1040601 OTA车端概要设计

文件制修订记录书

制修订记录				
文件版别	修改人	制修订日期	制修订页次	制修订摘要（增、减、改、项目）
v0.0.1	洪雕	2024.6.16		1.删除备选架构 2.补充主要业务交互时序 3.整体架构还原为层次关系
v0.0.2	喻志敏	2024.6.22		1.调整目录结构 2.补充整车拓扑和IVI部署图
v0.0.2	洪雕	2024.6.24		1、增加名词解释 2、图中模块名称与文档内容统一 3、删除安全模块中的车内零件间文件传输 4、删除安全模块中零件划分安全存储区域 5、删除车端信息存储安全

1 目的(Objective)

编写本文档主要用于说明公版FOTA项目的具体软件架构设计。其中，具体说明了软件架构设计的目标和约束需求，规定了系统的开发环境和运行环境，阐述了逻辑架构设计的整体思想和具体设计的实现，介绍了系统的主要业务流程和系统的功能模块的划分以及模块之间的联系，为团队成员在项目的各个阶段（从售前咨询到开发实施，再到后期的维护和升级）都能够有一个清晰的认识和指导。

2 范围(Scope)

本车端软件架构设计书的适用范围主要包括以下几个方面：

a. **整车FOTA升级系统的开发**：架构设计书作为整车FOTA升级系统开发过程中的指南，帮助项目团队理解系统的整体架构和各个模块之间的关系，从而有针对性地进行开发工作。
b. **系统实施和集成**：架构设计书可以作为系统实施和集成的依据，帮助实施团队了解系统的设计原理和核心逻辑，确保实施过程的顺利进行。
c. **系统维护和升级**：架构设计书可以作为维护和升级工作的参考，帮助团队在后续的维护和升级过程中更好地理解系统的结构和功能，提高工作效率。
d. **项目交流和沟通**：架构设计书可以作为项目交流和沟通的工具，帮助团队成员之间共同理解系统的设计和实现细节，减少沟通误差，提高工作效率。

总之，本软件架构设计书适用于整车FOTA升级系统的开发、实施、维护和升级等各个阶段，可以为项目团队提供指导和支持，确保项目的顺利进行和最终交付的成功。

3 缩写与定义(Abbreviations and definitions)

3.1 术语与缩略语描述

下列术语和缩略语适用于本文件，术语和缩略语描述如下所示：

缩写(Abbreviation)	英文全称(Full Name)	中文描述（Chinese description）
OTA	Over-the-Air Technology	远程无线升级技术
FOTA	Firmware Over-The-Air	固件空中升级技术
ECU	Electronic Control Unit	电子控制器单元
TSP	Telematics Service Provider	远程通信服务供应商
HMI	Human Machine Interface	人机交互接口
FBL	Flash Bootloader	闪存引导加载程序
CDN	Content Delivery Network	内容分发网络服务器系统模块，用于管理升级包
OTA Server	Over the air server	OTA管理平台，负责车辆的版本管理，任务发布，升级结果展示等
MCU	Micro controller Unit	微控制单元，作为单独升级模块
APK	Android application package	OTA 人机交互Android应用程序包
UCM	Update Control Master	OTA控制主模块，用来负责OTA升级主流程控制管理
Lite	Update Control Lite	OTA 控制从模块，用来负责提供OTA升级所需的关键能力支持（如文件下载，车辆状态信息获取等）。
Lite2hmi	Update Control Lite to HMI	OTA 控制从模块，用来负责对接HMI界面展现相关内容的交互
UA	Updata Agent	差分升级模块，负责SOC的升级
OTA Client	Over the air client	OTA车端车云通信客户端，用于车云通信
DIM	Device Interface Manager	设备交互管理模块，用于管理需要被升级的零件
UIM	User Interaction Manager	用户交互管理模块，用于管理人机交互
OTA-Client	OTA-Client Manager	OTA客户端管理模块，用于管理车云通信

注:

OTA-Master，完全等同于FOTA-Master，UC-Master，本文档范围内统一简称为UCM。

OTA-SubMaster，完全等同于FOTA-Lite，UC-Lite，本文档范围内统一简称为UCL。

Diagnostic Engine，完全等同于Vehicle Diagnostic Manager，本文档范围内统一简称为VDM。

3.2 文档与接口编号规则

编号命名规则旨在为事物或实体分配唯一标识，以便于组织、管理和辨识。本架构文档与接口编号规则约束如下：

- **文档编号规则:**
  - 文档编号应遵循以下格式：《项目_软件元素_车端软件架构设计书》。
- **接口编号规则:**
  - 接口编号应遵循以下格式：项目_SWE2_软件元素_一级组件编号_二级组件编号，其中组件编号默认从0001开始（保留四位有效数字）。

通过这样的编号规则，可以确保每个文档和接口都有一个唯一且描述性的标识符，便于项目的组织、管理和辨识。同时，这也有助于在项目团队成员之间提供清晰的沟通基础，以及在项目的整个生命周期中维护文档和接口的一致性和双向追溯。

4 预期读者和阅读建议(Intended readers and reading suggestions)

4.1 预期读者(Intended readers)

- 项目开发工程师
- 解决方案工程师
- 产品经理
- 项目经理
- 测试工程师
- FAE工程师

4.2 阅读建议(Reading suggestions)

本架构设计文档是为公版FOTA项目整车FOTA（Firmware Over-The-Air）软件架构而编制。它旨在作为后续FOTA软件开发的指导文件。在深入研究本文档之前，我们建议读者参考以下相关文档，以确保对项目的全面理解和技术背景的充分掌握。

5 架构设计目标和约束(Architectural design objective and constraints)

5.1 架构设计目标(Architectural design objective)

本架构设计文档目标如下：

a. **明确需求和功能**：明确FOTA车端软件架构平台软件系统的功能和需求，包括各种用户需求、业务需求和非功能性需求。
b. **定义系统结构：**根据需求和功能确定FOTA软件系统架构图，并明确各层次和组件的职能，定义组件间边界。
c. **确定系统依赖及集成消耗：**定义FOTA运行环境依赖、资源消耗、性能要求以及验证方式。
d. **指导开发和实施：**为实际开发确定基本框架，指导开发人员实施后续FOTA上应用开发任务，指导详细设计、编码和集成验证等工作。

综上，整车FOTA软件架构设计目标是为了确保系统能够满足客户的需求和要求，同时能够保持高可靠性、高安全性、良好的用户体验和易用性，以及良好的可维护性和性能优化能力。

5.2 约束(Constraints)

5.2.1 软硬件环境约束(Constraints on software and hardware environment)

5.2.1.1 开发集成环境(Development integrated environment)

基于本架构设计文档，OTA支持的操作系统环境描述如下：

操作系统	版本号
Linux	3.4
Android	M，N，O，P
QNX	7.0

FOTA软件开发依赖的第三方开源组件库依赖情况如下：

开源组件			许可证		大小
名称	版本	下载链接	类型	链接	库

openssl	1.1.1n	https://www.openssl.org/source/old/1.1.1/openssl-1.1.1n.tar.gz	GNU	https://github.com/openssl/openssl/blob/OpenSSL_1_1_1n/LICENSE	3.64MB（静态库）
curl	7.88.1	https://github.com/curl/curl/releases/download/curl-7_88_1/curl-7.88.1.tar.gz	MIT	https://github.com/curl/curl/tree/curl-7_88_1/LICENSES	10000KB（静态库）
leveldb	1.22	https://github.com/google/leveldb/archive/refs/tags/1.22.tar.gz	BSD	https://github.com/google/leveldb/blob/1.22/LICENSE	989KB（静态库）
iniparser	4.1	https://github.com/ndevilla/iniparser/archive/refs/tags/v4.1.tar.gz	MIT	https://github.com/ndevilla/iniparser/blob/v4.1/LICENSE	12KB（静态库）
EasyLogger	2.2.0	https://github.com/armink/EasyLogger/archive/refs/tags/2.2.0.tar.gz	MIT	https://github.com/armink/EasyLogger/blob/2.2.0/LICENSE	22KB（静态库）
cJSON	1.7.15	https://github.com/DaveGamble/cJSON/archive/refs/tags/v1.7.15.tar.gz	MIT	https://github.com/DaveGamble/cJSON/blob/v1.7.15/LICENSE	27KB（静态库）
zlib	1.2.11	https://github.com/madler/zlib/archive/refs/tags/v1.2.11.tar.gz	GNU	https://github.com/madler/zlib/blob/v1.2.11/README	115KB（静态库）

上述所选用的第三方开源组件在嵌入式领域运用中，稳定性、易用性、扩展性、性能、成本、复杂度，安全性等方面都有优秀的表现，故选择该三方开源组件库使用。

5.2.1.2 硬件配置建议(Hardware configuration suggestion)

配置参数	参数建议
芯片平台	建议为主流的4G通信能力平台，如高通海思等平台
CPU架构/主频/核心数	建议Cortex A7/1GHz/双核或以上配置

操作系统版本	Linux 3.4（或更高），包含基础类库
FLASH类型，大小	NAND/NOR/EMMC，16GB（或更高）
DDR类型	建议512MB（或更高），DDR3（或更高）
文件系统类型	EXT2/EXT4/UBIFS/YAFFS2/FAT
作为主控升级单元的可控空间（最低要求）	建议大于（所有ECU升级整包大小之和× 4.5） + APP自身程序 + 程序日志 + OTA配置信息

5.2.1.3 存储空间要求(Storge space requirements)

宿主ECU供应商需根据OTA业务需求，分配文件存储空间，用于缓存从OTA管理平台下载的升级文件。存储空间还需符合如下要求：

- 预留存储空间由产品需求、车机系统开发，和OTA开发根据实际业务情况共同协商定义；
- 建议有独立分区，该分区被OTA应用独占使用，用于存放升级包；
- 该存储空间具备一定的安全防护条件，确保数据库、配置文件、升级文件不被用户篡改，或者被第三方窃取；

5.2.2 接口/协议的约束(Interface/protocol constraints)

接口和协议的约束是确保系统各组件之间有效、一致和安全通信的关键要素。这些约束定义了组件如何相互连接、数据如何交换以及通信过程中必须遵守的规则。以下是一些常见的接口和协议约束：

- **内部接口/协议的约束**

在内部接口方面，模块内采用函数调用、参数传递、返回值等方式进行信息传递。接口传递的信息将是以基础数据类型或数据结构封装的数据，以参数传递或返回值的形式在模块内传输。

- **外部接口/协议的约束**
- 模块间及进程间通信时，需要使用ABUP XRPC服务化通信机制，具体通信协议需要参考各服务对外提供的Method、Event信息。
- FOTA与外部云平台通信，需要遵循FOTA约定的车云协议。
- 通信协议必须包含加密、认证和授权机制，以保护数据的机密性、完整性和可用性。
- 接口和协议必须与现有的系统组件兼容，确保新旧系统能够无缝集成。
- 接口和协议应设计得足够灵活，以支持未来功能的添加和系统的扩展。

5.2.3 软件质量的约束(Software quality constraints)

本项目代码需要符合如下质量标准和约束：

- 符合Helix QAC MISRA-2012(c++)/2016(c) 标准。
- 单元测试（VectorCAST）语句覆盖率达80%。
- 集成测试（VectorCAST）核心模块覆盖率达90%以上。
- 冒烟测试（台架模拟测试）7*24小时压力测试，软件正常运行、无异常抛送。
- 7*24小时压力测试，软件无内存泄漏。

5.2.3.1 性能约束(Performance constraints)

基于本架构设计文档，FOTA UCM运行时对宿主ECU的RAM/ROM/CPU要求如下（具体情况需要结合实际FOTA应用场景做调整）：

配置参数	参数要求	备注
ROM	20MB	软件自身占用内存空间大小
RAM	20MB ~ 25MB	软件运行时占用内存空间大小
CPU	6000DMIPS	

注：不同编译链编译输出的ROM会略有差异。

5.2.3.2 可靠性约束(Reliability constraints)

- 功能规约：明确定义和规划系统的功能需求，并确保软件系统能够按照需求进行正确的操作。功能规约提供了设计和实现软件的基础，确保软件系统的核心功能得以正确执行。
- 可行性约束：在软件系统设计和开发之前，要评估项目的可行性，并确定项目的限制条件。这些约束包括时间、预算、资源和技术等方面的限制，确保软件系统在现实条件下能够实现可靠性目标。
- 健壮性约束：软件系统应该具有健壮性，即能够处理异常情况和错误输入，保证系统能够正确地响应和恢复。在设计和实现过程中，需要考虑边界条件、异常处理和错误检测等机制，以增强系统的健壮性。
- 可用性约束：软件系统应该具备良好的可用性，即用户能够方便、高效地使用系统，并能够获得所需的服务和功能。通过提供友好的用户界面、良好的用户体验和合理的系统响应时间，增强软件系统的可用性。

- 安全性约束：软件系统的安全性是保护系统免受恶意攻击和非法访问的能力。软件系统的设计和实现应该遵循安全最佳实践，包括数据加密、身份验证、访问控制等安全机制，防止潜在的安全威胁。
- 可维护性约束：软件系统的可维护性是指对系统进行改进、修复和更新的能力。系统应该具备清晰的架构设计、模块化的代码、良好的文档和适当的注释，以便于后续的维护工作。
- 性能约束：软件系统应该具备一定的性能水平，能够满足用户的需求并在合理的时间内完成任务。在设计和实现过程中，要考虑系统的响应时间、吞吐量、资源利用率等性能指标，并进行性能测试和优化。
- 可测试性约束：软件系统应该具备良好的可测试性，即能够方便地进行单元测试、集成测试和系统测试等各个层面的测试工作。通过合适的测试策略和测试工具，可以提高软件系统的可靠性。

以上是XOTA可靠性的一些约束，这些约束在软件开发的不同阶段都需要考虑和满足。同时还需要根据具体的应用需求，制定适合的可靠性约束和相应的实施措施。

5.2.3.3 可扩展性约束(Extensibility constraints)

组件和应用程序的接入需要符合XOTA平台上的模块化及服务化框架的定义与约束规范。

6 系统总体设计(System overall design)

6.1 J0x整车架构

本项目是基于J0x车型架构展开OTA车端设计，车端OTA组件主要部署于IVI中，整车网络拓扑见下图。

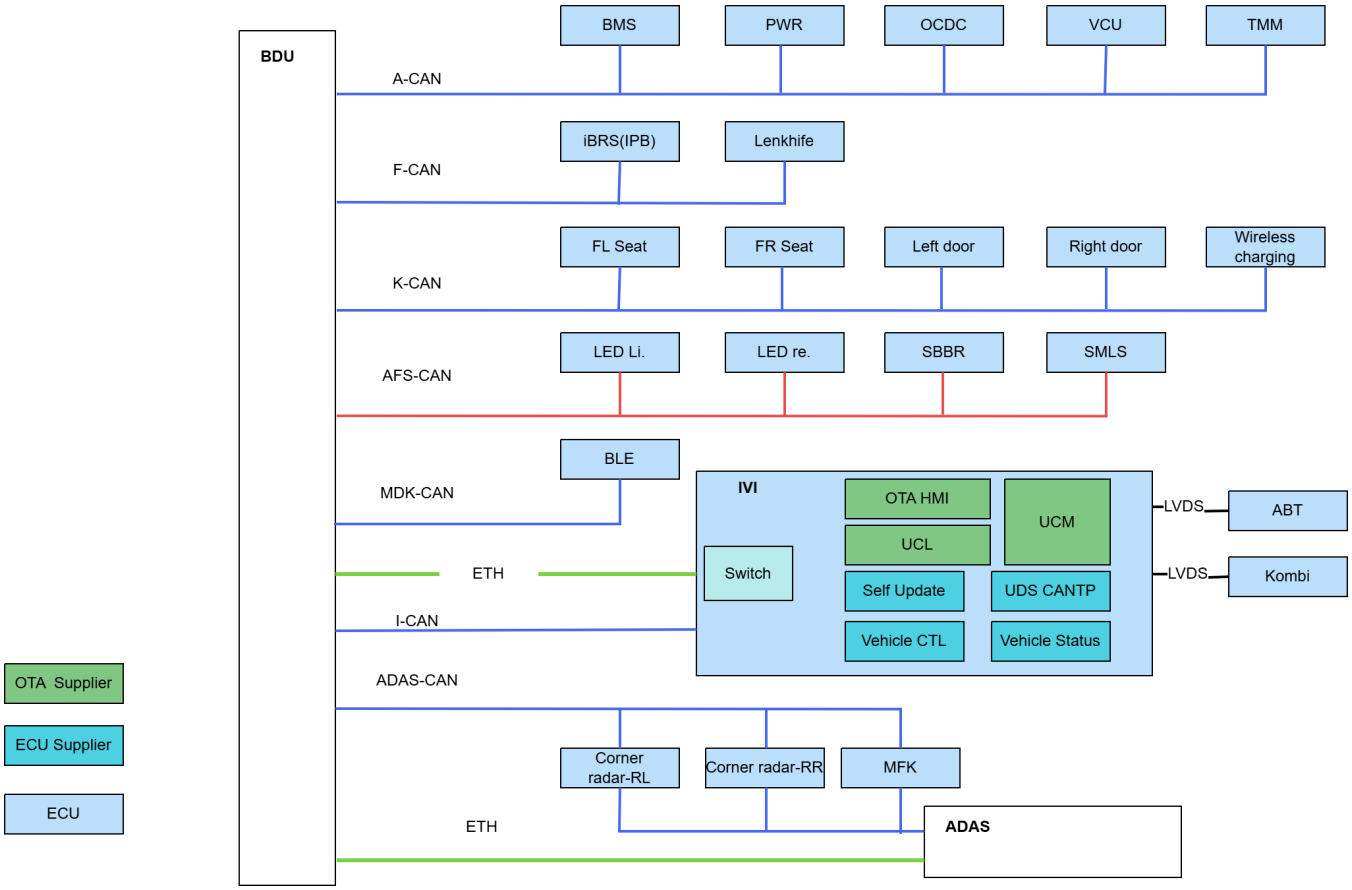


图6.1 整车网络拓扑图

6.2 设计思想(Design philodophy)

在本项目中，我们的设计思想是构建一个灵活、可扩展且高效的软件系统，以适应不断变化的业务需求和技术环境。为了实现这一目标，我们采纳了分层架构设计方法，相较于传统的三层架构（界面层、业务逻辑层和数据访问层），我们根据OTA软件的复用性考虑，将整个系统从顶层到底层划分为四个主要层次：

- **应用层(Application layer)**：应用层是系统的顶层，其主要职责是集成和协调FOTA应用的各项功能。在这一层，我们将整合和调度底层组件层所提供的服务，确保OTA应用程序能够无缝执行其核心功能，同时提供用户友好的交互界面。

- **组件层(Component layer)**：组件层致力于为OTA业务提供一系列标准化的组件实现。这些组件包括车辆状态控制、车辆诊断刷写、文件传输等关键服务。我们的目标是最大化组件的复用性，以便在不同的应用场景中快速部署和扩展。
- **基础软件层(Basic software layer)**：基础软件层为OTA平台提供了坚实的基础软件平台。在这一层，我们将实现事件处理、进程间通信和信息埋点等基础软件功能，为开发人员提供一个高效的系统级实现框架，从而加速上层应用场景的构建。
- **平台适配层(Foundation layer)**：平台适配层是系统架构的底层，它为OTA平台提供了可移植的操作系统接口，以及常用功能和工具的抽象实现。这一层确保了系统的跨平台兼容性，为上层架构提供了稳定和统一的运行环境。

分层架构的核心宗旨在于达成“高内聚低耦合”的设计理想，并促进组件的高效复用。在这一架构中，我们精心构建了四个层次，以确保系统的主要功能和业务逻辑得到恰当的处理和优化。

分层架构的原则是基于将系统分解为多个独立但相互协作的层次，以提高软件的可维护性、可扩展性和复用性。以下是分层架构设计的一些核心原则：

- **关注点分离**：每个层次应该专注于其特定的职责，避免跨层次的职责交叉。这有助于减少复杂性，使得每个层更容易理解和维护。
- **模块化**：系统应该被划分为多个独立的模块或组件，每个模块都有明确的接口。这样可以提高代码的复用性，便于替换和升级。
- **松耦合**：层次之间的依赖关系应该尽可能地弱化。这意味着一个层次的变化不应该对其他层次产生太大影响。
- **高内聚**：每一层都应该向上层提供清晰的抽象，隐藏下层的实现细节。这有助于上层开发者专注于业务逻辑，而不必关心底层的具体实现。
- **标准化接口**：层次之间的通信应该通过定义良好的接口进行。这有助于确保不同层次之间的兼容性和一致性。
- **可扩展性：**架构应该设计得足够灵活，以支持未来的扩展。这包括水平扩展（同层内增加更多的组件/模块）和垂直扩展（层与层之间允许扩充层）。
- **可测试性：**每一层都应该独立可测试，以便于进行单元测试和集成测试。这有助于提高软件质量，减少缺陷。
- **安全性：安全性应该在所有层次中得到考虑和实施，确保数据和系统的安全。**
- **性能优化**：在设计时考虑性能需求，确保系统在满足功能的同时，也能提供良好的性能。

除此之外，本架构中还约定：“**同一层中，不同模块间不允许直接接口调用，必须采用消息或者基础软件层中的数据库接口交互。不同层中，不推荐上层直接调用下层的接口，应尽量采用消息、基础软件层和平台适配层的接口交互**”。

6.3 软件架构图(Vehicle software architecture diagram)

结合当前整车架构和平台化设计思想，对OTA车端软件架构设计如下图：

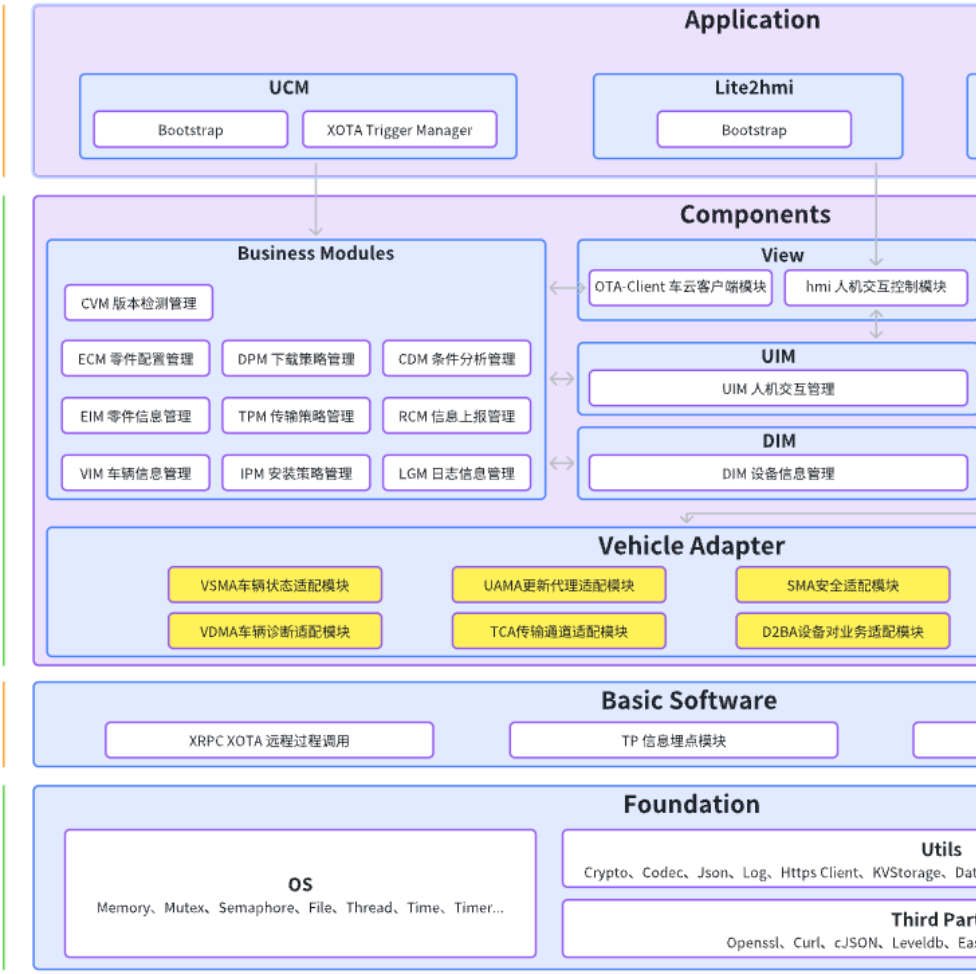


图6.3 OTA软件架构图

架构中组件功能描述如下:

名称	描述
UCM	OTA升级控制主应用，加载组件层中Business Modules，组织和驱动OTA车端行为，多以一个后台服务形式存在。
Lite2hmi	OTA人机交互应用，加载组件层中的View中hmi 人机交互控制模块，接受用户从车机端操作和向用户展示OTA过程画面，多以一个前端APP形式存在。
Lite (UCL)	OTA升级控制从模块，加载组件层中的Service Components组件，用来负责车内ECU升级，采集整车信息及OTA必要车控等车内本地相关操作，多以一个后台服务形式存在。
Business Modules	OTA业务模块抽象集合，为APP应用提供基础模块
View	为可交互模块提供功能服务
UIM	用户交互管理模块，用于管理人机交互
DIM	设备交互管理模块，用于管理需要被升级的零件
Service Components	服务组件，将OTA业务中用到的部分功能抽象成服务为已服务组件的形式为OTA业务提供功能支持
Vehicle Adapter	车辆OTA适配模块，主要适配OTA升级过程中的依赖，如车辆网络状态获取、车辆唯一标识获取，告知OTA任务开始/结束执行等。
Basic Software	基础软件层为OTA平台提供了坚实的基础软件平台。在这一层，我们将实现事件处理、进程间通信和信息埋点等基础软件功能，为开发人员提供一个高效的系统级实现框架，从而加速上层应用场景的构建。
Foundation	平台适配层是系统架构的底层，它为OTA平台提供了可移植的操作系统接口，以及常用功能和工具的抽象实现。这一层确保了系统的跨平台兼容性，为上层架构提供了稳定和统一的运行环境。

系统中UCM、Lite、Lite2hmi及OTA Server & CDN 之间的服务关系及通信协议如下:

服务消费者	服务提供者	说明	通信协议	协议接口
UCM	OTA Server	UCM向OTA管理平台发起车辆注册、车辆配置信息获取、版本检测、升级包下载、信息上报等	https (双向认证)	《公版5.0OTA平台接口设计书.docx》
UCM	Lite	UCM向Lite请求特定零部件的信息采集、升级执行等	ARPC (TCP、SSL TCP)	《VOTA Protocol详细设计文档_v0.1.5_20220424.docx》、8.1 接口说明
Lite2hmi	UCM	Lite2hmi向UCM转发人机界面的指令，如触发下载、安装等	ARPC (TCP、SSL TCP)	《VOTA Protocol详细设计文档_v0.1.5_20220424.docx》、8.2 接口说明
Lite	CDN	Lite向CDN服务器下载升级包	https (单项认证)	《公版5.0OTA平台接口设计书.docx》

6.4 OTA在IVI中的部署图(Software deployment diagramdiagram)

基于OTA软件架构设计，在IVI系统中的部署情况见下图。

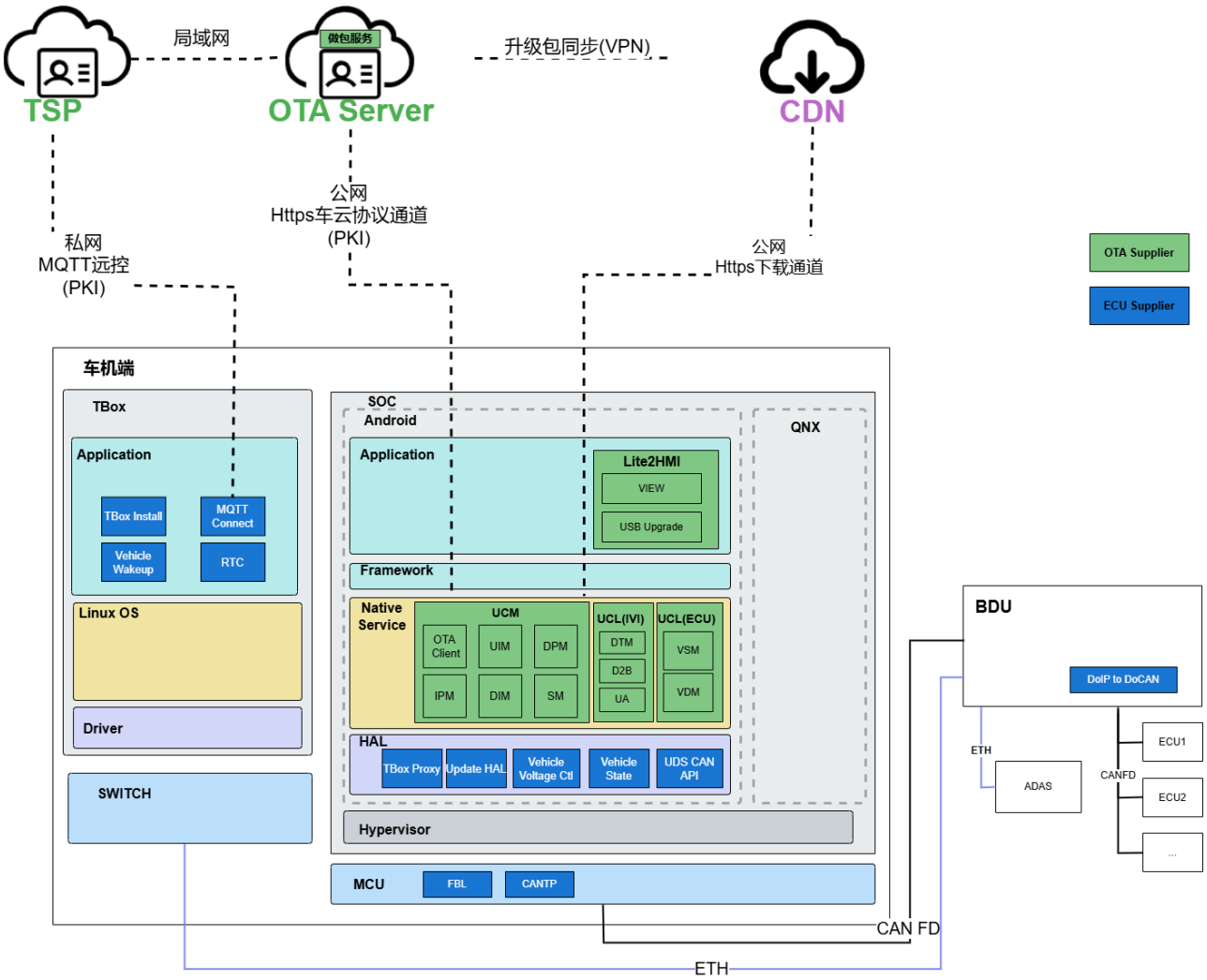


图6.3 OTA软件架构图

由于智能件升级方式与传统ECU诊断刷写的升级方式区别较大，这里通过部署2个UCL来分别管理车机自升级和通过传统诊断方式升级的ECU。

6.5 模块与组件设计(Modules and components design)

6.5.1 UCM升级控制主模块 (Update Control Master)

6.5.1.1 模块概述

UCM作为FOTA（Firmware Over-The-Air）的控制主模块，用来负责OTA升级主流程控制管理。UCM的设计目的是为了确保车辆软件更新过程的安全性、可靠性和有效性，特别是在涉及多个ECUs和复杂的更新策略时。主要负责OTA版本检测、升级包的下载与安装等关键步骤，以确保升级过程的顺利进行。

在OTA升级的不同阶段中，UCM会向Lite端请求对应服务，以确保ECU的顺利更新。如获取车辆的ECU信息、查询当前的车辆状态，以及执行对车辆状态的控制操作。此外，UCM还负责发起升级包的下载请求，并在适当时机触发ECU的升级过程。

在整个升级过程中，UCM不仅确保了升级操作的精确执行，还通过实时向Lite2hmi推送升级进度/结果信息，使得HMI端能够及时展示车辆的升级状态。同时，UCM还能够迅速响应来自Lite2hmi的指令，如接收下载触发信号和预约安装时间等任务。这种双向互动机制，不仅提高了升级过程的灵活性，也使得车主能够更加便捷地管理车辆的升级计划。

6.5.1.2 静态视图

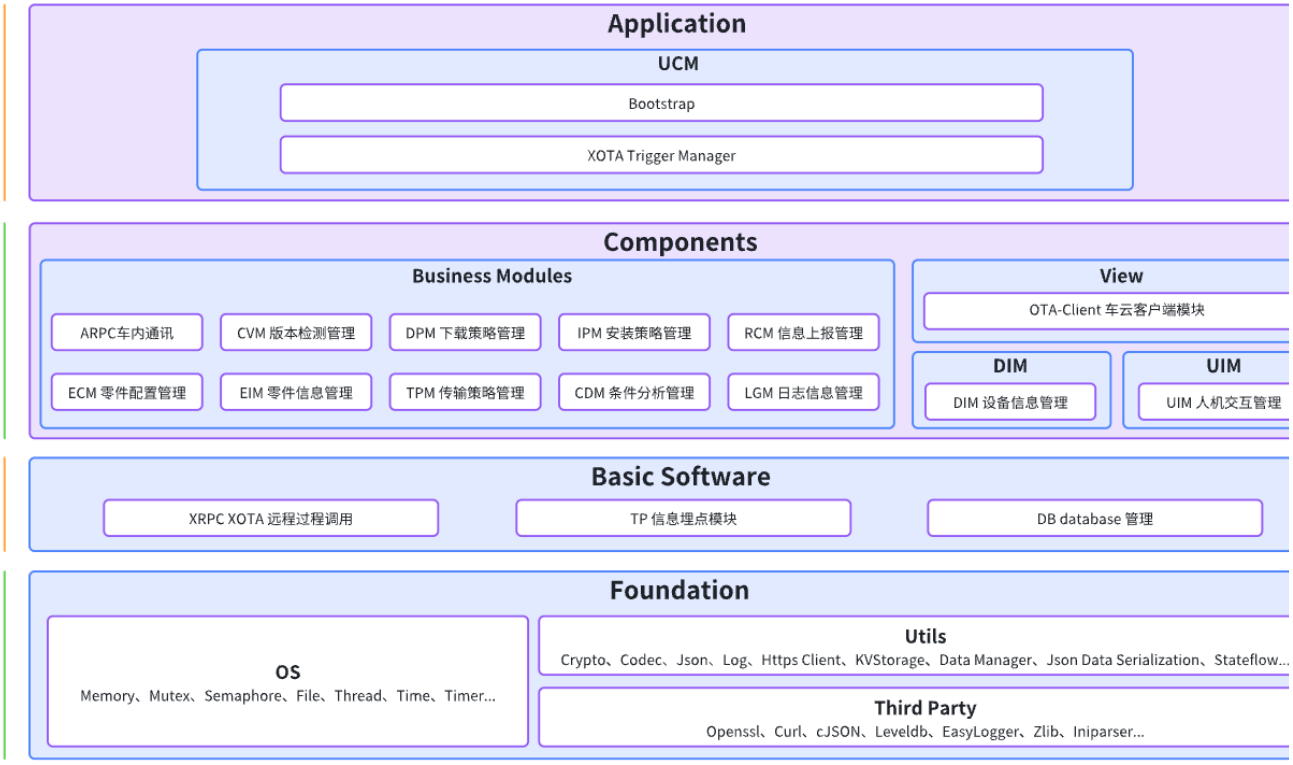


图6.5.1.2 UCM组件分层图

UCM中各个模块功能描述如下:

组件名称	组件功能
Bootup (启动)	负责整车上电后，启动UCM应用进程，并进行UCM应用的系统初始化、业务初始化及OTA状态的恢复等，确保软件更新流程的连续性和高效性。
XOTA TASK Manager (任务管理)	负责FOTA升级流程中的状态机管理，根据不同的状态和事件进行相应的调度。这包括监控升级过程中各个环节的完成情况，根据当前状态和事件触发相应的动作，以确保升级过程按照预期进行。
零件配置管理	负责管理向OTA管理平台获取ECU信息及升级相关配置信息，为版本检测做准备。
零件信息管理	基于零件配置管理模块从OTA管理平台获取的ECUs配置信息，来实现对车辆各个零件信息的采集。
车辆信息管理	负责在OTA过程中管理车辆的状态。例如车辆状态读取、车辆状态控制等。
版本检测管理	基于零件信息管理采集的零件信息，管理向OTA管理平台发送版本检测处理。
升级包下载管理	负责OTA过程中管理和控制待升级ECU升级包的下载策略处理流程。
升级包传输管理	负责协调主控设备（UCM）与特定ECU之间的升级文件传输。确保UCM设备下载的升级文件能够有效地传输到目标ECU，从而让这些ECU具备OTA升级基础。
升级包安装管理	负责协调整车ECU的安装流程，按照按照安装策略信息进行操作，并监控整个安装过程，以确保ECUs安装的正确性和可靠性。
条件分析管理	负责将OTA过程中产生的各种信息上报至OTA管理平台，以便OTA管理平台可以监控车辆升级状态。

信息上报管理	负责评估车辆当前状态是否符合执行特定OTA操作所需的前置条件。
日志信息管理	用于OTA系统运行中的日志收集、存储、上传等管理。
车云客户端管理	作为OTA系统与OTA管理平台之间的通信桥梁，负责发送和接收升级指令及日志上传等功能。
人机交互管理	负责OTA升级过程中与Lite2hmi的交互管理，进而实现用户界面控制（触发检测、下载、安装及相应进度和结果的展示）。
设备信息管理	负责在OTA升级过程中与Lite通信，进而实现对待升级的ECUs的控制管理。

6.5.1.3 业务时序图

6.5.1.4 资源消耗目标

ROM空间占用大小	RAM空间占用大小	CPU占用率	耗时
20MB	25MB	10%	N/A

6.5.1.5 组件接口

详见车云交互接口《公版5.0 OTA平台接口设计书.docx》。

6.5.2 Lite升级控制从模块(Update Control Lite)

6.5.2.1 模块概述

Lite作为FOTA的控制从模块，主要集成在被升级ECUs上。其不仅为UCM的升级流程提供关键支持，还赋予ECU远程升级的能力。根据OTA的功能需求，Lite能够集成多样化的服务，以满足不同应用场景的功能需求。这些服务按照其功能特点可划分为以下七个类别：

a. **DTM（Delegate Transfer Manager）委托传输服务**：负责从指定服务器下载和上传指定文件，确保数据传输的准确性和效率。
b. **SM（Security Manager）安全服务**：提供必要的安全措施，包括数据加密、身份验证和完整性校验，以保护Lite设备免受未经授权的访问和篡改。
c. **UAM（Update Agent Manager）更新代理服务**：在宿主设备是智能件时，负责升级刷写操作和部分零件信息的采集工作。
d. **VDM（Vehicle Diagnostic Manager）车辆诊断服务**：适用于非智能件的刷写及车辆诊断，以及执行部分零件信息的采集工作。
e. **VSM（Vehicle Status Manager）车辆状态服务**：在OTA过程中读取、设置和监控车辆状态，如车速、档位、电池电量等，以及设置整车的高压状态。
f. **D2B（Device to Business）设备对业务服务**：处理升级过程中依赖的OEM相关接口，如预约升级、进入OTA模式等，确保与车辆制造商的业务流程对接。
g. **MISC（Miscellaneous）杂项服务**：负责收集Lite相关日志信息，设置Lite进程日志输出等级等进程相关属性，以便于调试和监控Lite app的运行状态。

这些服务共同构成了Lite的功能框架，使其能够支持复杂的OTA升级流程，包括数据传输、安全保障、设备升级、车辆状态监控以及与OEM业务流程的集成。通过这些服务的集成，Lite app能够为ECU提供全面的OTA升级能力，实现车辆软件的远程更新和功能增强。

6.5.2.2 静态视图

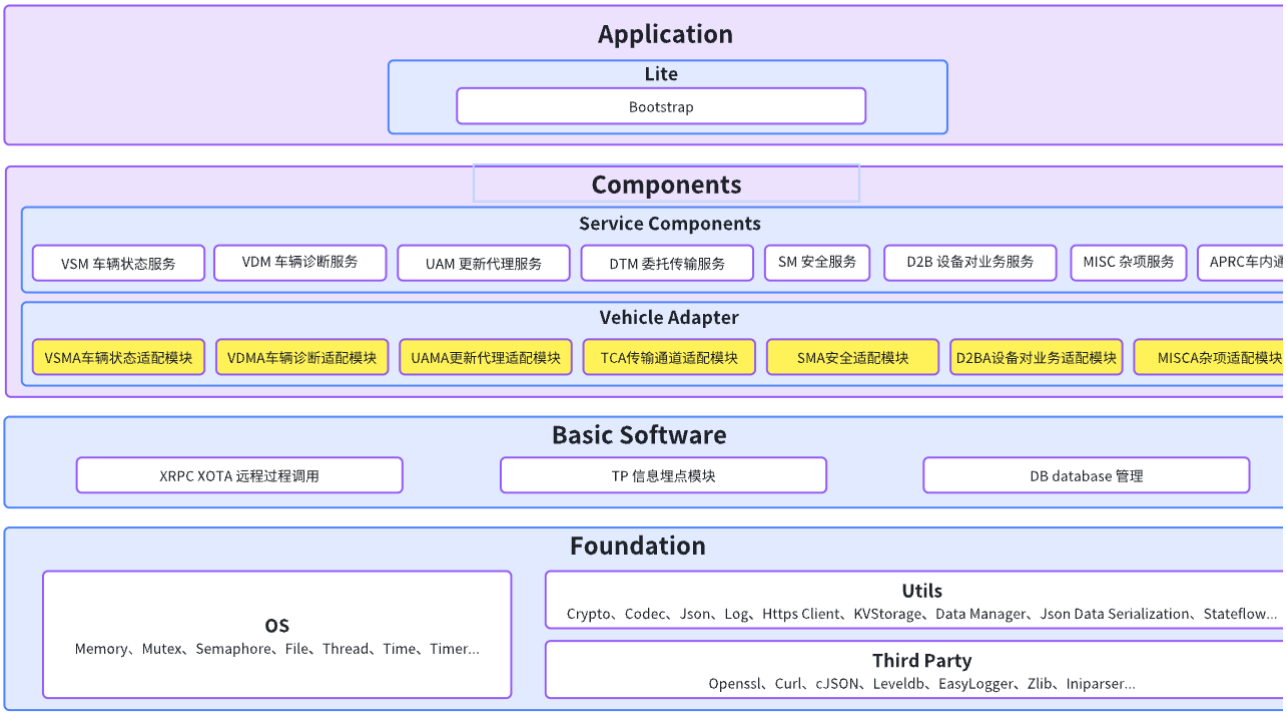


图6.5.2.2 Lite组件分层图

Lite中各个模块功能描述如下:

组件名称	组件功能
Bootup (启动)	负责整车上电后，启动Lite应用进程，并进行Lite应用的系统初始化、服务初始化及通信连接建立等。
车辆状态服务	用于OTA升级过程中的车辆状态管理，如车辆状态的查询、设置及监控。
车辆诊断服务	用于通过UDS协议对车辆诊断及ECU刷写。
更新代理服务	用于支持通过非诊断协议进行ECU的信息采集机升级。
委托传输服务	用于车辆升级过程中文件的传输功能，包括车内文件下载和上传，车云文件下载。
安全服务	为OTA升级过程提供安全保障，确保业务组件在各执行阶段的安全性和数据校验。
设备对业务服务	OTA升级过程中依赖的宿主ECU提供的一些功能，例如获取车辆唯一标识、设置预约时间、获取车辆网络类型等。
杂项服务	车端OTA系统中一些公共或者杂项功能集。
车辆状态适配模块	主要为车辆状态服务的具体实现。
车辆诊断适配模块	主要为车辆诊断服务的具体实现。
更新代理适配模块	主要为更新代理服务的具体实现。
传输通道适配模块	主要车端车云通信通道的封装实现（消息、文件上传/下载通道）。
安全适配模块	主要为安全服务的具体实现。
设备对业务适配模块	主要为设备对业务服务的具体实现。
杂项适配模块	主要为杂项服务的具体实现。

6.5.2.3 业务时序图

6.5.2.4 资源消耗目标

ROM空间占用大小	RAM空间占用大小	CPU占用率	耗时
20MB	25MB	基线3%（峰值60%）	N/A

6.5.2.5 组件接口

详见《公版VUS1.6项目_UCM与Lite交互接口设计书.docx》。

6.5.3 Lite2hmi升级控制从模块-人机交互SDK (Update Control Lite to HMI)

6.5.3.1 模块概述

Lite2hmi 是一种特殊的 Lite形态，是一款专为人机交互（HMI）集成设计的轻量级应用框架，它为开发者提供了一整套用于实现车载信息娱乐系统（HMI）与OTA升级过程的集成方案。Lite2hmi通过提供 OTA升级流程的回调通知机制，使得宿主HMI侧的应用能够与车端的OTA功能无缝对接，确保了OTA与用户交互体验的流畅性，同时增强了车载系统的用户满意度和可靠性。

6.5.3.2 静态视图

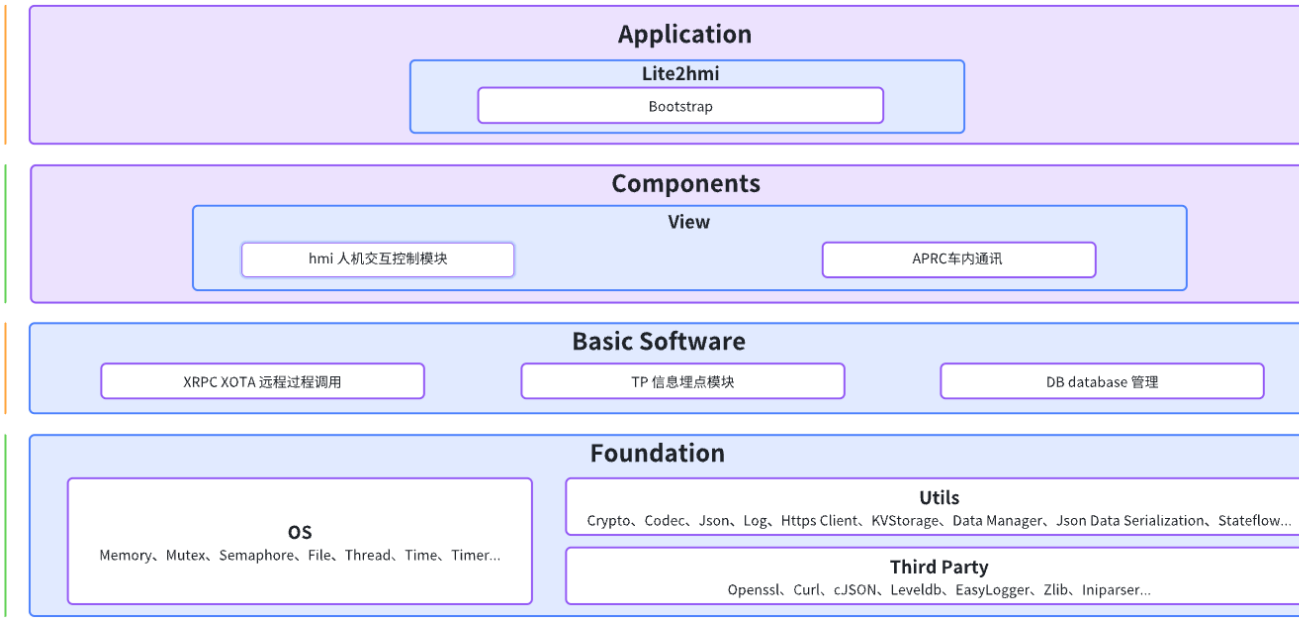


图6.5.3.2 Lite2hmi组件分层图

Lite2hmi中各个模块功能描述如下:

组件名称	组件功能
Initialization API	一组Lite2hmi与UCM通信链路建立的初始化接口。用于简化Lite2hmi与UCM的通信。
人机交互控制模块	一组Lite2hmi与UCM交互的接口，主要包含两类： l 用户主动触发：用户触发检测、下载和安装等。 l HMI展示：检测结果、下载进度/结果、安装进度/结果等。

6.5.3.3 业务时序图

6.5.3.4 资源消耗目标

ROM空间占用大小	RAM空间占用大小	CPU占用率	耗时
15MB	20MB	10%	N/A

6.5.3.5 组件接口

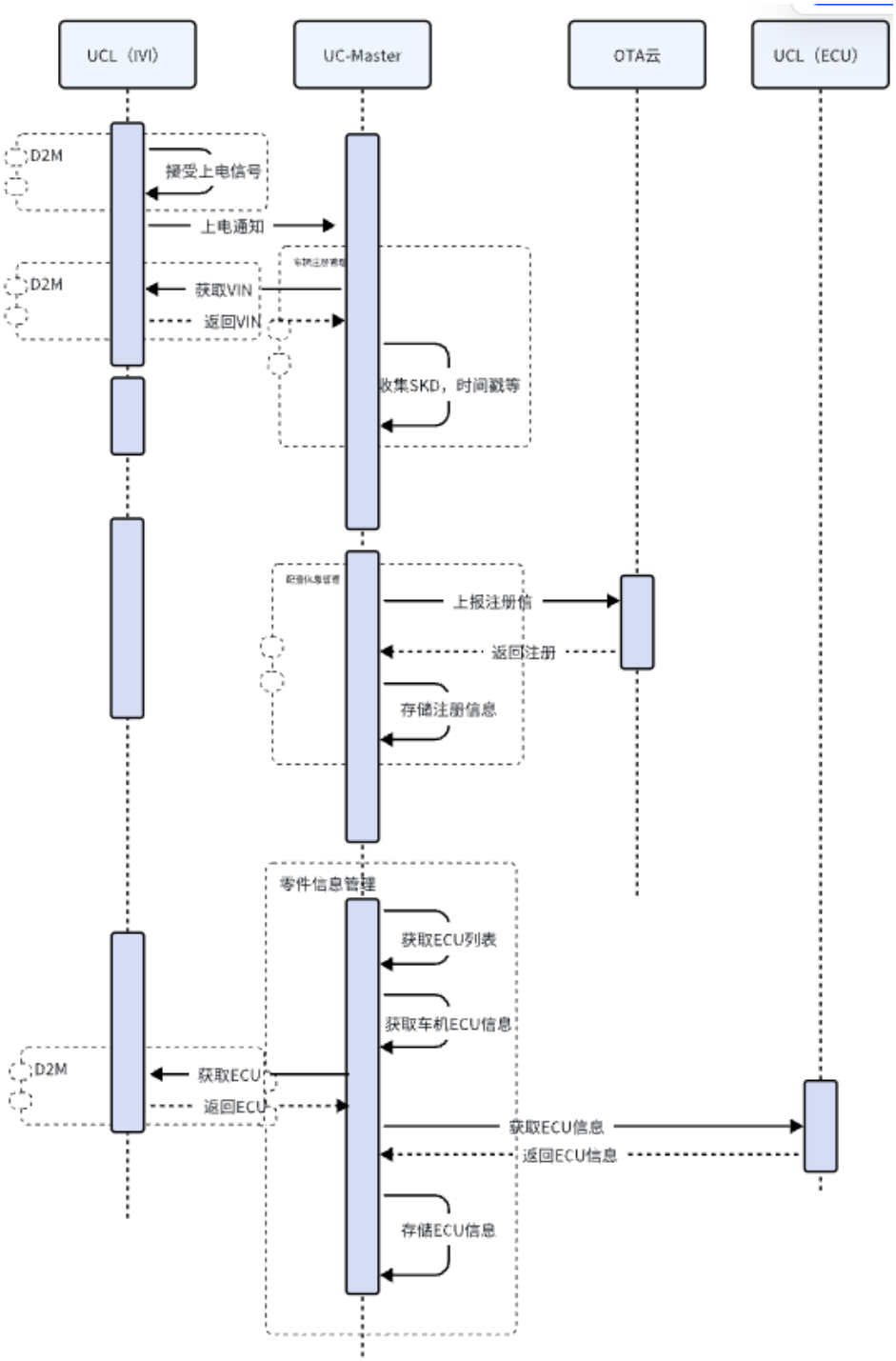
详见人机交互接口《公版VUS1.6项目_UCM与Lite2hmi交互接口设计书.docx》。

7 OTA主要业务流程设计(Main process design)

7.1 获取配置和资产上报流程

通过获取车辆配置信息，可以实现两个主要目标：首先，用户可以利用OTA管理平台对车辆的OTA客户端软件相关输出进行控制，例如调整车端OTA软件日志文件的上报周期和输出等级，以优化车辆的性能和用户体验。其次，通过提前获取待升级的ECU列表，车辆可以预先采集对应的ECU DID值，从而在进行版本检测时大幅缩短用户体验时间，提升整体的升级效率。

流程图：

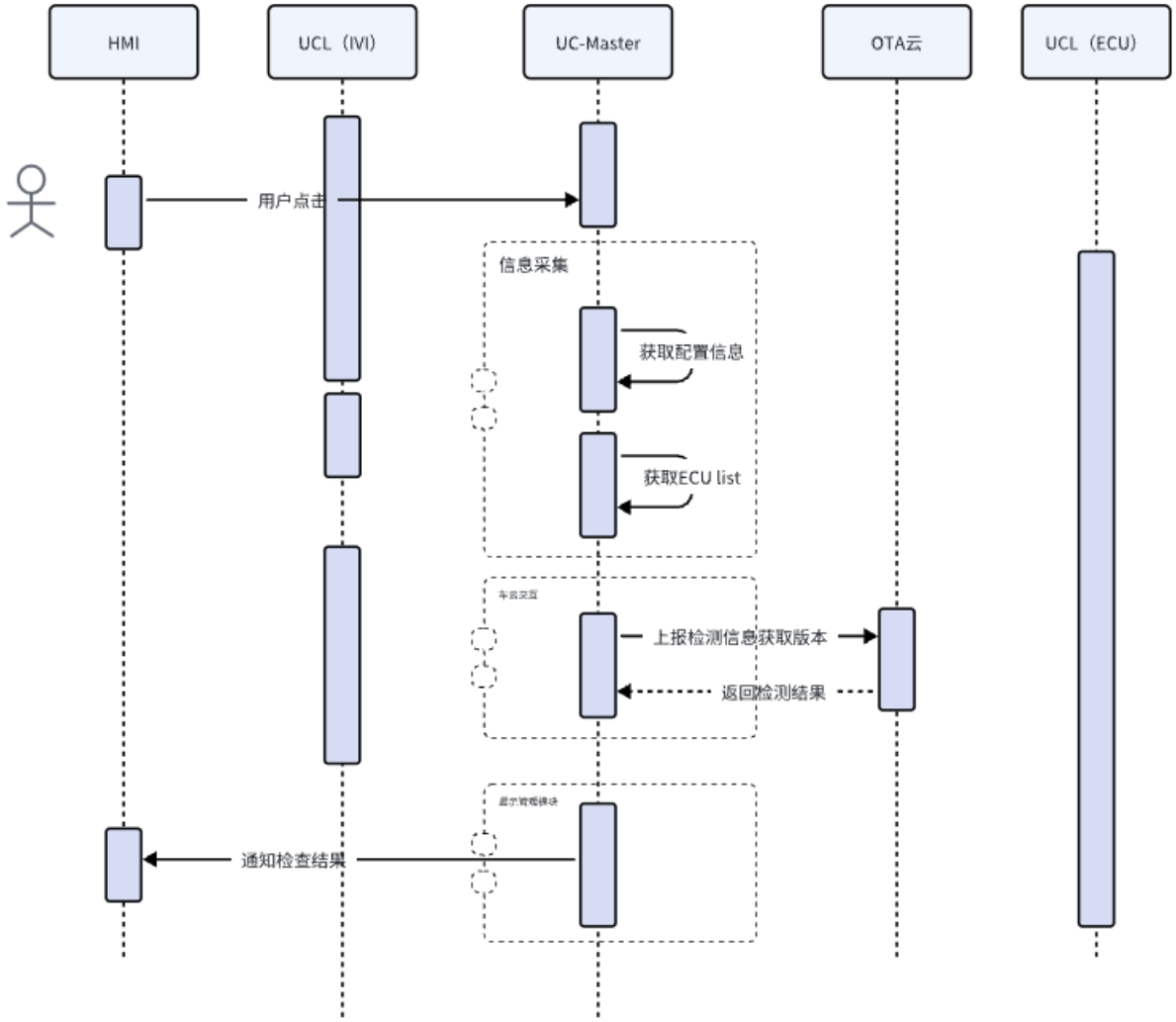


7.2 手动检测

版本检测流程是指汽车端上报ECU详细信息，OTA管理平台据此判断ECU版本有是否需要更新的过程。版本检测流程由UCM发起，OTA管理平台响应。

版本检测管理模块主要负责车端OTA版本检测的整体流程，包括获取零件配置、读取待升级零件信息、向OTA管理平台发送版本检测请求、处理新版本策略如下载HMI策略文件等。

流程图：

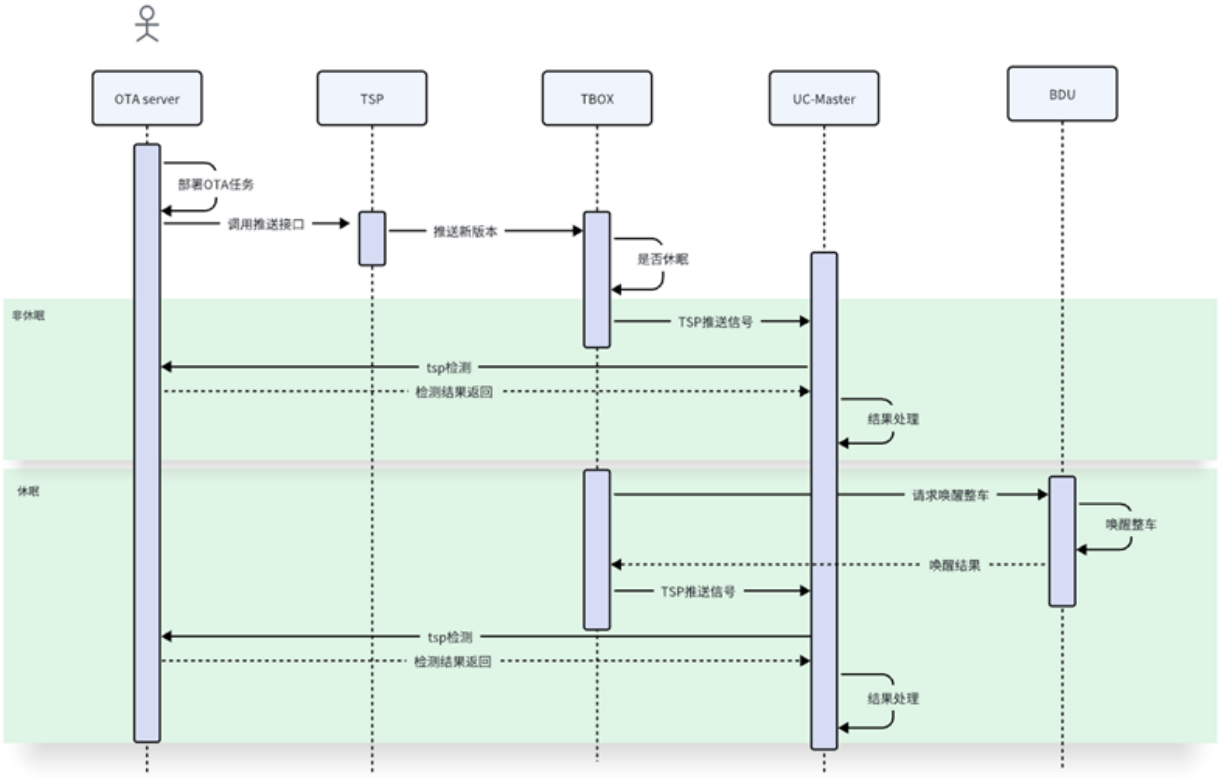


7.3 FOTA常规升级流程

在OTA场景中可以通过TSP功能完成OTA的的远程控制远程业务主要包括一下业务：新版本推送，立即安装，修改预约升级时间等。

7.3.1 新版本推送

流程图：

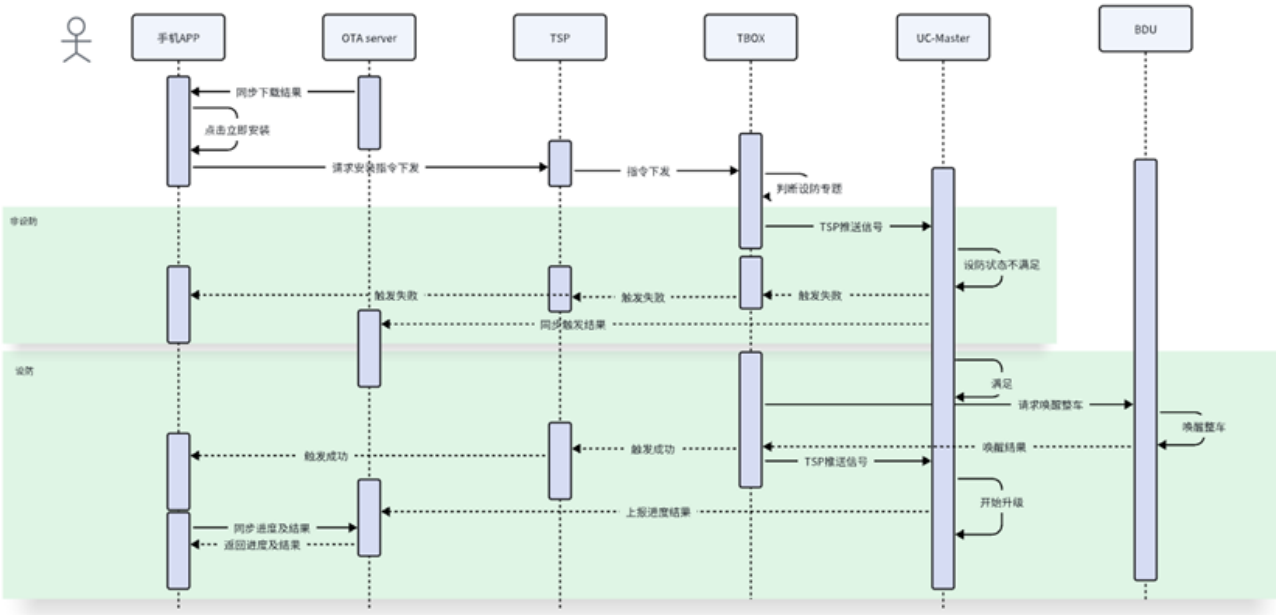


场景流程描述：

- OTA云端部署任务
- 部署完成后请求TSP下发推送指令到tbox，tbox判断当前处理状态是否为休眠状态
  - 非休眠：此时整车设备处于运行状态，OTA系统在运行中，可接收TSP推送信号触发检测流程获取任务信息，该状态下用户正在用车，只提示不弹框，上电信号或者下电信号时弹框提示用户。
  - 休眠：此时tbox和BDU处于休眠态，其他零件处于未运行状态。首先要唤醒整车，唤醒成功后OTA系统处于运行中。可接收TSP推送信号触发检测流程获取任务信息，根据任务类型进行下一步操作。

7.3.2 立即安装

流程图：

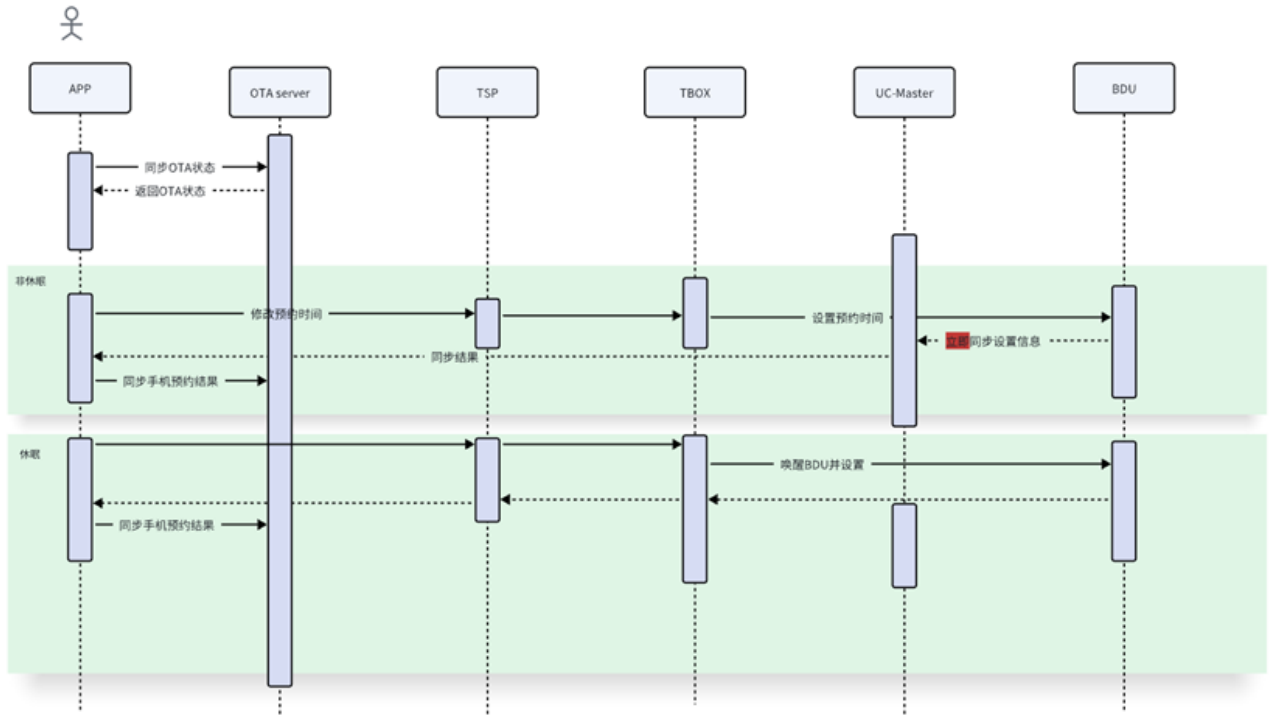


场景流程描述：

- 手机APP运行时向OTA云获取当前车辆状态如果是下载完成可以触发立即安装
- 手机APP调用tsp推送立即安装指令到tbox判断设防状态
    1. 非设防状态：当前条件不满足远程控制升级返回触发失败
    2. 设防状态：满足远程升级必要条件，tbox请求BDU唤醒整车，tbox接收到唤醒结果，同步结果到tsp，如果结果为成功，tbox发送信号给UM-Master进行tsp推送升级，手机APP向OTA云同步升级进度及结果；

7.3.3 远程修改预约升级时间

流程图：



场景流程描述：

- 手机APP运行时向OTA云获取当前车辆状态如果是下载完成或者已经设置预约，手机app和执行修改预约时间推送
- 手机APP调用tsp推送修改预约时间指令
    1. 休眠：tbox唤醒BDU，设置预约时间返回设置结果，通过TSP同步到app用于显示，同时将结果通过TSP同步到OTA云
    2. 非休眠：tbox设置预约时间BDU返回设置结果，通过TSP同步到app用于显示，同时将结果通过TSP同步到OTA云

8 OTA安全需求

8.1 车云链路安全

8.1.1 车辆与OTA管理平台的通信

车辆与OTA管理平台之间的消息通信（包括车辆注册、获取车辆配置信息、车辆版本检测等），采用TLS双向认证来确保通信安全。

为了保障消息通信交互报文的安全性，每个接口的请求和响应报文都必须进行全字段的验签即加密操作。具体的验签和加密算法详见《公版5.0 OTA平台接口设计书.docx》。

车辆在上传升级日志至OTA管理平台时，采用TLS单向认证机制，即车辆端仅对OTA管理平台的合法性进行认证。

8.1.2 车辆与CDN的通信

车辆在下载升级包时，使用TLS单向认证方式，即车辆端仅对服务器的合法性进行认证。之所以不采用双向认证，是因为CDN作为一个为多家提供数据分发服务的供应商，其相关资源是面向所有服务客户的。如果要求CDN对每个客户的设备都进行认证，将导致管理上的大量工作和开发负担，从而增加维护的复杂性。

这些认证方式及其相关证书均由配置文件动态加载，以适应不同的安全需求。

---具体内容参见《OTA安全方案》

8.2 车内链路安全

8.2.1 车内进程间消息通信

本架构中UCM与UCL、UCM与lite2HMI之间的消息通信，通信协议采用的是ARPC协议，这是OTA车端组件为实现车辆OTA功能场景定义的私有协议，支持域间通信。

ARPC协议认证支持如下几种方式：

- 支持无认证。
- TLS单向认证。
- TLS双向认证。

8.3 升级包安全

升级包安全方案，主要从两个角度，对升级包的安全进行保证

- 升级包合法性认证（签名）：为了保证升级包的合法性，防止升级包被人为篡改为不明来路的升级包，车辆升级失败所导致的安全隐患，所以在车辆下载完升级包，触发升级前，对升级包合法性进行校验。当前我们采用的主要认证方式：对升级包进行数据签名的方式实现。
- 升级包加密：主要是为了保证，升级包地址被破解获取，被非法下载导致的文件泄露，加密后的升级包，无法被正常使用。当前升级包加密的方案，是采用对称加密的方式，对升级包进行加密，车端下载完成升级包，对升级包解密后使用。

主要过程涉及升级包签名、升级包加解密、具体参见参见《OTA标准包方案》以及《OTA安全方案》。

9 软件其他考虑(Software other considerations)

9.1 异常处理(Exception handling)

9.1.1 软件错误监控机制，信号捕捉机制

在程序运行过程中，如果遇到异常崩溃，我们的系统已经设计了一套信号捕获机制，能够识别并响应SIGABRT（程序异常终止）、SIGSEGV（段错误）和SIGPIPE（管道破裂）等关键信号。一旦程序因这些信号而异常退出，系统会自动执行堆栈回溯操作，并将当前进程的堆栈信息输出到日志文件中。开发人员可以通过分析这些日志信息，精确定位程序异常发生的位置，从而迅速有效地诊断和修复问题。

这种设计确保了在程序出现不可预见的故障时，能够提供详细的堆栈信息，帮助开发人员快速定位问题根源，加速了从开发到部署的流程，并提升了软件的整体稳定性和可靠性。

9.1.2 数据库异常恢复

多进程同时操作同一个数据库，导致数据库异常。后续进程无法打开数据库，导致程序不断崩溃。处理方案当程序拉起时,需要打开本地数据库，当打开数据库失败时，就需要对数据库进行恢复,当恢复失败后，就需要进行对数据库数据的清除，重新打开数据库保证程序可以正常运行。

9.1.3 VIN变更的异常处理

在UCM程序的启动过程中，会执行一个关键步骤：获取车辆的VIN并向平台发起注册请求。这一过程确保了车辆信息在平台上的实时性和有效性。每次注册时，程序首先会验证VIN的长度，确保其为标准的17位。如果VIN长度不符合要求，程序会立即报告错误。一旦VIN验证通过，程序便会发起注册请求，并将注册信息妥善保存。这样，即使VIN发生变更，只要主控程序重新启动并执行注册流程，就能确保车辆信息在平台上的准确性和最新状态。这种机制确保了车辆数据的持续更新，为车辆管理和监控提供了可靠的基础。为车辆管理和监控提供了可靠的基础。

10 软件约束(Software constraints)

10.1 代码规范(Code specification)

代码规范严格按照

- 《OEM编码规范》。
- Helix QAC MISRA-2012(c++)/2016(c) 规则。

11 参照文件(Reference)

序号	文件名	版本号	存储位置
1	《J01-OTA 车机端PRD》		06. J01-OTA 车机端PRD - Jetta BEV - Confluence (vc.in)
2			
3			
4			
5			
6			