软件定义汽车

服务化开发白皮书

中国汽车工业协会　软件分会
软件定义汽车（SDV）专业委员会

2025年12月

软件定义汽车
服务化开发白皮书

中国汽车工业协会　软件分会
软件定义汽车（SDV）专业委员会

2025 年 12 月

编 委 会

主编单位

中国汽车工业协会
深圳引望智能技术有限公司
上海智能汽车软件园
中国长安汽车集团有限公司
赛力斯凤凰智创科技有限公司
吉利汽车研究院（宁波）有限公司
安徽江淮汽车集团股份有限公司
舍弗勒投资 (中国) 有限公司
北京海纳川汽车部件股份有限公司
东软睿驰汽车技术（上海）有限公司
麦格纳汽车系统苏州有限公司
电装智能科技（上海）有限公司
中国第一汽车集团有限公司
东风汽车集团有限公司
广州汽车集团股份有限公司
北京小米移动软件有限公司
北京汽车研究总院有限公司
科世达（上海）管理有限公司
芜湖埃泰克汽车电子股份有限公司
国汽（北京）智能网联汽车研究院有限公司
汉拿万都（北京）汽车部件研究开发中心有限公司
奇瑞汽车股份有限公司
欧摩威集团
中科创达软件股份有限公司
江铃汽车股份有限公司
比亚迪汽车工业有限公司
梅赛德斯—奔驰数字技术有限公司
镁佳（北京）科技有限公司
一汽解放汽车有限公司

目　录
CONTENTS

序　言
PREFACE

汽车产业正经历一场由“新四化”驱动的世纪变革，软件定义汽车（SDV）已成为共识，推动产品属性从单一的交通工具向集出行、办公、娱乐于一体的“智能移动空间”跃迁。这一根本性转变，正剧烈重塑着以整车厂为绝对核心的线性产业链，催生出融合了传统车企、零部件巨头、科技公司及软件服务商的网状产业新生态。

在此背景下，构建面向服务的架构（SOA）以实现深度的“软硬解耦”，已成为 SDV 从理念落地的核心实现路径。成功的服务化转型，能解锁用户体验的持续迭代、有效提升开发效率、开拓全新商业模式，是赢得未来竞争的关键。

然而，理想与现实之间存在巨大鸿沟。行业在向服务化迈进时，普遍面临电子电气架构转型、工具链协同、信息安全与功能安全体系的构建、开发流程与组织架构重塑等多维度的严峻挑战。这些系统性难题，非单一企业所能独立解决，亟需产业链各方凝聚共识，共同探索破局之路。

本白皮书正是在此背景下应运而生。我们旨在系统梳理软件定义汽车服务化的内涵、价值与挑战，并重点探讨其可行的落地路径与方法论，希望能为行业构建互联互通的“共同语言”，推动形成标准引领、生态协同的创新格局，助力中国智能网联汽车产业在新一轮竞争中把握机遇、做强做优。

前　言
FOREWORD

本白皮书由中国汽车工业协会软件分会提出，软件定义汽车（SDV）委员会联合各成员单位及行业伙伴共同编制而成。

编制工作基于对当前汽车产业现状的深入研究，旨在系统回应软件定义汽车发展中的共性挑战。白皮书详细阐述了软件定义汽车服务化开发的定义、核心价值与应用场景，针对性地分析了服务化开发在流程、操作系统、工具链、安全等方面面临的挑战，并提出了具体的落地方案与发展建议。

我们期待本白皮书的发布能够汇聚行业智慧，与业界同仁共同探讨并解决智能网联汽车发展过程中的关键问题，在技术革新、生态构建与应用创新等方面形成合力，为推动中国汽车产业的高质量发展贡献一份力量。

1 什么是软件定义汽车的服务化

1.1 软件定义汽车的概念

软件定义汽车（Software Defined Vehicles，SDV）是当前汽车产业在智能化与网联化浪潮推动下形成的新型发展范式。其核心内涵在于，汽车的功能实现与持续演进不再主要依赖硬件体系，而是转向以软件为核心驱动，通过软件深度参与并主导车辆的定义、设计、开发、验证、销售及服务等全生命周期环节，从而持续优化用户体验、提升车辆价值。软件定义汽车不仅改变了汽车的开发流程与技术架构，也重构了汽车产业的价值创造逻辑和生态结构。软件定义汽车的主要特征可归纳为以下三个方面：

1. 软硬解耦：传统汽车开发中软硬件高度耦合，功能迭代受限于硬件更新周期。而在软件定义汽车中，软硬件实现分层解耦，软件功能可独立于硬件进行开发、部署与升级，大幅提升了系统的灵活性和可扩展性。

2. 架构重塑：在硬件层面，分布式电子电气架构逐步被集中式架构取代，实现算力资源的统一调度与高效利用；在软件层面，采用面向服务的架构（Service-Oriented Architecture，SOA），支持功能模块的灵活调用与组合，为个性化体验和服务创新提供技术基础。

3. 持续进化：车辆在全生命周期内可通过软件升级不断拓展功能、优化性能，实现“车辆进化”，从而延长产品价值周期，推动汽车从“一次性交付品”向“可持续服务平台”转型。

软件定义汽车的颠覆性价值主要体现在以下维度：

1. 重构研发体系：打破“硬件先行”的传统开发模式，构建“软件主导、软硬协同”的新型研发流程，显著提升开发效率，功能迭代周期预计缩短 60% 以上。

2. 优化技术架构：算力资源集中化率预计提升至 80%，软件架构的灵活性和复用性显著增强，支撑差异化功能与服务的快速部署，全面提升系统响应能力和用户体验。

3. 重塑产业生态：通过软硬解耦与分层架构，推动产业链分工重组，形成以软件能力为核心、多方协同的服务化开发生态，提升整车软硬件协同开发效率，加速跨界融合与创新合作。

1.2 软件定义汽车的服务化

SOA 架构的总体思路，是通过对车辆底层能力进行梳理，设计出合理的服务模型，抽取共性基础能力，并通过“分层解耦”的方式定义恰当的 API（应用程序编程接口）。其核心特征在于，这些 API 接口独立于底层的硬件平台、操作系统与编程语言，从而确保构建在不同系统中的服务能够以一种统一和通用的方式进行交互，最终由上层应用通过调用这些服务接口来灵活地组合实现复杂的业务逻辑。这一架构范式的落地，必然要求软件开发模式的根本性变革，即从传统模式转向服务化开发。

在软件定义汽车的实现路径中，面向服务的架构作为核心的技术基石，正发挥着无可替代的作用。其基本理念在于，对整车智能化底层能力进行重构与组织，将传统的、以单一功能 ECU（电子控制单元）为核心的分布式连接方式，转变为以“服务”为中心的软件架构。通过将车辆的硬件能力与各项应用功能封装成标准化的可调用“服务”，并基于统一接口与通信协议进行交互，SOA 使得各个服务组件之间能够灵活、高效地相互访问与组合，最终为实现硬件资源的“即插即用”与软件功能的快速迭代奠定坚实基础。

面对汽车软件复杂性的指数级增长与市场对个性化、智能化体验的迫切需求，采用 SOA 服务化架构已成为产业发展的必然选择。

SOA 是电子电气架构变革的必然要求。 传统的分布式 ECU 架构在算力融合、数据传输与跨功能协同方面已显现出瓶颈。在电子电气架构向域控制乃至中央集中式演进的过程中，SOA 通过其松耦合、标准化的特性，成为实现跨域功能协同与计算资源高效利用的核心技术路径。

SOA 是软件定义汽车实现的技术前提。 软件定义汽车要求软件能够深度参与并主

导汽车的全生命周期。SOA 通过“软硬分离”与“服务抽象”，将车辆能力转化为可编程的接口，为软件功能的持续迭代、升级与商业模式创新提供了稳定的基础框架，从而推动汽车从静态的交通工具向可进化、可扩展的智能移动终端转变。

SOA 是构建开放产业生态的关键载体。 作为一种开放、标准化的架构，SOA 能够降低开发门槛，吸引更广泛的开发者与科技公司参与其中，共同构建一个充满活力的汽车软件应用与服务生态，催生新的价值增长点。

传统汽车架构软硬件高度嵌套，当需要在现有架构中增加一项新功能时，需改变与其相关的 ECU 软件，这增加了系统升级和提升车辆功能的复杂度。对于汽车制造商而言，集成验证新增功能也是一项挑战。特别是在实现较复杂功能时，需要同时开发多个控制器才能完成验证。因此，在传统架构下，硬件之间难以形成较强协同性，汽车软件的可复用性差，OTA 升级能力较弱，难以满足未来智能汽车的发展要求。

SOA 架构设计理念是将车端的不同功能以及相应硬件的能力划分为服务，对不同服务组件的接口进行标准化封装。通过既定的协议，服务可以相互访问和扩展组合。整车制造商可以将智能汽车原生能力封装为标准化的服务接口，并以服务的形式供应用层调用，实现资源的充分共享。应用层可以在不同的车型上实现复用，同时基于标准化服务接口，可以实现新功能的快速迭代和更新。因此，SOA 架构设计，结合未来新型集中式电子电气架构，为实现软件定义汽车奠定了坚实基础。

服务化开发是一种基于整车系统的全新的开发方式。它彻底改变了过去将汽车视为多个独立 ECU 简单相加的思维，不再局限于基于单个 ECU 进行软件设计与实现。取而代之的是，将整车所有 ECU 及其承载的软件视为一个统一的“资源池”，并在此资源池之上，以“服务”为基本单元对软件资产进行重新组织与管理。这种开发模式极大地提升了软件组件的复用性与功能的可配置性，从而能够支撑功能的快速迭代与敏捷交付，持续为用户创造更丰富、更智能的驾乘体验。

2 软件定义汽车服务化的价值

2.1 引领用户体验变革

2.1.1 用户体验的持续优化

软件定义汽车深刻重塑了用户与汽车的传统互动关系，推动用户角色完成从“被动的功能消费者”到“主动的体验参与者“的转型。服务化设计让汽车不再是普通的交通工具，而是能够持续进化的智能终端。

SOA 架构通过预埋标准化 API 接口，在实车硬件不做变更的情况下，可依托 OTA 升级完成软件迭代与新功能激活。这不仅让用户购车后持续获得功能新增与优化的体验，更构建起“购车 – 用车 – 升级”的全生命周期价值闭环，突破传统架构开发周期长、体验滞后的局限，最大化延伸产品价值。例如，智能交互灯语在刚量产的车上只有 10 个灯语效果，但是 API 接口软件已预埋了 200 个灯效效果，后期随着灯具灯语软件更新即可实现 200 个灯效功能。

此外，这种低耦合的软件架构能支持各项服务独立升级和维护，避免单一功能的调整或故障对整体系统运行产生连锁影响，有效保障用户的核心驾乘体验与用车安全，进一步强化用户对智能汽车的信任度与价值体验。

2.1.2 个性化定制能力的提升

软件定义汽车打破了传统汽车功能固化的局限，显著提升了车辆个性化定制能力。SOA 作为核心技术支撑，通过对车辆动力控制、座舱交互、智能互联等核心功能进行

服务化拆解，形成标准化、模块化的独立服务单元，并封装为统一规范的 API 接口，为个性化定制体系构建了坚实的技术底座。

从用户交互层面来看，APP 应用端基于开放的 API 接口，根据用户个性化需求灵活组合不同服务模块，实现功能的定制化搭配与调用。同时，软件系统可通过 API 接口的多样化组合，主动拓展个性化服务场景，例如结合灯光控制、音响调节、座椅角度调节等服务模块，打造专属灯光秀模式；融合空调温度调节、座椅按摩、氛围灯营造等功能，推出贴合用户需求的新场景，让车辆从“交通工具”升级为“个性化移动空间”。

相较于传统架构，SOA 服务化架构的个性化定制这一模式不仅大幅降低了个性化功能的开发门槛与落地成本，更实现了“一车一策”的精细化定制体验，也为用户提供兼具专属感、灵活性与场景化的高品质用车服务，推动车辆从固定功能载体向个性化移动空间升级。

2.1.3 生态互联与智能运维

软件定义汽车打通了车端与外部场景的生态壁垒，通过技术整合实现车内外智能设备与服务的无缝协同，将智能家居、移动支付、智慧出行等多元场景融入出行链路，为用户构建丰富、便捷的数字生活体验，重塑出行场景的功能边界与价值局限，推动出行从“单一交通行为”向“全场景数字服务节点”升级。

SOA 作为生态整合的核心技术支撑，通过标准化服务封装与统一 API 接口设计，将智能家居控制、移动支付结算等外部服务转化为可灵活调用的独立 APP 服务模块。这些服务模块通过车端开放的 API 接口生态实现深度集成，用户可在车机系统或关联移动端 APP 中，一键触发跨场景控制指令：例如通过车机提前联动智能家居设备，实现到家前灯光自动开启、空调预设温度；借助车载移动支付服务，完成加油、停车、充电等场景的无感结算，真正实现“车即服务入口”的生态整合目标。

这种基于 SOA 架构的生态整合模式，不仅打破了传统汽车作为孤立交通工具的功能局限，更以用户需求为核心，构建起跨终端、跨场景的智能互联生态，为用户带来前所未有的便捷性与体验多样性，推动汽车从“单一出行工具”升级为“全域智能生活枢纽”，成为贯通日常数字生活的核心载体。

2.2 重构企业价值体系

2.2.1 研发效率提升

服务化设计通过构建软硬件解耦的标准化架构，实现了汽车软件开发方式的根本性变革。传统高耦合的嵌入式软件被解耦成多个“基础服务”，这些服务如同标准化零部件，实现了软硬件的有效解耦，使功能开发不再高度依赖硬件迭代，开发者通过标准化的 API 接口，即可高效、灵活地构建新功能，实现“一套软件平台、覆盖多车型”的规模化协同效应，显著提升了软件复用程度与整体研发效率。

服务化架构遵循“高内聚、低耦合”的核心设计原则，建立了系统级的质量管控机制。每个服务的独立性与标准化接口设计，有效隔离了功能变更的影响范围，大幅降低了系统复杂度。经过充分验证的标准化服务组件可在多车型平台中实现规模化复用，既确保了功能实现的一致性，又通过持续迭代优化不断提升整体软件的可靠性与稳定性。

2.2.2 资产战略转型

服务化架构从根本上重塑了汽车软件的本质属性，使其从深度依附于特定硬件的“定制化成本中心”，成功转型为可沉淀、可复用并持续增值的“企业核心软件资产”。通过服务化的设计与开发，将核心能力原子化、标准化为一个个独立服务，使开发活动不再仅仅是满足单一车型需求的孤立项目，而可以持续沉淀至企业级的可复用服务库中，进而催生了可跨车型、跨平台的软件复用能力，奠定了软件资产化的基石。

至此，汽车软件开发得以从高成本、低效率的“重复劳动”中解放出来，升华为一场高效、灵活且能够持续积累的“战略资产构建”过程。服务化架构由此驱动企业实现从短期项目成本到长期战略资产的根本性转变，每一个经过标准化设计与验证的服务，都将作为核心数字资产注入企业平台，在后续产品开发中通过规模化复用，形成强大的累积效应与增值飞轮，构筑起企业的软件定义核心竞争力。

2.2.3 产业协作重塑

SOA 架构的核心技术特征在于实现了软件功能与硬件平台的彻底解耦。这一技术

突破使企业能够建立科学的硬件选型机制，基于性能指标、成本结构和供应链稳定性等客观参数，选择最优硬件方案。此种模式不仅有效规避了传统模式下对单一供应商的技术依赖，更显著增强了企业在供应链波动中的风险抵御能力。

在产业协作层面，标准化服务接口的建立为生态系统参与者提供了统一的能力接入标准。这种架构设计显著降低了第三方开发者的技术门槛，使产业链上下游企业能够基于统一的数字接口开展协同创新。这种变革促使产业组织形式由传统的线性链式结构，向网络化、平台化的开放生态体系演进。

通过构建开放的技术架构和标准化的协作接口，企业能够加速整合内外部创新资源，形成规模化的创新合力。这种生态化协作模式不仅提升了资源利用效率，更通过协同互动的方式推动整个产业实现从封闭式创新向开放式创新的范式转移。

2.2.4 商业模式拓展

服务化架构为汽车产业的数字化转型开辟了全新的价值创造路径。该架构通过将核心功能模块化为可独立部署、动态升级的标准化服务，使企业得以突破传统“制造－销售”的一次性交易模式，构建起“硬件为载体、软件为内核、服务为体验”的一体化商业模式。

在此模式下，车辆作为可迭代的价值载体，通过软件与服务的持续更新，不仅提升了用户粘性与品牌忠诚度，更开辟了由“制造利润”迈向“全生命周期价值”的全新增长路径。

2.3 多维驱动行业变革

2.3.1 技术架构的革新

软件定义汽车服务化所驱动的技术架构革新，其核心在于通过构建“服务导向”的软件层，实现了对传统汽车电子电气架构的彻底抽象与虚拟化。这一软件层如同在异构的硬件芯片与统一的业务功能之间，插入了一个强大的“操作系统”，它将所有底层硬件能力都封装成标准、统一的服务接口。正是这种架构层面的抽象，使得硬件资源得以被“池化”并动态调度，为上层应用的开发解除了对特定硬件的依赖。因此，这场革新本质上是为整个行业建立了一套全新的、以服务为中心的“数字基座”，它

不仅决定了软件如何被定义、组合与迭代，更从根本上奠定了硬件标准化、生态开放化与供应链重构的技术前提，宣告了汽车研发从硬件集成主导迈向软件架构主导的新范式。

2.3.2 技术生态的融合

在汽车产业向“软件定义汽车”加速演进的进程中，面向服务的架构以其高度的开放性与灵活性，成为了打破传统 ECU 封闭部署模式、实现技术生态深度融合的核心引擎。该架构通过定义标准化接口并依托服务总线的智能调度能力，成功将车辆转变为一座集成了车企自研核心服务与丰富第三方生态服务（如导航、支付、智能驾驶算法等）的“移动平台”。这一转变打破了汽车与互联网、人工智能及物联网等产业间的技术壁垒，促进了跨领域技术要素的自由流动与高效重组。形成了一个以“汽车 + 多产业”为特征的开放式创新生态，它为产品功能的快速迭代、商业模式的跨界重构提供了坚实的基础，最终强力驱动着汽车从功能单一的交通工具，向一个持续进化、可自定义的智能移动空间完成本质性转型。

2.3.3 接口标准化的推进

随着服务化理念被行业核心企业广泛采纳，接口标准化已从技术共识转化为产业实践。这一进程通过建立统一的交互协议，显著降低了产业链各环节的对接成本，使芯片、硬件、操作系统到应用服务等各层级能够高效协同，在减少重复开发投入的同时形成了规模效应。标准化更深层的价值在于推动研发范式的转变：当底层接口实现统一后，研发资源得以从繁琐的适配工作中解放，转而聚焦于核心应用创新与差异化功能开发，极大提升了行业响应市场需求与技术迭代的敏捷性。更重要的是，标准化架构为“软件定义汽车”提供了坚实基础，使各类软件组件能够如乐高积木般自由组合、持续演进，最终推动产业从一次性硬件销售向持续增值的服务模式与开放生态转型，构筑起未来汽车产业的新格局。

2.3.4 硬件资源化与供应链重构

SOA 的深入应用正在系统性地重塑汽车硬件价值定位与供应链格局。该架构通过“硬件资源化”理念，推动传统分散的 ECU 向域控制器和中央计算平台演进，使硬件从专用功能载体转变为可被上层软件通过标准化接口动态调用的共享资源池，这一转

变不仅显著提升了硬件利用率和通用性，降低了整车 BOM 成本与迭代复杂度，更重要的是通过软硬件解耦重构了产业分工体系：硬件不再与特定功能或供应商绑定，而是成为支持服务生态的标准化基础平台，这使得车企能够在全球范围内基于性能、成本等维度自由选择硬件供应商，打破了传统“软硬件捆绑”的供应模式。与此同时，这种变革催生了全新的产业分工格局——硬件供应商专注于提升基础硬件的性能与可靠性，软件供应商则致力于创新功能开发，推动供应链从垂直整合转向水平分工，最终构建起更加灵活、高效的专业化协同网络。

3 服务化开发的挑战

随着软件定义汽车的深入发展，服务化开发方式正逐步成为行业共识。然而，当前 SDV 服务化开发仍处于探索与实践深化阶段，其价值有待于通过体系化建设进一步释放。

3.1 适配现有开发模式与流程的挑战

服务化开发所倡导的敏捷、迭代与高度协作的理念，对现有成熟开发模式的运作流程提出了新的适配要求。面向开发模式和流程的升级，其核心差异与挑战体现在以下两个方面：

1. 串行递进流程与快速迭代节奏的差异

现有模式依托于分阶段、层层递进的串行流程，确保了开发过程的可控性与可追溯性。而服务化开发以后台能力服务化为核心，强调通过接口解耦实现功能的并行开发与独立部署，其本质要求一种支持快速迭代和持续验证的方式。两者在项目推进的“时间节拍”上存在差异，如何在保证高质量的前提下，提升迭代能力，是重要的适配挑战。

2. 硬件依赖的验证环境与软件先行的开发需求的差异

当前流程中，软件的集成与系统验证高度依赖后期可用的目标硬件环境。然而，

服务化开发要求软件团队能够提前并独立于硬件进行服务的开发与集成测试。这种“软件先行”的理念与现有“软硬件同步”的验证环境之间产生了缺口，如何构建并管理一套支持服务早期开发与测试的虚拟化环境，是实现平滑过渡的技术挑战。

3.2 工具链碎片化与兼容性挑战

在 SOA 的设计范式下，企业通过构建标准化、松耦合且可复用的服务组件，旨在打造灵活、健壮并能敏捷响应业务变化的 IT 架构。然而，当这种服务化开发由多个团队并行推进时，由于各自采用的开发工具链不统一，为这一理念的落地实践带来了新的挑战。工具链的异构性与不成熟性正成为制约服务化发展的关键瓶颈，主要体现在以下两方面：

1. 工具链兼容性挑战

各主机厂或供应商所选用的工具链来源各异，因其底层架构、设计目标与验证标准不一，存在兼容性问题。直接导致了开发过程中格式不支持、数据类型未对齐、通信属性冗余配置等技术障碍，不仅增加了集成的复杂度，也严重阻塞了服务化开发的高效流程。

2. 工具链成熟度挑战

为适应服务化趋势而新兴的工具，其自身仍处于快速迭代阶段。在功能完备性、场景覆盖度及稳定性方面往往存在考量不周、验证不足的情况，引入了一定的风险与不确定性，对服务化开发的效率与质量构成了新的制约。

综上所述，行业内在工具链层面存在的“碎片化”与“成熟度不足”的现状，已成为制约 SOA 规模化应用和效能释放的重要因素。需要行业各方凝聚共识，共同推动工具链的标准化与规范化建设，以更加规范统一的工具生态支撑服务化架构的持续演进与成功落地。

3.3 服务化开发对操作系统中间件提出的多维度挑战

随着汽车电子架构向中央计算和软件定义方向发展，软件系统正在从静态分布转

向以服务为核心的动态协作架构。这种模式极大提升了系统的灵活性和可扩展性，但也对操作系统提出了更高的要求。不同域的操作系统——从控制域的实时系统，到娱乐域的开放系统——都在面临新的挑战与演进压力。

1. 从静态到动态的运行机制

在传统车载系统中，操作系统运行的任务、通信方式和内存布局在设计阶段就已确定，运行后几乎不再变化。而在 SOA 环境下，系统中的服务可以根据需要被动态加载、更新或卸载。例如，智驾路径规划服务要求在车辆启动后按需加载，但该服务依赖的模块较多，包含地图解析、预测等；在服务加载阶段，CPU 瞬时占用率会明显提升，可能达到 80%–90%，进而导致系统启动到可驾驶状态的延时增大等。操作系统必须支持在运行中安全地创建、管理和回收服务，并具备完善的隔离和容错机制，防止个别服务故障影响整车稳定性。这意味着操作系统需要从“静态调度”转变为“动态管理”，具备更强的自适应能力。

2. 通信机制的复杂化

SOA 架构强调服务间的解耦与交互，系统通信不再局限于点对点模式，而通过中间件（如 SOME/IP、DDS、MQTT 等）实现跨域消息传递。这种机制带来了灵活性，但也增加了通信延迟和系统复杂度。操作系统需要提供高效的进程间通信和消息调度机制，支持多核并行通信、消息优先级控制、报文聚合和流控等，以保证关键任务的数据交换可靠和及时。

3. 资源管理与多任务调度

随着中央计算平台的普及，车内的多个功能域往往共用同一硬件资源。SOA 模式下，不同服务会并行运行，CPU、内存和带宽的竞争更加频繁。某些场景下，将座舱服务与智驾服务分别放在同一个SOC的CPU和实时核上，两者共享总线、缓存等资源，在座舱开启导航 + 音乐时，产生大量内存访问，导致智驾侧控制周期波动。操作系统必须具备实时的资源监控与分配能力，支持动态负载平衡，确保关键功能在高负载条件下仍能获得必要的算力与带宽。同时，多核和异构计算架构的使用，也对操作系统的调度策略提出了更高要求。

4. 实时性保障的难度提升

SOA 的灵活特性带来了运行路径的不确定性，导致系统实时性保障更具挑战。传

统实时操作系统（如 AUTOSAR Classic OS、RTOS）以确定性调度为核心，而服务化架构则引入了动态通信和跨域依赖。操作系统需要在灵活性与确定性之间实现平衡，例如通过多级优先级调度、核间隔离和资源锁定等方式，确保关键任务（如控制环路、传感器融合）的稳定运行。

5. 娱乐域操作系统的特殊挑战

在座舱和娱乐域中，SOA 服务化带来的挑战更集中地体现在系统复杂性和用户体验上。这些领域普遍采用开放式操作系统，如 Android Automotive、HarmonyOS、Linux 等，它们在功能扩展性和生态兼容性方面具有优势，但缺乏传统车载系统所要求的强实时能力和确定性。随着座舱与驾驶信息融合的趋势加快，娱乐域操作系统也需要承担部分安全关键功能（如摄像头显示、驾驶辅助信息呈现）。这对系统的启动时间、性能隔离、任务优先级调度等方面都提出了更高要求。同时，服务的频繁调用还要求这些系统具备更强的多进程管理、内存回收与稳定性保障机制，以防止应用崩溃或资源泄漏影响整车运行。开放系统将通过虚拟化、容器化技术与实时内核增强方案，与底层安全域协同运行，以兼顾功能灵活与运行可靠。

总体来看，SOA 服务化让汽车软件具备了更强的扩展性与协作能力，但也让操作系统的角色从“固定任务执行者”转变为“动态运行管理者”。控制域操作系统需要在确定性和灵活性之间寻找平衡；中央计算平台上的系统需强化资源调度与隔离；而娱乐域操作系统则要在开放性与稳定性之间实现兼容。随着车载计算能力提升和软件架构的成熟，操作系统中间件将逐步具备动态调度、服务感知、资源自适应等能力，成为支撑汽车智能化与服务化演进的核心基础。

3.4 信息安全挑战

车辆实现 SOA 服务化后，面临着多维度的网络安全挑战，涵盖针对车辆本身的攻击、来自云端服务器的攻击，以及周边配套设施的攻击等多个层面。

1. 通讯协议 & API 漏洞利用

整车通过 SOA 服务化后，通常采用 DDS 或 SOME/IP 等服务化的通讯协议，攻击者可以通过暴露到车辆对外接口，蓝牙、Wi-Fi 和蜂窝网络等手段，调用车内的服务

化接口，造成非法的服务请求和重放攻击等；同时攻击者还可以针对不同层级（服务层、硬件抽象层或操作系统层）的 API 发起攻击，以操纵传感器、获取硬件控制权，植入后门或破坏系统运行。例如部分传感器以服务化形式提供原始数据，用于图像识别、车道识别等功能。然而，攻击者若逆向解析服务接口，便可通过合法接口注入错误传感数据，致使系统产生误判识别，进而威胁车辆行驶安全。

2. 云端服务器的渗透

在机器学习模型的训练环节，攻击者可注入恶意数据，这会直接导致模型后续出现决策失误与不安全行为。具体来看，攻击者会先渗透进入云端服务器，通过篡改服务器内的数据、注入有毒训练数据等操作，使训练出的模型在特定场景下出现异常决策。当这类存在问题的模型通过云端服务器批量导入车辆后，最终会批量引发车辆的异常行为。

3. 第三方服务攻击

云服务中的错误配置（如配置不当的开放 MQTT 代理、API 密钥泄露），以及第三方服务存在的安全漏洞，均可能被攻击者利用，导致安全风险沿整个供应链蔓延。例如，攻击者可通过非法手段入侵充电桩，使其在为车辆充电时，向车辆输送不稳定电流或超幅震荡电压，最终造成车辆出现不可逆的硬件损坏。

整车服务对外暴露的选择性、车内数据加解密机制的完备性，以及合理的数据安全管理体系的建立，是影响服务化开发阶段信息安全的关键因素。

3.5 服务颗粒度界定与治理困境

在 SOA 的架构设计中，如何界定一个服务的合适“颗粒度”是一项关键且复杂的挑战。它深刻影响着硬件资源利用率、系统灵活性以及开发测试的可行性。

1. 对硬件资源与性能的影响

如服务颗粒度过粗（如集成导航、娱乐、车身控制），将导致服务与硬件强绑定，资源调度僵化。即使更新简单功能，也需部署整个服务，浪费计算与通信资源。同时，粗粒度服务通常伴随庞大而复杂的接口，难以适应硬件或业务逻辑的变化，影响系统扩展性。

如服务颗粒度过细（如每个传感器独立成服务），将引发服务数量激增，服务间通信带来显著网络延迟与 CPU/ 内存开销，可能超出车载平台负载能力。此外，细粒度服务接口数量多、调用链路复杂，若缺乏统一的设备抽象层进行整合，将进一步加剧系统复杂度和通信成本。

2. 对可测试性带来的实际痛点

如服务颗粒度过粗，将导致可测试性差。以某个“自动开窗”场景为例，测试需启动整个“车身服务”及其所有依赖，导致用例复杂、环境搭建困难、问题定位低效。

如服务颗粒度过细，将简化单元测试，但导致集成测试复杂性剧增。例如，同一场景需协调位置、车窗、安全策略等多个服务，并模拟其交互，使测试覆盖与执行异常困难。接口调用链路长、依赖复杂，若缺乏统一的原子服务接口规范和服务编排机制，集成测试难度增加。

服务划分应遵循“高内聚、低耦合”原则，以业务边界为导向，并结合稳定的接口设计，特别是通过设备抽象统一硬件访问接口，通过原子服务构建可复用的基础能力，从而在架构层面支撑合理的服务颗粒度划分。在此基础上，建立强大的服务治理与测试平台，通过高效的服务编排、全链路追踪和测试等手段，来管理和减轻因颗粒度与接口设计不当带来的复杂性，从而在灵活性与可控性之间找到最佳平衡点。

4 服务化开发的方法

4.1 服务化开发方法论

4.1.1 服务化开发的特征

汽车电子电气架构正经历从分布式架构向域集中式架构、中央计算架构的深刻变革。在此进程中，传统紧耦合开发范式逐渐显现出显著局限性，包括功能与硬件强绑定导致的硬件资源复用率低、跨域协同效率不足，升级难度大耗时长升级耗时长等问题。面向此技术挑战，SOA 服务化开发范式通过标准化的全生命周期管理流程，依托松耦合的技术特性，构建起包含服务定义、接口设计、软件组件实现、系统集成验证在内的完整技术链路，有效推动车载功能开发模式从传统的硬件驱动向软件驱动转型。

服务定义需结合整车功能属性进行服务分层设计，依据动力、底盘、座舱、智驾等核心域需求确定“服务颗粒度”，避免过粗或过细。需要联合主机厂、Tier1 供应商、芯片厂商等车辆研发相关参与方制定统一的车载 SOA 服务元数据规范，明确服务参数格式与 QoS 等指标。通过松耦合接口和标准化协议打破域间壁垒。接口遵循“面向契约”原则，确保组件按契约对接，降低上下游跨厂商协作成本。

在实现阶段，服务封装为高内聚的独立组件，对外暴露标准接口，结合操作系统和中间件进行开发，应支持组件独立编译测试。操作系统中间件作为关键的协同枢纽，承担着复杂的系统集成任务，其凭借服务注册、发现、路由及监控等核心功能，化解多域异构系统与多元组件的集成挑战。在集成完成后，需开展系统性的测试验证工作，包括通过兼容性测试验证不同供应商组件间的接口适配性，运用性能测试模拟并发服

务调用场景下的响应时延，实施可靠性测试验证服务故障时的自动降级与恢复能力，以此保障系统的高可用性与稳定性。

4.1.2 服务化开发的原则

1. 服务分层原则：服务分层是一种软件架构设计中的重要概念，它指的是将一个复杂的系统按照不同的功能和职责划分为多个层次，每个层次专注于完成特定的任务，并为上一层或相同层提供服务。服务分层架构可参考 2022 年中国汽车工业协会软件分会发布的《软件定义汽车产业生态创新白皮书》中的架构。

2. 服务调用原则：基于服务分层架构，需严格规范层级间的调用逻辑。关于纵向调用，为确保系统稳定性，控制类服务接口遵循单层向下调用规则，禁止跨层调用导致的功能耦合；通知类服务接口则可支持跨层级灵活调用，以满足消息广播需求。关于横向调用，同层级服务间可自由交互，实现模块间高效协作。

3. 单一职责原则：强调每个服务应聚焦于单一业务功能或领域，通过职责的清晰界定，确保服务功能明确且边界清晰。这一设计理念不仅大幅降低了服务在设计、开发及维护阶段的复杂度，更显著提升了服务的可复用性与可扩展性（例如，“胎压监测服务”仅负责实时采集与分析胎压数据，而将轮胎温度控制的逻辑剥离至独立的“轮胎温控服务”，以此实现功能解耦与高效管理）。

4. 松耦合原则：服务之间的依赖关系应尽可能松散，减少对其他服务内部实现的依赖。当一个服务发生变化时，尽量不影响其他服务的正常运行。可以通过使用异步通信机制、消息队列等方式来解耦服务之间的直接依赖。

5. 可复用性原则：设计的服务应该具有较高的复用性，能够在不同的业务场景和项目中被重复使用。（如“车速采集服务”可被“巡航控制”、“仪表显示”、“里程计算”等服务复用）。

6. 可扩展性原则：服务应具备良好的扩展性，能够方便地进行功能扩展和性能提升，以适应不断变化的业务需求和整车功能增长。

服务分层原则划定系统架构边界，明确各层依赖关系。服务调用原则作为分层的运行保障，通过规范跨层调用维系架构稳定。单一职责原则确保服务职责清晰，是实现松耦合与复用的前提。松耦合原则借助异步通信等手段降低服务间依赖，增强系统弹性。可复用性原则基于分层与单一职责设计通用服务，提升开发效率。可扩展性原则依托上述原则，保障系统灵活应对业务变化。

4.2 服务化开发的流程

SOA 服务化开发仍需要遵循汽车电子架构开发通用的“V 模型”流程，在此基础上补充融入面向服务的开发方法与实施流程，该调整不会影响模型上下游的需求追溯链路及开发环节的衔接逻辑。在 SOA 服务化开发过程中，应遵循服务化开发方法论，以服务为核心驱动力，通过标准化接口与松耦合架构实现功能模块化与灵活扩展。具体流程分为以下几个关键阶段。

1. 系统需求分析阶段

核心任务：从场景、用户功能、系统用例识别并定义系统的能力，形成系统需求，作为系统设计活动的输入。

输出物：系统需求分析文件。

依赖工具链：工具详见 5.1 章节工具链功能要求内容。通过整车系统设计工具，完成系统需求分析。

2. 系统设计阶段

核心任务：根据系统需求形成系统逻辑架构和实现架构。在服务化开发中，需要明确服务、服务接口及依赖关系，基于功能需求完成系统级服务和服务接口定义。

- 服务定义
  - 设计服务架构，将系统功能分解为 SOA 服务。
  - 基于整车功能需求和关键规范中对服务分层的定义，完成整车服务库的定义。
  - 明确服务的依赖关系，需明确各服务间的调用顺序与数据流向（如“自适应巡航服务”需依赖“雷达感知服务”“车速控制服务”及“制动执行服务”）。
  - 定义服务版本号，通过语义化版本控制区分接口兼容性（如主版本号变更表示不兼容更新）。
- 服务接口设计
  - 服务接口规范定义：依据 SOA 服务设计规范与接口设计规范，完成系统

描述接口通信模式定义（包含请求 / 响应式交互、发布 / 订阅式通信）、数据结构设计（例如车速信号采用 uint16 数据类型，物理量单位为 km/h）及服务质量（QoS）约束条件设定（例如关键控制类服务端到端延迟需满足≤ 100ms 的要求）。

- 服务部署与通信设计
  - 结合物理拓扑及 ECU 功能，设计服务部署方式。
  - 定义 SOME/IP 部署与实例，包括传输协议、事件组等。
  - 配置服务发现机制（如 SOME/IP SD 协议），确保消费方能动态发现服务提供方。

输出物：系统描述文件。

依赖工具链：工具详见 5.1 章节工具链功能要求内容。通过整车系统设计工具，完成服务定义与接口设计。

3. 软件组件（SWC）设计阶段

核心任务：将服务映射为 AUTOSAR 软件组件，定义服务的提供方与消费方，定义组件内部行为与外部接口。

- SWC 划分与映射
  - 定义独立的最小单元软件组件（Software Component，SWC）实现服务，例如“灯光调节服务”由“灯光控制 SWC”提供。
  - 以“智能雨刮服务”为例，该服务由“雨量传感器 SWC”、“车速 SWC”和“雨刮执行 SWC”构成的组合软件组件实现，各组件间通过内部端口完成数据交互。
- 内部行为与接口映射
  - 明确 SWC 的内部行为逻辑，包括状态机设计与算法实现机制。通过“提供端口”（Provide Port）对外暴露服务功能，通过“需求端口”（Require Port）声明外部服务依赖关系。
  - 端口接口设计需严格遵循服务接口规范要求。以“车门控制 SWC”为例，其提供端口需完整实现“DoorLockReq”请求接口，该接口包含“LockState”（枚举类型，取值为 Locked/Unlocked）、“LockCtrl”（枚举类型，取值为 Locked/Unlocked）等关键参数定义，同时该 SWC 的需求端口需要依赖车速服务进行内部逻辑判断。

输出物：SWC 描述文件

依赖工具链：工具详见 5.1 章节工具链功能要求内容。使用整车系统设计工具，完成服务软件组件设计。

4. 软件组件实现阶段

核心任务：实现 SWC 业务逻辑，配置 SWC 依赖的通信配置与基础软件，完成组件级验证。

- SWC 代码实现
  - 基于 AUTOSAR AP/CP 架构，建议使用基于面向对象的语言（例如：C++/JAVA/Python 编写 SWC AP 侧代码），对于 CP 侧代码则建议使用简单高效的面向微控制器开发友好的主流语言（例如 C 语言）。
  - 通过 RTE（CP 运行时环境）接口或 ARA（AP）中间件实现 SWC 间的数据交互。
- 中间件与通信配置
  - 可选择 SOME/IP 作为跨控制器通信协议，配置服务 IP 端口、报文 ID 及序列化方式（如 Protobuf）；本地 ECU 内 SWC（CP）通过 RTE 直接通信，无需额外协议封装。
  - 基于服务配置文件（Arxml）生成相关中间件配置文件，实现 SWC 间的数据交互、通信配置（生成 Comm、EM 配置文件）。
- 组件级验证
  - 通过工具进行 SWC 接口一致性测试，验证数据收发正确性；通过工具（如 CANoe.SOA 模块）模拟服务调用，测试异常场景（如服务超时、数据丢失）。

输出物：SWC 源代码、中间件配置代码、组件测试报告。

依赖工具链：工具详见 5.1 章节工具链功能要求内容。基于行为建模工具完成服务行为逻辑建模、自动代码生成和模型仿真验证。基于基础软件与中间件配置工具完成 SOA 中间件配置、操作系统适配配置及组件化代码与配置文件生成。使用软件验证与集成验证工具完成服务单元级的 SiL 验证。

5. 系统集成验证阶段

核心任务：将 SWC 部署至目标 ECU，验证服务跨域协同能力与系统级功能。

- 硬件映射与部署
  - 根据 ECU 算力、网络带宽等约束，将 SWC 映射至物理控制器（如“辅助驾驶服务”部署至智驾控制器，“娱乐服务”部署至座舱域控制器），生成 ECU 配置描述文件。
- 系统级验证
  - 开展系统级集成联调测试，基于台架、实车等测试场景完成系统级功能联调测试，验证服务组合逻辑（如“远程启动服务”需依次调用“动力唤醒服务”、“空调预冷服务”、“车门解锁服务”）；监控服务调用延迟、CPU 占用率等指标，确保满足功能安全要求。

输出物：ECU 可执行文件、系统集成测试报告。

依赖工具链：工具详见 5.1 章节工具链功能要求内容。使用软件验证与集成验证工具完成多服务集成 SiL 验证、车载网络通讯验证、性能与可靠性仿真测试，完成整车软件的实时性和可靠性分析。

5 使能服务化开发的工具链

5.1 工具功能要求

面向汽车 SOA 服务化软件开发的工具链，需具备端到端、模型化、支持多平台变体管理的能力，深度融合 ASPICE 开发流程要求，重点覆盖系统建模、行为建模、基于软件配置与代码生成、虚拟化仿真与服务通信验证等关键环节，实现从整车系统设计到嵌入式实现的全链路模型驱动开发（Model-Based Development，MBD），提升开发效率、降低集成风险、增强跨团队协同能力。

汽车 SOA 软件开发工具链体系应以模型为核心，贯穿 ASPICE 所要求的系统工程与软件工程全过程，实现从“文档驱动”向“模型驱动”的转型。通过端到端集成的工具链，不仅提升 SOA 服务开发的效率与质量，更为未来软件定义汽车的敏捷迭代、持续交付与功能安全合规奠定坚实基础。

5.1.1 整车系统设计工具

整车系统设计工具是 SOA 软件开发流程的顶层设计中枢，承担从整车功能需求到服务化软件架构的转化任务。它以模型驱动工程为核心方法，基于 UML/SysML/AUTOSAR 建模语言，构建可执行、可追溯、可复用的系统架构模型，作为后续软件开发、仿真验证与配置生成的唯一事实源（Single Source of Truth）。

- 系统需求分析与用例建模：基于 UML/SysML 进行整车功能场景建模，识别服务需求（如车身服务、座舱服务）。输出可追溯的用例图、活动图、时序图，支持与需求管理工具集成。

- EE 架构与通信设计：结合整车网络拓扑，将服务映射到具体 ECU 节点。设计服务通信协议（基于 SOME/IP 或 DDS），定义序列化格式、QoS 策略、传输周期等。
- 服务定义与功能设计：结合功能边界划分服务单元，定义服务接口，包括方法（Method）、事件（Event）、属性（Property）及数据类型，构建服务依赖图，明确服务间调用关系。
- 服务软件组件设计：将服务进一步细化为软件组件（如符合 AUTOSAR 标准的 SWC），定义其内部结构与端口接口。
- 服务软件组件部署设计：将软件组件部署到不同 ECU 节点，支持不同车型不同 EE 架构的按需部署，确保同一服务架构适配不同车型。
- 应用开发与基础软件配置衔接：基于服务定义自动化转换与抽取生成标准模型文件格式（如 ARXML），对接行为建模工具，用于服务应用逻辑开发，对接基础软件配置工具用于自动化生成协议栈配置（如 SOMEIP）。

5.1.2 软件开发工具

通过模型化方式实现服务内部行为逻辑与算法的设计、仿真及自动代码生成，软件开发工具是 SOA 服务“模型即代码”的核心载体。

- 服务行为逻辑建模：使用图形化建模语言（如状态机）实现服务核心逻辑，支持浮点 / 定点建模，满足高精度控制与资源受限场景需求。
- 模型仿真与 MiL（Model-in-Loop，模型在环）验证：输入虚拟信号（如传感器数据），验证算法逻辑正确性；同时可联合多个服务的行为模型进行协同仿真，验证服务交互时序是否符合设计；基于仿真结果优化服务行为模型。
- 自动代码生成：从行为模型自动生成符合汽车行业标准的代码（C/C++），包含服务接口实现、内部逻辑代码，符合编码规范。
- 模型联合仿真：基于接口标准，集成车辆动力学等模型进行模型交换与联合仿真。

5.1.3 基础软件与中间件配置工具

针对不同硬件平台与操作系统，实现 SOA 服务底层软件（OS、中间件）的配置与自动化部署，确保服务在目标环境的可运行性。

- SOA 中间件配置：配置服务通信中间件（如 SOME/IP 协议栈），定义服务发现参数、序列化方式、通信 QoS 等。
- 操作系统适配配置：针对高安全确定性平台（如用于底盘控制的 AUTOSAR CP 平台），配置 OS 的任务调度（如服务任务优先级、周期）、中断管理，确保实时性；针对高性能智能平台（如用于辅助驾驶的 AUTOSAR AP 平台），配置进程管理、内存隔离等。
- 组件化代码与配置文件生成：支持将模型配置自动生成符合功能安全、实时性与通信标准的底层代码，支撑服务组件在真实 ECU 或虚拟环境中的运行。

5.1.4 软件验证与集成验证工具

服务化软件在软件验证与系统集成验证阶段，通过验证可辅助工程师进行服务的功能 / 性能 / 通信负载等验证。

验证工具可构建纯软件的虚拟环境（SiL，Software-in-the-Loop，软件在环），验证 SOA 服务在目标操作系统与硬件抽象层中的运行逻辑、性能与兼容性。

- 服务单元级 SiL 验证：在虚拟环境中加载单个服务的代码与模型，模拟服务依赖的外部接口；验证服务功能正确性、异常处理。
- 多服务集成 SiL 验证：部署多个服务到虚拟 OS，模拟服务间通信；验证服务交互逻辑、资源占用。
- 性能与可靠性仿真：模拟高负载场景，测试服务响应时间、吞吐量，验证是否满足 QoS 要求；注入故障，测试服务的容错能力。

验证工具可在 SOA 软件开发中对软件系统的时间特性的分析和优化，包括基于实测数据的时间与性能分析以及基于仿真数据的 WCET 和时延分析，保障汽车软件的实时性和可靠性。

- 时间数据采集与解析分析：采集车辆实测的服务调用时序数据，对时序文件进行结构化解析，还原服务调用全链路的时间序列关系，量化各环节耗时、资源冲突等问题，为调度优化提供数据支撑。
- 调度问题检测与模拟优化：基于解析后的时序数据，自动检测常见调度问题（如优先级倒置、资源抢占导致的延迟）；复现问题场景，验证优化策略（如调整调度优先级、资源分配方案）的有效性，输出调度优化方案。

验证工具可进行 SOA 服务的车载网络通信验证，模拟总线环境与对手件，确保服

务通信的正确性、实时性与协议合规性。

- 服务通信协议验证：针对以太网服务通信（如 SOME/IP），验证服务发现（SD）流程（如客户端是否正确发现服务实例、消息编码 / 解码、会话管理（如连接超时处理）。
- 对手件模拟与场景仿真：模拟服务的上下游对手件，构建复杂通信场景，验证服务在真实场景下的通信延迟。
- 总线负载测试：测量服务通信对总线带宽的占用，确保不超过设计阈值。

为了更好地支撑 SDV 整车软件研发，应支持工具间数据互通和共享，进一步可构建端到端的“一站式”工具链，自动衔接设计，开发，部署，验证等环节，实现多方协同中的数据同源、数据可视可追溯，支撑高效开发与集成。

5.2 服务共享的要求

随着汽车软件定义趋势深化，SOA 成为实现软件模块化、敏捷开发的关键。但当前车企 SOA 开发中，“资产复用难”与“三方工具衔接不畅”两大问题突出：一方面，服务设计资产（如接口模型、组件单元）在不同项目、团队间难以共享，导致重复开发；另一方面，车企存量开发中积累的三方工具设计资产，因格式不兼容无法融入 SOA 工具链，造成资源浪费。为此，SOA 工具链生态扩展需聚焦两大核心：SOA 资产的高效共享与复用，打破内部资产壁垒；兼容三方存量设计资产的无缝迁移，盘活既有资源。

SOA 资产作为开发核心，涵盖服务接口模型、组件化服务单元、仿真测试用例等，其共享与复用需通过“资产库集成 – 便捷调用 – 协同管理”实现全链路贯通，最大化资产价值。SOA 工具链需搭建统一的“共享资产库”，并与整车系统设计、行为建模等核心工具深度集成，打破资产孤岛。

1. 资产的高效复用与价值落地

共享资产库需简化复用流程，降低开发门槛，实现“即取即用”。应支持可视化拖拽复用，无需手动重建模型或编写代码。资产复用后，工具可根据当前项目需求自动适配参数，并生成配套文件（如 SOME/IP 通信配置、服务部署清单等）。

2. 三方存量设计资产的兼容迁移

车企在传统开发中积累了大量基于三方工具的存量资产（如 Simulink 搭建的 ECU 控制模型），SOA 工具链需兼容这类资产，通过标准化格式与接口，实现无缝迁移至 SOA 开发流程，避免资源浪费。例如，结构化资产（如接口定义、配置参数）应符合 AUTOSAR ARXML 格式，支持服务化工具链直接解析。可考虑引入 AI 技术优化迁移流程，例如通过机器学习自动识别非标准格式的三方资产、推荐转换方案等。

6 使能服务化开发的操作系统中间件

6.1 支持应用平台化

操作系统中间件应提供统一的服务化开发框架，服务化开发框架应屏蔽平台差异，服务定义和服务设计与运行平台解耦，组件间以标准的服务化接口进行交互。服务基于分层架构设计，平台化应用应关注业务功能，与平台能力解耦。

操作系统中间件应支持标准的服务化通信规范。通信规范应能支持跨平台互通能力，同时应兼顾接入部件的算力资源。当前用于车载服务化通信的协议规范主要有SOME/IP 和 DDS。服务化通信规范应支持跨平台的服务发布与订阅。

操作系统中间件应支持应用使用建模的开发方法进行平台化的开发。

操作系统中间件应为服务提供标准的运行时环境，应能支持应用快速在不同的实现架构下迁移。

操作系统中间件应能支持部署功能安全相关业务。提供故障检测，故障隔离和故障恢复能力。

6.2 支持服务灵活部署

基于服务灵活部署的需要，操作系统中间件应能支持服务动态发现，服务路由灵活配置、部署调优等功能，从而支持服务灵活部署。

操作系统中间件应能支持灵活的服务订阅机制，结合动静态的订阅机制，减少服

务化场景下的系统负载和报文开销，简化服务部署设计。

操作系统中间件应能支持灵活配置服务间通信路由，从而应对通信矩阵的变化，服务使用者无需重新编译。

应对部件的新增、应用的改变等情况。对资源受限的部件，服务动态发现面临资源的挑战。应具备流控、延迟发送等机制，降低短期负载冲击。

服务应与平台能力解耦，服务迁移时，目标运行平台应提供对应的平台资源（计算、存储、通信、诊断等）。

由于 ECU 处理能力有限，需要根据计算资源、存储资源、通信开销等选择最佳部署策略，对 ECU 间的软件进行部署调优，且业务不感知，从而优化服务化性能，充分发挥服务化的优势。

6.3 支持多方协同开发

传统架构下 OS 与应用强耦合，OS 需要根据业务变更频繁更新。在服务化架构下，操作系统中间件需支持平台软件与应用的解耦，支持 SDV 多方开发者在独立运行环境中进行开发，即独立设计与开发服务、独立配置平台软件参数，独立生成代码、独立集成，独立验证，可支持独立更新。

服务间应通过标准的服务化接口交互。多方协同开发需要保证接口向前兼容，需要提供接口兼容性机制。

通过支持 SDV 软件多方协同开发，做到 OEM 和 tier1 解耦开发，以及 OEM 内部解耦开发。通过并行开发缩短开发周期，以满足服务化架构下敏捷开发和快速迭代的诉求。

6.4 提供整车服务框架

操作系统中间件应为应用软件提供整车服务框架，能够屏蔽整车 E/E 拓扑，屏蔽传感器、执行器、控制器等异构芯片的部署细节，提供跨核跨域融合的、整车级的、统一的业务标准访问接口，可以支撑业务层软件快速开发，缩短产品化周期。整车服务框架提供的功能可包括车调度，通信，诊断，存储，安全，系统管理等公共服务。

6.5 增强服务化通信能力

针对车内网络的特点和部件处理能力的差异，服务化通信应满足较高的规格要求。操作系统中间件应对服务化通信进行优化。

1. 降低服务报文流量，避免服务化通信的流量尖峰对处理能力弱的部件造成冲击，提升整体通信质量。

2. 提升服务化通信可靠性。提供服务化通信 Qos，保障关键业务数据通信优先处理。

3. 关键业务提供安全认证能力，控制访问权限，提升安全性。提供服务化通信的机密性，真实性和完整性的保护机制。

4. 服务通信报文聚合和流控能力。应对服务报文进行聚合，对报文流量进行监控，对于异常的报文流量应进行控制。

5. 服务化通信应能接入传统总线（如 CAN/LIN 等），应精简传统总线协议栈，减少开销。服务化通信应满足端到端的时延要求。

7 服务化开发的信息安全与功能安全

7.1 信息安全

SDV 信息安全应遵循三大原则：一是深度防御，为车辆、生产和后端建立多层防护，涵盖工厂 IT、网络、生产线，外部通信、电子电气架构、车内通信、ECU，以及网络、身份、端点、应用、数据等方面；二是设计安全，从开发伊始就将安全融入；三是持续风险管理，包括入侵检测与防御系统（IDPS）、分布式入侵检测系统（dIDS）、车辆安全运营中心（VSOC）、软件在线升级（OTA）等。企业应建立覆盖车辆全生命周期的信息安全管理体系，并严格遵循管理体系实施产品开发工作，覆盖开发、生产以及后生产各阶段。

SOA 服务安全应贯穿“设计、开发、部署、运维和下线”全生命周期，实现设计即安全（Secure by Design）。相关技术要求应满足相关信息安全国标的规定。其中针对 SOA 服务的信息安全要求包括：

1. 应对 SOA 架构下所有服务进行识别、评估、分类并进行相应的风险处置，并核实已识别风险得到处置。

2. 开发应满足安全编码规则与安全代码审计要求。

3. 应对服务和车内传输数据进行识别并分类分级。针对不同数据实施相关安全措施，应符合相关数据安全国标的规定。数据处理活动中的数据车内处理、默认不收集、精度范围适用、脱敏处理、个人同意及显著告知等要求，应符合符合相关数据安全国标的规定。

4. 应对服务的注册、发布及订阅的主体进行身份鉴权。应对服务的注册、发布及

订阅的主体实施权限管理，实施最小权限。应对服务的注册、发布及订阅的主体运行软件实施真实性和完整性进行验证，覆盖软件升级和启动。

5. 应对服务的真实性和完整性进行验证。

6. 应具备关键安全事件日志记录功能，如关键服务的注册、发布以及订阅。应采用安全机制保护存储的安全日志和事件，防止被修改和未经授权的删除。

7. 应建立安全监控平台，实施检测服务异常，检测到异常时记录并上报。

8. 服务下线注销时，应对相关敏感信息实施安全销毁，如密钥、证书以及服务敏感数据。

7.2 功能安全

软件定义汽车的发展导致了车辆功能开发模式的根本性变革。传统汽车中，各功能模块通常由独立的 ECU 实现，供应商需对整个 ECU 的功能安全负责。而在 SDV 基于 SOA 的开发模式中，软件服务可分布在多个硬件平台上，由不同供应商开发，这使得功能安全责任分散，需要建立新的责任边界与安全保障机制。同时，由于服务可动态更新与重组，功能安全必须贯穿软件全生命周期，从设计、开发、部署到运行时监控，形成持续的安全保障体系。

传统安全方法面临的挑战主要体现在三个方面：一是分布式服务架构导致的安全分析边界模糊，使得危害识别与风险评估更加复杂；二是软件服务的动态组合与更新，要求安全机制必须具备足够的灵活性与适应性；三是面向服务的通信模式引入了新的安全威胁，如服务未经授权访问、消息篡改或服务拒绝等。

为应对这些挑战，软件定义汽车的功能安全需构建多层次、全栈协同的安全框架。这一框架不仅需要涵盖系统性失效和硬件随机失效，还需考虑服务交互、资源竞争及动态部署带来的新型风险。同时，随着自动驾驶等级的提升，功能安全需要与预期功能安全（SOTIF）紧密结合，降低因设计不足或性能局限导致的安全风险。

7.2.1 全生命周期的功能安全

SDV 的服务化，核心在于 SOA 来实现硬件资源抽象、软件解耦和快速迭代。这不但要满足传统功能安全的要求，而且在全生命周期不同的阶段对其提出了更高、更复杂的要求。

1. 需求阶段：协同的安全目标定义

整车厂联合平台商，供应商开展联合 HARA（危害分析与风险评估）分析，定义安全目标和 ASIL（汽车安全完整性）等级，明确服务的安全职责与边界，针对服务订阅场景，预设不同服务组合下安全等级可配置机制。

2. 设计阶段：分层解耦下的安全架构构建

设备抽象层负责封装底层硬件差异，将硬件能力以标准化接口方式暴露给上层。在功能安全要求上，设备抽象层需确保硬件状态监控的可靠性，对传感器输入数据实施多重校验机制；对执行器输出设计安全状态，确保在发生故障时能够进入预设的安全状态。

原子服务层作为中间层，承担着承上启下的关键作用。该层需实现服务接口标准化，定义统一的服务接口，确保不同供应商开发的软件组件能够安全交互。原子服务层还需提供服务健康状态监控能力，检测服务异常并实施必要的服务降级或重启策略。

应用 / 组合服务层处于最顶层，负责用户需求逻辑的实现，通过调用原子服务层提供的接口，组合出多样化的场景化应用。该层需实现服务依赖管理，确保关键服务的可用性，并设计降级策略，当非关键服务失效时仍能保持基本安全功能。

3. 开发阶段：代码与模型的合规与验证

功能安全相关的服务代码需遵循行业标准，进行静态代码分析和动态单元测试。含 AI 模型的服务需满足训练数据覆盖极端场景，完成模型鲁棒性测试等。

将安全要求分解到每个服务，每个服务需通过 SiL 测试，验证独立运行及与依赖服务协同的安全性。例如，自适应巡航服务需单独测试“前车急刹响应”，并联合制动服务测试“减速协同逻辑”。

4. 部署阶段：OTA 升级的安全防护

基于服务影响范围实施分级验证，非功能安全相关服务升级仅需完成模块测试；功能安全相关服务升级可补充 HiL（Hardware-in-the-Loop，硬件在环）测试及整车验证，验证与传感器、执行器的兼容性。采用灰度发布，实时监控异常指标。

对含 AI 模型的服务，还需要进行模型特有的风险测试，如对抗样本注入，边界条件测试，算力压力测试等。

5. 运营阶段：车云协同的动态防御

部署车载安全监控服务，实时采集服务运行数据，云端通过大数据分析识别异常模式，结合用户行为数据与环境变化，定期更新 HARA 分析，调整安全目标。例如：在冰雪地区用户占比提升时，调整稳定性控制服务的 ASIL 等级。

7.2.2 功能安全开发流程与管理

功能安全的实现不仅依赖于技术措施，更需要系统化的流程与管理的支撑。在软件定义汽车的环境下，功能安全开发流程需与传统汽车开发流程深度融合，并覆盖完整的生命周期。软件定义汽车的特性要求将功能安全流程与软件开发流程、网络安全流程及质量管理流程进行深度整合，形成统一的工程体系。

软件定义汽车的服务化从根本上重塑了功能安全的流程与管理。它要求传统的、基于固定硬件和信号链的功能安全流程，向一个更灵活、更持续、更注重迭代和协作的模式演进。

服务化架构对功能安全开发流程的要求包括：

- “左移”与持续的安全活动。要求安全活动必须更早开始、更频繁地进行，并与敏捷开发节奏同步。
- 基于场景与正向开发。要求安全分析必须从基于信号转变为基于服务交互和用户场景。
- 扩展的分析方法与工具。要求传统的方法如 FMEA（失效模式与影响分析）和 FTA（故障树分析）需要升级，以应对服务动态绑定和软硬件解耦带来的复杂性。

服务化架构对管理体系的要求包括：

- 跨功能团队的深度融合。要求打破功能安全团队、网络安全团队、软件架构团队和云平台团队之间的壁垒。
- 工具链与数据管理的升级。要求建立端到端、互联互通的工具链，以管理复杂性。
- 全生命周期的安全管理。要求安全管理必须覆盖车辆从生产到报废的整个生命周期，特别是 OTA 升级环节。

总而言之，软件定义汽车的服务化开发对功能安全和信息安全的要求，可以概括为：在追求软件灵活性和迭代速度的同时，要求必须通过“设计安全”和“运行安全”

相结合的方式，为分布式的、动态的服务集群建立起一个具有韧性的安全体系；要求安全理念从“组件安全”升级为“系统架构安全”和“通信安全”，并贯穿于车辆的整个生命周期；要求在服务化架构下 AI 模型的测试不再是一个孤立的、静态的环节，而必须成为一个持续、集成、基于场景且高度自动化的系统级工程；最后，要求传统功能安全（如 ISO 26262）与新的预期功能安全（SOTIF）、车辆信息安全（ISO/SAE 21434，GB 44495）最佳实践的深度融合。软件定义汽车的服务化要求功能安全与信息安全紧密结合。

8 服务化开发的其它要求

8.1 系统级运维能力的要求

服务化开发相比传统开发模式，维测的难度进一步提升，快速定位定界是提升开发效率的重要一环，服务化开发对运维能力提出了更高的要求。

服务化开发的系统级运维能力需要一套贯穿汽车全生命周期，涵盖车端和云端，保障软件持续交付、车辆稳定运行、用户体验进化及数据价值挖掘的综合性工程与管理体系。

8.1.1 技术平台的运维能力

1. 车辆端运维方面，支持软件包管理，支持固件 / 软件的空中升级。支持状态管理，可实时采集车辆数据（日志，性能指标，故障码），并上传云端。支持远程诊断，能够远程定位问题、执行特定指令或配置。

2. 云端运维方面，应具备海量数据处理平台，能够存储分析来自车端的运维数据。应具有运维监控中心，统一监控所有车辆、云端服务和数据中心资源的健康状况。应具有 AI 与大数据分析能力，利用数据实现预测性维护（如预测电池衰减）、智能诊断（通过历史数据匹配故障模式）。

3. 主机厂研发端运维方面，应具备软件从代码提交、自动测试到车端部署的自动化流水线，应具有仿真测试环境，可利用数字孪生技术在虚拟环境中验证软件，降低实车测试成本和风险。

4. 供应商研发端运维方面，供应商的服务能力需覆盖从需求分析、模型验证到

OTA 升级的完整服务闭环，提供产品全生命周期支持。

8.1.2 维测日志管理

实施”端测智能采集，传输分级调度，云端智能分析”的全链路解决方案，降低运维成本，提升问题定位效率和准确性。

1. 端侧采集方面：应建立分级日志体系，明确定义日志级别和分类、统一日志格式规范、分层分级上报。支持异常条件触发高密度日志记录，支持基于特定的车辆信号或服务事件来开启或增强相关服务的日志记录。支持关键事件过滤，确保 FATAL 和 ERROR 级别的日志 100% 记录和上报，对 INFO 等低级别日志按百分比的采样率记录。日志支持分级存储和滚动删除策略，保护高优先级日志。上报前对日志进行压缩，减少网络传输量。

2. 日志传输方面：定义高优先级通道，用于上传 FATAL，ERROR 日志和关键警报，定义低优先级通道，用于上传采样，压缩的常规日志。

3. 日志分析方面：应具备集中式日志仓库。对日志做关联分析和原因定位，将日志信息与系统的性能指标、车辆信号进行关联分析，发现潜在因果关系。具备学习能力，自动从海量日志和指标中检测出异常模式并及时预警，自动聚类类似日志信息并识别出高频错误模式，支持预测性维护，基于历史日志和车辆数据，预测某些部件或服务可能发生的故障。

8.2 服务编排能力的要求

服务化开发模式下，应该具备服务编排能力，从而支持应用快速上线。通过服务编排引擎定义场景，按需编排服务。服务编排引擎应可以通过简单、便捷、易控的方式进行服务编排。

基于整车控制功能服务化，开发场景编排功能，在保证整车功能及信息安全的前提下，开放整车编排能力，为用户提供场景自定义功能，从而实现复杂场景（如“休憩模式”、“夏日通勤温控”、“后排休憩娱乐”等）。

服务编排引擎作为 SOA 架构中的“指挥家”，负责对整车原子化服务进行有机整合，按业务逻辑定义执行流程，从而将简单的整车服务转化为丰富的场景化体验。为实现通过服务编排对整车的精准、高效管理、服务编排引擎需满足以下核心要求。

1. 图形化与低代码的编排界面

编排引擎应提供直观的图形化拖拽界面，允许产品经理或业务工程师通过绘制流程图的方式定义场景逻辑，而非编写复杂的代码。这极大地降低了编排任务的技术门槛，实现了业务逻辑与技术实现的解耦，从而支持用户（非开发人员）快速构建和验证场景，显著加速应用功能的上市周期。

2. 流程控制与逻辑表达能力

支持丰富的流程控制原语，包括但不限于：

推荐场景：根据车型主要用户群体预设高频用车场景，支持用户直接使用或修改后使用，同时也能让用户快速熟悉场景编排逻辑；

创建场景：支持用户创建个性化场景，用户可自定义场景触发条件及执行动作，支持试运行场景和一键删除场景功能；

添加条件：被创建的场景需要满足的前置条件（如车速、电量、阳光强度、时间、地点等）。

添加动作：被创建的场景需要执行的动作（如车门开度、车窗开度、导航等），动作支持顺序执行、并行执行、分支执行、以及动作异常处理机制（即当某个服务调用失败时，引擎应能捕获异常并执行预设的回退逻辑，如在“雨天模式”编排中，若自动升窗服务失败，则应触发补偿动作（如向用户发送通知），确保系统状态的一致性）。

编辑场景：用户可根据个人驾驶习惯或临时场景需求对场景列表内任一场景卡片重新修改内容后保存（包含推荐场景）。

3. 服务治理与可靠性保障

场景编排引擎本身必须具备高可用性和强大的服务治理能力。

服务熔断与降级：当某个整车服务（如网络音乐服务）不可用时，引擎应能自动熔断对该服务的调用，并执行降级策略（如切换至本地音乐播放），保证核心流程的可用性。

超时控制：为每个服务调用设置合理的超时时间，避免因单个服务的长时间无响应导致整个场景“卡死”。

状态持久化：对于长时运行的编排流程，引擎需要能够持久化执行状态，即使在系统重启后也能从断点恢复，确保业务流程的完整性。

4. 版本管理与 OTA 部署能力

为了支持业务的持续迭代，场景编排引擎必须支持服务流程的版本管理。新的场景流程或流程更新应能通过 OTA 部署的方式上线，无需重启整个系统，从而实现功能的无缝升级与快速回滚，最大限度保障用户体验的连贯性。

8.3 硬件能力的要求

为了适应快速迭代且充满不确定性的软件需求，需要硬件具备预留能力和可扩展性，为未来 1 ~ 3 年甚至更久可能出现的新功能、新算法、新服务预留硬件能力。保障持续升级能力，确保硬件平台在整个生命周期内都稳定，高效的承载频繁的软件 OTA。。

例如，在选型芯片或域控制器时，其算力不只为当前软件需求设计，而是预留性能余量。设计网络时预留网络带宽，为关键控制信息设计冗余通讯路径。为关键的计算单元、传感器单元提供独立的电源供应和电源管理。

9 总结与展望

白皮书以“软件定义汽车的服务化开发”为主线，系统梳理了汽车产业在“新四化”背景下的技术演进逻辑与生态变革趋势。通过对软件定义汽车（SDV）及面向服务架构（SOA）的深入剖析，明确了服务化开发是推动汽车产业全链路变革的关键路径。白皮书以服务化价值为锚点，从服务化开发的挑战出发，结合行业服务化开发实践，提炼了面向服务化开发的开发方法、工具、操作系统中间件、安全与运维体系等关键要求，形成了指导 SOA 服务化落地的系统指南。

根据前文描述，总结出以下关键认识：

1. 服务化需要操作系统中间件、工具链、安全、硬件等多要素的协同支撑，进一步释放软件定义汽车的价值。

2. 服务化开发有助于研发效率提升，能够进一步促进软硬件解耦，提升软件部署灵活性，更好地支持跨车型复用软件。

3. 服务化开发方法在现有整车开发 V 模型的基础上嵌入面向服务的理念，在整车系统层面开展设计、开发、验证，充分发挥 SOA 优势。

4. 结合工具可进一步提升服务化开发效率，支持多方协同开发。现有工具链存在碎片化挑战，可构建端到端的“一站式”工具链，自动衔接设计，开发，部署，验证等环节，实现前后端信息全局共享，支撑高效开发与集成。

5. 随着软件复杂性的提高和服务报文的增加，对服务通信的能力要求越来越高，对通信总线的吞吐时延以及操作系统中间件的通信机制提出更高要求。

6. 为应对SDV生命周期内软件的快速迭代，硬件应具备资源预留能力和可扩展性。

下一步，建议结合本白皮书提出的服务化开发方法、工具链和操作系统中间件等要求，在行业范围内开展更加深入广泛的服务化开发研究与推广工作，推动服务化的

深化，进一步释放软件定义汽车的价值。

建议搭建行业共享共建的服务化开发社区，分享行业优秀实践和案例，及时整理总结服务化开发经验，提供基本的服务化开发环境和配套软件，形成丰富的服务化开发生态。

在前期已经开展的服务化架构和接口标准化基础上，建议进一步开展对服务化开发方法论、工具和操作系统中间件的功能和性能要求、安全与运维要求等方面的标准化工作，牵引行业更加有效地实现服务化，形成系统化的 SOA 技术标准体系。

展望未来，软件定义汽车的服务化将向更深层次、更广范围演进，成为推动汽车产业持续创新与数字化转型的核心力量。

中国汽车工业协会　软件分会
软件定义汽车（SDV）专业委员会
地址：北京市丰台区汽车博物馆东路诺德中心11号楼33层
邮政编码：110160
电话：010-63979900
网址：sdv.caam.org.cn