

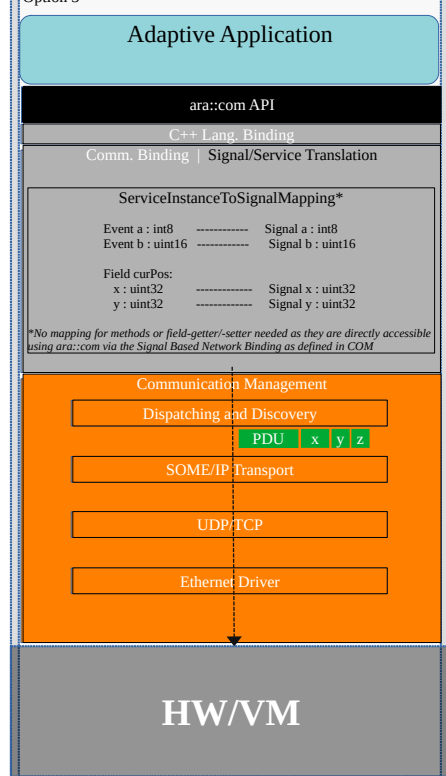
CP ← communication → AP

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 PDU's 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.

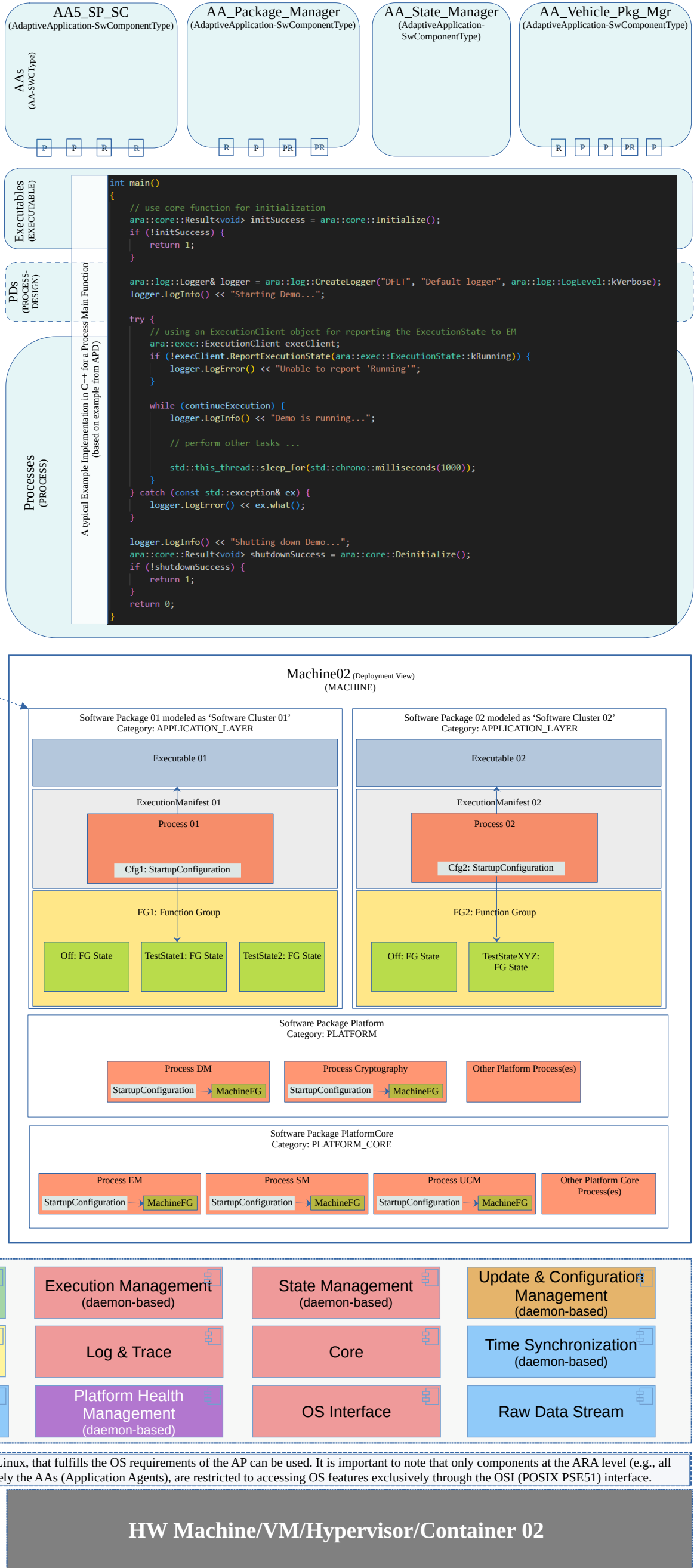
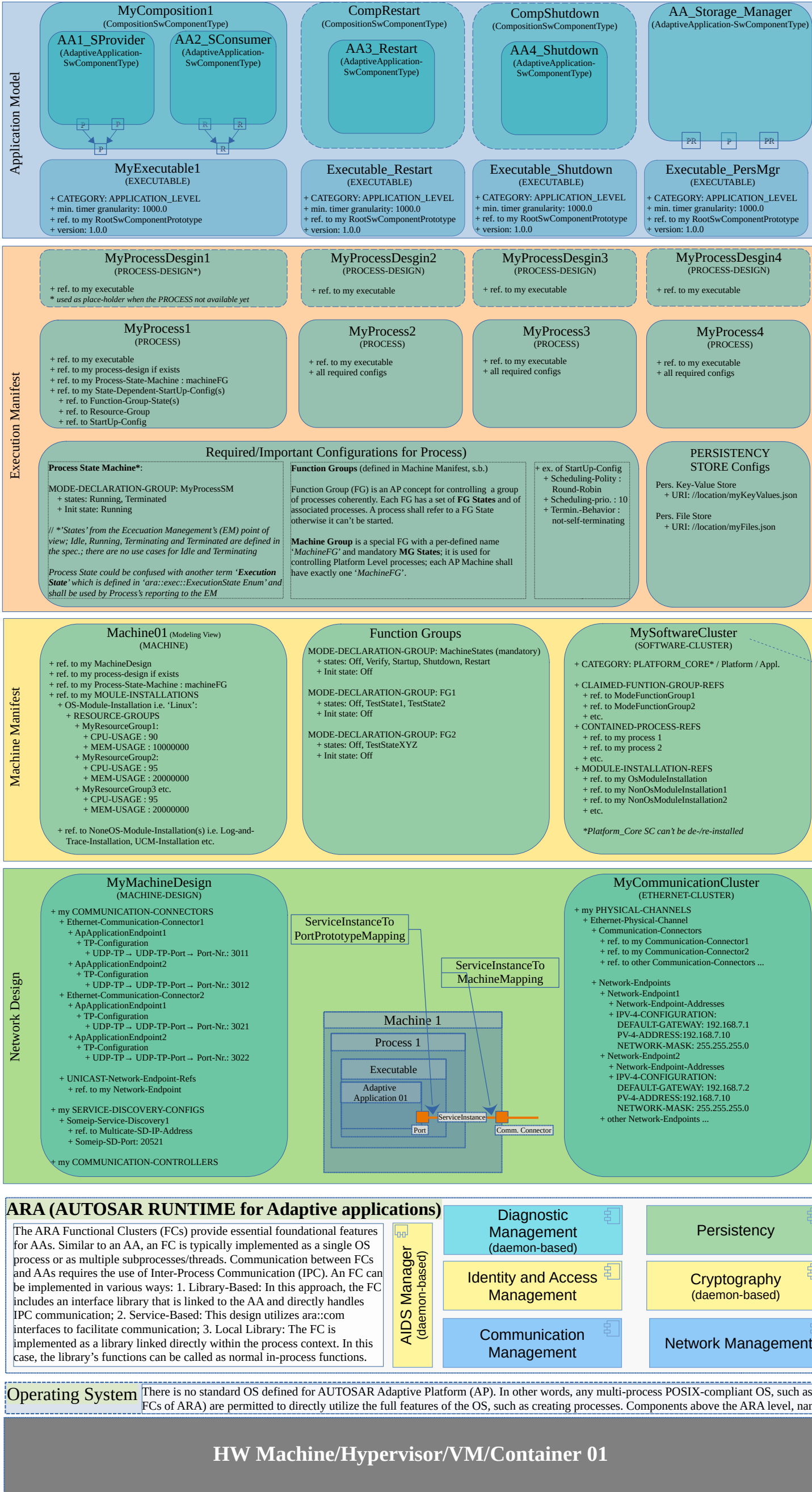
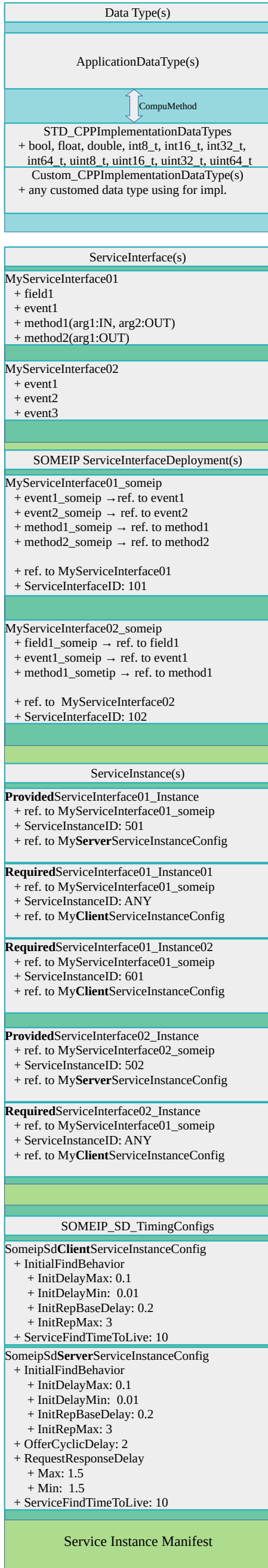
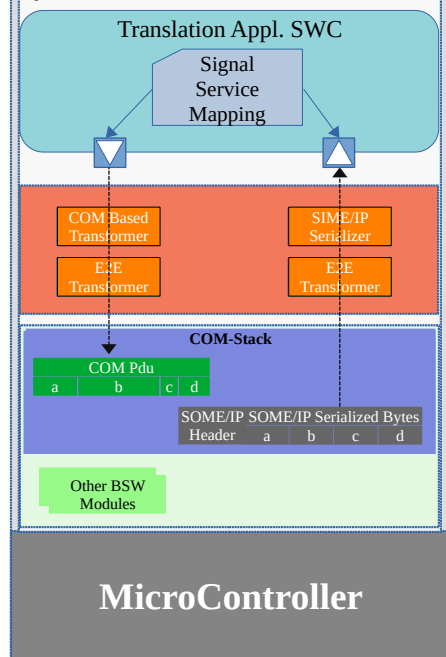
Machine (AP)



Gateway ECU (CP)



ECU (CP)



START, EXECUTION, SHUTDOWN / RESTART OF A MACHINE

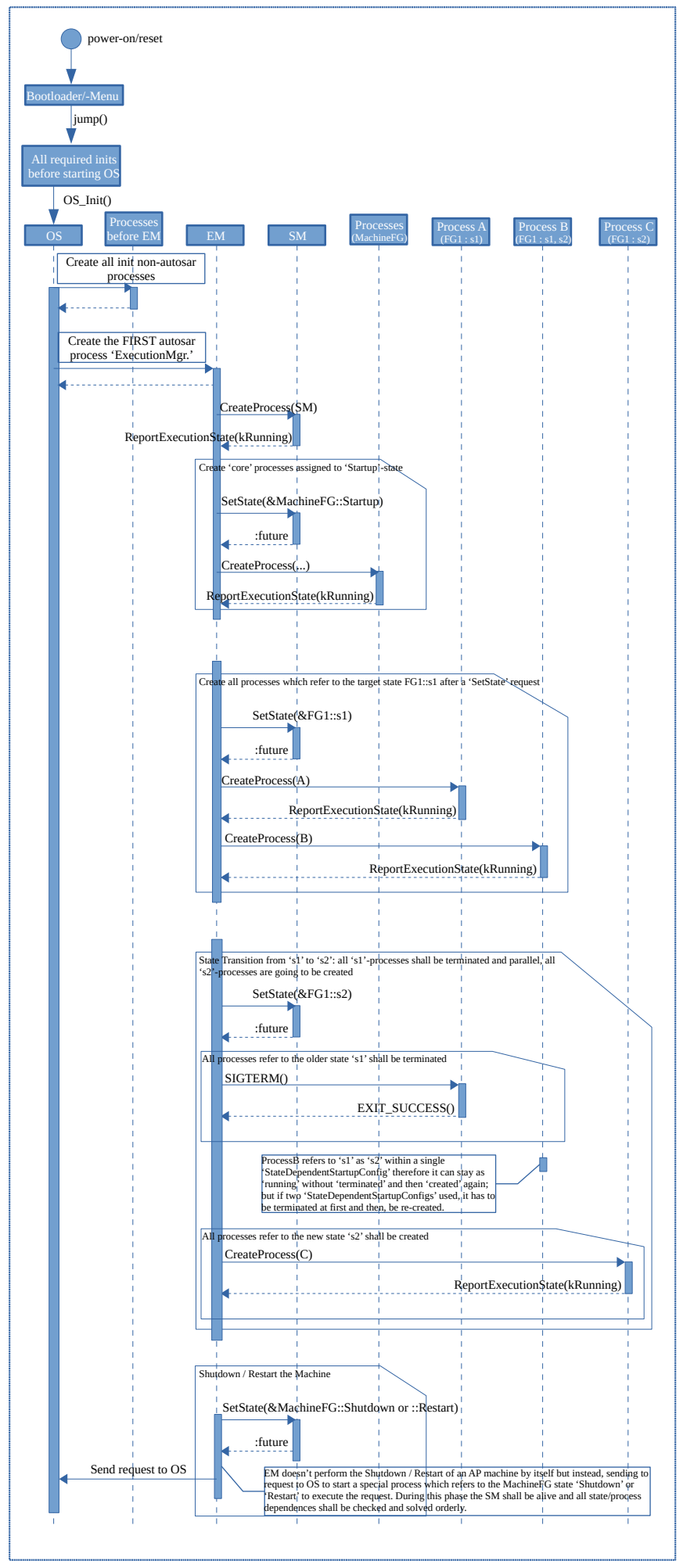
From the OS perspective, an AP Instance is essentially a collection of processes, which may include AAs (Adaptive Applications) operating above the ARA or FCs (Functional Clusters) running on the ARA, all executing in a multi-process environment.

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.



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

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 14229-1 (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 Intrusion Detection System Manager (IDSM) provides a standardized interface for receiving notifications of security events (SEs). The SEs 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	Functional	Communication	#NMX	Library-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	osi	Functional	Runtime	#OSI	Library-based	The Adaptive Platform does not specify a new Operating System for highly performance processors. Rather, it defines an execution context and Operating System Interface (OSI) for use by Adaptive Applications, after POSIX PSE51.
Persistence	per	Functional	Storage	#PER	Library-based	Persistence offers mechanisms to applications and other functional clusters of the Adaptive Machine. The data is available over boot and ignition cycles. Persistence 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	Functional	Runtime	#SMX	Library-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 Synchronization between different Applications and / or ECUs.
Update and Configuration Management	ucm	Functional	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 PersistenceToMachineMapping 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.