软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(12):5397−5418 [doi: 10.13328/j.cnki.jos.007081] [CSTR: 32375.14.jos.007081]

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

基于投影的嵌入式系统需求分析方法*

王小齐[1], 陈小红[1], 金 芝[2,3], 顾 斌[4], 綦艳霞[4]

[1](上海高可信计算重点实验室 (华东师范大学), 上海 200062)

[2](高可信软件技术教育部重点实验室 (北京大学), 北京 100871)

[3](北京大学 计算机学院, 北京 100871)

[4](北京控制工程研究所, 北京 100190)

通信作者: 金芝, E-mail: zhijin@pku.edu.cn; 綦艳霞, E-mail: qiyanxia369@sina.com

摘　要: 嵌入式系统正变得越来越复杂, 其软件系统的需求分析成为嵌入式系统开发的瓶颈问题. 设备依赖性和执行逻辑的交织性是嵌入式软件系统的典型特征, 要求需求分析方法能根据所依赖的设备进行有效的需求解耦. 从基于环境建模的需求工程思想出发, 针对嵌入式软件系统的特征, 提出一种基于投影的从系统需求到软件需求的分析方法, 帮助需求工程师对需求进行有效解耦. 首先总结嵌入式软件系统的系统需求与软件需求描述, 基于交互环境特性定义嵌入式软件系统的需求解耦策略, 并设计从系统需求到软件需求的规约过程. 在航天器太阳搜索系统进行实际案例研究, 并通过耦合度与内聚性两个指标对 5 个代表性案例场景进行量化评估, 展示所提方法的有效性.

关键词: 嵌入式系统; 需求分析; 需求解耦; 问题投影; 软件需求

中图法分类号: TP311

中文引用格式: 王小齐, 陈小红, 金芝, 顾斌, 綦艳霞. 基于投影的嵌入式系统需求分析方法. 软件学报, 2024, 35(12): 5397–5418. http://www.jos.org.cn/1000-9825/7081.htm

英文引用格式: Wang XQ, Chen XH, Jin Z, Gu B, Qi YX. Projection-based Requirements Analysis Approach for Embedded Systems. Ruan Jian Xue Bao/Journal of Software, 2024, 35(12): 5397–5418 (in Chinese). http://www.jos.org.cn/1000-9825/7081.htm

Projection-based Requirements Analysis Approach for Embedded Systems

WANG Xiao-Qi[1], CHEN Xiao-Hong[1], JIN Zhi[2,3], GU Bin[4], QI Yan-Xia[4]

[1](Shanghai Key Laboratory of Trustworthy Computing (East China Normal University), Shanghai 200062, China)

[2](Key Lab of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

[3](School of Computer Science, Peking University, Beijing 100871, China)

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

Abstract: Embedded systems are becoming increasingly complex, and the requirements analysis of their software systems has become a bottleneck in embedded system development. Device dependency and interleaving execution logic are typical characteristics of embedded software systems, necessitating effective requirement analysis methods to decouple the requirements based on device dependencies. Starting from the idea of environment-based modeling in requirement engineering, this study proposes a projection-based requirement analysis approach from system requirements to software requirements for embedded software systems, helping requirement engineers to effectively decouple the requirements. The study first summarizes the system requirement and software requirement descriptions of embedded software systems, defines the requirement decoupling strategies of embedded software systems based on interactive environment characteristics, and designs the specification process from system requirements to software requirements. A real case study is carried out in the spacecraft sun search system, and five representative case scenarios are quantitatively evaluated through two metrics of coupling and cohesion, which

* 基金项目: 国家自然科学基金 (62192731, 62272166, 62192730)
收稿时间: 2023-04-21; 修改时间: 2023-07-20, 2023-09-04; 采用时间: 2023-09-30; jos 在线出版时间: 2024-03-27
CNKI 网络首发时间: 2024-03-29

demonstrate the effectiveness of the proposed approach.

Key words: embedded system; requirement analysis; requirement decoupling; problem projection; software requirement

嵌入式系统已经从单片微型计算机、微控制器、片上系统发展到网络化, 其应用涉及智能家居、工业控制、交通管理、航空航天、汽车电子、轨道交通等领域, 有着良好的发展前景[1]. 与此同时, 嵌入式系统规模及其需求的复杂度也在不断增加[2]. 例如, 2007 年一辆典型的高级汽车包含大约 270 个与驾驶员互动的功能, 而 2010 年高级汽车包含 500 多个此类功能[3,4]. 不仅如此, 嵌入式系统需求复杂度还体现在连接设备的剧增, 2010 年全世界有 125 亿台智能互联设备. 而到 2020 年, 这个数字已经翻了 4 倍. 设备控制需求的复杂性给嵌入式软件系统的开发带来巨大的挑战. 传统的仅关注软件需求的方法已经不适合现在的复杂嵌入式系统的需求分析, 嵌入式系统的软件需求分析已经成为嵌入式系统开发的瓶颈问题.

一般来说, 一个嵌入式系统通常包含软件和多个设备, 如感应器和作用器等. 软件从感应器获得信息, 通过作用器控制外部操作对象, 其需求具有较强的设备依赖性. 这种依赖性使得嵌入式系统的需求相互交织, 需要能有效地将高层粗粒度需求分维度抽象并进行模块封装, 形成高内聚、低耦合的独立的控制需求集合. 目前已经有一些关于嵌入式软件系统需求分析的工作. 如 Lavi 等人提出的 ECSAM (embedded computer system analysis and modeling) 方法[5,6], 引导需求工程师采用结构化分析将嵌入式软件系统分解为若干子系统. Alford 提出了 SYSREM (the system requirements engineering methodology) 方法[7]指导需求工程师进行功能分解. Wrigley 用图形方式描述需求分析过程[8], 引导需求工程师从目标系统意图开始逐步进行功能精化. 这些工作研究采用了传统的功能分解或者精化策略, 它们以精确划分为原则, 无法处理嵌入式系统控制需求的交织性问题. 如何解耦这些纵横交错的控制需求是复杂嵌入式软件系统需求分析的难点.

在现有的需求工程方法中, Jackson 提出的问题框架方法[9]将软件即将交互的对象建模为外部交互环境, 并将需求表示为对交互环境的预期变化. 在嵌入式系统中, 设备作为一种特殊的交互环境, 其变化明显, 由于嵌入式系统需求的强设备依赖性, 使得问题框架方法非常适合描述其系统需求[10]. 特别地, 为了应对需求的复杂度, 问题框架方法还定义了投影. 与精确划分不同, 投影指采用特定的视角和角度, 只描述和视角相关的东西, 允许交互环境重叠. 在嵌入式系统中, 多个功能通常共享同一设备的情况, 导致交互环境大量重叠, 因此投影特别适合嵌入式系统的需求解耦. 在此基础上, 我们的前期工作[11−13]提出了一种以情景图为投影维度的软件需求问题投影方法. 然而, 该投影方法只关注与投影维度相关的需求解耦, 若投影维度不完整, 则可能导致信息缺失. 因此, 本文以问题框架方法为基础, 以基于环境建模的需求工程方法思想[10]为指导, 从嵌入式系统的交互环境关注点入手, 提出嵌入式软件系统的需求描述方法. 我们针对交互环境间的关系定义了问题解耦策略, 辅助完善投影维度, 避免信息的缺失. 在投影的基础上, 形成了一种系统化的嵌入式软件系统需求规约方法, 以应对控制需求交织问题. 本文的主要贡献如下.

- 根据嵌入式软件系统的需求特征, 提出了动 (情景图) 静 (问题图) 结合的方式描述嵌入式系统的系统需求与软件需求, 为需求问题解耦提供了基础.
- 定义了 5 种嵌入式软件系统需求问题解耦策略, 针对设备操作无约束、同一设备多种操作、直接数据依赖、复杂计算等情况, 设计了相应的需求问题解耦策略.
- 定义了系统边界识别、系统需求确定、软件需求推断、复杂需求解耦等活动, 形成了复杂嵌入式软件系统的需求规约过程, 建立了需求之间的可追踪关系, 为未来需求变更奠定了基础.

本文第 1 节对相关工作进行比较. 第 2 节介绍预备知识. 第 3 节给出嵌入式软件系统的系统需求与软件需求描述. 第 4 节定义问题解耦策略. 第 5 节设计嵌入式软件系统需求规约过程. 第 6 节以航天器太阳搜索系统为例进行案例研究, 并对本文方法进行了有效性评估. 最后总结全文.

1 相关工作

目前大部分关于嵌入式系统需求分析工作都是基于功能分解的. 例如, Lavi 等人使用 ECSAM 方法[5,6]提出了

从系统外部行为识别系统内部行为的方法. 其中, 外部行为指系统与其操作环境间的交互. 通过使用可操作情景, 并限制系统和接口的分析范围, 将外部行为转换为内部行为. 系统需求的精化由需求工程师根据概念子系统或结构化分析来确定子系统. 然而, 该方法并未提供需求分析的具体制导, 而是依赖需求工程师的经验. Alford 等人提出 SYSREM 方法[7], 该方法定义了系统需求并将它们分配到数据处理子系统中. 针对功能分解, 每个功能的输入或者输出项被分解为序列或者低层次的并发项. 该方法基于图形进行分析, 并给出了数据项分解所需要满足的条件. 然而, 该方法的数据分解过程主要依靠分析员的经验.

有一些相关工作采用需求精化的策略, 逐步将抽象需求精化为具体需求. 如 Wrigley 提出了一种方法来理解文本功能需求或者业务活动, 识别所需要的物理实体行为, 并根据行为类型进行组合或者进行精化, 形成原子行为需求[14]. 在进一步的工作中, Wrigley 用功能性 (functionality) 表达系统行为的抽象, 对应 SysML[15]的用例 (use case)、需求 (requirement) 等概念[8]. 其需求分析过程从目标系统的功能或意图出发, 采用图形化模型表示并逐步添加细节, 将原功能精化为最终可由已知系统组件提供的功能, 从而最终获得足够的细节功能描述. 然而, 这些方法都没有明确定义如何获取初始功能或意图, 也没有定义明确的精化策略.

还有一些工作结合了需求分解和精化. 例如, Penzenstadler 提出了 DeSyRe 方法[16,17], 在系统需求已知的情况下, 使用假设/保证推理和分解精化模式对原需求进行了分解. 其中的假设/保证推理使用 Hoare 逻辑的前置/后置条件进行形式描述, 还定义了 3 种分解精化模式, 指导需求工程师进行需求分析. 然而, 该方法要求使用较难获得的 Hoare 逻辑表达式, 基于 Hoare 逻辑的推理也相对困难. 另外, 还有一些工作针对特定领域进行了分析. 例如从 A-7 飞机软件开发衍生出的 SCR (software cost reduction) 方法[18,19], 提供了一种从系统需求到行为需求的规约方法. 它通过识别系统的输入和输出数据项, 将软件能力视作一组从输入数据项到输出数据项的函数.

功能分解和精化以精确划分为原则, 它们分解出的子功能之间是独立的, 不关心需求 (功能) 之间的交织, 也不关心交互环境重叠. 与它们不同, 投影是 2001 年 Jackson 在问题框架方法中提出的一种控制问题复杂度的解耦手段[9], 能有效解耦出子需求问题, 允许设备在需求描述中重叠. 我们前期对投影做了大量深入的工作. 早在 2006 年, Jin 等人就提出了情景图 (scenario graph) 的雏形, 即子句关联图, 并提出了投影的启发式规则[20]. 在 2009 年我们提出了基于问题图 (problem diagram) 和情景图的自动投影方法[11], 采用情景图作为投影维度, 对问题图进行了自动投影. 不仅如此, 我们还提出了良构情景的修正策略, 通过精化问题图和情景图[9], 保证投影出的子需求问题的完整性和正确性. 我们针对轨道交通领域[12]定义了基于变量约束以及情景的投影方法, 对轨道交通中共享设备和共享变量进行了解耦[13], 针对解耦出的每个子需求进行验证, 降低了安全需求验证的复杂度.

与功能分解和精化的工作相比, 本文所提出的方法遵循基于环境建模的需求工程理念, 强调嵌入式系统需求解耦要理清由于共享设备导致的控制需求交织问题, 不仅要做功能分解, 还要能解耦不同设备在不同功能中所处的角色. 与前期的投影工作只关注投影维度不同, 本文从不同的交互环境之间的关系出发, 定义了问题解耦策略, 以获取更多的投影维度. 这些维度可以在投影基础上帮助解耦交织的控制需求, 增加解耦后需求的内聚性并降低耦合度. 本文的解耦还可以提供需求之间的可追踪关系, 有助于领域专家理解问题解耦过程, 并便于在需求变更时进行需求重用.

2 预备知识

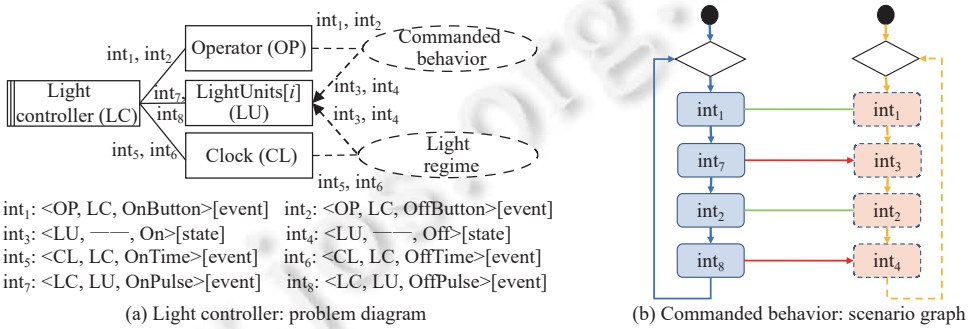
预备知识主要包括问题图、情景图以及问题投影.

2.1 问题图

问题图由 Jackson 在问题框架方法中提出[9], 它表达了对问题领域及需求的约束. 它明确区分了软件 (machine) 和问题领域 (problem domain), 其中软件是待开发软件, 用双竖线矩形框表示. 问题领域是软件将要交互的实体, 用矩形框表示. 问题领域包括 3 种: 因果领域、词法领域和自主领域. 其中因果领域指领域的现象之间存在可预测的因果关系, 如设备. 词法领域则指数据的物理存储, 如已有的数据库, 常用单竖线矩形表示. 自主领域主要指人. 软件与问题领域之间有交互, 用实线表示. 这些交互可以用五元组 $<ID,Ini,Rec,Con,Typ>$ 表示, 其中, ID 为交互编

号, *Ini* 为交互发起者, *Rec* 为交互接收者, *Con* 为共享现象, *Typ* 为现象类型, 包括事件 (event)、状态 (state) 和值 (value). 这些交互又称为行为交互. 不仅如此, 问题图还明确表达了需求 (用虚线椭圆表示), 以及需求对问题领域现象的引用或者约束. 需求引用 (reference, 用不带箭头的虚线表示) 和需求约束 (constraint, 用带箭头的虚线表示) 分别表示需求是“提到”这个现象, 还是“约束”该现象发生, 其中预期的问题领域发生的现象也可以用交互的形式进行表达. 为了与行为交互进行区别, 这些交互称为期望交互.

图 1(a) 是一个问题图的例子, 该例子源自文献 [9], 表达了要遵照操作员的要求开关灯 (comandded behavior) 以及按时间规律开关灯 (light regime) 两个需求. 其中灯光控制器 (light controller (LC)) 是待开发的软件, 操作员 (operator (OP))、灯光单元 (LightUnits[*i*] (LU)) 和时钟 (clock (CL)) 为该软件的问题领域, 灯光单元和时钟为因果领域, 操作员为服从式领域. 图中有 6 个行为交互: int_1, int_2, int_5–int_8, 描述了软件与这些问题领域之间的交互行为. 每个需求都对问题领域有需求引用或约束. 如遵照操作员的要求开关灯需求, 约束期望交互 int_3 和 int_4 的发生, 即问题领域要处于开和关状态 (On, Off).



(a) Light controller: problem diagram (b) Commanded behavior: scenario graph

图 1 问题图和情景图例子

2.2 情景图

情景图由本团队在 2009 年提出, 用于描述问题图中的行为交互如何实现需求中期望交互的过程[11]. 其中节点主要包括 3 种, 行为交互 (实线圆角矩形)、期望交互 (虚线圆角矩形) 与控制节点, 行为交互直接来自问题图, 期望交互为问题图中的需求引用或约束中的行为表示. 控制节点我们采用 UML 活动图中的图标表示, 如开始节点 (黑色实心圆)、终止节点 (圆环)、分支节点 (黑色短线)、合并节点 (菱形) 和判断节点 (菱形) 等活动控制节点.

情景图中边由交互节点之间的 5 种关系组成, 它们分别是行为序关系 (BO)、期望序关系 (EO)、行为使能关系 (BE)、需求期望关系 (RE) 和同步关系 (SC). 其中, 行为序关系表示行为交互之间或者行为交互与控制节点之间的先序关系, 用从行为交互/控制节点指向行为交互/控制节点的带箭头实线表示, 如图 1(b) 中 int_1 与 int_7 之间的关系. 期望序关系表示期望交互之间或者期望交互与控制节点之间的先序关系, 用由期望交互/控制节点指向期望交互/控制节点的带箭头虚线表示, 如图 1(b) 中 int_1 与 int_3 之间的关系. 行为使能关系表示行为交互使能期望交互发生, 用由行为交互节点指向期望交互节点的带箭头实线表示, 如图 1(b) 中 int_7 与 int_3 之间的关系. 需求期望关系表示期望交互要求行为交互发生, 用由期望交互节点指向行为交互节点的带箭头实线表示. 同步关系表示行为交互与期望交互同时发生, 用无箭头实线表示.

2.3 基于问题图和情景图的问题投影

投影可以将复杂问题解耦, 得到若干子问题. 每个子问题可以看作是问题在上下文环境中的投影, 我们在前期工作中定义了基于情景图的问题投影[11]. 具体表示为: $\pi_{sce}^{req}(PD) = SPD$. 其中, π 是投影算子, PD 是待投影的问题, sce 是投影维度——情景图, req 是 sce 要实现的需求, SPD 是投影后得到的子问题集合. 每个 (子) 问题都有独立的问题图和情景图.

例如, 以情景图 1(b) 为投影维度对问题图 1(a) 进行投影, 其投影出的子问题图如图 2 所示. 其中, 期望交互

int_1–int_4 以及行为交互 int_1、int_2、int_7、int_8 都是情景图 1(b) 中涉及的交互, 问题领域操作员 (OP) 和灯光单元 (LU) 是这些交互涉及到的领域, 需求是该情景图实现的需求遵照操作员的要求开关灯 (commanded behavior). 软件则可以人工改名. 图 2 和图 1(b) 分别是这个子问题的问题图和情景图.

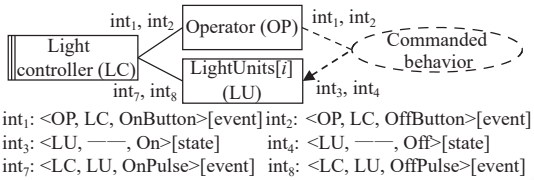


图 2 投影出的子问题图: 遵照操作员的要求开关灯

需要指出的是, 图 1(a) 所示问题图中的 2 个需求均约束设备——灯光单元 (LU) 打开与关闭, 两个需求之间存在控制设备的交织. 若某需求要求设备灯光单元打开, 而同时另一个需求要求灯光单元关闭时, 会出现灯光单元状态冲突问题, 为了解决这个问题, 我们在本文中根据解耦策略 (见第 4 节设备操作融合和抽象模式), 修改原来的问题图, 将所有对灯光单元的操作放在一个数据库中, 通过读取指令对灯光单元进行统一操作, 从而避免出现冲突问题, 而传统功能分解方法难以发现此类问题.

3 嵌入式软件系统需求描述

本节分析嵌入式软件系统的需求特征, 并由此提炼出嵌入式系统中系统需求与软件需求表达方式, 明确跨越系统需求与软件需求的表示——问题图和情景图, 以此作为需求问题解耦和软件需求规约的基础.

3.1 嵌入式软件系统需求特征

嵌入式系统是具备特定接口与功能的计算软硬件综合体. 其中, 嵌入式软件系统 (软件控制器, software controller) 一般通过感应器 (sensor) 和作用器 (actuator) 来实现对监测对象的感知和控制对象的作用, 它也可以接收并响应人类操作员 (operator) 的命令, 从数据存储 (data store) 中进行数据存储或者读取, 以及按时间进行数据处理等. 以嵌入式软件系统 (embedded software system) 为核心, 需要与之交互的对象作为环境实体 (environment entity), 可以画出嵌入式软件系统的结构图, 如图 3 所示. 其中环境实体包括: 操作员 (operator)、被监测实体 (monitored entity) 与被控制实体 (controlled entity). 复杂嵌入式软件系统一般由一组软件控制器、数据存储以及受软件调控的设备组成, 其中设备包括时钟 (clock)、感应器和作用器等.

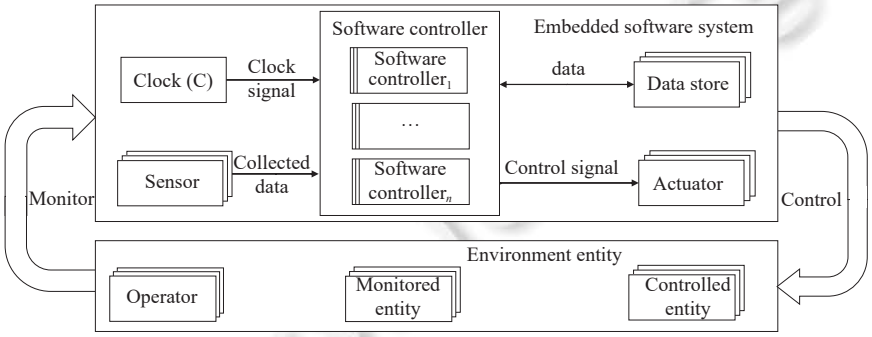


图 3 嵌入式软件系统结构图

嵌入式软件系统一般都是为完成特定用途而开发, 软件系统的需求包括控制和管理嵌入式系统资源的能力、通过和外界的交互完成面向特定应用功能的能力等. 前者是嵌入式操作系统的能力, 后者是嵌入式应用软件系统的能力, 本文主要关注后者, 即如何分析复杂嵌入式软件系统的应用需求.

根据图 3 所示的嵌入式软件系统结构, 软件系统的需求表现为软件系统通过感应器和作用器完成对外部环境

实体的感知和控制. 其控制过程需要实现特定用途指定的应用控制逻辑, 并规避所有可能出现的风险. 这里的控制过程具体表现为软件系统与感应器和作用器的交互, 以及感应器或作用器与它们能感知或作用的外部环境实体之间的交互, 这些交互的顺序需要遵循所要求的应用控制逻辑.

3.2 软件系统需求与软件需求

我们区分嵌入式软件的系统需求和软件需求. 其中, 嵌入式软件的系统需求 (system requirements) 为根据应用控制逻辑导出的预期发生在感应器或作用器与外部实体的现象, 以及它们必须遵循顺序. 这就是期望交互及其约束. 复杂嵌入式软件系统一般需要实现多个应用控制逻辑, 而它们可能相互交织, 在嵌入式软件系统需求获取的同时需要获取到所有的应用控制逻辑. 由此, 嵌入式软件系统需求可以用如下四元组表示:

$$< RS, EI, ES, ER >,$$

其中,

- $RS = \{rs_1, \ldots, rs_n\}$ 是嵌入式软件系统需求的集合;
- $EI = \{ei_1, \ldots, ei_n\}$ 是根据应用控制逻辑推导出的期望交互的集合;
- $ES : EI \rightarrow RS$ 是期望交互与系统需求间的支撑关系;
- $ER : EI \rightarrow EI$ 是期望交互发生的先序约束.

针对每个期望交互, 我们依然使用交互的五元组表示, 其中部分可以缺省. 特别地, 对共享现象类型, 在问题框架方法的分类基础上, 根据嵌入式系统特征, 本文进一步将事件分为指令 (instruction) 与信号 (signal) 两种类型. 因此可以将交互分为指令、信号、状态、值这 4 种类型.

软件控制器与感应器或作用器进行交互, 调度并协调这些设备完成系统需求, 我们称软件与感应器或作用器之间的交互为行为交互. 每个行为交互也用上述的交互形式表达. 行为交互及其约束形成了软件需求 (software requirements). 软件需求可以表达为四元组:

$$< CS, BI, BS, BR >,$$

其中,

- $CS = \{cs_1, \ldots, cs_n\}$ 是软件需求集合;
- $BI = \{bi_1, \ldots, bi_n\}$ 是行为交互的集合;
- $BS : BI \rightarrow CS$ 是行为交互与软件之间的支撑关系;
- $BR : BI \rightarrow BI$ 是行为交互发生的先序约束.

行为交互与期望交互关系如下. 行为交互使能了期望交互, 其使能过程要能满足期望交互的顺序约束, 从而实现应用控制逻辑. 特别地, 根据文献 [21], 软件需求与系统需求之间的鸿沟需要领域知识进行跨越. 在系统需求中涉及的期望交互是通过软件控制器联合设备、数据存储等实现的, 也就是说期望交互实际上是由软件行为联合设备、数据存储等领域行为实现的, 而领域行为则是领域知识, 通过它们, 可以建立在期望交互与行为交互之间的关系. 这些关系则可以通过情景图上的期望交互与行为交互之间的关系进行描述, 包括同步、行为使能、需求期望 3 种可能的关系. 由此我们发现, 问题图和情景图可以用来描述嵌入式软件系统的需求, 它们既涉及系统需求描述, 也涉及软件需求描述, 可以用情景图表达从系统需求推导软件需求的过程. 因此, 在本文中, 我们采用问题图和情景图作为跨越系统需求到软件需求的桥梁.

4 需求解耦策略

本文将投影作为一种嵌入式软件系统的解耦手段, 解决复杂嵌入式软件系统的需求解耦问题, 将复杂软件解耦为相对简单的软件. 为了得到更多的投影维度, 我们观察到, 嵌入式软件系统控制能力的解耦与设备和数据处理有密切的关系. 根据设备操作和数据处理的特性, 我们设计解耦策略以支持获取软件需求的投影维度. 根据高内聚与低耦合原则, 我们总结出 5 种解耦策略模式.

- 策略 1: 独立设备操作分离模式

该策略主要针对设备操作之间不存在约束关系的情况, 旨在将没有约束关系的需求解耦开. 具体来说, 如果一个系统需求涉及对不同设备的操作, 并且这些操作不存在约束关系, 可以将该需求按设备进行拆分, 形成一组独立的子需求, 使得每个子需求仅涉及单一设备. 例如, 在图 4(a) 中, “加电 (power-on)”需求要求分别对设备 1 (Device1) 和设备 2 (Device2) 进行加电操作. 在其情景图中 (图 4(b)), 这两个加电操作是并发关系, 没有约束, 因此可以将该需求拆分为两个子需求, 即“设备 1 加电 (Device1 power-on)”和“设备 2 加电 (Device2 power-on)”, 其问题图如图 4(c) 所示. 同样情景图也分为两个, 如图 4(d) 和图 4(e) 所示. 该策略可以形式地表示如下.

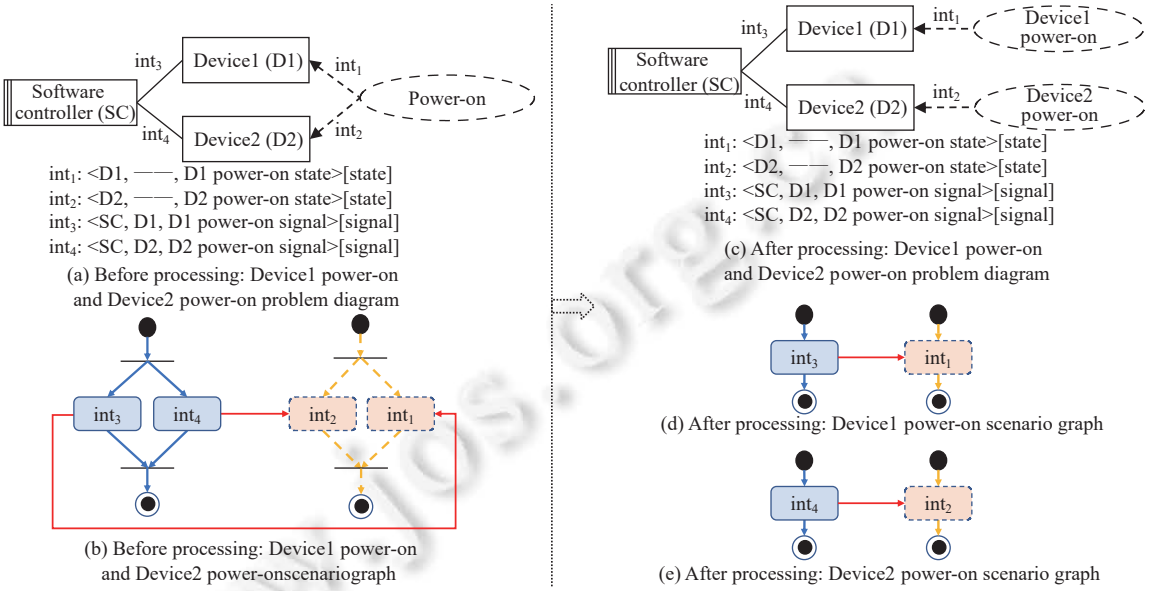


图 4　应用解耦策略 1 后设备加电例子的结果

解耦策略 1: 对问题图 $PD=<CS,DS,BI,RS,EI,RC>$ 及其对应的情景图 $SD_i=<BI_i,EI_i,CN_i,ES_i>$, 其中 $RS=\{rs_1,rs_2,\ldots,rs_n\}$, $1\leqslant i\leqslant n$, $BI=\bigcup_{i=1}^{n}BI_i$, $EI=\bigcup_{i=1}^{n}EI_i$. 若存在 $rs_i\in RS$ 要求对两个设备 (为简化叙述, 这里假设只有两个设备, 若有多个设备可以同理进行) 进行交互, 且这些交互在 rs_i 对应的情景图 SD_i 中不存在依赖约束, 则执行如下步骤.

1) 将需求 rs_i 拆分成两个需求 rs_{i1} 和 rs_{i2}, 其中 rs_{i1} 和 rs_{i2} 是分别涉及这两个设备的系统需求, 即 $RS=RS\setminus\{rs_i\}\cup\{rs_{i1},rs_{i2}\}$;

2) 在 RC 中修改相应的需求引用和需求约束;

3) 将情景图 SD_i 拆分为两个情景图 SD_{i1} 和 SD_{i2}, 其中 SD_{i1} 负责实现 rs_{i1}, SD_{i2} 负责实现 rs_{i2}.

通过策略 1, 我们将一个问题中的若干组无相关设备操作序列拆分到多个子问题中, 这样可以使得每个问题内的设备操作相关性增加, 从而提高了每个问题图的内聚性. 在策略 1 之前, 由于是一个问题, 不存在耦合度, 应用策略 1 后, 由于子问题之间没有依赖关系, 因此, 也没有耦合度.

● 策略 2: 设备操作融合和抽象模式

该策略主要针对多个需求要求软件向同一设备发起多种不同操作的情况, 每个操作具有优先级. 本策略旨在通过抽象, 将与设备交互的需求单独独立出来, 使得原来的软件仅需要负责产生与设备操作相关的指令, 而将具体指令的执行放到设备交互需求里. 这样就可以使得其他需求不需要关注与设备交互的细节. 在这种情况下, 增加一个新的“设备控制输出”需求负责与该设备进行实际交互, 它将所有原始需求中的指令转换为设备所需要的信号, 并按照每条指令的优先级将转化后的信号按照一定顺序发送给设备. 当存在多条不同优先级指令待处理时, 高优先级指令可以率先被转换为信号并发送给设备, 对于相同优先级指令, 按照先来后到顺序进行处理. 原始需求也将改变成负责生成指令. 这样的“指令”将使得新的需求“设备控制输出”与改进后的原始需求之间存在着数据依赖关

系. 这种数据依赖我们通过增加一个设计领域“设备指令数据存储”来解耦. 其中, 改进的原始需求产生的指令数据存储在“设备指令数据存储”, 而新的需求将从中读取指令, 由此将其转换为信号. “设备指令数据存储”内部根据优先级对指令进行排序, 保证每次取出的指令是当前所有指令中优先级最高的.

例如, 在图 5(a) 中, “设备 3 加电 (Device3 power-on)”需求对设备 3 (Device3) 进行加电操作, “设备 3 断电 (Device3 power-off)”需求对设备 3 进行断电操作. 情景图 (图 5(b)、图 5(c)) 表示实现这两个需求的交互关系.“设备 3 加电”与“设备 3 断电”具有相同优先级. 针对这种情形, 引入“设备 3 控制输出 (D3 control output)”需求, 以及“设备 3 指令 (D3 instruction)”数据存储领域, 其中“设备 3 控制输出”要求将指令转换为信号, “设备 3 指令”领域用于存储当前要转换的指令. 这样得到新问题图如图 5(d) 所示. 原“设备 3 加电”需求由“存储设备 3 加电指令”需求和“设备 3 控制输出”需求实现. 类似地, 原“设备 3 断电”需求由“存储设备 3 断电指令”需求和“设备 3 控制输出”需求实现. 得到 3 个新的情景图, 其中图 5(e) 实现“设备 3 控制输出”, 图 5(f) 实现“存储设备 3 加电指令”, 图 5(g) 实现“存储设备 3 断电指令”.

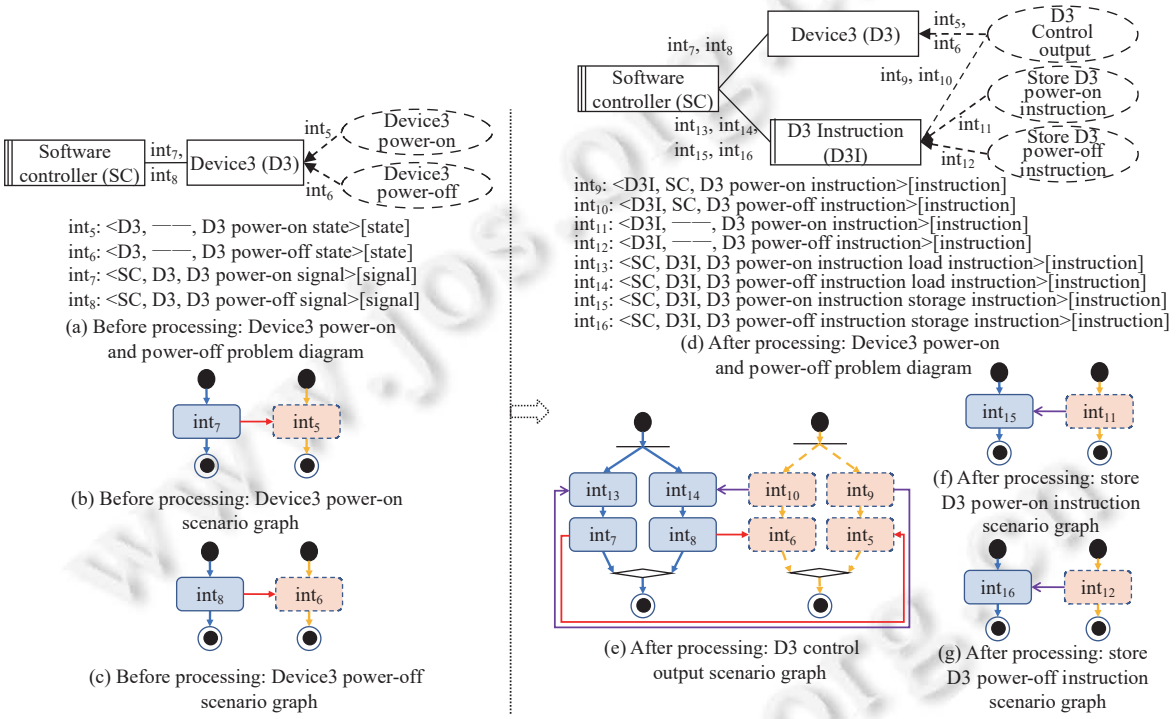


图 5　应用解耦策略 2 后设备加断电操作融合例子的结果

该策略可以形式地表示如下.

解耦策略 2: 对于当前问题图 $PD=<CS,DS,BI,RS,EI,RC>$ 和情景图 $SD_i=<BI_i,EI_i,CN_i,ES_i>$, 其中 $RS=\{rs_1,rs_2,\ldots,rs_n\}$, $1\leqslant i\leqslant n$, $BI=\bigcup_{i=1}^{n}BI_i$, $EI=\bigcup_{i=1}^{n}EI_i$. 若 PD 中包含两个需求 $rs_j,rs_k\in RS$ $(1\leqslant j\leqslant n, 1\leqslant k\leqslant n, 且 j\neq k)$ 要求对同一设备 dv 发送不同信号 (为简化叙述, 这里假设只有两个需求, 若有多个需求可以同理进行), 则进行如下操作.

1) 增加控制输出需求 rs 和设备指令数据存储领域 ds, 即 $RS=RS\cup\{rs\}$, $DS=DS\cup\{ds\}$, 其中 ds 负责存储这两个需求对设备的操作指令, rs 从 ds 取指, 将其转化为信号类型后发给设备;

2) 建立控制输出需求 rs 与设备 dv 和设备指令数据存储领域 ds 间的需求引用或约束, 去掉原需求 rs_j 和 rs_k 与设备 dv 间的需求引用或约束;

3) 修改原需求 rs_j 和 rs_k 为 rs'_j 和 rs'_k, 分别负责存储相应的指令, 并建立 rs'_j 和 rs'_k 与设备指令数据存储领域 ds 间的需求引用或约束;

4) 为控制输出需求 rs 创建其所属的情景图 SD_{rs}, 并修改 rs_j 和 rs_k 之前的情景图, 以实现 rs'_j 和 rs'_k.

策略 2 能够为每个设备提供一种设备操作的抽象接口, 它将其他问题需求中关于设备的操作统一为指令形式, 使得其他需求不需要关注与设备交互的细节, 便于将与设备交互的多种操作细节放在同一问题中进行统一处理, 这将增加了设备操作问题的内聚性, 同时也降低了设备操作中指令与信号生成的耦合度.

● 策略 3: 数据生产消费分离模式

在嵌入式系统中, 从各种设备获得的数据存在着被多个需求使用的情况, 在数据的获取与使用之间存在着直接的数据依赖. 为了解耦这种直接的数据依赖, 我们定义了解耦策略 3. 该策略依据生产者消费者模型, 在数据的获取与使用之间增加“数据存储”这一设计领域, 使得生产者产生的数据可以存储其中, 在消费时可以从中直接读取. 由此分离了数据的生产与消费, 降低了它们之间的耦合度, 也增加了内聚性.

例如, 在图 6(a) 中, “设备 4 数据生产 (Device4 data production)”需求从设备 4 (Device4) 采集数据, “设备 4 数据消费 (Device4 data consumption)”需求使用设备 4 数据, 并根据设备 4 数据的内容来判断设备 5 (Device5) 是否需要加电. 图 6(b)、图 6(c) 是这两个需求对应的情景图. 对这种情况, 可以增加“设备 4 数据 (D4 data)”数据存储领域, 原“设备 4 数据生产”“设备 4 数据消费”需求变为“设备 4 数据生产 2 (Device4 data production 2)”“设备 4 数据消费 2 (Device4 data consumption 2)”需求. “设备 4 数据生产 2”和“设备 4 数据消费 2”需求共享这个“设备 4 数据 (D4 data)”数据存储领域, 如图 6(d) 所示. 原情景图也需要修改, 结果如图 6(e) 与图 6(f) 所示.

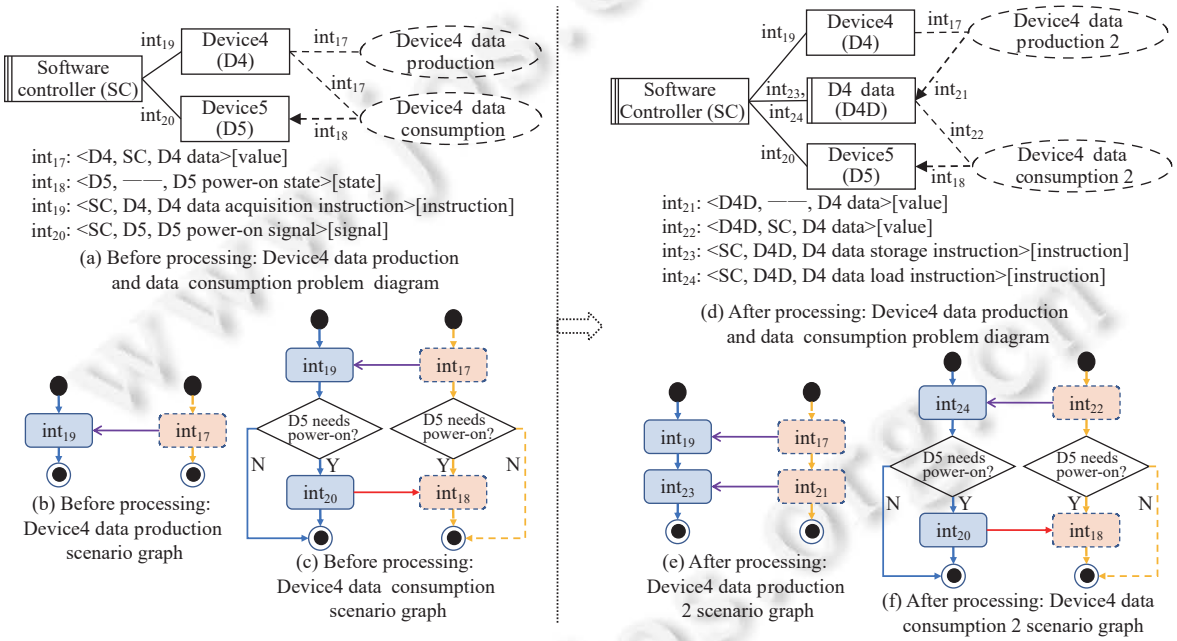


图 6 应用解耦策略 3 后设备数据生产消费例子的结果

该策略可以形式地表示如下.

解耦策略 3: 对于当前的问题图 $PD=<CS,DS,BI,RS,EI,RC>$ 和情景图 $SD_i=<BI_i,EI_i,CN_i,ES_i>$, 其中 $RS=\{rs_1,rs_2,\ldots,rs_n\}$, $1\leqslant i\leqslant n$, $BI=\bigcup_{i=1}^{n}BI_i$, $EI=\bigcup_{i=1}^{n}EI_i$. 若 PD 中存在需求 $rs_j,rs_k\in RS$ ($1\leqslant j\leqslant n$, $1\leqslant k\leqslant n$, 且 $j\neq k$), rs_j 产生数据 (data)、rs_k 需要使用这些数据, 则进行如下操作.

1) 增加数据存储领域 ds, 即 $DS=DS\cup\{ds\}$, 将需求 rs_j 和 rs_k 更新为 rs'_j 和 rs'_k, 即 $RS=RS\setminus\{rs_j,rs_k\}\cup\{rs'_j,rs'_k\}$, 使 ds 负责存储 rs'_j 产生的数据 (data), rs'_k 则从 ds 读取数据 (data);

2) 从 BI 中确定数据 (data) 的来源设备 dv, 在 RC 中去掉 dv 与需求 rs'_k 的需求引用或约束, 建立从 rs'_j 和 rs'_k 到 ds 的需求引用或约束, 识别其期望交互, 加入到 EI 中;

3) 在问题图 PD 增加软件与 ds 的行为交互, 修改实现 rs_j 和 rs_k 对应的情景图, 以实现新需求 rs'_j 和 rs'_k.

● 策略 4: 复杂计算级联分解模式

在嵌入式系统中, 存在着很多复杂的数据处理需求. 为了降低计算复杂度与支持中间结果复用, 我们定义了解耦策略 4. 该策略将可分步计算的数据处理需求按计算步骤进行分解, 增加数据存储领域存放中间结果, 将复杂计算需求分解为简单计算需求. 它将一组复杂计算划分为多个子步骤, 并保存每个子步骤的计算结果. 当存在其他问题需要使用子步骤的计算结果时, 无需再次重新计算, 直接复用已保存的子步骤计算结果即可. 策略 4 可以用于消除不同问题中的重复计算, 有效增加内聚性, 支持子问题的复用.

例如, 在图 7(a) 中, 计算 (computing) 需求要求从设备 6 数据 (D6 data) 和设备 7 数据 (D7 data) 取得数据, 完成计算后将结果保存至计算结果 (computing result) 领域, 图 7(b) 是实现该需求的情景图. 根据领域专家提供的信息, 计算需求包括两个子功能: 子计算 1 (Subcomputing 1)、子计算 2 (Subcomputing 2), 并且二者满足顺序关系. 因此, 软件可以先计算子计算 1, 然后将结果交给计算子计算 2 即可得到最终结果. 将计算 (computing) 需求分解为子需求子计算 1 与子计算 2, 并新增数据存储领域子计算 1 结果 (Subcomputing 1 result) 用于存储子计算 1 的计算结果. 其问题图如图 7(c) 所示, 其中, 两个子需求共享子计算 1 结果领域. 两个子需求的情景图分别如图 7(d) 与图 7(e) 所示. 子需求子计算 1 结果从设备 6 数据取数据计算, 将结果存储至子计算 1 结果, 然后子需求子计算 2 从领域子计算 1 结果和设备 7 数据中取数据, 完成计算后将结果存储至计算结果 (computing result).

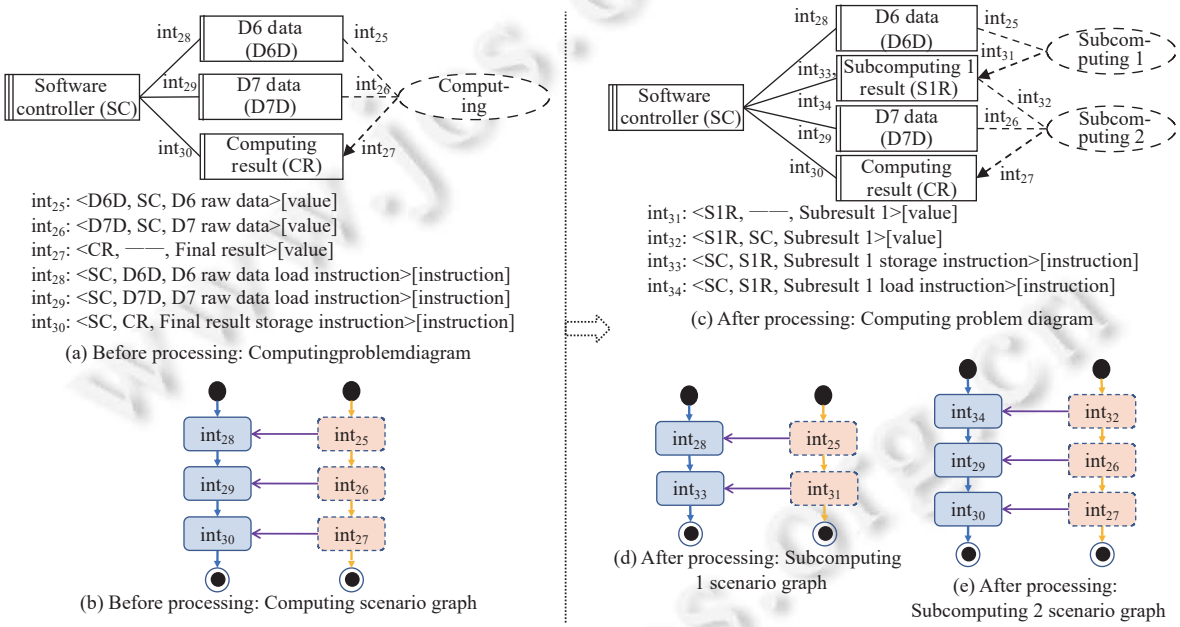


图 7 应用解耦策略 4 后数据计算例子的结果

该策略可以形式地表示如下.

解耦策略 4: 对于当前的问题图 $PD=<CS,DS,BI,RS,EI,RC>$ 和情景图 $SD_i=<BI_i,EI_i,CN_i,ES_i>$, 其中 $RS=\{rs_1,rs_2,\ldots,rs_n\}$, $1\leqslant i\leqslant n$, $BI=\bigcup_{i=1}^{n}BI_i$, $EI=\bigcup_{i=1}^{n}EI_i$. 若存在 $rs_i\in RS$, 其需要完成的计算任务可按步骤分解, 则进行如下操作.

1) 按照计算子功能分步信息将 rs_i 分解为若干个子计算需求 $rs_{i1},\ldots,rs_{im}$, 即 $RS=RS\setminus\{rs_i\}\cup\{rs_{i1},\ldots,rs_{im}\}$;

2) 增加数据存储领域 $ds_1,\ldots,ds_{m-1}$, 分别存储前 $m-1$ 个子需求的计算结果, 即 $DS=DS\cup\{ds_1,\ldots,ds_{m-1}\}$; 在 RC 中修改与 $rs_{i1},\ldots,rs_{im}$ 相关的需求引用和需求约束;

3) 修改 PD 的 BI 与 EI 中相应的行为交互与期望交互, 并为 $rs_{i1},\ldots,rs_{im}$ 创建对应的情景图.

● 策略 5: 控制-计算分离模式

在嵌入式系统中, 通常存在着控制需求和计算需求并存的情况. 当从设备或者数据存储中读取数据进行计算并以此控制设备时, 如果数据的计算特别复杂, 可以增加计算需求, 将控制和计算分离. 控制仅需要使用计算部分的结果, 计算内部操作与控制内部操作相关性并不高, 策略 5 将二者分开, 有助于提高每个问题内操作的相关程度, 进而提高内聚性.

例如, 图 8(a) 中, 设备 9 控制 (Device9 control) 需求将从设备 8 (Device8) 采集数据, 根据采到的数据生成设备 9 (Device9) 的控制信号用以控制设备 9, 图 8(b) 给出了其情景图. 根据策略 5, 若根据采到的数据生成设备控制信号的计算很复杂, 则增加新的需求设备 9 控制计算 (D9 control computing), 专门负责计算, 而将控制的需求留给设备 9 控制 2 (Device9 control 2), 二者联合起来完成原来的需求设备 9 控制. 具体的问题图如图 8(c) 所示, 这两个子需求共享设备 9 计算结果 (D9 computing result) 这一设计领域, 通过它解耦了数据依赖. 原来的情景图也改为这两个子需求的情景图 (图 8(d) 与图 8(e)).

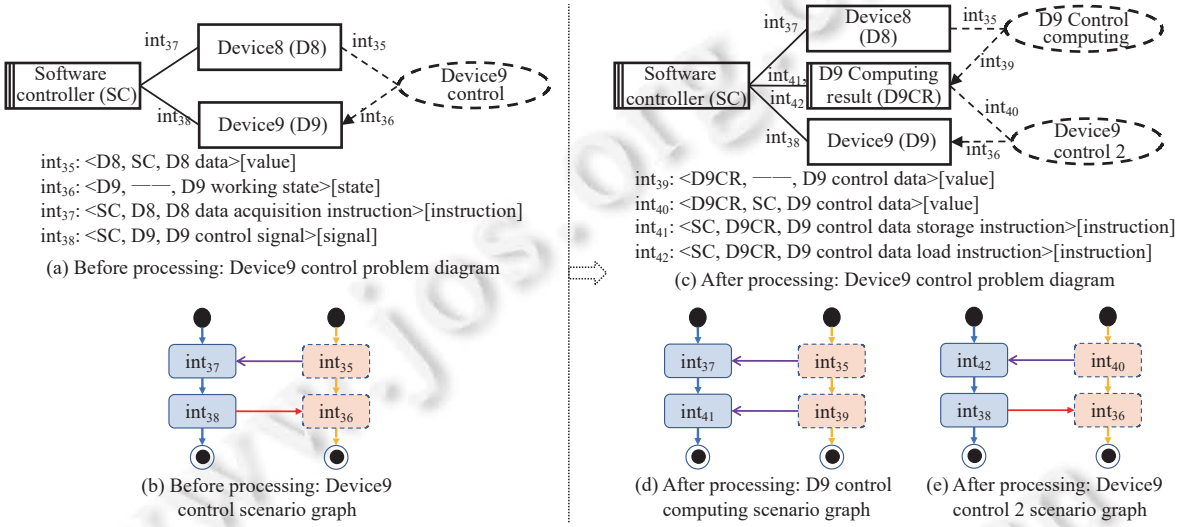


图 8 应用解耦策略 5 后设备控制例子的结果

该策略可以形式地表示如下.

解耦策略 5: 对于当前的问题图 $PD=<CS,DS,BI,RS,EI,RC>$ 和情景图 $SD_i=<BI_i,EI_i,CN_i,ES_i>$, 其中 $RS=\{rs_1,rs_2,\ldots,rs_n\}$, $1\leqslant i\leqslant n$, $BI=\bigcup_{i=1}^{n}BI_i$, $EI=\bigcup_{i=1}^{n}EI_i$. 若存在 $rs_i\in RS$, rs_i 产生数据 (data) 并且该需求使用这些数据控制设备, 则进行如下操作.

1) 增加数据存储领域 ds, 即 $DS=DS\cup\{ds\}$, 增加控制计算需求 rs_{i1} 和设备控制需求 rs_{i2}, 删除原需求 rs_i, 即 $RS=RS\setminus\{rs_i\}\cup\{rs_{i1},rs_{i2}\}$, 其中 rs_{i1} 负责采集数据, 并将计算得到的设备控制信息保存至数据存储领域 ds, rs_{i2} 从 ds 取数据, 将设备控制信息转化为设备控制信号后发送给设备;

2) 为控制计算需求 rs_{i1} 和设备控制需求 rs_{i2} 分别创建其所属的情景图 $SD_{rs_{i1}}$ 和 $SD_{rs_{i2}}$.

需要指出的是, 在这 5 个策略中, 策略 1、2、3、5 都是包含设备操作的解耦策略, 为针对嵌入式软件系统专属的需求解耦策略. 而策略 4 仅针对数据计算, 不涉及设备, 属于通用的解耦策略. 但由于现在的模型并不包含设备专有信息, 如通信协议、端口等, 目前的解耦策略中的设备其实表达了现象之间具有明确因果关系的问题领域 (在问题框架方法中称为因果领域), 从这种意义上来讲, 本文的 5 个策略也可以全部隶属于通用解耦策略.

5 软件需求规约过程

软件需求规约是根据系统需求所涉及的期望交互, 以及需求实现所基于的设备特性, 推导使能设备行为的软件行为交互的过程[10]. 本节定义了针对嵌入式软件系统的软件需求规约过程, 如图 9 所示.

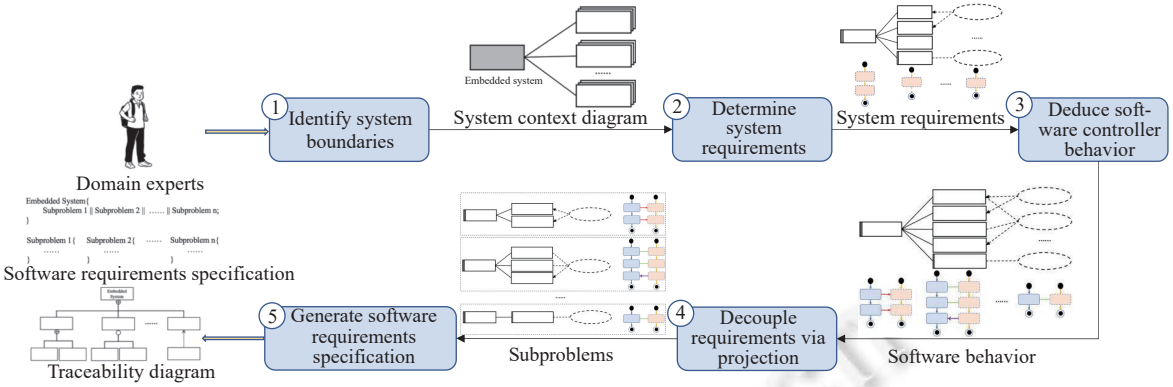


图 9 软件需求规约过程示意图

● 步骤 1: 识别系统边界

识别系统边界就是确定待设计的嵌入式系统需要协调或管理哪些外部设备, 包括如下步骤.

1) 标识嵌入式系统: 确定并命名将要构造的嵌入式系统软件.

2) 识别外部实体: 外部实体是物理世界中未来将与嵌入式系统发生交互的实体, 比如操作员、被监视的场景或设备以及被控制的设备或能调节的现实世界参数等.

3) 确定交互内容: 嵌入式系统与其外部实体间可能的共享现象.

4) 识别系统设备: 根据与外部环境实体的交互内容, 推断可能与软件直接交互的设备.

可以用上下文图来表达上面获得的信息.

● 步骤 2: 确定系统需求

嵌入式系统需求的描述包含如下步骤.

1) 识别系统需求并确定期望交互: 领域专家一般用简单自然语言描述系统需求, 从系统需求中识别出对直接交互设备的期望交互, 并建立需求和交互设备的需求引用.

2) 区分需求引用/约束: 对于每个需求引用, 识别系统需求对每个交互领域的期望交互的作用, 即需求引用/约束, 识别规则如下.

① 对时钟、感应器、作用器等设备领域, 若期望交互表达软件使能该设备进行状态变迁, 则为需求约束, 否则为需求引用.

② 对于接入操作员的设备, 若期望交互表达软件要向其发送数值/信号, 则为需求约束, 否则为需求引用.

3) 识别期望交互序关系: 需求相关者根据控制逻辑, 确定期望交互发生的先后顺序. 每个系统需求对应一个包含与其相关所有期望交互的独立的交互序列. n 条系统需求对应 n 个期望交互序列.

至此得到待开发软件的初始问题图和情景图.

● 步骤 3: 推断软件行为

本步骤解决软件行为确定问题. 领域特性将作为软件行为与期望交互之间的桥梁[22], 可以采取如下 3 个步骤来推断软件行为.

1) 确定行为交互: 根据期望交互和参与交互的领域的特性, 确定软件的行为交互, 有如下 4 种情况.

① 若期望交互表达设备状态变迁, 则需要软件向设备发送状态变迁信号/指令, 该交互为信号型/指令型交互. 根据设备特性, 可以进一步区分是信号型交互还是指令型交互.

② 若期望交互要求设备向软件发送数据, 则需要软件向设备发送数据采集指令, 为指令型交互.

③ 若期望交互表达向接入操作员设备发送数值, 则需要软件向该设备发数据发送指令, 为指令型交互.

④ 若不符合上述情况, 则保留软件和外部实体之间期望交互同样的内容作为行为交互.

2) 建立行为交互与期望交互间的关系: 由领域特性和交互类型决定.

① 若行为交互使能设备领域发生状态变迁, 则为行为使能关系.

② 若行为交互使能向接入操作员设备发数据或行为交互使能设备向软件发送数据, 则为需求期望关系.

③ 若行为交互和期望交互相同, 则为同步关系.

3) 确定行为交互序关系: 行为序关系的确定有两种方式, 一是根据与其对应的期望交互的序关系决定行为交互序关系, 二是领域专家根据任务控制逻辑确定.

至此得到问题图中所涉及交互 (期望交互和行为交互) 之间的关系.

● 步骤 4: 通过投影解耦软件需求

在介绍投影之前, 需要进行情景的良构性[11]检测, 以保证投影后子问题的完整性.

1) 良构情景检测: 对情景图进行良构性检测, 对非良构的情景图需要根据需求进行修改, 包括对不符合语法、不符合领域特性的问题图和情景图进行调整等.

2) 问题解耦策略应用: 依次用本文第 4 节中给出的问题解耦策略, 对问题图和情景图进行更新, 直到解耦得到的子需求的粒度达到软件软件设计的复杂度约减要求.

3) 进行投影: 基于文献 [11] 中的定义, 对基于情景图的问题图进行投影, 得到一组子问题图及其对应的情景图, 构成一组 (子) 软件.

在应用第 4 节的问题解耦策略对需求进行解耦时会逐步产生若干新需求, 造成了不同的需求层次, 形成可追踪关系. 这些追溯关系包括: 完全包含关系 (⊕)、部分包含关系 (O)、满足关系 (<satisfy>)、导出关系 (<derive>) 和踪迹关系 (<trace>). 其中, 完全包含关系表达了上层需求完全由下层需求描述; 而部分包含关系则表示上一层需求包含了部分下一层需求, 还有一部分需求它自己实现; 满足关系则表明上层需求可以由下层需求进行实现; 导出关系则表达了下层需求是由上层需求精化的结果; 踪迹关系则表达下层需求与上层需求有关系, 比较弱.

这些追溯关系的得来与解耦的策略有密切关系. 假设有需求 R_1, 使用解耦策略可以解耦为需求 R_2,R_3 或 R'. 若为策略 1, 则 $R_1\oplus(R_2,R_3)$ 或 $R_1\mathrm{O}(R_2,R_3)$; 若为策略 2, 则 $R_1<\mathrm{satisfy}>(R_2,R_3)$; 若为策略 3, 则 $R_1<\mathrm{derive}>R'$; 若为策略 4, 则 $R_1\oplus(R_2,R_3)$ 或 $R_1\mathrm{O}(R_2,R_3)$; 若为策略 5, 则 $R_1\oplus(R_2,R_3)$.

除了需求追溯关系, 各个需求之间还具有依赖关系. 根据其依赖的对象不同, 本文将依赖关系分为两类: 数据依赖和控制依赖. 其中数据依赖是指两个问题图共享同一个数据存储, 一个需求存储数据, 另一个需求读取数据, 如图 5 中的需求存储设备 3 加电指令 (store D3 power-on instruction) 和需求设备 3 控制输出 (D3 control output). 控制依赖则是指两个问题图共享同一个设备, 需求 rs_2 控制设备进入某状态 S, 而需求 rs_1 需要设备处于 S 状态才能计算, 如图 6 中的需求设备 4 数据生产 (Device4 data production), 它必须在需求设备 4 加电 (Device4 power-on) 对设备 4 (Device4) 加电后才能工作.

这些需求的数据依赖的形成与某些投影策略有密切关系. 策略 3 数据生产消费分离模式中, 数据生产者创建数据存储, 而消费者则从中读取数据. 策略 4 复杂计算级联分解模式通过增加数据存储领域存储中间结果供下一步使用, 也将产生数据依赖. 策略 5 控制-计算分离模式中, 计算需求产生的计算结果将会提供给控制需求使用. 这都将形成数据依赖. 而控制依赖的形成则与设备特性密切相关, 需要根据设备特性进行识别.

● 步骤 5: 生成软件需求规约

在解耦过程中, 会形成不同层次的需求, 为了精确地描述软件需求规约, 我们将软件需求规约分为两部分: 概述部分和详细描述部分. 概述部分描述子需求及其之间的关系, 子需求关系可根据需求间的追踪关系、需求依赖关系和领域特性推理得到. 详细描述部分给出每个解耦后的原子软件子需求涉及的软件行为交互及其序关系. 软件需求规约获取包含如下步骤.

1) 生成软件需求规约概述部分: 首先根据追溯关系获取软件需求规约概述部分的整体结构, 这是一种层级结构. 具体而言, 若 A 完全包含 B,C, 则 A 为 B,C 的上一层级, 即“$A\{B;C;\}$”; 若 A 部分包含 B,C, 令 A' 表示 A 中除去 B 和 C 后的剩余内容, 则 A',B,C 位于同一层级, 即“$A';B;C;$”; 若 B,C 满足 A, 则 A 为 B,C 的上一层级, 即“$A\{B;C;\}$”; 若由 A 导出 $A2$, 则直接使用 $A2$ 代替 A; 对于踪迹关系, 不做处理.

得到概述部分的结构之后, 使用依赖关系推导同一层级各个需求的顺序关系. 具体而言, 若 B 数据或控制依赖

于 A, 则在软件需求规约中, A 应在 B 之前, A 和 B 的上层需求仍然需要满足这一约束; 若 A 和 B 不具备依赖关系, 在单核处理器环境下使用随机方法进行排序, 在多核环境下用并行关系表示. 最后, 可以人工对软件需求规约概述部分进行约束, 如增加时间约束.

2) 生成软件需求规约详细描述部分: 详细描述部分给出每个软件子需求涉及的软件行为交互及其序关系. 对于软件发起的交互, 如$<\text{int}_{19}, \text{Software Controller}, \text{Device4}, \text{D4data acquisition instruction}, \text{instruction}>$, 我们可以将其转化为“Software Controller sends D4 data acquisition instruction to Device4”行为交互语句; 对于软件接收的交互, 如$<\text{int}_{17}, \text{Device4}, \text{Software Controller}, \text{D4data}, \text{value}>$, 我们可以将其转化为“Software Controller receives D4 data from Device4”行为交互语句. 软件行为交互的序关系遵循行为交互中的序关系, 包括顺序结构、分支结构、循环结构、并行结构、合并结构, 分别采用“;”“if(branching condition) {...} else {...}”“while(loop condition) {...} ”“||”以及“|”表示. 最后也可以人工增加时间约束.

6 案例研究与评估

本节使用航天太阳搜索控制软件子系统 (简称太阳搜索子系统) 进行案例研究, 并通过系统耦合度和内聚性两个指标在 5 个典型应用案例场景来对本文方法进行有效性评估, 最后讨论本文方法的优缺点.

6.1 案例研究

太阳搜索子系统案例由北京控制工程研究所提供, 太阳搜索子系统需要感知太阳位置和当前自身星体角度, 并改变自身星体角以实现对日巡航. 系统由 32 ms 中断定时器驱动, 从陀螺、太阳敏感器和三轴控制推力器收集数据, 通过数据管理计算机接收地面指令, 自动控制推力器喷气, 从而使卫星朝向太阳旋转.

● 步骤 1: 识别系统边界. 首先, 将待构造的嵌入式软件系统命名为太阳搜索子系统 (sun search subsystem). 然后从领域专家给出的系统描述中, 识别其外部实体以及与这些实体的交互. 比如, 根据描述“太阳搜索子系统需要确定当前卫星相对太阳的姿态, 控制卫星姿态进行太阳捕获, 以实现对日巡航”, 可以提取两个与系统交互的外部实体, “太阳 (sun)”和“卫星 (satellite)”, 得到“太阳搜索子系统感知太阳位置与卫星姿态、太阳搜索子系统改变卫星姿态”的交互内容. 根据描述“数据管理计算机接收地面命令并将遥控指令发送至软件”, 可以抽取实体“地面操作员 (ground operator)”和“数据管理计算机 (data management computer)”以及太阳搜索系统和它们的交互.

其次, 根据环境实体能力, 选择可能的交互设备. 例如, 由于太阳搜索子系统需要获取太阳可见标志和太阳测量角度, 可以选择“太阳敏感器 (sun sensor)”设备作为太阳搜索子系统的设备. 类似的, 选择“陀螺 (gyro)”“三轴控制推力器 (triaxial control thruster)”和“32 ms 中断定时器 (32 ms interrupt timer)”作为其他一些交互的设备等. 这样, 构造出太阳搜索子系统的以“软件 (software controller)”为中心的上下文图, 如图 10 所示.

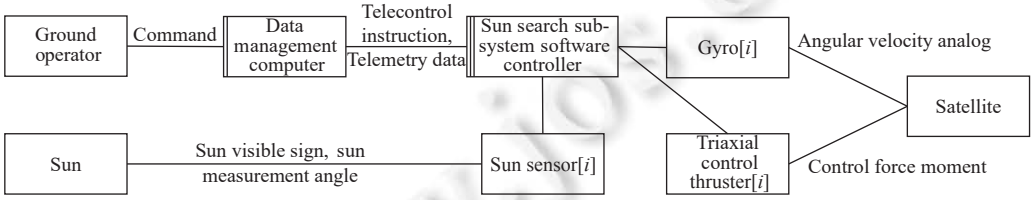


图 10 太阳搜索子系统上下文图

● 步骤 2: 确定系统需求. 先根据领域专家的描述识别系统需求, 包括“数据采集 (data acquisition)”“遥控指令处理 (telecontrol instruction processing)”“推力器三轴控制 (thruster triaxial control)”“遥测处理 (telemetry processing)”“初始化 (initialization)”和“故障诊断处理 (fault diagnosis processing)”.

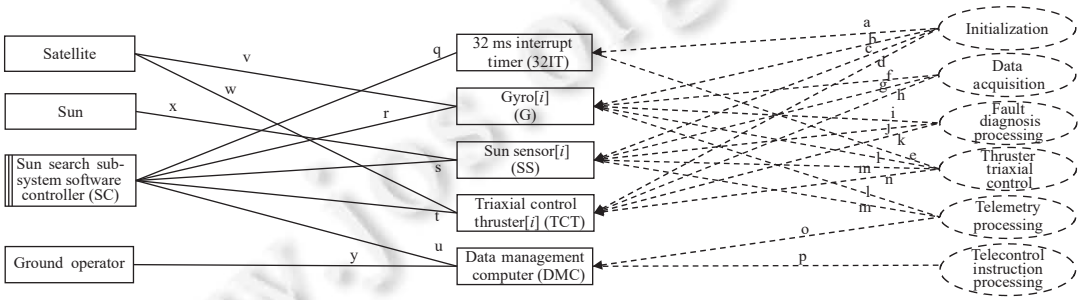
然后识别期望交互, 一个系统需求可能涉及若干个期望交互. 以故障诊断处理需求为例, 故障诊断处理需求要求当陀螺、太阳敏感器、三轴控制推力器出现故障时, 对相应的设备进行断电处理. 该系统需求将要求当陀螺故

障时, 出现陀螺断电状态 (gyro power-off state) 期望交互, 对其他需求进行相同的操作识别其期望交互.

接下来确定需求与期望交互之间是引用还是约束. 故障诊断处理需求包含期望交互“陀螺断电状态”, 表达了对陀螺预期状态的约束, 因此故障诊断处理需求约束该期望交互的发生; 遥控指令处理需求要求软件接收数据管理计算机发送的指令, 软件只能被动观察遥控指令现象的发生, 因此遥控指令处理需求引用该期望交互.

最后识别每个系统需求的期望交互间的序关系. 领域专家可以根据任务要求的控制逻辑来标注期望交互发生的先后顺序, 例如在故障诊断处理需求中, 太阳敏感器向软件发送“太阳可见标志”、若不可见则要求太阳敏感器处于“断电状态”, 这两个期望交互之间具有明显的先后关系, 并在其中存在判断关系, 可以画出其序关系. 类似地, 针对其他 5 个系统需求, 可以得到 5 个期望交互序列.

● 步骤 3: 推断软件行为. 首先, 针对每个期望交互, 推导其对应的软件行为, 建立期望交互与行为交互间的关系. 针对故障诊断处理需求中的“陀螺断电状态”期望交互, 根据设备陀螺的性质, 需要接收“陀螺断电信号”才能满足该期望交互. 因此, 可以推断出行为交互, 软件发送“陀螺断电信号”. 这样的期望交互与行为交互间关系是因为陀螺的性质而存在, 因此, 行为交互使能了期望交互, 属于行为使能关系. 类似地, 可以推导出其他期望交互的行为交互及其与期望交互关系. 结合步骤 2, 可以得到太阳搜索子系统的问题图, 如图 11 所示.



a: {int_1} b: {int_2} c: {int_3} d: {int_4} e: {int_5} f: {int_6, int_7} g: {int_8, int_9, int_{10}} h: {int_{11}, int_{12}} i: {int_{13}} j: {int_8, int_{14}} k: {int_{11}, int_{15}}
l: {int_6} m: {int_8, int_9} n: {int_{16}} o: {int_{17}} p: {int_{18}} q: {int_5, int_{19}} r: {int_{20}, int_{21}, int_{22}, int_{23}} s: {int_{24}, int_{25}, int_{26}, int_{27}, int_{28}}
t: {int_{29}, int_{30}, int_{31}, int_{32}, int_{33}} u: {int_{18}, int_{34}} v: {Angular velocity analog} w: {Control force moment} x: {Sun visible sign, Sun measurement angle} y: {Command}

int_1: <32IT, ——, 32ms interrupt timer start state>[state]
int_2: <G, ——, Gyro power-on state>[state]
int_3: <SS, ——, Sun sensor power-on state>[state]
int_4: <TCT, ——, Thruster power-on state>[state]
int_5: <32IT, SC, Interrupt signal>[signal]
int_6: <G, SC, Pulse-count>[value]
int_7: <G, SC, Gyro power state>[state]
int_8: <SS, SC, Sun visible sign>[value]
int_9: <SS, SC, Sun measurement angle>[value]
int_{10}: <SS, SC, Sun sensor power state>[state]
int_{11}: <TCT, SC, Jet interval>[value]
int_{12}: <TCT, SC, Thruster power state>[state]
int_{13}: <G, ——, Gyro power-off state>[state]
int_{14}: <SS, ——, Sun sensor power-off state>[state]
int_{15}: <TCT, ——, Thruster power-off state>[state]
int_{16}: <TCT, ——, Thruster jet state>[state]
int_{17}: <DMC, ——, Telemetry data>[value]
int_{18}: <DMC, SC, Telecontrol instruction>[instruction]
int_{19}: <SC, 32IT, 32ms interrupt timer start instruction>[instruction]
int_{20}: <SC, G, Pulse-count acquisition instruction >[instruction]
int_{21}: <SC, G, Gyro power state perception instruction>[instruction]
int_{22}: <SC, G, Gyro power-on signal>[signal]
int_{24}: <SC, SS, Sun visible sign acquisition instruction>[instruction]
int_{23}: <SC, G, Gyro power-off signal>[signal]
int_{25}: <SC, SS, Sun measurement angle acquisition instruction>[instruction]
int_{26}: <SC, SS, Sun sensor power state perception instruction>[instruction]
int_{27}: <SC, SS, Sun sensor power-on signal>[signal]
int_{28}: <SC, SS, Sun sensor power-off signal>[signal]
int_{29}: <SC, TCT, Jet interval acquisition instruction>[instruction]
int_{30}: <SC, TCT, Thruster power state perception instruction>[instruction]
int_{31}: <SC, TCT, Thruster power-on signal>[signal]
int_{32}: <SC, TCT, Thruster power-off signal>[signal]
int_{33}: <SC, TCT, Triaxial control signal>[signal]
int_{34}: <SC, DMC, Telemetry data transmission instruction>[instruction]

图 11 太阳搜索子系统-问题图

最后, 确立行为交互间序关系. 在故障诊断处理需求中, 有两个期望交互, 即太阳敏感器向软件发送太阳可见标志、太阳敏感器进入断电状态. 它们与两个行为交互之间有需求期望关系和行为使能关系. 这两个期望交互和软件根据“太阳可见标志”判断太阳是否可见的判断节点间存在期望序关系, 则这两个行为交互以及判断节点也满足行为序关系. 可以确定故障诊断需求中其他行为交互序关系, 得到故障诊断需求的情景图, 如图 12 所示. 类似地, 对太阳搜索子系统中的其他需求对应的行为交互建立序列, 获得 5 个交互序列.

● 步骤 4: 通过投影解耦需求. 首先, 对获得的 6 个情景图做良构检测, 无需调整. 其次, 对太阳搜索子系统 6 个原始需求依次重复应用问题解耦策略. 下面以“故障诊断处理”为例, 说明应用解耦策略过程. 初始的故障诊断处理情景图 (图 12) 分别对陀螺、太阳敏感器、推力器进行故障诊断操作, 但它们之间不存在约束关系, 符合策略 1 应用条件. 因此, 根据策略 1, “故障诊断处理”可以解耦为“陀螺故障诊断处理”“太阳敏感器故障诊断处理”“推力器故障诊断处理”, 据此对问题图 12 进行修正, 得到问题图 13(a) 和每个子需求的情景图, 如图 13(b)–图 13(d) 所示.

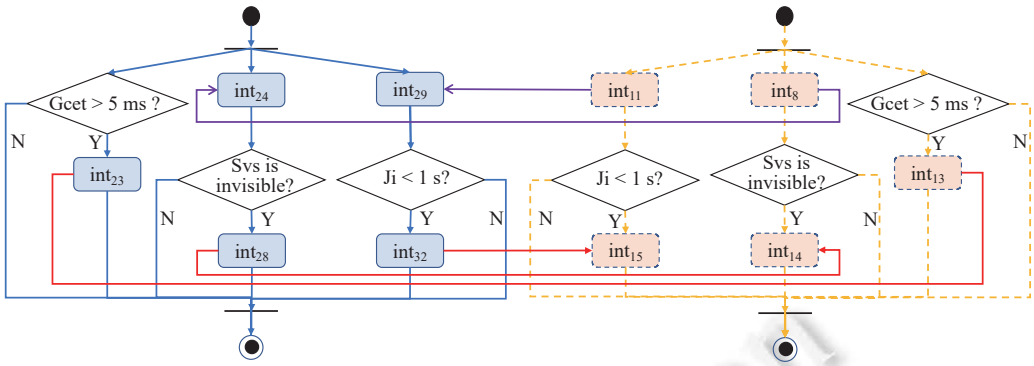


图 12 太阳搜索子系统-故障诊断处理情景图 (Gcet: 陀螺通信错误时间, Svs: 太阳可见标志, Ji: 喷气间隔)

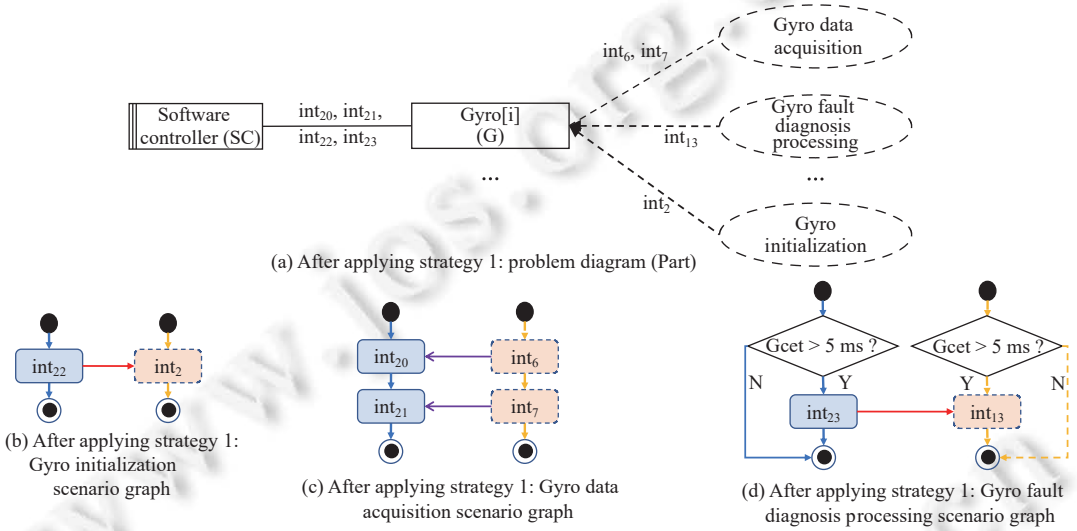


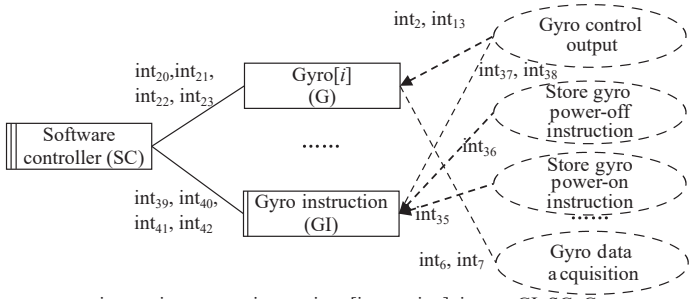
图 13 太阳搜索子系统解耦样例-应用策略 1 后

第三, 检查问题图 13(a) 及其情景图, 发现“陀螺初始化”要求软件向陀螺发送陀螺加电信号、“陀螺故障诊断处理”要求软件向陀螺发送陀螺断电信号, 这两个需求符合策略 2, 即存在多个需求要求软件向同一设备发送多种不同信号的情况. 根据策略 2, 将“陀螺初始化”与“陀螺故障诊断处理”分解为“存储陀螺加电指令”“存储陀螺断电指令”“陀螺控制输出”. 对问题图 13(a) 进行修正, 得到问题图 14(a) 和对应每个子需求的情景 (图 14(b)–图 14(d)).

第四, 检查问题图 14(a) 及其情景图, 发现“陀螺数据采集”“存储陀螺断电指令”需求符合策略 3 生产者消费者模式, “陀螺数据采集”需求产生陀螺数据, “存储陀螺断电指令”需求使用陀螺数据. 根据策略 3, 增加“陀螺数据”领域, 原来的“陀螺数据采集”需求更新为“陀螺数据采集 2”, 原来的“存储陀螺断电指令”需求更新为“存储陀螺断电指令 2”. 对问题图 14(a) 进行修正, 得到问题图 15(a), 以及每个子需求的情景图, 如图 15(b) 和图 15(c) 所示.

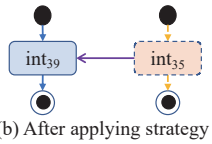
第五, 检查问题图 15(a) 及其情景图, 发现不符合策略 4 应用条件, 但“推力器三轴控制 3”符合控制和复杂计算并存的情况, 即符合策略 5. 根据策略 5, 可以将“推力器三轴控制 3”分解为“控制计算”和“推力器三轴控制 4”. 对问题图 15(a) 进行修正, 得到问题图 16(a), 以及每个子需求的情景图, 如图 16(b) 和图 16(c) 所示.

整个过程用了 4 次策略 1、3 次策略 2、3 次策略 3、1 次策略 4、1 次策略 5. 最后基于 20 个情景图对修正后的问题图进行投影, 得到 20 个子问题图及其对应的情景图. 经过投影解耦之后, 各需求之间的追踪关系共有 7 个完全包含关系、1 个部分包含关系、6 个满足关系、10 个导出关系和 1 个踪迹关系. 此外, 通过分析, 子问题之间有 13 个数据依赖关系、5 个控制依赖关系. 各需求间的追踪关系图、控制依赖关系请参见技术文档 (http://re4cps.org:8027/?pdf=Coupling_Cohesion.pdf).

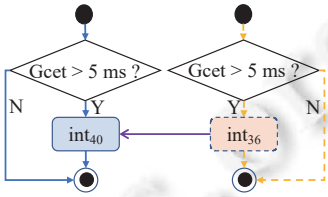


int_{39}: <SC, GI, Gyro power-on instruction storage instruction>[instruction] int_{38}: <GI, SC, Gyro power-off instruction>[instruction]
int_{40}: <SC, GI, Gyro power-off instruction storage instruction>[instruction] int_{37}: <GI, SC, Gyro power-on instruction>[instruction]
int_{41}: <SC, GI, Gyro power-on instruction load instruction>[instruction] int_{36}: <GI, ——, Gyro power-off instruction>[instruction]
int_{42}: <SC, GI, Gyro power-off instruction load instruction>[instruction] int_{35}: <GI, ——, Gyro power-on instruction>[instruction]

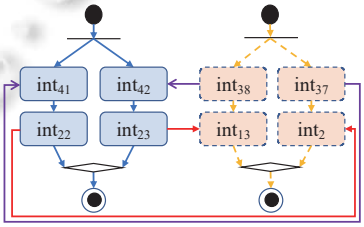
(a) After applying strategy 2: problem diagram (Part)



(b) After applying strategy 2: Store gyro power-on instruction scenario graph

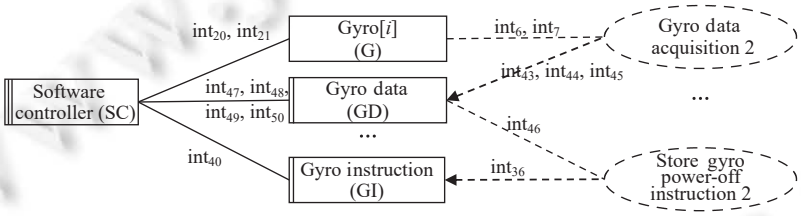


(c) After applying strategy 2: Store gyro power-off instruction scenario graph



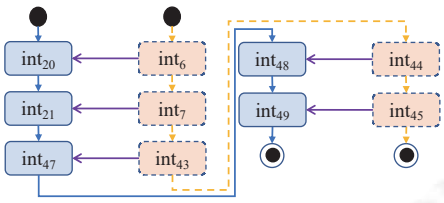
(d) After applying strategy 2: Gyro control output scenario graph

图 14 太阳搜索子系统解耦样例-应用策略 2 后

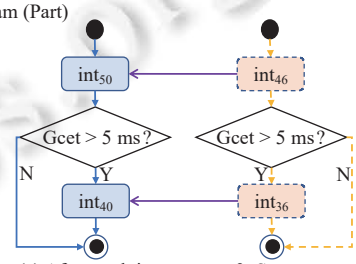


int_{47}: <SC, GD, Angular velocity analog storage instruction>[instruction] int_{45}: <GD, ——, Gyro communication error time>[value]
int_{48}: <SC, GD, Gyro power state storage instruction>[instruction] int_{46}: <GD, SC, Gyro communication error time>[value]
int_{49}: <SC, GD, Gyro communication error time storage instruction>[instruction] int_{43}: <GD, ——, Angular velocity analog>[value]
int_{50}: <SC, GD, Gyro communication error time load instruction>[instruction] int_{44}: <GD, ——, Gyro power state>[state]

(a) After applying strategy 3: problem diagram (Part)



(b) After applying strategy 3: Gyro data acquisition 2 scenario graph

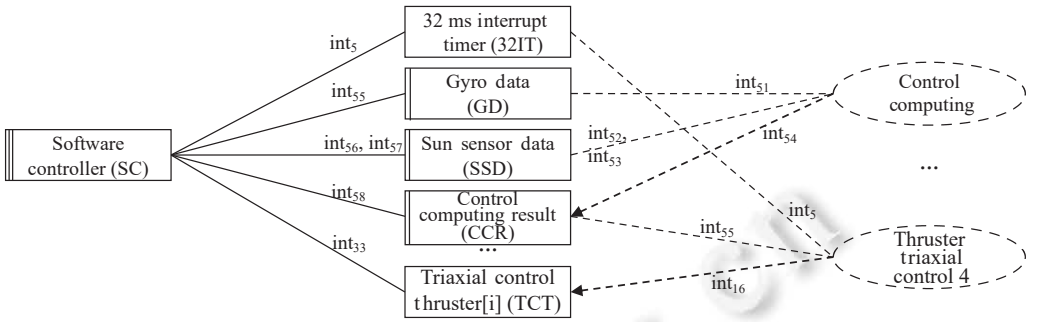


(c) After applying strategy 3: Store gyro power-off instruction 2 scenario graph

图 15 太阳搜索子系统解耦样例-应用策略 3 后

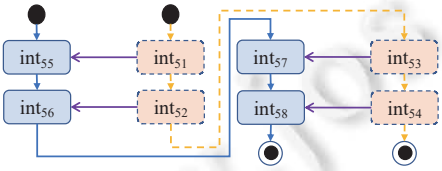
● 步骤 5: 输出软件需求规约. 根据需求追踪关系和依赖关系推导给出软件需求规约概述部分, 如图 17(a) 所示 (图 17(a) 仅给出了一部分的软件需求规约概述, 完整部分参考技术文档). 例如在追溯关系图中, “陀螺初始化”由“存储陀螺加电指令”和“陀螺控制输出”所满足, 而在依赖关系图中, “陀螺控制输出”数据依赖于“存储陀螺加电指令”保存的陀螺加电指令, 因此在“陀螺初始化”的软件需求规约中, “存储陀螺加电指令”发生在“陀螺控制输出”之

前. 类似地, 可以得到概述部分的其他内容. 最后再以人工添加的方式对概述部分进行修缮, 添加部分时间约束, 如 always@(128 ms)、while(160 ms). 对每个子软件需求, 按照规则输出其详细描述. 图 17(b) 给出了存储陀螺断电指令 2 的详细描述.

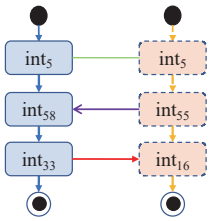


int_{51}: <GD, SC, Angular velocity analog>[value] int_{58}: <SC, CCR, Triaxial control quantity load instruction>[instruction]
int_{53}: <SSD, SC, Sun measurement angle>[value] int_{56}: <SC, SSD, Sun visible sign load instruction>[instruction]
int_{55}: <SC, GD, Angular velocity analog load instruction>[instruction] int_{54}: <CCR, ——, Triaxial control quantity>[value]
int_{57}: <SC, SSD, Sun measurement angle load instruction>[instruction] int_{52}: <SSD, SC, Sun visible sign>[value]

(a) After applying strategy 5: problem diagram (Part)



(b) After applying strategy 5: Control computing scenario graph



(c) After applying strategy 5: Thruster triaxial control4 scenario graph

图 16 太阳搜索子系统解耦样例-应用策略 5 后

我们将解耦后的规约结果交给两位北京控制工程研究所合作者进行确认. 他们首先核对了需求规约的概述部分, 然后又逐条详细核查了每条子软件需求. 他们很惊喜地发现我们解耦后的需求与他们人工靠经验分解的需求是一致的. 这表明解耦后的需求是有效的, 我们的方法是合理的.

6.2 方法评估

为了评估本文方法的有效性, 我们从耦合度和内聚性两个指标出发, 通过对比解耦前后的系统耦合度数值和内聚性数值, 评估解耦是否有效. 系统耦合度计算采用了一种常见的软件模块耦合性度量方法[22]. 该方法将软件模块之间的耦合关系分为 6 种, 并对每种关系进行了数值化表示: 内容耦合 (5)、公共耦合 (4)、控制耦合 (3)、标记耦合 (2)、数据耦合 (1)、无耦合 (0). 其中, 内容耦合指的是模块之间有代码重叠或者一个模块直接访问另一个模块的内部数据; 公共耦合指的是多个模块访问同一个公共数据环境; 控制耦合则是一个模块通过传送开关、标志、名字等控制信息, 明显地控制选择另一模块的功能; 标记耦合是指一组模块通过参数表传递记录信息; 数据耦合则是指一个模块访问另一个模块时通过数据参数 (不是控制参数、公共数据结构或外部变量) 来交换输入、输出信息; 无耦合则是不满足上述 5 种耦合关系. 这 6 种耦合的数值代表了其耦合的程度. 基于这 6 种耦合关系, 该方法给出了两个模块 x 和 y 的耦合度 M 计算公式: $M(x,y)=i+n/(n+1)$. 其中 i 表示 x 与 y 之间最坏耦合关系的数值化表示, 例如, 若 x 和 y 之间最坏耦合关系为公共耦合, 则 $i=4$; n 表示 x 与 y 之间相互连接的数量. 不同类型的耦合, 其 n 取值方法不同: 内容耦合中 n 为 x 和 y 共享的代码段数; 公共耦合中 n 为共享的全局复杂数据结构数量; 控制耦合中 n 为模块功能控制信息数量; 标记耦合中 n 为模块间传递的复杂数据结构数量; 数据耦合中 n 为模块间传递的简单数据的数量; 对于无耦合, n 为模块间存取相同数据的数目. 最终, 取所有模块耦合度的中位数 (median value) 作为系统耦合度.

本文在计算中, 将投影后的子问题 (解耦前、解耦后都可以投影) 视作一个软件模块, 子问题中的软件行为视作代码. 将有相同软件行为的模块记作内容耦合; 将读写同一全局数据视作公共耦合; 将具有控制依赖视作控制耦合; 将以参数的形式传入其他模块复杂数据结构视作标记耦合; 将以参数的形式传入其他模块简单变量视作数据耦合; 将模块间的数据依赖, 即存取相同数据视作无耦合 (这里的存取数据指将数据放入消息队列、从队列中取数, 与公共耦合中读写同一全局数据不同). 根据这些原则, 我们以太阳搜索子系统作为研究对象, 研究其解耦前 (6 个需求) 和解耦后 (20 个需求) 投影后的子问题之间的耦合度. 解耦前共有 6 个内容耦合、1 个公共耦合、19 个控制耦合. 解耦后共有 5 个控制耦合、28 个无耦合. 通过解耦, 内容耦合这种强耦合被消除, 控制耦合的数目也大大减少. 最终可以计算出解耦前系统耦合度为 3.75, 而解耦后只有 0.50, 说明本方法确实有效. 解耦前、解耦后系统的耦合关系图和耦合度计算可参见技术文档 (http://re4cps.org:8027/?pdf=Coupling_Cohesion.pdf).

```
Sun Search Subsystem {
    Initialization;
    while(160 ms) {
        Telecontrol Instruction Processing 2;
        Data Acquisition;
        Attitude Determination;
        Mode Switching Management;
        Control Computing;
        Fault Diagnosis Processing;
        Telemetry Processing 3;
        Thruster Triaxial Control 4;
        always@(128 ms) { Thruster Triaxial Control Output;}
    }
}

Fault Diagnosis Processing {
    Gyro Fault Diagnosis Processing;
    Sun Sensor Fault Diagnosis Processing;
    Thruster Fault Diagnosis Processing;
}

Gyro Fault Diagnosis Processing {
    Store Gyro Power-off Instruction 2;
    Gyro Control Output;
}
```

(a) Partial software requirements specification: Overview

```
Store Gyro Power-off Instruction 2 {
    Software Controller sends Gyro communication error time load instruction to Gyro Data;
    Software Controller receives Gyro communication error time from Gyro Data;
    if (Gyro communication error time > 5 ms) {
        Software Controller sends Gyro power-off instruction storage instruction to Gyro Instruction;
    }
}
```

(b) Software requirements specification: Store gyro power-off Instruction

图 17 部分太阳搜索子系统软件需求规约

由于嵌入式系统的强设备交互性, 其内聚性的计算应该充分考虑软件与设备之间的交互, 因此内聚性的计算采用了一种基于社区网络的软件模块内聚性度量方法[23]. 该方法将软件视为一种具有社区结构的复杂网络, 将每个软件模块视作一个社区, 里面包含了很多函数节点. 属于同一个节点群的多个节点之间的连接应该较密集, 而属于不同节点群之间的节点相互之间的连接应该较稀疏. 它使用模块度指标量化评价节点群社区划分是否合理, 模块度值越高说明划分越合理, 即该网络对应的软件内聚程度越高. 模块度值的计算公式为: $Q=\frac{1}{2m}\sum_{vw}\left[A_{vw}-\frac{k_v k_w}{2m}\right]\delta(c_v,c_w)$, 其中, m 是网络中边的数量, A_{vw} 表示网络的邻接矩阵, $A_{vw}=1$, 表示节点 v 和 w 之间有边, $A_{vw}=0$, 表示节点 v 和 w 之间没有边. k_v 和 k_w 是连接到节点 v 和节点 w 的边数. 若节点 v 和节点 w 在同一个社区, 则 $\delta(c_v,c_w)=1$, 否则 $\delta(c_v,c_w)=0$.

本文计算中, 将投影后的子问题视作一个节点群社区, 子问题中的软件行为视作节点. 当两个软件行为之间存在序关系时, 这两个软件行为对应的节点存在边连接; 当两个节点表示相同的软件行为时, 这两个节点存在边连接. 最终可以计算出解耦前系统模块度值为 0.47, 解耦后为 0.71, 软件内聚程度明显提高. 解耦前、解耦后系统的内聚关系图和模块度计算可同样参见技术文档.

同样地, 我们又将本文方法应用在灯光控制系统、轨道交通车载控制系统、包裹路由系统、XX 嵌入式系统 (保密) 等 4 个例子中 (前 3 个例子可参见网站 http://re4cps.org/examples). 这 4 个例子代表了不同的应用场景. 灯光控制系统包含 2 个原始需求, 分别以命令控制方式和时间驱动自动控制方式控制灯光单元的开关, 系统中存在

多个需求要求软件向灯光单元设备发起多种不同操作的情况. 轨道交通车载控制系统包含 2 个原始需求, 开门 (open doors) 要求软件监听火车信息, 并打开站点门与车门, 关门 (close doors) 要求软件接收驾驶员命令, 关闭站点门与车门, 此系统中站点门和车门设备被要求接收来自软件的开关操作. 包裹路由系统包含 3 个原始需求, 服从操作员的命令 (obey operators command) 要求操作者可以通过按钮控制传送带的启停, 正确路由 (correct routing) 要求计算机软件读取包裹信息、设置包裹走向、正确路由包裹, 报告错误路由包裹 (report misrouted package) 要求软件显示错误路由的包裹信息, 此系统中正确路由和报告错误路由包裹需求均要求软件向包路由器发起多种不同操作, 这两个需求也存在数据生产与消费、复杂计算操作的特征. XX 嵌入式系统具有 4 个原始需求, 此系统具有存在复杂计算操作的特点.

我们使用 5 种问题解耦策略分别对这些案例进行解耦. 然后再分别计算每个例子解耦前后的系统耦合度和模块度, 详细统计结果如表 1 所示. 从表中可以看出, 无论原需求数量多少, 经过了我们的策略解耦, 其系统都经过了进一步的精化, 需求数量增多, 需求耦合度均有较大程度的降低, 模块度标识的内聚性也有明显的提升. 其中耦合度最少降低了原系统耦合度的 60%, 模块度至少提升了 20%. 内聚性和耦合度两个方面的结果充分说明了本文方法的有效性.

表 1　内聚性与耦合度量化评价结果

例子	原需求数量	解耦后需求数量	使用的解耦策略 (编号)	系统耦合度		系统模块度	
				解耦前	解耦后	解耦前	解耦后
灯光控制系统	2	3	2	3.50	0.67	0.17	0.51
车载控制系统	2	4	2	3.67	0.50	0.39	0.65
包裹路由系统	3	6	2, 3, 4	1.75	0.67	0.44	0.53
XX嵌入式系统	4	7	4	1.50	0.50	0.66	0.80
太阳搜索系统	6	20	1, 2, 3, 4, 5	3.75	0.50	0.47	0.71

6.3 讨　论

本文系统地给出了嵌入式系统需求到软件需求的规约过程, 并给出了基于投影的需求解耦. 一方面, 通过解耦和规约, 建立了从系统需求到软件需求的可追踪关系. 随着嵌入式系统需求复杂度的增加以及需求变更的增多, 这种可追踪关系将为后期的维护奠定坚实的基础. 另外一方面, 我们建立了针对嵌入式系统的需求问题解耦策略, 这种策略是客观的, 不以人的意志为转移, 未来有可能自动化, 这将为后续的代码自动生成[24]奠定基础.

值得注意的是, 本文的需求追踪图中存在着 5 种需求追踪关系, 但是这些关系中很多都是投影中的高层需求与底层需求之间的关系, 比如完全包含、满足、导出、踪迹等关系. 这些需求最终在实现时, 仅会实现需求追踪图中的原子需求, 包括叶子节点和部分包含关系中的父需求, 追踪关系表达了原子需求的推导产生过程, 方便领域专家在发现某个需求推导错误时追根溯源、定位问题发生的位置, 它们之间的交互并不会增加需求的复杂性. 相反, 通过解耦, 使得需求的实现变得更简单. 方法评估也说明了这一点.

本文提出的基于投影的嵌入式系统需求分析方法, 其目的就是要为投影提供更完善的投影维度, 使得可以投影出更多的子需求. 方法具有如下风险: 首先, 本文中仅给出了 5 个问题解耦策略. 它们是否完备, 即是否能把所有的需求问题都解耦开? 目前这 5 个解耦策略是针对嵌入式系统中的数据处理与设备操作这两个特征提出的, 未来可以通过考虑嵌入式系统的其他特征, 增加解耦策略. 从我们应用的 5 个例子来看, 5 个解耦策略足够了, 但依然不能排除有其他策略的可能性. 从评估结果上看, 解耦策略确实可以帮助系统完成解耦, 实现内聚性增加和耦合度的降低. 我们相信, 在未来的实际应用中, 这 5 个解耦策略将会帮助领域专家实现系统 (部分) 解耦.

其次, 解耦的最终粒度问题. 也就是说, 解耦到什么时候为止? 在太阳搜索子系统的分解中, 我们发现有很多策略可以使用多遍才停止. 一般来说, 这个粒度只要可以实现就可以, 这个也是因人而异的. 我们在复杂嵌入式系统的需求模式[25]一文中提出了嵌入式系统的 7 个需求模式. 我们应用的 5 个案例投影出的子需求都可以匹配到这 7 个需求模式上, 这些需求模式也可以算一个可行的标准.

第三, 解耦后的最终结果与原需求是否等效? 也就是说, 解耦结果是否能达成跟原来需求同样的效果? 目前本

文中并没有给出解决方案, 仅是由需求提供者进行了人工的确认. 这种确认, 不仅是单个子需求的确认, 还要从系统层面进行需求确认. 从本文中的 5 个案例来看, 解耦过程中其实增加了很多需求信息, 如对设备操作不一致性的完善, 这将使得需求更加完整、准确. 但解耦是否会导致系统层面的信息缺失, 还需要系统的方法. 这将作为我们未来的工作.

7 结束语

针对复杂嵌入式系统的需求分析问题, 本文首先给出了嵌入式系统需求、软件需求的描述方式, 定义了嵌入式系统基于投影的软件需求规约过程, 形成了一种从系统需求到软件需求的需求问题解耦方法. 最后进行了实际应用——航天太阳搜索控制软件子系统的案例研究, 并采用耦合度和内聚性两个指标对提出的方法在 5 个典型应用案例场景进行了量化评估. 评估结果说明本文方法可以有效地降低耦合度, 提高内聚性, 达到了目标.

未来工作将考虑设计相应的支撑工具, 使需求工程师能根据工具的引导交互式地进行嵌入式系统需求识别并进行从系统需求到软件需求的规约. 我们还考虑将本文方法应用于更多实际的复杂案例, 捕捉更多的领域特征, 增加更多有针对性的问题解耦策略, 设计系统地需求解耦结果与原需求的等效性确认方法, 建立面向嵌入式软件智能合成的需求规约方法.

References:

[1] Rosero-Montalvo PD, López Batista VF, Rosero EA, Jaramillo ED, Caraguay JA, Pijal-Rojas J, Peluffo-Ordóñez DH. Intelligence in embedded systems: Overview and applications. In: Proc. of the 2019 Future Technologies Conf. (FTC). Switzerland: Springer, 2019. 874–883. [doi: 10.1007/978-3-030-02686-8_65]

[2] Pereira T, Ribeiro Q, Melo M, Magro S, Alencar F, Castro J. Requirements engineering for embedded systems: A systematic literature review. 2021. https://www.inf.puc-rio.br/wer/WERpapers/artigos/artigos_WER21/WER_2021_paper_11.pdf

[3] Braun P, Broy M, Houdek F, Kirchmayr M, Müller M, Penzenstadler B, Pohl K, Weyer T. Guiding requirements engineering for software-intensive embedded systems in the automotive industry: The REMsES approach. Computer Science-research and Development, 2014, 29: 21–43. [doi: 10.1007/s00450-010-0136-y]

[4] Pretschner A, Broy M, Kruger IH, Stauner T. Software engineering for automotive systems: A roadmap. In: Proc. of the 2007 Future of Software Engineering. Minneapolis: IEEE, 2007. 55–71. [doi: 10.1109/FOSE.2007.22]

[5] Lavi JZ, Kudish J. Systems modeling & requirements specification using ECSAM: An analysis method for embedded & computer-based systems. Innovations in Systems and Software Engineering, 2005, 1(2): 100–115. [doi: 10.1007/s11334-005-0010-4]

[6] Lavi JZ, Kudish J. Systems modeling & requirements specification using ECSAM: A method for embedded computer-based systems analysis. In: Proc. of the 11th IEEE Int'l Conf. and Workshop on the Engineering of Computer-based Systems. Brno: IEEE, 2004. 2–11. [doi: 10.1109/ECBS.2004.1316676]

[7] Alford. SREM at the age of eight; the distributed computing design system. Computer, 1985, 18(4): 36–46. [doi: 10.1109/MC.1985.1662863]

[8] Wrigley JC. Requirements decomposition using a graphical method. In: Proc. of the 15th IEEE Int'l Conf. of System of Systems Engineering (SoSE). Budapest: IEEE, 2020. 47–52. [doi: 10.1109/SoSE50414.2020.9130462]

[9] Jackson M. Problem Frames: Analyzing and Structuring Software Development Problems. Boston: Addison-Wesley, 2001.

[10] Jin Z. Environment Modeling-based Requirements Engineering for Software Intensive Systems. San Francisco: Morgan Kaufmann, 2018. [doi: 10.1016/C2014-0-00030-5]

[11] Jin Z, Chen XH, Zowghi D. Performing projection in problem frames using scenarios. In: Proc. of the 16th Asia-Pacific Software Engineering Conf. Batu Ferringhi: IEEE, 2009. 249–256. [doi: 10.1109/APSEC.2009.22]

[12] Yuan ZH, Chen XH, Liu J, Yu YJ, Sun HY, Zhou TL, Jin Z. Simplifying the formal verification of safety requirements in zone controllers through problem frames and constraint-based projection. IEEE Trans. on Intellignet Transportation Systems, 2018, 19(11): 3517–3528. [doi: 10.1109/TITS.2018.2869633]

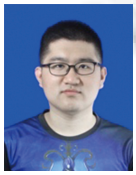
[13] Liu XS, Yuan ZH, Chen XH, Chen MS, Liu J, Zhou TL. Safety requirements modeling and automatic verification for zone controllers. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1374–1391 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5952.htm [doi: 10.13328/j.cnki.jos.005952]

[14] Wrigley JC. Structuring the problem space for model-based systems engineering. INSIGHT, 2019, 22(1): 45–50. [doi: 10.1002/inst.

12238]
[15] Friedenthal S, Moore A, Steiner R. Omg systems modeling language (OMG SysML™) tutorial. INCOSE Int'l Symp., 2008, 18(1): 1731–1862. [doi: 10.1002/j.2334-5837.2008.tb00914.x]
[16] Penzenstadler B. DeSyRe: Decomposition of systems and their requirements [Ph.D. Thesis]. Munich: Technische Universität München, 2011.
[17] Penzenstadler B. Exactly the information your subcontractor needs: DeSyRe—Decomposing system requirements. In: Proc. of the 1st Int'l Workshop on Requirements Patterns. Trento: IEEE, 2011. 1–10. [doi: 10.1109/RePa.2011.6046720]
[18] Heninger KL. Specifying software requirements for complex systems: New techniques and their application. IEEE Trans. on Software Engineering, 1980, SE-6(1): 2–13. [doi: 10.1109/TSE.1980.230208]
[19] Heitmeyer CL. Software Cost Reduction. Chichester: John Wiley Sons, Ltd., 2002. [doi: 10.1002/0471028959.sof307]
[20] Jin Z, Liu L. Towards automatic problem decomposition: An ontology-based approach. In: Proc. of the 2006 Int'l Workshop on Advances and Applications of Problem Frames. Shanghai: ACM, 2006. 41–48. [doi: 10.1145/1138670.1138678]
[21] Jackson M. The meaning of requirements. Annals of Software Engineering, 1997, 3: 5–21. [doi: 10.1023/A:1018990005598]
[22] Fenton N, Melton A. Deriving structurally based software measures. Journal of Systems and Software, 1990, 12(3): 177–187. [doi: 10.1016/0164-1212(90)90038-N]
[23] Newman MEJ, Girvan M. Finding and evaluating community structure in networks. Physical Review E, 2004, 69(2): 026113. [doi: 10.1103/PhysRevE.69.026113]
[24] 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]
[25] Wang XQ, Chen XH, Yang X, Yang B. Requirements patterns for complex embedded systems. In: Proc. of the 30th IEEE Int'l Requirements Engineering Conf. Workshops (REW). Melbourne: IEEE, 2022. 14–17. [doi: 10.1109/REW56159.2022.00011]

附中文参考文献:

[13] 刘筱珊, 袁正恒, 陈小红, 陈铭松, 刘静, 周庭梁. 区域控制器的安全需求建模与自动验证. 软件学报, 2020, 31(5): 1374–1391. http://www.jos.org.cn/1000-9825/5952.htm [doi: 10.13328/j.cnki.jos.005952]
[24] 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]



王小齐(1998－), 男, 硕士生, 主要研究领域为嵌入式软件需求分析, 模型驱动开发.

顾斌(1968－), 男, 博士, 研究员, CCF 高级会员, 主要研究领域为可信软件, 嵌入式系统, 计算机控制.

陈小红(1982－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为需求工程, 基于知识的软件工程和形式化方法.

綦艳霞(1973－), 女, 博士, 研究员, CCF 专业会员, 主要研究领域为可信软件, 嵌入式系统, 计算机控制.

金芝(1962－), 女, 博士, 教授, CCF 会士, 主要研究领域为软件需求工程, 知识工程.