Software Engineering and Applications 软件工程与应用, 2026, 15(4), 519-532
Published Online August 2026 in Hans. https://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2026.154048

Hans汉斯

高速公路机电软件制品协同部署方法研究

张　言[1]，李东博[1]，杨　娜[2*]

[1]河北高速公路集团有限公司承德分公司，河北　承德
[2]哈尔滨工业大学(威海)汽车工程学院，山东　威海

收稿日期：2026年5月29日；录用日期：2026年7月29日；发布日期：2026年8月6日

摘　要

针对高速公路机电系统中异构设备接入复杂、软件对象分散管理以及部署区域、安全边界和异常恢复策略难以统一约束等问题，提出一种高速公路机电软件制品协同部署方法。该方法将应用程序、AI应用、原子化服务、业务规则和设备插件等对象统一抽象为版本化制品，构建制品元模型和部署策略模型，将设备类型、接入协议、部署范围、授权条件和回滚关系等因素纳入统一描述，并设计“制品规范化生成－部署任务构建－云－边－端协同执行－运行反馈处置”的闭环流程，实现软件对象的统一审核、分发、部署和异常恢复。以梅岭隧道和老营盘隧道风速风向仪替换接入场景进行验证，并结合设备插件实例说明基于制品元数据、部署策略和运行反馈的快速回滚处置过程。结果表明，采用该方法后，设备接入周期由9天缩短至1天，应用部署时间由9 s缩短至5 s，故障响应时间由100 s缩短至10 s，分别降低约89%、44%和90%。结果说明，该方法能够提高异构设备接入效率和部署过程可控性，对高速公路机电系统软件对象统一治理具有一定参考价值。

关键词

高速公路机电系统，软件制品，协同部署，云－边－端协同，异构设备接入

Research on a Collaborative Deployment Method for Highway Electromechanical Software Artifacts

Yan Zhang[1], Dongbo Li[1], Na Yang[2*]

[1]Chengde Branch of Hebei Expressway Group Co., Ltd., Chengde Hebei
[2]School of Automotive Engineering, Harbin Institute of Technology (Weihai), Weihai Shandong

Received: May 29, 2026; accepted: July 29, 2026; published: August 6, 2026

*通讯作者。

文章引用: 张言, 李东博, 杨娜. 高速公路机电软件制品协同部署方法研究[J]. 软件工程与应用, 2026, 15(4): 519-532.
DOI: 10.12677/sea.2026.154048

Abstract

To address the problems of complex heterogeneous device integration, decentralized management of software objects, and the difficulty of uniformly constraining deployment areas, security boundaries, and exception recovery strategies in highway electromechanical systems, a collaborative deployment method for highway electromechanical software artifacts is proposed. The method uniformly abstracts applications, AI applications, atomic services, business rules, and device plugins as versioned artifacts, constructs an artifact meta-model and a deployment strategy model, incorporates device types, access protocols, deployment scopes, authorization conditions, and rollback relationships into a unified description, and designs a closed-loop process consisting of artifact standardized generation, deployment task construction, cloud-edge-device collaborative execution, and operation feedback handling. In this way, unified review, distribution, deployment, and exception recovery of software objects are realized. The method is verified in the replacement and integration scenario of wind speed and direction sensors in Meiling Tunnel and Laoyingpan Tunnel. In addition, a device plugin example is used to illustrate the rapid rollback handling process based on artifact metadata, deployment strategies, and operation feedback. The results show that, after adopting the proposed method, the device integration cycle is shortened from 9 days to 1 day, the application deployment time from 9 s to 5 s, and the fault response time from 100 s to 10 s, with reductions of approximately 89%, 44%, and 90%, respectively. The results indicate that the proposed method can improve the efficiency of heterogeneous device integration and the controllability of the deployment process, providing a reference for unified governance of software objects in highway electromechanical systems.

Keywords

Highway Electromechanical System, Software Artifacts, Collaborative Deployment, Cloud-Edge-Device Collaboration, Heterogeneous Device Integration

1. 引言

随着交通基础设施数字化、网络化和智能化建设不断推进，高速公路机电系统正由传统人工运维向平台化、协同化和智能化运维转型。《交通强国建设纲要》和《数字交通发展规划纲要》均强调推动交通基础设施数字化、网联化和智能化发展，为高速公路机电系统运维模式升级提供了政策基础[1] [2]。近年来，公路交通数字孪生、隧道运营管理创新应用和隧道机电智能运维平台等研究不断开展，为交通基础设施全生命周期管理提供了技术支撑[3]-[5]。同时，随着收费、通信、监控、供配电和隧道机电等设备规模持续扩大，设备类型多、协议差异大、软件对象分散、版本维护困难和现场处置依赖人工经验等问题日益突出[6]-[8]。

从软件工程和新型计算系统角度看，低代码开发、边缘计算、应用商店化分发和远程更新等技术，为复杂设备环境下的软件快速构建、分发和运维提供了新的条件。低代码开发能够通过模型化、可视化和参数化方式提升应用构建效率[9]；边缘计算可将计算、存储和服务能力下沉至网络边缘，提升分布式场景下的实时响应能力[10] [11]；应用商店模式在应用分发、版本维护和服务组织方面具有较强的管理能力[12]；面向物联网场景的安全 OTA 更新研究则表明，可信签名、策略控制和版本治理是异构设备环境

下保障软件持续交付的重要基础[13]。

近年来，云－边－端协同、云原生编排和微服务部署进一步推动了异构设备环境下的软件协同交付。相关研究表明，边缘计算驱动的物联网系统能够提升现场响应效率[14]，而资源调度、任务分配、服务编排和运行监测是云－边－端协同系统中的关键问题[15]。云原生工作负载在边缘侧的编排部署研究，为跨云端、边缘侧和终端侧的软件部署提供了参考[16]；云－边微服务架构和服务编排方法，也为复杂现场环境中的服务拆分、部署管理和运行协同提供了工程路径[17]。此外，低代码开发、基础设施即代码、持续集成和 DevSecOps 等研究表明，通过可视化建模、配置化环境描述、自动化构建测试和安全控制前置，可以提升软件交付的标准化、可追溯性和生命周期治理能力[18]-[21]。

上述研究为复杂设备环境下的软件构建、分发和运维提供了重要支撑，但高速公路机电系统在工程实践中仍具有较强的行业场景特征。首先，底层设备高度异构且协议繁杂。收费、监控、通信、供配电和隧道机电等系统中仍存在大量控制器、传感器和执行设备。由于部分设备通过串口、RS485、CAN 总线或厂商私有协议接入，上层平台通常需要借助网关、驱动或协议适配机制，将异构设备抽象为统一的对象接口[22]。其次，现场运行环境具有严格的物理与网络边界。机电系统与现场设备运行状态紧密关联，不同路段、隧道、收费站、机房和边缘节点之间存在明确的物理区域、网络边界和运维权限，其安全管理必须同时兼顾物理环境、系统可靠性和运行安全性要求[23]。因此，在高速公路机电场景中，软件部署不能仅停留在通用应用分发或服务编排层面，现有的通用 DevOps 或 IoT 管理平台在面对上述场景时，往往存在对异构设备兼容性差、以及对安全隔离和物理区域强关联的策略表达能力不足等缺陷。必须将该场景特有的设备接入方式、安全隔离机制以及物理区域强关联等约束条件纳入考量。

基于上述分析，面向高速公路机电系统这类典型云－边－端协同场景，现有研究仍有必要进一步结合行业特征，探索多类型软件对象的统一组织方式和可控部署过程。针对上述异构设备接入复杂、软件对象分散管理以及部署区域与安全边界约束难以统一表达等问题，本文结合 KaihongOS 的分布式协同能力与 KaihongOS Meta 的低代码开发能力，提出一种面向高速公路机电设备的软件制品协同部署方法，并基于“冀鸿 AppStore”完成平台化实现。该方法将应用、AI 应用、原子化服务、业务规则和设备插件等对象统一抽象为版本化软件制品，通过制品元模型、部署策略模型和运行反馈机制，将设备类型、接入协议、部署区域、授权范围和运行状态等约束纳入部署过程，实现软件对象的审核、授权、分发、部署、监测和异常处置闭环。

本文的主要工作如下：

(1) 构建软件制品统一治理框架，将应用、AI 应用、业务规则、原子化服务和设备插件等对象纳入统一生命周期管理过程；

(2) 设计云－边－端协同的软件制品分发与部署机制，由云端负责制品上架、版本管理和策略配置，由边缘侧承担预测试、策略解析和约束检查，由端侧完成部署执行、运行反馈和异常响应；

(3) 以梅岭隧道和老营盘隧道风速风向仪替换接入场景为例，对所提方法在设备接入、应用部署和故障响应方面进行验证，分析其在提升软件交付效率、增强部署可控性和降低现场运维复杂度方面的效果。

2. 软件制品协同部署设计

2.1. 问题定义与设计目标

高速公路机电系统涉及收费、通信、监控、供配电及隧道机电等多类设备，软件对象在类型、部署位置和维护方式上具有明显差异。传统依赖现场人工处理和脚本式运维的方式已难以满足异构设备环境下的以下需求：

(1) 统一治理需求。需要将应用、插件、规则、模型等不同形态的软件对象纳入统一管理框架，实现标识、版本、依赖、权限、部署范围和回退关系的一致管理。

(2) 协同部署需求。需要面向高速公路机电系统“云端统筹、边缘验证、端侧执行”的运行特点，构建云－边－端协同的软件制品分发与部署机制，弥补传统物联网平台在物理区域强关联与安全隔离策略表达上的不足，提高异构环境下的软件交付效率与部署可控性。

(3) 闭环处置需求。需要将制品部署、运行监测、异常处置和版本回退等过程关联起来，形成从制品生成到故障恢复的生命周期闭环，以降低现场处置成本并提升运维可追溯性。

基于上述需求，本文以“软件制品”为统一管理对象，构建面向高速公路机电系统的软件制品协同部署方法。该方法通过制品元模型统一描述多类型软件对象，通过部署策略约束制品分发范围和执行条件，并通过运行反馈与版本处置机制实现部署过程的可管、可控与可追溯。

2.2. 软件制品元模型与对象类型

为提高异构设备环境下软件治理的一致性，本文将高速公路机电系统中的软件交付对象统一定义为“软件制品”。软件制品是能够被独立描述、审核、分发、部署、监测和回退的软件交付单元，也是版本管理、权限控制、策略配置和故障处置的关联载体。本文将软件制品定义为一个八元组：

$$A = <I, V, D, E, P, T, S, R>$$

其中，I 表示制品标识，用于区分不同软件对象；V 表示版本信息，用于支持版本管理和更新控制；D 表示依赖关系，用于描述制品运行所需的系统、组件或服务条件；E 表示适配环境，用于限定制品可运行的设备类型、操作系统或边缘节点环境；P 表示权限约束，用于描述制品访问设备、数据或接口的权限范围；T 表示目标对象，用于标识制品作用的设备、控制器、网关或应用环境；S 表示部署范围，用于限定制品分发的路段、隧道、设备组或用户范围；R 表示回滚关系，用于描述异常情况下的版本回退或恢复路径。

通过上述形式化定义，应用程序、AI 应用、原子化服务、业务规则和设备插件等不同类型的软件对象能够在统一元模型下完成审核、分发、部署、监测和回退管理。为适配高速公路机电设备“多协议、多终端、多场景”的特点，本文将冀鸿 AppStore 管理的软件制品划分为五类，如表 1 所示。

Table 1. Software artifacts of Jihong AppStore
表 1. 冀鸿 AppStore 的软件制品

软件制品	特性
KaihongOS 应用	基于 KaihongOS UI 框架与 API 开发的前端应用/后台服务
AI 应用	以容器化或服务化方式交付的 AI 能力，面向边端推理与事件识别
原子化服务	轻量、即用即走、无需安装
业务规则应用	通过可视化编排快速形成场景化业务逻辑，实现跨设备联动与流程自动化
设备插件应用	集合协议解析、驱动与接入能力，以插件化方式降低多厂商设备集成成本

2.3. 部署策略模型与基础服务

在软件制品元模型基础上，制品能否被正确分发和执行，还取决于部署策略的约束。部署策略用于描述制品从云端下发至边缘侧和端侧过程中的目标范围、时间窗口、授权条件和异常恢复方式。为避免与制品元模型中的权限约束 P 混淆，本文将部署策略记为 S_d，定义如下：

$$S_d = <G, W, A, C, B>$$

其中，G 表示目标设备组，用于限定制品部署的设备集合或网关节点；W 表示部署时间窗口，用于约束制品下发和执行的时间范围；A 表示授权范围，用于确定制品可被订阅、下载和运行的用户或组织边界；C 表示约束条件，用于描述网络状态、设备状态、版本兼容性和依赖满足情况；B 表示回滚策略，用于规定异常情况下的暂停分发、重新下发、版本切换或回退方式。

基于制品元模型和部署策略模型，冀鸿 AppStore 为多类型制品提供基础管理服务，包括上架服务、管理功能、签名机制和分发系统四类，如表 2 所示。

Table 2. Basic services of Jihong AppStore
表 2. 冀鸿 AppStore 基础服务

服务类型	功能
上架服务	为开发者提供应用提交、资料生成与上架流程入口
管理功能	支持应用状态监控、版本更新与维护，形成可持续迭代能力
签名机制	保障制品来源可信与责任可追溯，为安全分发提供基础
分发系统	面向私有化用户提供订阅、下载、安装与更新能力

上述基础服务分别支撑制品提交、状态维护、可信校验和分发安装，共同形成“提交–管理–签名–分发–安装–更新”的制品服务流程。与传统按应用、插件或配置文件分别管理的方式不同，本文方法通过统一的制品元模型和部署策略模型，将多类型软件对象纳入同一生命周期治理框架，从而为云–边–端协同部署提供基础。

2.4. 闭环运维流程设计

在统一制品定义和部署策略约束的基础上，本文进一步构建面向高速公路机电系统的软件制品闭环运维流程。该流程将制品生成、审核治理、策略配置、分发部署、运行监测和异常处置纳入统一过程，不再将“部署”和“运维”视为彼此独立的阶段。

如图 1 所示，闭环流程主要包括制品提交、审核治理、策略配置、分发部署、订阅使用、运行反馈和故障处置等环节。开发者负责完成应用、插件或规则等制品的生成与提交；平台根据制品元模型对其进行规范性、兼容性和签名审核，并依据部署策略配置授权范围、目标设备组和部署时间窗口；用户通过应用市场完成制品订阅、下载、部署与使用，并在运行过程中回传告警、日志和执行状态。当出现运行异常时，平台根据运行反馈触发相应的版本处置流程，使异常处理结果重新回流至制品管理流程。

为保证闭环流程的有效运行，本文在总体设计中遵循以下原则：

(1) 统一描述原则。所有软件对象均按照制品元模型进行描述，避免因对象类型不同导致管理字段、版本记录和部署流程不一致。

(2) 策略约束原则。制品分发和部署过程均需绑定目标设备组、授权范围、时间窗口和约束条件，以降低误下发、版本不匹配和大范围故障扩散风险。

(3) 反馈驱动原则。运行日志、告警信息和故障处置结果应与制品版本关联，使运行反馈能够作为版本处置和后续迭代的依据。

因此，冀鸿 AppStore 的总体设计并非单一平台功能堆叠，而是面向高速公路机电系统提出的一种软件制品统一定义与协同部署方法。后续章节将在此基础上进一步阐述制品生成、协同部署过程及案例验证结果。

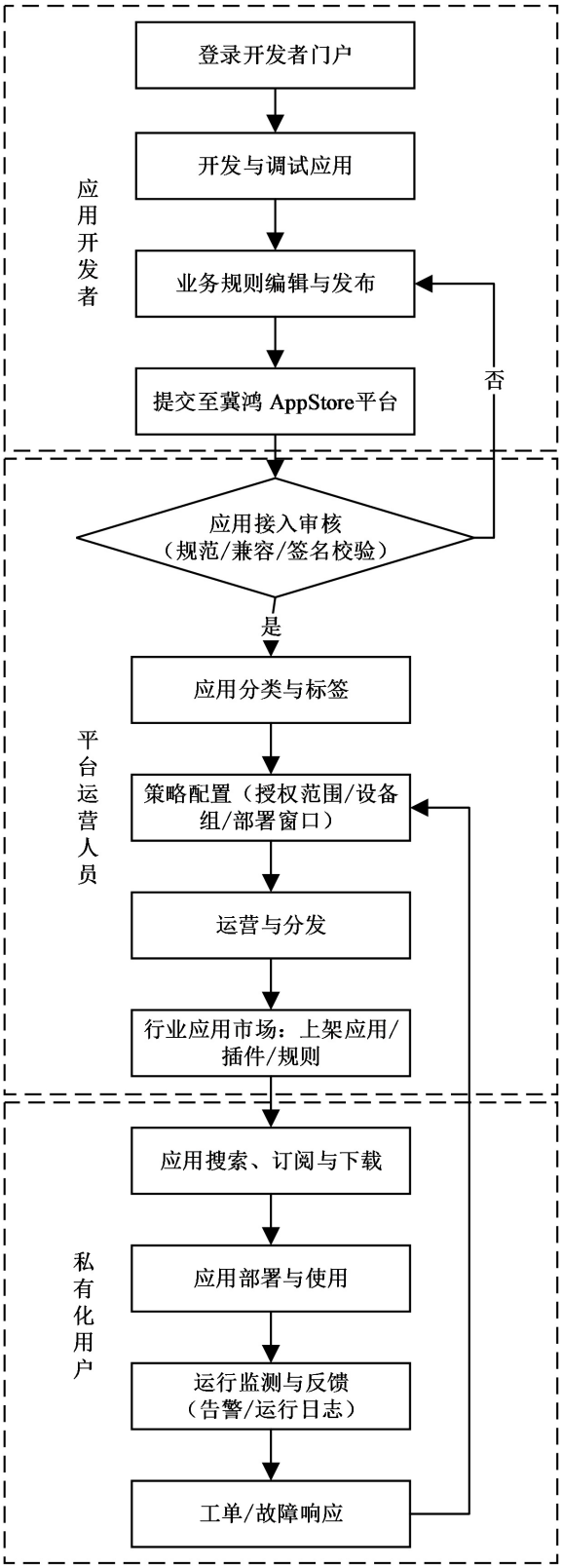

Figure 1. Closed-Loop operation and maintenance process of Jihong AppStore
图 1. 冀鸿 AppStore 闭环运维流程图

3. 软件制品协同部署方法

在第 2 章完成软件制品元模型、部署策略模型和闭环流程设计的基础上，本章进一步说明所提方法的执行过程。该方法以软件制品为基本交付单元，以部署策略为约束条件，围绕“制品规范化生成－部署任务构建－云－边－端协同执行－运行反馈处置”四个环节展开，实现多类型软件对象从平台治理到端侧运行的有序衔接。

3.1. 制品规范化生成

制品规范化生成的目标，是将原本分散存在的应用程序、设备插件、业务规则、AI 应用和原子化服务等软件对象，转换为可审核、可签名、可分发和可回退的标准制品。该过程并不改变各类软件对象的业务功能，而是通过统一的元数据补全、规范校验和交付物封装，使其能够进入同一生命周期管理流程。制品规范化生成流程如下：

(1) 元数据补全。开发者在提交制品时补全制品标识、版本、依赖、适配环境、权限、目标对象、部署范围和回滚关系等信息，为后续审核、部署和异常恢复提供基础。

(2) 规范与兼容性校验。平台对制品元数据完整性、目标运行环境匹配性、依赖条件和权限声明进行检查，避免不完整或不兼容的制品进入分发环节。

(3) 交付物封装。根据制品类型生成应用包、镜像、服务包、规则包或插件包等标准交付物，使不同类型软件对象具备统一的分发和安装入口。

(4) 签名与上架。平台对通过校验的制品进行签名和上架处理，形成可追溯、可订阅和可部署的版本化制品。

在实现上，冀鸿 AppStore 基于 KaihongOS Meta 提供低代码化制品生成能力。KaihongOS 应用主要通过工程配置、构建签名和上架资料生成形成应用包；AI 应用主要通过模型转换、服务封装和运行环境固化形成镜像或模型插件包；原子化服务通过元数据配置和服务逻辑封装形成服务包；业务规则通过触发条件、处理逻辑和执行动作的可视化编排形成规则包；设备插件通过协议能力封装和物模型映射形成插件包。五类制品的输入、交付物和部署位置如表 3 所示。

Table 3. Inputs, deliverables, and deployment locations of five types of artifacts
表 3. 五类制品输入、交付物及部署位置对照表

制品类型	主要输入	交付物	部署位置
KaihongOS 应用	工程配置、构建产物、签名资料	应用包	端侧(应用市场/运行环境)
AI 应用	模型文件、转换参数、服务代码	镜像/模型插件包	边缘/端侧(推理组件)
原子化服务	服务逻辑、卡片配置、元数据	服务包	端侧
业务规则应用	规则编排、节点参数、联动对象	规则包	边缘侧验证 + 端侧执行
设备插件	插件包、物模型映射、产品模板绑定	插件包	端侧(控制器)

3.2. 部署任务构建

制品完成规范化生成后，平台依据部署策略构建部署任务，将制品包、目标设备组、授权范围、部署时间窗口、约束条件和回滚策略等信息进行统一绑定。与传统人工逐台安装或脚本下发方式不同，部署任务将部署对象与部署条件关联起来，使制品分发过程具备明确的目标范围和执行边界。具体而言，平台首先根据路段、隧道、设备组、控制器或网关节点等信息确定部署对象，避免制品误下发至不相关

设备；随后对制品版本、目标环境、依赖关系、授权状态和部署时间窗口进行检查，确保制品满足分发条件；最后为部署任务绑定异常恢复策略，为后续运行反馈和异常处置提供依据。通过上述部署任务构建，软件部署过程由“人工判断后执行”转变为“策略约束下执行”，从而提高了部署范围、执行时机和异常处置方式的可控性。

3.3. 云－边－端协同执行

如图 2 所示，制品部署过程采用云－边－端协同方式完成。云端负责制品审核、版本管理和部署任务生成；边缘侧负责任务接收、环境检查、预测试和下发控制；端侧负责制品安装、加载、运行和状态反馈。

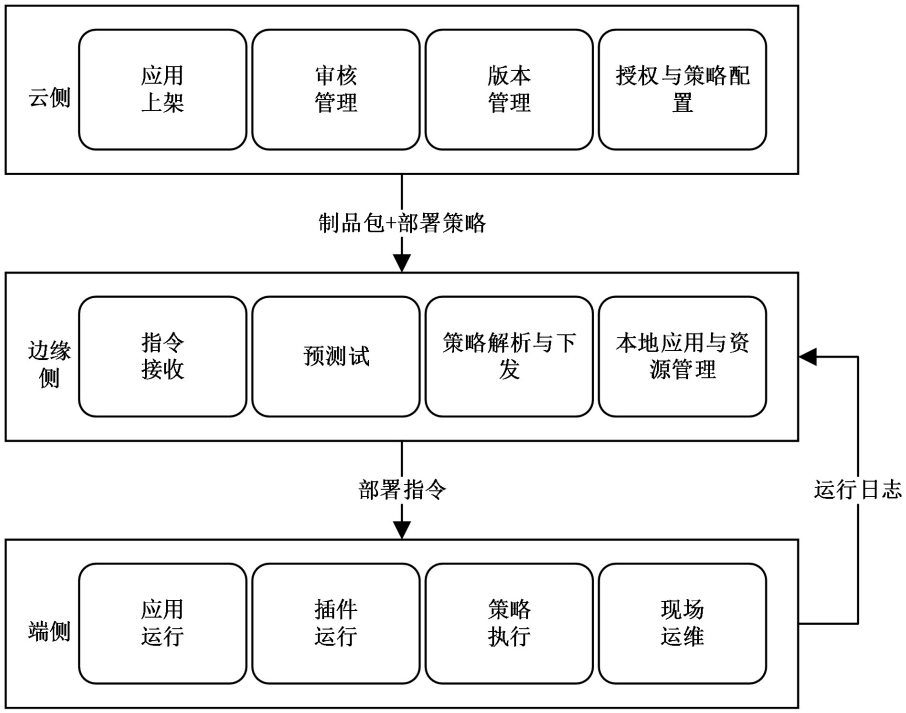

Figure 2. Cloud-Edge-Device collaborative distribution and deployment process of Jihong AppStore
图 2. 冀鸿 AppStore 云－边－端协同分发部署流程

云端首先根据制品状态和部署策略生成部署任务，并将“制品包 + 部署策略”下发至边缘侧。边缘侧接收任务后，对目标设备在线状态、运行环境、网络状态和版本兼容性进行检查；当条件满足时，边缘侧将制品下发至端侧设备。端侧根据制品类型执行安装、加载或更新，并将执行结果、运行日志和异常信息回传至边缘侧和云端。

不同制品在端侧执行方式上存在差异。应用类制品主要完成安装、更新和卸载；设备插件类制品主要完成协议解析、数据采集和指令下发；规则类制品主要完成事件响应和设备联动；AI 应用主要完成模型推理和事件识别。虽然交付物形态不同，但其执行过程均遵循“云端管理－边缘验证－端侧执行－状态反馈”的协同机制。

3.4. 运行反馈与异常处置

制品部署完成后，端侧持续回传运行状态、日志和告警信息。平台根据运行反馈判断制品执行结果，并将异常信息与制品版本、部署任务和回滚策略关联。当出现安装失败、运行异常、设备通信异常或版

本兼容性问题时，平台按照预设策略执行处置。异常处置主要包括以下方式：

(1) 暂停分发。当同一制品在多个目标设备上出现相似异常时，平台暂停后续分发任务，避免异常范围扩大。

(2) 重新下发。当异常由网络中断、临时状态异常或端侧执行失败引起时，平台可在条件恢复后重新下发部署任务。

(3) 版本切换。若当前版本与目标环境不匹配，平台会切换至已验证版本，以保障业务连续性。

(4) 版本回退。当制品部署后影响设备数据采集、协议解析或业务联动时，平台依据回滚关系恢复至上一稳定版本。

通过运行反馈与异常处置机制，软件制品部署过程由一次性下发转变为可监测、可追踪和可恢复的生命周期管理过程。

4. 对照基线与方法特性分析

为说明所提方法相较传统方式的改进作用，本文将传统人工运维和脚本式部署方式作为对照基线，并从部署对象、部署控制、异常处置和适用边界等方面进行分析。

4.1. 对照基线

传统运维中，高速公路机电系统的软件升级、设备接入和配置变更通常依赖现场人工或远程脚本逐台处理。其基本流程为：提出需求后，由现场人员或厂家技术人员进行设备适配和软件配置，随后开展设备联调和运行验证；当运行异常出现后，再通过人工排查、配置修改或手工恢复完成处置。

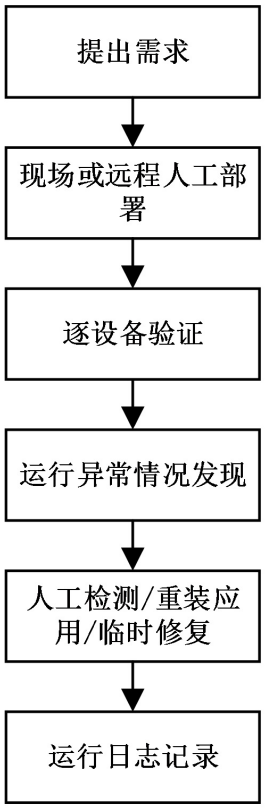

Figure 3. Operation and maintenance process of the traditional method
图 3. 传统方式运维流程图

如图 3 所示，传统方式存在三个主要问题。第一，软件对象分散管理，应用程序、设备插件、配置文件和规则逻辑缺少统一描述方式，容易造成版本记录不一致。第二，部署过程依赖人工判断，部署范围、执行时机和目标设备缺少策略化约束，容易引发误下发或版本不匹配。第三，异常处置依赖现场经验，故障恢复过程与软件版本、配置变更之间缺少关联记录，难以快速回退。

4.2. 方法特性分析

相较于传统人工部署方式，所提方法首先改变了软件对象的组织方式。传统模式下，应用程序、设备插件、业务规则和配置文件往往分别维护，版本记录和部署路径相对分散。本文将上述对象统一封装为版本化制品，并在同一流程下完成审核、签名、分发、部署和回退，使不同类型软件对象具备一致的管理入口，减少了多对象分散维护带来的流程割裂。

其次，所提方法提高了部署过程的可控性。部署任务将制品包、目标范围和执行条件关联起来，使制品下发具有明确的执行边界，可降低误下发、版本不匹配和异常扩散风险。同时，边缘侧在制品下发前对运行环境、设备状态和依赖条件进行检查，可减少端侧部署失败和重复联调。

最后，通过采集运行日志、告警信息和执行状态，将异常结果与制品版本、部署任务和回滚策略对应起来，使重新下发、版本切换和回滚等操作能够作为部署流程的一部分执行。由此，软件交付过程由单次下发转向“部署–监测–处置”的连续过程。

Figure 4. Workbench interface of Jihong AppStore
图 4. 冀鸿 AppStore 工作台界面

为进一步说明所提方法的平台化实现效果，冀鸿 AppStore 工作台界面如图 4 所示。该界面围绕软件制品治理与部署运维过程，对制品总数、已上架制品、部署中任务、运行正常实例和告警数量等关键状态进行集中展示，并通过制品类型分布、部署任务状态、最近告警和最近部署任务等模块，对制品分类、部署进度、运行异常和任务日志进行统一管理。由此可见，冀鸿 AppStore 能够将制品管理、部署执行和运行监测纳入同一平台界面，为软件制品的统一治理、过程跟踪和异常处置提供支撑。

4.3. 适用边界分析

本文方法更适用于设备类型较多、软件版本分散、部署范围需要控制且具备云–边–端协同基础的高速公路机电场景。例如，在多隧道、多控制器、多厂商设备共存的运行环境中，设备插件、应用程序和业务规则需要频繁更新或复用时，统一制品模型和策略化部署能够发挥较明显作用。

对于设备数量较少、软件更新频率较低、现场网络条件不足或缺少边缘节点支撑的场景，所提方法的应用收益可能相对有限。此外，本文方法依赖制品元数据、设备物模型和部署策略的规范化配置，若前期制品描述不完整或设备适配关系不明确，仍可能影响部署效果。

5. 案例验证与结果分析

5.1. 验证场景与指标设置

为验证所提软件制品协同部署方法的可行性，本文选取梅岭隧道和老营盘隧道风速风向仪替换接入场景作为验证对象。该场景涉及同功能、不同品牌及协议的机电设备替换，能够反映异构设备接入、设备插件适配和应用制品部署过程中的典型问题。

本文设置传统方式和冀鸿 AppStore 方式进行对比。传统方式主要依赖现场人员、原厂家或专业开发人员完成协议适配、应用安装和异常恢复；冀鸿 AppStore 方式则将设备插件、管理应用及相关配置封装为版本化制品，并通过平台完成审核、授权、分发、部署和版本处置。

本文选取以下三个指标进行评价：

(1) 设备接入周期：从风速风向仪硬件更换并具备接入条件，到管理应用能够稳定读取风速、风向数据所需时间。

(2) 应用部署时间：从执行应用部署指令，到端侧应用完成部署并正常运行所需时间。

(3) 故障响应时间：从运行异常触发处置，到通过重新下发、版本切换或回滚等方式恢复正常所需时间。

验证数据来源于工程实施记录、平台操作记录和现场故障处置记录。由于验证对象集中于风速风向仪这一单类设备，本文采用描述性对比方法分析两种方式下的指标变化。

5.2. 验证过程与结果

在梅岭隧道和老营盘隧道中完成风速风向仪替换后，分别采用传统方式和冀鸿 AppStore 方式进行设备接入、应用部署和异常恢复验证。

在具体实施过程中，以梅岭隧道风速风向仪设备插件为例，其制品元数据可实例化为：制品标识 I 为风速风向仪接入插件，版本 V 为 v1.2，依赖关系 D 为 KaihongOS 运行环境，适配环境 E 为 ARM 架构边缘节点，权限约束 P 为串口数据读写，目标对象 T 为 RS485 接入的风速风向仪设备，部署范围 S 为梅岭隧道指定设备组，回滚关系 R 指向上一稳定版本 v1.1。部署时，平台进一步构建部署策略 $S_d = <G, W, A, C, B>$，其中 G 为梅岭隧道环控网关组，W 为夜间低峰时段，A 为现场运维班组，C 为设备在线、网络状态正常且版本兼容性满足要求，B 为异常情况下切换至上一稳定版本。

在模拟“插件版本不兼容”的预测试中，端侧加载 v1.2 插件后出现协议解析异常，并将运行告警、异常日志和执行状态回传至边缘侧与云端。平台接收反馈后，判定其不满足部署策略中的版本兼容性约束，随即暂停该版本的后续分发任务，并依据回滚关系 R 将故障网关中的插件切换至 v1.1 稳定版本，从而实现由运行反馈触发的快速回滚处置。

表 4 所列数据来源于两处隧道实施过程中的平台操作记录、现场接入记录和故障处置记录，用于反映所选场景下两种方式的典型时间差异。结果如表 4 所示。

Table 4. Comparison of key indicators under different deployment methods
表 4. 不同部署方式关键指标对比

关键指标	传统方式	冀鸿 AppStore 方式
设备接入周期/天	9	1
应用部署时间/s	9	5
故障响应时间/s	100	10

由表 4 可知，采用冀鸿 AppStore 方式后，设备接入周期由 9 天缩短至 1 天，应用部署时间由 9 s 缩短至 5 s，故障响应时间由 100 s 缩短至 10 s，分别降低约 89%、44%和 90%。结果表明，所提方法在异构设备替换接入和平台化部署场景中能够缩短接入、部署和恢复时间。

5.3. 结果分析

从设备接入周期看，传统方式在更换不同品牌或协议的风速风向仪后，需要重新开展协议解析、配置修改和现场联调；冀鸿 AppStore 方式通过设备插件和物模型映射将部分适配工作前置，使现场过程转变为插件选择、配置绑定和运行确认，因此接入周期明显缩短。

从应用部署时间看，设备管理应用的打包、签名、分发和部署被纳入统一流程，减少了人工安装和脚本执行环节。由于本案例中应用规模较小，部署时间本身较短，因此该指标降幅相对有限，但平台化方式提升了部署过程的标准化和可追溯性。

从故障响应时间看，传统方式主要依赖人工判断和手工恢复，而所提方法将运行状态、版本记录和处置策略关联起来，使异常可通过重新下发、版本切换或回滚等方式处理，因此缩短了故障恢复过程。

需要说明的是，表 4 结果来源于风速风向仪替换接入场景，属于单类设备、两个隧道场景下的描述性验证结果。

6. 结论

本文围绕高速公路机电系统中异构设备接入复杂、软件对象分散管理、部署区域与安全边界约束难以统一表达，以及异常恢复策略与版本关系脱节等问题，将应用程序、AI 应用、原子化服务、业务规则和设备插件等对象抽象为统一的软件制品，并进一步给出了制品元模型、部署策略模型和云－边－端协同执行流程。通过这种处理，传统分散的软件安装、配置和恢复过程被转化为以制品版本和部署策略为核心的统一管理过程，使不同类型软件对象能够在同一框架下完成描述、审核、分发、部署、反馈和回退。

案例验证表明，在梅岭隧道和老营盘隧道风速风向仪替换接入场景中，所提方法使设备接入周期由 9 天缩短至 1 天，应用部署时间由 9 s 缩短至 5 s，故障响应时间由 100 s 缩短至 10 s。结果说明，制品化封装、策略化部署和反馈式处置能够减少重复适配和人工联调，并通过设备插件元数据实例化、部署策略约束和运行反馈回滚机制，将设备类型、接入协议、部署范围、授权条件和回滚关系等因素纳入统一管理，从而提高异构设备接入、部署过程控制和异常恢复能力。

与现有侧重设备状态监测、资源管理或养护决策的研究相比，本文的工作重点在于把高速公路机电系统中的多类型软件对象纳入统一的软件生命周期管理过程，补充了异构设备环境下软件制品定义、部署区域与安全边界约束表达、策略化部署和闭环处置方面的研究内容。验证结果说明，所提方法在单类异构设备替换接入和平台化部署场景中具有可行性，可为高速公路机电系统软件对象的统一治理提供方法参考。

后续研究将面向更多设备类型、多批次部署和复杂网络条件，进一步完善边缘侧预测试、分批部署、灰度发布和回滚判据等机制，检验该方法在规模化场景中的适用性。

致　谢

感谢河北高速公路集团有限公司承德分公司在研究过程中提供的工程场景、业务资料和实践条件支持。感谢哈尔滨工业大学(威海)汽车工程学院在论文撰写、方法设计和技术分析过程中给予的指导与帮助。感谢参与梅岭隧道和老营盘隧道相关设备接入、平台部署和现场验证工作的技术人员，为本文案例验证和数据整理提供了重要支持。感谢各位专家和同行对本文提出的宝贵意见，使本文在研究思路、结构安排和表达规范方面得到进一步完善。

基金项目

本文受 2025 年河北省省级重大科技支撑计划项目《全量鸿蒙高速公路智慧化应用场景》资助，项目编号：252Q0801D。

参考文献

[1] 中共中央 国务院印发《交通强国建设纲要》[EB/OL]. https://www.gov.cn/zhengce/2019-09/19/content_5431432.htm, 2026-03-22.

[2] 交通运输部关于印发《数字交通发展规划纲要》的通知: 交规划发〔2019〕89 号[EB/OL]. https://xxgk.mot.gov.cn/2020/jigou/zhghs/202006/t20200630_3321233.html, 2026-03-30.

[3] 伍朝辉, 徐建达, 符志强, 等. 交通强国建设视域下公路交通数字孪生体系架构、关键技术与实践案例[J]. 交通运输研究, 2023, 9(4): 104-124.

[4] 伍朝辉, 常莹, 李青, 等. 基于三维视频融合的隧道运营管理创新应用研究[J]. 隧道建设(中英文), 2022, 42(1): 154-161.

[5] 彭峰, 周凌峰. 基于知识图谱的隧道机电智能运维平台浅析[J]. 中国交通信息化, 2023(10): 138-140.

[6] 郝进. Kubernetes 在高速公路机电设备软件系统中的应用[D]: [硕士学位论文]. 西安: 长安大学, 2022.

[7] 张毅, 陈莎雯, 陈艳, 等. 高速公路机电设备动态运维评价及预测性维护[J]. 吉林大学学报: 工学版, 2026, 56(1): 170-182.

[8] 许珑璋, 丘天鹏, 林莉. 数字孪生技术在广西山区交通基础设施智慧运维中的应用[J]. 西部交通科技, 2025(10): 191-194.

[9] Bucaioni, A., Cicchetti, A. and Ciccozzi, F. (2022) Modelling in Low-Code Development: A Multi-Vocal Systematic Review. *Software and Systems Modeling*, **21**, 1959-1981. https://doi.org/10.1007/s10270-021-00964-0

[10] Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016) Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, **3**, 637-646. https://doi.org/10.1109/jiot.2016.2579198

[11] Satyanarayanan, M. (2017) The Emergence of Edge Computing. *Computer*, **50**, 30-39. https://doi.org/10.1109/mc.2017.9

[12] Kim, J., Park, Y., Kim, C. and Lee, H. (2014) Mobile Application Service Networks: Apple's App Store. *Service Business*, **8**, 1-27. https://doi.org/10.1007/s11628-013-0184-z

[13] El Jaouhari, S. and Bouvet, E. (2022) Secure Firmware over-the-Air Updates for IoT: Survey, Challenges, and Discussions. *Internet of Things*, **18**, Article 100508. https://doi.org/10.1016/j.iot.2022.100508

[14] Kong, L., Tan, J., Huang, J., Chen, G., Wang, S., Jin, X., *et al*. (2023) Edge-Computing-Driven Internet of Things: A Survey. *ACM Computing Surveys*, **55**, 1-41. https://doi.org/10.1145/3555308

[15] Chiang, Y., Zhang, Y., Luo, H., Chen, T., Chen, G., Chen, H., *et al*. (2023) Management and Orchestration of Edge Computing for IoT: A Comprehensive Survey. *IEEE Internet of Things Journal*, **10**, 14307-14331. https://doi.org/10.1109/jiot.2023.3245611

[16] Vaño, R., Lacalle, I., Sowiński, P., S-Julián, R. and Palau, C.E. (2023) Cloud-Native Workload Orchestration at the Edge: A Deployment Review and Future Directions. *Sensors*, **23**, Article 2215. https://doi.org/10.3390/s23042215

[17] Roda-Sanchez, L., Garrido-Hidalgo, C., Royo, F., Maté-Gómez, J.L., Olivares, T. and Fernández-Caballero, A. (2023) Cloud-Edge Microservices Architecture and Service Orchestration: An Integral Solution for a Real-World Deployment Experience. *Internet of Things*, **22**, Article 100777. https://doi.org/10.1016/j.iot.2023.100777

[18] Rokis, K. and Kirikova, M. (2023) Exploring Low-Code Development: A Comprehensive Literature Review. *Complex Systems Informatics and Modeling Quarterly*, **36**, 68-86. https://doi.org/10.7250/csimq.2023-36.04

[19] Kumara, I., Garriga, M., Romeu, A.U., Di Nucci, D., Palomba, F., Tamburri, D.A., *et al*. (2021) The Do's and Don'ts of Infrastructure Code: A Systematic Gray Literature Review. *Information and Software Technology*, **137**, Article 106593. https://doi.org/10.1016/j.infsof.2021.106593

[20] Soares, E., Sizilio, G., Santos, J., da Costa, D.A. and Kulesza, U. (2022) The Effects of Continuous Integration on Software Development: A Systematic Literature Review. *Empirical Software Engineering*, **27**, Article No. 78. https://doi.org/10.1007/s10664-021-10114-1

[21] Akbar, M.A., Smolander, K., Mahmood, S. and Alsanad, A. (2022) Toward Successful DevSecOps in Software Development Organizations: A Decision-Making Framework. *Information and Software Technology*, **147**, Article 106894. https://doi.org/10.1016/j.infsof.2022.106894

[22] Lagally, M., Matsukura, R., McCool, M., *et al*. (2023) Web of Things (WoT) Architecture 1.1. W3C Recommendation. https://www.w3.org/TR/wot-architecture11/

[23] Stouffer, K., Pease, M., Tang, C.Y., *et al*. (2023) Guide to Operational Technology (OT) Security. NIST Special Publication 800-82 Revision 3, National Institute of Standards and Technology.