Evolution of Simulink for Signal and SOA based Applications

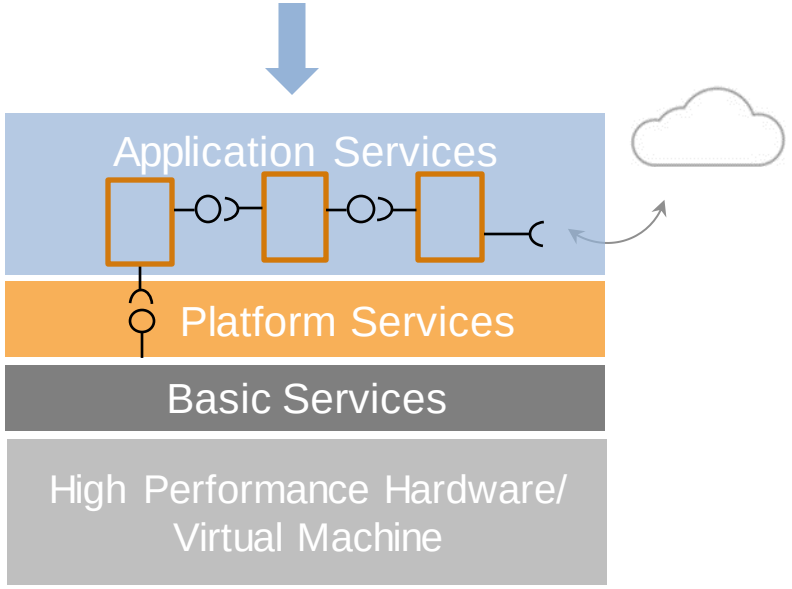
Oct 10, 2023 | MAC, Chennai

Shwetha B Patil

In the era of Software-Defined Vehicle

- Automotive industry is embracing Service-Oriented Architectures (SOA) as a new paradigm to design modern applications like Software-Defined Vehicles (SDVs)

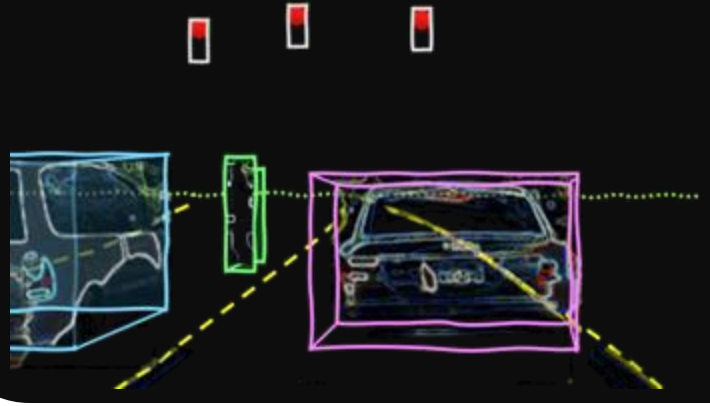
SW updates
- Frequent
- Selective
- Over-the-air



Higher HW abstraction:
Service-oriented architectures

Automakers are increasingly building software in-house with SOA based design

Software Defined Vehicle

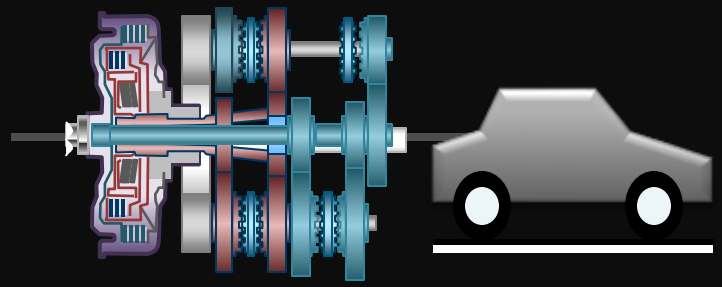


Autonomy
Connectivity
AI

Steering,
Braking

Speed,
Velocity

Traditional Vehicle



Controls
Real-time
CAN

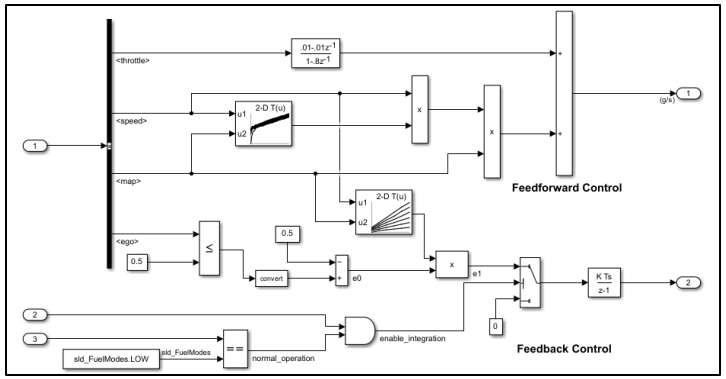
SOA based standards

AUTOSAR Adaptive Platform

ROS

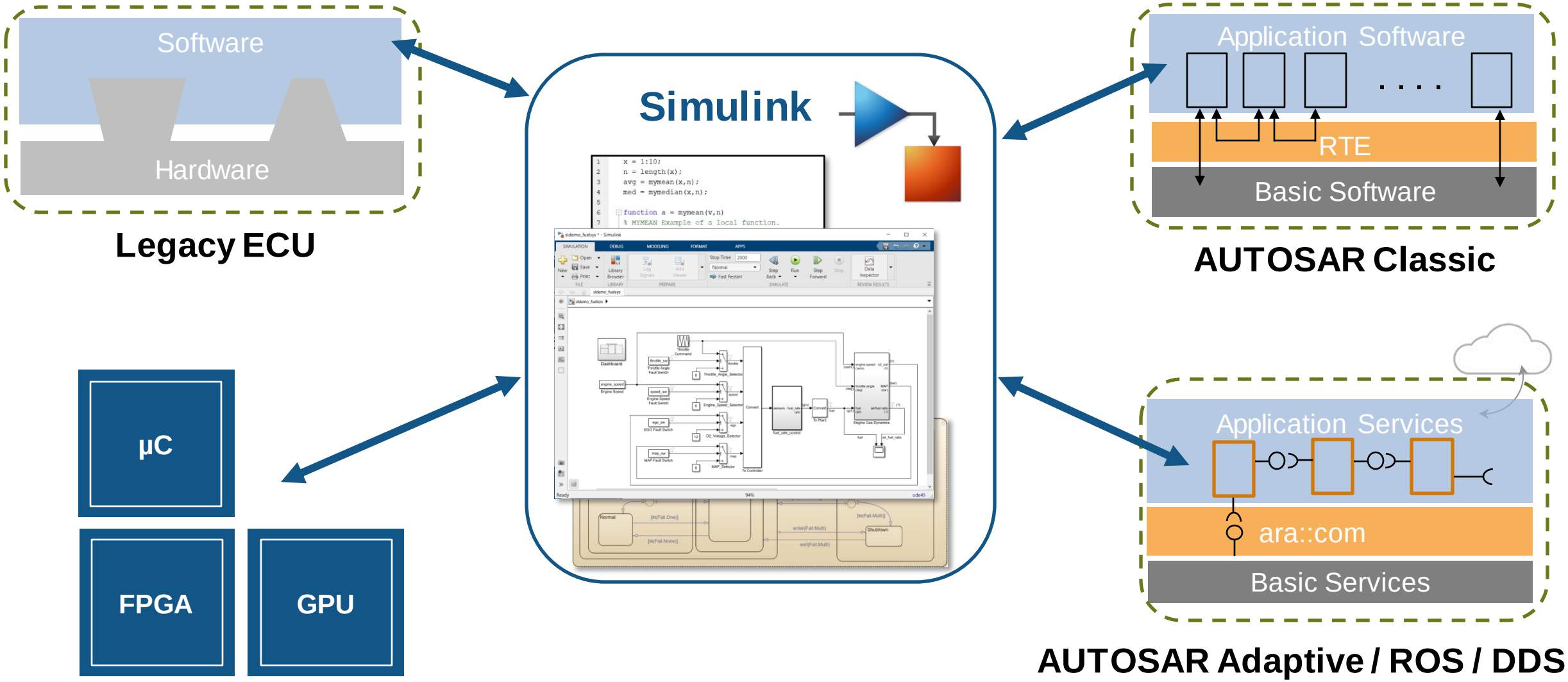
DDS

Model-Based Design



AUTOSAR

Simulink: deploy software to different targets and standards



Agenda

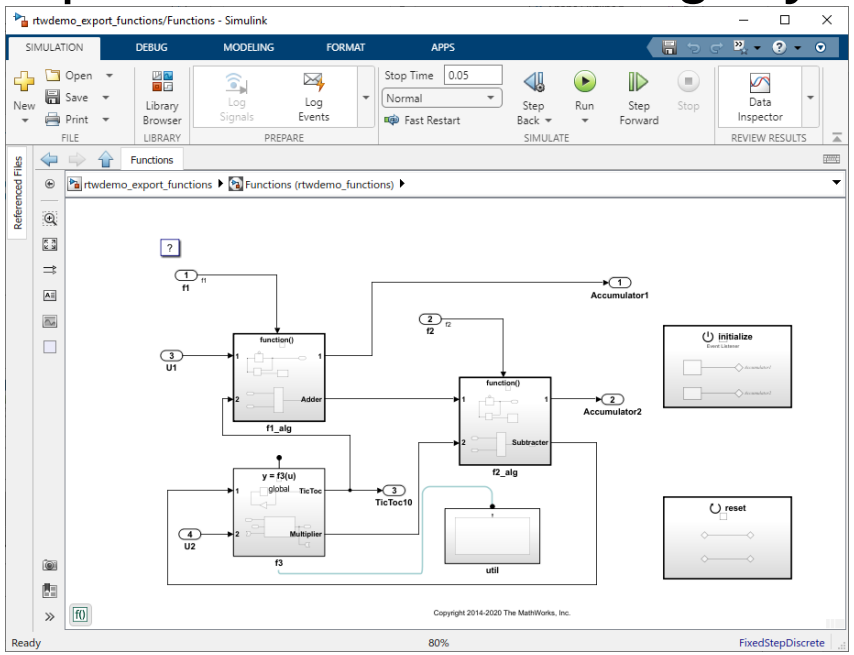
- Advanced Simulink semantics for the development of services
- Software architecture modeling
- Conclusions and key takeaways

Agenda

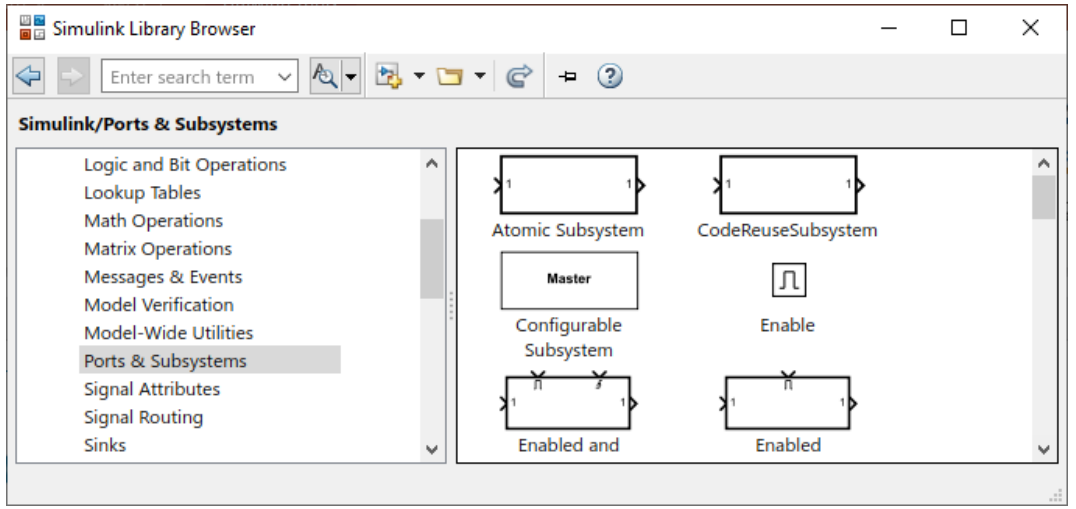
- Advanced Simulink semantics for the development of services
- Software architecture modeling
- Conclusions and key takeaways

Simulink Supports Exporting Callable Functions Well

- Export Function Modeling style

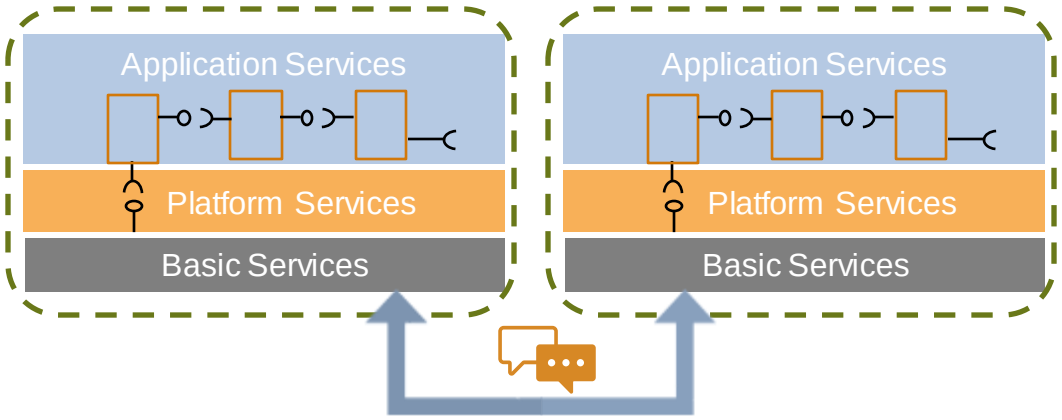


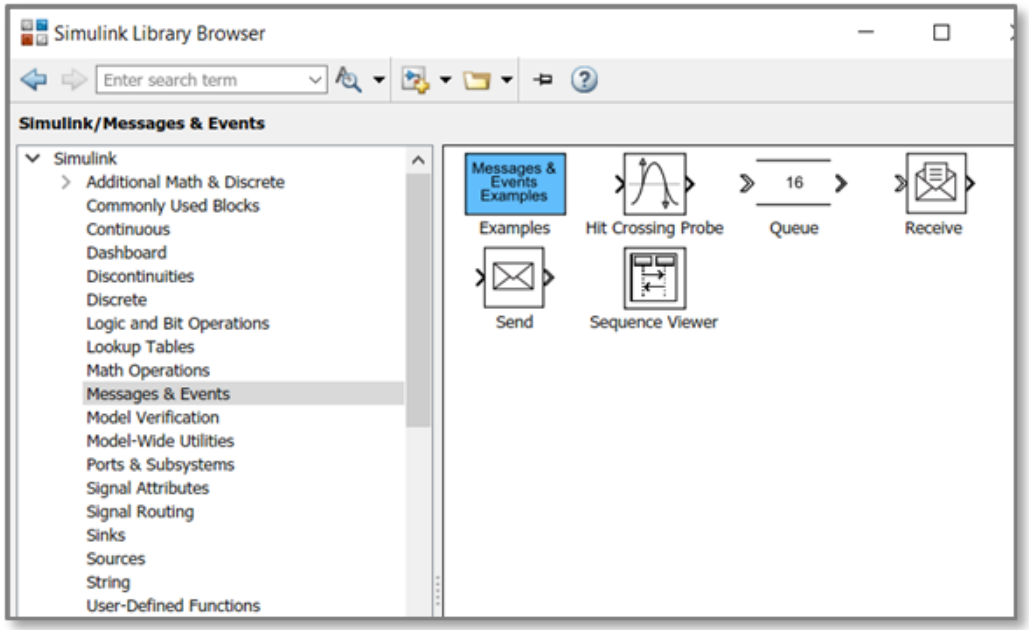
rtwdemo_export_functions

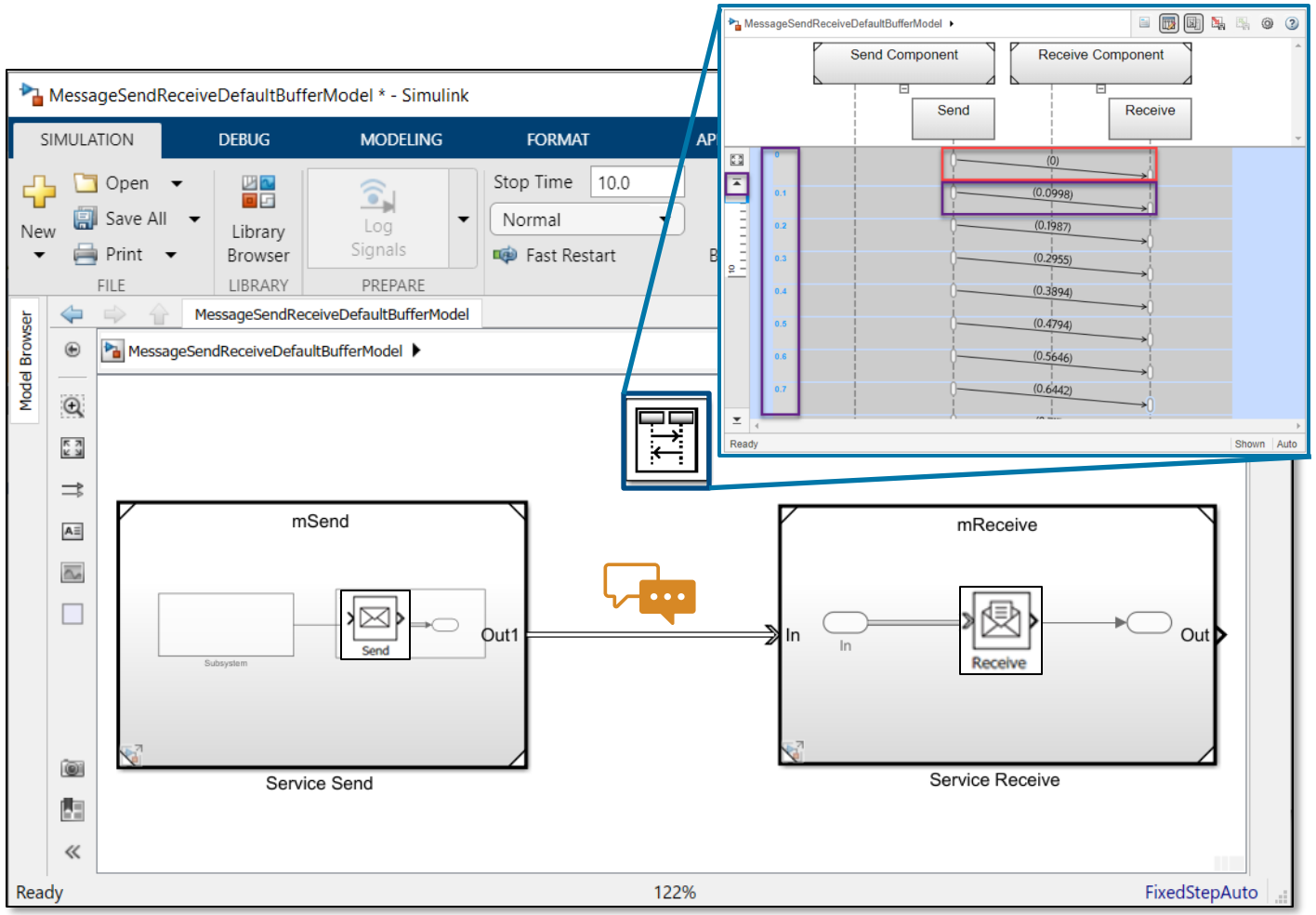


Ports and Subsystems

Service-Oriented Behavior Modeling with Simulink Messages





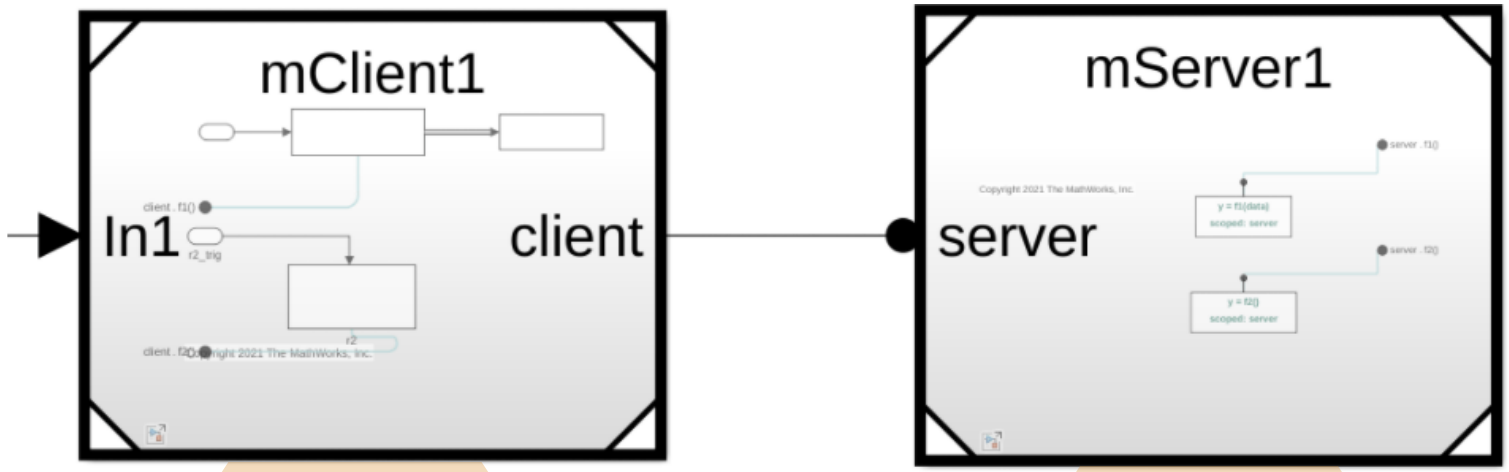


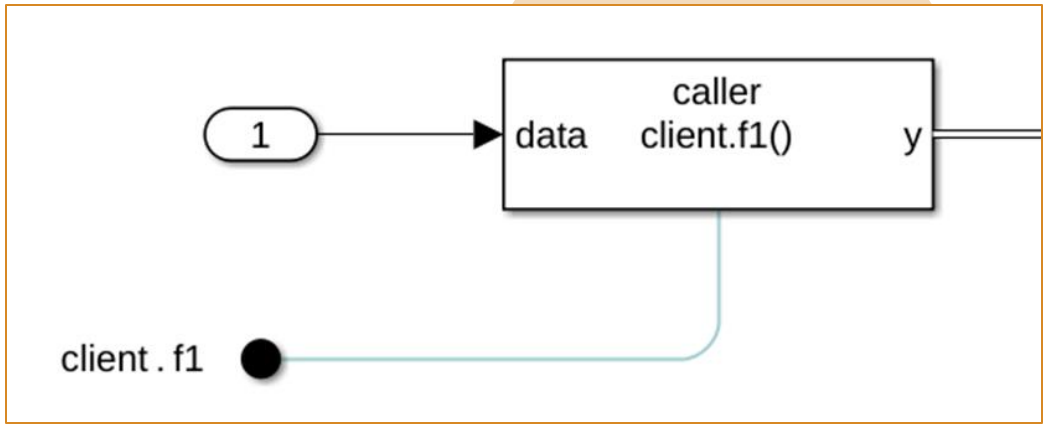
You can model service-oriented communication using messages (Send/Receive)

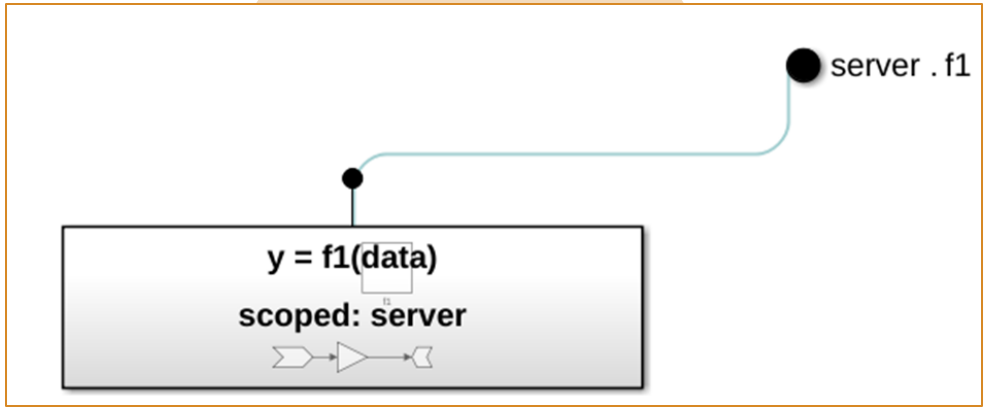
https://www.mathworks.com/help/simulink/ug/simulink-messages-overview.html

Service-Oriented Behavior Modeling with Service-Based Functions

Function Ports for SOA



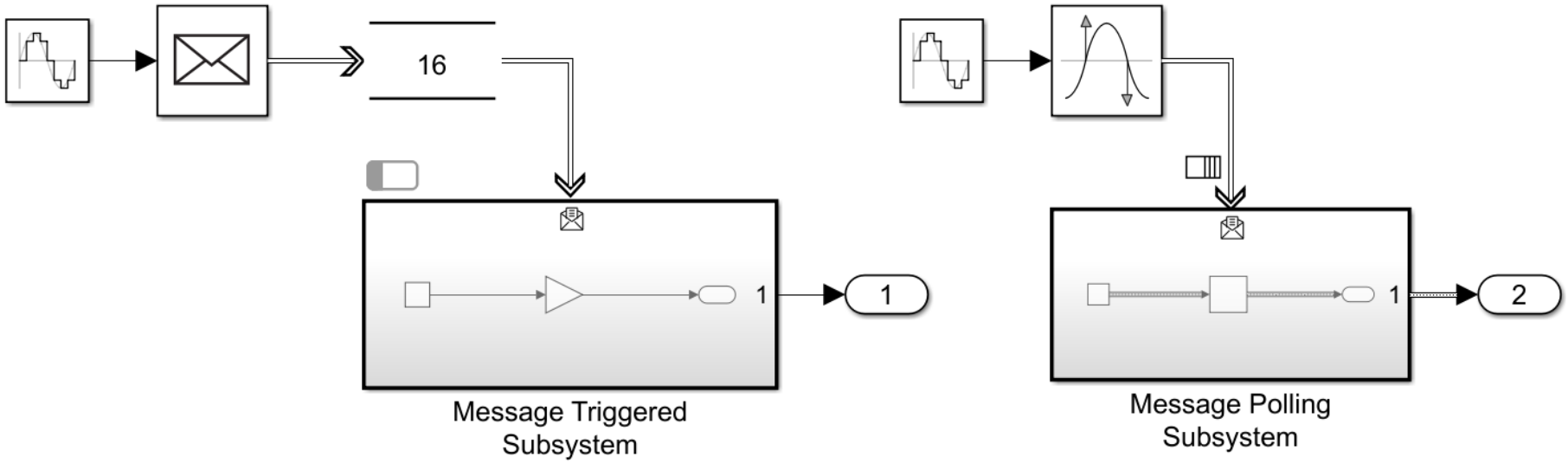




Model client and server components to facilitate data sharing using a functional interface between component models

https://www.mathworks.com/help/simulink/ug/call-simulink-functions-in-other-models-using-function-ports.html

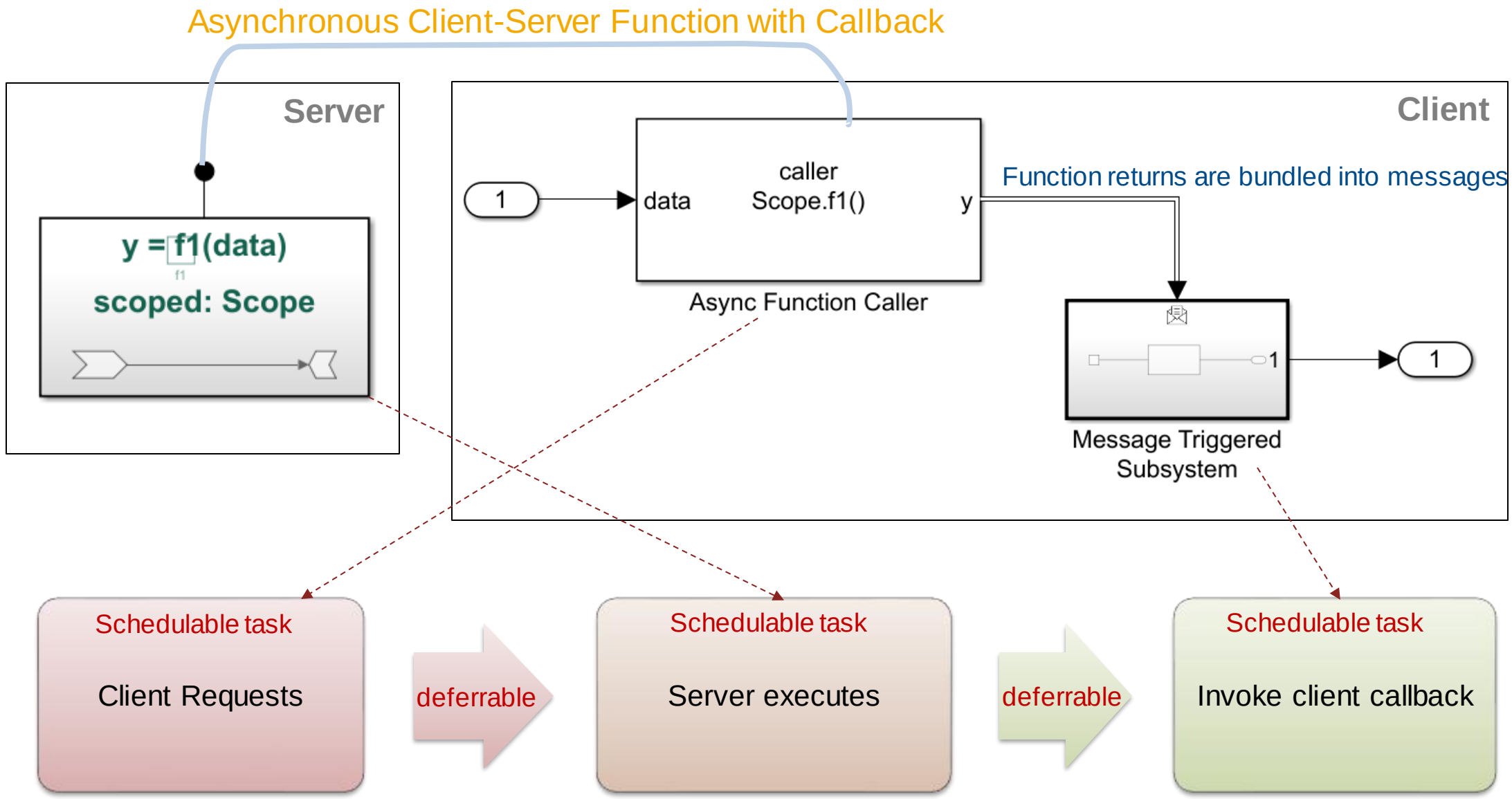
Service-Oriented Behavior Modeling with Message Triggered/Polling Subsystem



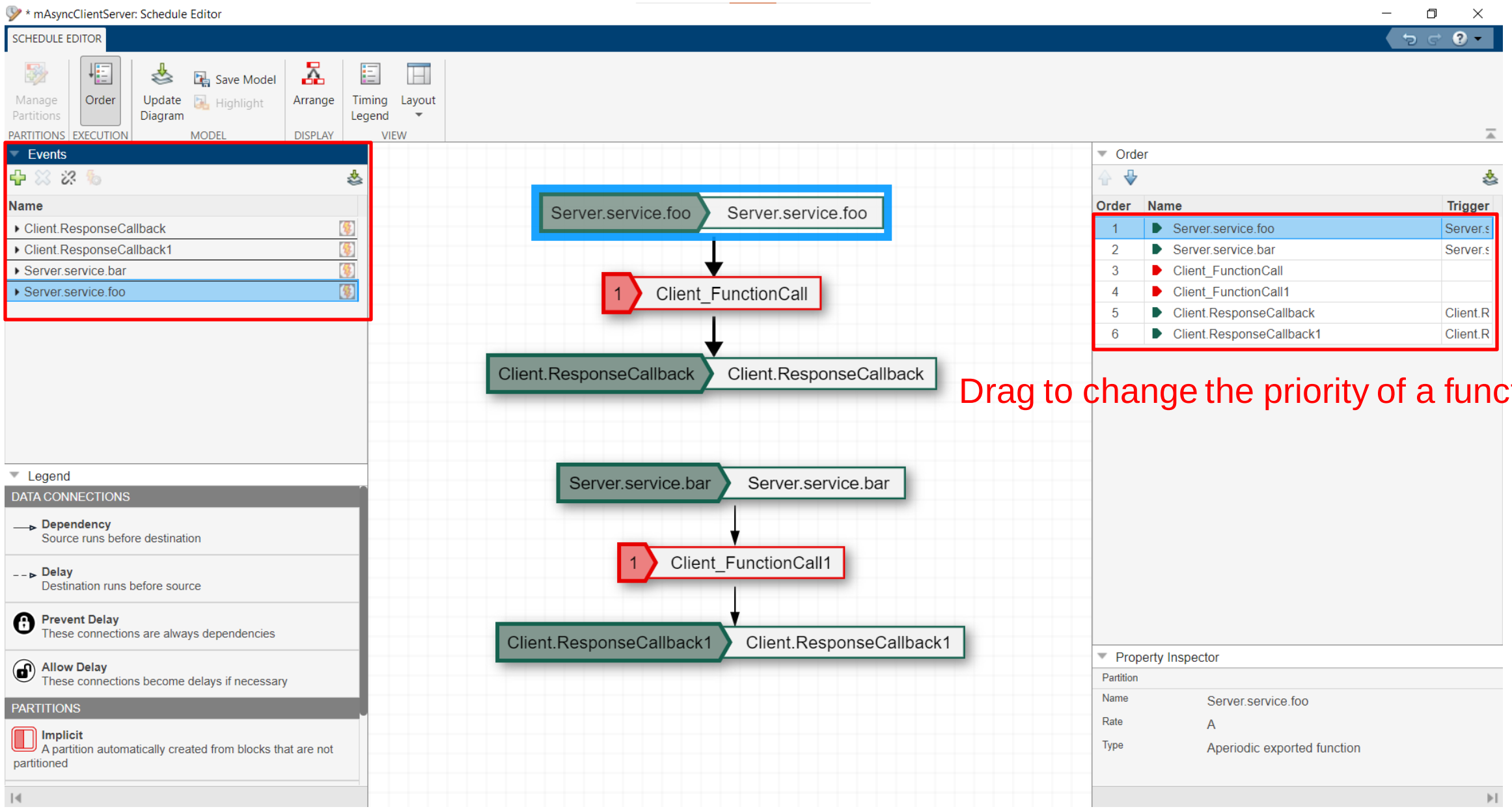
- New blocks to process messages by executing subsystem when message is available
- Model and generate code for components that are executed on message arrival

https://www.mathworks.com/help/simulink/ug/message-triggered-and-message-polling-subsystems.html

Service-Oriented Behavior Modeling with Service-Based Functions



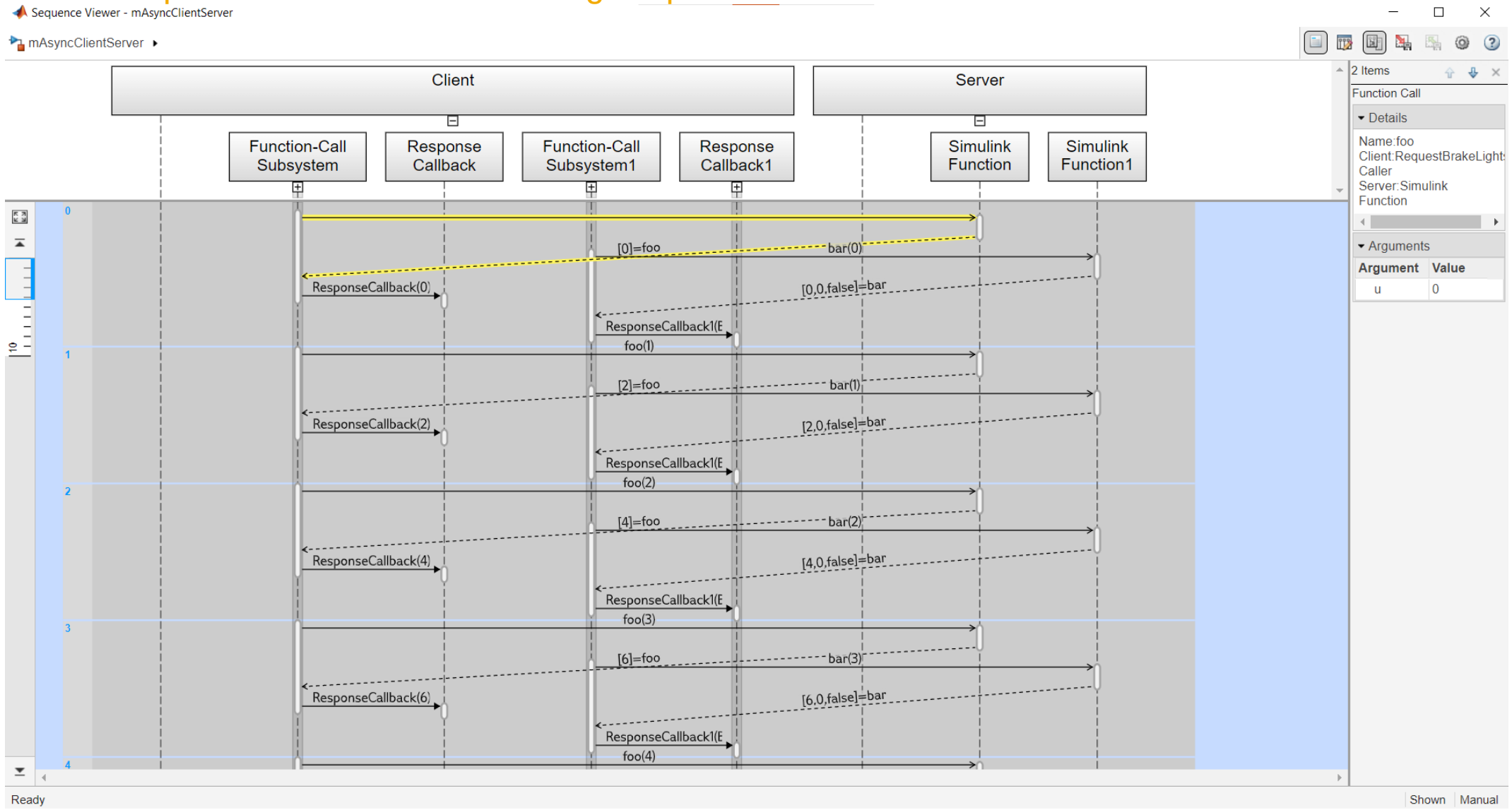
Service-Oriented Behavior



https://www.mathworks.com/help/simulink/ug/schedule-editor-overview.html

Service-Oriented Behavior Modeling

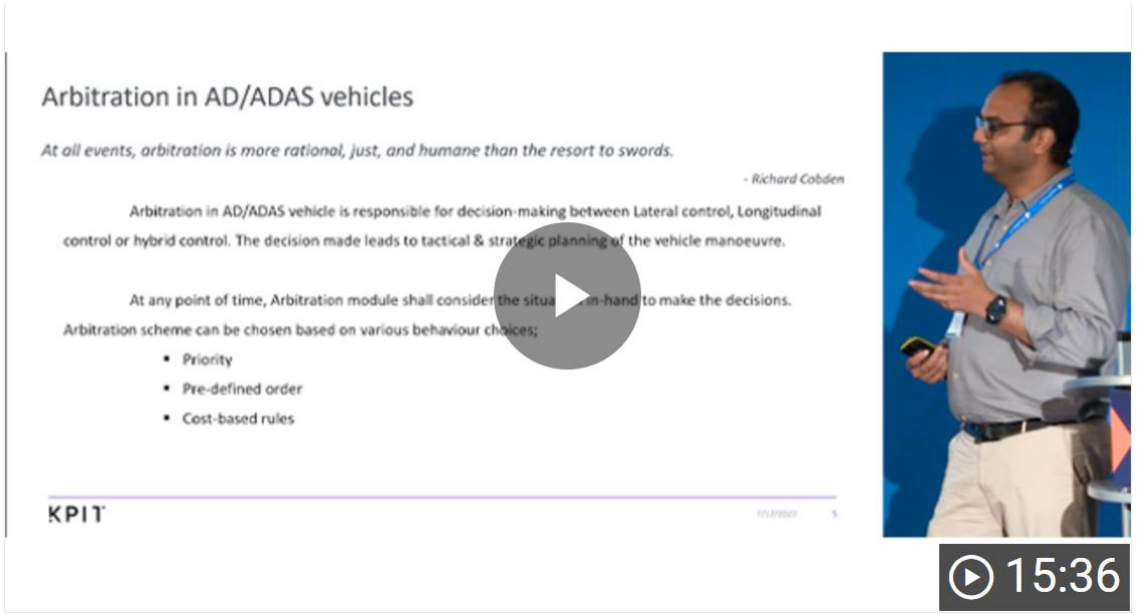
Visualize Sequence of Service Calls Using Sequence Viewer



https://www.mathworks.com/help/systemcomposer/ref/sequenceviewertool.html

User Presentations on SOA

[Zeekr - Using Model-Based Design to Develop SOA Applications for In-Vehicle OS]



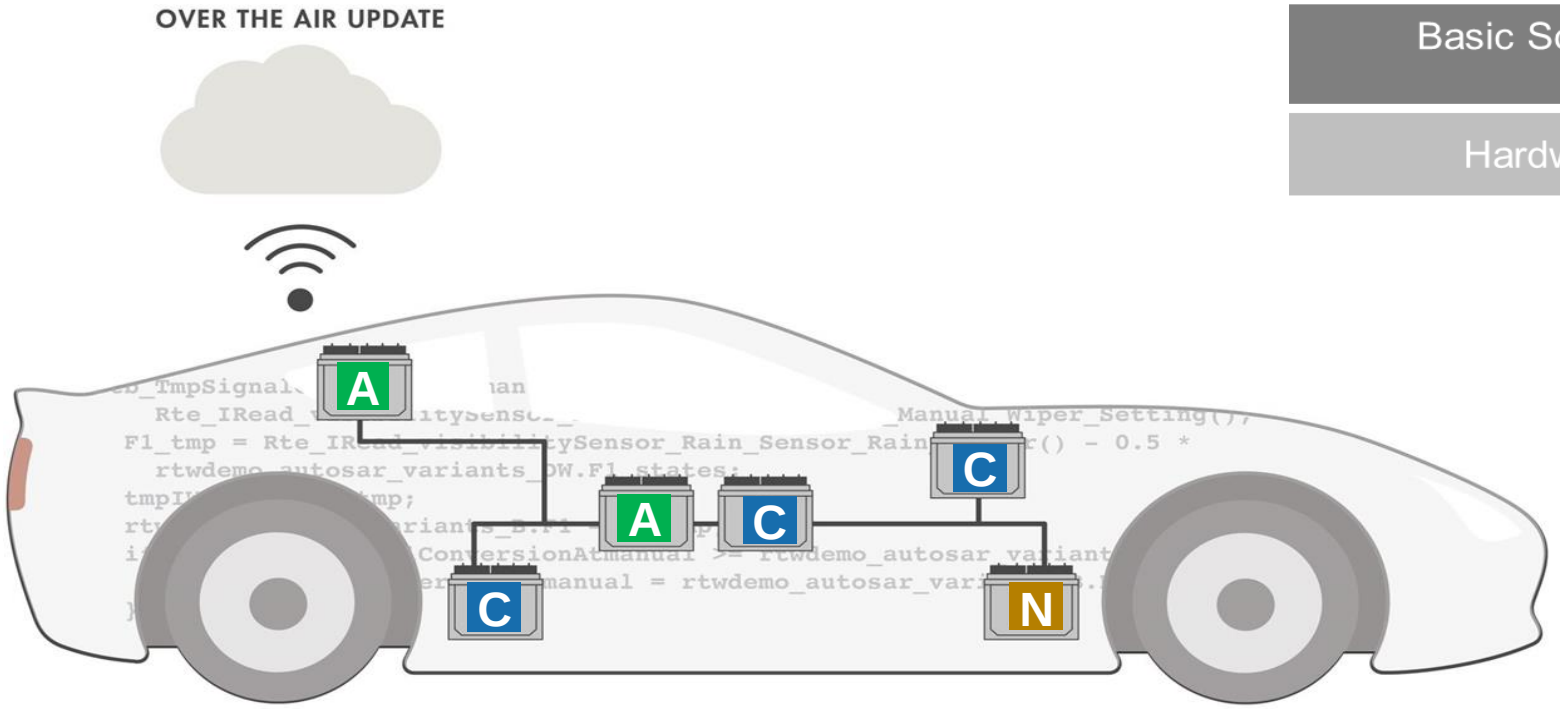
[KPIT - Service-Oriented Arbitration of ADAS Features with Model-Based Design]

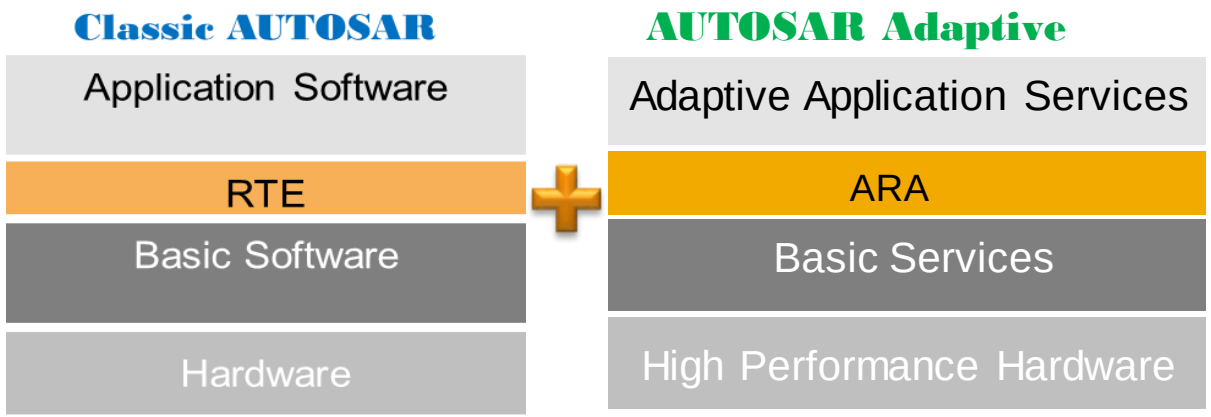
Agenda

- Advanced Simulink semantics for the development of services
- Software architecture modeling
  - AUTOSAR Classic – signal based
  - AUTOSAR Adaptive – services based
- Conclusions and key takeaways

AUTOSAR Classic & Adaptive

AUTOSAR® (**AUT**omotive **O**pen **S**ystem **AR**chitecture) is an open and standardized automotive software architecture





N Non - AUTOSAR

C Classic - AUTOSAR

A Adaptive - AUTOSAR

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

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

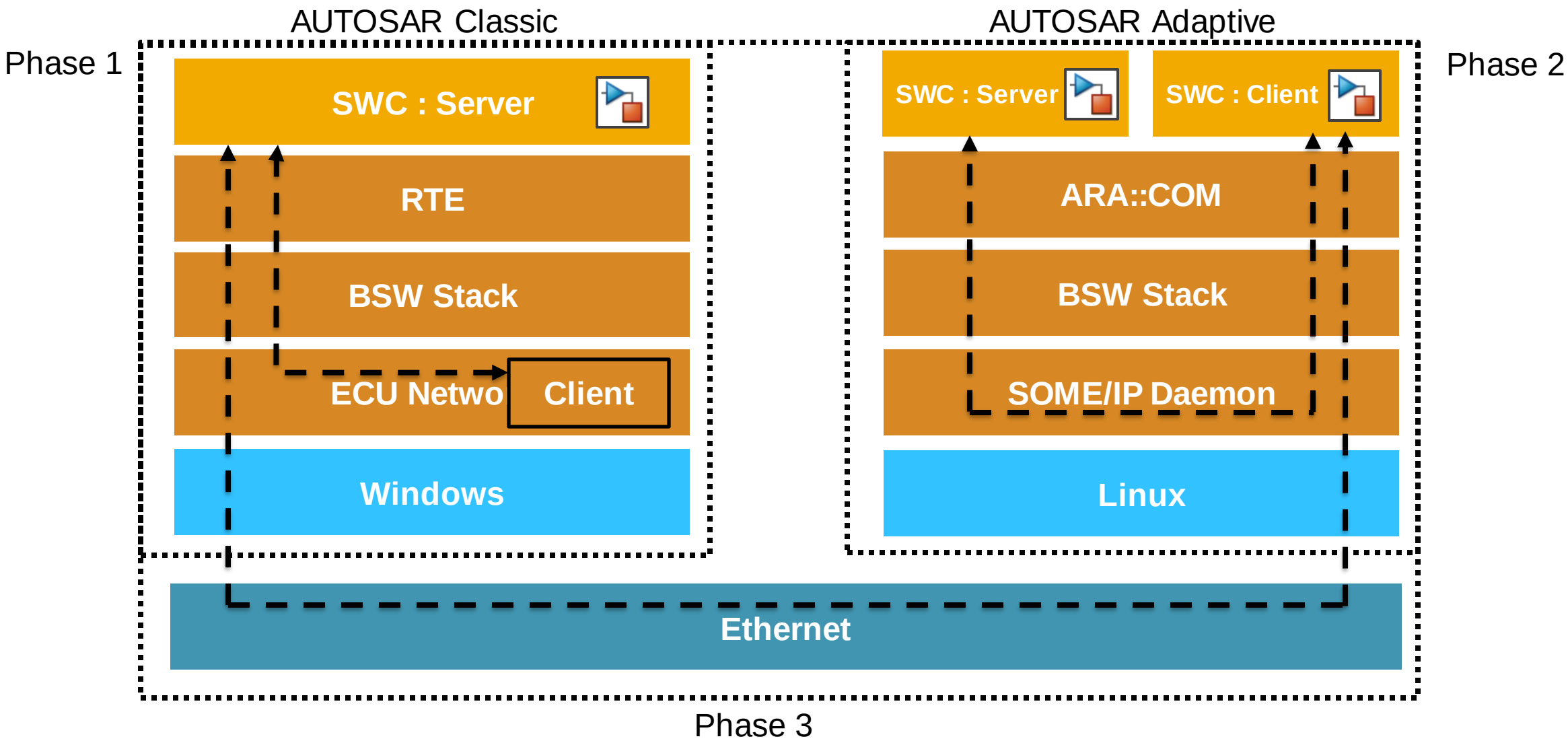
Shwetha Bhadravathi Patil
Technical Product Marketing
MathWorks
Natick, MA, United States
shwethap@mathworks.com

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

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

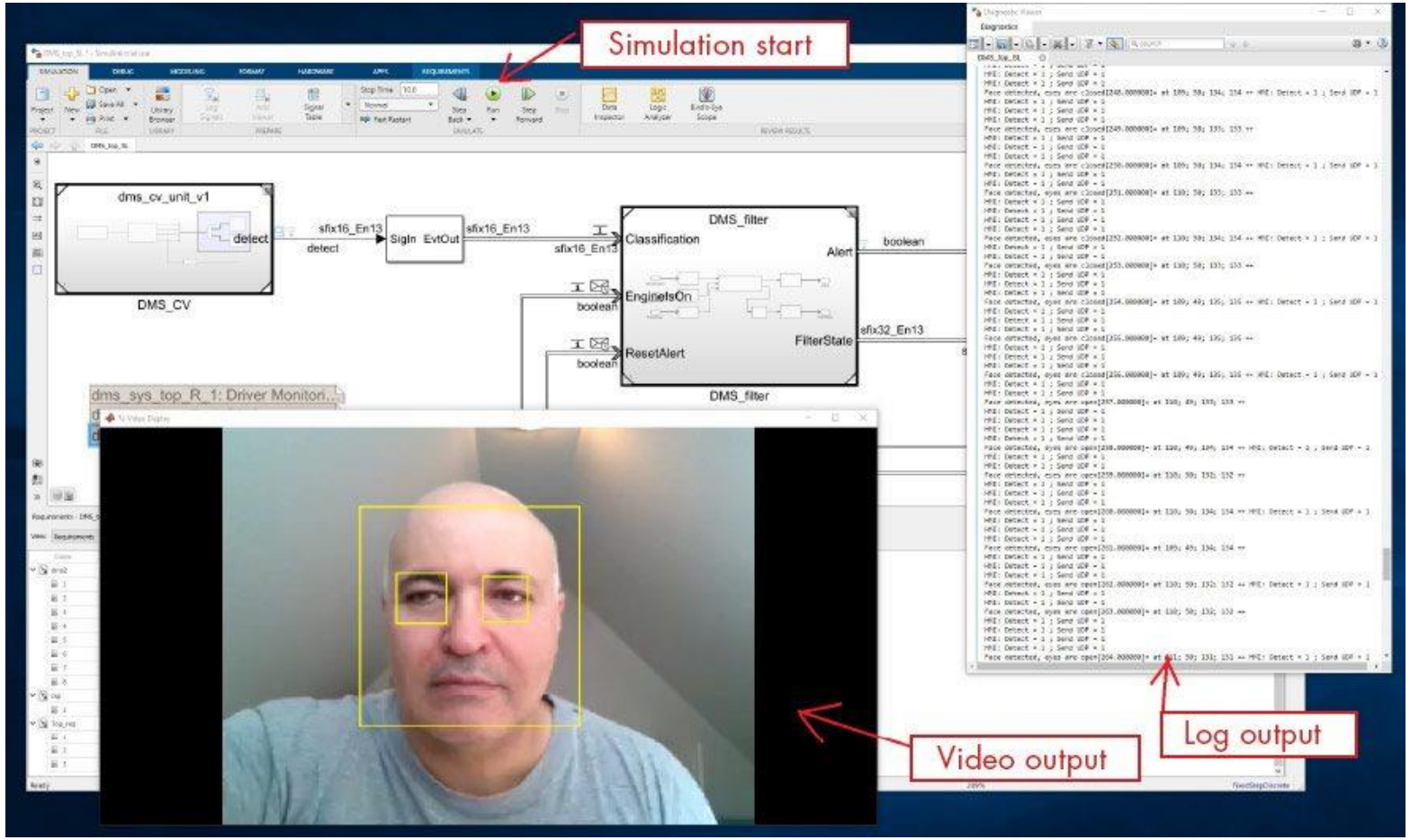
https://www.mathworks.com/content/dam/mathworks/conference-or-academic-paper/develop-and-integrate-autosar-classic-and-adaptive-applications-based-on-some-ip.pdf

Structure of Proof-of-Concept



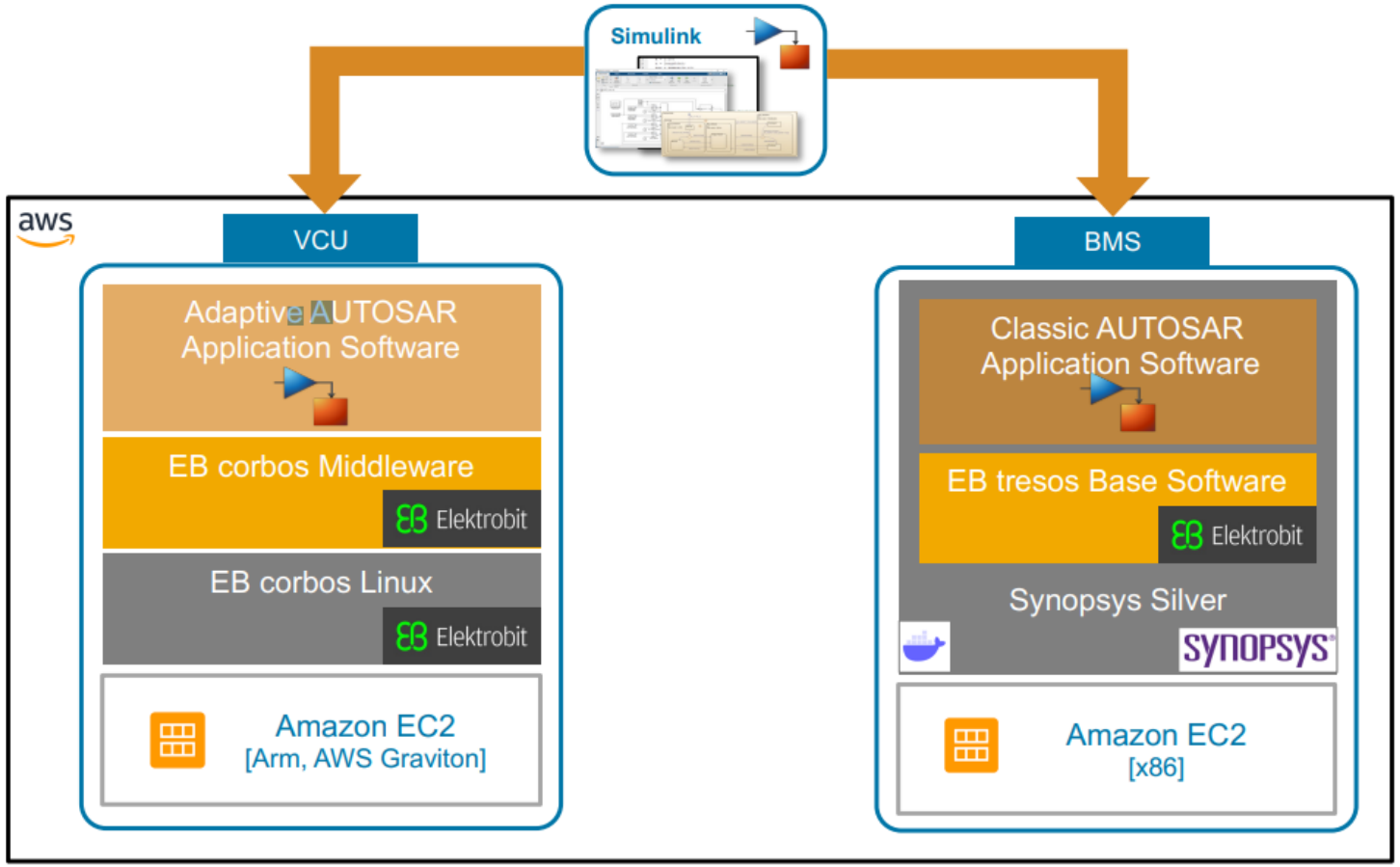
AUTOSAR Adaptive User Article

Elektrobit - Developing AUTOSAR Adaptive Software for a Driver Monitoring System with Model-Based Design

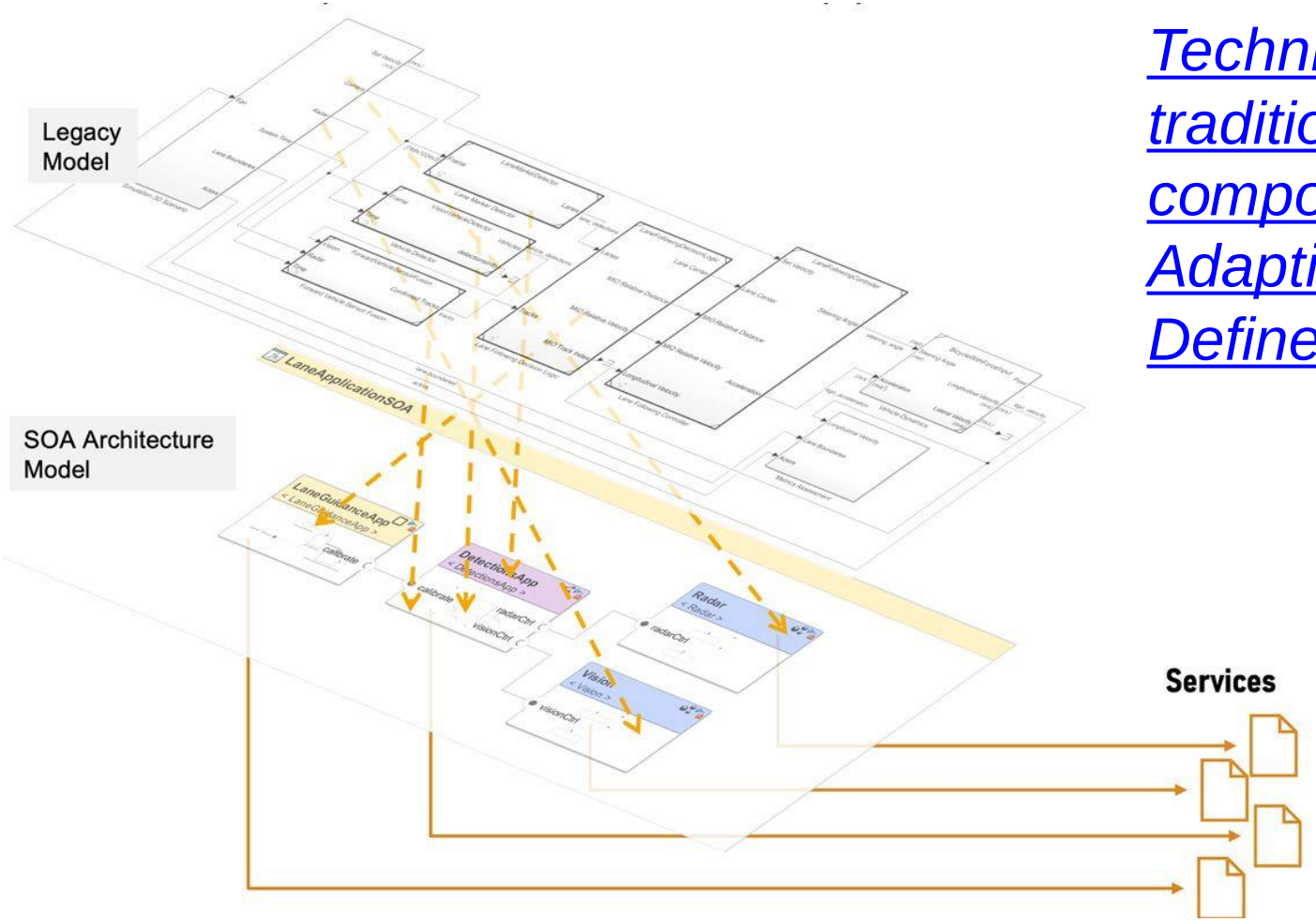


"Model-Based Design could accelerate development of end-to-end, AUTOSAR Adaptive software systems"

Cloud Deployment workflow with MBD

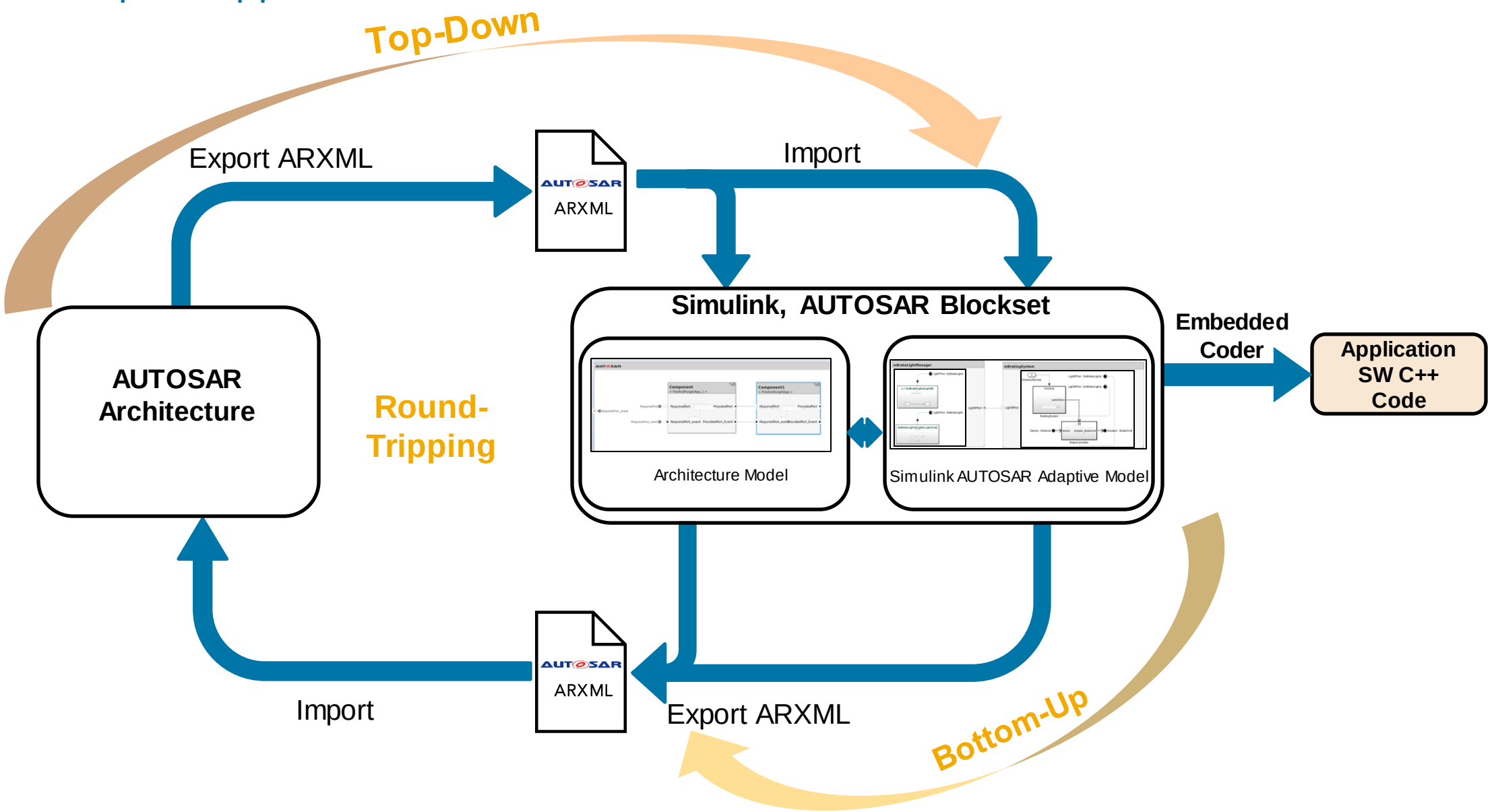


MathWorks presentation at AUTOSAR Open Conference 2023, May 11-12, San Diego, USA

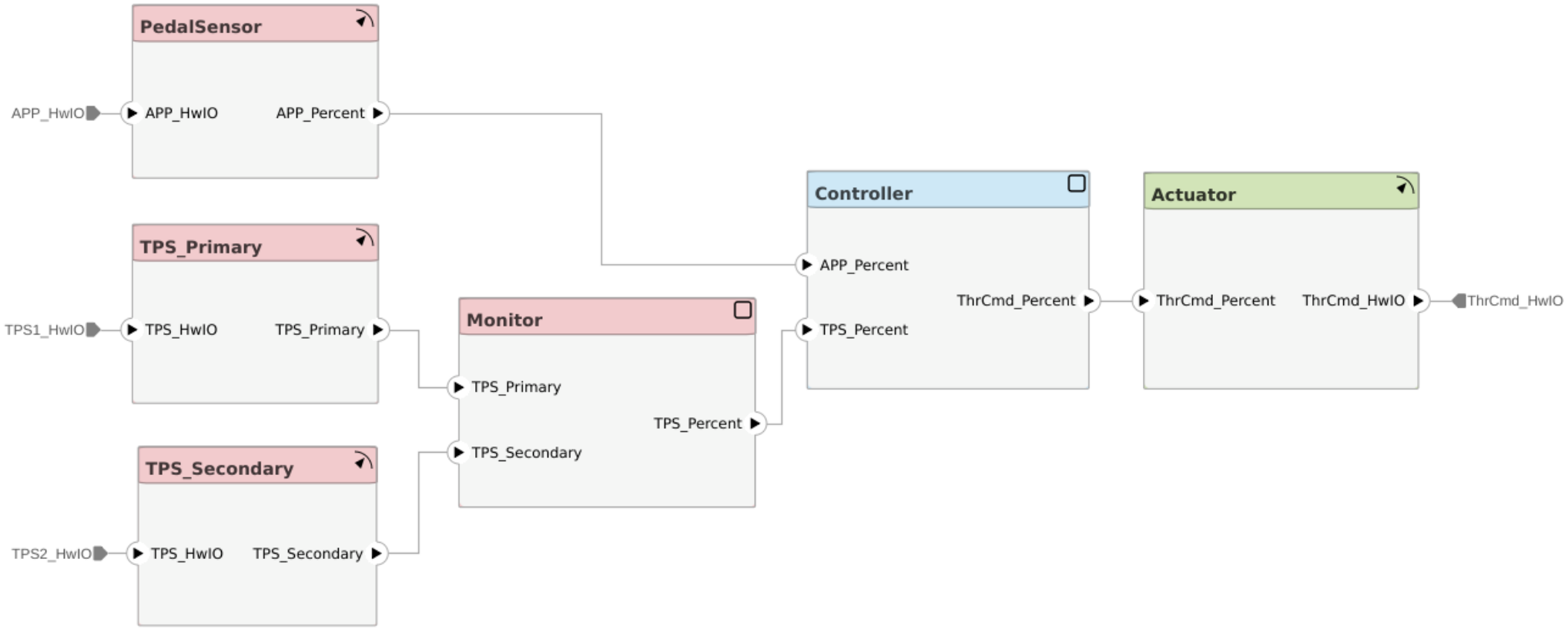


Technical Article - Migrating traditional automotive application compositions to AUTOSAR Adaptive services for Software Defined Vehicles

AUTOSAR Workflows - Importing and Exporting AUTOSAR Descriptions for Classic and Adaptive applications



Let's build the next AUTOSAR **Classic** based throttle position control system using System Composer and AUTOSAR Blockset



System Composer Architecture Model Templates

Standard Architecture Model Template

- Generic architecture design
  - Logical, functional, physical
  - Can have physical interfaces
  - Can compose software and AUTOSAR nodes in the architecture

Software Architecture Model Template

- Generic software architecture design
  - Can have client/server interfaces
  - Functions Editor

AUTOSAR Architecture Model Template

- AUTOSAR software architecture design
  - “is-a” software arch model
  - Purpose built canvas and resources for AUTOSAR
  - Out of the box support for **Classic** and **Adaptive** platforms

System Composer Architecture Model Templates

Standard Architecture Model Template

Software Architecture Model Template

AUTOSAR Architecture Model Template

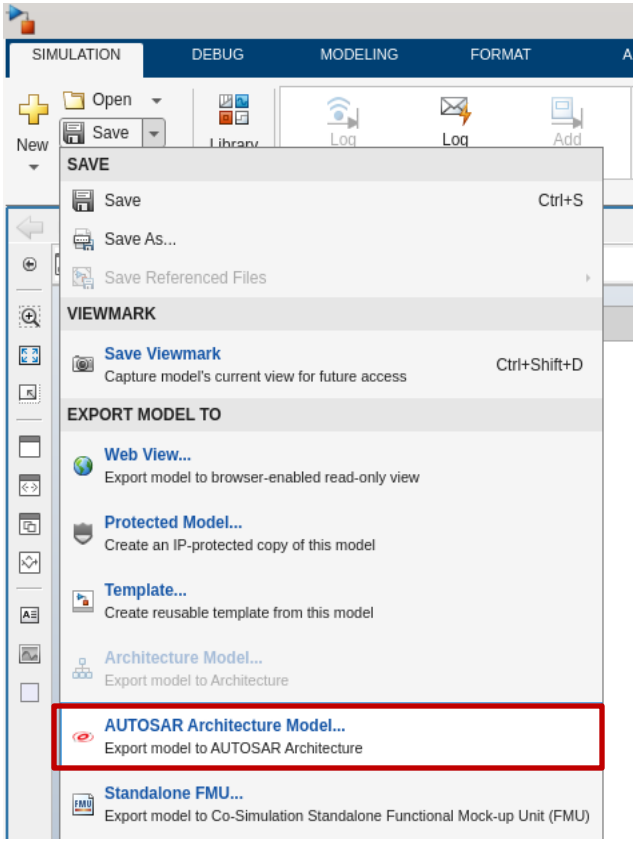
Export

- We support 2 workflows
  - Model AUTOSAR architectures directly
  - Export existing architecture model to AUTOSAR

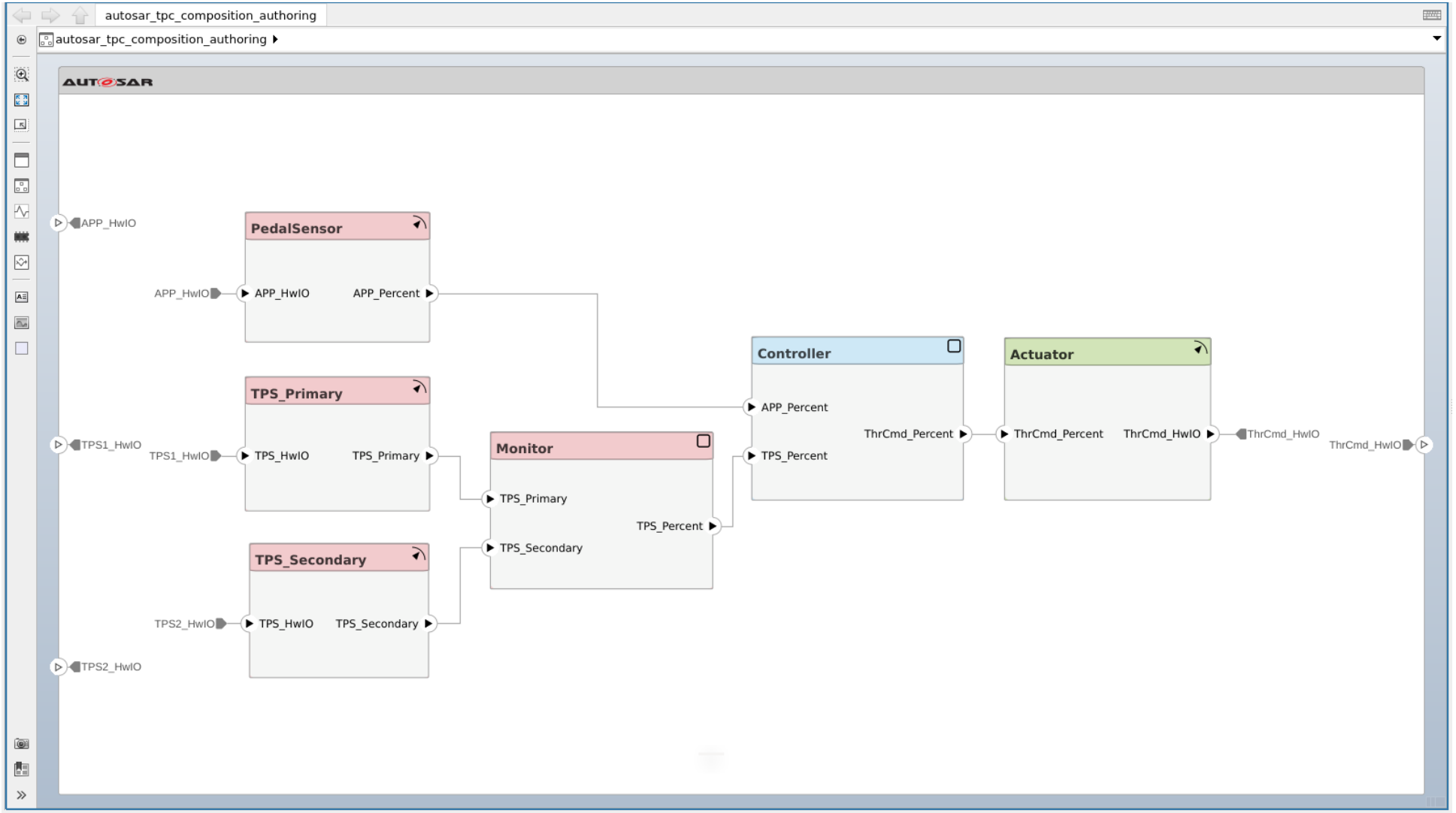
Let's use this workflow

R2019b

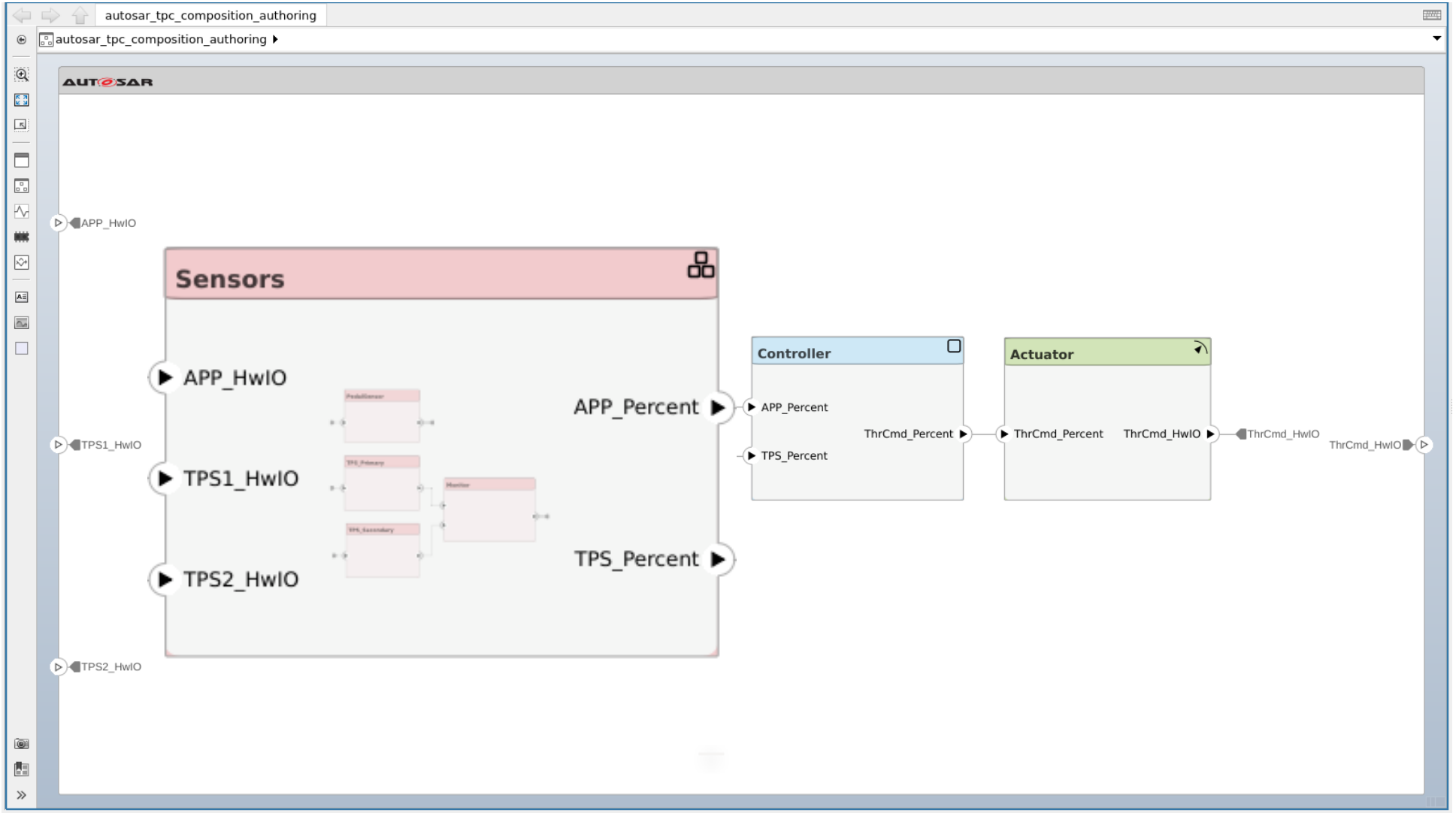
R2023b



Use Composition blocks to manage complexity

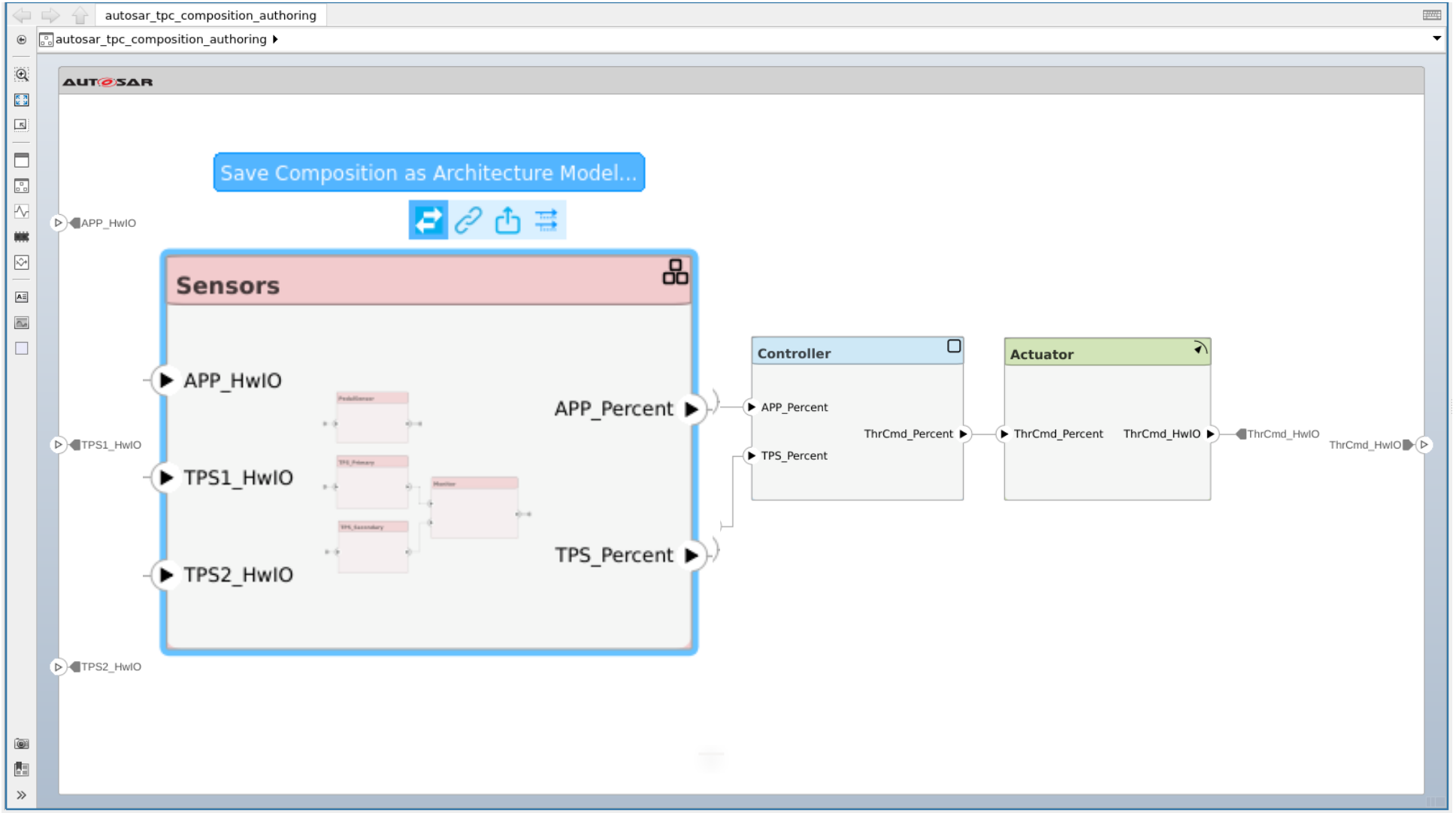


Use Composition blocks to manage complexity

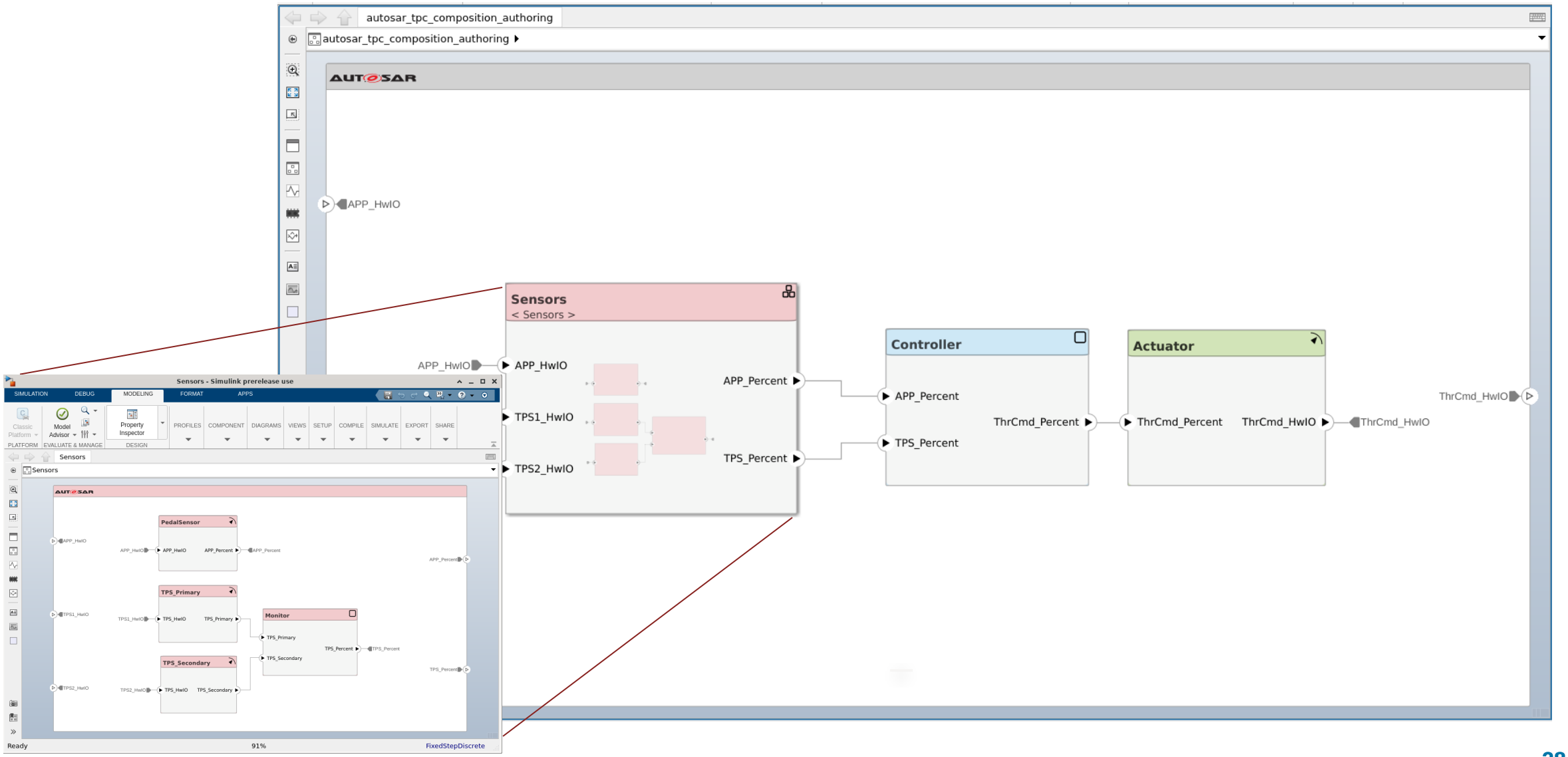


Save composition block as a separate AUTOSAR Architecture model

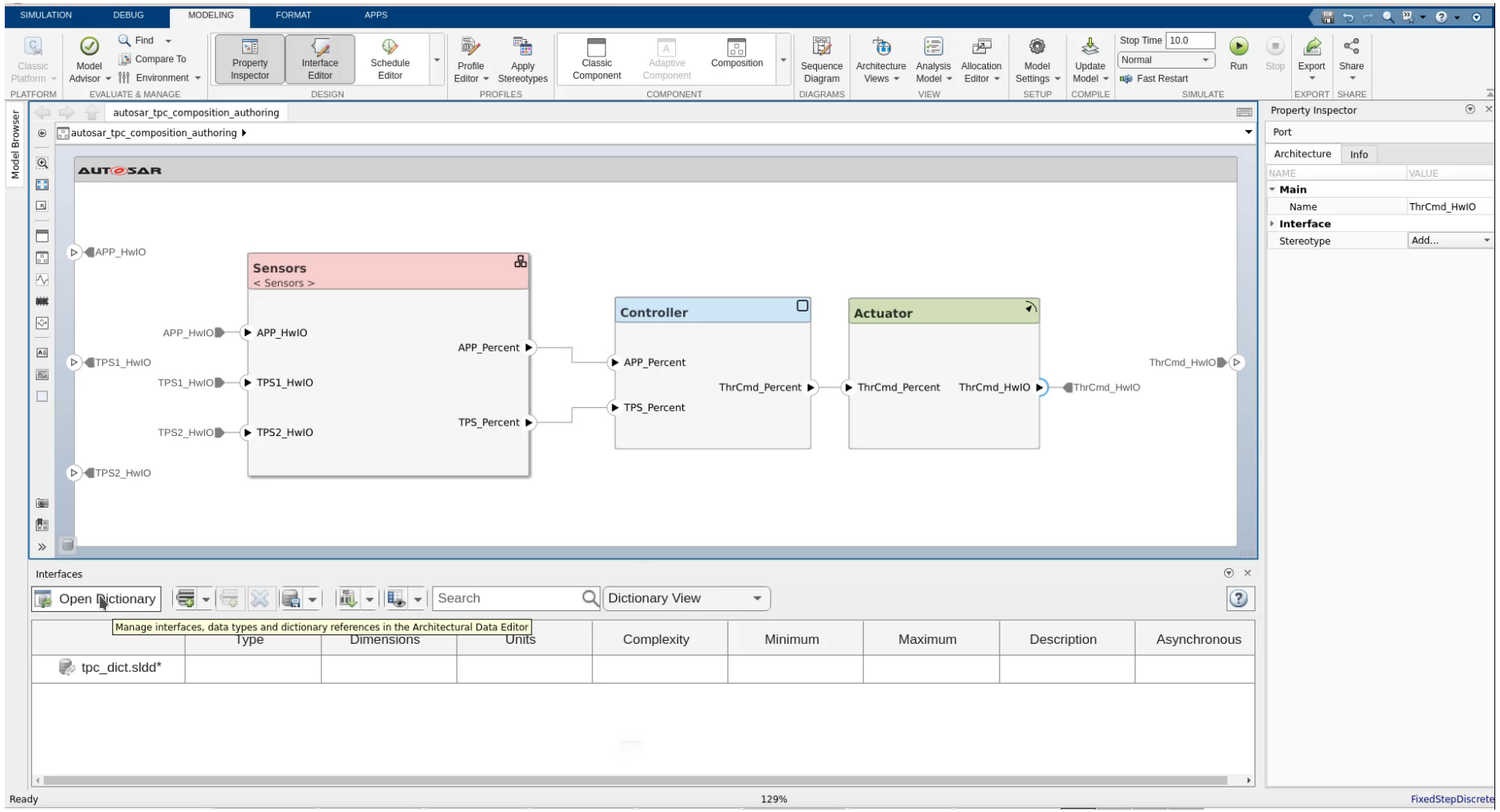
R2023b



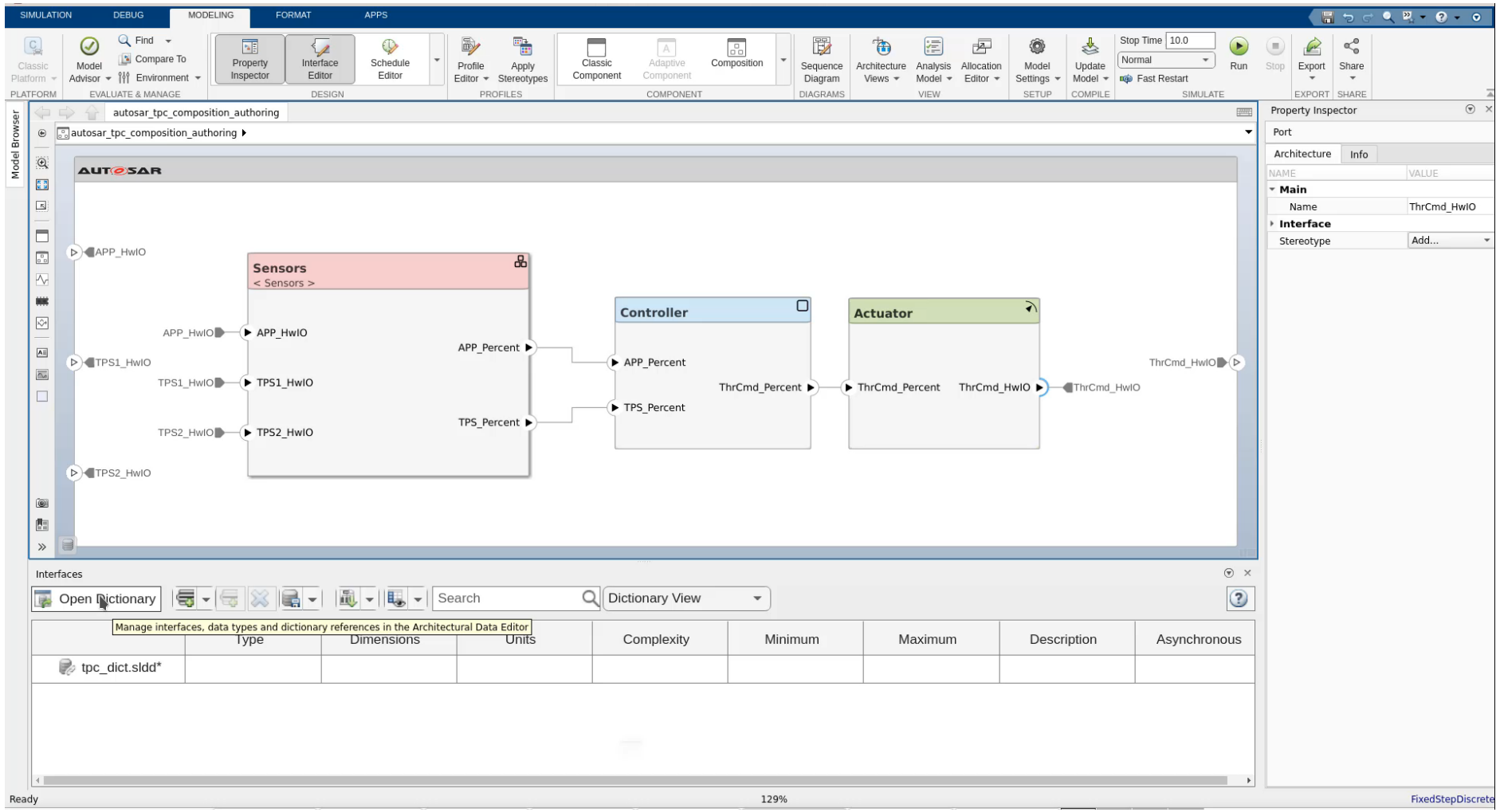
Save composition block as a separate AUTOSAR Architecture model



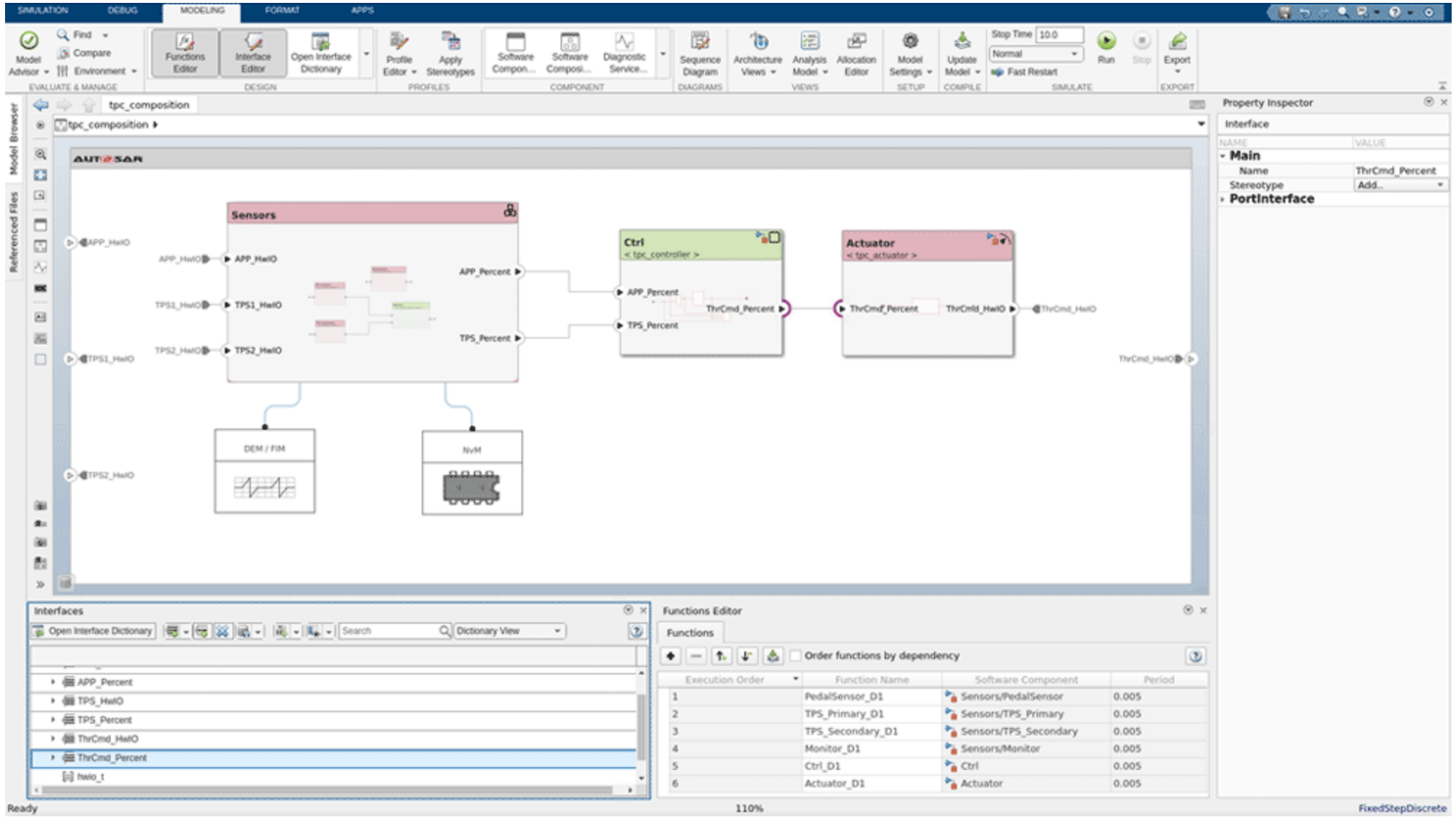
Use “Architectural Data” section in SLDD to manage interfaces and datatypes across architecture and algorithm models (Video)



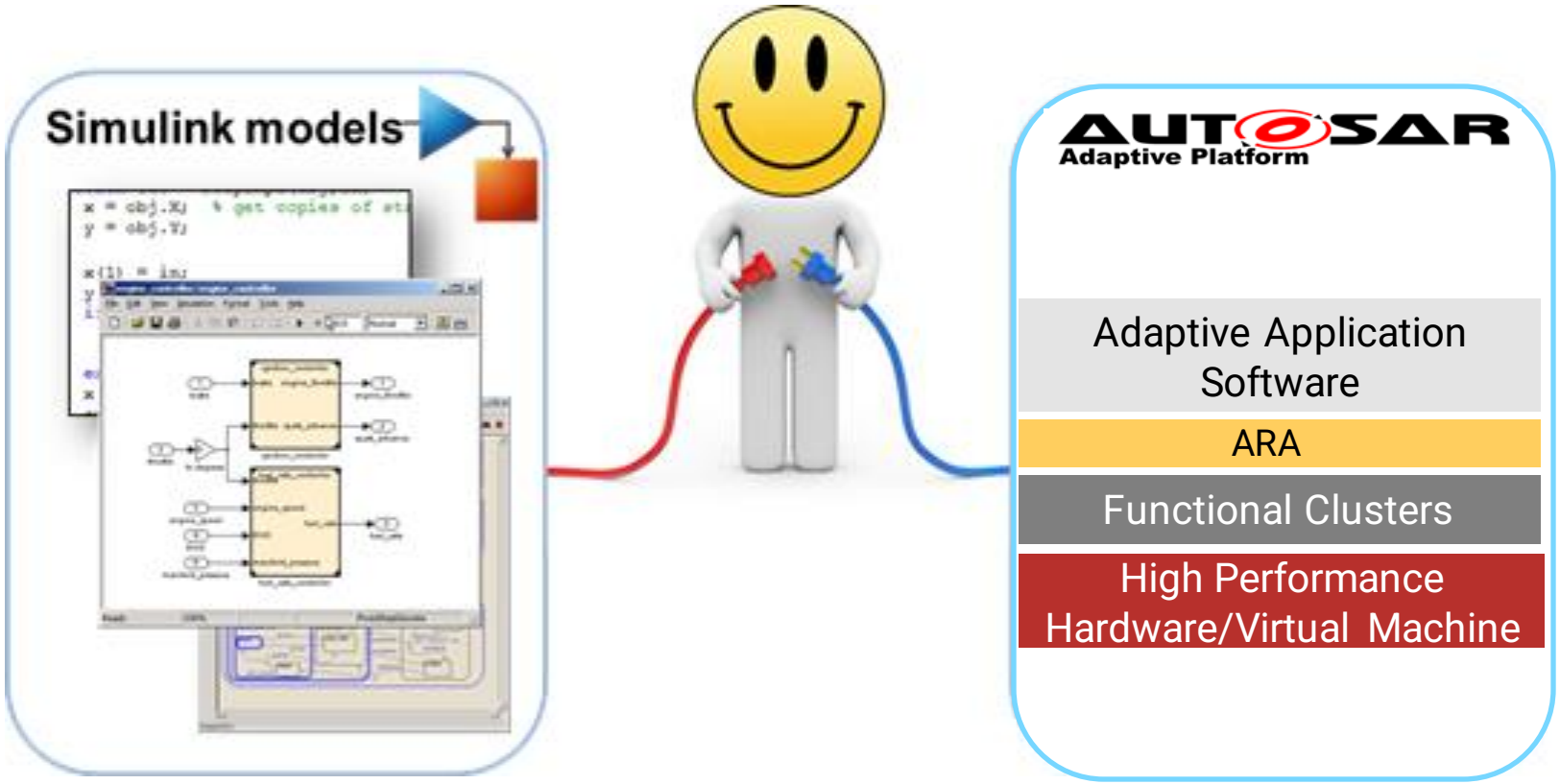
Use “Architectural Data” section in SLDD to manage interfaces and datatypes across architecture and algorithm models (Video)



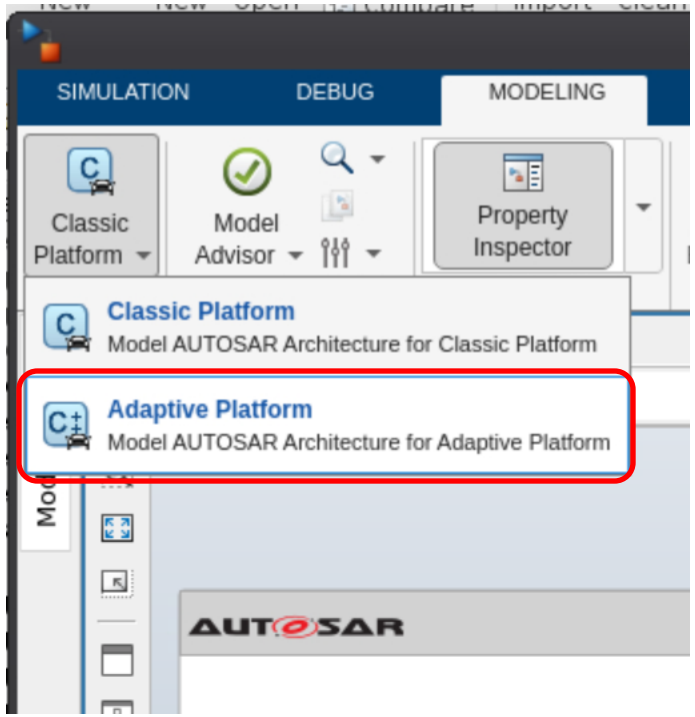
Simulate AUTOSAR Architecture

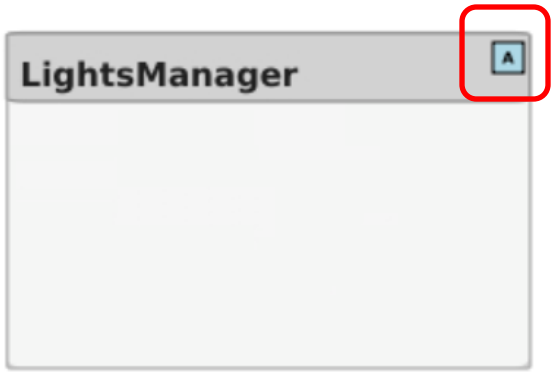


Plug your Simulink SOA models into the AUTOSAR Adaptive Platform

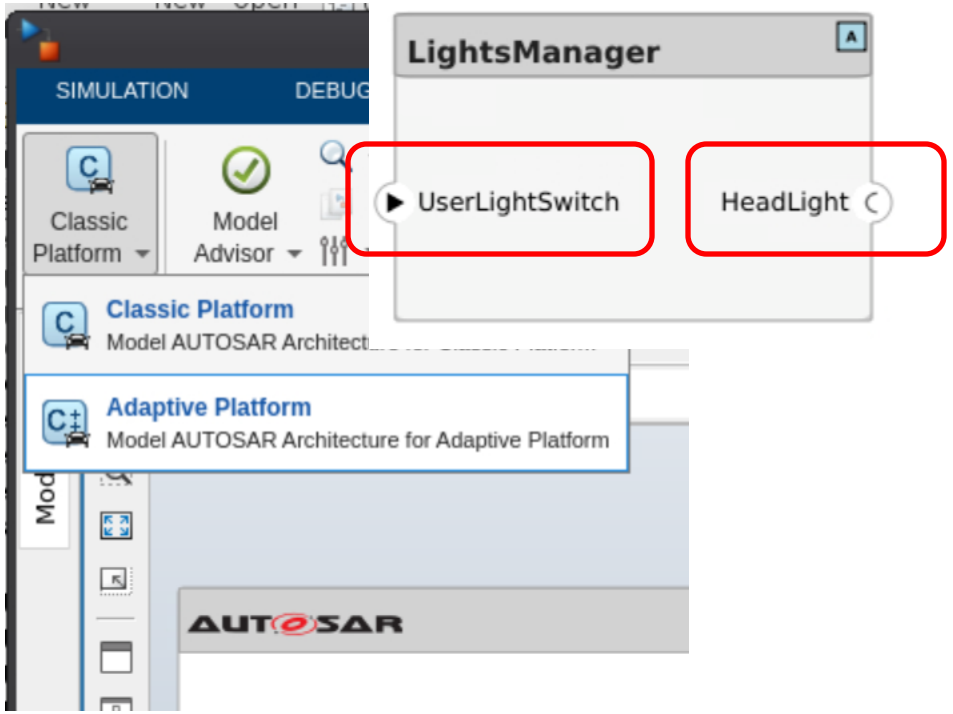


Develop Adaptive AUTOSAR Architectures

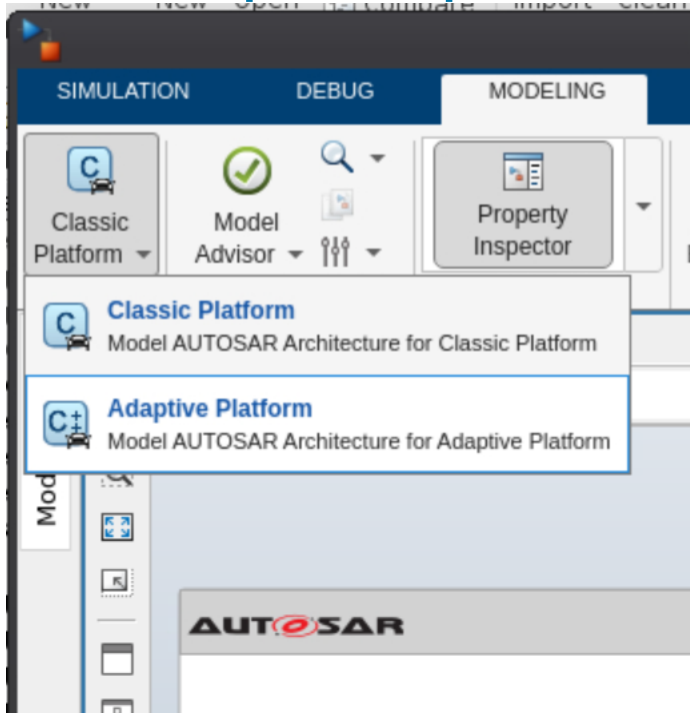


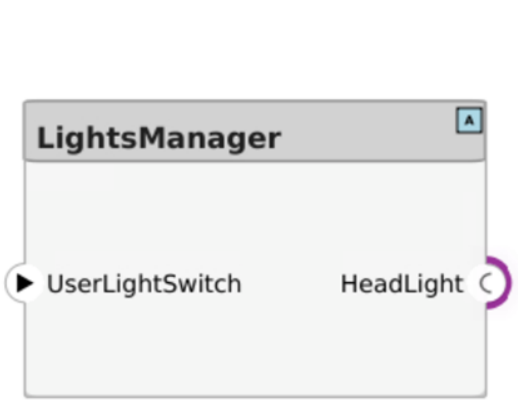


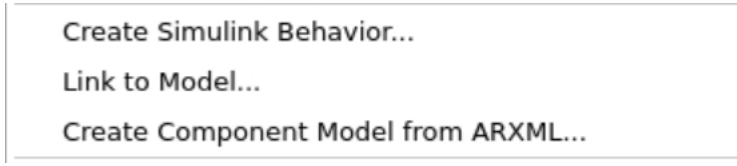
Develop Adaptive AUTOSAR Architectures

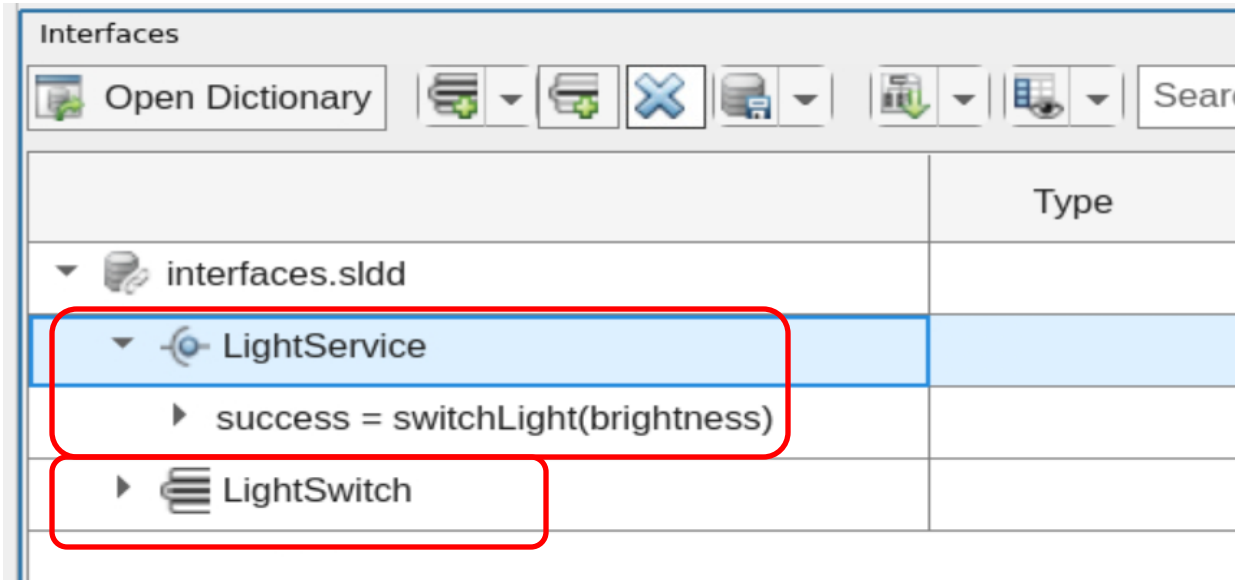


Develop Adaptive AUTOSAR Architectures

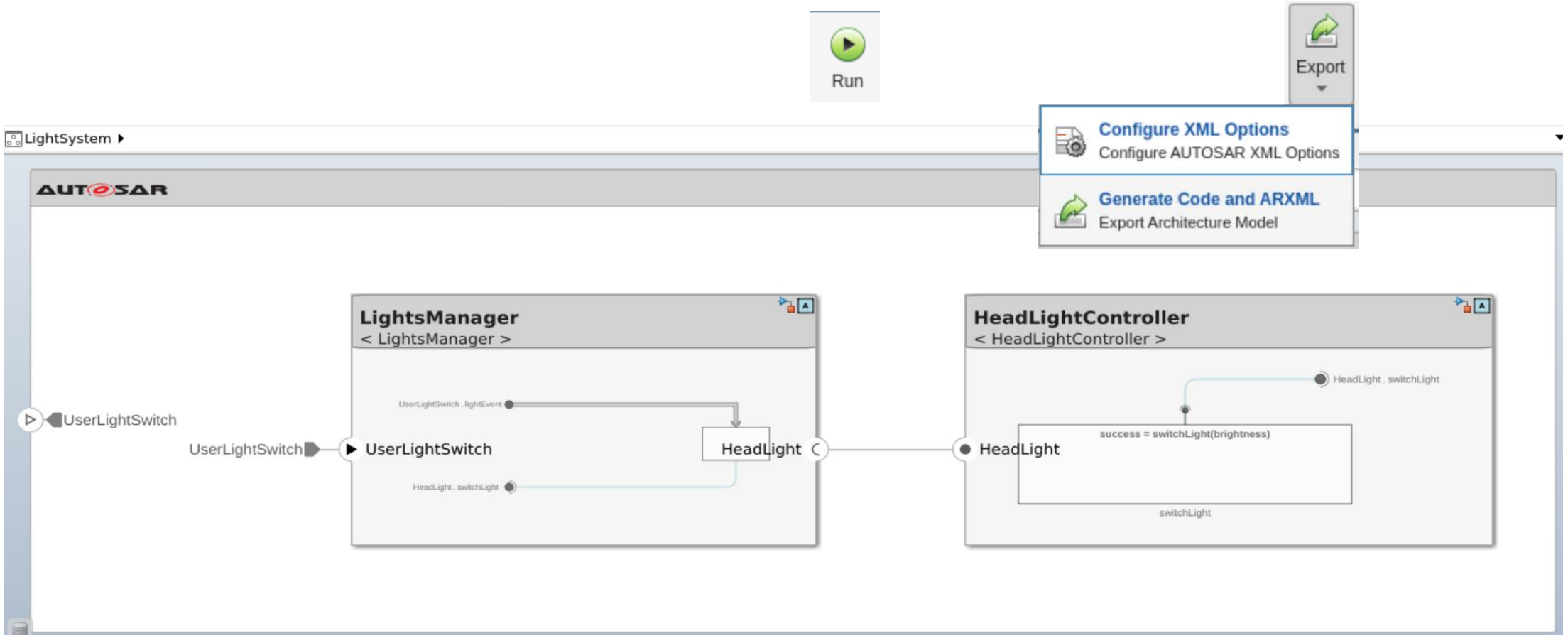




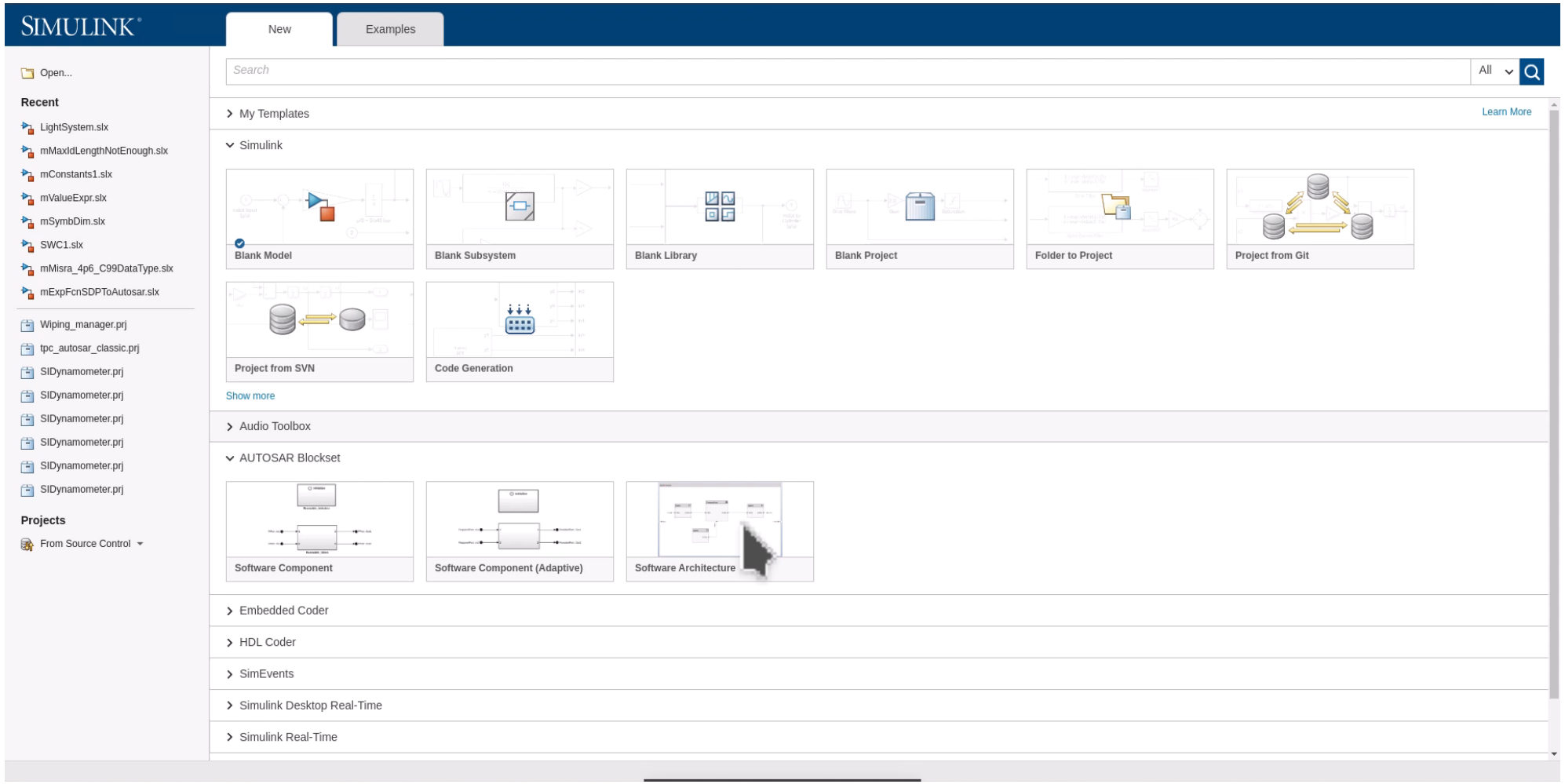




Export your Adaptive Architecture Models



Model Adaptive AUTOSAR Architectures in System Composer



Generated Code and ARXML for your system ready for deployment

- HeadLightController_autosar_adaptive
- LightsManager_autosar_adaptive

Editor - LightsManager.cpp | Project - sl4ap

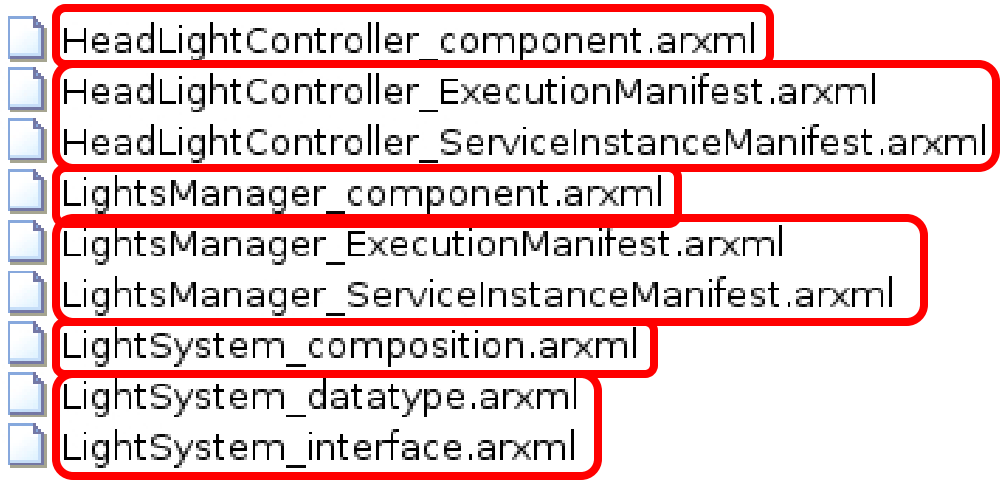
LightsManager.cpp

```
// Model step function
void LightsManager::MessageTriggeredSubsystem()
{
  ara::core::Future<proxy::methods::SwitchLight::Output> switchLightFuture;
  if (UserLightSwitch) {
    std::shared_ptr<ara::core::Result<size_t>> resultPtr;
    resultPtr = std::make_shared< ara::core::Result<size_t> >
      (UserLightSwitch->lightEvent.GetNewSamples(std::move(std::bind
         (&LightsManager::UserLightSwitchlightEventReceive, this, std::
          placeholders::_1)), 1U));
    if ((resultPtr->ValueOrThrow() > 0U) && HeadLight) {
      std::shared_ptr<ara::core::Result<proxy::methods::SwitchLight::Output>>
        switchLightResultPtr;

      // Received new event data
      // Outputs for Function Call SubSystem: '<Root>/Message Triggered Subsystem'
      // FunctionCaller: '<S1>/switchLight Caller'
      switchLightFuture = HeadLight->switchLight(LightsManager_B.Trigger);

      // End of Outputs for SubSystem: '<Root>/Message Triggered Subsystem'
      // Retrieve result on method switchLight's completion
      switchLightResultPtr = std::make_shared< ara::core::Result<proxy::methods::
        SwitchLight::Output> >(switchLightFuture.GetResult());

      // Check if method switchLight completed successfully and returned valid results
      if (switchLightResultPtr->HasValue()) {
        // Retrieve return arguments from method switchLight's Result container
        switchLightResultPtr->Value();
      }
    }
  }
}
```

LightsManager.cpp | main.cpp | HeadLightController.cpp

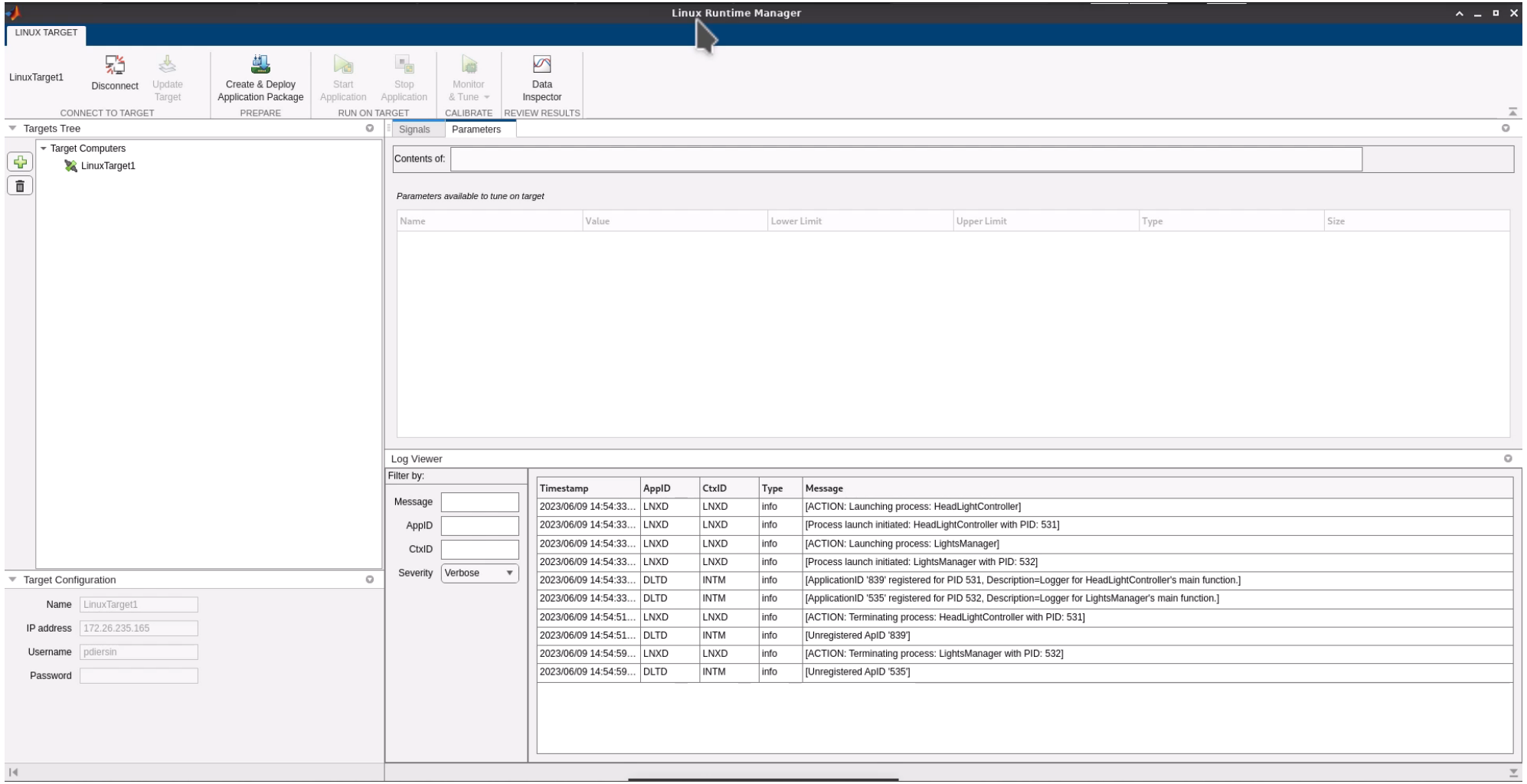
```
#include <stdbool.h>

// Model step function
void HeadLightController::switchLight(uint8_t brightness, bool *success)
{
  UNUSED_PARAMETER(brightness);

  // Outputs for Function Call SubSystem: '<Root>/switchLight'
  // SignalConversion generated from: '<S1>/success'
  *success = false;

  // End of Outputs for SubSystem: '<Root>/switchLight'
}

// Model initialize function
void HeadLightController::initialize()
{
  // Initialize service provider instance - HeadLight
  HeadLight = std::make_shared< skeleton::LightServiceSkeletonImpl >(ara::com::
    InstanceIdentifier(ara::core::StringView("1")), ara::com::
    MethodCallProcessingMode::kEventSingleThread);

  // Bind HeadLightController class implementation of switchLight method to service instance's implementation
  HeadLight->setFuncObjswitchLight(std::bind(&HeadLightController::switchLight,
    this, std::placeholders::_1, std::placeholders::_2));
  HeadLight->OfferService();
}
```

Generated Code and ARXML for your system ready for deployment



- Component Descriptions
- Composition Descriptions
- Data Types and Interfaces
- Deployment Configurations

Deploy Adaptive Architecture Models to Linux Targets



Conclusions

Challenges

- Development of SOA applications require a **change of mindset**
- **Centralize, re-architect** existing applications and partition in processes and services

Solutions

- **Design, simulate and generate** code to deploy Signal based AUTOSAR Classic and service-oriented AUTOSAR Adaptive in **Simulink**
- **Reuse your existing expertise and models** to mitigate the risk of migration to Adaptive applications