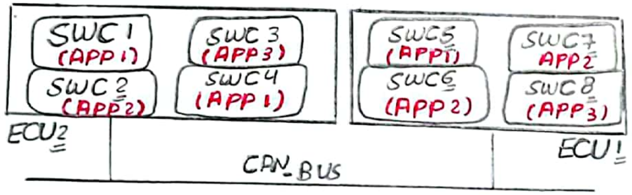
⇒ RTE : Runtime Environment

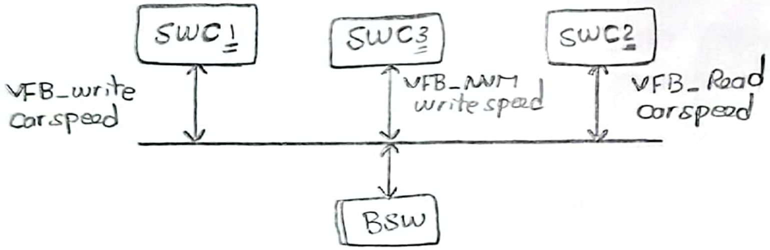
- Application may consist of more than one SWC
- SWCs may exist on different ECUs according
- to physical - requirement limitations.



↳ challenges:

- Application shall be HW - system Independent
- " development shall be accelerated as much as possible
- Application shall be maintainable / scalable

To overcome these challenges, AUTOSAR Introduce VFB: Virtual Function BUS



→ Through VFB : simulate
- Communicate with other SWCs
- " " sensors - Actuator
- " " standard services
- scheduling

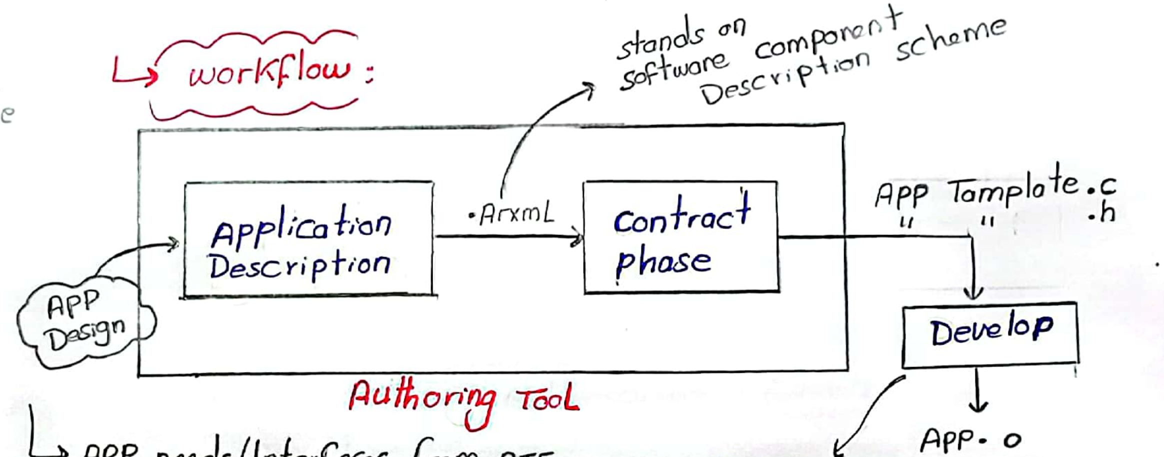
Note: All part of VFB simulated on Disk top APP

↳ Deployment of VFB :

- virtual connections between SWCs
  ↳ mapped to local connections or Network connections
- Virtual sensors - Actuators
  ↳ mapped to real Hardware
- standard services
  ↳ mapped to BSW services
- windows threads scheduling
  ↳ mapped to AUTOSAR Tasks

RTE:

↳ workflow :



↳ APP needs/Interfaces from RTE

developing required logic Inside Interfaces template

- Create SWC
- " port Interfaces
  - sender/Receiver
  - client/server
  - Mode switch
- Data element
- create port directions
  - provided - required

for each SWC
- Runables
- Events
- Access point

→ SWCD : Software Component Description:

- Application is organized in SWCs
- each SWC Implement apart of App functionality
- SWC is atomic : Can't be distributed in multiple ECUs
- SWC communicate with outside using ports

↳ ports :

- Provided port (output)
- Required port (Input)

→ Provide / Required < Data / Service

→ each port typed by port Interface

↳ port Interface :

- define type of communication between SWCs

Example :

- Sender / Receiver : Data Exchange
- Client / Server : operation "

↳ In Sender / Receiver Interface
- must define Exchanged data < Name / Type

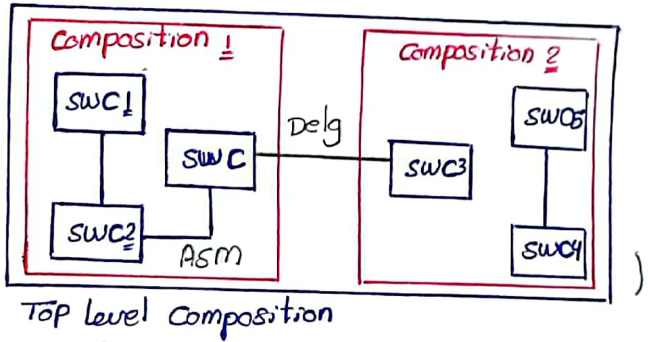
↳ In client / server Interface
- must define Exchanged operation prototype < Name / Args / Return

- Mode switch : Mode/state Exchange
- parameter Interface : Constant Exchange
- Nv Interface : Nv Data read/ write
- Trigger Interface : Trigger execution of other swc

↳ Compositions :

: grouping of SWCs based on functionalties

- reduce complexity when designing App
- can be nested



→ Assembly connectors : used to connect SWCs at the same composition

or between composition at the same level (P-R)

→ Delegation connectors : used to connent SWCs at diff composition

or between compositions at different level (P-P) (R-R);

↳ Interal Behavior : for each SWC

- Functions (Runnables)
- How Runnables are executed (Events)
- Provider / Required Info (Access points)

↳ Runnable Config :

- short Name : Name on .arxml file
- minimum start Interval : start Delay
- Can be Invoked concurrently < Reentrunt / Non Reentrunt
- symbol : function

mode

↳ Events (Triggers) :

- descripe how runnables will be Triggered
  - cyclic
  - response to certain trigger

↳ Events config :

- short Name
- Triggered Runnables
- Data received event
- periodicity
- :

↳ Access point : (IO for Runnables)

- define provided / Required info for each runnable
- Each access point mapped to RTE_API
- needed by RTE to manage Run time behavior

↳ Access point config :

- short Name
- port
- Element

↳ DataTypes :

- needed for the definition of
  - Data elements types for (SR)
  - args types for (CS)
- Datatypes ⟨ Application Datatype / Implementation ″
- All Datatypes defined in SWCD are generated by RTE In "Rte_Types.h"

↳ Application Datatypes :

- Represent physical Range
  - Example : carspeed [0 : 280] Km/H
- RTE use :
  - Computation method : for physical Range conversion
  - Data constrains : for range check
  - Unit :

↳ Implementation Datatypes:

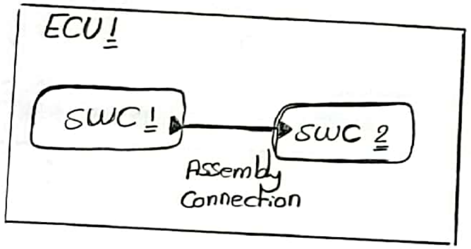
- Represents Implementation level Logical Range
- Can be : Arrays, pointers, structures.. premitives [uint8, ---]
  - Data Constrains
  - Computation method
  - Basetype [uint8, uint16, ---]

→ RTE ensures the runnables Invoked at the correct time

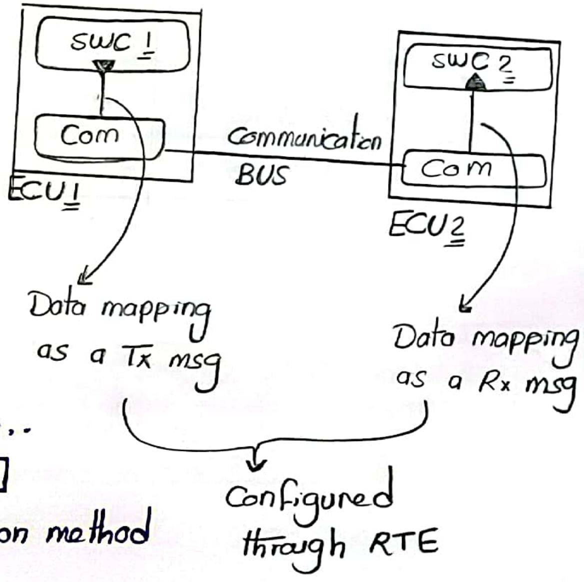
→ provide function to SWCs to access data or Invoke operation

→ provide all other resources the components needs

↳ Intra ECU:

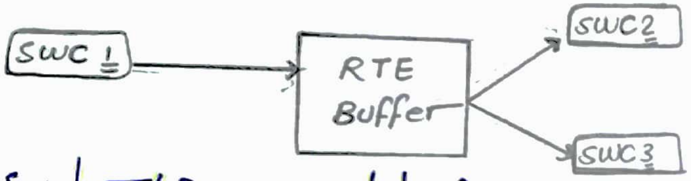


↳ Inter ECU:



Sender Receiver

- Data Exchange from n → 1
  1 → n
- this process managed through RTE "RTE protected buffer"



- Sender / Receiver Interface can contain one or more data element

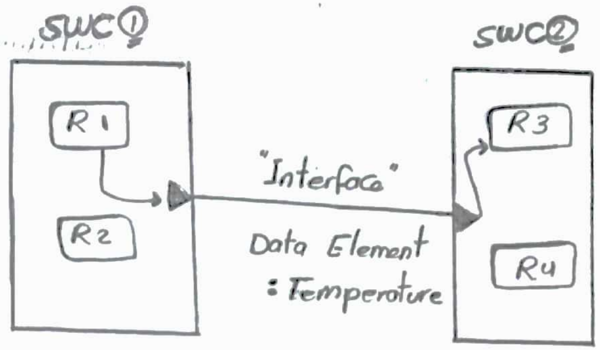
↳ Compatibility :

- provided - Required SR ports can be connected only
  - both ports are typed by the same sender receiver Interface
  - by compatible sender receiver Interface
    ↳ Compatible data elements
    - Same name
    - same type

we can ignore this step by port Interface mapping

→ this case may occur when two or more development team work on SWCs and doesn't have the same naming convension

↳ Access points : "a function generated by RTE and used on the assigned SWC Runnables to → provide data
→ Recive data



Now only R1 can provide the temperature and R3 can read the temp

```
#Include "Rte_swc2:h"
void R3 (void)
{ :        "Read" Access point"
state =Rte_Read_port_data( *data);
   :
}
Void R4 (void)
{ ... }
```

```
# Include "Rte_swc1.h"
void R1 (void)
{ :      "write Access point"
Rte_write_port_data (x);
   :
}
Void R2 (void){...}
```

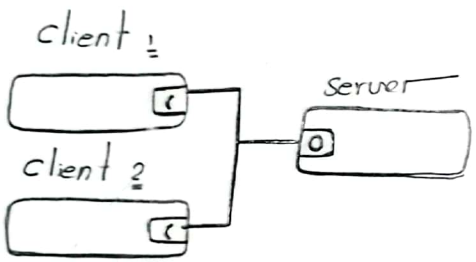
```
Rte.c file

static uint8 temp =0;

Rte_write_port_data (uint8 val)
{ suspendAll_Interrupts ();//protection
   temp = val;
  ResumeAll_Interrupts ();//protoction
}
Rte_Read_port_data (uint8 *val)
{ suspendAll_Interrpts ();// protection
   *val = temp;
  ResumeAll_Interrupts ();// protection
```

provide stable value for the same data element for all recivers during one task

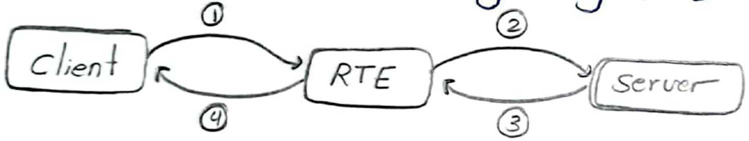
- Implicit Receiving : Data Element has two buffer ⟨ used
  updated
- Explicit Receiving : " " " one buffer

→ Client Server:

- Service Exchange
  - client : user (required)
  - server : provider
- (n) clients : (1) server



→ Service Invocation Handling through RTE:



1. Invoke request with arguments (client → RTE)
2. handling the client request (RTE → server)
3. returning results from (server → RTE)
4. RTE Invoke the request result (RTE → client)

↳ RTE role:

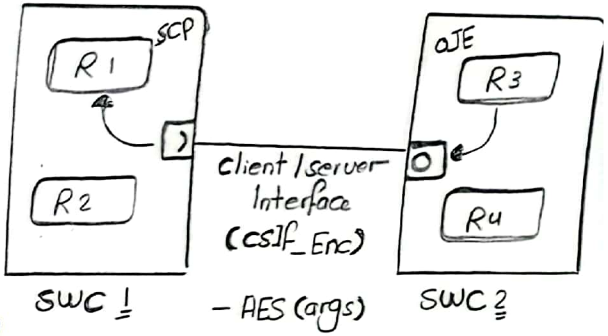
- Buffering for multiple requests for different clients
- Protection for arguments, results
- server Invokation can be synchronous/Asynchronous

↳ Access points:

- client runnables which will request the operation
  ↳ shall be defined through "server call point"
- server runnables which will Implement the operation
  ↳ shall be defined through "operation Invoked Events"

↳ Queue length: "server queue"

: define num of requests from client, to server can be buffered



SWC1.c

```
#Include "Rte_swc1.h"
Void R1 (void)
{ :
state= Rte_Call_rpEncry_AES (args);
  :
}
void R2(void) {..}
```

SWC2.c

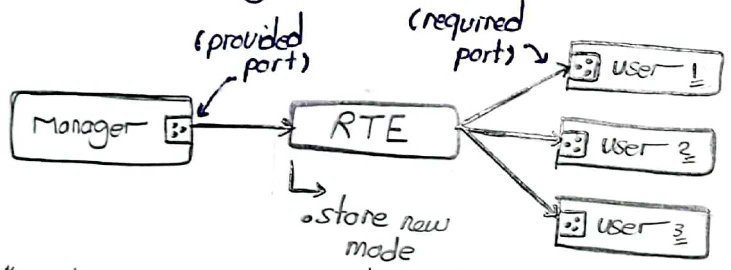
```
#Include "Rte_swc2.h"

Void R3 (Args)
{ "Implementation of operation"
}

Void R4 (void) {...}
```

Rte.c

```
std_returntype
        Rte_call-rpEncry_AES (args)
{       :
        R3 (args);
        :
        :
}
```

→ Mode switch:

- Control state or mode machine
- (1) mode manager : (n) user mode



↳ store new mode
↳ preform action based on new mode

"Mode Manager"
↳ : responsible to switch to new mode.

- Manager mode provide states to user mode

List of mode ← Mode Declaration Group

↳ Access points:

- at mode manager : mode manager Runnable has mode switch access point
- at mode user : Action Runnable triggered by mode switch event
  - ↳ on entry action
  - ↳ on exist from action
  - ↳ on transition

Every user preform action based on new mode, RTE action

MoIf_FanMode
- INIT
- AUTO
- MANUAL

↳ At manager:
- define MSP Mode switch point

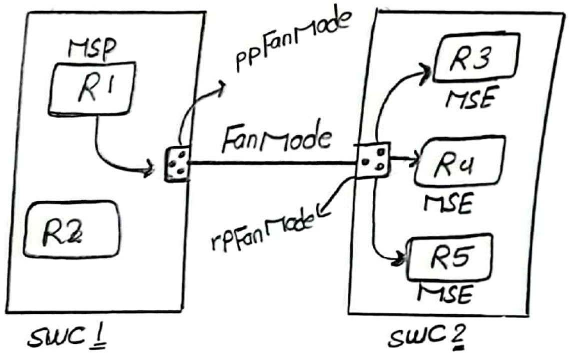
↳ At user:
- define MSE Mode switch Event

on Entry
on Exit
on Transitions

→ Mode Disabling Dependancy:

- for each Rte Event we can define mode disabling dependancies.

↳ Event to:
- disabling some periodic ~~Tasks~~ (runables)
- disabling data reception



SWC1 .C

```
#Include "Rte_swc1.h"
void R1 (void)
{
    FanMode = RTE_MODE_FAN_INIT;
 Rte_switch_ppFanMode_FanMode
            (FanMode);
 }
void R2(void) {     }
```

RTE.C

```
FanModcurrent = RTE_FANMODE_INIT;
FanModeNext = RTE_FANMODE_INIT;

Rte_switch_ppFanMode_FanMode
     (FanMode)
{
   FanModeNext = FanMode;
    set_Event (FanMode Events);
}

 Task (void)
 { waitEvent (FanMode Events);
    switch (FanModecurrent)
    { case c :    R3();
       case  :    R4();
       case  :     R5();
```