How does it work? An Overview of the AUTOSAR Adaptive Platform in Details (R23-11)

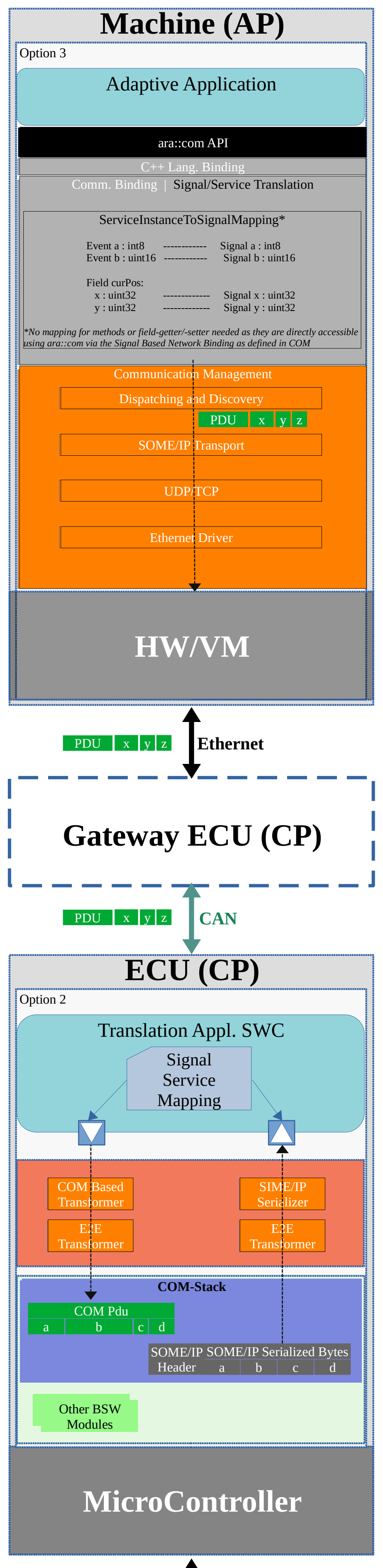
Version 1.0 ©Copyright by Yingtao WANG, Mar. 2025, Germany contact: Yingtao.Wang@stellantis.com

CP ← communication → AP

Option 1:
A CP ECU and an AP Machine can communicate directly without any adaptation if they are connected via Ethernet and use the same transfer protocol, such as SOME/IP.

Option 2:
In cases where direct communication is not feasible, an adaptation mechanism is required to convert between CP PDUs and AP Service Interface Messages. One solution is to implement an Adaptation SWC (Software Component) on the CP side. This component maps CP signals to the corresponding members (e.g., events, fields, methods) of the AP interface on a one-to-one basis. The data is then transferred using the respective communication protocols, as illustrated in the example below.

Option 3:
Similar to Option 2, the adaptation can also be implemented on the AP Machine. In this approach, a specialized software component, referred to as a 'Signal/Service Translator,' performs the necessary conversion. Additionally, if different physical channels such as Ethernet and CAN are used, a Gateway ECU may be introduced to bridge the communication between the two systems. This configuration is also depicted in the example below.


Machine (AP)
Option 3
Adaptive Application
ara::com API
C++ Lang. Binding
Comm. Binding | Signal/Service Translation
ServiceInstanceToSignalMapping*
Event a : int8 ----------- Signal a : int8
Event b : uint16 ----------- Signal b : uint16
Field curPos:
x : uint32 ------------ Signal x : uint32
y : uint32 ------------ Signal y : uint32
*No mapping for methods or field-getter/-setter needed as they are directly accessible using ara::com via the Signal Based Network Binding as defined in COM
Communication Management
Dispatching and Discovery
PDU x y z
SOME/IP Transport
UDP/TCP
Ethernet Driver
HW/VM
PDU x y z
Ethernet
Gateway ECU (CP)
PDU x y z
CAN
ECU (CP)
Option 2
Translation Appl. SWC
Signal Service Mapping
COM Based Transformer
E2E Transformer
SIME/IP Serializer
E2E Transformer
COM-Stack
COM Pdu
a b c d
SOME/IP SOME/IP Serialized Bytes
Header a b c d
Other BSW Modules
MicroController

Data Type(s)

ApplicationDataType(s)

CompuMethod

STD_CPPImplementationDataTypes
+ bool, float, double, int8_t, int16_t, int32_t, int64_t, uint8_t, uint16_t, uint32_t, uint64_t

Custom_CPPImplementationDataType(s)
+ any customed data type using for impl.

ServiceInterface(s)

MyServiceInterface01
+ field1
+ event1
+ method1(arg1:IN, arg2:OUT)
+ method2(arg1:OUT)

MyServiceInterface02
+ event1
+ event2
+ event3

SOMEIP ServiceInterfaceDeployment(s)

MyServiceInterface01_someip
+ event1_someip → ref. to event1
+ event2_someip → ref. to event2
+ method1_someip → ref. to method1
+ method2_someip → ref. to method2

+ ref. to MyServiceInterface01
+ ServiceInterfaceID: 101

MyServiceInterface02_someip
+ field1_someip → ref. to field1
+ event1_someip → ref. to event1
+ method1_sometip → ref. to method1

+ ref. to MyServiceInterface02
+ ServiceInterfaceID: 102

ServiceInstance(s)

ProvidedServiceInterface01_Instance
+ ref. to MyServiceInterface01_someip
+ ServiceInstanceID: 501
+ ref. to My**Server**ServiceInstanceConfig

RequiredServiceInterface01_Instance01
+ ref. to MyServiceInterface01_someip
+ ServiceInstanceID: ANY
+ ref. to My**Client**ServiceInstanceConfig

RequiredServiceInterface01_Instance02
+ ref. to MyServiceInterface01_someip
+ ServiceInstanceID: 601
+ ref. to My**Client**ServiceInstanceConfig

ProvidedServiceInterface02_Instance
+ ref. to MyServiceInterface02_someip
+ ServiceInstanceID: 502
+ ref. to My**Server**ServiceInstanceConfig

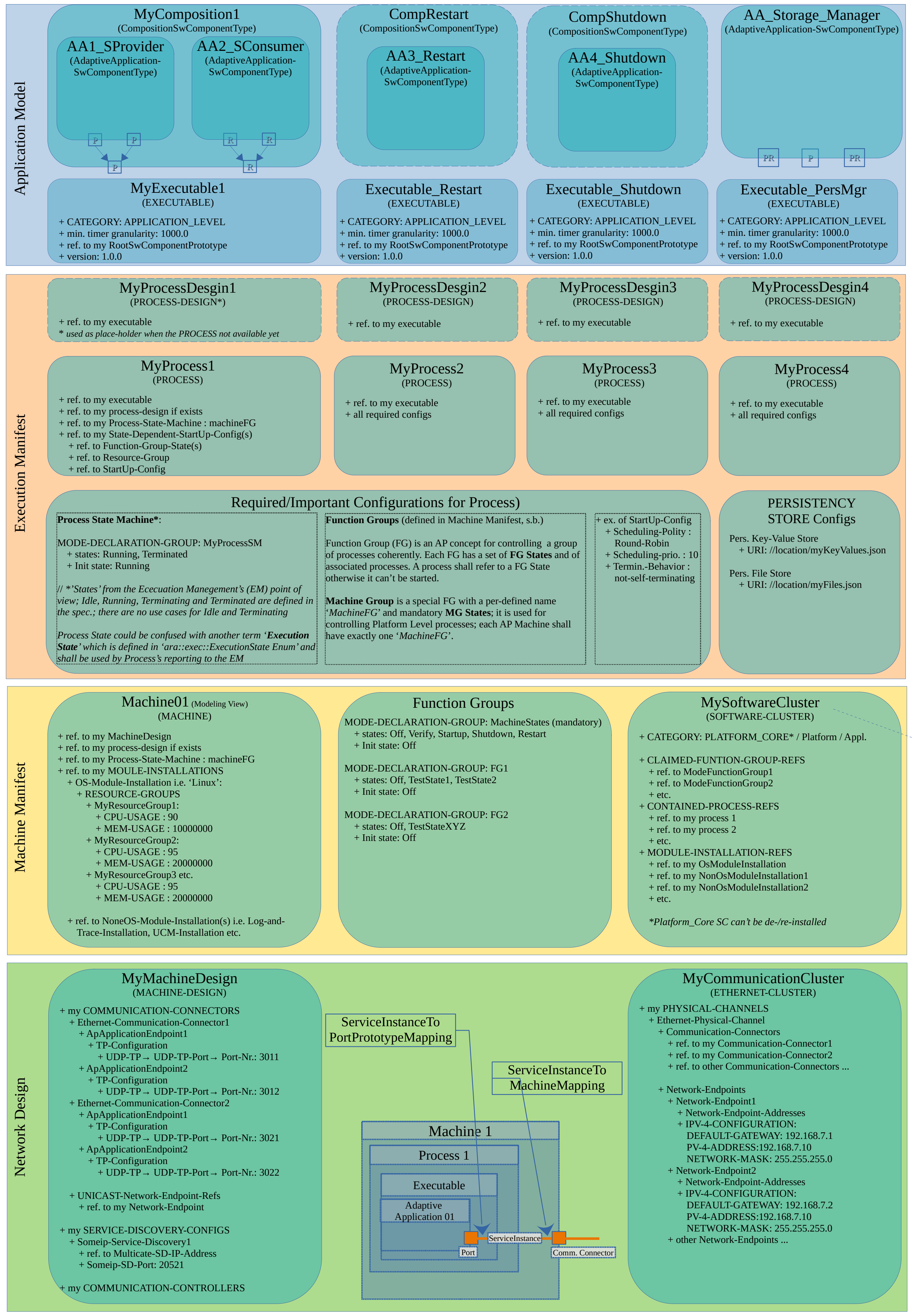
RequiredServiceInterface02_Instance
+ ref. to MyServiceInterface01_someip
+ ServiceInstanceID: ANY
+ ref. to My**Client**ServiceInstanceConfig

SOMEIP_SD_TimingConfigs

SomeipSd**Client**ServiceInstanceConfig
+ InitialFindBehavior
  + InitDelayMax: 0.1
  + InitDelayMin: 0.01
  + InitRepBaseDelay: 0.2
  + InitRepMax: 3
+ ServiceFindTimeToLive: 10

SomeipSd**Server**ServiceInstanceConfig
+ InitialFindBehavior
  + InitDelayMax: 0.1
  + InitDelayMin: 0.01
  + InitRepBaseDelay: 0.2
  + InitRepMax: 3
+ OfferCyclicDelay: 2
+ RequestResponseDelay
  + Max: 1.5
  + Min: 1.5
+ ServiceFindTimeToLive: 10

Service Instance Manifest


Application Model
MyComposition1
(CompositionSwComponentType)
AA1_SProvider
(AdaptiveApplication-SwComponentType)
AA2_SConsumer
(AdaptiveApplication-SwComponentType)
P P P R R R
CompRestart
(CompositionSwComponentType)
AA3_Restart
(AdaptiveApplication-SwComponentType)
CompShutdown
(CompositionSwComponentType)
AA4_Shutdown
(AdaptiveApplication-SwComponentType)
AA_Storage_Manager
(AdaptiveApplication-SwComponentType)
PR P PR
Execution Manifest
Machine Manifest
Network Design
ServiceInstanceTo PortPrototypeMapping
ServiceInstanceTo MachineMapping
Machine 1
Process 1
Executable
Adaptive Application 01
Port
ServiceInstance
Comm. Connector

MyExecutable1
(EXECUTABLE)
+ CATEGORY: APPLICATION_LEVEL
+ min. timer granularity: 1000.0
+ ref. to my RootSwComponentPrototype
+ version: 1.0.0

Executable_Restart
(EXECUTABLE)
+ CATEGORY: APPLICATION_LEVEL
+ min. timer granularity: 1000.0
+ ref. to my RootSwComponentPrototype
+ version: 1.0.0

Executable_Shutdown
(EXECUTABLE)
+ CATEGORY: APPLICATION_LEVEL
+ min. timer granularity: 1000.0
+ ref. to my RootSwComponentPrototype
+ version: 1.0.0

Executable_PersMgr
(EXECUTABLE)
+ CATEGORY: APPLICATION_LEVEL
+ min. timer granularity: 1000.0
+ ref. to my RootSwComponentPrototype
+ version: 1.0.0

MyProcessDesgin1
(PROCESS-DESIGN*)
+ ref. to my executable
*used as place-holder when the PROCESS not available yet

MyProcessDesgin2
(PROCESS-DESIGN)
+ ref. to my executable

MyProcessDesgin3
(PROCESS-DESIGN)
+ ref. to my executable

MyProcessDesgin4
(PROCESS-DESIGN)
+ ref. to my executable

MyProcess1
(PROCESS)
+ ref. to my executable
+ ref. to my process-design if exists
+ ref. to my Process-State-Machine : machineFG
+ ref. to my State-Dependent-StartUp-Config(s)
  + ref. to Function-Group-State(s)
  + ref. to Resource-Group
  + ref. to StartUp-Config

MyProcess2
(PROCESS)
+ ref. to my executable
+ all required configs

MyProcess3
(PROCESS)
+ ref. to my executable
+ all required configs

MyProcess4
(PROCESS)
+ ref. to my executable
+ all required configs

Required/Important Configurations for Process)

Process State Machine*:

MODE-DECLARATION-GROUP: MyProcessSM
+ states: Running, Terminated
+ Init state: Running

*// *'States' from the Ececuation Manegement's (EM) point of view; Idle, Running, Terminating and Terminated are defined in the spec.; there are no use cases for Idle and Terminating*

*Process State could be confused with another term '**Execution State**' which is defined in 'ara::exec::ExecutionState Enum' and shall be used by Process's reporting to the EM*

Function Groups (defined in Machine Manifest, s.b.)

Function Group (FG) is an AP concept for controlling a group of processes coherently. Each FG has a set of **FG States** and of associated processes. A process shall refer to a FG State otherwise it can't be started.

Machine Group is a special FG with a per-defined name '*MachineFG*' and mandatory **MG States**; it is used for controlling Platform Level processes; each AP Machine shall have exactly one '*MachineFG*'.

+ ex. of StartUp-Config
  + Scheduling-Polity : Round-Robin
  + Scheduling-prio. : 10
  + Termin.-Behavior : not-self-terminating

PERSISTENCY STORE Configs

Pers. Key-Value Store
+ URI: //location/myKeyValues.json

Pers. File Store
+ URI: //location/myFiles.json

Machine01 (Modeling View)
(MACHINE)
+ ref. to my MachineDesign
+ ref. to my process-design if exists
+ ref. to my Process-State-Machine : machineFG
+ ref. to my MOULE-INSTALLATIONS
  + OS-Module-Installation i.e. 'Linux':
    + RESOURCE-GROUPS
      + MyResourceGroup1:
        + CPU-USAGE : 90
        + MEM-USAGE : 10000000
      + MyResourceGroup2:
        + CPU-USAGE : 95
        + MEM-USAGE : 20000000
      + MyResourceGroup3 etc.
        + CPU-USAGE : 95
        + MEM-USAGE : 20000000

  + ref. to NoneOS-Module-Installation(s) i.e. Log-and-Trace-Installation, UCM-Installation etc.

Function Groups
MODE-DECLARATION-GROUP: MachineStates (mandatory)
+ states: Off, Verify, Startup, Shutdown, Restart
+ Init state: Off

MODE-DECLARATION-GROUP: FG1
+ states: Off, TestState1, TestState2
+ Init state: Off

MODE-DECLARATION-GROUP: FG2
+ states: Off, TestStateXYZ
+ Init state: Off

MySoftwareCluster
(SOFTWARE-CLUSTER)
+ CATEGORY: PLATFORM_CORE* / Platform / Appl.

+ CLAIMED-FUNTION-GROUP-REFS
  + ref. to ModeFunctionGroup1
  + ref. to ModeFunctionGroup2
  + etc.
+ CONTAINED-PROCESS-REFS
  + ref. to my process 1
  + ref. to my process 2
  + etc.
+ MODULE-INSTALLATION-REFS
  + ref. to my OsModuleInstallation
  + ref. to my NonOsModuleInstallation1
  + ref. to my NonOsModuleInstallation2
  + etc.

Platform_Core SC can't be de-/re-installed

MyMachineDesign
(MACHINE-DESIGN)
+ my COMMUNICATION-CONNECTORS
  + Ethernet-Communication-Connector1
    + ApApplicationEndpoint1
      + TP-Configuration
        + UDP-TP → UDP-TP-Port → Port-Nr.: 3011
    + ApApplicationEndpoint2
      + TP-Configuration
        + UDP-TP → UDP-TP-Port → Port-Nr.: 3012
  + Ethernet-Communication-Connector2
    + ApApplicationEndpoint1
      + TP-Configuration
        + UDP-TP → UDP-TP-Port → Port-Nr.: 3021
    + ApApplicationEndpoint2
      + TP-Configuration
        + UDP-TP → UDP-TP-Port → Port-Nr.: 3022

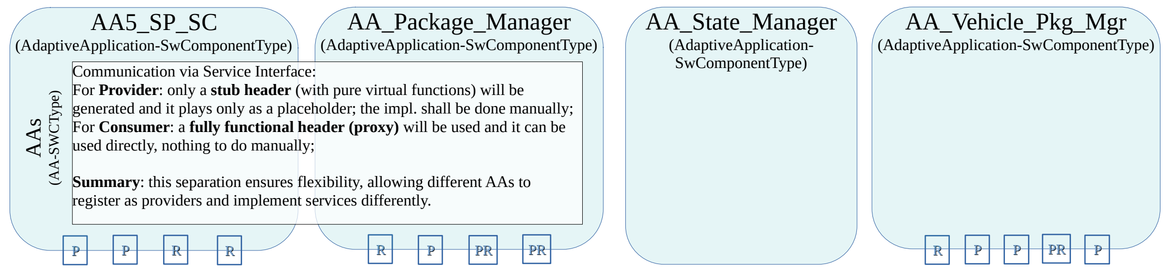
  + UNICAST-Network-Endpoint-Refs
    + ref. to my Network-Endpoint

+ my SERVICE-DISCOVERY-CONFIGS
  + Someip-Service-Discovery1
    + ref. to Multicate-SD-IP-Address
    + Someip-SD-Port: 20521

+ my COMMUNICATION-CONTROLLERS

MyCommunicationCluster
(ETHERNET-CLUSTER)
+ my PHYSICAL-CHANNELS
  + Ethernet-Physical-Channel
    + Communication-Connectors
      + ref. to my Communication-Connector1
      + ref. to my Communication-Connector2
      + ref. to other Communication-Connectors ...

    + Network-Endpoints
      + Network-Endpoint1
        + Network-Endpoint-Addresses
        + IPV-4-CONFIGURATION:
          DEFAULT-GATEWAY: 192.168.7.1
          PV-4-ADDRESS:192.168.7.10
          NETWORK-MASK: 255.255.255.0
      + Network-Endpoint2
        + Network-Endpoint-Addresses
        + IPV-4-CONFIGURATION:
          DEFAULT-GATEWAY: 192.168.7.2
          PV-4-ADDRESS:192.168.7.10
          NETWORK-MASK: 255.255.255.0
      + other Network-Endpoints ...


AAs
(AA-SWCType)
AA5_SP_SC
(AdaptiveApplication-SwComponentType)
AA_Package_Manager
(AdaptiveApplication-SwComponentType)
AA_State_Manager
(AdaptiveApplication-SwComponentType)
AA_Vehicle_Pkg_Mgr
(AdaptiveApplication-SwComponentType)
P P R R
R P PR PR
R P P PR P

Communication via Service Interface:
For **Provider**: only a **stub header** (with pure virtual functions) will be generated and it plays only as a placeholder; the impl. shall be done manually;
For **Consumer**: a **fully functional header (proxy)** will be used and it can be used directly, nothing to do manually;

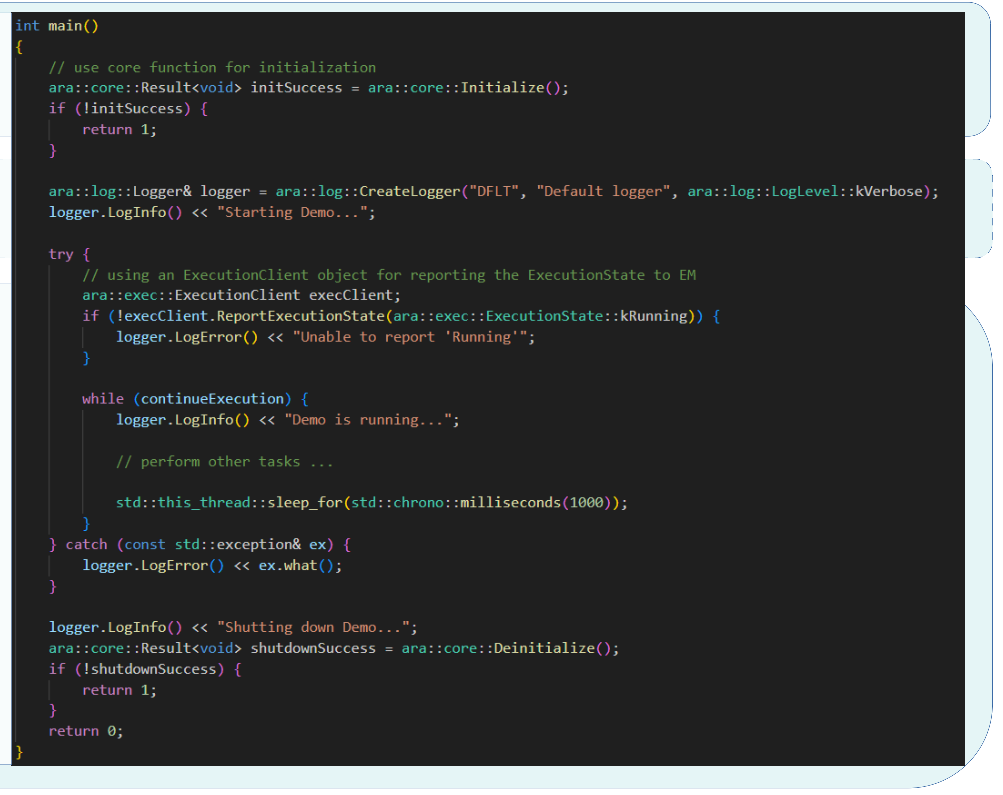
Summary: this separation ensures flexibility, allowing different AAs to register as providers and implement services differently.

Executables
(EXECUTABLE)

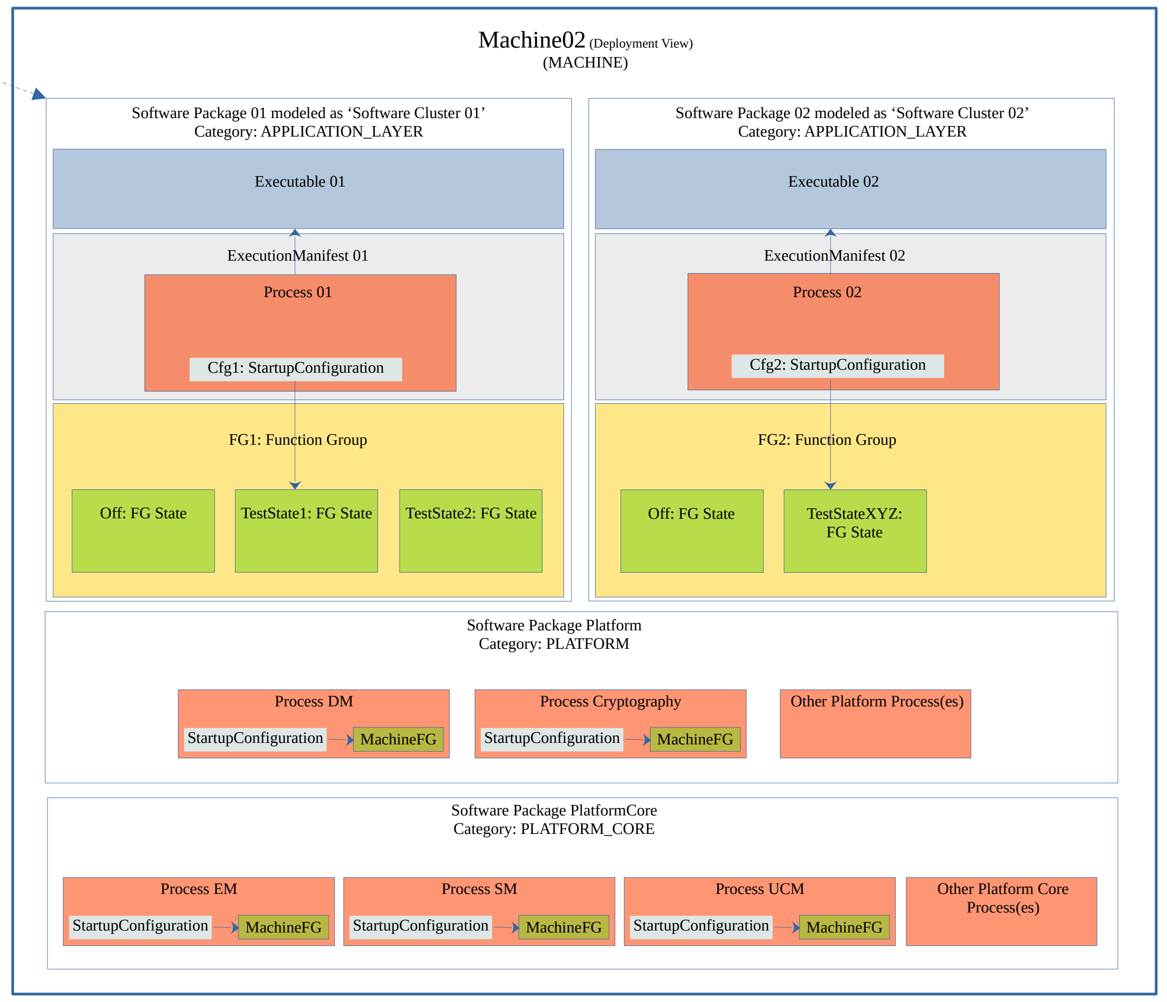
PDs
(PROCESS-DESIGN)

Processes
(PROCESS)

A typical Example Implementation in C++ for a Process Main Function
(based on example from APD)

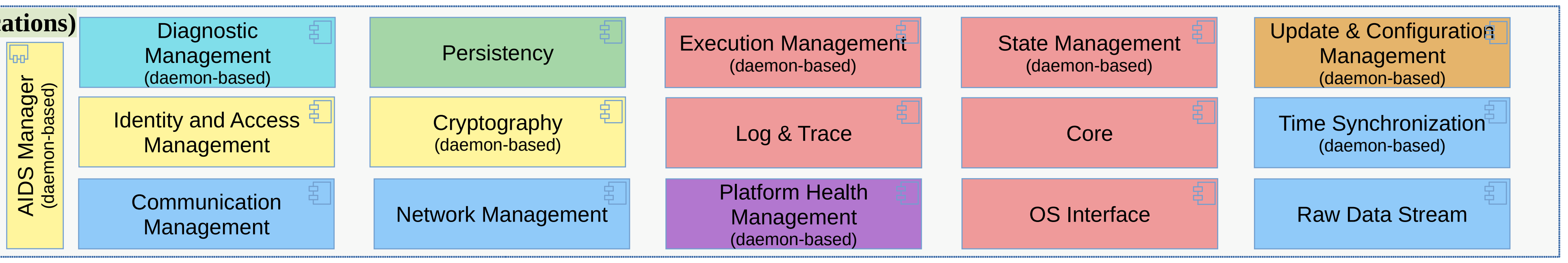


```
int main()
{
    // use core function for initialization
    ara::core::Result<void> initSuccess = ara::core::Initialize();
    if (!initSuccess) {
        return 1;
    }

    ara::log::Logger& logger = ara::log::CreateLogger("DFLT", "Default logger", ara::log::LogLevel::kVerbose);
    logger.LogInfo() << "Starting Demo...";

    try {
        // using an ExecutionClient object for reporting the ExecutionState to EM
        ara::exec::ExecutionClient execClient;
        if (!execClient.ReportExecutionState(ara::exec::ExecutionState::kRunning)) {
            logger.LogError() << "Unable to report 'Running'";
        }

        while (continueExecution) {
            logger.LogInfo() << "Demo is running...";

            // perform other tasks ...

            std::this_thread::sleep_for(std::chrono::milliseconds(1000));
        }
    } catch (const std::exception& ex) {
        logger.LogError() << ex.what();
    }

    logger.LogInfo() << "Shutting down Demo...";
    ara::core::Result<void> shutdownSuccess = ara::core::Deinitialize();
    if (!shutdownSuccess) {
        return 1;
    }
    return 0;
}
```



ARA (AUTOSAR RUNTIME for Adaptive applications)

The ARA Functional Clusters (FCs) provide essential foundational features for AAs. Similar to an AA, an FC is typically implemented as a single OS process or as multiple subprocesses/threads. Communication between FCs and AAs requires the use of Inter-Process Communication (IPC). An FC can be implemented in various ways: 1. Library-Based: In this approach, the FC includes an interface library that is linked to the AA and directly handles IPC communication; 2. Service-Based: This design utilizes ara::com interfaces to facilitate communication; 3. Local Library: The FC is implemented as a library linked directly within the process context. In this case, the library's functions can be called as normal in-process functions.



Operating System There is no standard OS defined for AUTOSAR Adaptive Platform (AP). In other words, any multi-process POSIX-compliant OS, such as Linux, that fulfills the OS requirements of the AP can be used. It is important to note that only components at the ARA level (e.g., all FCs of ARA) are permitted to directly utilize the full features of the OS, such as creating processes. Components above the ARA level, namely the AAs (Application Agents), are restricted to accessing OS features exclusively through the OSI (POSIX PSE51) interface.

HW Machine/Hypervisor/VM/Container 01

HW Machine/VM/Hypervisor/Container 02

Ethernet

Ethernet

START, EXECUTION, SHUTDOWN / RESTART OF A MACHINE

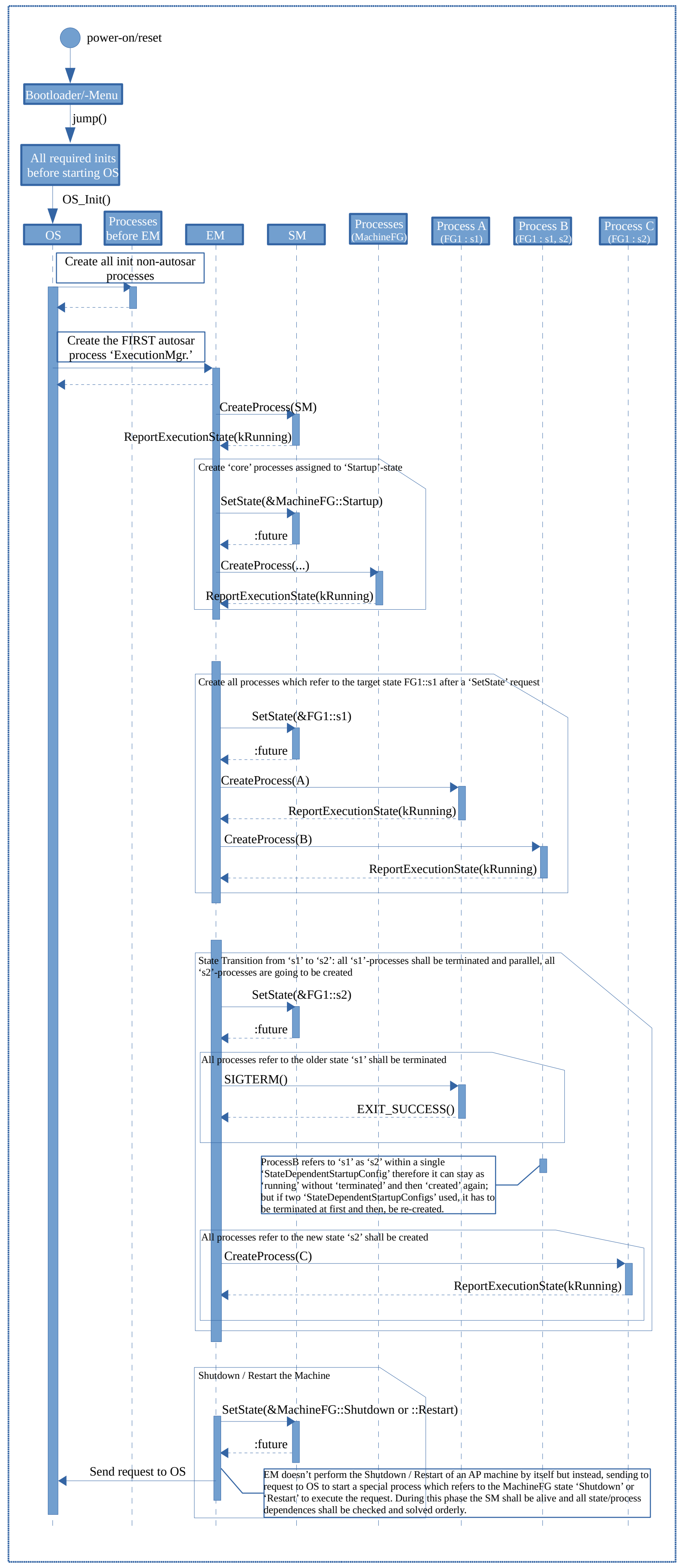
From the OS perspective, an AP Instance is essentially a collection of processes, which may include AAs (Adaptive Applications) operating above the ARA or FCs (Functional Clusters) running on the ARA, all executing in a multi-process environment.

The ARA, as a middleware platform, provides essential features such as Execution Management, State Management, and Communication Management, which can be reused by all AAs and FCs. Additionally, processes can leverage OS-level features like creating and scheduling processes or managing memory—either directly (for FCs) or indirectly (for AAs via ara::osi).

When starting an AP machine, the OS initializes the system and creates the EM process (Execution Manager) as the first AP process. This process acts as the parent for all subsequent AP processes, which are created by it. Following this, the SM process (State Manager) is created and started, transitioning the system state to 'Startup' for the Function Group MachineFG. This group typically includes all 'Platform Core' FCs, depending on their respective StartupConfigs.

At this stage, all core platform functionalities should be operational. Subsequently, application processes are started sequentially, based on their StartupConfigs and dependencies on other processes.

The description provided above is illustrated in the diagram below. Please note that due to space constraints, the example is highly simplified and intended only to provide a high-level overview.


power-on/reset
Bootloader/-Menu
jump()
All required inits before starting OS
OS_Init()
OS
Processes before EM
EM
SM
Processes (MachineFG)
Process A (FG1 : s1)
Process B (FG1 : s1, s2)
Process C (FG1 : s2)
Create all init non-autosar processes
Create the FIRST autosar process 'ExecutionMgr.'
CreateProcess(SM)
ReportExecutionState(kRunning)
Create 'core' processes assigned to 'Startup'-state
SetState(&MachineFG::Startup)
:future
CreateProcess(...)
ReportExecutionState(kRunning)
Create all processes which refer to the target state FG1::s1 after a 'SetState' request
SetState(&FG1::s1)
:future
CreateProcess(A)
ReportExecutionState(kRunning)
CreateProcess(B)
ReportExecutionState(kRunning)
State Transition from 's1' to 's2': all 's1'-processes shall be terminated and parallel, all 's2'-processes are going to be created
SetState(&FG1::s2)
:future
All processes refer to the older state 's1' shall be terminated
SIGTERM()
EXIT_SUCCESS()
ProcessB refers to 's1' as 's2' within a single 'StateDependentStartupConfig' therefore it can stay as 'running' without 'terminated' and then 'created' again; but if two 'StateDependentStartupConfigs' used, it has to be terminated at first and then, be re-created.
All processes refer to the new state 's2' shall be created
CreateProcess(C)
ReportExecutionState(kRunning)
Shutdown / Restart the Machine
SetState(&MachineFG::Shutdown or ::Restart)
:future
Send request to OS
EM doesn't perform the Shutdown / Restart of an AP machine by itself but instead, sending to request to OS to start a special process which refers to the MachineFG state 'Shutdown' or 'Restart' to execute the request. During this phase the SM shall be alive and all state/process dependences shall be checked and solved orderly.

Adaptive Platform Functional Cluster	Abb.	Architectural category	Functional category	Log&Trace Context ID	Suggested Impl.-Design	Functionality and/or Responsibility (based on the definitions from the 'AP_EXP_PlatformDesign')
Adaptive Platform Core	core	Functional	Runtime	#COR	Library-based	Core Types defines common classes and functionality used by multiple Functional Clusters as part of their public interfaces. One of the rationale to define Core Types was to include common complex data types that are often used in the interface definition.
Communication Management	com	Functional	Communication	#COM	Library-based	The Communication Management is responsible for all aspects (intra- / inter-processes / machines) of communication between applications in a distributed real-time embedded environment.
Cryptography	crypto	Functional	Security	#CRY	Library-based	The FC Crypto offers applications and other Adaptive AUTOSAR Functional Clusters a standardized interface, which provides operations for cryptographic and related calculations.
Diagnostics	diag	Functional	Diagnostics	#DIA	Library-based	The Diagnostic Management (DM) realizes the ISO 14229-5 (UDSonIP) which is based on the ISO 14229-1 (UDS) and ISO 13400-2 (DoIP).
Execution Management	exec	Functional	Runtime	#EXE	Library-based	Execution Management is responsible for all aspects of system execution management including initialization of the Adaptive Platform and the startup/shutdown of Processes. Execution Management works in conjunction with the Operating System to configure the run-time scheduling of Processes.
Firewall	fw	Functional	n/a	#FWX	Library-based	The functional cluster Firewall is responsible for managing these rules and configuring the firewall engine.
Identity and Access Management	iam	Functional	Security	#IAM	Library-based	IAM introduces privilege separation for Adaptive Applications and protection against privilege escalation in case of attacks. It provides a framework for access control for requests from Adaptive Applications on Service Interfaces, Functional Clusters of the Adaptive Platform Foundation and related modeled resources.
Intrusion Detection System Manager	idsm	Functional	Security	#IDS	Library-based	The functional cluster Intrusion Detection System Manager (IdsM) provides a standardized interface for receiving notifications of security events (SEv). The SEvs can be reported by security sensors implemented in other functional clusters and adaptive applications.
Log and Trace	log	Functional	Runtime	#LOG	Library-based	The Log and Trace Functional Cluster is responsible for managing and instrumenting the logging features of the AUTOSAR Adaptive Platform. The logging and tracing features can be used by the platform during development as well as in and after production.
Network Management	nm	Service	communication	#NMX	Service-based	The AUTOSAR NM is based on a decentralized network management strategy, which means that every network node performs activities independently depending only on the NM messages received and/or transmitted within the communication system.
Operating System Interface	n/a	Functional	Runtime	#OSI	Library-based	The Adaptive Platform does not specify a new Operating System for highly performant processors. Rather, it defines an execution context and Operating System Interface (OSI) for use by Adaptive Applications, after POSIX PSE51.
Persistency	per	Functional	Storage	#PER	Library-based	Persistency offers mechanisms to applications and other functional clusters of the Adaptive Platform to store information in the non-volatile memory of an Adaptive Machine. The data is available over boot and ignition cycles. Persistency offers standardized interfaces to access the non-volatile memory.
Platform Health Management	phm	Functional	Safety	#PHM	Library-based	The Platform Health Management supervises the execution of software. It offers the following supervision functionalities (all supervision functions can be invoked independently): Alive supervision, Deadline supervision, Logical supervision, Health Channel Supervision, Inform State Manager about supervision failures, Trigger watchdog.
State Management	sm	Service	Runtime	#SMX	Service-based	State Management is responsible for all aspects of the operational state of the AUTOSAR Adaptive Platform, including handling of incoming events, prioritization of these events/requests to set the corresponding internal states.It may consist of one or more state machines depending on the project needs.
Time Synchronization	tsync	Functional	Communication	#TSY	Library-based	TS FC defines the APIs which can be used to provide Time Sychronization between different Applications and / or ECUs.
Update and Configuration Management	ucm	Service	Configuration	#UCM	Service-based	UCM is responsible for updating, installing, removing and keeping a record of the software on an Adaptive Platform. Its goal is to make AP be able to flexibly update the software and its configuration through over-the-air updates (OTA).
Raw Data Stream	rds	Functional	Communication	#RDS	Library-based	Adaptive AUTOSAR also provides a standalone Communication API for processing raw binary data streams towards an external ECU, e.g. a sensor in an ADAS system. The API is static and implements functionality for a client application to establish a communication channel to a server, and for a server application to wait for incoming connections from a client.

Notices

Due to space limitations (as I wanted to include everything in a single picture), not all basic elements could be illustrated. For example, several types of mappings, such as PersistencyToMachineMapping or ProcessToMachineMapping, are defined and used to connect related components. While these are not shown in the illustration, they are straightforward to understand for those who are working in the corresponding domain.