embeddedworld2022
Exhibition&Conference
... it's a smarter world

Develop and Integrate AUTOSAR Classic and Adaptive Applications Based on SOME/IP

Shwetha Bhadravathi Patil
Technical Product Marketing
MathWorks
Natick, MA, United States
shwethap@mathworks.com

Aastha Kanwar
Application Engineering
MathWorks GmbH
Ismaning, Germany
akanwar@mathworks.com

Roy Park
Consulting Services
MathWorks
Seoul, South Korea
roypark@mathworks.com

***Abstract*—**Enormous changes are underway in the automotive software ecosystem. Driven in part by demanding autonomous driving (AD) applications and advanced driver-assistance systems (ADAS), automotive software now requires more frequent updates, higher integration flexibility, and greater computing power. As the AUTOSAR standard describes, Classic and Adaptive Platforms complement each other to serve the demands of modern automotive applications. However, little has been published to guide the development of complete integrated systems with AUTOSAR Classic and Adaptive applications. This paper describes a Model-Based Design approach to develop heterogenous software applications that communicate using SOME/IP (Scalable service-Oriented MiddlewarE over IP).

I. Introduction

The automotive software ecosystem has been undergoing a transformation in recent years. The changes are driven by the increase in demanding ADAS and highly automated driving applications, connected mobility, and a clear trend to transition to software-defined vehicles that use high-performance computers and centralized architectures. The next generation of automotive software will rely on high rates of data transfer and high computing power to support smart applications that process large quantities of sensor data, including point clouds, images, and video data. Off-loading demanding applications to cloud services is another key capability, which in turn requires secure, reliable communication with low latency. Higher integration flexibility and over-the-air updates enable customers to download new applications, software updates, and bug fixes on-the-fly. The challenge is to provide these new capabilities while reducing cost and complexity, following efficient software development workflows.

The AUTOSAR (AUTomotive Open System ARchitecture) organization provides standard platforms to support current and future generations of automotive electronic control units (ECUs). AUTOSAR provides the Classic Platform, used for safety-critical and hard real-time applications on cost-optimized microcontrollers, such as the powertrain and chassis. To address the needs of automated driving, Vehicle-to-X, and infotainment applications, the AUTOSAR organization introduced the AUTOSAR Adaptive Platform in 2017. The Adaptive Platform uses a service-oriented architecture that works on the principle of service providers and service consumers, which provides an efficient alternative to classical signal-based communication. The Adaptive Platform is designed to support high-performance processors (> 20.000 DMIPS) and fast Ethernet-based communication (100 Mbps -1 Gbps), in addition to capabilities such as over-the-air updates and dynamic application deployment. As the AUTOSAR standard describes, the Classic and Adaptive Platforms complement each other to serve the demands of modern automotive applications [1].

This paper describes an approach based on SOME/IP events and model-based design for system development and testing of communications among heterogenous ECU software applications in a PC environment. The approach has been validated in a proof of concept comprising three parts, detailed in Section IV:

1. Development of an AUTOSAR Classic application based on SOME/IP events
2. Development of an AUTOSAR Adaptive application based on SOME/IP events
3. Testing communication between the AUTOSAR Classic Server and Adaptive Client

II. Service-Oriented Communication in AUTOSAR

The AUTOSAR Adaptive Platform supports SOME/IP and data distribution service (DDS) for service-oriented communication between AUTOSAR Adaptive, AUTOSAR Classic, ROS, and other applications. Both SOME/IP and DDS

are middleware solutions that allow distributed applications to communicate using the publish-subscribe pattern. This paper describes the use of SOME/IP to communicate between Classic and Adaptive applications. [1]

A SOME/IP protocol has been specifically developed for automotive software for service-oriented communication over a network. The services are provided by servers, and a client can dynamically find the available services via service discovery and subscribe to them. Clients and servers can be distributed across different ECUs [2].

A service consists of zero to multiple combinations of methods, events, and fields. Figure 1 illustrates the concepts of methods, events, and fields for client-server communication. Methods allow the subscriber to issue remote procedure calls that are executed on the provider side. In the request/response method, a client can request asynchronous or synchronous calls to the server, and the server responds with an acknowledgment or an error message. SOME/IP also defines a fire/forget method, where a client makes a request to the server but does not receive any response from the server. If the client does not desire to request data from the server each time, events could be a better fit because events allow the client to subscribe to a service by sending a one-time subscription request. The server sends the subsequent data changes periodically to the subscribed clients, without additional requests [1].

Finally, fields may be interpreted as a combination of the method and event features. A field is a combination of one or more of the following:

a) A notifier that sends data on a change from the provider to the subscriber
b) A getter that can be called by the subscriber to explicitly query the provider for the value
c) A setter that can be called by the subscriber when it wants to change the value on the provider side

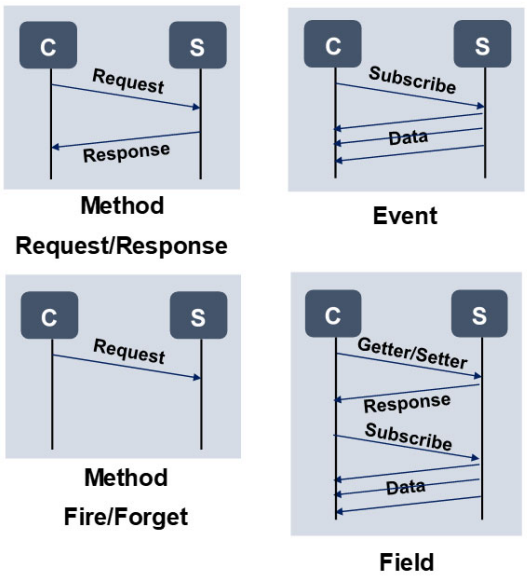

Figure 1 Methods, events, and fields in SOME/IP protocol

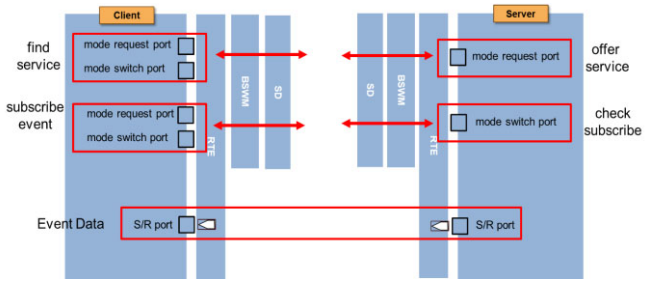

Figure 2 Handling of SOME/IP events in AUTOSAR Classic applications

III. Development of AUTOSAR Software Based on SOME/IP Events

The service-oriented event features in AUTOSAR are implemented in different ways in the Classic and Adaptive applications.

A. Handling of SOME/IP Events in AUTOSAR Classic

In Classic, services related to finding and offering services and event registration are handled by the Service Discovery (SD) module, the Basic Software Manager (BSWM), and the Run-Time Environment (RTE). As illustrated in Figure 2, separate mode request and switch ports are required for the find service/offer service and the event subscribe/check subscribe tasks at client and server software components (SWCs). [1]

Figure 2 shows the required port configuration and data exchange of a single event between a client and a server. In the client model, a mode request port requests *find service* and a mode switch port receives feedback on whether there is a service. Additionally, the client SWC needs another mode request port to request event registration to the server and a mode switch port to receive feedback on whether event service is possible from the actual server.

The server SWC has complementary ports. A mode request port offers service to the client and a mode switch port receives feedback on whether there is an event registration request from the actual client. Finally, the actual event data is exchanged via additional sender/receiver (S/R) ports.

B. Modelling of SOME/IP-Based Classic Applications using Model-Based Design

Model-based design is a well-established development methodology for automotive embedded software. In the previous section, we discussed the required communication ports for event data exchange between a client and a server. Figure 3 shows the implementation of this use-case for Classic client/server SWCs using model-based design. A comparison of Figure 2 and Figure 3 highlights the one-to-one mapping between the AUTOSAR concepts and

corresponding implementations using Model-Based Design. The ports have been configured as mode request, mode switch, and sender/receiver (S/R) ports, as in Figure 2. The algorithm has been implemented within the subsystems. At this stage, simulation can be employed to ensure the correct functionality of the modeled algorithms. After the design has been sufficiently verified, the application code can be automatically generated. Figure 4 shows the RTE function calls generated for each port in the application code for the client. [3]

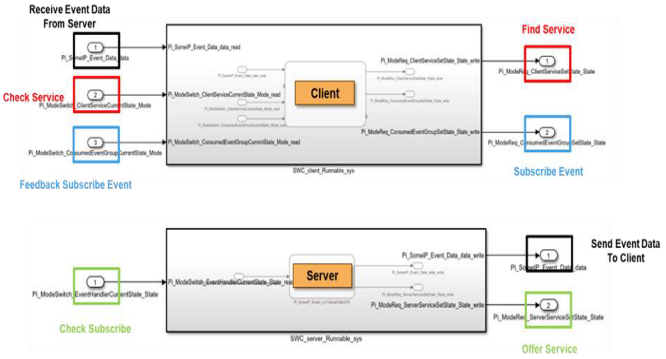

Figure 3 Modelling of event data exchange for AUTOSAR Classic client/server SWCs in Simulink

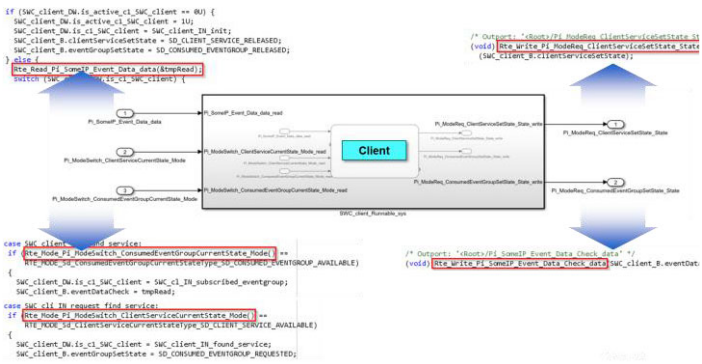

Figure 4 Application code for a Classic client

C. Handling of SOME/IP Events in AUTOSAR Adaptive

Unlike Classic, in the Adaptive applications, event-related tasks are handled in the AUTOSAR Runtime for Adaptive Applications by ara::com, a communication management layer that implements the service-oriented architecture pattern to achieve flexibility and scalability for distributed processing [6]. Therefore, the development of event-based applications is simplified compared to AUTOSAR Classic. Figure 5 illustrates the exchange of event data using event request/provide ports [1].

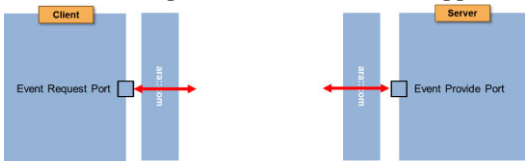

Figure 5 Handling of Events in AUTOSAR Adaptive Application

D. Event Modelling in Adaptive Applications using Model-Based Design

Event data exchange for Adaptive applications is modelled using message-based communication [5]. Figure 6 shows the modelling of a simple client/server mechanism for Adaptive applications. Figure 7 shows the corresponding application code for the adaptive client model. The generated code shows the handling of event registration and event data transmission and reception by ara::com [4].

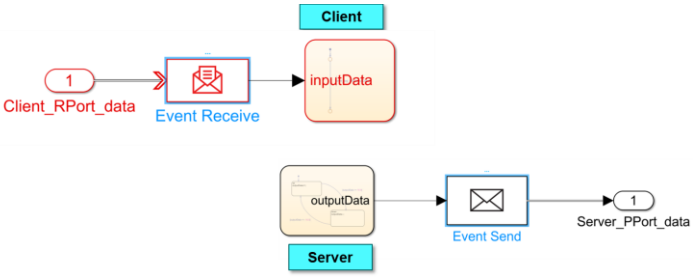

Figure 6 Modelling of event data exchange for Adaptive client/server

```
ara::com::ServiceHandleContainer< SimpleEvent::proxy::IncrementProxy::
  HandleType > handles;
handles = SimpleEvent::proxy::IncrementProxy::FindService(ara::com::
  InstanceIdentifier("1"));
if (handles.size() > 0U) {
  Client_RPort = std::make_shared< SimpleEvent::proxy::IncrementProxy >
    (*handles.begin());

  // Subscribe event
  Client_RPort->data.Subscribe(ara::com::EventCacheUpdatePolicy::kNewestN,
    1U);
}
// Fetch data for event "data" from ARA middleware
if (Client_RPort && Client_RPort->data.Update()) {
  // Access event data
  data_SmplCont = &Client_RPort->data.GetCachedSamples();

  // Copy event data to application
  Client_DW.dataIn_msgData = **data_SmplCont->begin();
  Client_DW.dataIn_msgDataPtr = &Client_DW.dataIn_msgData;

  // Received new event data
  status = 0;

  // Explicitly clean the event data cache
  Client_RPort->data.Cleanup();
}
```

Figure 7 Application code for Adaptive client

IV. Integration and Testing of Generated Code

As discussed in Section I, AUTOSAR Classic and Adaptive Platforms address different components of automotive software. Because of the complementary nature of AUTOSAR Classic and Adaptive Platforms, there is a strong need for efficient methodologies to integrate and test Classic and Adaptive applications. To demonstrate the suitability of the model-based design approach, a simple proof-of-concept was conducted. The application code generated from the models was integrated with a Classic and Adaptive stack. Finally, the communication, based on SOME/IP, was established using an Ethernet connection. This section describes the setup, development steps, and results.

The setup consisted of a Classic ECU, which provided simple event data as a server via SOME/IP. The server planned to service triangle wave data with a boundary of ±10. An Adaptive ECU served as a single client to the server, as shown in Figure 8.

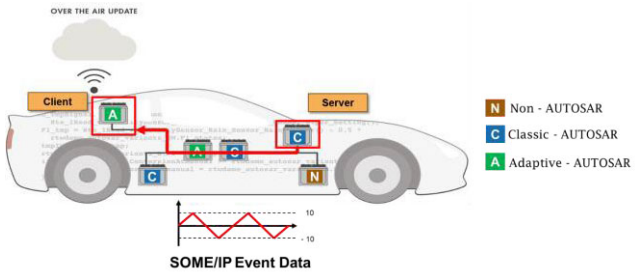

Figure 8 Classic ECU (server) and Adaptive ECU (client) for the proof-of-concept

This proof-of-concept consisted of three main phases as shown in Figure 9:

- Phase 1 - Development of AUTOSAR Classic ECU based on SOME/IP events
- Phase 2 - Development of AUTOSAR Adaptive ECU based on SOME/IP events
- Phase 3 – Testing the communication between AUTOSAR Classic server and Adaptive client

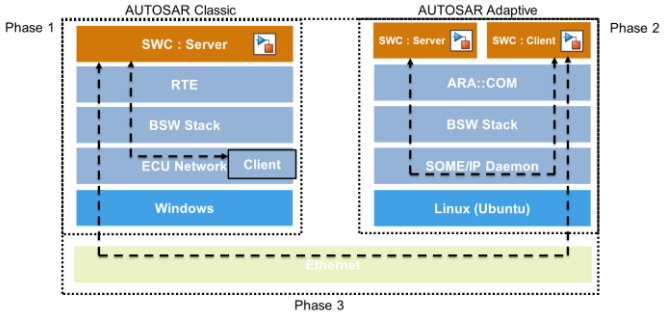

Figure 9 Development phases of the proof-of-concept

A. Phase 1: Development of AUTOSAR Classic Server application based on SOME/IP events

Phase 1 consisted of developing the AUTOSAR Classic server application using Simulink® from an ARXML file. After importing the ARXML file, a skeleton model was created, and the event data service behavior was implemented using AUTOSAR Blockset. Finally, application code was generated using Embedded Coder® as shown in Figure 10.

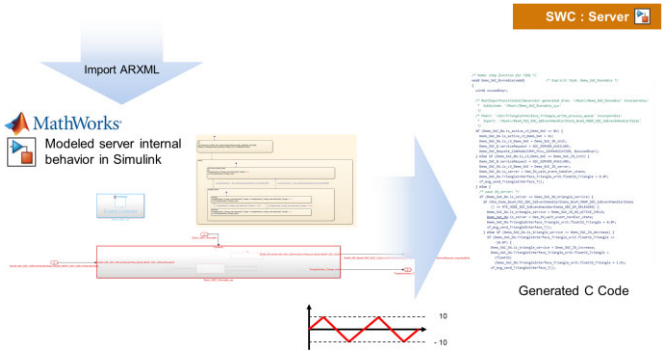

Figure 10 Development workflow of AUTOSAR Classic server application code

Next, the generated application code was integrated with a Basic Software stack to generate a .dll file. This .dll file was embedded on a virtual ECU network that operated as a node for testing. Finally, a separate virtual client node was created on an ECU network to validate the behavior of the server ECU.

B. Phase 2: Development of AUTOSAR Adaptive Server and Client Application Based on SOME/IP Events

Phase 2 consisted of developing the AUTOSAR Adaptive server and client application using Simulink from an ARXML file. After importing the ARXML file into Simulink, a skeleton model was created, and the implementation design of the event server/client behavior was completed using AUTOSAR Blockset. Finally, application code was generated using Embedded Coder as shown in Figure 11.

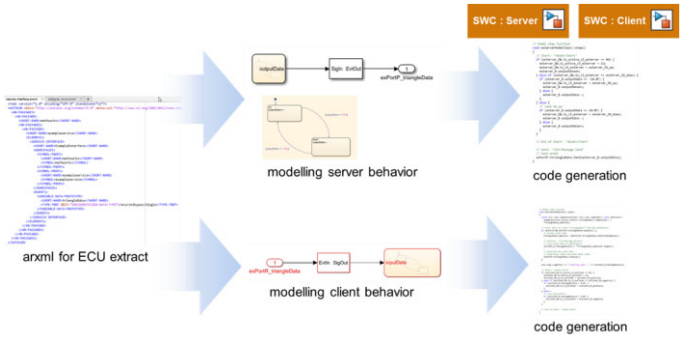

Figure 11 Development workflow of AUTOSAR Adaptive client and server application code

Next, the generated application code was integrated with the Adaptive Basic Software stack. Furthermore, a total of three

target files were created – one each for the server and the client, and a target file that played the role of ara::com, called the SOME/IP daemon. Then these three target files were run on Linux to verify that the developed client behaved as designed.

C. *Phase 3: Testing the Communication between AUTOSAR Classic Server and Adaptive Client*

Phase 3 tested the communication between the Classic server and the Adaptive client. On the Classic server side, a separate Ethernet device was allocated a unique IP address. Similarly, on the Adaptive client side, a separate Ethernet device was allocated a unique IP address with the client running on a LINUX virtual machine. These two devices were connected using an Ethernet cable as shown Figure 12.

To test the communication, the virtual client node on the ECU network was disabled, after having been previously created for the Classic server in Phase 1. Then, the ECU network was activated to run the Classic server, and the SOME/IP daemon was executed. At this stage, the SOME/IP daemon identified that the server was on an external node, and an Ethernet socket was created and assigned to the client, instead of using inter-process communication. The client can communicate with external nodes through this socket, and even with a change in the communication method, it is not necessary to change the application because the change is handled by the SOME/IP daemon. This is one of the advantages of ara::com. Finally, the Adaptive client node was executed to verify that the SOME/IP event data was successfully received from the Classic server over Ethernet.

Through this proof of concept, communication between the Classic server and the Adaptive client via event data using actual Ethernet communication was verified.

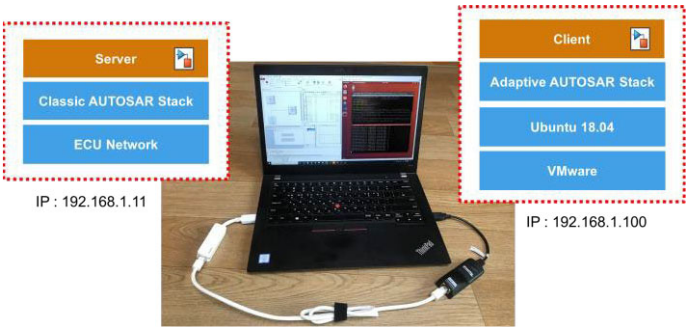

Figure 12 Setup to test the communication between AUTOSAR Classic server and Adaptive client

V. CONCLUSION

AUTOSAR Classic and Adaptive Platforms are expected to be key enablers for the software-defined autonomous vehicles of the future, handling both safety-critical real-time applications and demanding ADAS/AD, infotainment, and Vehicle-2-X applications of the future. The complementary nature of the two platforms demands methodologies that integrate and test Classic and Adaptive applications efficiently. This paper described how model-based design development methodologies can be used to develop AUTOSAR Classic and Adaptive applications based on SOME/IP events. The paper also demonstrated a workflow from ECU software development to integration with a third-party Basic Software stack, including testing of communication between an AUTOSAR Classic server and an Adaptive client application using real Ethernet communication.

REFERENCES

1. https://www.autosar.org/standards/
2. https://some-ip.com/
3. AUTOSAR Classic software component modeling https://www.mathworks.com/help/autosar/software-component-modeling.html?s_tid=CRUX_lftnav
4. AUTOSAR Adaptive software component modeling https://www.mathworks.com/help/autosar/adaptive-software-component-modeling.html?s_tid=CRUX_lftnav
5. Simulink Messages Overview https://www.mathworks.com/help/simulink/ug/simulink-messages-overview.html
6. Explanation of ara::com API https://www.autosar.org/fileadmin/user_upload/standards/adaptive/17-03/AUTOSAR_EXP_ARAComAPI.pdf

© 2022 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.