Lecture 9 - Hardware Proxy Driver Pattern

1. Hardware Proxy Drive Pattern

- It's a pattern that creates a software element (driver) responsible for the implementation and actions of a hardware element. Usually used for the files in the MCAL and HAL layers.
- The hardware elements can be pieces of memory, ports (DIO), interrupt mapped (UART, SPI), an external device (temperature sensor, motor).
- This pattern provides:
  - minimizing the changes in the hardware drivers.
  - easing the modification if needed in the drivers.
  - making sure that the clients don't worry about the actions taken by the hardware such as encryption, encoding, compression.

2. Features Supported by Proxy Pattern

2.1. `access()` or `read()`

- This is a public (can be accessed from other files) function(s) that returns a particular value from the hardware.
- This value has some kind of meaning to the client using the feature (function) to use it in its logic.
- E.g., the temperature of the temperature sensor, the speed of the motor, and the read APIs of the AUTOSAR DIO.
  - `u8 DIO_ReadChannel(u8 channel);`
  - `u8 DIO_ReadPort(u8 port);`
  - `u8 DIO_ReadChannelGroup(u8 *channel_group);`

2.2. `mutate()` or `write()`

- This is a public function(s) that set (write) (change) a state of the hardware.
- This function must have at least a single parameter, which is the value to be written to the hardware.
- E.g., set the speed of the motor, set the value in the ADC, and the write APIs of the AUTOSAR DIO.
  - `void DIO_WriteChannel(u8 channel);`

2.3. `configure()` or `init()`

- This is a public function that provides an interface used by clients to initialize or configure inital states for the hardware.
- The configuration may be set through the API in different ways:
  - post-build configurations: a struct or array (or address of any of them) is passed throught the `init()` API in which the client when calling the service pass the configurations in a struct. e.g.

```
/*
This is a struct data sturcture that has members
that defines the state of the GPIO ports such as
 its id,
 the direction of each pin,
 the pull up or pull down resistor state,
 and the maximum current going of the pins of this port. */
typedef struct{
    u8 portID;
    u8 portMask;
    u8 portDirection;
    u8 pullUp;
    u8 pullDown;
    u8 Use2mACrt;
    u8 Use4mACrt;
} port_configType;

void DIO_initPort(const port_configType* config_ptr);
```

2.4. `marshal()` and `unmarshal()`

- These are public functions that execute actions that don't include configuring initial states, reading data from the hardware, writiting states to the hardware.
- Usually these for the `marshal()` actions include encryption, bit-packing, compression, encoding that may be required for the device. And the opposite for the `unmarshal()`, i.e. decryption, bit-unpacking, decompression, decoding in which ensures that the extra information needed for the device is hidden from the client.

3. Car Motor Example

We need to design a proxy driver for the motor (assume DC).

3.1. `init()`

```
/* PSEUDO CODE */
void DCMotor_init(void)
{
    state = DISABLED;
    speed = 25;
    direection = CW; /* clock-wise*/
}
```

OR

```
 /* PSEUDO CODE */
typedef struct{
    u8 cfg_state;
    u8 cfg_speed;
    u8 cfg_direction;
} DCMotor_ConfigType;

void DCMotor_init(DCMotor_ConfigType *dcmotor_cfg)
{
    state = dcmotor_cfg -> cfg_state;
    speed = dcmotor_cfg -> cfg_speed;
    direection = dcmotor_cfg -> cfg_direction;
}
```

3.2. `write()`

3.2.1. `WriteSpeed()`

```
void DCMotor_WriteSpeed(u8 set_speed)
{
  speed = set_speed;
}
```

3.2.2. `WriteDirection()`

```
void DCMotor_WriteDirection(u8 set_direction)
{
  direction = set_direction;
}
```

3.3. `read()`

3.3.1. `ReadSpeed()`

```
u8 DCMotor_ReadSpeed(void)
{
  return speed;
}
```

3.3.2. `ReadDirection()`

```
u8 DCMotor_ReadDirection(void)
{
  return direction;
}
```

3.4. `marshal()`

3.4.1. `enable()`

```
void DCMotor_Enable(void)
{
  state = ENABLED;
}
```

3.5. `unmarshal()`

3.5.1. `disable()`

```
void DCMotor_Disable(void)
{
  state = DISABLED;
}
```

If we used a servo motor, more parameters like the angle, the gradient angle will be added in the same way...