

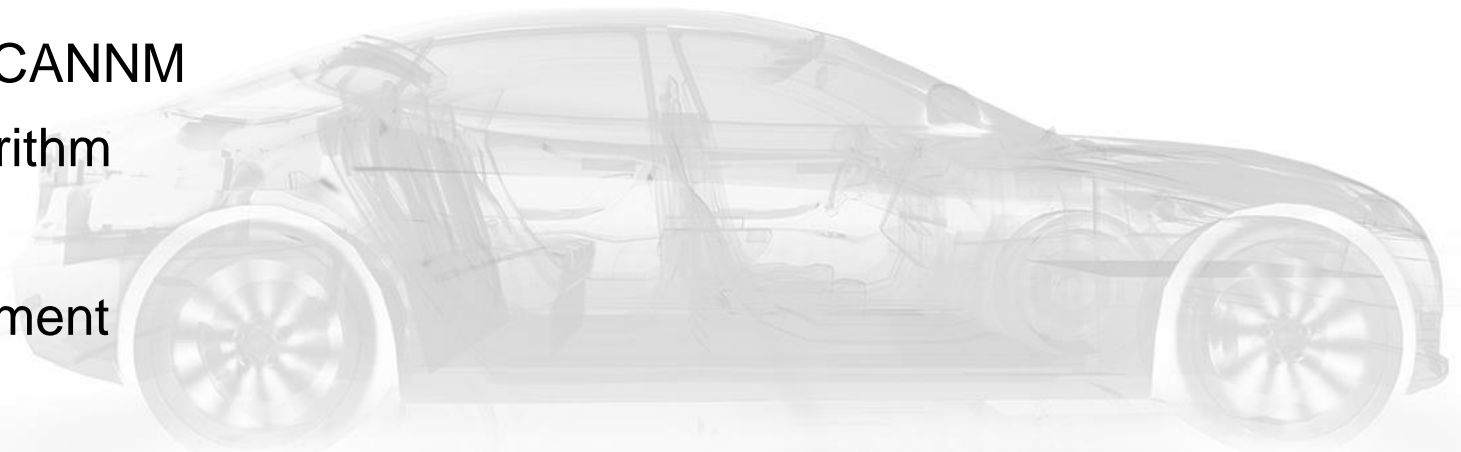
NIT **AUTOMOTIVE SOFTWARE** **WITH AUTOSAR**



AUTOSAR

Agenda

- › Mode Manager Concept
- › Mode Management
- › Wakeup Handling
- › BSWM configuration
- › ECUM configuration
- › COMM, CANSM, NM, and CANNM
- › Network Management Algorithm
- › Service Mapping
- › Exercise 4 - Mode Management

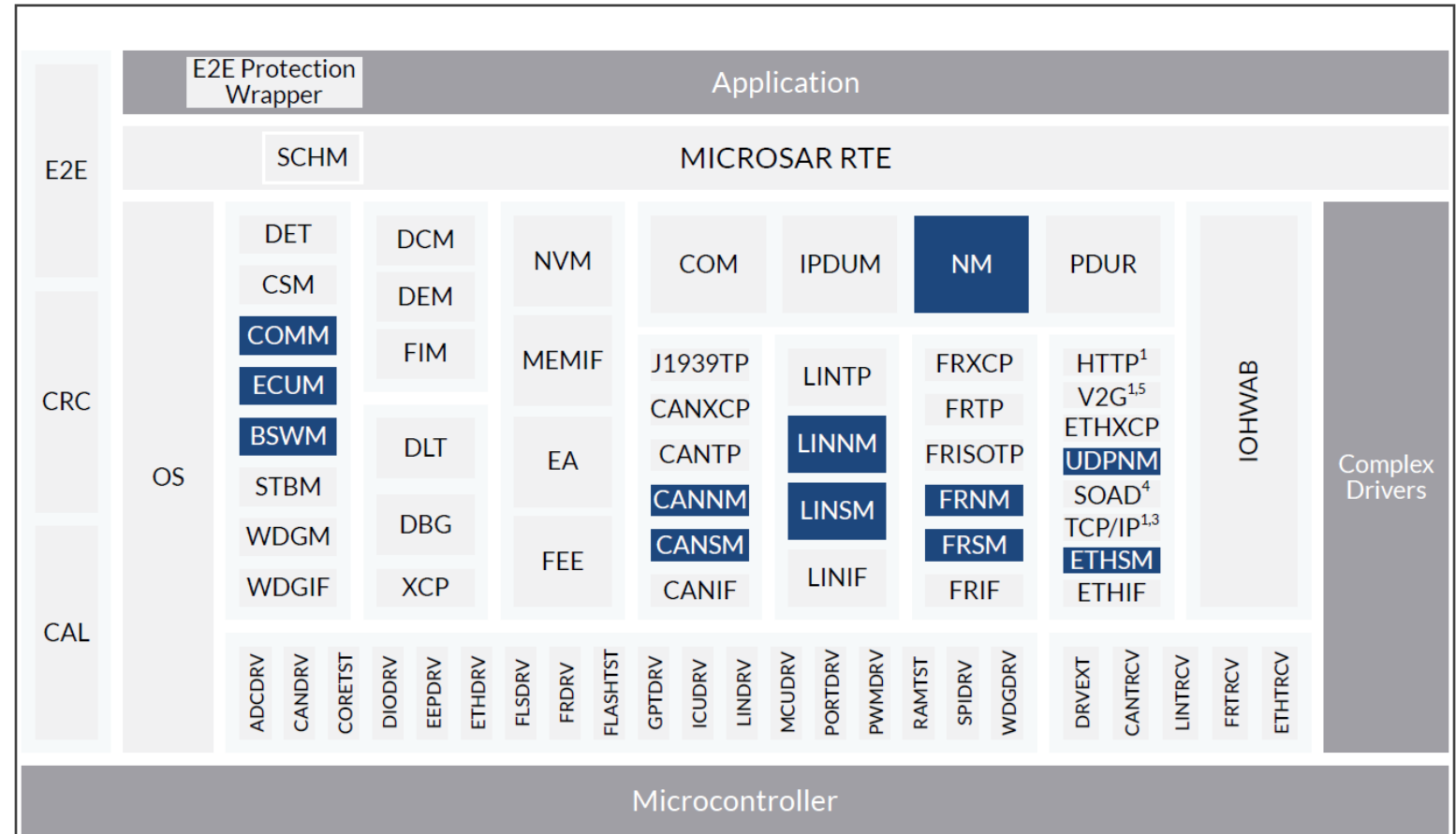


Mode Management

AUTOSAR BSW Modules for Mode Management

Affected BSW modules

- › ECUM
- › BSWM
- › COMM
- › NM
- › CANNM
(FRNM, LINNM, UDPNM)
- › CANSM
(LINSM, FRSM, ETHSM)



Mode Management

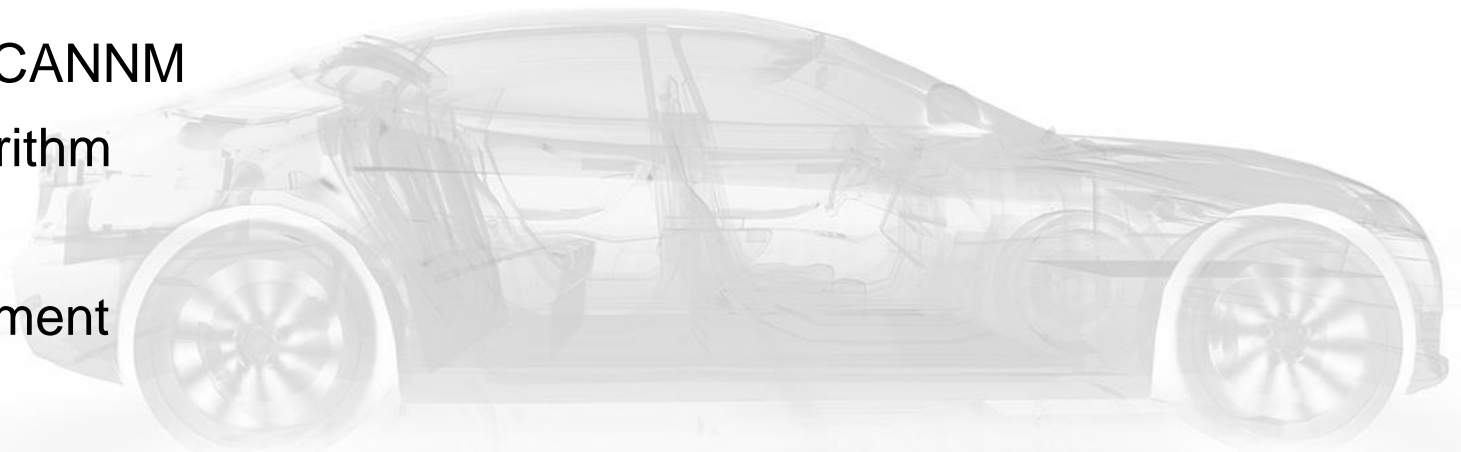
Motivation

- › The BSW shall satisfy specific requirements for an ECU
 - › "Wakeup and Sleep" functionality in the following cases, for example:
 - › ECU operation (μ C and local peripherals)
 - › Communication channel handling (vehicle networks)
- › Abstraction of technology from the application
(same treatment of different basic technologies like CAN, LIN, FR, ETH)
- › Generic concept for (both user-defined and BSW-defined) mode management
 - › Mechanisms for notification
 - › Service requests for mode changes (e.g., by SWCs)
 - › Mode arbitration (decision on how to handle transitions)
 - › Mode control (execution of mode transitions)

Agenda

› **Mode Manager Concept**

- › Mode Management
- › Wakeup Handling
- › BSWM configuration
- › ECUM configuration
- › COMM, CANSM, NM, and CANNM
- › Network Management Algorithm
- › Service Mapping
- › Exercise 4 - Mode Management



Mode Manager Concept

What is a Mode Manager?

- › We can distinguish the following roles
 - › Mode Manager, Mode User, Mode Requester
- › A **Mode Manager (MMgr)** is an arbitrary SWC or BSW module which has
 - › A set of one or more modes which the Mode Manager **provides** as published information to other SWCs (Mode Declaration Group(s))
 - › The ability to signalize current mode (state) and mode changes (transitions) over its Port Interface
 - › An implementation of the finite state machine corresponding to the published modes
- › **Mode User (MUsr)**
 - › Arbitrary SWC or the BSWM
 - › May **require** / subscribe the transition or state information
- › **Mode Requester (MRqr)**
 - › Arbitrary SWC or the BSWM with a Mode Request Port

Mode Manager Concept

Elements and mechanisms

› **Mode Declaration Group**

- › Set of supported modes by the MMgr
- › Available through Port Interface
- › Initial Mode necessary before any switching

› **Mode Switch Event**

- › Occurs upon transitions between modes
 - › On Mode Exit
 - › On Transition
 - › On Mode Entry

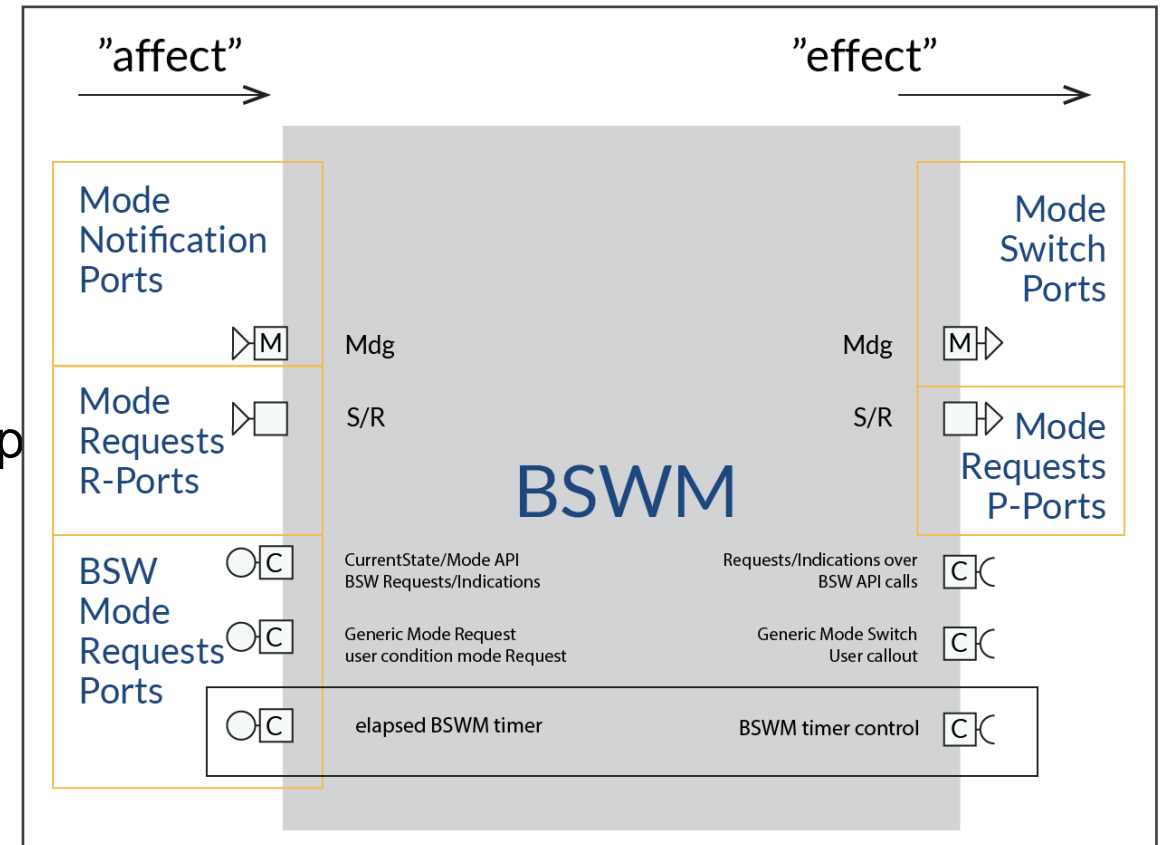
› **Mode Disabling Dependency**

- › Prevent a Runnable from being executed during a special mode state

Mode Manager Concept

Basic Software Mode Manager BSWM

- › **Mode Notification Port (R-Port)**
 - › SWC indicates Mode to BSWM over a Mode Declaration Group
- › **Mode Switch Port (P-Port)**
 - › BSWM notifies Mode switch to a SWC using a Mode Declaration Group
- › **Mode Request R-Port**
 - › SWC requests Mode from BSWM over S/R
- › **Mode Request P-Port**
 - › BSWM requests Mode from a SWC Mode Manager over S/R

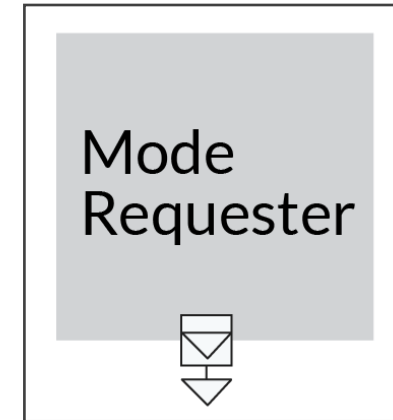


Module representation of the BSWM. The Ports labelled with "C" are only Standardized Interfaces or C APIs. Mode Ports (Mdg) or S/R Ports are AUTOSAR interfaces. Timers are only used internally in the BSWM; therefore, they are printed in grey.

Mode Manager Concept

Role: Mode Requester

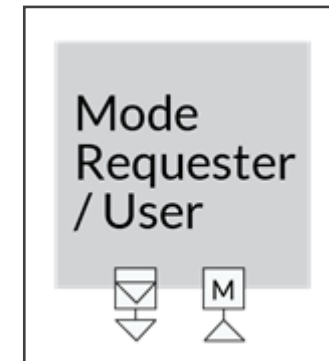
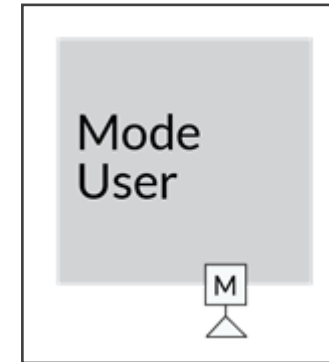
- › The **Mode Requester**
 - › A SWC can request a mode switch from the MMgr over a **Mode Request P-Port Prototype** (a Sender/Receiver communication)
 - › No special annotation in Developer
 - › AUTOSAR restriction: isService = TRUE
- › The Mode Requester doesn't need to be an SWC:
 - › In case the Mode Requester does not have a Mode Request port, it can use the `BswM_RequestMode()` API (Generic Mode)



Mode Manager Concept

Role: Mode User

- › The **Mode User**
 - › The Mode User has a **Mode Switch R-Port Prototype** to be informed about mode changes
 - › Can invoke `Rte_Mode` API to get knowledge about a Mode Change
 - › A Runnable can be triggered by the Mode Switch
 - › On Exit, On Transition, On Entry
- › Frequently, a Mode User is also a **Mode Requester**
 - › e.g., with a Mode Request P-Port Prototype (a Sender/Receiver communication)

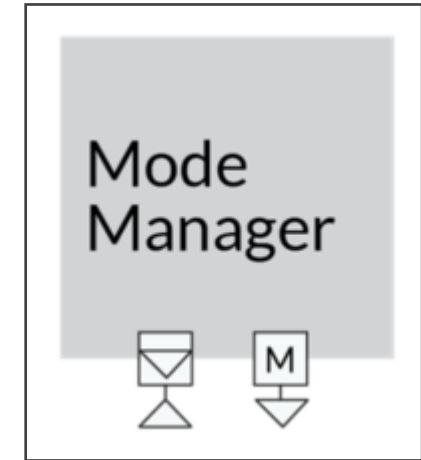


Note: In ASR3.x., the `Rte_Feedback` API could be used for modes by Mode users. In ASR 4.x. this is no longer the case. In ASR4.x. the Feedback API is dedicated to `DataSendCompletion` only.

Mode Manager Concept

Role: Mode Manager

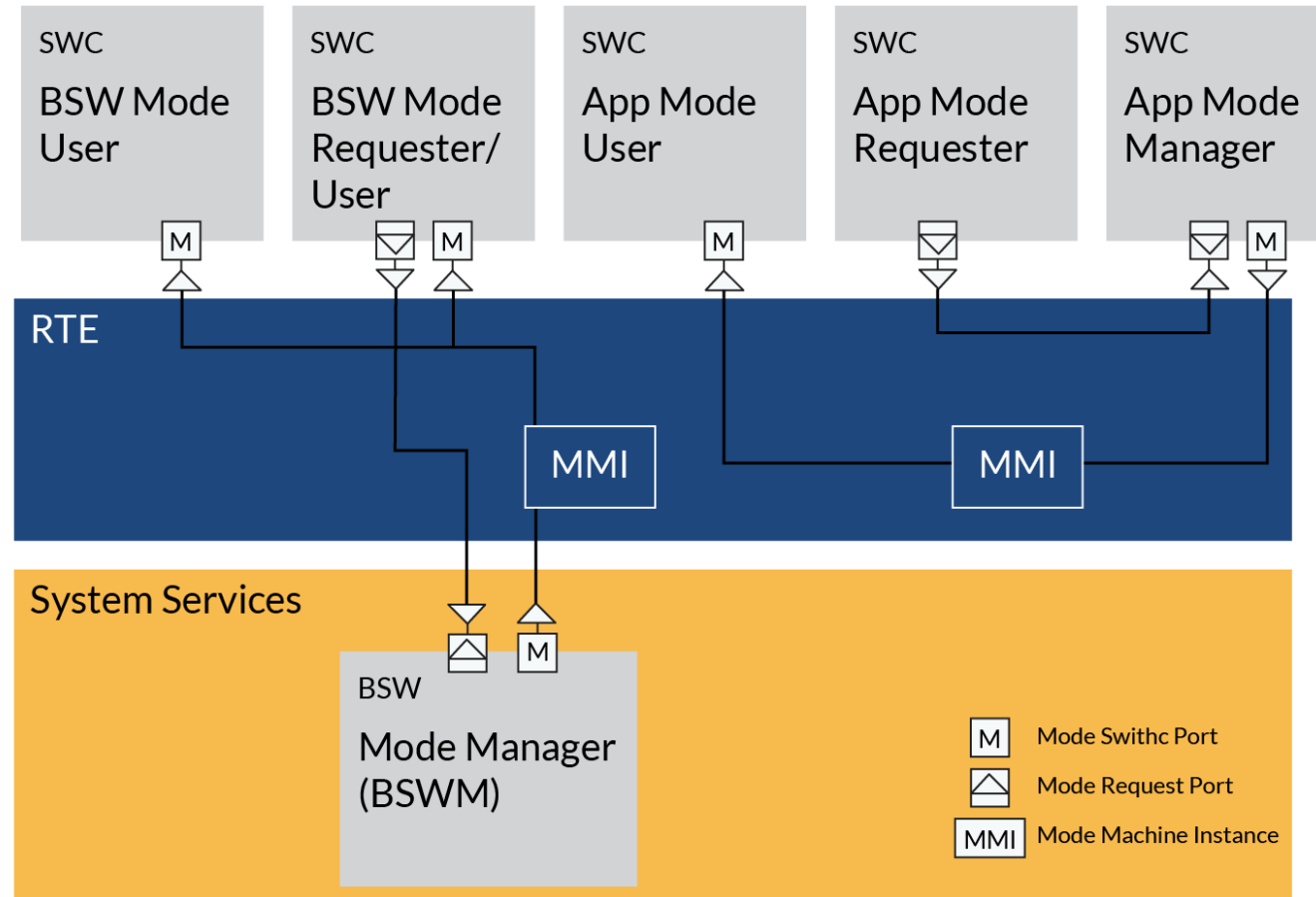
- › The **Mode Manager** uses
 - › `Rte_Switch` API to indicate the switch into a requested mode to the RTE
 - › The RTE keeps a Mode Machine Instance (MMI) to reflect the mode system-wide.
 - › `Rte_SwitchAck` API to get a Mode Switch Acknowledge in a non-blocking or blocking way.
 - › Additionally, `Rte_Mode` can be called to query the current MMI state in a non-blocking way.
- › The RTE indicates the Mode Switch over a **Mode Switch Port** (containing a Mode Declaration Group) to Mode Users as soon as the RTE initiates the mode transition.



Of course, the Mode manager can also use the `Rte_Mode` API to get the knowledge about the current MMI state.

Mode Manager Concept

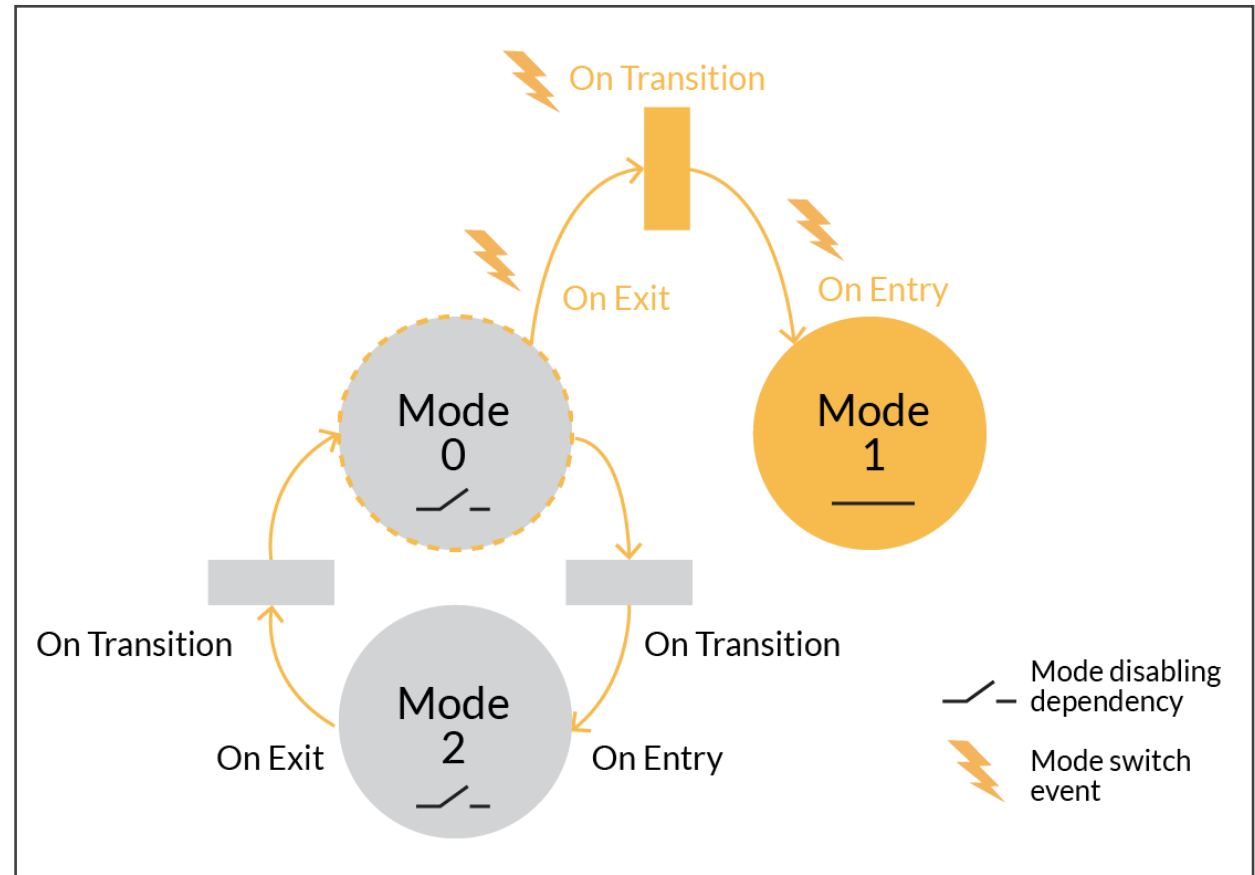
Mode Manager vs. Mode Users/Requesters



Mode Manager Concept

Mode Switching

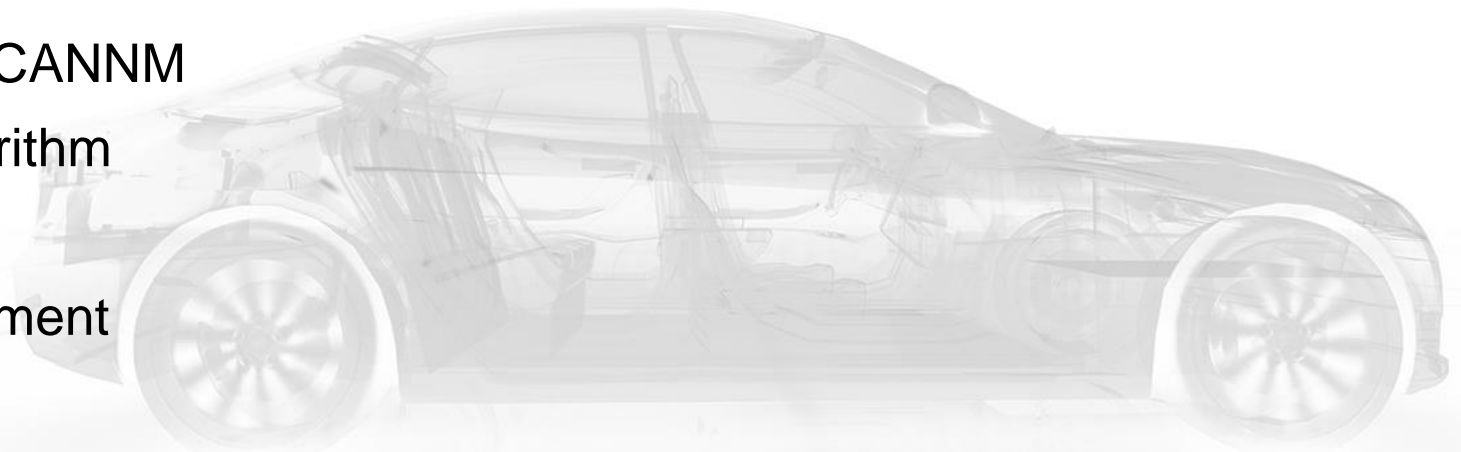
- › Mode Switch
 - › On Exit
 - › On Transition
 - › On Entry
- › Duration of Mode Disabling
 - › From On Exit of last mode until Mode itself
- › Mode Switching procedure
 - › Synchronous or Asynchronous



Why does Exit come before Entry? Cleaning up from last mode is necessary prior to switching to a new one!

Agenda

- › Mode Manager Concept
- › **Mode Management**
- › Wakeup Handling
- › BSWM configuration
- › ECUM configuration
- › COMM, CANSM, NM, and CANNM
- › Network Management Algorithm
- › Service Mapping
- › Exercise 4 - Mode Management



Mode Management

ECU State Manager

ECUM

- › Manages fundamental ECU phases like
 - › **OFF**, **UP** and **SLEEP**
 - › The necessary transitions between them
- › Initializes and de-initializes BSW, OS, and RTE
- › Wakeup event handling
- › Coordination of RUN requests from SWCs and BSWM
- › Two startup and two shutdown phases

Mode Management

Basic Software Mode Manager BSWM

BSWM

- › The BSWM is a generic, **rule-based** service module
 - › Used for mode arbitration and mode control
- › It can have a configured set of **rules** containing **default states, conditions, and actions**
 - › Each **condition** evaluates a set of **logical expressions***
 - › Evaluation result for a rule is either TRUE or FALSE
 - › **Action list(s)** must be given for at least one evaluation result per mode
 - › For example: call existing APIs, do a notification, perform a user-defined callout

Mode Management

Basic Software Mode Manager BSWM

1. BSW Modes (Standardized C-API)

- › `BswM<BSW>currentState/Mode` Standardized C language API
- › Functionality built into the Basic Software. Can be *Mode Switch Notifications* from BSW Modules acting as Mode Managers, but also *Mode Requests*, e.g., by the LINTP to switch the LIN schedule in the LINSM

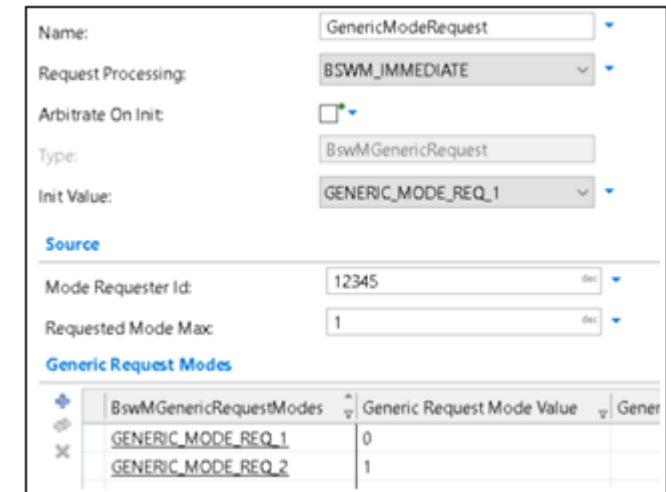
2. SWC Modes

- › `Rte_Switch / Rte_Mode` API, AUTOSAR Interface, i.e., ports
- › The SWC notifies its mode switches to the BSWM over a *Mode Declaration Group*, either directly or indirectly. The BSWM rules which react on the Mode Switch can trigger an action list.

3. Generic Modes (not using the RTE)

- › User defined C API, e.g.

`BswM_RequestMode (MRqrId0, BswMGenericRequestMode_001)`



The screenshot shows the configuration for a Generic Mode Request. The fields are as follows:

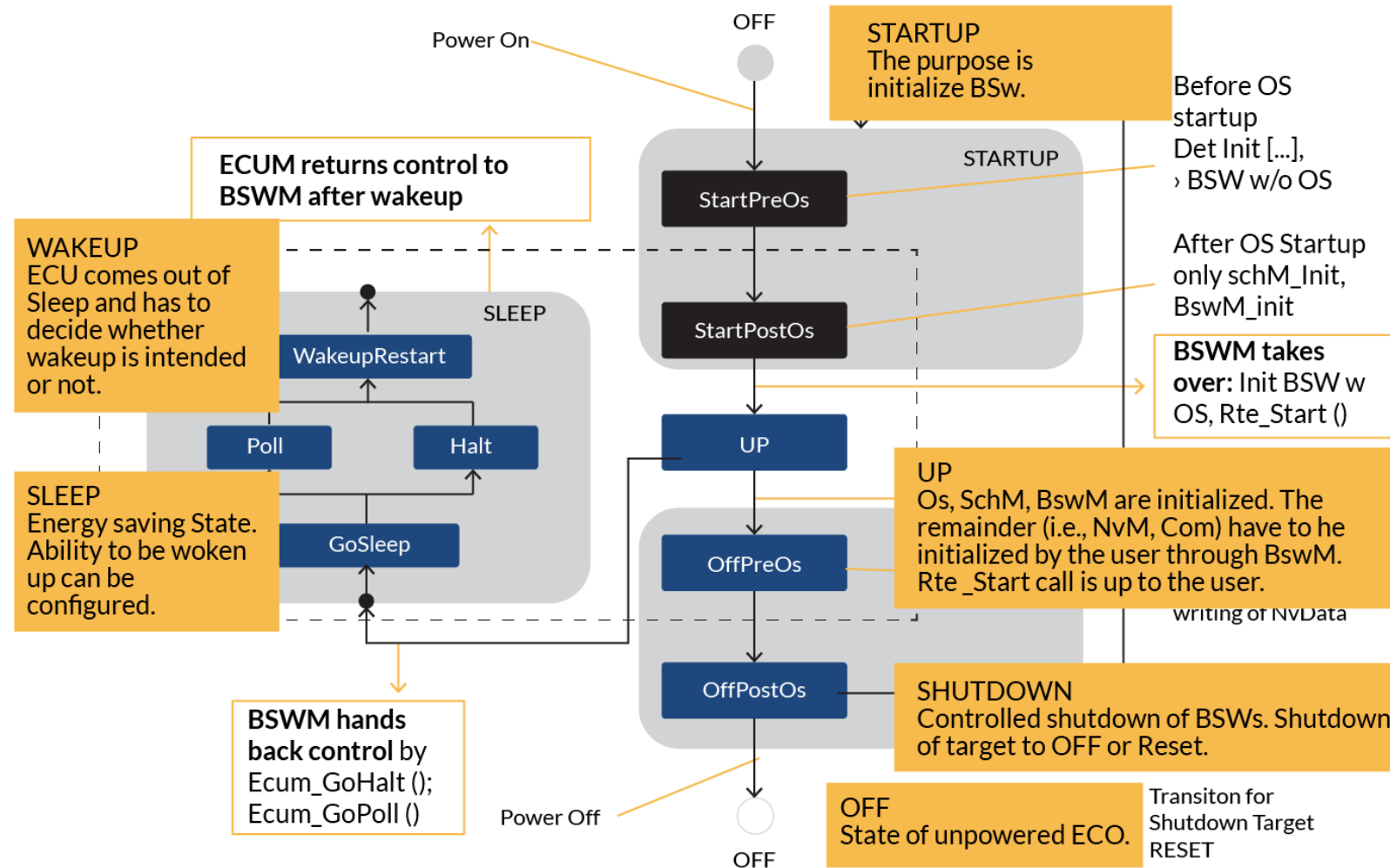
- Name:** GenericModeRequest
- Request Processing:** BSWM_IMMEDIATE
- Arbitrate On Init:** ☐
- Type:** BswMGenericRequest
- Init Value:** GENERIC_MODE_REQ_1
- Source:**
 - Mode Requester Id:** 12345
 - Requested Mode Max:** 1
- Generic Request Modes:**

BswMGenericRequestModes	Generic Request Mode Value	Gener
GENERIC_MODE_REQ_1	0	
GENERIC_MODE_REQ_2	1	

Generic Mode: If there is no built in `BswM_<BSWM>_currentMode/State` Notification like from the BSW and there is no Mode Switch Notification over the RTE, the Generic Mode can be used. Generic Modes are arbitrarily defined (not in terms of a Mode Declaration Group!).

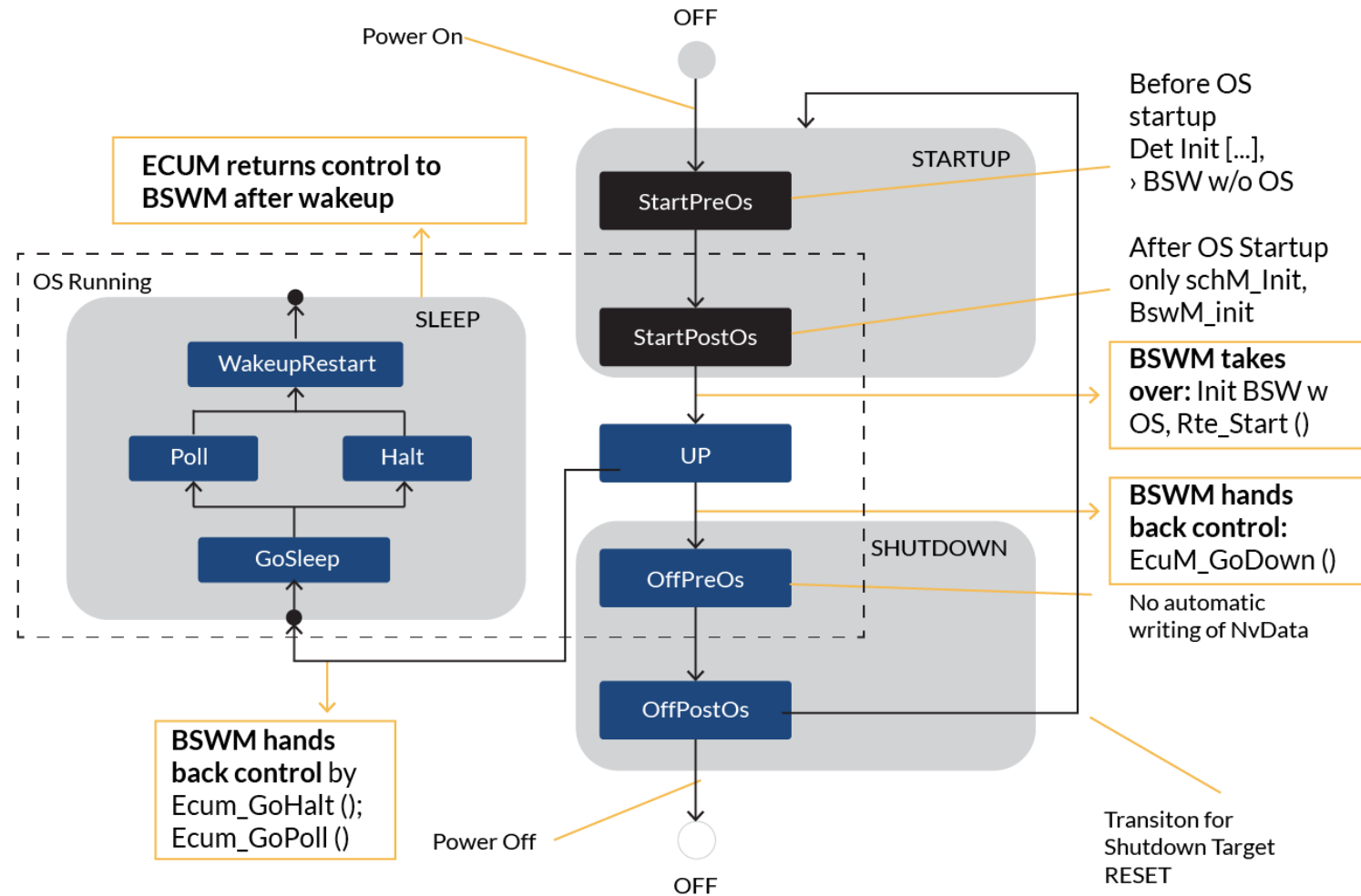
Mode Management

ECU State Manager



Mode Management

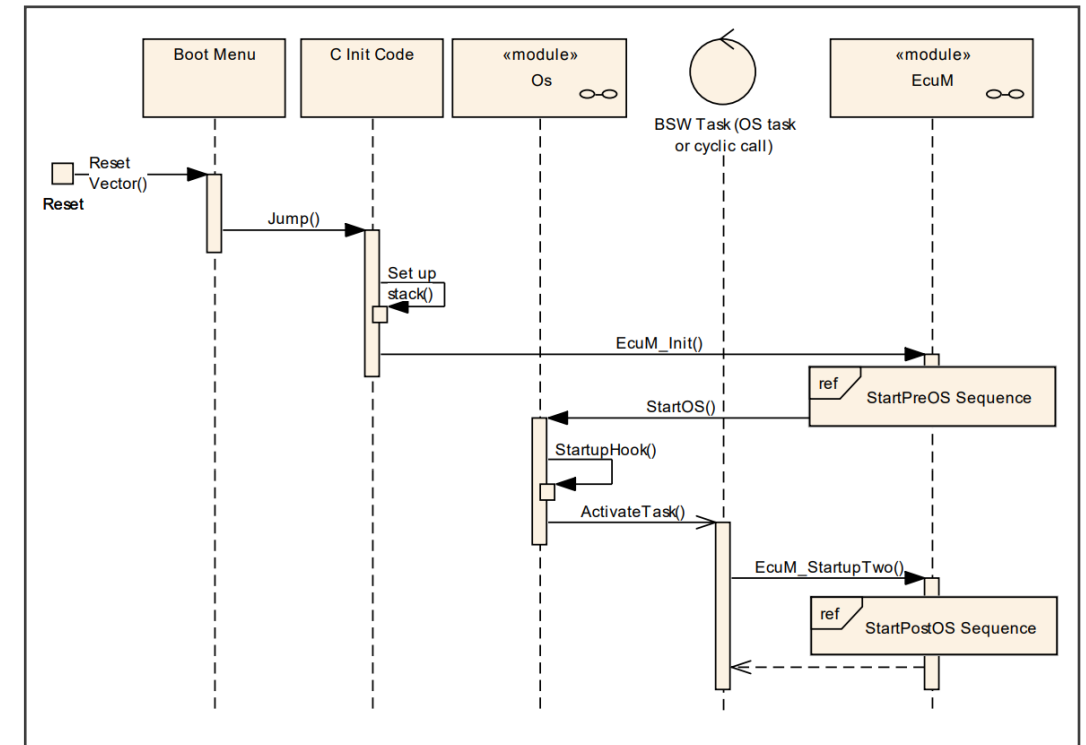
ECU State Manager



Mode Management

STARTUP PHASE

- › **StartPreOs Sequence**
 - › begins after the call of `EcuM_Init()`
 - › ends with the call of `StartOs()`
- › **StartPostOs Sequence**
 - › begins with the call of `EcuM_StartupTwo()`
 - › `BswM_Init()` executes the action lists
 - ends with the call of `SchM_StartTiming()`



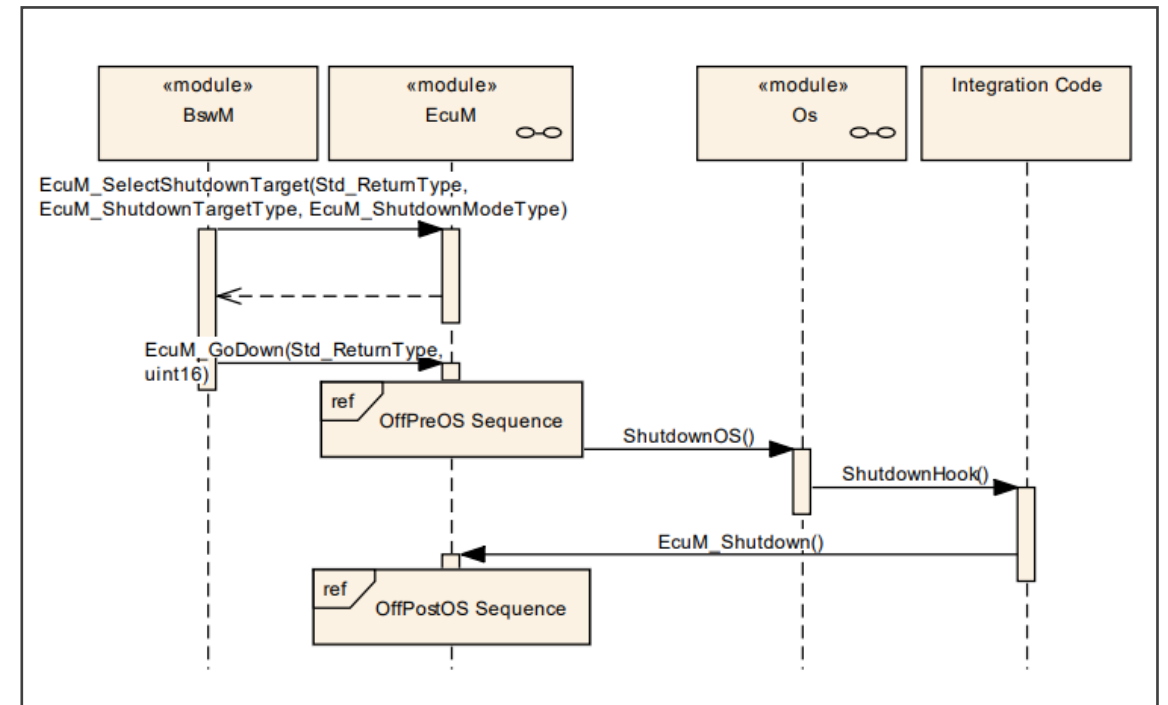
An overview behavior of the startup phase

Source: Specification of ECU State Manager AUTOSAR CP Release 4.3.1

Mode Management

SHUTDOWN PHASE

- › **OffPreOs Sequence**
 - › begins with the call of `EcuM_GoDown()`
 - › ends after the call of `ShutdownOs()`
- › **OffPostOs Sequence**
 - › begins with the call of `EcuM_Shutdown()`
 - › in `ShutdownHook()` the integrator has to insert the call to `EcuM_Shutdown()`
 - › ends in `EcuM_Al_SwitchOff()`

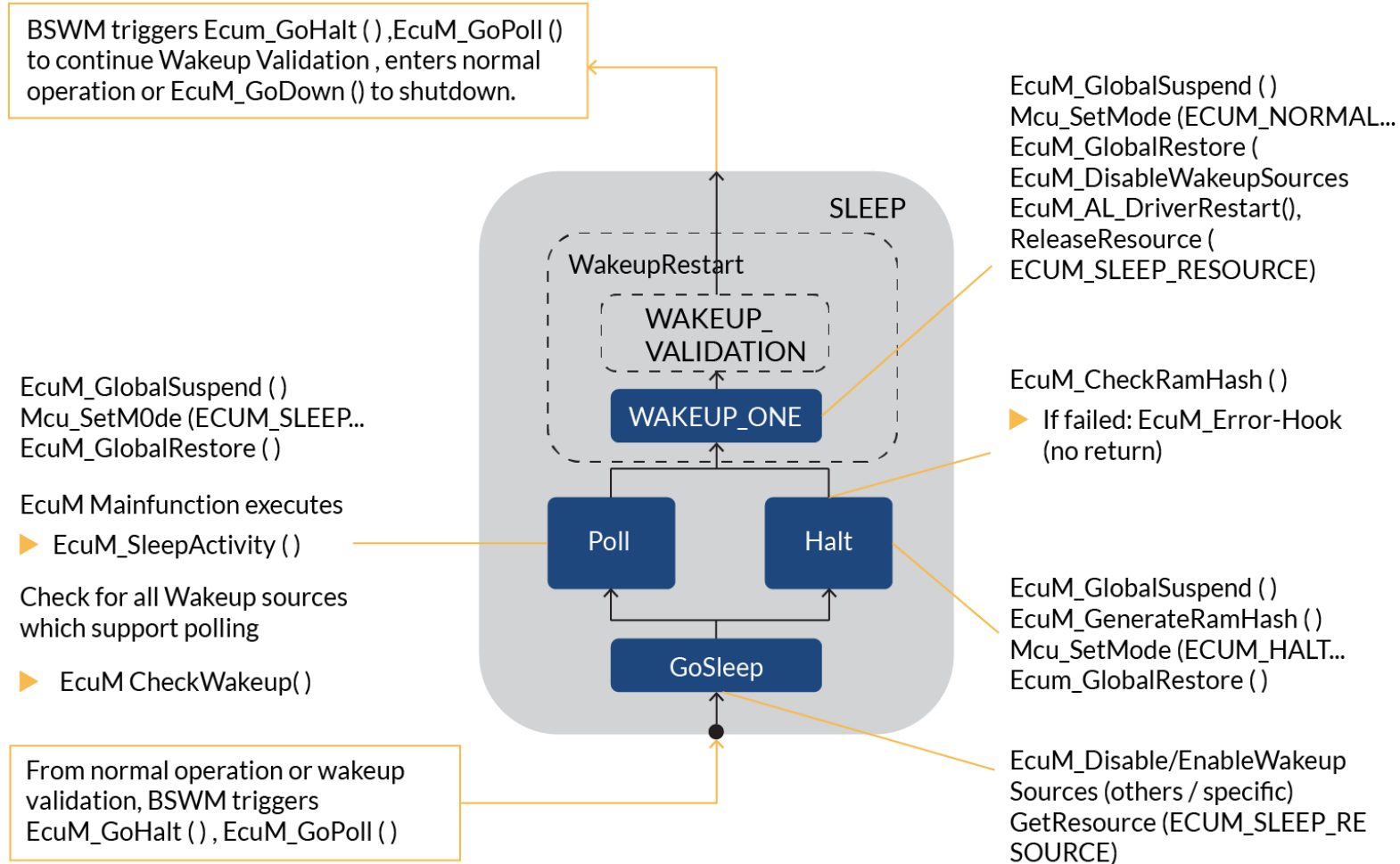


An overview behavior of the shutdown phase

Source: Specification of ECU State Manager AUTOSAR CP Release 4.3.1

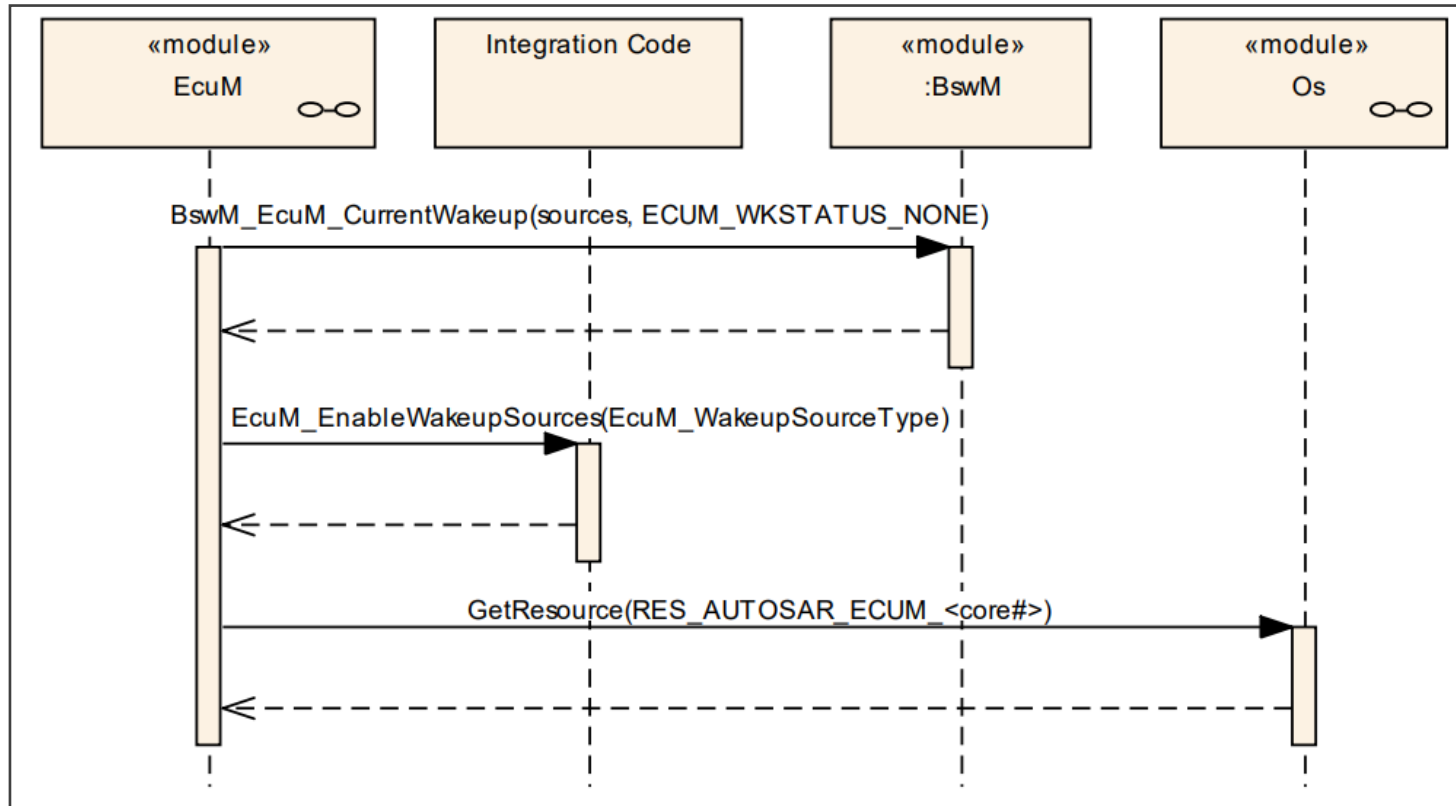
Mode Management

SLEEP PHASE: Poll or Halt



Mode Management

SLEEP PHASE: GoSleep

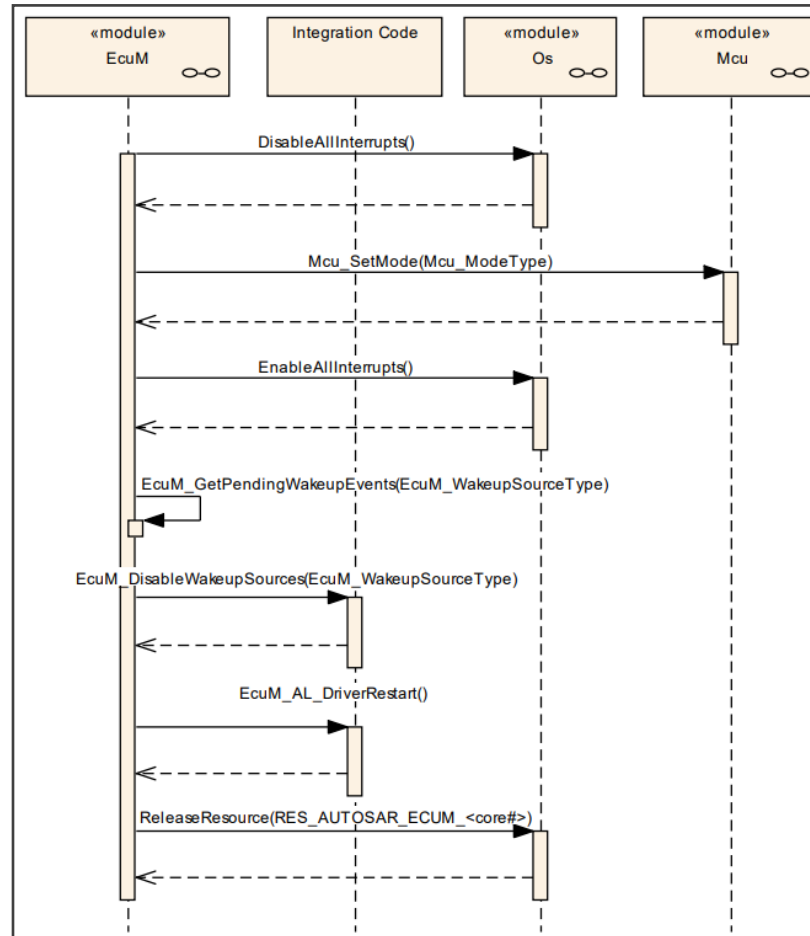


An overview behavior of the go sleep phase

Source: Specification of ECU State
Manager AUTOSAR CP Release 4.3.1

Mode Management

SLEEP PHASE: WakeupRestart

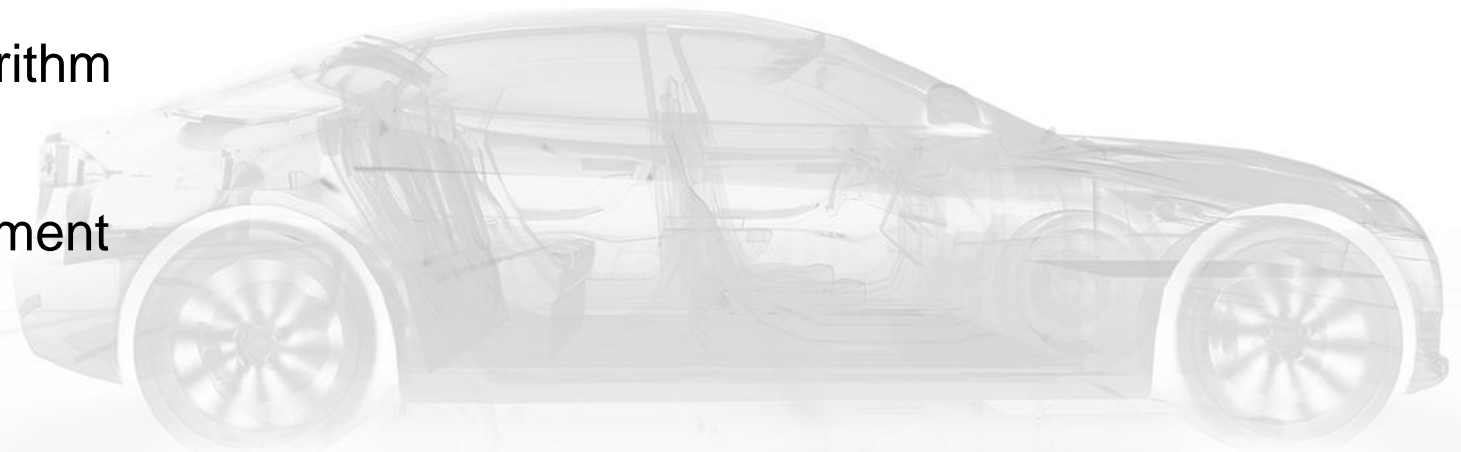


Source: Specification of ECU State Manager AUTOSAR CP Release 4.3.1

An overview behavior of the wakeup-restart phase

Agenda

- › Mode Management
- › **Wakeup Handling**
- › BSWM configuration
- › ECUM configuration
- › COMM, CANSM, NM, and CANNM
- › Network Management Algorithm
- › Service Mapping
- › Exercise 4 - Mode Management



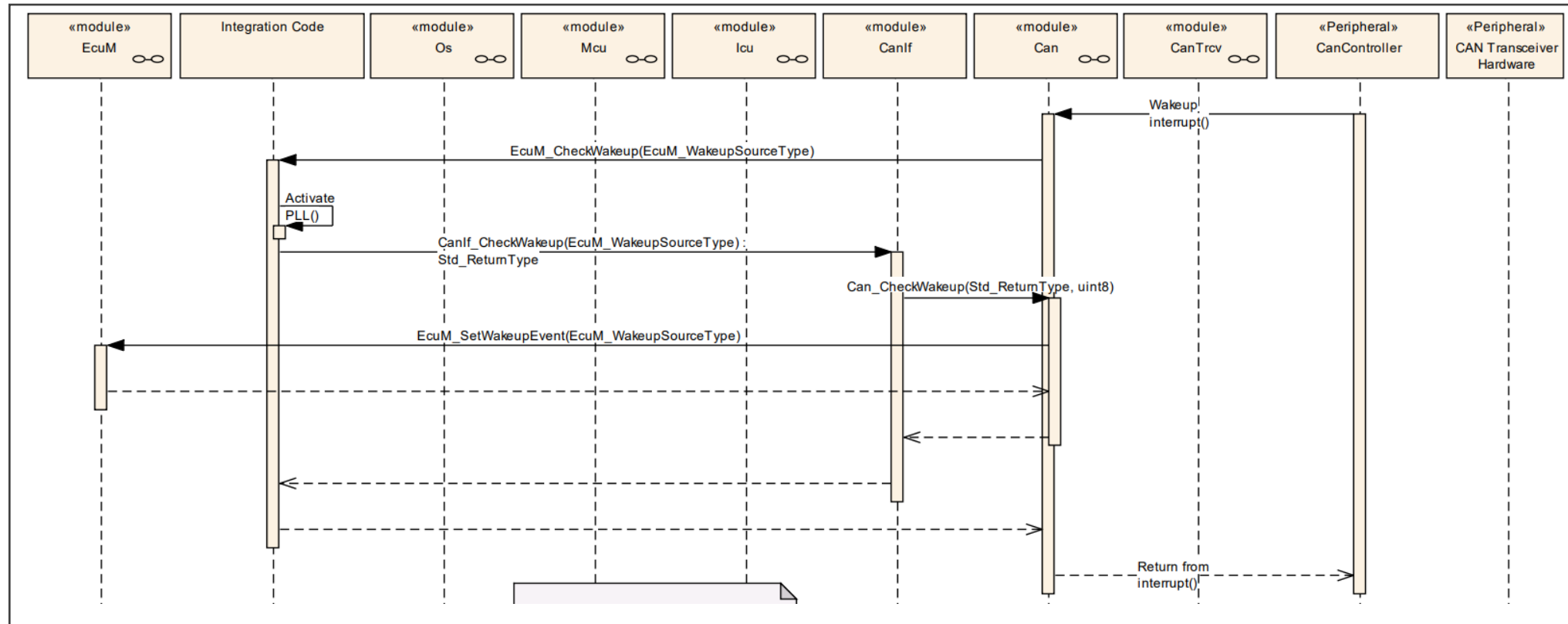
Wakeup Handling

ECUM Wakeup Sources

- › The AUTOSAR Wakeup procedure consists of two steps
 1. Check Wakeup Source & report Wakeup Event
 2. Wakeup Validation Protocol
- › Wakeup procedure independent from ECUM SLEEP PHASE
 - › But used to wakeup from the SLEEP PHASE
- › Each ECU owns a set of Wakeup Sources (maximum 32 sources)
 - › Wakeup is coordinated by the ECUM
 - › There are predefined Wakeup Sources
 - › Up to 27 user defined Wakeup Sources
- › Each Wakeup Source either
 - › Needs validation or
 - › Is validated per default

Wakeup Handling

Cooperation of CAN, CANIF and ECUM



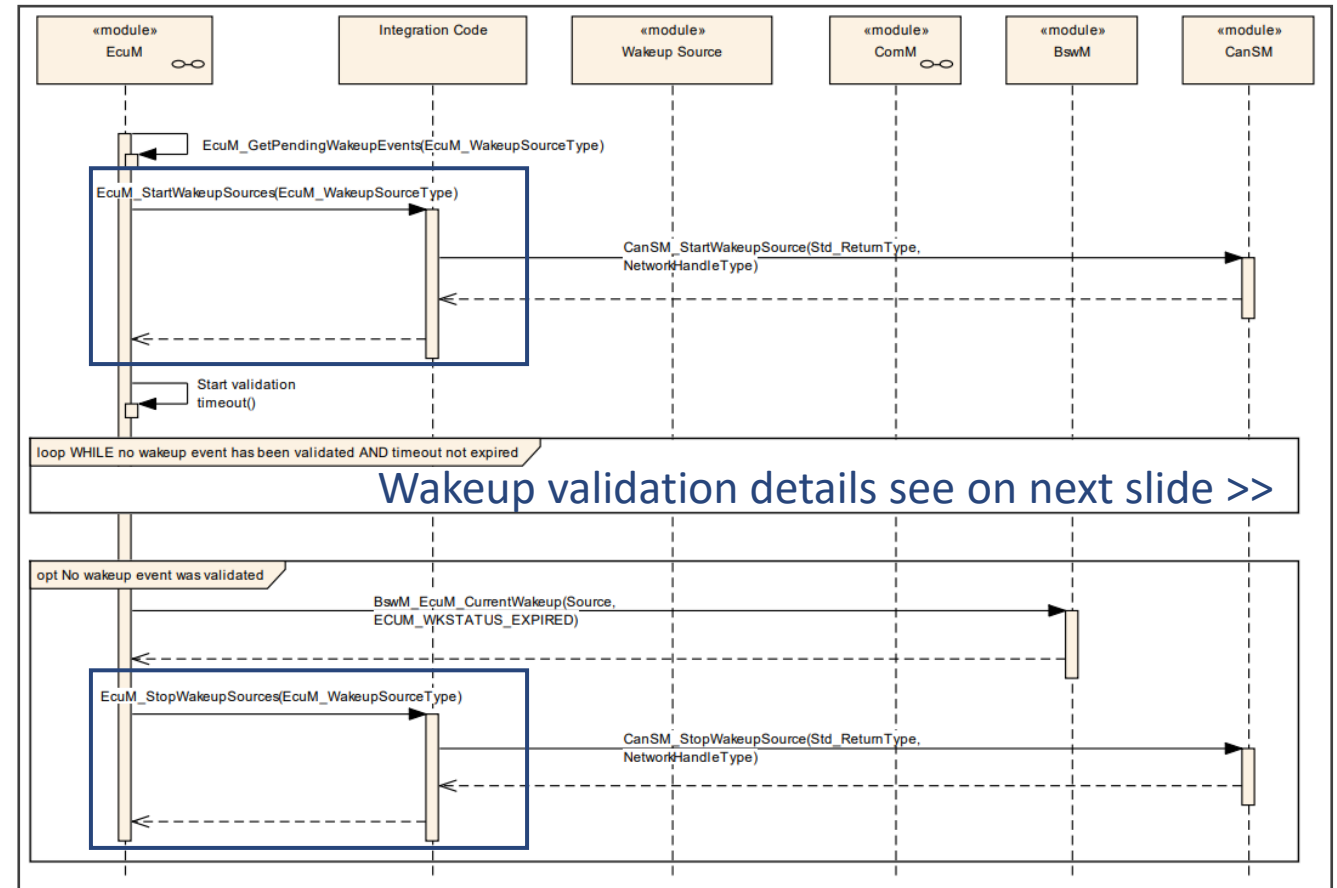
One possible wakeup scenario: CAN Controller Wakeup by Interrupt

Source: Specification of ECU State Manager AUTOSAR CP Release 4.3.1

Wakeup Handling

Wakeup Validation

- › The wakeup sources have to be **started** before the wakeup validation can proceed
`EcuM_StartWakeupSources()`
- › After the failed validation the wakeup sources have to be **stopped**
`EcuM_StopWakeupSources()`

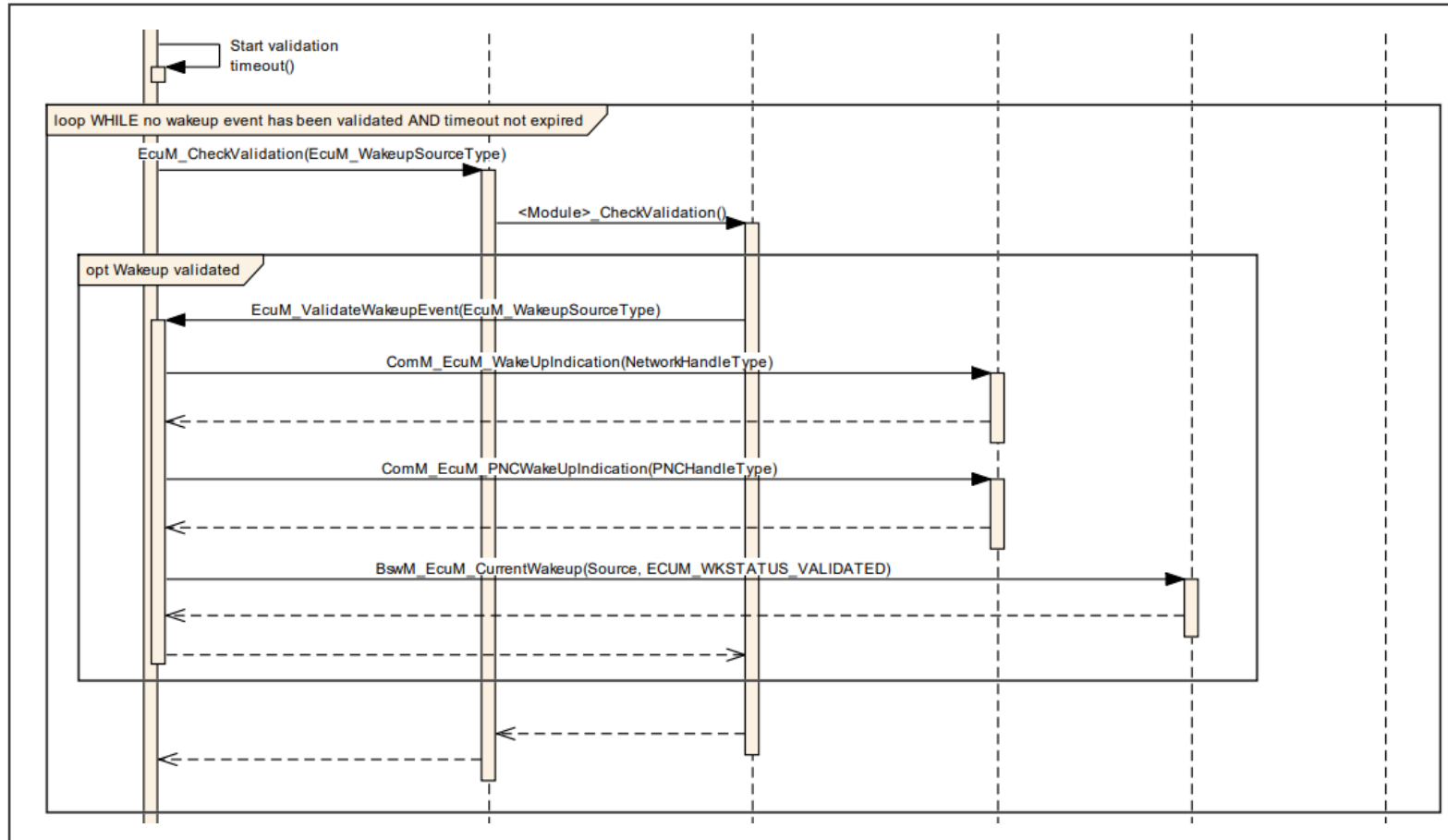


An overview behavior of the wakeup-validation phase

Source: Specification of ECU State Manager AUTOSAR CP Release 4.3.1

Wakeup Handling

Wakeup Validation

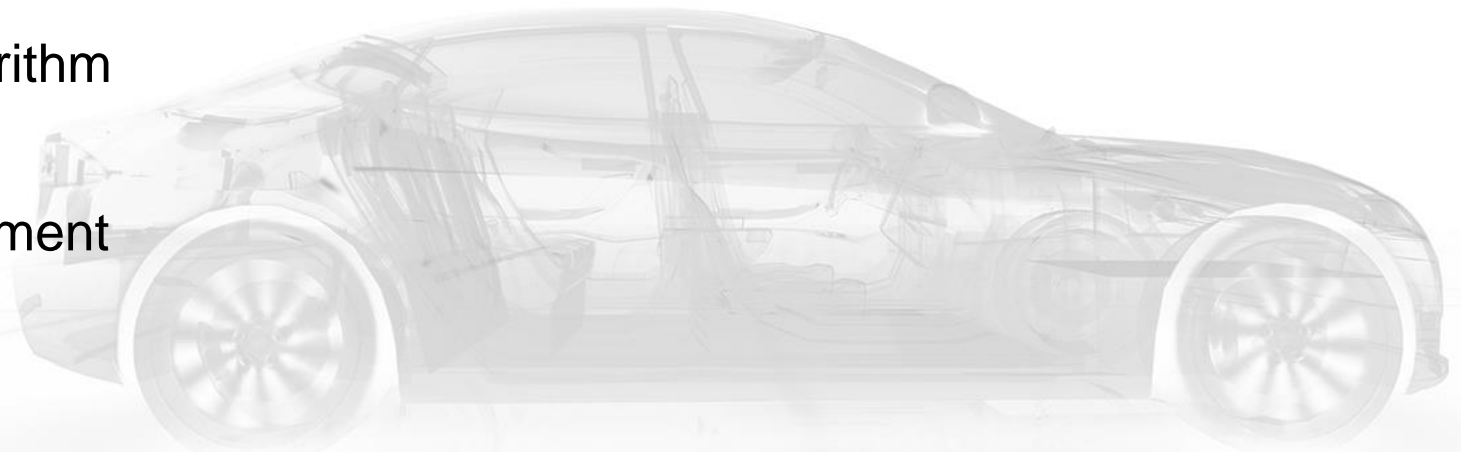


An example of CAN wakeup validation phase

Source: Specification of ECU State Manager AUTOSAR CP Release 4.3.1

Agenda

- › Mode Management
- › Wakeup Handling
- › **BSWM configuration**
- › ECUM configuration
- › COMM, CANSM, NM, and CANNM
- › Network Management Algorithm
- › Service Mapping
- › Exercise 4 - Mode Management

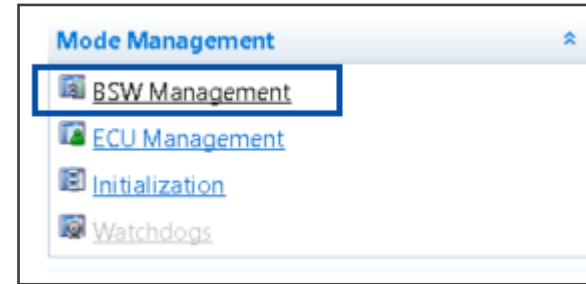


BSWM configuration

Auto Configuration

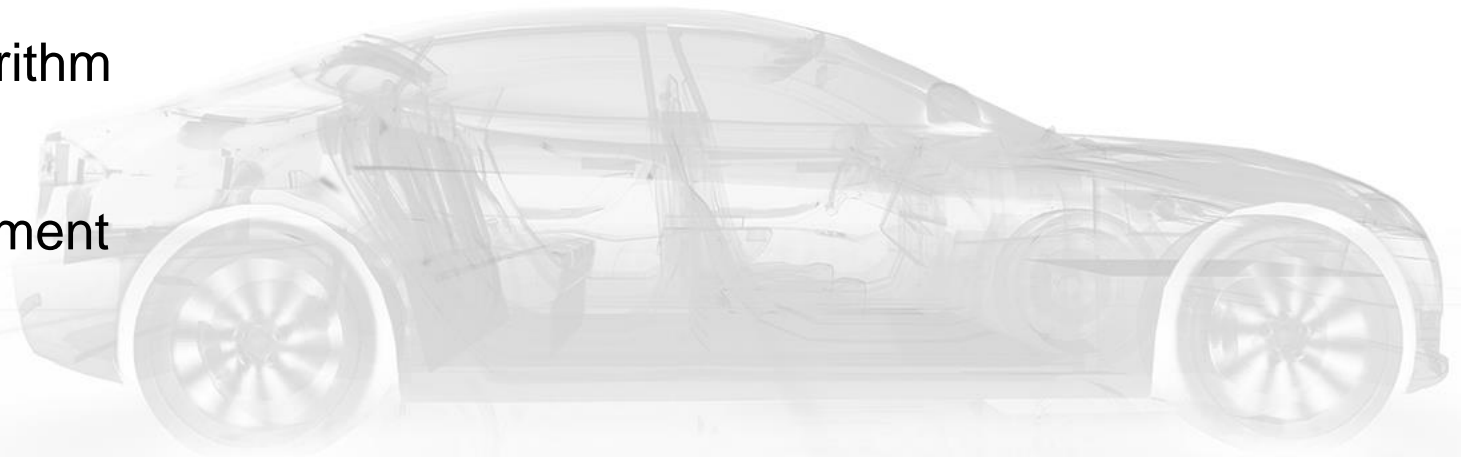
Auto configuration exists for

- 1) ECU state handling
- 2) Module initialization
- 3) Communication handling



Agenda

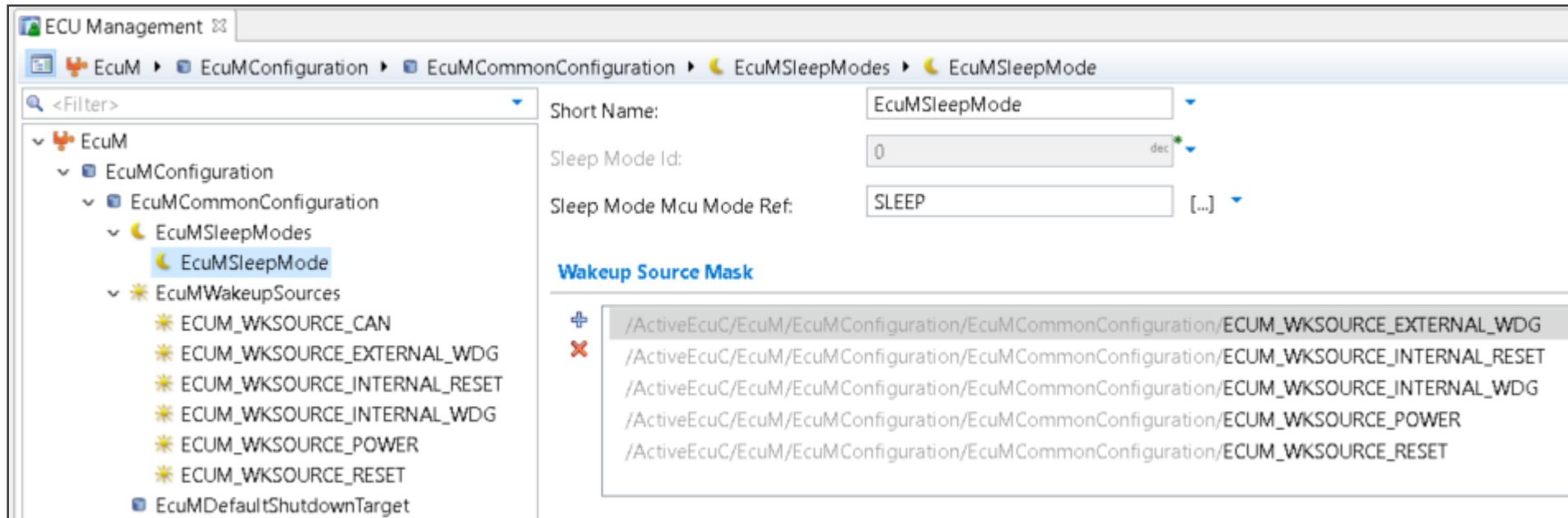
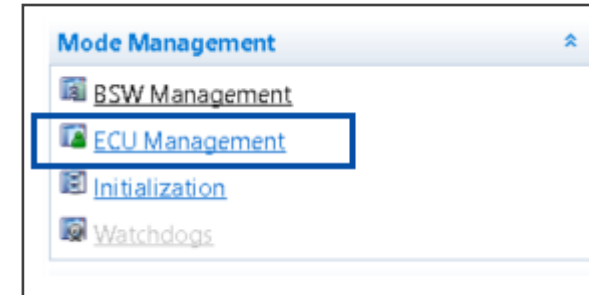
- › Mode Management
- › Wakeup Handling
- › BSWM configuration
- › **ECUM configuration**
- › COMM, CANSM, NM, and CANNM
- › Network Management Algorithm
- › Service Mapping
- › Exercise 4 - Mode Management



ECUM configuration

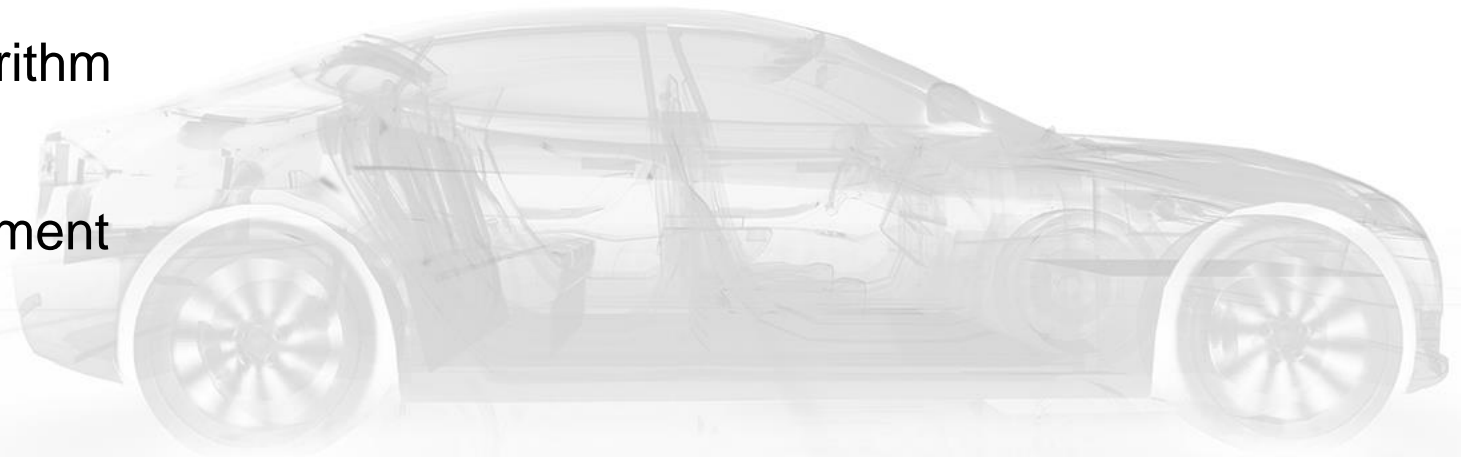
ECUM Configuration Settings

- › ECUM General Settings
- › ECUM fundamental Initialization steps
- › ECUM Sleep Modes and Wakeup Sources



Agenda

- › Mode Management
- › Wakeup Handling
- › BSWM configuration
- › ECUM configuration
- › **COMM, CANSIM, NM, and CANNM**
- › Network Management Algorithm
- › Service Mapping
- › Exercise 4 - Mode Management



COMM, CANSM, NM, and CANNM

Communication Manager

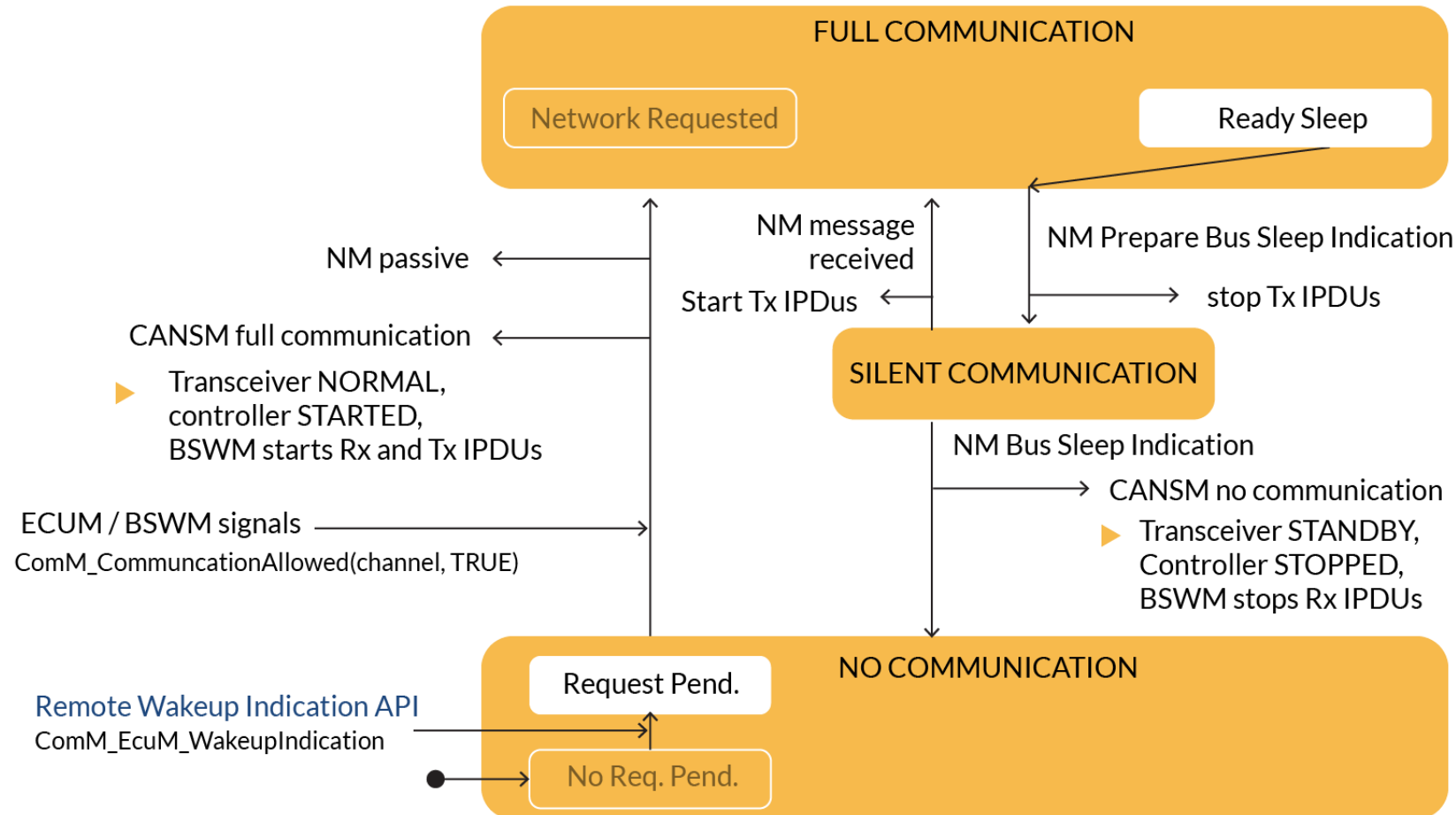
COMM

- › Controls the BSW regarding communication
- › Coordinates bus communication requests from SWCs
- › COMM cannot keep the ECU actively awake
 - › ECUM (or BSWM) indicate to COMM when communication is allowed
 - › ComM_CommunicationAllowed API
 - › For the transition of the ECU into shutdown, ECUM (or BSWM respectively) must call
 - › EcuM_GoDown API
- › Uses NM of the networks to keep the networks awake
- › Uses SM of the networks to control bus activity

COMM, CANSM, NM, and CANNM

Communication Manager – External Wakeup

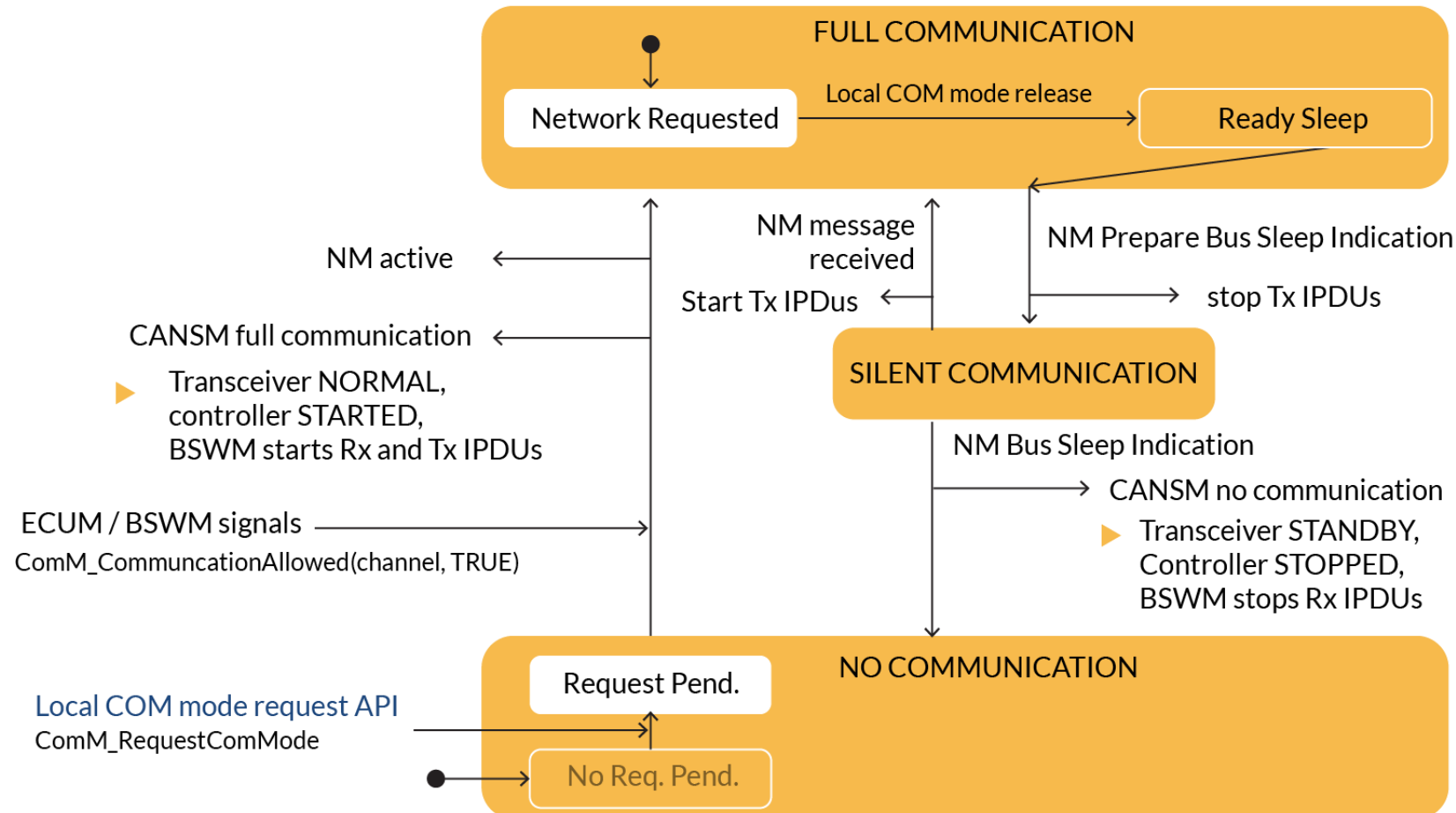
Always ready to sleep, cannot keep the network awake → start leads to **READY SLEEP** state.



COMM, CANSM, NM, and CANNM

Communication Manager – Internal Wakeup

can keep the network awake → start leads to **NETWORK REQUESTED** state.



COMM, CANSM, NM, and CANNM State Manager

CANSM

- › Translation of network communication mode requests
- › Control of peripherals (indirectly over CANIF)
 - › CAN Transceivers
 - › CAN Controllers
- › Handle the network mode via a separate state machine per network
- › CAN Bus error management: Bus-off recovery via a separate state machine per network

COMM, CANSM, NM, and CANNM

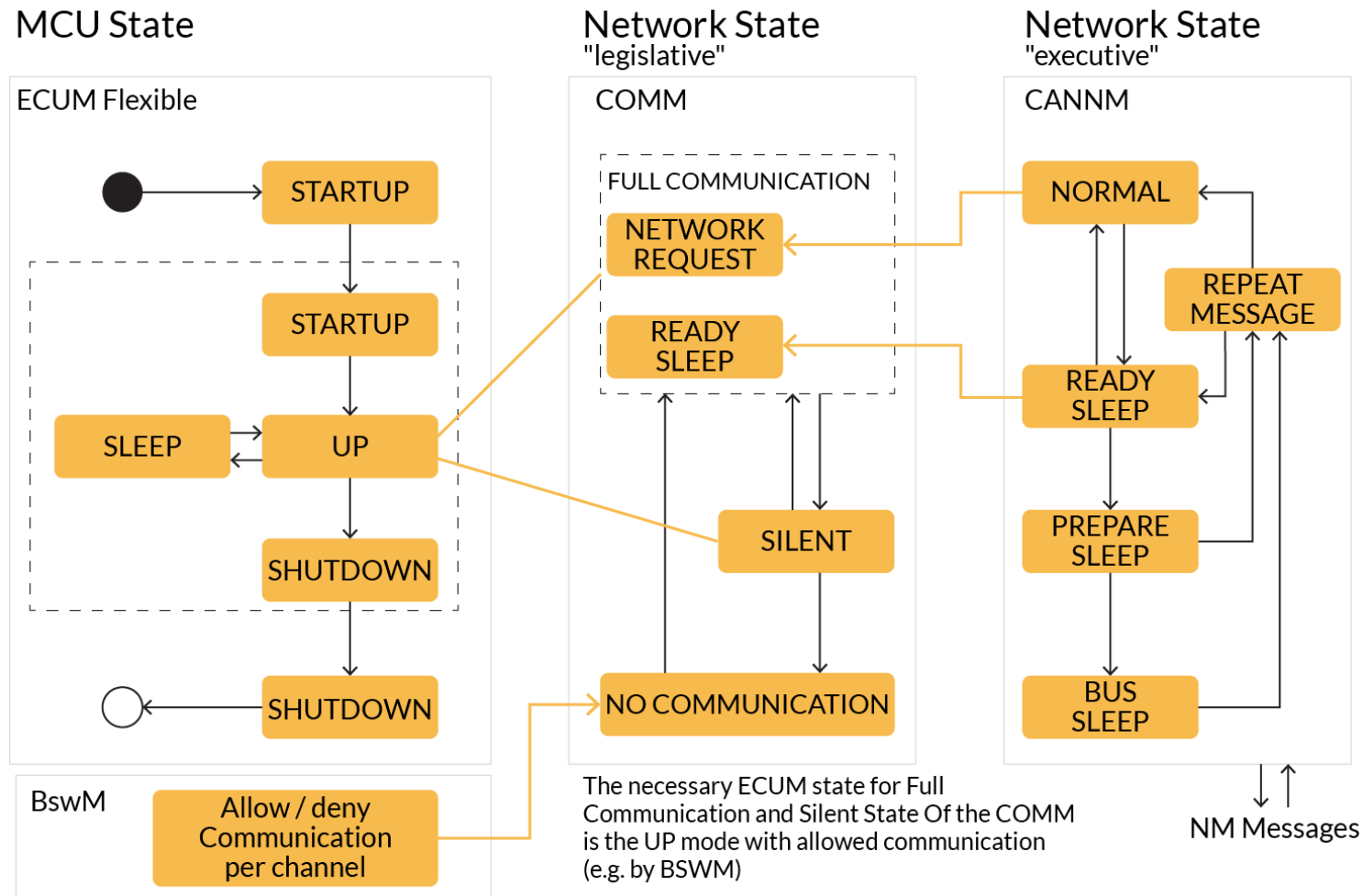
Network Management

NM

- › Controlled transition of all ECUs into bus-sleep mode and vice versa
- › Coordination functionality between different networks
- › Node detection handling
- › Abstraction of technology-specific NMs (CANNM, FRNM, LINNM, UDPNM) by a generic state machine

COMM, CANSM, NM, and CANNM

Interaction of ECUM, COM, CANNM and BSWM



COMM, CANSIM, NM, and CANNM

Network Management for CAN

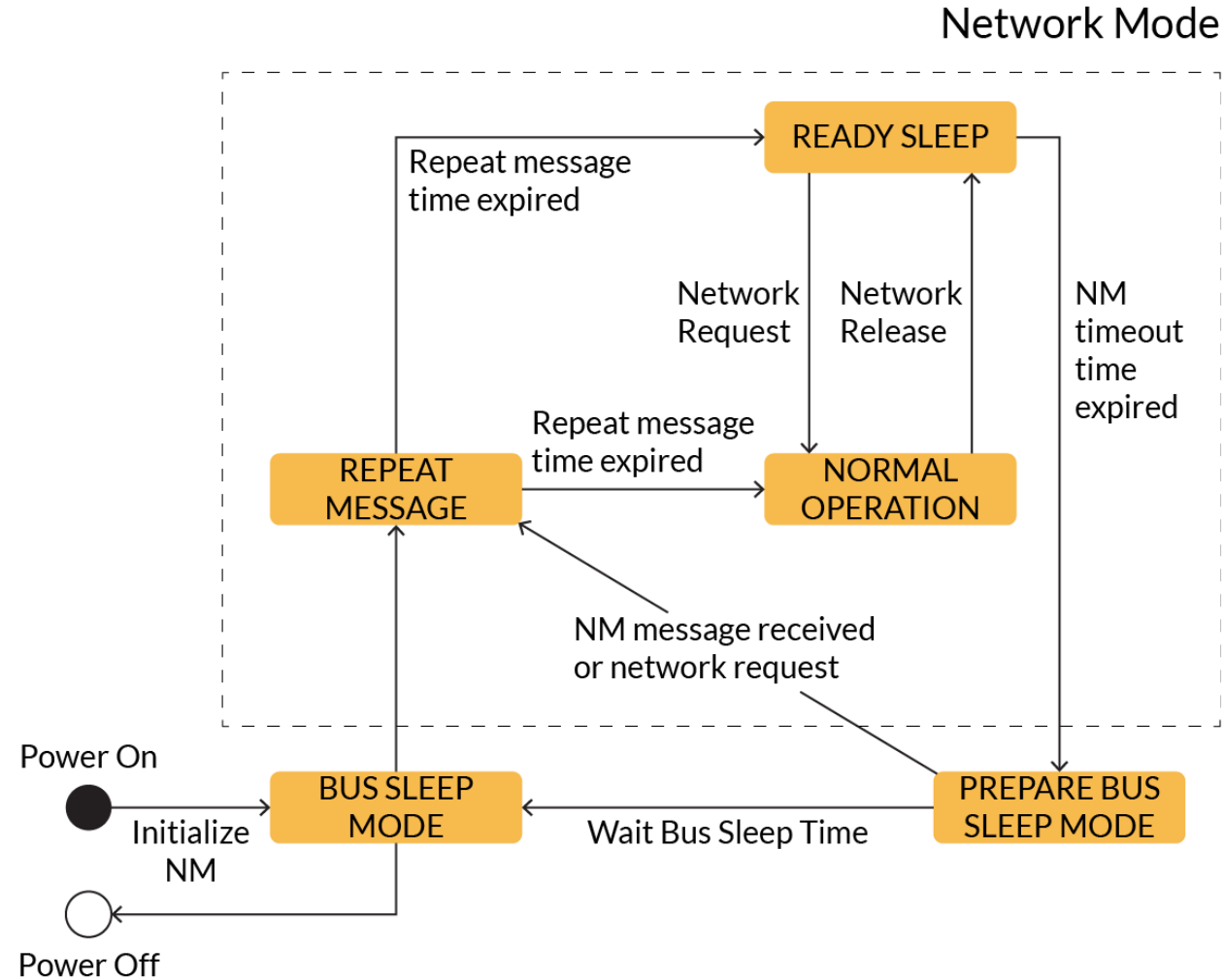
CANNM

The AUTOSAR NM is a **direct** and **distributed** Network Management

- › **Direct NM** - Each node has its own NM message
 - › The NM message is reserved for the Network Management
 - › The NM message signals that bus communication is needed
 - › The NM message data is **not relevant** for the NM mechanism
- › **Distributed NM** - All nodes are equal (there is no NM Master or NM Slave)
- › Optional bus load reduction
 - › Only two active nodes transmit NM messages
 - › Avoids the bus load being dependent on the number of participating nodes
- › Partial Networking
 - › Impose logical partial network clusters (PNCs)

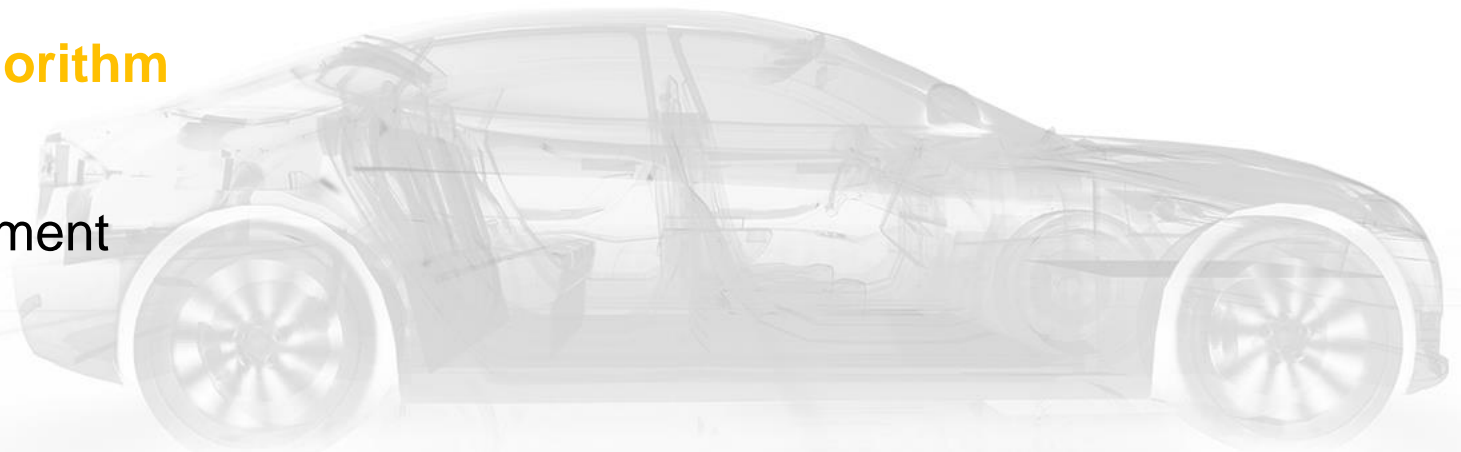
COMM, CANSIM, NM, and CANNM

Network Management for CAN



Agenda

- › Mode Management
- › Wakeup Handling
- › BSWM configuration
- › ECUM configuration
- › COMM, CANSM, NM, and CANNM
- › **Network Management Algorithm**
- › Service Mapping
- › Exercise 4 - Mode Management



Network Management Algorithm

Basic Mechanism – Single ECU (CAN)

PowerOn WakeUp NM messages of ECU



Repeat Message

Each ECU transmits its own NM message cyclically for NM_REPEAT_MSG_TIME.

Ready Sleep

ECU is ready to sleep, no NM message transmission, restart of timeout timer upon NM message Rx.

Normal

NM message transmission and restart of timeout timer upon NM message Rx and Tx.

Prepare Sleep

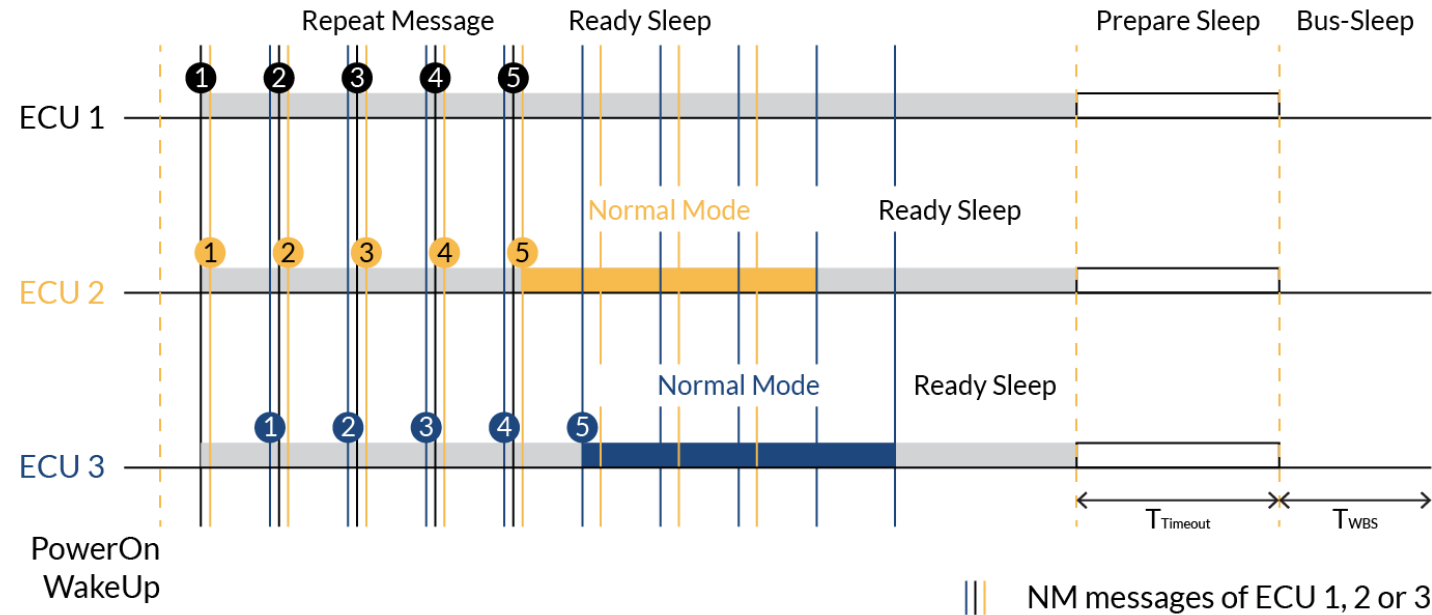
If NM_TIMEOUT_TIME expired and no NM message has been transmitted or received in the meantime.

Bus-Sleep

After NM_WAIT_BUS_SLEEP_TIME transition to bus-sleep mode.

Network Management Algorithm

Basic Mechanism – Network (CAN)



Repeat Message

Each ECU transmits its own NM message cyclically for $NM_REPEAT_MSG_TIME$.

Ready Sleep

ECU is ready to sleep, no NM message transmission, restart of timeout timer upon NM message Rx.

Normal

NM message transmission and restart of timeout timer upon NM message Rx and Tx.

Prepare Sleep

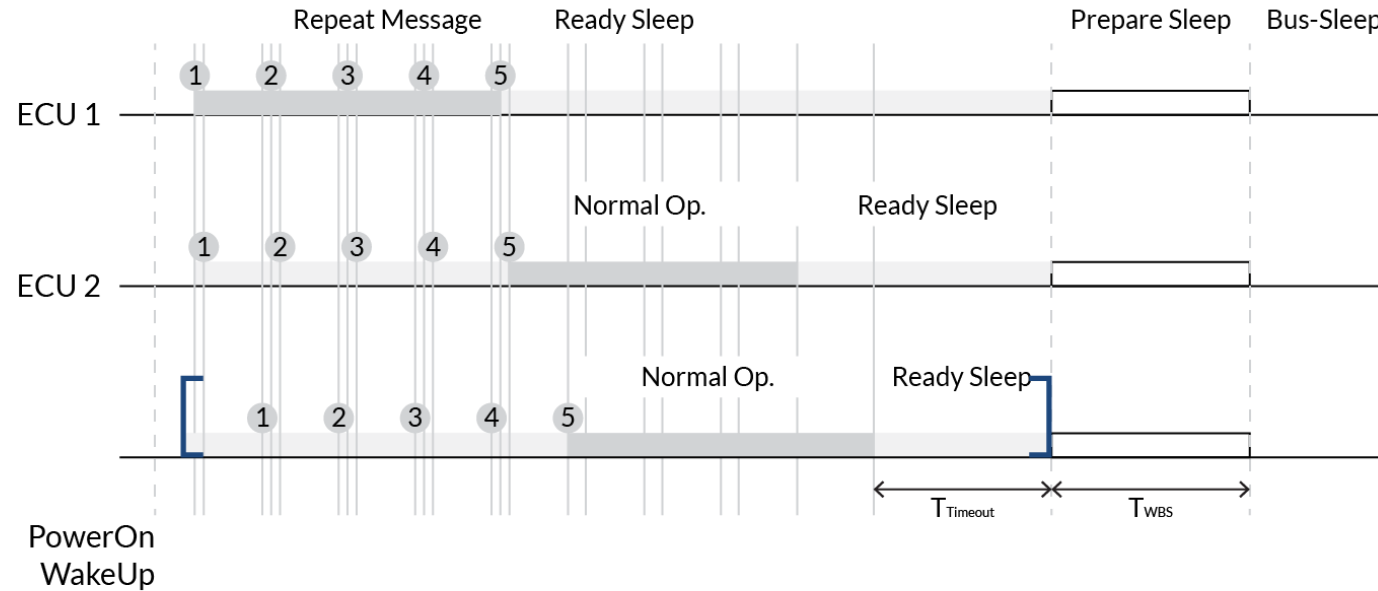
If $NM_TIMEOUT_TIME$ expired and no NM message has been transmitted or received in the meantime.

Bus-Sleep

After $NM_WAIT_BUS_SLEEP_TIME$ transition to bus-sleep mode.

Network Management Algorithm

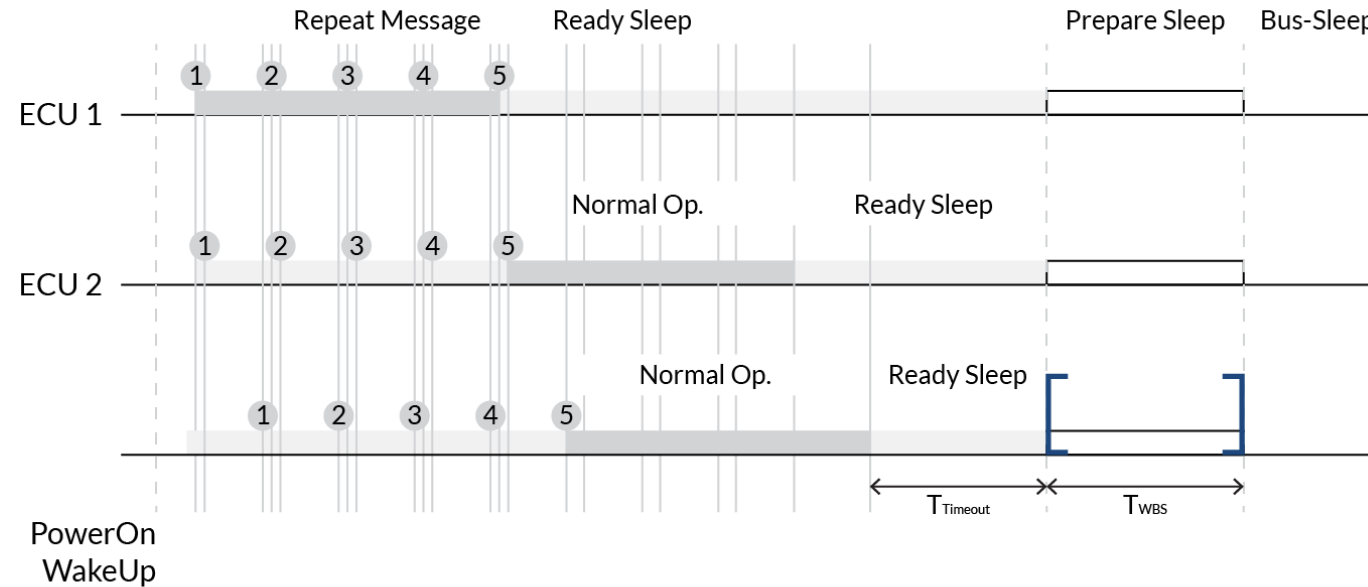
Network Mode (CAN)



- Reception and transmission of application messages enabled
- **Repeat Message State**
 - cyclic transmission of NM messages for a certain time
 - all NM active nodes participate in NM message communication
- **Normal Operation State**
 - transmission of NM messages (if activated reduced transmission)
- **Ready Sleep State**
 - transmission of NM messages is stopped

Network Management Algorithm

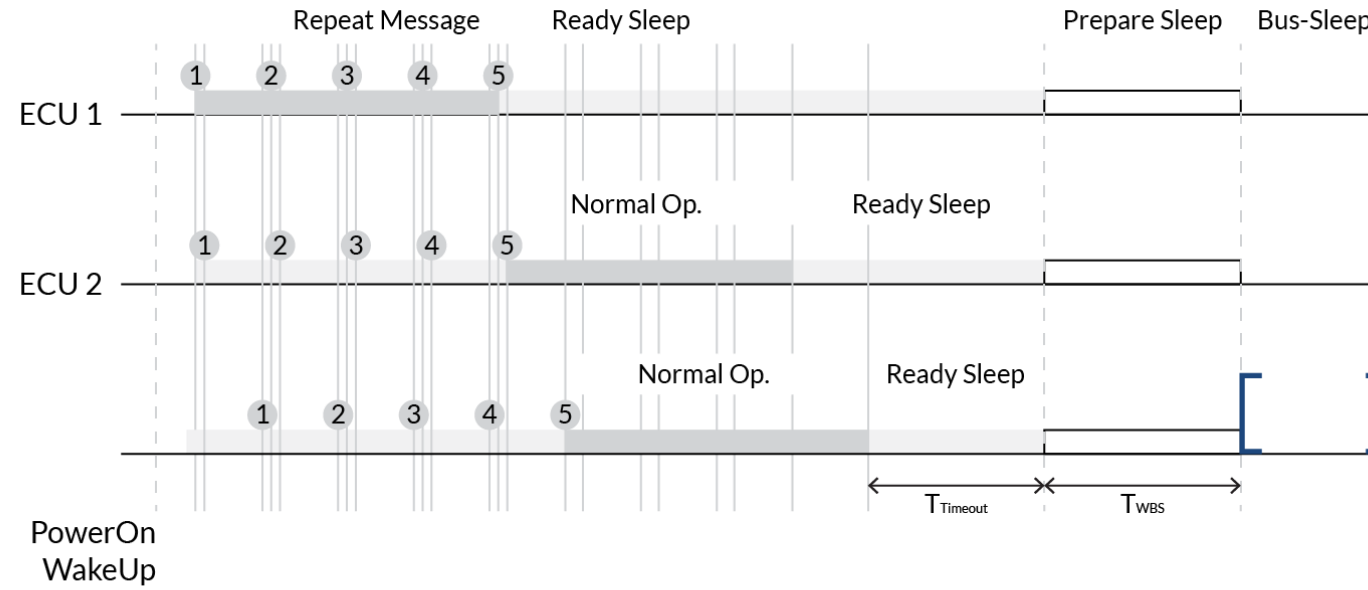
Prepare Bus-Sleep Mode (CAN)



- No active participation in the network
- Transmission of application message is stopped
- Bus calm down period
- Restart of network upon
 - reception of NM message (external)
 - bus communication request (internal)

Network Management Algorithm

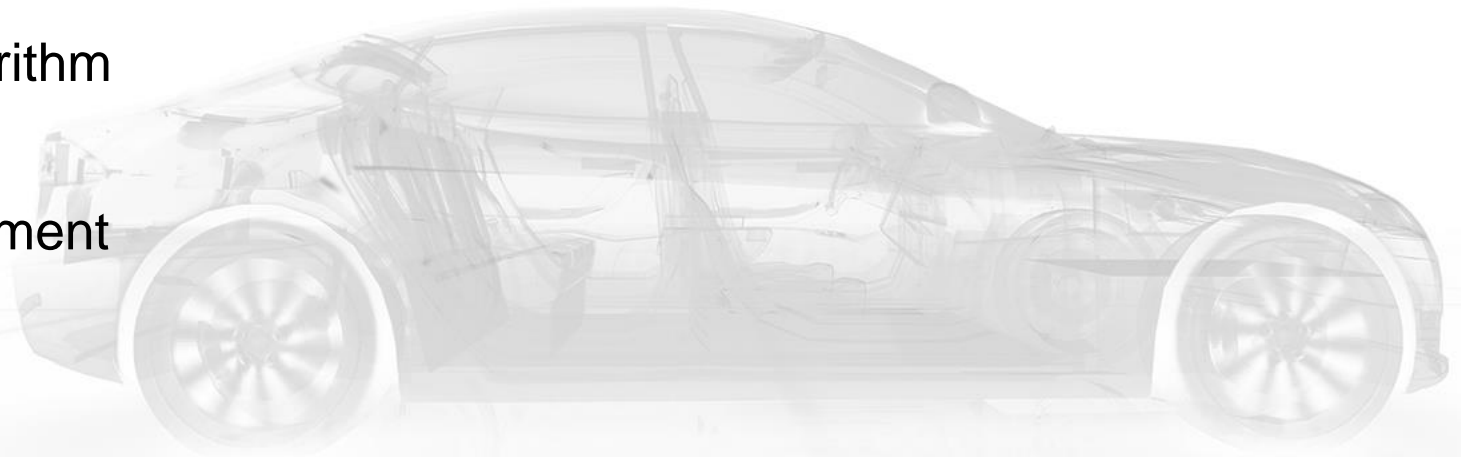
Bus-Sleep Mode (CAN)



- Communication bus is shut down
- Restart (wakeup) of network management
 - wake-up on communication bus (external)
 - reception of a NM message (external)
 - application requires bus communication (internal)

Agenda

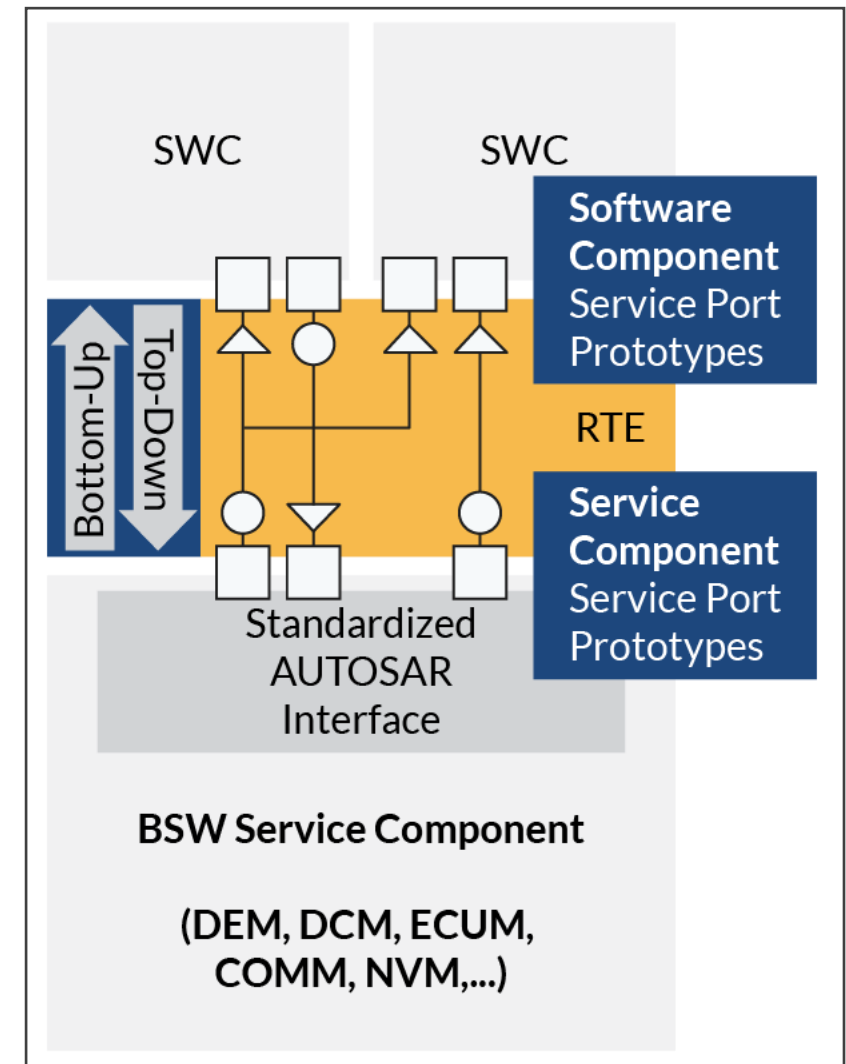
- › Mode Management
- › Wakeup Handling
- › BSWM configuration
- › ECUM configuration
- › COMM, CANSM, NM, and CANNM
- › Network Management Algorithm
- › **Service Mapping**
- › Exercise 4 - Mode Management



Service Mapping

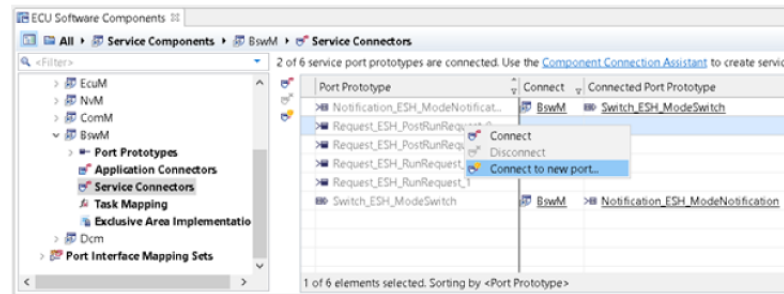
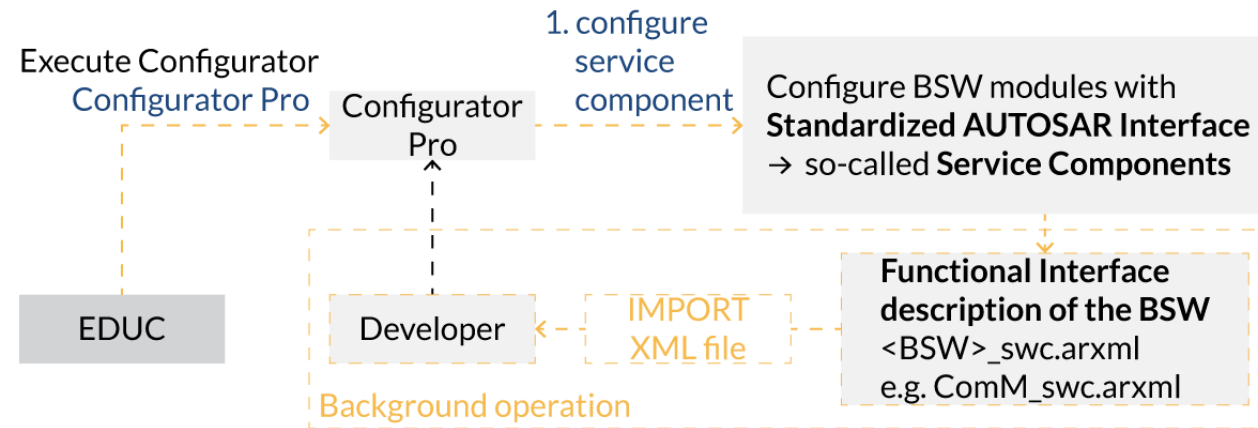
Service Mapping

- › Assigning Service Ports of Software Components to Service Ports of Service Components or vice versa
- › Bottom-up service configuration
 1. Configure a service module in the basic software
 2. Assign the respective service port prototypes to the SWCs where appropriate
- › Top-down service configuration
 1. Use Service Needs
 - › Document at the SWC level what is required from the BSW as a service
 2. Derive a service module configuration based on these requirements
 3. Configure the basic software service module in detail



Service Mapping

Service Component Configuration



2. Create Service Component Prototype

3. Equip SWCs with Service Port Prototypes and create the Service mapping and create Server Runnables in one step

Bottom-Up

In Configurator you can configure the Service Component, create Service Port Prototypes at the SWC and create the Service mapping

Agenda

- › Mode Management
- › Wakeup Handling
- › BSWM configuration
- › ECUM configuration
- › COMM, CANSM, NM, and CANNM
- › Network Management Algorithm
- › Service Mapping
- › **Exercise 4 - Mode Management**

