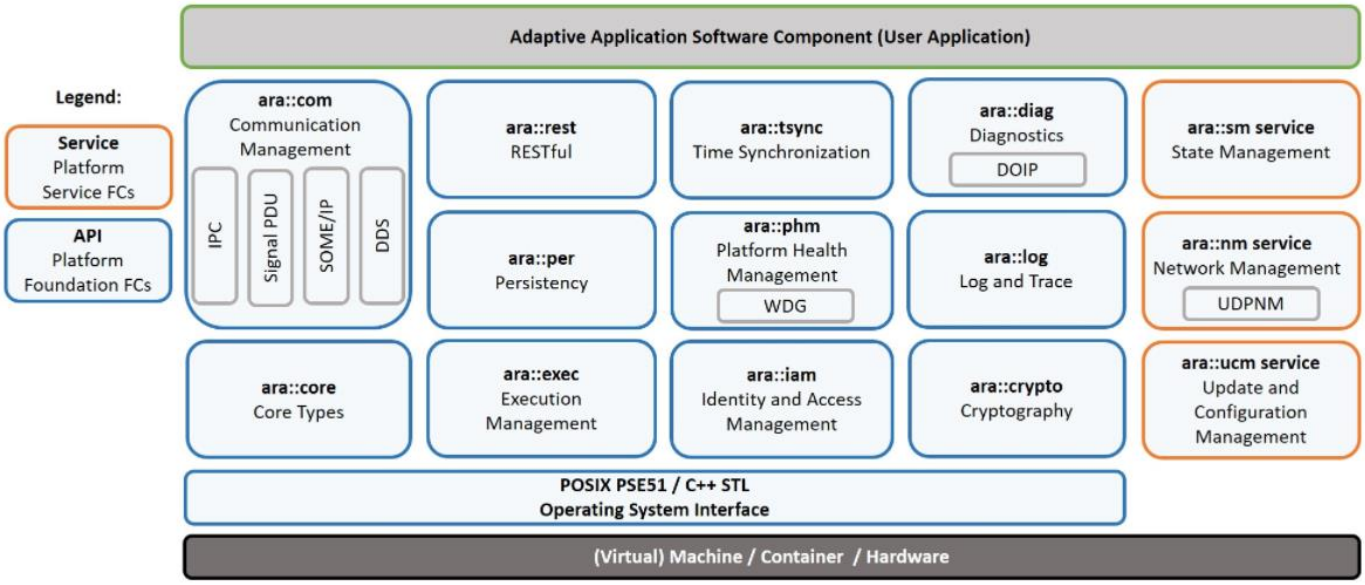
ADAPTIVE AUTOSAR

As the Adaptive Platform is AUTOSAR's solution for high- performance computing ECUs to build safety-related systems for use cases such as highly automated and autonomous driving.

The adaptive application can only use PSE51 POSIX profile APIS.



AUTOSAR Runtime for Adaptive applications (ARA):

ARA consists of application interfaces provided by Functional Clusters, which belong to either Adaptive Platform Foundation or Adaptive Platform Services.

Adaptive Platform comprises of the following Functional Clusters

Platform Health Management (PHM)

Communication Management (CM)

Diagnostics Management (DM)

Update and Configuration Management (UCM)

Log and Trace (L&T) , Cryptography (CRYPTO)

Execution Management (EM)

Persistency (PER)

Network Management (NM)

State Management (SM),

Time Synchronization (TS)

Adaptive Platform Foundation

The foundation consists of functional clusters, which can be loosely grouped into three categories: basic functionality, communication, and safety & security.

- Basic functionality

Execution Management (ara::exec) deals with the state of applications and the ECU as a whole. After power-on and kernel boot, it starts the system initialization and controls launch and termination of all other application software components.
So, in summary the Execution Management determines when to start or stop Processes.
Execution Management ensures that the authenticity of all Executables and Executable related data is checked.
Execution Management configures the operating system and its scheduling policy even at runtime.

Logging and Tracing (ara::log) provides APIs to Adaptive Applications and the rest of the platform to log events. It supports severity levels (debugging, informative, warning, error) and the concept of contexts and logger IDs to group and filter messages.

Persistency (ara::per) offers permanent storage for (read-only) configuration data, reading and writing of application-specific data formats. Feature-wise, it supports error detection and correction. Both file-based and key-value APIs are offered.

Time synchronization (ara::tsync) allows applications to set and retrieve a common time base. This functional cluster does not provide a time base itself, but allows its users to act either as a time provider (master) or requester (slave). Different time bases (local, global, synchronized) allow clients to either maintain their own time base, or be synchronized within or among different ECUs.

- Communication

Communication Management (ara::com) provides the basis for a service-based communication among applications and with platform services. It defines Service Interfaces, which are a flexible coupling mechanism to define provided and required interfaces.

Restful communication (ara::rest) offers an orthogonal communication paradigm that is less connection or service-based, but inspired by web-based stateless APIs. This functional cluster offers the basic building blocks to create Adaptive Applications that offer a REST-ful service (server part) or interact with another service (client part).

What is a REST-ful service?
1st source : shorturl.at/dtFGR
2nd Source, starting from page 46 in the adaptive autosar rest SWS : shorturl.at/kqEOV

- Safety & Security

Identity & Access Management (ara::iam) provides mechanisms akin to permission management, that allows to grant or deny access from Adaptive Applications to the Adaptive Platform Foundation and Services.

Cryptography (ara::Crypto) provides a standardized interface to access implementations of multiple cryptographic algorithms. These include random number generation, hashing algorithms as well as several symmetric and asymmetric cyphers.

Platform Health Management (ara::phm) allows to supervise Adaptive and Platform Applications by using the APIs of Execution Management. In case of failure, specific actions can be configured. Possible actions include restart of applications, or a state change within an application, function group or ECU.

Adaptive Platform Services

Diagnostics (ara::diag) offers services as specified in Unified Diagnostics (UDS, ISO 14229) and Diagnostics over IP (DoIP, ISO 13400). This includes the concepts of Diagnostic Events, i.e. errors, warnings or failures, and Diagnostic Communication, which deals with the transfer of the above mentioned events from the source to a persistent storage.

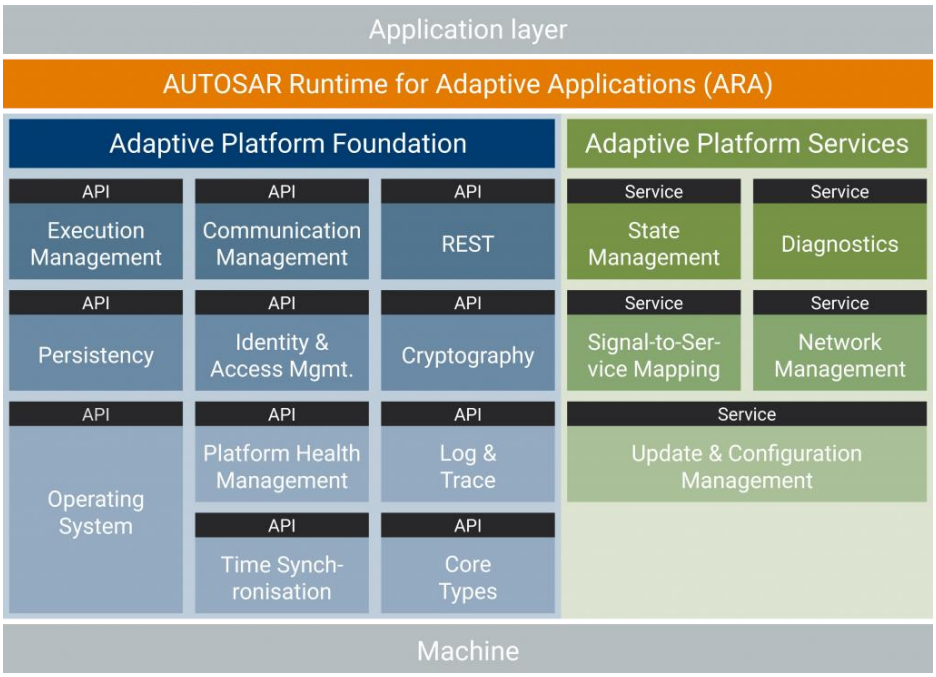
Signal-to-Services Mapping (ara::s2s) provides a compatibility layer towards the Classic Platform. It translates signals from the Classic to services in the Adaptive Platform. So interoperability between CP and AP software components can be realized.

Network Management (ara::nm) deals with the status management of any connected bus, possibly keeping the bus alive if needed for time-critical communication.

State management (ara::sm) is responsible for defining the current active state of the ECU. In case of a state change request, it triggers Execution Management for performing the necessary changes, as Execution Management performs the transitions between different states.

Update & Configuration Management (ara::ucm) provides means to install new and upgrade existing software components on ECUs. Both over-the-air (OTA) and workshop-based (through diagnostic devices) use cases are supported.

(Most Important cluster, It can be seen as a measure to the strength of the adaptive AUTOSAR).



Adaptive Application run on top of the ARA
=> ARA consists of application interfaces provided by Functional Clusters
Adaptive Application can provide Services (Non-platform service) to other Adaptive Applications
=> Adaptive Applications and Functional Clusters may use non-standard interfaces, but ensure not conflict with the standard AP functionalities and portability onto other AP implementations

ARA implementation ?
- The language binding is based on modern C++
- The C++ Standard library
- Only PSE51 interface (OS API)

Life-cycles of Adaptive Applications ?
=> Application launch and shutdown
- Managed by EM (Loading/launching an AA)
  => all the Functional Clusters are applications from EM point of view
- To launch an Adaptive Application it needs configuration at system integration time or at runtime.

Interaction between Adaptive Applications?
- Service requests/replies
  => uses CM (Communication Management) for both intra-machine and inter-machine and transparent to applications.

Interaction between Functional Clusters and Adaptive Applications?

- uses IPC under "Library-based" design or "Service-based" design
  => "Library-based" calls IPC directly, Uses "Library-based" locally in an AP instance
  =>"Service-based" uses CM functionality (AA <=> Cm or AA <=> Server proxy <=> CM)

Uses "Service-based" for other AP instance (distributed fashion)

IPC: Inter Process Communication

Interaction between Functional Clusters ?

- use "public" ARA interfaces of other Functional Clusters
- use "protected" interfaces of Functional Clusters to provide privileged access

AP Operating System point of view ?

=> required to provide multi-process POSIX OS capability
=> the AP and AA are just a set of processes
=> Each AA is implemented as an independent process
=> Functional Clusters are implemented as processes
=> These processes interact with each other through IPC or OS functionalities
=> The Adaptive Platform Services and the non-platform Services are also implemented as processes
=> These processes can have single-threaded or multi-threaded
=> If AAs running on top of ARA, they should only use PSE51.
=> If more features are needed they will be taken from the POSIX standard
=> If a process is the Functional Clusters, it can use available OS API or further POSIX calls
=> The use of specific calls will be left open to the implementer and not standardized.
=> the OS provide interfaces, such as creating processes, that are required by Execution Management to start an Application
=> They are platform implementation dependent (not belong to ARA)
=> OSI provides both C and C++ interfaces.
=> C program includes C function calls defined in the POSIX standard, namely PSE51 defined in IEEE1003.13 [1]
=> C++ program includes function calls defined in the C++ Standard and its Standard C++ Library

Scheduling ???

- Use standard scheduling policies of POSIX standard : SCHED_FIFO and SCHED_RR
- Other scheduling policies are allowed, but check portablity across different AP implementations

AP runs on as a Machine/hardware ???

=> a hardware can be one or more Machines and a single instance of AP runs on a machine
=> hardware may be virtualized using Hypervisor

Memory management ???

Ref: https://stackoverflow.com/questions/12488010/why-the-entry-point-address-in-my-executable-is-0x8048330-0x330-being-offset-of

- The process's address space is virtualized.
- Each process has its own address space (the addresses where code and data are located may or may not correspond to their underlying physical storage address)

Execution Management ???

- Working in cooperation with OS
  => responsible for platform initialization and the start-up/shut-down of Applications
  => run-time scheduling of Adaptive Applications
- Initialization process : OS first -> Execution Management -> Platform-level Applications -> AAs
- The startup order of the Platform-level Applications and the AAs are based on Machine Manifest and Application Manifest information
- Depending on the Machine State, deployed AAs are started during Platform-level Applications startup or later

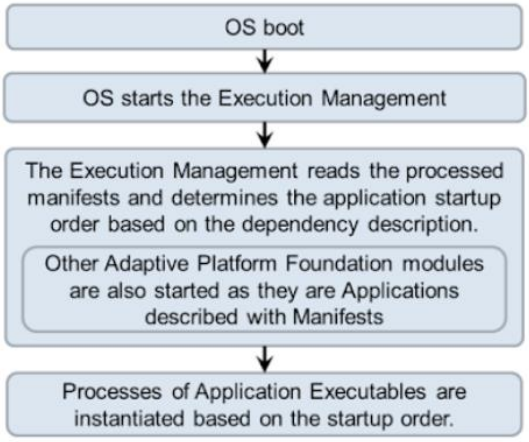
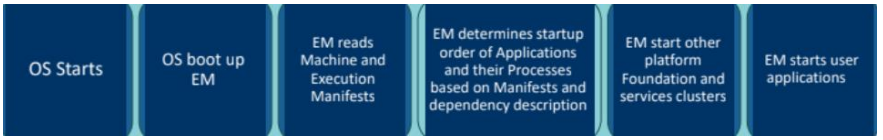
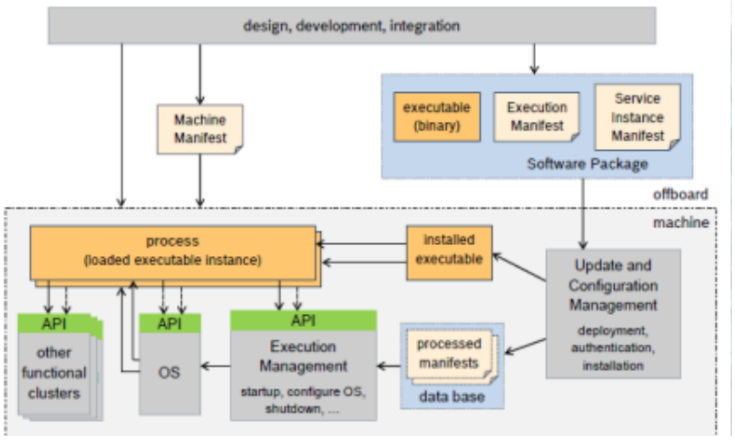


Figure: AP start-up process



Startup sequence repeated in another way..

- When the Machine is started, Execution Management is launched as the Machine's first process.
- Other functional clusters and platform-level Applications of the Adaptive Platform Foundation and Adaptive Platform Services are then launched by Execution Management.
- After the Adaptive Platform Foundation and Adaptive Platform Services are up and running, Execution Management continues to launch user-level Applications.
- Please note that an Application consists of one or more Executables.
- Therefore to launch an Application, Execution Management starts Processes as instances of each Executable.
- The startup order of the platform-level Processes shall be determined by Execution Management based on Machine Manifest and Execution Manifest information



Machine Manifest

Is a file to configure a Machine. The Machine Manifest holds all configuration information which cannot be assigned to a specific Executable or Process.

Execution Manifest

The Execution Manifest allows to define in which states the Processes have to run.

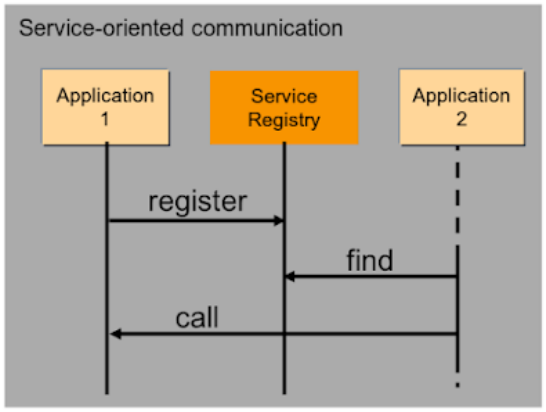
Two or more processes that reference the same executable. (Need to read more)

Machine State Management ???

- Machine State Management define the state of the operation for an Adaptive Platform.
- The Application Manifest allows definition in which Machine State the Application Executables shall run
- Grants full control over the set of Applications to be executed
- Composed of mandatory and non-standardized machine states
- Machine State Management can be integrated directly in the EM or as a separate Machine State Management Application.

Communication Management ???

=> how a defined service is presented to the application implementer (upper layer, Language Binding) as well as the respective representation of the service's data on the network (lower layer, Network Binding).



- Service Oriented Communication (intra-machine communication and inter-machine communication)
  + A service consists of : Events, Methods, Fields
  + Need a Service Registry acts as a brokering instance (services registers, services querying-Service Discovery)

Diagnostics???

- The Diagnostic Management realizes the ISO 14229-5 (UDSonIP) which is mainly based on the ISO 14229-1 (UDS) and ISO 13400-2 (DoIP).

Persistency ???

=> Offers a library based approach to access the non-volatile memory for Applications to store data over boot and ignition cycles.

- Fulfill the PSE51 requirements not to create or delete files during runtime
- Currently the used storage location identifiers are simply file names

=> 2 categories of storage locations: Key-Value Storage and Stream Storage

- Key-Value Storage : store and retrieve multiple Key-Value pairs in one storage location.
  + The supported value types are base types, PODs (C++ Plain Old Data structures) and arrays/containers derived from these types, AUTOSAR data types
- Stream Storage : raw access to file like structures

Safety ???

- protect the exchange of information inside the vehicle and with the external world including fault detection if any corruption has occurred but no mechanisms to guarantee the integrity of data (e.g: E2E-Protection)
- monitor the correct execution of platform functionalities and AAs
- safe and secure usage of programming languages
- Watchdog functionality : Alive supervision, Deadline supervision, Logical supervision, Error handling of supervision errors

Extra On E2E (End to end encryption):

End-to-end encryption is intended to prevent data being read or secretly modified, other than by the true sender and recipient(s). The **messages** are encrypted by the sender but the third party does not have a means to decrypt them, and stores them encrypted.

Among the several enhancements in AUTOSAR Release 4.0 is the addition of an End-to-End (E2E) Communication Protection Library. This library defines several E2E profiles, each of which implements a combination of End-to-End protection mechanisms such as sequence counters, data IDs and CRCs. Two of these profiles, Profiles 1 and 2, are intended to protect inter-ECU communication via databus systems like FlexRay or CAN, and are designed to address various communication faults.

Process Lifecycle & Execution States

Characterizes the internal lifecycle of a Process. In other words, they describe it from the point of view of a Process that is executed. The states visible to the Process are defined by the ExecutionState enumeration.

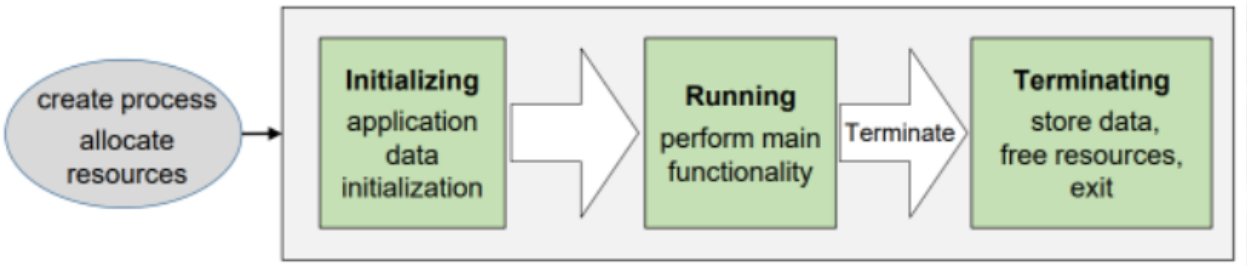
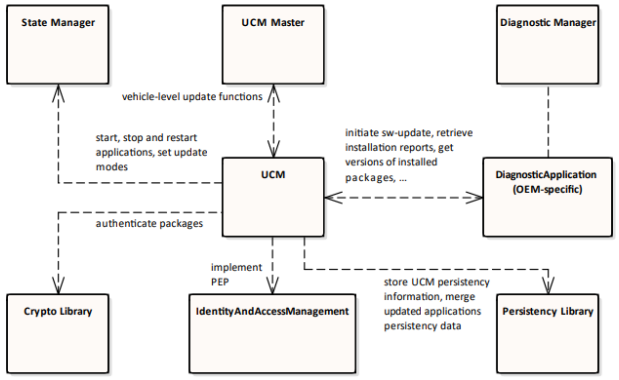


Figure 7.3: Execution States

-EM considers the initialization complete when process reports kRunning state.
-EM shall initiate process termination by sending SIGTERM signal to process
-EM considers process termination when process reports kTerminating state

More on the UCM cluster

UCM has multiple interfaces to other functional clusters on the Adaptive Platform as depicted in the figure below. The following subsections describe the implemented as well as the planned interfaces.

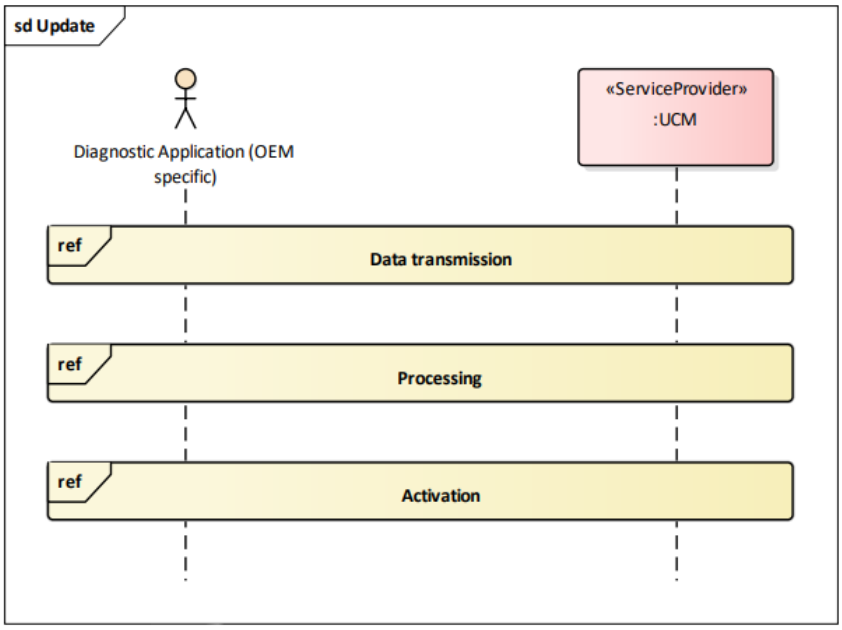


Update process done on three phases:

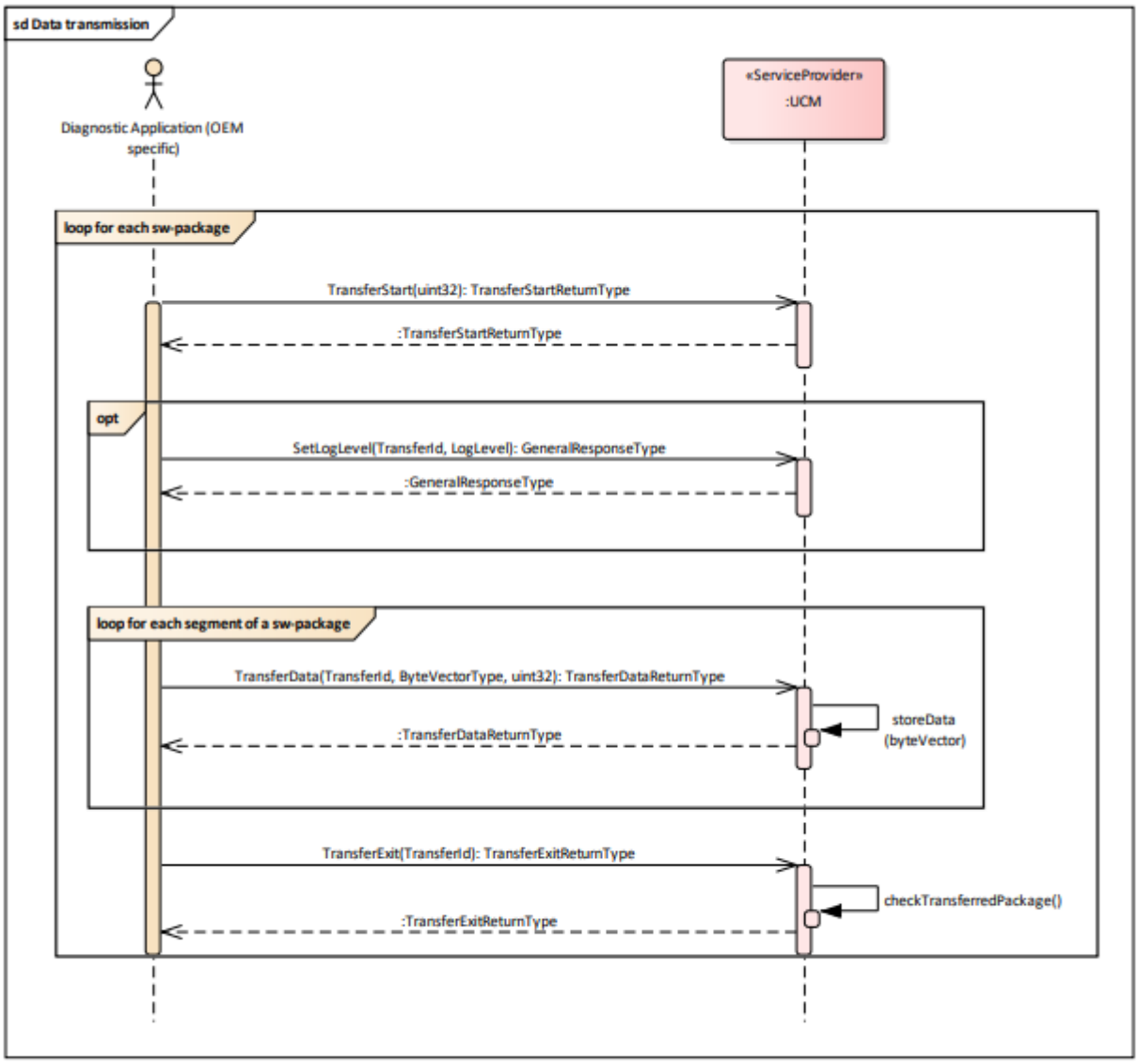
1. Data Transmission

2. Data Processing

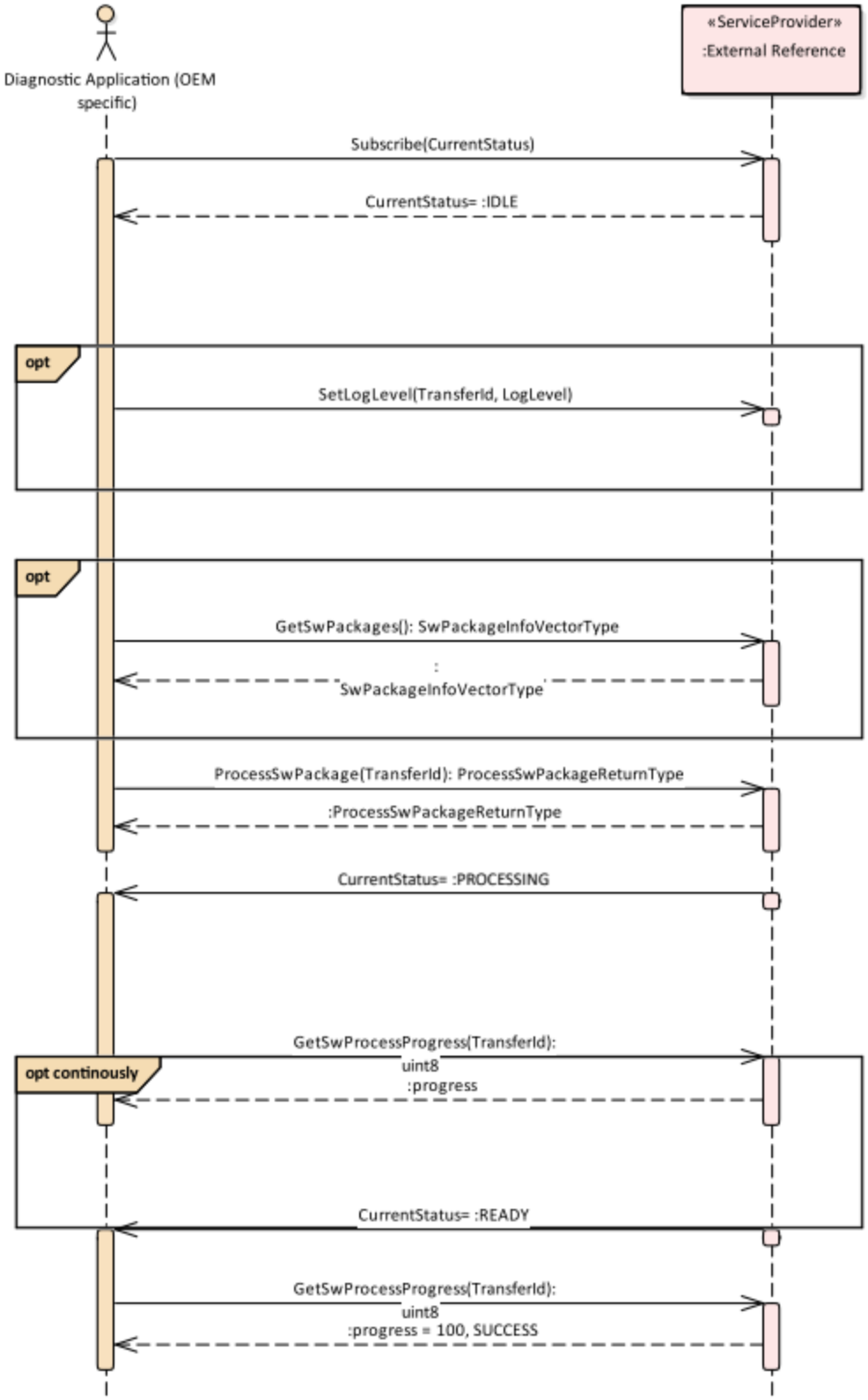
3. Activation



Sequence diagram showing the update process



Sequence diagram showing the data transmission

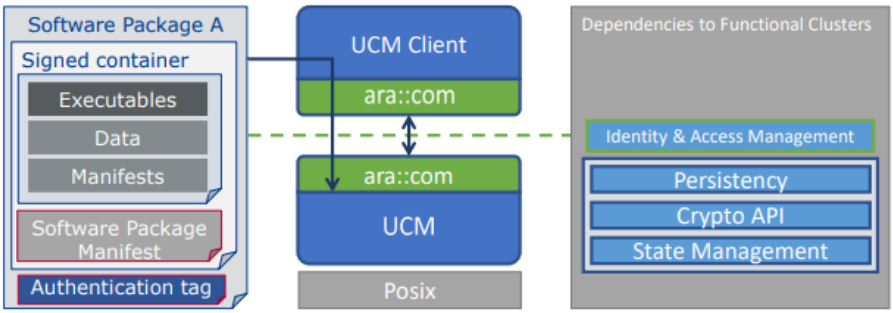


Sequence diagram showing the package processing

Dependencies of the UCM :

- Interfaces to Adaptive State Management
- UCM service over ara::com
- Interfaces to Adaptive Crypto Interface

The UCM functional cluster expose services to client applications via the ara::com middleware.



2.1.3 Persistency

UCM currently has no implemented interface towards Persistency. Later releases may contain functionality to merge existing persistency data of an Adaptive Application with persistency data supplied with a Software Package.

2.1.4 Diagnostics

In order to support Software Clusters having an own diagnostics target address UCM needs to have interfaces to functional cluster Diagnostics. These interfaces shall be used to inform Diagnostics about newly installed or removed Software Clusters so that Diagnostics can properly handle requests targeted to them.

As UCM currently only implements Software Clusters without a dedicated diagnostics target address these interfaces are neither defined nor implemented.

2.1.5 Identity and Access Management

UCM currently has no implemented interface towards Identity and Access Management. An implementation of a Policy Enforcement Point (PEP) is planned for later releases.

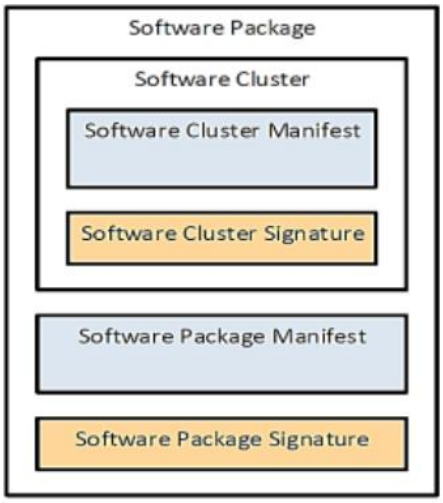
2.1.6 Crypto library

UCM needs to verify authentication of received packages using the ara::crypto library.

As the crypto library is not yet available this is planned for later releases.

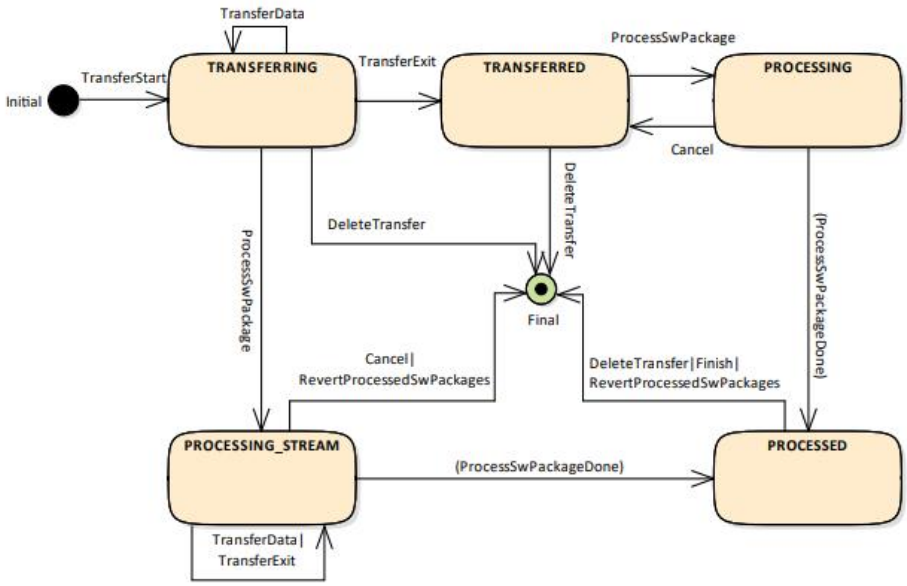
Software Package Structure

- Software package consists of Software cluster in addition to software cluster manifest.
- There is only one SW cluster per SW package.
- SW Cluster is “OEM/your app”
- There is map one to one between software package and software cluster, while software cluster can have more than one executable.
- Manifest files are used for configuration and check properties of each software package and cluster.
  1- **Software cluster manifest** contains the info (executable path/manifest)for each executable in this cluster
  2- **Software package manifest** contains info for all package like (should ... install or delete)
- Software Package Signature: confidentiality is used to make sure that only the intended device can use the software package.
  - One of the techniques is ‘Crypto Technique’



Transferring SW Package

- Each Software Package gets its own state as soon as it is being transferred to UCM.
- The state machine specifies the lifecycle of a Software Package that is transferred to and processed by UCM.
- During this lifecycle, a Software Package is uniquely identified with an id that UCM provides to the client.



Once Software Packages are transferred to UCM, they are ready to be processed to finally apply changes to the AUTOSAR Adaptive Platform.