

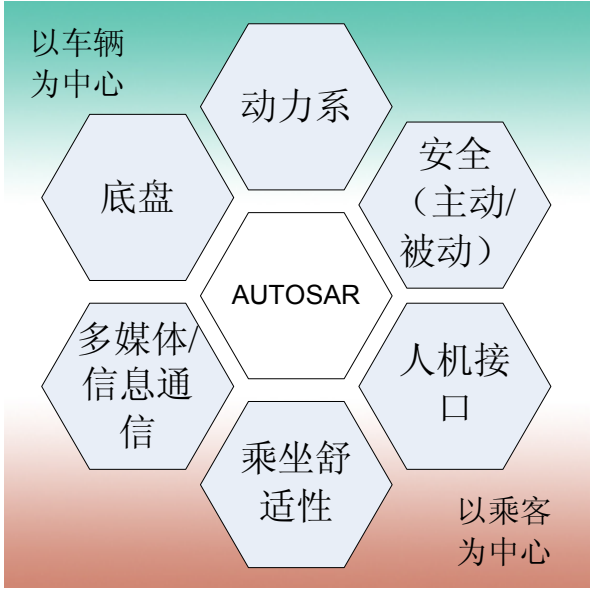
AUTOSAR 技术分析报告

1. AUTOSAR 简介

汽车电子领域的软件主要属于嵌入式软件。因此，其发展阶段类似于其他嵌入式系统的软件发展。由于受限于嵌入式硬件本身资源的匮乏，各种硬件产品的种类繁多和各自差异，以及整体嵌入式系统软件的逐步发展，起初的软件设计开发主要是封闭式的。这样有助于开发针对于特定硬件体，充分优化利用资源而特定设计的软件系统。这样的软件系统，是针对于特定硬件和特定应用而设计，其对于硬件资源的充分应用，以及软件本身的执行效率无疑是非常高。

然而，随着硬件本身的逐步发展，其可用资源已经十分充分。另一方面，汽车电子领域应用需求也日趋复杂，软件本身也变得越来越复杂。因此，无论汽车厂还是部件商都感到软件的标准化问题。软件的可管理性，可重复使用性，可裁减性，以及质量保证等等问题被提上了议程。AUTOSAR 的提出正是基于以上一些软件发展的要求，由几大主要汽车厂商以及部件提供商联合提出的，其中包括 BWM, DaimlerChrysler, Ford Motor, PSA Peugeot, Toyota Motor, Volkswagen AG, Bosch, Continetal, Siemens VDO 等。

AUTOSAR 是针对特定的汽车电子这一领域，提出的一套开放式软件结构。其主体思想是使得软件设计开发更易于管理，软件系统更易于移植、裁剪，以及更好的维护性和质量保证。AUTOSAR 组织所提出的目标以及它所关注的功能领域在下表中列出：

项目目标	功能领域
<ul style="list-style-type: none">• 解决汽车的可用性和安全性需求• 保持汽车电子系统一定的冗余• 可以移植到不同汽车的不同平台上• 实现标准的基本系统功能作为汽车供应商的标准软件模块• 通过网络共享软件功能• 集成多个开发商提供的软件模块• 在产品生命周期内更好的进行软件维护• 更充分的利用“货架产品”• 在车辆整个生命周期中进行软件更新与升级	 <p>The diagram illustrates the functional areas of AUTOSAR, centered around a white hexagon labeled 'AUTOSAR'. Surrounding this center are six blue hexagons, each representing a functional domain: '动力系' (Powertrain) at the top, '底盘' (Chassis) on the left, '多媒体/信息通信' (Multimedia/Information Communication) at the bottom-left, '乘坐舒适性' (Passenger Comfort) at the bottom, '人机接口' (Human-Machine Interface) at the bottom-right, and '安全 (主动/被动)' (Safety (Active/Passive)) on the right. The entire diagram is set against a background that transitions from light green at the top to light red at the bottom. The top green area is labeled '以车辆为中心' (Vehicle-centric), and the bottom red area is labeled '以乘客为中心' (Passenger-centric).</p>

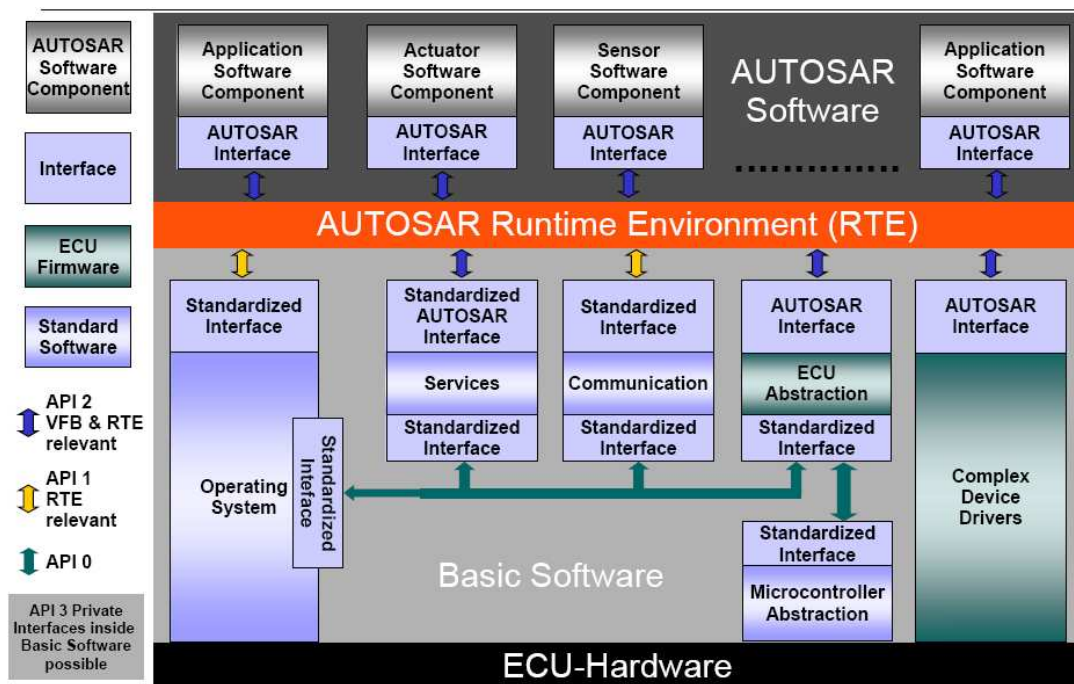
为了实现上述的项目目标，针对在汽车电子行业中面临的一些挑战，AUTOSAR 所采用的解决方案及其好处可以概述如下：

挑战	解决方法	好处
不成熟的过程，因为 ad-hoc 模式/缺少对功能需要的追踪能力。 缺少兼容的工具（供应商、OEM）	标准化的规范交换格式	对规范的改进（格式、内容） 提供无缝的工具链。
浪费在实现和优化组件上的努力，而顾客并不承认这些努力的价值。	基础软件核	软件质量的加强。 将工作集中在有价值的功能上。
微控制器模型缺乏可用性，很难适应现有软件。 （由新功能引起的）微控制器性能的扩展需求所导致的升级需要（如重新设计）。	微控制器抽象	微控制器能在不需要改变更高软件层的情况下调换。
重定位 ECU 之间的功能时需要做大量的工作。 功能重用时也需要做大量的工作。	运行时环境 (RTE)	功能封装导致的通信技术的独立性。 通过标准化机制，使得通信更加简单。 使功能分区和功能重定位变得可能。
非竞争性功能必须适应 OEM 的特定环境。 因为需要从其它组件供应接口需要很多功夫，所以哪怕是很微小的革新，也需要做很多工作。 基础软件和模型生成的代码间缺少清晰的接口。	接口标准化	减少/避免 OEM 和供应商之间的接口。 通过使用通用接口目录，使独立于软件功能的硬件实现所耗费的工作量。 简化模型驱动的开发，允许使用标准化的 AUTOSAR 代码生成工具。 OEM 间的模型的可重用性。 不同供应商之间模块的可交换性。

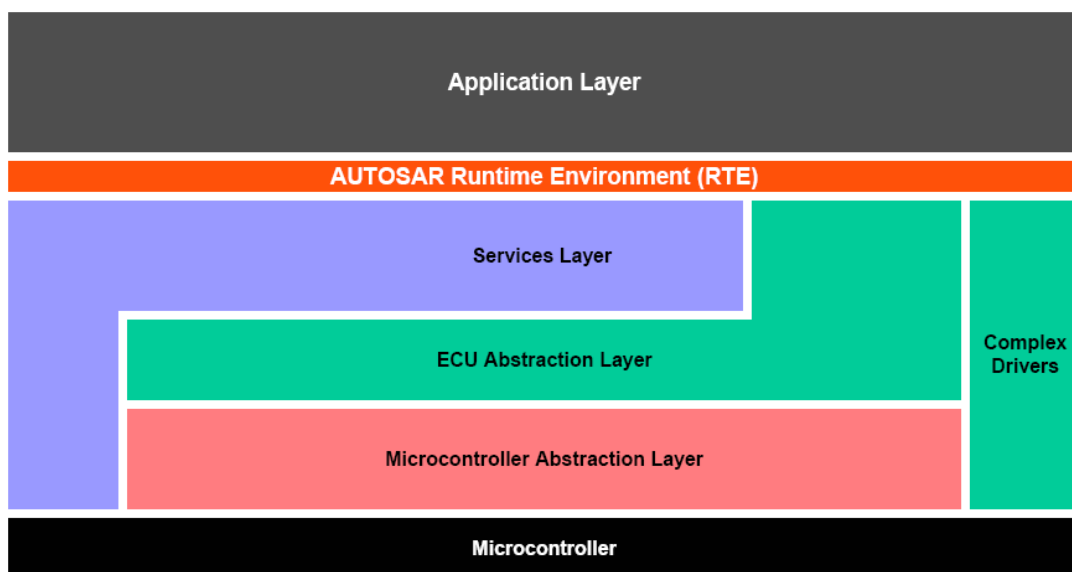
2. AUTOSAR 软件结构

2.1 AUTOSAR 软件的组成与分层

AUTOSAR 的软件组件可以用下图来表示：



对于上图所示的一些组件，可以根据功能及相互关系对其进行分层，如下图所示：



- **微控制器抽象层**

这一层是基础软件中的最低一层。它包含驱动，这些驱动是软件模块，用来对 μC 内部设备和映射了 μC 外部设备的内存进行访问。

- **ECU 抽象层**

这一层与微控制器抽象层进行对接。它也包含了外部设备的驱动。它为访问外设提供了 API，不管这些外设的位置(μC 内部或外部)，也不管它们与 μC 的连接(端口引脚，接口类型)。

- **服务层**

这层是基础软件中的最高层，而且它与应用软件之间有关联：当对 I/O 信号的访问包含 ECU 抽象层中时，服务层提供：

- 操作系统功能
- 车辆网络通信及管理服务
- 存储管理（NVRAM 管理）
- 诊断服务（包括 UDS 通信及错误内存）
- ECU 状态管理

2.2 RTE

运行时环境 RTE 是 AUTOSAR ECU 体系结构的核心组成部分。RTE 是 AUTOSAR 虚拟功能总线（Virtual Function Bus, VFB）的接口（针对某个特定 ECU）的实现，因此，它为应用程序软件组件之间的通信提供了基本的服务，同时也便于访问包含 OS 的基本软件组件。

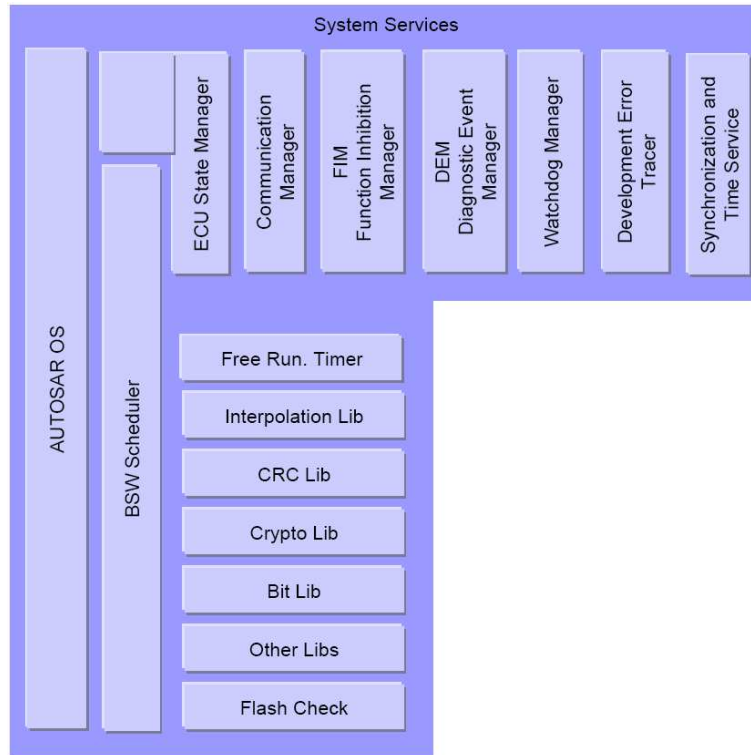
应用程序软件组件包含独立于 CPU 和所处位置的系统软件。这就意味着，为了满足系统设计者所做的一些限制，应用程序组件能够在系统配置期间被映射到任何有效的 ECU 上。RTE 负责确保这些组件能够通信。

RTE 和 OS, AUTOSAR COM 和其他的基础软件模块(BSW)是 VFB(Virtual Functional Bus) 概念的实现。RTE 实现了 AUTOSAR VFB 的接口，从而实现了 AUTOSAR 软件组件之间的通信。

RTE 是 AUTOSAR ECU 体系的核心,它提供了在 AUTOSAR 软件组件间通信的基础服务，扮演了一些方法，通过这些方法 AUTOSAR 软件组件能访问包括 OS 和通信服务在内的基础软件模块的。

2.3 系统服务

系统服务是一组可以由所有层次模块使用的模块和功能。例如实时操作系统、错误管理器和库功能。为应用和基本软件模块提供基本服务。它包含下图所示功能：



2.3.1 AUTOSAR OS

AUTOSAR OS 为实时应用提供了所有基本服务，即中断处理、调度、系统时间和时钟同步、本地消息处理，以及错误检测机制。所有服务都隐藏在良好定义的 API 之后。应用与 OS 和通信层的连接只通过 API。

AUTOSAR OS 的基本特征包括：

- 静态配置
- 能够推断实时系统性能
- 提供基于优先级的调度策略
- 提供运行时保护功能（存储、计时等）
- 可宿主在低端控制器上，并且不需要其他资源

它包含以下几个方面：

- **实时操作系统**

在嵌入式汽车 ECU 中的实时操作系统构成软件动态行为的基础。它管理任务和事件的调度，不同任务间的数据流，并且提供监控和错误处理功能。

但是，在汽车系统中，对操作系统的需求集中在特定领域。所使用的操作系统必须高效运行并且所占存储空间小。

在多媒体和远程信息处理应用中，操作系统提供的特征集以及可用计算资源有很大不同。在纯粹的任务管理之上，OS 中还包含了复杂的数据处理（例如，流、快速文件系统）、存储管理甚至图形用户接口。

汽车 OS 的典型领域涵盖了调度和同步的核心特征。在 AUTOSAR 中，上面讨论的附加特征在 OS 的范围之外，其他 WP4.2.2.1 工作包（例如 SPAL）涵盖了这些特征。在 AUTOSAR 的体系结构约束之下不可能把其他 OS（例如，QNX、VxWorks 和 Windows CE 等）的特征集集成到整体的 OS/通信/驱动结构中。因此，AUTOSAR OS 只考虑核心特征。

• **核心操作系统**

OSEK/VDK 操作系统广泛应用于汽车工业，并且已经证明了可以在现代车辆的所有 ECU 类型中使用。OSEK OS 引入的概念被广泛地理解，汽车工业领域在设计基于 OSEK OS 的系统方面有多年的经验。

OSEK OS 是一个事件触发的操作系统。这为基于 AUTOSAR 的系统的设计和维
护提供了高度的灵活性。事件触发使得可以自由地选择在运行时驱动调度的事件，例如角反转、局部时间源、全局时间源、错误出现等等。

由于这些原因，AUTOSAR OS 的核心功能必须基于 OSEK OS。OSEK OS 特别提供了以下特性以支持 AUTOSAR:

- 固定的基于优先级调度
- 处理中断的功能
- 只有中断有高于任务的优先级
- 一些防止错误使用 OS 服务的保护措施
- StartOS()和 StartupHook 启动接口
- ShutdownOS()和 ShutdownHook 关闭接口

AUTOSAR OS 基于 OSEK OS 意味着应用程序是向后兼容的。为 OSEK OS 编写的
应用程序可以在 AUTOSAR OS 上运行。但是，使用 AUTOSAR OS 引入的一些新
特性需要对已存在的 OSEK OS 特性的使用有所限制。例如：为定时器回调实现定时和
内存保护效率就会很低。此外，AUTOSAR OS 扩展了一些已存在的特性，例如直接通
过定时器驱动计数器。

AUTOSAR OS 提供的 API 向后兼容于 OSEK OS 的 API。新的需求作为功能扩展
来集成。

AUTOSAR OS 对 OSEK OS 扩展的 API 如下表:

服务名	语法
GetApplicationID	ApplicationType GetApplicationID (void)
GetISRID	ISRType GetISRID (void)
CallTrustedFunction	StatusType CallTrustedFunction (TrustedFunctionIndexType FunctionIndex, TrustedFunctionParameterRefType FunctionParams)

CheckISRMemoryAccess	AccessType CheckISRMemoryAccess (ISRTYPE ISRID, MemoryStartAddressType Address, MemorySizeType Size)
CheckTaskMemoryAccess	AccessType CheckTaskMemoryAccess (TaskType TaskID, MemoryStartAddressType Address, MemorySizeType Size)
CheckObjectAccess	ObjectAccessType CheckObjectAccess (ApplicationType ApplID, ObjectTypeType ObjectType, ...)
CheckObjectOwnership	ApplicationType CheckObjectOwnership (ObjectTypeType ObjectType, ...)
StartScheduleTableRel	StatusType StartScheduleTableRel (ScheduleTableType ScheduleTableID, TickType Offset)
StartScheduleTableAbs	StatusType StartScheduleTableAbs (ScheduleTableType ScheduleTableID, TickType Tickvalue)
StopScheduleTable	StatusType StopScheduleTable (ScheduleTableType ScheduleTableID)
NextScheduleTable	StatusType NextScheduleTable (ScheduleTableType ScheduleTableID_current, ScheduleTableType ScheduleTableID_next)
IncrementCounter	StatusType IncrementCounter (

	CounterType CounterID)
SyncScheduleTable	StatusType SyncScheduleTable (ScheduleTableType SchedTableID, GlobalTimeTickType GlobalTime)
SetScheduleTableAsync	StatusType SetScheduleTableAsync (ScheduleTableType ScheduleID)
GetScheduleTableStatus	StatusType GetScheduleTableStatus (ScheduleTableType ScheduleID, ScheduleTableStatusRefType ScheduleStatus)
TerminateApplication	StatusType TerminateApplication(RestartType RestartOption)
DisableInterruptSource	StatusType DisableInterruptSource (ISRTYPE DisableISR)
EnableInterruptSource	StatusType EnableInterruptSource (ISRTYPE EnableISR)
ProtectionHook	ProtectionReturnType ProtectionHook (StatusType Fatalerror)

- **静态定义的调度**

在许多应用中需要静态定义一组互相关联的任务的活动。这用于在基于数据流的设计中保证数据一致性，与时间触发的网络同步，保证正确的运行时定相，等等。

时间触发的操作系统通常作为这个问题的解决方法。然而，时间只是简单的事件，所以任何事件触发的 OS，包括 OSEK OS，会在汽车电子控制单元实现一个用于静态调度实时软件的调度器。

- **监控功能**

监控功能在适当的执行阶段检测错误，不是在错误发生的时刻。因此，监控功能是在运行时捕捉失效，而不是预防故障。

- **保护功能**

AUTOSAR 概念需要多来源的 OS 应用共存在同一处理器中。为了防止这些 OS 应用之间意想不到的交互，需要提供保护机制。

- **计时器服务**

计时器服务为应用和基础软件提供软件计时器。

计时机制的核心已经由 OSEK OS 的计数器和闹钟提供。如果要提供通用的软件计时，一些补充特性需要添加到 AUTOSAR OS。

- **时间触发操作系统**

时间触发操作系统在汽车电子控制单元实现了一个用于静态调度实时软件的调度器。

另外，操作系统为实时应用提供了所有基本服务，即中断处理、调度、系统时间和时钟同步、本地消息处理，以及错误检测机制。

所有服务都隐藏在良好定义的 API 之后。应用与 OS 和通信层的连接只通过 API。

对于特殊的应用，操作系统能够配置为只包含该应用需要的服务。因此操作系统的资源需求尽量少。

2.3.2 BSW 调度器

BSW 调度器是系统服务的一部分，因此它向所有层次的所有模块提供服务。但是，与其它 BSW 模块不同，BSW 调度器提供了**集成**的概念和服务。BSW 调度器提供方法用以：

- 把 BSW 模块的实现嵌入 AUTOSAR OS 上下文
- 触发 BSW 模块的主要处理功能
- 应用 BSW 模块的数据一致性机制

集成者的任务是**应用**所给的方法（AUTOSAR OS 提供的），在特定项目环境中以良好定义和有效的方式把 BSW 模块装配起来。

这也意味着 BSW 调度器只是**使用** AUTOSAR OS。它与 AUTOSAR OS 调度器并不冲突。

BSW 调度器的实现基于：

- BSW 模块的 BSW 模块描述
- BSW 调度器的配置

2.3.3 模式管理

模式管理簇包括三个基本软件模块：

- ECU 状态管理器，控制 AUTOSAR BSW 模块的启动阶段，包括 OS 的启动；
- 通信管理器，负责网络资源管理；
- 看门狗管理器，基于应用程序的生存状态触发看门狗。

2.3.3.1 ECU 状态管理器

ECU 状态管理器是一个基本软件模块，管理 ECU 的状态（OFF、RUN、SLEEP），以及这些状态之间的转换（过渡状态：STARTUP、WAKEUP、SHUTDOWN）。详细地，ECU 状态管理器：

- 负责初始化和 de-initialization 所有基本软件模块，包括 OS 和 RTE；
- 在需要时与所谓的资源管理器（例如，通信管理器）协作，关闭 ECU；
- 管理所有唤醒事件，并在被要求时配置 ECU 为 SLEEP 状态。

为了完成所有这些任务，ECU 状态管理器提供了一些重要的协议：

- RUN 请求协议，调整 ECU 是保持活动状态还是准备关闭，
- 唤醒确认协议，从“不稳定的”唤醒事件中区分出“真正的”唤醒事件，
- 时间触发的增多非工作状态协议（Time Triggered Increased Inoperation - TTII），允许 ECU 更多地进入节能的休眠状态。

ECU 状态管理器必须支持独立的预处理动作和过渡，以启动 ECU 或将其转换到低功耗状态（例如，休眠状态/备用状态）。通过良好使用 ECU 状态管理器的特性和能力，此模块能够用于执行电源消耗的预定义策略，因此提供了对 ECU 的有效能源管理。

ECU 状态管理器的特性和优势包括：

- 初始化和关闭基本软件模块。
- ECU 主要状态的标准化定义。
- 时间触发的更多非工作状态。

2.3.3.2 看门狗管理器

看门狗管理器是 AUTOSAR（服务层次）的标准化基本软件体系结构的基本软件模块。它监控与计时约束有关的应用执行的可靠性。

分层体系结构方法使得应用计时约束和看门狗硬件计时约束分离。基于此，看门狗管理器在触发看门狗硬件的同时提供了对一些独立应用的生存监控。

看门狗管理器提供以下特性：

- 监督多个处于 ECU 的单独应用，这些应用有独立的计时约束并且需要特别监督运行时的行为和生存状态。
- 每个独立的受监控实体都有故障响应机制。
- 可以关闭对单独应用的监督，而不会违反看门狗触发（例如，对于禁止的应用）。
- 通过看门狗驱动触发内部或外部、标准或窗口，看门狗。（internal or external, standard or window, watchdog）对内部或外部看门狗的访问由看门狗接口处理。
- 根据 ECU 状态和硬件性能选择看门狗模式（Off Mode, Slow Mode, Fast Mode）。

2.3.3.3 通信管理器

通信管理器收集并协调来自通信请求者（用户）的访问请求。

通信管理器的目的是：

- 简化通信协议栈的使用。包括通信栈的初始化，以及简单的网络管理。

- 调整 ECU 上多个独立软件组件的通信栈（允许发送和接收消息）的可用性。
- 暂时禁止发送消息以阻止 ECU（主动地）唤醒物理通道。
- 通过为每个物理通道实现一个状态机来控制 ECU 的多个物理通道。
- 可以强制 ECU 保持物理通道处于“silent 通信”模式。
- 分配所请求的通信模式需要的所有资源，简化资源管理。

通信管理器定义了“通信模式”，表示一个特定的物理通道对于应用是否可用，以及如何使用（发送/接收，只接收，即不发送也不接收）。

2.4 诊断服务

2.4.1 诊断事件管理器

诊断事件管理器 DEM（Diagnostic Event Manager）是一个子组件，如同 AUTOSAR 内诊断模块的诊断通信管理器（DCM）和功能禁止管理器（FIM）。它负责处理和存储诊断事件（错误）和相关 FreezeFrame 数据。DEM 进一步提供故障信息给 DCM（例如，从事件存储器读取所有存储的 DTC（Diagnostic Trouble Code））。DEM 给应用层、DCM 和 FIM 提供接口。定义了可选的过滤服务。

2.4.2 功能禁止管理器

功能禁止管理器 FIM（Function Inhibition Manager）负责提供软件组件和软件组件功能的控制机制。功能由一个、多个或部分有相同权限/禁止条件的可执行实体构成。通过 FIM 方法，对这些功能的禁止可以配置甚至修改。所以，极大地提高了功能在新系统环境下的适应性。

FIM 意义上的功能与可执行实体是不同并且独立的类型。可执行实体主要由调度需求来区分。与此相对的是，功能由禁止条件来分类。FIM 服务关注 SW-C 的功能，但是并不局限于此。BSW 的功能也能够使用 FIM 服务。

功能被指定了一个标识符（FID – function identifier），以及该特定标识符的禁止条件。功能在执行之前获得它们各自的权限状态。如果禁止条件应用于特定标识符，对应的功能将不再执行。

FIM 与 DEM 密切相关，因为诊断事件和它们的状态信息作为禁止条件被提供给 FIM。如果检测到了失效，并且事件报告给了 DEM，那么 FIM 禁止 FID 和对应的功能。

为了处理功能和关联事件的关系，功能的标识符和禁止条件必须引入到 SW-C 模板中（与 BSW 等价），并且在配置期间构造数据结构以处理标识符对于特定事件的敏感性。这些关系在每个 FIM 中是唯一的。

RTE 和 FIM 之间没有功能上的联系。在 AUTOSAR 中，RTE 按照接口和调度需求处理 SW-C。与此相对的是，FIM 处理禁止条件并通过标识符（FID）为控制功能提供支持机制。因此，FIM 概念和 RTE 概念不互相干扰。

2.4.3 开发错误跟踪器

开发错误跟踪器 DET（Development Error Tracer）主要用于在开发期间跟踪和记录错误。API 参数给出了追踪源和错误类型：

- 错误所在的模块

- 错误所在的功能
- 错误类型

由软件开发者和软件集成者在特定应用和测试环境下为 API 功能选择最优的策略。可能包括以下功能：

- 在错误报告 API 内设置调试器断点
- 计算报告的错误
- 在 RAM 缓存中记录调用和传递的参数
- 通过通信接口发送报告的错误到外部日志

DET 仅仅是对 SW 开发和集成的辅助，并不一定要包含在产品代码中。API 已经定义，但是功能由开发者根据特定需求来选择/实现。

2.4.4 诊断通信管理器

诊断通信管理器 DCM (Diagnostic Communication Manager) 确保诊断数据流，并且管理诊断状态，特别是诊断对话期和安全状态。另外，DCM 检查诊断服务请求是否被支持，以及根据诊断状态判断服务是否可以在当前对话期中执行。

DCM 为所有诊断服务提供连接到 AUTOSAR-RTE 的接口。另外 DCM 也处理一些基本诊断服务。

在 AUTOSAR 体系结构中，诊断通信管理器 (DCM) 处在通信服务中 (服务层)。DCM 是应用和 PDU 路由器封装的车辆网络通信 (CAN、LIN、FlexRay 和 MOST) 之间的中间层。DCM 与 PDU 路由器之间有一个接口。在通信过程中，DCM 从 PDU (Protocol Data Unit) 路由器接收诊断消息。

DCM 在其内部处理、检查诊断消息，并把消息传送到 AUTOSAR SW 组件进一步处理。根据诊断服务 ID，将执行应用层中的相应调用。

DCM 必须是网络无关的，所以协议和消息配置在 DCM 的下层。这需要连接到 PDU 路由器的网络无关接口。

DCM 由以下功能块组成：

DSP - Diagnostic Service Processing

DSP 主要包含了完整实现的诊断服务，这些服务在不同的应用之中是通用的（例如，访问故障数据），所以不需要由应用实现。

DSD-Diagnostic Service Dispatcher

DSD 的目的是处理诊断数据流。这里的处理意味着：

通过网络接收新的诊断请求并发送到数据处理器。

当被应用触发时，通过网络传送诊断响应 (AUTOSAR SW-Component或DSP)。

DSL-Diagnostic Session Layer

DSL 保证数据流与诊断请求和响应有关。DSL 也监督和确保诊断协议计时。进一步，DSL 管理诊断状态。

2.5 存储栈

2.5.1 存储服务

存储服务由一个 **NVRAM** 管理器模块构成，负责管理非易失性数据（从不同存储驱动读/写）。它需要一个 **RAM** 镜像作为数据接口提供给应用快速读取。

存储服务的任务是以统一方式向应用提供非易失性数据。这抽象了存储位置和属性。提供非易失性数据管理机制，如保存、加载、校验和保护和验证、可靠存储等。

2.5.1.1 存储硬件抽象的寻址方案

存储抽象接口和下层的闪存 **EEPROM** 仿真和 **EEPROM** 抽象层向 **NVRAM** 管理器提供虚拟线性 32 位地址空间。这些逻辑 32 位地址由 16 位逻辑块号和 16 位块地址偏移量组成。因此 **NVRAM** 管理器（理论上）可以有 65536 个逻辑块，每个逻辑块（理论上）可以有 64Kbytes。

NVRAM 管理器进一步将 16 位逻辑块号划分为以下部分：

- （16-NVM_DATASET_SELECTION_BITS）位的块标识符
- NVM_DATASET_SELECTION_BITS 位的数据索引，每个 **NVRAM** 块最多可以有 256 个数据集

2.5.1.2 NVRAM 管理器

非易失性 **RAM** 管理器（**NVRAM Manager**）管理所有非易失性存储器中数据的存储。

NVRAM 管理器本身与硬件无关，所有直接存取硬件的功能，例如内部或外部 **EEPROM**、内部或外部闪存中的仿真 **EEPROM** 等，封装在基本 **SW** 的较低层。在汽车环境中，**NVRAM** 管理器提供服务以根据各个数据的需求来保证数据存储和 **NV** 数据的维护。**NVRAM** 管理器要能够管理 **EEPROM** 和/或 **FLASH EEPROM** 仿真设备的 **NV** 数据。**NVRAM** 管理器为 **NV** 数据的管理和维护提供所需的同步/异步服务（初始化/读/写/控制）。**NVRAM** 管理器处理对非易失性数据的并行访问，并为单个数据元素提供可靠性机制，如校验和保护。

为了适用于汽车系统的所有领域，**NVRAM** 管理器需要具有高度的伸缩性（如定义请求队列的数目和大小，支持不同的块管理类型，**EEPROM** 仿真，等等）。

2.5.1.3 基本存储对象

NV块：NV块表示NV用户数据和CRC值（可选）组成的存储区；

RAM块：RAM块表示在RAM中用户数据和CRC值（可选）组成的区域；

ROM块：ROM块驻留在ROM（闪存）中，用于提供缺省数据以防NV块为空或被破坏；

管理块：管理块在 **RAM** 中，包含与 **Dataset NV** 块关联的块索引。另外，也包含相应 **NVRAM** 块的属性/错误/状态信息。

2.5.1.4 块管理类型

以下 **NVRAM** 存储类型应该由 **NVRAM** 管理器支持，并且由以下基本存储对象构成：

管理类型	NV 块	RAM 块	ROM 块	管理块
NVM_BLOCK_NATIVE	1	1	0..1	1
NVM_BLOCK_REDUNDANT	2	1	0..1	1
NVM_BLOCK_DATASET	1..	1	0..n	1

	(m<256)			
--	---------	--	--	--

Native NVRAM 块是最简单的块管理类型。以最小的开销存储/检索 NV 存储区。Native NVRAM 块由单个 NV 块、RAM 块和管理块组成。

Redundant NVRAM块有更高的容错性、可靠性和可用性，以及对数据被破坏的抵抗性。Redundant NVRAM块由两个NV块、一个RAM块和管理块组成。

Dataset NVRAM 块是相同大小数据块（NV/RAM）的阵列。应用一次只能存取其中的一个。Dataset NVRAM 块由多个 NV 用户数据和（可选）CRC 区域、一个 RAM 块和管理块组成。

2.5.1.5 NVRAM 管理器的 API 配置种类

为了使NVRAM管理器适合于有限的硬件资源，定义了3种不同的API配置种类：

- API 配置种类 3：
所有规定的API调用都可用。支持最大的功能性。
- API 配置种类 2：
API调用的中间集可用。
- API 配置种类 1：

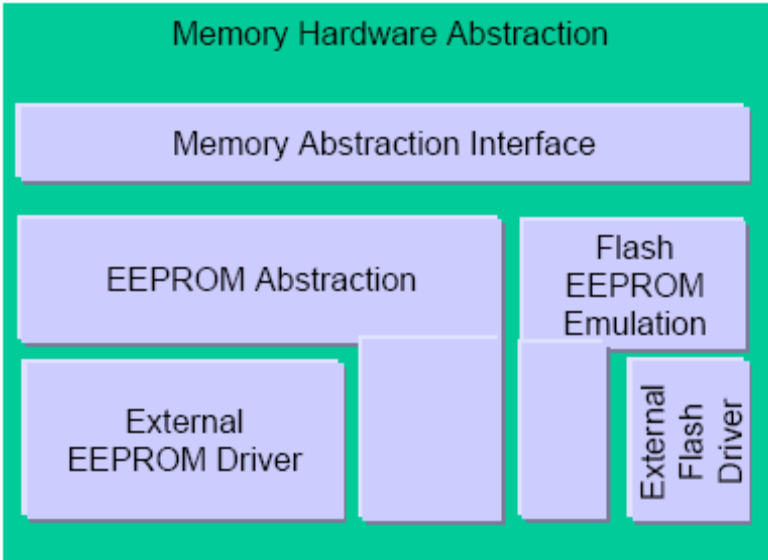
特别用于满足资源非常有限的系统，此 API 配置种类只提供所需要的 API 调用的最小集。

2.5.2 存储硬件抽象

存储硬件抽象是一组抽象于外围存储设备位置（片上或板上）和 ECU 硬件布局的模块。例如：片上 EEPROM 和外部 EEPROM 设备应该可以通过相同的机制存取。

通过存储器特有抽象/仿真模块访问存储驱动（例如 EEPROM 抽象）。通过仿真 EEPROM 接口和闪存硬件单元，就可以通过存储抽象接口访问这两种类型的硬件。

存储硬件抽象的任务是提供访问内部（片上）和外部（板上）存储设备和存储硬件类型（EEPROM、闪存）的相同机制。



2.5.2.1 EEPROM 抽象

EEPROM 抽象层 (EA) 扩展 EEPROM 驱动, 向上层提供线性地址空间上的虚拟分段和“实际上无限制的”擦除/写循环。除此之外, 它还应该提供与 EEPROM 驱动相同的功能。

2.5.2.2 闪存 EEPROM 仿真

闪存 EEPROM 仿真 (FEE) 按照闪存技术仿真 EEPROM 抽象层的行为。所以它与 EEPROM 抽象层有相同的功能和 API, 并且给出基于下层闪存驱动和闪存设备的相似配置。

2.5.2.3 内存抽象接口

内存抽象接口 (MemIf) 允许 NVRAM 管理器存取多个存储抽象模块 (FEE 或 EA 模块)。

内存抽象接口抽象于下层 FEE 和 EA 模块的数目, 并向上层提供统一线性地址空间上的虚拟分段。

2.5.3 存储驱动

2.5.3.1 EEPROM 驱动

EEPROM 驱动提供读、写、擦除 EEPROM 的服务。也提供了用于比较 EEPROM 中数据块和内存中数据块的服务。这些服务是异步的。有两类 EEPROM 驱动:

- 内部 EEPROM 驱动
- 外部 EEPROM 驱动

内部 EEPROM 驱动直接访问微控制器硬件, 并且定位在微控制器抽象层。外部 EEPROM 驱动使用处理程序 (handler) 或驱动访问外部 EEPROM 设备。它定位在 ECU 抽象层。

两种类型的驱动的功能需求和功能范围都是相同的。所以 API 在语义上是相同的。

2.5.3.2 闪存驱动

如果受到底层硬件的支持, 闪存驱动提供读、写和擦除闪存的服务, 以及设置写/擦除保护的配置接口。闪存驱动提供了一个内置加载器, 以加载闪存存取代码到 RAM 中, 并在需要的时候执行写/擦除操作。

在 ECU 应用模式下, 闪存驱动只用于闪存 EEPROM 仿真模块写数据。在应用模式下并不将程序代码写到闪存中。这应该由启动模式处理, 超出了 AUTOSAR 的范围。

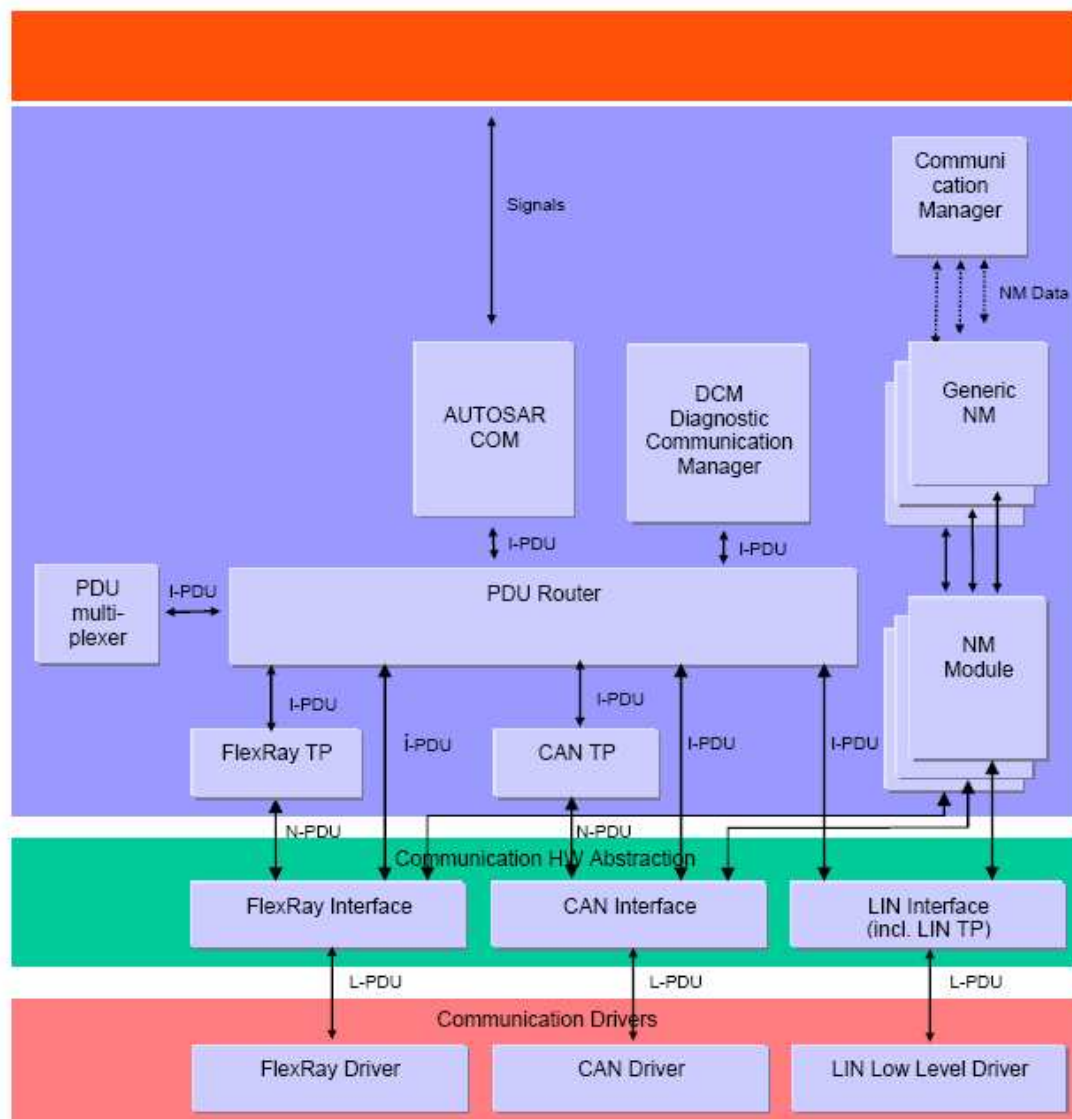
有两类闪存驱动:

- 内部闪存驱动
- 外部闪存驱动

内部闪存的驱动直接存取微控制器硬件, 并且定位在微控制器抽象层。外部闪存通常通过微控制器数据/地址总线连接, 然后闪存驱动使用总线的处理程序/驱动访问外部闪存设备。外部闪存设备的驱动定位在 ECU 抽象层。两种类型的驱动的功能需求和功能范围都是相同的。所以 API 在语义上是相同的。

2.6 通信栈

AUTOSAR 通信栈的概貌如下图：

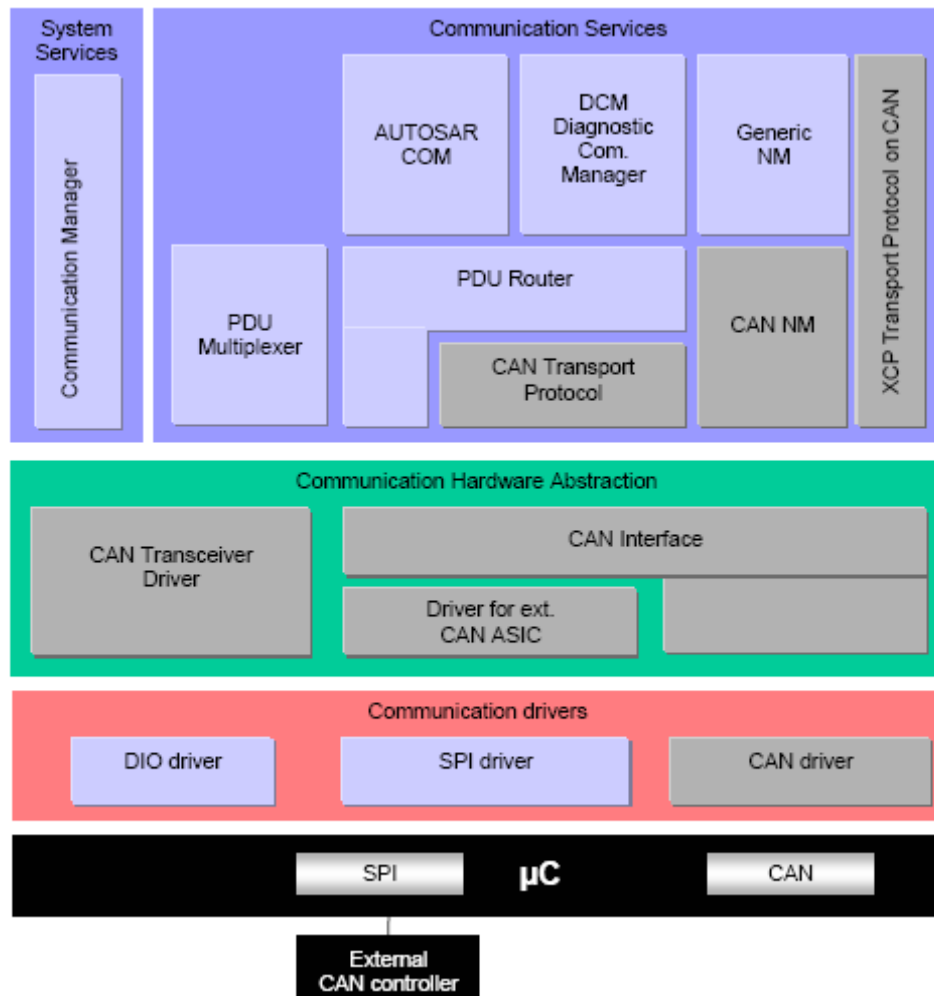


AUTOSAR 中的通信栈包含以下这些部分：

2.6.1 CAN

- **AUTOSAR CAN**

AUTOSAR CAN 模型如下图：



- **CAN 驱动**

CAN 驱动为上层使用者提供统一的接口——CAN 接口。CAN 驱动尽可能合理地隐藏了相关 CAN 控制器的硬件专用性。

CAN 驱动是最底层的一部分，为上层执行对硬件的访问和提供硬件无关的 API。上层中唯一能够访问 CAN 驱动的是 CAN 接口。

如果几个 CAN 控制器属于相同的 CAN 硬件单元，那么它们能够由 CAN 驱动来控制。

一个 CAN 控制器总是与一个物理通道相关联。它被允许与总线上的物理通道相连接，不管 CAN 接口是否将相关的 CAN 控制器分别对待。

- **CAN 接口（硬件抽象）**

CAN 接口提供标准化的接口，通过 ECU 的 CAN 总线系统来支持通信。其 API 与专用 CAN 控制器及其通过 CAN 驱动层的访问无关。CAN 接口能够通过统一的接口访问一个或多个 CAN 驱动。

CAN 接口仅能用于 CAN 通信，并且是为操作一个或多个底层 CAN 驱动而专门设计。涵盖不同 CAN 硬件单元的多个 CAN 驱动模块由一个在 CAN 驱动规范中指定的通用接口来表示。CAN 之外（也就是 LIN）的其他协议不支持。

- **CAN 传输层**

CAN 传输层是位于 PDU 路由和 CAN 接口模块之间的模块。其主要作用是分割和合并大于 8 字节的 CAN I-PDU。

根据 AUTOSAR 基本软件体系结构，CAN 传输层提供的服务有：

- 发送方向的数据分割；
- 接收方向的数据合并；
- 数据流控制；
- 分割期间内的错误检测。

AUTOSAR 体系结构定义了通信系统的各个具体的传输层(CanTp、包含 LinIf 的 LinTp、FlexRayTp)。因此，CAN 传输层仅涵盖了 CAN 传输协议的细节。

CAN 传输层拥有一个接口，该接口连接一个单独的下层 CAN 接口层和一个单独的上层 PDU Router 模块。

根据 AUTOSAR 发布的计划，该 CAN 传输层规范包含下面的限制：

- CAN 传输层仅运行在事件触发模式中，
- 没有传送/接收撤消。

- **CAN 收发器驱动**

CAN 收发器驱动负责处理 ECU 上的 CAN 收发器，依据的是与整个 ECU 当前状态相关的总线专用 NM 的状态。

CAN 收发设备驱动的目标：CAN 收发设备驱动抽象使用 CAN 收发设备硬件芯片。它向更高层提供硬件无关接口。它也可以通过 MCAL 层的 API 从 ECU 设计中抽象出来，访问 CAN 收发设备硬件。

CAN 收发设备硬件必须提供功能和接口，以映射到 AUTOSAR CAN 收发设备驱动的运行模式模型上。

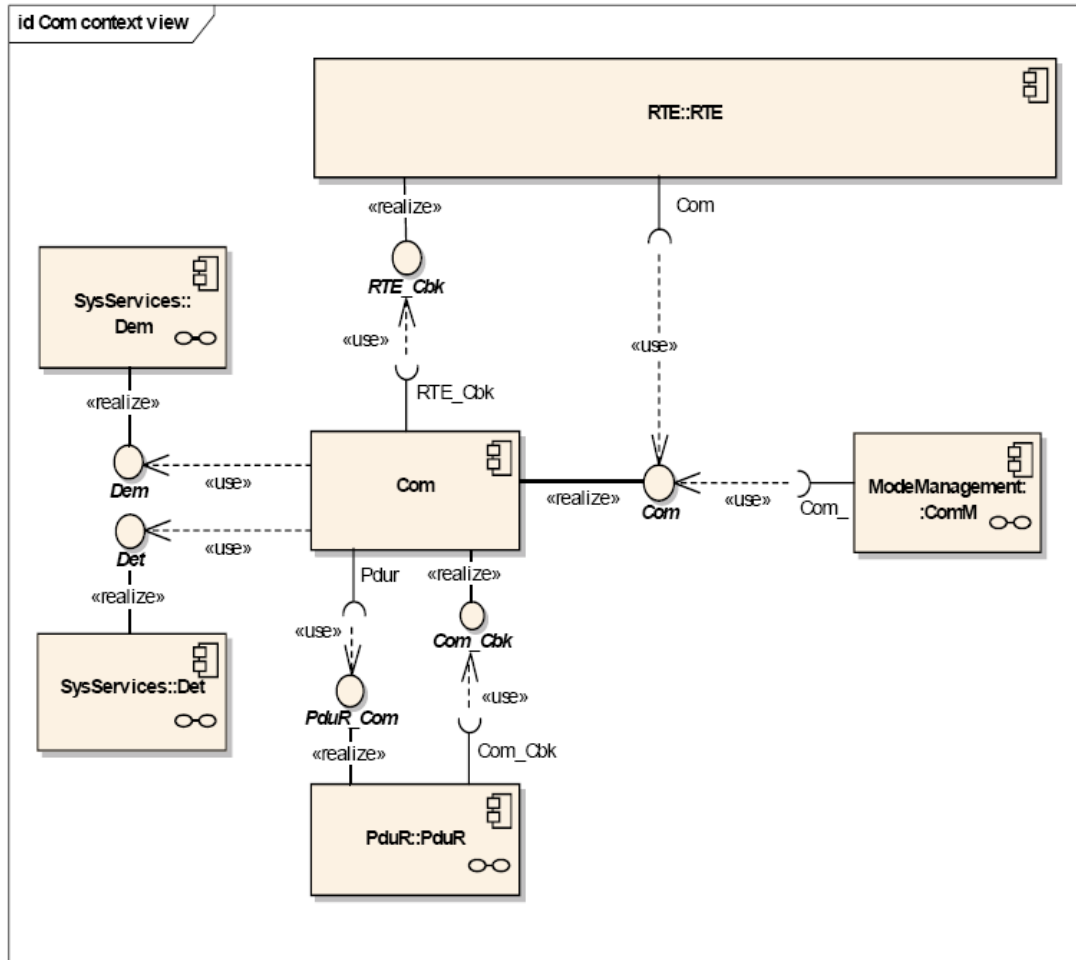
下层驱动（SPI 和 DIO）使用的 API 必须同步。不支持同步行为的下层驱动的实现不能与 CAN 收发设备驱动一起使用。

2.6.2 COM

- **AUTOSAR COM**

AUTOSAR COM 层位于 RTE 和 PDU 路由器之间。它来源于 OSEK_COM 标准。AUTOSAR COM 提供了信号网关功能。

COM 与其它模块的依赖关系如下图所示：



• COM Manager

COM Manager（COM 管理）是基本软件 Basic Software（BSW）的一个组件。它是囊括了下层通信服务的控制的资源管理。

COM Manager 控制的基本软件模块（BSW）与通信相关，而不是与软件组件或可运行实体相关。

COM Manager 从通信请求者那里收集总线通信访问请求，并协调总线通信访问请求。

COM Manager 的目标是：

（1）为用户简化总线通信栈的使用。这包括了总线通信栈的初始化和简化的网络管理处理。

（2）协调与多个软件组件（在一个 ECU 上）无关的总线通信栈（允许信号的发送和接收）的可用性。

（3）临时性取消信号的发送以阻止 ECU 唤醒通信总线。

（4）控制 ECU 的一个以上的通信总线通道，这通过为每个通道实现一种状态机制来实现。

（5）提供使 ECU 保持总线处于“静默通信”模式。

（6）通过分配对请求通信模式必需的所有资源来简化资源管理。

COM Manager 包含以下基本功能：

状态机制	无通信状态
	静默通信状态：网络释放状态、预备总线睡眠状态
	完全通信状态：网络请求状态、准备睡眠状态
扩展功能	状态期间范围
	通信阻止：总线唤醒阻止、静默通信模式的限制、无通信模式的限制
总线通信管理	
网络管理依赖	
总线错误管理	CAN 总线关闭处理 CAN Bus Off handling
	网络起动指示 Network Start Indication
	传输故障例外
	网络超时例外
测试支持需求	阻止完全通信请求计数器
错误分类	
错误检测	
错误通知	
非功能性需求	

• AUTOSAR COM 与 OSEK COM 的比较

根据通信部分提供的功能，对比两者在相同功能上的 API，以及两者各自所特有的 API，由于 AUTOSAR COM 较之 OSEK COM，多出了一个 COM Manager，即通信管理模块部分，所以整个 AUTOSAR COM Manager 为 AUTOSAR 标准所特有，下面先对两者的相同功能部分作比较。

1、相同功能及服务

（1）启动与控制服务

OSEK	AUTOSAR
------	---------

StartCOM	Com_Init
StopCOM	Com_DeInit
GetCOMApplicationMode	Com_IpduGroupStart
InitMessage	Com_IpduGroupStop
StartPeriodic	Com_DisableReceptionDM
StopPeriodic	Com_EnableReceptionDM
	Com_GetStatus
	Com_GetConfigurationId
	Com_GetVersionInfo

两者在通信的启动与控制服务部分的对比可以看出：首先，AUTOSAR 提供的 API 较多，表明它的功能较强；其次，AUTOSAR 的启动与控制服务中包含对 I-PDU（交互层协议数据单元）的处理和控制，如 Com_IpduGroupStart、Com_IpduGroupStop。

(2) 通信服务

OSEK	AUTOSAR
SendMessage	Com_SendSignal
ReceiveMessage	Com_ReceiveSignal
SendDynamicMessage	Com_UpdateShadowSignal
ReceiveDynamicMessage	Com_SendSignalGroup
SendZeroMessage	Com_ReceiveSignalGroup
GetMessageStatus	Com_ReceiveShadowSignal
COMErrorGetServiceId	Com_InvalidateSignal
COMError_Name1_Name2	Com_InvalidateShadowSignal
	Com_TriggerIPDUSend

通过对比可以看出，OSEK 通信服务中包含了对错误的一些简单的处理，如获得错误服务的 Id（COMErrorGetServiceId），而 AUTOSAR 通信服务仍然包含对 I-PDU 的处理，如 Com_TriggerIPDUSend。

(3) 通知机制支持服务（OSEK）与回调通知服务（AUTOSAR）

OSEK	AUTOSAR
------	---------

ReadFlag	Com_TriggerTransmit
ResetFlag	Com_RxIndication
	Com_TxConfirmation

两者在这个部分提供的功能差别不大，主要是对一些标志的修改和设置，以控制通信的状态和执行的功能。

2、不同功能及服务

(1) OSEK 为 I-PDU 的处理提供一类专门的服务，称为 OSEK 间接网络管理接口，包含 2 个 API: I-PDU 传输指示 (I_MessageTransfer) 和 I-PDU 超时指示 (I_MessageTimeOut)。

(2) OSEK 通信部分提供了一些例行程序对通信起扩展作用，包含 3 个 API: StartCOMExtension、COMCallouts、COMErrorHook。

(3) AUTOSAR 提供了一些调度函数，主要是对消息或信号的接收或发送起路由、调度的作用，包含 3 个 API: Com_MainFunctionRx、Com_MainFunctionTx、Com_MainFunctionRouteSignals。

(4) AUTOSAR 的通信部分有一个 COM Manager，这是一个通信管理模块，是 AUTOSAR 标准特有的，主要负责对通信进行监控、管理、诊断以及管理涉及通信的 ECU 状态。下表列出了它所提供的部分 API。

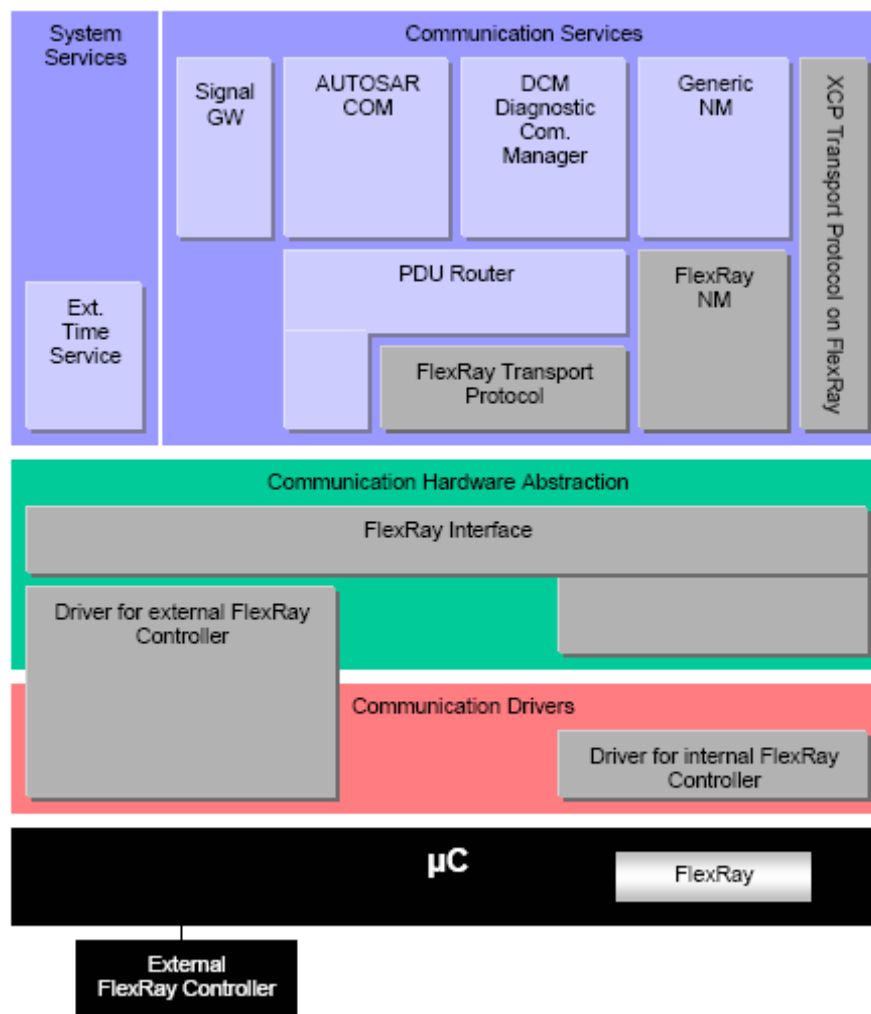
功能定义		ComM_Init
		ComM_DeInit
		ComM_GetStatus
		ComM_GetInhibitionStatus
		ComM_RequestComMode
		ComM_GetMaxComMode
		ComM_GetRequestedComMode
		ComM_GetCurrentComMode
	
专用函数	AUTOSAR 通用网络管理	ComM_Nm_NetworkStartIndication
		ComM_Nm_TransmissionFailure
		ComM_Nm_NetworkTimeout
	
	AUTOSAR 诊断通信管理	ComM_DCM_ActiveDiagnostic
		ComM_DCM_InactiveDiagnostic
	AUTOSAR ECU 状态管理	ComM_EcuM_RunModeIndication
		ComM_EcuM_WakeUpIndication

	总线接口	ComM_BusIf_BusOffIndication
	调度函数	ComM_MainFunction

2.6.3 FlexRay

- AUTOSAR FlexRay

AUTOSAR FlexRay 的分层体系结构如下图所示：



- FlexRay 接口

FlexRay 接口提供一种标准化的接口以访问 FlexRay 通信系统/硬件。FlexRay 接口必须与所使用的专用 FlexRay CC 及其通过 FlexRay 驱动器的访问无关。FlexRay 接口提供通过统一接口的对一个或几个 FlexRay 驱动器的访问。

FlexRay 接口的主要任务有：

- (1) 为上层提供到 FlexRay 通信系统的抽象接口。
- (2) FlexRay 接口通过一个或多个硬件专用驱动模块来访问 FlexRay 硬件，而不是直

接访问。

(3) 为了访问 FlexRay 通信控制器，FlexRay 接口使用一个或多个 FlexRay 驱动模块。

(4) 为了访问 FlexRay 收发器，FlexRay 接口使用一个或多个 FlexRay 收发器驱动模块。

(5) FlexRay 接口可执行代码与 FlexRay 通信控制器和 FlexRay 收发器完全不相关。

(6) FlexRay 接口允许代码模块的对象代码提交，遵循“one-fits-all”原则。

(7) FlexRay 接口提供给上层 AUTOSAR BSW 模块的功能如下：

- A. 初始化
- B. 配置/重配置
- C. 数据传送（发送和接收）
- D. 启动/停止/中断通信
- E. FlexRay 专用服务
- F. 设置运行模式
- G. 获取状态信息
- H. 各种计时器功能

- **FlexRay 驱动**

FlexRay 驱动模块必须为 FlexRay 接口模块、API 的使用者提供统一接口，以访问许多 FlexRay 通信控制器，这些控制器通常是相同类型的。FlexRay 驱动是一个软件层，它将抽象功能请求映射到 CC 专用硬件的序列上。CC 的硬件实现将从 FlexRay 接口隐藏。

- **FlexRay 传输层**

FlexRay 传输层为使用物理地址和功能地址的、分段式的确认过的和未确认过的 1 对 1 通信，以及分段式的未确认过的 1 对 n 通信提供支持。

- **FlexRay 收发器驱动**

FlexRay 收发器驱动负责处理 ECU 上的 FlexRay 收发器，其依据是总线专用 NM 的状态。

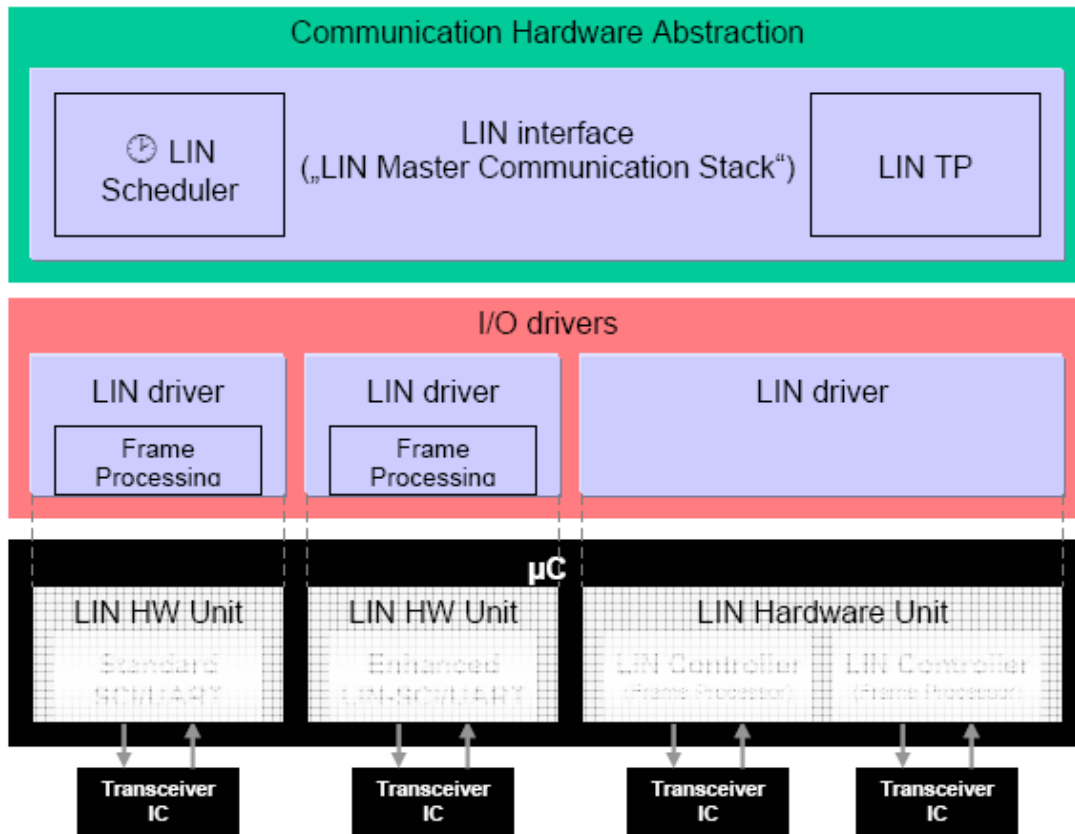
2.6.4 IPDUM

PDU 多路技术是指通过其 SDU（Service Data Unit）的一个以上的特定设计来使用一个 PDU（Protocol Data Unit）的相同 PCI（Protocol Control Information）。选择子字段是多路 PDU 的 SDU 的一部分。它用于区别多路 PDU 之间的内容。

2.6.5 LIN

- **AUTOSAR LIN**

AUTOSAR LIN 的分层体系结构如下图所示：



• LIN 驱动

LIN 驱动是最底层的一部分，执行硬件访问和为上层提供硬件无关的 API。上层唯一能够访问到 LIN 驱动的就是 LIN 接口。

一个 LIN 驱动能够支持一个以上的通道。LIN 驱动能够处理一个或多个属于相同 LIN 硬件单元的 LIN 通道。

• LIN 接口

LIN 接口被设计成硬件无关的。到上层模块（PDU 路由器）和下层模块（LIN 驱动）的接口被很好地定义。

LIN 接口可以处理一个以上的 LIN 驱动。一个 LIN 驱动能够支持一个以上的通道。这指的是 LIN 驱动能够处理一个或多个 LIN 通道。

LIN 接口负责向上层提供 LIN 2.0 主要功能有：

- (1) 为每个与 ECU 连接的 LIN 总线执行当前选择的调度。
- (2) 当上层请求到来时，切换调度表。
- (3) 从上层接收帧的传送，并传送数据部分作为适当 LIN 帧中的响应。
- (4) 当相应的响应在适当的帧中接收时，为上层提供帧接收通知。
- (5) 睡眠和唤醒服务
- (6) 错误处理

(7) 诊断传输层服务

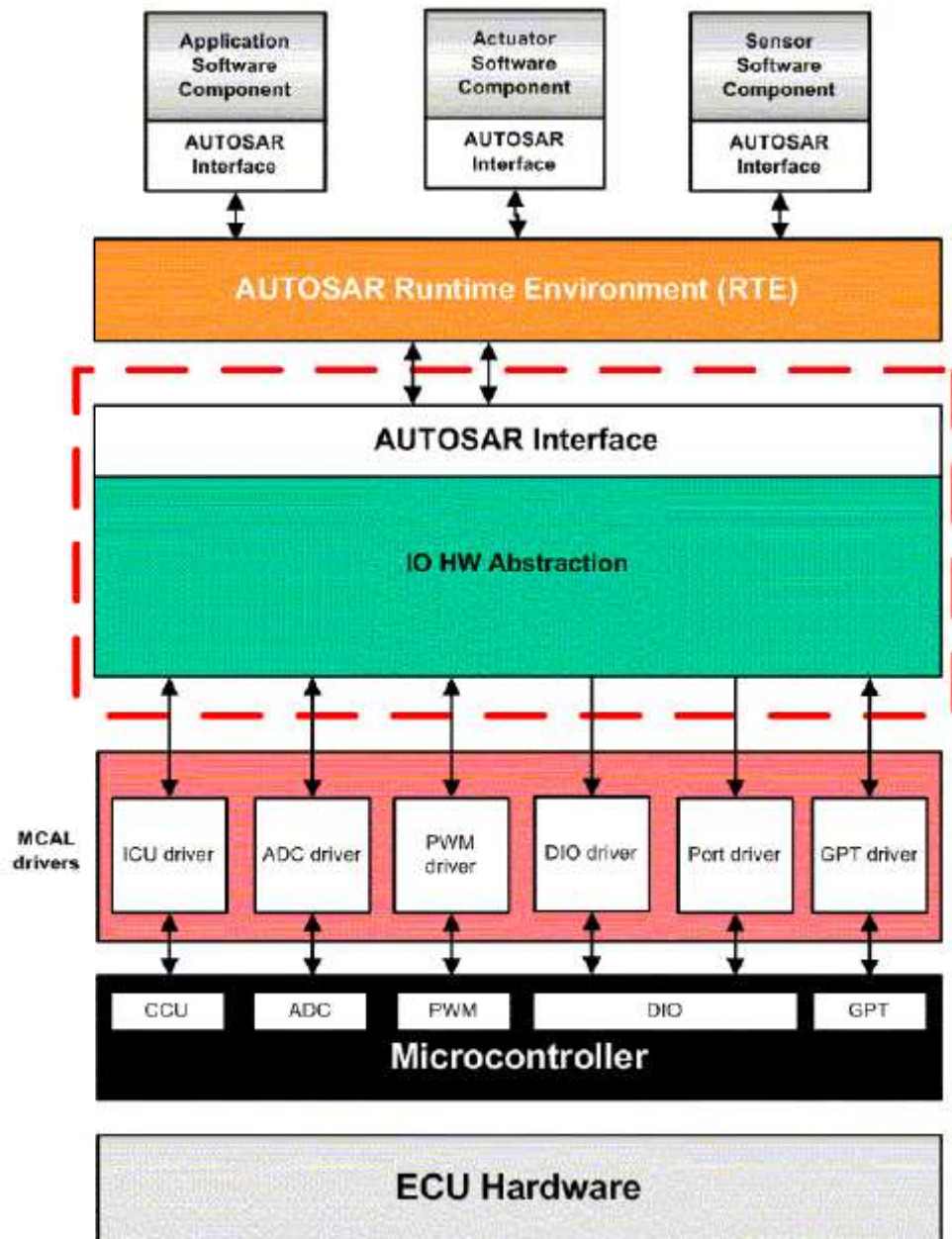
3 外设

3.1 IO 硬件抽象

IO 硬件抽象是 ECU 抽象层的一部分。IO 硬件抽象模块的目标是使数据通过 RTE 来传输，而完全不依赖于 ECU 硬件。这就是说软件组件设计者不需要更多的了解信号是如何影响物理层的。因此，该模块是特定于 ECU 的。

这主要通过把 ECU 信号映射到 IO 抽象端口上来实现。模块 IO 硬件抽象要提供用于初始化整个 IO 硬件抽象的模块。

下图显示了 IO 硬件抽象模块。它位于 MCAL 驱动之上。就是说 IO 硬件抽象模块要调用驱动 API 来管理片上设备。MCAL 驱动的配置依赖于将由 IO 硬件抽象模块提供的 ECU 信号的质量。例如，当引脚层发生相关的改变时（上升沿、下降沿），需要进行通知。系统设计者不得不配置 MCAL 驱动，以允许对给定信号进行通知。通知来自于驱动，并在 IO 硬件抽象模块中进行处理。



3.2 ADC 驱动

ADC 驱动初始化并控制微控制器内部的模数转换单元。该驱动包含一系列的基本功能函数。为了能够在某些特殊的应用中进行信号的频率分析（例如，快速傅立叶变换），就需要加强流式存取的功能。

ADC 驱动提供以下服务：

- 信号值结果的访问模式
- 流式访问。

通常，ADC 通道的变换请求通过 ADC 通道组来进行控制。通道组可以运行于持续的变换模式或者单触发变换模式。

变换处理和交互作用：

在同一时刻，ADC 驱动要管理一个以上的被配置成不同变换模式的组。

转换过程：

通常，ADC 通道的转换请求通过 ADC 通道组来进行控制。一个组可以运行于持续的转换模式或者单触发转换模式。单触发转换模式的触发条件也要被配置和控制。

如果通道运行于不同的模式（例如，在普通操作时采用持续的转换模式，在特定时间点的特殊转换时使用单触发或者按照命令的转换方式），通道必须被分配给拥有不同操作模式的多个组。

为了改变组间共享的通道的操作模式，应用程序必须停止任何对包含指定通道的组的当前转换，然后启动包含指定通道的新组的转换。

为了让应用程序能够在任何时候执行立即转换，就要定义一个按照命令的转换方式。它必须挂起组转换，然后在按照命令的转换活动完成后重新激活它。

3.3 DIO 驱动

DIO 驱动提供基于端口和通道的、对内部通用 I/O 断点的读和写访问。这里的读和写并不被缓冲。这个驱动的基本行为是同步的。

DIO 驱动提供了用于对下列设施进行读、写的服务：

- DIO 通道（引脚）
- DIO 端口
- DIO 通道组

这些服务的行为是同步的。该模块工作于引脚和端口上，由 PORT 驱动来对它进行配置。因此，在 DIO 驱动里面就没有对该端口结构进行配置和初始化。

端口驱动模块：

很多端口和端口引脚是由端口驱动模块分配给各种功能的，比如

- 常规 I/O
- ADC
- SPI
- PWM

DIO 驱动抽象了对微控制器硬件引脚的访问。此外，它能够对这些引脚进行分组。

DIO 驱动提供以下服务：

- 一个一个地修改端口或通道组的输出通道的等级。
- 一个一个地读取端口或通道组的输入和输出通道的等级。

DIO 驱动中的所有读写服务必须是可重入的。理由是：这些 DIO 驱动可以被不同的上层处理程序或驱动程序访问。这些上层模块可以并行的访问驱动。

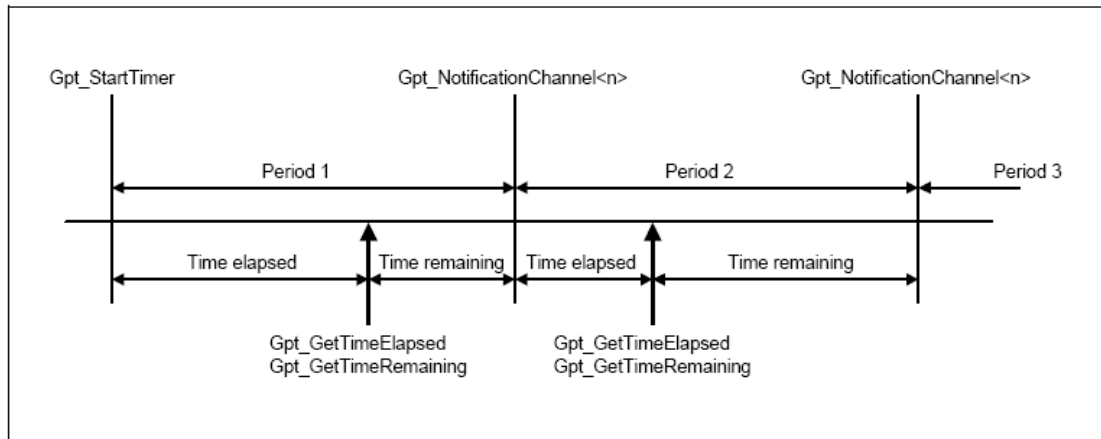
3.4 GPT 驱动

GPT 驱动允许产生单触发或持续的计时器通知。这个模块使用通用计时器的硬件计时通道，因此就为操作系统中或者其它基本软件模块（在这类模块中，OS 警告服务有过多的开销）中的使用提供了精确的、短期的计时。

GPT 驱动提供了用于启动和停止硬件计时模块中的功能计时实例（通道）的服务。它能够产生单个超时周期以及重复超时周期。如果必须调用一个通知，那么当所请求的超时周期过期时，用户就能够对它进行配置。可以在运行时启用或禁用通知。

不管是从上一个通知发生以来的相对时间消耗，还是到下一个通知之间的剩余时间，都

可以进行查询。



注意，GPT 驱动仅产生时间基础，而不服用于时间计数器。这个功能是由另一个模块（ICU 驱动）提供的。

GPT 驱动可以用来唤醒 ECU，不管预定义的超时周期是否过期。模式转换服务将 GPT 驱动在普通操作和睡眠模式之间进行转换。

该驱动不提供超时周期，这些超时周期超过了被时钟源、预定标器和计时寄存器所限制的最大值。用户必须对这个进行处理。

3.5 ICU 驱动

ICU 驱动控制微控制器的输入捕获单元。它提供了下列特性：

- 周期性的、低端的、高端的时间测量
- 边缘检测和通知
- 边缘计数
- 边缘时间戳
- 唤醒中断

对于信号边缘检测来说，需要使用捕获比较单元的边缘检测器或外部时间的中断控制器。

对于信号测量来说，需要一个捕获计时器以及至少一个捕获寄存器。

ICU 调制 PWM 信号，计算脉冲，测量频率和责任（duty）周期，产生简单的中断以及唤醒中断。

ICU 驱动提供以下服务：

- 信号边缘通知
- 控制唤醒中断
- 周期信号时间测量
- 边缘时间戳，可用于获取非周期的信号
- 边缘计数

为了保证数据一致性，应该提供可重入的代码。

时间单元节拍

为了从寄存器值中获得时间，需要知道振荡器频率、预定标器等等。因为这些设置是在 MCU 模块中或其它模块中产生的，不可能计算这些时间。因此，时间和节拍之间的转换是由上层负责的。

3.6 MCU 驱动

MCU 驱动提供用于基本微控制器的初始化，下电，复位和其它 MCAL 软件模块需要的微控制器特定功能的服务。初始化服务提供了灵活性，同时，除了启动代码之外，还提供了应用程序相关的 MCU 初始化。启动代码是完全特定于 MCU 的。MCU 驱动直接访问微控制器硬件，它位于微控制器抽象层（MCAL）中。

MCU 驱动的特征：

- 初始化 MCU 时钟，PLL，时钟预定标器和 MCU 时钟分布
- 初始化 RAM 部件
- 激活微控制器节电模式
- 激活微控制器复位

另外，还有一个服务来从硬件处获得复位的原因。

MCU 驱动微时钟和 RAM 初始化提供 MCU 服务。在 MCU 配置集中，特定于 MCU 的时钟（例如，PLL 设置）和 RAM（例如，段基地址和大小）设置必须被配置。

3.7 端口驱动

该模块初始化微控制器的整个端口结构。很多端口和端口引脚可以被分配给不同的功能，比如：

- 通用 I/O
- ADC
- SPI
- SCI
- PWM

由于这个原因，必须对这个端口结构进行总的配置和初始化。这些端口引脚的配置和使用是依赖于微控制器和 ECU 的。

该模块应该提供用于初始化微控制器的整个端口结构的服务。很多端口和端口引脚可以被分配给各种不同的功能。由于这个原因，必须有该端口结构的全部配置和初始化。这些端口引脚的配置和模式是依赖于微控制器和 ECU 的。

该端口驱动模块应该完成端口结构的全部配置和初始化，该端口结构是用在 DIO 驱动模块中的。因此，DIO 驱动工作在引脚和端口之上，由端口驱动对它进行配置。端口和端口引脚的配置顺序是由配置工具负责的。

端口驱动应该在使用 DIO 功能之前进行初始化。否则 DIO 驱动会产生未定义的行为。

端口访问的原子性：端口驱动应该通过使用原子指令或者利用 OS 的中断屏蔽功能来提供对端口的原子访问。

3.8 PWM 驱动

每个 PWM 通道都连接到一个属于微控制器的硬件 PWM 上。该驱动提供了初始化和控制微处理器内部的 PWM 的服务。PWM 模块产生有不同脉冲宽度的脉冲。

3.9 SPI 处理程序/驱动

SPI 总线是一种主从多节点总线系统，主节点设置片选（CS）来选择一个从节点来进行数据通信。SPI 有一个 4 线的同步串行接口。使用片选线来激活数据通信。

下列 SPI 模块提供基于通道的对 SPI 总线上的不同设备的读、写和传输访问。SPI 通道代表数据元素（8 到 16 比特）。这些通道可能是顺序组合的，不能够被中断。通道有一个

静态配置定义的波特率、片选等等。SPI 设备通常由所使用的 SPI 硬件单元和相关的片选线来标识。这个模块能够作为 SPI 主节点来使用。

这个软件模块的功能范围应该是可静态配置的，以尽可能多的适应每个 ECU 的时间需要。那就是说，比如同步的、异步的、或者两者都有的 SPI 访问都可以存在于 ECU。因此，两个 SPI 驱动可以存在，但仅有一个处理接口。

SPI 处理程序/驱动提供了一些服务来对通过 SPI 总线连接的设备进行读写。它提供了所需的机制来配置片上 SPI 外设。

单片式的 SPI 处理程序/驱动包含处理和驱动功能。它的主要目标是充分利用每个微控制器的特性，使得依赖于静态配置的实现最优化，以尽可能多的适应 ECU 的需要。

3.10 看门狗接口

内部看门狗驱动控制 MCU 的内部看门狗计时器。它提供触发器功能和模式选择服务。
外部看门狗驱动

外部看门狗驱动控制外部硬件看门狗。它提供触发器功能和模式选择服务。它有和内部看门狗驱动一样的功能作用域。

如果在一个 ECU 内使用了多于一个的看门狗设备和看门狗驱动（例如，内部软件看门狗和外部硬件看门狗），该模块就使得看门狗管理程序能够选择合适的看门狗驱动，以及看门狗设备。

看门狗驱动接口提供了对下层看门狗驱动的服务的统一访问，比如模式转换和触发。有设备索引选择适当的看门狗设备。看门狗驱动的服务的行为（同步/异步/计时）是受保护的。

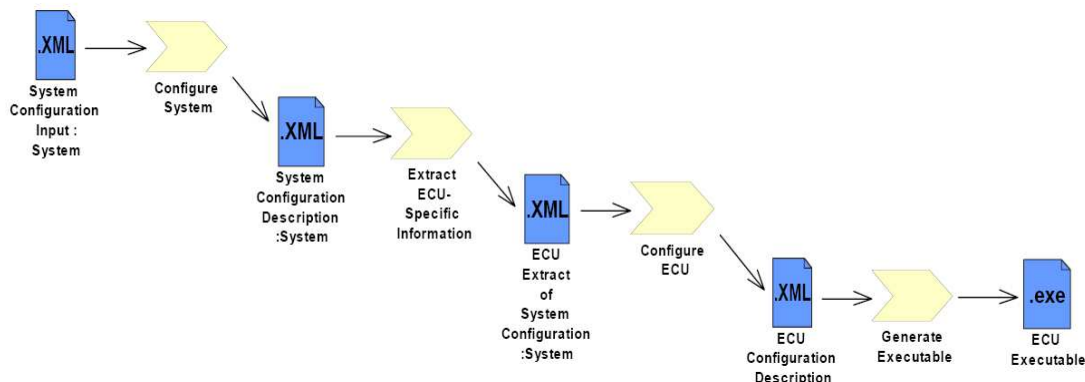
看门狗驱动接口没有给看门狗驱动增加额外的功能。看门狗驱动接口也没有从看门狗属性中进行抽象，比如 toggle 或窗口模式，超时周期等，就是说，该驱动接口没有隐藏下层看门狗驱动和看门狗设备的任何特性。

4 AUTOSAR 方法、模型、工具和一致性测试

4.1 AUTOSAR 方法

AUTOSAR 在系统开发的某些步骤需要通用的技术方法。这一方法就叫“AUTOSAR 方法”。“AUTOSAR 方法”既不是完整的过程描述也不是商业模型，这个方法中并没有定义“角色”和“责任”之类的东西，而且也不规定要执行那些活动。AUTOSAR 方法仅仅是一个“工作产品流”（work-product flow），定义“工作产品流”中活动的相互依赖性。

AUTOSAR 方法并不定义整体的时间线，也并不定义迭代怎样和何时执行。例如在系统设计中，同样的行为（即系统配置行为）会在不同的精确度上重复执行。第一个“粗糙”配置和最后一个“精确”配置依赖于实际配置甚至是 ECU 的实现。



AUTOSAR 方法概述

上图给出了运用 AUTOSAR 方法描述 ECU 从设计到构建、集成的过程。

1. 首先要定义 **System Configuration Input**，选择软、硬件组件，标识系统总体限制，这是系统设计或者体系的任务。AUTOSAR 倾向于通过信息交换格式（软件组件、ECU 资源、系统限制）和模板来减少这些初始系统设计决定的正式描述。所以定义 **System Configuration Input** 就意味着填写和编辑适当的模板。是从头填写模板还是重用模板（可能也需要一些改动）取决于用例。基本上 AUTOSAR 方法允许对模板的高度重用。
2. 活动 **Configure System** 主要是将软件组件映射到关于资源和计时要求的 ECU 上。
3. **Configure System** 的输出是 **System Configuration Description**。这一描述包括所有系统信息（如总线映射、拓扑等）和关于软件组件定位到哪个 ECU 的映射。
4. 活动 **Extract ECU-Specific Information** 从 **System Configuration Description** 中提取特定 ECU 所需的信息。
5. 然后输出到 **ECU Extract of System Configuration**。
6. 活动 **Configure ECU** 为实现添加了所有必需的信息，如任务调度、必需的 BSW（基础软件）模块、BSW 的配置、任务中可运行实体的赋值等。
7. 活动 **Configure ECU** 的结果将输出给 **ECU Configuration Description**，它负责收集所有关于特定 ECU 的局部信息。通过这些信息可以构建该特定 ECU 的可执行软件。
8. 在最后一步中，活动 **Generate Executable** 根据从 **ECU Configuration Description** 中得到的信息生成可执行软件。这一步通常涉及生成代码（如为 RTE 和 BSW 生成代码）、编译代码（编译生长的代码或编译软件组件的源代码）、将所有编译后的代码连接成为可执行软件。
9. 得到可执行 ECU 软件。

在这些简短介绍的 AUTOSAR 方法过程中，同时还需要将软件组件集成为整个的系统，比如生成组件 API，实现组件功能等。虽然这些没有在上图中表现出来，不过软件组件的实现或多或少与 ECU 的配置无关。

4.2 AUTOSAR 模型

4.2.1 起源

AUTOSAR 允许通过对嵌入式控制器和对应软件执行单元组成的分布式系统的各个方

面进行精确的和正式的描述，以建立一个非常灵活却又稳定而可靠的软件工程生命周期。

这个描述的覆盖范围从高层的软件组件的接口要求，到底层的特定总线消息的字节限制。由 AUTOSAR 中的不同工作包决定需要从各种描述中获得的信息。而这些描述就是 AUTOSAR 模型。

AUTOSAR 模型属于 UML2.0 的一个子集，是 UML2.0 元模型的简化。因为 UML2 中高度模块化的结构和对类、属性、关联重定义的过度使用，有时很难在用一两副图展现某个特定方面的同时又保持清晰的可读性。所以，这里简化了 UML2.0 元模型，只包含部分元素。

4.2.2 术语

名词	解释
AUTOSAR 元模型	AUTOSAR 元模型是一种用来定义描述 AUTOSAR 系统的语言的 UML2.0 模型。AUTOSAR 元模型是模板（在 AUTOSAR 中，模板定义了如软件组件和 ECU 之类的结构来创建 AUTOSAR 软、硬件系统）的图形化表示。用 UML2.0 的类图来描述元模型的属性和相互关联。
AUTOSAR 模型	AUTOSAR 模型是 AUTOSAR 元模型的实例。AUTOSAR 模型所包含的信息可以是任何能用 AUTOSAR 元模型表示的内容。AUTOSAR 模型既可以作为文件存储在文件系统中，也可以是一些软件工具所需的 XML 流、数据库或内存。
元数据	元数据包括和数据有关的信息，如创建人、版本、访问权限、时间戳等。
元层	模型和其元模型分别处于不同的元层。标准的元层体系是所谓的四层元模型体系，包含四个元层 M0、M1、M2、M3，在 M0 级的实体用 M2 实体的方式表示，M1 用 M2 实体表示，等等。
反射式元模型	如果元模型可以用自身实体来描述，那么它就可以称为是反射式的。反射式元模型用于终结元模型体系。如，在 OMG 四层元模型体系中，M3 级的元模型就是反射式的。
UML Profile	Profile 可以用来扩展 UML 规范定义的实体集，以适应某个特定领域。在 profile 中定义的新元类称为版类（stereotype）。一般通过增加语义和限制来扩展现有的元类。

4.2.3 元模型体系

完整的 AUTOSAR 模板元模型体系共有五层，如下图所示：

M0 层：AUTOSAR 对象

这是对 AUTOSAR 系统的实现：真实的 ECU 执行包含雨刷控制软件的软件映像。

M1 层：AUTOSAR 模型

这一元层的模型是由 AUTOSAR 终端用户（汽车工程师）构建的。由他们定义名为“雨刷”的软件组件和一系列连接其它软件组件的接口等等。在这一层 AUTOSAR 系统被细分成可重用的组件和特定实例。

M2 层：AUTOSAR 元模型

这一层定义之后将被 AUTOSAR 终端用户使用的“词汇表”，比如，这层定义了在了 AUTOSAR 中有名为“软件组件”的实体和另一个名为“端口”的实体。这些实体之间的联系和语义都属于整个模型的一部分。

M3：AUTOSAR 模板的 UML profile

M2 层的模板是由 M3 层定义的元模型构建的。正如之前讨论过的，这是 UML 加上一个特定的 UML profile，以更好的支持模板建模工作。严格意义上 M2 层上的模板仍然是 UML 的实例，但同时也采用了模板 profile。

M4：元对象设施（meta object facility）

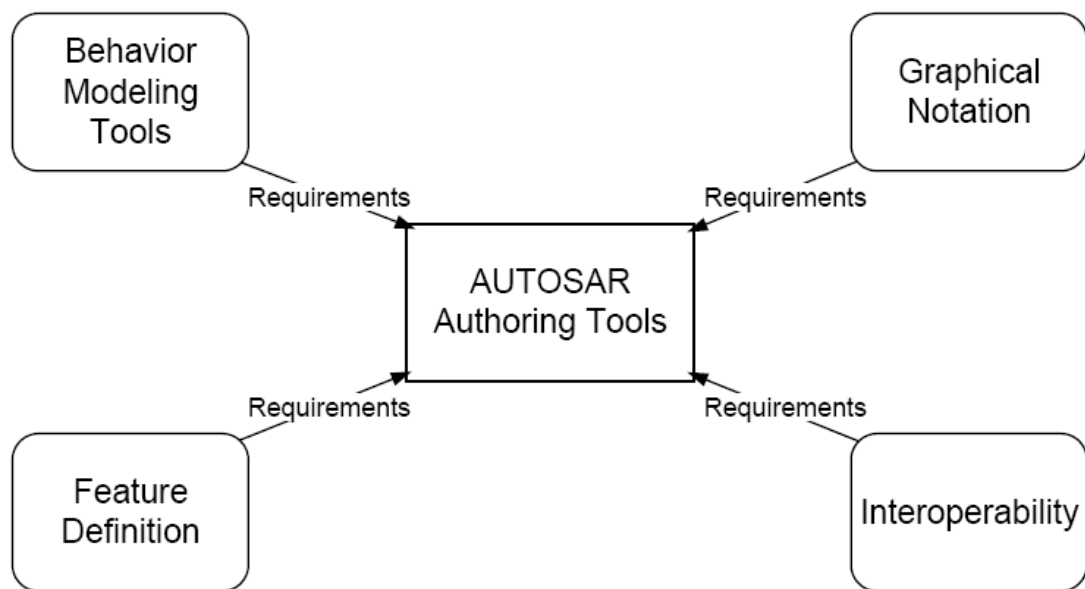
为了概念上的完整性，OMG 将 MOF 放在最后一层元层上。因为 MOF 被定义为是反射式的，所以不再需要进一步的元层。

4.3 AUTOSAR 工具

“AUTOSAR 创作工具”是指所有支持解释、修改、创建用于描述系统的 AUTOSAR XML 描述（AUTOSAR 模型的 XML 表示）的工具。这些模型由以下模板产生：

1. 软件组件模板，
2. ECU 资源模板，
3. AUTOSAR 系统模板。

AUTOSAR 创作工具包含几个重要的方面，如下图所示。



AUTOSAR 创作工具

A 创作工具的特征定义

“创作工具的特征定义”建议逐步实现 AUTOSAR 整体概念中关于交换描述的部分，即软件组件模板、ECU 资源模板、系统模板。在第一次实现的基础上，定义 AUTOSAR 模板子集的 AUTOSAR 创作工具。

B 创作工具的协同工作

“创作工具的协同工作”着重于那些在不同工具间交换 AUTOSAR 模型时可能会出现的问题。本文档首先描述一些数据交换的基本概念，然后简单勾勒出解决这些问题的策略。

C 行为模型的交互

“行为模型的交互”列出了 AUTOSAR 中行为模型的用例。并标识出与行为模型有关的部分 AUTOSAR 元模型。

D 图形符号

“图形符号”为 AUTOSAR 创作工具定义了 AUTOSAR 图形符号。例如，文档为图形建模 CompositionTypes 提供了详尽的图式。这些图形符号应作为实现 AUTOSAR 创作工具的指南。

4.4 一致性测试

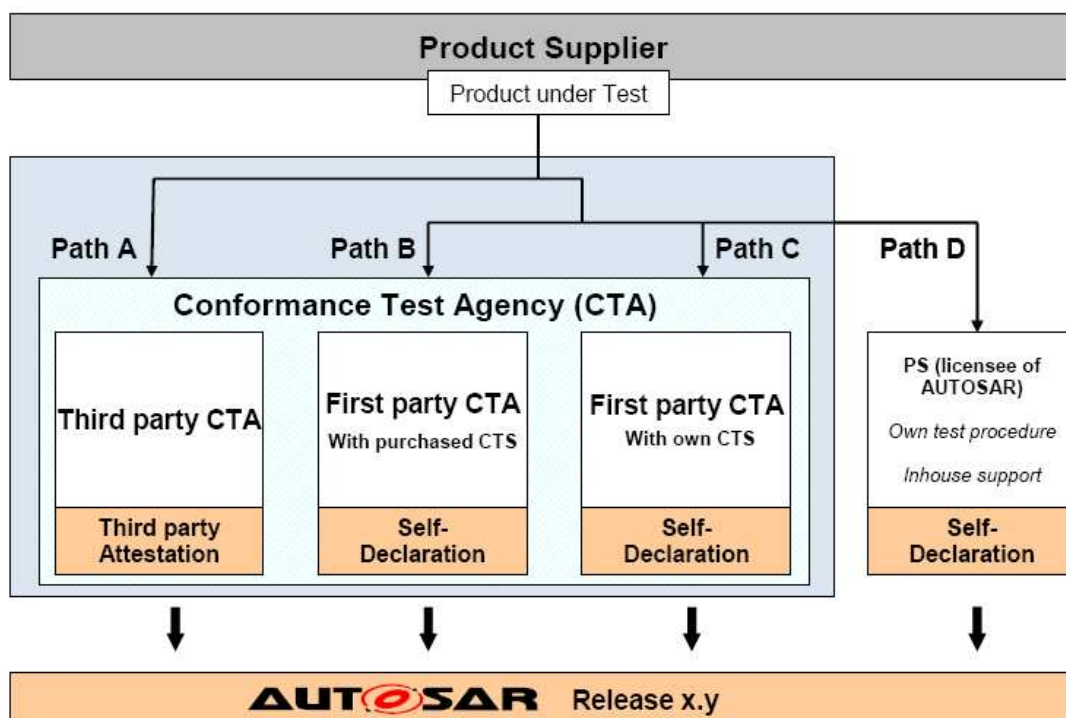
AUTOSAR 一致性测试的目的是为了验证产品是否符合 AUTOSAR 规范。这些产品需要在互操作性、重用/移植性、可扩展性上证明符合 AUTOSAR 标准。

因为 AUTOSAR 是一个开放标准，所以所有最终规范都是标准的一部分。

本文档关注为证明特定产品符合 AUTOSAR 标准所必需的几条相关路径。测试过程的复杂度应尽可能的低，但具体应根据供应商和客户之间的关系来确定最合适的测试方案。

一致性测试中的角色有：

- (1) AUTOSAR（维护标准，监控 AUTOSAR 规范的使用），
- (2) Conformance Test Agency（分为第一方 CTA 和第三方 CTA，主要任务是提供测试包，执行测试，提供支持和证明服务），
- (3) Product Supplier（开发和测试产品）。



5 总结

AUTOSAR 的提出对于汽车电子软件的发展，具有很大的影响，可从以下几个方面看出：

（1） 对于 OEM 的影响

AUTOSAR 的提出对于 OEM 来说最为有利。使得他们对有软件采购和控制有了更灵活和更大的权利，软件的提供商会逐步增多，使得他们又更多的选择；同时，软件系统的开放化，使得软件的质量监督也会相应提高。这些无疑对他们百益而无害。

（2） 对于部件提供商的影响

对于部件提供的影响须从两个角度来分析。从技术角度而言，软件技术的发展对于部件提供商带来的效率提高是毋庸置疑的。增强了软件的移植性，可利用率，可维护性等等，对于软件开发者带来的好处是巨大的。

从商业角度考虑，公开软件标准的提出对于所有软件提供商都是有益的。但同时，理论上讲，以前一些软件提供商的垄断模式随着这一开放式系统的提出而打破。新的市场将是开放的，竞争与机遇共存。传统部件提供商由于其基础积累仍在市场上处于优势，但新的提供商相对于以前将有更多的机会。

（3） 对于芯片提供商的影响

对于芯片提供商的影响是共同的，主要是提供一些支持工作。由于基础软件的开发仍需要芯片提供商的大力支持，因此芯片提供将不余气力的对这一标准提供支持。尽管属于非核心业务，但如果彻底放弃支持，市场份额或许会受到一些影响。