使用MBD进行AUTOSAR平台的SOA软件开发

长城汽车 – **杜建福 博士**

迈斯沃克 – 龚小平

2025-03

Content

- **Evolution of E/E & Software Architecture**
- **MathWorks SOA solution**
- **Practices of SOA Development**
- **Conclusion**

GWM Global R&D Layout

13
Regions

170+
Exporting countries and regions

14.9million
Global Cumulative Sales

长城Hi4技术体系
GWM Hi4 technology system

Hi4序列
Hi4 Series

Hi4

智能四驱电混架构
Intelligent 4WD Electric Hybrid System

性能和效能平衡
balance of performance and energy efficiency

Hi4性能版
Hi4 Performance Version

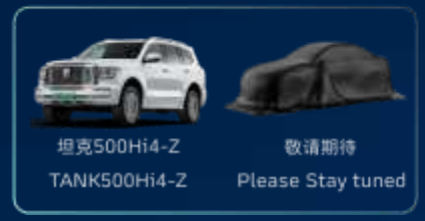
智能四驱电混架构
Intelligent 4WD Electric Hybrid System

极致动力，经济，NVH，安全
Ultimate Power, Economy, NVH, and Safety

Hi4-Z

泛越野超级电混架构
General Off-road Super Hybrid Architecture

超长续航，超高能效，超级性能，全面安全
Extended Range, Ultra-high Efficiency,
Superior Performance, and Comprehensive Safety



Hi4-T序列
Hi4-T Series

Hi4-T

越野超级混动架构
Off-road Super Hybrid Architecture

专注极致越野体验
Dedicated to Ultimate Off-road Experience

The Technological Great Wall – Smart

Coffee Pilot 智慧行驶
Intelligent driving system

Iteration
3

Vehicle Models
20
No. of car models equipped with this system exceeded 20

Mileage
亿km
Duration of data-driven intelligent learning 1 million hours

Coffee OS 智慧空间
Intelligent cockpit system

Start time
2.64秒

List Scroll Frame Rate
60FPS

App Start time
436ms

Voice wake up time
293ms

Map Scroll response time
153ms

Coffee Intelligence, as Great Wall Motors' (GWM) intelligent technology brand for complete vehicles, adheres to the concept of "user-centricism" and focuses on creating safer, more practical, and more cost-effective intelligent travel experiences for users. It provides comprehensive intelligent solutions for complete vehicles.
GWM's third-generation intelligent driving system, Coffee Pilot Ultra, is first integrated into the new Blue Mountain Intelligent Driving Edition, aiming to achieve national-scale capabilities of "driving anywhere roads exist" and "parking anywhere spaces are available" by the end of the year. As a cross-domain and cross-system technology, Service-Oriented Architecture (SOA) will undoubtedly accelerate the rapid implementation of intelligent vehicles.

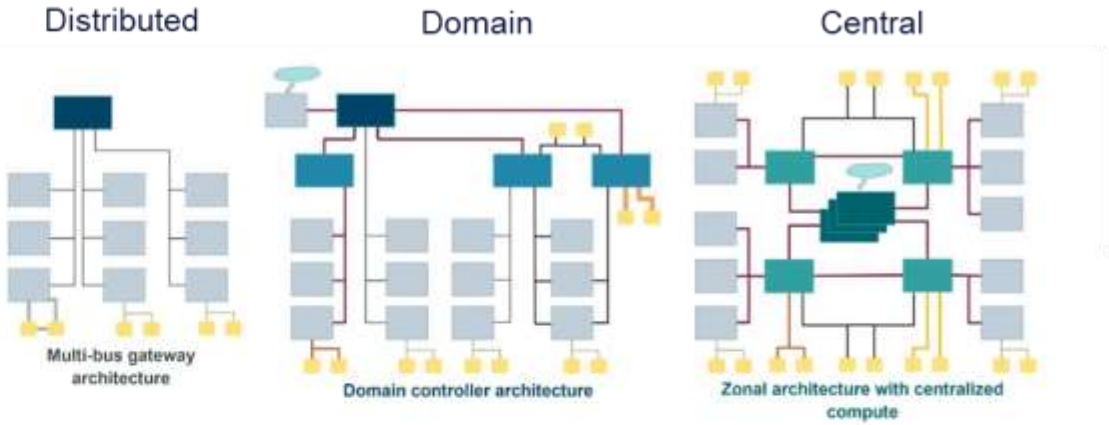
Content

- **Evolution of E/E & Software Architecture**
- MathWorks SOA solution
- Practices of SOA Development
- Conclusion

The Evolution of EE Architecture

Distributed ECU Architecture

- A single ECU control a single or multi functions
- A lot of ECUs(ranging from dozens to over 100)
- Wasted computing power
- Deep coupling of software and hardware
- Long wiring harnesses, high costs, and low automation levels in assembly
- OEMs are heavily reliant on suppliers



Source from web

Domain ECU Architecture

- Integrating multiple ECUs into three or five domain controllers including vehicle control domain(powertrain domain, chassis domain, body domain), cockpit domain, autonomous driving domain
- Reduction in the number of ECUs
- Concentration of functional software into multiple domain controllers

Central Computing Architecture

- Integration of domain controllers into in-vehicle computers
- Concentration of core functions into in-vehicle computers
- Concentration of auxiliary functions into zone controllers

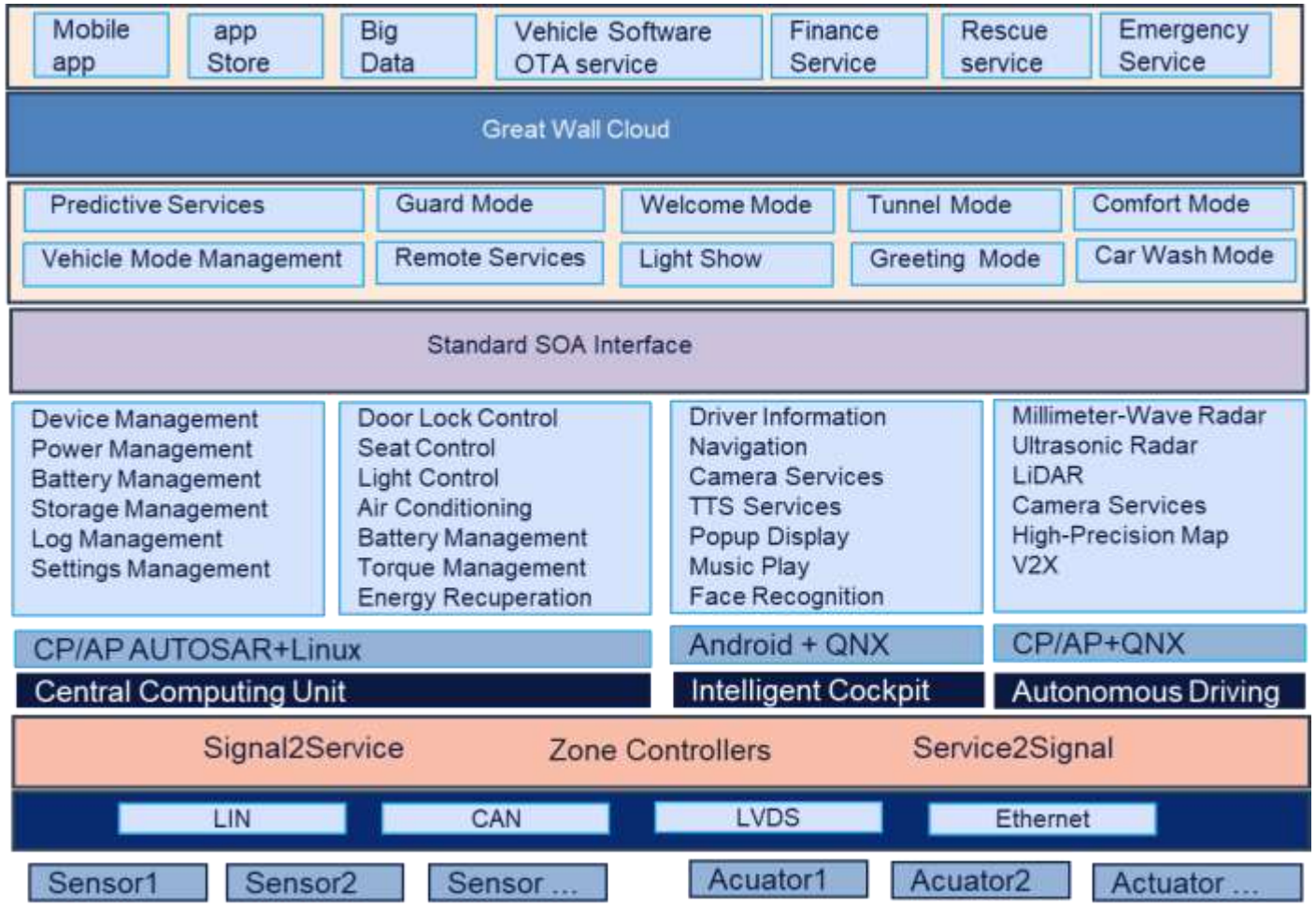
Extension of software architecture concepts

Software Architecture:
Single ECU

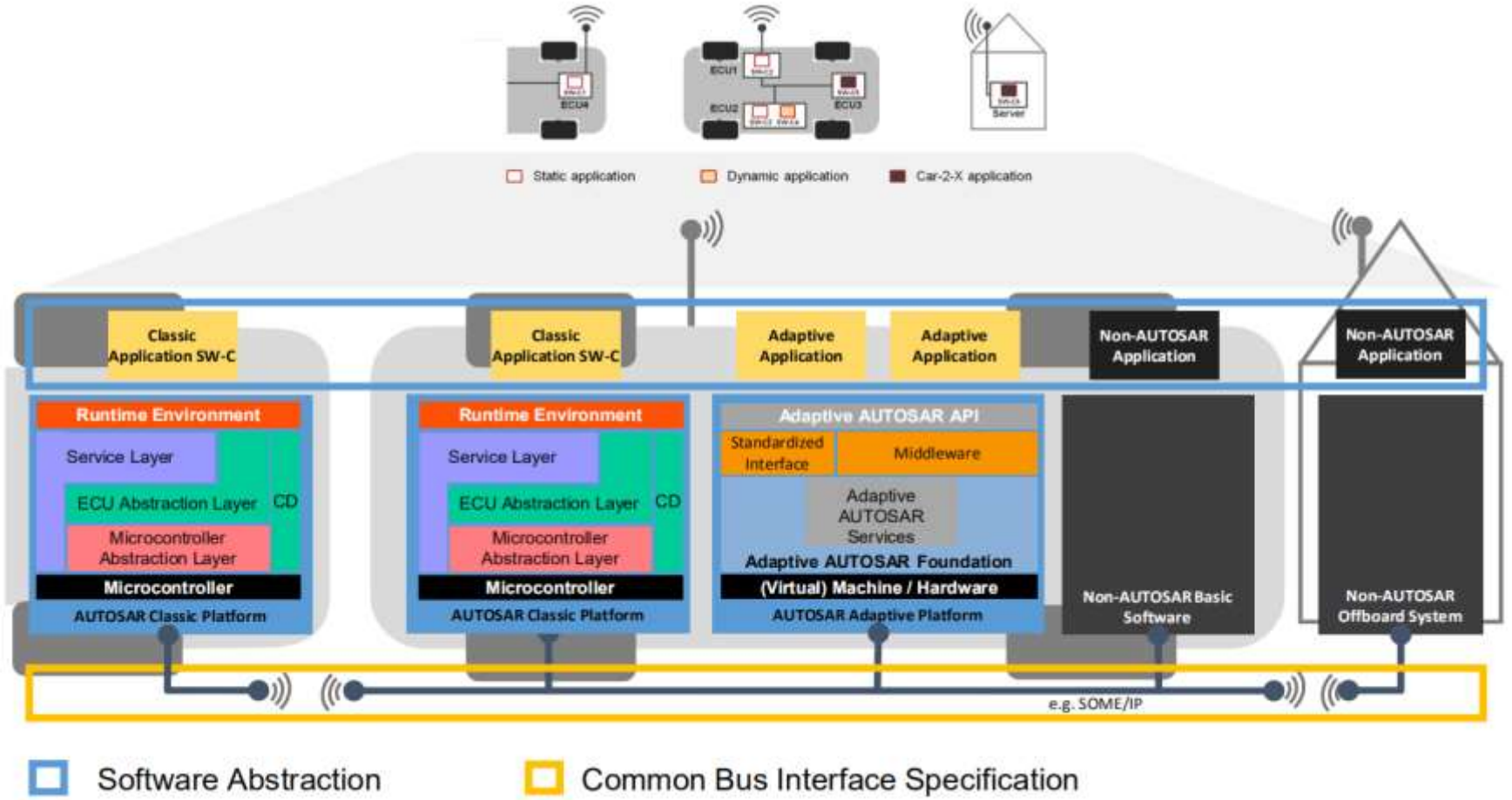
SOA

Software Architecture:

Vehicle-Cloud Integration



Future Automobile Software Architecture

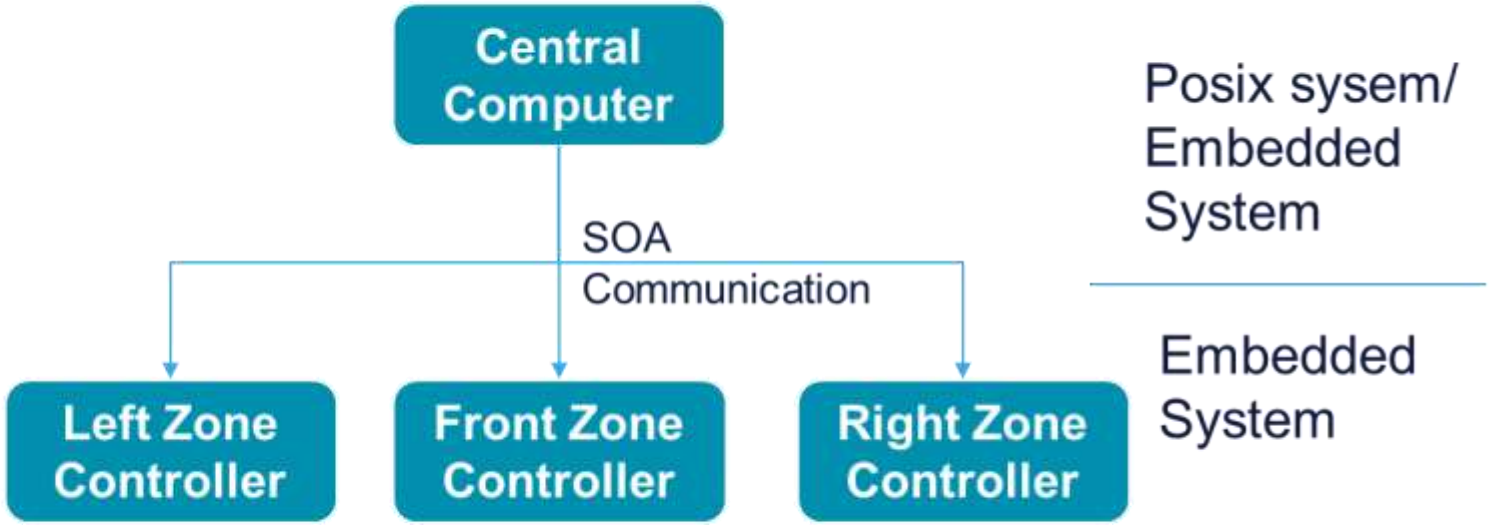


Source from 2016 AUTOSAR conference: *Ethernet and the AUTOSAR Adaptive Platform as basis for future E/E Architecture*

Future automotive software architecture will feature the coexistence of AP AUTOSAR , CP AUTOSAR and non-AUTOSAR.

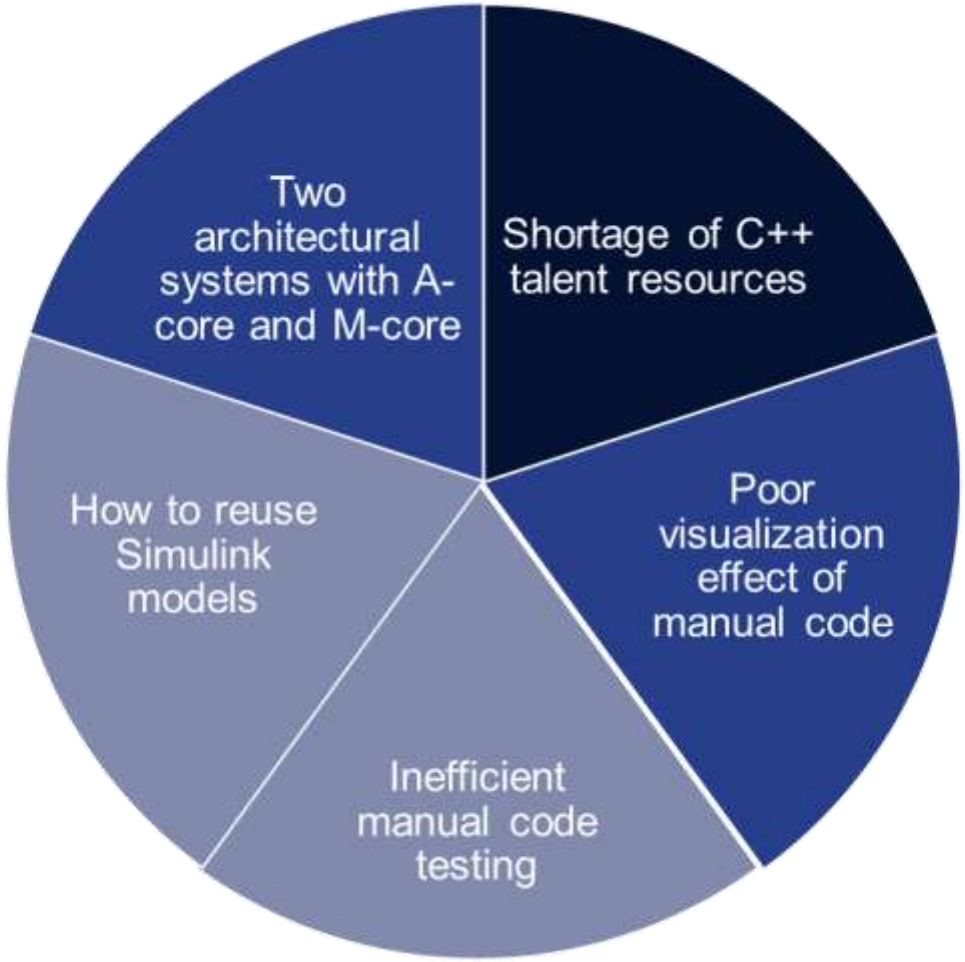
Challenges of SOA Development

SOA software need be developed in A core running Posix compatible system and also be developed in M core(R core) running classic RTOS.



Source: The ARM Processors: A, R, and M Categories and Their Specifics - Sirin Software

Challenges of SOA Software Development

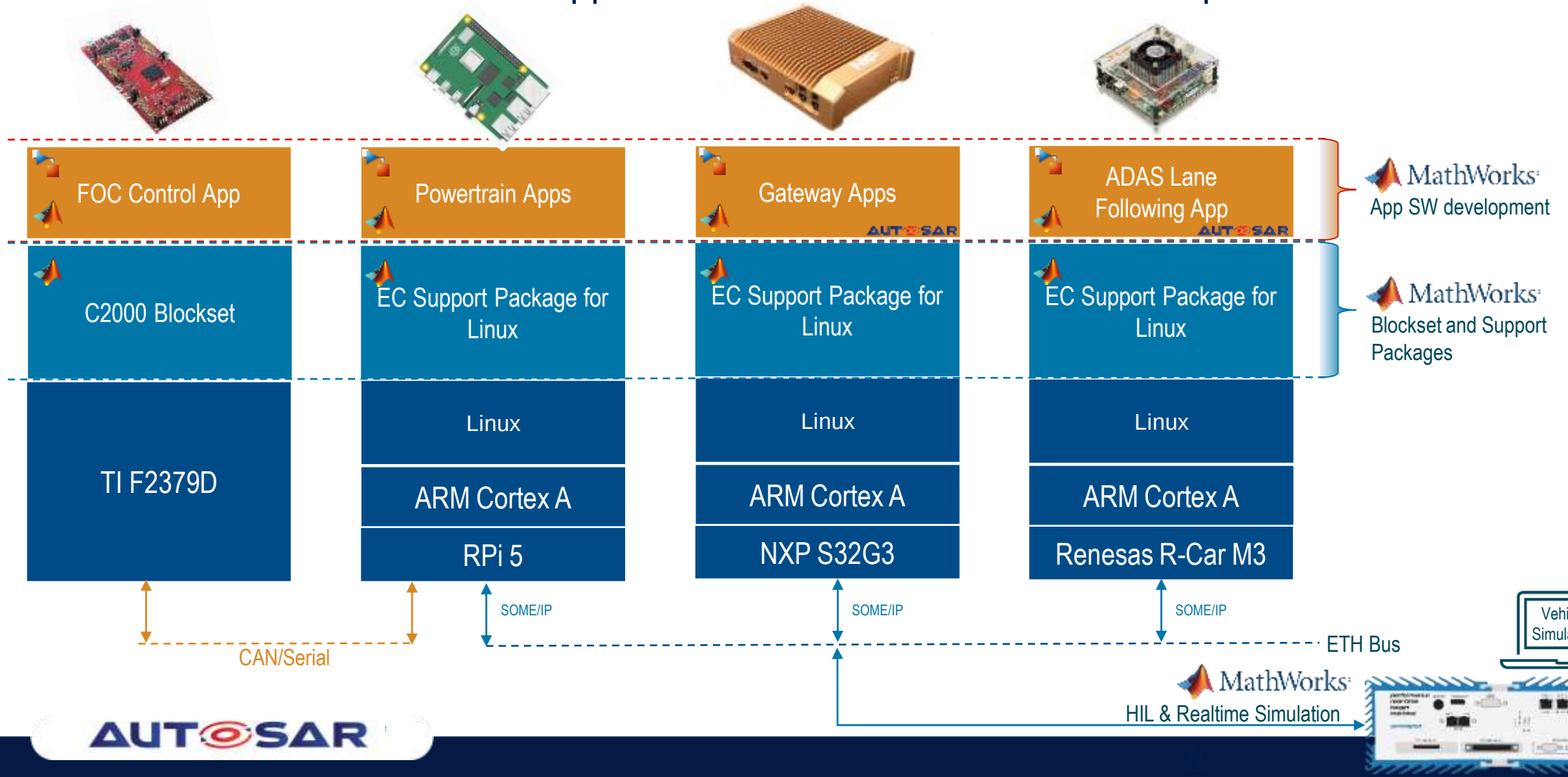


Content

- Evolution of E/E & Software Architecture
- **MathWorks SOA solution**
- Practices of SOA Development
- Conclusion

MathWorks Solutions

For Next Gen EE Architecture Apps - Domain, Zonal and Central Compute



AUTOSAR

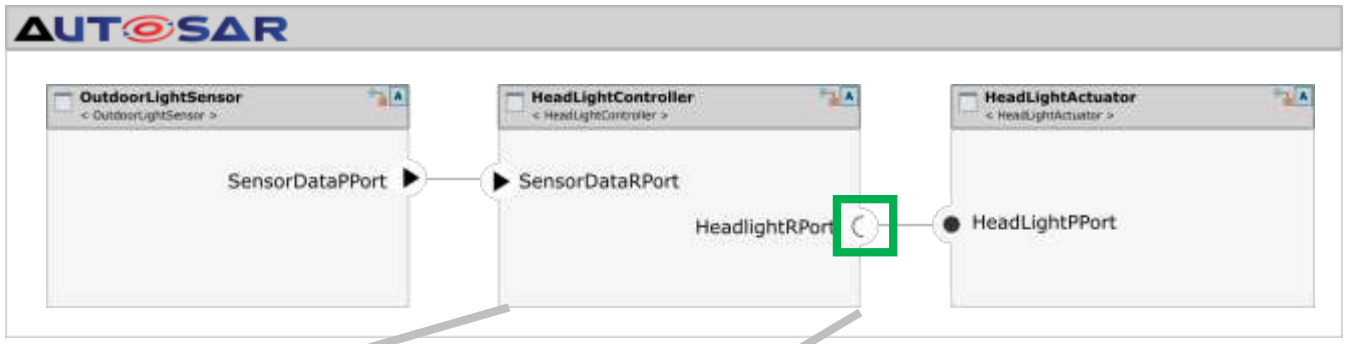
MathWorks Solutions

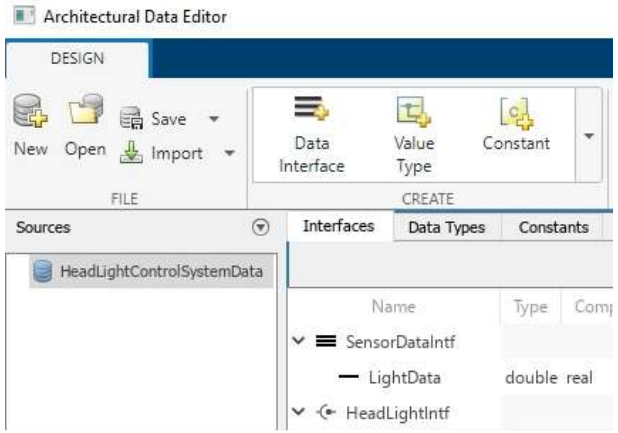
A holistic view for SOA development

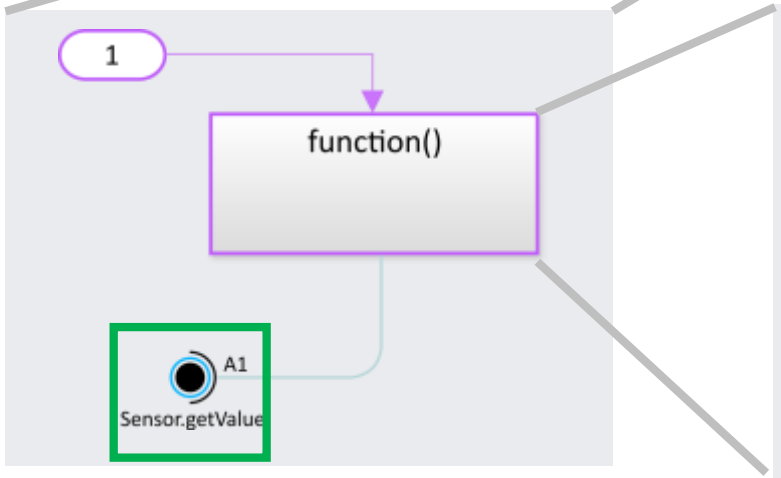
- From architecture to implementation
- From design to deployment
- From prototyping to production
- From physical to virtual

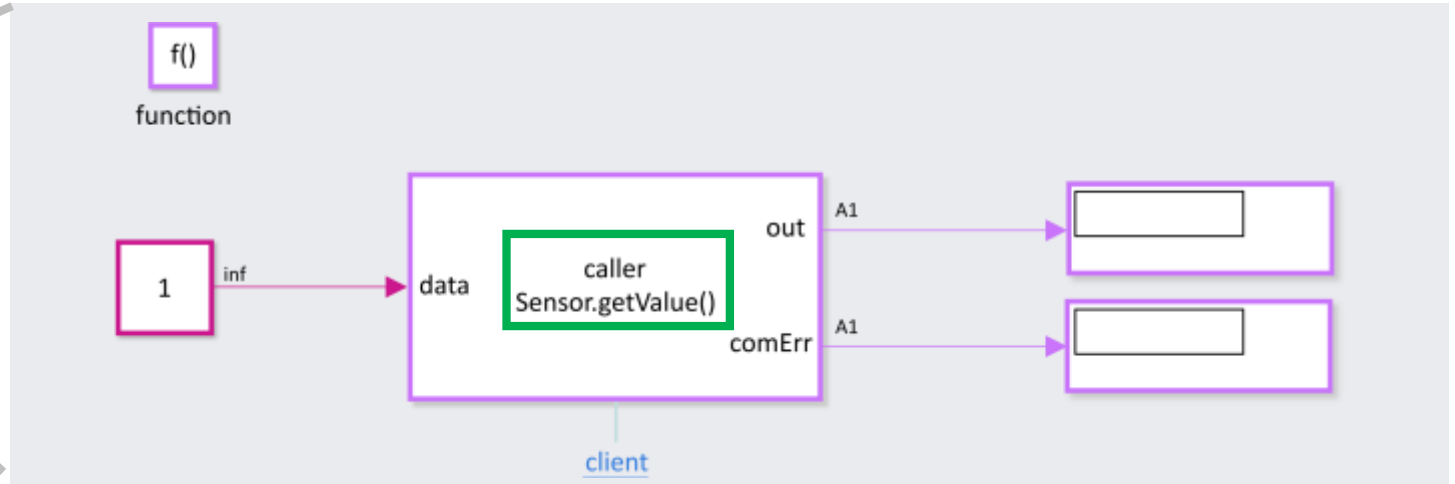
From architecture to implementation

Define SOA interfaces and components at composition level





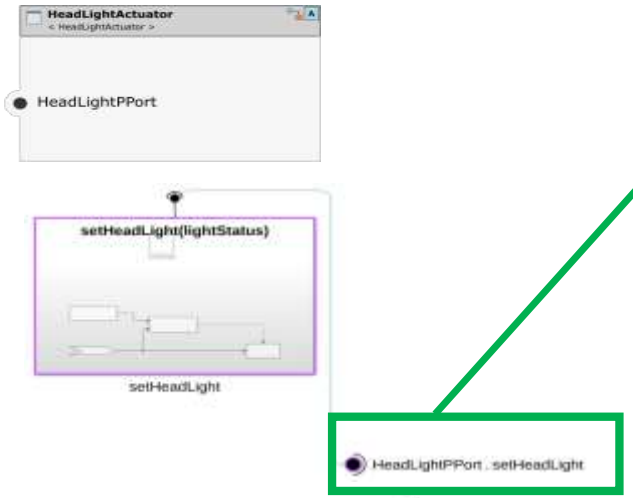




From architecture to implementation

Generate C++ code integrated to adaptive stack seamlessly

- e.g. Service Method
  - Offer service
  - Service Body
  - Terminate Service



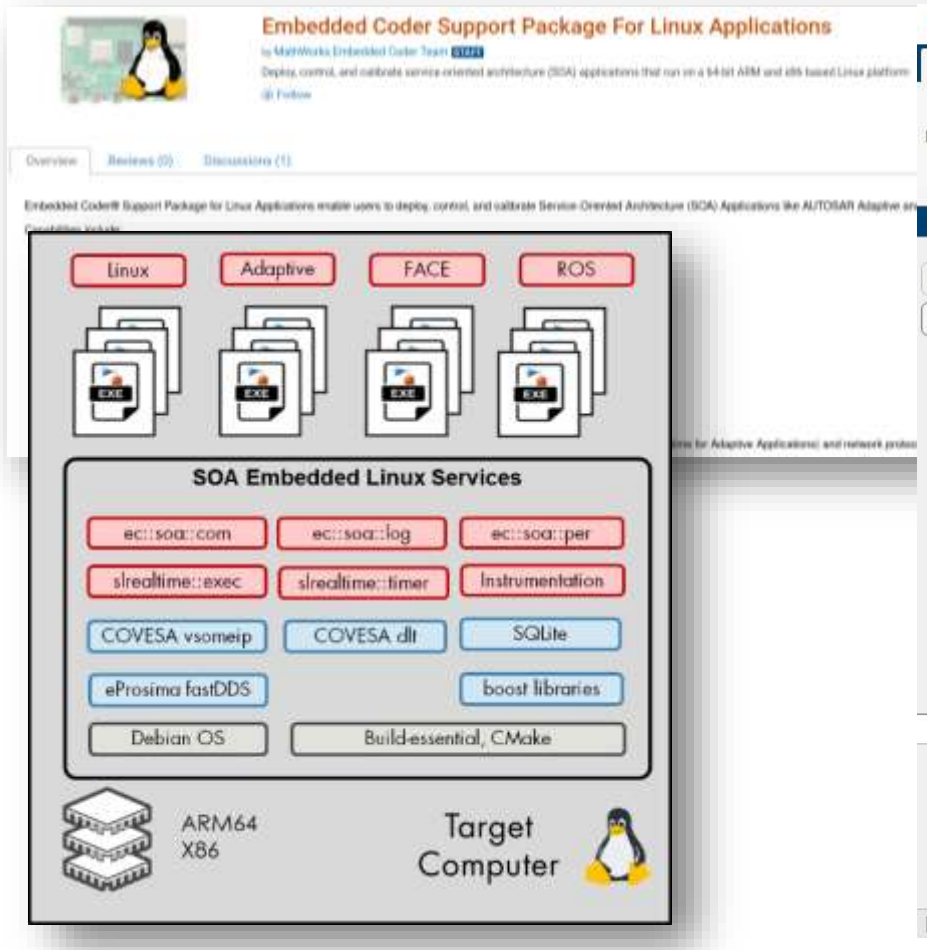

```cpp
// Model initialize function (Offer Service)
void HeadLightActuator::initialize()
{
  // Initialize service provider instance -
HeadLightPPort
  this->HeadLightPPort = makeSkeletonImpl(instanceId,
kEventSingleThread);

  // Bind HeadLightActuator class implementation of
setHeadLight method to service instance's
implementation
  HeadLightPPort->setFuncObjsetHeadLight(
    [this](auto lightStatus){this-
>setHeadLight(lightStatus)});
```

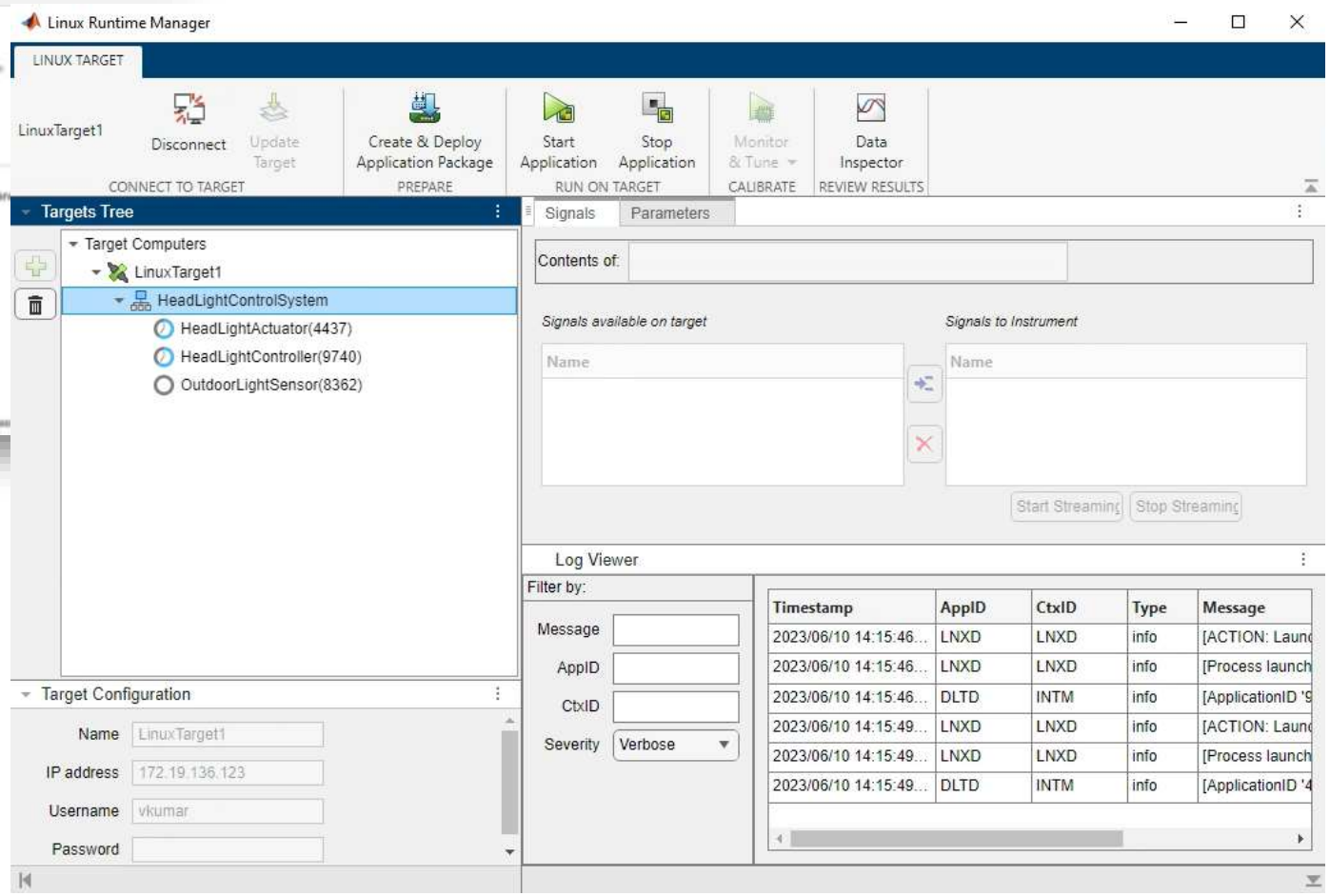
```cpp
// Model step function (service body)
void HeadLightActuator::setHeadLight(boolean_T lightStatus)
{
    // Store value on model instance
    HeadLightActuator_DW.HeadLightStatus =
      lightStatus;
    // S-Function (aralog): '<S2>/ara::log'
    araLog->LogVerbose()<< "Head light status: " <<
      HeadLightActuator_B.HeadLightStatus;
}
```

```cpp
// Model terminate function (Stop Offer Service)
void HeadLightActuator::terminate()
{
  HeadLightPPort->StopOfferService();
}
```

From design to deployment

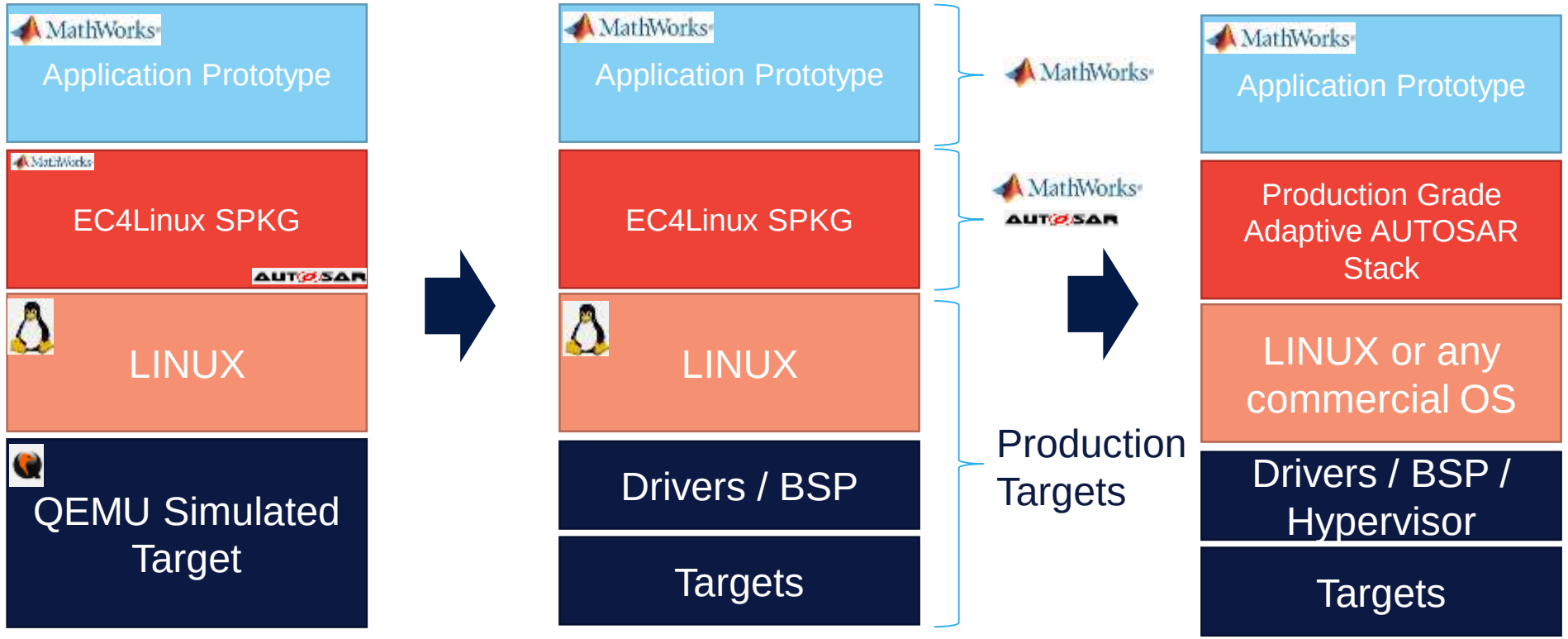
Quick deployment to service stack for prototyping and validation





From prototyping to production

Design and validation incrementally



Rapid Prototyping

Production Software

From prototyping to production

SOA deployment on ARM HPC

ADAS Lane Keeping Waring Algorithm

AUTOSAR

EC Support Package for Linux

Linux

ARM Cortex A

Production Targets

MathWorks

App SW development

MathWorks

Blockset and Support Packages

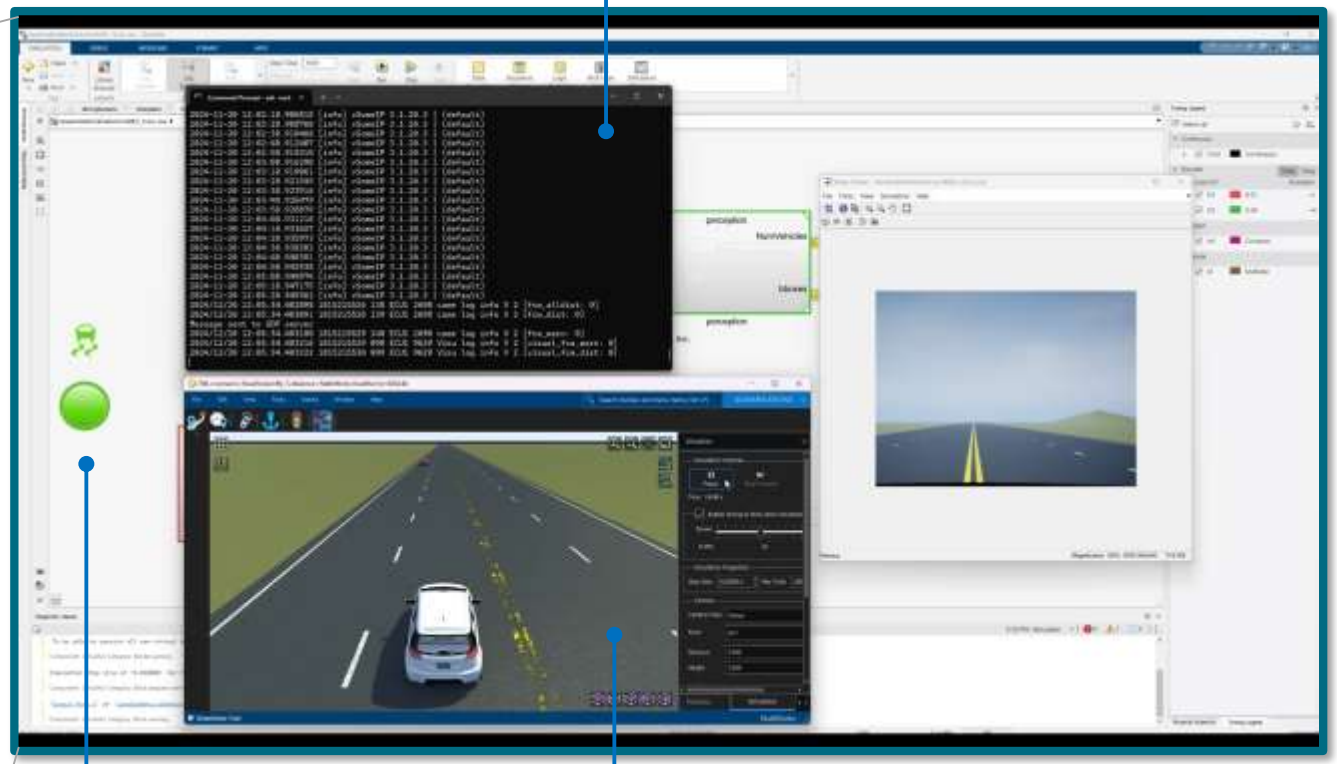
For prototyping

UDP SOME/IP

MathWorks

Vehicle Simulation

Demo Visualization



SSH inside the target running core algorithm run (FCW in this case)

Scene & Scenario in Roadrunner for creation of multiple test scenarios & photorealistic visualization

Simulink in Loop for Vehicle Model and Warning function activation check

AUTOSAR

From physical to virtual

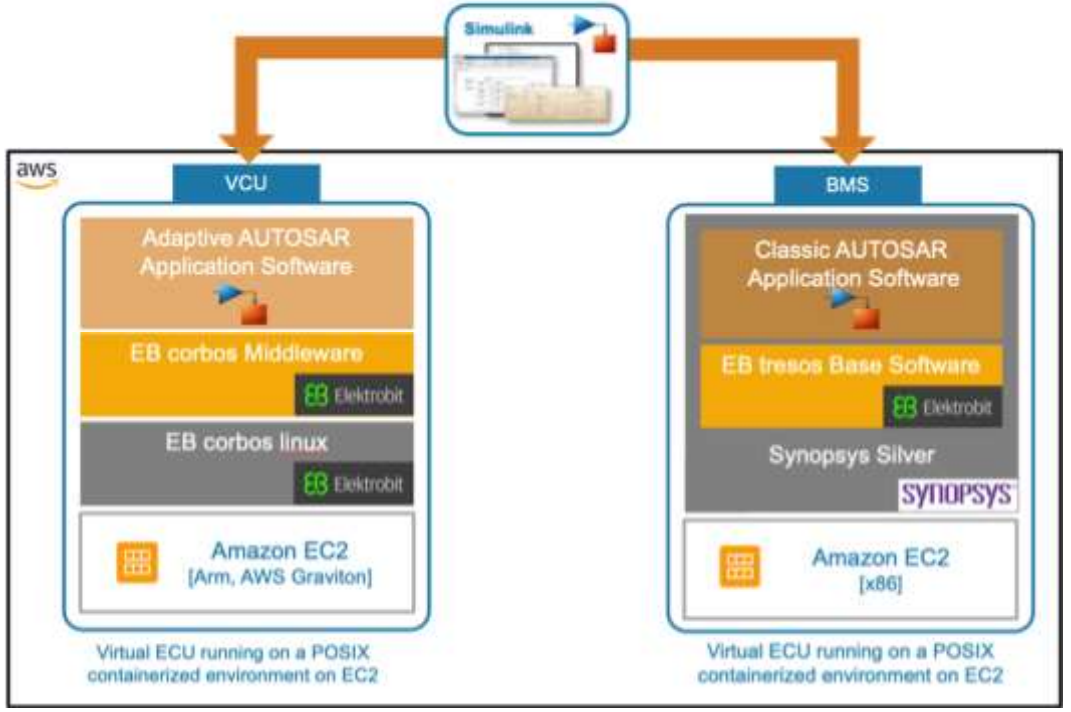
Cloud-based deployment and testing

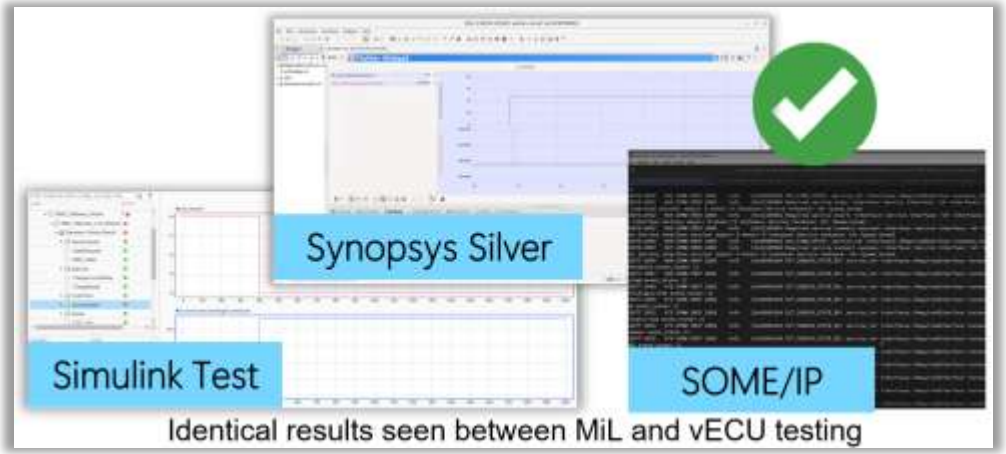
Challenge:

- Ensure application software can integrate into a variety of environments, depending on hardware architecture & middleware
- Verify model based AUTOSAR applications on ‘road ready’ middleware in the cloud without target hardware

Solution:

- Customize application code interface for specific middleware/hardware
  - Classic AUTOSAR
  - Adaptive AUTOSAR / DDS / ROS
- Integrate application code with underlying middleware to deploy within **Level 3** virtual ECUs
- Run virtual ECUs in a cloud native environment and verify the vECU is equivalent to MiL/SiL tests





Content

- Evolution of E/E & Software Architecture
- MathWorks SOA solution
- **Practices of SOA Development**
- Conclusion

SOA Software development methods

Manual Development

- Manually/automatically write the APP framework code
- Manually write the algorithm logic

Simulink Development

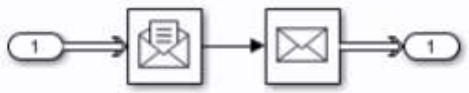
- Utilize Simulink to generate framework code and algorithm logic
- Adopt the adaptive_autosar.tlc template
- Directly utilize the main function and model step function generated by Simulink

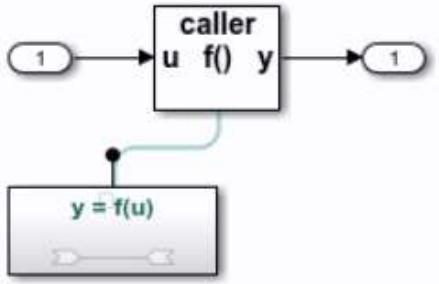
Combination of Manual and Simulink Development

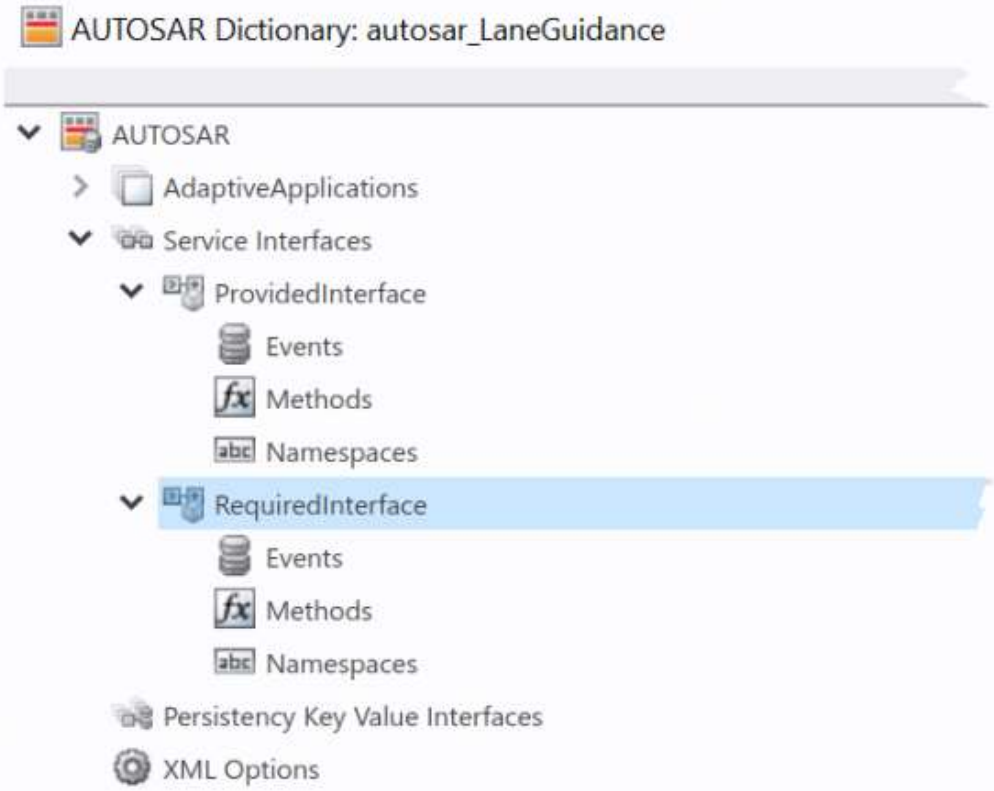
- Manually/automatically write the APP framework code
- Utilize Simulink to generate the algorithm logic
- Adopt the ert.tlc template
- Integrate the algorithm code (Skeleton and Proxy code) generated by Simulink, and call the Step function

Adaptive AUTOSAR Interface implemented by Simulink

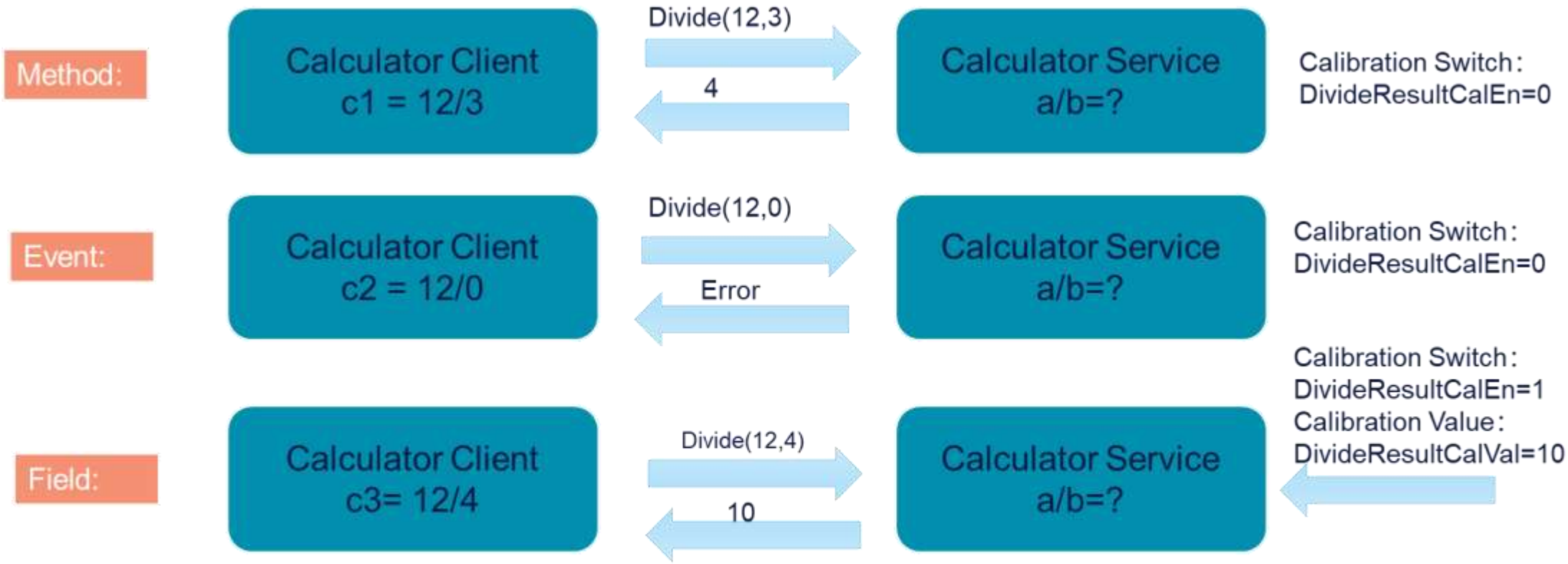
- Event: Input and output
- Method: Simulink Function 与 Function Caller
- Field： Event/Method





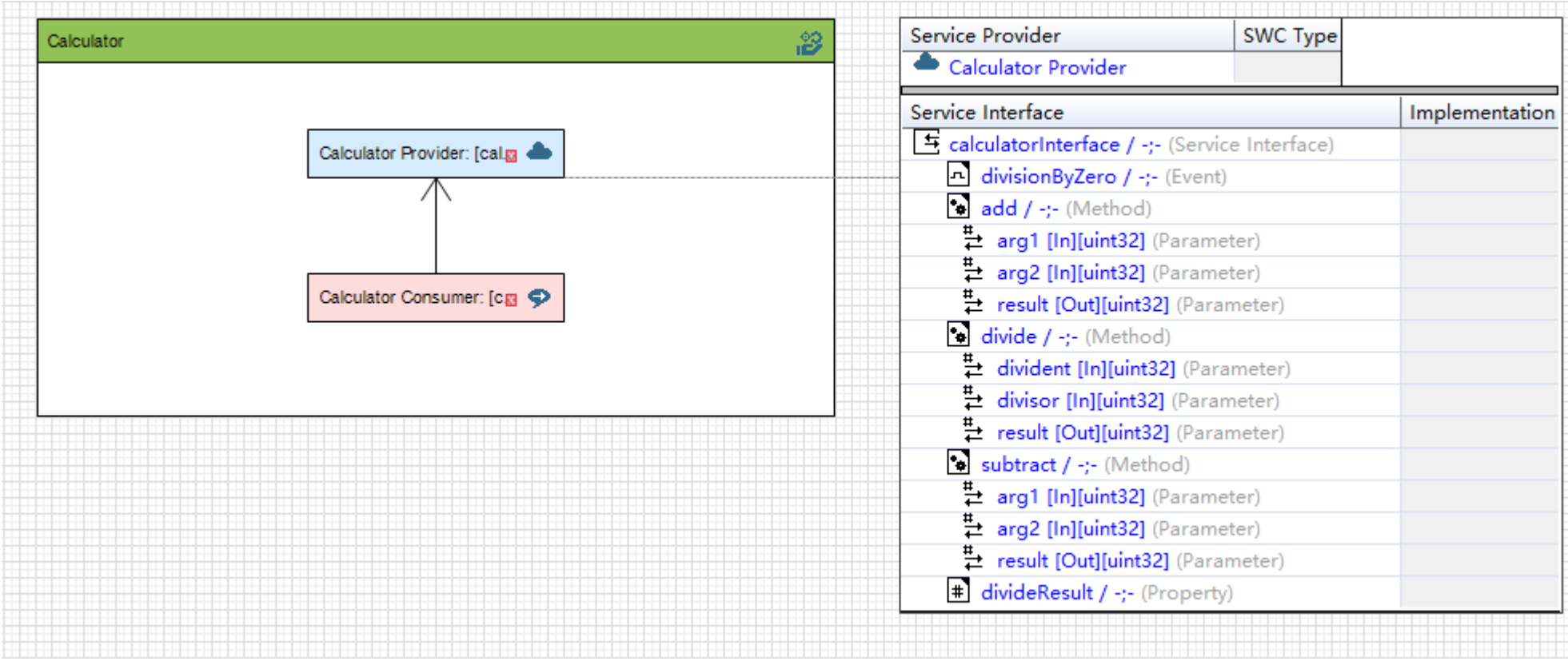


Demo - 32-bit unsigned division calculator



The result of the division operation can be calibrated, and the calibration value is stored in a file. The program reads the calibration value when it starts and saves it when it closes. Through this demo, the features of event/methods/fields/log/persistence are verified.

Demo – Service Architecture Design

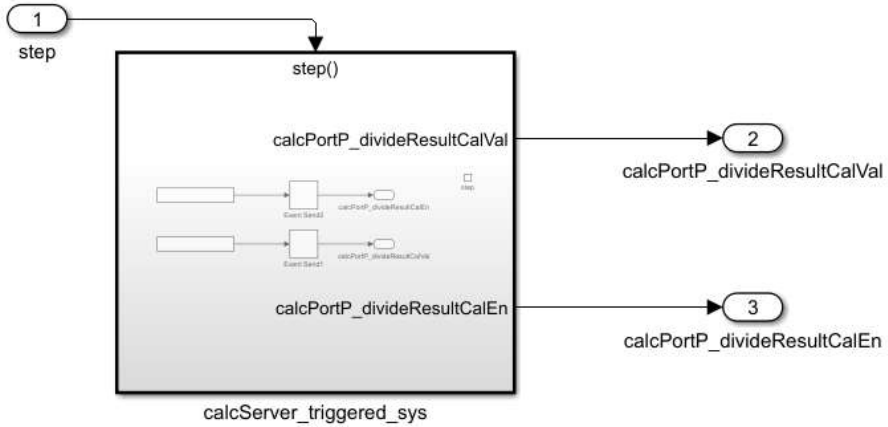


Demo – Create Simulink Model

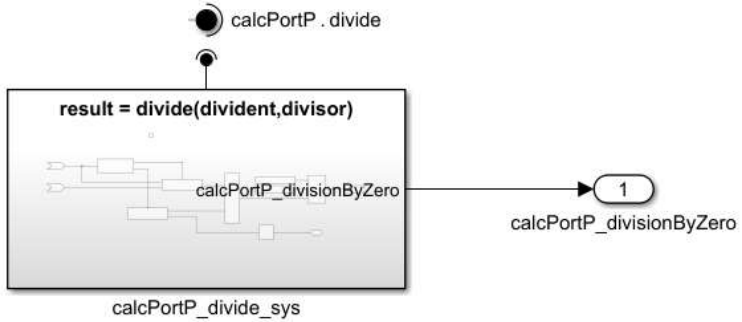
```
% Step1. 导入模型文件
ar = arxml.importer({'GWM_calculator_SI.arxml','standard_datatypes.arxml','GWM_calculator_server.arxml', ...
'GWM_calculator_Server_machine.arxml','GWM_calculator_PSI.arxml','GWM_calculator_Client.arxml', ...
'GWM_calculator_Client_machine.arxml','GWM_calculator_CSI.arxml','GWM_calculator_Per.arxml'})

% Step2：创建服务端和客户端
% 创建服务端
createComponentAsModel(ar,'/vector/calculatorService/calcServer')
% 创建客户端
createComponentAsModel(ar,'/vector/calculatorClient/calcClient')
end
```

Demo – Algorithm Development（Server）

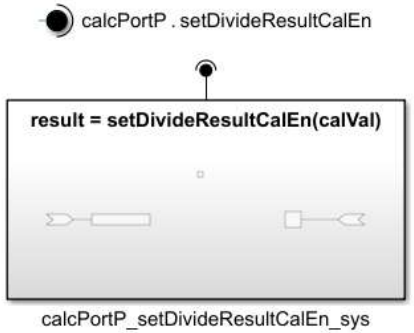
Field: Notifier divide result calibration value and calibration variable



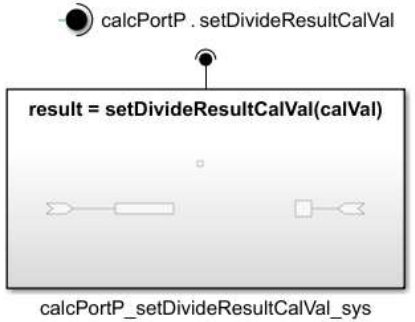
Division calculator method



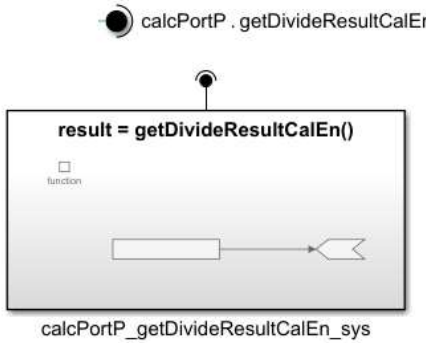
Field: Set divide result calibration variable enable/disable



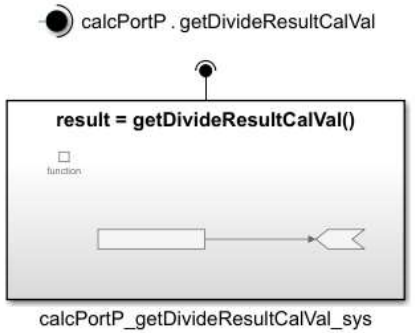
Field: Set divide result calibration value



Field: Get divide result calibration variable enable/disable

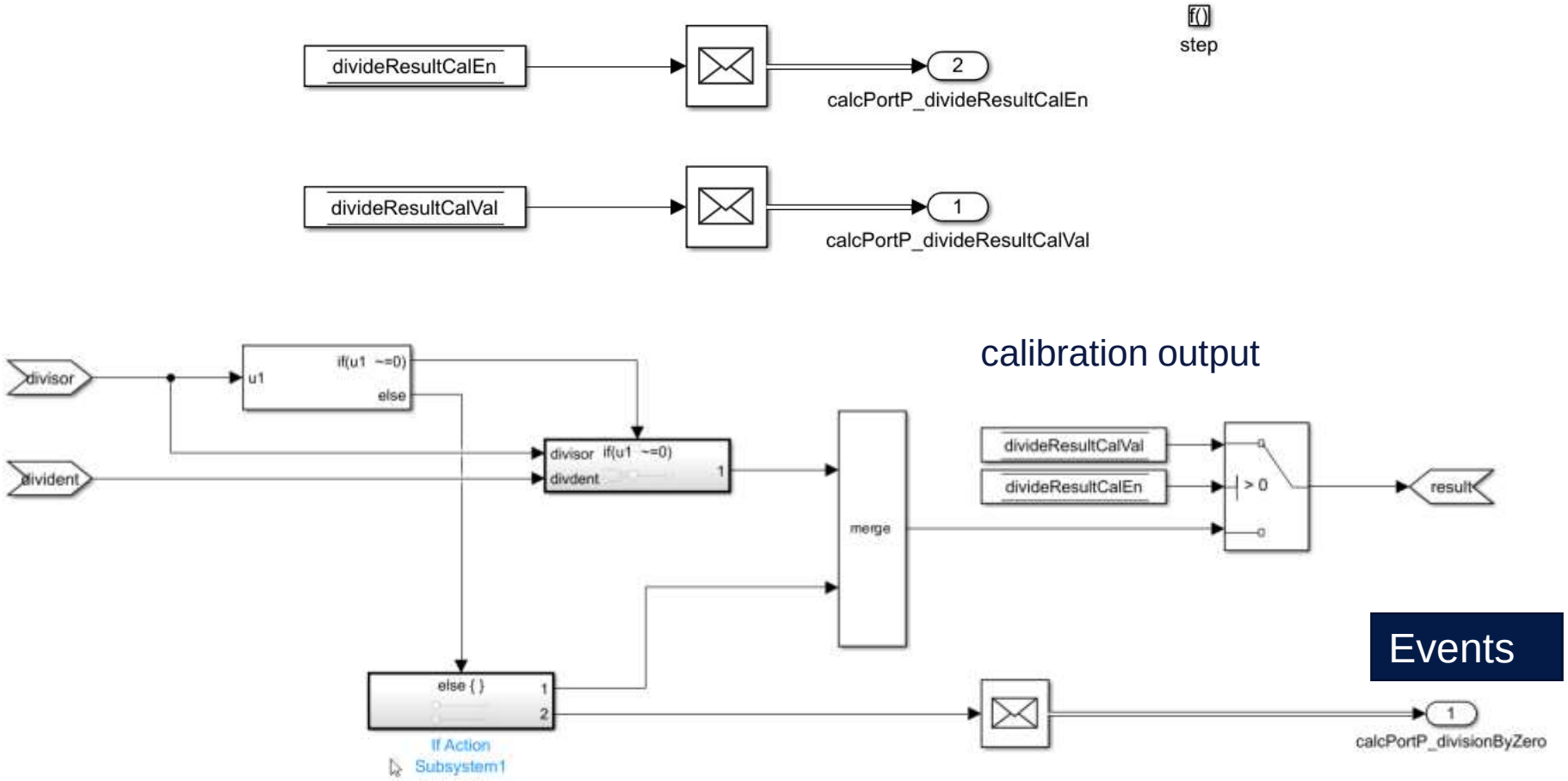


Field: Get divide result calibration Value

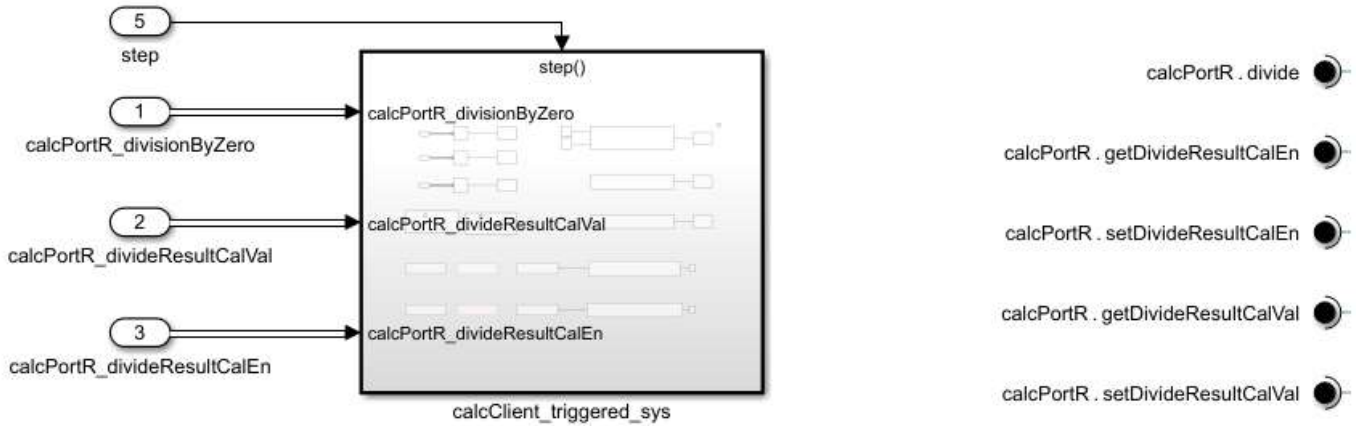


Demo – Algorithm Development（Server）

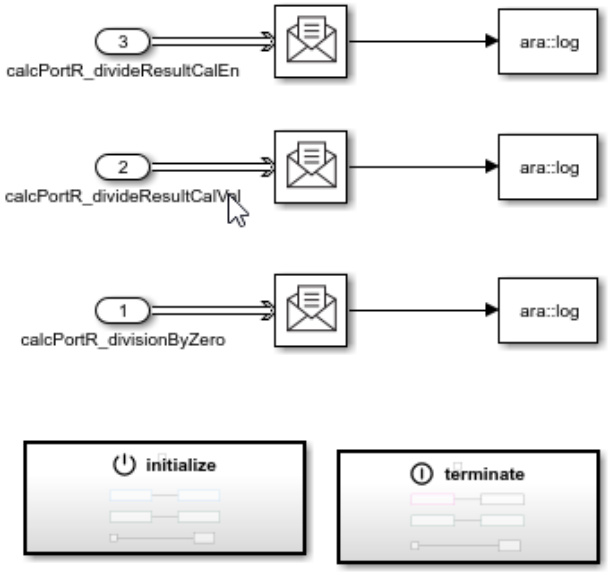
Step Function, output divide result calibration value and calibration variable



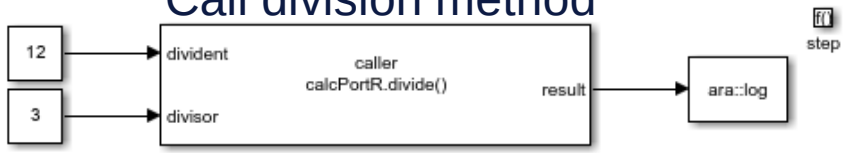
Demo – Algorithm Development（Client）



Demo– Algorithm Development（Client）



Call division method



Call division result calibration variable method

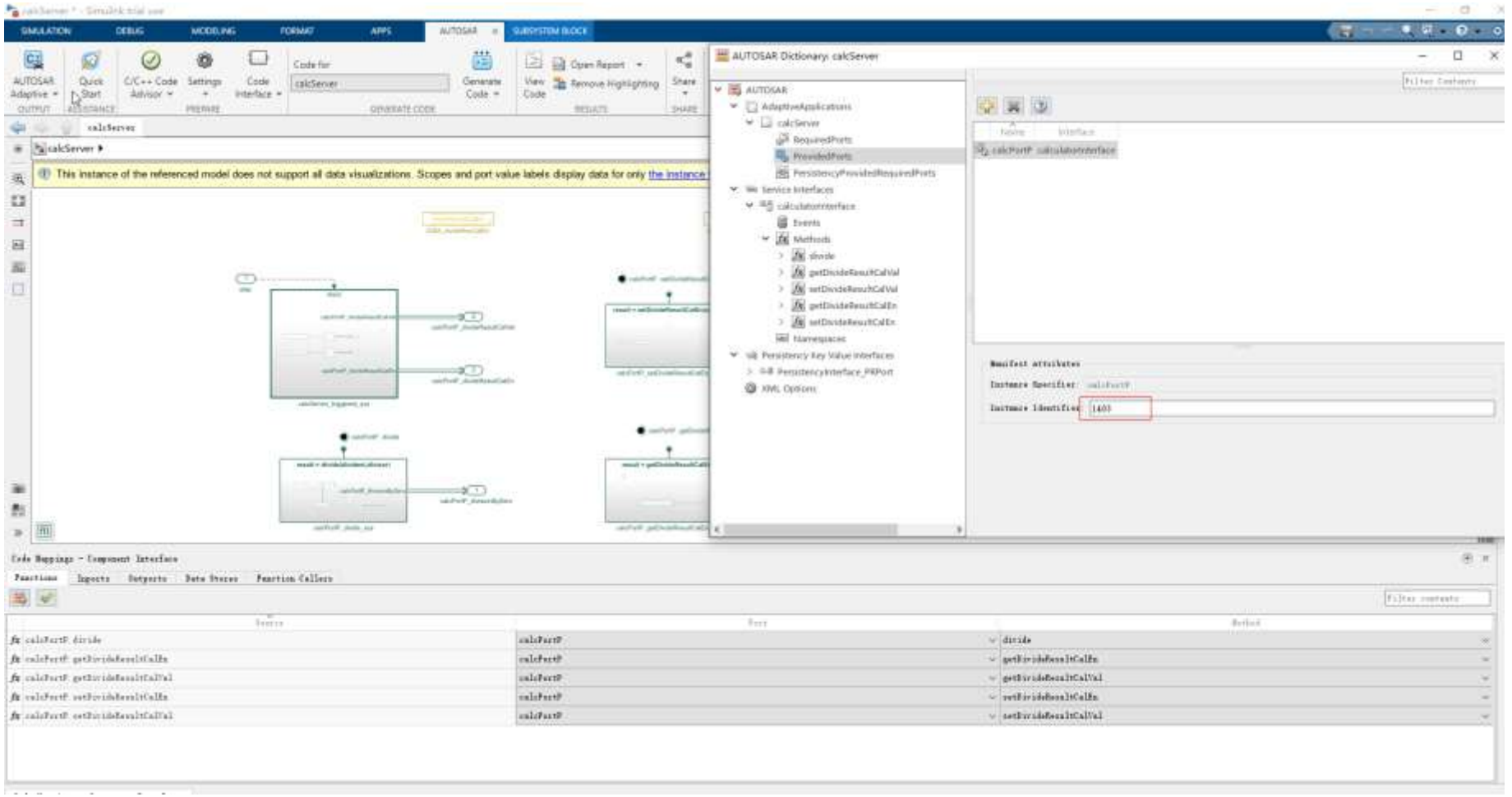
Call division result calibration value method

Call set division result calibration variable method

Call division result calibration value method

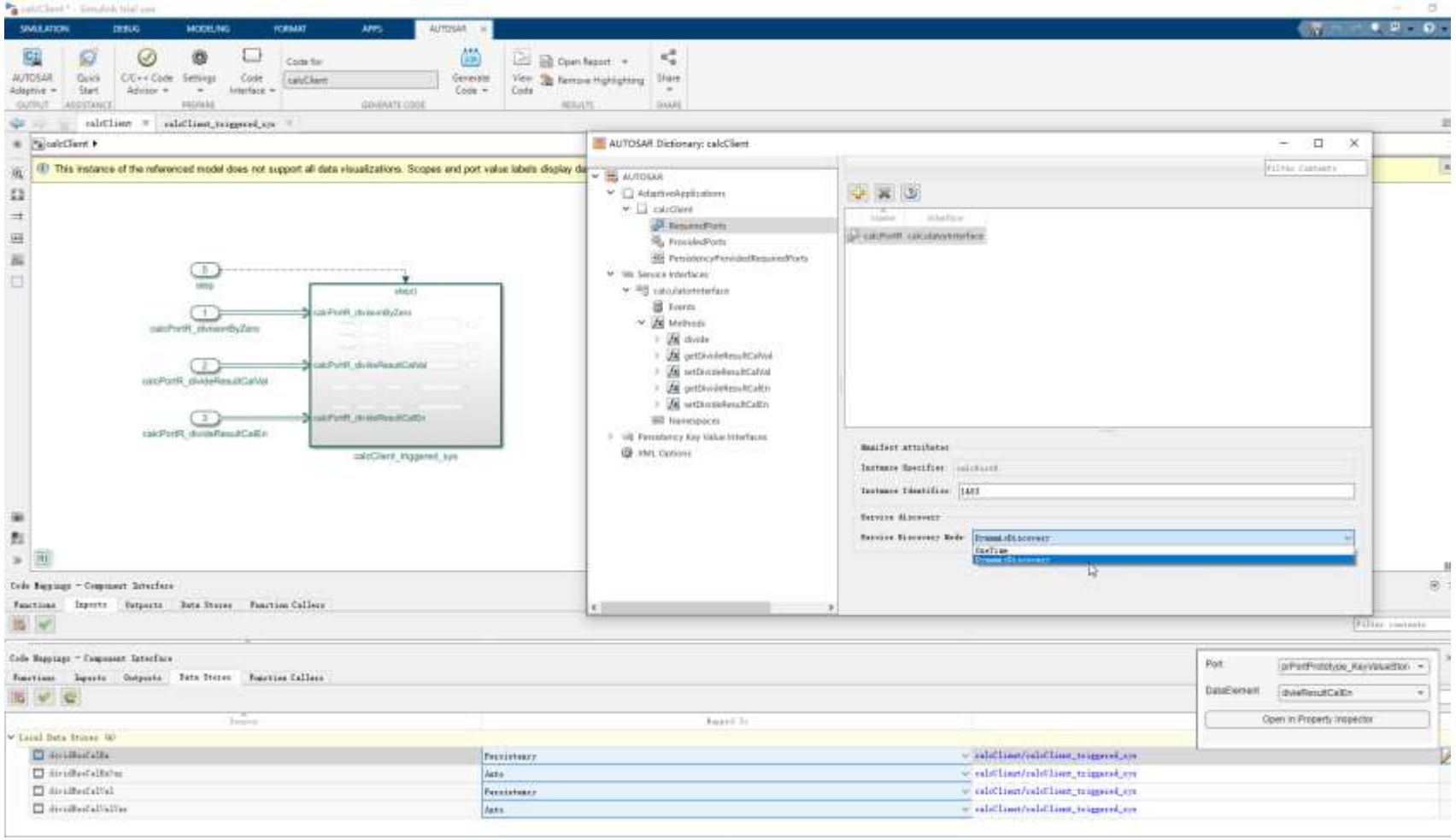
Demo– Model Configuration（Server）

Update Instance Identifier

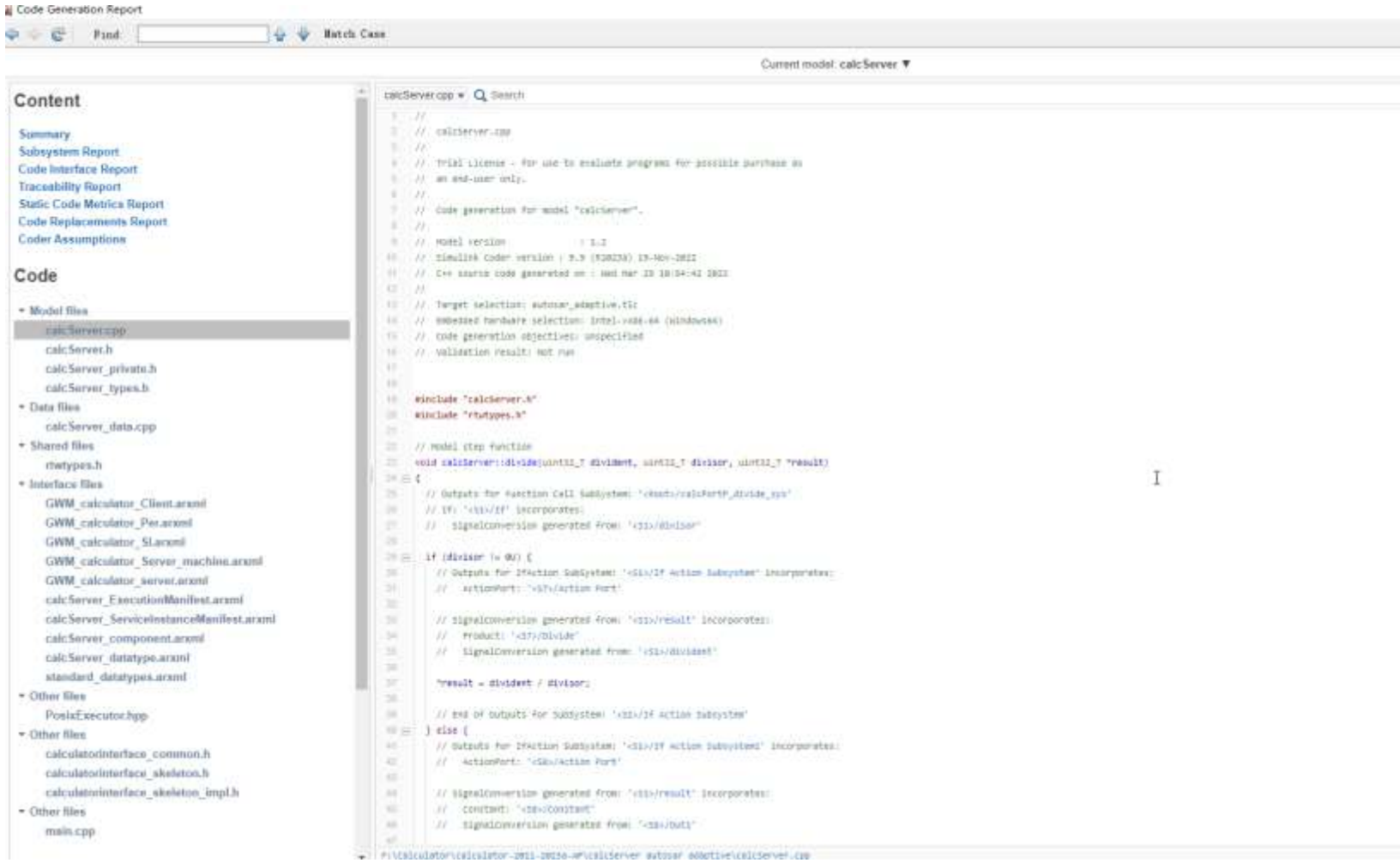


Demo– Model Configuration（Client）

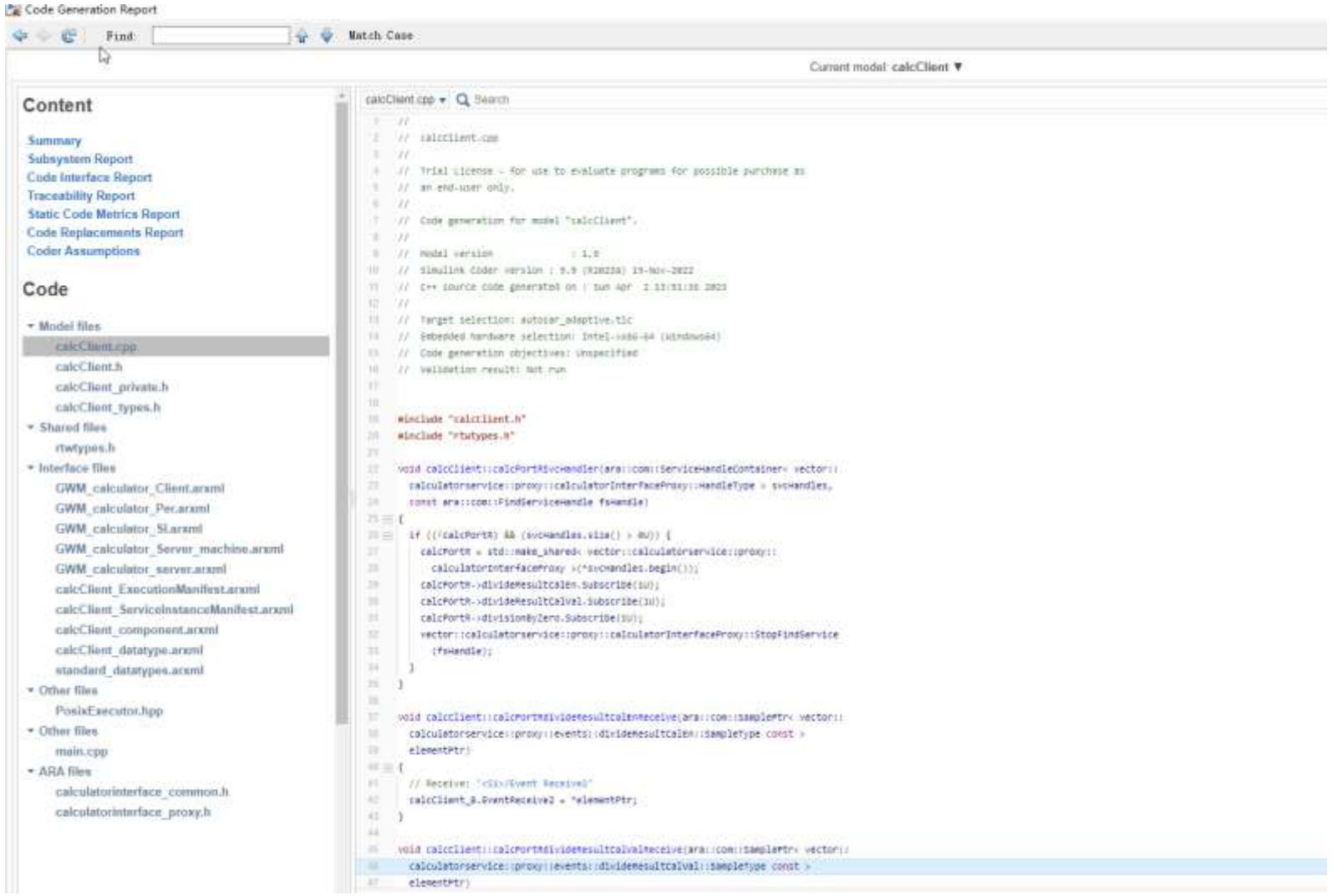
1）Update Service Discovery Mode　2) Mapping Persistence Port Data Element



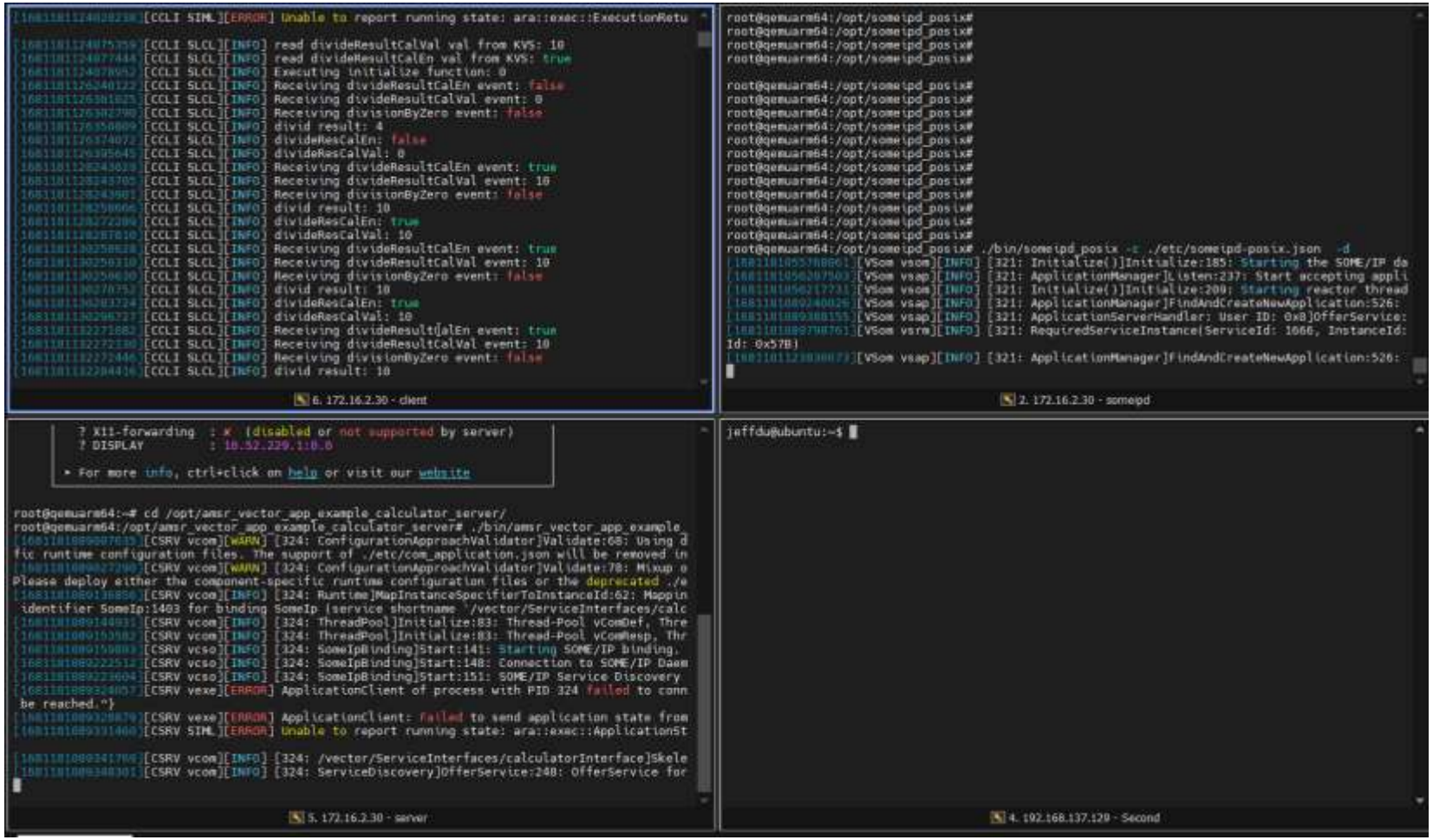
Demo – Code Generation - Server



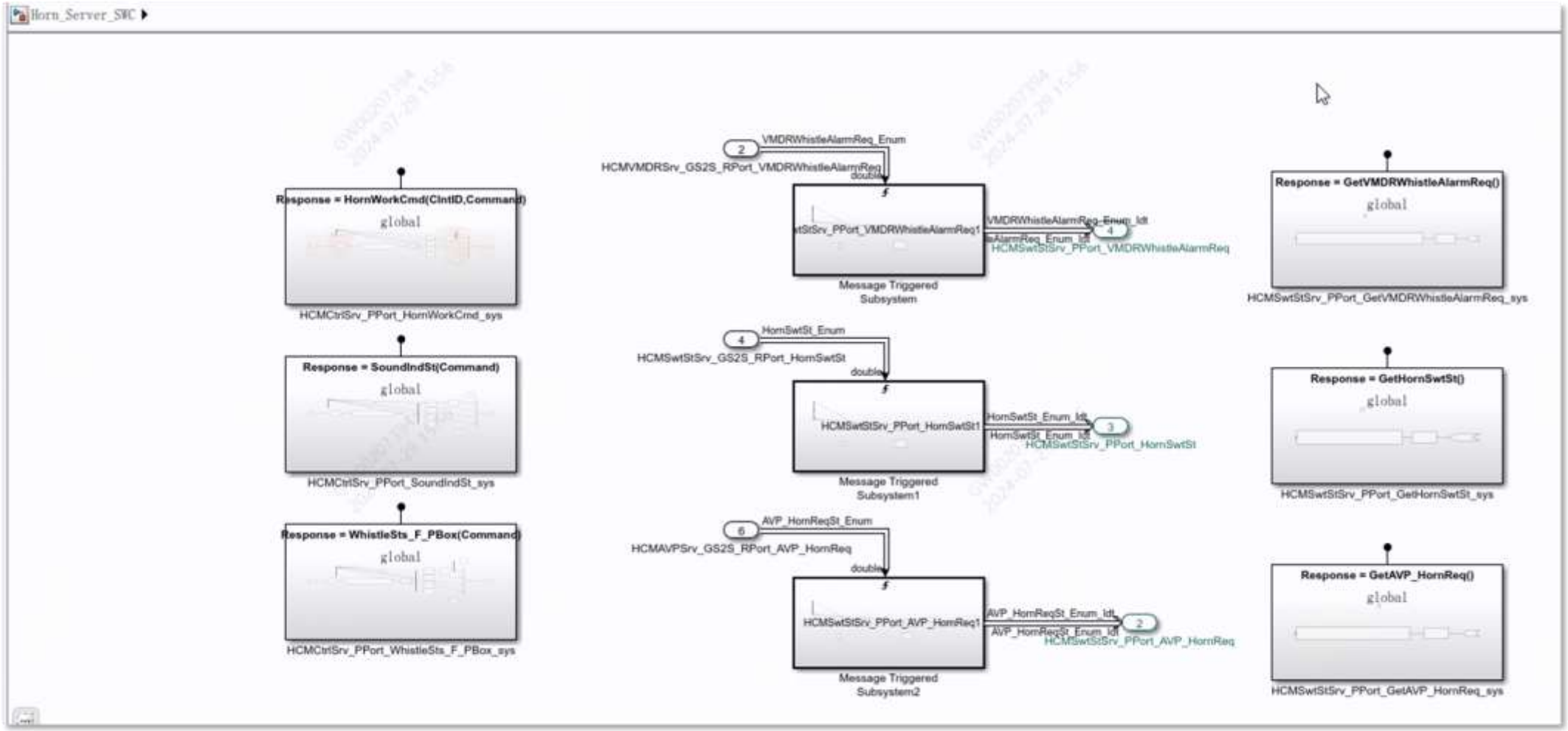
Demo – Code Generation - Client



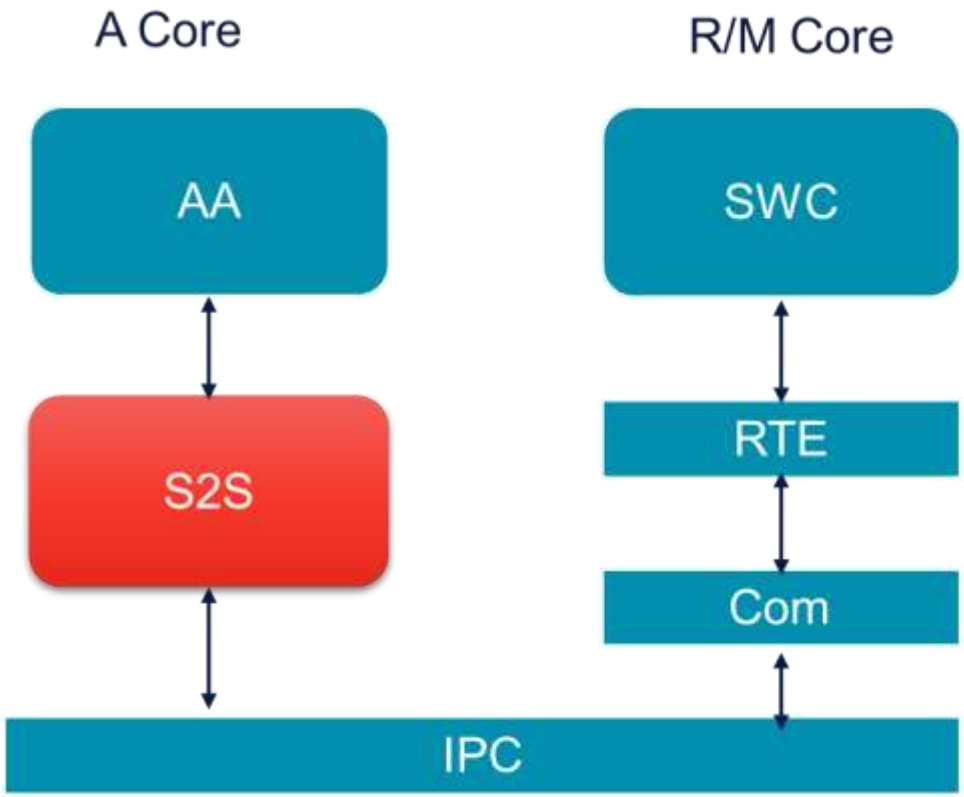
Demo – Running Result



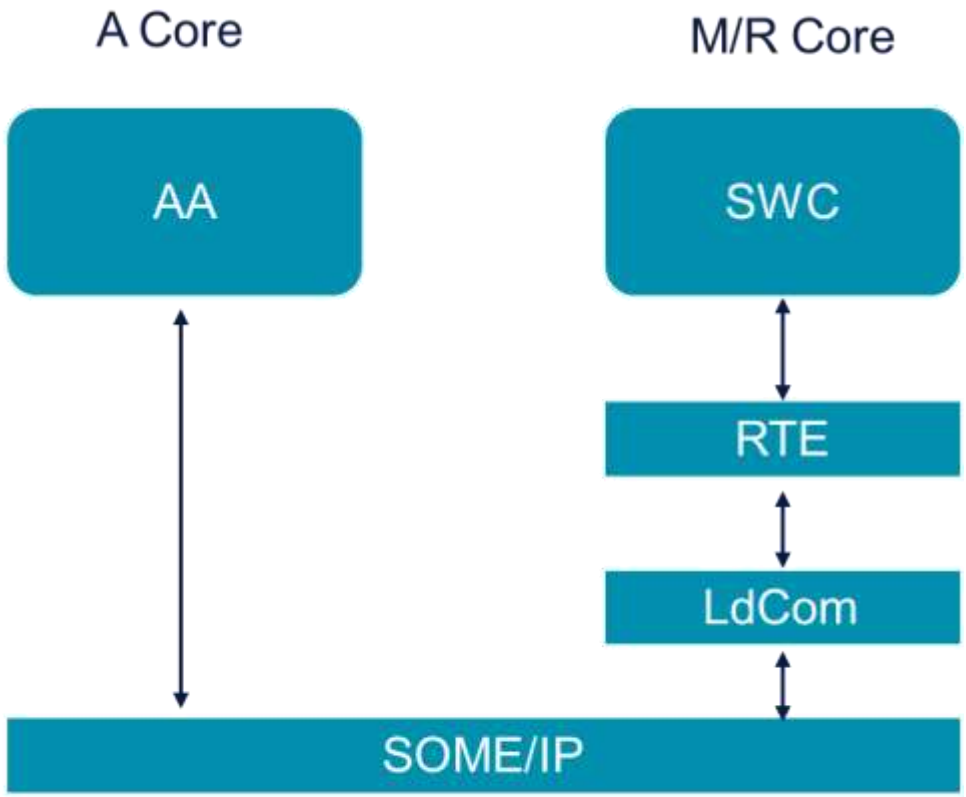
GWM GEEP4.0 Projects – Horn Server



Communication Between A core and R/M core(Signal2Service)



Method 1 (used in current project)



Method 2

Problems during SOA Development using Simulink

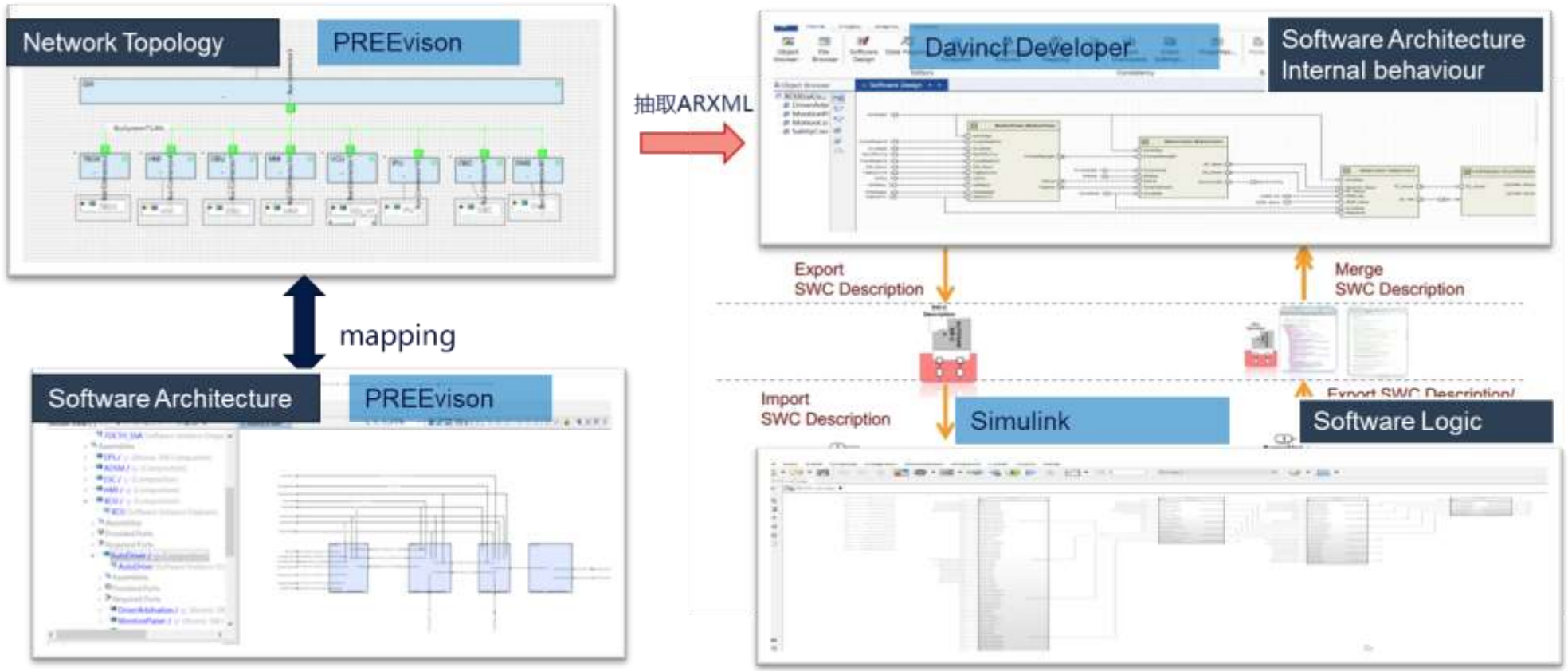
Inconsistencies in Standard Implementation

- Specifications adopted by various entities do not fully conform to the standards, resulting in generated code from architecture models not perfectly matching the code produced by Simulink, necessitating manual modifications.

Simulink currently does not support PHM

Simulink's support for Log & Trace needs improvement

SOA Software Development based on CP



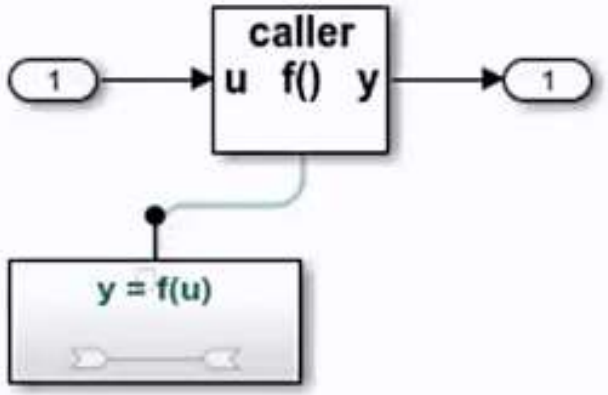
The toolchain is relatively mature comparing with AP.

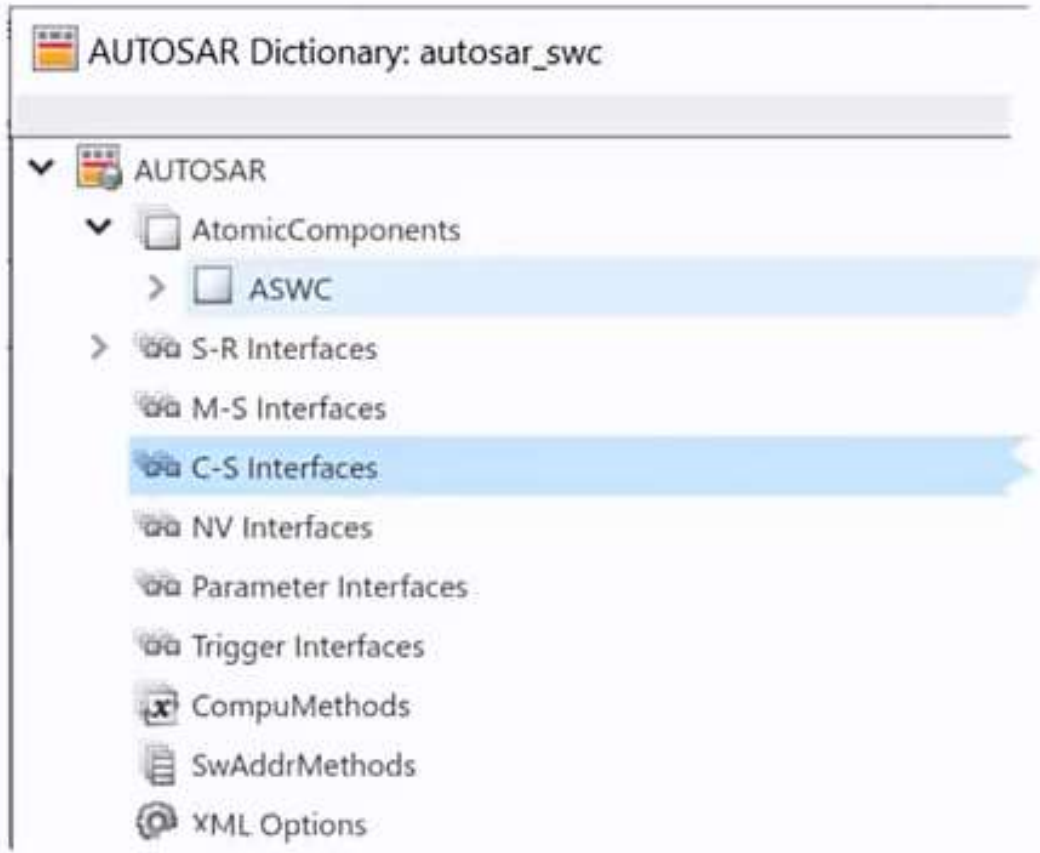
CP Event/Methods implementation using Simulink

RTE 接口

- S-R : 输入与输出

- C-S: Simulink Function 与 Function Caller





Content

- Evolution of E/E & Software Architecture
- MathWorks SOA solution
- Practices of SOA Development
- **Conclusion**

Conclusion

SOA software compliant with both CP and AP standards can be developed based on Simulink.

Existing software can be reused when developing SOA software using Simulink.

Developing AP software with Simulink can circumvent the issue of insufficient C++ personnel resources.

The development toolchain in a CP environment is relatively mature.

The development toolchain in an AP environment still needs further refinement.


AUTOSAR
Thanks for listening
BOSCH
Continental
Ford
gm
STELLANTIS
TOYOTA
VOLKSWAGEN GROUP