AUTOSAR

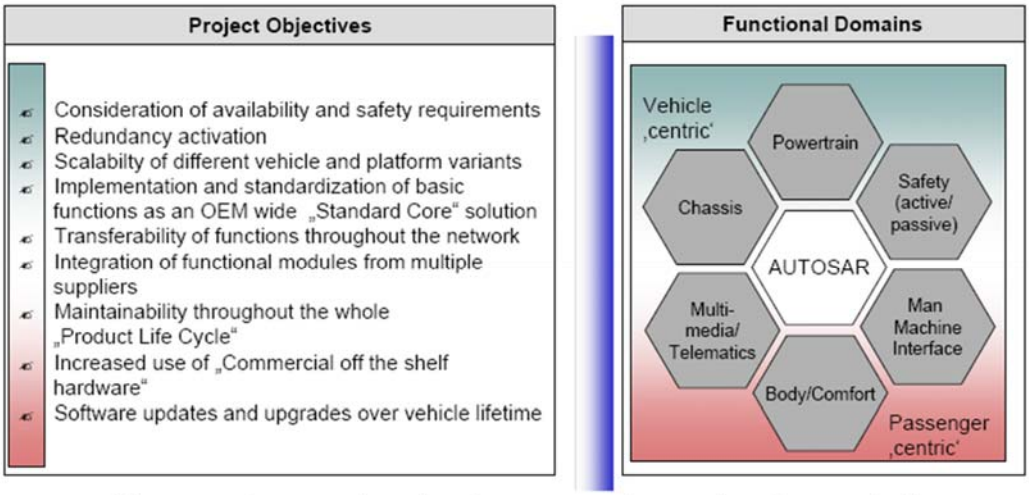
Main

2011/9/19

Why do we need AUTOSAR?

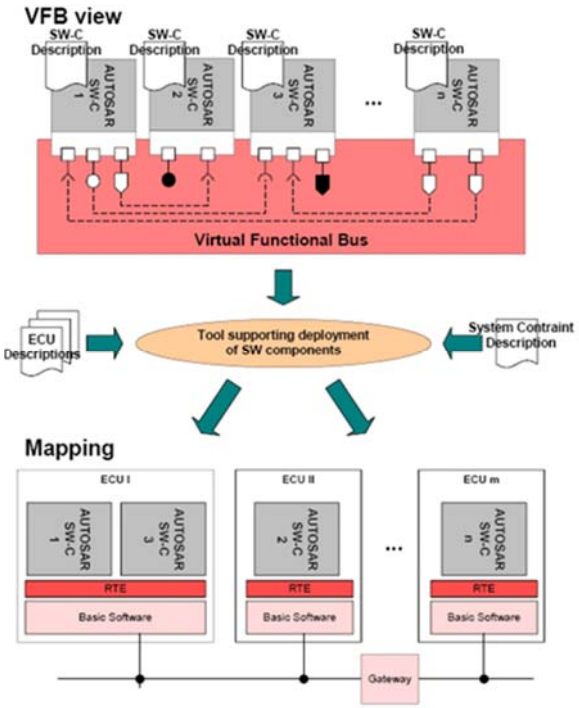
- AUTOSAR?
  - AUTomotive Open System Architecture.
  - Is a partnership of automotive manufacturers and suppliers working together to development and establish a de-facto open industry standard for automotive E/E architectures.

Objectives and Technical Benefits of AUTOSAR



Cooperate on standards, compete on implementation

Basic AUTOSAR approach



AUTOSAR Software Components?

- A fundamental design concept of AUTOSAR is the separation between:
  - **Application**
    - An application in AUTOSAR consists of interconnected "AUTOSAR Software Components".
    - Each AUTOSAR Software Component encapsulates a part of the functionality of the application.
  - **Infrastructure**

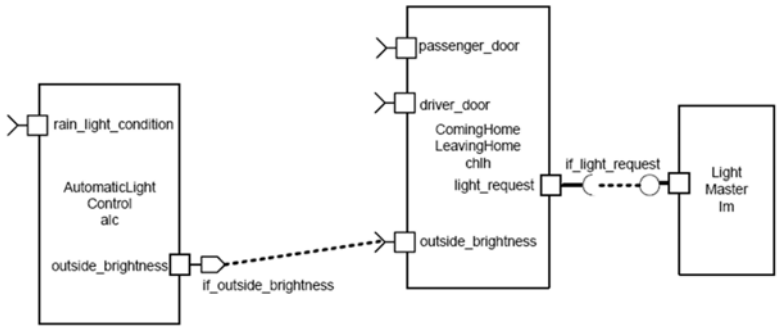


Figure 3: Example of interconnected AUTOSAR Software Components

AUTOSAR Software Components?

- The "**AUTOSAR Software Component**"
  - is an atomic piece of software that implements part of an application
  - is independent of the infrastructure
  - can be mapped on an ECU.
- The "**Sensor/Actuator Software Component**" is a special kind of AUTOSAR Software Component which encapsulates the dependencies on specific sensors and/or actuators.
- A component can also be a logical interconnection of other components(called a "**Composition**") which can be distributed over several ECUs.
  - And the following entities in an AUTOSAR system are also considered "**components**" – in "ECU Abstraction", "Complex Device Driver" and "AUTOSAR Service".

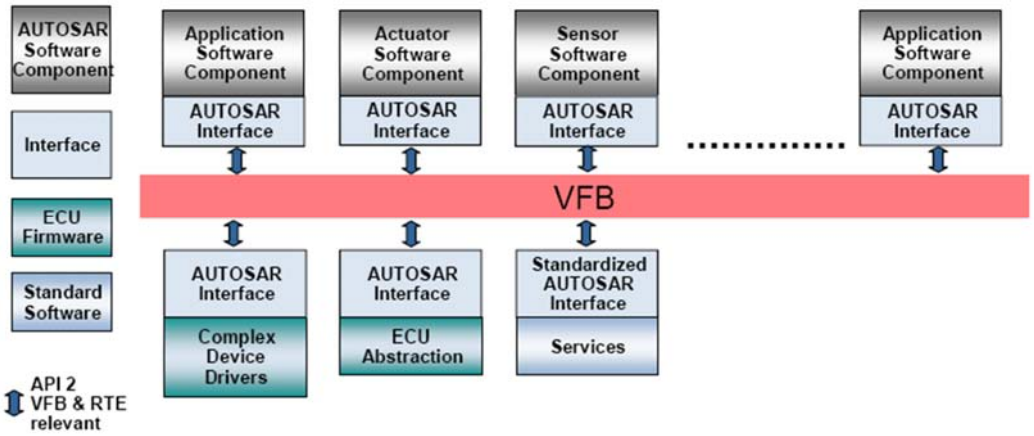
The AUTOSAR SW-C Description

- Contains
  - The operations and data elements that the SW-C provides and requires
  - The requirements that SW-C has on the infrastructure
  - The resources needed by the SW-C
  - The information regarding the specific implementation of the SW-C
- The structure and format of this AUTOSAR SW-C description is called the **"software component template"**.

VFB

- In order to fulfill the goal of **relocatability**, AUTOSAR Software Components are implemented independently from the underlying hardware.
- The independence is achieved by providing the **virtual functional bus** as **a means for a virtual hardware and mapping independent system integration**.

VFB Context



Communication mechanisms

- Components, Ports and AUTOSAR Interfaces
  - Components can interact with other components through well-defined **ports**.
  - A port always belongs to exactly one component and represents a point of interaction between a component and other components.
  - **PPort** and **RPort**
    - **PPort provides** an AUTOSAR Interface.
    - **RPort requires** an AUTOSAR Interface.

Communication mechanisms

- The AUTOSAR interface concept is introduced to define the services or data that are provided on or required by a port of a component.
  - Client-Server interface
  - Sender-Receiver Interface

Communication mechanisms

- Client-Server Communication

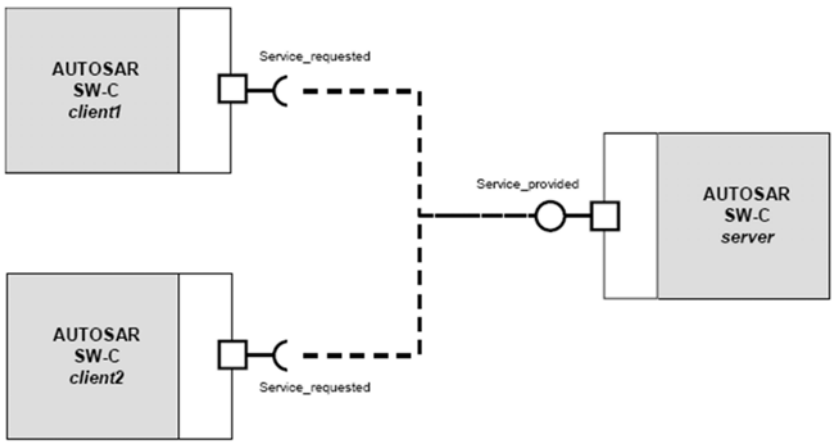


Figure 6: Client-server communication in the VFB view

Communication mechanisms

- Sender-Receiver Communication

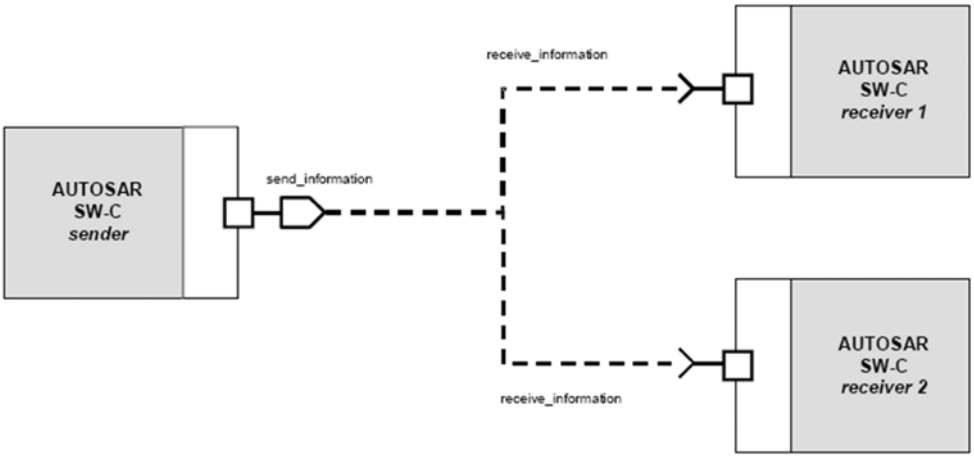


Figure 7: Data distribution by asynchronous non-blocking communication in the VFB view

AUTOSAR ECU Software Arch.

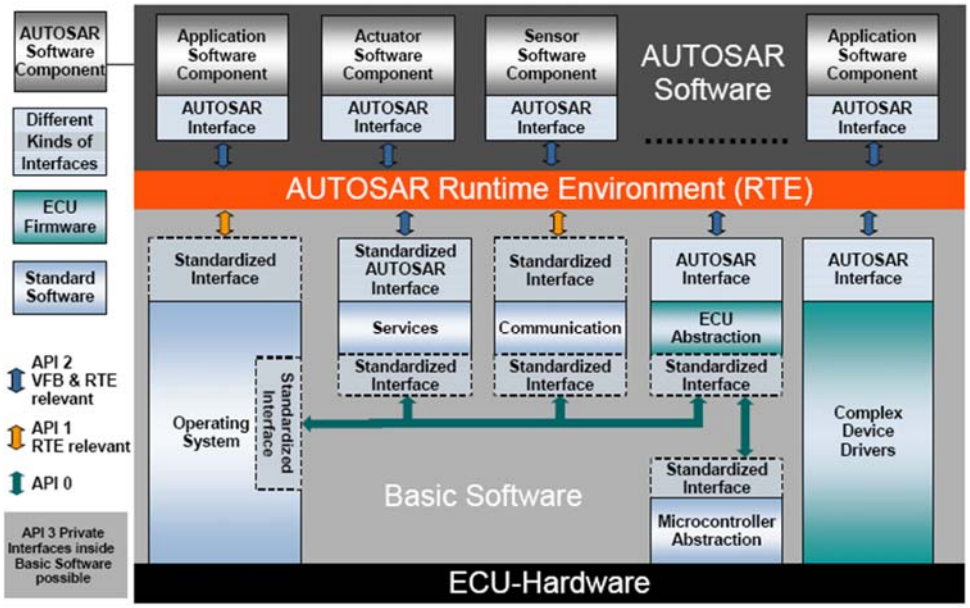


Figure 8 Schematic view of AUTOSAR ECU software architecture

Detailed ECU Architecture

Layered Software Arch.

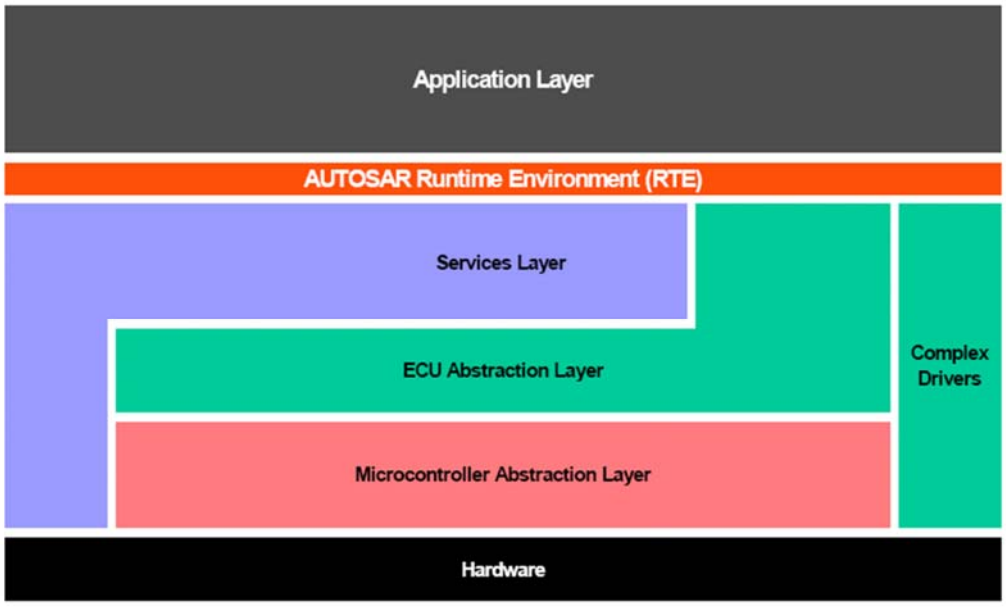
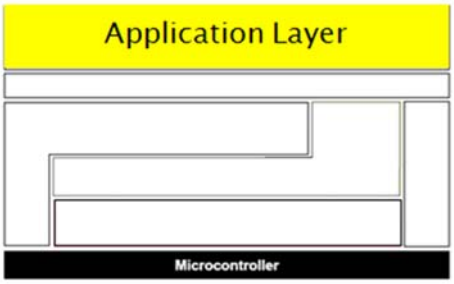


Figure 14: Overall layered architecture

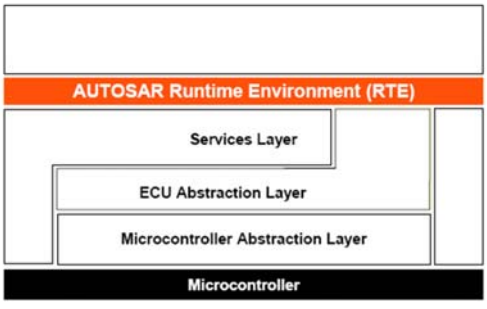
Application Layer

Application Layer:	
Mapping from ECU SW-Architecture:	The application layer of the ECU software architecture has been left untouched. A layering within the application layer is not the focus of AUTOSAR.
Purpose:	In the application layer all AUTOSAR Software Components, the Application Software Components as well as the Sensor/Actuator Software Components, are located.
Responsibility:	The software components of the application layer shall communicate internally via RTE and access all ECU resources via RTE.
Properties:	Implementation: µC independent, ECU independent, HW independent Interface: standardizable, syntactics pre-defined by AUTOSAR



AUTOSAR Runtime Environment

AUTOSAR Runtime Environment (RTE):	
Mapping from ECU SW-Architecture:	The RTE has also not been changed compared to the ECU software architecture.
Purpose:	The purpose of the RTE is to enable the implementation of AUTOSAR Software Components independent from the mapping to a specific hardware/ECU
Responsibility:	The RTE is a hardware independent layer providing communication services for the application containing Application Software Components and Sensor/Actuator Software Components. Above the RTE the software architecture style changes from „layered“ to „component style“. The AUTOSAR Software Components communicate with other components (inter and/or intra ECU) via the RTE.
Properties:	Implementation: ECU and application specific (generated individually for each ECU) Upper Interface: completely ECU independent

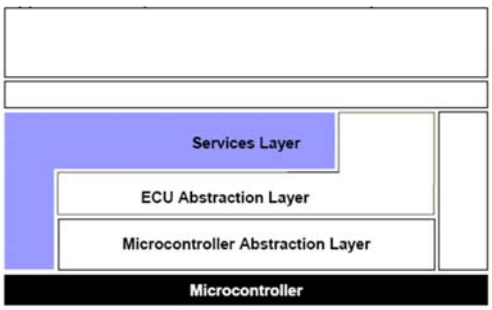


The Runtime Environment(RTE)

- Overview
  - The AUTOSAR Runtime Environment(RTE) is the runtime representation of the Virtual Function Bus for a specific ECU.
  - It must provide a uniform environment to AUTOSAR Software Components.
  - The purpose is to make the implementation of the software components independent from the communication mechanisms and channels.
  - Access to ports from a software component implementation
  - Implementation of connectors
  - Access to Basic Software
  - Lifecycle management
  - Multiple instantiations of software components

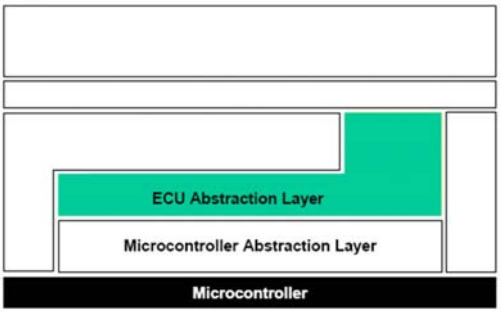
Services Layer

Services Layer:	
Mapping from ECU SW-Architecture:	The service layer is a layer which consists of the communication, services and operating system blocks shown in the ECU software architecture.
Purpose:	It provides basic services for application and Basic Software modules to abstract the microcontroller as well as the ECU hardware from the layers above.
Responsibility:	It is the highest layer of the Basic Software which also applies for its relevance for the application software: while access to I/O signals is covered by the ECU abstraction layer, the services layer offers • Operating system services • Vehicle network communication and management services • Memory services (NVRAM management) • Diagnosis Services (including diagnosis communication interface and error memory) • ECU state management
Properties:	Implementation: partly µC, ECU hardware and application specific Upper Interface: µC and ECU hardware independent



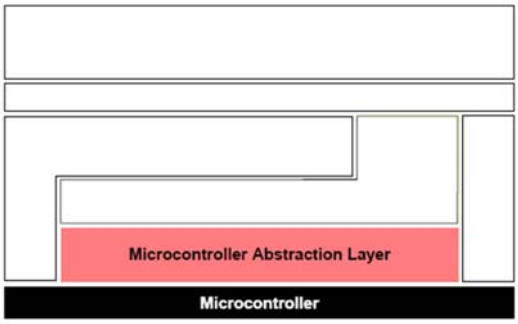
ECU Abstraction Layer

ECU Abstraction Layer:	
Mapping from ECU SW-Architecture:	The ECU abstraction layer is the most obvious change of the layered software architecture compared to the ECU software architecture. Nearly all standardized interfaces within the Basic Software shown in the ECU software architecture have been mapped into that layer.
Purpose:	To abstract the ECU layout from the above layer.
Responsibility:	It contains handlers, which are software modules which abstract how peripherals are connected to the CPU (e.g. on chip, implemented in ASIC and connected via SPI, …).
Properties:	Implementation: µC independent, ECU hardware dependent Upper Interface: µC and ECU hardware independent, dependent on signal type



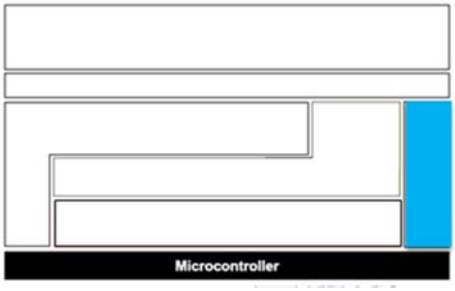
Microcontroller Abstraction Layer

Microcontroller Abstraction Layer (MCAL):	
Mapping from ECU SW-Architecture:	According to the ECU software architecture the microcontroller abstraction layer has been placed between the ECU abstraction and the real hardware.
Purpose:	The MCAL abstracts the microcontroller from the above layer to make upper layers µC independent.
Responsibility:	The MCAL layer is the lowest software layer of the Basic Software. It contains drivers, which are software modules with direct access to the µC internal peripherals and memory mapped µC external devices.
Properties:	Implementation: µC dependent, Upper Interface: standardizable and µC independent



Complex Drivers

Complex Drivers:	
Mapping from ECU SW-Architecture:	The Complex Device Driver could not be mapped to a specific layer and has therefore been taken over from the ECU software architecture with no change.
Purpose:	The Complex Device Drivers fulfill the special functional and timing requirements for handling complex sensors and actuators.
Responsibility:	A Complex Driver implements complex sensor evaluation and actuator control with direct access to the µC using specific interrupts and/or complex µC peripherals (like PCP, TPU), e.g. • injection control • electric valve control • incremental position detection
Properties:	Implementation: highly µC, ECU and application dependent Upper Interface: specified and implemented as AUTOSAR Interfaces.



Refinement of the Layered Arch.

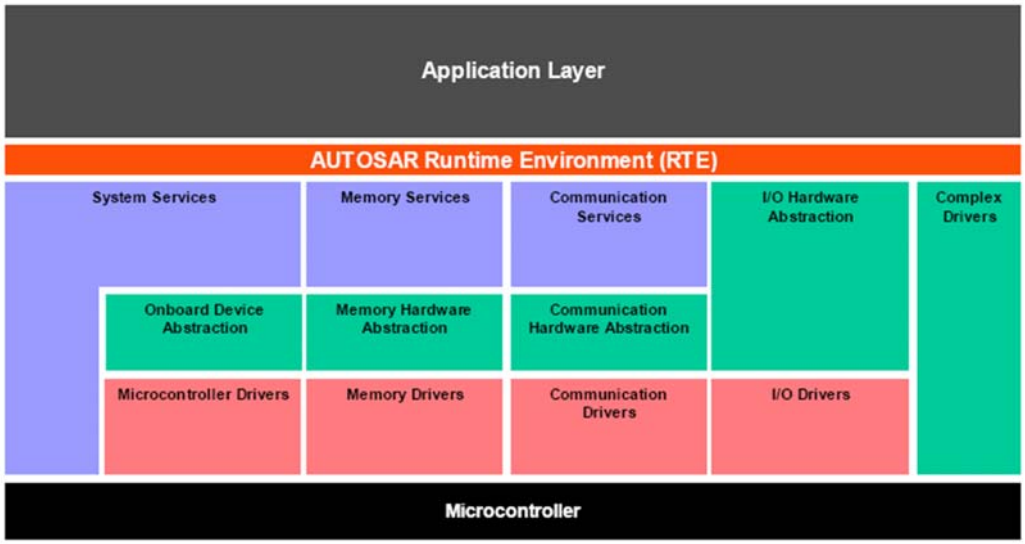
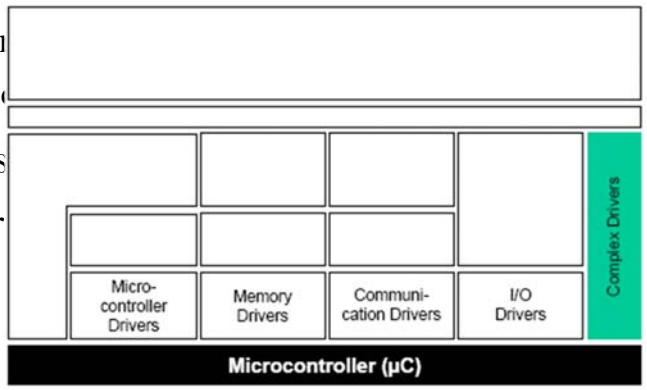


Figure 15: Refined layered software architecture

Microcontroller Abstraction Layer

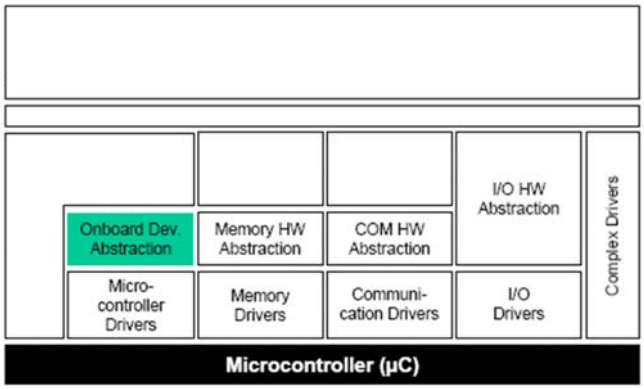
- I/O Drivers
  - Drivers for analog and digital I/O
- Communication Drivers
  - Drivers for ECU onboard and vehicle communication
- Memory Drivers
  - Drivers for on-chip mei
    external memory devic
- Microcontroller Drivers
  - Drivers for internal per
    microcontroller access



ECU Abstraction Layer

- I/O Hardware Abstraction
  - Is a group of modules which al
    peripheral I/O devices and the

The task of this group of modules is
- to represent I/O signals as they a
  current, voltage, frequency), and
- to hide ECU hardware and layout



- Communication Hardware Abstraction
  - Is a group of modules which abstracts from the location of communication controllers and the ECU H/W layout.

The task of this group of modules is
- to provide equal mechanisms to access a bus channel regardless of it's location (on-chip / on-board).

ECU Abstraction Layer

- Memory Hardware Abstraction
  - Is a group of modules which abstracts from the location of peripheral memory devices and the ECU H/W layout.

The task of this group of modules is
- to provide equal mechanisms to access internal (on-chip) and external (on-board) memory devices.

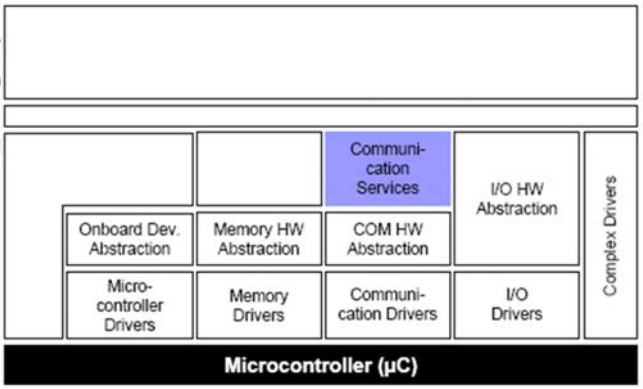
- Onboard Device Abstraction
  - Contains drivers for ECU onboard devices which cannot be seen as sensors or actuators like system basic chip, external watchdog etc. those drivers access the ECU onboard devices via the MCAL.

The task of this group of modules is
- to abstract from ECU specific onboard devices.

Service Layer

- Communication Services
  - Are a group of modules for vehicle network communication.
  - Are interfacing with the communication drivers via the communication H/W abstraction.

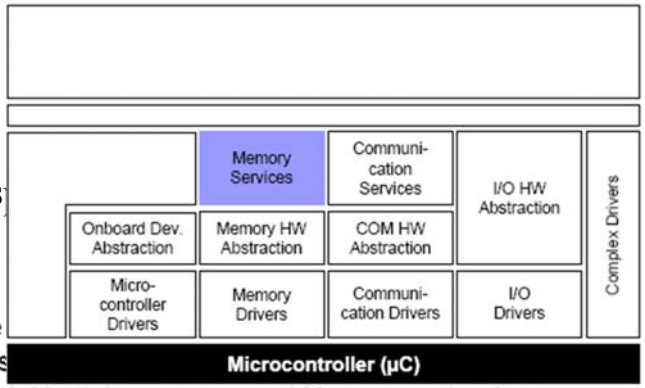
The task of this group of modules is
- to provide a uniform interface to the vehicle network for communication between different applications,
- to provide uniform services for r
- to provide a uniform interface to communication, and
- to hide protocol and message p



Service Layer

- Memory Services
  - Are a group of modules res
    volatile data.

The task of this group of modules is
- to provide non volatile data to the
- to abstract from memory location
- to provide mechanisms for non volatile data management like saving, loading, checksum protection and verification, reliable storage etc.



- System Services
  - Are a group of modules and
    modules of all layers

The task of this group of modules is
- to provide basic services for ap