

## 面向限定自然语言需求的AADL自动生成工具

刘承威<sup>1</sup>, 杨志斌<sup>1,2</sup>, 周勇<sup>1</sup>, 袁胜浩<sup>1</sup>, 许金森<sup>1</sup>, 薛垒<sup>3</sup><sup>1</sup>(南京航空航天大学 计算机科学与技术学院, 南京 211106)<sup>2</sup>(软件新技术与产业化协同创新中心, 南京 210093)<sup>3</sup>(上海航天电子技术研究所, 上海 201109)

E-mail: lcwj3nuaa@nuaa.edu.cn

**摘要:** 在航空、航天、交通、能源等安全关键领域中, 软件的失效可能导致系统处于危险状态, 从而导致财产损失、环境破坏甚至人员伤亡, 如何保障这类软件的可靠性和安全性一直是学术界和工业界共同面临的难题。近年来, 形式化模型驱动的安全关键软件设计与验证方法逐渐受到重视, 并被认为是切实可行的重要手段。然而, 形式化模型驱动开发方法的生命周期一般较少涉及需求阶段, 主要原因是当前工业界的软件需求主要通过自然语言文本描述。而安全关键软件引起严重事故的问题链的最上端原因往往又是软件需求尤其是安全性需求的问题。AADL(Architecture Analysis & Design Language)是一种广泛应用于安全关键软件领域的建模语言标准。本文针对自然语言需求和AADL模型驱动开发方法之间还存在鸿沟的问题, 研究基于限定自然语言的安全关键软件需求建模及AADL模型自动生成方法。首先, 提出一种基于限定自然语言的需求规约方法, 通过结构化的需求组织方式及受限的自然语言以减少需求表达中存在的二义性。其次, 给出限定自然语言需求到AADL模型的自动转换方法。此外, 本文给出一种结构化的验证性质描述模板, 并自动转换到AADL组合验证附件AGREE(Assume Guarantee REasoning Environment)Annex, 从而支持对AADL模型进行形式化验证。最后, 在AADL开源工具环境OSATE中实现了原型工具, 并基于航天导航制导控制系统进行案例分析。

**关键词:** 安全关键软件; AADL; 限定自然语言; 组合验证

中图分类号: TP311

文献标识码: A

文章编号: 1000-1220(2019)05-0984-12

## Automated Tool to Derive AADL Models from Requirements in Restricted Natural Language

LIU Cheng-wei<sup>1</sup>, YANG Zhi-bin<sup>1,2</sup>, ZHOU Yong<sup>1</sup>, YUAN Sheng-hao<sup>1</sup>, XU Jin-miao<sup>1</sup>, XUE Lei<sup>3</sup><sup>1</sup>(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)<sup>2</sup>(Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210093, China)<sup>3</sup>(Shanghai Aerospace Electronic Technology Institute, Shanghai 201109, China)

**Abstract:** Embedded systems have become increasingly complex and are widely used in safety-critical domains such as automotive and aerospace. Malfunctions of Safety-Critical Softwares (SCSs) can lead to accidents that can potentially put people, environment, property and mission in serious risks such as environmental catastrophes and loss of lives. Nowadays, Model-Driven Development (MDD) and Formal Method (FM) has been increasingly used in the design and verification of SCSs. However, the requirement phases are scarcely involved in MDD, this is because requirements are mostly written as free natural language text, which is often ambiguous, and difficult to be processed automatically and generate meaningful downstream design artifacts (e.g., models) from such requirements specifications, especially safety requirements. Architecture Analysis & Design Language (AADL) is a standardized architecture description language for embedded systems, which is widely used in avionics and aerospace industries to model safety-critical applications. Therefore, to bridge the gap between textual requirements and AADL models, we propose a requirements specification methodology named as RNLreq, which is equipped with a set of restriction rules and templates to structure and restrict the way how users document requirements, and an automated transformation of RNL2AADL to automatically derive AADL architecture models from requirements specified with RNLreq. Besides, to simplify the formal verification of AADL models, we propose a template for describing verification properties, which can assist engineers to generate AADL AGREE Annex for composition verification. Finally, we realize a tool in OSATE and evaluate these proposed methods with an industrial case study.

**Key words:** safety-critical software; AADL; restricted natural language; compositional verification

收稿日期: 2018-08-13 收修改稿日期: 2018-09-15 基金项目: 国家自然科学基金项目(61502231)资助; 国家重点研发计划项目(2016YFB1000802)资助; GF基础科研重点项目(JCKY2016203B011)资助; 江苏省自然科学基金项目(BK20150753)资助; 航空科学基金项目(2015ZC52027)资助。 作者简介: 刘承威, 男, 1994年生, 硕士研究生, CCF会员, 研究方向为模型驱动、软件工程等; 杨志斌, 男, 1982年生, 博士, 副教授, CCF会员, 研究方向为安全关键软件、形式化验证、模型驱动等; 周勇, 男, 1975年生, 博士, 副教授, CCF会员, 研究方向为软件工程、形式化方法等; 袁胜浩, 男, 1994年生, 硕士研究生, CCF学生会员, 研究方向为同步语言、形式化方法等; 许金森, 男, 1994年生, 硕士研究生, 研究方向为软件工程、模型驱动等; 薛垒, 男, 1982年生, 硕士, 高级工程师, 研究方向为嵌入式软件验证、嵌入式软件系统设计。

## 1 引言

安全关键软件(Safety-Critical Software)<sup>[1]</sup>是指应用于航空、航天、交通、能源等领域的安全关键系统中的一类软件,其运行情况可能引起系统处于危险状态,从而导致财产损失、环境破坏或者人员伤亡。如2016年,日本宇航局JAXA宣布耗资310亿日元的X射线天文探测卫星Hitomi由于姿控软件故障,导致异常翻滚,最终彻底失控。安全关键软件一般为安全关键系统的一部分,它可能引起或者助长不安全的条件,这样的软件被认为是安全关键的<sup>[2]</sup>。如何保障这类软件的可靠性和安全性一直是学术界和工业界共同面临的难题。

近年来,模型驱动(Model-Driven)尤其是采用形式化模型驱动的安全关键软件设计与开发方法逐渐受到重视,并被工业界认为是保障软件安全性与可靠性切实可行的重要手段。例如,国际民航领域使用的机载系统适航审定中的软件开发标准DO-178C<sup>[3]</sup>就将模型驱动和形式化方法(即DO-331<sup>[4]</sup>和DO-333<sup>[5]</sup>)作为其核心标准的重要技术补充。模型驱动开发方法以模型为整个软件开发过程的核心,通过在设计阶段建立软件的体系结构模型,实现模型的尽早验证和分析。同时,模型的重用以及基于模型转换的求精过程,都有助于降低软件开发的时间和成本。

然而,模型驱动开发生命周期一般较难涉及需求阶段<sup>[6]</sup>,而是从软件的分析或设计开始,其主要原因是当前工业界的软件需求主要通过自然语言文本描述。但是,已有研究表明,安全关键软件引起严重事故的问题链的最上端原因往往又是软件需求尤其是安全性需求的问题<sup>[7]</sup>。如著名的欧洲阿里安娜5火箭爆炸事件,事后分析其主要原因是阿里安娜5火箭是在阿里安娜4的基础上开发的,它的初始加速度比后者高了很多,这导致阿里安娜5火箭的水平速度是阿里安娜4的6倍。但这一部分软件恰恰是重用了阿里安娜4的模块,开发者没有意识到有关的需求已经发生了变化,而没有根据新的需求对重用的软件进行必要的修改<sup>[8]</sup>。因此,如何将自然语言需求和模型驱动开发方法进行有效链接,即实现自然语言需求到形式化设计模型的自动或半自动转换是模型驱动开发方法在安全关键软件设计与开发中实际应用的一个主要挑战<sup>[6,9,10]</sup>。

目前,安全关键软件的需求表达方式主要包括自然语言自由文本或限定自然语言文本(模板、表格等),用例图以及少量直接使用形式化的需求描述方式等。而常用于模型驱动开发的建模语言主要包括UML、SysML、AADL、EAST-ADL等,其中AADL(Architecture Analysis & Design Language)<sup>[11]</sup>是由美国汽车工程师协会SAE(Society of Automotive Engineers)提出的面向安全关键嵌入式系统的一种建模语言,并发布为SAE AS5506标准。AADL采用单一模型支持多种分析的方式,支持对安全关键嵌入式系统的软硬件混合建模,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<sup>[30]</sup>,是一种非常适用于安全关键系统的体系结构建模与分析语言。

我们将已有需求表达达到设计模型的转换分为以下三类:

1)针对自由文本描述的软件需求,通过自然语言处理(Natural Language Processing, NLP)的方式解析自由文本,并

对解析结果进行分析处理,最后转换到设计模型。例如Keletso J. Letsholo等提出的TRAM(Tool for Transforming Textual Requirements into Analysis Models)方法<sup>[12]</sup>,主要考虑自然语言需求到UML类图的转换,而Imran Sarwar Bajwa等则提出自然语言需求到的UML类图的文本说明即OCL(Object Constraint Language)的转换<sup>[13]</sup>;

2)给出结构化的限定自然语言需求表达方式,然后转换到设计模型。挪威Simula实验室Tao Yue等提出的RUCM(Restricted Use Case Modeling)<sup>[14,15]</sup>方法是通过对UML用例图的文本说明进行自然语言限定,以此降低自然语言带来的二义性与不精确性,并基于RUCM给出了aToucan工具<sup>[16]</sup>,支持RUCM到UML活动图、顺序图等自动化生成;正在制定的SysML V2.0也将会为SysML需求图中的文本说明设计自然语言约束规则,从而减少SysML建模过程中由自然语言带来的二义性等问题;由Rolls Royce公司提出的EARS(Easy Approach to Requirements Syntax)<sup>[17,18]</sup>方法则是提出了一些简单需求结构来捕获和表达用户需求,从而降低自然语言需求的二义性;

3)针对半形式化或形式化需求模型,抽取相关信息转换到设计模型,如Jonathan Lasalle等提出的SysML到UML的转换<sup>[19]</sup>、Van der Gaag等提出的SysML到SLIM的转换<sup>[20]</sup>等。

限定自然语言是一种重要的需求表达方式,它既能够减少自然语言带来的二义性,又能够较少改变工程师已有的需求撰写习惯。但是已有工作较少面向安全关键软件需求,同时,还没有考虑到AADL设计模型的自动转换。另外,在AADL模型验证与分析方面,验证性质一般包括功能正确性、时间正确性和资源利用正确性等多个方面,目前主要是将AADL模型转换到另外一种已有的形式化工具并通过模型检测的方式进行形式化验证,例如:AADL2Fiacre<sup>[21]</sup>,AADL2BIP<sup>[22]</sup>,AADL2IF<sup>[23]</sup>,AADL2ACSR<sup>[24]</sup>,AADL2RT-Maude<sup>[25]</sup>,AADL2Lustre<sup>[26]</sup>,AADL2Signal<sup>[27]</sup>,AADL2TASM/UPPAAL<sup>[28]</sup>等。但是模型检测方法容易存在状态空间爆炸问题,基于Contract理论的组合验证方法成为AADL模型验证的一个新的研究热点。

因此,本文提出一种基于限定自然语言的安全关键软件需求建模及其到AADL模型的自动转换方法。主要包括:首先,在综合分析RUCM<sup>[14]</sup>、EARS<sup>[17]</sup>等需求表达方式的基础上,提出基于限定自然语言需求模板的需求规约方法RNLreq,该规约方法能够在尽量不改变工程师的需求撰写习惯的前提下,降低自然语言需求存在的二义性与模糊性。其次,给出基于元模型的RNLreq到AADL模型的自动转换方法RNL2AADL。为了支持AADL模型的形式化验证,我们提出一种结构化的验证性质描述模板,并自动转换到AADL组合验证附件AGREE(Assume Guarantee REasoning Environment)Annex,以支持对AADL模型进行组合验证<sup>[29]</sup>。此外,我们在AADL开源建模工具OSATE<sup>[18]</sup>中实现工具原型,并给出案例分析。

论文第二节介绍AADL语言的基本建模概念及其组合验证扩展附件AGREE Annex。第三节给出限定自然语言需求建模方法RNLreq;第四节给出基于元模型的RNL2AADL模

型转换方法;第五节给出工具实现与案例分析;第六节对相关工作进行分析;第七节是本文的总结和展望。

## 2 体系结构分析与设计语言 AADL

体系结构分析与设计语言 AADL 是一种面向安全关键嵌入式系统的软硬件建模语言,本节主要介绍 AADL 基本建模概念及基于 Contract 理论的 AADL 组合验证扩展附件 AGREE Annex.

### 2.1 AADL 基本建模概念

2004 年,美国汽车工程师协会 SAE(society of automotive engineers)在 MetaH、UML 等建模语言的基础上,提出了嵌入式实时系统体系结构分析与设计语言 AADL<sup>[11]</sup>,目的是提供一种标准而又足够精确的方式,设计与分析嵌入式实时系统的软、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<sup>[30]</sup>. AADL 的这些特性使其具有广阔的应用前景,得到了以航空航天领域(如空客、波音、霍尼韦尔等)为首的欧美工业界的支持,此外学术界也对 AADL 也展开了深入的研究和扩展,其中,美国卡耐基梅隆大学为 AADL 开发的开源集成开发环境 OSATE 已被广泛使用.

安全关键系统是应用软件、运行时环境以及硬件平台深度融合的复杂系统, 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<sup>[30]</sup>.

AADL 是一种基于构件(Component)的软硬件建模语言,其软件构件主要用于描述系统的软件体系结构,主要包括线程(Thread)、线程组(Thread Group)、进程(Process)、数据(Data)、子程序(Subprogram)等;硬件构件主要用于描述系统的硬件体系结构,主要包括处理器(Processor)、虚拟处理器(Virtual Processor)、存储器(Memory)、外设(Device)、总线(Bus)、虚拟总线(Virtual Bus)等;此外, AADL 还提供了系统构件(System),整合分发协议(Dispatch)、通信协议(Communication)、调度策略(Scheduling)、模式变换协议(Mode Change)以及分区机制(Partition)等属性来描述系统的运行时环境,从而层次化地建立系统的体系结构模型.

在每个构件中, AADL 通过特征 features(port、data access)表示构件的事件、数据交互端口,通过连接 connection 描述构件间的交互行为,通过流 flow 描述系统中信息传输的逻辑路径,通过属性 property 描述体系结构中的约束条件,通过模式 mode 描述运行时体系结构的动态演化等.

此外, AADL 定义了属性集扩展和附件扩展两种方式,进一步丰富了 AADL 语言的表达能力. 其中,属性集扩展丰富了 AADL 在系统非功能约束方面的描述能力;而附件扩展则增强了 AADL 对构件实际功能行为的详细描述能力.

已有的扩展附件主要有: Graphical AADL Notation Annex、Error Model Annex、Behavior Annex、Data Modeling Annex、ARINC653 Annex 以及 AGREE Annex 等.

### 2.2 AADL AGREE Annex

一个嵌入式系统通常都会包含由不同领域计算模型组成的独立部分,每个独立部分都按照一定标准封装成构件(Component). 随着嵌入式系统功能不断加强,软件规模和复

杂性迅速提高,系统构件化特征显著,由此产生的状态空间爆炸问题使得传统的单构件/单模块形式化验证技术已经无法满足实际需求. 因此,组合验证理论被引入系统体系结构的验证过程中. 组合验证技术在传统单构件验证的基础上,以下层构件所满足的性质作为上层构件验证的基础,将系统验证分解成底层构件的形式化验证及不同体系结构层次的逻辑推理与组合,降低了系统形式化验证的实现难度.

基于契约(Contract)的组合验证方法是一种常见的组合验证方法,一个基本的构件契约具有如下描述:  $Contract = (Assume, Guarantee)$ , 其中 Assume 为假设部分,描述构件对运行环境的需求; Guarantee 为保证部分,描述当运行环境被满足时,构件可保证的外部行为特性,及构件承担的责任<sup>[38]</sup>. 对于一个软件构件而言,假设为功能正常执行的要求,保证为在要求满足时的产出结果.

AADL 作为一种面向构件化、层次化的嵌入式软硬件体系结构描述语言,对组合验证也提供支持. 2012 年 Rockwell Collins 公司基于 AADL 开源工具环境 OSATE 提出了假设保证推理环境 AGREE<sup>[29]</sup>,并发布为 AADL 组合验证附件 AGREE Annex,旨在构建一套基于 AADL 的组合验证平台工具,用于支持在 AADL 设计模型的基础上对嵌入式软硬件体系结构进行组合验证. AGREE Annex 以附件的形式,增强了 AADL 对验证性质、逻辑推理公式等的表达能力. 以下给出 AGREE 的 EBNF 语法:

$AgreeSubclause ::= (SpecStatement) + ;$

$SpecStatement ::=$

```
'assume' STRING ':' Expr ';'
| 'guarantee' STRING ':' Expr ';'
| EqStatement
| PropertyStatement
| ConstStatement
| FunDefExpr
| NodeDefExpr
| 'assert' Expr ';'
| 'lift' NestedDotId ';'
| LemmaStatement;
```

$LemmaStatement ::= 'lemma' STRING ':' Expr ';;'$

$PropertyStatement ::= 'property' ID '=' Expr ';;'$

$ConstStatement ::= 'const' ID ':' Type '=' Expr ';;'$

$EqStatement ::= 'eq' Arg (',' Arg) * '=' Expr ';;'$

$FunDefExpr ::= 'fun' ID '(' Arg (',' Arg) * ':' Type '=' Expr ';;'$

$NodeDefExpr ::= 'node' ID '(' Arg (',' Arg) * ')' ':' 'returns' '(' Arg (',' Arg) * ')' ':' NodeBodyExpr;$

$Arg ::= ID ':' Type;$

$NodeBodyExpr ::= ('var' (Arg ';' ) + ) ?$

$'let' (NodeStmt) + 'tel' ';;'$

$NodeStmt ::=$

$Arg (',' Arg) * '=' Expr ';;'$

$| LemmaStatement$

在 AGREE Annex 中, assume 和 guarantee 分别表示对应构件对环境的需求,以及需求满足条件下所能保证的输出. 此外, AGREE 还定义了等式(Eq)、性质语句(Property)、常量(Const)、函数(Fun)、节点(Node)、断言(Assert)、提升(Lift)、引理(Lemma)等,用以描述组件的执行过程.

AGREE 在 OSATE 的基础上,通过插件的形式提供了 AGREE 解析器,将 AGREE Annex 解析成 AGREE 抽象语法树,并提供 AGREE 分析器,将 AGREE 抽象语法树转换成同步语言 Lustre 程序,最后通过 JKind 模型检测工具对生成的 Lustre 程序进行验证,最终反馈出 AGREE Annex 中可能存在的缺陷或错误。

3 限定自然语言需求规约方法

本文在 RUCM、SPARDL 和 EARS 等基础上,提出限定自然语言需求规约方法 RNLreq,主要包括四部分,数据字典、领域词库、需求模板以及限定句式。

3.1 领域词库和数据字典

1)领域词库,主要针对具体的领域需求,将需求中所涉及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、软件模块名、硬件设备、模式变迁状态等。

领域词库中的名词可以定义为一个三元组,  $Noun::=<NounZhName, NounEnName, NounType>$ ,包括名词中文名称,英文名称以及类型,其中类型可以是系统、子系统、任务、子任务、共享功能模块、外设、总线、存储器、处理器、模式等类型。

2)数据字典,将需求文档中的各种数据都集中起来,并采用比较规范的方式对它们的属性进行描述,主要包括软件系统中的各种输入数据、输出数据、静态数据等。

数据字典中每条数据可以定义为一个七元组,  $Data::=<DataName, DataEnName, DataType, DataUnit, DataRange, DataAccuracy, DataDescription>$ ,包括中文名称、英文名、数据组成/类型、数据单位、数据值域、数据精度、数据描述等属性。其中  $DataType$  可以是基本数据类型 ( $Integer, Boolean, String$  等),也可以是由多个数据组成的复杂数据类型(类似 C 语言中的结构体)。

3.2 需求模板

嵌入式系统常常使用系统、子系统、功能、子功能的概念来对软件进行分层组织,并通过共享模块调用来实现跨系统或跨功能之间的模块共享。在 RNLreq 中,系统、子系统、功能、子功能以及共享模块都采用需求模板的方式表达需求,需求模板之间的层次关系则显式地表达软件系统的体系结构。其中,系统( $System$ )需求模板可以分解为若干个子系统( $Subsystem$ )需求模板,而子系统可以继续划分为多个子系统或功能( $Function$ ),功能也可划分为若干个子功能( $SubFunction$ )。此外,共享功能模块( $ShareFuctionBlock$ )作为独立的模块挂在系统中,可以被功能或子功能调用,共享功能模块可以用于描述一些经常被调用到的算法(如傅里叶变换)或与外设交互(如从 RS422 总线读写数据)。

限定自然语言需求模板是对需求的一种规范描述。根据文献[18]的建议,本文根据系统分解的思想,分别从系统、子系统、功能、子功能 4 个层次进行需求描述。基本的需求模板如表 1 所示。

需求模板由标识符 ID ( $Component\_ID$ )、名称 ( $Component\_Name$ )、输入 ( $Input$ )、输出 ( $Output$ )、子模块成组成 ( $Component\_Composition$ ) 以及需求约束 ( $Requirement\_Con-$

$straint$ ) 等项组成。其中 ID 是每个模板即软件模块的唯一标识;名称是从领域词库中选择获得,实现中英文对照,这样可以减少手工输入带来的不一致性;输入输出描述了软件模块与外界的交互,一般分为数据和事件两种类型。

表 1 限定自然语言需求模板  
Table 1 Restricted natural language requirement template

ID	唯一标识符
构件名称	限定为名词,或者多个名字的组合或简写,用于描述系统/子系统/功能/子功能的名称
输入	系统/子系统/功能/子功能的输入事件/数据,没有写 NONE
输出	系统/子系统/功能/子功能的输出事件/数据,没有写 NONE
子系统	一个复杂系统可以分解为若干子系统
功能	每个系统将实现若干功能
子功能	每个功能可以分解为若干原子功能
功能需求	每个功能单元在功能方面的描述
模式变换	每个功能单元在模式变换方面的描述
性能需求	规定每个功能单元在性能方面的约束
接口需求	各层组件数据流、控制流的协议描述
设计约束	软硬件设计约束,安全性可靠性的约束描述

需求约束包括功能处理 ( $Functional\_Process$ )、模式变换 ( $Mode\_Transition$ )、性能需求 ( $Performance\_Requirement$ )、接口需求 ( $Interface\_Requirements$ ) 及设计约束 ( $Design\_Restrictions$ ) 等。其中功能处理通过事件流来表达正常功能处理流程,其中的具体功能需求可以标定为安全关键功能;模式与模式变换用来描述当前模块在上一层系统中所处的工作模式(即源模式),以及当前模块的模式变换的触发条件和目标模式;性能需求则是对软件完成任务的能力做出的一些定量要求,如:实时性、精度、功耗、最大处理能力等;接口需求描述各模块与外界交互接口的要求,如接口数据传输协议等;设计约束描述功能模块在软硬件、安全性、可靠性等方面的约束。

由此,本文给出限定自然语言需求模板的 EBNF 语法如下:

```
RNLreq::= Glossary Dictionary TemplateSet SentencePatterns
Glossary::= Noun + ;
Noun = ( NounZhName NounEnName NounType ) + ;
DataDictionary::= Data + ;
Data = ( DataName DataEnName DataType DataUnit DataRange
DataAccuracy DataDescription ) + ;
TemplateSet::= ( RNLreq_Template ) + ;
RNLreq_Template::= Component_ID Component_Name Input Output
Component_Composition Requirement_Constraints
Requirement_Constraint::= Functional_Process [ Performance_require-
ments ] [ Mode_Transitions ] [ Interface_requirements ] [ Design
_constraints ] ;
Noun_Category::= Software_Category
| Hardware_Category
Template_Category::= 'System' | 'Subsystem' | 'Function' | 'SubFunc-
tion' | 'Shared_Function_Block' ;
Hardware_Category::= 'Device' | 'Bus'
```

```
'Processor' | 'Memory';
Component_Name ::= Noun_zh;
Component_Composition ::= ( RNL_Req_Template ) +
Functional_Requirements ::= ( Sentence_Pattern ) +
Performance_requirements ::= ( Sentence_Pattern ) +
Mode_Transitions ::= ( Sentence_Pattern ) +
Interface_requirements ::= ( Sentence_Pattern ) +
Design_constraints ::= ( Sentence_Pattern ) +
Input ::= { VarName_zh [ Assume_Specification ] } + ;
Output ::= { VarName_zh [ Guarantee_Specification ] } + ;
```

其中, *Noun* 表示领域专有名词, *Data* 表示需求中的数据, 对应 *Component\_Name* 必须是领域词库中已“注册”的名词, *Input*、*Output* 必须是“数据字典”中已注册的数据, *Component\_Composition* 描述需求模板间的层次关系, *Functional*、*Performance*、*Mode Transitions*、*Interface*、*Design constraints* 分别描述功能、性能、模式变换、接口、设计约束等不同类型的

3.3 限定句式

基于限定自然语言的需求规约方法不仅包括在需求组织结构上的约束, 还包括对每一条需求描述的约束. 在已有工作<sup>[31]</sup>中我们已经给出了限定自然语言需求模板的通用约束规则, 用于限定自然语言的表述方式, 以此减少在需求规约过程中人为引入的二义性和不确定性, 通用约束规则如表 2 所示.

表 2 通用自然语言约束规则  
Table 2 General restriction rules for natural language

约束规则	描述
R1	句子的主语必须是模块名称或“该模块”
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	准确的描述模块间的交互, 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

本文在已有工作的基础上, 分别为各类需求约束制定了一系列的需求描述限定句式, 本文以功能处理为例进行说明.

针对功能处理, 本文指定了 5 种限定句式, 涵盖在功能处理中常用的时间约束、判断选择等关系, 具体如表 3 所示.

其中, *Behavior* 表示一个或一组顺序执行的简单操作, 操作可以是数据收发、数据赋值、功能模块调用等; *Condition* 表示功能执行的条件, 可以分为触发条件(当接收到触发事件时执行)和判断条件(当值满足一定条件时执行)两种; *Time-Restrain* 表示 *Behavior* 执行的时间约束.

基于上述的约束规则及限定句式, 本文给出一个导航、引导与控制系统中的实际模块作为 RNLreq 的示例如表 4 所示.

表 3 功能需求描述句型  
Table 3 Sentence patterns for functional requirements

NO	ID	唯一标识符
SP1	<i>Behavior</i>	表示一个简单句, 单纯执行一条功能;
SP2	<i>Condition</i> + <i>Behavior</i>	表示在某种条件下完成一个功能行为
SP3	<i>TimeRestrain</i> + <i>Behavior</i>	表示在一定时间范围内完成一个功能行为;
SP4	<i>Condition</i> + <i>Behavior</i> + <i>else</i> + <i>Behavior</i>	表示在一定条件下完成一个功能行为, 若不满足条件则执行另一个功能行为
SP5	<i>TimeRestrain</i> + <i>Condition</i> + <i>Behavior</i>	表示在连续时间范围内一直满足某些条件, 执行一个功能行为

表 4 GNCC 控制数据转发模块  
Table 4 GNCC data retransmission module

名称	GNCC 控制数据转发模块
ID	TX_GNCC
是否安全关键	是
输入数据	GNCC 数据转发控制指令 GNCC 控制数据
输出数据	GNCC 控制数据 MF 标志
功能处理	1. 在 10ms 内, 该模块向 GNCC 发送握手信息. 2. 如果功能处理 1 失败, 那么执行 3 次功能处理 1; 如果 3 次功能处理 1 失败, 那么执行赋值 MF 标志位为 1, 该模块向日志模块发送错误信息. 3. 该模块向 GNCC 发送 GNCC 控制数据. 4. 该模块向 GNCC 发送结束信息.
性能需求	1. 该模块的周期 = 50ms.

4 RNL2AADL 自动转换方法

在 RNLreq 的基础上, 本文通过模型转换技术实现 RNLreq 到 AADL 初始模型的自动转换.

从限定自然语言需求模板到 AADL 模型自动转换的主要包括两部分:

1) 数据字典与领域词库的转换: 数据字典转换到 AADL 数据构件 (*Data Component*) 及数据构件的属性 (*Properties*), 例如第二章示例中的 GNCC 控制数据需在数据字典中定义, 并将被转换到 AADL 的数据构件, 其数据组成将转换到数据构件的子构件 (*subcomponent*). 由于领域词库主要用于领域专有词汇的“注册”与中英文映射, 因此无需转换到特定的 AADL 模型元素.

2) 限定自然语言需求模板的转换: 在 RNL2AADL 自动转换的实现过程中, 文本提出中间转换模型 RAInterM (RNL2AADL Intermediate Model), 将需求到 AADL 模型的转换(多对多关系)拆分成限定自然语言需求到 RAInterM (多对一关系)以及 RAInterM 到 AADL (一对多关系)的转换关系, 从而简化了转换的实现并保证了转换的可扩展性.

4.1 RNLreq 元模型

基于第二章给出的 RNLreq 的 EBNF 语法规则, 本文给出了 RNLreq 的元模型如图 1 所示.

RNLreq 元模型主要分为三个部分: *DataDictionary & Glossary*, *Template*, 以及 *Sentence*. *DataDictionary & Glossary* 表示数据字典及领域词库, 其中 *DataWord* 和 *Term* 分别表示数据和领域词汇. *Template* 表示各层模板, 每一类模板都

继承自 *AbstractTemplate*, 且低层模板可以是高层模板的父模板. 各层模板中每一类需求都是由一组限制句式描述的, 每一类限制句式继承于 *Sentence*, 而 *Sentence* 中的 *gen()* 方法中定义了该句式的转换规则.

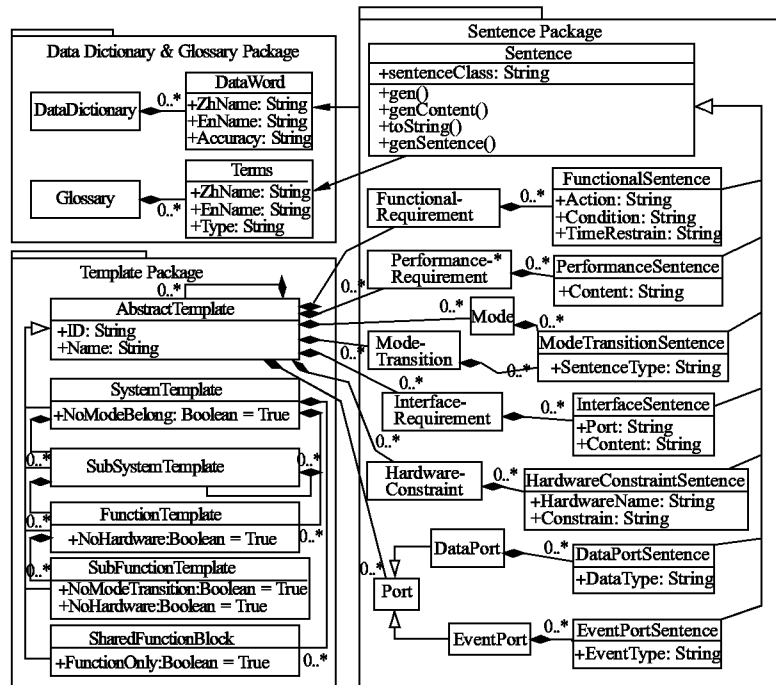


图 1 RNLreq 元模型

Fig. 1 Meta-Model of RNLreq

#### 4.2 RAInterM 元模型

为了简化 RNL2AADL 的转换, 并保证转换的可扩展性, 本文提出 RAInterM 中间模型, 并给出其元模型如图 2 所示:

其中 Model 是顶层概念, 表示整个系统; *Component* 表示系统中的实体; *StateMachine* 可以分为 *MTStateMachine* 和

*BHVStateMachine* 两种, 分别对应 RNLreq 中的模式变换和功能处理, 以及 AADL 中的 *mode transition* 和 *Behavior Annex*; *Connection* 和 *Port* 对应 RNLreq 中的输入输出, 表示系统中构件间的通信通道, 其中 *Port* 中定义了端口的数据类型, *Property* 定义了 *Component* 等元素的约束性质。

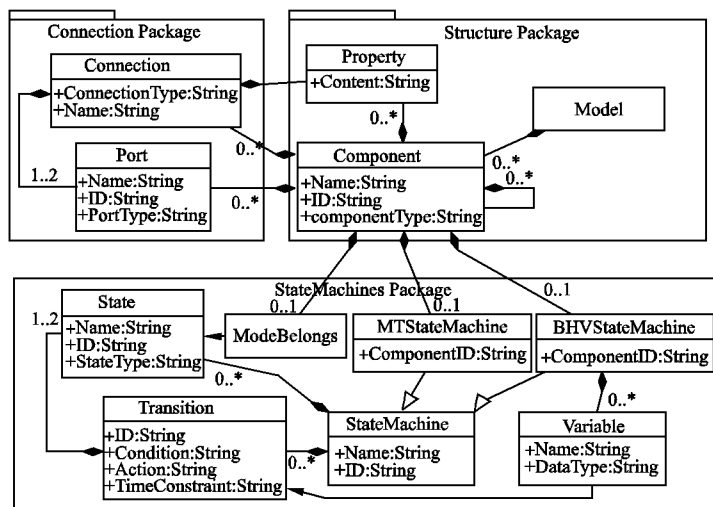


图 2 RAInterM 元模型

Fig. 2 Meta-Model of RAInterM

#### 4.3 RNLreq 到 RAInterM 的转换

基于 RNLreq 及 RAInterM 元模型, 本文给出 RNLreq 到 RAInterM 的转换算法如算法 1 所示:

##### 算法 1.

Transformation from RNL2AADL to RAInterM

Input: RNL2AADL

**Output:** RAInterM

```

1: for each Template t in RNL2AADL.getTemplates do
2:   Component c = new Component(t.getType);
3:   for each Port p in RAInterM.getPorts do
4:     p.gen(RAInterM, c, p.PortType);
5:   end for
6: end for
7: for each Port p in RAInterM.getPorts do
8:   if p.NoSameNamePort then
9:     p.type = DATAACCESS;
10:    RAInterM.add(data = newComponent(DATA));
11:    RAInterM.addDataAccessConnections(p, data);
12:   else
13:    RAInterM.addConnections(p, p.getSameNamePort);
14:   end if
15: end for
16: for each Sentence s in t.getRequirements do
17:   s.gen(RAInterM, c);
18:   // Each type of requirements transformed into different parts
      in RAInterM
19: end for

```

大致思路如下:

1) 系统框架的转换: 首先将整个系统的框架转换到中间模型 RAInterM, 其中模板转换到的 *Component* 组件, 输入输出转换到 *Port*, 其中端口上添加的约束性质对应转换到端口所关联 *Connection* 的 *Property* 中。

2) *Connection* 生成: 在需求模板中, 联通的输入输出端口在定义时数据类型是相同的。因此, 相同数据类型的 *Port* 通过 *Connection* 连接起来, 而单独的 *Port* 通过共享数据组件以共享数据访问的方式建立 *Connection*, 其中输入输出端口上的约束性质转换到 *Port* 的 *properties*。

3) 需求的转换: 针对 RNLreq 中定义的多种需求, 如功能处理、性能需求、接口需求、安全性设计约束、模式变换等, 通过在各需求描述句式的 *gen()* 方法中定义各种需求描述句式到中间模型的转换方法, 实现需求描述到中间模型的转换, 如模板 A 发送数据 D 到模板 B, 可以转换成 RAInterM 模型中, *component A* 的 *out port D* 与 *component B* 的 *in port D* 相连, 并在 *component A* 与 *B* 的最近公共父组件的 *BHVStateMachine* 中描述对应子组件发送数据这一行为。

#### 4.4 RAInterM 到 AADL 的转换

在从 RNLreq 到 RAInterM 的转换完成之后, 我们再给出从 RAInterM 到 AADL 的转换规则, 其中 AADL 的元模型可以在开源 AADL 工具环境 (OSATE) 中直接获得。本文给出的 RNLreq 到 AADL 的转换算法, 如算法 2 所示:

**算法 2.**

Transformation from RAInterM to AADL

**Input:** RAInterM

**Output:** AADL

```

1: for each Component c in
      RAInterM.getComponents do
2:   if c.isSystem then
3:     AADL.add(new System n);
4:   else if c.isProcess then

```

```

5:     AADL.add(new Process n);
6:   else if c.isThread then
7:     AADL.add(new Thread n);
8:   else if c.isSubprogram then
9:     AADL.add(new Subprogram n);
10:  else
11:    AADL.add(new Abstract n);
12:  end if
13:  AADL.addInstance(n.newInstance);
14:  c.getPorts -> n.features;
15:  c.SubComponents -> n.instance.subcomponents;
16:  c.Connections -> n.instance.connections;
17:  c.MTStateMachine.getStates -> n.instance.modes;
18:  c.MTStateMachine.getTransitions -> n.instance.transitions;
19:  c.Properties -> n.instance.properties;
20:  c.BHVStateMachine -> n.instance.BA;
21: end for

```

RAInterM 到 AADL 的转换主要包括两部分:

1) AADL 组件类型的转换, 主要是对 AADL 组件类型和端口声明的生成, 其中 RAInterM 中 *component* 转换到 AADL 中的软硬件构件, 如 *system*、*process*、*thread*、*bus*、*processor*、*device* 等, 而 *port* 则转换到 AADL 声明中的 *features*, *port* 上对应的约束规则转换到 AGREE Annex 的 *assume* 与 *guarantee*。

2) AADL 组件实例化的转换, 主要包括 AADL 模型的主体信息, 其中, RAInterM 中的 *component* 间关系转换到 AADL 的软硬件组件关系, *connection*、*property*、*statemachine* 等转换到 AADL 中的连接 (*Connection*)、性质 (*Property*)、模式变化 (*mode transition*)、行为附件 (*behavior annex*) 等。

完成初始 AADL 模型的自动转换之后, 还需要通过精化的方式进一步完善 AADL 设计模型, 使其能够更加完整地表达安全关键软件设计。

#### 4.5 AGREE Annex 自动生成

通过 RNLreq 到 AADL 的转换, 生成初始 AADL 设计模型。为了增强 AADL 设计模型对组合验证的支持, 需在 AADL 设计模型中添加组合验证所需的约束信息。因此, 本文面向 AGREE 假设保证附件, 提出一个面向组合验证的性质描述模板, 用以辅助用户增加组合验证所需的相关信息, 其结构如表 5 所示。

表 5 组合验证性质描述模板

Table 5 Compositional verification specification template

需求模板名称	对应 AADL 设计模型中的一个组件
假设	该组件所需的外界条件, 如输入值的范围等
保证	该组件在外界条件满足的前提下所能保证的性质
约束性质	该组件中涉及的数据的约束条件, 可以是端口数据或临时变量的约束不等式
临时变量	中间计算过程的临时变量
等式	中间数据计算过程

其中, 需求模板名称对应 AADL 组件的名称, 假设和保证分别表示对应 AADL 组件正常工作所需的外界条件及在该外界条件的前提下该组件所能保证的性质, 约束性质、临时

变量、等式等为验证求精过程中增加的中间过程。其中,约束性质可以从需求模板中性能需求、接口需求中抽取出来的约束条件;临时变量和等式可以是需求模板中的功能需求、模式变换中抽取的组件内功能执行逻辑。

在工程师通过组合验证求精模板以自然语言的方式增加相关描述之后,可以自动转换成 AADL AGREE Annex 并插入到 RNL2AADL 生成的初始 AADL 模型中。

本文给出组合验证性质描述模板到 AADL AGREE Annex 的转换算法,如算法 3 所示:

### 算法 3.

### Transformation from VRT to AGREE

**Input:** VRT, AADL

**Output: AGREE**

```

1: for each Component c in AADL.getComponents do
2:   VRTTemplate v = VRT.getVRTTemplate(c) ;
3:   for each AssumeStmt as in v.getAssumeStmts do
4:     AGREE.getComponent(c).addAssumeStmt(cs) ;
5:   end for
6:   for each GuaranteeStmt as in v.getGuaranteeStmts do
7:     AGREE.getComponent(c).addGuaranteeStmt(cs) ;
8:   end for
9:   for each ConstStatement cs in v.getConstStatements do
10:    AGREE.getComponent(c).addConstStatement(cs) ;
11:   end for
12:   for each Property p in v.getProperties do
13:    AGREE.getComponent(c).addProperty(p) ;
14:   end for
15: end for

```

在此基础上,用户可基于 OSATE 中集成的 AGREE 假设保证插件对 AADL AGREE Annex 进行组合验证.

## 5 工具实现和案例分析

## 5.1 工具实现

基于限定自然语言的需求模型到 AADL 设计模型的转换工具是基于 AADL 开源环境 OSATE 开发的,按照 MVC 的设计思想,将工具拆分为前端(模板、领域词库、数据字典的限定自然语言输入方式等)、中间模型(RAInterM,数据字典、领域词库的数据模型等)和(后端)中间模型到 AADL 的代码生成三个部分.其中,增加的组合验证性质分别定义在模板、数据字典、Req2AADL 和代码生成等部分中,工具程序框架如图 3 所示.

## 5.2 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,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 (Attitude and Orbit Control System, 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。

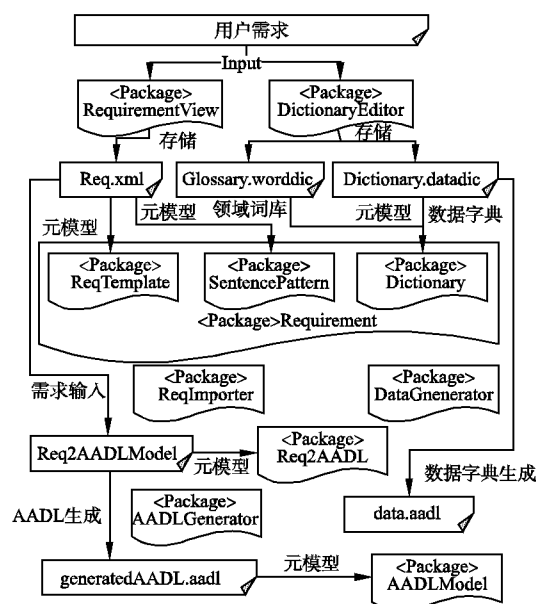


图3 原型工具实现框架

**Fig. 3** Framework of prototype tool

### 5.2.1 需求建模及 AADL 模型生成

GNC 案例对应的数据字典、领域词库如图 4 所示:

[illegible]

图4 GNC案例数据字典及领域词库

Fig.4 GNC DataDictionary & Glossary

建立数据字典和领域词库后,通过需求模板进行 GNC 需求规约,并对需求进行精化,此处以 GNC 中姿态控制系统为例,需求结构如图 5 所示,同时以图 6 中的消初偏模块(即消除星箭分离所产生的偏差)为例,给出其功能需求描述:

通过工具生成的 AADL 模型如图 7 所示。

### 5.2.2 模型验证

在生成 GNC 案例的 AADL 模型之后,我们考虑对生成的 AADL 模型进行形式化验证.在已有工作<sup>[31]</sup>中,我们已实现了单构件验证工具 AADL2TASM/UPPAAL.本文在对生成 AADL 模型进行单构件验证的基础上,通过 AGREE 插件工具对 AADL 模型进行组合验证.

首先,在 RNL2AADL 的基础上,生成了 AADL 初始设计



模型,以其中 `attitude_filter_init` 模块为例,一个简化的初始设计模型的代码如图 5 所示.

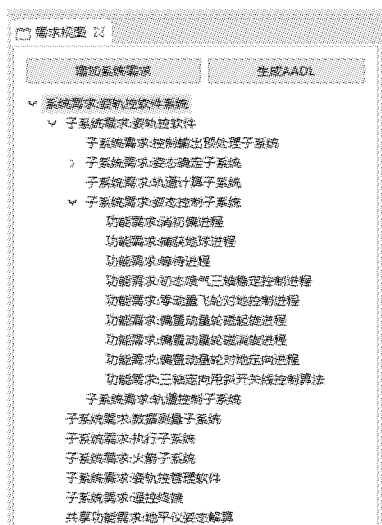


图 5 RNLreq 描述 GNC 需求结构

**Fig. 5** Requirement structure of GNC in RNLreq

**thread attitude\_filter\_init**

## features

**Attitude\_Measure: in data port**

```
Base_Types::Integer;
```

Attitude\_Angular\_Belocity:in data port

```
Base_Types :: Integer;
```

Attitude\_Estimation:out data port

```
Base_Types : : Integer;
```

**end attitude\_filter\_init;**

在此基础上,通过 AGREE 验证精化模板,对 attitude\_filter\_init 模块进行精化,添加假设、保证、约束性质等组合验证信息.精化后 Attitude\_filter\_init 模块对应转换出的 AGREE Annex 如下所示:

**annex agree** { \* \*

```
const Attitude_Rate;real = 1.2;
```

**eq** Compute\_Attitude : **real** = Atti-

$$\text{tude\_Angular\_Velocity} * \text{Attitude\_Rate};$$

**assume** "Attitude\_Filter input range " : Attitude\_Measure < 5;

**assume** "Attitude\_Filter input range " : Attitude\_Angular\_Be-

```
locity > 9;
```

**guarantee " Attitude\_Filter output range " : Attitude\_Estimation**

```
< Attitude_Measure + Compute_Attitude;
```

\* \* }

编号	基本流	
1	在连续3ms内,执行(将接收到的X轴陀螺初始角速度发送到姿态角速度X输出端口,将接收到的Y轴陀螺初始角速度发送到姿态角速度Y输	添加
2	若X轴陀螺初始角速度积分-角速度积分<=12、Z轴陀螺初始角速度积分-角速度积分<=12、X轴陀螺初始角速度-角速度值<=0.3、	删除
3	执行(将接收到的0发送到飞轮转速输出端口)。	
4	执行(将接收到的0发送到磁力矩器三轴驱动电流指令输出端口)。	上移
5	在连续5sec内,若X轴陀螺初始角速度积分-角速度积分<=12、卫星姿态偏差角Z-角度值<=26、Z轴陀螺初始角速度积分-角速度积	下移

图6 GNC案例消初偏模块功能需求

Fig. 6 Functional requirement of initial offset elimination module in GNC

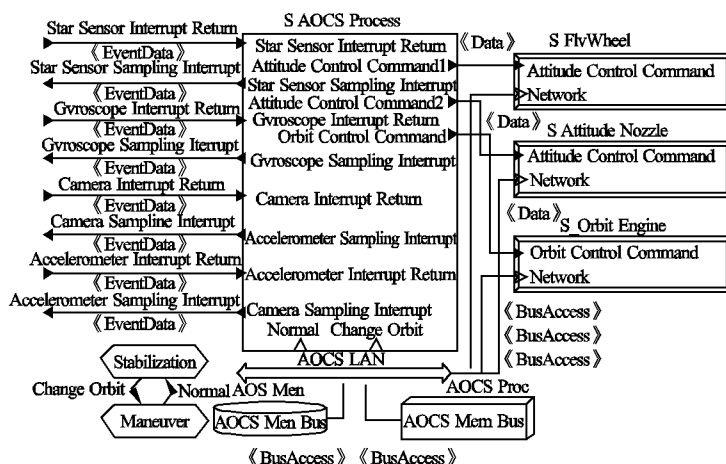


图7 AADL模型图形化表示

Fig. 7 AADL model expressed in graphics

最后,通过 Ostate 的 AADL AGREE 插件工具运行执行对 AGREE 附件的组合验证.

本文分别对 GNC 中的 6 项非功能性质进行验证,包括基于测速仪和节流阀的速度验证、基于导航相机和加速度计的加速度验证、对 AOCS 子系统执行周期的验证、对 AOCS 外

设电源分配的验证、对各执行机构并行工作情况的验证、对陀螺仪是否正常工作的验证。

以速度验证为例,验证性质为当输入的速度值为 0 到 150 范围内时,保证每个逻辑时间的速度差的绝对值小于 2。基于此,通过本文提出的方法生成了相应构件中的 *assume* 和

guarantee,并通过需求精化模板增加了中间计算过程,得到的父组件 Terminal 中的 AGREE Annex 代码如下:

```
annex agree { * *
    const max_accel:real = 2.0;
    assume" target speed is positive" :
    Targer_speed_Input.val >= 0.0;
    assume" reasonable target speed" :
    Targer_speed_Input.val < 150.0;
    property const_tar_speed = Agree_Nodes.H
    (Targer_speed_Input.val = prev(Targer_speed_Input.val, 0.0));
    guarantee" actual speed is less than constant target speed" :
    const_tar_speed => (Actual_Out.val <= Targer_speed_Input.val);
    guarantee" acceleration is limited" : Agree_Nodes.abs(Actual
    _Out.val - prev(Actual_Out.val, 0.0)) < max_accel;
    * * };
```

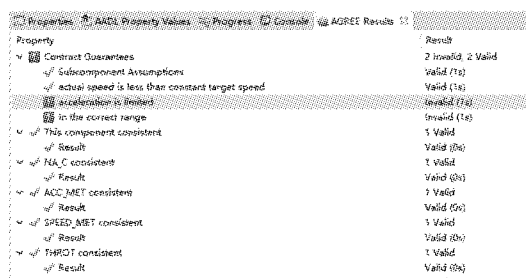
测速仪中 AGREE 代码如下:

```
annex agree { * *
    const P:real = 0.2;
    const D:real = 0.1;
    const I:real = 0.1;
    eq e:real = Targer_speed.val - Actual.val;
    eq e_int:real = prev(e, 0.0) + e;
    eq e_dot:real = prev(e, 0.0) - e;
    eq u:real = P * e + D * e_dot + I * e_int;
    guarantee" Actuator" : Actuator_Input = u;
    guarantee" Differ Speed" : Differ_speed.val = e;
    * * };
```

节流阀中 AGREE 代码如下:

```
annex agree { * *
    guarantee" Throttle_Behavior" : Actual.val = prev(Actual.
    val, 0.0) + 0.1 * Actuator_Input;
    * * };
```

在此基础上,可以通过 AGREE 组合验证插件对生成的 AGREE 代码进行验证,验证结果如图 8 所示。



Property	Result
Contract Guarantees	2 Invalid, 2 Valid
Subcomponent Assumptions	Valid (1s)
actual speed is less than constant target speed	Valid (1s)
acceleration is limited	Invalid (1s)
In the correct range	Invalid (1s)
This component consistent	1 Valid
Result	Valid (1s)
NAC consistent	1 Valid
Result	Valid (1s)
ACC/MET consistent	1 Valid
Result	Valid (1s)
SPEED/MET consistent	1 Valid
Result	Valid (1s)
THROT consistent	1 Valid
Result	Valid (1s)

图 8 使用 AGREE 工具验证速度

Fig. 8 Verifying speed properties with AGREE

图中可以看出,在给定的 assume 满足,及给出的中间计算过程的情况下,检测出性质“acceleration is limited”无法满足,从而可以说明模型中关于速度控制的功能构件存在问题,进而说明需求中存在不合理的设计。

## 6 相关工作

本文的相关工作主要包括 3 部分:

- 1) 限定自然语言需求建模;
- 2) 自然语言需求到设计模型的自动转换;
- 3) 基于契约的组合验证。

### 6.1 限定自然语言需求建模

自然语言具有很好的表达能力,但是存在着二义性、模糊性以及难以自动化分析处理等缺点。Rolls Royce 公司的 Alistair Mavin 等人针对用户需求难以描述和表达的问题,在大量工程经验的基础上,提出了一种 EARS(Easy Approach to Requirements Syntax)<sup>[17,18]</sup> 方法进行需求规约。该方法通过将需求分为 6 种类型,有效简化了需求的描述与表达难度,同时有效降低自然语言的二义性。德国帕德博恩大学的 Jörg Holtmann 教授等人针对汽车工程领域的软件系统的需求规约展开研究,提出了一种受控自然语言(Controlled Natural Language, CNL)用于汽车领域嵌入式软件需求描述。CNL 还可以实现需求的自动化验证,以及对需求的不一致性和不完整性进行检测<sup>[32]</sup>。

挪威 Simula 实验室的 Yue Tao 教授提出了一种改进的受限用例建模方法 RUCM(Restricted Use Case Modeling)<sup>[14-16]</sup>, RUCM 包含一个相对完善的用例文本说明模板和一系列自然语言限定规则(Restricted Rules),使得用例描述更加易于理解和精确,减少歧义,并且可以自动化产生分析模型。目前,已有相应的研究方法和工具支持将 RUCM 的用例模型自动转换为初步的 UML 分析模型(类图、活动图、顺序图等)。此外,针对 RUCM 较难描述机载嵌入式软件的安全性需求的不足,北航的吴际等人对 RUCM 进行了扩展,添加相应的模板与限定规则,提出 Safety RUCM,使其支持安全性需求的规范化描述<sup>[33]</sup>。另外,面向对象管理组织 OMG(Object Management Group)在正在制定的 SysML 2.0 版本<sup>[18]</sup>中也提出了限定自然语言需求建模的思想,旨在提高需求描述的精确性和有效性。

### 6.2 自然语言需求到设计模型的自动转换

如何将基于自由文本、受限的自然语言或半形式化的设计模型转换到形式化模型是一个热点问题。

英国曼彻斯特大学 Keletso J. Letsholop<sup>[12]</sup>等为了实现自由文本需求到设计模型的自动化转换,提出了 TRAM(Tool for Transforming Textual Requirements into Analysis Models)工具,实现了从自由文本描述的需求到设计模型的自动化转换。在转换过程中,采用了 Stanford 分词器将自然语言文本进行切分,并通过 SOM(Semantic Object Models)构造器建立中间 SOMi 模型,模型生成器(Model Generator)将 SOMi 模型通过模板匹配的方式映射到目标模型,最后通过模型验证器(Model Validator)对生成的设计模型进行分析验证并反馈给领域专家,然后进行下一轮需求迭代。

意大利 Insubria 大学的 Pietro Colombo 和 Ferhat Khenek<sup>[34]</sup>等针对需求与设计之间存在鸿沟的问题提出了一种从需求文档自动抽取生成早期设计模型的方法,该方法是以通过抽取经分析后产生的问题表(Problem Frames)中的信息,采用 ATL(ATLAS Transformation Language)模型转换语言制定转换规则,分四个步骤分别抽取 Blackboard 和 Knowledge Source 信息并不断精化,最后生成 SysML 模块定义图、活动图等设计模型。

代尔夫特理工大学的 Van der Gaag 等<sup>[35]</sup>提出了一种将 SysML 转换到 SLIM 的方法 SSTM(SysML to SLIM Transformation Methodology),支持 SysML 模型到 SLIM 模型的自动化转换,使得缺乏软件工程相关知识的系统工程师可以直接进行 SysML 系统建模,通过自动转换后使用 COMPASS 工具集直接分析系统的动态功能行为并发现开发过程中可能出现的约束和错误,有助于减少系统开发的时间和成本。

### 6.3 基于契约的组合验证

契约理论最早是由 Reussner 等<sup>[36]</sup>首次引入到基于构件的软件设计与开发过程中的,通过将契约从传统 C/S 架构的设计方法推广到基于组合理论的组合契约,从而使其在组合设计与开发过程中得到应用。

Atkinson 等<sup>[37]</sup>提出了一种组合时动态契约检查方法.通过设计一种 Built-in Test 构件,实现了测试任务的内建.该方法主要包括两个部分,Tester 构件和 Testing 构件,其中 Tester 构件负责测试构件是否实现了其外部行为特性,Testing 构件负责测试外部环境提供的服务是否满足构件的需求,通过在软件中植入检测构件,提取相关运行时信息进行分析,以确定构件可组合性<sup>[38]</sup>.

John D. Backes 等<sup>[37]</sup>在研究使用规约模式把自然语言需求简化成形式化规约的结构的过程中,通过增加需求规约语言 RSL(Requirements Specification Language)模式和 calendar automata 的方式,支持实时系统的分析并证明了 AGREE 中隐藏的性质:所有子组件的契约满足,保证系统的契约满足。

## 7 结束语

本文提出了一种基于限定自然语言的需求规约方法 RN-Lreq,通过结构化的数据字典、领域词库、需求模板及限定句式,约束需求的描述方式以减少其二义性;其次给出了 RNLreq 到 AADL 模型的自动转换,包括 RNLreq 到中间模型 RAInterM 及 RAInterM 到 AADL 的转换规则,简化了限定自然语言需求到 AADL 模型的自动转换;此外,提出了 AADL 模型组合验证性质描述模板,并自动生成 AADL 的组合验证附件 AGREE Annex,以支持对 AADL 模型进行组合验证;最后,在 AADL 开源建模环境 OSATE 实现了原型工具,并基于实际工业案例 GNC 系统进行了案例分析。

在未来的工作中,在需求建模方面,将进一步考虑对限定自然语言需求 RNLreq 的扩展,增强 RNLreq 对安全性、可靠性、实时性等非功能属性的表达能力;在模型验证方面,将研究从限定自然语言需求模型中自动生成验证性质,包括线性时序逻辑公式和分支时序逻辑公式,形成完整的模型验证方法。

### References:

- [1] Leveson N. Engineering a safer world: systems thinking applied to safety[M]. MIT Press, 2011.
- [2] Rierson L. Developing safety-critical software: a practical guide for aviation software and DO-178C compliance[M]. CRC Press, 2013.
- [3] DO R. 178C. Software considerations in airborne systems and equipment certification[S]. RTCA. Inc. Washington DC, USA, 2011.
- [4] DO R. 331. Model-based development and verification supplement

to DO-178C and DO-278A [S]. RTCA. Inc. Washington, DC, 2011.

- [5] DO R. 333. Formal methods supplement to DO-178C and DO-278A[S]. RTCA. Inc. Washington, DC, 2011.
- [6] Yue T, Briand L C, Labiche Y. A systematic review of transformation approaches between user requirements and analysis models [J]. Requirements Engineering, 2011, 16(2): 75-99.
- [7] Vilela J, Castro J, Martins L E G, et al. Integration between requirements engineering and safety analysis: a systematic literature review [J]. Journal of Systems and Software, 2017, 125: 68-92. <https://www.sciencedirect.com/science/article/pii/S0164121216302333>, doi:10.1016/j.jss.2016.11.031.
- [8] China Aerospace Science and Technology Corporation. Technical Guide and Implementation of Space Software Engineering [Z]. China Aerospace Science and Technology Corporation, 2005.
- [9] Martins L E G, Gorschek T. Requirements engineering for safety-critical systems: overview and challenges [J]. IEEE Software, 2017, 34(4): 49-57.
- [10] Martins L E G, Gorschek T. Requirements engineering for safety-critical systems: a systematic literature review [J]. Information and Software Technology, 2016, 75: 71-89.
- [11] SAE S A E. Architecture analysis & design language [S]. Embedded Computing Systems Committee, SAE, 2017.
- [12] Letsholo K J, Zhao L, Chioasca E V. TRAM: a tool for transforming textual requirements into analysis models [C]. Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, IEEE Press, 2013: 738-741.
- [13] Bajwa I S, Lee M, Bordbar B. Translating natural language constraints to OCL [J]. Journal of King Saud University-Computer and Information Sciences, 2012, 24(2): 117-128.
- [14] Yue T, Briand L C, Labiche Y. Facilitating the transition from use case models to analysis models [J]. ACM Transactions on Software Engineering and Methodology, 2013, 22(1): 1-38. doi:10.1145/2430536.2430539.
- [15] Zhang G, Yue T, Wu J, et al. Zen-RUCM: a tool for supporting a comprehensive and extensible use case modeling framework [C]. Demos/Posters/StudentResearch@ MODELS, 2013: 41-45.
- [16] Yue T, Briand L C, Labiche Y. aToucan: an automated framework to derive UML analysis models from use case models [J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2015, 24(3): 1-52.
- [17] Mavin A, Wilkinson P, Harwood A, et al. Easy approach to requirements syntax (EARS) [C]. Requirements Engineering Conference, RE'09. 17th IEEE International, IEEE, 2009: 317-322.
- [18] Gregory S. Easy EARS: rapid application of the easy approach to requirements syntax [C]. 19th IEEE International Requirements Engineering Conference (RE'11), 2011.
- [19] Lasalle J, Bouquet F, Legeard B, et al. SysML to UML model transformation for test generation purpose [J]. ACM SIGSOFT Software Engineering Notes, 2011, 36(1): 1-8.
- [20] van der Gaag J T. SysML to SLIM transformation methodology: Connecting model-based space systems engineering and model-based software engineering [D]. Holland: Delft University of Technology, 2017: 1-129.
- [21] Bodeveix J-P, Filali M, Garnacho M, et al. Towards a verified

- transformation from AADL to the formal component-based language FIACRE [J]. *Science of Computer Programming*, 2015, 106:30-53. <http://oatao.univ-toulouse.fr/14914/>, doi:10.1016/j.scico.2015.03.003.
- [22] Chkouri M Y, Robert A, Bozga M, et al. Translating AADL into BIP-application to the verification of real-time systems[C]. *International Conference on Model Driven Engineering Languages and Systems*, Springer, Berlin, Heidelberg, 2008:5-19.
- [23] Abdoul T, Champeau J, Dhaussy P, et al. AADL execution semantics transformation for formal verification[C]. *Engineering of Complex Computer Systems (ICECCS 2008)*, IEEE, 2008:263-268.
- [24] Sokolsky O, Lee I, Clarke D. Schedulability analysis of AADL models[C]. *Parallel and Distributed Processing Symposium (IP-DPS 2006)*, IEEE, 2006.
- [25] Ölveczky P C, Boronat A, Meseguer J. Formal semantics and analysis of behavioral AADL models in Real-Time Maude[M]. *Formal Techniques for Distributed Systems*, Springer, Berlin, Heidelberg, 2010:47-62.
- [26] Jahier E, Halbwachs N, Raymond P, et al. Virtual execution of AADL models via a translation into synchronous programs[C]. *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software*, ACM, 2007:134-143.
- [27] Ma Y, Talpin J P, Gautier T. Virtual prototyping AADL architectures in a polychronous model of computation[C]. *Sixth ACM and IEEE International Conference on Formal Methods and Models for Co-design, MEMOCODE*, 2008:139-148.
- [28] Yang Z, Hu K, Ma D, et al. From AADL to timed abstract state machines: a verified model transformation[J]. *Journal of Systems and Software*, 2014, 93:42-68. doi:10.1016/j.jss.2014.02.058, <http://oatao.univ-toulouse.fr/12950/>.
- [29] Cofer D, Gacek A, Miller S, et al. Compositional verification of architectural models [C]. *NASA Formal Methods Symposium*, Springer, Berlin, Heidelberg, 2012:126-140.
- [30] Yang Zhi-bin, Pi Lei, Hu Kai, et al. AADL: an architecture design and analysis language for complex embedded real-time systems [J]. *Journal of Software*, 2010, 21(5):899-915.
- [31] Wang Fei, Yang Zhi-bin, Huang Zhi-qiu, et al. Generating the AADL model based on restricted natural language requirement template[J]. *Journal of Software*, 2018, 29(8):2350-2370.
- [32] Fockel M, Holtmann J, Meyer J. Semi-automatic establishment and maintenance of valid traceability in automotive development processes[C]. *Proceedings of the Second International Workshop on Software Engineering for Embedded Systems*, IEEE Press, 2012:37-43.
- [33] Wu X, Liu C, Xia Q. Safety requirements modeling based on RUCM [C]. *Computing, Communications and IT Applications Conference (ComComAp)*, IEEE, 2014:217-222.
- [34] Colombo P, Khendek F, Lavazza L. Bridging the gap between requirements and design: an approach based on problem frames and SysML [J]. *Journal of Systems and Software*, 2012, 85(3):717-745.
- [35] Reussner R, Poernomo I, Schmidt H. Contracts and quality attributes for software components[C]. *Proceedings of the Eighth International Workshop on Component-Oriented Programming*, 2003.
- [36] Atkinson C, GroB H G. Built-in contract testing in model-driven, component-based development [C]. *ICSR Workshop on Component-Based Development Processes*, 2002.
- [37] Lynch N, Segala R, Vaandrager F. Hybrid i/o automata[J]. *Information and Computation*, 2003, 185(1):105-157.
- [38] Wang Bo, Bai Xiao-ying, He Fei, et al. Survey on modeling and verification techniques of composable embedded software [J]. *Journal of Software*, 2014, 25(2):234-253.

#### 附中文参考文献:

- [8] 航天软件工程实施技术指南及范例[Z]. 中国航天科技集团公司, 2005.
- [30] 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[J]. *软件学报*, 2010, 21(5):899-915.
- [31] 王飞, 杨志斌, 黄志球, 等. 基于限定自然语言需求模板的 AADL 模型生成方法[J]. *软件学报*, 2018, 29(8):2350-2370.
- [38] 王博, 白晓颖, 贺飞. 可组合嵌入式软件建模与验证技术研究综述[J]. *软件学报*, 2014, 25(2):234-253.