus.sae.org/content/as5506

[10] Aerospace SAE. Architecture analysis & design language (standard SAE AS5506A). 2009. https://saemobilus.sae.org/content/as5506a

[11] Yang ZB, Pi L, Hu K, Gu ZH, Ma DF. AADL: An architecture design and analysis language for complex embedded real-time systems. Ruan Jian Xue Bao/Journal of Software, 2010,21(5):899−915 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3700.htm [doi: 10.3724/SP.J.1001.2010.03700]

[12] Farail P, Gaufillet P, Canals A, Camus CL, Sciamma D, Michel P, Crégut X, Pantel M. The TOPCASED project: A toolkit in open source for critical aeronautic systems design. Embedded Real Time Software (ERTS), 2006,781:54−59.

[13] Gaufillet P, Heim S, Bonnin H, Dissaux P. ITEA SPICES AADL experimentation at airbus. In: Proc. of the 14th Int'l Conf. on Reliable Software Technologies, Ada-Europe. Washington: IEEE Computer Society Press, 2009. http://public.enst-bretagne.fr/~kermarre/RST2009/p.gaufillet_airbus.pdf

[14] Lewis BA, Feiler PH. Multi-Dimensional model based engineering using AADL. In: Proc. of the 19th IEEE/IFIP Int'l Symp. on Rapid System Prototyping. Washington: IEEE Computer Society Press, 2008. xv−xviii.

[15] Athalye P, Maksimovic D, Erickson R. High-Performance front-end converter for avionics applications [aircraft power systems]. IEEE Trans. on Aerospace and Electronic Systems, 2003,39(2):462−470.

[16] Alexander P, Kong C. Heterogeneous modeling support for embedded systems design. In: Thomas A, Kirsch HCM, eds. Proc. of the Embedded Software. Heidelberg: Springer-Verlag, 2001. 1−13.

[17] Sztipanovits J, Karsai G. Embedded software: Challenges and opportunities. In: Thomas A, Kirsch HCM, eds. Proc. of the Embedded Software. Heidelberg: Springer-Verlag, 2001. 403−415.

[18] He JF, Li XS, Liu ZM. Component-based software engineering—The need to link methods and their theories. In: Van Hung D, Wirsing M, eds. Proc. of the Theoretical Aspects of Computing (ICTAC 2005). Heidelberg: Springer-Verlag, 2005. 70−95.

[19] Crnkovic I, Larsson M. A case study: Demands on component-based development. In: Proc. of the 22nd Int'l Conf. on Software Engineering (ICSE 2000). Limerick: IEEE Computer Society, 2000. 23−31.

[20] Crnkovic I. Component-Based software engineering-new challenges in software development. Software Focus, 2001,2(4):127−133.

[21] Hu J. Formal analysis and verification for component-based embedded software design [Ph.D. Thesis]. Nanjing: Nanjing University, 2005 (in Chinese with English abstract).

[22] Elmqvist J, Nadjm-Tehrani S. Safety-Oriented design of component assemblies using safety interfaces. Electronic Notes in Theoretical Computer Science, 2007,182(29):57−72.

[23] Sakugawa B, Cury E, Yano ET. Airborne software concerns in civil aviation certification. In: Maziero CA, ed. Proc. of the Dependable Computing. Heidelberg: Springer-Verlag, 2005. 52−60.

[24] Yue T. Restricted use case modeling approach (User manual). Technical Report, Simula Research Laboratory, 2010.

[25] Yue T, Briand LC, Labiche Y. A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation. In: Schürr A, ed. Proc. of the Model Driven Engineering Languages and Systems. Heidelberg: Springer-Verlag, 2009. 484−498.

[26] Gu B, Dong YW, Wang Z. Formal modeling approach for aerospace embedded software. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):321−331 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4784.htm [doi: 10.13328/j.cnki.jos.004784]

[27] Mavin A, Wilkinson P, Harwood A, Novak M. Easy approach to requirements syntax (EARS). In: Proc. of the 17th IEEE Int'l Requirements Engineering Conf. New York: IEEE, 2009. 317−322. [doi: 10.1109/RE.2009.9]

[28] França RB, Bodeveix JP, Filali M, Rolland JF, Chemouil D, Thomas D. The AADL behaviour annex-experiments and roadmap. In: Proc. of the 12th IEEE Int'l Conf. on Engineering Complex Computer Systems (ICECCS 2007). Washington: IEEE Computer Society, 2007. 377−382. [doi: 10.1109/ICECCS.2007.41]

[29] Thomas A, Joël C, Philippe D, Pierre YP, Jean CR. AADL execution semantics transformation for formal verification. In: Proc. of the 13th IEEE Int'l Conf. on Engineering of Complex Computer Systems. Washington: IEEE Computer Society, 2008. 263−268. [doi: 10.1109/ICECCS.2008.24]

[30] SAE Aerospace. Architecture analysis and design language (AADL) annex Vol.1 (standard SAE AS5506/1). 2011. https://saemobilus.sae.org/content/as5506/1

[31] Feiler P. SAE aadl error model annex: An overview. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2011. https://wiki.sei.cmu.edu/aadl/images/1/13/ErrorModelOverview-Sept222011-phf.pdf

[32] SAE Aerospace. Architecture analysis and design language (AADL) annex D: Behavior model annex (standard SAE AS5506TM/3). 2017. https://saemobilus.sae.org/content/AS5506/3/

[33] SAE Aerospace. SAE architecture analysis and design language (AADL) annex Vol.2 (standard SAE AS5506/2). 2011. https://saemobilus.sae.org/content/AS5506/2/

[34] Delange J. ARINC653 AADL annex. ARINC653 annex overview. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2013. https://wiki.sei.cmu.edu/aadl/images/7/78/Arinc653annex-jul13.pdf

[35] The OSATE Website. http://osate.org/

[36] Li ZS, Gu B. Application research of AADL in design of space craft control system. Aerospace Control and Application, 2011, 37(1):55−58, 62 (in Chinese with English abstract).

[37] Medvidovic N, Dashofy EM, Taylor RN. The role of middleware in architecture-based software development. Int'l Journal of Software Engineering and Knowledge Engineering, 2003,13(4):367−393.

[38] Zhang W, Mei H, Zhao H, Yang J. Transformation from CIM to PIM: A feature-oriented component-based approach. In: Briand L, ed. Proc. of the Model Driven Engineering Languages and Systems. Berlin, Heidelberg: Springer-Verlag, 2005. 248−263.

[39] Yue T, Briand LC, Labiche Y. Facilitating the transition from use case models to analysis models: Approach and experiments. ACM Trans. on Software Engineering and Methodology (TOSEM), 2013,22(1):Article No.5.

[40] Yue T, Briand LC, Labiche Y. Automatically deriving a UML analysis model from a use case model. Technical Report, 2010-15, Oslo: Simula Research Laboratory, 2010.

[41] Yue T, Briand LC, Labiche Y. An automated approach to transform use cases into activity diagrams. In: Proc. of the 6th European Conf. on Modelling Foundations and Applications (ECMFA). Heidelberg: Springer-Verlag, 2010. 337−353.

[42] Yue T, Briand LC, Labiche Y. aToucan: An automated framework to derive UML analysis models from use case models. ACM Trans. on Software Engineering and Methodology (TOSEM), 2015,24(3):Articale No.13.

[43] Lu S, Halang WA, Zhang L. A component-based UML profile to model embedded real-time systems designed by the MDA approach. In: Proc. of the 11th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA 2005). New York: IEEE, 2005. 563−566. [doi: 10.1109/RTCSA.2005.6]

[44] Shih C, Wu CT, Lin CY, Hsiung PA, Hsueh NL, Chang CH, Koong CS, Chu WC. A model-driven multicore software development environment for embedded system. In: Proc. of the 33rd Annual IEEE Int'l Computer Software and Applications Conf. New York: IEEE, 2009. 261−268. [doi: 10.1109/COMPSAC.2009.148]

[45] Wu X, Liu C, Xia Q. Safety requirements modeling based on RUCM. In: Proc. of the Computing, Communications and IT Applications Conf. (ComComAp). New York: IEEE, 2014. 217−222. [doi: 10.1109/ComComAp.2014.7017199]

[46] Holtmann J, Meyer J, von Detten M. Automatic validation and correction of formalized, textual requirements. In: Proc. of the 4th Int'l Conf. on Software Testing, Verification and Validation Workshops (ICSTW). New York: IEEE, 2011. 486−495. [doi: 10.1109/ICSTW.2011.17]

[47] Fockel M, Holtmann J, Meyer J. Semi-Automatic establishment and maintenance of valid traceability in automotive development processes. In: Proc. of the 2nd Int'l Workshop on Software Engineering for Embedded Systems. Piscataway: IEEE Press, 2012. 37−43.

[48] Fockel M, Holtmann J. A requirements engineering methodology combining models and controlled natural language. In: Proc. of the 4th Int'l Model-Driven Requirements Engineering Workshop (MoDRE) at Requirements Engineering 2014. New York: IEEE, 2014. 67−76. [doi: 10.1109/MoDRE.2014.6890827]

[49] Daun M, Fockel M, Holtmann J, Tenbergen B. Goal-Scenario-Oriented requirements engineering for functional decomposition with bidirectional transformation to controlled natural language: Case study "body control module". ICB-Research Report, No.55, Essen: Institut für Informatik und Wirtschaftsinformatik (ICB), Universität Duisburg-Essen, 2013. 1−68.

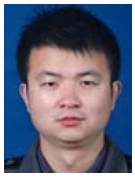
附中文参考文献:

[11] 杨志斌,皮磊,胡凯,顾宗华,马殿富.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. http://www.jos.org.cn/1000-9825/3700.htm [doi: 10.3724/SP.J.1001.2010.03700]

[21] 胡军.构件化嵌入式软件设计的分析与验证[博士学位论文].南京:南京大学,2005.

[26] 顾斌,董云卫,王政.面向航天嵌入式软件的形式化建模方法.软件学报,2015,26(2):321−331. http://www.jos.org.cn/1000-9825/4784.htm [doi: 10.13328/j.cnki.jos.004784]

[36] 李振松,顾斌.AADL 在航天器控制系统设计中的应用研究.空间控制技术与应用,2011,37(1):55−58,62.



王飞(1990－),男,安徽安庆人,博士生,主要研究领域为软件工程,安全关键嵌入式软件,需求工程.

刘承威(1994－),男,硕士生,主要研究领域为软件工程,安全关键嵌入式软件,需求工程.

杨志斌(1982－),男,博士,副教授,CCF 专业会员,主要研究领域为安全关键嵌入式软件,形式化方法.

章文炳(1992－),男,硕士,主要研究领域为软件工程,安全关键嵌入式软件,模型驱动开发.

黄志球(1965－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为软件工程,软件安全性,形式化方法.

薛垒(1982－),男,高级工程师,主要研究领域为嵌入式软件设计验证.

周勇(1975－),男,博士,副教授,CCF 专业会员,主要研究领域为软件工程,形式化方法.

许金淼(1994－),男,硕士生,主要研究领域为软件工程,安全关键嵌入式软件.