

**Automotive Open System Architecture**

A grey logo on a black background

Description automatically generated

Table of Contents

[AUTOSAR Platform 3](#_Toc184998448)

[Why AUTOSAR? 3](#_Toc184998449)

[Classic AUTOSAR Platform 3](#_Toc184998450)

[AUTOSAR Architecture 3](#_Toc184998451)

[AUTOSAR Methodology 7](#_Toc184998452)

[AUTOSAR Application Interfaces 9](#_Toc184998453)

[Adaptive AUTOSAR Platform 12](#_Toc184998454)

[Adaptive AUTOSAR Architecture 12](#_Toc184998455)

[Briefly Explain All Modules in Adaptive Autosar 15](#_Toc184998456)

[Adaptive Autosar Application Implementation 27](#_Toc184998457)

[Communication between AUTOSAR Platforms 29](#_Toc184998458)

[Salient feature of Adaptive Autosar Platform 29](#_Toc184998459)

[Classic AUTOSAR VS Adaptive AUTOSAR 30](#_Toc184998460)

[Applications of AUTOSAR 31](#_Toc184998461)

[Advantages and Disadvantages of AUTOSAR 32](#_Toc184998462)

[Definitions of AUTOSAR Keywords 32](#_Toc184998463)

**Acronyms and Abbreviations of AUTOSAR**

* **OEM** – Original Equipment Manufacturing
* **TIER 1** – Direct Suppliers to an OEM.
* **BSW** – Basic Software
* **RTE** – Run Time Environment
* **MCAL** – Microcontroller Abstraction Layer
* **SWS** – Software Component
* **CDD** – Complex Device Driver
* **SOA** – Service Oriented Architecture
* **ECU** – Electronic Control Unit
* **ARA** – AUTOSAR Runtime Adaptive Environment
* **POSIX** – Portable Operating System Interface
* **OSEK** – Operating System Embedded Kernel
* **CAN** – Control Area Network
* **API** – Abstract Programming Interface

The **AUTOSAR** standard defines variations of the software architecture called “**AUTOSAR platforms”**.

# AUTOSAR Platform

The AUTOSAR or Automotive Open System Architecture was developed to create a common standardized software architecture for designing automotive electronic control units (ECUs).

AUTOSAR (Automotive Open System Architecture) was founded in 2004 by a consortium of leading automotive manufacturers (OEMs), suppliers, and service providers from the electronics, semiconductor, and software industries.

Improve the reusability and exchangeability of software modules between the OEMs and Suppliers.

## Why AUTOSAR?

A diagram of software

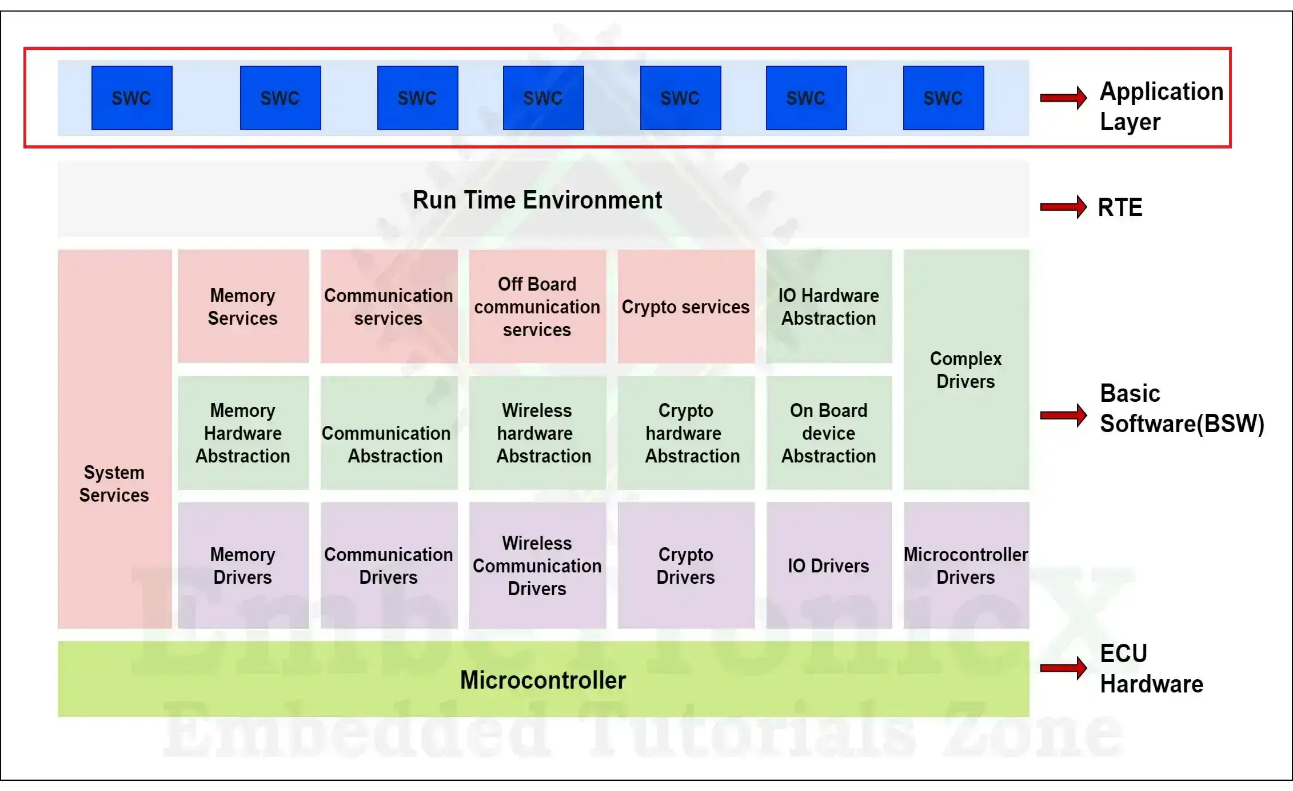
Description automatically generated

## Classic AUTOSAR Platform

There are 3 main working topics in classic AUTOSAR.

### AUTOSAR Architecture

* Architecture provides a structured platform for hardware independent software application with the help of AUTOSAR basic software stack.
* layered architecture, that supports the realization of functional requirement into an ECU.
* Abstract, hardware components from application layer.

**Block diagram of AUTOSAR architecture:**

**The AUTOSAR specifies a three-layer architecture, which are categorized into following modules:**

#### Application layer

* The Application layer is the first layer of the AUTOSAR software architecture and supports custom functionalities implementation.
* The AUTOSAR application layer includes various application specific software components that are designed to execute specific set of tasks, as per the use-cases.

**Block diagram of application layer:**A black and white text

Description automatically generated

Note: To explore the more about application layer, click on below link.

[https://www.embitel.com/blog/embedded-blog/decoding-the-component-concept-of-the-](https://www.embitel.com/blog/embedded-blog/decoding-the-component-concept-of-the-application-layer-in-autosar) [application-layer-in-AUTOSAR](https://www.embitel.com/blog/embedded-blog/decoding-the-component-concept-of-the-application-layer-in-autosar)

#### Runtime environment (RTE)

* The RTE layer acts as a middleware between the AUTOSAR application layer and the lower layers. Basically, the RTE layer manages the inter- and intra-ECU communication between the Application layer components as well as the BSW layer components.
* Provides the communication and scheduling services to application software’s.
* The main purpose of RTE is to make application software independent of ECU.

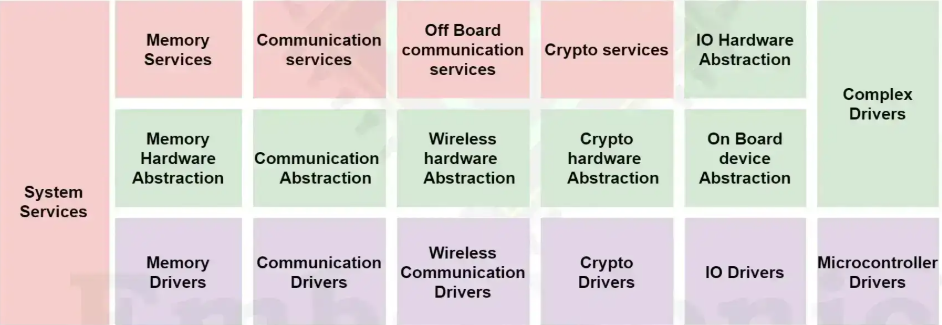
**Block diagram of RTE layer:**



#### Basic software (BSW)

Responsible for managing hardware resources and providing common services for application software.

**Block diagram of basic software layer:**



**The AUTOSAR Basic Software is further divided in the layers:**

##### Services layer

* The service layer is the topmost layer of the AUTOSAR Basic Software Architecture.
* Provides background services for applications, RTE, BSW modules.
* The main work of the services layer is to provide services from the application layer to the microcontroller layer.
* The service layer constitutes an operating system, which runs from the application layer to the microcontroller layer. The OS has an interface between the microcontroller and the application layer and can schedule application tasks.
* The service layer in BSW is responsible for services like network services, memory services, diagnostics service, communication service, ECU state management, and more.

##### ECU Abstraction layer

* The ECU abstraction layer interface the drive of the micro controller abstraction layer. it also contains driver for external devices.
* It offers an API for access to peripherals and devices regardless of their location (Microcontroller internal/external) and their connection to the microcontroller (port pins and type of interface).
* This layer and its drivers are independent of the microcontroller and dependent on the ECU hardware and provide access to all the peripherals and devices of ECU, which supports the functionalities like communication, memory, I/O, etc.
* ECU abstraction layer makes the higher levels independent of ECU hardware.
* ECU abstraction layer provides the uniform to all functionalities of ECU. Block diagram of ECU abstraction layer in BSW.

##### Microcontroller Abstraction layer

* The Microcontroller Abstraction Layer is the lowest software layer of the Basic Software. It contains internal drivers, which are software modules with direct access to the µC and internal peripherals.
* Provides direct access to all on-chip microcontroller peripheral and external devices which are mapped to memory.
* The Microcontroller Abstraction Layer makes the application and basic software layer independent of the underlying hardware platform.
* Every microcontroller has its own MCAL derivers.

#### Detailed Diagram of AUTOSAR Layered Architecture

#### autosar_layered_architecture_details

### AUTOSAR Methodology

**AUTOSAR methodology divided into 5 types below here explained in detail:**

#### System Extract

The combination of all ECU information’s is called as “system extract”.

#### ECU Extract

ECU extract file contain the 1 ECU information and ECU contains the Frame details, PDU details, received frame details, transmit frame details, Count of signals, Length of the signal, PDU length, Frame length this all details will available inside ECU extract file.

#### PDF

PDF file nothing but a Parameter Definition File. PDF Is the template for this configuration parameters which is nothing, but a declaration of containers and parameters will be there inside the PDF ARXML files. All the modules information will be provided by the AUTOSAR documentation. Based on the information will be configured accordingly.

A screenshot of a computer

Description automatically generated

#### BSW MD

BSW MD file nothing but it’s containing the Vender ID (vender id is nothing but, in our implementation, a particular module will change any values inside the PDF’s that PDF’s called as a Vender ID), Static and Dynamic code versions, AUTOSAR version, Code Generated version, in a module how many API’s are available and what is main function derived each module. So, all this details information available inside the BSW MD file.

A computer screen shot of a computer code

Description automatically generated

#### ECU Discerption

ECU discerption is nothing but an if you provide the actual values of the containers or parameters this information will be there in ECU Discerption file. After compilation of this process will get .h and .C or C++ dynamic files.

The process of conversion from the System Extract to ECU Discerption is called as a “AUTOSAR METHODOLOGY”.

### AUTOSAR Application Interfaces

The AUTOSAR interface will be helpful to make a communication between the AUTOSAR software components.

**AUTOSAR Interface block diagram:**

A diagram of software components

Description automatically generated

**The AUTOSAR interface of the basic software are grouped in 3 categories:**

#### AUTOSAR Interface

* AUTOSAR interface will be used in the Application layer.
* The main role of AUTOSAR interface to make a communication of between the software components at the same time it will make a communication between the BSW as well.
* The AUTOSAR interface will communicate via RTE.
* AUTOSAR Interfaces of software components are defined by ports and port interfaces.
* The SWS interact with other components only through AUTOSAR interfaces.
* Relevant of modeling of software components.
* Definition of information exchanged between the software components.

**AUTOSAR Interface block diagram:**

A close up of a sign

Description automatically generated

#### Standardized AUTOSAR Interface

* The Standardized AUTOSAR Interface will be used in the system services.
* This can be called via RTE from the Application layer
* This typically used for services in BSW layer.
* AUTOSAR services interact with other software components through a standardized AUTOSAR interface.
* Syntax and semantics are standardized (C-API).
* Used to define AUTOSAR services.

**EX:** Service Name: Dem\_setEventStatus

Syntax: Std\_ReturnType Dem\_setEventStatus(Dem\_EventIdType EventId, Dem\_EventStatusType EventStatus)

A close-up of a diagram

Description automatically generated**Standardized AUTOSAR Interface block diagram:**

#### Standardized Interface

* The Standardized Interface will not use for software components.
* It will used within the BSW layer.
* Only C-API is Standardized.
* Standardized Interface are typically used between the software modules which are always on the same ECU.
* Standardized Interface cannot use communication between the ECU to ECU.

**Standardized Interface block diagram:**

A diagram of a software

Description automatically generated

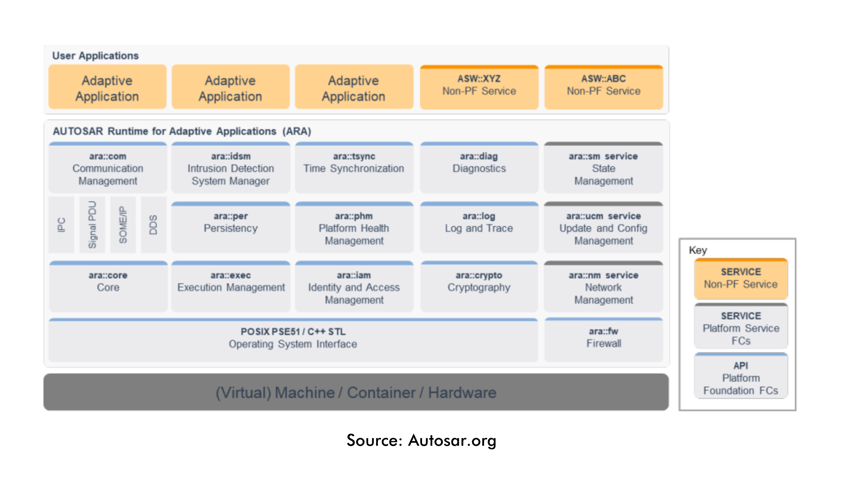
## Adaptive AUTOSAR Platform

* The Adaptive AUTOSAR Platform is an extension of the classic AUTOSAR standard that provides a framework for developing complex software systems for the next generation of connected and autonomous vehicles.
* Adaptive AUTOSAR is designed to provide developing automotive software that can be support a wide range of use cases from simple infotainment systems to complex autonomous driving applications.
* Adaptive AUTOSAR platform is the emergence of automotive ECUs that not only communication within the in-vehicles network but also communication outside the vehicles.
* Internally AUTOSAR only consists of classic AUTOSAR then later as the need changes with increasing technology then later a new architecture was developed it is called as "Adaptive AUTOSAR " It was developed in 2018.
* Adaptive platform will not replace classic AUTOSAR platform or non-AUTOSAR platform. rather it will interact with these platforms and external backend system such as roadside infrastructures form integrated system. like traffic lights and v2x vehicles.
* The AUTOSAR Adaptive platform is a standardized architecture for high-performance computing ECUs to build safety systems such as highly driving, autonomous systems, more powerful and more flexible E/E architectures in the vehicle.
* The big advantage is Adaptive ECUs make it possible to update applications over a vehicle’s entire life cycle and add new software functions at a later time.

### Adaptive AUTOSAR Architecture

* The Adaptive Platform is a distributed computing and service-oriented architecture (SOA).
* The platform provides high-performance computing, message-based communication mechanisms, and flexible software configuration for supporting applications, such as automated driving and infotainment systems.

**Adaptive AUTOSAR Architecture Block Diagram:**



**The AUTOSAR specifies a 4-layers architecture, which are categorized into following modules:**

#### Application layer

* The adaptive applications can be integrated at runtime. This implies that different software can be developed and distributed for an ECU, completely independent of each other. At the heart of an AUTOSAR adaptive platform, there is an POSIX operating system, and each adaptive application is implemented as a process in this OS.
* To ensure the communication between local applications and applications on other ECUs including the interaction with the Adaptive platform services, middleware protocols must be defined. The most noticeable changes in the use of AUTOSAR Adaptive are the universal use of Ethernet based communication systems.
* In the Adaptive platform, the applications utilize the “AUTOSAR Runtime for Adaptive Applications,” also known as ARA. This runtime environment gives users standardized interfaces to efficiently integrate different applications into the system.
* The Adaptive platform now offers the option of removing, updating, or adding individual applications at run-time.
* An Adaptive application contains at least one executable. in order to be deployable on different
* adaptive platform, it is only using ARA programming interface. They are implemented by one or several executables.

#### AUTOSAR Runtime for Adaptive Application (ARA)

* One of the core features of this adaptive platform is called AUTOSAR Runtime for Adaptive Applications (ARA). ARA gives users all the interfaces and infrastructure needed to communicate and execute adaptive applications into the system and allows data exchange between ECUs regardless of their internal architectures. In addition, this runtime offers direct access to the operating system functions known as the “Minimum Real time system Profile” (PSE51).
* The module operating system interface based on a subset of POSIX is responsible for run-time resource management such as signals, timer and thread handling for all adaptive applications and functional clusters that establish the platform.
* ARA offers mechanisms for ECU-internal and inter-network communications as well as access to basic services such as diagnostics and network management. The adaptive AUTOSAR applications are formed in software components that communicate via services. These services may be requested or provided. In addition, the application programmer can directly access a subset of operating system functions. In terms of communication, the AUTOSAR Adaptive defines a new feature called ara::com. ara::com is a standard C++ API based on SOA more specifically based on SOME/IP.
* ARA works in a different way with service or client based dynamically to improve the responsiveness, reliability and portability feature.
* A set of standard application interface provided by functional clusters, which belong to either adaptive platform foundation or Adaptive platform services.

#### Functional Clusters

* The software of the Adaptive Foundation and Adaptive Services is presented in the form of functional clusters.
* **Adaptive platform foundation:** it is part of an adaptive platform implementation which provides standardized platform functionality to applications via software interface (APIs). which are required without this cannot implemented Adaptive Autosar Platform.
* **Adaptive platform service:** standard platform services that is provided by an application which is part of Autosar platform implementation may or may not require services. based on requirements we can use it.
* Their main purpose is to offer functionalities in the form of services to the applications.

#### Operative System

* The Adaptive Autosar Platform uses POSIX based OS so here it provides the standard interface for the communication between the application and OS.
* Normally in our classic Autosar the application component was connected to the RTE so all the work the scheduling and everything was done using RTE only. But new thing in Adaptive Autosar is each application contains the OS for creation a process for that. for example, we have 4 applications inside our Adaptive Autosar so 4 processors will be created inside OS and priority will be assigned to them.

A screenshot of a computer

Description automatically generated

* The module operating system interface based on a subset of POSIX is responsible for run-time resource management such as signals, timer and thread handling for all adaptive applications and functional clusters that establish the platform.
* It is one of the pre-requisites of using complex processors and offers developers the necessary building blocks to create high-performance automotive applications to run on different systems without modification.
* The POSIX standards cover a wide range of functionalities, including file I/O, process management, thread management, inter-process communication, and more. It was originally developed by the IEEE (Institute of Electrical and Electronics Engineers) as IEEE Std 1003.

#### Machine

* The new term, “Machine,” given by Adaptive AUTOSAR is, in simple terms, the entity where the software runs. It is understood that this Machine can be of virtual nature. The actual hardware can in turn host one or multiple Machines.

### Briefly Explain All Modules in Adaptive Autosar

#### COM

The **COM (Communication)** module in Adaptive AUTOSAR facilitates **Service-Oriented Communication** between different software components or applications. Unlike the Classic AUTOSAR COM module, which is signal-based, the Adaptive AUTOSAR COM module is designed for **service-oriented communication** using protocols like **SOME/IP**.

**Key Responsibilities of the COM Module:**

**Service-Oriented Communication**:

* Manages communication between services and applications on different ECUs.
* Uses service descriptions to enable discovery and interaction between components.

**Protocol Abstraction**:

* Provides a protocol-agnostic API for applications to use.
* Internally supports protocols like **SOME/IP** for communication.

**Service Discovery**:

* Handles **service registration** and **discovery** using mechanisms like **SOME/IP Service Discovery (SD)**.
* Ensures that applications can dynamically discover and connect to available services.

**Data Serialization/Deserialization**:

* Serializes data (application-specific data structures) into a format suitable for transmission.
* Deserializes received data back into the original structure.

**Runtime Binding**:

* Supports dynamic binding of services and clients during runtime.
* Allows components to communicate flexibly in a dynamic system environment.

**How the COM Module Works:**

1. **Service Description**:

* Each service is defined by a **Service Interface**, which specifies:
* **Methods:** Functions exposed by the service (e.g., GetSpeed()).
* **Events:** Notifications sent by the service (e.g., SpeedExceeded).
* **Fields:** Data values that represent the state of the service (e.g., currentSpeed).

1. **Service Discovery**:

* When an application (client) starts, it queries the network to discover available services.
* The COM module uses **SOME/IP-SD (Service Discovery)** to locate and bind to the desired service.

1. **Communication**:

* Once the service is discovered, the COM module facilitates communication between the client and service provider:
* **Request/Response** for method calls.
* **Publish/Subscribe** for event notifications.

1. **Data Transmission**:

* The COM module serializes the data using a serialization protocol (e.g., **SOME/IP serialization**) before sending it over the network.
* At the receiver's end, the data is **deserialized** back into a usable format.

#### EM

The **Execution Management (EM)** is responsible for managing **lifecycle** and **execution states** (**starting, stopping, and monitoring**) of applications and services on the Adaptive AUTOSAR platform.

**Key Points:**

**Lifecycle Management**:

* EM controls when applications start and stop.
* It ensures services are running before dependent applications start.

**State Management**:

* Tracks and manages the states of individual applications, such as **Running**, **Stopped**, **Suspended**, etc...

**Error Handling**:

* Restarts applications or services if they fail.

**Dependency Handling**:

* Starts applications in the right order, based on their dependencies.

**Dynamic Management**:

* Supports dynamic addition, removal, or update of applications at runtime in some implementations.

**How EM Works:**

1. **Application Configuration**:

* Applications declare their lifecycle requirements in the **Manifest File** (part of Adaptive AUTOSAR configuration).

1. **Startup**:

* At system boot, the EM reads the configuration, initializes platform services, and starts the applications based on dependencies.

1. **State Transitions**:

* EM manages transitions between states like **Stopped**, **Initializing**, **Running**, and **Terminated**.

1. **Fault Monitoring and Recovery**:

* If an application fails, the EM can attempt to restart it or notify the platform of a critical failure.

#### SM

The **State Management (SM)** module in Adaptive AUTOSAR manages the **system's operational** **modes** and **states** to keep everything running smoothly. It ensures coordination between applications, services, and platform states to keep the system functioning efficiently.

**Key Points:**

**Manages System States:**

* Handles states like Startup, Running, Shutdown, and Error.

**Synchronizes Applications:**

* Ensures all applications follow the system state.

**Example:** Apps start only when the system is in Running state.

**Manages Modes:**

* Handles modes like Normal Mode or Low Power Mode for efficient operation.

**Handles State Transitions:**

* Smoothly transitions between states (e.g., from Startup to Running).

**Error Handling:**

* Switches to Error state when something fails.

**How EM Works:**

**1. Startup:**

* SM initializes the system by starting necessary services and applications.

**2. Running:**

* Once ready, the system moves to the Running State where applications and services operate normally.

**3. Error Handling:**

* If something goes wrong (e.g., a service fails), SM detects it and takes action like restarting the service or notifying the system.

**4. Shutdown:**

* When it's time to turn off, SM ensures all services and applications stop safely.

#### Core

The **ARA::CORE module** in Adaptive AUTOSAR is responsible for managing core functionalities such as **memory management**, **thread scheduling**, and **resource allocation** across the system. It acts as the underlying framework that enables applications and services to interact with the hardware and system resources.

**Key Functions:**

**Memory Management**:

* **Allocates and frees memory** for applications to use, ensuring there's no memory wastage.

**Thread Scheduling**:

* **Decides which tasks** (threads) should run and when, ensuring fair use of the system’s processing power.

**Resource Allocation**:

* **Manages system resources** (like CPU, memory) to avoid conflicts between applications.

**Task Synchronization**:

* Ensures that tasks or applications can work together without interfering with each other.

**Inter-Process Communication (IPC)**:

* Allows different applications to exchange information, like sending signals or data.

**How ARA::CORE Works:**

1. **Application Initialization**:

* When the system starts, **ARA::CORE initializes** memory, resources, and threads for running applications.

1. **Running Applications**:

* During operation, **ARA::CORE manages** tasks by deciding which thread gets CPU time.
* It also manages **resource allocation** to ensure efficient operation of both foreground and background tasks.

1. **Handling Communication**:

* ARA::CORE facilitates **communication** between apps, ensuring they can exchange necessary data (like audio or navigation commands).

1. **System Shutdown**:

* Before shutting down, ARA::CORE **frees memory** and **releases resources**, ensuring a clean exit and preventing memory leaks.

#### OS

In **Adaptive AUTOSAR**, the **Operating System (OS)** is responsible for managing the **hardware resources** and **running the applications** in the system. It provides the environment in which all the software applications and services run.

**Key Functions:**

**Task Scheduling**:

* Decides which application or task should run at any given time, ensuring that everything gets a chance to run without interruptions.

**Memory Management**:

* Allocates and frees memory for applications, making sure there’s enough space for them to run properly.

**Interrupt Handling**:

* Reacts to urgent events from hardware, like sensor data or user inputs, and makes sure the system responds quickly.

**Communication**:

* Allows different applications to send messages or data to each other, ensuring smooth interaction between them.

**Resource Protection**:

* Ensures that multiple applications don’t try to use the same resources at the same time, preventing crashes or conflicts.

**Time Management**:

* Keeps track of time and makes sure tasks that need to run periodically (like checking sensors) happen on time.

**How OS Works:**

1. **System Initialization**:

* The OS initializes hardware components, allocates memory for system tasks, and configures scheduling policies.

1. **Task Execution**:

* The OS starts execution of various applications, scheduling them according to their priorities. Applications like navigation, media, diagnostics, and communication services run concurrently but are allocated CPU time as per OS rules.

1. **Handling Events**:

* The OS reacts to events such as hardware interrupts, sensor data, or user inputs by prioritizing and managing responses (e.g., launching safety features in response to sensor data).

1. **Shutdown**:

* Before the system shuts down, the OS gracefully stops all running tasks, frees resources, and ensures that no critical data is lost.

#### DIAG

The **Diagnostic (DIAG) Module** in **Adaptive AUTOSAR** is responsible for **detecting** and **managing faults** in the vehicle's electronic systems. It helps in finding issues, reporting them, and allowing repairs.

**Key Functions:**

**Fault Detection**:

* It identifies problems in vehicle systems, like a sensor failure, and generates fault codes (DTCs).

**Communication with Diagnostic Tools**:

* It allows external tools (like OBD-II scanners) to read fault codes and interact with the vehicle’s systems for troubleshooting.

**Fault Memory**:

* It stores detected faults and can clear them once the issue is fixed.

**Read and Clear Fault Codes (DTCs)**:

* It lets technicians check and reset the fault codes in the system after repairs.

#### PHM

**PHM (Platform Health Management)** in **Adaptive AUTOSAR** is a system that **helps monitor the health** of vehicle components and **predict possible failures** before they happen. It enables proactive maintenance, reducing the chances of unexpected breakdowns.

**Key Functions of PHM:**

**Health Monitoring**:

* Continuously monitors the condition of critical vehicle components (like sensors, actuators, etc.) to detect any signs of degradation or abnormal behavior.

**Fault Prediction**:

* Uses data and algorithms to predict potential future failures or performance issues in components, helping to schedule maintenance before a failure occurs.

**Data Analysis**:

* Collects and analyzes data from vehicle sensors to assess the overall health and performance of the system.

**Maintenance Recommendations**:

* Provides recommendations or alerts about when a component may need servicing or replacement based on predicted failures or wear.

**Lifecycle Management**:

* Helps in tracking the lifecycle of components, ensuring they are replaced or serviced at the right time to maintain vehicle performance.

**How PHM Works:**

1. **Monitoring**:

* PHM constantly checks the condition of important vehicle parts, like sensors or the engine, to ensure they are working correctly.

1. **Data Collection**:

* It collects data from sensors that measure things like temperature, pressure, and speed to assess the health of different components.

1. **Analysis and Prediction**:

* Using algorithms, PHM analyses the collected data to detect patterns that might indicate a future problem, such as wear or degradation in a part.

1. **Alerts and Recommendations**:

* If PHM predicts a potential failure, it sends alerts or maintenance recommendations, so the issue can be addressed before it leads to a breakdown.

**Example:**

If PHM detects that the brake system's wear is reaching a critical level, it might predict that the brakes will soon need replacing. It will send a warning to the driver or maintenance team to replace the brakes, preventing a failure.

#### Crypto

The **Crypto module** in **Adaptive AUTOSAR** is used **to protect data** and ensure secure communication between vehicle systems. It performs tasks like encryption, decryption, and verifying data integrity to keep sensitive information safe.

**Key Functions:**

**Encryption/Decryption**:

* Protects data by converting it into unreadable form and then back into readable form only for authorized systems.

**Hashing**:

* Creates a unique code for data to check if it has been changed during transmission.

**Digital Signatures**:

* Verifies the source of data to ensure it comes from a trusted sender.

**Key Management**:

* Handles and stores keys used for encryption and decryption securely.

**How it Works:**

* When two vehicle systems need to communicate, the Crypto module encrypts the data to keep it secure.
* The receiving system decrypts the data using a key to read it.
* The module also ensures that data is not altered during transmission using hashing and can verify the sender’s identity with digital signatures.

**Example:**

If the car’s infotainment system needs to send a secure message to another system, the Crypto module encrypts the message before sending it. The other system decrypts it to read, ensuring the data is secure and hasn’t been tampered with.

#### UCM

**Update and Configuration Management (UCM)** in **Adaptive AUTOSAR** is responsible for managing **software updates** and ensuring the vehicle’s software is correctly configured.

**Key Functions:**

**Software Updates**:

* Keeps vehicle software up-to-date by managing updates for ECUs (Electronic Control Units).

**Configuration Management**:

* Ensures the correct configuration of software and settings for different vehicle systems.

**Version Control**:

* Tracks and manages software versions to ensure compatibility.

**Integrity Checking**:

* Verifies that software updates are not tampered with and are applied correctly.

**Error Handling**:

* Deals with issues during updates and ensures recovery if something goes wrong.

**How it Works:**

* When an update is needed, UCM downloads, checks, and applies it to the system.
* It ensures only compatible software is used and can roll back to a previous version if there are any issues.

**Example:**

If a new version of the vehicle’s navigation system is available, **UCM** will update the system, verify it’s safe, and ensure it’s working correctly.

#### LOG

**Logging in Adaptive AUTOSAR** is a way to record important information about the system’s activities, errors, and events. This helps developers and engineers to track what’s happening in the vehicle’s software, identify problems, and improve system performance.

**Key Functions:**

**Record Events**: Logs actions or events happening in the system.

**Error Tracking**: Records errors or issues that occur.

**Monitor Performance**: Tracks how well the system is performing.

**Security**: Logs security-related events to protect the system.

**How it Works:**

* The system creates logs whenever something important happens, such as an error or a change in state.
* These logs are stored and can be reviewed later to understand what happened or fix problems.

**Example:**

If a sensor fails in the vehicle, the system will log an error with details about the failure. This helps engineers fix the issue quickly.

#### PER

**Persistency in Adaptive AUTOSAR** means **saving important data** so that it stays even when the **system is turned off** or **restarted**. This ensures that things like settings, configurations, or user preferences are not lost and can be used again when the system is back on.

**Key Points:**

**Data Saving**: Data is saved in memory (like flash or EEPROM) so it remains after a reboot.

**System Continuity**: It keeps important settings or states intact across reboots.

**User Experience**: It helps restore user settings, like volume or language, when the system restarts.

**How it Works:**

* Data is saved to non-volatile memory before the system shuts down.
* When the system restarts, the saved data is loaded back into memory to continue from where it left off.

**Example:**

If you adjust the volume in your car’s infotainment system, **Persistency** ensures that the volume setting stays the same even after turning the car off and on again.

#### TSync

**Time Sync** in **Adaptive AUTOSAR** ensures that all parts of a system (like different ECUs in a car) have the same, accurate time. This is important for tasks that need to happen at the right time, such as coordinating systems or recording events.

**Key Points:**

**Same Time Across Systems**: Ensures that all ECUs in the vehicle share the same time.

**Accurate Timing**: Helps in tasks like controlling systems or triggering actions at the correct time.

**Protocols Used**: Uses methods like **NTP** (Network Time Protocol) to synchronize time.

**How it Works:**

* One ECU or device provides the correct time (master time source).
* Other ECUs synchronize their clocks to this master time, ensuring they are all in sync.

**Example:**

If your car’s braking system needs to coordinate with other systems, **Time Sync** makes sure that all ECUs involved use the same time, so everything happens at the right moment.

#### Manifest Files In Adaptive Autosar

##### Execute Manifest file

This provides the details of what is the name of executable, when it should be executed, when it should be started and stop all those details are present inside file.

**Example:**

{

"ExecutionManifest": {

"Applications": [

{

"name": "App1",

"Executable": {

"FilePath": "/path/to/app1\_binary",

"StartTime": "Boot", // The app starts when the system boots up

"StopTime": "Shutdown", // The app stops when the system shuts down

"Permissions": "read-write-execute" // Permissions for the app

},

"Resources": {

"CPU": "2 cores",

"Memory": "512MB"

}

}

]

}

}

##### Machine Manifest file

This provides the details of hardware configuration, such as CPU, memory, and network setup.etc.

{

"MachineManifest": {

"MachineConfig": {

"MachineName": "Machine1",

"CPU": {

"CoreCount": 4,

"Type": "ARM Cortex-A72"

},

"Memory": {

"RAM": "8GB",

"ROM": "128GB"

},

"Network": {

"IP\_Address": "192.168.1.100",

"Ports": ["8080", "5000"]

}

}

}

}

##### Service Instance Manifest

* **Purpose**: Configures **service-oriented communication** between applications or components.
* **What it contains**: Details about how services interact with each other and the transport protocols used for communication (e.g., Ethernet, CAN, etc.).

**Example:**

This manifest might define how a specific service communicates over Ethernet or another protocol, including port numbers and communication requirements.

{

"ServiceInstanceManifest": {

"ServiceInstances": [

{

"name": "DataService",

"TransportProtocol": "Ethernet",

"Communication": {

"Port": 5000,

"Protocol": "TCP"

},

"ServiceBehavior": {

"StartOnDemand": **true**,

"Timeout": "30s"

}

}

]

}

}

### Adaptive Autosar Application Implementation

To run an application in Adaptive AUTOSAR Platform, during runtime (**ARA**) several functional clusters are essential, including the Execution Management (**EM**), State Management (**SM**), **Log and Trace**, **Core**, and Operating System (**OS**).

**let's explain brief about all functional clusters:**

* **OS:** The operating system interface provides functionality for implementing multi-thread real time embedded applications and corresponding to the POSIX PSE51 profile. The OS starts first because it contains all the application-related processors and executables. Once the OS is up and running, it boots up the EM.
* **Core:** It provides functionality for initialization and de-initialization of the Autosar runtime for Adaptive applications as well as termination of processes.
* **Log and Trace:** It provides interface for applications to forward logging and tracing information onto the communication bus, the console, onto the file.
* **Execution Management (EM):** The element of the Autosar adaptive platform responsible for the order **startup** and **shutdown** of the **Autosar Adaptive platform** and **Adaptive applications**. EM will start based on Execute manifest and Machine manifest files.
* **Execute manifest file:** This provides the details of what is the name of executable when it should be executed, when it should be start and stop all those details are present inside file.
* **Machine manifest file:** This provides the details of system memory, Ram, Rom, Ip Address, ports etc.
* **EM Startup Process in Adaptive AUTOSAR** The OS starts first because it contains all the application-related processors and executables. Once the OS is up and running, it boots up the EM. The EM reads the Execute Manifest and Machine Manifest files. Based on these manifests and the dependency descriptions, the EM determines the startup order of applications, and their processes and EM starts other platform foundation and service clusters. After initializing the necessary platform services, the EM starts the user applications.

A blue rectangular box with white text

Description automatically generated

* **State Management (SM):** The element defining modes (Startup, Running, shutdown) of operation for Autosar Adaptive platform.it allows flexible definition of functions which are active on the platform at any given time.
* **Ex:** SM knows when to change from startup to driving and when to change from driving to parking so that info will be provided by our SM.it will provides the details of changes of state to EM. After EM will start running our new executable present in that mode.

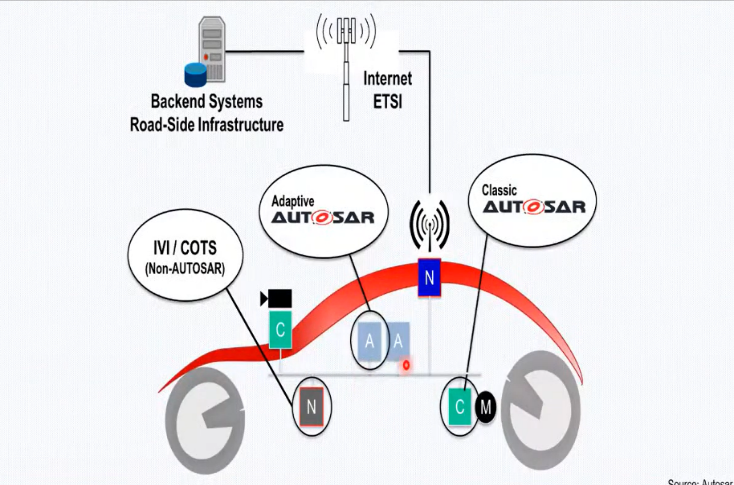
A diagram of a car

Description automatically generated

### Communication between AUTOSAR Platforms

How to communicate between Adaptive and Classic ECUs is a mandatory question. In such a scenario, the ECUs which are interconnected over Ethernet use service-oriented communication over SOME/IP. In this example, the AUTOSAR Classic ECU1 is connected to multiple bus systems to which other ECUs are connected. ECU1 operates as a gateway in this configuration and it’s responsible for transferring the message signals from the bus side into a service so that they can be accessed directly by the AUTOSAR Adaptive platform. The communications layout is a fixed component of the design of AUTOSAR ECUs, whether it is a Classic or Adaptive platform. Because the configuration format is different for the two platforms, it is necessary to map the service configuration in the form of a conversion. The situation is somewhat more multifaceted for communicating with an AUTOSAR Classic ECU whose operation is exclusively signal based. In this scenario, the ECU1 is designed as a signal gateway, and it converts message signals directly into UDP frames (“Specification of UDP Network Management,” n.d.) on Ethernet. The AUTOSAR Adaptive ECU now converts signals from the UDP frame to a service that is available within ECU2.

A diagram of a computer system

Description automatically generated

**Fig:** Adaptive Autosar interacted with classic Autosar platform and non-Autosar platform

### Salient feature of Adaptive Autosar Platform

* **Service -Oriented Architecture:** Here with the help of SOA the functionality is available as the services can be dynamically integrated at runtime. the meaning of SOA is whatever the service that we want only at that time itself the service will be executed if we do not request for the service that service will not be executed in that ways power consumption can be reduced but classic Autosar what are the service are there It will continuously execute over the lifetime of the ECU.
* **POSIX Based OS:** The Adaptive Autosar Platform uses POSIX based OS so here it provides the standard interface for the communication between the application and OS. Normally in our classic Autosar the application component was connected to the RTEso all the work the scheduling and everything was done using RTE only. But the new thing in Adaptive Autosar is each application contains the OS for creation a process for that. for example, we have 3 applications inside our Adaptive Autosar so 3 processors will be created inside OS and priority will be assigned to them.
* **C++ as the coding Language:** Because C programming did not have as many features as C++ has so with help of the C++ all the provide so developing in the Adaptive as a bit easier and it is more efficient.
* **Planned dynamics:** The platform supports over-the-air updates, allowing for new features and security patches to be deployed without interrupting the vehicle’s operation. Available dynamic software updated at runtime.
* **Safety:** The platform adheres to functional safety standards such as ISO 26262, ensuring that the system can handle faults and failures without compromising the vehicle’s safety. This includes features like redundancy, fault detection, and fail-safe mechanisms.
* **Security:** The platform incorporates robust security measures to protect against cyber threats. This includes:
* **Intrusion Detection Systems (IDS):** Monitoring for unusual activities that could indicate a security breach.
* **Cryptography:** Ensuring data integrity and confidentiality through encryption.
* **Access Control:** Restricting access to sensitive functions and data based on predefined policies.
* **Parallel processing and Multi-Core Processors:** we have larger amount of data that can be parallelly processed with help of multi-core. using this feature processing time will be reduced and response time will be increased.
* **Interoperable with non-Autosar Components:** Adaptive Autosar can interact with classic Autosar as well as any other architecture that is present inside the market. Using Ethernet or getaway it will communicate.

## Classic AUTOSAR VS Adaptive AUTOSAR

**Classic:**

* First released on 2014.
* It works on Signal based communication like CAN, LIN, MOST and Flex Ray etc.
* Implementation of deeply embedded functionalities.
* Software update at run time is not possible, communication between the software components are hard-wired.
* classic AUTOSAR are written in C Language.
* The software executes from ROM memory.
* Classic AUTOSAR is based on OSEK operating system.
* Update in a Classic Platform implies replacement of the entire ECU code.
* does support such multicore processors in latest version.
* It is configured in a static manner.
* Computing power low.
* It supports up to ASIL-D standard.
* **Examples of future systems:** Engine Control, Braking systems, Airbag Control Unit, sensor and actuator interfacing etc.

**Adaptive:**

* First released on 2018.
* It works on Service based communication like SOMEIP and Ethernet.
* Implementation of high-performance functionalities.
* Adaptive AUTOSAR RTE is independent of the applications and hence, Over-The-Air update is possible.
* Adaptive AUTOSAR are written in C++ Language
* The software executes on RAM memory.
* Adaptive AUTOSAR is based on POSIX operating system.
* Adaptive platforms provide the options to remove/update individual applications in an ECU.
* does support such multicore processors.
* It is configured in a Dynamically manner.
* Computing power high.
* It supports up to ASIL-B standard
* **Examples of future systems:** Over-The-Air updates (OTA), Sensor fusion Data processing, Persistence, Dynamic choosing of application packages over run-time of vehicle, ADAS, AD, AI, ML and complex ECU development etc.

## Applications of AUTOSAR

* Sensors like LIDAR and RADAR
* Electrification
* ADAS Functions with a Camera
* v2x
* Map Updates
* Automotive Apps
* Over-the-Air (OTA) Updates
* ECU Integration and Compatibility
* ECU Software Development

## Advantages and Disadvantages of AUTOSAR

**Advantages:**

* Reusability of software component
* Software code can be reused
* Design flexibility is more
* Cost and development time will be reduced
* To process large amount of data in short period of time.
* update the software with time to get optimized software.
* To support all the other vendors’ software.
* Available dynamic software updated at runtime.

**Disadvantages:**

* Complexity
* Initial Investment
* Learning Curve

## Definitions of AUTOSAR Keywords

#### OSEK

* The OSEK (“**Open Systems and their Interfaces for the Electronics in Motor Vehicles** “) is a standards body that has produced specifications for an embedded operating system, a communications stack, and a network management protocol for automotive embedded systems. It has also produced other related specifications. OSEK was designed to provide a standard software architecture for the various electronic control units (ECUs) throughout a car.
* The OSEK was founded in 1993 by a German automotive company consortium (BMW, Robert Bosch GmbH, DaimlerChrysler, Opel, Siemens, and Volkswagen Group) and the University of Karlsruhe. In 1994, the French car manufacturers Renault and PSA Peugeot Citroën, which had a similar project called VDX (Vehicle Distributed executive), joined the consortium. Therefore, the official name is OSEK/VDX.

A diagram of a software company

Description automatically generated with medium confidence

#### SOMEIP

SOME/IP (**Scalable Service-Oriented Middleware over IP**) is the network protocol that acts as the middle layer. SOME/IP is a communication protocol used in the automotive industry for inter-ECU communication within a vehicle. It is a part of the AUTOSAR (Automotive Open System Architecture) standard and provides a standardized mechanism for exchanging data and services between electronic control units (ECUs) in a vehicle over an IP-based network.

#### CAN

CAN, short for **Controller Area Network**, is a widely used communication protocol in the automotive industry and other domains where real-time and reliable communication between electronic devices is crucial. It was originally developed by Robert Bosch GmbH in the 1980s to address the challenges of interconnecting various electronic control units (ECUs) within a vehicle.

#### Ethernet

* **Ethernet** is a widely used networking technology that was originally developed for local area networks (LANs) in office and enterprise environments. Over time, Ethernet has evolved and become a pervasive technology in various domains, including the automotive industry, industrial automation, data centers, and more.
* Complex ECU operations in applications such as ADAS and [Over-the-Air (OTA)](https://www.embitel.com/firmware-over-the-air-fota-updates-for-iot-and-automotive-devices) [update](https://www.embitel.com/firmware-over-the-air-fota-updates-for-iot-and-automotive-devices) require higher bandwidth; something that conventional [CAN protocol](https://www.embitel.com/can-stack-software-solution-for-in-vehicle-network-communication) cannot
* achieve. **Ethernet** solves this issue by offering higher bandwidth which enables accurate transfer of large messages and point-to-point communication, among others.

#### ECU

ECU stands for **Electronic Control Unit**. It is a specialized electronic device or embedded system that controls and manages various functions and subsystems within a vehicle or any other complex system. ECUs are critical components in modern automobiles, where they play a crucial role in controlling engine performance, transmission, braking, airbags, infotainment systems, and other vehicle systems.

#### Dynamic Configuration

Dynamic Configuration refers to the capability of modifying certain parameters and configurations of software components at runtime, i.e., while the system is running on the target Electronic Control Unit (ECU). This feature allows for greater flexibility and adaptability in automotive systems.

#### Static Configuration

Static Configuration refers to the process of defining and setting up various parameters and configurations of software components at compile-time or during the initial system configuration.

#### Software Component

* Software component is a piece of code which carries out an application.it is a modular building block that can be used to construct an AUTOSAR software systems.
* The main use of the software component in AUTOSAR is to make it reusability and to design a software module module independent of embedded hardware’s.

#### P-Port

In AUTOSAR (Automotive Open System Architecture), a P-port (Provided-port) is a type of port used in the communication between different Software Components (SWCs) within the AUTOSAR software architecture. Ports facilitate the exchange of data and signals between SWCs, allowing them to interact and cooperate to perform the desired functionalities.

#### R-Port

R-port is a nothing, but receiver port also called as server or receiver, it mainly used for receiving the data via specific interface.

#### PR-Ports

PR is a provider and receiver ports; it can take the role of both required and provided port prototype.