软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(4):904–933 [doi: 10.13328/j.cnki.jos.006223]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

安全关键异构软件混合建模及代码生成方法*

宗 喆[1,2], 杨志斌[1,2], 袁胜浩[1,2], 周 勇[1,2], Jean-Paul BODELEIX[3], Mamoun FILALI[3]

[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信部重点实验室(南京航空航天大学),江苏 南京 211106)

[3](IRIT-University of Toulouse, Toulouse 31062, France)

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

摘 要: 随着系统复杂性的急剧增加,未来安全关键软件越来越多地采用异构构件组合架构,各种构件可能使用不同的计算模型或实现语言,使得整个软件系统呈现异构性.因此,设计此类系统需要使用复杂的建模方法.AADL (architecture analysis and design language)是一种安全关键系统体系结构多范式建模语言,丰富的可表达方式和可扩展性使其成为安全关键异构软件设计的重要选择.提出一种 AADL 和 SDL(specification and description language)混合建模方法,支持以自底向上的方式对安全关键软件系统进行混合建模,并给出面向多核处理器平台的代码自动生成方法.首先,通过扩展 AADL 属性集,以支持使用 SDL 建模语言表达软件构件的功能行为.其次,以 Ada 作为目标语言,给出 AADL-SDL 混合模型的多任务代码生成方法.最后,实现了原型工具支持 AADL 和 SDL 混合建模及多任务 Ada 代码生成,并基于导航、制导与控制系统案例对所提方法的有效性进行分析.

关键词: 安全关键异构软件;混合建模;AADL;SDL;多核;代码自动生成

中图法分类号: TP311

中文引用格式: 宗喆,杨志斌,袁胜浩,周勇,Jean-Paul BODEVEIX,Mamoun FILALI.安全关键异构软件混合建模及代码生成方法.软件学报,2021,32(4):904–933. http://www.jos.org.cn/1000-9825/6223.htm

英文引用格式: Zong Z, Yang ZB, Yuan SH, Zhou Y, Bodeveix JP, Filali M. Co-modeling and code generation for safety-critical heterogeneous software. Ruan Jian Xue Bao/Journal of Software, 2021,32(4):904–933 (in Chinese). http://www.jos.org.cn/1000-9825/6223.htm

Co-modeling and Code Generation for Safety-critical Heterogeneous Software

ZONG Zhe[1,2], YANG Zhi-Bin[1,2], YUAN Sheng-Hao[1,2], ZHOU Yong[1,2], Jean-Paul BODELEIX[3], Mamoun FILALI[3]

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

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

[3](IRIT-University of Toulouse, Toulouse 31062, France)

Abstract: Safety-critical systems have evolved to use heterogeneous components to implement complex requirements, each component may adopt different computation models or modeling languages. Therefore, it is necessary to use complex modeling approaches to design those systems. AADL, as a multi-paradigm modeling language for safety-critical system architecture, is a good choice to design safety-critical heterogeneous systems because of its rich expressibility and well scalability. This study proposes a bottom-up AADL-SDL co-modeling approach that integrates functionality modeled by SDL through the AADL architecture model and provides a multi-task code

* 基金项目: 国家自然科学基金(62072233); 航空科学基金(201919052002); 中央高校基本科研业务费专项资金(NP2017205); 国防基础科研项目(JCKY2020205C006)

Foundation item: National Natural Science Foundation of China (62072233); Aviation Science Fund of China (201919052002); Fundamental Research Funds for the Central Universities (NP2017205); National Defense Basic Scienfic Research (JCKY 2020205C006)

本文由“面向领域的软件系统构造与质量保障”专题特约编辑潘敏学教授、魏峻研究员、崔展齐教授推荐.

收稿时间: 2020-09-13; 修改时间: 2020-10-26; 采用时间: 2020-12-19; jos 在线出版时间: 2021-01-22

© 中国科学院软件研究所 http://www.jos.org.cn

generation approach for multi-core platforms. Firstly, AADL property sets are extended to support the capability of modeling functionality. Secondly, a multi-task code generation approach is proposed to transform AADL-SDL models to Ada code. Finally, a prototype tool is implemented to support AADL-SDL co-modeling and multi-task Ada code generation. The effectiveness of the method proposed in this study is analyzed based on the guidance, navigation, and control system scenarios.

Key words: safety-critical heterogeneous software; co-modeling; AADL; SDL; multi-core; code generation

安全关键软件(safety-critical software)[1]是指应用于航空、航天和核能等领域的安全关键系统中,且其运行失效会引起系统处于危险状态,从而导致人员伤害、重大财产损失或者环境破坏等灾难性后果的一类软件,它对功能正确性、实时性、安全性等性质有极高的要求.随着系统复杂性的急剧增加,未来安全关键软件越来越多地采用异构构件组合架构,即构件由不同供应商以 OEM(original equipment manufacturer)方式提供,各构件可能具有不同特性,例如使用不同的计算模型(状态机、同步数据流、异步执行模型、连续时间模型等)或实现语言,使得整个软件系统呈现异构性[2,3].异构软件系统的执行与交互语义也从完全同步发展到全局异步-局部同步 GALS(globally asynchronous locally synchronous)方式,即不同构件具有各自的时钟控制(称为多时钟,multi-rates),构件之间采用异步通信方式.

近年来,多范式建模方法 MPM(multi-paradigm modeling)[4,5]已成为安全关键异构软件设计的研究热点.多范式建模方法可以通过模型转换、模型组合、混合建模和混合仿真等方式对软件系统中使用的不同领域知识、不同视图以及不同抽象层次进行建模,充分发挥各种建模语言的描述能力.在安全关键系统领域,常用的建模语言主要包括 Modelica[6]、SysML[7]、UML Marte[8]、AADL[9]、EAST-ADL[10]、SCADE[11]、Simulink[12]、Ptolemy II[13]等.其中,AADL(architecture analysis and design language)是由美国汽车工程师协会 SAE 提出的面向安全关键系统的一种建模语言标准(SAE AS5506).AADL 以层次化构件的方式表达系统的软硬件架构.一方面,AADL 可以非常方便地表达多时钟、GALS、异步通信等异构软件特征;另一方面,AADL 提供定义新属性集和附件(annex)等多种扩展方式,使得 AADL 逐渐成为安全关键异构软件多范式建模的重要选择.

在支持多领域知识的多范式建模方面:Modelica 是一种面向对象的建模语言,用于对大型、复杂和异构系统进行建模,并支持多领域建模.例如,航空航天领域的安全关键异构系统涉及机械、电气、液压控制等多种领域模型.Sodja 等人[14]提出一种模型简化技术,对基于 Modelica 的信息物理融合系统 CPS(cyber-physical system)多领域模型进行模型简化,以降低基于 Modelica 的异构模型的验证与执行过程的复杂性.

在不同抽象层次的多范式建模方面:Hugues 等人[15]提出一种基于 SysML 和 AADL 的安全关键系统设计、验证及代码生成方法.其中,SysML 主要用于系统高层建模与分析,当该系统分解给软件来实现时,就转换到 AADL,用 AADL 以逐步求精的方式进行软件设计与实现.Wang 和 Hugues 等人[16]则进一步提出面向未来开放式航空电子系统的多范式建模方法,即 SysML 用于系统工程建模,转换到基于 AADL 的软件架构设计,并进一步转换到基于 FACE[17]的开放式航空电子系统架构实现和基于 SCADE 的软件构件功能实现.

在支持多种功能行为表达的多范式建模方面:AADL 提供行为附件 BA(behavior annex)[18]对基于控制流方式的构件功能行为进行表达;正在制定中的 AADL Hybrid Annex[19]则支持基于 Hybrid CSP(communicating sequential processes)对构件的连续行为模型进行构造;Zhan 和 Zhan 等人[20]使用 AADL 与 Simulink 进行混合建模,扩展 AADL 描述系统连续行为的表达能力;欧空局 ESA 提出基于 AADL、Simulink 和规范与描述语言 SDL(specification and description language)[21,22]的多范式建模方法 TASTE[23−25].TASTE 基于 AADL 语言子集描述系统框架,并使用 Simulink、SDL、C/Ada 等描述系统功能行为,目前,TASTE 主要支持串行 Ada 代码自动生成和集成.

随着安全关键异构软件对计算性能要求的不断增加,使用多核处理器成为航空航天领域的迫切需求.我们提出 AADL 及行为附件 BA、同步语言 SIGNAL、SDL、Simulink、C、Ada 的多范式建模方法:AADL 用于表达安全关键异构软件架构,AADL 行为附件 BA、同步语言 SIGNAL、SDL、Simulink 分别支持状态机、同步数据流、异步执行模型、连续时间模型等多种计算模型,并基于抽象语法标记 ASN.1(Abstract Syntax Notation One)[26,27]对异构模型间的交互数据建模.SDL 作为一种异步建模语言,其主要特征是:可以准确描述软件的异步

© 中国科学院软件研究所 http://www.jos.org.cn

功能行为,用于表达安全关键软件异步行为.我们在前期研究[28,29]中提出了同步语言 SIGNAL 和 AADL 的混合建模及多核代码自动生成方法,本文则主要介绍 AADL 和 SDL 的混合建模(两种语言的多范式建模,我们称其为混合建模)及 Ada 多任务代码生成方法.两者的区别在于:同步语言 SIGNAL 主要基于数据流等式表达 AADL 构件的功能行为,侧重于对数值计算相关的算法过程进行建模,例如 GNC 系统中对地斜开关算法等;而 SDL 则侧重基于控制流方式表达构件功能行为,侧重于对系统控制流程进行建模,如 GNC 系统中的航天器三轴姿态角度修正控制模块等.

AADL-SDL 混合建模过程可分为自顶向下和自底向上两个方向:在自顶向下的建模过程中,首先,采用 AADL 描述系统体系结构,并基于 AADL 数据组件描述系统中涉及的数据类型;其次,对于具体功能行为采用 SDL 模型进行描述,并给出 AADL 数据组件到 ASN.1 的转换,以保证异构构件间交互数据的一致性.而在自底向上的建模过程中,首先,采用 SDL 语言描述具体组件功能,并基于 ASN.1 描述 SDL 模型和外部环境(即其他组件或系统输入/输出)之间的交互;其次,基于 AADL 描述系统体系结构,并通过属性集扩展的方式规定 AADL 体系结构模型和 SDL 功能行为模型之间的调用关系和数据交换方式.例如,安全关键系统已经存在多个功能模块的 SDL 模型,如何基于 AADL 将这些不同功能模块自底向上地集成起来,构成完整的系统体系结构模型.

本文考虑自底向上的 AADL-SDL 混合建模过程,主要贡献包括:

(1) 提出一种 AADL 与 SDL 混合建模方法,包括 AADL-ASN.1 和 AADL-SDL 扩展属性集方法.其中,AADL-ASN.1 扩展属性集主要用于描述混合模型中不同构件间的数据类型,AADL-SDL 属性集用于支持在 AADL 体系结构模型中集成 SDL 模型对应的功能行为.

(2) 提出一种面向多核处理器的 AADL-SDL 混合模型到 Ada 多任务代码生成方法.首先,基于 AADL 体系结构模型生成 Ada 框架代码;其次,根据混合模型中的 ASN.1 数据属性生成 Ada 数据类型代码.然后,基于混合模型中的线程运行时属性生成 Ada 运行时代码;最后,基于 SDL 模型生成 Ada 多任务代码,并将所有生成代码进行集成.

(3) 基于 AADL 开源建模工具 Osate[30,31]实现了 AADL-SDL 混合建模工具 ASCM(AADL and SDL co-modeling tool)和多任务 Ada 代码生成工具 AS2MTA(AADL and SDL to multi-task ada code generator),并使用实际工业案例 AOCS 对本文所提方法的有效性进行了分析.

(4) 相较于前期工作,本文扩展了 AADL 多范式建模框架对软件控制流程的描述能力,在支持状态机、同步数据流、连续时间模型等多种计算模型的基础上,增加了异步执行模型的建模过程.生成的 Ada 代码相较于前期工作,增加了流程控制代码的并发过程,提升了代码的运行效率.

本文第 1 节介绍 AADL、SDL 建模语言和 ASN.1 标准的基本概念.第 2 节概述面向安全关键异构软件的 AADL 多范式建模框架.第 3 节对本文提出的 AADL-SDL 混合建模方法进行详细阐述.第 4 节给出 AADL-SDL 混合模型到多任务 Ada 代码生成方法.第 5 节介绍 AADL-SDL 混合建模工具 ASCM 和多任务 Ada 代码生成工具 AS2MTA.第 6 节通过对工业界案例导航、制导与控制系统 GNC 建模与代码生成进行分析并与部分前期研究进行比较,对本文所提方法及原型工具进行评估.第 7 节对多范式建模、混合建模和代码生成相关工作进行分析.第 8 节总结全文,并对未来研究内容进行初步探讨.

1 研究背景

1.1 体系结构分析与设计语言 AADL

AADL 作为一种针对嵌入式系统的多范式建模语言,通过 AADL 构件以及构件之间的连接,从软件结构、软件运行时环境和硬件结构这 3 个方面对嵌入式系统体系结构进行建模描述.

1) 软件结构:支持通过线程、线程组、进程、数据、子程序等构件以及连接对软件的内部结构进行描述.通过上述构件的组合,建立具有层次化的软件体系结构模型.

2) 软件运行时环境:支持通过分发协议、通信协议、调度策略、模式变换协议以及分区机制等属性对软件执行模型进行建模.

© 中国科学院软件研究所 http://www.jos.org.cn

3) 硬件结构:支持通过处理器、虚拟处理器、存储器、外设、总线、虚拟总线等构件以及连接对系统的硬件执行平台进行建模.

此外,AADL 支持基于自定义属性集(property set)和基于附件的扩展.其中,基于自定义属性集的扩展支持 AADL 构件与多种异构模型之间通过自定义属性进行关联,以实现 AADL 模型与多种异构模型间的集成.基于附件的扩展支持在 AADL 核心构件的基础上通过增加附件的方式,提升 AADL 建模语言的描述能力,例如:Behavior Annex 扩展基于控制流方式的构件功能行为表达能力;Hybrid Annex 扩展 AADL 支持对构件的连续行为模型进行构造;Error Model Annex[32]扩展了 AADL 描述系统故障行为的能力.AADL 核心构件、AADL 扩展属性集和 AADL 附件共同组成完整的 AADL 模型.

1.2 规范与描述语言 SDL

规范与描述语言 SDL 支持使用半图形、半文本的方式描述特定类型的嵌入式系统的功能行为.SDL 有图形表示法 GR(graphical representation)和文字短语表示法 PR(phrase representation)两种.其中,GR 用一系列的符号和图形来描述系统,比较直观;PR 用语句来描述系统,便于计算机处理.这两种表示方法在语义上是等效的,它们之间可以互相转换.SDL 建模元素主要分为结构元素、定义元素和行为元素.

结构元素:主要用来描述整个系统模型的分层结构,包括系统(system)、功能块(block)、进程(process)和过程(procedure).图 1 给出了一个 SDL 系统模型结构,其中,SDL 系统层包括两个 SDL 功能块 B1 和 B2,B1、B2 分别通过 SDL 信号 c1、c3 与系统外界交互,B1、B2 之间则通过 SDL 信号 c2 进行交互.SDL 功能块 B1 中进程 P1、P2 分别通过 SDL 信号 R1 和 R2 与外界交互,P1 和 P2 之间通过信号 R3、R4 进行交互.SDL 进程 P1 中描述了当前进程的行为,并且定义了 SDL 过程 Proc1,Proc1 对当前过程的功能行为进行建模.

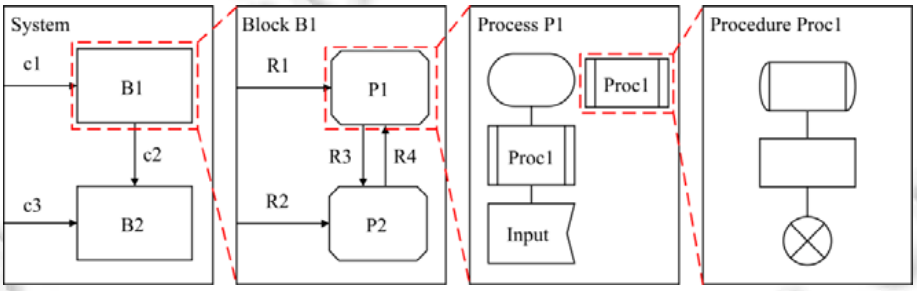


Fig.1 An example of SDL system model structure

图 1 SDL 系统模型结构示例

定义元素:对软件中需要使用到的各种数据(data)、临时变量(variable)和子功能模块之间的信号(signal)进行建模描述.

行为元素:对进程(process)/过程(procedure)模块中的功能行为进行建模.行为元素包括开始状态(start)、状态(state)、触发器(trigger)、行为(action)、表达式(expression)等.其中,开始状态和状态用来基于自动机的方式对系统行为进行建模;触发器包含输入(input)和保存(save)元素,用来对子系统的中断行为进行建模,通常与通信信道组合使用,描述中断信号的传入和相关数据的保存;行为包含输出(output)、任务(task)过程调用(procedure call)、分支选择(switch)和进程创建等,主要用来对具体功能行为中需要的输出操作、顺序流程、外部过程调用、分支选择和进程创建等过程进行建模;表达式(expression)支持以伪代码的形式对具体行为进行建模描述.

1.3 抽象语法标记 ASN.1

ASN.1 作为一种国际标准,用于描述通过电信协议传输的数据.ASN.1 提供了一种对数据进行表示、编码、传输和解码的标准格式,能够有效地对异构系统之间的通信数据进行建模.在网络管理、安全电子邮件、移动网络和空中交通管制等领域有着广泛的运用.国际电信联盟 ITU(International Telecommunication Union)也在标准 ITU-T Z.105[33]中推荐在 SDL 模型中使用 ASN.1 描述数据类型.

© 中国科学院软件研究所 http://www.jos.org.cn

ASN.1 中定义了整型(INTEGER)、布尔(BOOLEAN)、字符串(IA5String,UniveraslString,…)和位串(BIT STRING)等基本数据类型,并且支持有序集合(SEQUENCE)、有序数组(SEQUENCE OF)、无序集合(SET)和无序数组(SET OF)等复杂数据类型的构造.此外,ASN.1 还支持压缩编码规则 PER(packed encoding rule)、可辨别编码规则 DER(distinguished encoding rule)和 XML 编码规则 XER(XML encoding rule)等数据编码、解码规则的描述.因此,ASN.1 也广泛应用于需要计算机通信和其他需要编码数据的行业.

2 安全关键异构软件的 AADL 多范式建模框架

安全关键异构软件的 AADL 多范式建模框架如图 2 所示.核心思路是基于 AADL 描述系统体系结构,基于 AADL BA、同步语言 SIGNAL、SDL、Simulink、C 和 Ada 等描述功能行为,以及基于 AADL 数据构件和 ASN.1 扩展属性集描述异构模型间的交互数据.

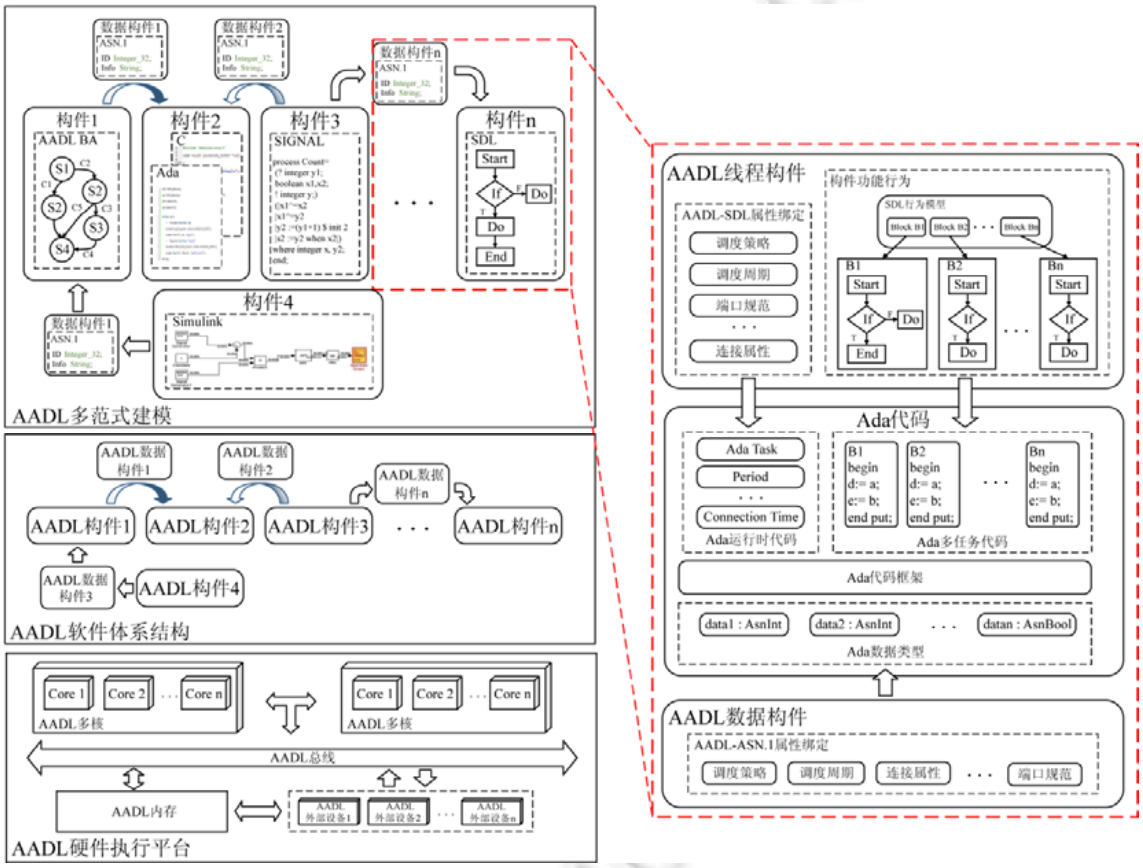


Fig.2 AADL multi-paradigm modeling framework

图 2 AADL 多范式建模框架

本文主要给出 AADL 和 SDL 混合建模及面向多核处理器平台的 Ada 代码自动生成方法,主要涉及图 2 中红色虚线标注的部分.

(1) 提出一种自底向上的 AADL 与 SDL 混合建模方法.首先,为了保证 AADL 模型与 SDL 模型交互数据的一致性,提出 AADL-ASN.1 扩展属性集,以支持在 AADL 模型中使用 ASN.1 进行数据建模.其次,为了支持在 AADL 模型中使用 SDL 描述构件功能行为,提出 AADL-SDL 扩展属性集,主要包括 AADL 模型与 SDL 模型间的接口映射、多线程调度、线程分发策略等相关属性,从而支持 AADL 和 SDL 的混合建模.

© 中国科学院软件研究所 http://www.jos.org.cn

(2) 提出一种面向多核处理器平台的 AADL-SDL 混合模型到 Ada 多任务代码的生成方法.首先,基于 AADL 体系结构模型生成对应的 Ada 框架代码;其次,基于 AADL 数据构件与 AADL-ASN.1 属性集生成 Ada 数据类型代码;然后,基于 AADL 构件与 AADL-SDL 属性集生成 Ada 运行时代码;最后,基于 SDL 行为模型生成 Ada 多任务代码,并将所有生成的代码进行集成.

(3) 基于 AADL 开源建模工具 Osate 与 SDL 开源建模工具 OpenGEODE[34]实现了 AADL-SDL 混合建模工具 ASCM 和 Ada 多任务代码生成工具 AS2MTA,并且使用实际的工业案例 GNC 系统对本文所提方法进行评估.

3 AADL-SDL 混合建模方法

本节主要对 AADL-SDL 混合建模方法的具体内容进行详细介绍.首先,为了支持 SDL 模型与 AADL 模型间的数据交互建模,在第 3.1 节给出 AADL-ASN.1 交互数据属性集扩展.其次,在第 3.2 节给出 AADL-SDL 混合建模属性集扩展以支持将 SDL 模型以自底向上的方式集成到 AADL 架构模型中.

3.1 AADL-ASN.1交互数据属性集扩展

由于 SDL 模型通过 ASN.1 描述数据类型,在进行自底向上的 AADL-SDL 混合建模的过程中,为了保证 SDL 模型与 AADL 模型之间数据交互的一致性,本文提出 AADL-ASN.1 交互数据属性集扩展以支持在 AADL 模型中使用 ASN.1 进行数据建模.该属性集主要包括对基本数据类型和复杂数据类型进行建模的相关属性.其中,基本数据类型包括整型(INTEGER)、实数(REAL)、布尔(BOOLEAN)和字符串(IA5String,UniveraslString,…)等;而复杂数据类型包括有序集合(ASN1_Sequence)、有序数组(ASN1_Sequence_of)、无序集合(ASN1_Set)和无序数组(ASN1_Set_of)等.

AADL-ASN.1 属性集扩展的主要属性见表 1.

Table 1 The ASN.1 data properties

表 1 ASN.1 数据性质

ASN.1 数据性质	描述
Is_Base_Type	判断当前数据构件是否为基本数据类型
Real_Value_Range	浮点类型数据的取值范围
Integer_Value_Range	整型数据的取值范围
String_Size	串类型的长度
Sequence_Of_Range	数组的长度范围
Data_Type	数据的类型
ASN1_Sequence	有序集合
ASN1_Sequence_of	有序数组
ASN1_Set	无序集合
ASN1_Set_of	无序数组
Target_Language	当前构件生成数据文件的格式
ASN1_Config	生成数据文件的相关配置

其中,属性 Is_Base_Type 支持与 AADL 数据构件绑定,属性值的类型为 aadlboolean,用于描述当前 AADL 构件是否为基本数据类型.属性的定义如下:

```
--Property Is_Base_Type:
Is_Base_Type: aadlboolean applies to (data);
```

由于 ASN.1 语法中支持通过数据范围定义数据类型,因此,在 AADL-ASN.1 扩展属性集中增加了实数(浮点)取值范围 Real_Value_Range、整数取值范围 Integer_Value_Range、串长度 String_Size、数组长度范围 Sequence 等属性.具体定义如下:

```
--Property Real_Value_Range
Real_Value_Range: range of aadlreal applies to (data);
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
--Property Integer_Value_Range
Integer_Value_Range: range of aadlinteger applies to (data);

--Property String_Size
String_Size: aadlinteger applies to (data);

--Property Sequence_Of_Range
Sequence_Of_Range: range of aadlinteger applies to (data);
```

ASN.1 支持的数据类型包括基本数据类型和复杂数据类型.具体定义如下:

```
--Property Supported_Data_Type:
Supported_Data_Type: type enumeration
(
ASN1_Boolean,ASN1_Null,ASN1_Integer,
ASN1_Real,ASN1_Enumerated,ASN1_Bit_String,
ASN1_Octet_String,ASN1_Character_String,
ASN1_Sequence,ASN1_Sequence_of,
ASN1_Set,ASN1_Set_of
);

--Property Data_Type:
Data_Type : ASN1_Properties::Supported_Data_Type
applies to (data);
```

属性 Support_Data_Type 的类型为枚举类型,定义了 AADL-ASN.1 属性集支持的所有 ASN.1 数据类型,作为属性 Data_Type 的属性值,与 Data_Type 组合使用描述数据类型.例如,使用 Support_Data_Type 中的元素 ASN1_Boolean 为 Data_Type 赋值,Data_Type=>ASN1_Boolean 定义了当前 AADL 数据构件的类型为 ASN.1 布尔类型.

除了使用 Data_Type 定义数据类型,AADL-ASN.1 属性集还支持对 4 种复杂数据类型进行详细描述.需要使用的属性定义如下:属性 ASN1_Item 是一个 record 类型,定义了复杂数据类型中的基本元素,即复杂数据类型中的成员变量.其中,ID_Value 定义的当前变量在有序集合(ASN1_Sequence)和有序数组(ASN1_Sequence_of)中的序号,如果是无序的结构,ID_Value 默认值为–1.Name_Value 定义当前变量的变量名,Type_Value 定义当前变量的类型,Default_Value 定义当前变量的默认值.属性 ASN1_Sequence、ASN1_Sequence_of、ASN1_Set 和 ASN1_Set_of 分别定义上述 4 种复杂数据类型.其中,ASN1_Sequence 和 ASN1_Set 的属性值的类型为 list,list 中的元素为 ASN1_Item.ASN1_Set_of 和 ASN1_Sequence_of 的属性值为 record,record 的成员 item 类型为 list 定义数组中的元素,成员 Type_Value 定义数组中元素的类型,成员 Count 定义数组中元素的个数,Type_Rename_To 定义引用类型.

```
--Property ASN1_Item:
ASN1_Item: type record (
        ID_Value : aadlinteger;
        Name_Value : aadlstring;
        Type_Value : classifier(data);
        Default_Value : aadlstring;
);

--Property ASN1_Sequence
ASN1_Sequence : list of ASN1_Properties::ASN1_Item
applies to (data);

--Property ASN1_Sequence_of
ASN1_Sequence_of : record (
        Item :list of ASN1_Properties::ASN1_Item;
        Type_Value : ASN1_Properties::Supported_Data_Type;
        Type_Rename_To: classifier(data);
        Count : aadlinteger;
)applies to (data);

--Property ASN1_Set
ASN1_Set : list of ASN1_Properties::ASN1_Item
applies to (data);
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
--Property ASN1_Set_of
ASN1_Set_of : record (
        Item :list of ASN1_Properties::ASN1_Item;
        Type_Value : ASN1_Properties::Supported_Data_Type;
        Type_Rename_To: classifier(data);
        Count : aadlinteger;
)applies to (data);
```

3.2 AADL-SDL混合建模属性集扩展

在第 3.1 节中主要介绍了 AADL-ASN.1 交互数据属性集扩展,本节将给出 AADL-SDL 混合建模属性集扩展的主要内容.在 AADL-SDL 混合模型中 AADL 构件的功能行为使用 SDL 模型表达,SDL 功能行为模型在执行过程中的运行时属性通过 AADL-SDL 混合建模属性集进行建模.属性集中主要包括 SDL 模型运行时的分发策略、分发周期等运行时属性,数据构件的操作类型、访问权限属性,SDL 行为模型的相关属性等.AADL-SDL 混合建模属性集扩展的主要内容见表 2.

Table 2 Properties of the SDL model

表 2 SDL 模型相关属性

SDL 模型属性	描述
Connection_Types	构件端口的数据传输类型
Dispatch_Protocols	线程构件的分发策略
Dispatch_Period	线程构件的分发周期
Period_Offset	线程构件的分发周期偏移
Deadline	线程构件的执行时间限制
Data_Kind	数据构件的操作类型
Access_Permission	数据构件的访问权限
Element_Name	构件对应 SDL 模型名称
Element_Type	构件对应 SDL 模型元素类型
Source_Language	实现构件行为的建模语言名称
Signal_Type	映射到 SDL signal 中参数的类型

属性 Connections_Types 定义 AADL 构件间端口数据传输的类型,支持立即传输(immediate)和延迟传输(delay),属性定义如下:

```
--Property Supported_Connection_Types
Supported_Connection_Types: type enumeration
(Immediate,Delay);
--Property Connection_Types
Connection_Types:SDL_Properties::Supported_Connection_Types applies to (all);
```

属性 Supported_Connection_Types 的类型为枚举类型,定义了两种支持的连接类型 Immediate 和 Delay,作为属性 Connection_Types 的属性值.例如将 Supported_Connection_Types 中的 Immediate 赋值给属性 Connections_Types,Connections_Types=>Immediate 定义立即传输的数据传输连接.

AADL-SDL 支持对周期(periodic)、偶发(sporadic)、非周期(aperiodic)等线程分发策略进行建模,具体使用到的属性如下:

```
--Property Supported_Dispatch_Protocols
Supported_Dispatch_Protocols: type enumeration
        (Periodic,Sporadic,Aperiodic);
--Property Dispatch_Protocols
Dispatch_Protocols: SDL_Properties::Supported_Dispatch_Protocols applies to
(thread);
--Property Deadline
Deadline: inherit Time applies to (thread);
--Property Dispatch_Period
Dispatch_Period: inherit Time applies to (thread);
--Property Period_Offset
Period_Offset: inherit Time applies to (thread);
```

© 中国科学院软件研究所 http://www.jos.org.cn

属性 Supported_Dispatch_Protocols 是枚举类型,作为属性 Dispatch_Protocols 的属性值支持上述 3 种分发策略.例如,Supported_Dispatch_Protocols 中的 Periodic 定义了周期分发策略,Dispatch_Protocols=>Periodic 绑定的 AADL 线程构件采用的就是周期分发策略.

属性 Deadline 继承了 AADL 的时间属性 Time,描述 AADL 线程的最长执行时间.例如,Deadline=>500ms,当前线程构件的最长执行时间为 500ms.

属性 Dispatch_Period 描述不同分发策略下 AADL 线程构件的需要描述的时间片段.例如,AADL 线程构件绑定 Dispatch_Period=>10ms,在周期分发策略下,定义了线程的分发周期为 10ms.在偶发(sporadic)分发策略下,定义了相邻触发事件的最小时间间隔为 10ms.

属性 Period_Offset 定义了采用周期分发策略的线程的启动偏移时间.例如,在未设置当前属性的周期线程中,默认在系统启动时就启动线程.Period_Offset=>10ms 绑定的线程,在系统启动后延迟 10ms 启动线程.

除了线程的分发过程,AADL-SDL 属性集还支持对数据的操作与访问类型.数据的操作类型包含保护数据(protected)与非保护数据(unprotected).数据的访问类型包括只读、只写和读写.属性的具体定义如下所示:

```
--Property Support_Operation_Kinds
Support_Operation_Kinds: type enumeration
        (Protected,Unprotected);
--Property Support_Permissions
Support_Permissions: type enumeration
        (Read_Only,Write_Only,Read_Write);
--Property Data_Kind
Data_Kind: SDL_Properties::Support_Operation_Kinds applies to (data);
--Property Access_Permission
Access_Permission : SDL_Properties::Support_Permissions applies to (data);
```

其中,属性 Support_Operation_Kinds 与 Support_Permissions 是枚举类型,分别定义属性 Data_Kind 与 Access_Permission 的属性值.例如,Data_Kind=>Protected、Access_Permission=>Read_Only 描述了当前 AADL 数据构件定义的数据为保护类型并且只读.

AADL-SDL 属性集定义了对 SDL 模型的接口描述,以确保 AADL 模型能够与 SDL 模型的接口兼容.使用到的主要属性如下所示:

```
--Property Supported_Element_Types
Supported_Element_Types: type enumeration(sdl_signal,sdl_system);
--Property Element_Name
Element_Name: aadlString
      applies to (thread,subprogram,subprogram access,port);
--Property Element_Type
Element_Type:SDL_Properties::Supported_Element_Types
      applies to (classifier,subprogram access,port);
--Property Source_Language
Source_Language: enumeration (SDL)=>SDL
       applies to (port,data access,thread,subprogram);
--Property Signal_Type
Signal_Type: ASN1_Properties::Supported_Data_Type applies to (port);
```

AADL-SDL 扩展属性集支持通过 SDL system 实现 AADL 构件的功能行为,通过 SDL signal 实现数据交互.其中,属性 Supported_Element_Types 定义了 AADL 构件支持的 SDL 元素,属性 Element_Name 描述当前 AADL 构件对应的 SDL 实现的名称,属性 Element_Type 定义了当前构件在 SDL 中实现为 signal 或 system,属性 Source_Language 定义当前构件的具体实现.此外,如果 AADL 中的数据传输过程含有参数,则使用属性 Signal_Type 描述参数的类型.

4 Ada 多任务代码生成

AADL-SDL 混合模型的 Ada 多任务代码生成主要包括两个部分:AADL 体系结构模型到 Ada 多任务代码的生成和 SDL 功能行为模型到 Ada 多任务代码的生成.本节将详细介绍上述代码生成方法的具体内容.

© 中国科学院软件研究所 http://www.jos.org.cn

4.1 AADL模型到Ada多任务代码生成方法

AADL 模型到 Ada 多任务代码的生成方法主要包括 3 个部分.(1) AADL 结构模型到 Ada 框架代码的生成方法;(2) AADL 数据构件到 Ada 数据类型代码的生成方法;(3) AADL 线程构件到 Ada 运行时代码的生成方法.

4.1.1 AADL 结构模型到 Ada 框架代码的生成方法

AADL 结构模型主要由系统构件、进程构件、线程构件和子程序构件等多种构件组成.AADL 结构模型生成的 Ada 框架代码主要包含多个 Ada 过程(procedure)、函数(function)、任务(task)以及相互之间的调用过程.AADL 系统构件与进程构件对应 Ada 代码中的过程,系统构件和进程构件中的特征(feature)、数据子构件(data subcomponent)和连接(connection)分别对应 Ada 过程的形参、局部变量和保护对象.AADL 线程构件和子程序构件分别对应 Ada 代码中的任务和函数.Ada 框架代码中每个 Ada 过程、函数、任务只包含输入输出参数、局部变量声明与调用关系,不包含软件功能行为代码.

4.1.2 AADL 数据构件到 Ada 数据类型代码的生成方法

AADL 数据构件生成 Ada 数据类型代码的过程包括如下步骤:首先,AADL 数据构件通过绑定 AADL-ASN.1 属性集对 AADL-SDL 混合模型中使用的 ASN.1 数据类型进行建模,得到 AADL 数据模型;其次,通过 AADL2ASN1 转换算法生成 ASN.1 数据文件;然后,通过 ASN.1 编译工具 Asn1Scc[35]读取 ASN.1 数据文件,并生成对应的 Ada 数据类型代码.AADL 数据模型到 ASN.1 数据文件的转换算法如下所示:

```
01.  procedure AADL2ASN1
02.  Input: AADL_Data %AADL 数据模型
03.  Output: ASN1_Data %ASN.1 数据类型
04.  data_properties←getProperties(AADL_Data) %获取 AADL 数据模型绑定的所有属性
05.  ASN1_Data←setConfig(ASN1_Data,data_properties) %设置数据名称,文件路径等基本配置信息
06.  ASN1_Data←setType(ASN1_Data,data_properties) %设置数据类型
07.  Is_Base_Type←getIs_Base_Type(data_properties)
08.  If Is_Base_Type=true then %判断是否为基本数据类型
09.     ASN1_Data←setID(ASN1_Data,data_properties) %设置数据编号
10.     ASN1_Data←setRange(ASN1_Data,data_properties) %设置数据范围
11.     ASN1_Data←setValue(ASN1_Data,data_properties) %设置数据值
12.  Else %如果为复杂数据类型
13.     subdatas←getSubDatas(data_properties) %获取当前数据类型的成员
14.     For each subdata in subdatas do %遍历当前数据类型的成员
15.        asnsubdata←AADL2ASN1(subdata) %递归生成 ASN1 数据类型
16.        ASN1_Data←addSubData(ASN1_Data,asnsubdata) %添加 ASN.1 数据类型的成员
17.     end For
18.  end If
19.  return ASN1_Data % 生成 ASN.1 数据类型
20.  end procedure
```

算法输入为 AADL 数据模型 AADL_Data,输出为 ASN.1 数据 ASN1_Data.首先获取 AADL 数据模型中的全部属性 data_properties(第 4 行).通过函数 setConfig 和 setType 分别从属性中解析出 ASN.1 数据的名称、ASN.1 数据文件路径等配置信息和 ASN.1 数据的类型,并赋值给 ASN1_Data(第 5 行~第 6 行);其次,通过函数 getIs_Base_Type 获取属性 Is_Base_Type 的值(第 7 行),判断当前数据是否为基本数据类型(第 8 行).如果是基本数据类型,则通过函数 setID、setRange 和 setValue 分别获取 ASN.1 数据的编号、范围和默认值,并赋值给 ASN1_Data(第 8 行~第 11 行).如果不是基本数据类型,则通过函数 getSubDatas 获取复杂数据类型所有的成员变量,并存入链表 subdatas 中,遍历链表 subdatas,对链表中每个元素递归调用算法 AADL2ASN1,并将返回值添加到 ASN1_Data 的成员变量中(第 12 行~第 18 行).最后,返回 ASN.1 数据 ASN1_Data.

4.1.3 AADL 线程构件到 Ada 运行时代码的生成方法

AADL 线程构件生成的 Ada 代码主要包括两部分:Ada 运行时代码和 Ada 任务执行代码.其中,Ada 任务执行代码通过 SDL 行为模型生成(详见第 5.2 节),因此本节主要介绍 Ada 运行时代码的生成.

根据第 4.4 节中对 AADL-SDL 属性集的介绍,本文提出的 AADL-SDL 混合建模方法中,主要支持 AADL 周期调度、偶发调度和非周期调度这 3 种线程调度策略.

© 中国科学院软件研究所 http://www.jos.org.cn

Ada 运行时代码生成主要采用 Ada 语言提供的 generic 机制,即提供一种参数化模板的方式来定义 Ada 运行时代码模板.本节主要考虑 Ada 运行时代码的生成过程,具体任务功能通过调用 SDL 模型生成的 SDL_procedure 来实现.

AADL 周期线程调度策略生成对应的 Ada 周期任务代码模板 APTCT(Ada periodic task code template)见表 3.AADL 线程通过属性 Dispatch_Protocols=>Periodic 定义当前线程的分发策略为周期性分发,通过属性 Dispatch_Period=>TIME(ms)定义当前线程的周期.TIME 用来表示时间数值.例如,绑定属性 Dispatch_Protocols=>Periodic 与 Dispatch_Period=>20ms 的 AADL 线程对应的 Ada 任务通过时间触发执行,以 20ms 为周期,每隔 20ms 触发 1 次.

周期性任务代码模板的主要参数包括变量与过程.例如,变量 Task_Period 表示任务执行周期.过程 SDL_procedure 为 SDL 行为模型生成的 Ada 任务执行过程代码,通过关键字 in/out 设置 Ada 任务执行过程的传入传出参数.

Table 3 Ada periodic task code template (APTCT)

表 3 Ada 周期任务代码模板

AADL 周期任务调度策略	Ada 周期任务代码模板

thread Periodic_Thread
……
properties
--当前thread构件通过SDL实现
SDL_Properties::Source_Language=>SDL;
--当前thread构件的分发策略为Periodic
SDL_Properties::Dispatch_Protocols=>Periodic;
--分发周期为TIME ms
SDL_Properties::Period=>TIME ms;
……
end Periodic_Thread;
	generic
	//模板参数:执行周期
	Task_Period : in Ada.Real_Time.Time_Span;
	//模板参数:SDL 生成 Ada 代码
	with procedure SDL_Procedure(Param1: in Type1,
	Param2: out Type2,…);
	……
	task body The_Periodic_Task is
	Next_Start:Ada.Real_Time.Time;
	begin
	accept Start;
	//初始化
	Activate_Entrypoint;
	//设置任务启动时间
	Next_Start:=System_Startup_Time+Dispatch_Offset;
	delay until Next_Start;
	//设置任务循环周期与最大执行时间
	Next_Start:=Next_Start+Task_Period;
	Next_Deadline_Val:=System_Startup_Time+
	Dispatch_Offset+Task_Deadline;
	loop
	//调用执行任务行为代码
	SDL_Procedure(Param1:in Type1,
	Param2:out Type2,…);
	delay until Next_Start;
	//设置下一个周期
	Next_Start:=Next_Start+Task_Period;
	Next_Deadline_Val:=Ada.Real_Time.Clock+
	Task_Deadline;
	end loop;
	end The_Periodic_Task;

AADL 偶发线程调度策略对应的 Ada 偶发任务代码模板见表 4.AADL 线程通过属性 Dispatch_Protocols=>Sporadic 定义当前线程的分发策略为偶发,通过属性 Dispatch_Period=>TIME(ms);规定了任务连续分发(dispatch)的最小时间间隔为 TIME(ms).例如,绑定属性 Dispatch_Protocols=>Sporadic 与 Dispatch_Period=>20ms 的 AADL 线程对应的 Ada 任务通过接收外部参数或者队列内参数触发执行,即如果队列不为空,那么读取队列内数据并触发过程;如果队列为空,接收到外部数据,那么读取外部数据并触发任务.并且,由于属性 Dispatch_Period=>20ms 规定了触发最小时间间隔为 20ms,即每次任务执行结束后,需要等待 20ms 才能从队列或者外部读取下一个数据并触发任务.

Ada 偶发任务代码模板 ASTCT(Ada sporadic task code template)与 Ada 周期任务代码模板不同,首先,Ada

© 中国科学院软件研究所 http://www.jos.org.cn

偶发任务的阻塞发生在任务体中,通过调用函数 Wait_For_Incoming_Events 监听端口是否有事件到达;其次,计算最小到达间隔时间 Minimal_Inter_Arrival 来确保相连相邻任务分发之间存在最小时间间隔.

Table 4 Ada sporadic task code template (ASTCT)

表 4 Ada 偶发任务代码模板

AADL 偶发线程调度策略	Ada 偶发任务代码模板

thread Sporadic_Thread
……
properties
--当前thread构件通过SDL实现
SDL_Properties::Source_Language=>SDL;
--当前thread构件的分发策略为Sporadic
SDL_Properties::Dispatch_Protocols=>Sporadic;
--分发周期为TIME ms
SDL_Properties::Period=>TIME ms;
……
end Sporadic_Thread;
	generic
	//模板参数:执行周期
	Task_Period:in Ada.Real_Time.Time_Span;
	//模板参数:SDL 生成 Ada 代码
	with procedure SDL_Procedure(Param1:in Type1,
	Param2:out Type2,…);
	……
	task body The_Sporadic_Task is
	begin
	//初始化
	Activate_Entrypoint;
	//阻塞任务
	Block_Task (Entity);
	delay until System_Startup_Time;
	//任务体
	loop
	//阻塞,等待端口触发事件
	Wait_For_Incoming_Events(Entity,Port);
	Await_Dispatch_Condition();
	//计算最小到达间隔时间
	Next_Start:=Ada.Real_Time.Clock+
	Minimal_Inter_Arrival;
	//根据 SDL 模型生成的 Ada procedure
	SDL_Procedure (Param1:in Type1,
	Param2:out Type2,…);
	//保证最小到达间隔时间
	delay until Next_Start;
	end loop;
	end The_Sporadic_Task;

AADL 非周期线程调度策略对应的 Ada 非周期任务代码模板 AATCT(Ada aperiodic task code template)见表 5.

Table 5 Ada aperiodic task code template (AATCT)

表 5 Ada 非周期任务代码模板

AADL 非周期线程调度策略	Ada 非周期任务代码

thread Aperiodic_Thread
……
properties
-- 当前thread构件通过SDL实现
SDL_Properties::Source_Language=>SDL;
--当前thread构件的分发策略为Aperiodic
SDL_Properties::Dispatch_Protocols=>Aperiodic;
……
end Aperiodic_Thread;
	generic
	//模板参数:SDL 生成 Ada 代码
	with procedure SDL_Procedure(Param1:in Type1,
	Param2:out Type2,…);
	……
	task body The_Aperiodic_Task is
	begin
	//初始化
	Activate_Entrypoint;
	//阻塞任务
	Block_Task (Entity);
	delay until System_Startup_Time;
	//任务体
	loop
	//阻塞,等待端口触发事件
	Wait_For_Incoming_Events(Entity,Port);
	//根据 SDL 模型生成的 Ada procedure
	SDL_Procedure (Param1:in Type1,
	Param2:out Type2,…);
	end loop;
	end The_Aperiodic_Task;

© 中国科学院软件研究所 http://www.jos.org.cn

AADL 线程通过绑定属性 Dispatch_Protocols=>Aperiodic 定义当前线程的分发策略为非周期,非周期调度没有时间属性.例如,绑定属性 Dispatch_Protocols=>Aperiodic 的 AADL 线程对应的 Ada 任务在执行过程中通过接收外部参数或者队列内参数触发执行,通过参数触发执行的过程与偶发调度相同,不同的是偶发调度设置了触发最小时间间隔,而非周期调度没有.非周期任务只要接收外部参数或者队列不为空就会触发执行.

Ada 非周期任务模板和偶发任务模板的主要区别是,前者生成的代码中没有最小到达间隔时间 Minimal_Inter_Arrival 的限制,当 Wait_For_Incoming_Events 监听到事件时,就触发当前线程,执行当前线程的功能任务.

通过对上述 Ada 任务代码模板设置不同的参数,实现 Ada 任务代码的生成,对于多个任务间的数据交互,使用 Ada 队列代码模板 AQCT(Ada queue code template)实现不同任务间的数据交互,将队列创建为 Ada 保护类型(protected),以支持多任务调用,Ada 队列对应的代码模板声明如下所示:

```
package Param_Queue is
    //队列大小
    queue_size:constant:=100;
    //队列缓存区
    type action is access all Signal_Type;
    type index is mod queue_size;
    type todolist is array (index) of action;
    //队列操作接口
    protected type genericqueue is
        entry put (s:in action,t: Time);
        entry get (s:out action,t: Time);
        procedure Initialize;
        function size return Integer;
    end genericqueue;

end Param_Queue;
```

其中,queue_size 定义当前队列容量;Signal_Type 通过参数化的方式定义队列中元素的类型;通过进入点 put,get 实现入队出队操作,put 和 get 参数 s 为队列中的元素,t 为入队出队延迟时间,通过设置参数 t 的值,支持在周期不同的任务之间进行数据交互;Initialize 实现队列的初始化操作;Size 获取当前队列中的元素个数.

4.2 SDL模型到多任务Ada代码生成

欧空局 ESA 开发的 OpenGEODE 开源工具支持从 SDL 模型中生成串行 Ada 代码,在此基础上,我们提出面向多核的 SDL 多任务代码生成方法.如图 3 所示,首先,将 SDL 系统结构(system 和 block)转换为 Ada 多任务框架代码;其次,将 SDL 块结构(block)之间的异步通信(无延迟/延迟)转换为基于 Ada 非延迟/延迟队列的通信机制,从而支持目标多任务通信;最后,基于分别编译技术,利用 OpenGEODE 将 SDL 块中的进程和过程层编译为目标 Ada 代码.

首先,图 3 深色部分给出了 SDL 系统结构到 Ada 多任务框架内代码的生成.如表 6 所示,SDL system 构件转换为 Ada procedure,其中,SDL 模型与环境的交互转换为 procedure 中的参数,以支持被 AADL 模型生成的 Ada 线程所调用.而 SDL 中层次化的 Block 结构,将转换为对应的嵌套 Ada Task 结构.

Table 6 Ada code generated from SDL system structure

表 6 SDL 系统结构生成方法

SDL	Ada
System S; **Channel** c **from** env **to** S **with** in1,in2; **from** S **to** env **with** out; **enchannel**; **endsystem**;	Procedure S(in1,in2: access ASN_SIGNAL; out: access ASN_SIGNAL) is begin… end S;
Block A **Block** A1;…**endblock**; **Block** A2;…**endblock**; **endblock**;	Task body A is Task type A1 is…; Task body A1 is…; Task type A2 is…; Task body A2 is…; ta1:A1; ta2:A2; begin…ta1.Start; ta2.Start;…end A;

© 中国科学院软件研究所 http://www.jos.org.cn

其次,SDL 异步通信转换方法包括非延迟通信和延迟通信(如图 3 红色部分所示).对于 SDL 非延迟通信,本文采用 Ada 队列的方式实现 SDL 不同块间采用异步通信机制,具体实现见表 7:异步队列(asyn_queue)定义为保护类型,以保证并发调用入队/出队(put/get)操作过程的正确性.

对于延迟通信,由于实际的延迟取决于真实物理环境(网络、线路)中的约束,因此仅定义出队和入队操作.为了方便后续案例的分析与实验,本文通过对出队操作进行随机设置延迟时间访问,从而实现仿真延迟通信.具体实现见表 7,包含参数传入和随机值设置两种方式,其中,RandomGenerator 函数用于生成随机延迟时间(默认范围为 0~10s),delay 表达式用于设置延迟出队操作.

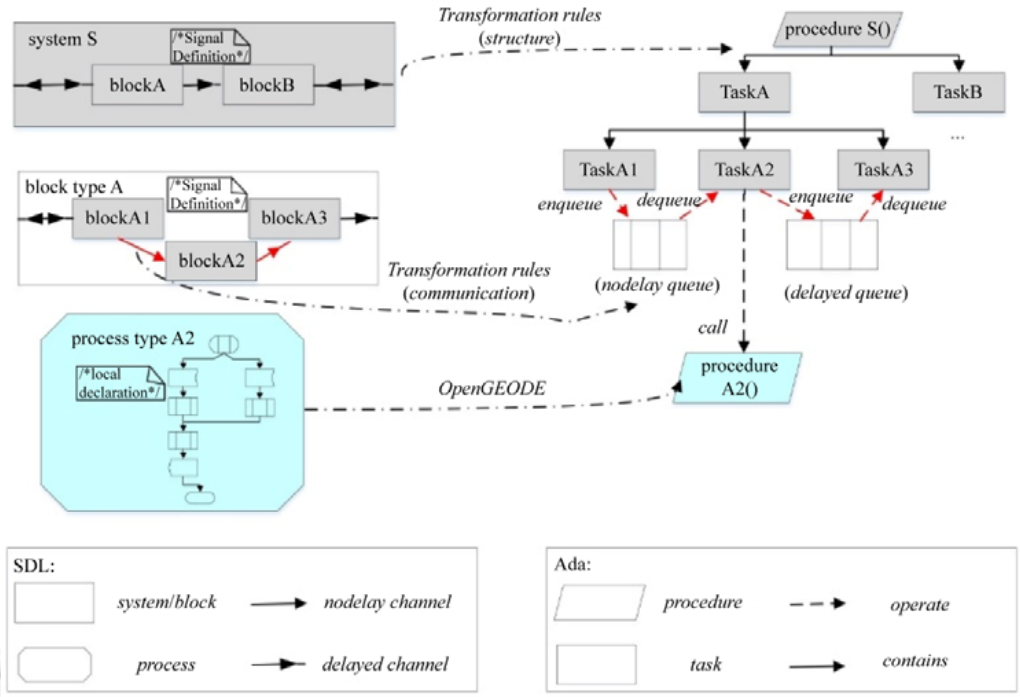


Fig.3 Ada multi-task code generated from SDL

图 3 SDL 到 Ada 多任务代码生成

Table 7 Ada queue code template

表 7 Ada 队列代码模板

Ada 代码声明	Ada 代码实现

queue_size: constant:=…
type action is access all…
type index is mod queue_size;
type todolist is array (index) of action;
protected type asyn_queue is
entry put (s:in action,t: Time);
entry get (s:out action,t: Time);
function size return Integer;
private
todo:todolist:=(others=>**null**);
head:index:=**0**;
tail:index:=**0**;
count:Integer range **0**..queue_size:=**0**;
end asyn_queue;
	entry put (s:in action,t: Time)
	when count<queue_size is
	begin
	todo(tail):=s;
	tail:=tail+1;
	count:=count+1;
	end put;
	entry get (s:out action) when count>**0** is
	r:Float;
	begin
	-- *RandomGenerator*(*r*);
	-- *delay Duration*(*r*);
	s:=todo(head);
	todo(head):=null;
	head:=head+1;
	count:=count-1;
	end get;

© 中国科学院软件研究所 http://www.jos.org.cn

最后,对于 SDL 模型中的功能行为(由 process 和 procedure 组成,如图 3 青色部分所示),本文利用 OpenGEODE 已有串行 Ada 代码生成功能,自动生成对应的 Ada procedure,并根据通信类型将其对应接口和异步通信队列操作相关联,其中,输入对应出队操作,输出对应入队操作.

5 原型工具

原型工具主要包含两部分:AADL+SDL 混合建模工具 ASCM(AADL and SDL co-modeling tool)和多任务 Ada 代码生成工具 AS2MTA(AADL and SDL to multi-task Ada code generator).

5.1 AADL-SDL混合建模工具ASCM

ASCM 支持 AADL+ASN.1 和 AADL+SDL 的混合建模功能,支持 ASN.1 数据文件的生成.ASCM 工具整体框架如图 4 所示.

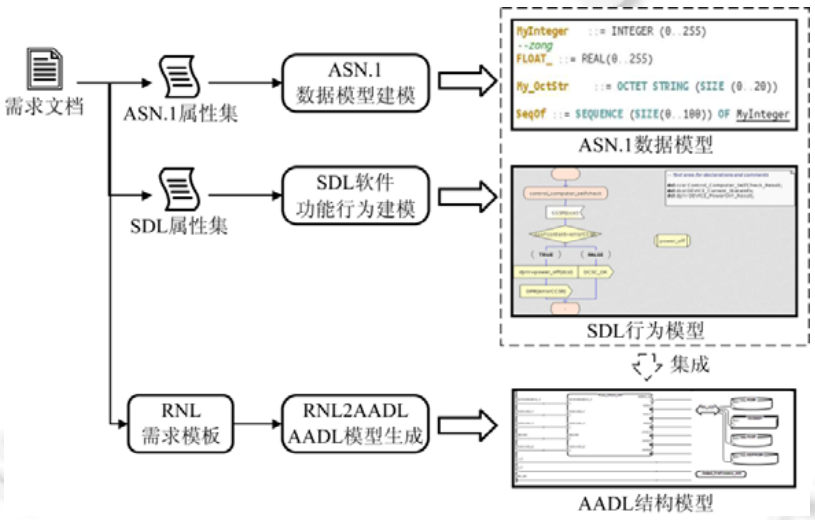


Fig.4 The structure of ASCM

图 4 ASCM 工具结构

ASCM 支持 AADL 软件体系结构建模、ASN.1 数据建模和 SDL 功能行为建模,并通过 ASN.1 属性集和 SDL 属性集实现 AADL 体系结构模型、ASN.1 数据模型和 SDL 功能行为模型这 3 种模型的集成,最终实现 AADL-SDL 的混合建模.其中,对于 AADL 体系结构建模 ASCM 使用 RNL2AADL[36]支持从 RNL(restricted natural language)到 AADL 架构模型的自动生成.

5.2 Ada多任务代码生成工具AS2MTA

AS2MTA 支持 AADL 模型到 Ada 代码框架的生成、SDL 模型到 Ada 行为代码的生成以及 Ada 代码框架与 Ada 行为代码的集成,AS2MTA 工具的整体框架如图 5 所示.

AS2MTA 主要分为 5 个部分.

1) AADL2Ada:基于 Ada 代码生成工具 AADL2Ada,AS2MTA 支持 AADL 软件体系结构模型生成 Ada 代码框架.

2) ASN2Ada:基于开源 ASN.1 编译工具 Asn1Scc,AS2MTA 支持 ASN.1 数据类型生成 Ada 数据定义代码.

3) SDL2Ada:基于开源 SDL 建模工具 OpenGEODE,AS2MTA 支持 SDL 模型到 Ada 任务行为代码的生成.

4) AS2MTA 支持 ASN.1 属性集与 AADL 数据构件生成 Ada 数据访问接口,以支持 Ada 数据定义代码的外部调用.

5) AS2MTA 支持 SDL 属性集与 AADL 构件生成 Ada 运行时代码,以支持 Ada 框架代码对多个 Ada 任务

© 中国科学院软件研究所 http://www.jos.org.cn

行为代码的多任务调度.

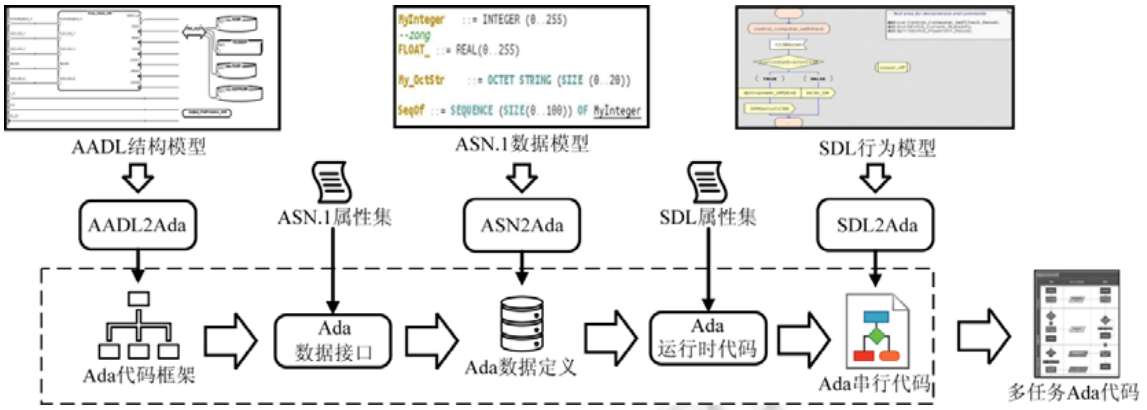


Fig.5 The structure of AS2MTA

图 5 AS2MTA 工具结构

ASCM 与 AS2MTA 通过 Java 实现,工具的具体模块与代码规模见表 8.

Table 8 Statistics of tool implementation

表 8 工具实现数据统计

原型工具	具体功能	代码规模(行)
ASCM	AADL 建模	400+
	ASN.1 建模	2 300+
	SDL 建模	2 200+
AS2MTA	AADL2Ada	900+
	ASN2Ada	800+
	SDL2Ada	1 200+
	Ada 数据接口生成	4 500+
	Ada 运行时代码生成	5 200+

工具的主要特点包括:

(1) 建模方面:ASCM 工具基于 AADL 开源建模工具 Osate 进行功能扩展,集成了 SDL 开源建模工具 OpenGEODE,提供了 AADL-SDL 混合建模平台;扩展了 AADL-ASN.1 属性集和 AADL-SDL 属性集,并集成到 ASCM 工具中,支持 ASN.1 属性集与 AADL 数据构件到 ASN.1 数据文件的自动生成与同步,简化了 ASN.1 数据模型的建模过程,并保证 AADL 模型与 SDL 模型交互数据的一致性.

(2) 代码生成方面:AS2MTA 基于 Ada 代码生成工具 AADL2Ada 进行功能扩展,集成了 ASN.1 开源编译器 Asn1Scc 与 SDL 开源建模工具 OpenGEODE,支持 AADL-SDL 混合模型到多任务 Ada 代码的生成.

(3) 系统应用方面:与实际工业界合作,使用 ASCM 工具对 GNC 系统进行了 AADL-SDL 混合建模,并使用 AADL-SDL 混合模型通过 AS2MTA 工具生成对应的多任务 Ada 代码.

6 案例分析与方法对比

本节将使用 AADL-SDL 混合建模工具 ASCM 和多任务 Ada 代码生成工具 AS2MTA 对实际的工业界案例进行 AADL-SDL 混合建模与多任务 Ada 代码生成.并且,在仿真环境下对生成的多任务 Ada 代码进行运行测试,并与前期研究进行对比分析.最后,结合对比分析结果对本文方法的有效性进行分析评估.

6.1 姿态轨道控制系统AOCS

导航、制导与控制系统 GNC(guidance navigation & control)主要负责航天器姿态和轨道确定与控制.GNC 由导航传感器(例如,导航相机、星敏传感器、陀螺仪和加速度计)、姿态轨道控制系统 AOCS(attitude and orbit

© 中国科学院软件研究所 http://www.jos.org.cn

control system)和执行器(例如,反作用飞轮、喷嘴和发动机)组成.其中,AOCS 主要负责执行轨道确定、轨道控制、姿态确定和姿态控制等任务.此外,通常需要在导航传感器和 AOCS 之间添加一个数据处理单元 DPU(data processing unit),用来对导航传感器发送的数据进行预处理.图 6 所示为 GNC 系统的简化框图.

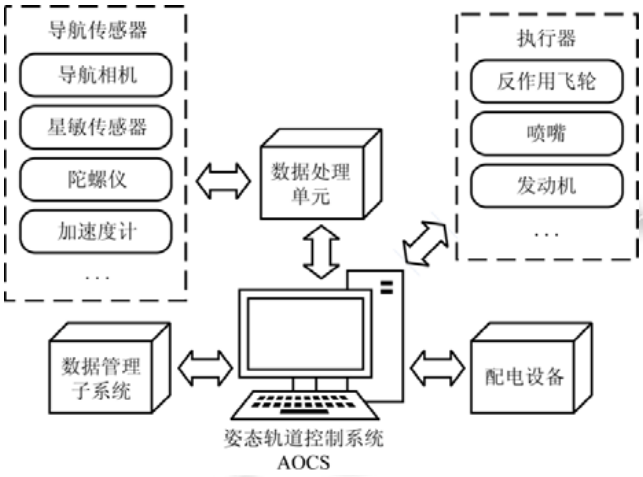


Fig.6 GNC system structure

图 6 GNC 系统结构

为了方便介绍本文所提出的方法,我们主要以 AOCS 为例,详细介绍 AADL-SDL 混合建模方法和 Ada 多任务代码生成方法.AOCS 系统主要包括姿态确定、姿态控制、轨道计算和轨道控制等 9 个模块,总计 124 个子模块和 21 种计算模式.

6.2 AADL-SDL混合建模

AADL-SDL 混合建模过程主要分为 4 个部分.

(1) 基于 SDL 对子系统的功能行为建模.

(2) 基于 AADL-ASN.1 交互数据属性集对系统中的交互数据进行建模.

(3) 基于 AADL 对系统体系结构进行建模.

(4) 基于 AADL-SDL 混合建模属性集将 SDL 功能行为模型与对应的 AADL 构件进行集成.

6.2.1 基于 SDL 的功能行为建模

以 AOCS 系统中轨道计算模块的轨道根数计算(Orbital_Elements_Calculation)为例,详细介绍 SDL 功能行为建模过程.轨道根数计算的输入参数包括轨道倾角、轨道角速度、航天器当前时钟、地面注入时间、轨道递推初始值等,其计算过程包括长期项计算(Long_Term_Calculation)、短周期项计算(Short_Period_Term_Calculation)、平根数计算(Mean_Orbit_Elements_Calculation)和瞬根数计算(Instantaneous_Elements_Calculation)等.对应的 SDL 模型结构如图 7 所示.

首先,使用 ASN.1 标准对轨道根数计算过程的数据进行建模,以轨道倾角 i_0 为例,轨道倾角的数据类型为浮点数,范围为 0~180 的闭区间.因此,定义范围在 0~180 之间的浮点数类型 FLOAT_::=**REAL**(0..180),并使用该浮点数类型定义轨道倾角 i_0:=FLOAT_.其次,使用 SDL 信号(signal)将轨道倾角数据从 SDL 系统传递到 SDL 功能块中.轨道倾角的 SDL 信号的定义为 **signal** i_0(FLOAT_),参数的类型与轨道倾角的类型相同.然后,定义读取轨道倾角并进行相关数值计算的 SDL 功能块 roq,并将 signal i_0(FLOAT_)作为 roq 功能块的输入信号之一.最后,在 roq 功能块内定义 roq 参数计算的 SDL 进程,并在 SDL 进程内部定义具体计算行为.

© 中国科学院软件研究所 http://www.jos.org.cn

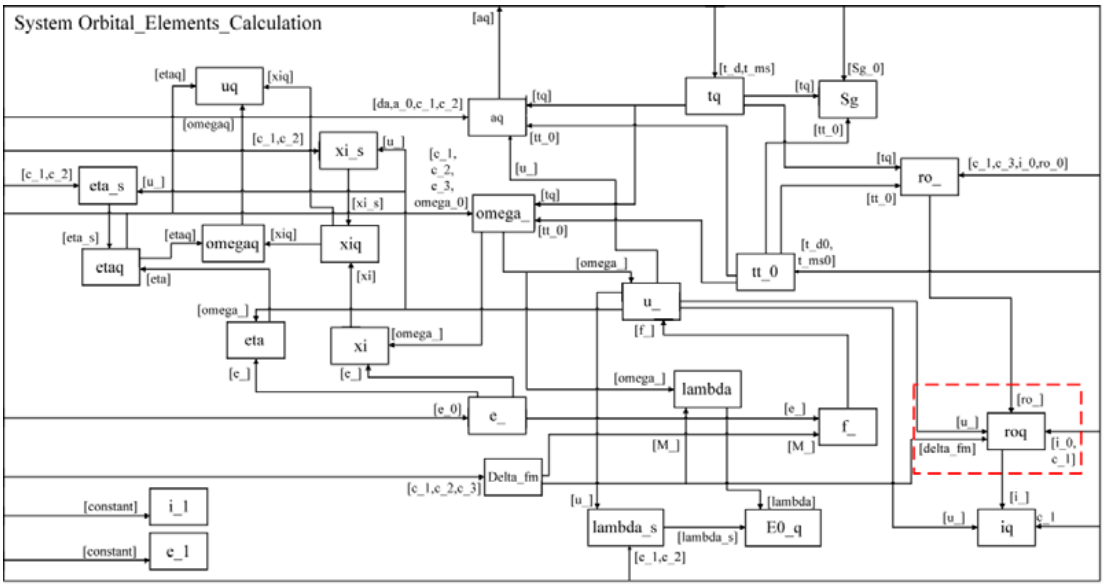


Fig.7 SDL behavior model of Orbital_Elements_Calculation

图 7 轨道根数计算过程 SDL 行为模型

6.6.2 基于 AADL-ASN.1 交互数据属性集的交互数据建模

在 AADL-SDL 混合建模过程中,通过 AADL-ASN.1 交互数据属性集描述异构模型间的交互数据.本节以 AOCS 控制系统中的轨道根数为例,详细介绍 AADL-SDL 混合模型中数据类型的建模过程.轨道根数主要包含轨道倾角(Orbital_Inclination)、升交点黄经(Longitude_Of_The_Ascending_Node)、离心率(eccentricity)、近日点辐角(Argument_Of_Perihelion)、半长轴(Semi_Major_Axis)和平近点角(Mean_Anomaly)这 6 个必要参数.其中,轨道倾角、升交点黄经、近日点幅角和平点近角描述的数据使用 ASN1_Real 进行建模,其角度范围为 0~180 的闭区间.以轨道倾角数据为例,对应的 AADL 数据构件定义如下所示:

```
--轨道倾角数据构件
data Orbital_Inclination
        properties
                ASN1_Properties::Target_Language=>ASN_1;
                --基本数据类型
                ASN1_Properties::Is_Base_Type=>True;
                --当前数据构件类型为ASN1_Real;
                ASN1_Properties::Base_Type=>ASN1_Real;
                --角度范围为0到180
                Asn1_Properties::Real_Value_Range=>0.0..180.0;
end Orbital_Inclination;
```

离心率对应的数据构件的数据类型为浮点数,使用 ASN1_Real 进行建模,且离心率的范围为 0~1.因此,为属性 Asn1_Properties::Real_Value_Range 赋值为 0.0..1.0.半长轴对应飞行器运行轨道的半长轴,数据类型也为浮点数.离心率和半长轴对应的 AADL 数据构件定义如下:

```
--离心率
data Eccentricity
        properties
                ASN1_Properties::Target_Language=>ASN_1;
                --基本数据类型
                ASN1_Properties::Is_Base_Type=>True;
                --当前数据构件类型为ASN1_Real
                ASN1_Properties::Base_Type=>ASN1_Real;
                --离心率为0到1
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
                Asn1_Properties::Real_Value_Range=>0.0..1.0;
end Eccentricity;
--半长轴
data Semi_Major_Axis
        properties
                ASN1_Properties::Target_Language=>ASN_1;
                --基本数据类型
                ASN1_Properties::Is_Base_Type=>True;
                --当前数据构件类型为ASN1_Bit_String;
                ASN1_Properties::Base_Type=>ASN1_Real;
end Semi_Major_Axis;
```

轨道根数数据构件(Orbital_Elements)的类型为包含 6 个元素的无序集合,通过绑定属性 Asn1_Properties::Base_Type=>ASN1_Set 与 Asn1_Properties::ASN1_Set 对 Orbital_Elements 的内部结构进行建模,对应的 AADL 数据构件定义如下:

```
data Orbital_Elements
        properties
                --实现方式为ASN.1
                ASN1_Properties::Target_Language=>ASN_1;
                --当前数据构件对应ASN1_Sequence
                ASN1_Properties::Base_Type=>ASN1_Set;
                --ASN1_Sequence中的具体内容
                ASN1_Properties::ASN1_Set=>
                (
                        --轨道倾角数据构件
                        [Name_Value=>"OI";
                        Type_Value =>
classifier(Orbital_Inclination);],
                        ……
                );
end Orbital_Elements;
```

6.2.3 基于 AADL 的系统体系结构建模

本节主要介绍 AOCS 系统的 AADL 体系结构模型,如图 8 所示.

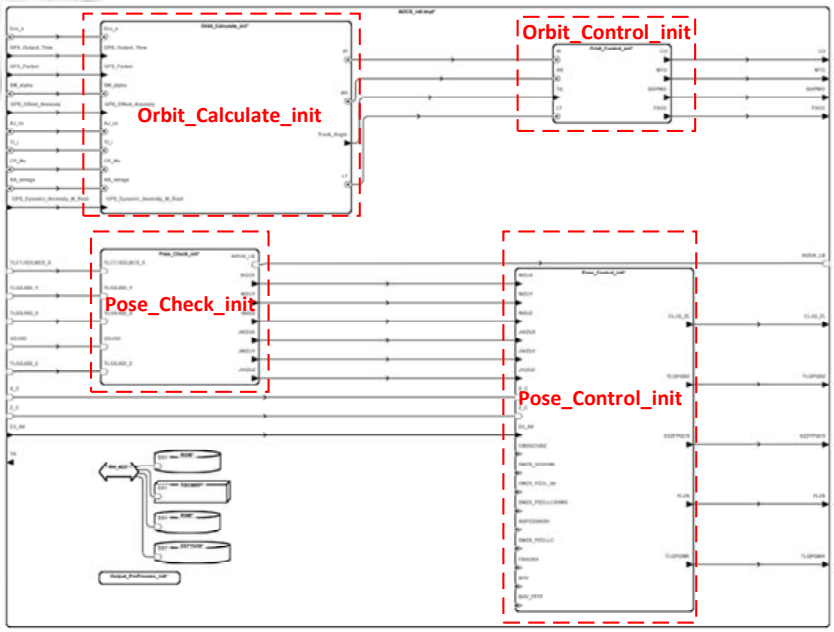


Fig.8 AADL architecture model of AOCS system

图 8 AOCS 系统体系结构模型

© 中国科学院软件研究所 http://www.jos.org.cn

AOCS 系统的体系结构模型包括 AADL 系统构件(AOCS_Init).在 AOCS_Init 构件中定义姿态确定子系统构件(Pose_Check_Init)、姿态控制子系统构件(Pose_Control_Init)、轨道计算子系统构件(Orbit_Calculate_Init)和轨道控制子系统构件(Orbit_Control_Init).Orbit_Calculate_Init 构件中定义轨道根数计算对应的 AADL 线程构件 Orbital_Elements_Calculation;然后,使用 AADL 数据构件与 AADL-ASN.1 属性集扩展对各个子系统间的交互数据进行建模(见第 4.2 节);最后,使用 AADL-SDL 属性集扩展将 Orbital_Elements_Calculation 与轨道根数计算过程的 SDL 模型进行关联(见第 4.3 节).

此外,AOCS 系统的各个子系统构件中的不同功能模块可采用不同计算模型表达功能行为,并通过扩展属性集的方式实现多种异构构件与 AADL 体系结构模型的集成.例如,针对具有较多数据流计算特征的构件使用同步语言 SIGNAL 建模(如,姿态控制子系统中的消除偏模块[28,29]);针对具有较多控制流特征的构件,可以使用 AADL Behavior Annex 的状态机进行描述;对于部分构件,可以重用已有的 C/Ada 代码描述;对于具有异步行为特征的构件,可以使用 SDL 进行建模(如,轨道根数计算模块).

6.2.4 AADL-SDL 模型集成

本节以 AOCS 系统的轨道根数计算过程为例详细阐述基于 AADL-SDL 混合建模属性集的 AADL-SDL 模型集成过程.首先,通过需求文档获取轨道根数的计算过程的相关参数与具体的计算行为细节;其次,以第 4.3 节中的轨道根数数据构件 Orbital_Elements 作为轨道根数计算过程的输入参数,对航天器运行过程中的轨道瞬根数、平根数等参数进行周期性的迭代计算,平均执行周期为 1.5s.

其中,AADL 子程序构件 OE_Msg 用来描述轨道根数计算构件与其他 AADL 线程构件间的数据交互.交互数据的类型为 Orbital_Elements,属性 Source_Language=>SDL 和 Element_Type=>signal 定义当前 subprogram 构件对应 SDL 模型中的 signal.OE_Msg 定义如下:

```
subprogram OE_Msg
        features
                --数据传输过程中需要传输的数据,类型为Orbital_Elements
                orbital_elements: requires data access
Dataview_Uniq::Orbital_Elements;
        properties
                --subprogram OE_Msg通过SDL实现
                SDL_Properties::Source_Language=>SDL;
                --OE_Msg  实现为SDL Signal
                SDL_Properties::Element_Type=>signal;
end OE_Msg;
```

轨道根数计算过程对应的 AADL 构件 Orbital_Elements_Calculation 其主要结构如下所示.首先,子程序构件 OE_Msg 定义了 Orbital_Elements_Calculation 的输入输出端口 in_msg 和 out_msg;其次,属性 Signal_Name 描述了对应的 SDL 行为模型的输入输出端口为 OEC_In 与 OEC_Out;然后,属性 dispatch_Protocols=>Periodic 和 Dispatch_Period=>1500ms 描述了当前线程为周期线程,周期大小为 1 500ms.Orbital_Elements_Calculation 的详细定义如下所示:

```
thread Orbital_Elements_Calculation
        features
                --输入数据
                in_msg: provides subprogram access Channels::OE_Msg{
                      --对应SDL模型中signal
                      SDL_Properties::Element_Type=>signal;
                      --对应SDL signal的名称为OEC_In
                        SDL_Properties::Signal_Name=>"OEC_In";
                };
                --输出数据
                out_msg: requires subprogram access Channels::OE_Msg{
                        --对应SDL模型中signal
                        SDL_Properties::Element_Type=>signal;
                        --对应SDL signal的名称为OEC_Out
                        SDL_Properties::Signal_Name=>"OEC_Out";
                };
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
        properties
                --当前thread构件通过SDL实现
                SDL_Properties::Source_Language=>SDL;
                --当前thread构件的分发策略为Periodic
                SDL_Properties::Dispatch_Protocols=>Periodic;
                --分发周期为1500ms=1.5s
                SDL_Properties::Dispatch_Period=>1500ms;
end Orbital_Elements_Calculation;
```

6.3 Ada多任务代码生成

6.3.1 Ada 框架代码和数据类型代码的生成

首先,基于 Ada 代码生成工具 AADL2Ada 生成对应的 Ada 代码框架.然后,基于 ASN.1 开源编译工具 Asn1Scc 生成 Ada 数据类型代码.AOCS 系统对应的代码整体结构如图 9 所示,其中,system_satellite_attitude_orbit_control_init_impl 为顶层系统,包含功能函数库(system_commonfunc_init)、主控系统(system_ctrl_process_init)、轨道计算(system_obtcalmain_init)、轨道控制(system_obtctrl_init)、姿态确定(system_poscheck_init)和姿态控制(system_posctrl_init)这 6 个子系统和数据类型定义(dataview-uniq).其中,主控系统(system_ctrl_process_init)中还包含 8 个子功能模块.

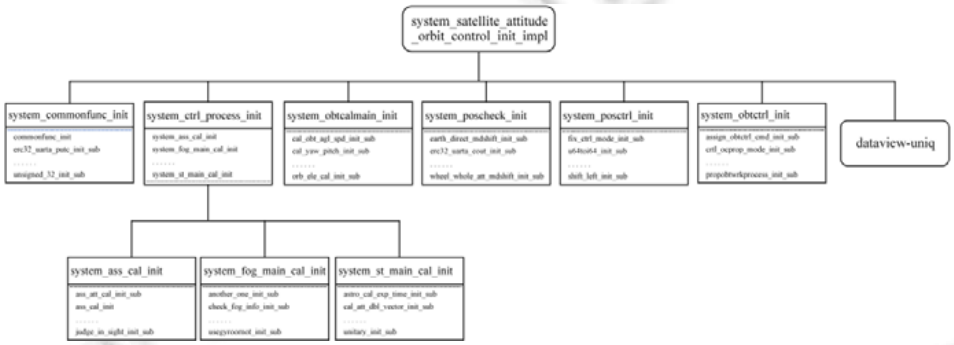


Fig.9 Ada framework code for AOCS

图 9 AOCS 系统 Ada 框架代码

6.3.2 Ada 运行时代码生成

本节结合第 4.1.3 节给出的 Ada 运行时代码的生成过程,以 Orbital_Elements_Calculation 为例进行 AADL-SDL 混合模型到多任务 Ada 代码的生成.轨道根的计算过程属于周期执行过程,因此,AS2MTA 选用第 4.1.3 节中提到的 Ada 代码模板 APTCT 进行代码生成工作,由于轨道根计算过程的执行周期为 1 500ms,因此,APTCT 的参数设置可见表 9.

Table 9 The parameters of Ada runtime code template in Orbital_Elements_Calculation

表 9 轨道根计算过程的 Ada 运行时代码模板参数设置

构件名称	System_Startup_Time	Task_Period	Dispatch_Offset
Orbital_Elements_Calculation	默认系统启动时间	1 500ms	0ms

通过 Ada 代码模板生成对应的运行时代码,通过队列模板生成各个任务间的交互代码.图 10 展示了多个 Ada 任务通过 Ada 队列代码进行任务间的交互过程,AADL 模型中包括轨道根数计算线程在内一共 n 个线程构件.以 Orbital_Elements_Calculation 线程构件与 T2 线程构件为例,Orbital_Elements_Calculation 构件通过 Ada 周期任务代码模板 APTCT 生成对应的 Ada 周期任务代码 Orbital_Elements_Calculation_a.T2 构件生成对应 Ada 周期任务代码 T2_a.Orbital_Elements_Calculation 构件与 T2 构件之间的交互数据 OE_Msg 通过 Ada 队列代码模板 AQCT 生成 Orbital_Elements_Calculation_a 与 T2_a 进行交互的队列代码 OEC_2_T2_Queue.

© 中国科学院软件研究所 http://www.jos.org.cn

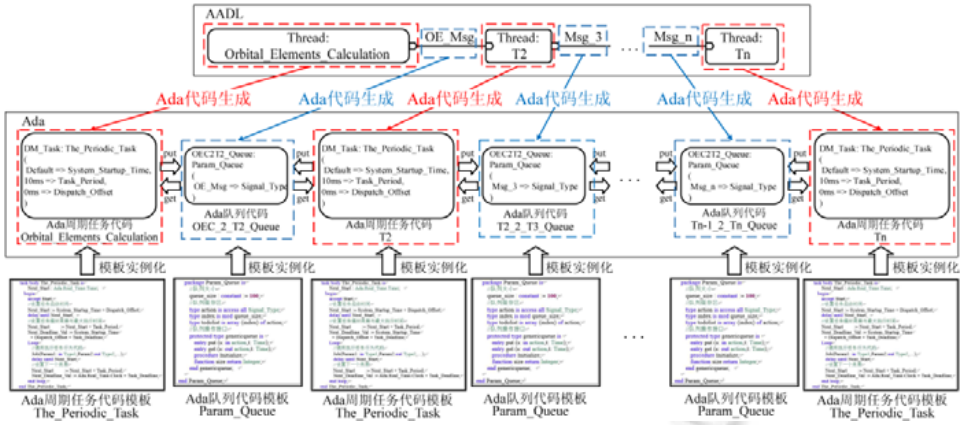


Fig.10 Ada runtime code

图 10 Ada 运行时代码

6.3.3 SDL 模型到多任务代码生成

本节以第 6.2.1 节轨道根数计算 SDL 行为模型为例,详细介绍 SDL 行为模型到 Ada 多任务代码的生成过程.轨道根数计算 SDL 行为模型分别对应于编译后的部分目标 Ada 代码,见表 10.

Table 10 Multi-tasking Ada code from SDL

表 10 SDL 多任务 Ada 代码

Ada 任务执行	Ada 任务数据交互
Procedure Orbital_Elements_Calculation (in_msg:in ASN_SIGNAL, out_msg:out ASN_SIGNAL) is task type U_ is entry Start; end U_; t_u_:U_; task body U_ is begin PI_U_(…); U_(); RI_AQ(…); … end U_; begin t_U_.Start; … t_Aq.Start; end Orbital_Elements_Calculation;	dp_queue: delayedqueue; ccs_queue: delayedqueue; … procedure PI_U_(DPC_param: access Device_PowerOn_Command) is begin … dp_queue.get(…); … end p_dp; procedure RI_AQ(CCSR_param: access Control_Computer_SelfCheck_Result) is begin ccs_queue.put(…); … end p_dp;

其中,Ada 过程 Orbital_Elements_Calculation 根据 SDL 模型中 system 组件生成,其参数列表根据 SDL 模型与环境交互方向分别使用关键字 in 和 out 进行标注.Orbital_Elements_Calculation 过程中包括 25 个 Ada task,分别根据源 SDL 模型中的 block 组件生成.对于不同块之间的异步通信,根据输入信号对应的块名声明对应的延迟队列.对于块中的具体功能,基于 OpenGEODE 生成对应的 Ada 目标代码.以 AQ 块与 U_块为例,U_过程为 U_task 的计算部分,由 OpenGEODE 生成.此外,U_task 中还包括两类特殊的 Ada 过程:RI_U_和 PI_AQ,其中,PI_U_负责从 DP 块对应的延迟队列中取数据,而 RI_AQ 负责将计算结果放入 AQ 块对应的延迟队列中.

6.4 AADL-SDL混合模型分析与评估

首先,使用 SDL 对 AOCS 系统的软件功能模块内部行为进行建模.SDL 模型的统计数据见表 11.

© 中国科学院软件研究所 http://www.jos.org.cn

Table 11 SDL model statistics of AOCS system

表 11 AOCS 系统 SDL 模型统计

AOCS 系统	姿态确定	姿态控制	轨道计算	轨道控制	总计
SDL 模型(行)	4 600+	3 300+	2 800+	5 100+	15 800+
Block 数	57	44	32	61	194
Signal 数	100+	50+	50+	400+	600+
Procedure 数	200+	300+	100+	200+	800+

其次,使用限定自然语言生成 AADL 工具 RNL2AADL[36],通过自然语言需求模板对 AOCS 系统自然需求文本中的软件架构信息进行抽取,并通过抽取的软件架构信息生成 AADL 系统体系结构模型,体系结构模型主要包括 4 个主系统和 309 个软硬件构件,其中主要包括 1 个顶层系统构件、4 个一级子系统构件、125 个二级子系统构件、72 个 3 级子系统构件、259 个子程序构件、632 个线程构件.除了描述系统结构的构件外,还包括 148 个数据构件,主要对整个系统中各个构件之间进行通信交互的数据报文建模;1 146 个数据访问连接;902 个外部过程访问连接.AADL 软件体系结构模型的统计数据见表 12.

Table 12 The statistical data AADL model of AOCS system

表 12 AOCS 系统 AADL 模型统计

AOCS 系统	姿态确定	姿态控制	轨道计算	轨道控制	总计
AADL 模型(行)	3 500+	2 600+	3 800+	2 700+	12 600+
子系统数	40	27	25	21	113
子程序数	100+	50+	50+	100+	300+
线程数	200+	100+	100+	100+	500+
数据访问数	400+	200+	300+	500+	1 400+
数据端口数	200+	100+	100+	300+	700+

最后,通过 AADL-SDL 扩展属性集实现上述 SDL 功能行为与 AADL 体系结构模型的集成.

在第 2 节中,我们提出了 AADL 及行为附件 BA、同步语言 Signal、SDL、Simulink、C、Ada 的多范式建模总体框架,本文主要研究 AADL 和 SDL 混合建模方法.通过和工业界的合作与确认,这里选用适用范围、生成代码特征等指标对 AADL-SDL、AADL-BA[37]、AADL-Signal[28,29]这 3 种混合建模方法进行评价,分析结果见表 13.

Table 13 The comparison and evaluation of AADL-BA, AADL-Signal, AADL-SDL

表 13 AADL-BA、AADL-Signal、AADL-SDL 混合模型对比评估结果

建模方法	适用范围	生成代码特征
AADL-BA	需要进行频繁状态切换的反应式系统	以系统状态变化为核心.生成代码包含状态机结构
AADL-Signal	数值计算算法与同步并发算法调度	以可执行任务为单元,对每个任务内的算法进行单独封装.同时生成代码支持多任务之间的同步.代码主要内容为算法与并发任务的同步
AADL-SDL	流程控制,业务解耦,复杂数据结构操作,异步系统	以对应子系统功能模块的业务流程为核心,包括主要功能的执行流程、数据输入、数据输出、异常处理、异步队列等在编程过程中常见的代码结构.并且,对不同功能对应的代码进行了独立封装,实现了代码流程上的解耦

如表 13 所示,AADL-BA、AADL-Signal 与 AADL-SDL 混合建模方法的对比分析结果如下.

(1) 适用范围:在通过混合模型进行代码生成的过程中,AADL-BA 通过自动机模型生成代码,代码生成过程采用的编程策略以系统的状态变化为核心,生成代码适用于需要进行频繁状态切换的反应式系统.AADL-Signal 通过数据流等式描述数值的计算过程,具有同步并发的描述语义,适用于对系统中数值计算的算法与同步调度算法进行建模与代码生成.如航天嵌入式系统中的轨道计算、姿态计算等相关算法.AADL-SDL 侧重于描述系统实现过程的流程控制与异步行为,可以清晰地展示出系统不同功能模块间的流程划分,实现代码的解耦,提升各部分代码的复用性.

(2) 生成代码特征:针对 AADL-BA 混合建模,由于 BA 将系统行为抽象为状态与状态迁移.生成代码中引入

© 中国科学院软件研究所 http://www.jos.org.cn

了状态机结构,适合对软件行为中的状态变换行为进行建模.针对 AADL-Signal 混合建模,Signal 采用数据流等式的方式进行建模,模型抽象程度较高,适用于对软件行为中数值计算相关的算法内容进行建模.针对 AADL-SDL 混合建模,SDL 支持对系统行为中的业务流程与异步控制进行建模.可以准确、清晰地描述出软件行为中的流程控制行为.

对比分析结果,正是体现了安全关键异构软件系统需要进行多范式建模的必要性.AADL-BA、AADL-Signal 及 AADL-SDL 混合建模方法能够较完整地支持安全关键异构软件系统的建模要求.

6.5 生成代码分析与评估

6.5.1 与 AOCS 已有代码对比分析

首先,针对 AOCS 系统 AADL-SDL 混合模型,基于 AS2MTA 进行多任务的 Ada 代码生成,并且,选取生成代码规模和代码运行时间两个指标来对本文所提出的方法进行评估.针对这两个指标,我们分别选取 AOCS 系统源代码和 AS2MTA 生成代码在多核仿真环境下进行实验结果统计,其中,AOCS 系统源代码的相关实验由实际工业界进行,并为我们提供处理后实验结果数据.实验具体结果见表 14.

通过表 14 中的数据可知,AS2MTA 生成的代码规模比 AOCS 系统源码要大,主要原因包括:(1) 生成多任务运行时;(2) 本文的代码生成方法还未包含编码策略的优化等过程.

Table 14 The test results of generated code (I)

表 14 生成代码运行测试结果(I)

AOCS 系统		姿态确定	姿态控制	轨道计算	轨道控制	总计
系统源码	代码规模(行)	6 100+	7300+	4 000+	12 600+	30 000+
	单核仿真 平均执行时间(s)	31.510 9	34.424 7	22.348 7	19.223 0	
	多核仿真 平均执行时间(s)	20.321 1	19.544 7	12.664 3	9.009 8	
AS2MTA 生成代码	代码规模(行)	4 200+	3 100+	4 100+	23 000+	517 00+
	多核仿真 平均执行时间(s)	21.283 2	16.720 0	18.048 9	8.002 1	

AS2MTA 生成的代码的平均执行时间与 AOCS 系统源码的平均执行时间的数据统计情况如图 10 所示.

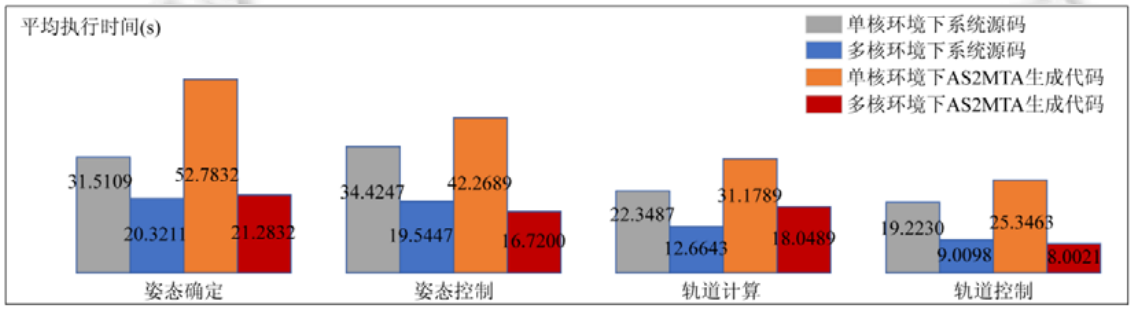


Fig.11 Average code execution time

图 11 代码平均执行时间统计

通过分析表 14 和图 10,我们得出 AS2MTA 生成的代码的平均执行时间与 AOCS 系统源码的平均执行时间对比分析结果.相较于单核仿真环境下 AS2MTA 的生成代码,多核仿真环境下的生成代码平均执行时间有所提升.但是,与 AOCS 系统源码相比,在运行代码体量较大的情况下,AS2MTA 生成多任务代码的运行时间较短.但在运行代码体量较小的情况下,AOCS 系统源码的执行时间较短.造成这种现象的主要原因可能是,虽然并发运行多任务代码会提升运行速率,但是创建多任务的过程会占用部分时间,在运行代码体量较小的情况下,创建多任务的时间占用了大部分的运行时间,描述功能逻辑的代码实际运行时间只占用了运行时间的一小部分,代码的并行执行对代码运行效率产生的提升效果无法弥补创建任务对代码运行效率产生的降低效果,导致多任

© 中国科学院软件研究所 http://www.jos.org.cn

务代码的运行效率低于 AOCS 系统源代码的运行效率.在运行代码体量较大的情况下,描述功能逻辑的代码实际运行时间远大于任务创建时间,则创建任务对代码运行效率产生的影响就可以忽略不计,并行代码的运行效率就会高于源代码的运行效率.未来,我们将对 AS2MTA 的生成代码进行优化.

6.5.2 与 AADL-BA、AADL-Signal 生成代码对比分析

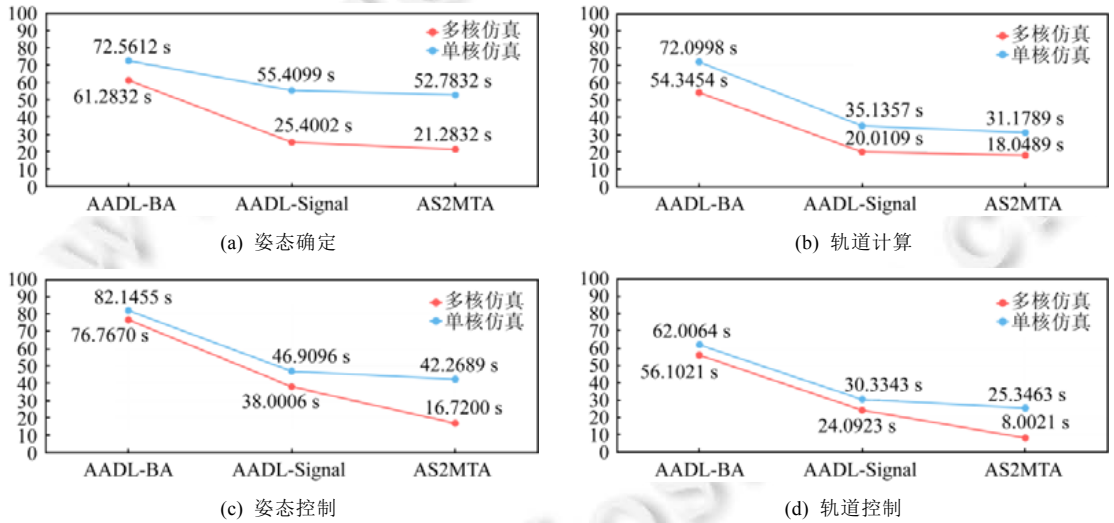
本节针对代码规模和代码运行时间两个指标,分别选取前期研究中 AADL-BA 混合模型生成代码与 AADL-Signal 混合模型生成代码和 AS2MTA 生成代码在仿真环境下进行实验.实验具体结果见表 15.

Table 15 The test results of generated code (II)

表 15 生成代码运行测试结果(II)

AOCS 系统		姿态确定	姿态控制	轨道计算	轨道控制	总计
AADL-BA 生成代码	代码规模(行)	3 700+	2 100+	3 300+	24 000+	34 000+
	单核仿真 平均执行时间(s)	72.561 2	82.145 5	72.099 8	62.006 4	
	多核仿真 平均执行时间(s)	61.283 2	76.767 0	54.345 4	56.102 1	
AADL-Signal 生成代码	代码规模(行)	2 200+	1 900+	4 700+	24 000+	33 200+
	单核仿真 平均执行时间(s)	55.609 9	47.909 6	35.135 7	30.334 3	
	多核仿真 平均执行时间(s)	25.400 2	38.000 6	20.010 9	24.092 3	
AS2MTA 生成代码	代码规模(行)	4 200+	2 100+	3 100+	22 000+	31 700+
	单核仿真 平均执行时间(s)	52.783 2	42.268 9	31.178 9	25.346 3	
	多核仿真 平均执行时间(s)	21.283 2	16.720 0	18.048 9	8.002 1	

AADL-BA 生成代码、AADL-Signal 生成代码的平均执行时间与 AS2MTA 生成代码的平均执行时间的数据统计情况如图 11 所示.



(a) 姿态确定 (b) 轨道计算

(c) 姿态控制 (d) 轨道控制

Fig.12 Average code execution time

图 12 代码平均执行时间统计

通过分析表 15 和图 11,AS2MTA 生成的代码的平均执行时间与 AADL-BA 和 AADL-Signal 生成代码的平均执行时间对比分析结果如下.

1) 单核仿真环境下的平均执行时间:AS2MTA 生成代码相较于 AADL-BA 与 AADL-Signal 的生成代码的平均执行时间,是最短的.在单核仿真环境下运行,多任务并发对代码执行效率的提升影响较低,影响代码运行效率的主要因素为代码结构.AADL-BA 为所有生成代码引入状态机结构,对于功能行为中的状态变换,通过状态机代码可以提升运行效率,对于不适用状态机的功能行为,则由于状态机代码的结构复杂而降低了代码的执行效率,因此,AADL-BA 的平均执行时间最长.AADL-Signal 侧重于考虑数值计算行为中的并发,运行测试也只

© 中国科学院软件研究所 http://www.jos.org.cn

针对于数据计算的代码实现过程,因此,相较于 AADL-BA 代码的平均执行时间有所降低.AADL-SDL 侧重于多个控制流程间的并发,在单核仿真环境下,AADL-Signal 与 AADL-SDL 生成代码的平均执行时间没有明显差距,AADL-SDL 相较于 AADL-BA,由于代码生成结构的优化,缩短了代码的平均执行时间.

2) 多核仿真环境下的平均执行时间:相较于单核仿真环境,多核仿真环境下 3 种建模方法的生成代码运行效率都有所提升.与前期研究相比,AS2MTA 生成代码的平均执行时间最短,运行效率最高.并且,通过分析图 10 可知,AADL-Signal 侧重于数据计算过程的并发行为,姿态确定和轨道计算两个模块的主要内容是航天器各种姿态参数与轨道参数的计算过程.因此,在姿态确定和轨道计算两个模块中,AADL-Signal 生成代码的运行效率较高,AS2MTA 生成代码的运行效率没有明显提升.但是,姿态控制与轨道控制模块的主要职责是根据参数对航天器的姿态与轨道进行调整,代码结构以流程控制为主,因此,相较于 AADL-Signal,使用 AS2MTA 生成代码的执行效率有明显提升,代码平均执行时间大幅度缩短.

7 相关工作

本文的相关工作主要包括 3 个方面:多范式建模、混合建模和基于模型的代码生成.

7.1 多范式建模相关工作

Vangheluwe 在文献[4]中介绍了多范式建模与仿真,并将多范式建模的研究方向分为 3 个部分:(1) 多形式化建模,涉及到描述的不同形式化模型之间的耦合和转换.(2) 模型抽象,涉及到不同抽象级别上模型之间的关系.(3) 元建模,描述模型的模型.

Mosterman 在文献[5]中将 Vangheluwe 描述的多范式建模方法的 3 个研究方向抽象为多范式建模的 3 个维度:多抽象、多形式化和元模型.并且,基于这 3 个维度引入了领域独立的多范式建模框架,通过一个混合动力系统的案例,描述了多范式建模方法的统一框架.

Morozov 将多范式建模方法应用在信息物理系统 CPS(cyber-physical systems)中[38],使用 UML 定义了 CPS 系统的元模型,并使用汽车喷漆的案例概述了元模型的使用过程.

Blouin 提出了一种对嵌入式系统架构模型进行定量分析的多范式建模语言 QAML(quantitative analysis modeling language)[39],QAML 通过利用 AADL、SysML、MARTE、AUTOSAR 等多领域建模语言对嵌入式系统架构模型进行定量分析.

Blouin 和 Borde 在文献[40]中提出 AADL 作为一种多范式建模语言,由于具有可以扩充其他形式语义的特性,因此可以作为系统多范式建模方法的核心.同时,介绍了 ADL 在 MPM4CPS 中的应用,并使用一个机器人案例说明 AADL 多范式建模的具体用法,描述了自顶向下以 AADL 架构模型为中心组合使用 SysML、MARTE 和 UML 等模型,从需求分析开始,然后进行系统设计、设计分析和验证,最后自动生成代码的设计流程.

SysML[7]作为一种针对系统工程领域的图形化建模语言,侧重于复杂系统的整体架构.适合在系统开发生命周期的初期对系统进行需求捕获、边界定义、逻辑架构设计等,由于缺乏形式语义,在系统详细设计与实现阶段的应用较少.AADL 基于构件对软件体系结构进行建模,提供进程、线程、子程序、处理器、外部设备等多种构件,适合在系统开发生命周期后面的阶段进行详细的设计和具体实现.本文提出的混合建模方法主要面向多任务 Ada 代码生成工作,需要对系统详细设计以及实现细节进行建模,因此,本文使用 AADL 对系统体系结构进行建模.

7.2 混合建模以及代码生成相关工作

多领域的模型集成作为一种多范式建模方法,不同领域建模语言之间的混合建模过程是多领域的模型集成的重要组成部分.AADL 主要通过扩展附件和混合建模实现多领域的模型集成.

AADL 目前支持 Behavior Annex[18]通过集成时间自动机模型对软件行为中的状态控制进行描述;BLESS Annex[41]则在 Behavior Annex 的基础上为自动机添加 BLESS Assertions,使其支持自动化的推理证明.Error Model Annex[32]通过集成故障属性、故障概率等故障模型对软件行为中的失效事件进行描述;Hybird Annex[19]

© 中国科学院软件研究所 http://www.jos.org.cn

通过集成 Hybrid CSP 对系统物理层的连续行为进行描述.

Zhan 提出了 AADL 与 Simulink/Stateflow 混合建模方法[20],相较于 Hybrid Annex 对系统物理层连续行为的描述,Zhan 使用 Simulink/Stateflow 描述 AADL 系统体系结构模型中物理层的连续行为,通过 contract 对 AADL 物理层构件进行约束.

Ouni 使用 AADL 与 Capella 进行混合建模[42],通过 INGEQUIP 保持交互信息在不同模型间的一致,通过 Capella 描述系统设计初期的功能需求、非功能需求以及平台约束,并将 Capella 模型集成到 AADL 模型中,组合两种模型进行调度分析、行为验证、结构验证等工作.

欧空局 ESA 提出 AADL、Simulink、SDL 的多范式建模方法 TASTE[23−25],TASTE 通过 AADL 部分子集描述系统体系结构,并通过 TASTE 扩展属性集将 AADL 系统体系结构模型与 Simulink、SDL 等行为模型进行集成.通过集成代码生成工具 Ocarina[43]进行基于 AADL 模型的代码生成.通过集成 Simulink Coder、Qgen、OpenGEODE 等工具实现 Simulink、SDL 等行为模型代码的生成.TASTE 基于 OpenGEODE 实现 SDL 模型到 Ada 代码的生成,目前主要支持串行 Ada 代码自动生成和集成.本文结合 TASTE 的工作,使用分别编译技术实现了 SDL 行为模型到多任务 Ada 代码的自动生成与调度,并且支持与 AADL 模型生成 Ada 代码进行集成.

Baouyaa 使用 TASTE 工具集对火车控制系统进行建模[44],使用 AADL 描述系统体系结构,使用 SDL 描述系统行为,并对铁路系统的相关性质进行了验证.

在面向安全关键软件的 AADL 多范式建模的前期研究中[28,29],我们基于同步语言 Signal 对 AADL 语义进行了扩展,使用 Signal 对 AADL 模型中的并发行为进行了描述,并支持 AADL-Signal 混合模型到多任务代码的生成.本文基于 SDL 进一步扩展了 AADL 语义,对软件系统中的异步行为进行了描述,并且实现了 AADL-SDL 到多任务 Ada 代码的生成.

8 总结与未来工作

本文提出一种 AADL 和 SDL 混合建模方法,支持以自底向上的方式对安全关键软件系统进行混合建模,并给出面向多核处理器平台的代码自动生成方法.首先,提出 AADL 与 SDL 混合建模方法,包括 AADL-ASN.1 和 AADL-SDL 扩展属性集方法.其中,AADL-ASN.1 扩展属性集主要用于描述混合模型中不同构件间的数据类型,AADL-SDL 属性集用于支持在 AADL 体系结构模型中集成 SDL 模型对应的功能行为.其次,提出面向多核处理器的 AADL-SDL 混合模型到 Ada 多任务代码生成方法,包括框架代码、数据类型代码、运行时代码以及多任务功能代码的自动生成.然后,基于 AADL 开源建模工具 Osate 实现了 AADL-SDL 混合建模工具 ASCM 和多任务 Ada 代码生成工具 AS2MTA,并且使用实际工业案例对本文所提方法的有效性进行了分析.

在未来的工作中,我们将进一步开展以下 3 个方面的研究工作.

1) 工具扩展方面:目前的代码生成工作依赖于开源 SDL 建模工具 OpenGEODE,因此,AADL-SDL 混合模型中采用的 SDL 标准与 OpenGEODE 工具保持一致,SDL 标准目前已经更新到 SDL 2010,未来考虑开发支持 SDL 2010 标准的 SDL 建模工具,并实现 ASCM 和 AS2MTA 到 SDL 2010 的移植.

2) 组合验证方面:研究 AADL-SDL 混合模型的形式化组合验证方法.由于混合模型中存在异步通信,因此需考虑与异步通信有关的形式化验证.

3) 生成代码的智能优化方面:面对复杂编程场景有时需要依靠编程人员的经验灵活选择编程策略,未来将研究基于人工智能的编程策略自动优化.

References:

[1] Leveson NG. Engineering a Safer World: Systems Thinking Applied to Safety. The MIT Press, 2016.

[2] Paz A, El Boussaidi G, Hafedh M. ChecSDM: A method for ensuring consistency in heterogeneous safety-critical system design. IEEE Trans. on Software Engineering, 2020. [doi: 10.1109/TSE.2020.2966994]

[3] Benveniste A, Caillaud B, Carloni LP, Sangiovanni-Vincentelli AL. Composing heterogeneous reactive systems. ACM Trans. on Embedded Computing Systems (TECS), 2008,7(4):1–36. [doi: 10.1145/1376804.1376811]

© 中国科学院软件研究所 http://www.jos.org.cn

[4] Vangheluwe H, De Lara J, Mosterman PJ. An introduction to multi-paradigm modelling and simulation. In: Proc. of the AIS' 2002 Conf. (AI, Simulation and Planning in High Autonomy Systems). Lisboa, 2002. 9–20.

[5] Mosterman PJ, Vangheluwe H. Computer automated multi-paradigm modeling: An introduction. Simulation, 2004,80(9):433–450. [doi: 10.1177/0037549704050532]

[6] Fritzson P, Bunus P. Modelica—A general object-oriented language for continuous and discrete-event system modeling and simulation. In: Proc. of the 35th Annual Simulation Symp. IEEE, 2002. 365–380. [doi: 10.1109/SIMSYM.2002.1000174]

[7] Specification OMG. OMG System Modeling Language (OMG SysML) Specification. Version 1.6.

[8] Gérard S, Selic B. The UML—marte standardized profile. IFAC Proc. Volumes, 2008,41(2):6909–6913. [doi: 10.3182/20080706-5-KR-1001.01171]

[9] 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]

[10] Blom H, Lönn H, Hagl F, Papadopoulos Y, Reiser MO, Sjöstedt CJ, Chen DJ, Kolagari RT. EAST-ADL: An architecture description language for automotive software-intensive systems. In: Embedded Computing Systems: Applications, Optimization, and Advanced Design. IGI Global, 2013. 456–470. [doi: 10.4018/978-1-4666-3922-5.ch023]

[11] Le Sergent T. SCADE: A comprehensive framework for critical system and software engineering. In: Proc. of the Int'l SDL Forum. Berlin, Heidelberg: Springer-Verlag, 2011. 2–3. [doi: 10.1007/978-3-642-25264-8_2]

[12] Ursu C, Bhat R, Damodaran R. Simulink® modeling for vehicle simulator design. SAE Technical Paper, 2011. [doi: 10.4271/2011-01-0746]

[13] Ptolemaeus C. System Design, Modeling, and Simulation: Using Ptolemy II. Berkeley: Ptolemy.org, 2014.

[14] Sodja A, Škrjanc I, Zupančič B. Cyber-physical modelling in Modelica with model-reduction techniques. Journal of Systems and Software, 2020,163:110517. [doi: 10.1016/j.jss.2019.110517]

[15] de Saqui-Sannes P, Hugues J. Combining SysML and AADL for the design, validation and implementation of critical systems. ERTS, 2012.

[16] Zhe W, Hugues J, Chaudemar JC, LeSergent T. An integrated approach to model based engineering with SysML, AADL and FACE. SAE Technical Paper, 2018. [doi: 10.4271/2018-01-1942]

[17] Cebrowski AK. FACE™ Technical Standard. Edition 3.0, 2017.

[18] Dissaux P, Bodeveix JP, Filali M, Gaufillet P, Vernadat F. AADL behavioral annex. In: Proc. of the DASIA Conf. Berlin, 2006. 32.

[19] Ahmad E, Larson BR, Barrett SC, Zhan NJ, Dong Y. Hybrid annex: An AADL extension for continuous behavior and cyber-physical interaction modeling. In: Proc. of the 2014 ACM SIGAda Annual Conf. on High Integrity Language Technology. 2014. 29–38. [doi: 10.1145/2692956.2663178]

[20] Zhan H, Lin Q, Wang S, Talpin JP, Xu X, Zhan N. Unified graphical co-modelling of cyber-physical systems using AADL and simulink/stateflow. In: Proc. of the Int'l Symp. on Unifying Theories of Programming. Cham: Springer-Verlag, 2019. 109–129. [doi: 10.1007/978-3-030-31038-7_6]

[21] Sandhu KK. Specification and description language (SDL). In: Proc. of the IEE Tutorial Colloquium on Formal Methods and Notations Applicable to Telecommunications. IET, 1992. 3/1–3/4.

[22] Belina F, Hogrefe D. The CCITT-specification and description language SDL. Computer Networks and ISDN Systems, 1989,16(4):311–341. [doi: 10.1016/0169-7552(89)90078-0]

[23] Perrotin M, Grochowski K, Verhoef M, Galano D, Mosdorf M, Kurowski M, Denis F, Graas E. TASTE in action. 2016. https://www.researchgate.net/publication/316857258_TASTE_in_action

[24] Perrotin M, Conquet E, Delange J, Schiele A, Tsiodras T. TASTE: A real-time software engineering tool-chain overview, status, and future. In: Proc. of the Int'l SDL Forum. Berlin, Heidelberg: Springer-Verlag, 2011. 26–37. [doi: 10.1007/978-3-642-25264-8_4]

[25] Perrotin M, Conquet E, Dissaux P, Tsiodras T, Hugues J. The TASTE Toolset: Turning human designed heterogeneous systems into computer built homogeneous software. 2010. https://www.researchgate.net/publication/44708161_The_TASTE_Toolset_turning_human_designed_heterogeneous_systems_into_computer_built_homogeneous_software

© 中国科学院软件研究所 http://www.jos.org.cn

[26] Barry PT. Abstract syntax notation-one (ASN. 1). In: Proc. of the IEE Tutorial Colloquium on Formal Methods and Notations Applicable to Telecommunications. IET, 1992. 2/1–2/3.

[27] ITU-T RX. Specification of Abstract Syntax Notation One (ASN.1), 1988.

[28] Yuan S, Yang Z, Bodeveix JP, Filali M, Wang T, Zhou Y. Automated Ada code generation from synchronous dataflow programs on multicore: Approach and industrial study. In: Proc. of the Int'l Workshop on Formal Techniques for Safety-critical Systems. Cham: Springer-Verlag, 2019. 57–73. [doi: 10.1007/978-3-030-46902-3_4]

[29] Yang ZB, Yuan SH, Xie J, Zhou Y, Chen Z, Xue L, Bodeveix JP, Filali M. Multi-threaded code generation tool for synchronous language. Ruan Jian Xue Bao/Journal of Software, 2019,30(7):1980–2002 (in Chinese with English abstract). http.//www.jos.org.cn/1000-9825/5754.htm [doi: 10.13328/j.cnki.jos.005754]

[30] Feiler P, Delange J. Automated fault tree analysis from AADL models. ACM SIGAda Ada Letters, 2017,36(2):39–46. [doi: 10.1145/3092893.3092900]

[31] Lasnier G, Pautet L, Hugues J, Wrage L. An implementation of the behavior annex in the AADL-toolset OSATE2. In: Proc. of the 16th IEEE Int'l Conf. on Engineering of Complex Computer Systems. IEEE, 2011. 332–337. [doi: 10.1109/ICECCS.2011.39]

[32] AS5506 SAE. 1A: Architecture analysis and design language (AADL) Annex Volume1: Annex E: Error model Annex. 2015. https://www.sae.org/standards/content/as5506/1a

[33] Verhaard L. An introduction to Z. Computer Networks and ISDN Systems, 1996,28(12):1617–1628. [doi: 10.1016/0169-7552(95)00121-2]

[34] Verhoef M, Perrotin M. TASTE for overture to keep SLIM. In: Proc. of the 13th Overture Workshop. 2015. 132–139.

[35] Mamais G, Tsiodras T, Lesens D, Perrotin M. An ASN.1 compiler for embedded/space systems. 2012. https://www.researchgate.net/publication/229422184_An_ASN1_compiler_for_embeddedspace_systems

[36] 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 with English abstract). http://www.jos.org.cn/1000-9825/5530.htm [doi: 10.13328/j.cnki.jos.005530]

[37] Feng SZ, Yang ZB, Xue L. Automatic generation method of Ada code for aerospace embedded software based on AADL. Computer and Modernization, 2020(6):52–59 (in Chinese with English abstract).

[38] Morozov D, Lezoche M, Panetto H. Multi-paradigm modelling of cyber-physical systems. IFAC-PapersOnLine, 2018,51(11):1385–1390. [doi: 10.1016/j.ifacol.2018.08.334]

[39] Senn E, Blouin D, Zendra O. A multi-paradigm DSML for quantitative analysis of embedded system architecture models. In: Proc. of the 15th ACM/IEEE Int'l Conf. on Model Driven Engineering Languages & Systems-MODELS. 2012. [doi: 10.1145/2508443.2508450]

[40] Carreira P, Amaral V, Vangheluwe H. Multi-paradigm modelling for cyber-physical systems: Foundations. In: Foundations of Multi-paradigm Modelling for Cyber-physical Systems. Cham: Springer-Verlag, 2020. 1–14. [doi: 10.1007/978-3-030-43946-0_1]

[41] Larson BR, Chalin P, Hatcliff J. BLESS: Formal specification and verification of behaviors for embedded systems with software. In: Proc. of the NASA Formal Methods Symp. Berlin, Heidelberg: Springer-Verlag, 2013. 276–290. [doi: 10.1007/978-3-642-38088-4_19]

[42] Ouni B, Gaufillet P, Jenn E, Hugues J. Model driven engineering with Capella and AADL. 2016. https://www.researchgate.net/publication/308507803_Model_Driven_Engineering_with_Capella_and_AADL

[43] Lasnier G, Zalila B, Pautet L, Hugues J. Ocarina: An environment for AADL models analysis and automatic code generation for high integrity applications. In: Proc. of the Int'l Conf. on Reliable Software Technologies. Berlin, Heidelberg: Springer-Verlag, 2009. 237–250. [doi: 10.1007/978-3-642-01924-1_17]

[44] Baouya A, Mohamed OA, Bennouar D, Ouchani S. Safety analysis of train control system based on model-driven design methodology. Computers in Industry, 2019,105:1–16. [doi: 10.1016/j.compind.2018.10.007]

附中文参考文献:

[9] 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]

© 中国科学院软件研究所 http://www.jos.org.cn

[29] 杨志斌,袁胜浩,谢健,周勇,陈哲,薛磊,Jean-Paul BODEVEIX,Mamoun FILALI.一种同步语言多线程代码自动生成工具.软件学报,2019,30(7):1980−2002. http.//www.jos.org.cn/1000-9825/5754.htm [doi: 10.13328/ j.cnki.jos.005754]

[36] 王飞,杨志斌,黄志球,周勇,刘承威,章文炳,薛垒,许金淼.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 [doi: 10.13328/j.cnki.jos.005530]

[37] 冯思喆,杨志斌,薛垒.基于 AADL 的航天嵌入式软件 Ada 代码自动生成方法.计算机与现代化,2020(6):52−59.

宗喆(1996－),男,硕士,CCF 学生会员,主要研究领域为安全关键嵌入式软件.

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.

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.

Bodeveix Jean-Paul(1963－),男,博士,教授,博士生导师,主要研究领域为实时系统,形式化方法.

袁胜浩(1994－),男,硕士,主要研究领域为形式化方法,定理证明.

Filali Mamoun(1957－),男,博士,高级研究员,博士生导师,主要研究领域为实时系统,形式化方法.

© 中国科学院软件研究所 http://www.jos.org.cn