计算机科学
COMPUTER SCIENCE

面向ARINC653操作系统的综合化航空电子软件代码自动生成方法

凌仕翔, 杨志斌, 周勇

引用本文

凌仕翔, 杨志斌, 周勇. 面向ARINC653操作系统的综合化航空电子软件代码自动生成方法[J]. 计算机科学, 2024, 51(7): 10-21.

LING Shixiang, YANG Zhibin, ZHOU Yong. Integrated Avionics Software Code Automatic Generation Method for ARINC653 Operating System [J]. Computer Science, 2024, 51(7): 10-21.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230600216

面向 ARINC653 操作系统的综合化航空电子软件代码自动生成方法

凌仕翔　杨志斌　周　勇
南京航空航天大学计算机科学与技术学院　南京 211106
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 211106
(1596103924@qq.com)

摘　要　综合化航空电子系统(Integrated Modular Avionics,IMA)是一类典型的安全关键系统,具有分布式、异构、计算资源和物理资源强耦合等特征。随着 IMA 系统趋于复杂化和智能化,系统的功能越来越多地采用软件来实现,如何对这类复杂软件进行建模并自动生成代码成为一个重要挑战。文中提出了一种基于 AADL(Architecture Analysis and Design Language)的综合化航空电子系统代码生成方法。首先,提出 HMC4ARINC653(Heterogeneous Model Container for ARINC653)属性集扩展,使其具备描述 IMA 软件架构、异构功能行为和非功能属性的能力;其次,提出 IMA 模型到 C 代码及 ARINC653 系统配置文件的映射规则,并遵守 MISRA C 安全编码规范,生成的代码能够在 ARINC653 操作系统上部署并仿真执行;最后,设计并实现了相应的原型工具,以 ARINC653 操作系统和工业界实际案例,验证了所提方法和工具的有效性。

关键词:综合化航空电子系统;ARINC653 操作系统;AADL;代码自动生成
中图分类号　TP311

Integrated Avionics Software Code Automatic Generation Method for ARINC653 Operating System

LING Shixiang,YANG Zhibin and ZHOU Yong
School of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 211106,China
Key Laboratory of Safety-critical Software,Ministry of Industry and Information Technology,Nanjing 211106,China

Abstract　Integrated modular avionics(IMA) is a typical safety-critical system characterized by its distributed,heterogeneous nature and strong coupling of computing and physical resources. With the increasing complexity and intelligence of IMA systems, software is increasingly being used to implement system functionalities. Modeling and generating code for such complex software pose significant challenges. This paper presents a code generation approach for IMA systems based on the architecture analysis and design language(AADL). Firstly,an extension of the HMC4ARINC653(heterogeneous model container for ARINC653) attribute set is proposed to enable the description of IMA software architecture,heterogeneous functional behavior,and non-functional attributes. Secondly,mapping rules from the IMA model to C code and ARINC653 system configuration files are defined, adhering to the MISRA C safety coding guidelines. The generated code can be deployed and simulated on the ARINC653 operating system. Finally,the corresponding prototype tool is designed and implemented to validate the effectiveness of the methodology and tools proposed in this paper with the ARINC653 operating system and real cases from the industry.

Keywords　Integrated avionics system,ARINC653 operating system,AADL,Automatic code generation

1　引言

航空电子系统先后经历了分立式、联合式、综合化(Integrated Modular Avionics,IMA)[1]等阶段。分区(Partition)是综合化航空电子系统的核心概念,即每个分区包含独立的地址空间、上下文数据以及实时任务,一个分区的错误行为不会影响到其他分区。为此,综合化航空电子操作系统标准 ARINC653[2]定义了分区管理、调度、通信等接口,以保证 IMA 系统的可靠性和安全性。IMA 系统作为一类典型的安全关键系统[3],对安全性、可靠性、实时性有极高的要求。

到稿日期:2023-06-29　返修日期:2023-11-29
基金项目:国家自然科学基金(62072233);国防基础科研项目(JCKY2020205C006);航空科学基金(201919052002);南京航空航天大学科研与实践创新计划(xcxjh20221602)
This work was supported by the National Natural Science Foundation of China(62072233),National Defense Basic Scientific Research Project (JCKY2020205C006),Aeronautical Science Foundation of China(201919052002) and Postgraduate Research & Practice Innovation Program of NUAA(xcxjh20221602).
通信作者:杨志斌(yangzhibin168@163.com)

随着系统复杂性急剧增加，IMA 系统的功能越来越多地采用软件实现。如何在有限的开发时间和成本下设计与实现高质量的综合化航空电子软件成为国内外学术界与工业界面临的重要挑战。近年来，模型驱动（Model-Driven）尤其是采用形式化模型驱动的综合化航空电子软件设计与开发方法逐渐受到重视，并被工业界认为是切实可行的重要手段[4]。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[5] 就将模型驱动和形式化方法（即 DO-331[6] 和 DO-333[7]）作为其核心标准的重要技术补充。

模型驱动的安全关键系统设计与开发方法中常用的建模语言有 SCADE[8]，Simulink[9]，SysML[10]，AADL[11]（Architecture Analysis & Design Language，AADL）等。SCADE 和 Simulink 主要用于功能模块设计和验证[12]。AADL 作为一种能对嵌入式系统的软件以及硬件体系结构进行建模与分析的模型驱动语言，在 IMA 系统的模型驱动开发方法方面的应用得到了学术界与工业界的认可[13]。

随着 IMA 系统趋向于复杂化和智能化，综合化航空电子软件常采用异构构件（Heterogeneous Components）组合架构，例如构件由不同供应商以 OEM 方式提供[14]，各个构件可能使用不同的计算模型、不同的实现语言甚至不同的人工智能模型，使得整个软件呈现异构性。如何对这类复杂异构软件建模并自动生成代码成为一项重要挑战。

在建模方面，中科院软件所詹乃军研究员和法国 INRIA Jean-Pierre Talpin 教授[15]提出了面向 CPS 系统的 AADL 和 Simulink/Stateflow 混合建模方法。本文的前期工作研究了 AADL 与同步语言的混合建模方法[16]。欧空局（ESA）提出了基于 AADL、Simulink 和规范与描述语言 SDL 的异构建模方法 TASTE[17]，基于 AADL 语言子集描述系统框架，并使用 C/Ada，Simulink，SDL 以及 SCADE 等描述系统功能行为。但这些方法都未考虑 IMA 系统的建模。另外，AADL 标准定义了 ARINC653 扩展附件，给出了基于 AADL 核心语言表达 ARINC653 架构的建模方式，并扩展了部分针对 ARINC653 的特殊属性，但不支持异构软件建模。

在代码生成方面，综合化航空电子软件设计往往采用 SCADE，Simulink 等建模工具，这些工具主要生成平台无关功能代码[18]。法国 Lasnier 等[19]研究基于 ARINC653 分区应用的从 AADL 模型到 C 代码的映射，设计并实现了原型工具 OCARINA，其中平台相关部分信息主要通过代码生成器硬编码实现，模型和源代码之间存在较大的语义差异。Rahmoun 等提出了面向具体执行平台的 AADL 模型求精工具 RAMSES[20]，主要考虑 POSIX，OSEK 和 ARINC653 等平台上的模型求精及其代码生成，但不能完全支持 ARINC653 标准。上述工作一方面对平台相关信息支持不完善；另一方面，生成的代码仍需要工程师手工添加相关功能代码，这一过程不仅繁琐而且容易出错，既费时费力又难以保证代码质量。

为此，本文提出了一种面向 ARINC653 操作系统的综合化航空电子软件代码自动生成方法。相比 Lasnier 和 Rahmoun 等的工作，本文方法支持 IMA 异构软件建模以及国产 ARINC653 操作系统平台。本文方法总体框架如图 1 所示。首先，解析 IMA 模型生成抽象语法树，为每个叶子节点添加具体属性生成实例语法树；其次，根据转换规则和 MISRA C 安全编码规范遍历语法树生成 C 代码及配置文件；最后，将代码和配置文件部署到 ARINC653 操作系统上进行仿真执行。

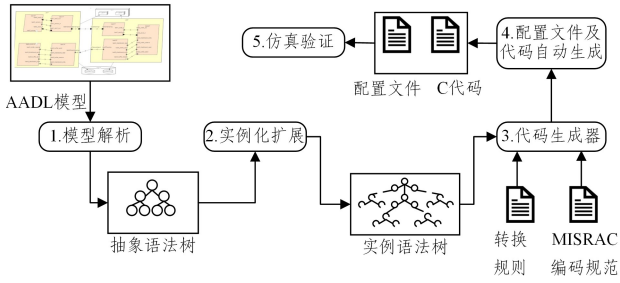

图 1 总体框架

Fig. 1 Overall framework

本文的主要贡献包括：

1）提出了扩展属性集 HMC4ARINC653，使其具备描述综合化航空电子系统的软件架构、异构功能行为和非功能属性的能力，并给出了 IMA 软件建模方法。

2）设计代码转换规则并考虑安全编码规范，针对每个分区生成相应的框架代码、分区运行时代码、数据结构代码、功能代码以及系统配置文件，生成的源代码能够通过安全编码检查并支持直接部署到 ARINC653 操作系统上进行仿真执行。

3）设计并实现了相应的原型工具，并基于 ARINC653 操作系统和工业界实际案例验证了本文方法和工具的有效性。

2 研究背景

2.1 IMA 体系架构

ARINC653 标准（Avionics Application Software Standard Interface）是由国际标准组织 SAE 发布的航空电子应用软件标准接口规范，其目的是在航电实时操作系统与应用软件之间定义一套通用的应用执行接口标准。IMA 体系架构包括应用层、APEX 接口、系统内核和硬件层。应用层主要包括应用分区和系统分区。APEX 接口位于应用软件和操作系统之间，它定义了系统为应用软件提供的一组设施，以控制有关其内部处理元素的调度、通信和状态信息。系统内核提供分区隔离环境以及 APEX 接口的具体实现。

ARINC653 标准提供了 IMA 软件分区操作环境[21]，根据软件的安全级别进行分区，每个分区在时间和空间上相互隔离，分区中的故障不会影响同一处理器上其他分区的运行。

2.2 ARINC653 操作系统

ARINC653 操作系统架构如图 2 所示，系统通过配置文件降低模块之间的依赖性。

ARINC653 操作系统运行时，首先根据配置文件信息为分区应用分配资源，如内存要求、时间要求、外部接口等，并指定分区间的通信；然后对分区内的进程进行资源配置（如周期、执行时间、WCET），检查进程配置信息与框架代码中的参数是否一致，将分区部署在指定的核心模块，确保系统在初始化期间每个分区应用的完整性；最后通过模块支持层的链接器和相关的链接脚本将配置文件和代码加载到系统中，生成工程目标文件，能在目标板或 QEMU 上仿真运行。

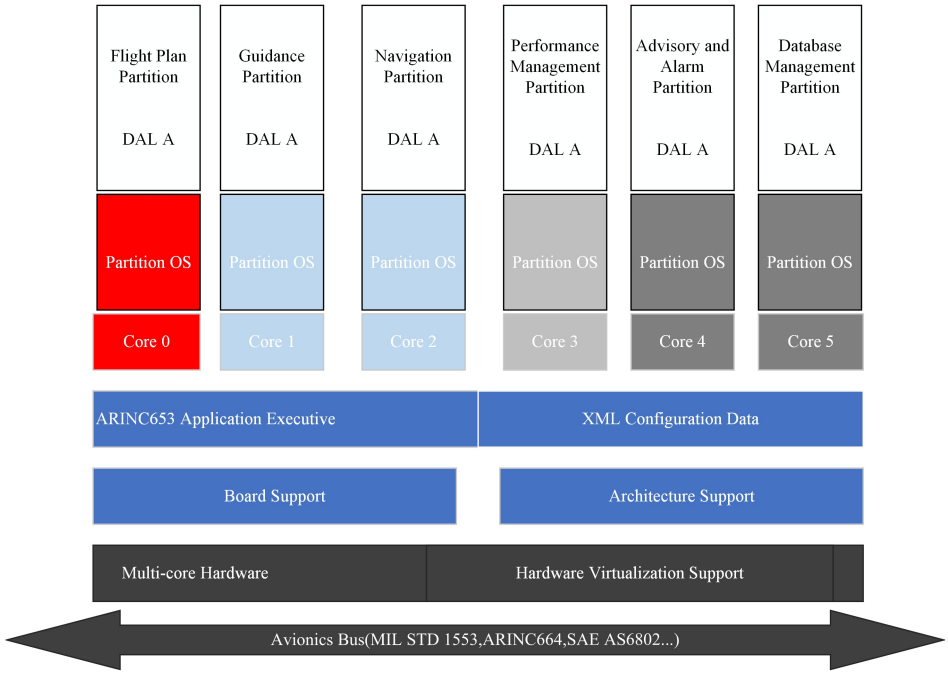

图 2 ARINC653 操作系统体系架构

Fig. 2 Architecture of ARINC 653 operating system

2.3 AADL 语言

体系结构分析与设计语言(Architecture Analysis and Design Language,AADL)是一种面向安全关键嵌入式系统的建模标准,用于设计和分析嵌入式实时系统的体系结构,描述系统的软硬件部分。AADL 定义了软件构件(数据、线程、线程组、子程序、进程)、执行平台构件(内存、总线、处理器、设备、虚拟处理器、虚拟总线)和混合构件(系统)。

AADL 支持通过属性集扩展和附件扩展两种方式进行核心语义的扩展。例如:AADL 行为附件[22](Behavior Annex,BA)能够以状态机的形式对构件内部的功能行为和构件之间的交互行为进行建模;AADL 错误模型附件(Error Model Annex)[23]支持 AADL 对系统故障行为进行建模;AADL ARINC653 附件扩展了 AADL 子集,给出了基于 AADL 语言核心表达 ARINC 653 基本元素,并给出了部分属性集扩展的定义。

3 基于扩展 AADL 的 IMA 建模方法

本章首先给出 AADL 扩展属性集 HMC4ARINC653 的定义,其次给出具体的 IMA 建模方法。

3.1 HMC4ARINC653 属性集扩展

AADL 国际标准会为了便于对符合 ARINC653 规范的综合化航空电子软件建模,制定了 AADL ARINC653 附件,给出了基于 AADL 语言表达 ARINC653 架构的建模方式,并扩展了部分针对 ARINC653 的特殊属性,但仍有部分非功能属性无法表达且不支持异构软件建模。为此,本文在 ARINC653 附件基础上进一步提出 HMC4ARINC653 属性集,使 AADL 模型通过属性绑定和参数绑定调用 C,SDL 和 SCADE 等异构构件,描述 IMA 软件的具体功能行为。部分属性集扩展如表 1 所列。

表 1 HMC4ARINC653 属性集扩展

Table 1 HMC4ARINC653 attribute set extension

HMC4ARINC653 扩展属性	描述
Max_Message_Size	采样/队列端口大小
Max_Nb_Messages	队列单条消息最大字节数
Period_Seconds	分区周期
Current_Value	信号量通信当前值
Maximum_Value	信号量通信最大值
Source_Language	异构模型实现语言
File_Path	异构模型文件路径
Module_Name	异构模型模块名称
Exception_Types	支持的异常类型

属性 Max_Message_Size 定义为 aadlinteger 类型,用于描述 AADL 数据或事件数据端口大小。属性 Max_Nb_Messages 定义为 Size 类型,用于描述 AADL 事件数据端口通信时单条消息的最大字节数。属性 Period_Seconds 定义为 Time 类型,用于描述处理器绑定的分区周期。属性 Current_Value 定义为 aadlinteger 类型,用于描述数据构件作为信号量通信时的信号量当前值。属性 Maximum_Value 定义为 aadlinteger 类型,用于描述数据构件作为信号量通信时的信号量最大值。具体属性定义如下:

Max_Message_Size: aadlinteger applies to(data port, event data port);

Max_Nb_Messages: Size applies to(event data port);

Period_Seconds: Time applies to(processor, virtual processor);

Current_Value: aadlinteger applies to(data);

Maximum_Value: inherit Time applies to(data).

属性 Source_Language 定义为枚举类型,描述 AADL 建模过程中支持的异构构件类型,如 Simulink,SCADE,SDL,C 等(默认为 C 语言)。属性 File_Path 定义为 aadlstring 类型,用于描述异构构件所在的文件路径。属性 Module_Name

定义为 aadlstring 类型，用于描述异构构件中调用的具体模块名称。属性定义如下：

Source_Language：enumeration（Simulink，SCADE，SDL，C，Ada）=> C applies to（port，thread，subprogram）；

File_Path：aadlstring applies to（thread）；

Module_Name：aadlstring applies to（subprogram，thread）.

属性 Exception_Types 定义为枚举类型，用于描述系统不同阶段出现的异常类型，目前支持端口类型异常（PORT_TYPE_EXCEPTION）、数据转换异常（DATA_CAST_EXCEPTION）、数据访问异常（DATA_ACCESS_EXCEPTION）和任务派遣异常（TASK_EXCEPTION）和警告（WARNING）这 5 种异常。属性定义如下：

Exception_Types：enumeration（

PORT_TYPE_ EXCEPTION，

DATA_CAST_ EXCEPTION，

DATA_ACCESS_ EXCEPTION，

TASK_ EXCEPTION，WARNING）=> WARNING

applies to（thread，subprogram）.

3.2 IMA 建模方法

IMA 建模框架如图 3 所示，首先对系统需求进行分解，建立 AADL 架构模型（即平台无关模型）。为满足 IMA 系统的功能属性和非功能属性，根据求精规则添加相关 ARINC653 附件元素和 HMC4ARINC653 属性集元素，对模型进行迭代求精，将平台无关模型转换为平台相关模型。

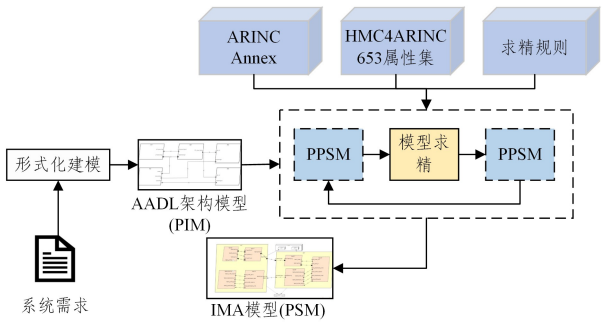

图 3 IMA 建模框架

Fig. 3 IMA modeling framework

IMA 实体与 AADL 建模元素之间的映射如表 2 所列。

表 2 IMA 实体与 AADL 建模元素映射

Table 2 Mapping of IMA entities to AADL modeling elements

IMA 实体	AADL 建模元素
模块	处理器构件
分区	绑定到虚拟处理器构件和内存构件的进程构件
进程	线程构件
队列通信	跨进程构件的事件数据端口连接
采样通信	跨进程构件的数据端口的连接
缓冲区通信	跨线程构件的事件数据端口的连接
信号量通信	进程构件内共享数据组件
事件通信	跨线程构件的事件数据端口的连接
黑板通信	进程构件内共享数据组件或跨线程构件的事件数据端口的连接
内存	存储构件

黑板通信和信号量通信可以通过进程构件内共享数据组件表示。根据两者通信语义的区别，黑板通信允许多个线程同时访问，而信号量通信只允许线程互斥访问。因此，使用属性 Concurrency_Control_Protocol 定义对共享资源访问的并发控制协议。如果数据构件中包含该属性，则认为数据构件表示信号量通信，反之，则表示黑板通信。

根据系统需求建立 AADL 模型后，通过模型求精将平台无关模型（Platform Independent Model，PIM）逐步迭代，添加设计信息及平台相关信息，每次迭代生成部分平台相关模型（Partial Platform-Specific Models，PPSM），通过多次迭代，将其转换为平台相关模型（Platform Specific Model，PSM）。以队列通信模型为例，队列端口初始时仅定义端口属性类型，为满足 IMA 系统队列通信要求，需要添加扩展属性 Max_Message_Size 定义队列端口大小，Timeout 定义进程超时等待时间，Max_Nb_Message 定义队列端口每条消息最大字节数，以及 Queueing_Discipline 定义队列通信阻塞时进程队列排队规则，最终得到符合 IMA 软件要求的队列通信模型。

4 代码生成方法

本章首先给出代码生成总体思路，其次给出框架代码、分区运行时代码、数据结构代码、功能代码和配置文件的转换规则。

4.1 总体思路

代码生成框架如图 4 所示。首先，定义考虑安全编码规范模块的代码生成模板；其次，解析 IMA 模型，并基于模板引擎分别生成对应的代码，其中 IMA 架构模型生成框架代码及系统配置文件，IMA 端口通信生成分区运行时代码，IMA 数据模型生成数据结构代码，IMA 功能行为生成具体的功能代码，生成的源代码能够通过安全检查；最后，上述代码及配置文件可以直接部署到 ARINC653 操作系统进行仿真执行。

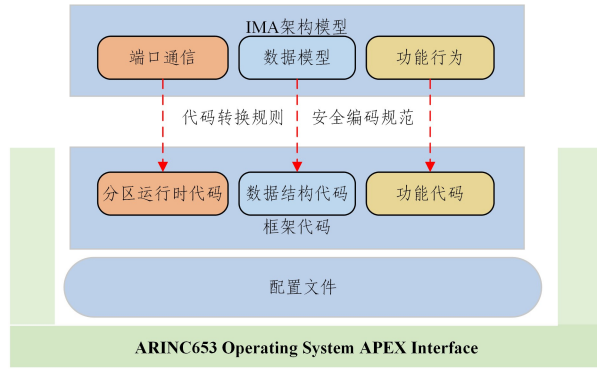

图 4 代码生成总体框架

Fig. 4 Overall framework of code generation

4.2 框架代码生成

为保证代码生成的安全性，代码生成过程中需要遵循 MISRA C：2012 开发标准，该标准通过提供编写 C 代码的指南和规则来提高嵌入式系统的安全性和可靠性。标准包括数据类型、控制结构、函数和内存管理等方面，下面列出部分安全编码规范。

S-Rule1：定义的参数或关键字禁止与基本类型关键字或 C 语言中关键字重复，参数必须使用类型声明。

S-Rule2：禁止 switch 的 case 语句无任何可执行语句或

无 default 语句。

S-Rule3：源代码中不得出现三段式序列或二段式序列。

S-Rule4：表达式的值不应该被分配给具有更窄的基本类型或不同基本类型类别的对象。

S-Rule5：内部范围内的标识符不得与外部范围内的标识符使用相同的名称。

S-Rule6：禁止将参数指针赋值给过程指针。

S-Rule7：禁止在同一表达式中调用多个相关函数。

IMA 架构模型主要由系统构件、进程构件、线程构件等组成。IMA 架构模型生成的框架代码主要包括分区进程资源分配、创建和启动、分区通信端口创建等内容。具体的转换规则如下。

Rule1：AADL 线程构件的属性映射为进程属性类型结构体 PROCESS_ATTRIBUTE_TYPE 中的元素。线程构件名映射为参数 NAME，线程构件转换的任务函数映射为参数 ENTRY_POINT，栈内存大小映射为参数 STACK_SIZE，优先级映射为参数 BASE_PRIORITY，最坏执行时间映射为参数 TIME_CAPACITY，截止期类型映射为参数 DEADLINE。当线程构件 Dispatch_Protocol 属性值设置为 Periodic 或 Sporadic 时，表示周期性或偶发性进程，PERIOD 初始化参数值等于属性值，Dispatch_Protocol 属性值为 Aperiodic 时表示非周期性进程，PERIOD 设置为 0 或 INFINITE_TIME_VALUE。

Rule2：队列端口调用函数 CREATE_QUEUING_PORT 函数进行初始化。端口属性与函数参数映射如下：端口名对应参数 QUEUING_PORT_NAME，端口大小对应参数 MAX_MESSAGE_SIZE，端口可保存的最大消息条数对应参数 MAX_NB_MESSAGE，端口方向对应参数 PORT_DIRECTION，端口排队规则对应参数 QUEUING_DISCIPLINE。

Rule3：采样端口调用函数 CREATE_SAMPLING_PORT 函数进行初始化。端口属性与函数参数映射如下：端口名对应参数 SAMPLING_PORT_NAME，端口大小对应参数 MAX_MESSAGE_SIZE，端口刷新率对应参数 REFRESH_PERIOD，端口方向对应参数 PORT_DIRECTION。

Rule4：函数 CREATE_BUFFER 用于初始化缓冲区信息。数据事件端口属性与函数参数映射如下：端口名对应参数 BUFFER_NAME，端口大小对应参数 MAX_MESSAGE_SIZE，端口最大消息条数对应参数 MAX_NB_MESSAGE。

Rule5：函数 CREATE_BLACKBOARD 用于初始化黑板信息。数据端口属性与函数参数映射如下：端口名对应参数 BLACKBOARD_NAME，端口大小对应参数 MAX_MESSAGE_SIZE。

Rule6：事件通信调用 CREATE_EVENT 函数初始化信息。事件端口属性与函数参数映射如下：端口名对应参数 EVENT_NAME。

Rule7：函数 CREATE_SEMAPHORE 用于初始化信号量信息。数据构件属性与函数参数映射如下：构件名映射为参数 SEMAPHORE_NAME，构件信号量当前值映射为参数 CURRENT_VALUE，构件信号量最大值映射为参数 MAXIMUM_VALUE，进程阻塞规则对应参数 QUEUING_DISCIPLINE。

Rule8：AADL 线程构件实例映射为函数入口方法，方法名使用〈线程名_job〉命名，线程体的循环使用 while(1)表示，循环体内包含分区通信以及功能函数调用等内容。周期性进程或偶发性进程执行完后，执行函数 PERIODIC_WAIT 挂起当前周期，直到下一个释放点到达；非周期进程执行完后，下一次执行则通过具体的事件触发。

4.3 分区运行时代码生成

IMA 端口通信由进程构件间的连接、进程构件与线程构件之间的连接，以及线程构件之间的连接等多种构件的端口连接组成。IMA 端口通信生成的分区运行时代码主要包括分区间通信、分区内通信、任务调度等内容。

分区运行时代码转换规则如图 5 所示，其中 AADL 构件以图形化方式展示。无论是分区间通信还是分区内通信，通信双方在接收或发送消息前，都需要获取端口 ID 以及端口状态，图 5 中使用 GetID 和 GetStatus 函数表示。

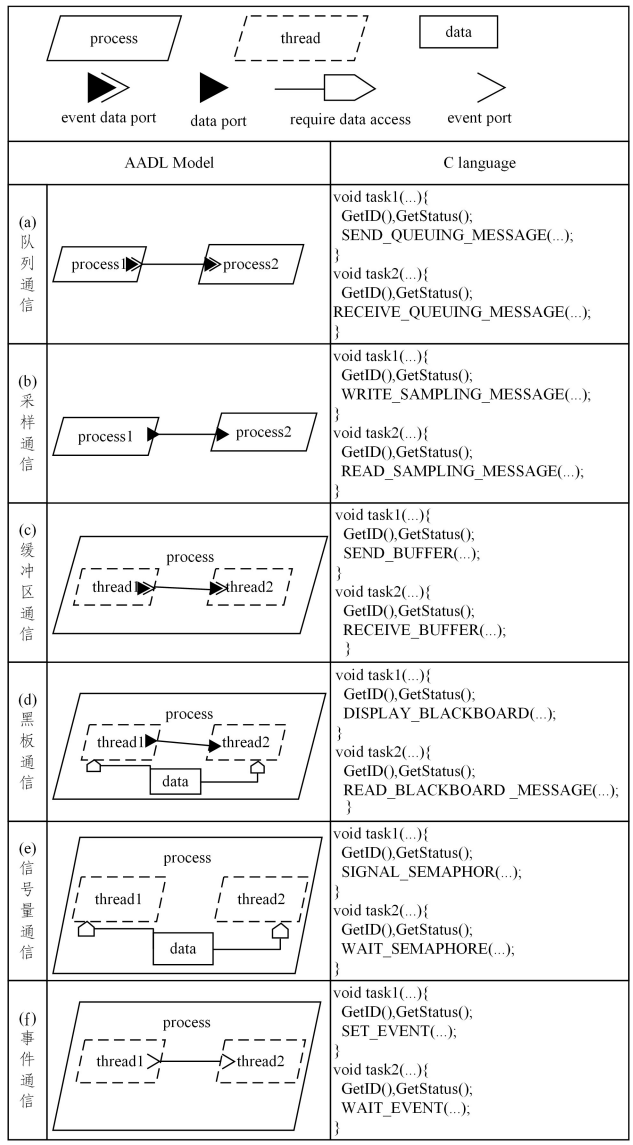

图 5 分区运行时代码转换规则

Fig. 5 Code conversion rules for partition runtime

图 5 中 6 种通信转换规则具体定义如下：

Rule9：图 5(a)表示队列通信，输入事件数据端口作为消息的接收端口，调用函数从消息队列中接收消息，输出事件数据端口向消息队列中发送消息。

Rule10：图 5(b)表示采样通信，输入数据端口作为消息的接收端口，在超时时间内等待其他端口消息。输出数据端口向指定采样端口发送消息。

Rule11：图 5(c)表示缓冲区通信，输入事件数据端口负责从缓冲区中读取消息，输出事件数据端口调用相应函数向缓冲区发送消息。

Rule12：图 5(d)表示黑板通信，输入数据端口在指定的黑板上写消息，输出数据端口从黑板上读取消息。若采用进程内共享数据构件实现黑板通信，则拥有写访问权限的线程在黑板上写消息，拥有读访问权限的线程从黑板上读取消息。

Rule13：图 5(e)表示信号量通信，定义有限信号量资源，进程通过函数获取或释放相应资源。

Rule14：图 5(f)表示事件通信，输入事件端口调用函数获得指定的事件，输出事件端口调用函数将指定事件状态置为 UP 态，所有等待该事件的进程从等待态变为就绪态。

4.4 数据结构代码生成

IMA 数据模型到 C 数据类型代码的生成主要分为简单数据类型转换和复杂数据类型转换，具体的转换算法如算法 1 所示。

算法 1 数据模型转换算法

Input：AADL 数据模型 DataImpl

Output：C 语言数据类型 CDateType

1. CDateType←setVariableName(DataImpl)
2. dataTypeImpl←getDataFeature(DataImpl)
3. aadlType←getDataType(dataTypeImpl)
3. is_Base_Type←checkBaseType(dataType)
4. If is_Base_Type＝true then
5. CDataType←transType(aadlType)
6. Else
7. dataExpre←getDataExpre(DataImpl)
8. name←getElementNames(DataImpl)
9. basetypes←getBaseTypes(DataImpl)
10. CDataType←setConfig(dataExpre，basetypes，name)
11. end If
12. return CDataType

算法 1 输入为 AADL 数据模型 DataImpl，输出为 C 数据类型 CDataType。首先将 AADL 数据模型标识符名转换为 C 变量名，解析模型中的数据特征和数据类型。然后通过函数 checkBaseType 判断 AADL 数据类型是否为简单的基本数据类型。如果是基本数据类型，则通过函数 transType 将 AADL 简单数据类型转换成 C 基本数据类型，表 3 列出了部分基本数据类型的映射；反之，则分别获取 AADL 复杂数据类型名、成员变量以及它们对应的基本数据类型，通过 setConfig 将上述信息添加到 CDataType 中。最后，返回对应的 C 数据结构代码。

表 3 基本数据类型映射

Table 3 Basic data type mapping

AADL 数据类型	C 数据类型
character	char
Integer	int
Booleans	bool
String	char[255]
Unsigned_16	short
Unsigned_64	long
Float	float
Double	double

图 6 给出了自定义的数据构件转换为对应的 C 语言数据结构，其中 A_Struct 表示结构体，包含元素 f1，f2，类型分别为 float 和 char；B_Array 表示数组，数组类型为 Integer，大小为 42；C_Float 表示基本数据类型，直接调用基本数据类型映射规则。

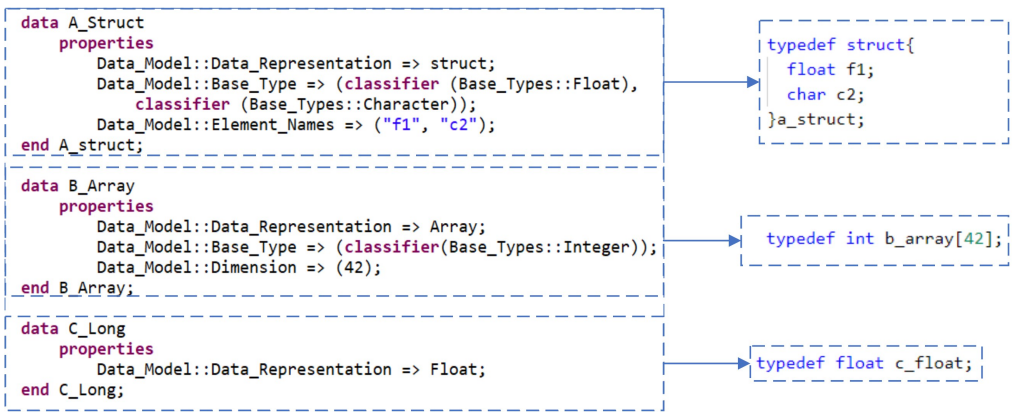

图 6 数据模型转换示例

Fig. 6 Example of data model transformation

4.5 功能代码生成

IMA 模型功能行为主要由子程序构件实现。子程序构件可以调用具体的异构构件实现相应功能。若通过 SDL，Simulink，SCADE 等构件实现，则调用第三方代码生成工具生成相应的 C 代码，在生成代码中引用相关的文件和函数；若通过 C 语言实现，则直接引用相应头文件，调用指定功能函数。

Rule15：AADL 的子程序映射为 C 语言中的函数，子程序构件的特征转换为 C 语言中的参数列表，函数名为〈subprograms_子程序名〉。子程序构件的 out parameter 对应 C

语言的输出参数，in parameter 表示 C 语言的输入参数，Source_Language 表示子程序构件调用的功能代码实现语言，Source_Name 表示调用的具体函数名，Source_Text 表示调用的文件名。

如图 7 给出了子程序构件转换示例。子程序构件 P_Spg 使用 C 语言实现，根据 Rule15 将子程序构件 P_Spg 转换为函数 subprograms_p_spg，子程序构件特征转换为 C 语言的参数，将参数传递给调用的 hash 函数；子程序 msg_handle 负责处理消息，使用 SDL 构件执行相应功能，转换为函数 subprograms_msg_handle，将参数传递给调用的 send 函数；子程序 calculate 使用 SCADE 模型实现，将特征转换为参数传递给 analyze 函数。

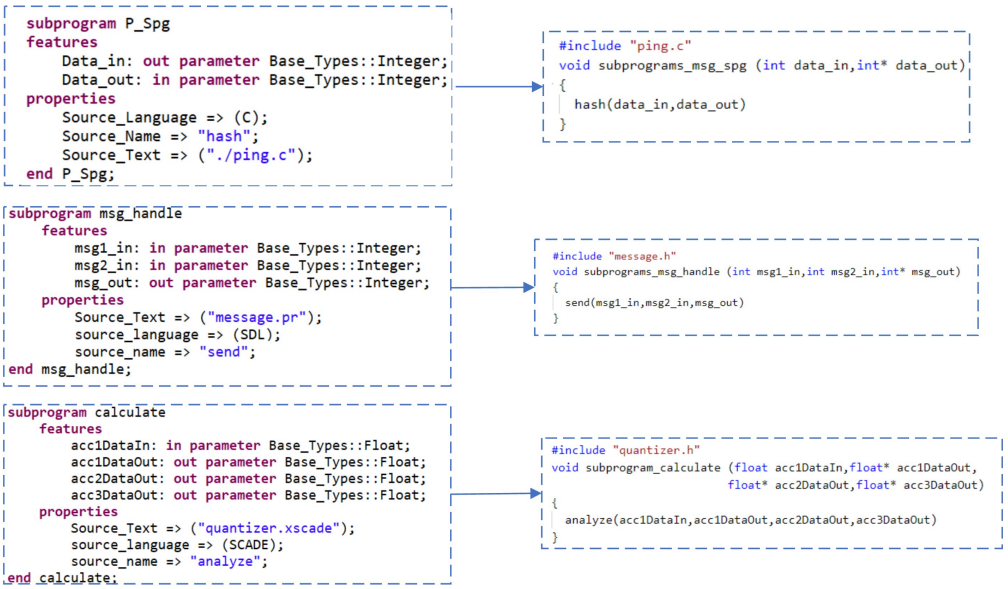

图 7 子程序构件转换示例

Fig. 7 Example of subroutine component conversion

4.6 配置文件生成

ARINC653 标准提供了可扩展的 XML 模式，用于定义 ARINC653 配置所需的数据结构。ARINC653 配置文件由 7 个主要模块组成，按照元素类型可分为：分区信息（分区基本信息表、分区内存表、分区调度表、分区间通信表）和健康监控（Health Monitoring，HM）。

IMA 架构模型到系统配置信息转换算法如算法 2 所示。

算法 2 配置文件生成算法

输入：IMA 实例模型 System

输出：ARINC653 系统配置信息 systemConfig

```
processImpls←getProcesses(System)
memoryMap←getMemoryBinding(System)
processorMap←getProcessorBinding(System)
connections← getAllConnections(System)
for each process in processImpls
    infos←walkPartitionInfo(process)
    features←getAllFeaturesInfo(processImpl)
    basicConfig←genPartitionInfo(infos,features)
    memory←memoryMap.get(process)
    memInfo←walkMemoryInfo(memory)
    memConfig←genMemConfig(infos,memInfo)
    processor←processorMap.get(process)
    scheduleInfo←walkScheduleInfo(processor)
    hmInfo←walkHMInfo(process,processor)
    scheduleConfig←genScheduleConfig(infos,scheduleInfo)
    hmConfig←genHMConfig(hmInfo)
end for
for each connection in connections
    source←getSourceContext(connection)
    dest←getDestinationContext(connection)
    commConfig←genCommConfig(source,dest)
end for
systemConfig←Integrate config
```

算法 2 的输入为模型实例化对象 SystemImpl，输出为系统配置信息表。首先，获取系统构件下所有的进程子构件、进程构件与内存构件、处理器构件之间的绑定以及进程间的连接信息；其次，对进程构件依次遍历，解析进程构件的基本信息和特征信息，将进程信息和进程端口信息写入分区基本信息表中（basicConfig），将进程信息和存储构件的属性信息写入分区内存表中（memConfig），将进程信息和分区调度信息写入分区调度表中（scheduleConfig），解析进程构件和处理器构件，获取系统中定义的健康监控信息，写入健康监控表中（hmConfig），对进程间连接进行遍历，获取连接的源端口信息和目标端口信息，将上述信息写入分区间通信表中（commConfig）；最后，将上述信息进行集成得到系统配置文件。

5 工具设计与案例分析

5.1 代码生成工具

IMACGT（Integrated Modular Avionics Code Generation Tool）工具采用模块化结构，其提供了扩展接口，可以灵活配置代码模板，支持 AADL 模型到 C 代码以及配置文件的生成，工具整体框架如图 8 所示。

IMACGT 工具主要分为 4 个模块：

1）IMA 模型解析模块：将 IMA 实例化模型解析为抽象语法树，为叶子节点添加相关属性生成实例语法树。

2）AADL2C：将实例语法树转换为目标语言法树，生成相应的框架代码、功能代码、分区运行时代码、数据结构代码。

3）AADL2XML：抽取模型中的关键信息，生成符合 ARINC653-scheme 的系统配置文件。

4）代码集成：将上述生成的代码集成，得到面向 ARINC653 操作系统的可执行代码。

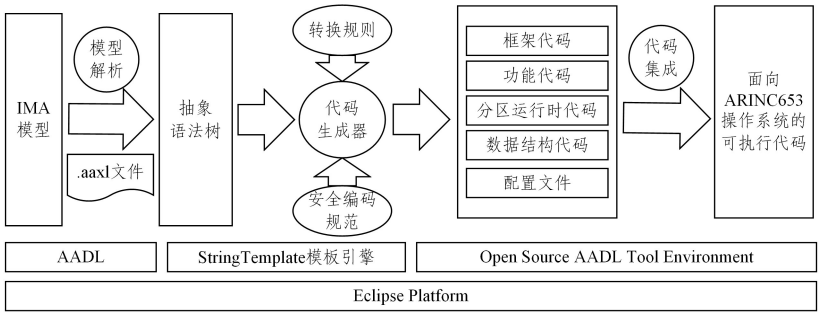

图 8 IMACGT 工具结构

Fig. 8 Structure of IMACGT

IMACGT 工具基于 Java 实现，开发环境为基于 Eclipse 平台的 OSATE 插件环境。工具每个模块的代码规模如表 4 所列。

表 4 工具实现数据统计

Table 4 Statistics of tool implementation

原型工具	具体功能	代码规模(行)
IMACGT	IMA 模型解析	2100+
	AADL2C	4700+
	AADL2XML	3900+
	代码粘合	1600+
	代码模板	1400+

工具的主要特点包括：

1）代码生成方面：IMACGT 工具采用模块化架构，允许用户针对不同的目标语言进行扩展，通过自定义代码模板，支持面向不同类型的 ARINC653 操作系统的代码自动生成。

2）系统应用方面：与工业界合作，针对典型 IMA 系统，通过 IMACGT 工具，能够生成面向 ARINC653 操作系统的可执行代码及系统配置文件，并在真实环境下进行仿真执行。

5.2 案例分析

5.2.1 案例介绍

图 9 给出了一个简化的航空电子飞行管理系统（Flight Management System，FMS）架构图。FMS 由 6 个主要模块组成：导航、性能管理、飞行引导、飞行计划管理、数据库管理、咨询和警报，它们分别具有以下功能：

1）导航模块：负责确定飞机当前时间的位置，执行导航计算，并自动调整导航站管理；完成飞机横截面飞行管理，并按照预定路线引导飞机到达目的地。

2）性能管理模块：在飞行过程中，计算飞机的相关性能指标，包括飞行高度、速度、爬升速度和下降速度，以获得最佳的垂直预选飞行路径。基于性能数据库提供的基准数据、外部传感器的一些信号数据，以及控制显示部件上的输入数据，完成飞机的纵向截面管理。

3）飞行引导模块：当飞机沿着预选航线飞行，受干扰或导航不确定性的影响而偏离预选航线时做出相应的决策。它分为水平引导和垂直引导。水平引导是在水平面内按照一定的控制速率控制飞机实际轨迹与预选路径在水平平面上的偏差；垂直引导则是基于舱容率控制实际轨迹相对于预选轨迹在垂直平面上的偏差。

4）飞行计划管理模块：通过控制显示部件将飞机飞行计划输入 FMS 中，包括飞行水平轨迹和垂直飞行计划中的速度和高度倾斜角。在计算出飞机的位置并将其与实际位置进行比较后，根据它们之间的误差生成飞机控制计算和推力管理计算指令；然后，飞行控制计算机和推力管理计算机生成相应的控制命令，通过 FMS 实现航线的自动驾驶控制。

5）数据库管理模块：飞行管理系统包含性能数据库和导航数据库。性能数据库是飞机性能管理的基础，用于执行相应的性能计算，确保飞机沿着最佳的垂直轮廓飞行。导航数据库旨在为飞机提供自动导航能力，从起飞到降落。它存储整个区域的导航信息，包括机场、路标、导航站地理位置、频率和飞机飞行区域的路线构成结构。

6）咨询和警报模块：飞行员可以通过控制显示部件获取咨询信息，例如飞行轮廓相关信息、性能相关信息和系统故障信息等。同时，FMS 还提供自动警报功能，如风切变警报、接近地面警报、TCAS 警报等。

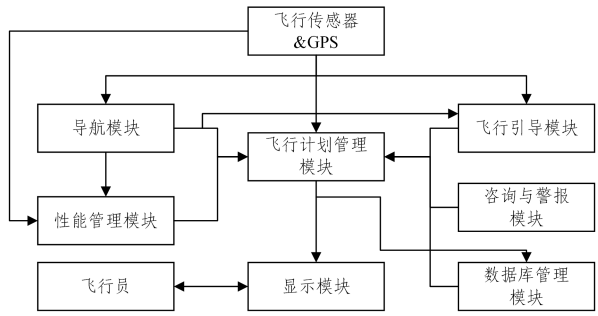

图 9 飞行管理系统简化架构图

Fig. 9 Simplified architecture of FMS

我们将 FMS 部署在 ARINC653 操作系统上，根据系统每个任务的执行时间构建任务向量 $\boldsymbol{T}=\{E,P,W\}$，其中 E 表示任务的执行时间，P 表示任务的周期，W 表示任务的最坏执行时间，时间单位为 ms。最终得到的案例数据如表 5 所列。

表 5 FMS 系统案例数据

Table 5 Example data of FMS

分区名	任务	任务向量	优先级
导航	Tsk11	(20,40,22)	2
	Tsk12	(15,50,18)	3
	Tsk13	(25,40,30)	4
	Tsk14	(40,100,42)	5
	Tsk15	(50,200,54)	6
性能管理	Tsk21	(50,200,56)	2
	Tsk22	(10,100,11)	3
	Tsk23	(25,100,27)	4
飞行引导	Tsk31	(20,50,26)	2
	Tsk32	(10,50,12)	3
	Tsk33	(15,50,18)	4
飞行计划管理	Tsk41	(60,200,68)	2
	Tsk42	(80,200,90)	3
	Tsk43	(40,200,42)	4
	Tsk44	(55,100,56)	5
	Tsk45	(15,50,18)	6
	Tsk46	(75,300,77)	7
数据库管理	Tsk51	(50,150,54)	2
	Tsk52	(60,150,63)	3
咨询和警报	Tsk61	(40,120,43)	2
	Tsk62	(60,120,65)	3

5.2.2 飞行管理系统建模

针对飞行管理系统，我们构建了 AADL 的体系结构模型，包括 6 个进程、21 个线程、149 个子程序、23 个数据类型、6 个存储器、192 个端口和 274 个连接。如图 10 所示，根据飞行管理系统需求建立 6 个分区，分别是：导航分区、性能管理分区、飞行引导分区、飞行计划管理分区、数据库管理分区和咨询与警报分区。

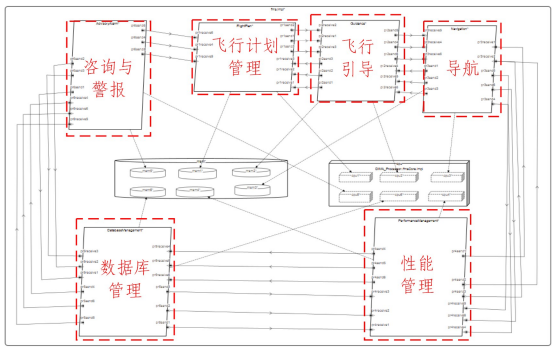

图 10 FMS 系统 AADL 体系结构模型

Fig. 10 AADL architecture model of FMS

5.2.3 代码及配置文件生成

首先，基于配置文件生成工具 AADL2XML 生成系统配置文件，定义系统每个分区的端口信息、内存信息、调度信息和分区间通信信息；然后通过代码生成工具 AADL2C 为每个分区生成对应的框架代码、分区运行时代码、功能代码、数据结构代码；最后将上述代码和配置文件进行集成。

对 FMS 模型进行实例化得到 AAXL 文件，通过 AADL2XML 工具生成系统配置文件。表 6 列出了各个分区的配置信息的行数，导航分区配置共计 306 行，性能管理分区配置共计 267 行，飞行引导分区配置共计 302 行，飞行计划管理分区配置共计 593 行，数据库管理分区共计 283 行，咨询与警报分区共计 220 行。其中飞行计划管理模块作为 FMS 系统的核心模块，负责多个模型信息的接收和处理，占系统配置文件的 30.08%。

表 6 配置文件数据统计

Table 6 Data statistics of configuration files

分区名	分区基本信息	分区内存信息	分区调度信息	分区间通信	健康监控
导航	82	48	22	86	68
性能管理	24	70	34	52	87
飞行引导	18	58	81	49	96
飞行计划管理	49	77	141	184	142
数据库管理	34	64	64	73	48
咨询警报	27	42	52	47	52

通过 AADL2C 工具为每个分区生成相应的代码，各个分区代码量如表 7 所列。

表 7 分区代码统计

Table 7 Partition code statistics

分区名	代码行数
导航	3800+
性能管理	2800+
飞行引导	3400+
飞行计划管理	5100+
数据库管理	2400+
咨询预警报	1700+

以 FMS 系统中的导航分区为例，介绍导航分区代码生成过程。导航分区中共包含 5 个进程，分别负责水平导航处理、垂直导航处理、导航数据处理、传感器数据处理和导航控制。图 11 展示了导航分区内定义的进程配置信息，包括起始地址、基本优先级、最坏执行时间、周期等信息。

```
<PROCESSCONF description="Navigation">
    <PROCESS Name="task11" BasePriority="2" DeadLineType="SOFT_DEADLINE" EntryAddress="84519"
            ExecTime="20000000" FloatDemand="false"  Period="40000000" StackSize="2048" WCET="22000000"
    <PROCESS Name="task12" BasePriority="3" DeadLineType="SOFT_DEADLINE" EntryAddress="175489"
            ExecTime="15000000" FloatDemand="false"  Period="50000000" StackSize="1024" WCET="18000000"
    <PROCESS Name="task13" BasePriority="4" DeadLineType="SOFT_DEADLINE" EntryAddress="261401"
            ExecTime="25000000" FloatDemand="false"  Period="40000000" StackSize="8192" WCET="30000000"
    <PROCESS Name="task14" BasePriority="5" DeadLineType="SOFT_DEADLINE" EntryAddress="453975"
            ExecTime="40000000" FloatDemand="false"  Period="50000000" StackSize="4096" WCET="42000000"
    <PROCESS Name="task15" BasePriority="6" DeadLineType="SOFT_DEADLINE" EntryAddress="956204"
            ExecTime="50000000" FloatDemand="false"  Period="50000000" StackSize="2048" WCET="54000000"

</PROCESSCONF>
```

图 11 导航分区进程配置信息

Fig. 11 Configuration information of navigation partition processes

如图 12 所示，Tsk11 定义了水平导航进程的参数信息，其中周期为 40ms，优先级为 2，栈大小为 8Kbyte，最坏执行时间为 30ms，生成的框架代码如图 12 右侧所示。根据 AADL 模型定义的信息，创建相应的分区通信端口，对 AADL 模型中的数据进行转换，为进程分配相应的资源，创建并启动进程。

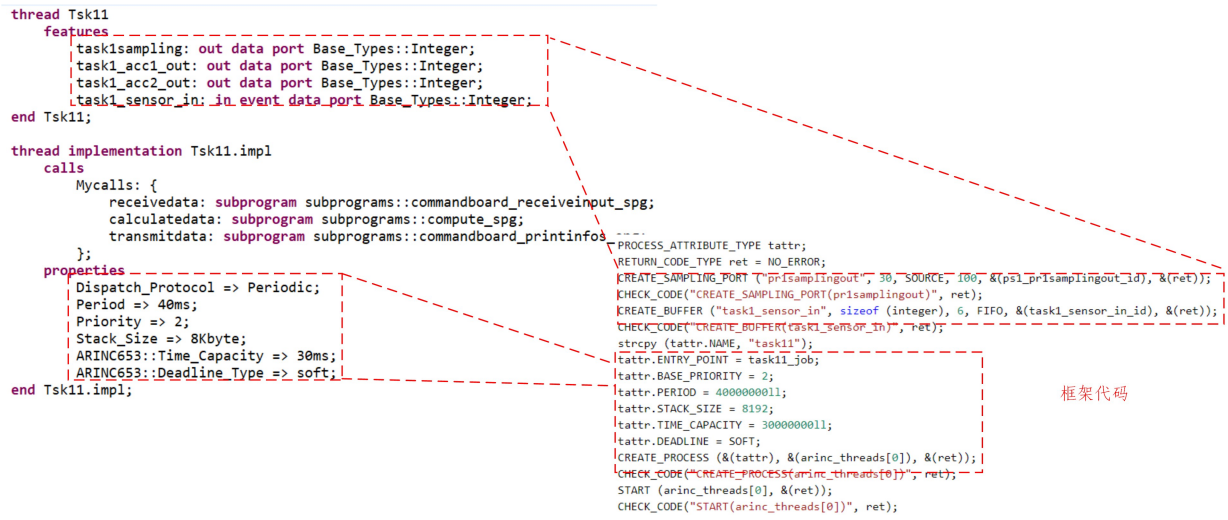

图 12 框架代码映射示例

Fig. 12 Example of framework code mapping

在 ARINC653 操作系统开发环境中创建工程项目，将生成的配置文件及代码复制到对应的分区目录下，对项目进行构建，启动 QEMU 虚拟机，通过网络进行数据传输，将构建成功的工程文件通过 RAM 部署到虚拟机上，代码运行效果如图 13 所示。

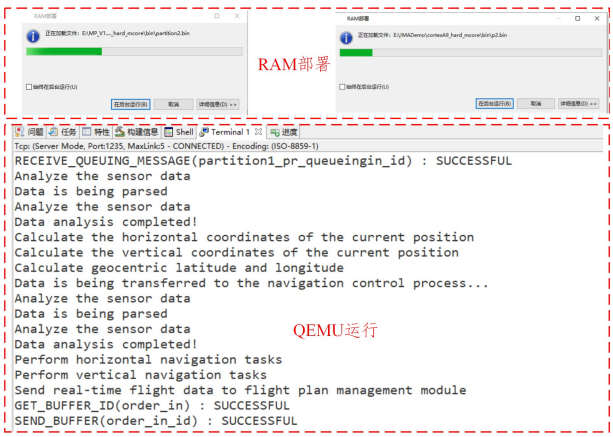

图 13 QEMU 部署运行

Fig. 13 Deployment and execution of QEMU

5.3 实验结果分析

Cppcheck 是一个开源的静态代码分析工具，用于检查 C 和 C++代码中的错误、警告和代码质量问题，包括语法错误、内存泄漏、未初始化的变量等问题，还支持用户自定义规则。我们将生成的代码导入 Cppcheck 进行静态分析，分析结果如图 14 所示，代码符合 MISRA C 安全编码规范且不存在语法错误。

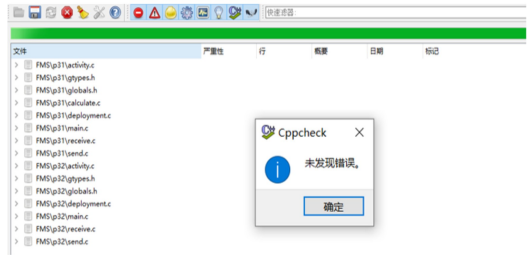

图 14 代码分析结果

Fig. 14 Results of code analysis

将本文的 IMA 平台配置及代码生成方法与其他方法进行对比，评估结果如表 8 所列。

表 8 代码生成工具方法和评价

Table 8 Code generation tool methodology and evaluation

代码生成方法	支持标准	可扩展性	平台相关性	功能代码集成	输出	工具
Gilles Lasnier	ARINC653	一般	部分支持	不支持	C,Ada,XML	OCARINA
Laurent Pautet	POSIX,ARINC653	较好	部分支持	不支持	C,XML	RAMSES
John Hatcliff	seL4,Linux	较好	不支持	不支持	C,Scala	HAMR
Gianluca	ARINC653	一般	支持	不支持	C	HIL
Hu 等	—	一般	不支持	不支持	C	—
Wang 等	ARINC653	一般	部分支持	不支持	RT-JAVA	—
本文方法	ARINC653	较好	支持	支持	C,XML	IMACGT

在文献[19]中，工具支持 C 和 Ada 及配置文件生成，平台相关部分信息主要通过代码生成器硬编码实现，而且还需要工程师手工集成功能代码；文献[20,24]支持 C 代码及配置文件生成，但生成的配置文件与 ARINC653 定义的 XML 文件标准存在部分差异，生成的代码仅包含部分平台相关信息；文献[25-26]针对的是通用代码生成，不包含平台相关信息；文献[27]基于 ARINC653 架构的 AADL 模型自动生成 RT-Java 代码，但 RT-Java 在嵌入式软件中较少使用且 ARINC653 标准中没有定义 RT-JAVA 相关接口。本文方法能够完整支持 ARINC653 平台相关信息并且支持功能代码的集成，生成的代码及配置能够直接部署到 ARINC653 操作系统上仿真执行。

6 相关工作

6.1 异构建模

华东师范大学何积丰院士和刘静教授等提出了面向 CPS 系统建模的 AADL 扩展，即 AADL+[28]；清华大学贺飞教授等给出了可组合嵌入式软件建模与验证技术研究综述[29]，总结了嵌入式软件组合理论的基本概念，并指出基于组合设计与开发方法在大规模、复杂嵌入式软件开发中有良好的应用前景。Zhe 等[30]提出了基于 SysML 和 AADL 的安全关键系统设计、验证及代码生成方法，SysML 用于系统顶层建模与分析，AADL 用于实现系统的具体功能行为，通过对 AADL 模型不断精化进行软件设计与实现。

6.2 代码自动生成

Rahmoun 等介绍了一种基于 AADL 的实时嵌入式系统的模型细化和时序分析框架，并给出了求精后模型的代码生成框架，生成的代码符合改进后的模型，并实现了相应的原型工具 RAMSES，支持 POSIX，ARINC653，以及 OSEK 等平台上的模型求精及其代码生成。美国堪萨斯州立大学 John Hatcliff 等开发了“基于模型的高保障嵌入式系统快速工程”(High-Assurance Model-based Rapid engineering for embedded systems，HAMR) 工具，它可以从 AADL 系统架构模型中生成高保障软件。目前 HAMR 支持多个平台的代码生成，包括 Java 虚拟机、Linux 等快速原型开发平台，以及通过形式验证的 seL4 空间分区微内核。Conlin 等提出了一种 Keras2C[31]的代码自动生成方法，能够将 Keras/TensorFlow 等神经网络模型转换为实时兼容的 C 代码，但在处理大型神经网络模型时需要使用动态内存分配，生成的代码不支持 WCET(Worst Case Execution Time)分析，无法满足适航认证要求。Silva 等[32]提出了一种面向航空电子平台的代码自动生成方法，通过 TensorFlow/Keras 训练的全连接神经网络实现 C 代码自动生成，并支持使用 OTAWA 进行代码 WCET 分析。然而，该方法生成的代码并未考虑航空电子应用软件对内存和强实时性的严格限制。

结束语 本文提出了一种面向 ARINC653 操作系统的综合化航空电子软件代码生成方法。首先，提出扩展属性集 HMC4ARINC653，使其具备描述综合化航空电子系统的软件架构、异构功能行为和非功能属性的能力，并给出了基于 ARINC653 附件和 HMC4ARINC653 属性集的 IMA 软件建模方法；其次，设计代码转换规则并考虑安全编码规范，针对每个分区生成相应的框架代码、分区运行时代码、数据结构代码、功能代码以及系统配置文件，上述代码和配置可以直接部署到 ARINC653 操作系统上仿真执行；最后，实现了原型工具 IMACGT，并基于实际工业界案例和 ARINC653 操作系统，验证了本文所提方法和工具的有效性。

在未来的工作中，我们将首先考虑使用定理证明器 Coq 来完成从模型到代码的语义一致性证明；其次，随着多核处理器的广泛应用，研究面向多核处理器的并行代码生成；最后，引入机器学习或大模型方法，对现有代码进行特征提取训练，在代码生成过程中对相应代码片段进行智能化推荐。

参 考 文 献

[1] GARSIDE R, PIGHETTI F J. Integrating modular avionics: A new role emerges[J]. IEEE Aerospace and Electronic Systems Magazine, 2009, 24(3): 31-34.

[2] Airlines Electronic Engineering Committee. Avionics Application Software Standard Interface: ARINC Specification 653P1-2 [M]. Aeronautical Radio, 2006: 11-21.

[3] YANG Z B, YUAN S H, XIE J, et al. A synchronous language multithreaded code automatic generation tool[J]. Journal of Software, 2019, 30(7): 1980-2002.

[4] SINGH P, SINGH L K. Reliability and Safety Engineering for Safety Critical Systems: An Interview Study With Industry Practitioners[J]. IEEE Transactions on Reliability, 2021, 70(2): 643-653.

[5] RTCA DO-178C. Software Considerations in Airborne Systems and Equipment Certification [S]. Washington, DC: RTCA, 2011.

[6] DO-331, Model-based development and verification supplement to DO-178C and DO-278A[S]. RTCA: Washington, DC, USA, 2011.

[7] DO-333, Formal methods supplement to DO-178C and DO-278A [S]. RTCA: Washington, DC, USA, 2011.

[8] LE SERGENT T. SCADE: A comprehensive framework for critical system and software engineering[C]// International SDL Forum. Berlin: Springer, 2011: 2-3.

[9] URSU C, BHAT R, DAMODARAN R. Simulink © modeling for vehicle simulator design[R]. SAE Technical Paper, 2011.

[10] FRIEDENTHAL S, MOORE A, STEINER R. OMG systems modeling language(OMG SysML) tutorial[C]// INCOSE Intl. Symp. 2006: 65-67.

[11] YANG Z B, PI L, HU K, et al. AADL: An Architecture Design and Analysis Language for Complex Embedded Real-Time Systems[J]. Journal of Software, 2010, 21(5): 899-915.

[12] FONS-ALBERT B, USACH-MOLINA H, VILA-CARBÓ J, et al. Development of integrated modular avionics applications based on Simulink and XTRATUM[J]. Data Systems in Aerospace, 2013, 720: 1-15.

[13] DING R, YU Q H. Growth Framework of Autonomous Unmanned Systems Based on AADL[J]. Computer Science, 2020, 47(12): 87-92.

[14] LEE E A. Fundamental limits of cyber-physical systems modeling[J]. ACM Transactions on Cyber-Physical Systems, 2016, 1(1): 1-26.

[15] ZHAN H, LIN Q, WANG S, et al. Unified graphical co-modeling of cyber-physical systems using AADL and simulink/stateflow[C]// Unifying Theories of Programming: 7th International Symposium (UTP 2019). Springer International Publishing, 2019: 109-129.

[16] YANG Z, BODEVEIX J P, FILALI M. Towards a simple and safe Objective Caml compiling framework for the synchronous language SIGNAL[J]. Frontiers of Computer Science, 2019, 13: 715-734.

[17] PERROTIN M, GROCHOWSKI K, VERHOEF M, et al.

TASTE in action[C]// 8th European Congress on Embedded Real Time Software and Systems(ERTS 2016). 2016:1-3.

[18] TAN S Y. Fast Design and Verification of Flight Control Law for Small Compound UAV[J]. Computer Science, 2020, 47(S1): 651-656.

[19] LASNIER G, ZALILA B, PAUTET L, et al. Ocarina: An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications[J]. Ada Europe, 2009, 5570: 237-250.

[20] RAHMOUN S, MEHIAOUI-HAMITOU A, BORDE E, et al. Multi-objective exploration of architectural designs by composition of model transformations[J]. Software & Systems Modeling, 2019, 18: 107-127.

[21] RUAN W, ZHAI Z. Kernel-level design to support partitioning and hierarchical real-time scheduling of ARINC 653 for VxWorks[C]// 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing. IEEE, 2014: 388-393.

[22] DISSAUX P, BODEVEIX J P, FILALI M, et al. AADL behavioral annex[C]// Proceedings of DASIA Conference. 2006.

[23] LARSON B, HATCLIFF J, FOWLER K, et al. Illustrating the AADL error modeling annex(v. 2) using a simple safety-critical medical device[J]. ACM SIGAda Ada Letters, 2013, 33(3): 65-84.

[24] CORRARO G, BOVE E, GARBARINO L, et al. A novel approach for the development and coding of avionics functionalities for IMA architectures[C]// 2018 IEEE/AIAA 37th Digital Avionics Systems Conference(DASC). IEEE, 2018: 1-8.

[25] HATCLIFF J, ROBBY B J, CARPENTER T, et al. HAMR: An AADL multi-platform code generation toolset[C]// Leveraging Applications of Formal Methods, Verification and Validation: 10th International Symposium on Leveraging Applications of Formal Methods(ISoLA 2021). Springer International Publishing, 2021: 274-295.

[26] HU K, DUAN Z, WANG J, et al. Template-based AADL automatic code generation[J]. Frontiers of Computer Science, 2019, 13: 698-714.

[27] WANG Y, MA D, ZHAO Y, et al. Automatic RT-Java code generation from AADL models for ARINC653-based avionics software[C]// 2012 IEEE 36th Annual Computer Software and Applications Conference. IEEE, 2012: 670-679.

[28] LIU J, LI T, DING Z, et al. AADL+: a simulation-based methodology for cyber-physical systems[J]. Frontiers of Computer Science, 2019, 13: 516-538.

[29] WANG B, BAI X Y, HE F. Survey on Modeling and Verification Techniques of Composable Embedded Software[J]. Journal of Software, 2014, 25(2): 234-253.

[30] ZHE W, HUGUES J, CHAUDEMAR J C, et al. An integrated approach to model based engineering with SysML, AADL and FACE[R]. SAE Technical Paper, 2018.

[31] CONLIN R, ERICKSON K, ABBATE J, et al. Keras2c: A library for converting Keras neural networks to real-time compatible C[J]. Engineering Applications of Artificial Intelligence, 2021, 100: 104182.

[32] SILVA I D A, CARLE T, GAUFFRIAU A, et al. Automatic predictable C code generation of machine learning models for avionics systems[J/OL]. https://etr2021. ensma. fr/files/01_phdstudents_session_iryna. pdf.

LING Shixiang, born in 2000, postgraduate. His main research interests include safety-critical system and formal verification.

YANG Zhibin, born in 1982, Ph.D, professor, is a member of CCF (No. 08632M). His main research interests include safety-critical system, formal verification and AI software engineering.

（责任编辑：何杨）