Autosar Architecture

Software Component and Compositions

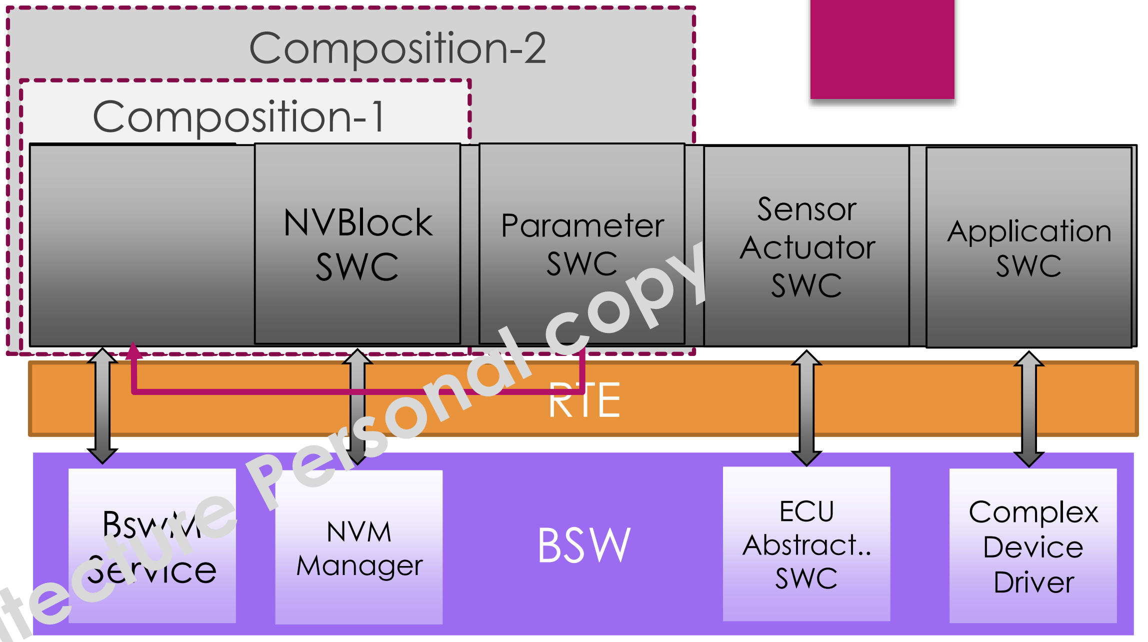
Software Components

- Application software within Autosar is organized in self contained units called Atomic Software Component types.
- Such Atomic Software components together form the complete functional implementation of the software

Software components takes one of the below types

- ApplicationSwComponent ✓
- NVBlockSwComponent ✓
- ComplexDeviceDriverSwComponent ✓
- ServiceSwComponent ✓
- ServiceProxySwComponent ✓
- EcuAbstractionSwComponent ✓
- SensorActuatorSwComponent ✓

- ParameterSwComponent ✓
- CompositionSwComponent ✓



- **Application** software component holds the functionality of the software. Example: Calculations, Functional/decision making Algorithms etc..
- **NVBlock** component is used when we have interfaces on the application layer to be stored on NVM memory. It interacts with the NVRAM Manager
- **CDD** component provides an easy access to hardware directly from application layer to fulfil special timings and functional requirements
- **Service** component is used for configuring services for a particular control unit
- **Service proxy** component is used when a particular service component is to be accessed from different control units
- **EcuAbstraction** component is a part of BSW, which acts as an interface between MCAL and SensorActuator component on the ASW
- **SensorActuator** component is used on the application layer to interact with the BSW ECUAbstraction layer, and acts as a interface to the other application compoents
- **Parameter** component is a part of software component types that provide only calibrations to the software. Example: Tuning vehicle performance using parameters during testing's
- **Composition** aggregates components and connections between its sub Components

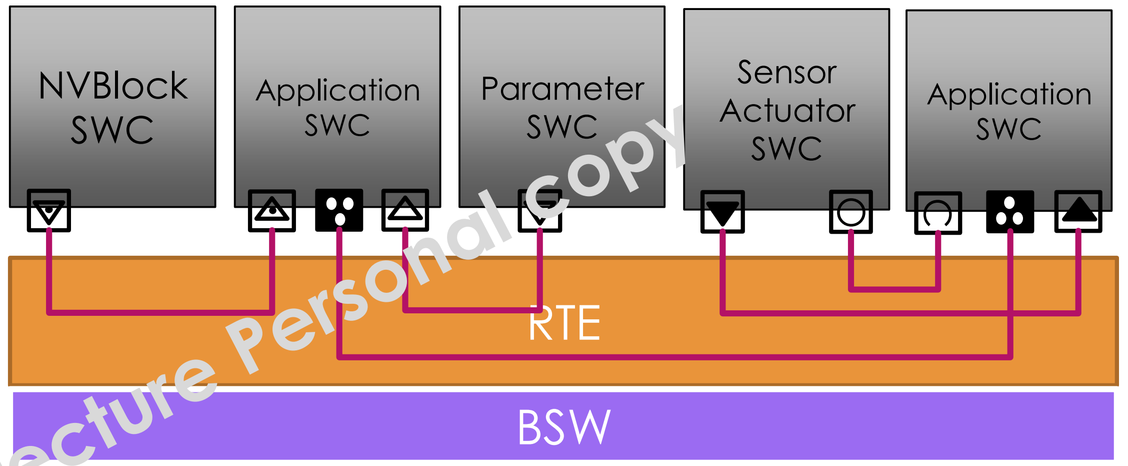
Autosar Ports and Port Interfaces

Ports: Autosar architecture proposes Ports as the mode of communication between Autosar modules

- Provider Port (P-Port)
- Receiver Port (R-Port)
- ProviderReceiver Port (PR-Port)

Port Interfaces: The kind of information that are communicated between ports are defined by port interfaces

• Sender Receiver Interface	P-Port ▼	R-Port ▲
• Client Server Interface	P-Port ◎	R-Port ◌
• NVData Interface	P-Port ▽	R-Port △
• Parameter Interface	P-Port ▽	R-Port △
• ModeSwitch Interface	P-Port ⁘	R-Port ⁘
• Trigger Interface	P-Port ∨	R-Port ∧



SR Interface is used to communicate interfaces between components Port writing the interface is the Provider and receiving end is the Receiver

CS Interface is used to call services or functions from another components Component owning the service is the server and the caller is the Client

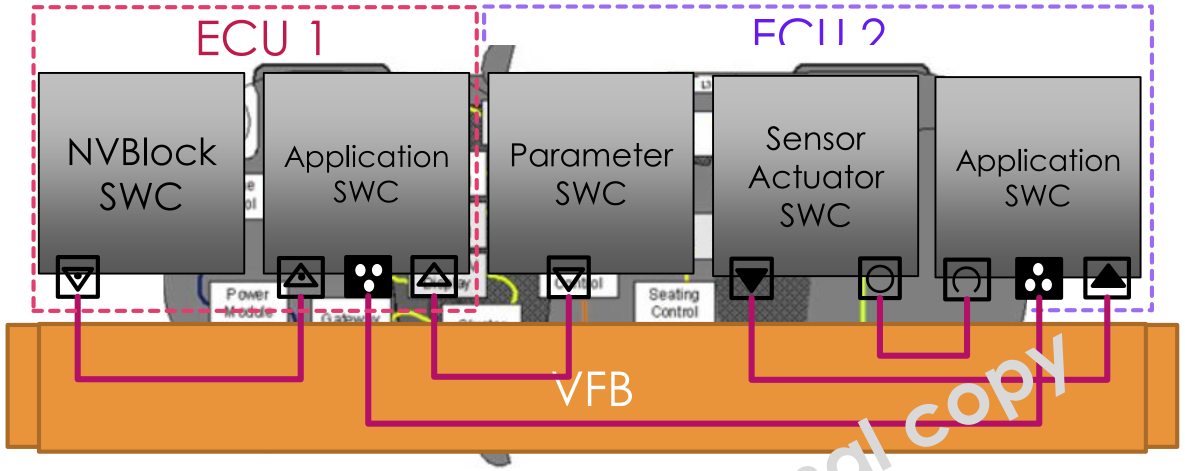
NVData Interface is to communicate with NVBlock SWC to send and receive non volatile memory interface

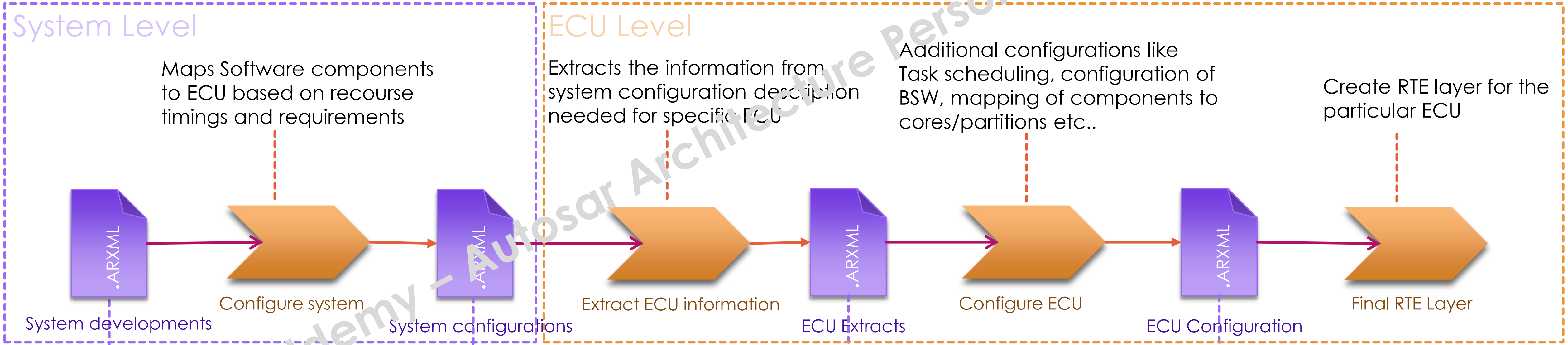
Parameter interface is used for exchanging calibrations or constants across components

Mode switch interface is used for notification of a software component of different states that the system can enter

Trigger interface induces as a trigger execution for other components

Autosar Methodology





Maps Software components to ECU based on recourse timings and requirements

Extracts the information from system configuration description needed for specific ECU

Additional configurations like Task scheduling, configuration of BSW, mapping of components to cores/partitions etc..

Create RTE layer for the particular ECU

Software Components, Hardware Selection, Overall System, ECU recourses

Output configuration system in system configuration description. Includes BUS mapping, Topology etc.. And mapping of software components to ECU

Outputs of Ecu Extract. Which Mainly Contains

Flatview.arxml
Flatmap.arxml
EcuExtract.arxml

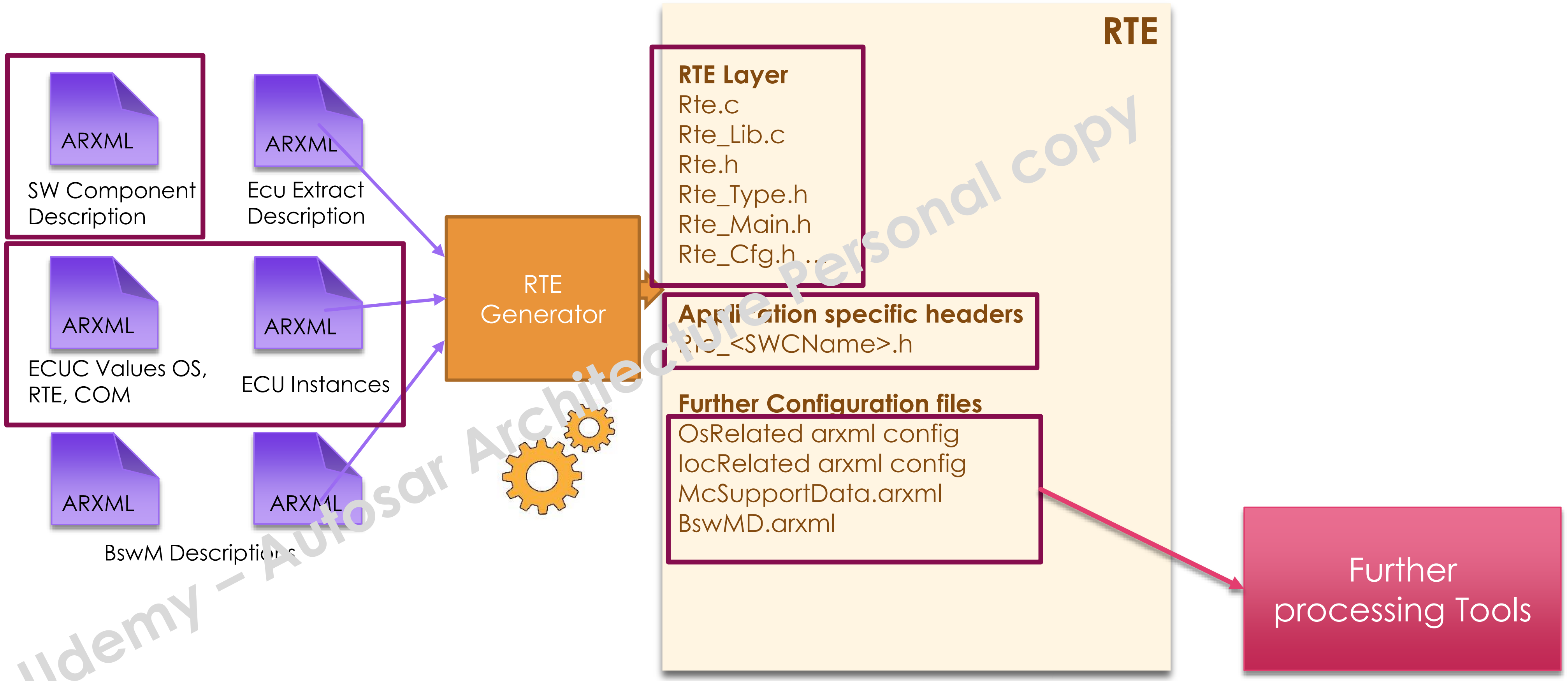
Outputs of ECU configuration. We have all inputs related to a local ECU

Autosar RTE API's

Specification AR4.4: AUTOSAR_SWS_RTE.pdf
Section- 5.6 API Reference

	Functionality	RTE API's
Sender Receiver Interface	dataReadAccess (Implicit)	Rte_IRead , Rte_IStatus, Rte_IsUpdated
	dataWriteAccess (Implicit)	Rte_IWrite, Rte_IWriteRef, Rte_IInvalidate, Rte_IFeedback
	dataSendPoint (Explicit)	Rte_Write (Non-Queued), Rte_Send (Queued)
	dataReceive- PointByArgument (Explicit)	Rte_Read
	dataReceive- PointByValue (Explicit)	Rte_DRead
Client Server Interface	Clientserver- ServerCallPoint	Rte_Call
	AsynchronousServerCallResultPoint	Rte_Result
Mode Switch Interface	ModeSwitchPoint	Rte_Switch, Rte_SwitchAck
	ModeAccessPoint	Rte_Mode
Parameter Interface	Port - ParameterInterface	Rte_Prm
	ParameterDataPrototype - shared or PerInstance	Rte_CData
	PerInstanceMemory	Rte_Pim
	readLocalVariable - Explicit IRV	Rte_IrvRead
	readLocalVariable - Implicit IRV	Rte_IrvIRead
	writeLocalVariable - Explicit IRV	Rte_IrvWrite
	writeLocalVariable - Implicit IRV	Rte_IrvIWrite
	InternalTriggeringPoint	Rte_IrTrigger
	ExternalTriggeringPoint	Rte_Trigger
	PortAPIOption-indirectAPI	Rte_Port
	ExclusiveArea	Rte_Enter, Rte_Exit

Autosar RTE Generator


ARXML
SW Component Description
ARXML
Ecu Extract Description
ARXML
ECUC Values OS, RTE, COM
ARXML
ECU Instances
ARXML
ARXML
BswM Descriptions
RTE Generator
RTE
RTE Layer
Rte.c
Rte_Lib.c
Rte.h
Rte_Type.h
Rte_Main.h
Rte_Cfg.h ...
Application specific headers
Rte_<SWCName>.h
Further Configuration files
OsRelated arxml config
IocRelated arxml config
McSupportData.arxml
BswMD.arxml
Further processing Tools