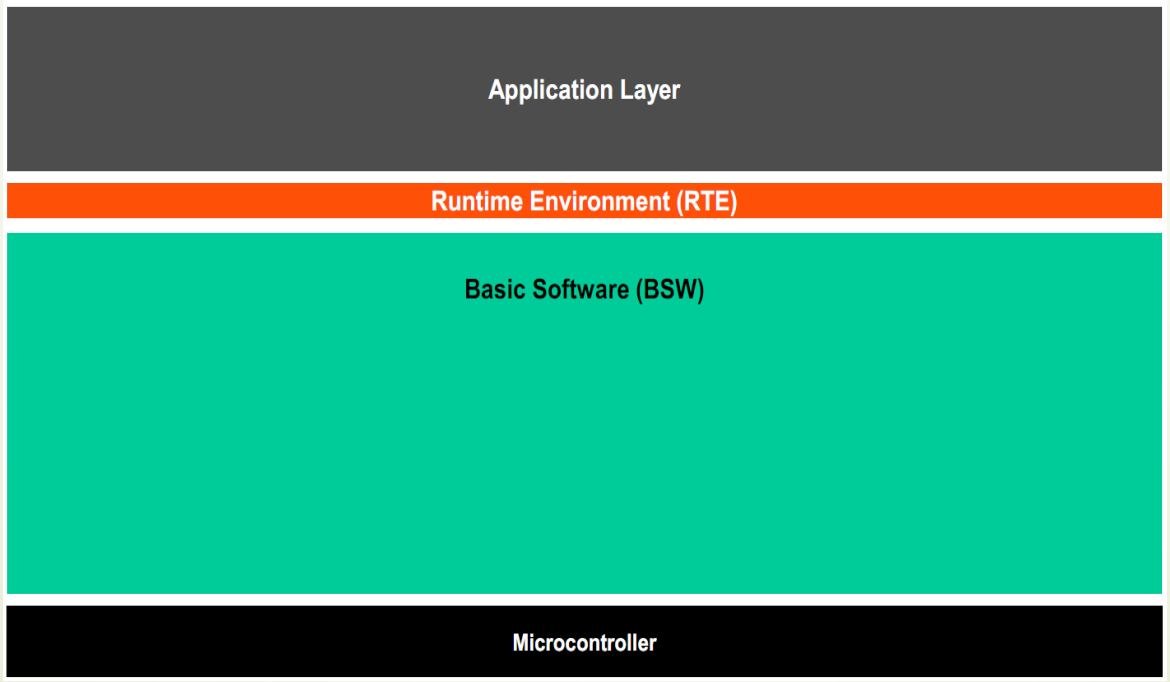
AUTOSAR Software Component

AUTOSAR Software Component

Software component: software component(SWC) Is a piece of code that carries out an application or part of an application.

In AUTOSAR, software components are not limited to the application layer, i.e. they also exist in the RTE and BSW layer.

For the time-being our focus is on the application layer software components.

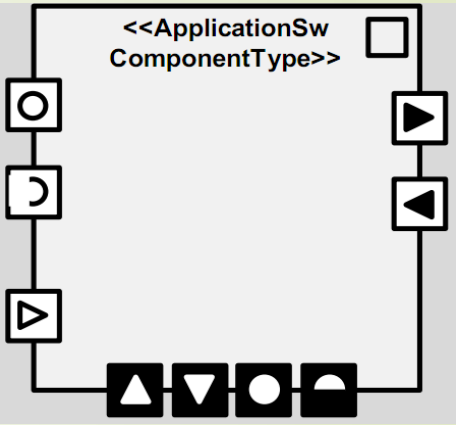


SWC Types

- Application software component
- Sensoractuator software component
- Parameter software component
- Composition software component
- Service proxy software component
- Service software component
- Ecuabstraction software component
- Complex driver software component
- Nvblock software component

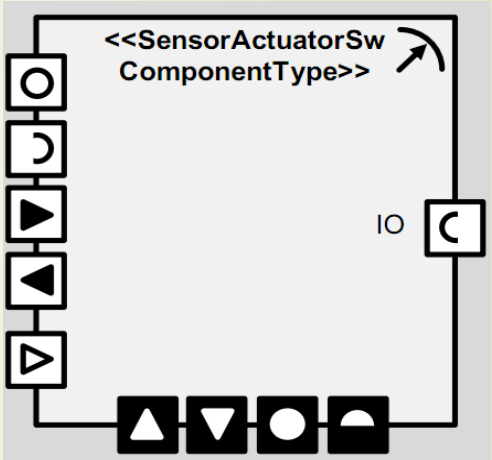
Application SWC

- is an atomic software component that carries out an application or part of it. It can use ALL AUTOSAR communication mechanisms and services
- Application SWCs are our main building blocks



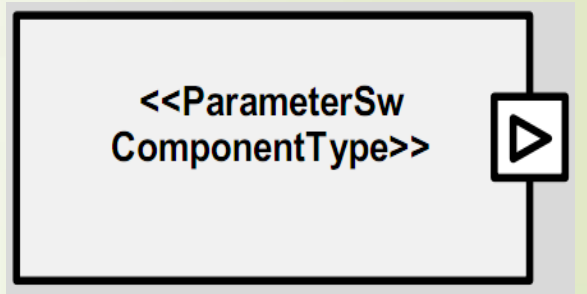
Sensoractuator software component

- is an atomic SWC that handles the specifics of sensors or actuators. It directly interacts with the ecu-abstraction



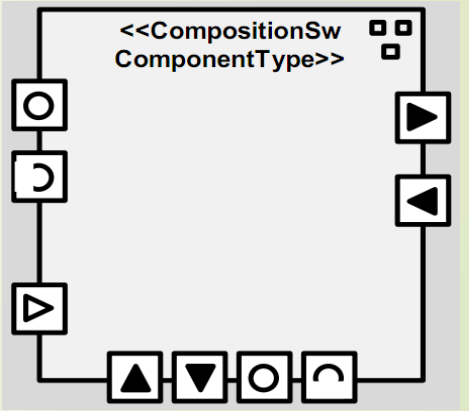
Parameter Software Component

- Atomic SWC
- it provides parameter values. They can be fixed data, const or variable. It allows access to fixed data or calibration data
- They don't have an internal behavior
- They only have PPorts of ParameterInterfce type
- Need to be on the same ecu as the SWCs accessing them since a parameter SWC represents the memory containing the calibration parameter



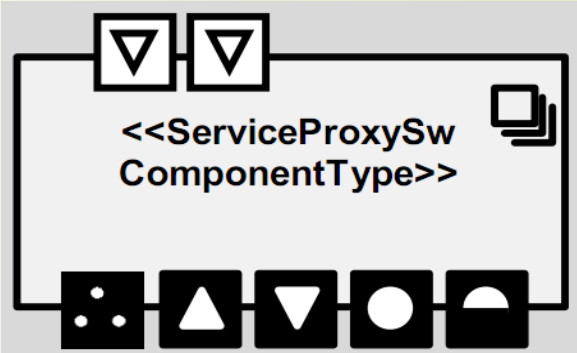
Composition software components

- a composition of atomic or non-atomic (composite) software components which is an encapsulation
- Non-atomic
- Has no binary footprint
- Primarily used to abstract a bunch of SWCs from other SWCs on VFB level



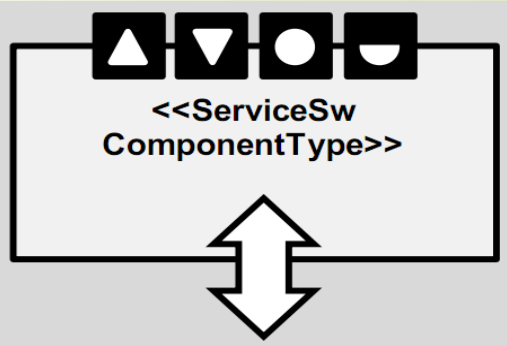
Service proxy SWC

- responsible for distribution of modes through the system (inter-ECU) since AUTOSAR's mode switch system only supports intra-ECU communication
- each ECU will need a copy of this since service proxy SWCs are to take care of informing ECUs of the mode changes



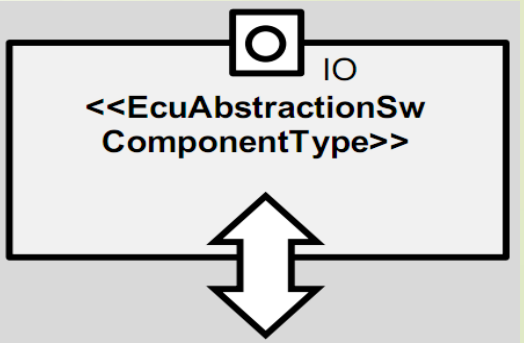
Service software component

- provides services specified by AUTOSAR through interfaces specified by AUTOSAR. This component may interact directly with modules from BSW
- Represents the different BSW Module services in the VFB view



ECU-abstraction software component

- it provides access to the ECU's IOs. It can interact with certain BSW modules. (again that arrow thingy at the bottom). The services are usually provided through PPorts and are used by SensorActuator SWCs
- BSW layer
- Represents the ECUAbstraction layer and its services
- The only SWC that is allowed to access IO ports



SWC Connectors

- Assembly connector: all the connections between SWCs are of this type
- Delegation connector: in a composition SWC, the ports of the inner SWCs that need to be visible from the outside of the composition, i.e. the composition SWC's ports need to be connected to delegation connectors

Example of access pattern to sensors and actuators:

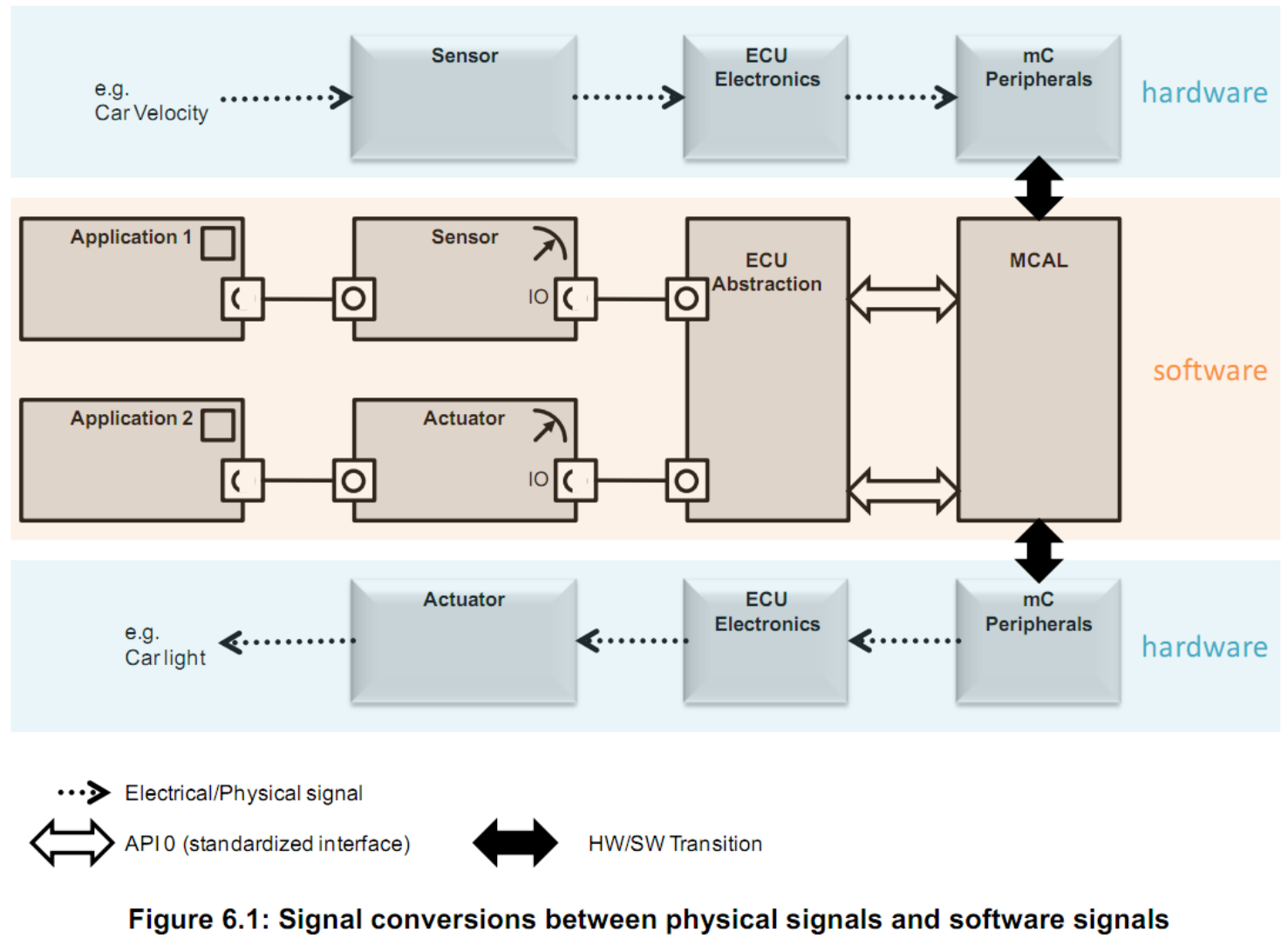
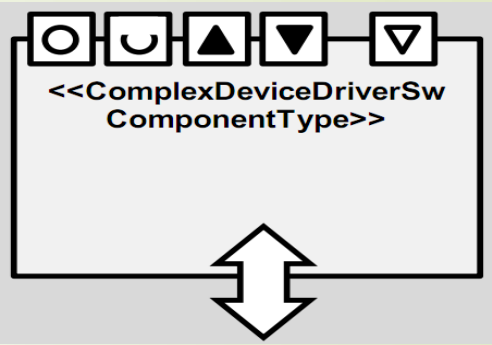


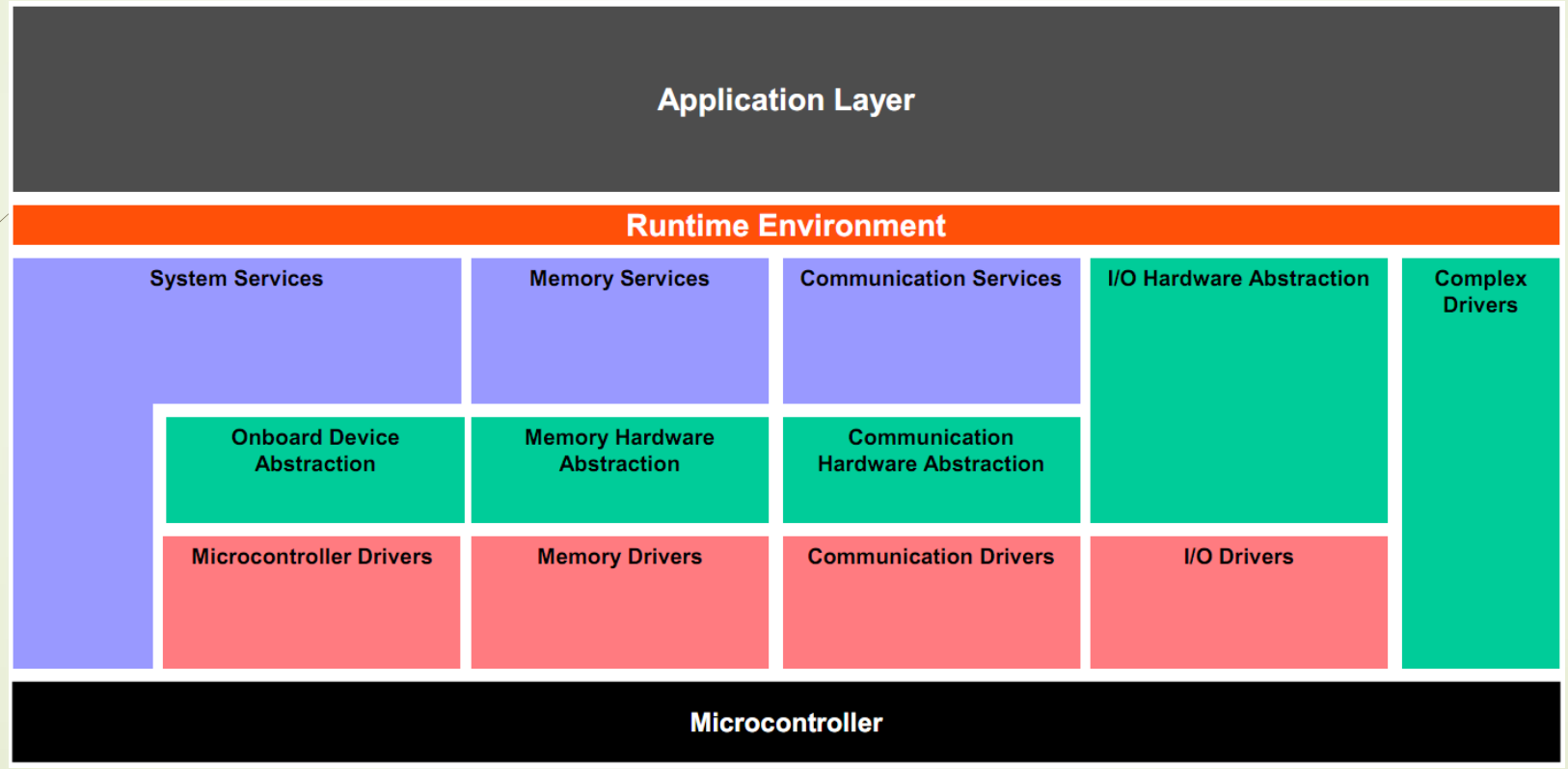
Figure 6.1: Signal conversions between physical signals and software signals

Complex driver software component

- it generalizes the ECUAbstraction component. It can define ports to interact with components in specific ways and can also directly interact with BSW modules
- Complex device drivers can use BSW services
- Complex device drivers exist to fulfill certain needs:
  - Implementing a complex application that cannot be otherwise implemented due to the AUTOSAR BSW layered architecture
  - Timing critical applications
  - Non-AUTOSAR applications within AUTOSAR ECU

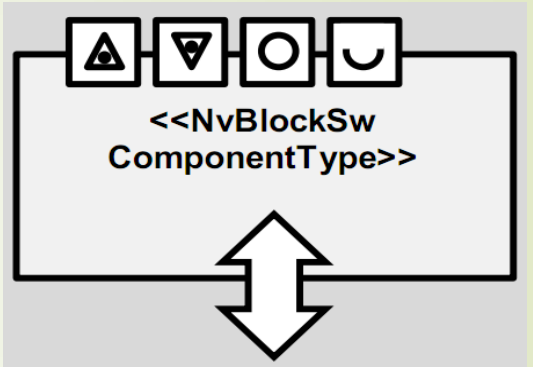


AUTOSAR Layered Architecture



NVBlock SWC

- it allows SWCs to access NV data
- It represents the Nvmanager from BSW layer

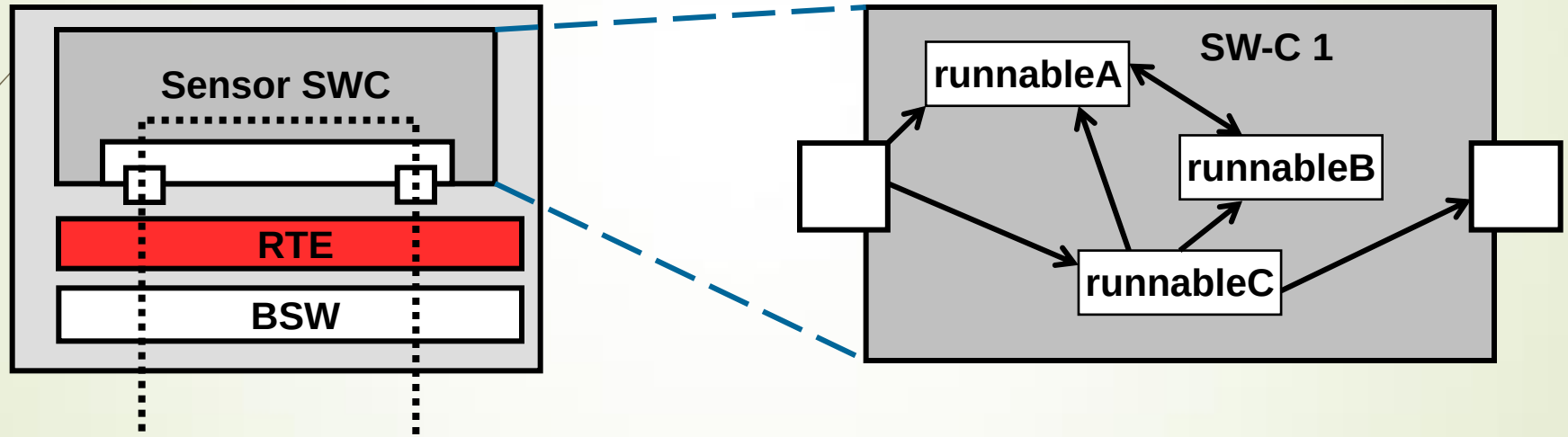


SWC elements

- Ports
  - PPort → provide port
  - Rport → require port
  - PRPort → provide require port
- Internal Behavior
  - Runnables
    - RTE Events
    - InterrunnableVaribales
- Implementation (source or object code)

SW Components and Runnables

- SW-Components
  - atomic components with respect to *mapping*
  - provided by one supplier
- Runnables
  - atomic components with respect to *execution*
  - attached to different OS Tasks

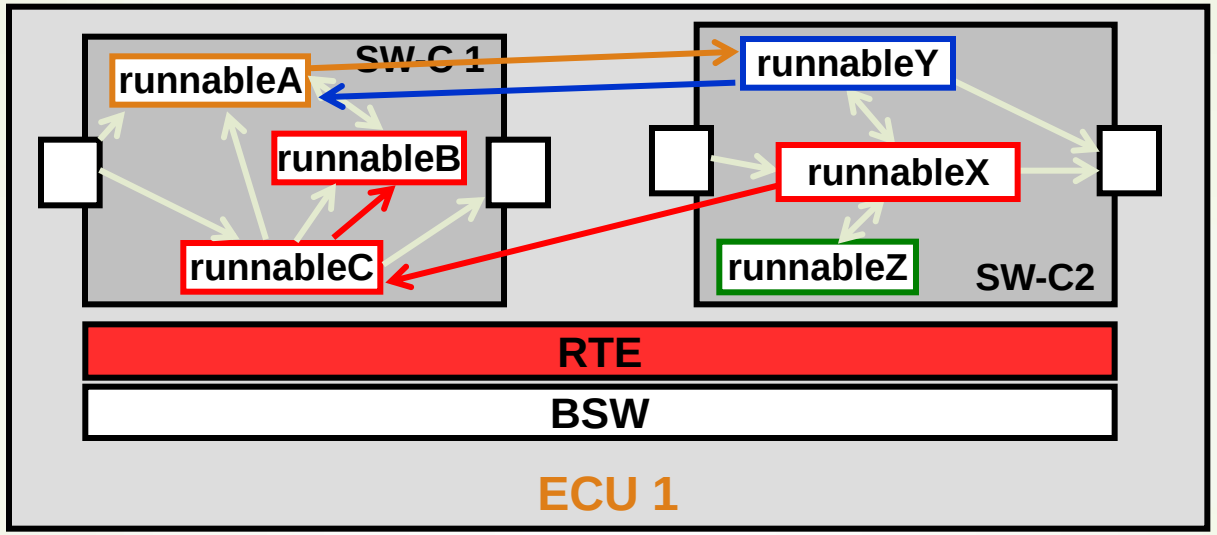


SWC Description and Elements

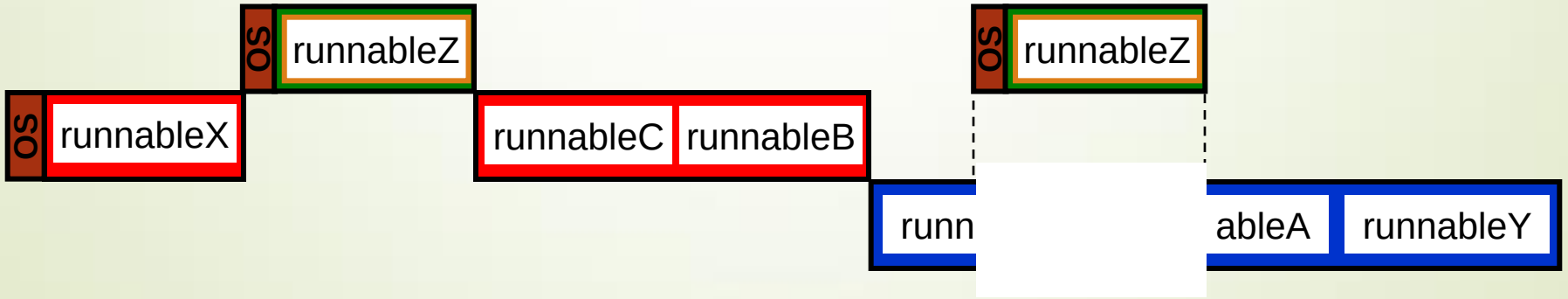
```
component application appswc{
    ports{
        sender swcpport provides sri1
        receiver swcrport requires sri2

    }
}
internalBehavior appswcib for appswc{

    runnable appswcrun1 [0.0]{
        symbol "appswcrun1"

        dataReceivedEvent swcrport.data2 as appswcgotit

    }

}
implementation appswcimpl for appswc.appswcib{
    language c
    codeDescriptor "src"
}
```

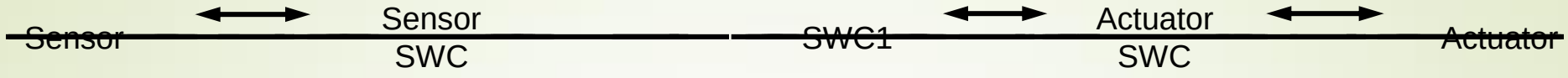
Runnables and Tasks

SW architecture example: 2 SW components, 6 runnables

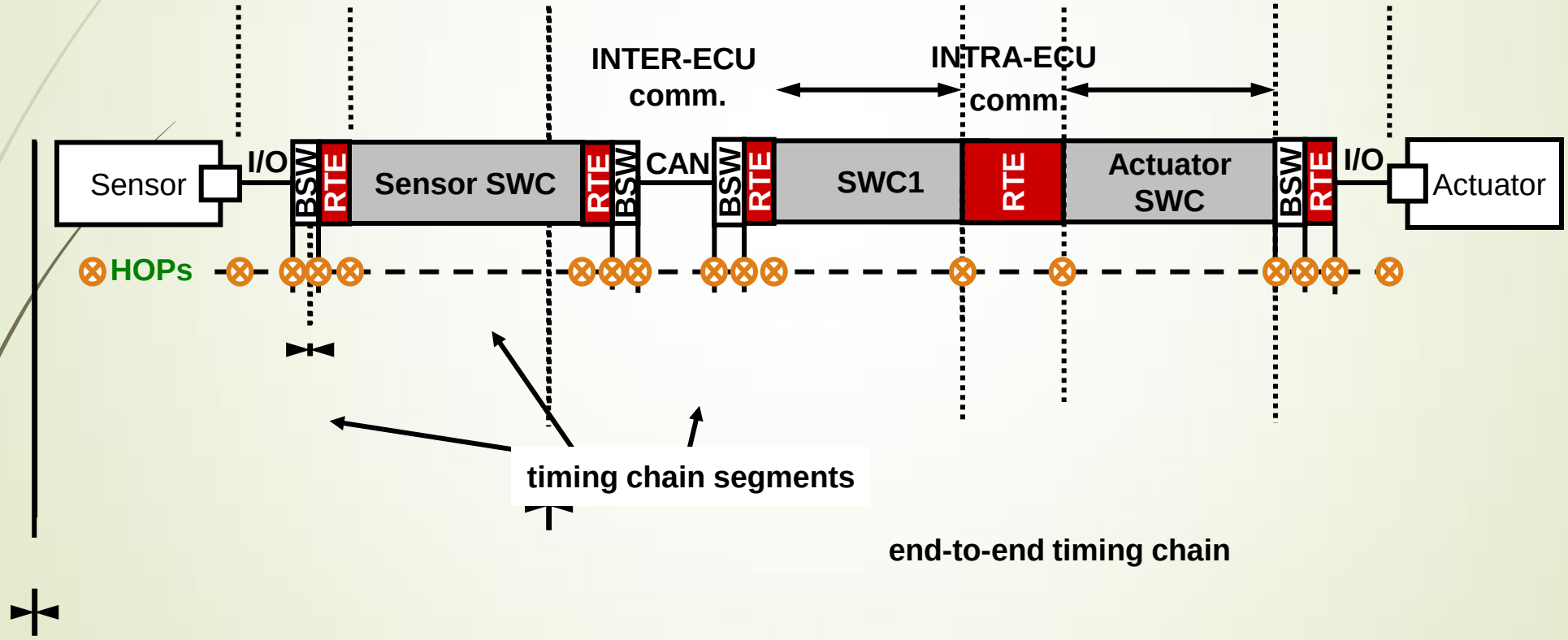


Schedule and timing dependencies





Timing Chains and Hand-Over Points



Port Interfaces

- SenderReceiverInterface
- NvDataInterface
- ParameterInterface
- ModeSwitchInterface
- ClientServerInterface
- TriggerInterface

Supported value encoding

Supported value encodings that can be used inside an AUTOSAR port:

- 2C: two's complement
- IEEE754: floating point numbers
- ISO-8859-1: ASCII-Strings
- ISO-8859-2: ASCII-Strings
- WINDOWS-1252: ASCII-String
- UTF-8: UCS Transformation format 8
- UCS-2: universal character set 2
- NONE: unsigned integer
- BOOLEAN: this represents an integer to be interpreted as boolean

The following types are applicable if the port typed by the interface is not a service port and hence, is a data port

SenderReceiverInterface

- Allows for the specification of the typically asynchronous communication pattern where a sender provides data that is required by one or more receivers (1:n or n:1)
- For SenderReceiverInterface, n:m while n or m are bigger than one is not possible
- Can invalidate receiving data
- handleInvalidEnum
  - dontInvalidate: invalidation is switched off
  - Keep: the error code returned by the RTE API will be used
  - Replace: replace a received invalid value. The replacement value is the initvalue

Example

```
interface senderReceiver sri1 {
    data uint8 data1 queued initValue 15
    data uint32 data2
    data float32 data3
}
```

ClientServerInterface

- A client may initiate the execution of an operation by a server that supports the operation
- The server executes the operation and immediately provides the client with the result(synchronous operation call) or else the client checks for the completion by itself (asynchronous function call)
- A client may not connect to more than one server such that one specific operation call would be handled by multiple servers (n:1)
- It is not possible to pass a reference to a ClientServerOperation as an argument in another ClientServerOperation

ClientServerInterface

- In a ClientServerInterface, a client requests an operation that is carried out by the server. The client will be notified of the operation's completion either by asking or waiting for the server to acknowledge the completion
- Client needs to provide values for ArgumentDataPrototypes that are "in" or "inout"
- A component can be both client and server

Example

```
interface clientServer csi1{
    error error1 1
    error error2 2
    operation btkint possibleErrors error1{
        in uint16 myarg1 policy useVoid
    }
    operation btkbool possibleErrors error2{
        out ^boolean myarg2 policy useArgumentType
        out uint32 myarg3 policy useArgumentType
    }
}
```

ClientServerInterface

- ServerArgumentImplPolicyEnum
  - useArgugemtType: the argument type of the runnable entity is derived from the AutosarDataType of the ArgumentPrototype
  - useArrayBaseType: the argument type of the runnable entity is derived from the AutosarDataType of the elements of the array that corresponds to the ArgumentPrototype. This represents the base type of the array in C
  - useVoid: the argument type of the runnable entity is void

Modes and ModeSwitchInterfaces

- Mode requester: the component that asks for a mode change from the mode manager. Uses the SenderReceiverInterface.
- Mode manager: the component or BSW module that own the modeGroup and can change the mode requested by a mode requester through a SenderReceiverInterface. The mode manager is responsible for changing the current mode through ModeSwitchInterfaces
- Mode user: a component that is notified by the mode manager of the new mode through a ModeSwitchInterface

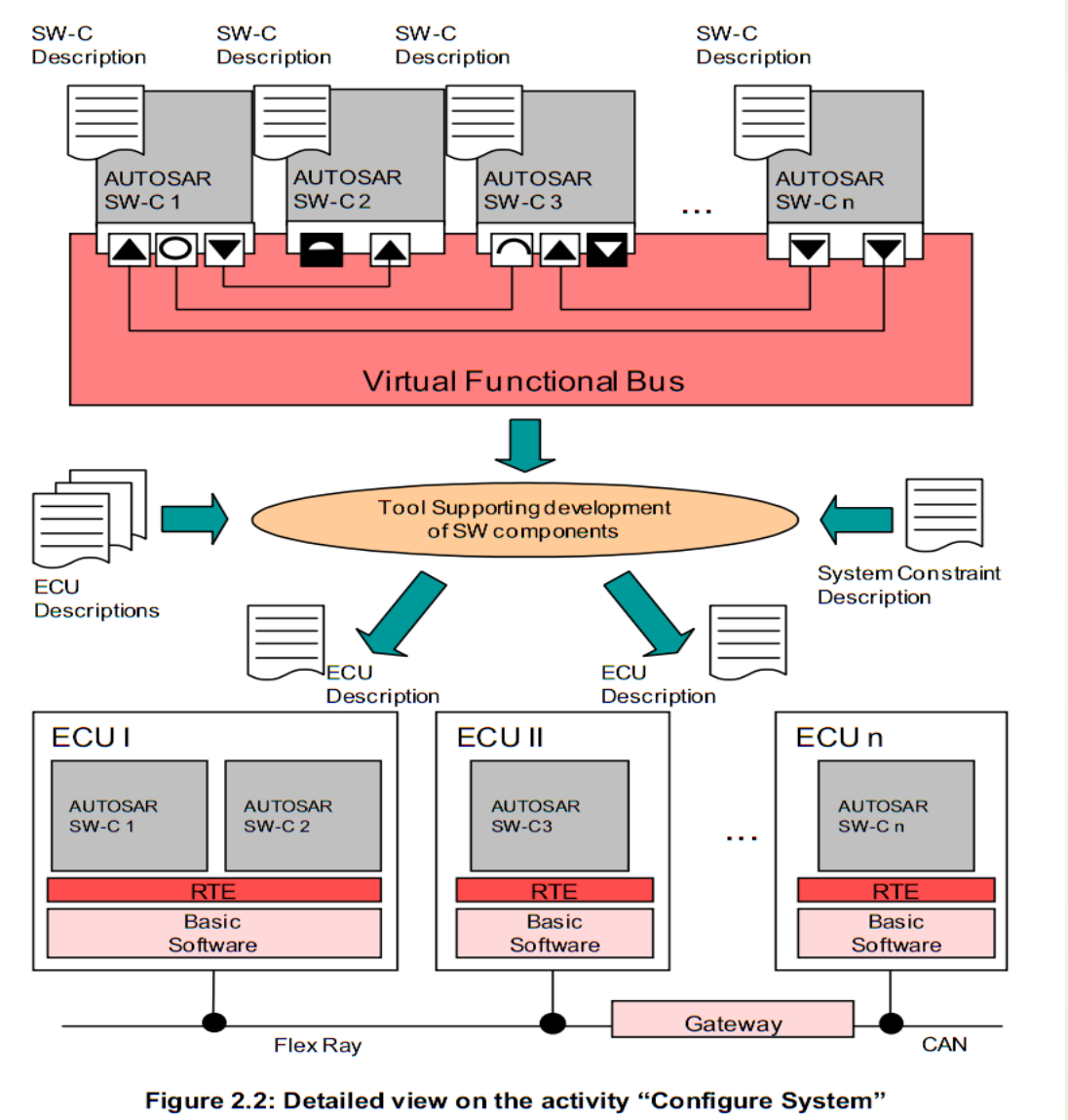
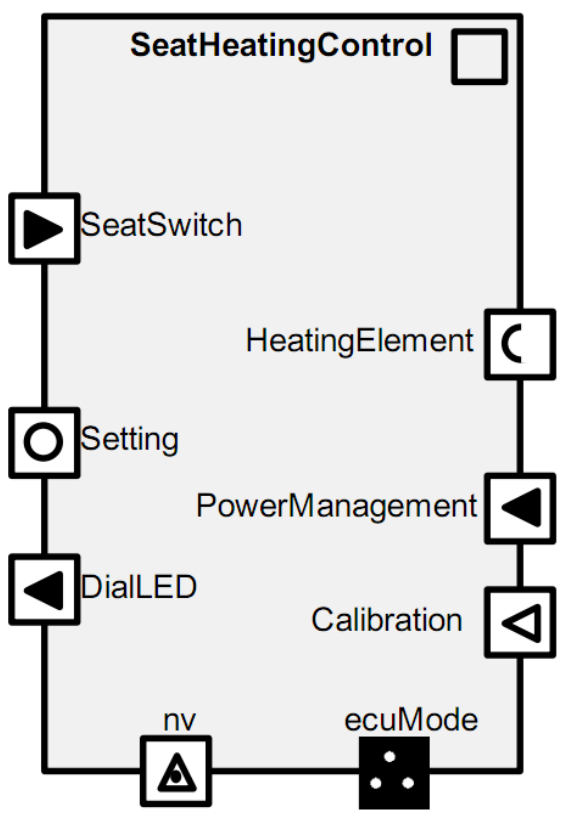


Figure 2.2: Detailed view on the activity "Configure System"

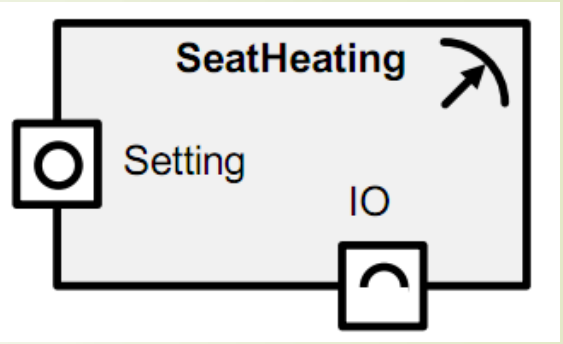
Example: Seat Heating Control

- Application SWC called SeatHeatingControl
- Ports:
  - Require ports:
    - If the seat is taken (SeatSwitch)
    - Setting of seat temperature dial(Setting)
    - Info from a power manager to decide when to turn off the heating (PowerManagement)
  - Provide ports:
    - dialled that is associated with the seat temperature dial (DialLED)
    - Heating element (HeatingElement)
  - The component can be calibrated (Calibration)
  - It needs the status of the ECU on which it runs (ecuMode)
  - Requires access to non-volatile memory (nv)

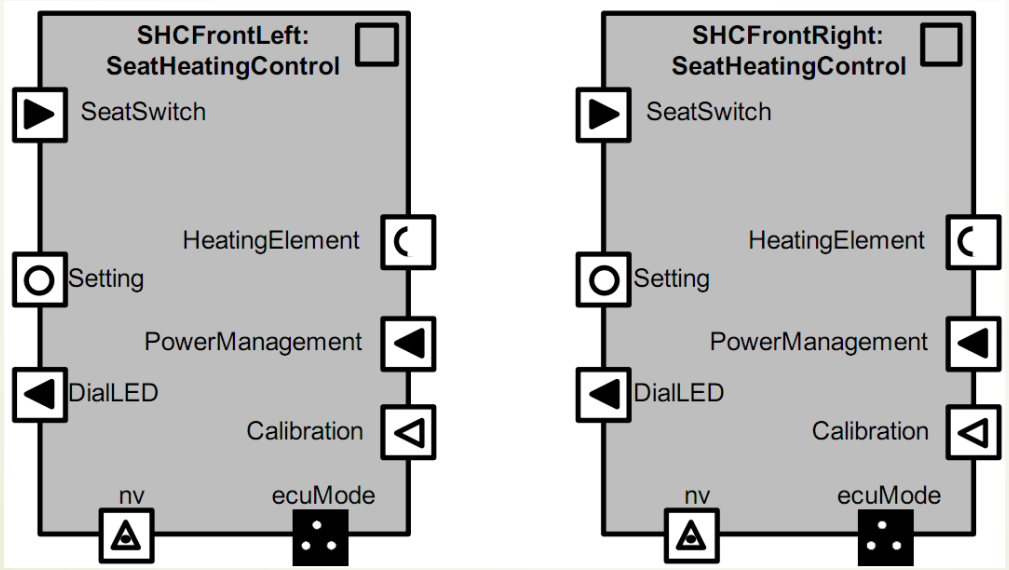


Example

- A SensorActuator SWC called SeatHeating
  - Inputs the desired setting of the heating element (Setting)
  - Directly controls the seat heating hardware (IO)

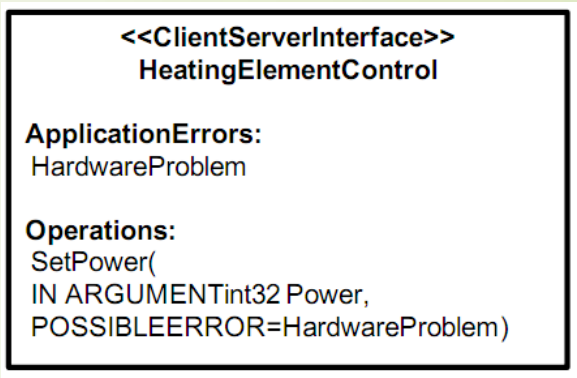


- AUTOSAR supports multiple instantiation

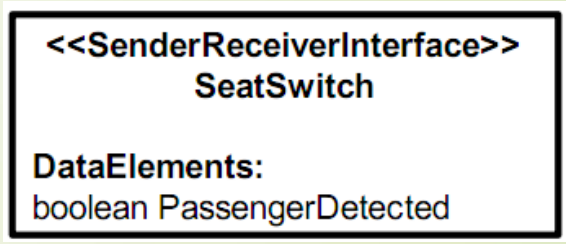


Example

- A ClientServerInterface defines a set of operations that can be invoked by a client and carried out by a server

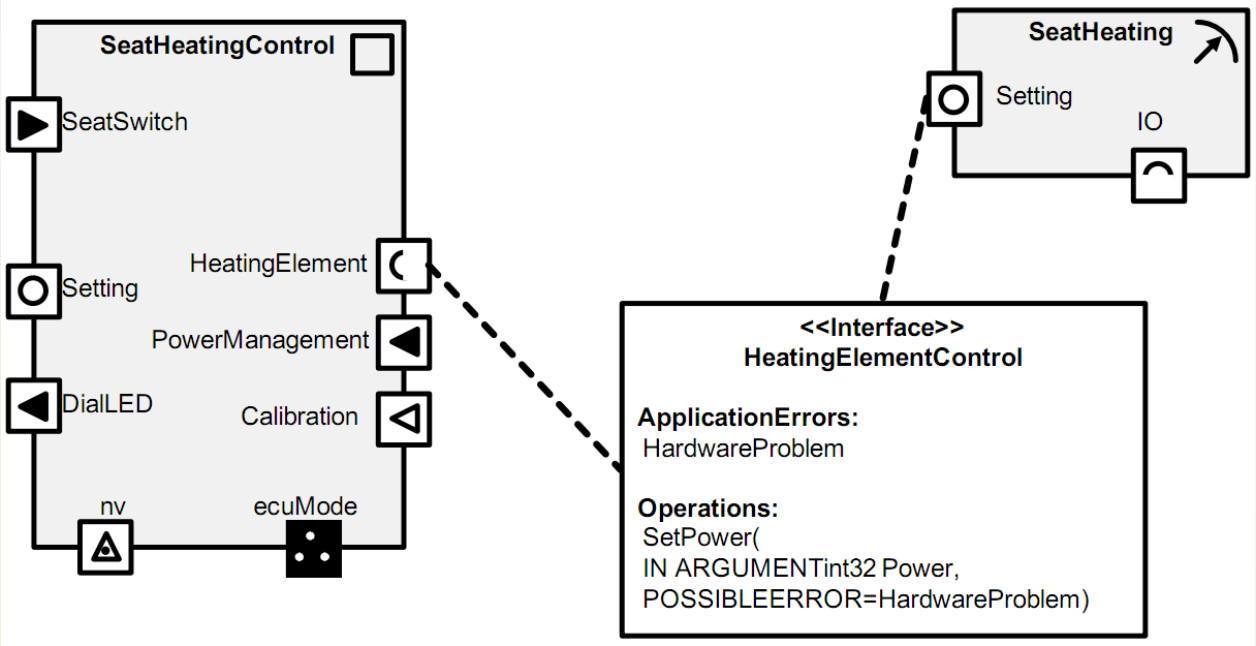


- A SenderReceiverInterface defines a set of data-elements that are sent and received



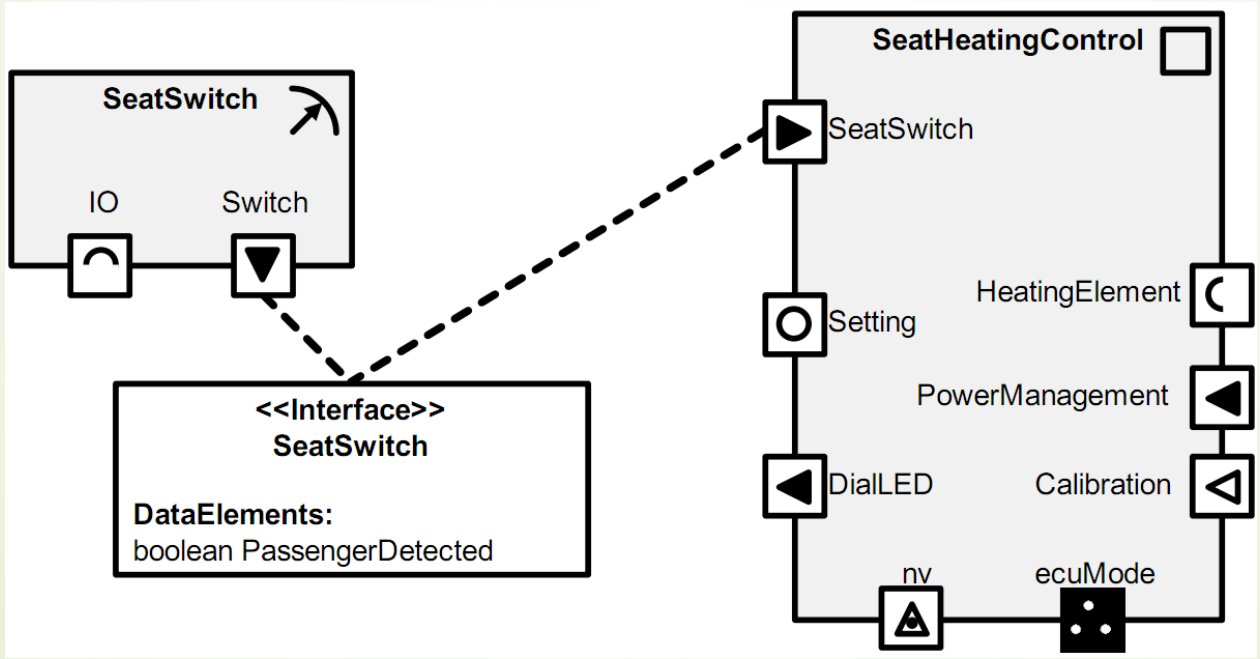
Example

- When a PPort provides a ClientServerInterface, the component to which the port belongs provides an implementation of the operations defined in the interface



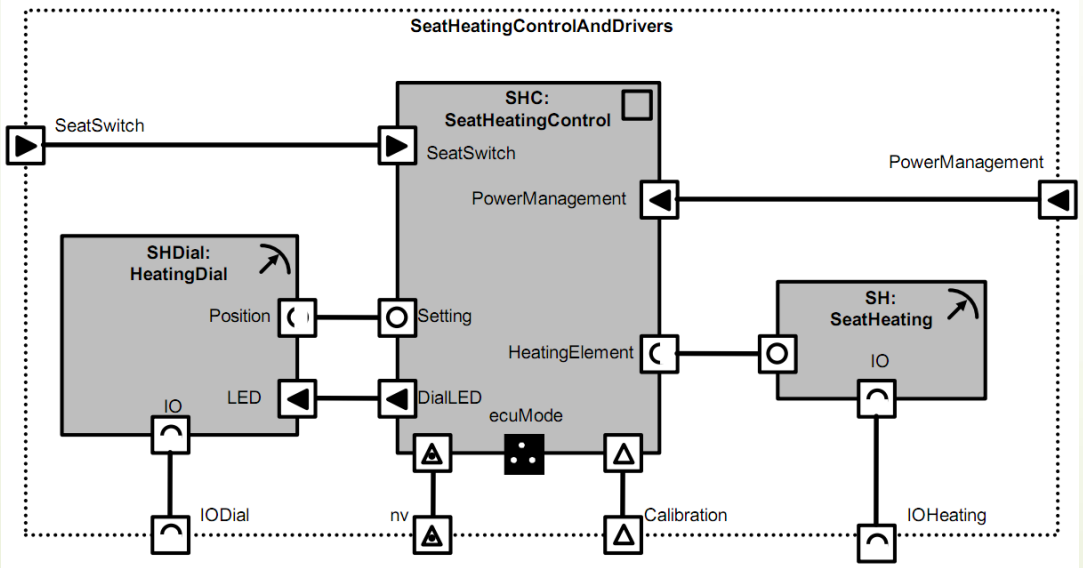
Example

- A component providing a SenderReceiverInterface generates values for the boolean value “PassengerDetected” through its port “Switch”. Similarly the component “SeatHeatingControl” can read the data-element “PassengerDetected” through its port “SeatSwitch”



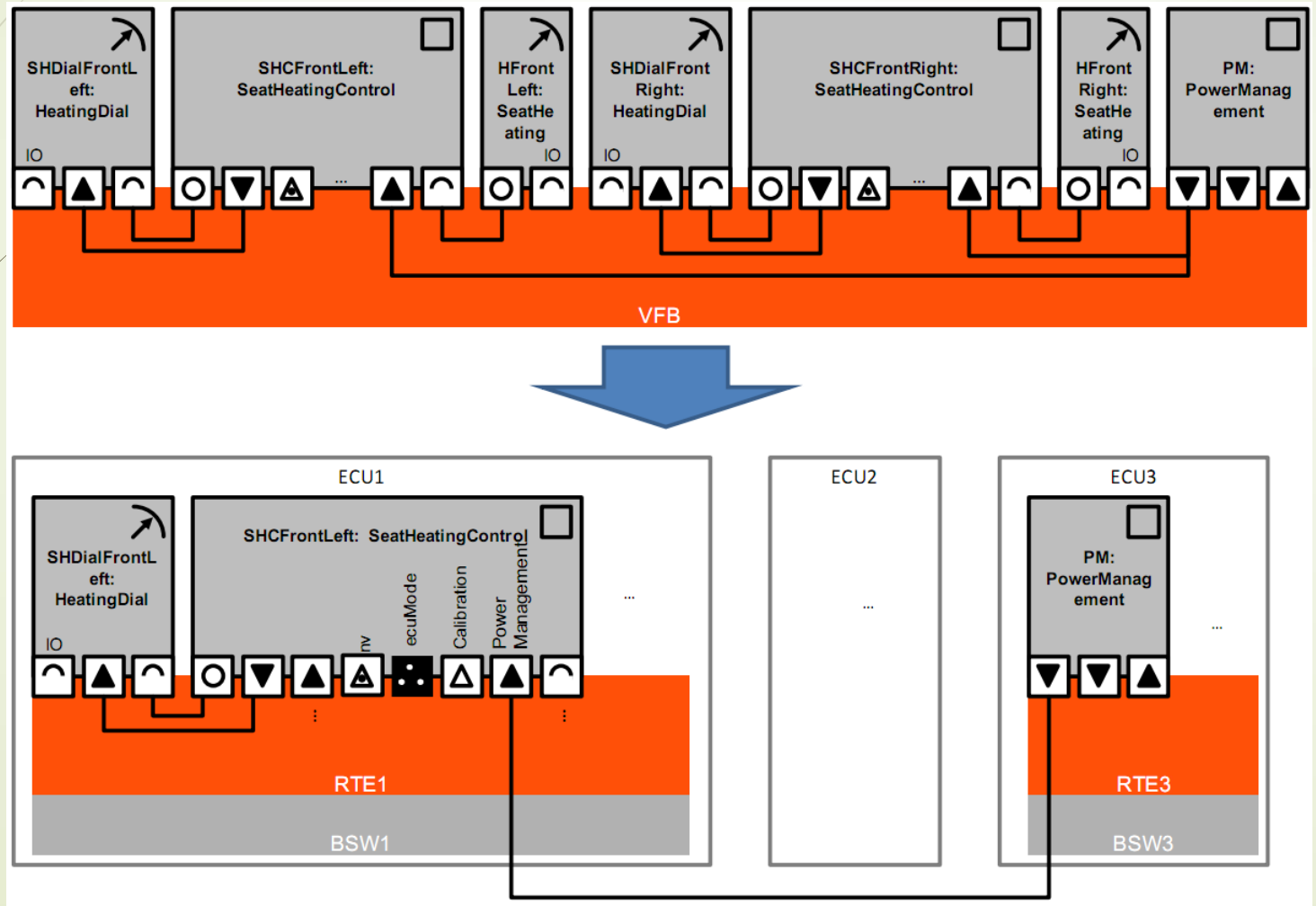
Example

- A sub-system consisting of usages of components and connectors is packaged into a composition. In AUTOSAR the usage of a component-type within a composition is called a prototype.
- This composition contains 3 prototypes: SHDial (type HeatingDial), SHC (type SeatHeatingControl), SH (type SeatHeating)

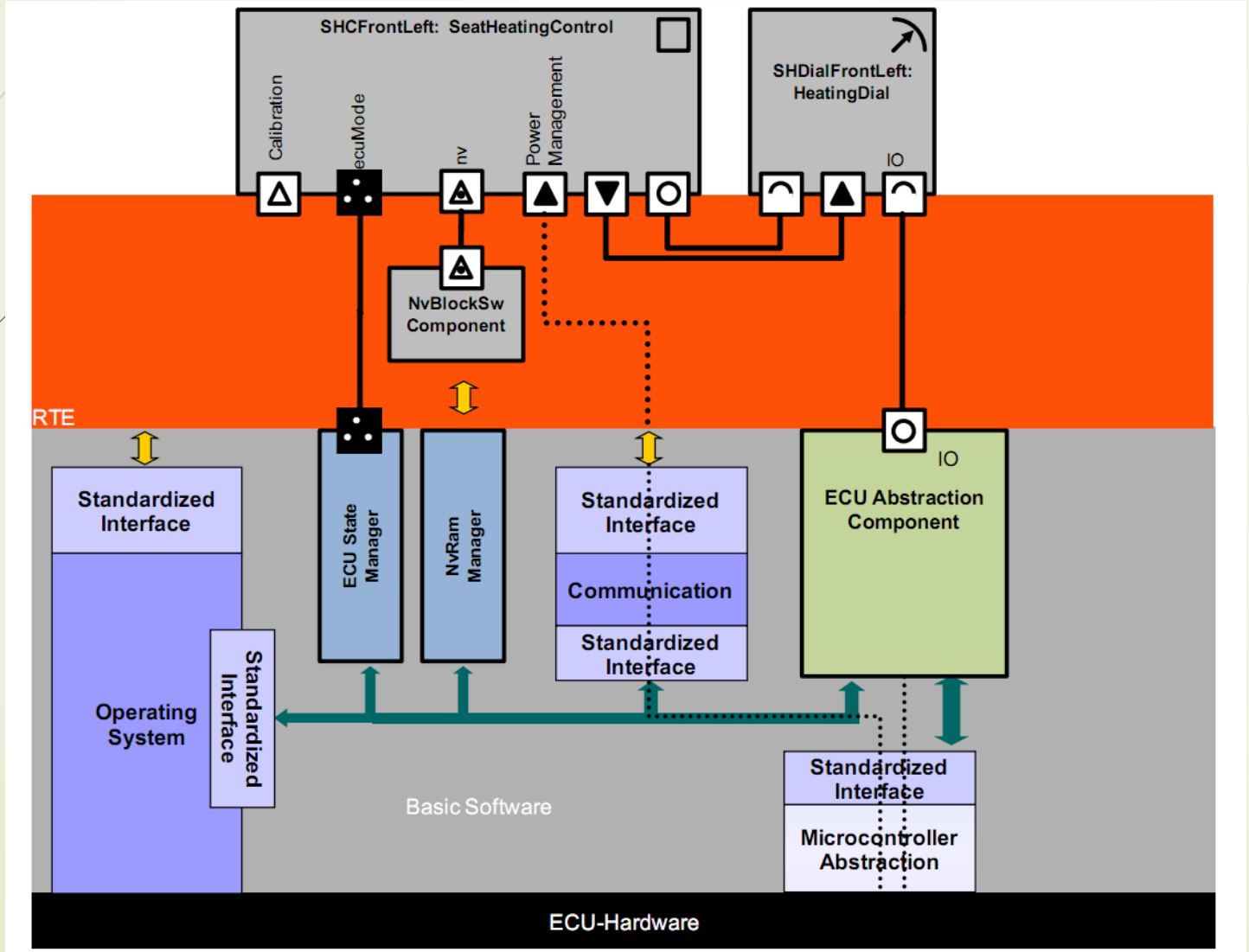


Example

- Mapping SWCs to ECUs



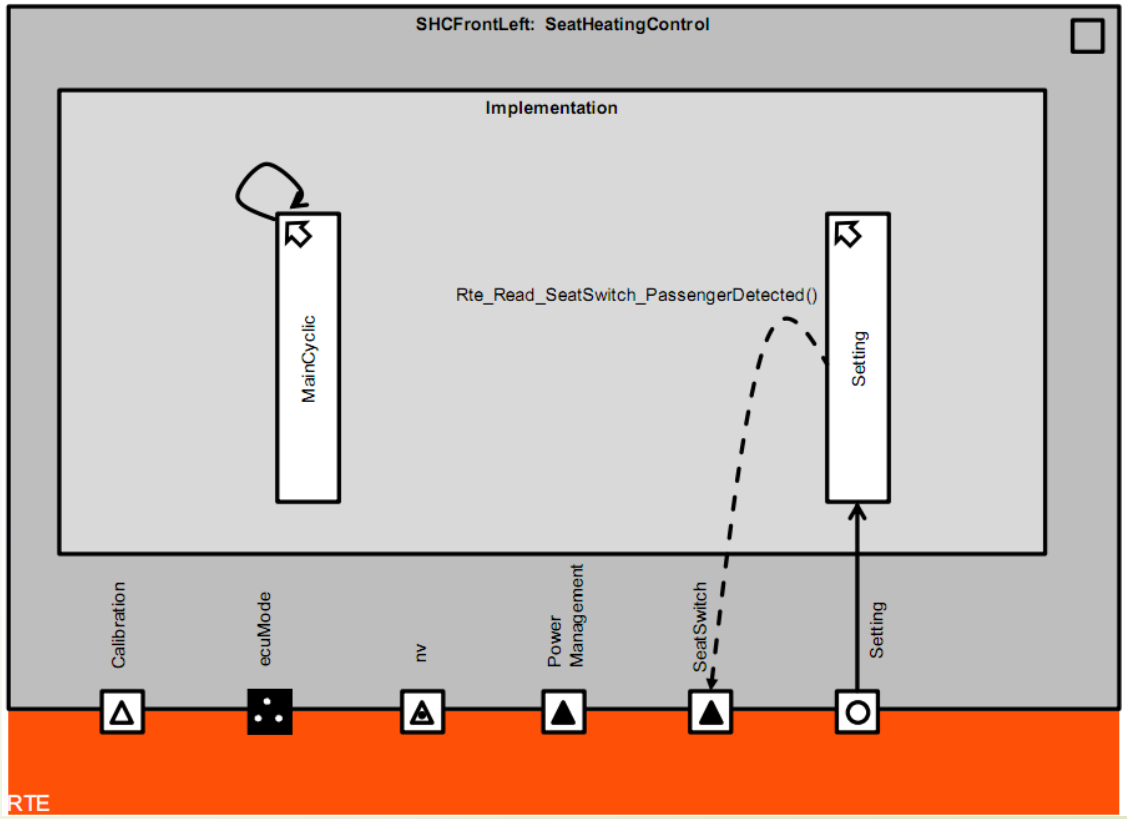
Example


SHCFrontLeft: SeatHeatingControl
Calibration
ecuMode
nv
Power Management
SHDialFrontLeft: HeatingDial
IO
NvBlockSw Component
RTE
Standardized Interface
ECU State Manager
NvRam Manager
Standardized Interface
Communication
Standardized Interface
IO
ECU Abstraction Component
Operating System
Standardized Interface
Basic Software
Standardized Interface
Microcontroller Abstraction
ECU-Hardware

Example

- Runnable entities
- A runnable entity is a sequence of instructions that can be started by the RTE
- A runnable runs in the context of a task (OS task)
- A task provides resources such as context

and stack size to a runnable



RTE Events

- asynchronousServerCallReturns: raised when an asynchronousservercall is finished
- dataReceiveErrorEvent: raised by RTE when the com layer detects and notifies an error regarding the received data is reported, references a variabledataprototype. Two cases where the com layer notifies the event:
  - The data value is equal to a predefined invalid value
  - The last update time of a periodic signal exceeds the alivetimeoutvalue
- dataReceivedEvent: raised when a referenced variabledataprototype is received
- dataSendCompletedEvent: raised when a sender has completed the transmission of the reference dataprototype or when an error is raised
- dataWriteCompletedEvent: raised when an implicit write access was successful or an error occurred

RTE Events

- modeSwitchEvent: raised when a mode change is received
- operationInvokedEvent: raised when an operation referenced by the interface is requested by the client
- Timingevent: raised periodically by RTE

Runnable Entity Attributes

- asynchronousServerCallResultPoint: the owning runnable entity is entitled to fetch the result of the asynchronous server call
- dataReadAccess: runnable entity has implicit read access to data element of a senderreceiver or nv portprototype
- dataReceivePoint: runnable entity has explicit read access to data element of a senderreceiver or nv portprototype
  - dataReceivePointByArgment: the result is passed back to the application by means of an argument in the function signature
  - dataReceivePointByValue: the result is passed back to the application by means of the return value
- dataSendPoint: runnable entity has explicit write access to dataElement of a senderreceiver or nv portprototype

Runnable Entity Attributes

- dataWriteAccess: runnable entity has implicit write access to data element of a senderreceiver or nv port
- Modeaccesspoint: a mode access point is required by a runnable entity owned by a mode manager or mode user. Its semantics implies the ability to access the current mode
- modeSwitchPoint: required by a runnable entity owned by a mode manager. Its semantics imply the ability to initiate a mode switch
- parameterAccess: the presence of a parameteraccess implies that a runnable entity needs read only access to a parameterdataprototype which may either be local or within a portprototype

Runnable Entity Attributes

- serveCallPoint: if a runnable entity owns a server call point it is entitled to invoke a particular client server operation of a specific rportprototype of the corresponding atomicswcomponenttype
- Waitpoint: has a trigger that its waiting for
- externalTriggeringPoint: if a runnable entity owns it, it is entitled to raise an externaltriggeroccurredevent
- internalTriggeringPoint: if a runnable entity owns it, it is entitled to trigger the execution of runnable entities of the corresponding SWC
- The term implicit is used for communication based on data-access and explicit is used for data-point based communication

Measurement

- Only the following can be measured in AUTOSAR:
  - In the context of communication between SWCs:
    - VariableDataPrototypes enclosed in a SenderReceiverInterface
    - Argument of ClientServerOperations enclosed in a ClientServerInterface
  - In the context of a single SWC (internal):
    - Content of InterrunnableVariables which are used for communication between runnables of one AUTOSAR SWC

Timing

- VFBTiming: this view deals with timing information related to the interaction of SwComponentTypes at VFB level
- SwcTiming: this view deals with timing information related to the SwcInternalBehavior of AtomicSwComponentTypes
- SystemTiming: this view deals with timing information related to a system utilizing information about topology, software deployment and signal mapping
- BswModuleTiming: this view deals with timing information related to the BswInternalBehavior of a single BswModuleDescription
- EcuTiming: this view deals with timing information related to the EcuValueCollection, particularly with the EcuModuleConfigurationValues

- Queued communication is not available for dataElements owned by PRPorts
- If swImplPolicy is set to any other value (other than queued, i.e. FIFO) than queued then LIFO applies.

- Variant handling: it allows designers at many levels to put together a superset of functionality and choose which actual pieces of this functionality will be enabled in a specific variant
- AUTOSAR supports several discrete binding times:
  - System design
  - Code generation
  - Pre compile
  - Link time
  - Post build