软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(3):1264−1289 [doi: 10.13328/j.cnki.jos.007549] [CSTR: 32375.14.jos.007549]

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

空间飞行器控制软件在轨自适应可信演化框架*

李晓锋[1,2], 罗懿行[2], 王路桥[3], 董晓刚[2], 李建文[4], 顾 斌[2], 董云卫[1], 杨孟飞[5]

[1](西北工业大学 软件学院, 陕西 西安 710129)

[2](北京控制工程研究所, 北京 100190)

[3](西安电子科技大学 计算机科学与技术学院, 陕西 西安 710071)

[4](华东师范大学 软件工程学院, 上海 200062)

[5](中国空间技术研究院, 北京 100194)

通信作者: 董晓刚, E-mail: dongxiaogang@263.net

摘 要: 空间飞行器的智能自主水平和在轨稳定运行能力是提升航天任务成功率的关键, 而软件自适应演化技术则是实现这一目标的重要途径, 并成为当前软件工程领域的研究热点. 首先对空间飞行器控制软件自适应演化的研究现状和存在问题进行综述. 然后, 针对空间飞行器在轨运行环境的开放性、宿主计算资源的有限性以及飞行任务对实时响应的高要求, 提出名为 MAPE-KV (monitor-analyze-plan-execute over knowledge and verification) 的空间飞行器控制软件在轨自适应可信演化框架. 此外, 在该框架的指导下对航天器软件应用逻辑、自适应控制逻辑、可信保障逻辑以及支撑知识库进行设计. 最后, 通过非预期故障和业务变更两类典型案例验证所提框架的有效性, 仿真结果表明该方法能够有效应对星载软件在运行过程中面临的异常事件挑战.

关键词: 空间飞行器; 控制软件; 在轨; 自适应; 可信保障

中图法分类号: TP311

中文引用格式: 李晓锋, 罗懿行, 王路桥, 董晓刚, 李建文, 顾斌, 董云卫, 杨孟飞. 空间飞行器控制软件在轨自适应可信演化框架. 软件学报, 2026, 37(3): 1264–1289. http://www.jos.org.cn/1000-9825/7549.htm

英文引用格式: Li XF, Luo YX, Wang LQ, Dong XG, Li JW, Gu B, Dong YW, Yang MF. On-orbit Self-adaptive and Trustworthy Evolution Framework for Spacecraft Control Software. Ruan Jian Xue Bao/Journal of Software, 2026, 37(3): 1264–1289 (in Chinese). http://www.jos.org.cn/1000-9825/7549.htm

On-orbit Self-adaptive and Trustworthy Evolution Framework for Spacecraft Control Software

LI Xiao-Feng[1,2], LUO Yi-Xing[2], WANG Lu-Qiao[3], DONG Xiao-Gang[2], LI Jian-Wen[4], GU Bin[2], DONG Yun-Wei[1], YANG Meng-Fei[5]

[1](School of Software, Northwestern Polytechnical University, Xi'an 710129, China)

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

[3](School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

[4](Software Engineering Institute, East China Normal University, Shanghai 200062, China)

[5](China Academy of Space Technology, Beijing 100194, China)

Abstract: The intelligent autonomy and stable on-orbit operation capabilities of spacecraft are crucial to enhancing the success rate of space missions. Software self-adaptive evolution technology is an essential approach to achieving this goal and has become a research hotspot in the field of software engineering. This study first provides an overview of the research status and existing issues in the self-adaptive evolution of spacecraft control software. Then, in response to the openness of the spacecraft on-orbit operating environment, the limited computing resources of the host, and the high real-time response requirements of flight missions, an on-orbit self-adaptive and

* 基金项目: 国家自然科学基金 (U21B2015, 62192730, 62192735, 62402038)
收稿时间: 2024-09-26; 修改时间: 2025-05-06; 采用时间: 2025-09-02; jos 在线出版时间: 2025-12-24
CNKI 网络首发时间: 2025-12-26

trustworthy evolution framework for spacecraft control software, termed MAPE-KV (monitor-analyze-plan-execute over knowledge and verification), is proposed. In addition, under the guidance of this framework, the application logic, self-adaptive control logic, trustworthiness assurance logic, and supporting knowledge base for spacecraft software are designed. Finally, the effectiveness of the proposed framework is validated through two typical scenarios: unexpected failures and service changes. Simulation results demonstrate that the proposed method can effectively address the challenges posed by unexpected events during the operation of spaceborne software.

Key words: spacecraft; control software; on-orbit; self-adaptive; trustworthy assurance

1 引 言

重大复杂的航天任务如火星取样返回、载人登月等, 对空间飞行器的智能水平和自主能力提出了多项要求[1]. 作为空间飞行器系统的核心和关键部分, 星载软件负责保证系统在运行过程中与环境动态交互时的安全性, 包括实现基本的自主健康管理和故障状态下的重构能力, 以应对新型任务需求的任务规划和再分配. 软件自适应演化理论通过实时检测需求变化、环境变化以及自身状态等多种系统可变因素, 自主调整软件系统的体系结构、软件单元的能力及其协作关系, 从而持续提供符合用户期望的服务, 并提高系统的适应性、可靠性和鲁棒性. 这种能力在应对空间飞行器面临的复杂环境变化时尤为重要. 因此, 自该概念提出以来, 软件自适应演化方法受到了各航天大国的高度重视.

美国在航天领域自适应软件的研究方面取得了显著进展. 1998 年发射的深空一号 (DS-1) 首次实现了指令生成、分发和执行, 并通过自主故障诊断系统 Livingstone 进行模式识别与系统重构[1]. 两年后, 美国发射地球观测 1 号 (EO-1) 卫星, 并搭载了自主优化决策软件 ASE, 该软件能够自主选择通信链路, 将实验数据快速传回地面, 显著降低地面运维成本[2]. 2014 年, 美国国家航空航天局 (NASA) 利用 F/A-18 飞机模拟了早期飞行段的火箭, 以测试 NASA 下一代运载火箭的自适应软件在飞行过程中对环境变化的适应能力. 此外, 美国国防部高级研究计划署 (DARPA) 资助了一系列与自适应软件相关的项目, 目前正在开展意图定义的自适应软件项目 IDAS (intent-defined adaptive software) 的研究. 该项目旨在开发自动化的软件代码生成和演化技术, 以快速应对软件任务需求和运行环境的变化, 使软件开发和维护成本实现数量级降低.

经过数十年的研究和实践, 中国所研制的各类空间飞行器已具有一定的自调整能力, 典型的如载人飞船应用全系数自适应控制理论和思想设计了返回再入控制[3], 嫦娥四号探测器采用复杂环境动力下降智能自主控制技术, 完整实施了月球着陆避障[4]等. 空间飞行器的自调整能力是通过在控制软件中预先设定数据有效性判断、部件或系统的故障诊断等策略, 然后根据策略决策结果进行组件切换或系统重构等操作实现. 即现有的这种自调整能力是基于设计师认知的空间飞行器在轨运行时可能发生的各种情况的, 并将相应的对策以固化代码的形式在星上运行, 这种方法在一定程度上是有效的, 随着知识和经验的积累, 对策会不断完善, 从而提升空间飞行器在轨稳定运行能力. 然而, 当面对新的用户需求 (战争/自然灾害等紧急态势改变或增加相应功能)、不确定的环境变化 (如空间高能粒子打翻存储器等) 与未知的在轨故障时, 现有的软件往往不能在线及时应对, 需要地面人工分析和决策后, 重新编写软件补丁代码, 依靠地面站在有限通信时间窗口内发出注入指令来完成软件维护, 整个过程需要制定各种详细的运行标准和流程, 并建造装备与此配套的地面基础设施和仪器设备, 耗费庞大的人力物力.

总而言之, 目前空间飞行器控制软件在轨自主能力主要存在以下 3 方面不足: (1) 软件逻辑紧密耦合, 当面临变更时需要大量的源代码修改, 不仅限制了软件在轨动态调节的能力, 也引入了潜在安全隐患, 缺少软件在轨主动动态调节自身行为的机制和方法; (2) 软件缺少针对积累经验的归纳和推理机制, 较难应对需求变化和未知的各种故障; (3) 软件可信保障手段均需占用较大的系统资源, 仅能在地面实施, 未有在轨条件下实时验证的相关设计.

针对以上问题, 本文主要贡献为: (1) 提出一种面向软件变更的空间飞行器控制软件在轨自适应可信演化框架 MAPE-KV. 实现了控制软件应用逻辑、自适应逻辑和可信保障逻辑的全面解耦, 并设计支持灵活修改的知识库结构, 从而确保软件持续可靠地自主演化; (2) 实现面向资源受限航天场景的轻量级自适应决策方法与运行时验证技术, 显著提升系统在强实时性、低资源条件下的自适应能力与可信保障能力; (3) 通过卫星太阳搜索系统的非预期故障与任务规划系统需求拓展这两个在轨控制软件典型应用场景, 验证了本文框架在实际工程中的可行性与有效性.

2 自适应演化技术研究现状

合理的软件模型有助于提高软件运行时的自适应能力, 优质的自适应策略能够指导软件系统的演化调整过程, 全面的策略验证方法能够确保演化结果的可信. 因此, 本节从自适应系统建模方法、自适应演化决策方法和自适应过程可信性保障方法这 3 个方面对研究现状进行综述.

2.1 自适应系统建模方法

针对需求建模, 现有研究主要集中于功能性需求和非功能性需求两类问题. 自适应软件的功能性需求建模常用方法有 Z 语言和特征模型. Z 语言具备高可信特征, 但不适合描述软件动态过程[5]; 特征模型允许运行时动态重配置[6], 但难以描述需求关系. 非功能性需求建模方法有排队模型和 KAOS 模型, 排队模型适合建模性能或响应时间等需求[7], KAOS 模型支持形式化建模[8]. 在运行时动态修复场景下, Chu 等人[9]采用信号时序逻辑 (STL) 对系统需求进行建模, 通过削弱与增强需求, 实现自适应的降级与恢复协调机制. 针对环境建模, 现有研究主要集中在本体模型和概率模型. 本体模型用于描述由一套对象、属性以及关系所构成的系统, 用来支持对该领域的属性进行推理[10]. 概率模型是对一连串随机事件动态关系的定量描述, 能够将环境建模为概率过程, 用随机变量建模环境的状态[11]. 针对结构建模, 现有研究主要集中在体系结构模型. 如 Cetina 等人[12]采用领域特定建模语言描述体系结构. 少量研究采用其他模型建模系统的结构, 其中, 类图将软件结构分类建模并描述类与类之间的结构和关系, 适用于结构化、柔性的软件结构建模工作. 如 Salehie 等人[13]采用类图对系统组成进行建模, 发掘各个类与功能性需求、安全性需求之间的关联关系. 组件图能够表示组件如何互相组织以构建复杂系统, 如 Filho 等人[14]在 EWS 系统中使用组件图描述系统的体系结构, 描述各个组件在系统中的关联关系. 针对行为建模, 现有研究主要集中于状态图模型、活动图模型以及概率模型等方面. 状态图模型描述对象在生存期间的动态行为, 如 Lee 等人[15]对软件演化的空间进行了建模. 活动图展现了系统内活动的流程, 利于识别并行的活动, 典型的活动图模型有 Petri 网等[16]. Maia 等人[17]在 Dragonfly 系统中使用活动图模型描述了无人机系统实现自适应的流程.

现有自适应软件方法在需求建模、环境感知、结构调节和行为演化等方面取得了丰富的研究成果, 广泛应用于汽车电子、机器人、无人机等 CPS (cyber-physical system) 典型场景中. 然而, 这些方法通常仅从单一维度进行建模, 未能全面覆盖自适应软件的多维特性, 这在一定程度上限制了其在空间飞行器自适应软件的演化与验证中的应用. 此外在航天领域, 系统面临强实时性、资源受限与高可信保障的多重挑战, 现有方法在模型表达精度、决策解释性和在轨可控性等方面尚难满足其严苛要求. 因此, 亟需面向航天控制软件特点, 构建具有多维建模能力、可信演化保障与知识驱动决策机制的一体化方法体系.

2.2 自适应演化决策方法

软件自适应决策的目标是制定出当前最优策略, 以指导软件调整自身参数、行为以及结构. 针对空间飞行器控制软件强实时、高可靠以及硬件资源受限等需求的自适应决策方法较为匮乏, 现有研究主要集中在基于规则的决策技术、基于搜索的决策技术和基于学习的决策技术这 3 方面.

基于规则的决策技术主要有 Drools 规则引擎和 ECA 规则等方法. Drools 规则引擎基于 RETE 算法, 完全开源且易于编辑更新和管理策略[18]. ECA (事件-条件-动作) 规则描述简洁清晰, 因此被应用到多个领域, 并被视作是执行协调演化的最高效的策略方法之一[19]. 例如 Vlahavas 等人[20]将 ECA 规则应用于知识系统, 并行开展规则匹配和执行; 张震[21]将 ECA 规则应用到空间数据库的条件匹配和规则执行; 胡利平等人[22]采用 ECA 规则对航空集群决策规则进行规范化表达, 提高了航空集群的灵活性和适应性; 李青山等人[23]将 ECA 规则引入软件自适应领域, 提出了一种软件自适应动态演化机制; Moreno 等人[24]则在其提出的 DARTSim 系统中引入 ECA 规则, 实现了一组无人驾驶飞行器在敌对和未知环境中执行侦察任务的高级模拟. 相比于 Drools 规则, ECA 规则具有专门的监测机制, 当环境发生变化时 ECA 会主动根据变化进行决策演化. 同时, ECA 方法的运行效率高, 契合航天领域专用软件的强实时性需求. ECA 规则能够以简洁的语言描述策略的触发和执行过程, 减少大量的策略解析时间, 符合空间飞行器控制软件资源受限条件. 但是该类方法存在一个缺陷问题, 即需要预先静态定义所有的候选策略, 可

扩展性较差, 航空航天技术应综合考虑可扩展性、动态、柔性等要求, 因此仅使用该方法难以满足航天领域要求.

在基于搜索的决策方面, 刘道文等人[25]提出了一种基于混沌局部搜索的粒子群算法, 并将其应用于选址问题的决策. 该算法利用混沌优化进行粒子群局部搜索, 避免陷入局部极小值, 并实现在全局范围上搜索目标函数的最优值. 王璐等人[26]提出一种基于并行搜索优化的自适应决策方法, 将自适应决策视作在调整策略组成的搜索空间中, 评价并选择最优策略的搜索优化问题. 航天领域需要稳定的计算资源和高效的计算效率, 以便航天飞行器能顺利完成更多要求严苛的任务. 但该类方法在效率提升与策略选择方面仍有待改进, 其计算效率受搜索空间规模影响, 且当针对多目标进行权衡决策时, 会产生一个最优策略集合, 需从集合中选择出当下最适用策略, 因此应优化该方法使其符合航天领域资源受限的场景特征.

基于学习的决策技术主要从基于强化学习、深度学习和迁移学习的决策方法开展研究. Wu 等人[27]提出了一种基于强化学习的自适应演化方法, 该方法通过生成新的策略来处理未知情况. 深度学习将软件自适应调整等任务转化为回归问题[28,29], 当获得了自适应软件的相关原始数据, 即可采用深度学习模型进行求解. Wang 等人[30]在 EWS 系统中使用强化学习方法确定运行时的最佳策略. 将深度学习与控制论结合, 能在外界发生变化时将软件系统自动调整到设定的运行状态. 常见的深度强化学习网络有 DQN 网络等[31]. 迁移学习可以在数据不足时, 利用在训练数据充分问题上的已训练模型, 通过迁移操作使模型适合于新的问题[32]. 将软件自适应决策方法与强化学习、深度学习和迁移学习结合是未来软件自适应演化发展的重要方向. 在目前航天领域中, 空间飞行器提供可靠、稳定、高性能的算力仍是一个巨大挑战, 但是该类方法对硬件要求高, 模型选择依赖于先验知识或基本假设, 因此, 现阶段还很难拓展到航空航天应用场景.

针对空间飞行器控制软件强实时及硬件资源受限等实际应用需求, 基于搜索与学习的方法存在资源要求高、无法实时决策等缺点, 因此现阶段很难直接采用此类方法进行在轨决策. 而基于规则的决策技术通过使用既定规则匹配演化事件来调整软件, 虽然决策迅速, 但是需要引入学习机制不断更新规则以适应非预期情况.

近年来, 在 CPS 系统的其他典型应用领域, 自适应决策技术取得了诸多进展. 例如, Aravind 等人[33]针对自动驾驶车辆中复杂电子机械系统的维护问题, 提出融合物理建模与人工智能的预测性维护框架, 通过构建振动-旋转关节等物理退化模型, 联合多模态传感数据与带宽动态调整策略, 实现了对突发性硬件老化与非线性扰动的提前感知与系统优化调度. 在工业机器人领域, Yokoyama 等人[34]提出了自适应技能协调 (ASC) 方法, 通过构建导航、抓取、放置等预训练技能库, 并引入混合专家模型与在线纠正策略, 在未知环境中实现了高鲁棒性的动作协调与零样本迁移部署. 在无人机系统中, Abbaspour 等人[35]面向非线性六自由度飞行模型, 设计了结合神经网络自适应结构 (NNAS) 与非线性动态逆控制器 (NDI) 的主动容错控制方案, 结合扩展卡尔曼滤波实现故障信号识别与反馈补偿, 在执行器故障场景下实现了亚弧度级的精确控制. 这些方法在运行机制上呈现出明显的共性, 均通过引入学习、自适应调整与推理控制机制, 提升系统对未知扰动或突发故障的容错与动态恢复能力.

然而, 上述方法多部署于地面或资源相对充足的运行平台, 具备宽裕的计算能力、传感器与通信带宽, 能够支持复杂神经网络结构、多模态数据融合以及延迟容忍度较高的推理过程. 相比之下, 空间飞行器控制软件在轨运行面临极端资源受限、强实时性与高度可靠性等综合约束, 决策策略必须以毫秒级时间进行响应, 并保障系统在长时间任务中的持续稳定性, 因而难以直接应用复杂模型或高度依赖数据驱动的 CPS 方案.

因此, 本文聚焦航天控制软件面向未知事件的策略演化问题, 提出将 ECA 规则机制的快速响应能力、启发式搜索策略的灵活组合能力与强化学习方法的动态适应能力相融合, 构建适用于空间飞行器资源条件的软件自适应演化机制. 该机制在吸收 CPS 系统中已有方法共性的基础上, 结合航天系统运行环境的特殊性, 实现了具备快速响应、策略优化与运行时修复能力的轻量级自适应控制策略框架, 为航天领域在轨软件的演化与自治提供了关键技术支撑.

2.3 自适应过程可信性保障方法

自适应软件可信性保障的目标是对软件系统的演化策略和演化结果进行验证与评估. 当前验证的研究已经有很多成果, 按模型的获得方式可分为静态验证 (static verification) 和运行时验证 (runtime verification) 两种方法.

静态验证包含定理证明[36,37]和模型检测[38,39]等方法. 定理证明是使用数学知识来严格推导证明待验证的系统满足给定的性质, 现今被工业界和学术界广泛接受的主流的定理证明器有 Coq[40]和 Isabelle[41]. Lean[42]代表性的成果有通过定理证明技术完全地证明了操作系统微内核 seL4[43]和 C 程序编译器 Comcert[44]的正确性. 近年来, 还有一些用深度学习或大模型 LLM 构建的自动定理证明方法[45,46]. 模型检测是使用自动化搜索的策略来穷举待验证系统的所有行为从而验证系统是否满足给定的性质, 如 Vogel 等人[47]在 mRUBiS 系统中引入一组验证器, 对返回的结果进行验证适配, 以监控 mRUBiS 体系结构中是否存在问题. 其对应代表性的模型检查器包括 SPIN[48]、CBMC-C[49]、 NUXMV[50]等经典算法, 以及 IC3[51]和 CAR[52]等先进模型检测技术. 研究界还发展了一系列启发式优化策略, 如 i-good-lemmas 引理生成技术[53]和预测引理[54]辅助推理方法等. 取得的主要成就有帮助火星探测器成功完成任务[55], 有效解决了复杂电路验证[56]和芯片设计验证[57]难题. 而商用 EDA 软件有效保障了硬件设计的正确性. 定理证明的优势在于不存在所谓的“空间爆炸问题”, 但其主要问题是无法实现推理完全自动化, 多数情况下需要大量的人工介入; 而模型检测可以实现自动化的验证过程, 需要很少甚至不需要人工介入, 但其性能主要受限于“状态空间爆炸”[40]问题.

运行时验证 (runtime verification, RV) 是一种轻量级且严谨的形式化方法, 用于在系统运行期间动态分析其执行轨迹是否满足预定义的规范, 指给定一个形式化规范, 检查当前待验证的系统轨迹是否满足规范. 与传统的静态验证方法相比, 运行时验证无需对系统进行完整建模, 具有资源开销小、部署灵活等优势, 因此在多个关键领域得到了广泛应用. 在航天领域, 美国空间航天局 NASA 已经将该技术集成到火星探测机器人 Robonaut2 上用于保证其运行的安全性[47,49]. 此外, 美国国家航空航天局 (NASA) 开发了 R2U2 (realizable, responsive, unobtrusive unit) 系统[58], 专为航天器、无人机和机器人等嵌入式系统设计, 能够在资源受限的环境下实时分析形式化系统需求. R2U2 已成功部署于 NASA 的 Robonaut2 机器人、探空火箭以及月球门户计划中, 展示了其在实际航天任务中的可行性和有效性. 在铁路系统中, 随着控制系统复杂性的增加, 传统的预编程安全条件监控已难以满足需求. 研究人员提出了参数化模态实时序列图 (PMLSCs) 作为监控规范语言, 用于对中国高速铁路的 RBC 系统进行运行时验证, 提高了监控效率并降低了误报率[59]. 此外, 运行时验证在网络安全、金融安全和法律合规等领域也发挥着重要作用[50,55,60]. Maia 等人[17]在 Dragonfly 系统中使用监控器检测环境数据与传感器数据以验证变量是否在合理范围内; Tsigkanos 等人[61]在 AMELIA 系统中为实现运行时的需求验证设置了监控器以监控环境信息与系统上下文. 运行时验证所需的模型可以直接从系统真实运行轨迹中获得, 所需的硬件资源少. 但是, 由于运行时验证仅用到了部分系统行为对应的模型, 其只能用于检测系统存在的问题, 而不能用来证明系统的正确性. 因此, 运行时验证通常与其他验证方法 (如静态分析和模型检查) 结合使用, 以实现更全面的系统验证.

针对空间飞行器在轨自适应软件系统的验证需求, 由于验证技术需要集成到在轨自适应软件系统中自动运行, 定理证明需要人工参与不适合作为可选方案, 模型检测技术可用于针对演化策略的验证, 但是需要进一步提升处理模型的范围和验证效率. 运行时验证技术可用于针对演化结果的验证, 但需要优化运行时验证的资源配置, 并扩展其能处理的规范性质范围.

3 在轨自适应演化体系

与传统的空间飞行器软件在轨维护需要大量人工干预的过程不同, MAPE-KV 关注演化需求与系统行为间的语义关系表达, 并以此为基础建立资源受限条件下具有自学习能力的决策与重构机制实现对环境变化、在轨故障以及扩展任务的精准识别与快速重构, 从而减少设计师地面人工分析、决策、补丁生成及验证等工作, 提升空间飞行器的智能自主及稳定运行能力, 基于 MAPE-KV 的自适应控制软件体系结构如图 1 所示.

在空间飞行器的在轨运行中, 自适应控制软件系统通过 MAPE-KV 框架实现高效的动态演化与自主管理. 该框架的核心在于其自适应控制层, 它由感知 (monitor, M)、分析 (analyze, A)、决策 (plan, P) 和执行 (execute, E) 这 4 个关键阶段组成, 每个阶段都运用了先进的技术手段来确保系统的自适应性和可靠性. 在感知阶段, 系统利用传感器网络和数据采集模块实时收集飞行器的运行状态数据, 包括姿态、轨道参数以及关键部件的工作状态等.

这些数据通过高效的数据传输机制被传递到分析模块, 为后续的决策提供实时依据. 分析阶段则运用机器学习和数据挖掘技术, 对收集到的大量数据进行深度分析, 以识别潜在的故障迹象和需求变化. 通过模式识别和异常检测算法, 系统能够快速准确地判断当前状态是否符合预期, 从而为规划阶段提供决策支持. 决策阶段是自适应控制的关键环节, 它基于分析阶段的结果, 运用优化算法和决策理论, 制定出最优的演化策略. 这些策略不仅考虑当前的故障或需求变化, 还兼顾系统的长期稳定性和资源利用效率. 执行阶段则负责将规划好的策略转化为具体的动作指令, 通过精确的控制算法和执行机制, 确保策略能够准确无误地实施. 在整个过程中, 知识库 (knowledge base) 作为核心的知识存储单元, 存储了系统的运行经验和自适应规则, 为各个阶段提供知识支持. 同时, 可信保障逻辑 (verification) 对生成的策略进行实时验证, 确保其安全性和有效性. 通过这种结构化的自适应控制框架, 空间飞行器能够在复杂的在轨环境中实现自主的动态演化, 无需人工干预即可快速响应各种变化, 显著提升了系统的可靠性和运行效率.

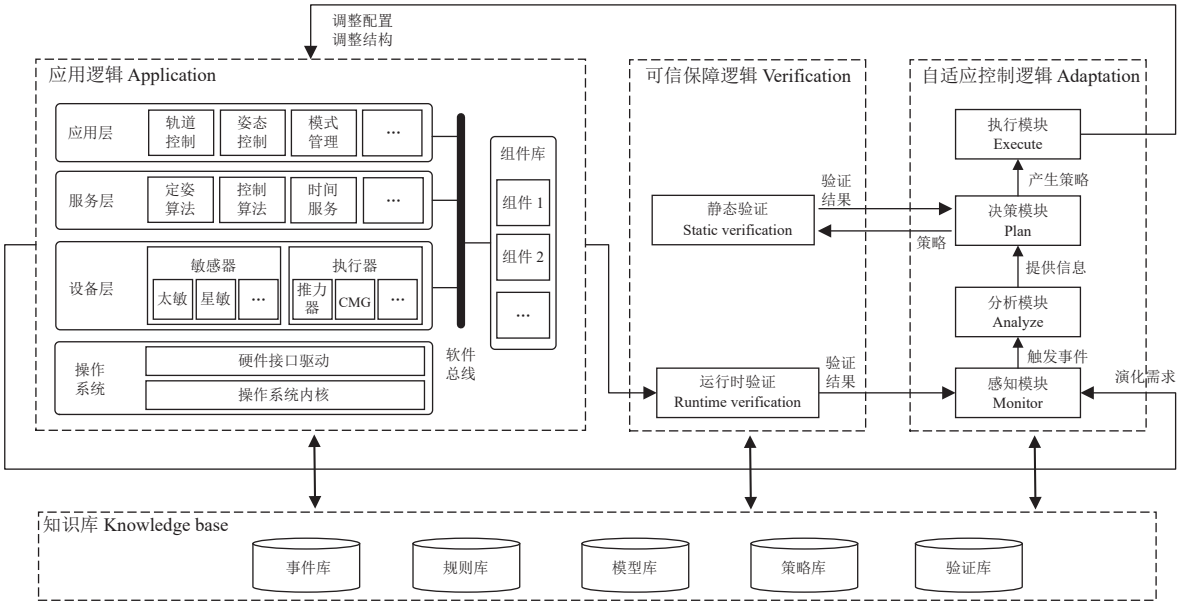


图 1 自适应控制软件体系结构示意图

3.1 航天器软件应用逻辑框架

航天器软件应用逻辑以实现空间飞行器控制任务为目标, 可以分为设备层、服务层和应用层, 其中设备层用来管理飞行器的敏感器及执行机构, 具体包括陀螺仪、星敏感器、推力器等, 每个部件对外提供数据通信、加断电等软件组件. 服务层主要提供 I/O 数据缓存、时间处理、协议解析、运行事件记录以及定姿、控制算法等功能组件, 例如定姿方案中有星敏感器加陀螺仪组合定姿方案、星敏感器定姿方案、太阳敏感器定姿方案等组件. 应用层提供面向应用的功能组件, 例如对飞行模式管理组件具体包括对日捕获与定向模式组件、对日三轴稳定模式组件、大角度姿态机动模式组件等. 软件系统的可扩展功能组件存放在组件库中, 供自适应演化过程动态调用. 软件系统的各功能组件按照模块化进行封装, 组件具备参数调整及动态重构的能力, 软件系统可以根据需求变化动态选择使用的组件. 为了实时感知软件系统的运行状态, 在飞行器控制逻辑中设置监控点追踪软件运行轨迹, 提取组件可能出现的异常事件和非预期状态, 对航天器软件应用逻辑框架中的组件和行为建立模型.

● 对应用逻辑框架中的组件建立模型. 由于组件具有易变特性, 所以首先需对引起组件行为的数据进行描述. 本文将组件运行过程中涉及的数据分为内部数据和端口参数两类, 内部数据是组件运行过程中产生, 不与外部交互的数据; 端口参数是参与组件有关功能运算的基础, 包括外部输入的数据和组件生成并输出的数据. 内部数据的使用与外系统无关, 而端口参数涉及与外系统的交互, 在数据获取或发送过程需要考虑数据更新、数据可用等问题. 因此, 本文将组件的模型描述按其属性定义划分为组件类型、组件端口、端口参数这 3 个元素组合, 其中端口

及参数是可变元, 通过对不同的端口和参数进行调整, 从而很好地表征组件的动态特性. 用 S 表示组件模型, *s_type* 表示组件类型, *port* 表示组件端口, *par* 表示端口参数, 表示为 S={*s_type*, *port*, *par*}. 以“Roll_shaft-Bias_10_degree_flight”需求为例, 组件参数将被动态调整, 并反馈至行为模型.

● 对应用逻辑框架中的行为建立模型. 行为模型包括控制软件需要执行的操作和这些操作引起的软件状态变化. 因需求变化导致软件行为变化, 所以本文的行为模型首先描述状态转移信息, 并重点声明行为源状态、行为目标状态等可变元; 其次, 本文将行为的方向性抽象为行为模型的描述元素, 对软件行为的具体操作进行描述, 表示行为指导下的行为源与行为目标间的关联关系; 最后, 由于状态转移可能因为组件之间的相互调用而发生, 且复杂需求通过一定的行为序列去实现, 故行为模型还包含软件行为间的参数传递. 因此, 行为模型最终可描述为包含上述可变元、行为方向及相关参数的元组. 用 B 表示行为模型, *origin* 表示源状态, *target* 表示目标状态, d 表示行为方向, 相关参数用 *para* 表示, 表示为 B={*origin*, *target*, d, $para_i$,…}, i=1, 2, 3,…. 例如“capturing_the_sun”动作, 则行为模型将定义 solar_sensor 和 solar_panel 为行为源, 行为目标为“sensorTone”“panelTurn”等, 并对行为基本信息进行描述.

3.2 软件自适应控制逻辑框架

空间飞行器控制软件的自适应部分以控制论为核心, 采用 MAPE 控制循环模型来建模控制逻辑. 通过与应用逻辑部分以及地面测控系统的交互, 感知并分析系统变化, 随后生成并执行应对策略. MAPE 模型将自适应软件系统分为感知、分析、决策和执行这 4 个环节, 重点关注这些功能的循环执行.

感知环节通过针对性的感知方法, 确保自适应系统能够精准捕获空间飞行器控制软件的当前系统状态. 该环节通过收集运行时的外部环境数据、地面业务需求以及系统故障信息, 为后续的分析提供基础数据支持. 分析环节主要针对运行时的外部环境、地面业务需求以及系统故障, 通过感知过程所收集的信息, 对系统运行中发生的异常事件进行深度解析. 分析的目的是识别需求类型和需求内容, 并将 3 种类型异常事件进行统一描述, 从而为决策提供依据, 并辅助简化决策过程. 决策环节通过叶子需求-元行为拆解以及人工智能算法, 为系统动作执行机构提供应对异常事件的策略. 该环节的核心是针对目标需求制定出最优策略, 以确保系统能够迅速恢复迁移至正常运行状态, 或根据新的任务需求进行调整. 执行环节负责将决策环节生成的策略具体化, 并与应用逻辑紧密耦合, 负责确保策略的高效实现和有效执行. 执行环节包括实际具体的执行设备系统操作, 如调整飞行器姿态调整、轨道修正以及子系统的重启等. 图 2 展示了自适应控制软件状态迁移过程, 详细说明了每个环节的具体作用和相互关系.

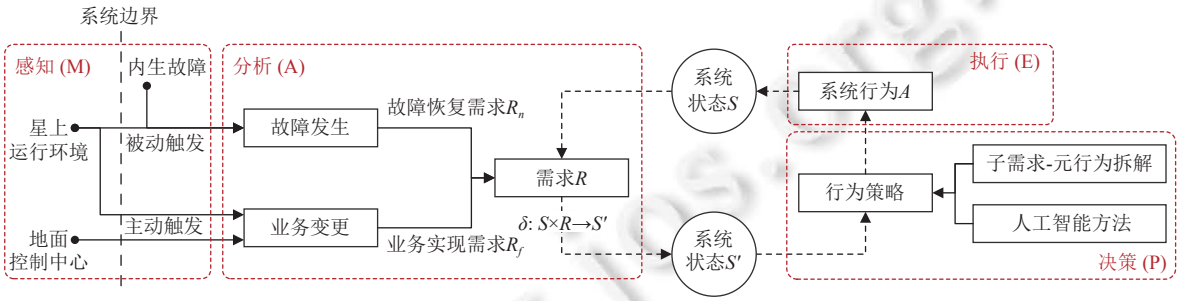


图 2 自适应控制逻辑状态迁移示意图

在资源受限的环境下, 空间飞行器控制软件的自适应框架需要高效利用有限的内存和计算资源. 为此, 我们采取了以下策略: 首先, 在内存管理方面, 通过优化数据结构和采用数据压缩技术, 减少了感知和分析环节中数据存储的内存占用. 同时, 利用缓存机制和优先级调度, 确保关键数据的快速访问和处理. 其次, 在计算复杂度控制方面, 通过引入启发式算法和分层决策机制, 降低了决策环节的计算负担. 例如, 在感知环节, 采用特征提取和降维技术, 减少数据处理的复杂度. 在分析环节, 利用机器学习模型的轻量化设计, 提高分析效率. 在决策环节, 通过预定义的规则库和动态策略生成的结合, 快速生成应对策略, 避免复杂的实时计算. 这些策略确保了自适应框架在资源

受限的条件下仍能高效运行, 满足空间飞行器对实时性和可靠性的要求.

(1) 自适应触发要素

自适应触发要素决定了系统需动态响应的基本情形, 涵盖了空间飞行器运行过程中所有需要系统演化的外部和内部事件. 一般而言, 自适应触发要素主要包括环境变化、故障突发和业务变更这 3 个维度. 环境变化指的是星载软件所处的外部物理或信息环境发生了显著变化 (如空间环境辐照、热控变化、通信条件变化等), 这类变化可能进一步引发新的功能需求或导致系统异常. 故障突发则是指系统内部元器件、传感器、执行机构等出现异常, 如部件失效、参数漂移、通信中断等, 需系统即时诊断并采取措施. 业务变更则主要指任务业务的调整或升级 (如成像目标切换、观测模式变化、临时应急指令等), 需系统重新适配、重构控制逻辑和参数.

本文将上述 3 类自适应触发要素进一步统一描述. 进一步分析可见, “环境变化”最终可以归约为“故障突发”或“业务变更”. 本质上, 所有环境扰动事件最终都可通过系统监测与故障诊断、业务管理机制等环节转化为需要处理的新“需求”——包括故障修复需求和任务重配置需求. 因此, 无论起因于外部环境还是系统内部异常, 最终均可以用“需求未满足/已满足”这一统一视角进行抽象表达. 这样, 环境变化、故障突发和业务变更这 3 类事件最终都成为系统自适应控制中的“需求触发点”. 具体的, 故障处理的过程是当前异常状态到目标稳定状态的迁移, 而业务变更过程可被视为当前业务未实现到业务实现状态的迁移. 这两种状态迁移过程在模式上具有高度相似性, 因此, 将“故障发生未恢复”与“业务变更未实现”均定义为“需求未满足状态”, 将“故障已恢复”和“业务已实现”均定义为“需求已满足状态”. 通过将自适应触发点建模为需求描述模型, 以实现自适应要素的统一建模. 图 3 展示了自适应要素的潜在交互关系.

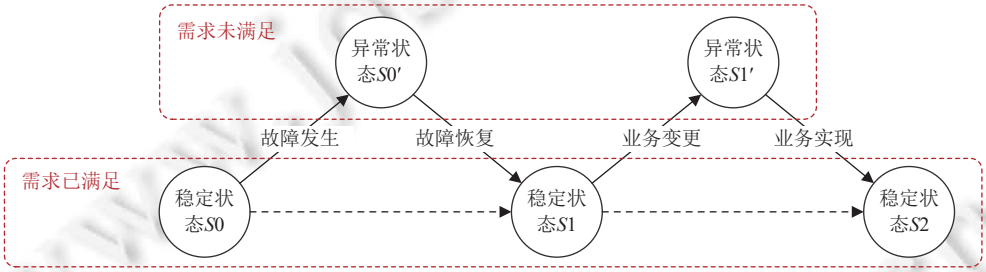


图 3 自适应要素的潜在交互关系

为实现多维度自适应要素建模的统一描述, 本文提出一种基于状态迁移的自适应要素统一描述方法:

$$T = (R, S, A, \delta),$$

其中, T 表示系统状态迁移, R 表示需求集合 $R=(R_f, R_n)$, 包括功能性需求 R_f 和非功能性需求 R_n, S 表示状态集合, δ 表示状态转移函数. δ 状态转移函数定义为:

$$\delta : S \times R \to S',$$

其中, $\delta(s, r)=s'$表示系统在状态 s 下, 当需求 r 产生时, 系统状态转移到 s'.

功能性需求 R_f 以飞行器业务变更为主, 具体涉及指令序列的调整. 任务变更通常由地面指挥中心发起, 并通过一系列精确的操作步骤影响系统的当前状态. 例如, 对于红外成像业务而言, 目标变更请求 $r_{\text{target\_change}}$ 会触发系统状态到指令重规划状态 $s_{\text{instruction\_replanning}}$, 非功能性需求 R_n 以故障发生为主, 具体涉及子系统故障的快速诊断和隔离. 故障事件 $e_{\text{fault\_occurrence}}$ 会触发系统从正常运行状态 s_{normal} 转移到故障检测状态 $s_{\text{fault\_detection}}$, 并通过分析和决策过程确定故障的性质和位置. 随后, 系统状态转移到故障处理状态 $s_{\text{fault\_handling}}$, 执行相应的恢复措施, 最终恢复到正常状态 s_{normal}.

本文建模方法将所有外部环境变化与系统内部故障、业务调整全部用需求状态进行归约抽象, 使不同触发来源下的自适应过程具备一致的逻辑基础. 与传统单一维度模型相比, 本方法在模型层实现了多维度 (环境-故障-业务) 统一表达, 并可支持需求的动态扩展、重用和组合. 在具体实现上, 模型支持对各类触发情形的参数化描述和状态追踪, 为航天高可信软件的在轨自适应演化奠定坚实基础.

(2) 自适应可变要素

自适应可变要素与决策和执行过程紧密关联. 在决策过程中, 由规则知识库牵引, 基于目标模型将需求建模为树形结构, 以实现对抽象主题需求的逐层分解. 随后, 通过策略知识库中存储的“叶子需求-元行为”关系进行策略匹配. 图 4 展示了基于目标模型的需求分解过程.

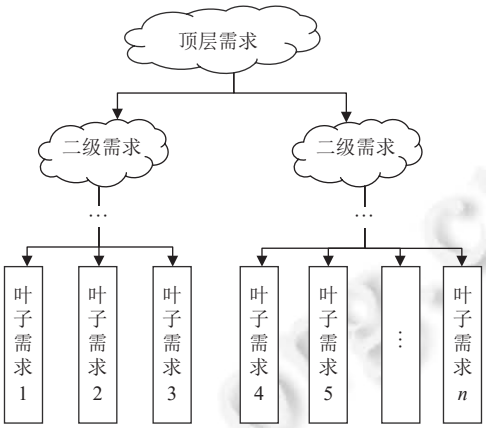


图 4 基于目标模型的需求分解示意图

将需求分解为子需求有助于为需求的具体实现提供清晰路径. 顶层需求代表自适应分析阶段的输入, 通常是系统在特定情境下需要实现的宏观需求, 例如“星敏异常问题排查”. 二级需求代表实现顶层主题需求所需的中间子需求, 例如“陀螺仪状态确认”“太阳敏感器状态确认”等. 通过逐步分解二级需求, 宏观的顶层需求被转化为具体且可实现的目标, 使得复杂的系统需求能够层层细化, 直至每个需求都变得具体和可实现. 最终, 这些需求形成不可再分的叶子需求, 如“陀螺仪重置”. 将该描述需求分解的目标模型 T 定义为 $T=(V,E)$, 其中, V 为表示需求的节点集合. $E\subseteq V\times V$ 是节点之间的有向边的集合, 表示父子关系. 根节点 $R\in V$ 表示主题需求. 分支节点 $B_i\in V$ 表示子需求, 满足 $\forall B_i,\exists(R,B_i)\in E$. 叶子节点 $L_{ij}\in V$ 表示叶子需求, 满足 $\forall L_{ij},\exists(B_i,L_{ij})\in E$.

元行为作为具体的行为指令, 与叶子需求相对应, 提供了实现叶子需求的明确指导. 这种方法不仅使每个需求的实现路径得以明确, 还有效地管理和协调了系统各部分的行为, 从而确保系统在复杂环境中高效、稳定地运行. 针对已知目标, 采用 ECA 规则方法从知识库中匹配相应策略. ECA 规则是一种高效的决策方法, 通过预定义的规则库对事件进行快速匹配和处理. 本文将“事件 E”定义为“目标为已知目标”, 将“条件 C”定义为“元行为可执行条件”, “行为 A”定义为“向执行设备发出元行为指令”. 例如, 当系统单元目标为“陀螺仪重置”, 而该目标与知识库中的已知单元目标“$g1$: 陀螺仪重置”成功匹配, 则事件 E 触发. 当条件 C (“陀螺仪已通电”) 满足, 则执行动作 A, 即“陀螺仪断电再加电”.

对于规则知识库未覆盖的新型需求, 采用人工智能方法进行处理, 这些方法特别适用于复杂和动态的环境. 主要包括进化算法、强化学习和深度学习. 进化算法通过模拟自然来选择优化规则知识, 强化学习通过环境交互来调整元行为的组合以最大化奖励, 深度学习利用神经网络来捕捉复杂需求-行为模式. 这些算法使系统能够自主识别和分析新需求, 并动态调整叶子需求与元行为的映射策略. 通过应用这些方法, 系统不仅能够处理已知需求, 还能在面对非预期需求时迅速适应和优化, 确保在复杂环境中保持高效稳定的运行. 在空间飞行器控制场景中, 学习机制需要重点获取以下核心知识: (1) 环境动态演化规律, 包括空间辐射环境、设备老化特性等长期变化模式; (2) 历史故障处理经验, 通过分析星载系统日志中的异常事件处理记录, 提炼潜在规则; (3) 任务需求变更模式, 挖掘不同任务场景下控制参数调整的关联特性.

然而, 该学习过程面临双重挑战: 其一, 规则完整性方面, 由于在轨环境的开放性和故障模式的不可穷举性, 难以预先构建完备的规则集合; 其二, 学习准确性方面, 受限于星载计算资源约束, 需在有限算力下实现高置信度的规则推理. 上述挑战的根源在于空间任务特殊性: 首先, 在轨数据的稀疏性导致单一故障场景可能仅有个位数样本

记录; 其次, 动态环境的不确定性使得传统监督学习的独立同分布假设失效; 最后, 实时性要求与计算密集型 AI 算法存在固有矛盾.

图 5 展示了基于人工智能方法的非预期需求满足过程. 将叶子需求集合定义为 $L=\{L_1,L_2,\ldots,L_m\}$, 其中 L_i 代表一个具体的需求项. 将元行为集合定义为 $A=\{a_1,a_2,\ldots,a_n\}$, 其中每个 a_j 代表一个针对需求的具体行为或操作. 因此, 存在关系 $R\subseteq L\times A$, 其中, $(L_i,a_j)\in R$ 表示叶子需求 L_i 可以通过元行为 a_j 来满足或实现. 以及约束关系: 每个叶子需求 L_i 可以与一个或多个元行为 a_j 相关联, 即 $\exists j$ 使得 $(L_i,a_j)\in R$. 每个元行为 a_j 可以与一个或多个叶子需求 L_i 相关联, 即 $\exists i$ 使得 $(L_i,a_j)\in R$.

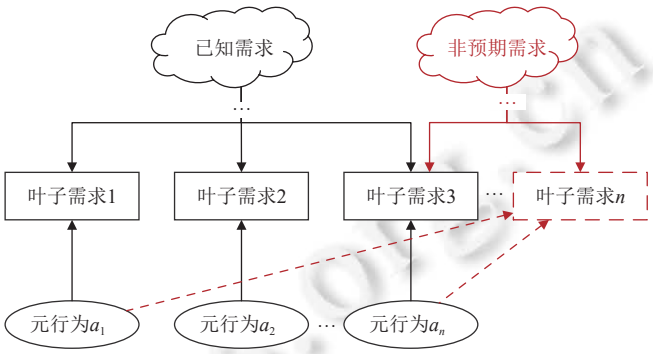


图 5 基于人工智能方法的非预期需求满足过程

3.3 可信保障逻辑框架

可信保障部分是另一种反馈环路, 用于对 MAPE 控制循环进行验证, 包含静态和动态验证两个环节. 在轨自适应演化具有过程多样性、结果可变性等特征. 传统形式化方法一般是采用基于数学和逻辑的技术, 对系统的行为和属性进行精确描述、建模、分析和验证, 以确保其正确性和可靠性. 本文采用模型检查技术来验证在轨自适应演化过程中的关键安全属性和行为正确性, 特别关注由于过程多样性和结果可变性可能引发的非预期状态.

然而, 在轨运行环境对验证模块施加了严苛约束, 设计并实现相关验证技术时, 必须充分考量可用计算资源 (如内存、处理器时间) 的严格限制以及验证结果的实时性需求. 传统的形式化验证方法因其高昂的资源消耗与计算复杂度, 在上述约束下难以满足在轨自适应系统的验证需求. 为此, 本文聚焦于自适应软件系统的核心——策略演化模块, 提出一种旨在保障其策略在演化前后均维持正确性与一致性的验证方法. 该方法在策略演化前, 首先对决策模块生成的演化策略执行轻量级静态验证; 仅当策略通过验证后, 才由执行模块部署实施. 软件演化生效后, 则运用高效的运行时验证技术, 对系统关键性质进行持续的动态监测与验证. 验证结果将实时反馈至自适应控制层的感知模块, 为后续的适应性调整与演化决策提供闭环的经验依据与修正指导, 从而形成一个可信的自适应调控环路.

如图 6 所示, 我们设计并实现了一个可信保障逻辑框架, 由“静态验证”和“动态验证”两个主要部分协同构成, 覆盖从策略生成到运行时监控的全流程验证机制, 以全面保障系统在复杂环境下的决策正确性与行为可靠性.

在静态验证阶段, 框架从左至右依次包括演化策略、模型检查与输出反馈这 3 个关键模块. 我们采用由三元组<E, C, A>构成的策略集合, 分别表示演化事件、约束条件与应对动作. 这些候选策略首先输入至模型检查模块, 该模块集成了死锁检测、传播技术、重启机制及字级别扩展等验证手段, 用于全面分析策略在形式模型下的正确性与安全性. 若策略通过验证, 将进入策略生成模块用于系统部署; 若验证失败, 则自动输出反例, 为后续策略修正提供依据.

在动态验证阶段, 我们引入系统运行时的行为日志与 MLTL 性质作为输入, 通过公式匹配与逻辑求解判断系统行为是否满足预设性质. 行为日志记录了关键变量的运行值 (如 a=0, b=1, c=0.5 等), 而 MLTL 性质则规定了系统在时间上的行为约束 (如 $G_{[0,10]}a$=0 表示变量 a 在 [0, 10] 时间区间内恒为 0). 当公式判断为 true, 表示行为符合预期; 若判断为 false, 则触发决策模块, 标记当前结果为异常并进行处理.

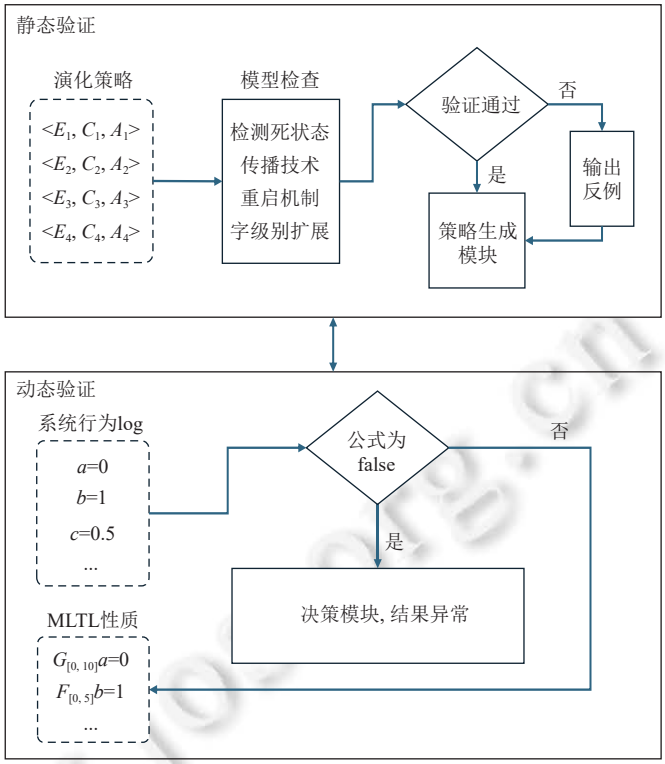


图 6 可信保障逻辑框架

通过将静态策略验证与动态行为监控有机融合, 该逻辑框架实现了对系统运行状态的全局感知与实时评估, 切实解决了资源受限、自适应性强等在轨环境下的核心难题, 不仅提升了系统的可信性和鲁棒性, 同时也为应对复杂不确定环境中的任务调度与自适应控制提供了坚实的决策保障基础.

3.4 控制软件知识库框架

控制软件知识库框架不仅为应用逻辑、可信保障逻辑和自适应控制逻辑的运行提供全面的知识支持, 同时也从这些逻辑中获取运行信息, 并形成反馈, 通过抽取和提炼后形成或更新知识, 从而保证空间飞行器控制软件的持续可信和自适应演化. 知识库由模型库、事件库、规则库、策略库和验证库这 5 个子库组成, 分别存储与自适应控制相关的各种信息和策略.

(1) 模型库主要存储自适应控制过程中的各种模型, 包括模型定义、内容和形式等信息. 模型库的核心功能是为系统的建模和仿真提供基础, 确保在各种情况下系统都能基于精确的模型进行分析和决策.

(2) 事件库以规则的形式存储异常事件到主题需求的映射关系. 事件库中包含已知和未知的异常事件信息, 确保在事件分析过程中能够快速识别和处理各种异常情况. 例如, 当系统检测到某种异常事件时, 可以通过事件库中的规则迅速确定其对应的需求, 从而启动相应的处理流程.

(3) 规则库存储从主题需求到二级需求以及最终的叶子需求的关系. 通过规则库, 系统可以将复杂的主题需求逐层分解为具体的、可实现的目标. 例如, 从宏观的“姿态异常问题排查”需求出发, 通过规则库逐步细化为“陀螺仪状态确认”“太阳敏感器状态确认”等具体叶子需求, 最终形成不可再分的叶子需求.

(4) 策略库存储叶子需求到元行为的映射关系, 并为非预期需求的满足方法提供推理支持. 策略库通过预设的 ECA 规则 (事件-条件-动作) 实现已知需求的快速匹配和处理, 同时利用人工智能算法处理非预期需求, 确保系统能够灵活应对各种复杂和动态的环境. 例如, 策略库中的 ECA 规则可以快速匹配“陀螺仪重构”的需求, 并发出相应的陀螺加断电执行指令.

(5) 验证库用于存储和管理系统约束条件, 在静态验证过程中发挥关键作用. 通过将系统的约束条件以一阶谓词逻辑公式的形式存储在验证库中, 系统可以灵活地从中调用相关约束, 进行验证和推理. 验证库还支持动态更新, 通过引入外部知识或调整已有约束, 能够迅速适应不同任务需求, 确保系统在变化的环境中持续保持高效、准确的验证能力.

演化知识库采用统一的 3 层结构体系 (如图 7 所示), 分别为标识层、关联层和完全信息层, 并在模型库、事件库、规则库、策略库和验证库这 5 类功能库中横向部署, 实现知识的结构化管理与形式化表达. 标识层负责管理知识元数据, 采用四元组表示为 $K_{id}=(Type,Name,Timestamp,Version)$, 其中 *Type* 表示知识类型 (如模型、规则、事件、策略、约束等), *Name* 为名称标识, *Timestamp* 表示创建时间, *Version* 为版本号. 该层支持基于属性的模糊检索和组合查询, 有效提升知识访问效率. 关联层建模知识之间的语义依赖关系, 统一以有向图结构 $G_{\text{rel}}=(V,E,\phi)$ 表示, 其中 V 为知识节点集合, E 为依赖边集合, $\phi: E\to\{trigger,dependency,scope\}$ 为边的语义类型映射函数. 该层支持知识间关系的动态维护, 保障演化过程中的一致性. 完全信息层是知识内容的表达核心, 5 类功能库中的不同类型知识 (如模型公式、事件映射、规则结构、策略动作与约束条件) 均在该层以结构化和形式化方式存储与调用. 例如, 模型可表示为四元组, 规则知识可用一阶逻辑表示, 策略则可表达为 ECA 规则. 通过该 3 层结构在所有知识类型中的一致部署, 系统实现知识的统一建模、可追溯演化与高效推理支撑.

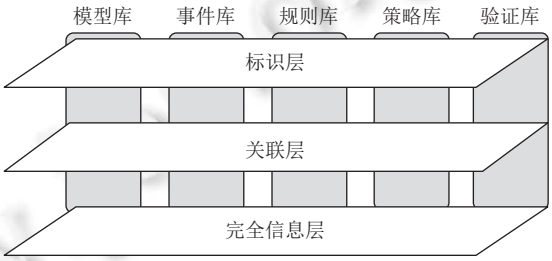


图 7 演化知识库基本结构图

此外, 知识库的管理机制采用了动态更新与版本控制相结合的策略, 以适应空间飞行器控制软件的在轨演化需求. 具体包括以下 4 个方面.

(1) 动态更新机制: 知识库通过实时监测运行状态和环境变化, 自动抽取关键数据并进行知识提炼. 例如, 当系统检测到新的异常事件时, 事件库能够通过数据挖掘算法提取事件特征, 并结合专家知识生成新的异常事件规则, 经过可信验证后对关联层中的映射关系进行更新.

(2) 版本控制机制: 为保证知识库的可靠性和可追溯性, 系统对每一条知识的变更都进行版本管理. 每次更新时, 系统会记录知识的修改历史, 包括修改时间、修改内容和修改原因等信息. 通过版本控制机制, 系统能够在需要时回滚到之前的版本, 或对不同版本的知识进行对比分析, 确保演化过程的可信性.

(3) 知识表示方法: 知识库中的相似知识采用统一的表示方法, 以兼顾表达能力和计算效率. 例如, 模型库中的数学模型采用参数化表示, 规则库和策略库中的逻辑规则采用一阶谓词逻辑或生产规则形式, 而验证库中的约束条件则以形式化验证语言进行描述.

(4) 知识推理与查询优化: 知识库内置了一套高效的推理引擎, 能够基于关联层和完全信息层的信息进行知识推理. 例如, 当系统需要处理某一复杂需求时, 推理引擎能够从规则库中逐层分解需求, 通过策略库匹配具体的执行动作, 并结合验证库确保推理结果符合约束条件. 此外, 知识库还采用了基于索引的查询优化技术, 充分利用标识层的信息加速查询过程, 减少推理时间.

4 案例分析

本节将以卫星太阳搜索系统的非预期故障、任务规划系统需求拓展两类控制软件在轨自适应演化场景为例, 进行在轨自适应演化体系设计和案例分析.

4.1 非预期故障

4.1.1 控制软件应用逻辑

控制软件作为空间飞行器的核心, 其运行在星载计算机中, 主要完成飞行器的轨道与姿态控制功能. 通过星敏感器测量航天器本体坐标系相对于某个基准坐标系的相对角位置和角速度, 以确定航天器的姿态. 星敏工作状态通过其遥测数据进行判断. 如果星敏通信正常, 地面能够正常获得测量数据, 包括姿态数据 (四元数) 和状态数据 (星敏工作模式、识别星数、姿态质量). 如果状态数据连续一段时间异常, 则认为星敏发生故障. 现有方法在星敏数据出现异常时, 无法自动应对, 需要耗费大量人力和时间成本. 在某些紧急情况下, 故障处理不及时, 甚至可能导致航天器受损或任务失败.

航天器控制软件总框架分为系统软件和应用软件两部分, 两者经过联合编译后一起存放在程序存储器 NorFlash 中. 系统软件负责维护计算机系统的正常运行以及所有的任务调度, 应用软件负责完成卫星姿态控制任务, 其中故障诊断和恢复应用逻辑如图 8 所示. 首先针对逐个敏感器和执行机构的部件进行故障诊断, 包括通信层、数据层、方案层这 3 个维度. 通信层中, 对敏感器采集数据包的帧头和校验和进行检验, 若发现错误则认为通信故障, 相应执行故障链路重启的操作, 再次发送数据传输请求. 数据层中, 对敏感器采集数据包中关键信息进行检验, 判定所采集数据有效性 (例如, 星敏数据包中识别星数应大于等于 3, 否则该星图数据无效), 若检测出该故障对相应部件进行加断电重构. 在方案层中, 根据型号定制化的方案层面需求对数据进行额外的判定或校验 (例如, 星敏常值持续 10 s 判定为数据无效), 在此情况下启用备选方案 (例如, 启用备份星敏). 之后进行组件级故障诊断, 若该故障发生则会进行相应模式转换, 避免将故障敏感器数据引入控制闭环. 最后是系统级故障诊断, 在此情况下会触发切换控制计算机或卫星停控的操作以保持卫星能源.

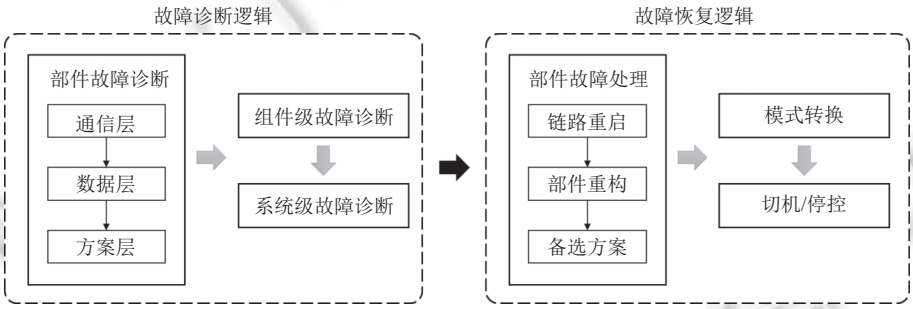
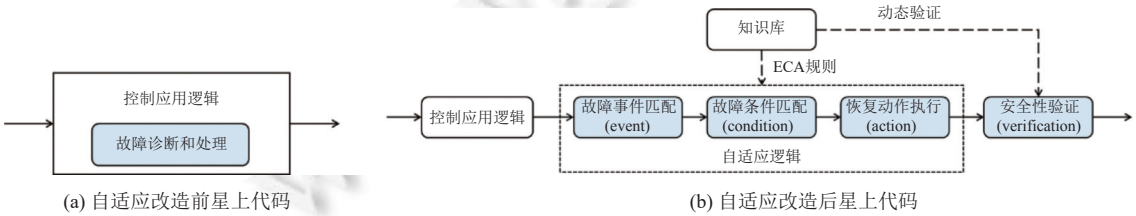


图 8 对日定向控制软件故障处理流程

4.1.2 软件自适应控制逻辑

在故障诊断案例中, 自适应改造前后的星上代码如图 9 所示. 改造前, 原星上代码故障诊断和处理功能与控制应用逻辑强耦合; 改造后, 将故障诊断和处理功能解耦, 作为独立的自适应逻辑依次实现感知 (故障事件匹配)-决策 (故障条件匹配)-执行 (恢复动作执行) 的自适应回路. 同时, 设计安全性验证模块, 用于在轨动态验证, 保障所执行动作的可靠性.



(a) 自适应改造前星上代码 (b) 自适应改造后星上代码

图 9 在轨故障诊断与恢复自适应逻辑改造框架图

根据第 3.2 节中自适应触发要素的定义, 在轨故障作为自适应触发要素之一, 会使系统从正常的工作状态, 跳转为异常的工作状态. 这一结果可以由飞行器姿态数据遥测发现, 例如 X 轴、Y 轴、Z 轴姿态角的观测值和预期值偏离 (即, 星地姿态发散), 最终导致飞行器姿态不稳定、喷气频繁、能源耗尽, 严重影响飞行器正常任务的执行和寿命, 带来不可挽回的损失.

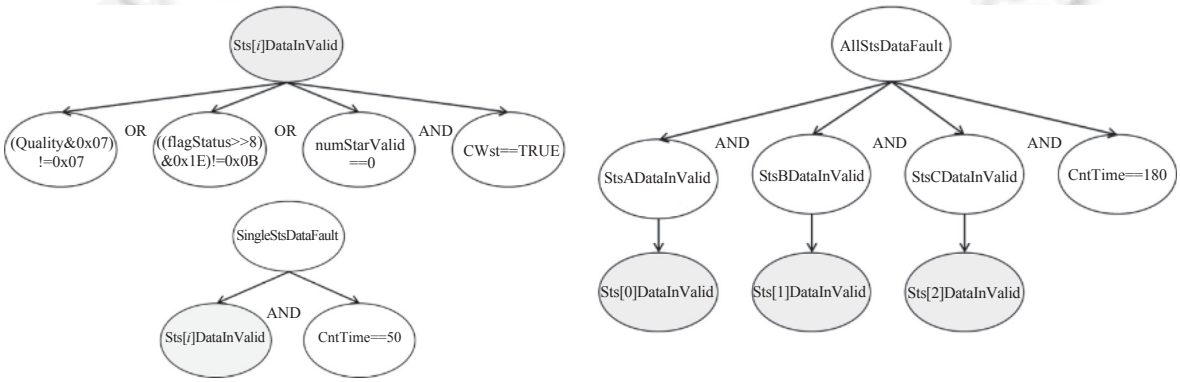
(1) 自适应触发要素分析

为了减轻和避免在轨故障所带来的影响, 首先需要对自适应触发要素进行分析. 案例中建立基于树结构的故障事件 ($e_{\text{fault\_occurrence}}$) 匹配方法, 从而触发系统从正常运行状态转移到故障检测状态. 每个控制周期, 控制软件会与星敏建立通信获取星敏采集数据, 除了对数据包的帧头、校验和进行通信层校验, 还需要对姿态数据包中关键数据的有效性进行判断, 当发生数据错误时, 则判定星敏数据无效. 表 1 中整理了案例场景中 3 种典型的星敏数据无效故障事件, 优先级依次由低到高.

表 1　星敏数据无效故障事件表

序号	事件名称	优先级
1	单个星敏数据无效累计时间达50 s	III
2	所有星敏数据无效累计时间达180 s	II
3	所有星敏数据无效累计时间达1 380 s	I

对于已知故障事件的识别, 首先通过领域知识建立故障事件树. 原星上代码的故障诊断逻辑多为 IF-ELSE 条件判断语句, 案例中抽取最外层判定逻辑作为故障事件的判定规则, 它一般描述故障触发的外部条件, 例如星敏作为控制计算机的下位机所传输的姿态数据包中星数识别错误. 由表 1 所建立的故障事件规则树如图 10 所示. 为了实现故障事件的模块化, 这里建立原子事件 Sts[i]DataInValid, 它的判定结果可以由后序遍历其叶子节点的运算结果得到, 同一层级的节点计算优先顺序为从左至右. 在原子事件的基础上, 构建事件 1 (图 10(a)) 和事件 2 (图 10(b)). 据此, 故障诊断逻辑分解为多个层级, 处理不同粒度的事件, 使故障感知过程更加清晰高效, 同时可以支持故障规则的动态添加、删除或修改, 支持规则的继承和重用, 实现规则的模块化和复用, 减少规则重复定义的工作量, 提升在轨自适应能力.



(a) 事件1　　(b) 事件2

图 10　星敏数据无效故障事件规则树

(2) 自适应可变要素生成

在自适应可变要素中, 针对已知的故障事件, 采用 ECA 规则解析的方式实现故障的诊断和恢复, 在这一过程中, 最重要的步骤就是定义自适应决策规则, 即 ECA 规则. 在此案例场景中, 故障条件匹配则是通过知识库中预先定义好的规则进行故障处理, 在由外部设备、实体等引起的事件触发故障事件的基础上, 根据系统内部条件 (例如, 系统属性、部件状态), 选择合适的恢复动作. 故障条件的表达式可以是简单的系统变量的真假判断; 也可以是

一系列关联若干系统变量的条件子句的组合判断. 表 2 中给出了星敏数据无效故障的 ECA (事件-条件-动作) 映射规则. 当故障事件匹配后, 将进行故障条件匹配, 进而制定相应的故障恢复动作.

表 2 故障诊断和恢复 ECA 规则

编号	事件 (event)	条件 (condition)	动作 (action)
1	星敏数据无效累计达50 s	—	故障星敏断电, 故障星敏重新加电
2	所有星敏数据无效累计180 s	有备份星敏且备份星敏健康	备份星敏加电
3	所有星敏数据无效累计1 380 s	使用星敏进行姿态测量, 且至少3个陀螺数据有效	1 h后转太阳捕获模式 (SAM)
4	所有星敏数据无效累计1 380 s	使用星敏进行姿态测量, 且少于3个陀螺数据有效	切控制计算机

在故障条件匹配后, 将输出的故障恢复动作精化为系统的单元目标并按步骤执行, 完成故障恢复操作. 以表 2 中第 1 条 ECA 规则的动作为例, 可以将故障恢复这一高层目标转化为针对星敏数据无效故障事件 1 特定的目标, 即故障星敏重构, 这一顶层目标又可以精化为故障星敏断电, 等待 4 s, 故障星敏加电子目标, 对应原子行为分别由断电指令、4 s 倒计时计数器和加电指令依次顺序执行. 目标精化与原子行为的对应关系详见图 11.

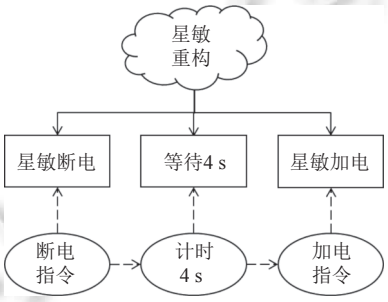


图 11 单个星敏数据无效故障恢复目标精化

4.1.3 动态策略验证

为快速定位和解决空间飞行器在轨时的异常状态, 本文开发了一种基于规则校验的运行时验证工具, 通过检查对日定向过程中航天器的日志数据, 特别是遥测数据, 来判断其是否符合指定的规则 (即约束条件). 一旦检测到不符合规则的情况, 系统会提供详细的错误信息, 明确指出违反的具体规则, 从而为进一步的故障排除和问题解决提供有力支持. 运行时验证工具的整体框架如下, 分为输入数据处理和 (单步) 运行时验证两大模块; 输入数据处理部分再分为两个子模块: 从二进制 UDP 数据包解析得到各字段对应数值变量值的二进制数据解析模块, 以及处理数值变量数据得到 LTL (线性时序逻辑) 公式中布尔变量赋值的算术表达式处理模块, 设计框架如图 12 所示.

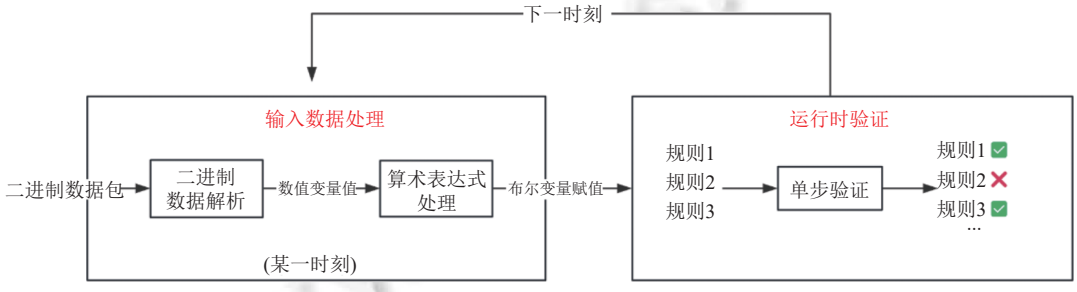


图 12 空间飞行器在轨运行时可信验证逻辑设计

在运行时验证模块中, 规则采用 LTL 公式表示. 这是因为 LTL 具有强大的逻辑表达能力, 能够包含时序逻辑和信息, 从而以更形式化的方式处理这些信息和规则. 这种方式不仅减少了遗漏和错误的风险, 还提高了处理效

率. 与嵌入在代码中的测试相比, 使用 LTL 公式的规则与被测程序更加分离. 这样做有两个主要好处: 首先, 它减少了对被测程序的影响; 其次, 规则的修改和测试变得更加容易灵活.

4.1.4 知识库设计

知识库在应用逻辑、可信保障逻辑和自适应控制逻辑的运行中提供必要的知识支持. 同时, 它通过获取应用逻辑、可信保障逻辑和自适应控制逻辑的运行信息, 形成反馈. 这些反馈经过抽取和提炼后被用于形成或更新知识, 从而确保空间飞行器控制软件能够持续进行可信自适应演化. 图 13 展示了“单个星敏数据无效”故障事件的知识示例, 用“Event”作为事件的标识符, 对于每个已知故障事件定义唯一的事件名称, 并给出详细的事件描述, 即自适应触发要因. 案例中故障事件的触发要素为存在姿态质量错误、模式错误或识别星数错误情况之一, 同时当前允许进行星敏故障诊断, 并且累计时间达到 50 s 时, 则判断该故障事件触发.

```
1 Event                                                                                    // 事件标识符
2 StsDataInvalid_Event                                                                     // 事件名称
3 ((Quality&0x07)||(((flagStatus>>8)&0x1E)==0x08)||(numStarValid!=0)&&(CWst==TRUE)&&CntTime==50s)
  // 事件描述: 单个星敏数据无效累计时间达50 s
```

图 13 已知故障“单个星敏数据无效”事件知识实例

案例中故障恢复规则以<*E*, *C*, *A*> 三元组的形式在知识库中存储, 以此将自适应触发要素和自适应可变要素关联起来. 对于“单个星敏数据无效故障”, 有规则<StsDataInvalid_Event, StsDataInvalid_Condition, StsDataInvalid_Action>, 其中 *C* 代表根据当前自适应触发要素, 选择合适的自适应可变要素的前提条件. 图 14 给出关于此条件的实例. “Condition”作为条件的标识符, 为每个已知故障触发要素和可变要素确定唯一的转化条件, 并给出详细的条件描述. 案例中当触发故障事件后, 进行如下的条件判断再匹配合适的故障恢复行为.

```
1 Condition                 // 条件标识符, 关键字保留
2 StsDataInvalid_Condition  // 条件名称
3 NULL                      // 条件描述, 即条件的触发条件, NULL代表条件永真
```

图 14 已知故障“单个星敏数据无效”条件知识实例

图 15 展示了“单个星敏数据无效”故障恢复策略在知识库中的形式. “Action”作为行为的标识符, 对于每个已知故障恢复策略定义唯一的行为名称, 并给出详细的行为描述, 即自适应可变要素. 案例中当触发故障事件后, 进行规则匹配, 完成如图 12 所示的故障星敏重构操作, 包括断电、等待 4 s、加电这 3 步操作.

```
1 Action                                                  // 行为标识符
2 StsDataInvalid_Act                                      // 行为名称
3 <(∅, Sts_PowerOff), (Time=4s, ∅), (∅, Sts_PowerOn)>     // 行为描述: 故障星敏重启
```

图 15 已知故障“单个星敏数据无效”策略知识实例

后文图 16 展示了目前运行时验证工具所支持的部分简单规则. 将规则解析为 LTL 公式的过程, 分为以下两步: (a) 提取原子事件: 从规则中提取出原子事件, 并使用算术表达式来表示. 这些表达式将在后续的专门解析过程中进行处理. (b) 编码表达规则: 基于已提取的原子事件, 按照 LTL 公式的语法对这些规则进行编码表达.

4.1.5 仿真实验结果

针对空间飞行器对日定向任务场景, 搭建控制软件在轨自适应演化仿真系统, 仿真界面如图 17 所示. 在依次点击“初始化”“开始仿真”控件后, 依次发送校时、轨道注入、转 IPM 对地模式的遥控指令, 使得卫星正常进入 IPM 惯性指向模式. 此时通过右侧的“ECA 输入框”注入表 2 所示的星敏数据无效 ECA 规则, 这里按照预定义的 ECA 语法结构编写规则, 以便于后台程序自动解析.

空间飞行器在轨时的实时系统状态可以通过遥测数据包下传得到, 结果如图 18 中的遥测界面所示, 其中的曲线图为飞行器 XYZ 三轴姿态角, 列表则记录当前系统状态的部分关键变量. 平滑曲线为星上真实姿态, 三角点曲

线为动力学仿真计算结果, 开始仿真时选用星敏 A 定姿, 飞行器星上真实姿态和地面计算结果一致. 当飞行器运行至第 300 s 时, 通过图 12 中的仿真系统注入星敏 A 常值故障, 遥测界面显示星地姿态发生偏差. 此时在建立好的 ECA 规则库的基础上, 在第 330 s 时修改 ECA 规则, 将星敏常值故障作为数据无效原子事件 Sts[i]DataInValid 的一种并完成规则上注. 发现当星敏常值故障连续发生 50 s 后, 触发了星敏数据无效故障的 ECA 规则, 即故障事件 1. 由于星敏 A 数据无效, 飞行器选用星敏 B 定姿, 星地姿态偏差减小, 飞行器恢复正常.

原始规则	定义的原子事件	原子事件的算术表达式表示	与原始规则等效的LTL公式
飞行阶段值恒定为11	p_1: 飞行阶段值为11	p_1: *flightPhase*=11	$G(p_1)$
对日定向阶段标志按照时间先后顺序0→1→2, 不可跨越或者颠倒顺序	p_2, p_3, p_4: 对日定向阶段标志分别为0, 1, 2	p_2: *phaseMark*=0 p_3: *phaseMark*=1 p_4: *phaseMark*=2	$G(p\rightarrow X(F(p\rightarrow X(F(p_4)))))$ 可以改为$G(p_2\rightarrow)\wedge G(p_3\rightarrow)$
左模拟太阳敏感器有效标志和右模拟太阳敏感器有效标志这两个标志均有效后, “对日定向阶段标志”才会1→2	p_5: 左模拟太阳敏感器有效标志有效 p_6: 右模拟太阳敏感器有效标志有效	p_5: *leftSimulationSunMark*=true p_6: *rightSimulationSunMark*=true	$G((p_3\wedge Xp_4)\rightarrow(p_5\wedge p_6))$
船时值单调递增	p_7: 船时值随时刻单调递增	p_7: *shiptime*++	$G(p_7)$
“对日定向成功标志”由未成功变为成功必须在子模式2下	p_8: “对日定向成功标志”为成功 子模式2, 即对日定向阶段标志为2, 对应原子事件p_4	p_8: *successMark*=true	$G((\neg p_8\wedge Xp_8)\rightarrow p_4)$ $G((\neg p_8\wedge Xp_8)\rightarrow p_4\wedge Xp_4)$
对日定向成功后, 子模式保持在2下	—	—	$G(p_8\rightarrow G(p_4))$

图 16 在轨动态验证 LTL 规则知识实例

图 17 空间飞行器控制软件在轨自适应演化仿真系统界面

在实验过程中加入未知的星敏常值故障的测试验证, 通过对比传统固化策略与本文自适应方法的表现, 验证软件自适应控制逻辑的有效性. 实验设置两种场景: (a) 已知故障: 注入表 1 定义的 III 级星敏数据无效事件; (b) 非预期故障: 模拟未预定义的星敏常值故障. 实验结果表明, 传统方法没有非预期故障的处理能力, 需地面人工分析后注入新规则, 耗时为小时级; 而本文方法通过基于人工智能方法的动态规则扩展, 成功识别大部分非预期故障, 平均响应时间维持在毫秒级. 实验结果表明, 本文方法使系统对新故障类型的响应效率提升 3 个数量级, 且减少了

人工干预需求, 直接印证了框架在经验归纳与推理机制方面的改进.

在实验过程加入星敏常值故障的性质验证, 可以检测到性质不通过的验证结果, 实现空间飞行器在轨动态验证. 经过实验测试, 运行时验证工具能够准确高效地检测到被违反的规则, 具体结果如图 19 所示.

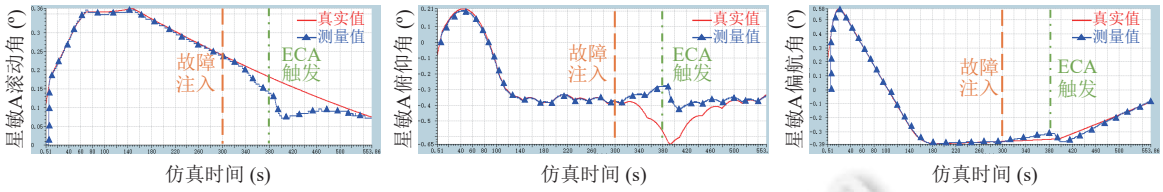


图 18 空间飞行器控制软件在轨遥测显示界面

```
please input the formula(input 'finish' to end input):
G(STS_A_on->STS_A_error)
please input the formula(input 'finish' to end input):
G(STS_B_on->STS_B_error)
please input the formula(input 'finish' to end input):
G(STS_C_on->STS_C_error)
please input the formula(input 'finish' to end input):
finish
(false R ((! STS_A_on) | STS_A_error))
(false R ((! STS_B_on) | STS_B_error))
(false R (STS_C_error | (! STS_C_on)))
-------------------------program START-------------------
```

图 19 星敏常值性质验证动态验证结果

在相同的实验条件下, 对本文所研究的空间飞行器控制软件在轨自适应演化仿真系统进行了 10 次任务仿真测试. 表 3 展示了控制软件在轨自适应演化仿真系统的仿真结果. 在测试中, 对于知识库中存在的所有故障均能够做到 100% 识别. 对于 I 级故障, 故障响应的平均时间为 0.26 ms, 远低于可接受响应时间标准. 对于 II 级和 III 级故障, 相比 I 级故障, 用时会相对较久, 造成该现象的原因主要是该系统采用严格的分级检测机制, 按照 I 级 (最高)→II 级→III 级 (最低) 的固定顺序执行故障判断. 当存在高优先级故障时, 必须完成当前优先级的全流程判断后, 才会进入下一级检测. 这种串行处理模式导致 II/III 级故障检测耗时包含其之前所有优先级任务的判断时间. 同时, 系统在每个优先级判断周期内存在反馈时隙, 用于执行状态同步、结果回传等操作. 当连续处理多个优先级任务时, 这些离散的反馈时段会形成累积延迟. 总体而言, 本文所研究的自适应演化仿真系统将故障响应的时间控制在毫秒级, 证明该方法具有高效的故障检测能力.

表 3 控制软件在轨自适应演化仿真系统测试结果

故障优先级	故障检测成功率 (%)	故障响应平均时间 (ms)	故障检测平均时间 (ms)	平均延迟时间 (ms)	可接受的响应时间 (ms)
I	100	0.26	0.21	0.05	1.28
II	100	4.85	0.62	4.23	10.24
III	100	6.34	0.47	5.87	38

在保障实时故障诊断的同时, 该系统在内存管理方面展现出卓越的优化成效. 卫星数据处理采用流式处理模式, 配合基于优化数据结构的性质公式解析方案, 使得核心功能的内存开销接近理论下限. 然而, 在扩展规范性质处理范围时, 系统面临数据类型支持广度与二进制文件体积、运行效率之间的本质性矛盾. 为此, 本文设计了分层式轻量级数学公式解析架构: 基础层通过整型、浮点型等基本数据类型的逻辑运算模块 (涵盖大小比较等核心操作) 保障基础功能; 增强层则采用指定位宽的原码/补码精准解析技术, 实现对复杂数据类型的扩展支持. 该架构在维持 800 KB 紧凑体积的前提下, 成功平衡了功能扩展性与存储效率.

为实现上述性能指标, 相较于传统基于预构建自动机的验证范式, 本文在技术实现层面采用公式传播 (formula progression) 动态计算方法. 该方法通过 3 项核心机制显著提升系统效能: 1) 按需计算机制仅在实际需求触发时展开性质公式运算, 避免冗余计算; 2) 即时内存回收策略确保内存占用严格与当前处理的性质公式复杂度成正比; 3) 并行处理不同的性质, 保障毫秒级实时响应能力. 实验数据显示, 与传统方法相比, 该技术在典型应用场

景下内存占用降低 40%–60%, 同时保持毫秒级实时响应能力, 为航天器在复杂空间计算环境中的可靠运行提供了关键技术支撑.

4.2 业务变更

4.2.1 指令规划软件应用逻辑

业务的顺利实现是航天任务成功的基本保障, 尤其是对非预期业务的正确应对是空间飞行器智能化的重要体现. 当航天器业务内容发生变更时, 控制软件需要进行相应的调整和更新. 以可见光成像为例, 当成像对象、成像时间、成像模式等基本需求发生变化时, 飞行器的指令和指令序列需要相应调整或重构. 传统方法依赖于地面人工修改, 并在通信时间窗口内上注至星上, 该过程费时费力, 且存在一定的安全风险. 若控制软件具备自适应演化能力, 则仅需要接收任务变更信息, 由控制软件自主分析新增业务需求, 并生成相应的控制策略, 实时更新飞行器载荷设备参数, 从而确保针对业务变更的顺利响应.

在空间飞行器中, 任务实现以业务逻辑为核心, 图 20 展示了空间飞行器任务的基本实现流程. 首先, 对任务基本信息、通信窗口以及其他相关信息进行分析, 以全面了解任务的具体要求. 当任务被判定为可行时, 通过任务预处理和规划制定详细的任务计划. 随后, 进入指令序列处理阶段, 指令构造模块会生成相应的指令, 这些指令通过分系统约束消解进行校验和优化, 确保各个子系统能够协调工作. 此外, 经过指令序列规划生成最终的指令序列, 经过验证后, 由遥测方上注至空间飞行器中, 确保任务的顺利执行.

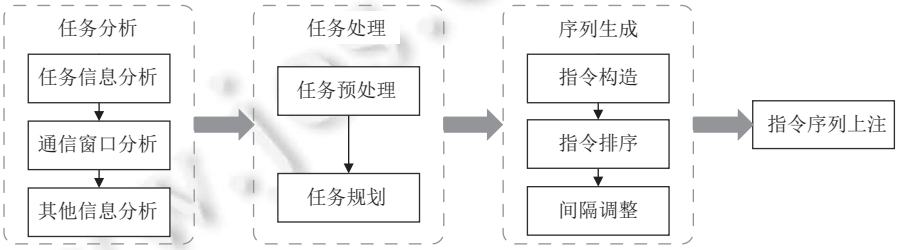


图 20 空间飞行器任务基本实现流程

现有任务处理系统依赖于地面预制的静态业务规则, 这导致在面对非预期任务时难以灵活调整业务逻辑. 由于无法适应不断变化的任务需求, 这种缺乏灵活性的机制常导致处理效率低下, 甚至可能引发任务失败, 限制了空间飞行器在复杂环境中执行任务的能力.

4.2.2 软件自适应控制逻辑

在业务变更的情景中, 自适应事件的触发依赖于现有的任务管理方法. 本文重点探讨自适应系统中的可变因素, 具体分为基于参数变更的指令构造, 以及面向业务实现的指令序列生成两种类型. 基于参数变更的指令构造涉及在系统参数调整后, 重新构造所需指令, 以通过动态调整指令来应对系统内外部的变化. 面向业务实现的指令序列生成则关注在业务需求变化时, 如何高效生成符合新需求的指令序列. 此过程旨在提升系统的灵活性和响应能力, 确保在业务变更时能够迅速适应新的操作要求, 保障任务的顺利进行.

(1) 基于参数变更的指令构造

对业务实现规则进行分析, 指令构造过程可变因素以参数设定为主. 图 21 展示了指令构造业务规则变更的样例. 在该样例中, 当任务采用 2 号相机, 并以第 3 种类型进行成像时, 所构造的指令代码应为 11H, 且该指令的持续时长为 120 s. 由于卫星任务的调整, 预设的 120 s 成像时长无法满足现有业务背景, 因此, 通过外置自适应控制软件识别可变更的代码参数, 并采用修订策略对参数进行调整, 最终确保实现业务变更.

(2) 面向业务实现的指令序列生成

军事、救灾等突发场景对空间飞行器紧急响应非预期任务提出了挑战. 例如, 高分辨率成像卫星能够执行持续成像和数据传输两种任务类型. 通常情况下, 该卫星在成像后会进行图像处理, 并在下一个周期内进行数据传输. 然而, 当自然灾害突然发生时, 地面遥测中心可能需要该卫星实现图像的实时传输. 基于本文自适应控制逻辑

及框架对指令排序和间隔调整过程进行演化. 图 22 展示了非预期任务实时成像实现的关键过程样例. 对于“高分成像”这一已知业务需求, 可以将其分解为姿态控制、高分传感器控制以及指令管理等关键子需求. 同样, “数据传输”需求可以分解为指令管理、数据处理、链路建立以及传输控制等关键子需求. 每个子需求进一步细化, 直至成为不可再分的叶子需求.

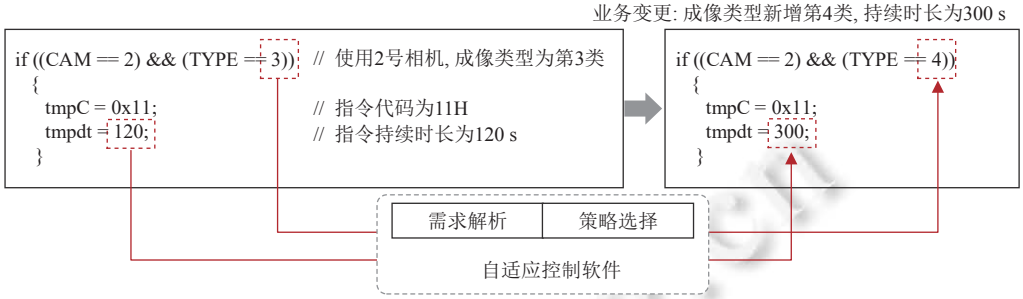


图 21 指令构造业务规则变更样例

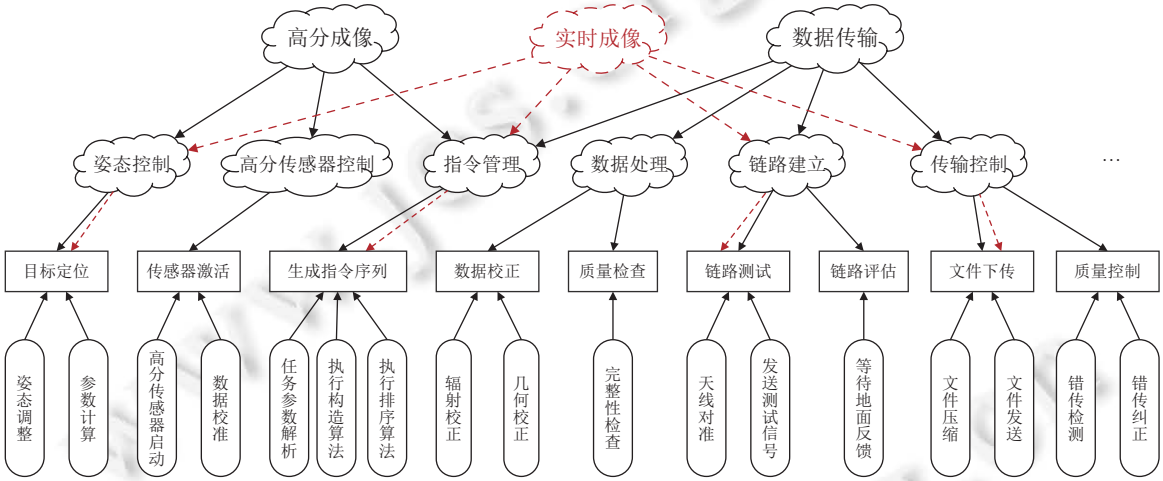


图 22 非预期任务“实时成像”实现的关键过程样例

对于非预期需求“实时成像”, 首先将成像任务的基本信息以常规任务方式上传至空间飞行器. 任务处理系统通过分析任务信息生成业务实现需求 $R_f=(r_1, r_2)$, 其中, r_1 为成像需求, r_2 为数据传输需求. 随后, 系统根据需求分解过程, 将该需求进一步细化为具体的二级需求, 包括姿态控制、指令管理、链路建立以及传输控制. 每个二级需求通过进一步分解形成预设的最小叶子需求, 最终, 由策略知识引导对应的元行为进行组合, 以实现非预期任务“实时成像”.

4.2.3 静态指令验证

为了提高卫星指令序列自主处理系统的可靠性, 提出一种基于约束求解的 SMT (satisfiability modulo theory) 静态验证方法. 该方法通过将单个子系统作为独立模块进行建模和验证, 确保各子系统之间的互不干扰操作, 并通过中央星务系统实现全局协调, 从而提高系统整体的稳健性.

具体而言, 每个子系统会将其操作的约束条件以自然语言描述, 并转换为一阶谓词逻辑公式. 接着, 这些公式会提交给 SMT 求解器处理. 如果求解器判断这些逻辑条件无法同时满足, 则意味着指令序列存在冲突, 便于迅速识别并修正问题. 相反, 如果所有条件均能满足, 则表明指令序列符合预期要求. 此外, 该方法还利用 SMT 求解器生成导致冲突的不可满足核心, 通过分析这些核心, 可以精准定位问题指令及其相关冲突的约束条件, 从而加快问题解决, 设计框架如图 23 所示.

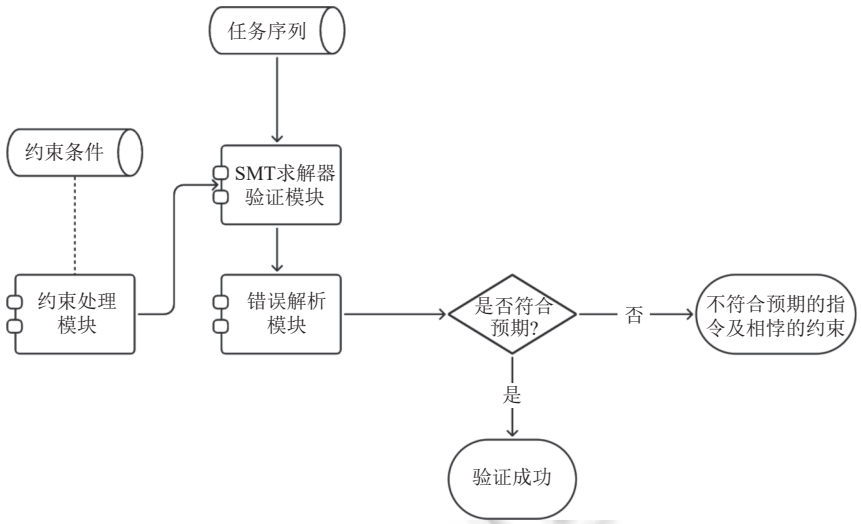


图 23　空间飞行器指令序列静态验证框架图

在静态验证阶段, 约束条件被定义为一阶谓词逻辑公式. 一阶谓词逻辑因其强大的表达能力, 不仅能够全面覆盖系统的各类约束, 还可以通过外部知识库灵活更新, 简化了系统约束条件的维护和调整.

4.2.4　知识库设计

图 24 展示了非预期业务变更“实时成像”事件的知识实例. 其中, 事件属性值为“Real-time Imaging Task”, 表示该事件为实时成像任务. 该事件关联了两个需求, 即高分成像和数据传输, 分别对应需求 ID “R1”和“R2”. 这些需求通过“Requirements”属性进行描述. 其中, 需求 R1 (High-resolution Imaging): 要求实现高分辨率的成像功能. 需求 R2 (Data Transmission): 要求实现数据的实时传输功能.

```
"Event": "Real-time Imaging Task",          // "事件": "实时成像任务",
"Requirements": {                           // "需求": {
  "R1": "High-resolution Imaging",          // "R1": "高分成像",
  "R2": "Data Transmission"                 // "R2": "数据传输"
}                                           //
```

图 24　非预期事件“实时成像任务”事件知识实例

图 25 展示了规则库中“高分成像”需求分解知识实例. 在该实例中, 主题需求为高分成像 (High-resolution Imaging), 此需求进一步细分为 3 个二级需求: 姿态控制 (Attitude Control, ID: R1.1), 确保航天器在成像过程中能够稳定且精确地控制姿态, 以保证成像质量; 高分传感器控制 (High-resolution Sensor Control, ID: R1.2), 对成像传感器进行精确的控制和校准, 以获取高分辨率的图像数据; 指令管理 (Instruction Management, ID: R1.3), 涉及成像任务相关指令的生成、调度和执行, 以确保成像任务按计划进行.

```
"Requirement": "High-resolution Imaging",                    // 主题需求: 高分成像
"Sub-requirements": [                                        // 二级需求映射规则
  { "ID": "R1.1", "Name": "Attitude Control" },              // 姿态控制
  { "ID": "R1.2", "Name": "High-resolution Sensor Control" }, // 高分传感器控制
  { "ID": "R1.3", "Name": "Instruction Management" }         // 指令管理
]
```

图 25　“高分成像”需求分解知识实例

图 26 展示了“目标定位”叶子需求的 ECA 规则知识实例. 在该实例中, 叶子需求为目标定位 (Target Positioning), 并通过 ECA (事件-条件-动作) 规则详细描述了其对应的行为. 具体规则为: 当事件为发生姿态控制需求 (Need for Attitude Control), 且条件为检测到姿态偏差 (Attitude Deviation Detected) 时, 执行姿态调整 (Perform Attitude Adjustment) 的动作; 当事件为发生参数计算需求 (Need for Parameter Calculation), 且条件为检测到参数异常 (Parameters Anomaly Detected) 时, 执行参数更新 (Perform Parameter Update) 的动作.

```
"Sub-requirement": "Target Positioning",          // 子需求: 目标定位
"ECA Rules": [                                    // ECA规则列表
  {
    "Event": "Need for Attitude Control",         // 事件: 发生姿态控制需求
    "Condition": "Attitude Deviation Detected",   // 条件: 检测到姿态偏差
    "Action": "Perform Attitude Adjustment"       // 动作: 执行姿态调整
  },
  {
    "Event": "Need for Parameter Calculation",    // 事件: 发生参数计算需求
    "Condition": "Parameters Anomaly Detected",   // 条件: 检测到参数异常
    "Action": "Perform Parameter Update"          // 动作: 执行参数更新
  }
]
```

图 26 “目标定位”叶子需求应对策略知识实例

4.2.5 仿真实验结果

针对非预期业务变更设计指令序列生成仿真系统. 通过上传“成像”和“数传”两种任务序列模拟地面遥测方发布非预期任务; 通过上传星务约束、控制约束与数传约束模拟非预期任务约束变更, 实现对新型任务特性的补充; 此外, 由于不同型号航天器运行环境、功能、载荷等差异性, 设计载荷约束上传模块模拟多型号. 图 27 展示了非预期任务指令序列自适应处理仿真界面.

图 27 非预期任务实时成像实现的关键过程

在相同实验条件下, 对常规任务处理系统和本文所研究自适应任务处理系统进行 10 次任务仿真测试. 表 4 展示了两种任务处理系统的仿真结果. 指令构造成功率采用保留一位小数的方式进行统计. 结果显示, 自适应任务处

理系统的指令构造成功率为 96.8%, 相比常规任务处理系统的 98.6% 降低了 1.8%; 同时, 任务实现成功率也降低了 10%. 这一现象的主要原因在于航天任务的实现与多个子系统的协同工作密切相关, 而本次实验中未充分考虑非关键子系统的影响, 从而导致部分任务未能成功完成.

表 4 自适应任务处理系统仿真结果

仿真系统	指令构造成功率 (%)	常规任务实现成功率 (%)	非预期任务实现成功率 (%)	非预期任务实现时间开销 (s)
常规任务处理系统	98.6	100	20	43
自适应任务处理系统	96.9	90	60	7

在非预期任务的处理能力方面, 自适应任务处理系统表现出显著优势, 任务实现成功率达到 60%, 相比常规任务处理系统提升了 40%. 为进一步评估自适应任务处理系统在应对非预期任务需求时的时间开销优势, 实验中额外引入了模拟传统场景的专家人工变更软件环节, 以实现非预期任务处理的对比分析. 结果表明, 在常规任务处理系统中, 非预期任务的平均实现时间为 43 s, 而自适应任务处理系统仅需 7 s. 尽管任务的复杂程度对时间开销具有直接影响, 部分简单任务能够通过快速调整实现, 但常规任务处理系统由于需要精准定位修改位置, 以及源代码修改和二次编译, 显著增加了时间开销.

上述结果表明, 自适应任务处理系统在非预期任务的处理效率和响应能力方面表现出明显优势.

5 结 论

随着航天事业的发展, 空间飞行器智能自主能力吸引了越来越多的研究关注. 本文提出了一种空间飞行器控制软件在轨自适应可信演化框架, 并阐述了自适应控制逻辑、可信保障逻辑和知识库设计这 3 个方面的关键技术, 为提高空间飞行器自主能力提供了参考, 弥补了传统在轨飞行器控制软件无法主动调节软件行为的不足, 如当空间飞行器感知到空间碎片逼近, 控制软件应自适应调节轨道高度, 规避后再返回原轨道. 提高了应对不确定环境与未知故障的处理效率, 以及在资源受限环境下在轨实时验证保障可靠性. 以上 3 方面为空间飞行器控制软件智能化发展奠定了基础. 除此之外, 本文在策略生成和在轨修正阶段采用了基于 ECA 的规则匹配方法, 但在航天系统资源受限的情况下, 仍存在优化空间, 特别是在演化知识库的构建方面, 如知识库的规模、查询效率和更新速度等, 均需进一步提升. 同时, 在实时验证阶段, 性质验证器模块基于有穷状态自动机, 其状态规模受验证规范的复杂性影响, 资源占用问题仍存在优化空间. 未来, 随着这些关键技术的进一步突破, 软件自适应演化技术将为空间飞行器提供更为出色的在轨应变能力.

References

[1] 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]

[2] Chien S, Sherwood R, Tran D, *et al*. The EO-1 autonomous science agent. In: Proc. of the 3rd Int'l Joint Conf. on Autonomous Agents and Multiagent Systems, 2004. New York: IEEE, 2004. 420–427.

[3] Wang Y, Wu HX. Space rendezvous and berthing and study of the control method. Journal of Astronautics, 2001, 22(5): 15–19 (in Chinese with English abstract). [doi: 10.3321/j.issn:1000-1328.2001.05.003]

[4] Zhang HH, Liang J, Huang XY, Zhao Y, Wang L, Guan YF, Cheng M, Li J, Wang PJ, Yu J, Yuan L. Autonomous hazard avoidance control for Chang'E-3 soft landing. SCIENTIA SINICA Technologica, 2014, 44(6): 559–568 (in Chinese with English abstract). [doi: 10.1360/092014-51]

[5] Chen H. Research on key issues in high assurance system software of embedded systems [Ph.D. Thesis]. Chengdu: University of Electronic Science and Technology of China, 2018 (in Chinese with English abstract).

[6] Batory D. Feature models, grammars, and propositional formulas. In: Proc. of the 9th Int'l Conf. on Software Product Lines. Rennes: Springer, 2005. 7–20. [doi: 10.1007/11554844_3]

[7] Arcelli D. Exploiting queuing networks to model and assess the performance of self-adaptive software systems: A survey. Procedia Computer Science, 2020, 170: 498–505. [doi: 10.1016/j.procs.2020.03.108]

[8] Cheng BHC, Sawyer P, Bencomo N, Whittle J. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: Proc. of the 12th Int'l Conf. on Model Driven Engineering Languages and Systems. Denver: Springer, 2009. 468–483. [doi: 10.1007/978-3-642-04425-0_36]

[9] Chu S, Koe J, Garlan D, Kang E. Integrating graceful degradation and recovery through requirement-driven adaptation. In: Proc. of the 19th IEEE/ACM Int'l Symp. on Software Engineering for Adaptive and Self-managing Systems. Lisbon: IEEE, 2024. 122–132. [doi: 10.1145/3643915.3644090]

[10] Aradea, Supriana I, Surendro K. Self-adaptive software modeling based on contextual requirements. TELKOMNIKA, 2018, 16(3): 1276–1288. [doi: 10.12928/TELKOMNIKA.v16i3.7032]

[11] Moreno GA, Cámara J, Garlan D, Schmerl B. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo: ACM, 2015. 1–12. [doi: 10.1145/2786805.2786853]

[12] Cetina C, Giner P, Fons J, Pelechano V. Autonomic computing through reuse of variability models at runtime: The case of smart homes. Computer, 2009, 42(10): 37–43. [doi: 10.1109/MC.2009.309]

[13] Salehie M, Pasquale L, Omoronyia I, Ali R, Nuseibeh B. Requirements-driven adaptive security: Protecting variable assets at runtime. In: Proc. of the 20th IEEE Int'l Requirements Engineering Conf. Chicago: IEEE, 2012. 111–120. [doi: 10.1109/RE.2012.6345794]

[14] Filho RR, Alberts E, Gerostathopoulos I, Porter B, Costa FM. Emergent Web server: An exemplar to explore online learning in compositional self-adaptive systems. In: Proc. of the 2022 Int'l Symp. on Software Engineering for Adaptive and Self-managing Systems. Pittsburgh: IEEE, 2022. 36–42. [doi: 10.1145/3524844.3528079]

[15] Lee E, Kim YG, Seo YD, Seol K, Baik DK. RINGA: Design and verification of finite state machine for self-adaptive software at runtime. Information and Software Technology, 2018, 93: 200–222. [doi: 10.1016/j.infsof.2017.09.008]

[16] Ding ZH, Zhou Y, Zhou MC. Modeling self-adaptive software systems by fuzzy rules and Petri nets. IEEE Trans. on Fuzzy Systems, 2018, 26(2): 967–984. [doi: 10.1109/TFUZZ.2017.2700286]

[17] Maia PH, Vieira L, Chagas M, Yu YJ, Zisman A, Nuseibeh B. Dragonfly: A tool for simulating self-adaptive drone behaviours. In: Proc. of the 14th IEEE/ACM Int'l Symp. on Software Engineering for Adaptive and Self-managing Systems (SEAMS). Montreal: IEEE, 2019. 107–113. [doi: 10.1109/SEAMS.2019.00022]

[18] Zhang QY. Design and implementation of workflow integrated with rule engine on SaaS platform [MS. Thesis]. Beijing: Beijing University of Posts and Telecommunications, 2017 (in Chinese with English abstract).

[19] Jiang YP, Wang W, Shi BL, Dong JR. Model and behavioral determinism theory for ECA rules. Ruan Jian Xue Bao/Journal of Software, 1997, 8(3): 190–196 (in Chinese with English abstract). https://www.jos.org.cn/jos/article/abstract/19970305

[20] Vlahavas I, Bassiliades N. Parallel, Object-oriented, and Active Knowledge Base Systems. New York: Springer, 1998. [doi: 10.1007/978-1-4757-6134-4]

[21] Zhang Z. Research on technologies of rules in spatial database [MS. Thesis]. Changsha: National University of Defense Technology, 2010 (in Chinese with English abstract).

[22] Hu LP, Liang XL, He LL, Zhang JQ, Ren BX, Qi D. Construction method of aviation swarm decision rule base based on scenario analysis. Acta Aeronautica et Astronautica Sinica, 2020, 41(S1): 723737 (in Chinese with English abstract). [doi: 10.7527/S1000-6893.2019.23737]

[23] Li QS, Wang L, Chu H, Zhang M. Agent-based software adaptive dynamic evolution mechanism. Ruan Jian Xue Bao/Journal of Software, 2015, 26(4): 760–777 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4757.htm [doi: 10.13328/j.cnki.jos.004757]

[24] Moreno G, Kinneer C, Pandey A, Garlan D. DARTSim: An exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems. In: Proc. of the 14th IEEE/ACM Int'l Symp. on Software Engineering for Adaptive and Self-managing Systems (SEAMS). Montreal: IEEE, 2019. 181–187. [doi: 10.1109/SEAMS.2019.00031]

[25] Liu DW, Yang YJ. Particle swarm algorithm based on chaos local search and its application. Computer Technology and Development, 2021, 31(4): 216–220 (in Chinese with English abstract). [doi: 10.3969/j.issn.1673-629X.2021.04.037]

[26] Wang L, Huo QE, Li QS, Wang Z, Jiang YX. Self-adaptation decision-making based on parallel search optimization for command and control information system. Ruan Jian Xue Bao/Journal of Software, 2022, 33(5): 1774–1799 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6561.htm [doi: 10.13328/j.cnki.jos.006561]

[27] Wu T, Li QS, Wang L, He L, Li YJ. Using reinforcement learning to handle the runtime uncertainties in self-adaptive software. In: Proc. of the 2018 STAF Collocated Workshops on Software Technologies: Applications and Foundations. Toulouse: Springer, 2018. 387–393. [doi: 10.1007/978-3-030-04771-9_28]

[28] Bahrampour S, Ramakrishnan N, Schott L, Shah M. Comparative study of deep learning software frameworks. arXiv:1511.06435, 2016.

[29] Verstaevel N, Boes J, Nigon J, d'Amico D, Gleizes MP. Lifelong machine learning with adaptive multi-agent systems. In: Proc. of the 9th

Int'l Conf. on Agents and Artificial Intelligence. Porto, 2017. 275−286. [doi: 10.5220/0006247302750286]

[30] Wang XZ, Zhang TL, Wang R. Noniterative deep learning: Incorporating restricted Boltzmann machine into multilayer random weight neural networks. IEEE Trans. on Systems, Man, and Cybernetics: Systems, 2019, 49(7): 1299–1308. [doi: 10.1109/TSMC.2017.2701419]

[31] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing Atari with deep reinforcement learning. arXiv:1312.5602, 2013.

[32] Zhuang FZ, Luo P, He Q, Shi ZZ. Survey on transfer learning research. Ruan Jian Xue Bao/Journal of Software, 2015, 26(1): 26–39 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4631.htm [doi: 10.13328/j.cnki.jos.004631]

[33] Aravind R, Shah CV. Physics model-based design for predictive maintenance in autonomous vehicles using AI. Int'l Journal of Scientific Research and Management, 2023, 11(9): 932–946. [doi: 10.18535/ijsrm/v11i09.ec06]

[34] Yokoyama N, Clegg A, Truong J, Undersander E, Yang TY, Arnaud S, Ha S, Batra D, Rai A. ASC: Adaptive skill coordination for robotic mobile manipulation. IEEE Robotics and Automation Letters, 2024, 9(1): 779–786. [doi: 10.1109/LRA.2023.3336109]

[35] Abbaspour A, Yen KK, Forouzannezhad P, Sargolzaei A. A neural adaptive approach for active fault-tolerant control design in UAV. IEEE Trans. on Systems, Man, and Cybernetics: Systems, 2020, 50(9): 3401–3411. [doi: 10.1109/TSMC.2018.2850701]

[36] Loveland DW. Automated Theorem Proving: A Logical Basis. Fundamental Studies in Computer Science Vol. 6. New York: North-Holland Publishing Company, 1978.

[37] Clarke EM, Grumberg O, Peled D. Model Checking. Cambridge: MIT Press, 1999.

[38] Clarke EM, Emerson EA, Sifakis J. Model checking: Algorithmic verification and debugging. Communications of the ACM, 2009, 52(11): 74–84. [doi: 10.1145/1592761.1592781]

[39] Baier C, Katoen JP. Principles of Model Checking. Cambridge: MIT Press, 2008.

[40] Bertot Y, Castéran P. Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions. Berlin: Springer, 2004. [doi: 10.1007/978-3-662-07964-5]

[41] Paulson LC. Isabelle: The next 700 theorem provers. In: Odifreddi P, ed. Logic and Computer Science. Berlin: Academic Press, 1990. 361–385.

[42] de Moura L, Ullrich S. The Lean 4 theorem prover and programming language. In: Proc. of the 28th Int'l Conf. on Automated Deduction. Springer, 2021. 625–635. [doi: 10.1007/978-3-030-79876-5_37]

[43] Klein G, Elphinstone KJ, Heiser G, Andronick J, Cock D, Derrin P, Elkaduwe D, Engelhardt K, Kolanski R, Norrish M, Sewell T, Tuch H, Winwood S. seL4: Formal verification of an OS kernel. In: Proc. of the 22nd ACM SIGOPS Symp. on Operating Systems Principles. Big Sky Montana: ACM, 2009. 207–220. [doi: 10.1145/1629575.1629596]

[44] Leroy X. Formal verification of a realistic compiler. Communications of the ACM, 2009, 52(7): 107–115. [doi: 10.1145/1538788.1538814]

[45] Wang HM, Yuan Y, Liu ZY, Shen JH, Yin YC, Xiong J, Xie EZ, Shi H, Li Y, Li L, Yin J, Li ZG, Liang XD. DT-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In: Proc. of the 61st Annual Meeting of the Association for Computational Linguistics, Vol. 1 (Long Papers). Toronto: ACL, 2023. 12632–12646. [doi: 10.18653/v1/2023.acl-long.706]

[46] Liu XP, Liu HZ, Yi XD, Wang J. LLM-enhanced theorem proving with term explanation and tactic parameter repair*. In: Proc. of the 15th Asia-Pacific Symp. on Internetware. Macao: ACM, 2024. 21–30. [doi: 10.1145/3671016.3674823]

[47] Vogel T. mRUBiS: An exemplar for model-based architectural self-healing and self-optimization. In: Proc. of the 13th IEEE/ACM Int'l Symp. on Software Engineering for Adaptive and Self-managing Systems. Gothenburg: IEEE, 2018. 101–107.

[48] Holzmann GJ. The model checker spin. IEEE Trans. on Software Engineering, 1997, 23(5): 279–295. [doi: 10.1109/32.588521]

[49] Kroening D, Tautschnig M. CBMC-C bounded model checker. In: Proc. of the 20th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Grenoble: Springer, 2014. 389–391. [doi: 10.1007/978-3-642-54862-8_26]

[50] Cavada R, Cimatti A, Dorigatti M, Griggio A, Mariotti A, Micheli A, Mover S, Roveri M, Tonetta S. The NUXMV symbolic model checker. In: Proc. of the 26th Int'l Conf. on Computer Aided Verification. Vienna, 2014. 334–342. [doi: 10.1007/978-3-319-08867-9_22]

[51] Bradley AR. SAT-based model checking without unrolling. In: Proc. of the 12th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation. Austin: Springer, 2011. 70–87. [doi: 10.1007/978-3-642-18275-4_7]

[52] Li JW, Zhu SF, Zhang YL, Pu GG, Vardi MY. Safety model checking with complementary approximations. In: Proc. of the 2017 IEEE/ACM Int'l Conf. on Computer-aided Design (ICCAD). Irvine: IEEE, 2017. 95–100. [doi: 10.1109/ICCAD.2017.8203765]

[53] Xia YC, Becchi A, Cimatti A, Griggio A, Li HW, Pu GG. Searching for i-good lemmas to accelerate safety model checking. In: Proc. of the 35th Int'l Conf. on Computer Aided Verification. Paris: Springer, 2023. 288–308. [doi: 10.1007/978-3-031-37703-7_14]

[54] Su YH, Yang QS, Ci Y. Predicting lemmas in generalization of IC3. In: Proc. of the 61st ACM/IEEE Design Automation Conf. San Francisco: ACM, 2024. 208. [doi: 10.1145/3649329.3655970]

[55] Holzmann GJ. Mars code. Communications of the ACM, 2014, 57(2): 64–73. [doi: 10.1145/2560217.2560218]
[56] Chien PC, Lee NZ. CPV: A circuit-based program verifier. In: Proc. of the 30th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Luxembourg City: Springer, 2024. 365–370. [doi: 10.1007/978-3-031-57256-2_22]
[57] Wang W, Zhang N, Tian C, Duan ZH, Xu ZJ, Yu CF. Verifying chips design at RTL level. In: Proc. of the 17th Int'l Symp. on Theoretical Aspects of Software Engineering. Bristol: Springer, 2023. 146–163. [doi: 10.1007/978-3-031-35257-7_9]
[58] Johannsen C, Jones P, Kempa B, Rozier KY, Zhang P. R2U2 version 3.0: Re-imagining a toolchain for specification, resource estimation, and optimized observer generation for runtime verification in hardware and software. In: Proc. of the 35th Int'l Conf. on Computer Aided Verification. Paris: Springer, 2023. 483–497. [doi: 10.1007/978-3-031-37709-9_23]
[59] Chai M, Wang HF, Tang T, Liu HJ. Runtime verification of train control systems with parameterized modal live sequence charts. Journal of Systems and Software, 2021, 177: 110962. [doi: 10.1016/j.jss.2021.110962]
[60] Ceci M, Sannier N, Abualhaija S, Shin D, Bianculli D, Halling M. Toward automated compliance checking of fund activities using runtime verification techniques. In: Proc. of the 2024 IEEE/ACM Workshop on Software Engineering Challenges in Financial Firms. Lisbon: IEEE, 2024. 19–20
[61] Tsigkanos C, Nenzi L, Loreti M, Garriga M, Dustdar S, Ghezzi C. Inferring analyzable models from trajectories of spatially-distributed Internet of Things. In: Proc. of the 14th IEEE/ACM Int'l Symp. on Software Engineering for Adaptive and Self-managing Systems (SEAMS). Montreal: IEEE, 2019. 100–106. [doi: 10.1109/SEAMS.2019.00021]

附中文参考文献

[1] 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]
[3] 王颖, 吴宏鑫. 基于椭圆轨道的空间交会停靠全系数自适应控制方法研究. 宇航学报, 2001, 22(5): 15–19. [doi: 10.3321/j.issn:1000-1328.2001.05.003]
[4] 张洪华, 梁俊, 黄翔宇, 赵宇, 王立, 关轶峰, 程铭, 李骥, 王鹏基, 于洁, 袁利. 嫦娥三号自主避障软着陆控制技术. 中国科学: 技术科学, 2014, 44(6): 559–568. [doi: 10.1360/092014-51]
[5] 陈昊. 高可信嵌入式系统软件的关键技术研究 [博士学位论文]. 成都: 电子科技大学, 2018.
[18] 张源清. SaaS 平台下融合规则引擎的工作流服务的设计与实现 [硕士学位论文]. 北京: 北京邮电大学, 2017.
[19] 姜跃平, 汪卫, 施伯乐, 董继润. ECA 规则的模型和行为特定理论. 软件学报, 1997, 8(3): 190–196. https://www.jos.org.cn/jos/article/abstract/19970305
[21] 张震. 空间数据库规则技术研究 [硕士学位论文]. 长沙: 国防科学技术大学, 2010.
[22] 胡利平, 梁晓龙, 何吕龙, 张佳强, 任宝祥, 齐铎. 基于情景分析的航空集群决策规则库构建方法. 航空学报, 2020, 41(S1): 723737. [doi: 10.7527/S1000-6893.2019.23737]
[23] 李青山, 王璐, 褚华, 张曼. 一种基于智能体技术的软件自适应动态演化机制. 软件学报, 2015, 26(4): 760–777. http://www.jos.org.cn/1000-9825/4757.htm [doi: 10.13328/j.cnki.jos.004757]
[25] 刘道文, 杨拥军. 基于混沌局部搜索的粒子群算法及其应用. 计算机技术与发展, 2021, 31(4): 216–220. [doi: 10.3969/j.issn.1673-629X.2021.04.037]
[26] 王璐, 霍其恩, 李青山, 王展, 姜宇轩. 基于并行搜索优化的指控系统自适应决策方法. 软件学报, 2022, 33(5): 1774–1799. http://www.jos.org.cn/1000-9825/6561.htm [doi: 10.13328/j.cnki.jos.006561]
[32] 庄福振, 罗平, 何清, 史忠植. 迁移学习研究进展. 软件学报, 2015, 26(1): 26–39. http://www.jos.org.cn/1000-9825/4631.htm [doi: 10.13328/j.cnki.jos.004631]

作者简介

李晓锋, 研究员, CCF 专业会员, 主要研究领域为可信软件, 软件自适应, 智能化软件工程.
罗懿行, 博士, 工程师, CCF 专业会员, 主要研究领域为软件需求工程, 软件自适应.
王路桥, 博士, 工程师, CCF 专业会员, 主要研究领域为软件自适应, 任务指令智能规划.
董晓刚, 研究员, 主要研究领域为嵌入式软件, 软件自适应.
李建文, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为形式化验证.
顾斌, 博士, 研究员, 博士生导师, 主要研究领域为可信软件, 嵌入式系统, 计算机控制.
董云卫, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为嵌入式系统, 形式化验证, 可信软件.
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系统设计, 可信软件, 计算机控制.