软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(3):1240−1263 [doi: 10.13328/j.cnki.jos.007495] [CSTR: 32375.14.jos.007495]

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

嵌入式软件 IP 通用模型*

徐 雄[1], 马 旭[1,2], 高 强[1,2], 计泽坤[1,2], 王淑灵[1], 詹博华[1,2], 詹乃军[1,2], 李晓锋[3], 顾 斌[3], 董晓刚[3], 杨孟飞[4], 刘 洋[5], 冀振燕[5]

[1](计算机科学国家重点实验室 (中国科学院 软件研究所), 北京 100190)

[2](中国科学院大学, 北京 100049)

[3](北京控制工程研究所, 北京 100190)

[4](中国空间技术研究院, 北京 100094)

[5](北京交通大学 软件学院, 北京 100044)

通信作者: 顾斌, E-mail: gubinbj@sina.com

摘 要: 软件 IP (intellectual property) 是具有知识产权的可复用的软件知识实体, 是软件智能合成的基石. 针对嵌入式系统的关键特性, 提出了面向嵌入式系统的软件 IP 通用模型, 包括知识模型、形式模型和实现, 并且讨论了它们三者之间的一致性关系. 与目前的主流模型相比, 该方法充分考虑了嵌入式系统的关键特性、系统对环境和平台的假设、软件中间知识的表示和使用、模型组装的正确性, 以及模型与实现之间的关系, 因此具有明显的优势. 提出的嵌入式软件 IP 通用模型在一定程度上揭示了软件构成的本质, 即软件并非代码的集合, 而是知识、规约和代码的三位一体. 此外, 为了简化软件 IP 的使用, 根据不同的使用目的 (关注点), 将软件 IP 表示为不同的视图. 最后, 提出从存量嵌入式软件资产提取软件 IP 的方法, 并且通过实际案例展示了提取方法的有效性和可行性.

关键词: 嵌入式系统; 软件 IP; 智能合成; 知识; 形式模型

中图法分类号: TP311

中文引用格式: 徐雄, 马旭, 高强, 计泽坤, 王淑灵, 詹博华, 詹乃军, 李晓锋, 顾斌, 董晓刚, 杨孟飞, 刘洋, 冀振燕. 嵌入式软件IP通用模型. 软件学报, 2026, 37(3): 1240–1263. http://www.jos.org.cn/1000-9825/7495.htm

英文引用格式: Xu X, Ma X, Gao Q, Ji ZK, Wang SL, Zhan BH, Zhan NJ, Li XF, Gu B, Dong XG, Yang MF, Liu Y, Ji ZY. General Model of Embedded Software IP. Ruan Jian Xue Bao/Journal of Software, 2026, 37(3): 1240–1263 (in Chinese). http://www.jos.org.cn/1000-9825/7495.htm

General Model of Embedded Software IP

XU Xiong[1], MA Xu[1,2], GAO Qiang[1,2], JI Ze-Kun[1,2], WANG Shu-Ling[1], ZHAN Bo-Hua[1,2], ZHAN Nai-Jun[1,2], LI Xiao-Feng[3], GU Bin[3], DONG Xiao-Gang[3], YANG Meng-Fei[4], LIU Yang[5], JI Zhen-Yan[5]

[1](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](Beijing Institute of Control Engineering, Beijing 100190, China)

[4](China Academy of Space Technology, Beijing 100094, China)

[5](School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China)

Abstract: Software intellectual property (IP) is a reusable software knowledge entity with intellectual property, serving as the basis for intelligent software synthesis. In view of the key features of the embedded systems, this study proposes a general model of software IP for the systems, including the knowledge model, formal model, and implementation, and discusses the consistency between the three. Compared with the current mainstream models, the proposed method fully considers the key features of embedded systems, assumptions

* 基金项目: 国家自然科学基金 (62192732, 62192730)
收稿时间: 2024-02-01; 修改时间: 2025-03-21; 采用时间: 2025-06-20; jos 在线出版时间: 2025-11-13
CNKI 网络首发时间: 2025-11-14

made by the system about the environment and platform, representation and utilization of the immediate knowledge of software, correctness of model assembly, and relations between model and implementation. Therefore, this method has remarkable advantages. The proposed general model reveals the essential nature of software construction to some extent. Software is not merely a set of codes, but should be a combination of knowledge, specification, and codes. Additionally, for simplifying the utilization, this study shows software IP in different views according to the utilization purposes (focuses). Finally, this study puts forward an approach of extracting software IP from existing embedded software assets and validates the effectiveness of this extraction method through practical cases.

Key words: embedded system; software IP; intelligent synthesis; knowledge; formal model

复杂嵌入式系统广泛应用于国民经济、国防等关键领域, 例如航空航天、轨道交通、核电等. 在这些领域, 一方面嵌入式系统实现功能越来越多, 软件规模变得越来越庞大和复杂; 另一方面, 系统安全至关重要, 如果发生错误将会给生命和经济等带来重大灾难性后果. 因此, 如何高效开发可靠的复杂嵌入式软件是一个重大挑战. 嵌入式系统通常具有如下几个关键特性.

(1) 硬件依赖: 嵌入式系统包括硬件和软件的组合, 软件需要通过硬件才能与物理环境进行交互, 并且软件的性能依赖于硬件平台的配置.

(2) 领域相关: 嵌入式系统的设计通常是面向领域的, 不同的领域对系统的关注点也不同.

(3) 资源限制: 嵌入式系统都是以尽可能少的资源运行所需的计算.

(4) 实时性: 嵌入式系统必须对环境的实时变化做出相应的反馈.

(5) 安全性: 嵌入式系统的运行错误可能导致灾难性后果.

基于嵌入式系统的这些特性, 人们提出了各种提高嵌入式软件开发效率和质量保障的方法. 这些方法的基本思路是通过抽象/精化、组合/分解、软件复用等技术, 对软件进行建模和封装, 构造能够复用的软件功能单元, 最后将不同的软件单元进行组合形成满足需求的嵌入式软件系统.

目前主流的方法包括模型驱动设计 (model-driven design, MDD)[1−5]、基于构件的设计 (component-based design, CBD)[6−9]和基于契约的设计 (design by contract, DbC)[10]等. MDD 从嵌入式系统的需求出发, 首先构造满足需求的抽象模型, 进而对模型进行仿真、测试和验证, 最后生成模型的代码. MDD 的模型模块的功能是独立的、完整的, 但是, ① MDD 主要通过仿真发现设计中的错误, 而仿真无法覆盖系统的所有运行场景, 因此会有设计缺陷在系统实现时甚至部署后才被发现; ② 并不是所有代码都可以基于模型自动生成, 并且模型和实现之间的一致性难以保证; ③ MDD 难以描述模型功能及对其他模块和执行平台的假设和约束. CBD 将软件封装成可复用的构件, 通过对可复用的构件进行组装来进行软件资产的重复利用. CBD 中具有详细的功能描述和接口协议, 但是, ① 构件对嵌入式环境的假设以及对环境保证的行为的规约描述尚显不足; ② 由于构件缺乏明确的统一标准并且构件通常由不同人员开发, 导致文档不全, 质量得不到保证, 由构件组装成的嵌入式系统的正确性很难直接由构件的正确性保证, 存在构件的错误使用和配置的风险; ③ 构件只是功能的部分实现, 需要通过请求接口请求外部的服务才能完全实现构件的功能, 因此构件不是功能独立的; 此外, ④ 构件采用的请求/提供服务模式更适用于分布式系统, 而不是专门针对嵌入式系统. DbC 使用严谨的数学语言对构件的行为、构件之间的交互以及关于环境的假设建立形式化规约, 部分解决了构件可理解性和正确组合的问题. 然而, ① DbC 中的构件不是功能完整的, 它缺少对实现和功能的具体描述, 对契约的描述和证明通常需要人工参与, 很多研究尚停留在理论阶段; ② 忽略了契约与实现间的关系, 很难灵活实现已有软件资产的复用. 总而言之, 现有这些主流方法的缺陷主要体现在以下几点.

(1) 未能充分考虑嵌入式系统的资源受限和实时性等关键特征.

(2) 在嵌入式环境及平台的要求和假设方面存在表述不清晰的问题.

(3) 未能充分重视对中间知识的表示和使用.

(4) 系统的正确性难以由组成它的模块的正确性保证.

(5) 忽略了模型与实现之间的关系.

此外, 嵌入式系统的实时和资源需求、与物理环境的深度融合也对以上这些方法构成挑战.

相对于软件领域的系统开发, 硬件领域提出硬件 IP (intellectual property) 的概念[11−13], 对芯片中常用的功能模

块进行封装, 使之具有高度的可复用性、可组合性和可验证性, 极大提升了芯片设计的可靠性和效率. 基于现有软件和硬件的设计思想, 文献 [14] 提出了软件 IP 的概念以及基于软件 IP 的嵌入式软件智能合成 (IP-based embedded software intelligent synthesis, IPESIS) 框架. 该框架主要包括以下 4 个部分.

(1) 设计一种基于软件 IP 的嵌入式软件需求描述语言, 该语言支持对软件需求、软件 IP 及系统行为的多维度抽象表达, 同时利于机器的理解和处理, 利用该语言可以编写软件需求形成软件需求规约.

(2) 构建软件 IP 的表示模型, 以对软件 IP 的功能和性能、输入输出接口等进行抽象和表达, 在此基础上, 从软件存量资产中提取软件 IP, 构建软件 IP 知识库, 为软件合成提供知识引擎.

(3) 程序合成, 以软件需求规约和软件 IP 知识库中的软件 IP 为输入, 通过智能化技术合成程序代码.

(4) 建立面向合成过程和合成产品的软件质量保障体系, 确保合成程序代码的正确性.

与传统的软件开发过程不同, IPESIS 关注任务需求的抽象和表达, 基于已有的软件资产自动构造程序, 因此开发者可以将更多精力集中在抽象层次更高的软件需求分析和设计上, 而不被具体的编程实现所困扰.

针对 IPESIS 框架的第 (2) 部分, 本文将定义嵌入式软件 IP 通用模型, 包括知识模型、形式模型和实现这 3 部分. 知识模型封装软件开发者的知识, 能够帮助开发者和使用者从各个方面理解软件 IP, 并能够辅助软件 IP 的搜索、适配和组装过程. 知识模型的定义体现了软件 IP 与传统模型表示方法的主要不同. 形式模型形式地定义了软件 IP 的接口规约以及对外部环境的依赖, 支持软件 IP 的验证和正确的组合, 能够为软件智能合成提供安全保障. 实现则包括软件 IP 的相关代码. 为了支持开发者不同的需求, 本文还提出基于不同关注点的软件 IP 视图, 包括基本视图、设计视图、验证视图、开发视图以及用户自定义视图, 分别对应不同粒度和不同需求的软件开发.

本文第 1 节介绍嵌入式软件建模、分析与开发的相关工作. 第 2 节介绍面向嵌入式系统的软件 IP 通用模型, 包括知识模型、形式模型和实现, 这 3 部分分别在第 3、4、5 节中进行详细描述. 第 6 节讨论软件 IP 知识模型、形式模型和实现这三者之间的一致性关系. 第 7 节介绍软件 IP 的各种视图. 第 8 节通过实际案例展示从嵌入式软件存量资产提取软件 IP 的方法及其有效性和可行性. 最后一节总结本文提出的方法.

1 嵌入式软件建模、分析与开发相关工作

1.1 模型驱动开发

UML[15]和 SysML[16]是广泛使用的图形建模语言, 它们均由 OMG 负责维护[8]. UML 提供不同种类的图对系统的功能、结构和行为进行建模, 主要应用于软件工程的需求分析和设计中. SysML 是 UML 的轻量级版本, 但是提供一些额外的支持系统工程的建模机制. MARTe[17]是由 OMG 发布的 UML 的面向实时和嵌入式系统的扩展. 然而, 它们均缺乏严格的形式语义, 并且不支持对系统行为的仿真与验证.

目前已有多个基于模型的嵌入式系统开发环境. Simulink[18]是由 MathWorks 研制的嵌入式系统图形建模、仿真与综合的集成开发环境. 它同时支持离散和连续数据流的刻画, 以模块作为基本单元, 通过图、子系统等组合方式建立层次化系统模型. AADL[4]是美国汽车工程师协会 SAE 于 2004 年提出的嵌入式实时系统体系结构的建模语言, 被广泛应用于航空航天、汽车控制等工业领域中. AADL 通过定义基本构件和复合构件, 能够同时描述系统的软硬件体系结构和功能行为. 然而, 它对构件的功能和行为描述能力不够, 需要相关领域的专用语言来辅助实现. Modelica[19]是一个适用于大规模物理系统和动态系统的建模与仿真环境. 它以面向对象为基础, 允许非因果关系的数据流刻画, 适用于不同的运行上下文, 因此更强调系统的复用性. 近年来 Modelica 提出 FMI 标准[20,21], 致力于将不同来源的计算模型进行组合并利用各组件所拥有的仿真工具进行协同仿真. Ptolemy[22]以角色作为系统的基本构造单元, 通过端口及层次结构来构造复杂的嵌入式系统, 支持包括顺序、并发、实时等不同类型的计算模型. 以上两个框架能够充分描述系统不同部件之间的组合和交互, 然而它们均不支持嵌入式系统体系结构以及连续动态行为的建模. Metropolis[23]是一个基于特定平台的嵌入式系统建模与仿真环境, 它所提供的元模型语言能够同时描述计算行为、软硬件体系结构以及功能与体系之间的映射, 并且支持与第三方工具和算法的集成. 文献 [24] 提出了一个嵌入式系统模型语言框架 ESMoL, 支持系统控制行为、硬件平台以及软件行为多个侧面的集成, 通过描述它

们之间的交互关系, 将混合模型转换到一个统一的中间语言进行分析与代码生成. 以上基于模型的嵌入式系统开发环境对系统行为进行建模和仿真分析, 然而它们存在的共同问题是缺乏严格的形式语义, 难以支持形式验证.

支持形式验证的模型设计方法已有一些研究. SCADE[3]是一个高安全性的嵌入式软件开发环境, 集成了嵌入式系统建模、仿真、形式验证, 代码生成等功能. 它的核心语言是同步数据流语言 LUSTRE[25], 具有严格的语义. 然而, 它对系统构件和封装的支持较弱. Event-B[26]是一个基于事件的系统建模和分析方法, 它支持系统各抽象层次的建模并通过逐步精化和严格的数学证明来保证不同层次行为的一致性. Event-B 被扩展到混成系统[27,28], 能够处理连续物理环境和离散控制系统的建模和分析. 欧洲 COMPASS 项目致力于建立成体系系统 (systems of systems) 的集成建模框架, 提出的建模语言 CML[29]集成了 Z 方法[30]、CSP[31]和 VDM[32], 定义了基于程序统一理论 UTP[33]的语义, 并且支持与 SysML[34]的无缝集成来描述软硬件体系结构. 欧洲 CESAR 项目提出的 Polychrony[35]是一个异构的嵌入式与实时系统开发框架, 它的核心语言是 Signal[36], 能够描述硬件特性, 支持多时钟以及同步/异步通信, 通过精化技术逐步对系统加以实现. CML 和 Polychrony 主要处理离散系统. Zélus[37−39]以同步数据流语言 LUSTRE 为基础进行扩展, 支持连续行为的刻画, 实现了一个较强的类型系统, 能够静态地检查出部分错误行为. 上面所提到的基于模型的形式框架基本停留在软件模型的形式化层次上, 对于可复用软件构件的支持 (包括界面封装、组合等) 没有进行深入的研究.

1.2 基于构件的开发

基于构件的设计方法 (CBD) 的基本特点是抽象掉软件模块实现细节, 提供其接口及规约, 封装成可复用的构件, 并提供构件组合操作, 从而有效构造复杂系统并提高系统开发效率. CBD 方法支持构件的替换, 保证系统正确性. 工业界主流的 CBD 技术包括 J2EE/EJB[7]、CORBA/CCM[8]和 COM/DCOM[9]等. J2EE/EJB 采用基于软件构件模型的分布对象计算体系, 可以显著简化复杂的企业级开发. 所有开发者必须遵循构件的开放规范, 从而实现构件的兼容性和可移植性等. CORBA 是一种跨越网络、操作系统和机器实现分布对象之间互操作的工业标准, CORBA/CCM 是用于开发和部署 CORBA 应用程序的服务器端构件规范. COM 是微软提出的基于构件的开发技术, 在此基础上继续推出了 DCOM, 使基于构件的网络应用的开发成为可能. 此外, 针对航天飞行器软件, NASA 的 JPL 喷气推进实验室提出了 F Prime 开源框架[40], 该框架将构件分为 3 类: 主动构件、被动构件和队列构件, 这些构件的对外接口分为指令、时间、事件、参数、遥测等[41]. F prime 提供了一种面向航天飞行软件体系结构方法, 该方法可以生成代码框架[42], 加快飞行控制程序的开发过程. 但是以上这些技术对构件没有进行形式化描述.

学术界已有一些基于构件的形式设计框架. Reo 是一种抽象行为类型的构件形式模型[43], 其中构件行为被定义为在构件端口上输入数据流和输出数据流之间的关系. BIP 是实时系统的构件模型[44], 通过行为层、交互层和优先级层来描述构件的行为和它们之间的交互关系. 最近, BIP 的扩展能够描述动态构件和带参构件的行为[45−48]. rCOS 是一个构件和对象系统的统一语义模型[49−53], 其将构件定义为 4 个部分: 接口、契约、实现和发行, 并定义了一组构件组合操作以及构件间的黏合代码, 从而可以将若干已知构件黏合成一个复杂构件. 然而, 以上的构件框架主要关注功能性, 缺少对于实时、资源、连续行为等方面的支持. 这方面的工作目前比较少并且不够深入. 文献 [54,55] 将 Simulink 转换到微分动态逻辑然后进行分析验证, 然而该方法没有考虑转换的正确性. CyPhyML 是一种基于构件的信息物理融合系统的组合和交互语言, 使用 FORMULA 约束逻辑来描述包括连续行为在内的构件行为以及不同构件间的约束关系和一致性[56]. 文献 [57] 实现了一个信息物理融合系统的模型和工具集成平台 OpenMETA, 结合了模型驱动的设计和基于构件的设计, 允许将不同的构件组装在一起, 基于 FORMULA 定义不同部件之间的语义一致性, 并且在工具集成阶段, 支持包括混成系统在内的多种模型和对应的验证算法.

1.3 基于契约的设计

基于契约的设计 (design by contract) 方法能够独立地描述各个系统构件的责任以及对构件所在环境的假设, 检查构件之间组装时是否相容、一致, 从而保证组装后系统的正确性, 并支持构件替换. 基于契约的设计思想最早由 Meyer 在软件工程领域提出[58,59], 这个思想起源于 Floyd-Hoare 逻辑[60]. 给定程序 M, 它的 Hoare 三元组表示为 {p}M{q}, 其中 p 表示输入满足的性质, q 表示 M 执行后输出满足的性质. 从而, 将 (p, q) 定义为 M 的契约. 契约的

一个重要概念是精化: 对于任意的程序, 如果它是契约 A 的实现蕴含着它也是契约 B 的实现, 那么契约 A 是契约 B 的精化. Meyer 又将基于契约的设计推广到面向对象领域. 如 Eiffel 语言[59]定义了类的契约.

基于契约的设计后来被推广到反应式系统和信息物理融合系统. 反应式系统 (reactive system) 是指需要连续地与环境交互的系统, 而信息物理融合系统是集连续演化、离散控制、通信、并行等一体的混合系统. Abadi 等人[61,62]首次提出了反应式系统的假设/保证 (assume/guarantee) 规约, 基于博弈论来定义组件和环境的行为, 并定义规约之间的精化、规约的并行组合和相容性. 假设 (assumption) 表示对环境行为的假设, 保证 (guarantee) 表示在这个环境假设下, 系统本身能够保证的行为, 它们都定义为性质. 最典型的性质表示形式是执行迹的集合, 每个迹表示状态序列或事件序列. de Alfaro 等人[63]提出接口自动机 (interface automata), 并利用接口自动机描述系统构件接口与环境之间的契约关系. 接口模型被扩展以处理更加复杂的行为, 例如实时、资源等问题[64,65]. 接口输入输出自动机[66]与接口自动机不同, 它将环境的假设和系统的保证分别定义为两个不同的自动机.

除了安全性, 系统还有对于实时、资源约束等方面的需求. 文献 [67] 提出了多视点模型的数学基础. 针对实时性质, 文献 [68,69] 提出了实时接口, 定义了实时接口的相容性、精化, 以及并行组合、合取、商等组合方式, 在此基础上实现了工具 ECDAR[70]. 模态规约的实时扩展在文献 [71] 中给出. 文献 [72] 扩展了实时接口契约, 能够描述周期性、任务完成时间以及与任务调度有关的假设/保证性质, 一些相关的工作参见文献 [73]. 文献 [74] 介绍了实时调度规约, 接口可以定义时间、资源、调度策略等多方面的需求, 实现了分析工具 CARTS[75], 并应用在 ARINC 分区调度中[76]. 另外, 还有一些专门针对资源约束的接口契约方面的工作[77]. 假设/保证契约设计还被推广到概率和随机系统中, 能够处理定量概率性质的描述和验证[78−80]. 基于契约的设计方法需要从已有软件资产中提取契约, 并在契约上进行复杂的操作和推理, 在理论上仍是一个挑战, 因而无法在实践中应用到大规模嵌入式软件.

另一方面, 在电子电路设计领域, 硬件 IP 对芯片中常用的功能模块进行封装, 具有高度的可复用性、可组合性和可验证性, 提升了芯片设计的效率. 为方便设计者使用, 硬件 IP 的生产者需要提供详细的使用文档. 为了促进 IP 产业的发展, SPIRIT Consortium 出台了 IP-XACT 标准[11]. IP-XACT 使用 XML 格式的语言来定义和描述 IP 的规约以及 IP 之间相互连接的细节, 给出了通用的设计表示, 可以在 IP 设计的不同流程内部进行交换. 硬件 IP 的设计理念与成功经验为嵌入式软件描述与建模提供了参考. 此外, 知识工程领域提出了知件的概念. 知件是独立的、计算机可操作的、商品化的、有完备文档的、可被某一类软件调用的知识模块[81]. 知件工程的愿景是形成与软件和硬件并列的第 3 大产业[82]. 与知件不同, 本文提出的软件 IP 是一个软件知识实体, 其本质是一种软件模块, 支持嵌入式软件的智能合成.

2 嵌入式软件 IP 通用模型设计思路

本文的主要贡献是提出面向嵌入式系统的软件 IP 通用模型, 在介绍具体的模型之前, 本节就该模型的设计思路予以介绍.

2.1 软件功能模块通用模型

一个软件功能模块都可以抽象成一个输入输出模型, 如图 1 所示. 功能模块的一次执行包含顺序的 3 个步骤: 输入、计算、输出. 从输入端获取计算需要的资源, 然后执行计算, 最后将计算结果从输出端输出. 对模型的输入和输出进行不同的解释, 会得到不同的软件模型.

如果将图 1 中的输入端解释为请求接口, 输出端解释为提供接口, 那么会得到软件构件模型, 如图 2 所示. 软件构件是软件系统中具有相对独立功能、可以明确辨识、接口由契约指定、和语境有明显依赖关系、可独立部署的可组装软件实体[83]. 构件的请求接口描述了为实现构件的功能所需要的外部服务, 提供接口以 API 的形式向外部环境提供构件的至少一个功能 (服务).

如果将图 1 中的输入和输出解释为数据和事件端口, 那么会得到数据流/事件流模型, 如图 3 所示. 模型驱动开发 (MDD) 通常采用这种数据流/事件流模型. MDD 方法一般分为 3 个阶段: 首先是建模, 根据系统设计需求从模型库中选择合适的数据流/事件流模块完成组装; 其次是分析, 通过仿真、测试、形式验证等方法确认组装的模型是否满足需求; 最后是代码生成, 利用模型转换技术, 将抽象的模型逐步精化为可执行代码.

图 1 软件功能模型通用模型

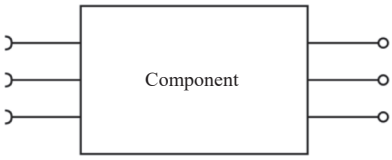

图 2 软件构件模型

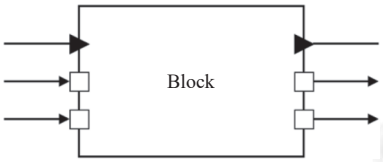

图 3 数据/事件流模型

2.2 嵌入式软件 IP 通用模型

与一般的软件功能模块不同, 软件 IP 是具有知识产权的可复用的软件知识实体的总称, 是具有特定功能的、上下文依赖关系及接口定义明确、设计文档规范完整, 经过严格质量验证的软件[14]. 然而软件 IP 本质上也是一种软件功能模块, 因此软件 IP 模型也遵照图 1 的模型, 如图 4 所示, 其中输入和输出端口构成了软件 IP 的接口.

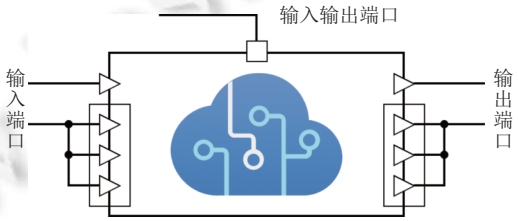

图 4 嵌入式软件 IP 通用模型

接口部分描述了软件 IP 与外部环境的交互, 每个软件 IP 只对外开放一个接口. 接口描述了软件 IP 对外可见的部分, 不在接口中的内容对外不可见. 软件 IP 的功能是通过一个函数来实现的, 但是实现细节对外不可见. 外部环境通过调用 IP 的名称, 实现对这个 IP 功能的调用.

软件 IP 是面向嵌入式实时系统的, 在嵌入式实时系统中资源是至关重要的, 因此在软件 IP 模型中, 接口中的输入端口、输出端口以及输入输出端口是对资源需求的一种表示. 关于这些端口的类型、存储位置以及占用空间的大小等信息必须在软件 IP 中有明确的说明.

图 4 中的软件 IP 模型是一种将控制流从数据流分离的软件模型: 如果一个软件 IP 被外部调用, 那么它将执行它的功能, 即从输入端口 (包括输入输出端口) 读取数据, 进行计算之后从输出端口 (包括输入输出端口) 输送出去. 实际上软件 IP 的这种工作模式类似于“管道-过滤器”交互式模型[84], 如图 5 所示. 所谓“管道-过滤器”交互式模型是指功能模块可以视为处理数据的“过滤器”, 模块之间的连接称作“管道”, 用于在模块之间传递数据; 控制流的作用是将某个模块激活, 使其能够响应特定的信号然后执行相应的功能. 文献 [84] 认为, “管道-过滤器”模型在嵌入式与实时系统领域广泛使用的原因是控制理论可以很容易地映射到这种交互式模型.

一般的软件构件模型采用“请求-响应”模式, 如图 6 所示. 一个构件在请求另一个构件的服务之后, 被请求方可能会提供给请求方返回值. 这种控制流和数据流不分的模式符合一般软件系统的开发习惯, 因此 CBD 方法更适用于一般软件系统的开发.

文献 [84] 对比了“管道-过滤器”和“请求-响应”这两种交互式模式, 并且认为一般软件构件模型会使用“请求-响应”模式, 但是在一些特殊领域 (尤其是嵌入式系统), “管道-过滤器”模式是占据主导地位的. 软件 IP 是面向嵌

入式系统的, 因此软件 IP 模型的设计采用“管道-过滤器”风格, 而不是一般的软件构件模型采用的“请求-响应”风格.

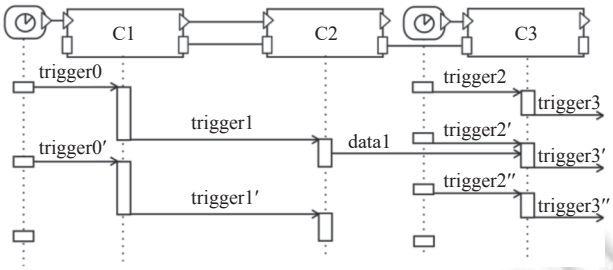

图 5 “管道-过滤器”交互式模型

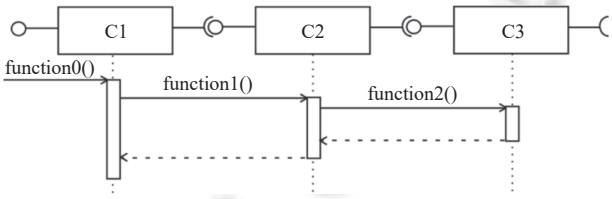

图 6 “请求-响应”交互式模型

另外, 图 4 中的软件 IP 模型是功能独立的, 即外部环境通过调用软件 IP 的名称来访问 IP 的功能, IP 的执行只依赖于接口里的输入, 而不会依赖于外部的其他功能. 相比于软件 IP 模型, 软件构件模型不是功能独立的, 它需要通过请求接口请求外部的服务, 然后才能通过提供接口向外提供服务. 从知识产权的角度来说, 用户购买的软件功能模块必须是功能独立的, 否则用户需要购买更多的模块来支持之前购买的模块的功能, 这是不合理的. 因此, 这也是本文将软件 IP 模型设计成图 4 的形式而不是图 2 构件形式的一个重要原因.

2.3 知识模型、形式模型和实现

一个良构的软件 IP 应该能够全面、正确地反映软件 IP 在创建和使用过程中的关键特性, 它是开发者关于软件知识的抽象和汇集, 构成了软件智能合成理论与方法的核心概念. 总的来说, 软件 IP 的表示模型可以定义为如下三元组:

$$\text{SoftwareIP} = (\text{KM}, \text{FM}, \text{IMP}),$$

其中, KM 表示知识模型, FM 表示形式模型, IMP 表示实现, 如图 7 所示.

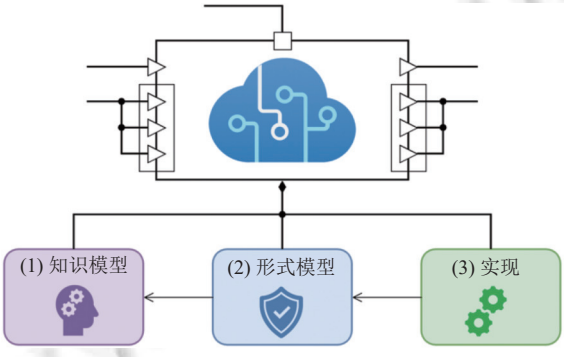

图 7 嵌入式软件 IP 通用模型

知识模型是软件 IP 中知识的结构化表示, 它是软件 IP 中抽象程度最高、内容最丰富的部分. 知识模型基本采用非形式的自然语言和图形模型等方式描述; 形式模型采用形式语言进行描述; 实现部分主要包括程序代码. 因

此, 从抽象/精化的角度看, 虽然知识模型最抽象, 但是可以覆盖软件 IP 的整个生命周期, 所以其内容最丰富. 知识模型首先是帮助开发者和使用者从各个方面理解软件 IP, 其次能够辅助软件 IP 的搜索、适配和组装. 形式模型是知识模型的一种形式化表示. 相对于知识模型, 形式模型能够在更深层次上帮助理解软件 IP, 它能最大程度地减少软件 IP 的创建者和使用者对软件 IP 的认知差距, 能够为软件智能合成提供安全保障. 最后, 实现部分中的程序代码需要和形式模型中的描述保持一致.

3 软件 IP 的知识模型

现有的方法 (CBD、MDD 和 DbC 等) 缺乏对嵌入式软件设计与开发过程中产生的中间知识的表示, 因此难以支持二次开发. 在合成软件时, 需要去搜索一些功能模块完成组装, 如果没有这些中间知识, 那么很难进行功能模块的搜索和适配. 软件 IP 可以通过知识模型来表达这些与设计和开发相关的知识. 与知件[81,82]的关注点不同, 软件 IP 知识模型的关注点是支持软件 IP 的自动搜索与适配, 为基于软件 IP 的智能合成奠定知识基础.

3.1 软件 IP 的知识及使用模式

软件 IP 是开发者知识的一种体现, 开发者复用软件的能力直接来源于拥有的关于软件资产及其属性的知识, 因此研究软件 IP 的知识对使用和理解软件 IP 起到了关键的作用. 具体来说, 软件 IP 知识的使用模式如图 8 所示. 首先, 软件 IP 创建者感知应用领域的环境, 从知识库中获取领域知识, 然后推理产生结论 (软件 IP 的分析与设计), 执行相应的动作 (软件 IP 的创建与执行), 最后验证期望结果和实际结果之间的差异, 判断软件 IP 的执行效果, 这一过程产生的新的软件 IP 将被存入知识库中; 另一方面, 软件 IP 使用者感知使用环境以及用户需求, 从已有的知识库中获取知识, 在此基础上完成软件 IP 的选取、适配与演化, 最终合成新的软件 IP, 这个过程也会产生新的知识, 进一步丰富软件 IP 知识库的内容.

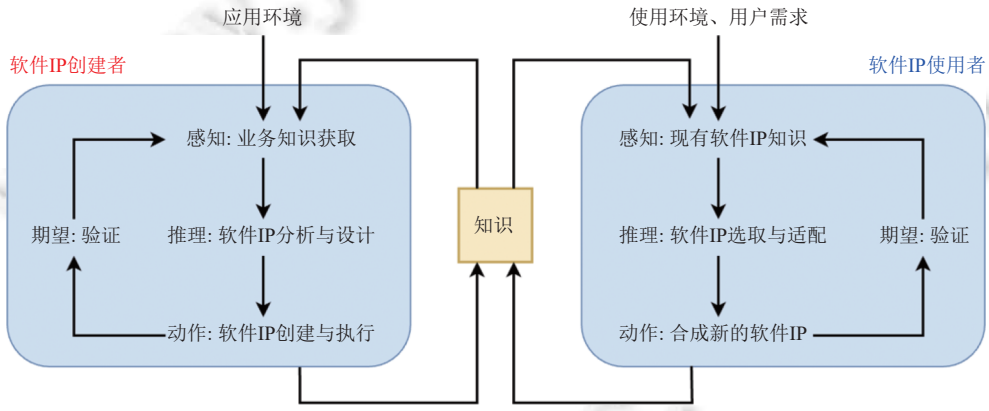

图 8 软件 IP 知识的使用模式

3.2 软件 IP 知识模型的结构

软件 IP 作为一种可独立部署、可组装的知识实体, 其中最重要的就是它的知识模型. 知识模型是具有指导意义的结构化信息, 可以定义为如下三元组:

$$KL = (Basic, Expert, SoftDev),$$

其中, Basic 表示基本知识, Expert 是领域专家知识, SoftDev 表示软件开发过程知识, 如后文图 9 所示.

3.2.1 基本知识

软件 IP 的基本知识是对软件 IP 的一个总体概述, 表示该 IP 本身的一些属性信息, 具体包括 IP 名称、ID 编号、版本、版权归属、作者以及开发的时间、关键词、所属应用领域和类别等.

(1) IP 名称是一个名词性短语, 用中文描述同时给出英文缩写, 并且要求 IP 名称能够直接反映该 IP 要解决的

问题, 例如“双精度浮点数有效性判断 (FloatValid64)”.

(2) ID 编号是由英文字母 (包含大小写)、符号 (点、斜线、短横、下划线等) 和数字的字符串组成, 每个软件 IP 对应唯一的 ID, 并且可以从 ID 中解析出该 IP 的名称、版本、所属应用领域和类别等信息.

(3) 版本也是由英文字母 (例如 beta)、符号和数字组成的字符串, 并且遵循领域的命名规范.

(4) 版权归属是一个中文字符串, 指明版权归属单位.

(5) 作者指软件 IP 的创建者, 可以是多个人, 也可以是一个团队或者部门.

(6) 开发时间采用中国标准时间进行描述, 支持多种日期表示格式.

(7) 关键词是能够体现软件 IP 特征的词语, 并且要求关键词必须出自领域术语库 (术语词典).

(8) 所属应用领域描述该软件 IP 适合用于哪些应用领域, 应用领域的描述同样需要依据领域术语库.

(9) 类别表明该软件 IP 的分类信息, 遵循领域的分类标准.

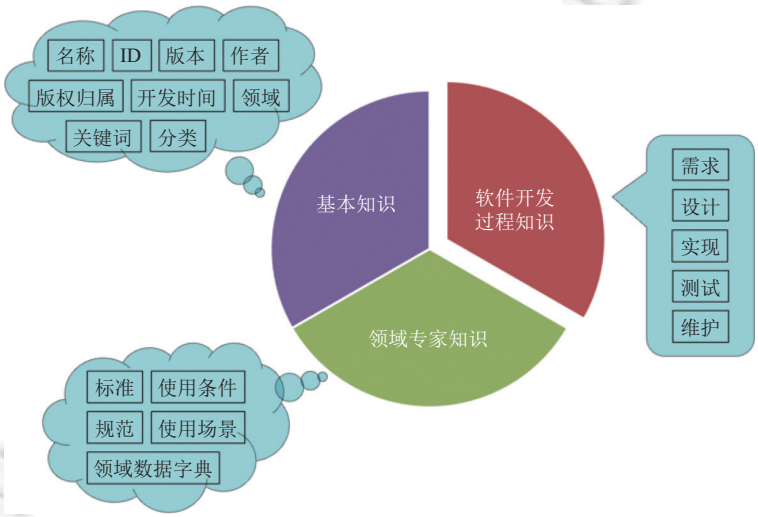

图 9 软件 IP 的知识模型

3.2.2 领域专家知识

软件 IP 的领域专家知识包括该 IP 所遵循的开发标准和规范、IP 的使用条件 (例如偏心率为 10^{-2} 量级) 和典型使用场景 (例如低轨、高轨)、领域数据字典等. 软件 IP 遵循的开发标准和规范通常以文档的形式存在, 因此在知识模型中只需要说明这些文档的名称, 例如《航天器软件编程规范 (C 语言部分)》.

3.2.3 软件开发过程知识

该部分知识覆盖了软件 IP 开发的整个生命周期, 包括需求、设计、实现、测试和维护, 如后文图 10 所示.

3.2.3.1 需 求

需求部分包括软件 IP 的功能描述, 体现软件 IP 特征, 该部分可以采用自然语言或者半结构化自然语言 (伪自然语言) 描述. 需求的描述要尽量简洁, 而且必须基于领域术语库 (术语词典), 即不能使用自创的或者语义不明确的名词. 从需求规格说明文档中提取描述需求的半结构化自然语言描述相对容易, 然而这种方式难以确认提取的需求描述的一致性和完备性. 因此, 可以考虑使用 UML、AADL、Simulink/Stateflow 等为软件 IP 的需求描述建立图形模型, 继而可以采用仿真等方式对需求的一致性和完备性进行确认. 需求的图形模型可以从文档中直接提取 (如果有), 或者根据需求的自然语言描述自动生成相应的图形模型. 然而后者的实现难度较大, 因为涉及自然语言理解等比较困难的工作.

3.2.3.2 设 计

设计部分包括接口、参数、流程图和异常处理这 4 部分.

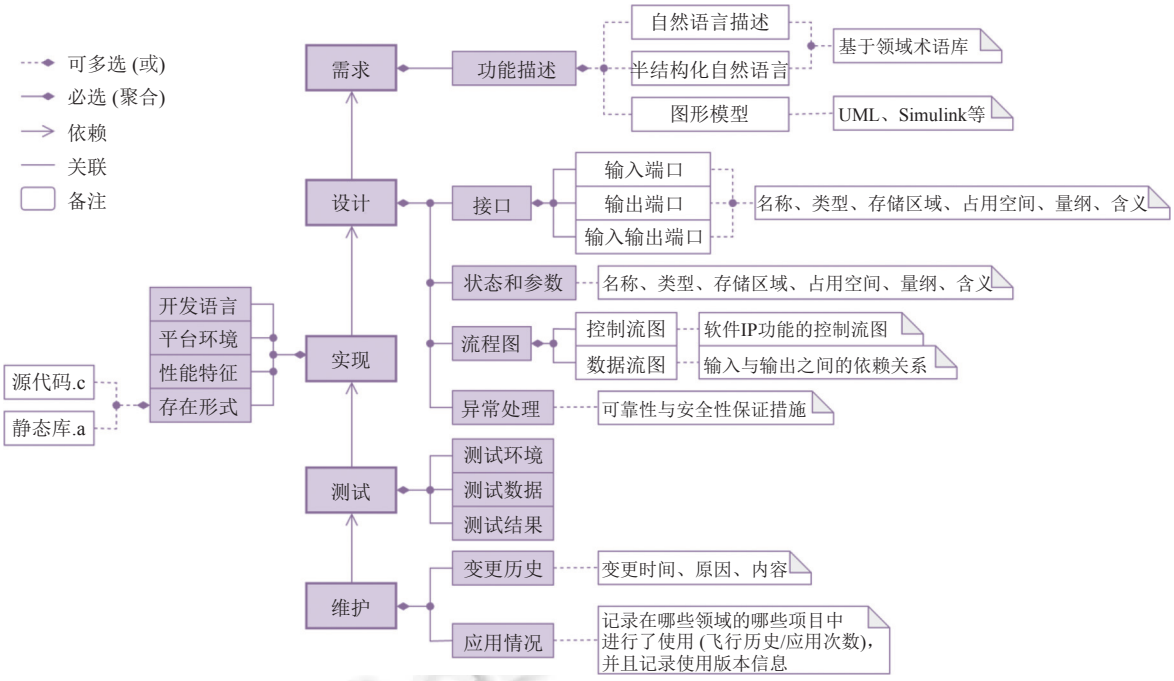

图 10 软件开发过程知识

(1) 接口中包含了软件 IP 的输入端口、输出端口和输入输出端口. 这些端口是软件 IP 对资源需求的一种表示, 因此对于每一个端口, 需要明确它的名称、类型、存储位置、存储空间、量纲和含义等.

(2) 状态和参数部分定义软件 IP 的状态变量和参数变量.

(3) 流程图包括控制流图和数据流图. 控制流图展示了软件 IP 功能的控制流; 数据流图展示了软件 IP 输入与输出端口之间数据依赖关系, 即每一个输出端口依赖于哪些输入端口.

(4) 异常处理包括采用的可靠性与安全性的保证措施, 用于预防可能出现的故障以及明确故障出现后可采取的应对措施.

3.2.3.3 实 现

实现部分包括开发语言、环境配置、性能特征和存在形式.

(1) 开发语言部分声明了软件 IP 的代码是由哪种编程语言编写.

(2) 平台环境包括软件 IP 的运行对处理器、编译器和操作系统等的要求.

(3) 性能特征包括时间、空间、精度 (数据类型)、大小端、圈复杂度等.

(4) 存在形式包括源代码 (.c 文件) 或者静态库 (.a 文件) 等.

3.2.3.4 测 试

测试部分包括测试环境、测试数据和测试结果.

(1) 测试环境包括测试运行其上的软件和硬件环境的描述.

(2) 测试数据中列出一些比较有代表性的测试数据, 这些测试数据构成的输入-输出对支持软件 IP 的搜索. 在一些情况下, 用户可能无法清晰地描述需要使用的 IP, 但是知道给定这个 IP 一个输入会得到特定的输出, 那么用户可以根据这些输入-输出对在知识库中搜索满足需求的 IP.

(3) 测试结果中包括测试类型 (例如单元测试或者集成测试)、是否达到测试目标以及针对某项指标 (例如分支覆盖) 的覆盖率等, 根据测试结果可以对软件 IP 进行可信等级评估 (certification).

3.2.3.5 维 护

维护部分包括变更历史和应用情况.

(1) 变更历史包括变更日期、变更内容和变更原因.

(2) 应用情况包含软件 IP 的运行记录, 例如该 IP 在哪些领域的哪些项目中进行了使用.

4 软件 IP 的形式模型

形式化方法是全面系统地使用基于数学的语言、技术和工具精确地说明、开发和验证软件系统, 使用形式化方法描述的规约具有规范性和无二义性. 采用形式化方法描述软件 IP 可以最大程度地减少软件 IP 创建者和使用者对软件 IP 的认知差距, 有利于正确地理解软件 IP 的含义, 为基于软件 IP 的软件智能合成提供安全性保障. 一般地, 给定一个程序 M, 根据 Floyd-Hoare 逻辑, 它的形式模型可以表示为一个 Hoare 三元组: {p}M{q}, 其中 p 表示 M 输入满足的性质 (前置条件), q 表示 M 执行之后输出满足的性质 (后置条件). 前后断言 (p, q) 定义了程序 M 的形式规约. 另外, 不变式也是描述程序满足的性质的重要手段. 程序不变式与程序的前后断言 (p, q) 不同: (p, q) 描述程序输入和输出的性质, 它不关注程序执行过程; 而不变式则描述了程序在运行过程中, 程序的状态始终满足的约束. 通过不变式可以证明程序满足的性质 (例如可终止性等). 对于嵌入式软件, 除了它的功能行为, 我们通常也关注它的非功能行为 (例如性能、资源要求、安全性、健壮性等) 的形式描述, 即非功能约束. 综上所述, 对于软件 IP 形式模型, 可以定义以下三元组:

$$FM = (Spec, NonFun, Inv),$$

其中, Spec 表示形式规约, NonFun 表示非功能约束, Inv 表示 IP 不变式.

4.1 形式规约

以上提到的前后断言 (p, q) 定义了程序的形式规约, 又称为程序的契约. 实际上, 这种基于契约的设计 (DbC) 思想正是来源于 Floyd-Hoare 逻辑. 基于契约的设计方法能够独立地描述每个 IP 的责任以及对 IP 所需环境的假设, 检查 IP 之间组装时是否相容、一致, 从而保证组装后系统的正确性, 即系统的正确性可以由组成它的 IP 的正确性来保证. 契约描述了一个 IP 的输入与输出之间的逻辑公式. 可以考虑采用统一程序理论 (unifying theories of programming, UTP)[33]作为软件 IP 契约 (形式规约) 的表示方法. UTP 的思想是来自于物理学中的大统一理论, 它的愿景是统一所有风格的编程范式. 在 UTP 中, 变量 x 本身表示该变量的初始值, x'表示该变量的终止值, 那么程序的行为可以表示成所有 x 和 x'之间的逻辑公式 (谓词关系). 例如, 以下是用 UTP 书写的程序形式规约.

Variable declaration:

● Inputs: u, y: **R**

● Outputs: x, z: **R**

Assumptions: $y \neq 0$

Guarantees: $x' > u \wedge z' = x' / y$

该形式规约是一个假设/保证 (assume/guarantee)-契约, 它描述了程序的输入是 u 和 y, 输出是 x 和 z, 它们都是实数类型. 如果输入 $y \neq 0$ (assume), 那么该软件 IP 将保证输出 $x' > u$ 并且 $z' = x' / y$ (guarantee); 反之, 如果输入 $y = 0$ (违反假设), 那么该软件 IP 将不会有任何保证, 即输出 x 和 z 可以是任意值.

4.2 非功能约束

软件 IP 的非功能约束主要包括运行环境约束、资源约束和性能约束等. 运行环境约束包括对处理器的要求 (处理器版本与主频)、编译器的要求 (编译器版本与编译选项), 以及操作系统的要求 (操作系统版本); 资源约束包括 IP 占用的内存空间约束; 性能约束包括 IP 在运行时的响应时间和一次执行时间约束. 这些约束可以通过一阶逻辑公式表示.

4.3 IP 不变式

一个软件 IP 可能包含状态变量 (详见第 5 节), IP 不变式描述了软件 IP 在动态运行过程中状态变量值始终满

足的约束. 因此, IP 不变式类似于面向对象编程中类不变式的概念.

5 软件 IP 的实现

软件 IP 的实现包含两部分: 头文件 (.h) 和源代码文件 (.c). 源代码文件实现了软件 IP 的功能, 而头文件主要包含了源代码中使用的变量的声明. 这些声明的变量包括如下.

(1) 输入变量: 对应软件 IP 的输入端口, 软件 IP 从输入变量 (端口) 读取数据.

(2) 输出变量: 对应软件 IP 的输出端口, 软件 IP 的计算结果通过输出变量 (端口) 传送出去.

(3) 状态变量: 用于记录在软件 IP 后续执行中会被再次用到的值. 例如, 累加器的软件 IP 中会使用状态变量来存储上一次的累加结果.

(4) 参数变量: 可以被视为软件 IP 的“出厂配置”. 例如, 上面累加器软件 IP 的参数变量可以是累加值的上限和下限值. 与以上状态变量不同, 参数变量在软件 IP 的执行过程中应保持不变, 因为更改“出厂配置”通常是不合理的.

除了声明变量之外, 头文件中还会声明软件 IP 的句柄程序. 该句柄程序相当于该软件 IP 的“主函数”, 其具体实现包含在源代码文件 (.c) 中. 通过这种方式, 环境可以通过调用该软件 IP 句柄函数来使用其功能, 而无需暴露其实现细节. 具体而言, 环境可以通过包含 (#include) 该软件 IP 的头文件来使用它. 关于软件 IP 实现的具体形式及使用方法, 可参见案例文档: https://github.com/BearHeroWithErrow/poweronjudge-ip-instance/blob/main/PowerOnJudge 的软件 IP 实例.pdf.

6 软件 IP 的一致性

软件 IP 由知识模型 (第 3 节)、形式模型 (第 4 节) 和实现 (第 5 节) 这 3 部分构成, 因此这 3 部分之间应保持一致性关系. 知识模型通常采用自然语言文本、图表等非形式的方式描述; 形式模型以形式化的方式描述软件 IP 的功能和非功能行为; 实现部分通过代码的方式实现了软件 IP 的功能. 软件 IP 三要素之间的相互转换如图 11 所示.

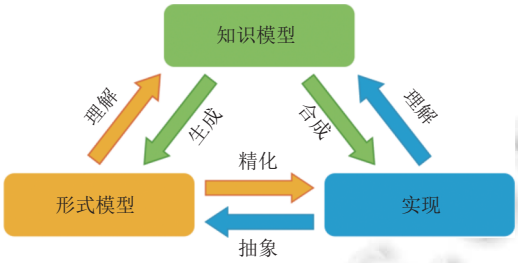

图 11 软件 IP 三要素之间的转换关系

6.1 知识模型与形式模型

知识模型包括软件 IP 功能的自然语言和/或图形描述 (如 Simulink/Stateflow 和 UML 图等), 据此可生成形式描述作为该软件 IP 的形式模型. 例如, 已有相关研究[85]将 (非形式的) AADL 和 Simulink/Stateflow 模型转换为形式 HCSP 模型[86], 之后可使用相应的工具[87]对其进行形式验证. 近年随着大型语言模型 (large language model, LLM) 的普及, 从自然语言描述生成形式描述的研究成果也不断涌现. 例如, 工具 nl2spec[88]可交互式地将用英文描述的需求转换为 LTL 公式. 在从知识模型生成形式模型之后, 我们可以借助某些工具检查原始形式模型与生成的形式模型之间的逻辑等价性, 以此验证一致性.

另一方面, 可以将软件 IP 的形式模型输入到训练好的 LLM 中, 然后得到基于文本的描述作为输出, 即对该形式模型的自然语言理解. 可以通过检查知识模型中的原始文本与从形式模型中生成的自然语言理解之间的语义等

价性, 来确认知识模型与形式模型之间的一致性. 然而, 就目前而言, 形式模型的理解以及检查文本之间的语义等价性在自然语言理解领域都是巨大的挑战.

6.2 知识模型与实现

软件 IP 的实现 (代码) 可从其知识模型中合成, 之后可以使用一些测试和形式化技术[89]来检查原始代码与合成代码之间的功能等价性. 近年来, 基于人工智能技术[90−92]从自然语言描述合成代码的方法已得到长足进步, 然而通过人工智能生成的代码的正确性和安全性难以得到保证[93,94]. 尤其是涉及安全攸关嵌入式软件时, 如何合成满足安全要求的正确代码仍然是一个开放的问题. 此外, 可以借助 LLM 将软件 IP 实现部分的代码生成自然语言描述, 即对代码的理解. 与形式模型的理解 (第 6.1 节) 类似, 将代码转换为正确的文本描述也是自然语言理解领域的一大挑战.

6.3 形式模型与实现

基于模型驱动开发 (MDD) 的思想, 软件 IP 的形式模型可以逐步精化为可执行代码 (实现). 与基于人工智能技术的代码生成不同, 通过精化生成的代码更为可靠, 尽管生成的代码可能存在效率较低、冗长等问题. 例如, 最近有一些研究工作将经过验证的 HCSP 模型转换为 SystemC[95]和 C[96]代码, 并保证了形式模型与生成代码之间的一致性.

另一方面, 软件 IP 实现部分的代码也可以抽象为形式模型. 目前存在一些工具可以自动或半自动地从代码执行轨迹中推断出契约、断言和不变式, 例如 Daikon[97]、EvoSpex[98]等. 近年也出现一些基于 LLM 生成不变式和程序形式规约的研究工作[99]. 此外, 还有许多其他从代码生成规范的方法, 例如在 K 语义框架[100]中合成规约, 将代码注释转换为形式规约[101]等.

7 软件 IP 的视图

以上介绍的知识模型、形式模型和实现构成了软件 IP 的主体部分, 此外, 一个软件 IP 实体还包括一些说明性文档. 具体来说, 软件 IP 的目录结构如后文图 12 所示, 主要由以下部分构成.

(1) 数据单 (Datasheet): 以文档的形式呈现, 是软件 IP 的总览, 包含软件 IP 的功能、性能、接口描述、可靠性和安全性约束、运行环境、使用范例及仿真测试数据等信息.

(2) 知识模型: 以 XML 文档的形式对软件 IP 的知识进行表示, 同时包含知识模型中涉及的一些图形模型及其可解析文档.

(3) 形式模型: 包括软件 IP 的形式规约、非功能约束和 IP 不变式的可解析文档.

(4) 实现: 包含接口的.h 文件和实现的源代码.c 文件 (或者编译后的静态链接库.a 文件).

(5) 制品: 包括如下.

- 说明目录: 对应管理视图下的说明部分, 主要包含一些说明性的文档以供查阅.
- 测试目录: 对应管理视图下的测试部分, 包含测试报告和测试用例.
- 仿真目录: 对应管理视图下的仿真部分, 包含仿真报告和仿真数据.
- 维护目录: 对应管理视图下的维护部分, 包含维护记录.

然而, 软件 IP 的复杂性将会给基于软件 IP 的智能合成带来困难. 因此, 为了简化软件 IP 的表示, 本节将从软件 IP 的不同观察视角出发, 定义软件 IP 的层次化视图, 包括基本视图、设计视图、验证视图、开发视图、管理视图和用户自定义视图, 如后文图 13 所示. 此外, 后文图 12 的目录结构可以看作是软件 IP 的全视图.

7.1 基本视图

软件 IP 的基本视图相对于软件 IP 的其他视图处于最抽象的层次, 它代表领域内的一个概念 (concept). 例如, 工程师在设计一个软件系统的过程中会产生一些概念, 而系统的雏形就是由这些概念组合而成[102]. 实际上, 这些概念就是所谓的知识, 因此软件 IP 的基本视图可以看作是软件 IP 的最小知识模型.

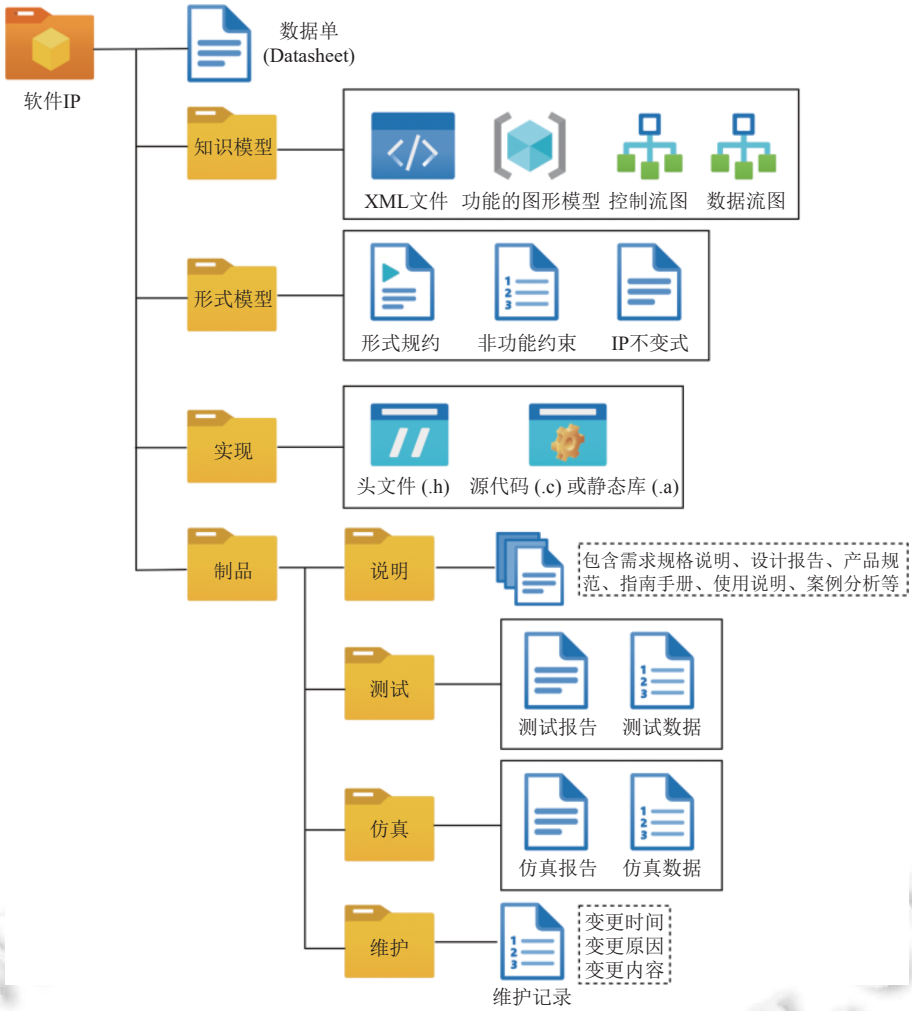

图 12 软件 IP 的目录结构

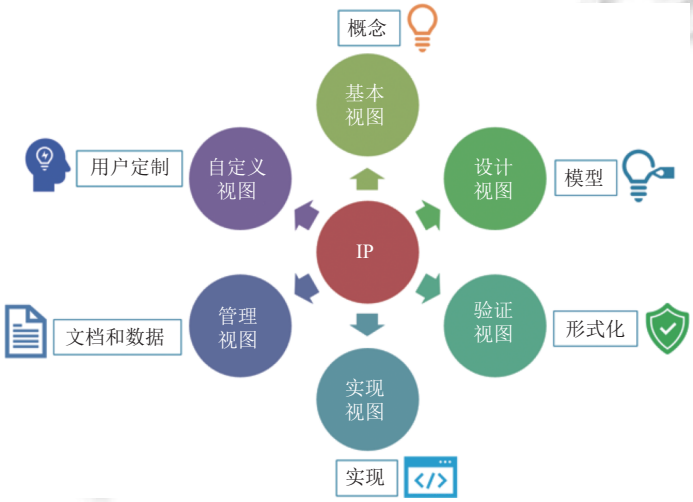

图 13 软件 IP 的视图

软件 IP 的基本视图如图 14 所示, 它展示了软件 IP 作为一个知识实体应该包含的基本要素.

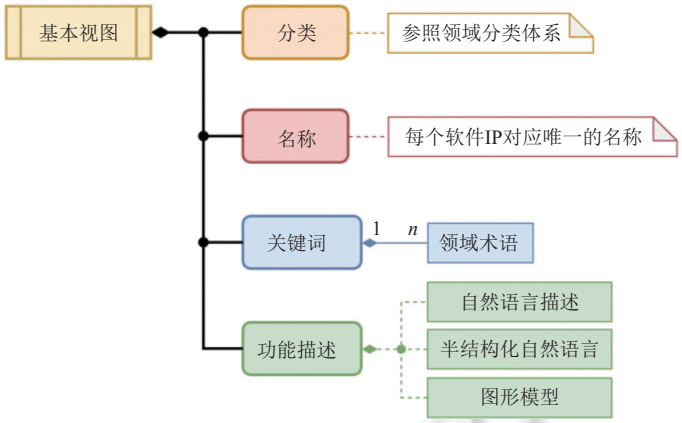

图 14 软件 IP 的基本视图

(1) 分类: 一个特定的领域应当具备一个完备的分类标准或者分类体系. 对于每一个软件 IP 都应该有一个固定的分类, 根据分类信息, 可以实现对软件 IP 的快速查找.

(2) 名称: 每一个软件 IP 有一个唯一的名称.

(3) 关键词: 相当于该软件 IP 的标签或者特征, 这里的每一个关键词都必须出自领域术语库. 关键词支持搜索, 但是和基于分类的搜索不同: 基于分类的搜索粒度更大, 搜索效率更高; 而基于关键词的搜索粒度更小, 搜索效率更低, 但是搜索结果更准确.

(4) 功能描述: 包括利用自然语言、半结构化自然语言或者图形模型描述的软件 IP 功能. 功能的描述蕴含着创建该软件 IP 的目的, 如果无法清楚地描述一个软件 IP 的需求, 那么说明创建这个 IP 的动机是不明确的, 也就是说该 IP 所传达的概念还不成熟, 还不能称之为知识实体.

基本视图中的分类、名称和关键词来自知识模型中的基本知识, 功能描述来自知识模型中软件开发过程知识的需求部分.

7.2 设计视图

软件 IP 的设计视图如图 15 所示. 相比于基本视图, 设计视图增加了软件开发过程知识中的设计部分内容, 包括接口、状态和参数、流程图和异常处理. 设计依赖于基本视图中的功能描述.

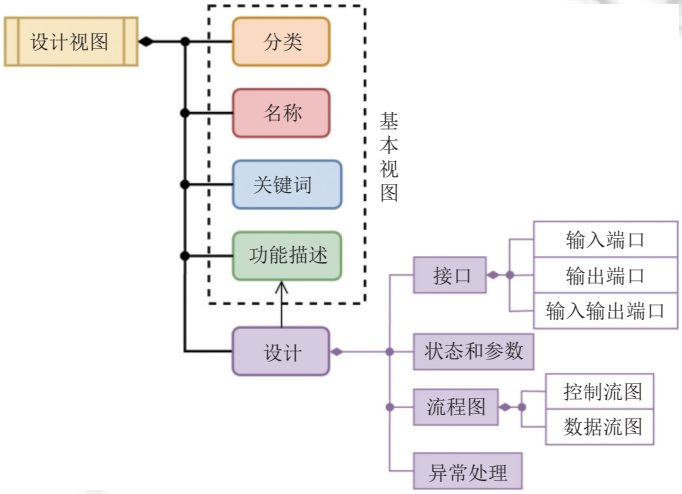

图 15 软件 IP 的设计视图

7.3 验证视图

软件 IP 的验证视图如图 16 所示. 相比于设计视图, 验证视图增加了形式模型的内容, 包括形式规约、非功能约束和 IP 不变式.

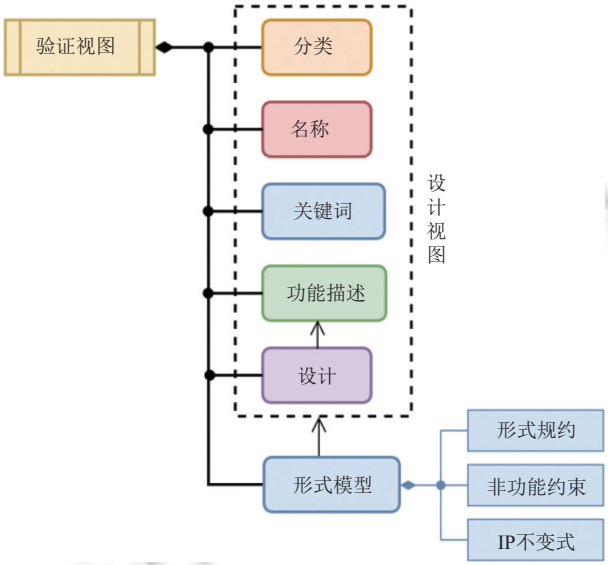

图 16 软件 IP 的验证视图

7.4 开发视图

软件 IP 的开发视图如图 17 所示. 开发视图在设计视图的基础上增加了知识模型中的实现部分 (包括开发语言、环境配置、性能特征和存在形式) 以及与软件 IP 代码相关的内容.

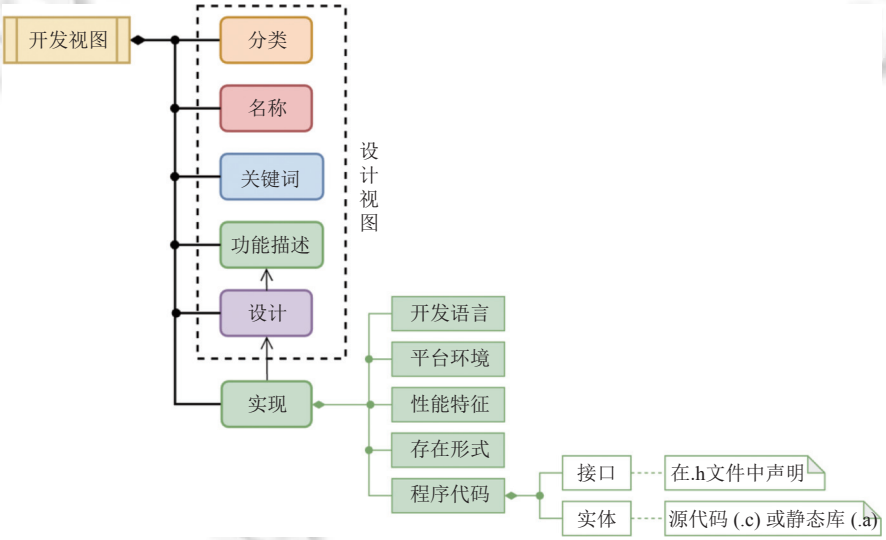

图 17 软件 IP 的开发视图

7.5 管理视图

软件 IP 的管理视图与其他视图不同, 它主要包括软件 IP 的基本信息以及各种文档和数据. 从 IP 的管理视图

可以获取与该 IP 相关的详细信息, 这些信息可以辅助基于 IP 的软件智能合成. 例如, 如果在 IP 合成或者验证的过程中发现了问题, 那么可以切换到管理视图去查阅一些资料和文档, 以检查是否对 IP 的理解有误, 然后再对 IP 进行相应的修改.

如图 18 所示, 软件 IP 的管理视图包含以下 5 部分.

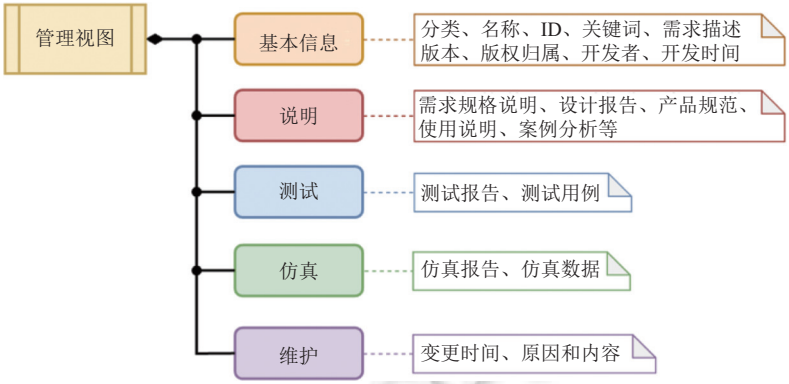

图 18 软件 IP 的管理视图

(1) 基本信息: 包括分类、名称、ID、关键词、需求描述、版本、版权归属、开发者、开发时间等; 基本信息已经包括了 IP 的基本视图.

(2) 说明: 包括需求规格说明、设计报告、产品规范、使用说明、案例分析等文档.

(3) 测试: 包括测试报告和测试用例. 根据测试结果可以对 IP 的可信等级进行评估 (certification), 如果 IP 的测试较为充分, 那么可以考虑省去对该 IP 的形式验证.

(4) 仿真: 包括仿真报告和仿真数据. 如果软件 IP 需求的图形模型 (如果有) 是可仿真的, 那么可以对该 IP 的行为进行仿真, 产生的仿真报告和仿真数据可以在管理视图下查阅.

(5) 维护: 包括变更文档, 主要记录变更时间、变更原因与变更内容等.

7.6 用户自定义视图

基本视图是软件 IP 视图的“必选”项, 它反映了一个软件 IP 作为一个软件知识实体所包含最本质的信息. 软件 IP 的其他视图是在基本视图的基础上添加相应的信息形成. 除了以上定义的设计、验证、开发和管理视图之外, 用户可以在“必选”的基本视图基础上, 根据自己的偏好, 任意勾选一些“可选”项以形成相应的视图, 即用户自定义视图.

8 案例分析: 软件 IP 提取

文献 [14] 首次提出了基于软件 IP 的嵌入式软件智能合成框架 (IPESIS), 但并未明确给出软件 IP 模型的具体定义. 本文的关注点在于提出嵌入式软件 IP 通用模型的概念, 以支撑文献 [14] 的智能合成框架. 这一工作是迈出“嵌入式软件智能合成”的第 1 步, 为后续研究 (包括软件 IP 的搜索和组装等) 奠定了理论基础.

为推动文献 [14] 中提出的 IPESIS 在实际场景中的应用落地, 本节将讨论软件 IP 的提取方法. 在实际的软件工程项目中, 软件存量资产大多以程序代码和非形式化语言撰写的各类设计与需求文档等形式存在. 这给开发者在不同应用场景下高效复用这些资产带来实质性困难. 此外, 由于软件存量资产通常规模庞大, 难以通过人工的方式提取软件 IP. 因此, 本节提出一种从嵌入式软件存量资产自动提取软件 IP 的方法, 并通过实际案例来展示提取方法的有效性和可行性. 这部分工作将为 IPESIS 的后续研究提供实践基础.

8.1 软件 IP 提取方法

给定一个嵌入式软件工程项目作为输入, 我们将工程项目中的每个函数视为一个软件 IP, 并提取其知识模型、

形式模型和实现. 本文按照“知识模型-形式模型-实现”的顺序对软件 IP 进行介绍, 但在提取软件 IP 时, 我们从“实现”部分开始. 针对输入的工程项目中的每个函数, 我们先提取其“实现”, 在此基础上继续提取其知识模型和形式模型.

根据第 5 节的描述, 软件 IP 的实现由头文件 (.h) 和实现该软件 IP 功能的源代码 (.c) 组成, 其中头文件包含输入和输出变量 (对应接口中的输入和输出端口) 以及状态和参数变量的声明. 因此, 软件 IP 实现的提取包括: (1) 声明变量提取. 根据第 3 节中关于输入、输出、状态和参数变量的定义, 我们可以为软件工程项目中每个函数提取这 4 种变量. 具体而言, 函数的输入变量是指函数形参和函数中使用的全局变量 (排除先写后读的变量); 函数的输出变量是指被修改的函数形参和全局变量 (变量或其别名被修改); 函数的状态变量首先即是该函数输入变量同时也是输出变量, 如果没有其他函数也使用了该变量, 那么该变量则是该函数的状态变量; 函数的参数变量首先是该函数的输入变量, 如果没有别的函数以该变量为输出变量, 那么该变量则是该函数的参数变量. (2) 将每个函数转换为软件 IP 实现: 基于第 (1) 步提取的变量以及项目的源代码生成软件 IP 实现 (头文件和源代码). 具体而言, 首先创建头文件 (.h) 并在其中以结构体的形式声明第 (1) 步提取的 4 类变量; 其次, 将项目中的函数转换成软件 IP 实现中源代码 (.c) 的格式; 最后, 函数的头文件 (.h) 和格式化的源代码 (.c) 构成了该函数对应的软件 IP 的实现, 外部环境可以通过引入 (#include) 软件 IP 的头文件来使用其功能.

基于提取出来的软件 IP 实现, 我们采用 LLM 的上下文学习方法, 进一步提取其知识模型. 该方法充分利用了 LLM 在代码理解和模式识别方面的能力, 并结合少量示例, 建立了一个高效的少样本学习框架. 知识模型的提取分为两个阶段: 提取阶段和验证阶段. 在提取阶段, 通过 API 调用 LLM, 并将温度系数设置为 0.3, 以平衡输出的确定性和创造性. 同时, 采用约束解码方法以确保输出格式的规范性; 验证阶段包括一个 3 层验证系统: 格式验证 (使用 JSON schema 验证输出是否符合预定义的格式要求)、交叉验证 (通过抽样进行人工审查确保提取结果的准确性) 和反向验证 (利用提取的知识模型生成软件 IP 代码片段, 并与原始代码进行比对, 验证一致性), 以确保提取结果的准确性和可靠性. 如果验证阶段未通过, 则重新进行提取过程.

针对软件 IP 知识模型提取问题, 我们从软件 IP 的实现提取相应的契约. 然而, 目前用于契约生成的开源工具[97,103,104]存在一些局限性, 例如生成效率低和自动化程度低等, 这源于它们对静态/动态分析的严重依赖以及缺乏轻量级架构. 此外, 它们对预定义的验证目标的需求, 或仅仅专注于循环不变式生成, 导致了工具的灵活性受限. 因此, 我们开发了一种面向软件 IP 的契约提取工具, 该工具利用 LLM 的能力, 以 ANSI C 规约语言 (ACSL)[105]格式为 C 程序生成形式规约 (契约), 并利用 Frama-C[106]框架进行自动化验证.

提取出来的软件 IP 通过封装软件开发者的知识, 提供了独立于具体实现的功能模块, 这些模块可以在不同的嵌入式系统开发项目中重复使用. 知识模型明确了软件 IP 的功能、接口、使用条件等关键信息, 使得开发者能够快速理解并集成这些模块到新的项目中; 形式模型通过形式化方法准确描述了软件 IP 的行为和接口规约, 减少了开发者对软件 IP 理解的偏差, 提高了复用的准确性.

8.2 案例与实验评估

本实验中从航天嵌入式领域的太搜软件 (以下简称太搜) 提取相应的软件 IP. 太搜是嵌入在卫星中的软件, 它通过陀螺仪和太阳传感器感知太阳的位置, 并根据卫星的姿态角和位置数据, 计算控制命令, 最后通过控制推进器来调整卫星的姿态. 太搜包含复杂的数据结构, 这对自动化提取工具构成很大的挑战, 其复杂性源于函数之间错综复杂的相互依赖关系、对指针的广泛使用以及复杂的数据操作流程等.

本实验的测试环境配置在 Ubuntu 24.04.1 系统上, 配备了 AMD Ryzen 9 7940HX 的 CPU 和 32 GB 的 RAM. 这一配置为执行计算密集型任务提供了均衡的环境, 同时确保有足够的内存资源来处理大型数据集. 该工具主要使用 C++和 Python 实现.

我们从太搜中提取了 49 个软件 IP. 这些软件 IP 实现部分的提取耗时近 9 s, 这表明了工具在处理具有复杂相互依赖关系但数据结构相对可控的中等复杂嵌入式系统代码时的高效性. 关于太搜软件 IP 知识模型的提取, 我们在 GPT-4o-0806 模型上对两种提示模板 (基本提示词 BP 和基于上下文学习的提示词 ICP) 进行评估, 结果如表 1

所示, 其中, *FV*=(有效输出数量/输出总数量), *MV*=(专家验证为正确的提取结果数量/经审核的提取结果总数量), *RCC*=(由生成代码片段覆盖的代码行数/原始代码总行数). 实验结果表明, 知识模型提取方法通过了 *FV* 验证, 并且基于上下文学习的提示词模板 (ICP) 在 *MV* 和 *RCC* 指标上表现更为优秀.

表 1 软件 IP 知识模型提取结果评估 (%)

提示词模板	*FV*	*MV*	*RCC*
BP	100.00	82.00	43.69
ICP	100.00	88.00	45.74

软件 IP 契约的提取工具中 LLM 使用的提示和参数如下所示: (1) 模型温度: 固定为 0.3, 以确保输出的稳定性和一致性; (2) 迭代次数: 0 表示不进行迭代, 3 表示进行迭代优化; 使用 3 次迭代是基于先前的测试, 测试表明虽然迭代优化提高了准确性, 但超过 3 次迭代后准确性的增益十分微小, 表明存在收益递减; (3) 提示词模板: 本实验考虑两种提示词: ① 基本提示词 (BP): 最简单的指令, 不提供上下文的指导; ② 增强提示词 (EP): 优化的模板, 包含结构化的设计和逐步推理. 表 2 展示了软件 IP 契约的提取结果, 其中 *SCR*=(语法正确规约的数量/生成规约的总数量), 表示语法正确率; *SVR*=(有效规约的数量/语法正确规约的数量),表示规约有效率; *TCR*=(*SCR*×*SVR*), 表示总体正确率. 该实验采用两阶段设计: 在第 1 阶段, 使用 GPT-4o-mini, 证明了提示词和迭代优化都能独立地提高性能 (第 1–4 行). 它们的结合应用导致了性能的进一步增强 (第 3, 4 行), 这表明了这两种策略之间的协同增强作用; 在第 2 阶段, 使用两种模型 GPT-4o 和 DeepSeek-V3, 它们都达到了 90% 的 SCR 值 (第 5, 6 行). 然而, 所有模型的语义理解都达到了一个平台期, GPT-4o 和 DeepSeek-V3 的 SVR 值仅略高于 GPT-4o-mini.

表 2 软件 IP 契约提取结果评估

模型	提示词模板	迭代	*SCR* (%)	*SVR* (%)	*TCR* (%)
GPT-4o-mini	BP	0	4.00	64.52	2.58
		3	18.00	71.43	12.86
	EP	0	20.00	79.01	15.80
		3	64.00	48.11	30.79
GPT-4o	EP	3	94.00	57.83	54.36
DeepSeek-V3			90.00	58.87	52.98

总之, 通过对太搜这一复杂案例开展实验研究, 我们验证了本节提出的软件 IP 提取方法在应对复杂嵌入式软件时的有效性和可行性. 该实验结果表明, 该方法具备高效处理复杂嵌入式软件的能力, 能够为文献 [14] 中提出的 IPESIS 的落地应用提供有力的技术支撑, 推动其从理论走向实际.

9 总 结

由于嵌入式系统实现功能越来越多, 软件规模变得越来越庞大和复杂, 安全攸关领域对软件可靠性要求越来越高, 因而如何高效开发可靠嵌入式软件是一个重大挑战. 为了提高嵌入式软件开发效率和质量保障, 文献 [14] 提出了软件 IP 的概念和基于软件 IP 的嵌入式软件智能合成开发模式及其框架. 在此基础上, 本文提出了嵌入式软件 IP 通用模型及其不同层次的视图表示, 进一步丰富了软件 IP 的语义. 与现有方法相比, 本文提出的软件 IP 模型的优势主要体现在以下 3 点.

(1) 中间知识制品. 现有的方法缺乏对嵌入式软件设计与开发过程中产生的中间知识的表示, 因此会给二次开发带来困难. 软件 IP 模型包含了嵌入式软件开发过程中的所有必要领域知识以及中间产品, 支持软件 IP 的自动搜索与适配, 为软件智能合成奠定知识基础.

(2) 组合正确性. MDD 和 CBD 缺乏对嵌入式环境的假设以及支撑平台的约束, 这使得组装变得很困难, 特别是组装后的系统正确性很难由模型或者构件的正确性保证. 软件 IP 模型吸取了 DbC 思想, 它对嵌入式环境以及平台都有明确的假设和约束, 所以嵌入式系统的正确性在一定程度上可以由组成它的软件 IP 的正确性来保证. 因

此, 基于软件 IP 的方法具有非常好的组合性, 可以提高嵌入式系统的开发效率以及安全性.

(3) 功能的独立性、完整性以及全局变量的使用. CBD 的构件不是功能独立的, 它依赖于外部的其他功能; DbC 描述的契约没有功能的具体实现, 因此它不是功能完整的. MDD 模型是功能独立的且完整的, 软件 IP 模型借鉴了 MDD 模型这一优点. 此外, CBD 和 DbC 中没有全局变量的定义, 部分 MDD 模型支持全局变量的使用, 但是需要在模型中引入表示全局变量的数据模块. 软件 IP 模型本身不包含全局变量的定义, 但是在代码合成阶段支持使用全局变量对合成的代码进行优化.

然而, 本文提出的理论需要经过实践的检验, 因此, 在本文的基础上, 我们下一步将研究航天控制领域更多的案例, 并且构建软件 IP 知识库, 为面向嵌入式系统的软件 IP 知识建模奠定实践基础.

致谢 感谢中国科学院软件研究所硕士生杨范芃在软件 IP 形式模型提取方面所做出的重要贡献; 感谢国防科技大学陈振邦教授及其博士生刘坤林、杜一德在软件 IP 实现提取方面提供的宝贵指导与深入的技术讨论; 感谢北京交通大学硕士生冯浩杰、熊泽昌在软件 IP 知识模型提取过程中给予的技术支持. 各位同仁的专业见解与协助, 为本研究的顺利开展提供了有力支撑, 在此谨致诚挚谢意.

References

[1] Estefan JA. Survey of model-based systems engineering (MBSE) methodologies. INCOSE MBSE Focus Group, 2007. 1–12.

[2] Karris ST. Introduction to Simulink with Engineering Applications. Fremont: Orchard Publications, 2006.

[3] Ansys Inc. Ansys SCADE suite: Model-based development environment for critical embedded software. 2025. https://www.ansys.com/products/embedded-software/ansys-scade-suite

[4] SAE International. Architecture analysis & design language (AADL) AS5506C. 2017. https://legacy.sae.org/standards/content/as5506c/

[5] Fritzson P. Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-physical Approach. Hoboken: John Wiley & Sons, 2014. [doi: 10.1002/9781118989166]

[6] McIlroy M. Mass-produced software components. In: Proc. of the 1968 NATO Conf. on Software Engineering. Garmisch: Springer, 1968. 88–98.

[7] Oracle Corporation. Enterprise JavaBeans technology. 2025. https://www.oracle.com/java/technologies/enterprise-javabeans-technology.html

[8] Object Management Group. The common object request broker: Architecture and specification. Revision 2.5. 2001. https://www.omg.org/spec/CORBA/2.5/PDF/

[9] Microsoft Corporation. [MS-DCOM]: Distributed component object model (DCOM) remote protocol. 2025. https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom/4a893f3d-bd29-48cd-9f43-d9777a4415b0

[10] Benveniste A, Caillaud B, Nickovic D, Passerone R, Raclet JB, Reinkemeier P, Sangiovanni-Vincentelli A, Damm W, Henzinger TA, Larsen KG. Contracts for system design. Foundations and Trends® in Electronic Design Automation, 2018, 12(2–3): 124–400. [doi: 10.1561/1000000053]

[11] IEEE. IEEE standard for IP-XACT, standard structure for packaging, integrating, and reusing IP within tool flows. New York: IEEE, 2023. [doi: 10.1109/IEEESTD.2023.10054520]

[12] Intel Corporation. Introduction to Intel® FPGA IP cores: UG-01056. 2020. https://www.intel.com/content/www/us/en/docs/programmable/683102/20-3/introduction.html

[13] Saleh R, Wilton S, Mirabbasi S, Hu A, Greenstreet M, Lemieux G, Pande PP, Grecu C, Ivanov A. System-on-chip: Reuse and integration. Proc. of the IEEE, 2006, 94(6): 1050–1069. [doi: 10.1109/JPROC.2006.873611]

[14] Yang MF, Gu B, Duan ZH, Jin Z, Zhan NJ, Dong YW, Tian C, Li G, Dong XG, Li XF. Intelligent program synthesis framework and key scientific problems for embedded software. Chinese Space Science and Technology, 2022, 42(4): 1–7 (in Chinese with English abstract). [doi: 10.16708/j.cnki.1000-758X.2022.0046]

[15] Booch G, Jacobson I, Rumbaugh J. The unified modeling language. Unix Review, 1996, 14(13): 5.

[16] Hause M. The SysML modelling language. In: Proc. of the 5th European Systems Engineering Conf. INCOSE, 2006. 1–12.

[17] Neto AC, Sartori F, Piccolo F, Vitelli R, de Tommasi G, Zabeo L, Barbalace A, Fernandes H, Valcarcel DF, Batista AJN. MARTe: A multiplatform real-time framework. IEEE Trans. on Nuclear Science, 2010, 57(2): 479–486. [doi: 10.1109/TNS.2009.2037815]

[18] MathWorks. Simulink documentation. 2021. https://www.mathworks.com/help/simulink

[19] Modelica Association. Modelica™—A unified object-oriented language specification for physical systems modeling: Version 1.4. 2000. https://modelica.org/documents/ModelicaTutorial14.pdf

[20] Blochwitz T. Functional mock-up interface for model exchange and co-simulation. 2014. https://www.fmi-standard.org/assets/releases/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf

[21] Blochwitz T, Otter M, Akesson J, Arnold M, Clauß C, Elmqvist H, Friedrich M, Junghanns A, Mauss J, Neumerkel D, Olsson H, Viel A. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In: Proc. of the 9th Int'l Modelica Conf. Munich: Linköping University, 2012. 173–184. [doi: 10.3384/ecp12076173]

[22] Eker J, Janneck JW, Lee EA, Jie Liu N, Xiaojun Liu N, Ludvig J, Neuendorffer S, Sachs S, Yuhong Xiong N. Taming heterogeneity—The ptolemy approach. Proc. of the IEEE, 2003, 91(1): 127–144. [doi: 10.1109/JPROC.2002.805829]

[23] Davare A, Densmore D, Guo LP, Passerone R, Sangiovanni-Vincentelli AL, Simalatsar A, Zhu Q. METROII: A design environment for cyber-physical systems. ACM Trans. on Embedded Computing Systems, 2013, 12(S1): 49. [doi: 10.1145/2435227.2435245]

[24] Porter J, Hemingway G, Nine H, vanBusKirk C, Kottenstette N, Karsai G, Sztipanovits J. The ESMoL language and tools for high-confidence distributed control systems esign. Part 1: design language, modeling framework, and analysis. Nashville: Institute for Software Integrated Systems, 2010. 109

[25] Halbwachs N, Caspi P, Raymond P, Pilaud D. The synchronous data flow programming language LUSTRE. Proc. of the IEEE, 1991, 79(9): 1305–1320. [doi: 10.1109/5.97300]

[26] Abrial JR. Modeling in Event-B: System and Software Engineering. Cambridge: Cambridge University Press, 2010. [doi: 10.1017/CBO9781139195881]

[27] Banach R, Butler M, Qin SC, Verma N, Zhu HB. Core hybrid Event-B I: Single hybrid Event-B machines. Science of Computer Programming, 2015, 105: 92–123. [doi: 10.1016/j.scico.2015.02.003]

[28] Banach R, Butler M, Qin SC, Zhu HB. Core hybrid Event-B II: Multiple cooperating hybrid Event-B machines. Science of Computer Programming, 2017, 139: 1–35. [doi: 10.1016/j.scico.2016.12.003]

[29] Woodcock J. Engineering UToPiA: Formal semantics for CML. In: Proc. of the 19th Int'l Symp. on Formal Methods. Singapore: Springer, 2014. 22–41. [doi: 10.1007/978-3-319-06410-9_3]

[30] Spivey JM. The Z Notation: A Reference Manual. New York: Prentice Hall, 1992.

[31] Hoare CAR. Communicating sequential processes. Communications of the ACM, 1978, 21(8): 666–677. [doi: 10.1145/359576.359585]

[32] Bjørner D. The vienna development method (VDM). In: Proc. of the 1979 Int'l Conf. on Mathematical Studies of Information Processing. Kyoto: Springer, 1979. 326–359. [doi: 10.1007/3-540-09541-1_33]

[33] Hoare CAR, He JF. Unifying Theories of Programming. London: Prentice Hall, 1998.

[34] Friedenthal S, Moore A, Steiner R. A Practical Guide to SysML: The Systems Modeling Language. 3rd ed., San Francisco: Morgan Kaufmann, 2014.

[35] Le Guernic P, Talpin JP, Le Lann JC. Polychrony for system design. Journal of Circuits, Systems and Computers, 2003, 12(3): 261–303. [doi: 10.1142/S0218126603000763]

[36] Le Guernic P, Benveniste A, Bournai P, Gautier T. Signal—A data flow-oriented language for signal processing. IEEE Trans. on Acoustics, Speech, and Signal Processing, 1986, 34(2): 362–374. [doi: 10.1109/TASSP.1986.1164809]

[37] Bourke T, Pouzet MZ. Zélus: A synchronous language with ODEs. In: Proc. of the 16th Int'l Conf. on Hybrid Systems: Computation and Control. Philadelphia: ACM, 2013. 113–118. [doi: 10.1145/2461328.2461348]

[38] Bourke T, Carcenac F, Colaço JL, Pagano B, Pasteur C, Pouzet M. A synchronous look at the simulink standard library. ACM Trans. on Embedded Computing Systems (TECS), 2017, 16(S5): 176. [doi: 10.1145/3126516]

[39] Benveniste A, Bourke T, Caillaud B, Colaco JL, Pasteur C, Pouzet M. Building a hybrid systems modeler on synchronous languages principles. Proc. of the IEEE, 2018, 106(9): 1568–1592. [doi: 10.1109/JPROC.2018.2858016]

[40] NASA Jet Propulsion Laboratory. F Prime: Flight software & embedded systems framework. 2020. https://nasa.github.io/fprime

[41] Bocchino R, Canham T, Watney G, Reder L, Levison J. F Prime: An open-source framework for small-scale flight software systems. In: Proc. of the 32nd Annual Small Satellites Conf. Logan, 2018. https://digitalcommons.usu.edu/smallsat/2018/all2018/328/

[42] Bocchino RL, Levison JW, Starch MD. FPP: A modeling language for F prime. In: Proc. of the 2022 IEEE Aerospace Conf. (AERO). Big Sky: IEEE, 2022. 1–15. [doi: 10.1109/AERO53065.2022.9843754]

[43] Arbab F. Abstract behavior types: A foundation model for components and their composition. Science of Computer Programming, 2005, 55(1–3): 3–52. [doi: 10.1016/j.scico.2004.05.010]

[44] Basu A, Bozga M, Sifakis J. Modeling heterogeneous real-time components in BIP. In: Proc. of the 4th IEEE Int'l Conf. on Software Engineering and Formal Methods. Pune: IEEE, 2006. 3–12. [doi: 10.1109/SEFM.2006.27]

[45] Bozga M, Jaber M, Maris N, Sifakis J. Modeling dynamic architectures using Dy-BIP. In: Proc. of the 11th Int'l Conf. on Software Composition. Prague: Springer, 2012. 1–16. [doi: 10.1007/978-3-642-30564-1_1]

[46] El-Hokayem A, Bozga M, Sifakis J. A temporal configuration logic for dynamic reconfigurable systems. In: Proc. of the 36th Annual ACM Symp. on Applied Computing. ACM, 2021. 1419–1428. [doi: 10.1145/3412841.3442017]

[47] Konnov I, Kotek T, Wang Q, Veith H, Bliudze S, Sifakis J. Parameterized systems in BIP: Design and model checking. In: Proc. of the 27th Int'l Conf. on Concurrency Theory. Québec City: Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2016. 30. [doi: 10.4230/LIPIcs.CONCUR.2016.30]

[48] Bozga M, Iosif R, Sifakis J. Checking deadlock-freedom of parametric component-based systems. Journal of Logical and Algebraic Methods in Programming, 2021, 119: 100621. [doi: 10.1016/j.jlamp.2020.100621]

[49] He JF, Li XS, Liu ZM. Component-based software engineering. In: Proc. of the 2nd Int'l Colloquium on Theoretical Aspects of Computing. Hanoi: Springer, 2005. 70–95. [doi: 10.1007/11560647_5]

[50] He JF, Li XS, Liu ZM. A theory of reactive components. Electronic Notes in Theoretical Computer Science, 2006, 160: 173–195. [doi: 10.1016/j.entcs.2006.05.022]

[51] He JF, Li XS, Liu ZM. rCOS: A refinement calculus of object systems. Theoretical Computer Science, 2006, 365(1–2): 109–142. [doi: 10.1016/j.tcs.2006.07.034]

[52] Chen X, He JF, Liu ZM, Zhan NJ. A model of component-based programming. In: Proc. of the 2007 Int'l Conf. on Fundamentals of Software Engineering. Tehran: Springer, 2007. 191–206. [doi: 10.1007/978-3-540-75698-9_13]

[53] Zhan NJ, Kang EY, Liu ZM. Component publications and compositions. In: Proc. of the 2nd Int'l Symp. on Unifying Theories of Programming. Dublin: Springer, 2008. 238–257. [doi: 10.1007/978-3-642-14521-6_14]

[54] Liebrenz T, Herber P, Glesner S. Deductive verification of hybrid control systems modeled in Simulink with KeYmaera X. In: Proc. of the 20th Int'l Conf. on Formal Engineering Methods. Gold Coast: Springer, 2018. 89–105. [doi: 10.1007/978-3-030-02450-5_6]

[55] Herber P, Adelt J, Liebrenz T. Formal verification of intelligent cyber-physical systems with the interactive theorem prover KeYmaera X. In: Proc. of the 2021 Software Engineering Satellite Events. Braunschweig: Software Engineering, 2021.

[56] Simko G, Lindecker D, Levendovszky T, Neema S, Sztipanovits J. Specification of cyber-physical components with formal semantics—Integration and composition. In: Proc. of the 16th Int'l Conf. on Model Driven Engineering Languages and Systems. Miami: Springer, 2013. 471–487. [doi: 10.1007/978-3-642-41533-3_29]

[57] Sztipanovits J, Bapty T, Koutsoukos X, Lattmann Z, Neema S, Jackson E. Model and tool integration platforms for cyber-physical system design. Proc. of the IEEE, 2018, 106(9): 1501–1526. [doi: 10.1109/JPROC.2018.2838530]

[58] Meyer B. Applying 'design by contract'. Computer, 1992, 25(10): 40–51. [doi: 10.1109/2.161279]

[59] Meyer B. Touch of class. Learning to program well with object technology and design by contract, an introduction to software engineering. 2009. https://se.inf.ethz.ch/~meyer/down/touch/TOUCH.pdf

[60] Floyd RW. Assigning meanings to programs. Mathematical Aspects of Computer Science, 1967, 19: 19–32.

[61] Abadi M, Lamport L, Wolper P. Realizable and unrealizable specifications of reactive systems. In: Proc. of the 16th Int'l Colloquium on Automata, Languages and Programming. Stresa: Springer, 1989. 1–17. [doi: 10.1007/BFb0035748]

[62] Abadi M, Lamport L. Composing specifications. ACM Trans. on Programming Languages and Systems (TOPLAS), 1993, 15(1): 73–132. [doi: 10.1145/151646.151649]

[63] de Alfaro L, Henzinger TA. Interface automata. ACM SIGSOFT Software Engineering Notes, 2001, 26(5): 109–120. [doi: 10.1145/503271.503226]

[64] Chakrabarti A, de Alfaro L, Henzinger TA, Stoelinga M. Resource interfaces. In: Proc. of the 3rd Int'l Workshop on Embedded Software. Philadelphia: Springer, 2003. 117–133. [doi: 10.1007/978-3-540-45212-6_9]

[65] Henzinger TA, Jhala R, Majumdar R. Permissive interfaces. In: Proc. of the 10th European Software Engineering Conf. Held Jointly with 13th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Lisbon: ACM, 2005. 31–40. [doi: 10.1145/1081706.1081713]

[66] Larsen KG, Nyman U, Wąsowski A. Interface input/output automata. In: Proc. of the 14th Int'l Symp. on Formal Methods. Hamilton: Springer, 2006. 82–97. [doi: 10.1007/11813040_7]

[67] Reineke J, Tripakis S. Basic problems in multi-view modeling. In: Proc. of the 20th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Grenoble: Springer, 2014. 217–232. [doi: 10.1007/978-3-642-54862-8_15]

[68] de Alfaro L, Henzinger TA, Stoelinga M. Timed interfaces. In: Proc. of the 2nd Int'l Workshop on Embedded Software. Grenoble: Springer, 2002. 108–122. [doi: 10.1007/3-540-45828-X_9]

[69] David A, Larsen KG, Legay A, Nyman U, Wasowski A. Timed I/O automata: A complete specification theory for real-time systems. In: Proc. of the 13th ACM Int'l Conf. on Hybrid Systems: Computation and Control. Stockholm: ACM, 2010. 91–100. [doi: 10.1145/1755952.1755967]

[70] David A, Larsen KG, Legay A, Nyman U, Wąsowski A. ECDAR: An environment for compositional design and analysis of real time systems. In: Proc. of the 8th Int'l Symp. on Automated Technology for Verification and Analysis. Singapore: Springer, 2010. 365–370. [doi: 10.1007/978-3-642-15643-4_29]

[71] Bertrand N, Pinchinat S, Raclet JB. Refinement and consistency of timed modal specifications. In: Proc. of the 3rd Int'l Conf. on

Language and Automata Theory and Applications. Tarragona: Springer, 2009. 152–163. [doi: 10.1007/978-3-642-00982-2_13]

[72] Bhaduri P, Stierand I. A proposal for real-time interfaces in speeds. In: Proc. of the 2010 Design, Automation & Test in Europe Conf. & Exhibition. Dresden: IEEE, 2010. 441–446. [doi: 10.1109/DATE.2010.5457163]

[73] Stierand I, Reinkemeier P, Gezgin T, Bhaduri P. Real-time scheduling interfaces and contracts for the design of distributed embedded systems. In: Proc. of the 8th IEEE Int'l Symp. on Industrial Embedded Systems. Porto: IEEE, 2013. 130–139. [doi: 10.1109/SIES.2013.6601485]

[74] Anand M, Fischmeister S, Lee I. A comparison of compositional schedulability analysis techniques for hierarchical real-time systems. ACM Trans. on Embedded Computing Systems, 2013, 13(1): 2. [doi: 10.1145/2501626.2501629]

[75] Phan LTX, Lee J, Easwaran A, Ramaswamy V, Chen SJ, Lee I, Sokolsky O. CARTS: A tool for compositional analysis of real-time systems. ACM SIGBED Review, 2011, 8(1): 62–63. [doi: 10.1145/1967021.1967029]

[76] Easwaran A, Lee I, Sokolsky O, Vestal S. A compositional scheduling framework for digital avionics systems. In: Proc. of the 15th IEEE Int'l Conf. on Embedded and Real-time Computing Systems and Applications. Beijing: IEEE, 2009. 371–380. [doi: 10.1109/RTCSA.2009.46]

[77] Reinkemeier P, Stierand I. Real-time contracts—A contract theory considering resource supplies and demands. Technical Report, SFB/TR 14 AVACS, Automatic Verification And Analysis of Complex Systems, 2014.

[78] Delahaye B. Modular specification and compositional analysis of stochastic systems [Ph.D. Thesis]. Rennes: Université Rennes, 2010.

[79] Delahaye B, Caillaud B, Legay A. Probabilistic contracts: A compositional reasoning methodology for the design of stochastic systems. In: Proc. of the 10th Int'l Conf. on Application of Concurrency to System Design. Braga: IEEE, 2010. 223–232. [doi: 10.1109/ACSD.2010.13]

[80] Caillaud B, Delahaye B, Larsen KG, Legay A, Pedersen ML, Wasowski A. Compositional design methodology with constraint Markov chains. In: Proc. of the 7th Int'l Conf. on the Quantitative Evaluation of Systems. Williamsburg: IEEE, 2010. 123–132. [doi: 10.1109/QEST.2010.23]

[81] Lu RQ, Jin Z. From knowledge based software engineering to knowware based software engineering. Scientia Sinica (Series E): Information Sciences, 2008, 38(6): 843–863 (in Chinese with English abstract). [doi: 10.1360/zf2008-38-6-843]

[82] Lu RQ. From hardware to software to knowware: IT's third liberation? IEEE Intelligent Systems, 2005, 20(2): 82–85. [doi: 10.1109/MIS.2005.27]

[83] State Administration for Market Regulation, National Standardization Administration. GB/T 36455-2018 Software component model. Beijing: Standards Press of China, 2018 (in Chinese).

[84] Crnkovic I, Sentilles S, Vulgarakis A, Chaudron MRV. A classification framework for software component models. IEEE Trans. on Software Engineering, 2011, 37(5): 593–615. [doi: 10.1109/TSE.2010.83]

[85] Xu X, Wang SL, Zhan BH, Jin XY, Talpin JP, Zhan NJ. Unified graphical co-modeling, analysis and verification of cyber-physical systems by combining AADL and Simulink/Stateflow. Theoretical Computer Science, 2022, 903: 1–25. [doi: 10.1016/j.tcs.2021.11.008]

[86] He JF. From CSP to hybrid systems. In: Roscoe AW, ed. A Classical Mind: Essays in Honour of C. A. R. Hoare. New York: Prentice Hall Int'l (UK) Ltd., 1994. 171–189.

[87] Sheng HH, Bentkamp A, Zhan BH. HHLPy: Practical verification of hybrid systems using Hoare logic. In: Proc. of the 25th Int'l Symp. on Formal Methods. Lübeck: Springer, 2023. 160–178. [doi: 10.1007/978-3-031-27481-7_11]

[88] Cosler M, Hahn C, Mendoza D, Schmitt F, Trippel C. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In: Proc. of the 35th Int'l Conf. on Computer Aided Verification. Paris: Springer, 2023. 383–396. [doi: 10.1007/978-3-031-37703-7_18]

[89] Churchill BR, Padon O, Sharma R, Aiken A. Semantic program alignment for equivalence checking. In: Proc. of the 40th ACM SIGPLAN Conf. on Programming Language Design and Implementation. Phoenix: ACM, 2019. 1027–1040. [doi: 10.1145/3314221.3314596]

[90] Jain N, Vaidyanath S, Iyer A, Natarajan N, Parthasarathy S, Rajamani S, Sharma R. Jigsaw: Large language models meet program synthesis. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 1219–1231. [doi: 10.1145/3510003.3510203]

[91] Li YJ, Choi D, Chung J, et. al. Competition-level code generation with AlphaCode. Science, 2022, 378(6624): 1092–1097. [doi: 10.1126/science.abq1158]

[92] Ross SI, Martinez F, Houde S, Muller MJ, Weisz JD. The programmer's assistant: Conversational interaction with a large language model for software development. In: Proc. of the 28th Int'l Conf. on Intelligent User Interfaces. Sydney: ACM, 2023. 491–514. [doi: 10.1145/3581641.3584037]

[93] Perry N, Srivastava M, Kumar D, Boneh D. Do users write more insecure code with AI assistants? In: Proc. of the 2023 ACM SIGSAC Conf. on Computer and Communications Security. Copenhagen: ACM, 2023. 2785–2799. [doi: 10.1145/3576915.3623157]

[94] Xu FF, Vasilescu B, Neubig G. In-IDE code generation from natural language: Promise and challenges. ACM Trans. on Software Engineering and Methodology (TOSEM), 2022, 31(2): 29. [doi: 10.1145/3487569]

[95] Yan GG, Jiao L, Wang SL, Wang LT, Zhan NJ. Automatically generating systemc code from HCSP formal models. ACM Trans. on Software Engineering and Methodology (TOSEM), 2020, 29(1): 4. [doi: 10.1145/3360002]

[96] Wang SL, Ji ZK, Xu X, Zhan BH, Gao Q, Zhan NJ. Formally verified C code generation from hybrid communicating sequential processes. In: Proc. of the 15th ACM/IEEE Int'l Conf. on Cyber-physical Systems. Hong Kong: IEEE, 2024. 123–134. [doi: 10.1109/ICCPS61052.2024.00018]

[97] Ernst MD, Perkins JH, Guo PJ, McCamant S, Pacheco C, Tschantz MS, Xiao C. The Daikon system for dynamic detection of likely invariants. Science of Computer Programming, 2007, 69(1–3): 35–45. [doi: 10.1016/j.scico.2007.01.015]

[98] Molina F, Ponzio P, Aguirre N, Frias M. EvoSpex: An evolutionary algorithm for learning postconditions. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 1223–1235. [doi: 10.1109/ICSE43902.2021.00112]

[99] Pei KX, Bieber D, Shi KS, Sutton C, Yin PC. Can large language models reason about program invariants? In: Proc. of the 40th Int'l Conf. on Machine Learning. Honolulu: PMLR, 2023. 27496–27520.

[100] Alpuente M, Pardo D, Villanueva A. Abstract contract synthesis and verification in the symbolic K framework. Fundamenta Informaticae, 2020, 177(3–4): 235–273. [doi: 10.3233/FI-2020-1989]

[101] Blasi A, Goffi A, Kuznetsov K, Gorla A, Ernst MD, Pezzè M, Castellanos SD. Translating code comments to procedure specifications. In: Proc. of the 27th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Amsterdam: ACM, 2018. 242–253. [doi: 10.1145/3213846.3213872]

[102] Jackson D. The Essence of Software: Why Concepts Matter for Great Design. Princeton: Princeton University Press, 2021.

[103] Alpuente M, Feliú MA, Villanueva A. Automatic inference of specifications using matching logic. In: Proc. of the 2013 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation. Rome: ACM, 2013. 127–136. [doi: 10.1145/2426890.2426914]

[104] Wen C, Cao JL, Su J, Xu ZW, Qin SC, He MD, Li HK, Cheung SC, Tian C. Enchanting program specification synthesis by large language models using static analysis and program verification. In: Proc. of the 36th Int'l Conf. on Computer Aided Verification. Montreal: Springer, 2024. 302–328. [doi: 10.1007/978-3-031-65630-9_16]

[105] Baudin P, Filliâtre JC, Marché C, Monate B, Moy Y, Prevosto V. ACSL: ANSI C specification language. Technical Report, v1.2, CEA-list, 2008.

[106] Kirchner F, Kosmatov N, Prevosto V, Signoles J, Yakobowski B. Frama-C: A software analysis perspective. Formal Aspects of Computing, 2015, 27(3): 573–609. [doi: 10.1007/s00165-014-0326-7]

附中文参考文献

[14] 杨孟飞, 顾斌, 段振华, 金芝, 詹乃军, 董云卫, 田聪, 李戈, 董晓刚, 李晓锋. 嵌入式软件智能合成框架及关键科学问题. 中国空间科学技术, 2022, 42(4): 1–7. [doi: 10.16708/j.cnki.1000-758X.2022.0046]

[81] 陆汝钤, 金芝. 从基于知识的软件工程到基于知件的软件工程. 中国科学 E 辑: 信息科学, 2008, 38(6): 843–863. [doi: 10.1360/zf2008-38-6-843]

[83] 国家市场监督管理总局, 国家标准化管理委员会. GB/T 36455-2018 软件构件模型. 北京: 中国标准出版社, 2018.

作者简介

徐雄, 博士, 助理研究员, 主要研究领域为信息物理融合系统设计, 模型驱动开发, 形式化方法.

马旭, 硕士生, 主要研究领域为软件工程, 形式化方法.

高强, 硕士, 主要研究领域为形式化方法.

计泽坤, 硕士, 主要研究领域为形式化方法.

王淑灵, 博士, 副研究员, CCF 专业会员, 主要研究领域为信息物理融合系统形式验证.

詹博华, 博士, CCF 专业会员, 主要研究领域为形式化方法, 定理证明, 程序验证.

詹乃军, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为信息物理融合系统设计, 程序验证, 模态和时序逻辑.

李晓锋, 研究员, CCF 专业会员, 主要研究领域为嵌入式高可信软件开发, 软件缺陷检测, 软件智能开发.

顾斌, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为计算机控制, 可信软件, 信息物理系统.

董晓刚, 研究员, 主要研究领域为嵌入式高可信软件开发, 软件缺陷检测, 软件智能开发.

杨孟飞, 博士, 研究员, 博士生导师, CCF 会士, 主要研究领域为空间飞行器设计, 控制计算机, 可信软件, 软件智能开发.

刘洋, 硕士, 主要研究领域为软件工程, 自然语言处理.

冀振燕, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为人工智能, 信息安全, 软件工程.