How to Write a Classic AUTOSAR Application with RTE and COM: A Simple Example Code

Published in Level Up Coding
6 min read · Apr 8

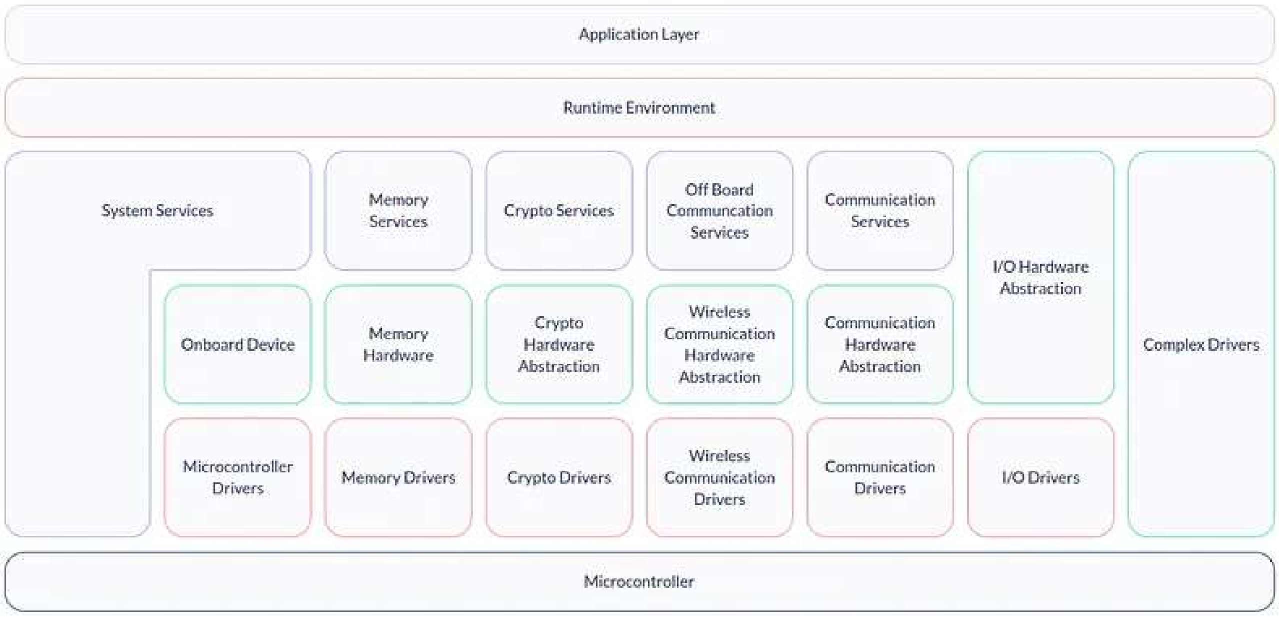
https://www.autosar.org/standards/classic-platform

Classic AUTOSAR (AUTomotive Open System ARchitecture) is a software architecture framework that is widely used in the automotive industry to develop complex, safety-critical software systems.

The goal of Classic AUTOSAR is to provide a standardized framework for developing automotive software that is modular, scalable, and portable across different hardware platforms.

RTE (Run-Time Environment) Layer

At its core, Classic AUTOSAR is based on a layered architecture that consists of three main layers: the Application Layer, the RTE (Run-Time Environment) Layer, and the BSW (Basic Software) Layer.



https://www.autosar.org/standards/classic-platform

The Application Layer contains the software components that implement the vehicle functions, while the RTE Layer provides a standardized interface between the Application Layer and the BSW Layer.

The BSW Layer provides low-level software services that are required by the Application Layer, such as device drivers and communication protocols.

One of the key concepts of Classic AUTOSAR is Component-Based Software Engineering (CBSE). In the context of Classic AUTOSAR, CBSE is used to design and implement software components that can be easily integrated into larger systems. Components are designed to be modular and self-contained, with well-defined interfaces that allow them to be easily combined with other components.

To illustrate the concepts of Classic AUTOSAR, let’s take a look at an example code of a simple application that controls the speed of a motor.

This example uses the RTE layer to provide a standardized interface between the application code and the BSW layer.

```
#include "Rte_SpeedControl.h"

int main(void)
{
    /* Initialize the RTE */
    Rte_Start();
```

```
        /* Set the speed to 50 km/h */
        Rte_Write_speed(50);

        /* Wait for 5 seconds */
        delay(5000);

        /* Set the speed to 0 km/h */
        Rte_Write_speed(0);

        /* Shutdown the RTE */
        Rte_Stop();

        return 0;
    }
```

In this example, the main function initializes the RTE layer, sets the speed of the motor to 50 km/h using the Rte_Write_speed function, waits for 5 seconds using the delay function, sets the speed of the motor to 0 km/h using Rte_Write_speed again, and finally shuts down the RTE layer.

The Rte_SpeedControl.h header file defines the interface between the application code and the RTE layer.

```
#ifndef RTE_SPEEDCONTROL_H
#define RTE_SPEEDCONTROL_H

/* Prototypes for RTE functions */
void Rte_Start(void);
void Rte_Stop(void);
void Rte_Write_speed(uint16_t speed);

#endif /* RTE_SPEEDCONTROL_H */
```

This header file declares the Rte_Start, Rte_Stop, and Rte_Write_speed functions, which are used by the application code to interact with the RTE layer.

Communication

Next, let's talk about communication infrastructure between different components in classic Autosar.

In Classic AUTOSAR, The communication layer(COM) provides a set of standardized communication mechanisms that can be used to send and receive messages between components, regardless of their physical location within the system.

To illustrate how COM is used in Classic AUTOSAR, let's take a look at an example of an application that consists of two software components: a motor control component and a user interface component. The motor control component is responsible for controlling the speed of the motor, while the user interface component is responsible for displaying information to the user and accepting input from the user.

To enable communication between these two components, we can define a set of messages that can be sent between the components using COM.

For example, we could define a "set speed" message that the user interface component can send to the motor control component to set the speed of the motor.

To use COM, we need to define a set of communication interfaces that describe the messages that can be sent and received between components.

In Autosar, AUTOSAR XML is used as a standard format for exchanging and storing Autosar models and configurations, including interface descriptions.

In this example, the simplified version of the arxml file would look like this,

```
          <SHORT-NAME>UserInterfacePort</SHORT-NAME>
          <PROVIDED-INTERFACE-TREF DEST="P-PORT-PROTOTYPE">/Interface/MotorCont
        </P-PORT-PROTOTYPE>
      </PORTS>
    </APPLICATION-SOFTWARE-COMPONENT-TYPE>
  </ELEMENTS>
  <ELEMENTS>
    <INTERFACE>
      <SHORT-NAME>MotorControlInterface</SHORT-NAME>
      <OPERATIONS>
        <CLIENT-SERVER-OPERATION>
          <SHORT-NAME>set_speed</SHORT-NAME>
          <DIRECTION>CLIENT-TO-SERVER</DIRECTION>
          <ARGUMENTS>
            <ARGUMENT>
              <SHORT-NAME>speed</SHORT-NAME>
              <TYPE-TREF DEST="ARGUMENT">/DataType/uint16</TYPE-TREF>
            </ARGUMENT>
          </ARGUMENTS>
        </CLIENT-SERVER-OPERATION>
        <CLIENT-SERVER-OPERATION>
          <SHORT-NAME>get_speed</SHORT-NAME>
          <DIRECTION>SERVER-TO-CLIENT</DIRECTION>
          <ARGUMENTS>
            <ARGUMENT>
              <SHORT-NAME>speed</SHORT-NAME>
              <TYPE-TREF DEST="ARGUMENT">/DataType/uint16</TYPE-TREF>
            </ARGUMENT>
          </ARGUMENTS>
        </CLIENT-SERVER-OPERATION>
      </OPERATIONS>
    </INTERFACE>
  </ELEMENTS>
</AR-PACKAGE>
```

If we translate it to simpler IDL just to make it more understandable what it is defining, it looks like this,

```
interface MotorControlInterface {
    void set_speed(uint16_t speed);
    uint16_t get_speed();
}
```

The ARXML file contains the interface definitions for MotorControlInterface and UserInterfaceInterface. These definitions specify the operations that can be called on each interface and the data types that are used as parameters and return values.

The ARXMl specifies an interface called MotorControlInterface with two methods: set_speed and get_speed. The set_speed method takes a uint16_t argument called speed, and it does not return any value. The purpose of this method is to set the speed of the motor. The get_speed method returns a uint16_t value, and it does not take any arguments. The purpose of this method is to retrieve the current speed of the motor. This interface can be used to define the communication between the Motor Control Application and other software components that need to interact with it.

To use these interfaces in our application code, we need to generate code from the ARXML file that specifies the interfaces and provides the necessary communication infrastructure. This code can then be integrated into the software components to enable communication between them.

When the ARXML file is processed by the AUTOSAR tools, it generates code that implements the interfaces and provides functions for calling the operations on the interfaces.

Here's an example of the generated codes and how we could use these interfaces in our applications:

Motor Controller Application:

```
        /* Loop indefinitely */
        while(1)
        {
            /* Wait for the speed to be set by the User Interface App */
            uint16_t speed = MotorControlInterface_get_speed(&motor_control);

            /* Set the motor speed */
            set_speed(speed);
        }
    }

    /* Initialize the Motor Control App */
    void initialize_motor_control(void)
    {
        /* TODO: Implement initialization code for the Motor Control App */
    }

    /* Set the motor speed */
    void set_speed(uint16_t speed)
    {
        /* TODO: Implement code to set the motor speed */
    }
```

User Interface Application:

```
            speed = new_speed;

        }
    }

    /* Wait for the user to update the speed */
    uint16_t wait_for_speed_update(void)
    {
        /* TODO: Implement code to wait for user input and return the new speed */
    }
```

MotorControlInterface_get_speed is a function generated from the ARXML description of the MotorControlInterface. This function is used to retrieve the current speed set by the User Interface App via the MotorControlInterface interface.

MotorControlInterface_set_speed function is generated from the ARXML file too. This function takes in a new speed value as an argument and updates the speed in the Motor Control app using the interface from User Interface Application.

Summary

This article provided a brief overview of Classic AUTOSAR and explained the general idea of developing Classic AUTOSAR applications with RTE and COM using simple example codes. Classic AUTOSAR is a layered software architecture framework that employs Component-Based Software Engineering to design and implement modular, self-contained software components with well-defined interfaces. The article highlighted the importance of the RTE Layer as a standardized interface between the Application Layer and the BSW Layer, and how the BSW Layer provides low-level software services required by the Application Layer. By employing these concepts, Classic AUTOSAR enables the development of complex, safety-critical automotive software systems that are modular, scalable, and portable across different hardware platforms.

Embedded Systems Autosar Automotive C Cplusplus