

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)

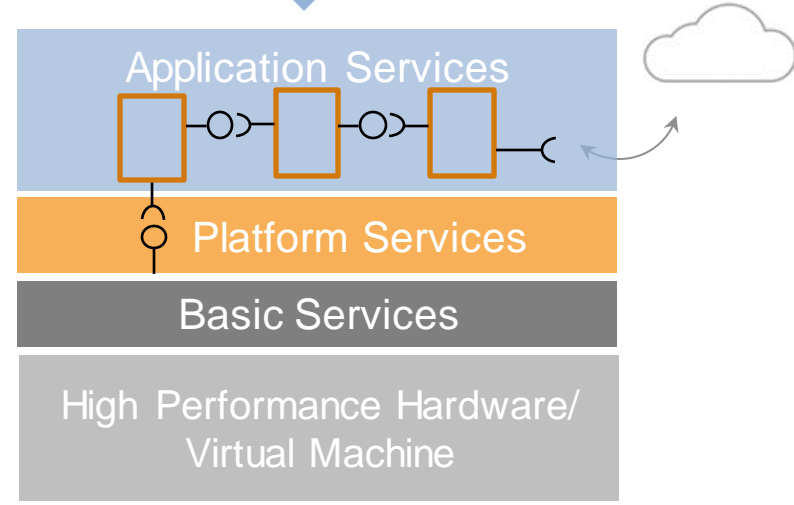


```

100110
001010
1001100010
001010
100110
001010
010010
    
```

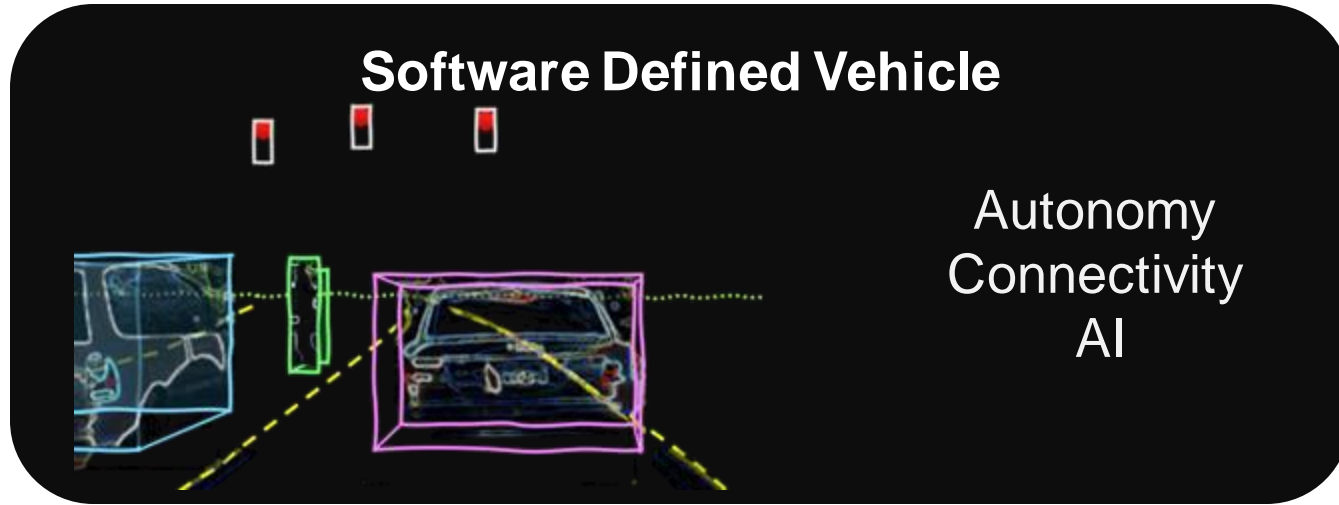


- SW updates
- Frequent
 - Selective
 - Over-the-air



Higher HW abstraction:
Service-oriented architectures

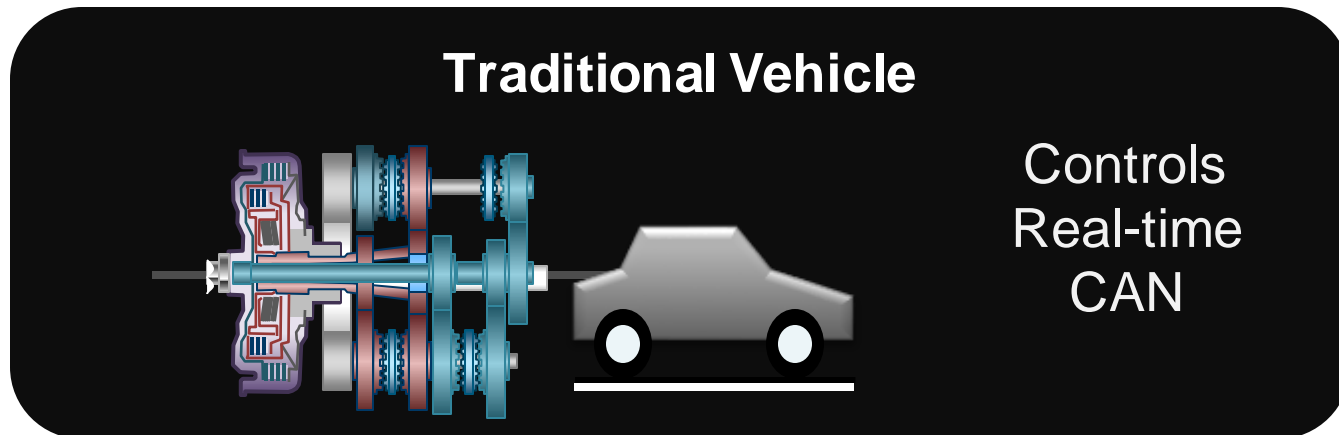
Automakers are increasingly building software in-house with SOA based design



Steering,
Braking



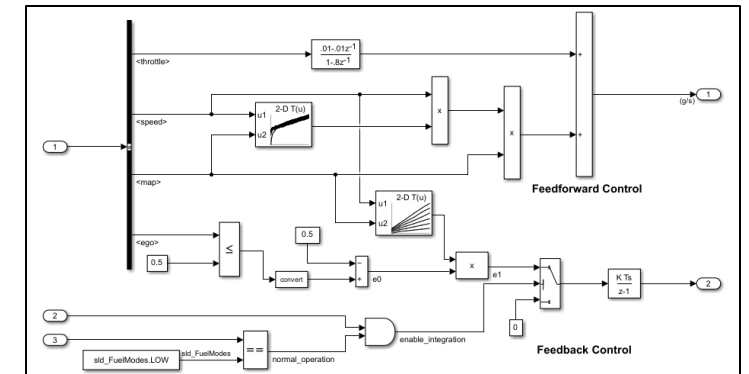
Speed,
Velocity



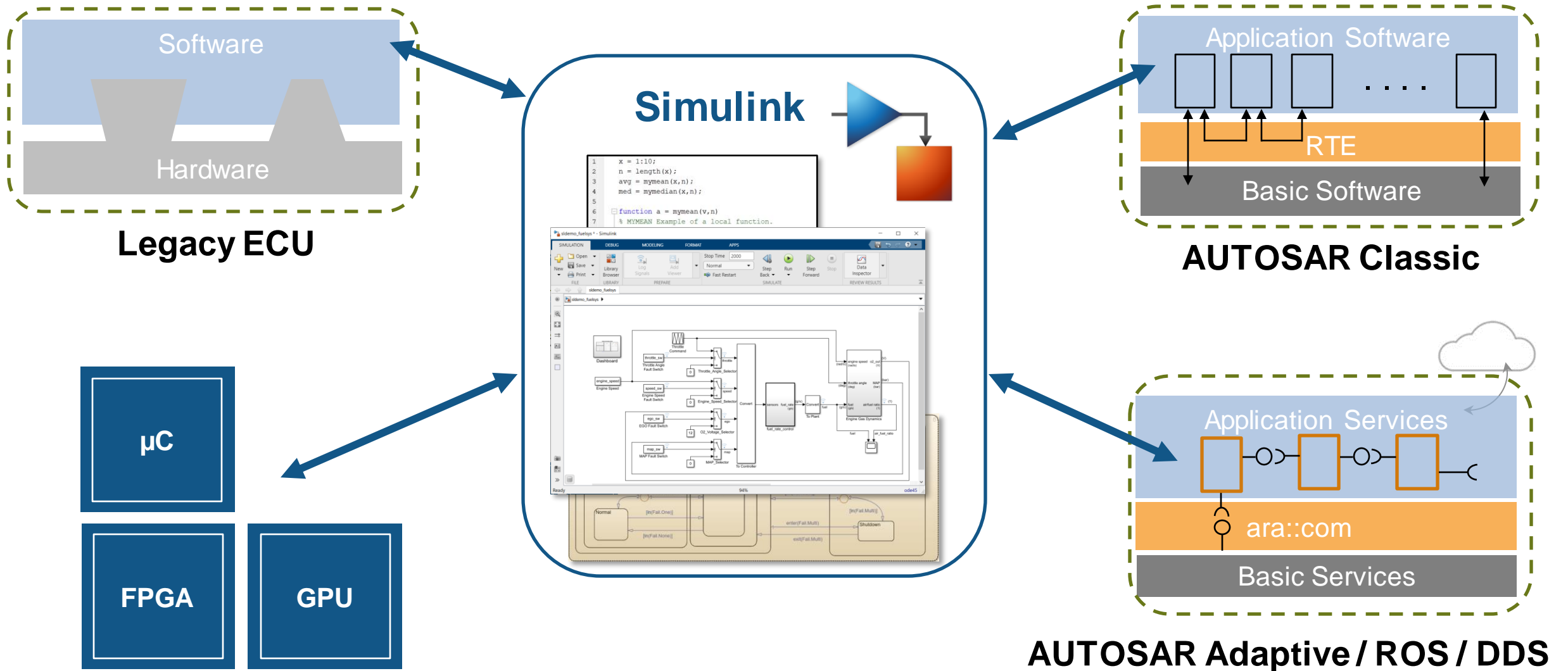
SOA based standards



Model-Based Design



Simulink: deploy software to different targets and standards



Agenda

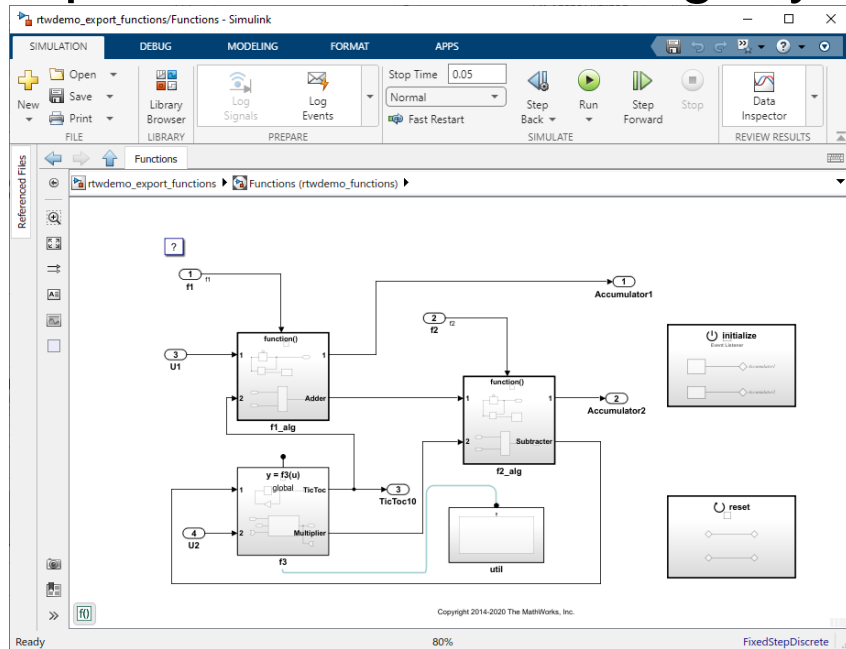
- Advanced Simulink semantics for the development of services
- Software architecture modeling
- Conclusions and key takeaways

Agenda

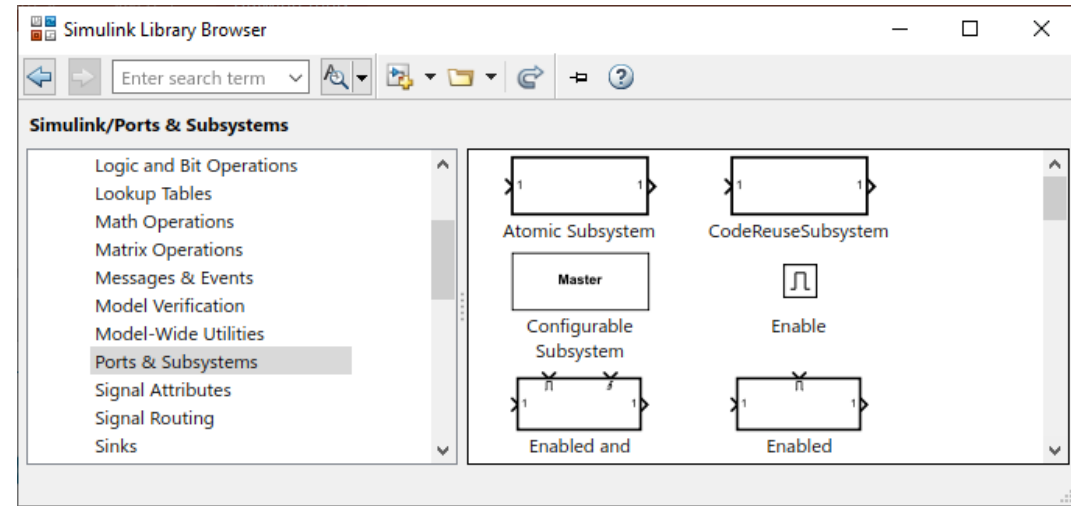
- **Advanced Simulink semantics for the development of services**
- Software architecture modeling
- Conclusions and key takeaways

Simulink Supports Exporting Callable Functions Well

- Export Function Modeling style

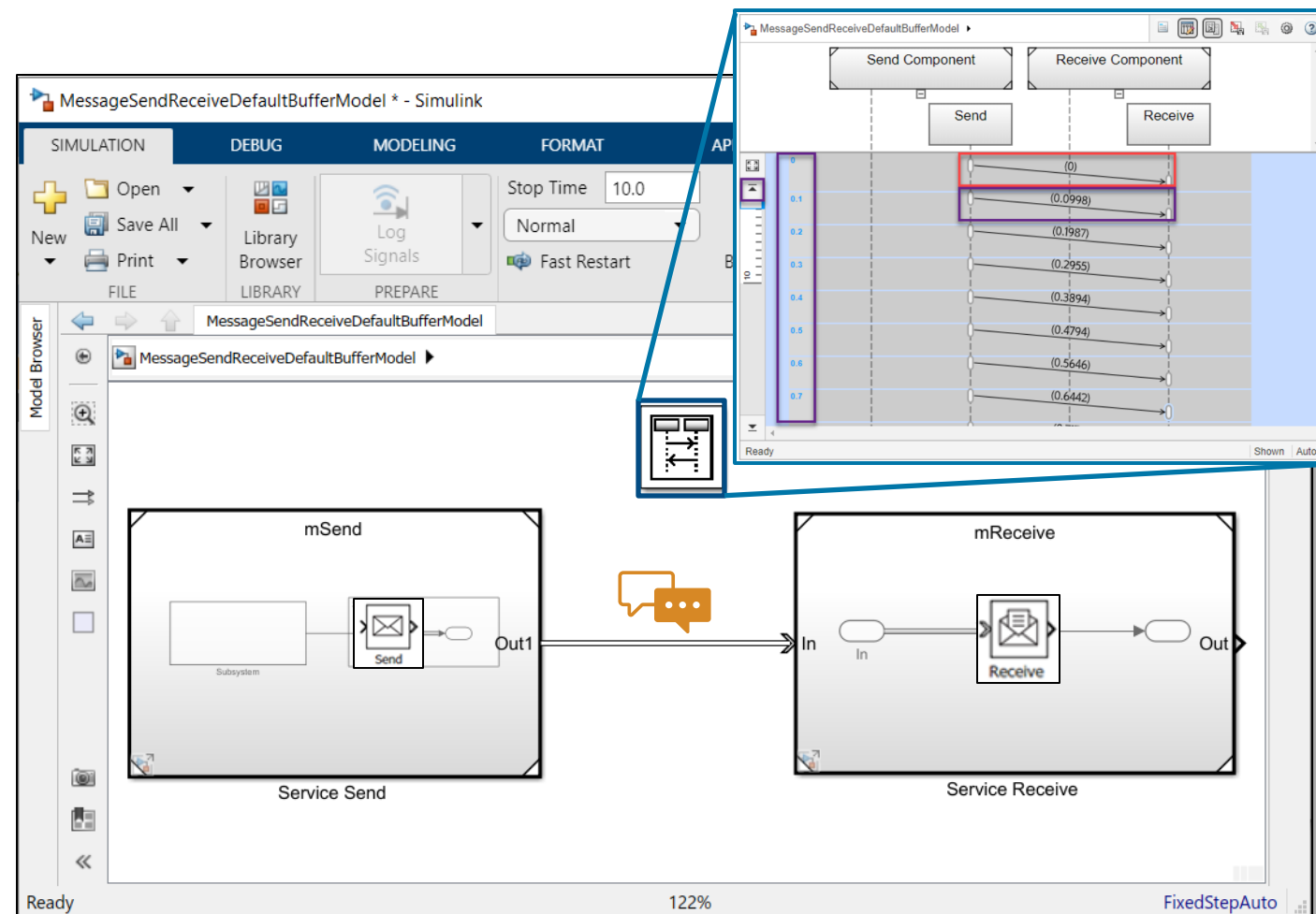
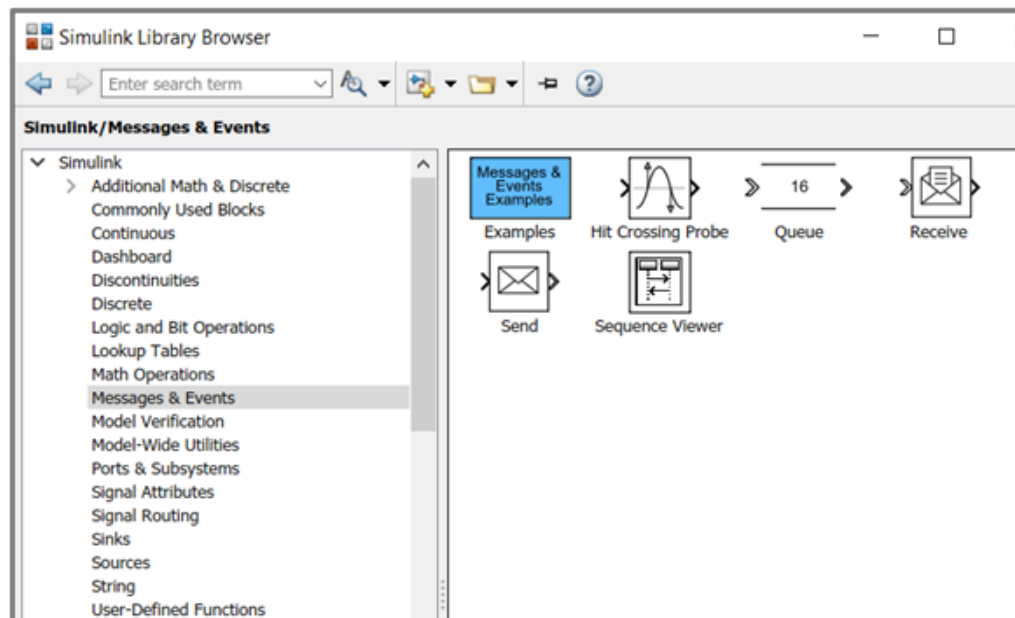
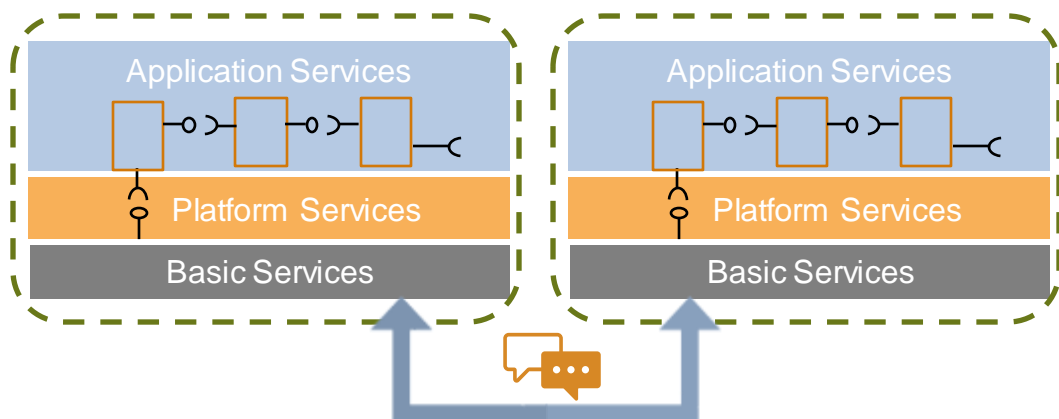


rtwdemo_export_functions



Ports and Subsystems

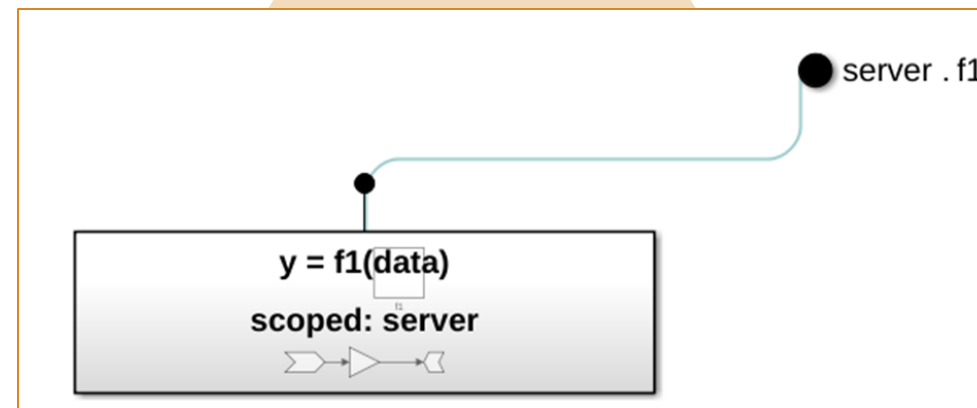
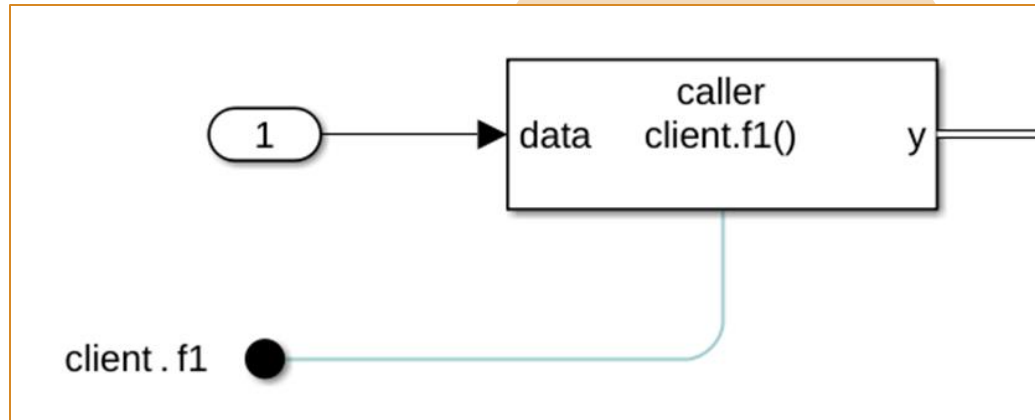
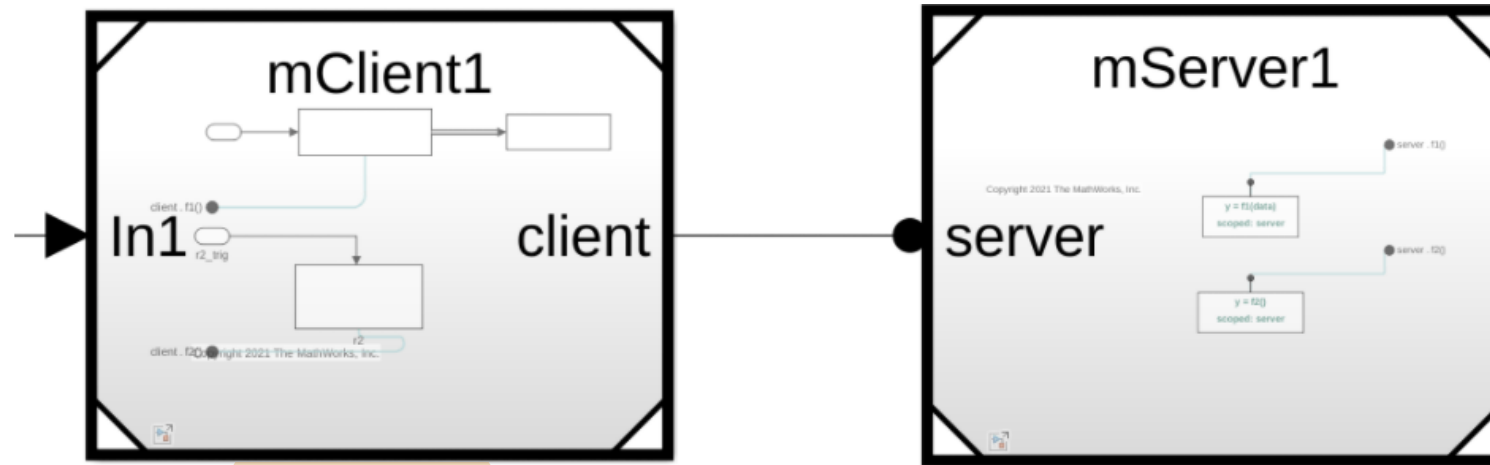
Service-Oriented Behavior Modeling with Simulink Messages



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

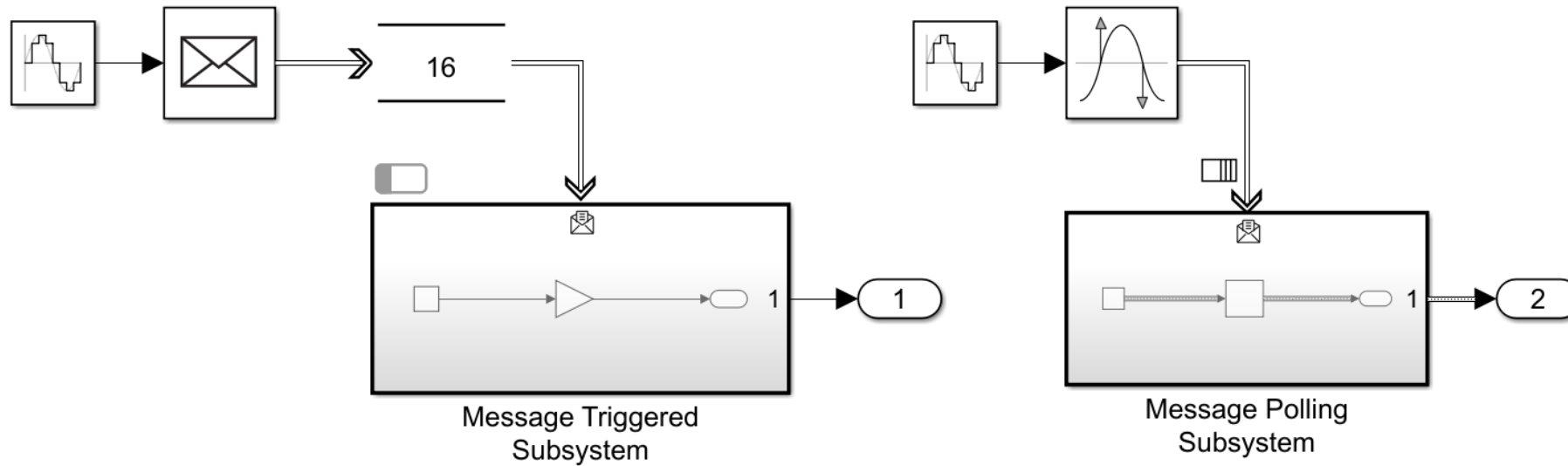
Service-Oriented Behavior Modeling with Service-Based Functions

Function Ports for SOA



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

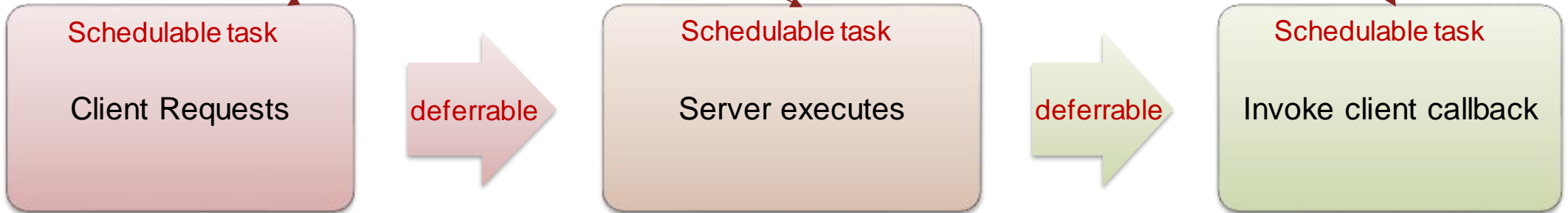
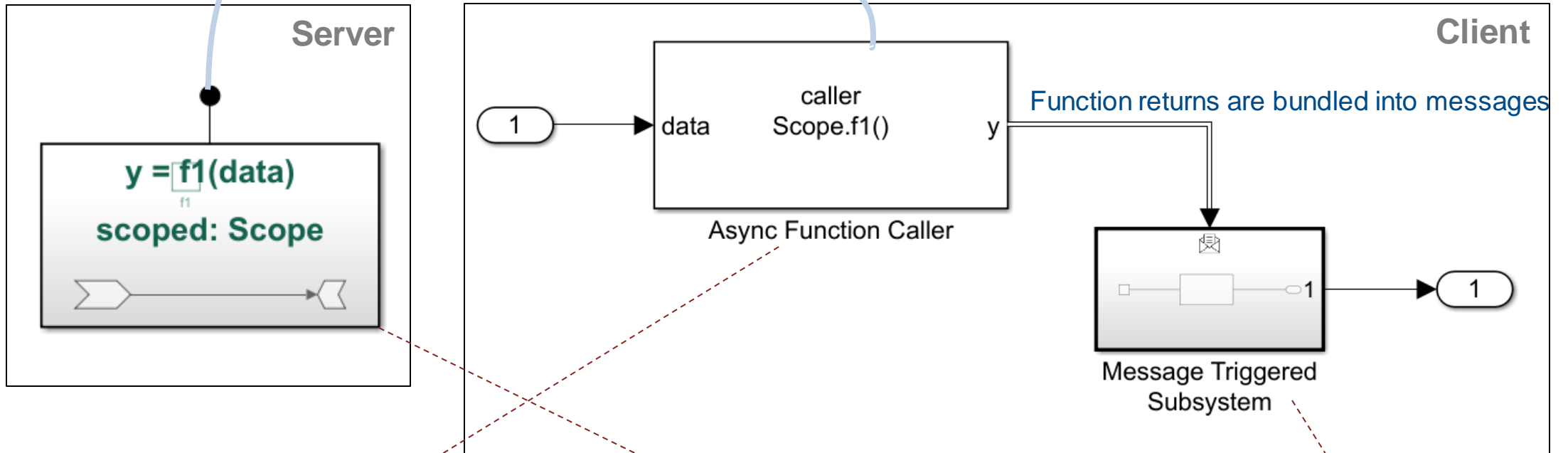
Service-Oriented Behavior Modeling with Message Triggered/Polling Subsystem



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

Service-Oriented Behavior Modeling with Service-Based Functions

Asynchronous Client-Server Function with Callback



Service-Oriented Behavior

SCHEDULE EDITOR

Manage Partitions | Order | Update Diagram | Save Model | Highlight | Arrange | Timing Legend | Layout

PARTITIONS EXECUTION MODEL DISPLAY VIEW

Events

Name
Client.ResponseCallback
Client.ResponseCallback1
Server.service.bar
Server.service.foo

```

graph TD
    subgraph TopFlowchart
        A[Server.service.foo] --> B[Client_FunctionCall]
        B --> C[Client.ResponseCallback]
    end
    subgraph BottomFlowchart
        D[Server.service.bar] --> E[Client_FunctionCall1]
        E --> F[Client.ResponseCallback1]
    end
    
```

Order

Order	Name	Trigger
1	Server.service.foo	Server.s
2	Server.service.bar	Server.s
3	Client_FunctionCall	
4	Client_FunctionCall1	
5	Client.ResponseCallback	Client.R
6	Client.ResponseCallback1	Client.R

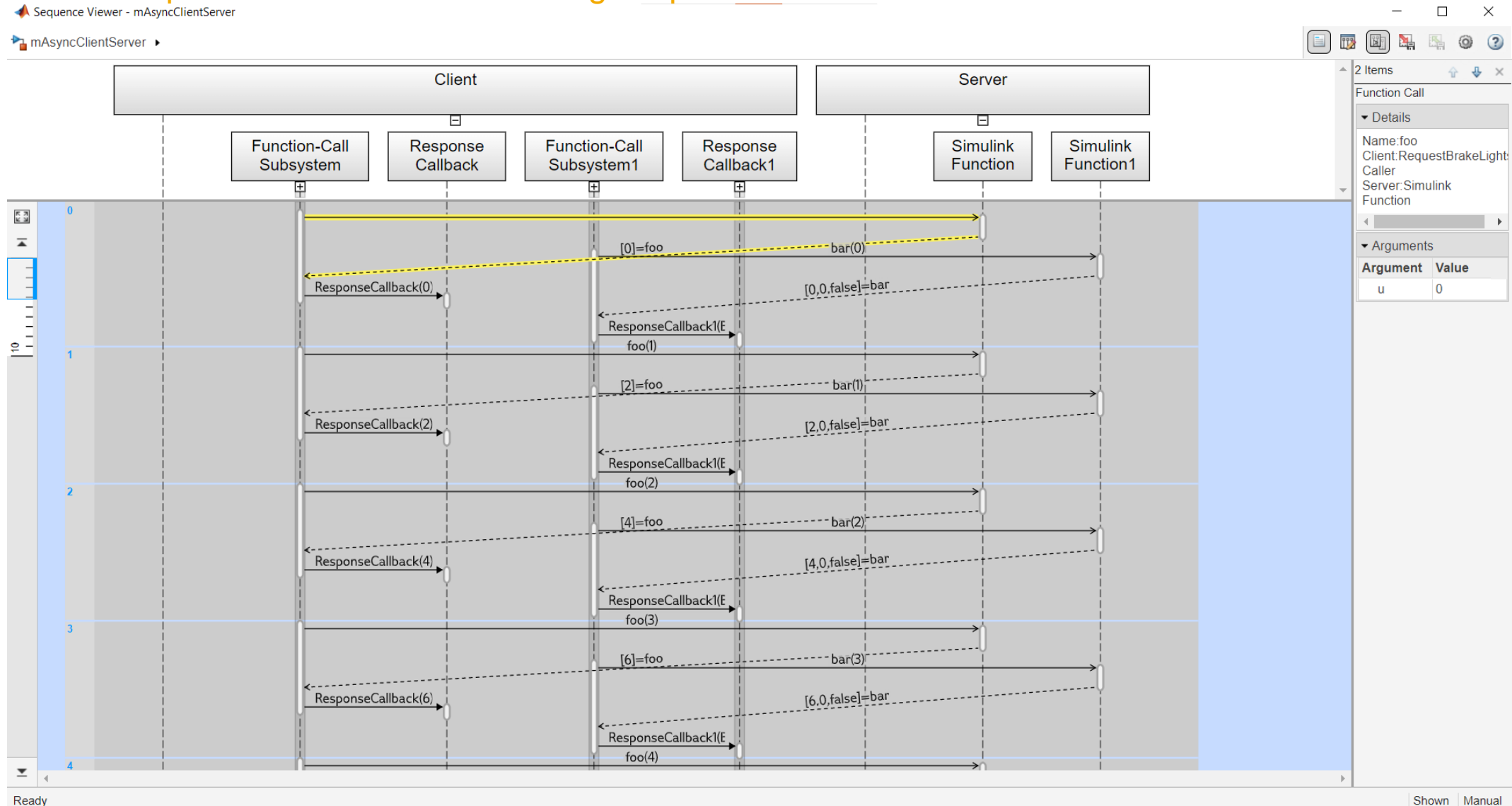
Drag to change the priority of a function

Property Inspector

Partition	
Name	Server.service.foo
Rate	A
Type	Aperiodic exported function

Service-Oriented Behavior Modeling

Visualize Sequence of Service Calls Using Sequence Viewer



User Presentations on SOA

How to Model on In-vehicle OS

MATLAB SIMULINK

InterfaceClass
 {
 // Filled with OS API
 ...
 }

Application
 Simulink Generated Code
 Wrapper Code

New Features
 - Support SOA Behavior

New Modeling Principle
 - Specific rules based on practice

Wrapper Code generator
 - Link Simulink side and OS side

Deployment
 - Deployed just like normal application

Copyright (c) 2023, ZEEKR Inc. All rights reserved.

15:53

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

Arbitration in AD/ADAS vehicles

At all events, arbitration is more rational, just, and humane than the resort to swords.
 - Richard Cobden

Arbitration in AD/ADAS vehicle is responsible for decision-making between Lateral control, Longitudinal control or hybrid control. The decision made leads to tactical & strategic planning of the vehicle manoeuvre.

At any point of time, Arbitration module shall consider the situation in-hand to make the decisions.

Arbitration scheme can be chosen based on various behaviour choices;

- Priority
- Pre-defined order
- Cost-based rules

KPIT

15:36

[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



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

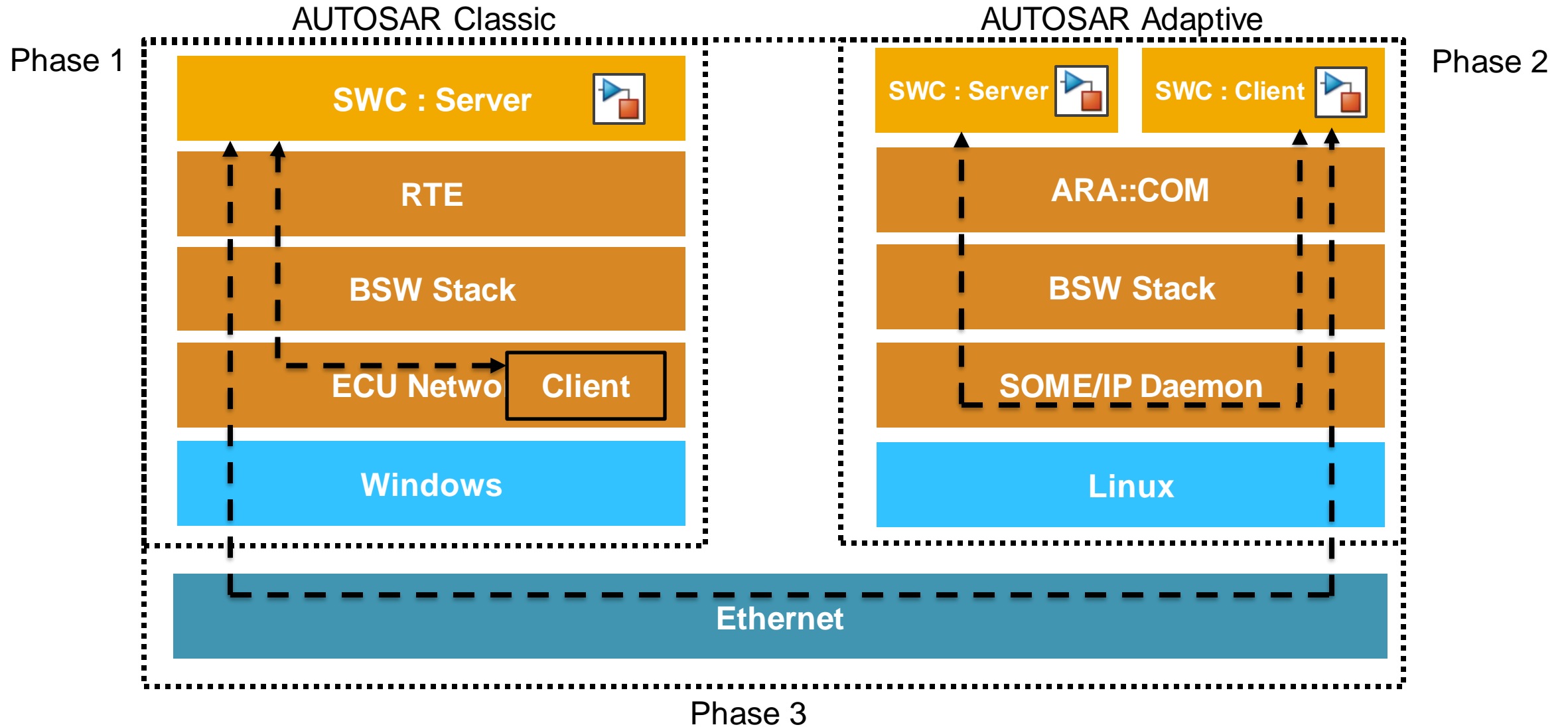
Shwetha Bhadravathi Patil
Technical Product Marketing
MathWorks
Natick, MA, United States
shwethap@mathworks.com

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

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

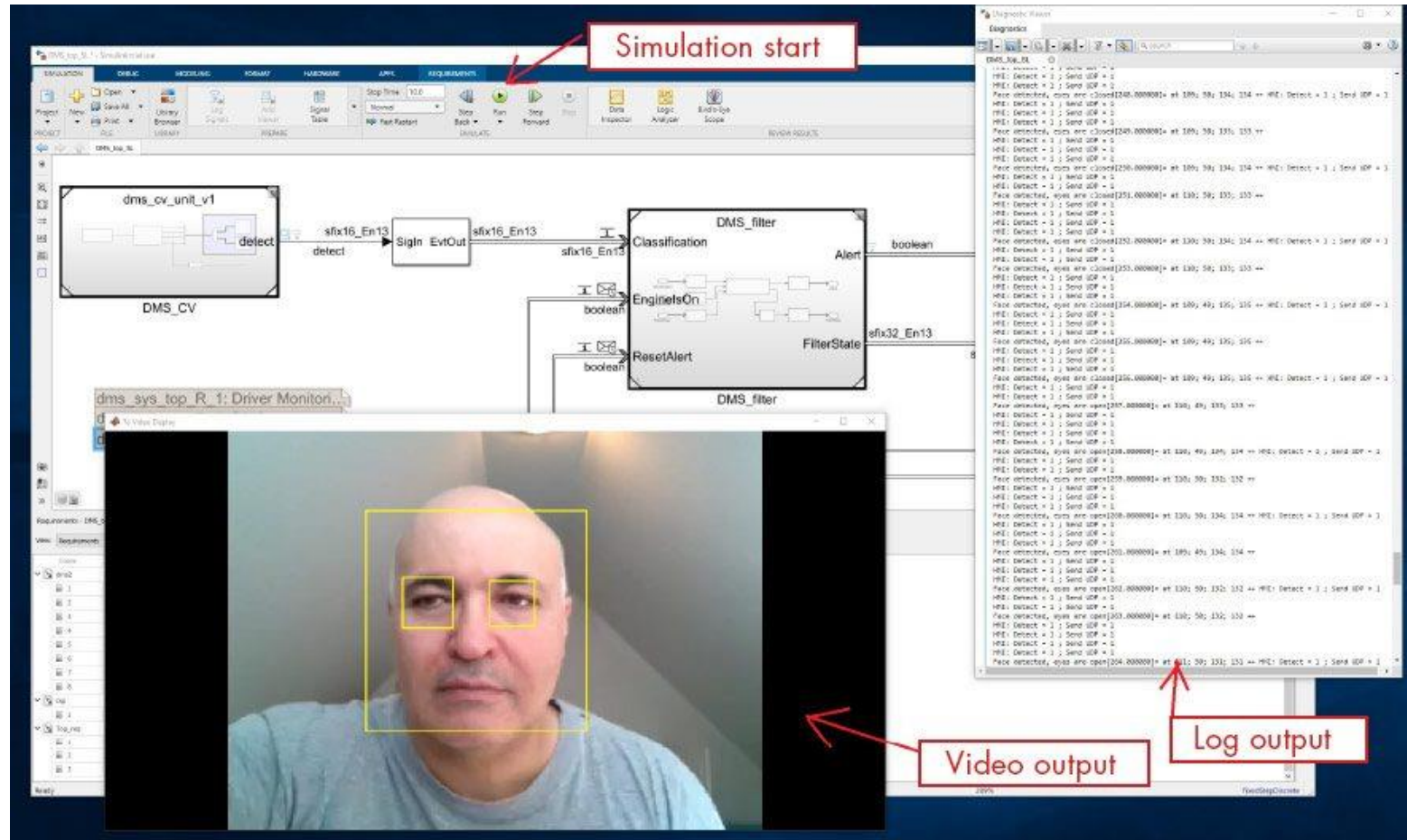
<https://www.mathworks.com/content/dam/mathworks/conference-or-academic-paper/develop-and-integrate-autosar-classic-and-adaptive-applications-based-on-some-ip.pdf>

Structure of Proof-of-Concept



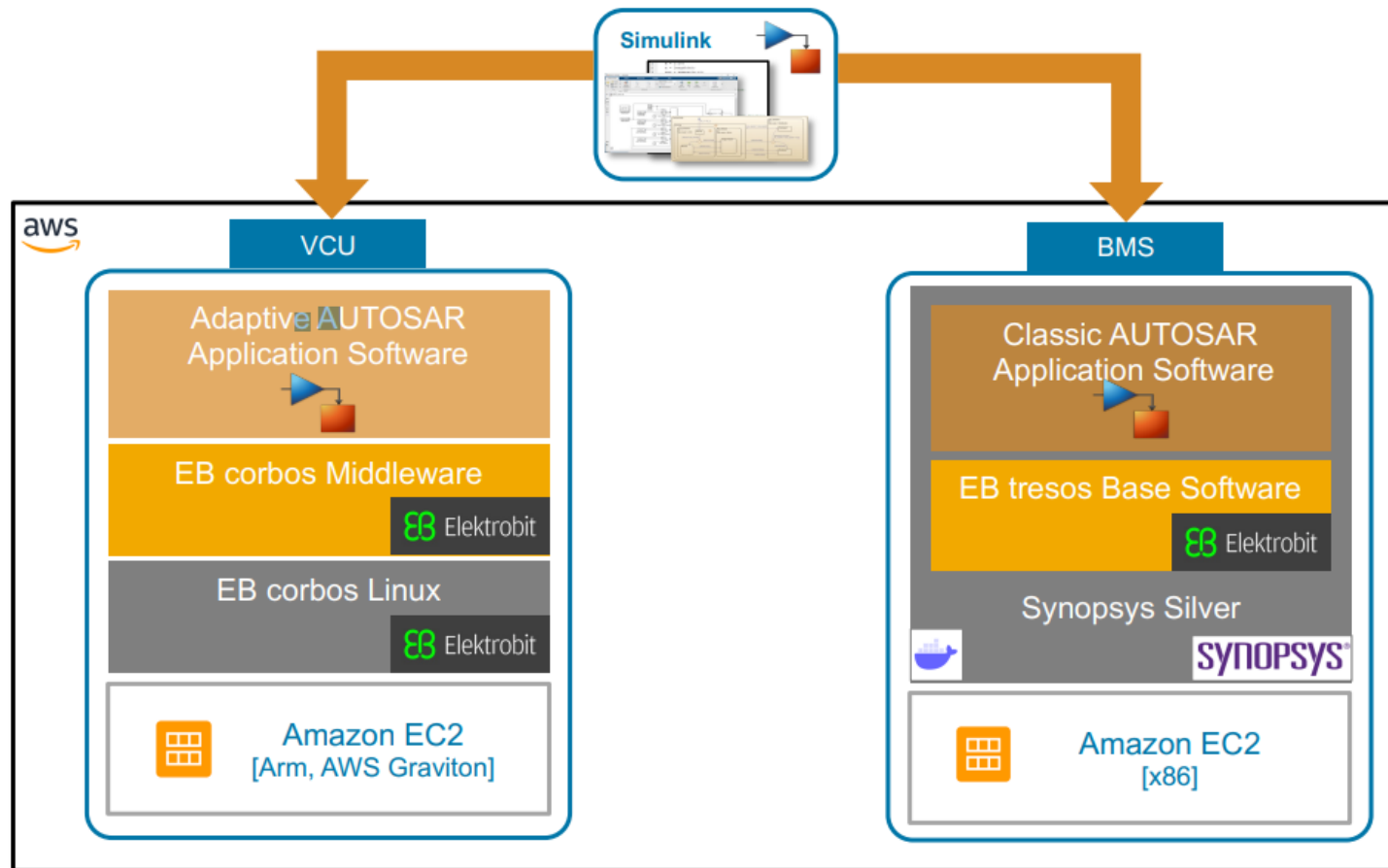
AUTOSAR Adaptive User Article

Elektrobit - [Developing AUTOSAR Adaptive Software for a Driver Monitoring System with Model-Based Design](#)

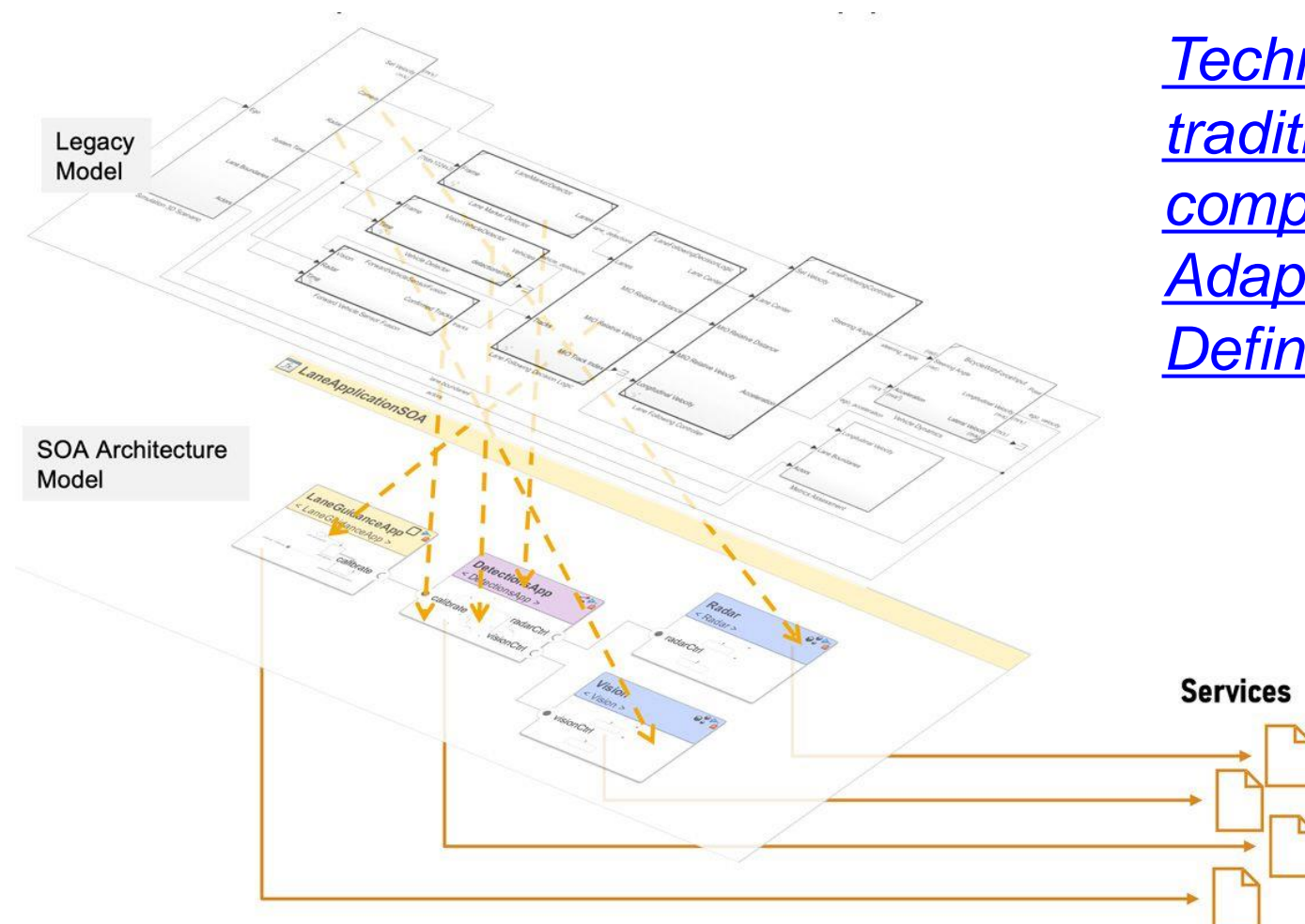


“Model-Based Design could accelerate development of end-to-end, AUTOSAR Adaptive software systems”

Cloud Deployment workflow with MBD

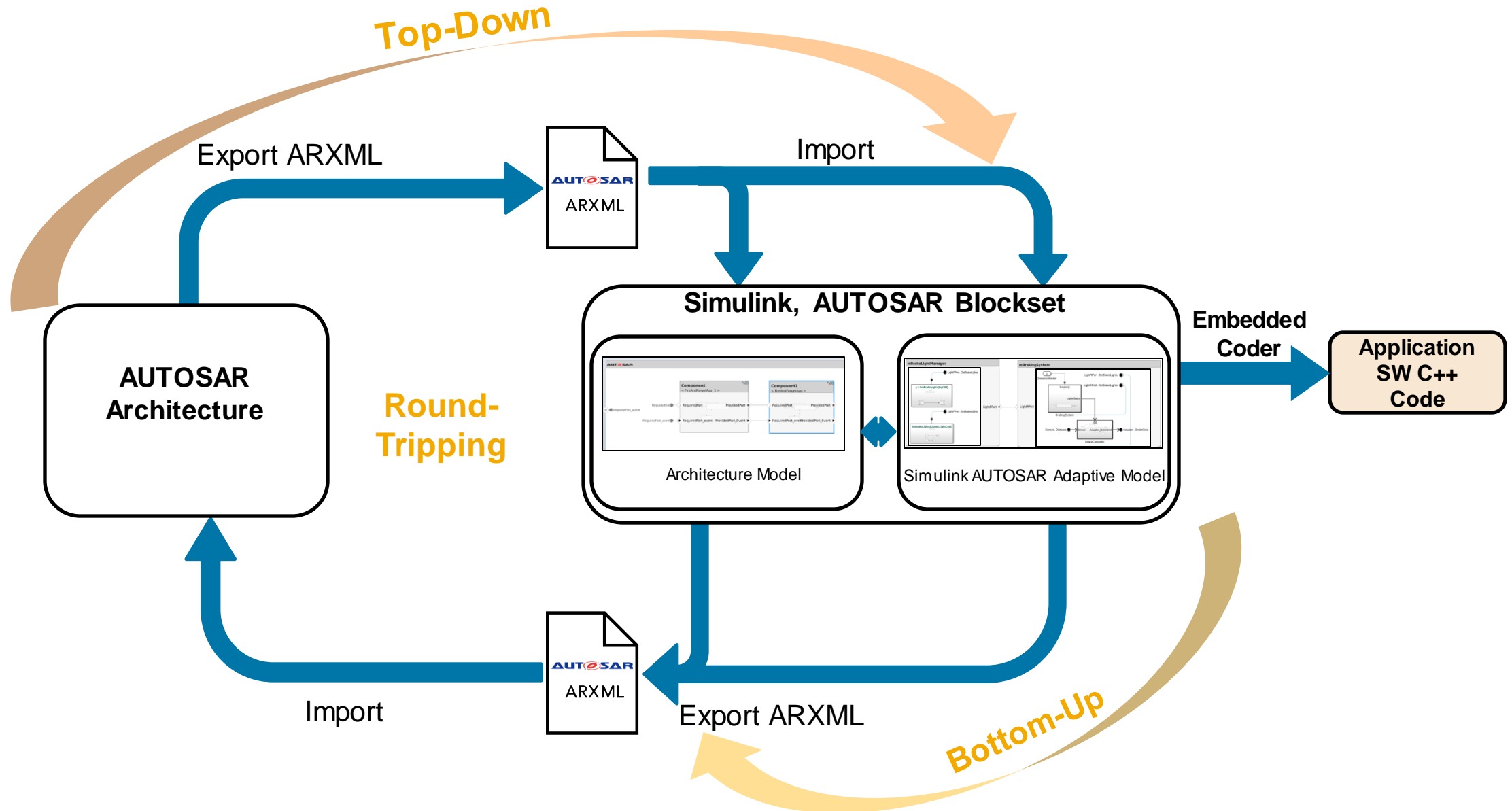


MathWorks presentation at AUTOSAR Open Conference 2023, May 11-12, San Diego, USA

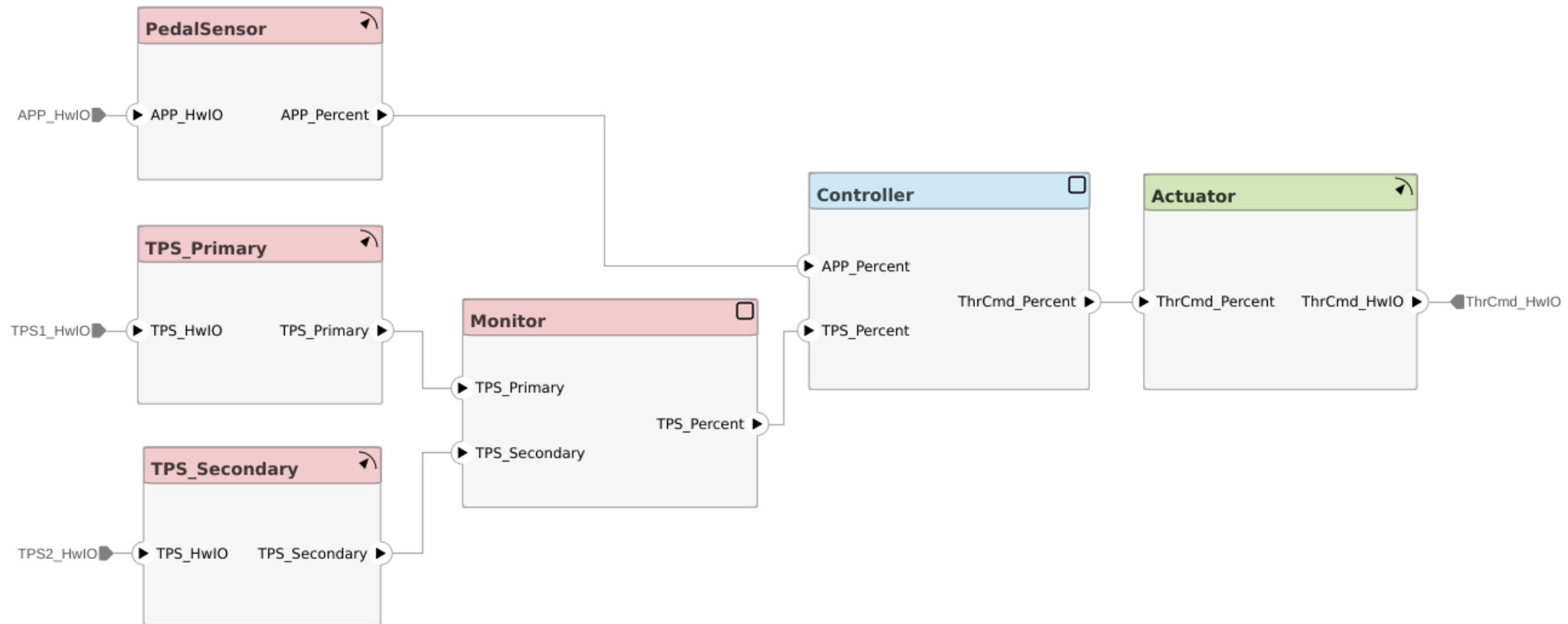


[Technical Article - Migrating traditional automotive application compositions to AUTOSAR Adaptive services for Software Defined Vehicles](#)

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

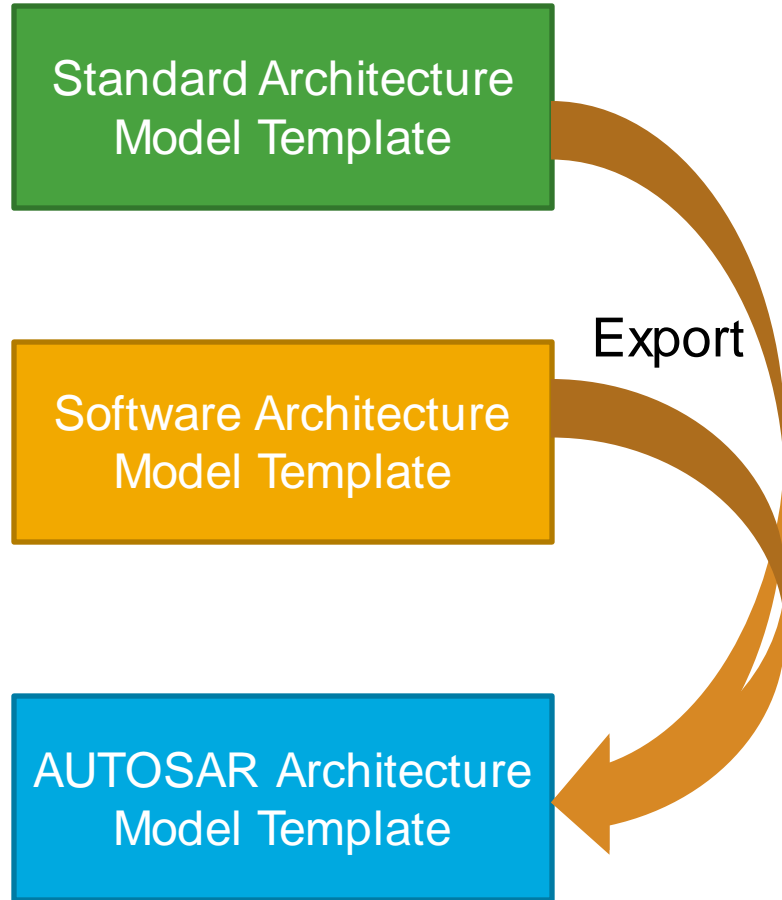
Software Architecture Model Template

- 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

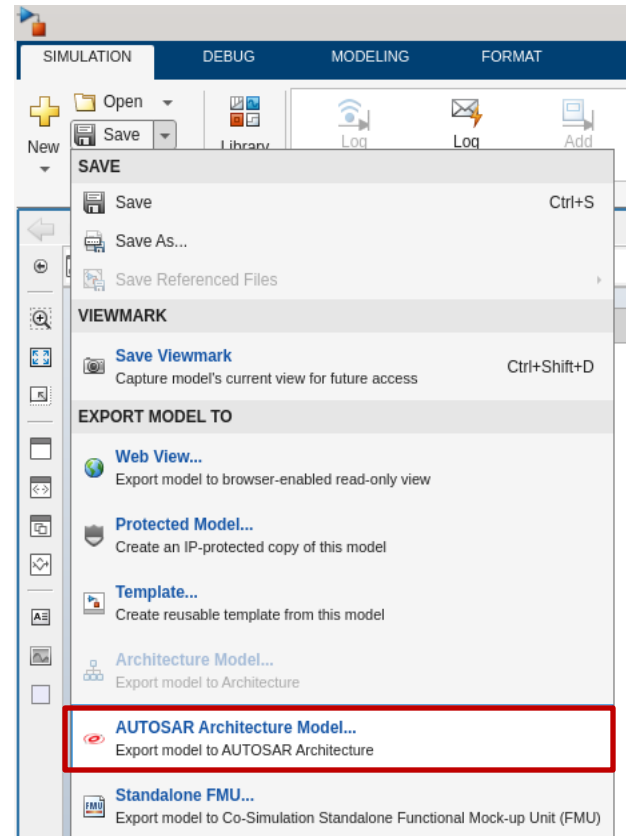


- We support 2 workflows
 - Model AUTOSAR architectures directly
 - Export existing architecture model to AUTOSAR

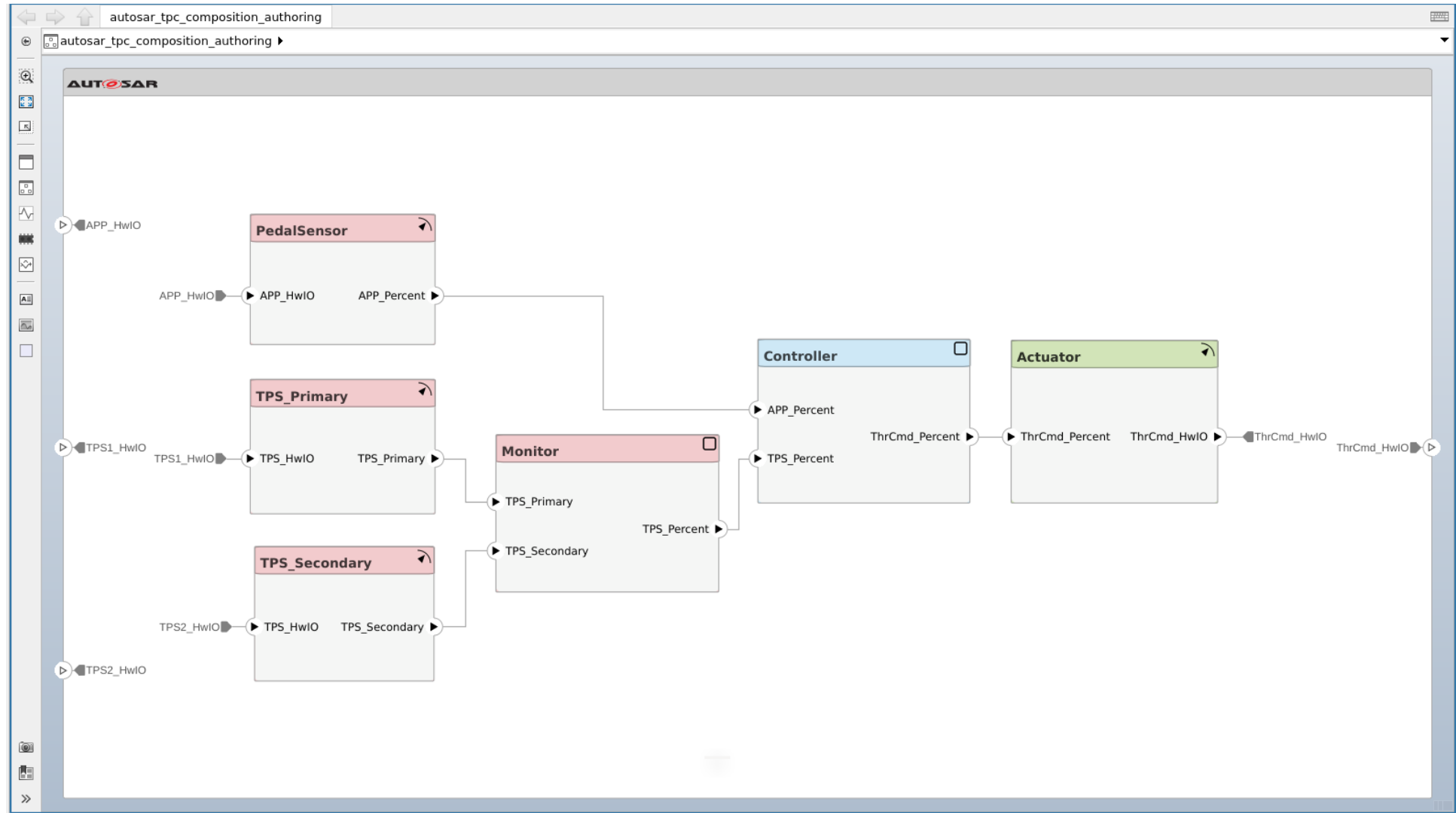
Let's use this workflow

R2019b

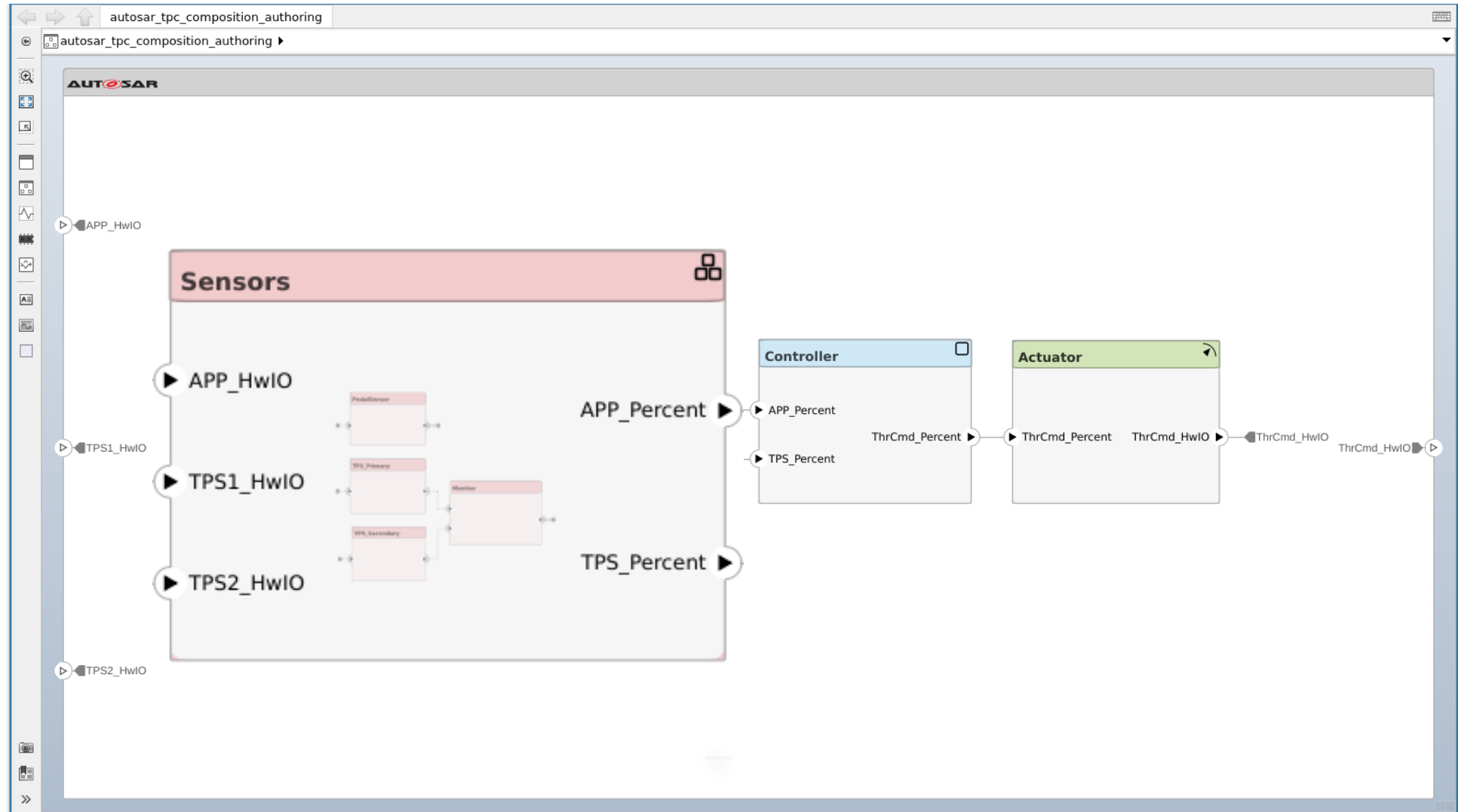
R2023b



Use Composition blocks to manage complexity

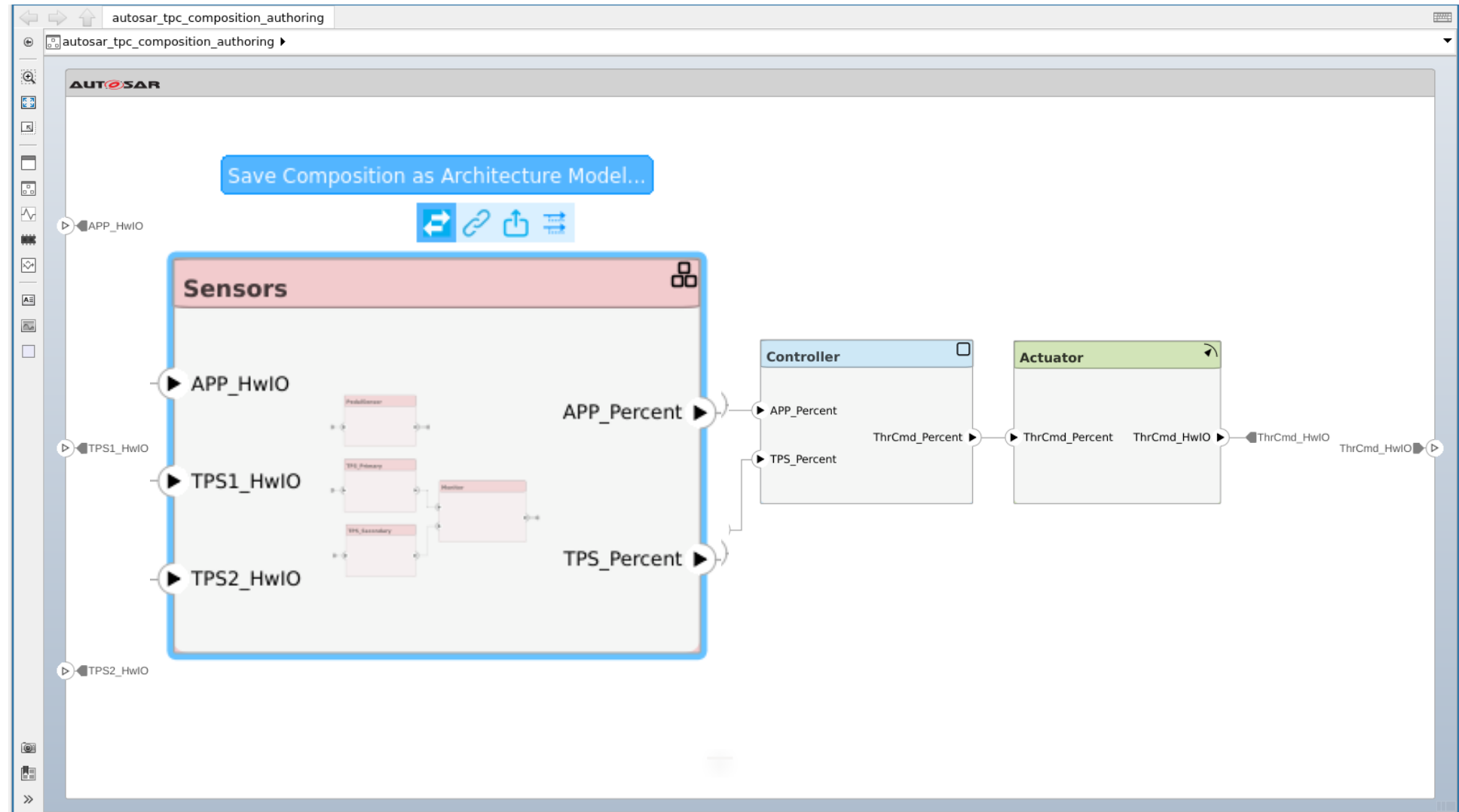


Use Composition blocks to manage complexity

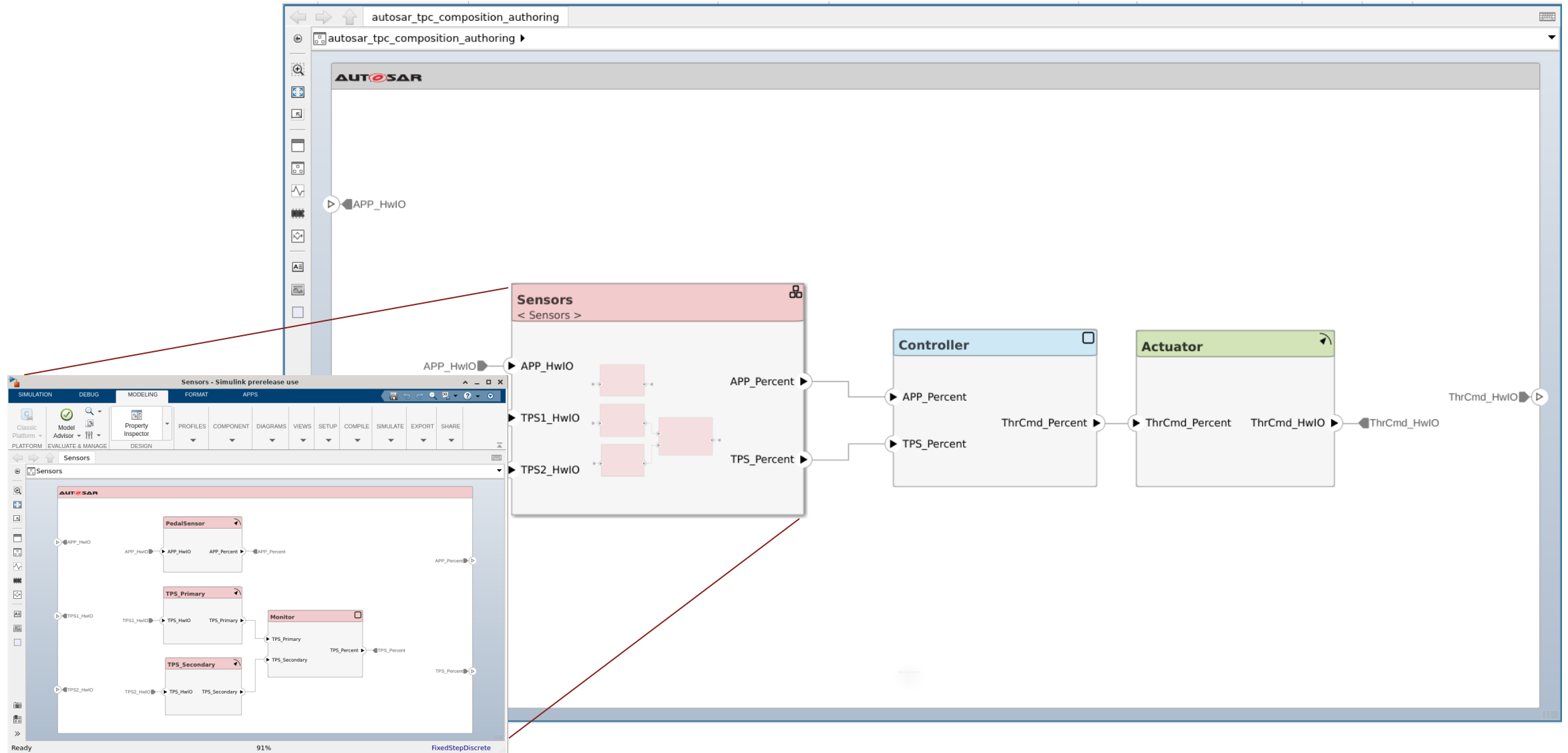


Save composition block as a separate AUTOSAR Architecture model

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)

The screenshot displays the MATLAB/Simulink SLDD (Software-in-the-Loop Design) environment. The main workspace shows an AUTOSAR block diagram with three primary components: **Sensors** (pink), **Controller** (blue), and **Actuator** (green). The Sensors block has three input ports: APP_HwIO, TPS1_HwIO, and TPS2_HwIO. The Controller block has two input ports: APP_Percent and TPS_Percent. The Actuator block has one input port: ThrCmd_Percent. The output of the Actuator is ThrCmd_HwIO. The diagram is connected to external hardware interfaces (APP_HwIO, TPS1_HwIO, TPS2_HwIO, ThrCmd_HwIO).

Below the diagram is the **Architectural Data Editor** (ADE) window, which is used to manage interfaces, data types, and dictionary references. It includes a search bar and a table with the following columns: **type**, **Dimensions**, **Units**, **Complexity**, **Minimum**, **Maximum**, **Description**, and **Asynchronous**. The table currently contains one entry:

type	Dimensions	Units	Complexity	Minimum	Maximum	Description	Asynchronous
tpc_dict.sldd*							

The right-hand side of the interface shows the **Property Inspector** for the selected port, displaying details such as **Name** (ThrCmd_HwIO) and **Stereotype** (Add...).

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

The screenshot displays the Simulink Architectural Editor interface. The main workspace shows a block diagram with three primary blocks: **Sensors** (pink), **Controller** (blue), and **Actuator** (green). The **Sensors** block has three input ports: APP_HwIO, TPS1_HwIO, and TPS2_HwIO. The **Controller** block has two input ports: APP_Percent and TPS_Percent. The **Actuator** block has one input port: ThrCmd_Percent. The output of the Controller is ThrCmd_Percent, which is connected to the input of the Actuator. The output of the Actuator is ThrCmd_HwIO. The diagram is titled "AUTOSAR" and "autosar_tpc_composition_authoring".

At the bottom of the window, the **Interfaces** section is visible, containing a table with the following structure:

Manage interfaces, data types and dictionary references in the Architectural Data Editor							
type	Dimensions	Units	Complexity	Minimum	Maximum	Description	Asynchronous
tpc_dict.sldd*							

The status bar at the bottom indicates "Ready", "129%", and "FixedStepDiscrete".

Simulate AUTOSAR Architecture

The screenshot displays the Simulink AUTOSAR modeling environment. The main workspace shows a system architecture diagram with the following components and connections:

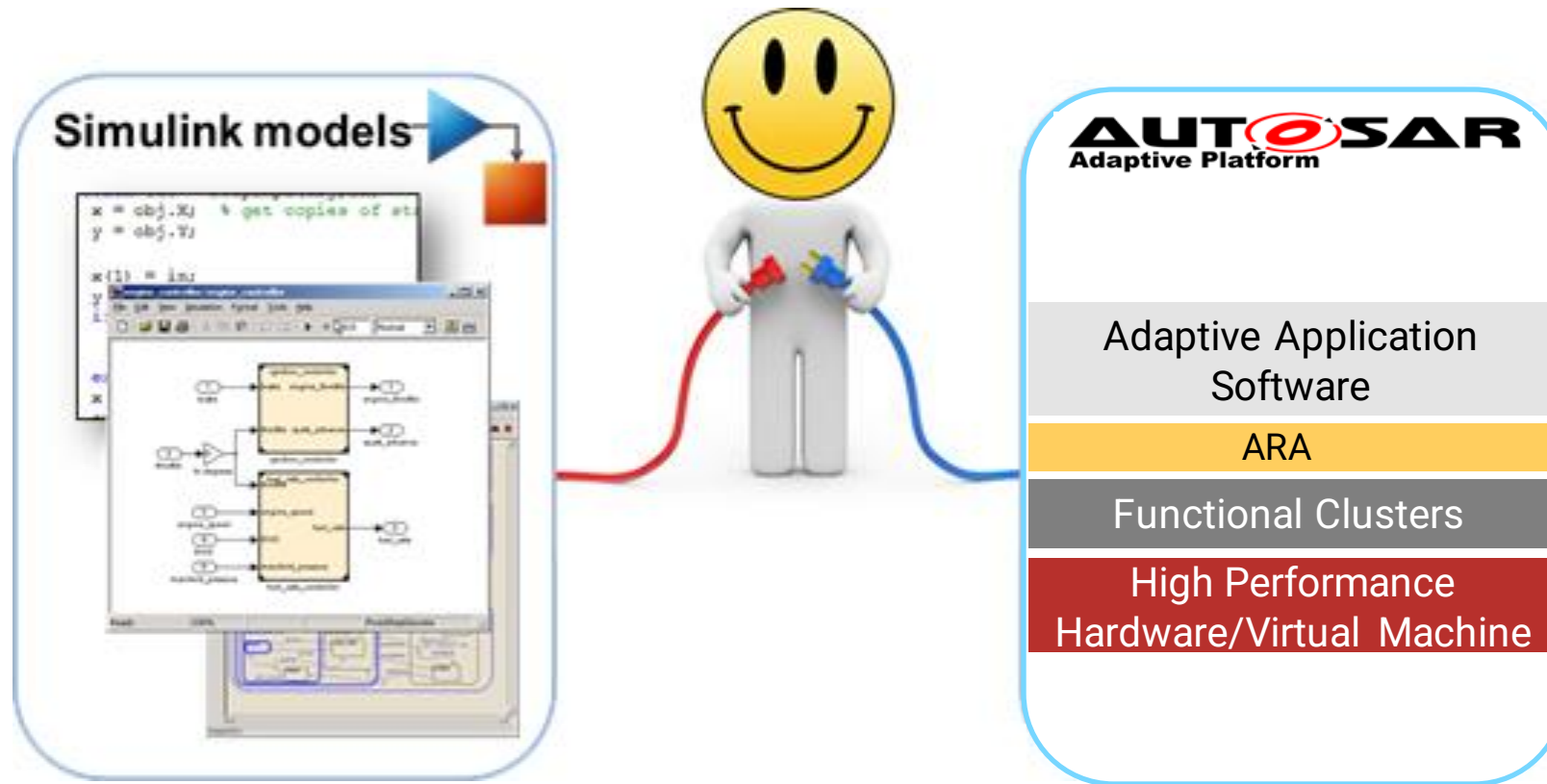
- Sensors** (red box): Contains sub-components for APP, TPS1, and TPS2. It has external interfaces for APP_HwIO, TPS1_HwIO, and TPS2_HwIO. Internal outputs are APP_Percent, TPS1_Percent, and TPS2_Percent.
- Ctrl** (green box): Labeled as '< tpc_controller >'. It receives APP_Percent and TPS_Percent as inputs and outputs ThrCmd_Percent.
- Actuator** (red box): Labeled as '< tpc_actuator >'. It receives ThrCmd_Percent as input and outputs ThrCmd_HwIO.
- DEM / FIM** (white box): Connected to the Sensors component.
- NvM** (white box): Connected to the Sensors component.

Additional panels and toolbars include:

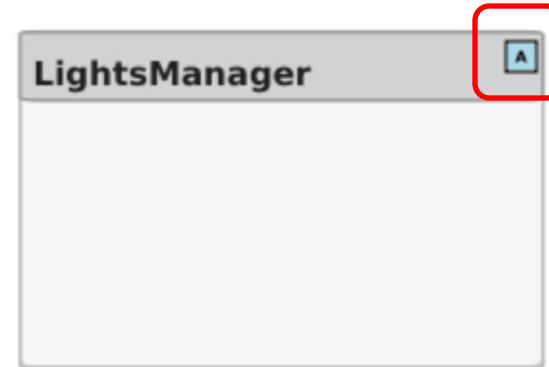
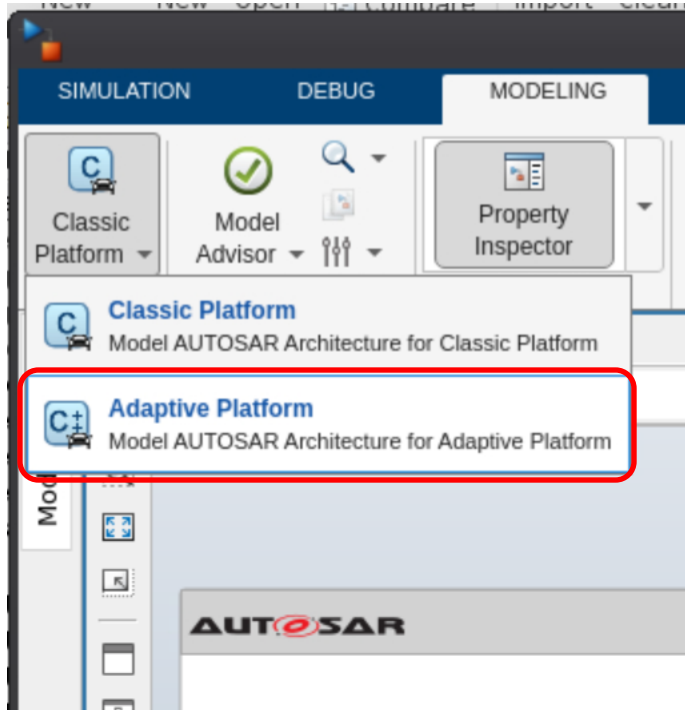
- Property Inspector** (top right): Shows the selected 'ThrCmd_Percent' interface with a 'Name' of 'ThrCmd_Percent' and a 'Stereotype' of 'Add...'.
- Interfaces** (bottom left): A list of interfaces including APP_Percent, TPS_HwIO, TPS_Percent, ThrCmd_HwIO, ThrCmd_Percent (selected), and hwio_t.
- Functions Editor** (bottom right): A table listing the execution order of functions in the model.

Execution Order	Function Name	Software Component	Period
1	PedalSensor_D1	Sensors/PedalSensor	0.005
2	TPS_Primary_D1	Sensors/TPS_Primary	0.005
3	TPS_Secondary_D1	Sensors/TPS_Secondary	0.005
4	Monitor_D1	Sensors/Monitor	0.005
5	Ctrl_D1	Ctrl	0.005
6	Actuator_D1	Actuator	0.005

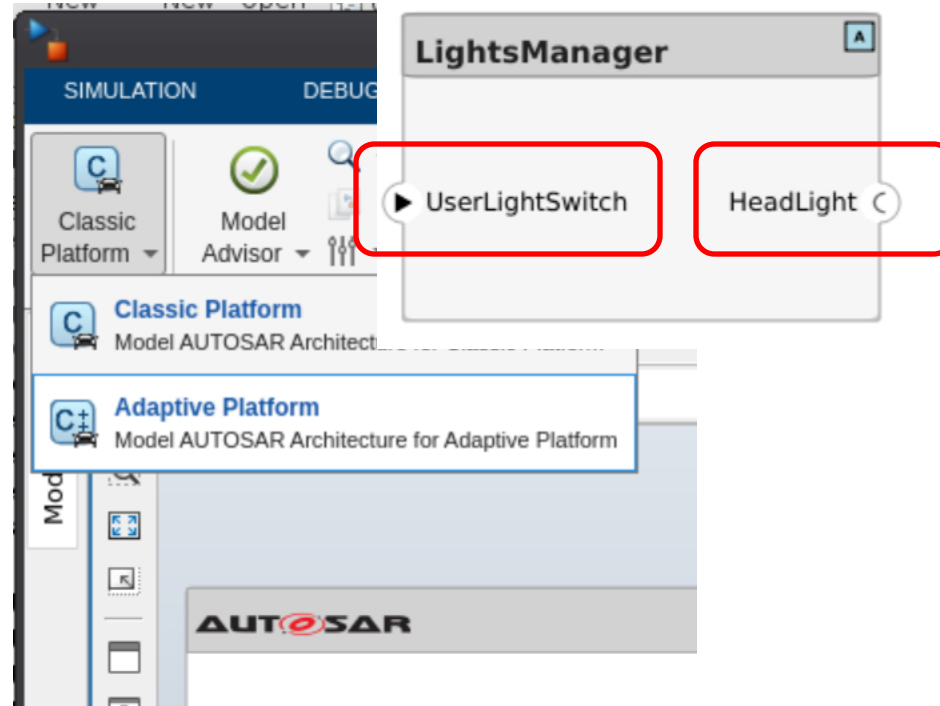
Plug your Simulink SOA models into the AUTOSAR Adaptive Platform



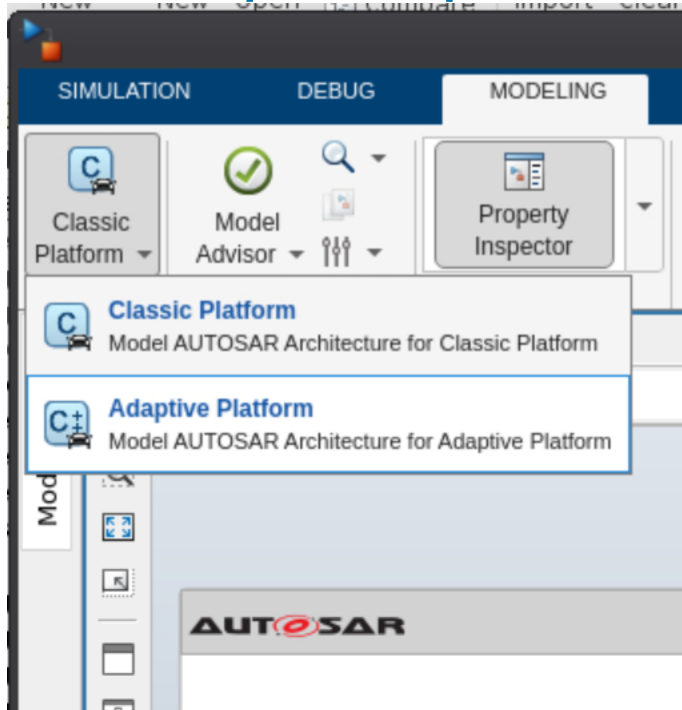
Develop Adaptive AUTOSAR Architectures



Develop Adaptive AUTOSAR Architectures



Develop Adaptive AUTOSAR Architectures



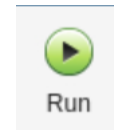
- Create Simulink Behavior...
- Link to Model...
- Create Component Model from ARXML...

Interfaces

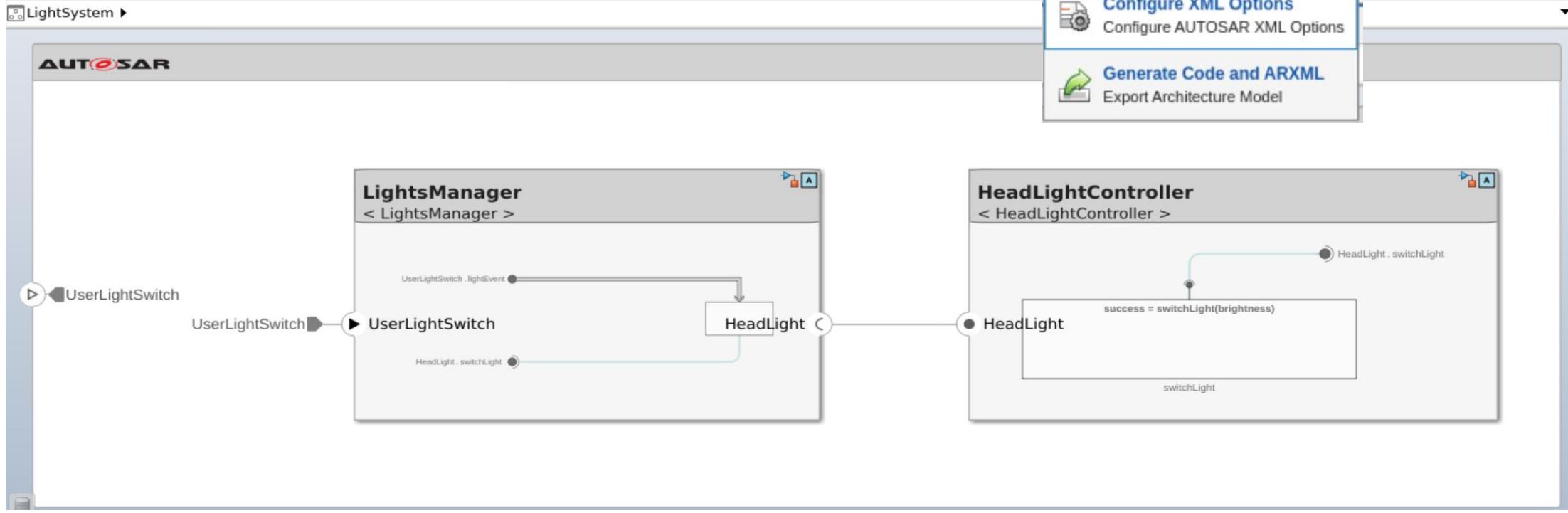
Open Dictionary [Icons] Search

	Type
▼ interfaces.sldd	
▼ LightService	
▶ success = switchLight(brightness)	
▶ LightSwitch	

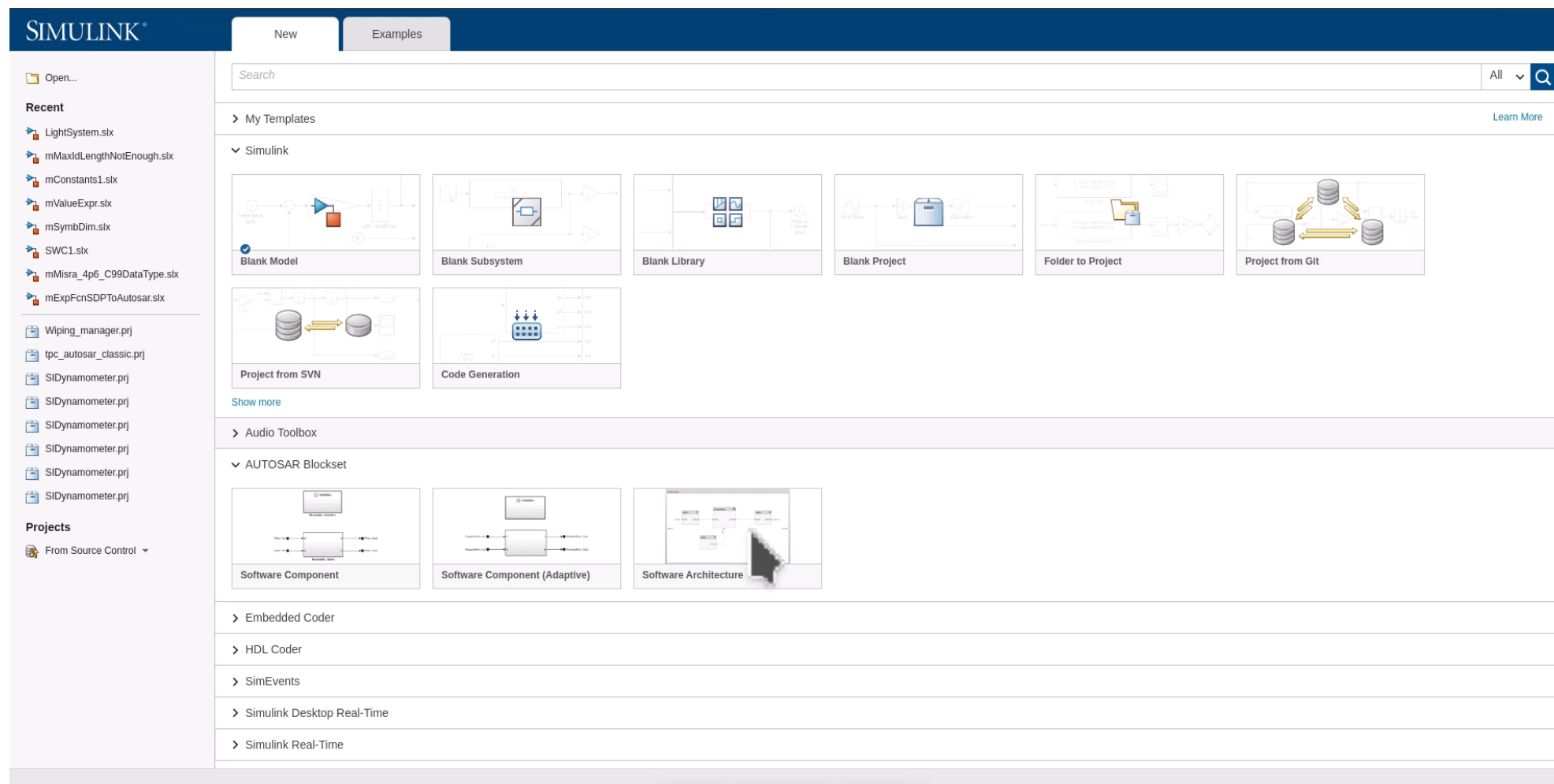
Export your Adaptive Architecture Models





- Configure XML Options**
Configure AUTOSAR XML Options
- Generate Code and ARXML**
Export Architecture Model

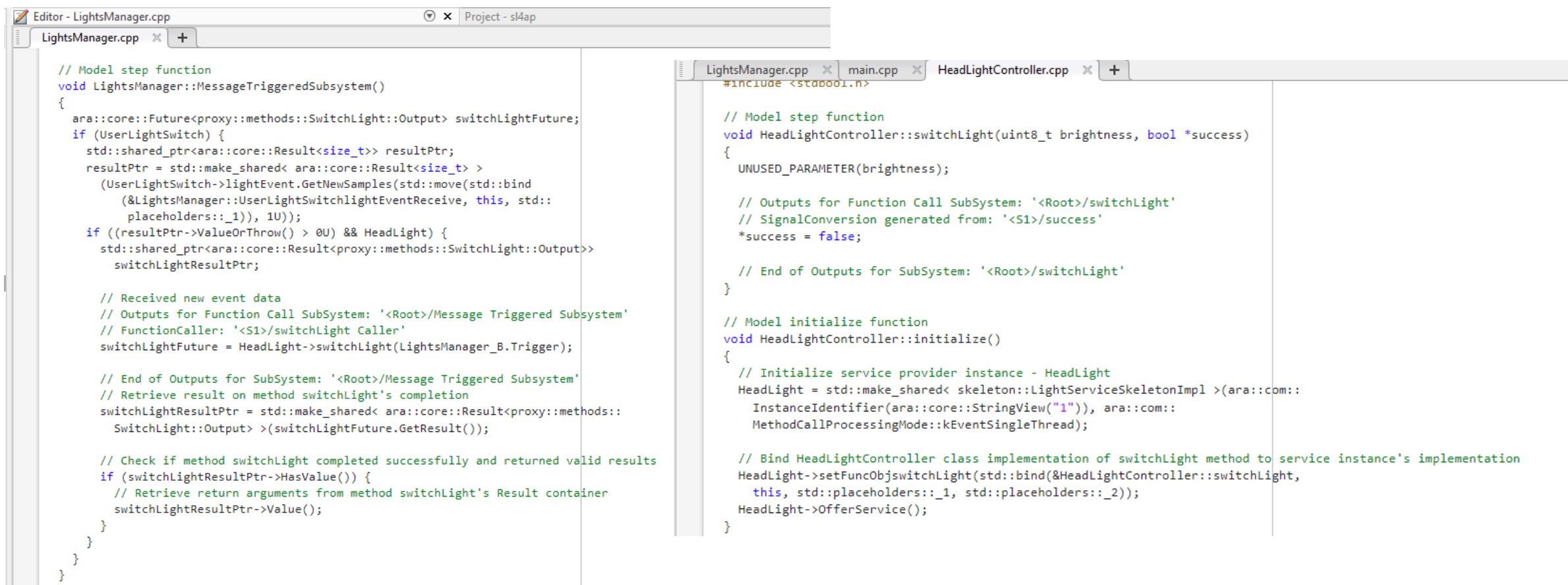


Model Adaptive AUTOSAR Architectures in System Composer



Generated Code and ARXML for your system ready for deployment

- +  HeadLightController_autosar_adaptive
- +  LightsManager_autosar_adaptive



```

Editor - LightsManager.cpp
Project - sl4ap
LightsManager.cpp x +

// 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 x main.cpp x HeadLightController.cpp x +
#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;

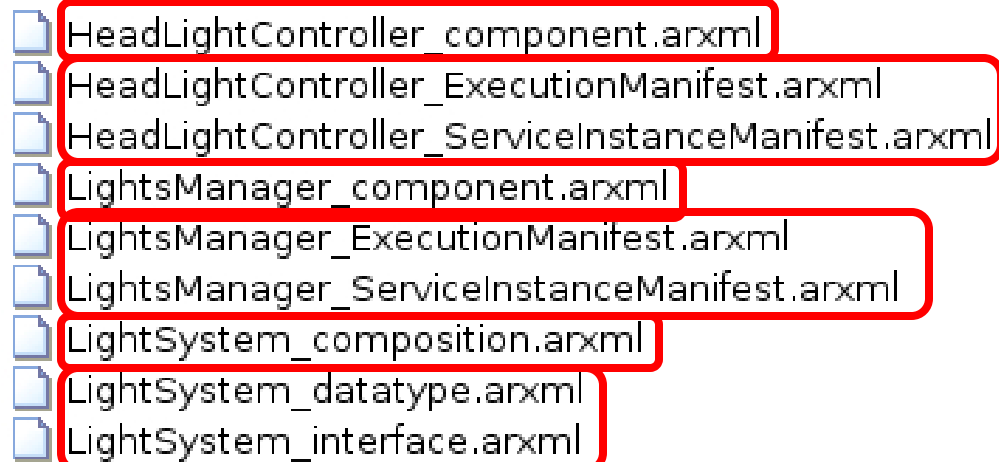
    // End of Outputs for SubSystem: '<Root>/switchLight'
}

// 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

The screenshot displays the Linux Runtime Manager interface. At the top, there is a toolbar with icons for Disconnect, Update Target, Create & Deploy Application Package, Start Application, Stop Application, Monitor & Tune, and Data Inspector. Below the toolbar is a 'Targets Tree' on the left showing 'LinuxTarget1'. The main area is divided into 'Parameters' and 'Log Viewer' sections.

Target Configuration:

- Name: LinuxTarget1
- IP address: 172.26.235.165
- Username: pdlersin
- Password: [Redacted]

Log Viewer:

Timestamp	AppID	CtxID	Type	Message
2023/06/09 14:54:33...	LNXD	LNXD	info	[ACTION: Launching process: HeadLightController]
2023/06/09 14:54:33...	LNXD	LNXD	info	[Process launch initiated: HeadLightController with PID: 531]
2023/06/09 14:54:33...	LNXD	LNXD	info	[ACTION: Launching process: LightsManager]
2023/06/09 14:54:33...	LNXD	LNXD	info	[Process launch initiated: LightsManager with PID: 532]
2023/06/09 14:54:33...	DLTD	INTM	info	[ApplicationID '839' registered for PID 531, Description=Logger for HeadLightController's main function.]
2023/06/09 14:54:33...	DLTD	INTM	info	[ApplicationID '535' registered for PID 532, Description=Logger for LightsManager's main function.]
2023/06/09 14:54:51...	LNXD	LNXD	info	[ACTION: Terminating process: HeadLightController with PID: 531]
2023/06/09 14:54:51...	DLTD	INTM	info	[Unregistered ApID '839']
2023/06/09 14:54:59...	LNXD	LNXD	info	[ACTION: Terminating process: LightsManager with PID: 532]
2023/06/09 14:54:59...	DLTD	INTM	info	[Unregistered ApID '535']

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