

MATLAB EXPO

Developing Service-Oriented Architecture and Implementing Using Adaptive AUTOSAR and DDS

Rajat Arora, MathWorks



Nukul Sehgal, MathWorks



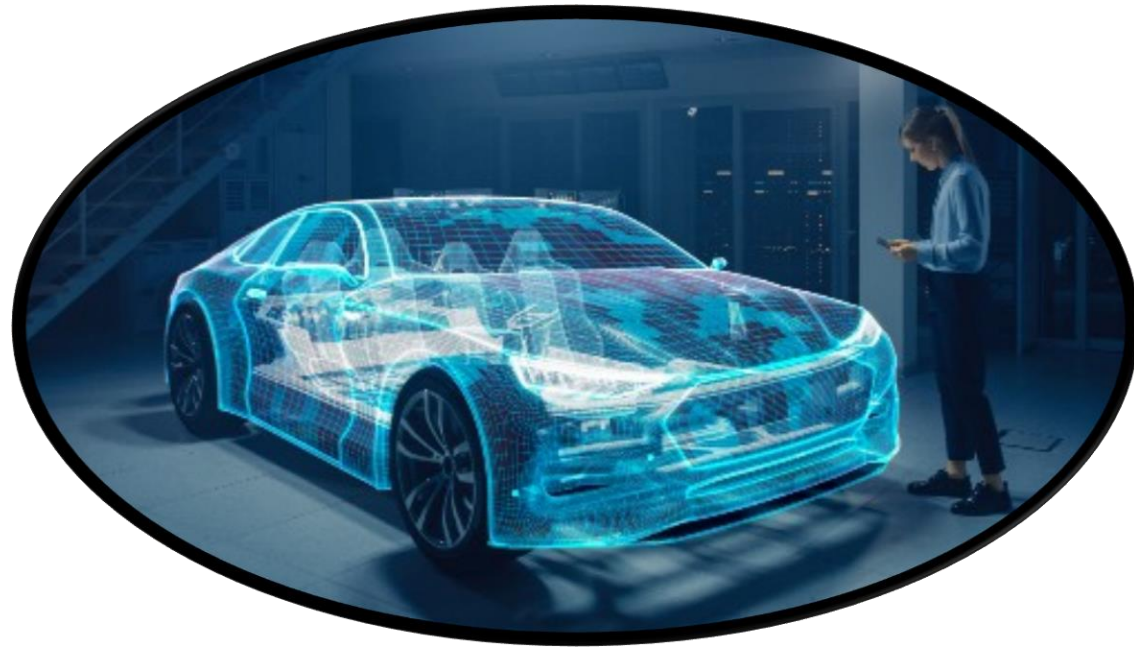
Agenda

- Software-defined vehicles and new architectures (SOA)
- SOA Concepts
- MathWorks Solutions for SOA
 - Adaptive AUTOSAR
 - DDS/ROS
- Conclusions and key takeaways

Agenda

- **Software-defined vehicles and new architectures (SOA)**
- SOA Concepts
- MathWorks Solutions for SOA
 - Adaptive AUTOSAR
 - DDS/ROS
- Conclusions and key takeaways

Software-defined vehicles



Brand-distinctive features and main value for the customer will come from Software

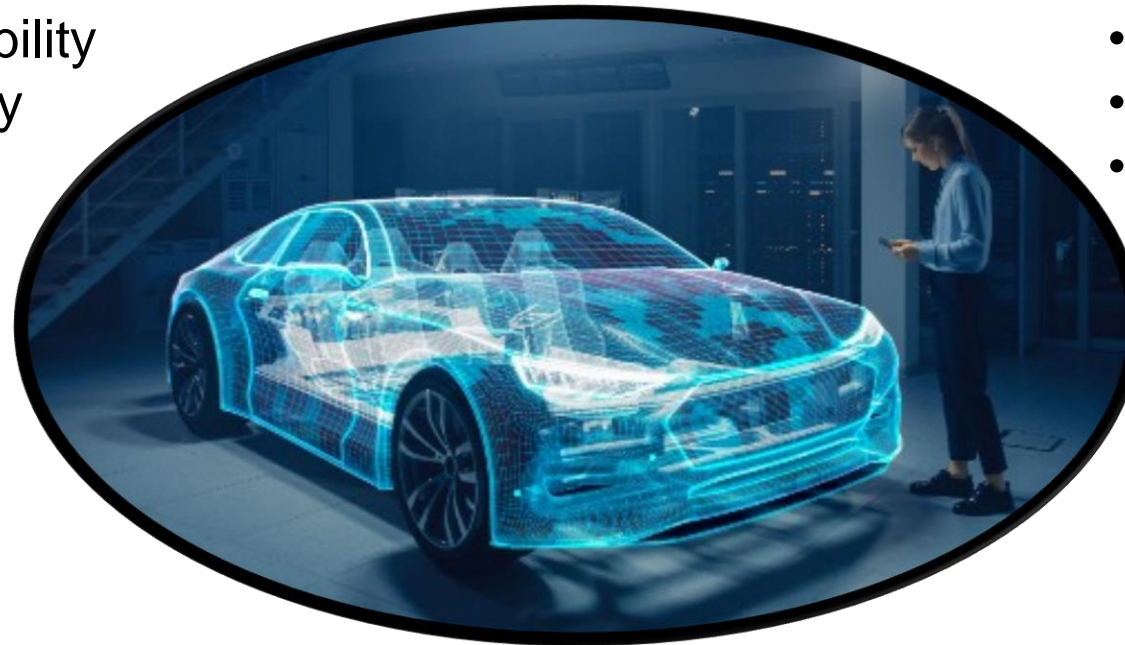
Software-defined vehicles

Customer expectations

- Clean and Safe mobility
- Digital Life continuity

Technology & Innovation

- Electrification
- Autonomy
- Connectivity



monetize

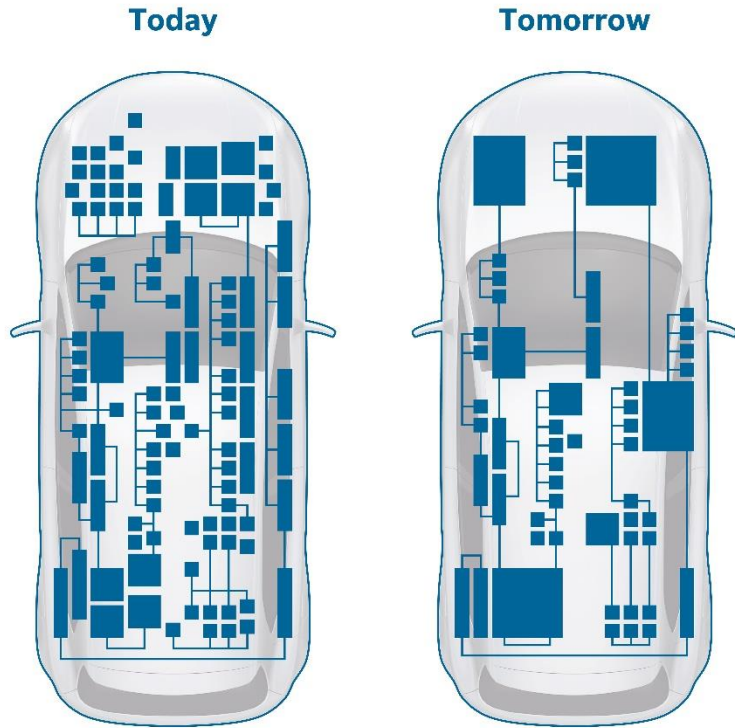
Business opportunity

- App stores, SW features on demand
- SW services subscription plans

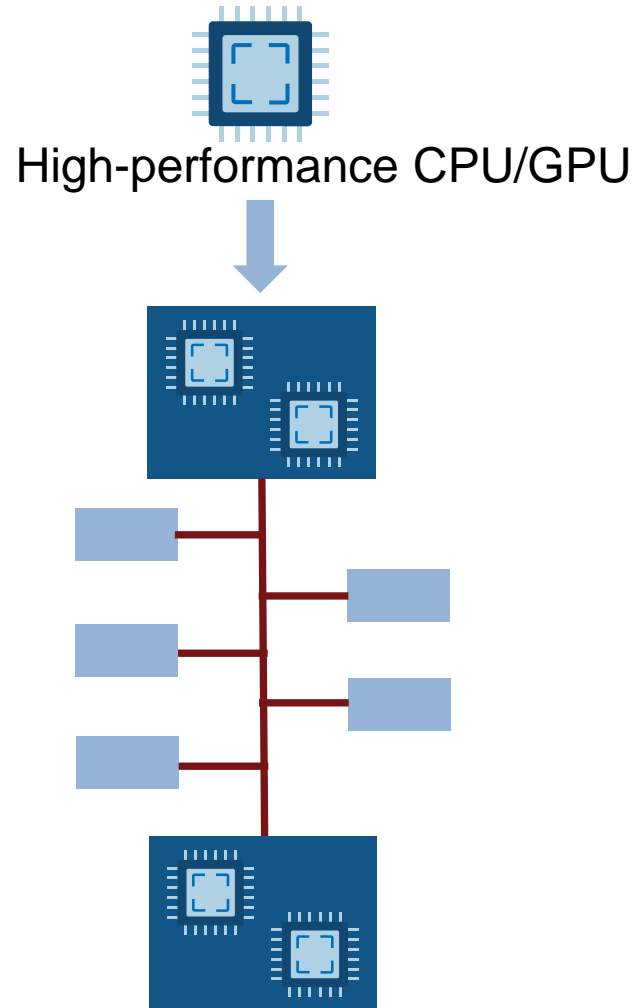
invest

demand

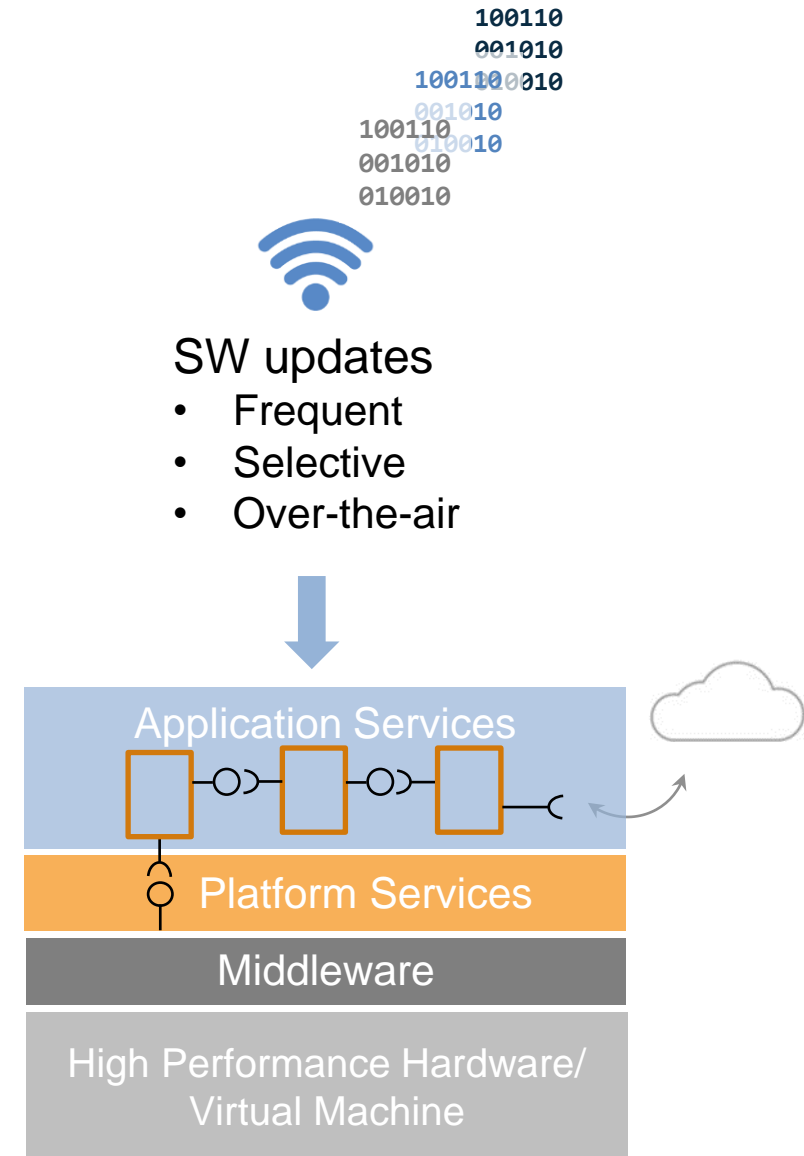
Centralization of computing and SOA



Consolidation and centralization of computing



New E/E zonal architectures



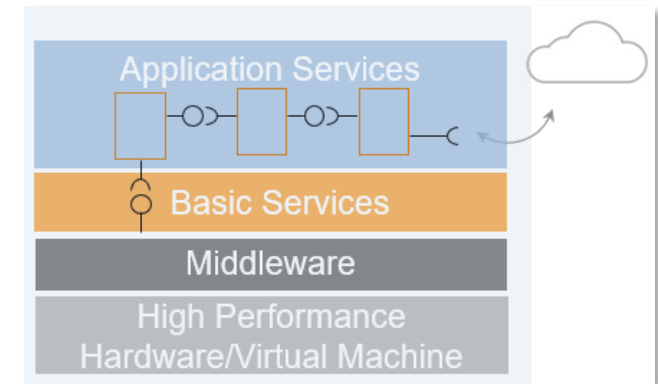
Higher HW abstraction:
Service-oriented architectures

Agenda

- Software-defined vehicles and new architectures (SOA)
- **SOA Concepts**
- MathWorks Solutions for SOA
 - Adaptive AUTOSAR
 - DDS/ROS
- Conclusions and key takeaways

SOA – What's it all about?

- With SOA, applications are standalone processes that provide and/or require services distributed across the vehicle computing platform and the cloud
- SOA provides flexibility to add, remove, or update applications without impacting the entire, typically large, software system
- SOA is used by multiple industrial standards:
 - AUTOSAR Adaptive Platform
 - DDS (Data Distribution Services)
 - ROS (Robot Operating System)



AUTOSAR Blockset

Design and simulate AUTOSAR software

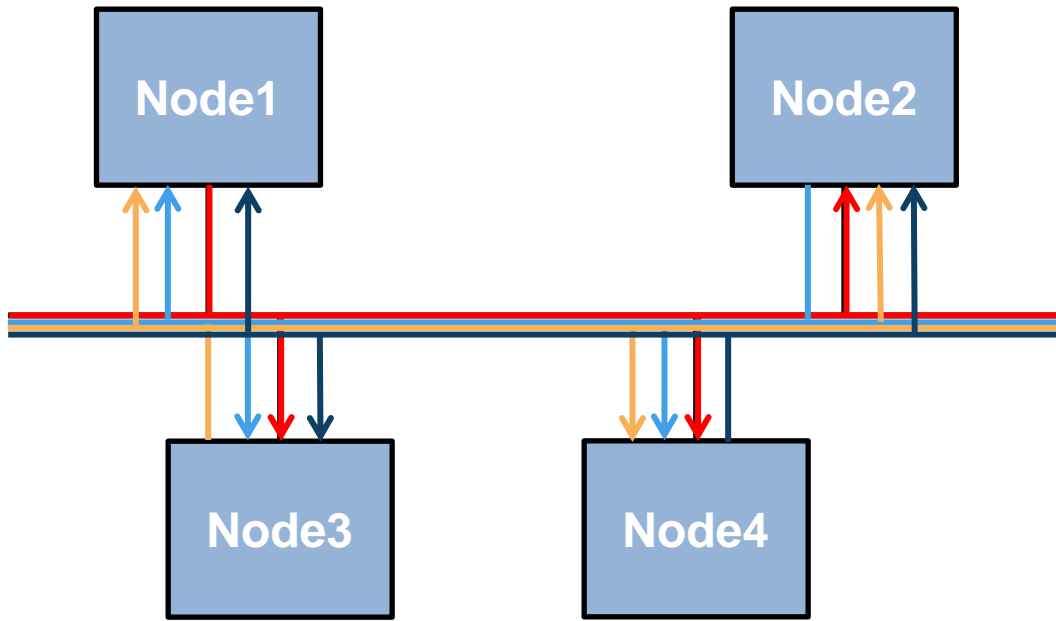
DDS Blockset

Design and simulate DDS applications

ROS Toolbox

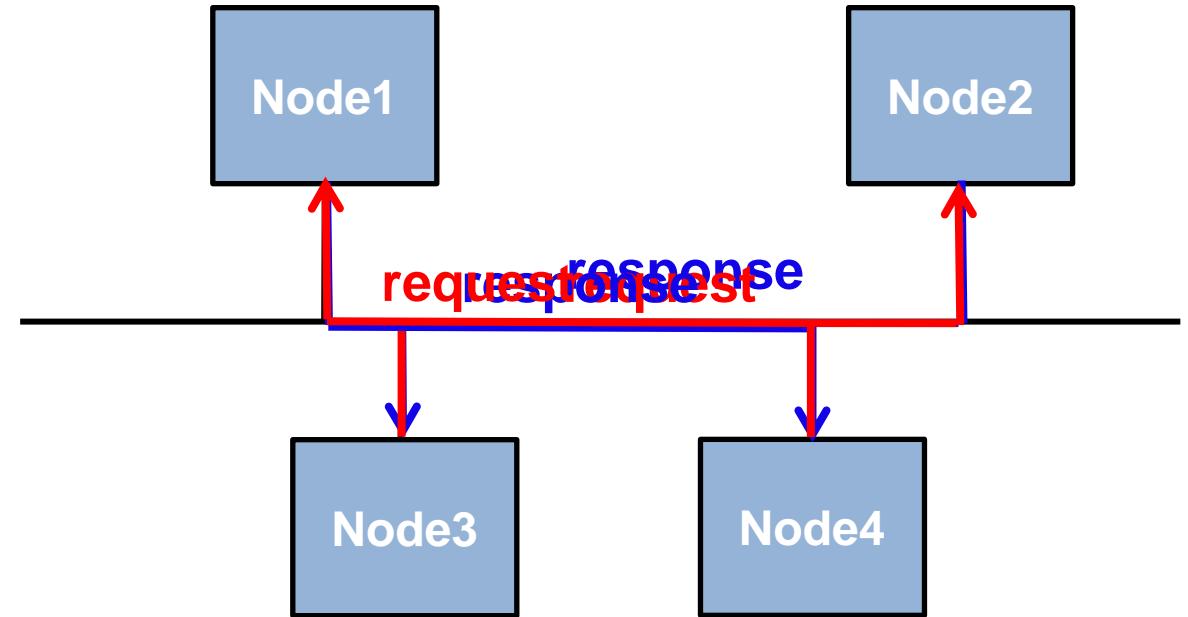
Design, simulate, and deploy ROS-based applications

SOC (Service Oriented Communication)



signal-oriented communication

- send data independent of needs
- high bus load
- not efficient



service-oriented communication

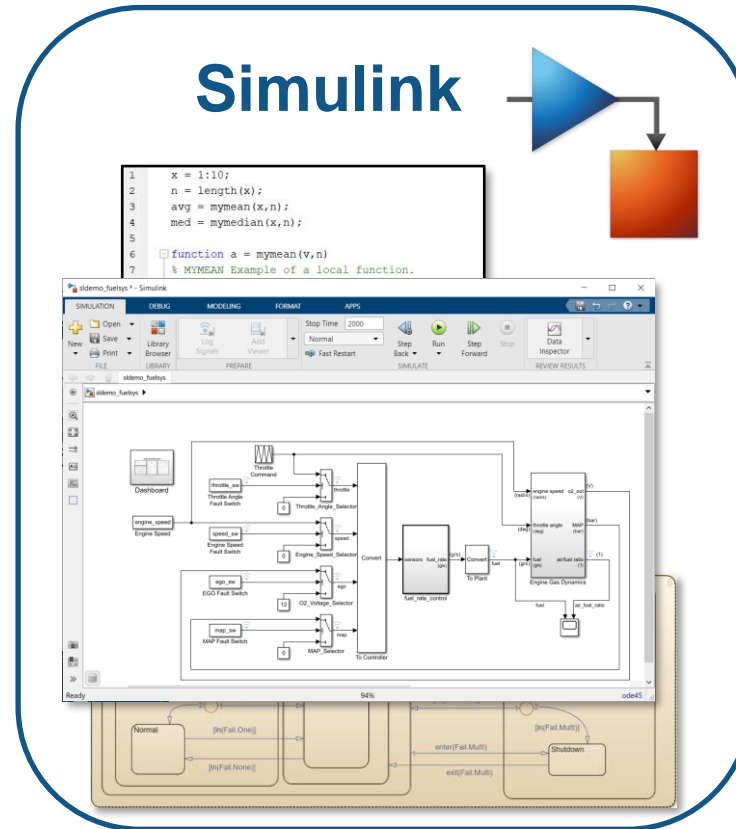
- send data dependent of needs
- low bus load
- more efficient

Key Challenges

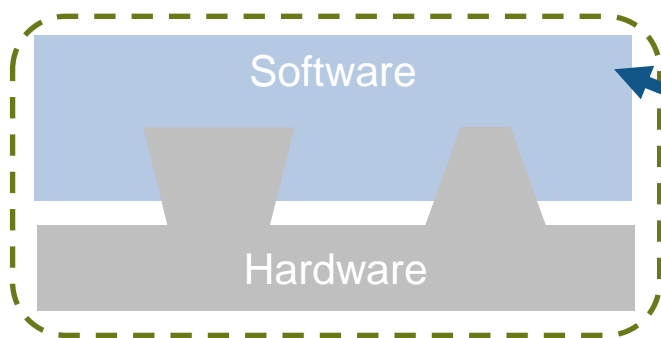
- Service-oriented architectures require a **change of mindset**
 - Shift from time-driven to event-driven execution
- **Centralize, re-architect** existing applications and partition in processes and services
 - e.g. Centralize energy management and path planning
- **Reuse of existing expertise**, workflows and software assets (don't start from scratch)
 - Migrate software components from AUTOSAR Classic to AUTOSAR Adaptive

MathWorks is collaborating with OEMs and Suppliers to address these challenges

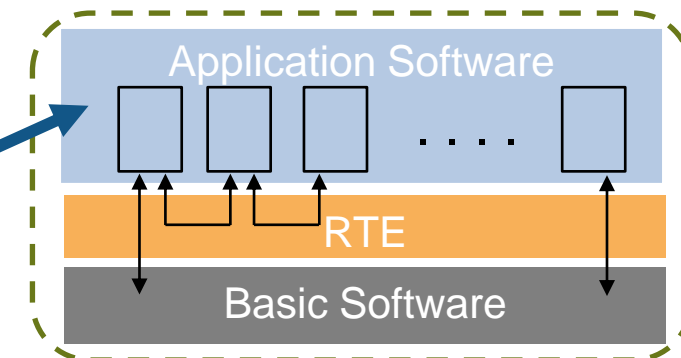
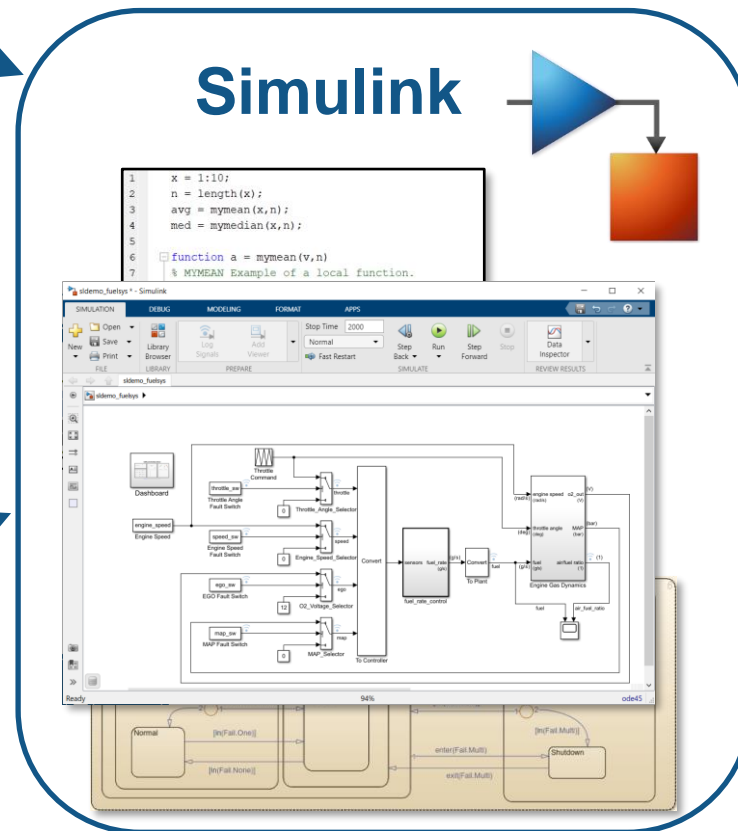
Simulink: Deploy software to different targets and standards



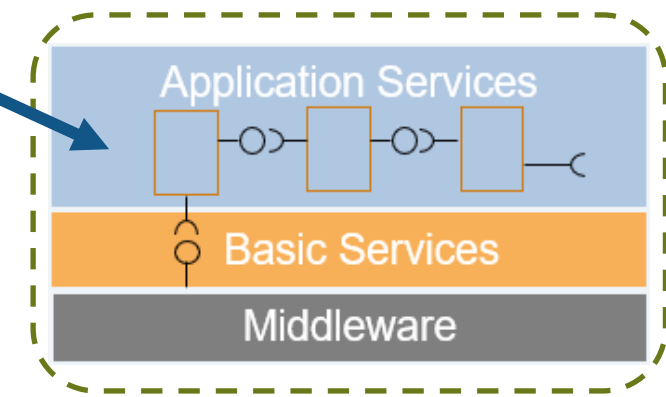
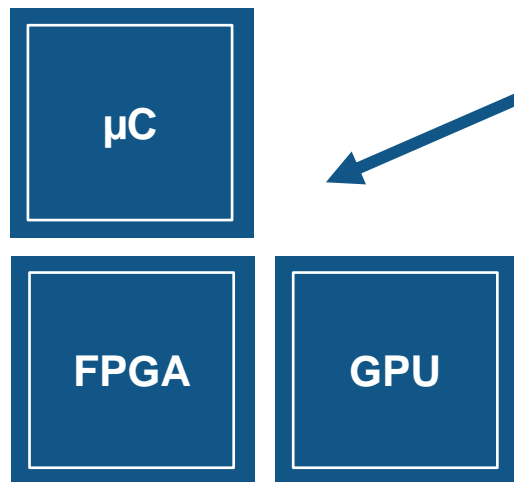
Simulink: Deploy software to different targets and standards



Legacy ECU

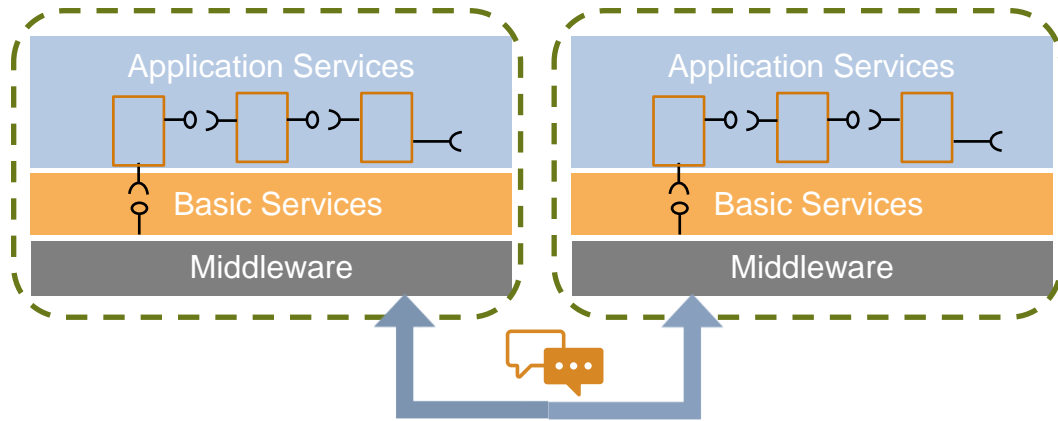


AUTOSAR Classic

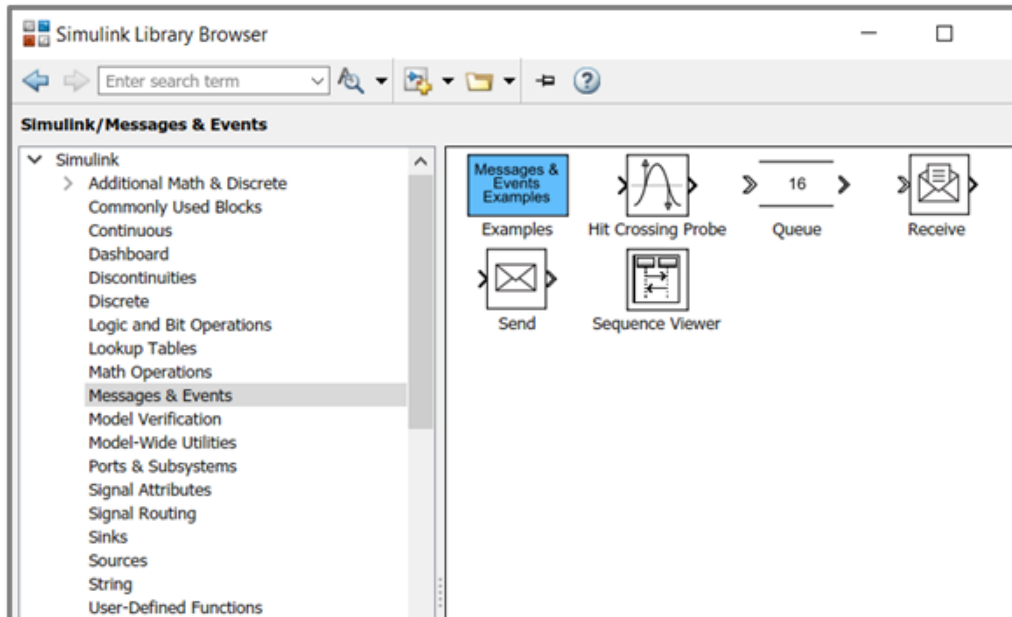
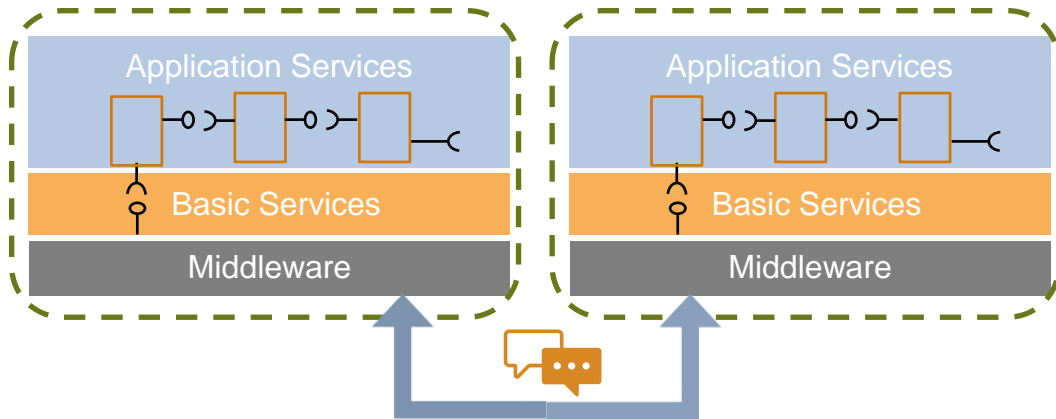


AUTOSAR Adaptive / ROS / DDS

Simulink Messages for Service-oriented communication

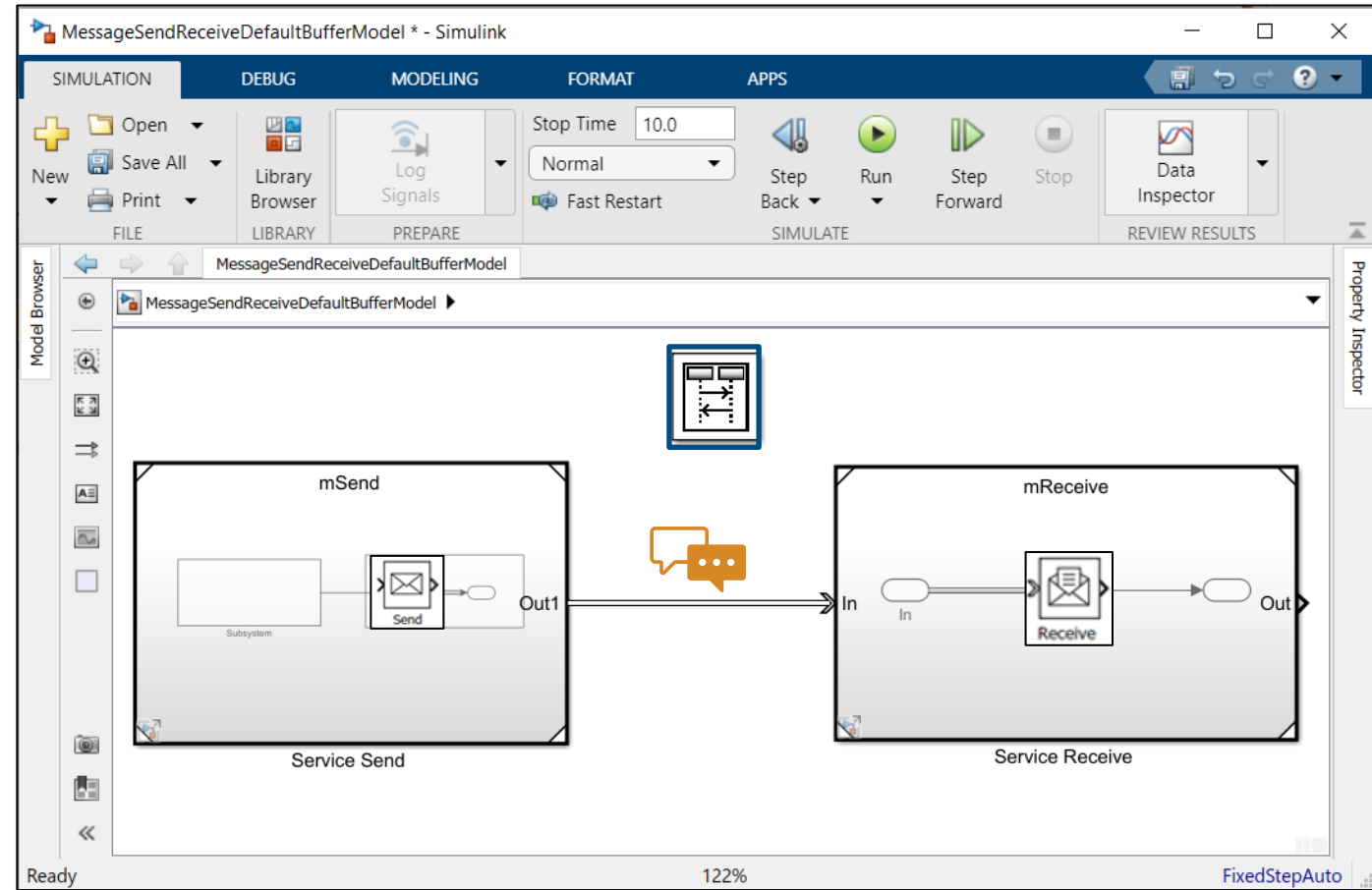
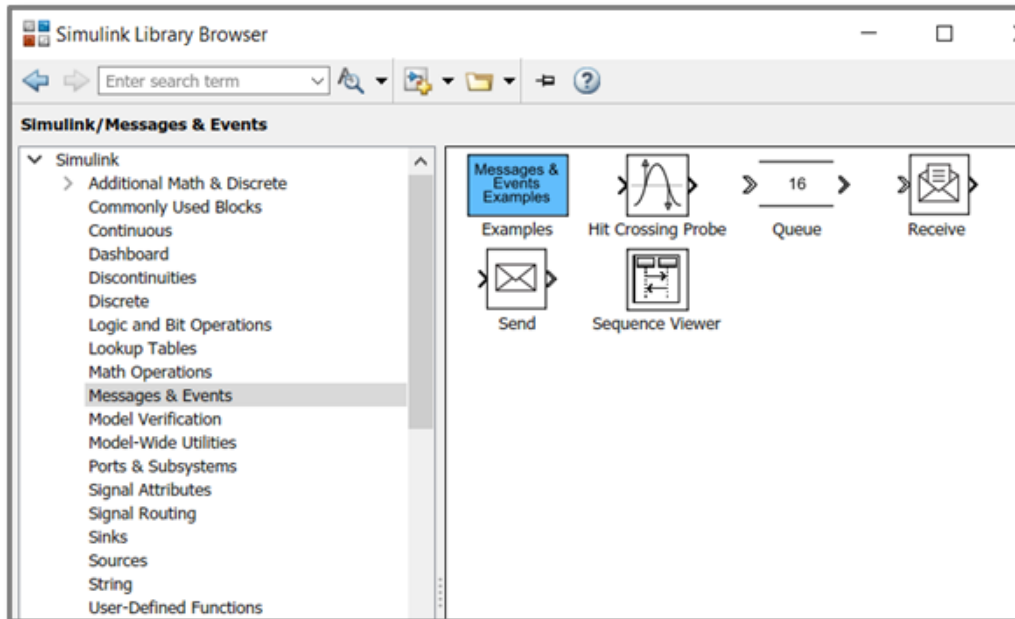
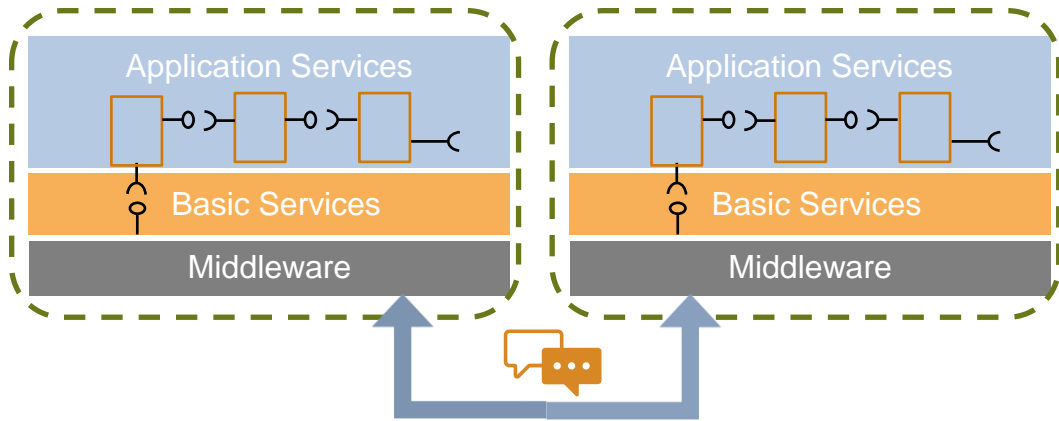


Simulink Messages for Service-oriented communication



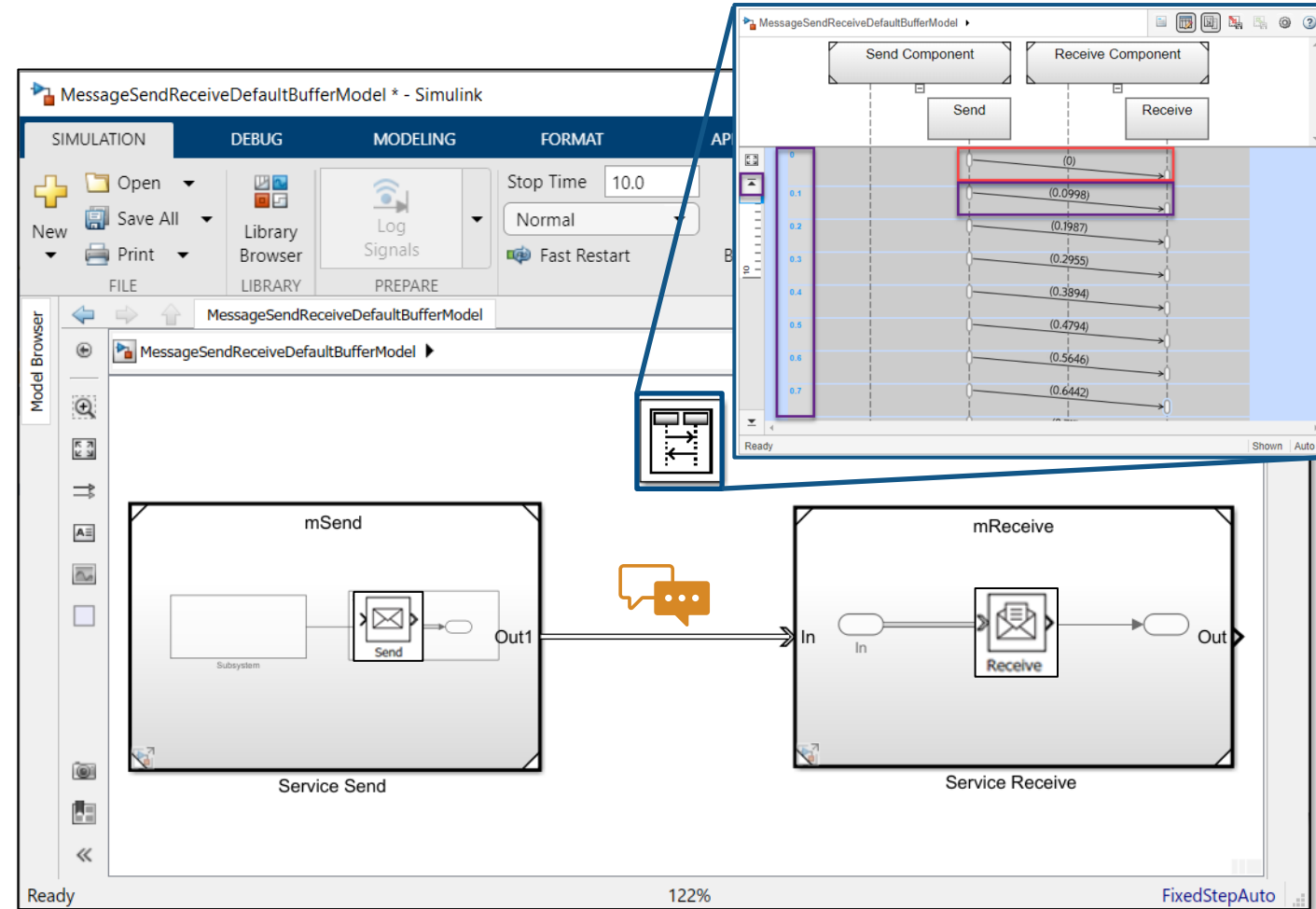
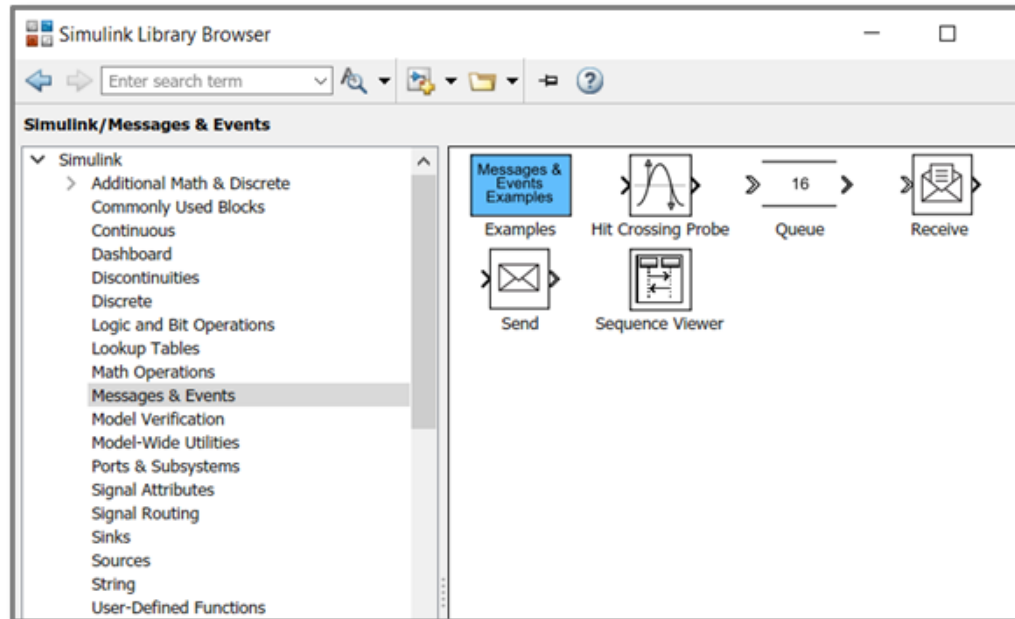
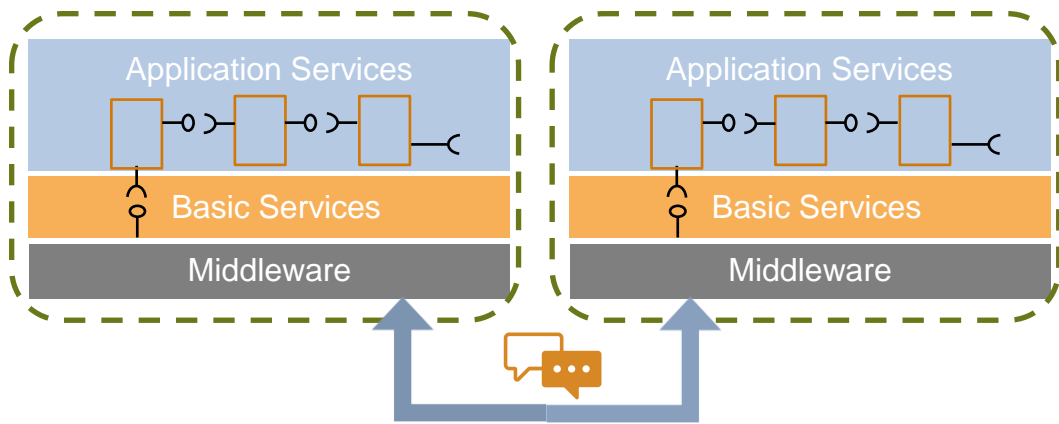
You can model service-oriented communication using messages (Send/Receive).

Simulink Messages for Service-oriented communication



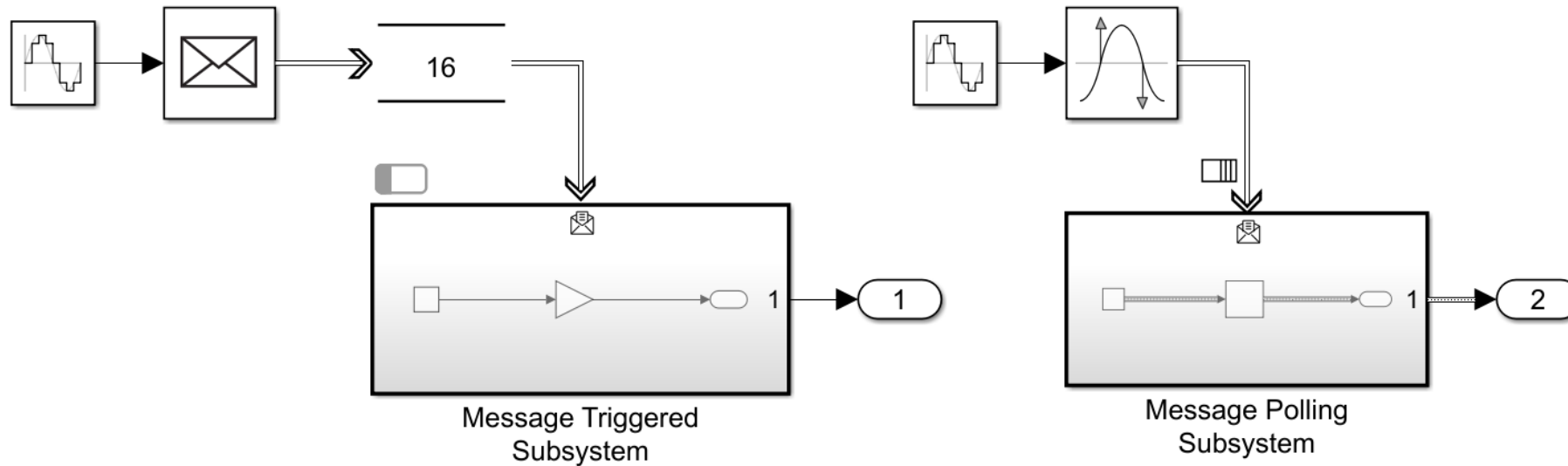
You can model service-oriented communication using messages (Send/Receive).

Simulink Messages for Service-oriented communication



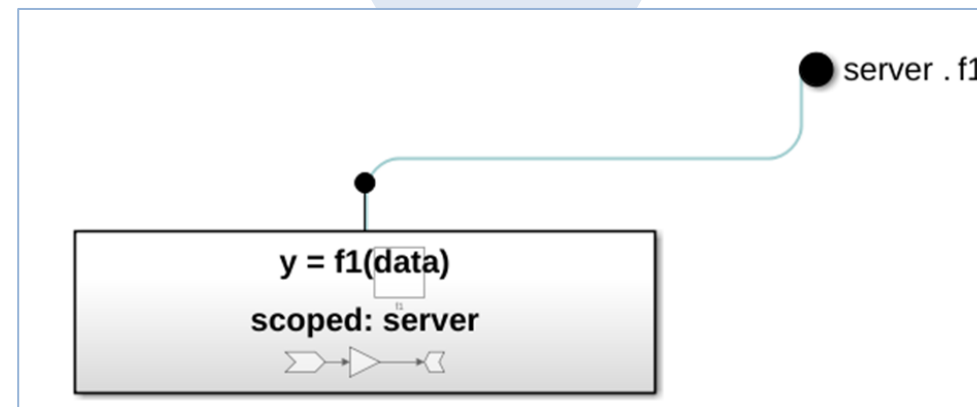
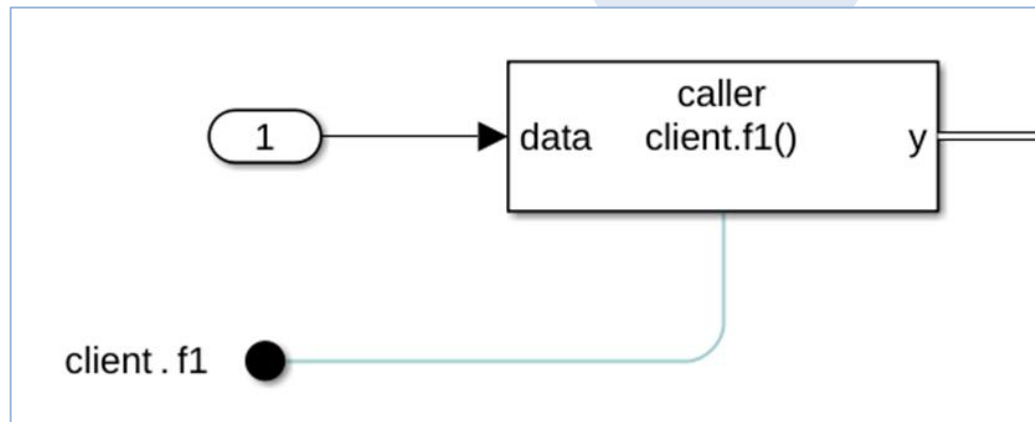
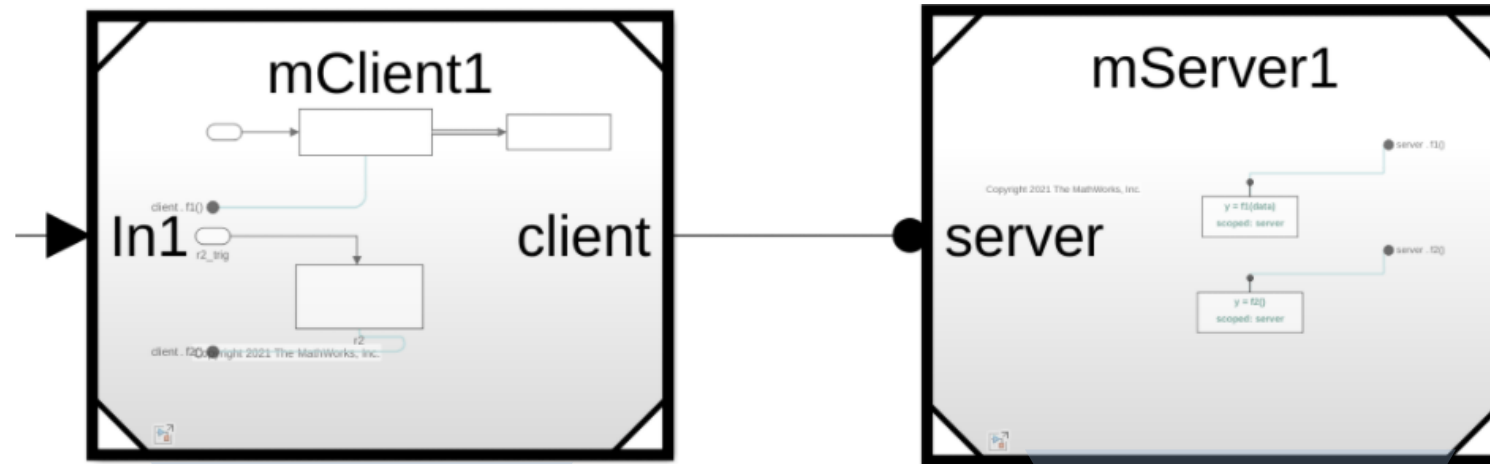
You can model service-oriented communication using messages (Send/Receive).

Message Triggered/Polling Subsystem for SOA



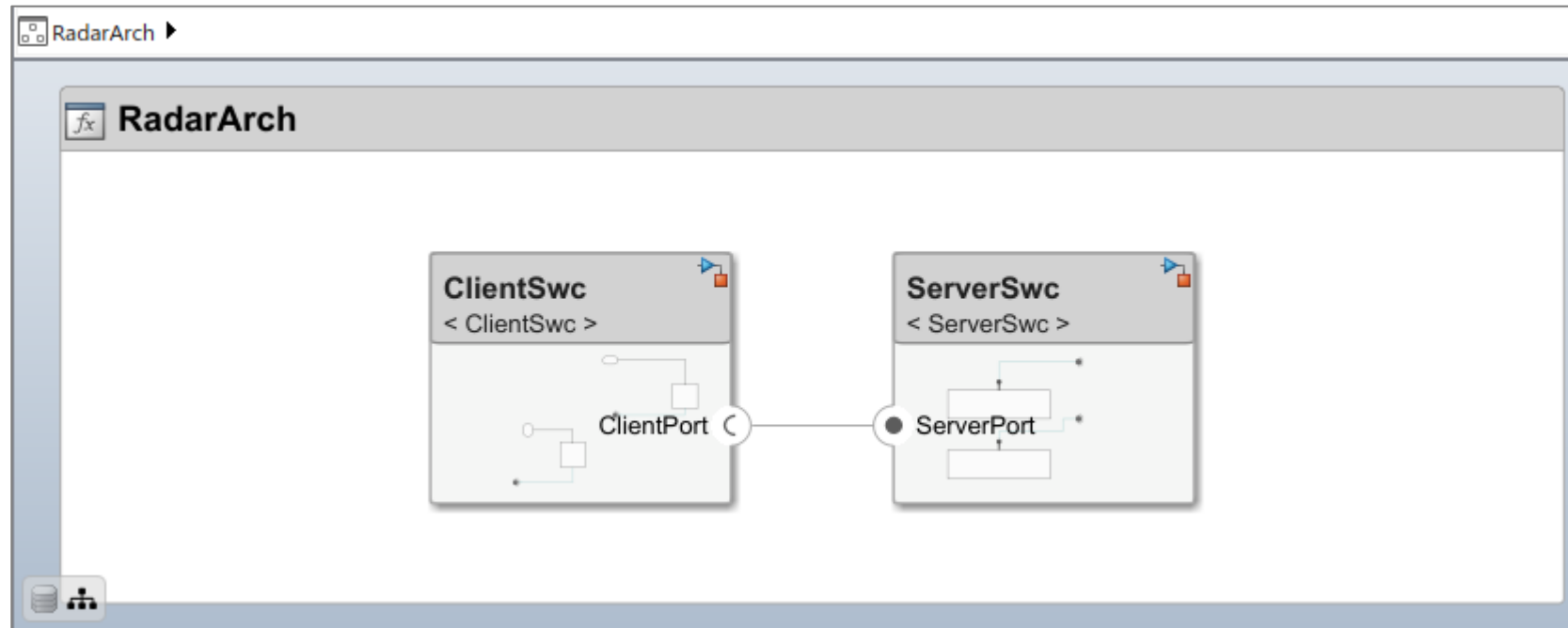
- New blocks to process messages by executing subsystem when message is available
- Model and generate code for components that are executed on message arrival

Function Ports for SOA



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

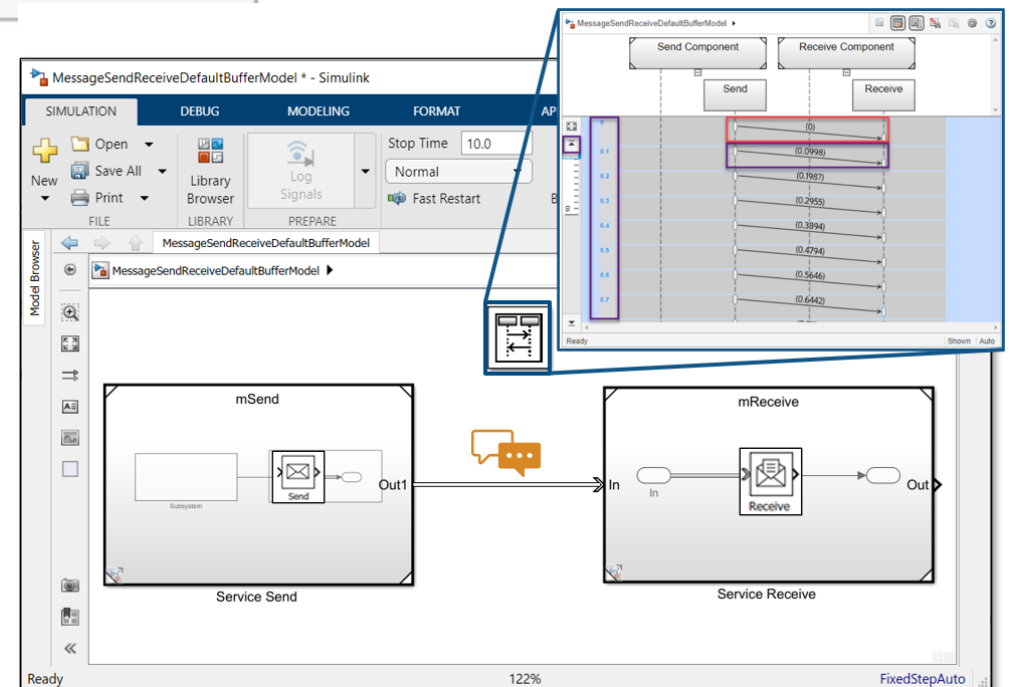
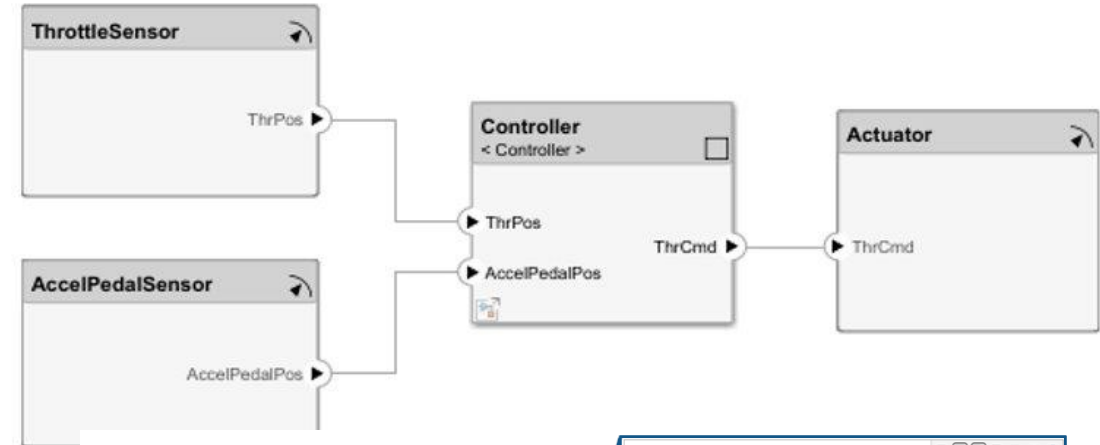
Author SOA applications in software architecture models



Model client-server connections between software components in software architectures in System Composer

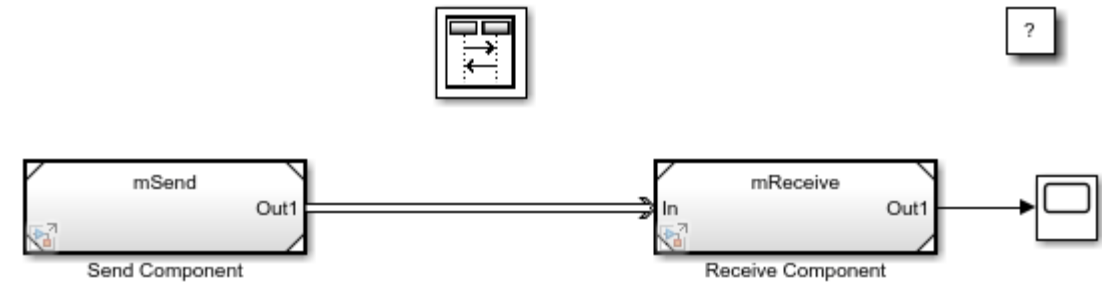
MathWorks investments in **SW architecture design and simulation**

- Intuitive, collaborative, graphical environment to design **software architectures**
 - Manage complexity
 - Maximize sharing and reuse
- High-level language to model and simulate **service-oriented applications**
 - Messages and queues
 - Client / server relationship
 - Sequence diagrams



Deploying SOA using C++

Generate SOA-based C and C++ application code from software services modeled in Simulink for deployment.



Copyright 2019 The MathWorks, Inc.



```
// Constructor
MessageSendReceiveDefaultBufferModelModelClass::
MessageSendReceiveDefaultBufferModelModelClass():
MessageSendReceiveDefaultBuff_B()
,MessageSendReceiveDefaultBuf_DW()
,ReceiveComponentRecvData(*this)
,SendComponentSendData(*this)
,Receive_ComponentMDLOBJ0(get_ReceiveComponentRecvData())
,Send_ComponentMDLOBJ1(get_SendComponentSendData())
,MessageSendReceiveDefaultBuf_M()
{
// Currently there is no constructor body generated.
}
```

```
// Forward declaration
class MessageSendReceiveDefaultBufferModelModelClass;
class
MessageSendReceiveDefaultBufferModelModelClassMessa_ReceiveComponent_RecvDataT
: public RecvData_real_T
{
private:
MessageSendReceiveDefaultBufferModelModelClass & aProvider;
public:
MessageSendReceiveDefaultBufferModelModelClassMessa_ReceiveComponent_RecvDataT
(MessageSendReceiveDefaultBufferModelModelClass & aProvider);
virtual void RecvData(real_T *data, int32_T length, int32_T *status);
};

class
MessageSendReceiveDefaultBufferModelModelClassMessa_ReceiveComponent_SendDataT
: public SendData_real_T
{
private:
MessageSendReceiveDefaultBufferModelModelClass & aProvider;
public:
MessageSendReceiveDefaultBufferModelModelClassMessa_ReceiveComponent_SendDataT
(MessageSendReceiveDefaultBufferModelModelClass & aProvider);
virtual void SendData(const real_T *data, int32_T length, int32_T *status);
};
```

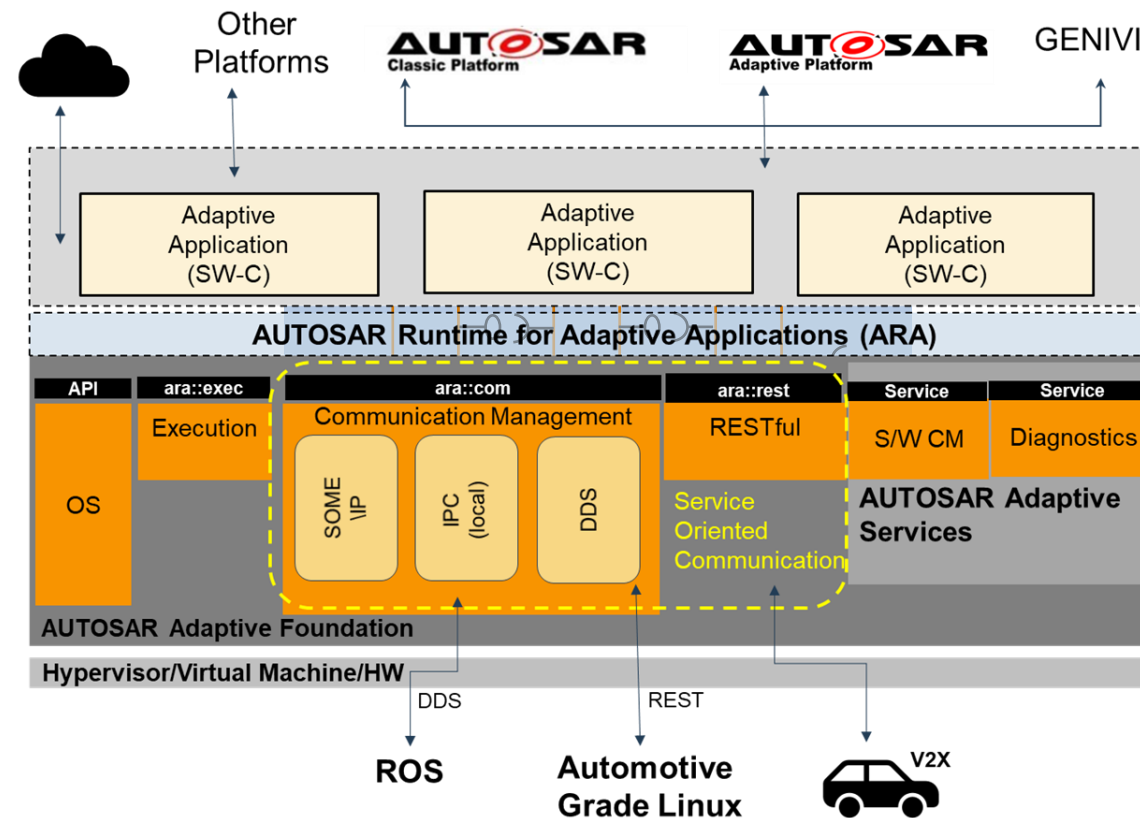
>> [Generate C++ Messages to Communicate Data Between Simulink Components](#)

Agenda

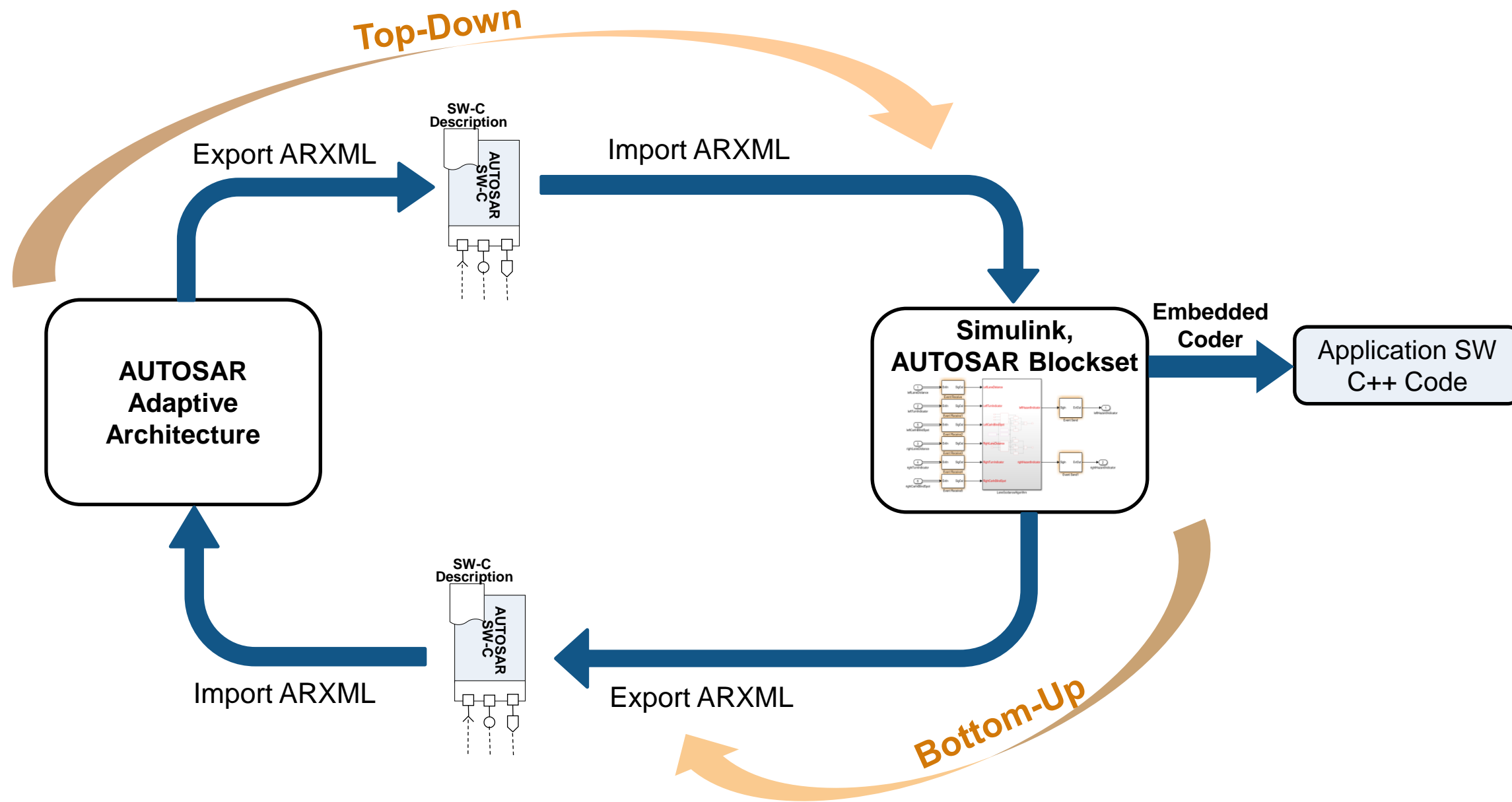
- Software-defined vehicles and new architectures (SOA)
- SOA Concepts
- **MathWorks Solutions for SOA**
 - Adaptive AUTOSAR
 - DDS/ROS
- Conclusions and key takeaways

AUTOSAR Adaptive

AUTOSAR Adaptive Platform implements the AUTOSAR Runtime for Adaptive Applications (ARA) for automotive industry.

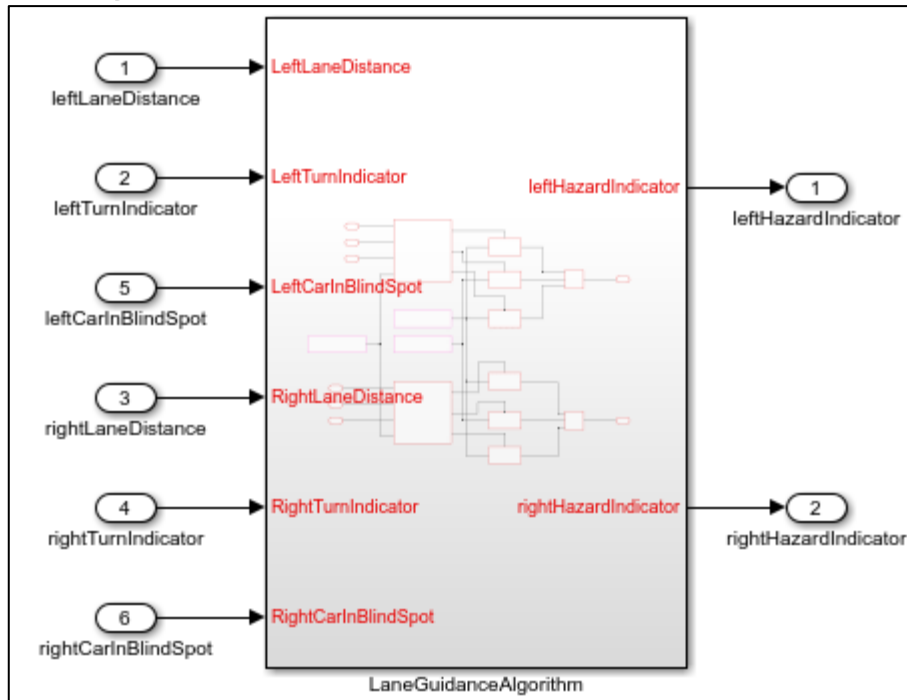


AUTOSAR Adaptive workflows



AUTOSAR Adaptive in action

Legacy Simulink model



Bottom-Up

OR

Start from an AUTOSAR Adaptive ARXML

```

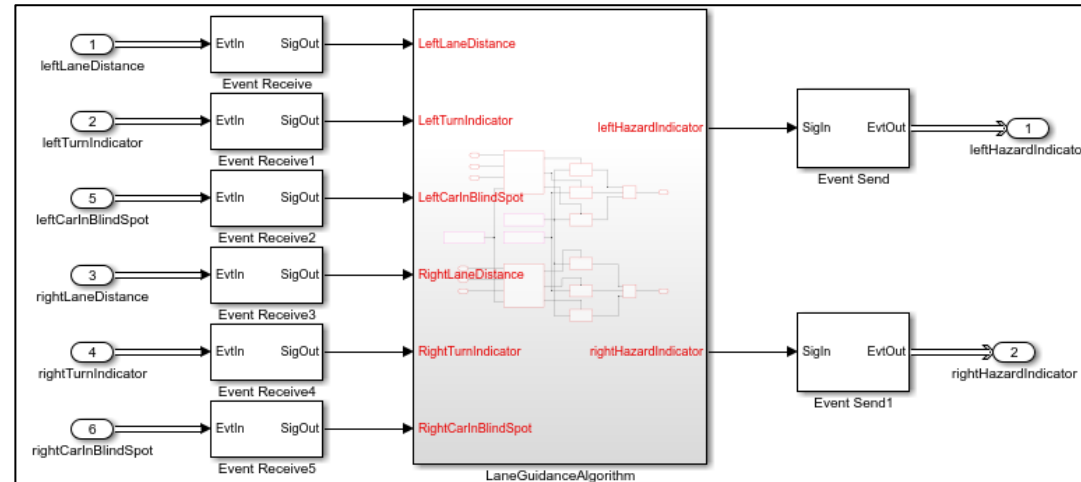
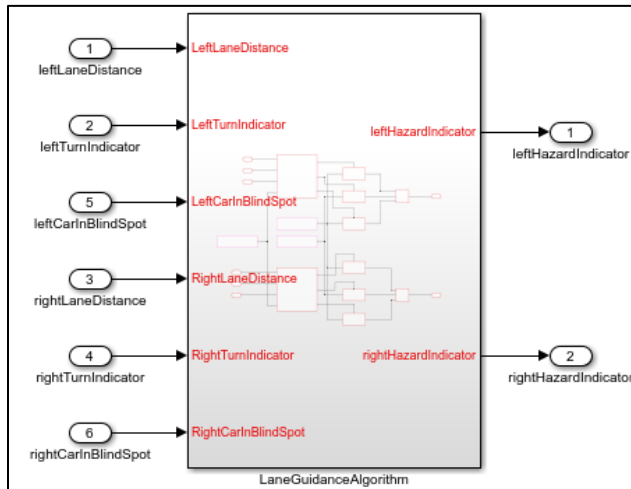
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 Auto generated XML Component Description for model autosar_LaneGuidance
4 Model version : 1.224
5 Simulink Coder version : Simulink Coder 9.2 (R2019b) 23-May-2019
6 XML source code generated on : Wed Jul 24 16:11:51 2019
7 Model Checksum : 3376303272 3457889089 3078584661 1517304406
8 -->
9 <AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
10 <AR-PACKAGES>
11 <AR-PACKAGE>
12 <SHORT-NAME>LaneGuidance_pkg</SHORT-NAME>
13 <AR-PACKAGES>
14 <AR-PACKAGE>
15 <SHORT-NAME>LaneGuidance_sw</SHORT-NAME>
16 <ELEMENTS>
17 <ADAPTIVE-APPLICATION-SW-COMPONENT-TYPE UUID="6574ed24-7dad-53cc-e7ac-01f60699f406">
18 <SHORT-NAME>LaneGuidance</SHORT-NAME>
19 <PORTS>
20 <R-PORT-PROTOTYPE UUID="a8adc3c3-bbb1-575e-fbc6-0fcf8164f622">
21 <SHORT-NAME>RequiredPort</SHORT-NAME>
22 <REQUIRED-COM-SPECS>
23 <QUEUED-RECEIVER-COM-SPEC>
24 <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE"/>LaneGuidance_pkg/LaneGuidance_if/R
25 <HANDLE-OUT-OF-RANGE>NONE</HANDLE-OUT-OF-RANGE>
26 <USES-END-TO-END-PROTECTION>false</USES-END-TO-END-PROTECTION>
27 <QUEUE-LENGTH>1</QUEUE-LENGTH>
28 </QUEUED-RECEIVER-COM-SPEC>

```

Top-Down

AUTOSAR Adaptive in action: bottom-up

Add blocks to make the necessary event and signal connections



Add Message ports and Message-Signal conversion blocks

Modelling an AUTOSAR Adaptive application in Simulink

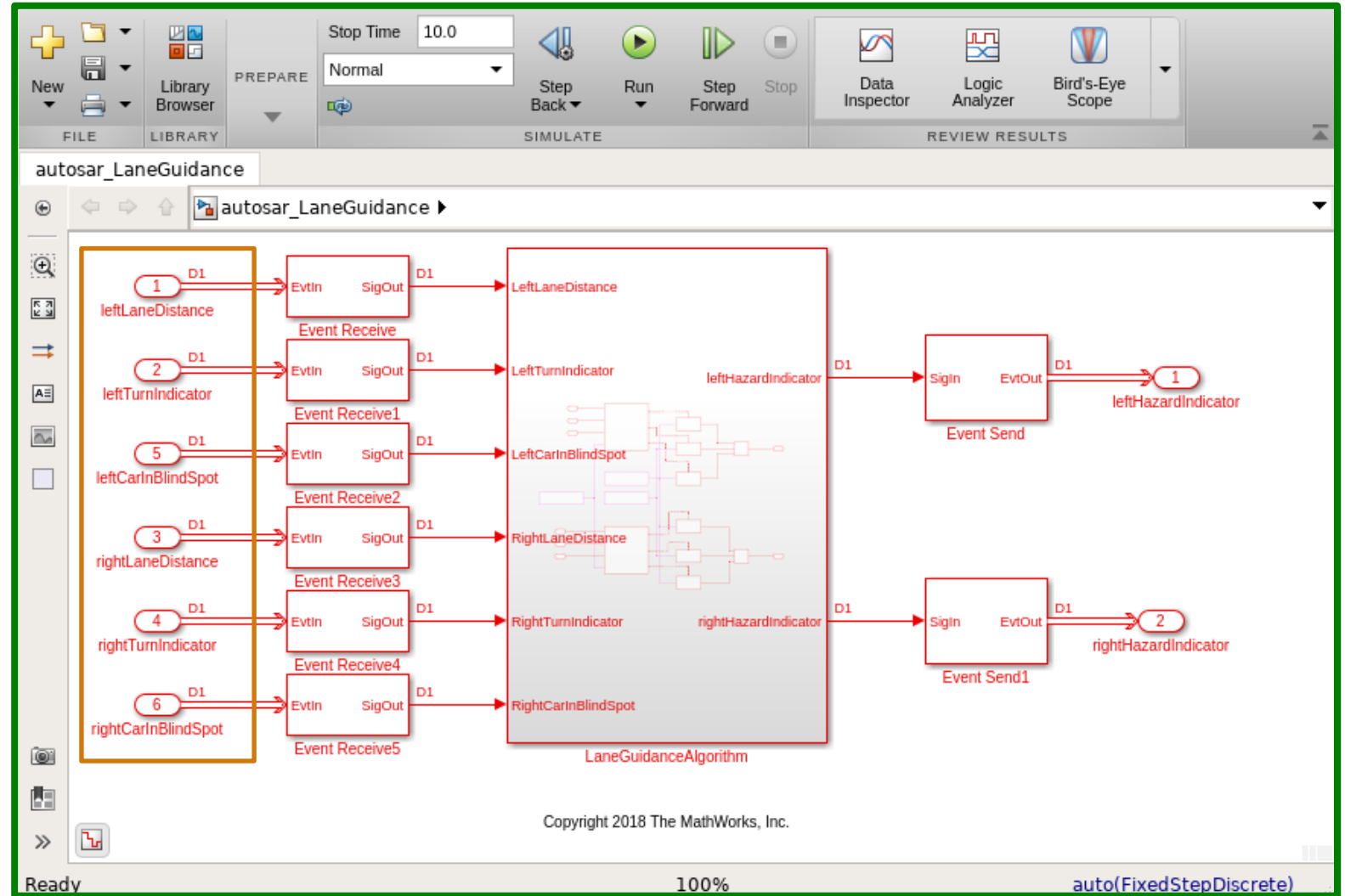


Adaptive Application

RequiredPort

```

"Radar" : {
  // events
  "event" : {
    "leftLaneDistance"
    "leftTurnIndicator"
    "leftCarInBlindSpot"
    "rightLandDistance"
    "rightTurnIndicator"
    "rightCarInBlindSpot"
  },
  // methods
  "method" : {
    "Calibrate"
    "Adjust"
  },
  // fields
  "field" : {
    "updateRate"
  }
}
    
```



Modelling an AUTOSAR Adaptive application in Simulink

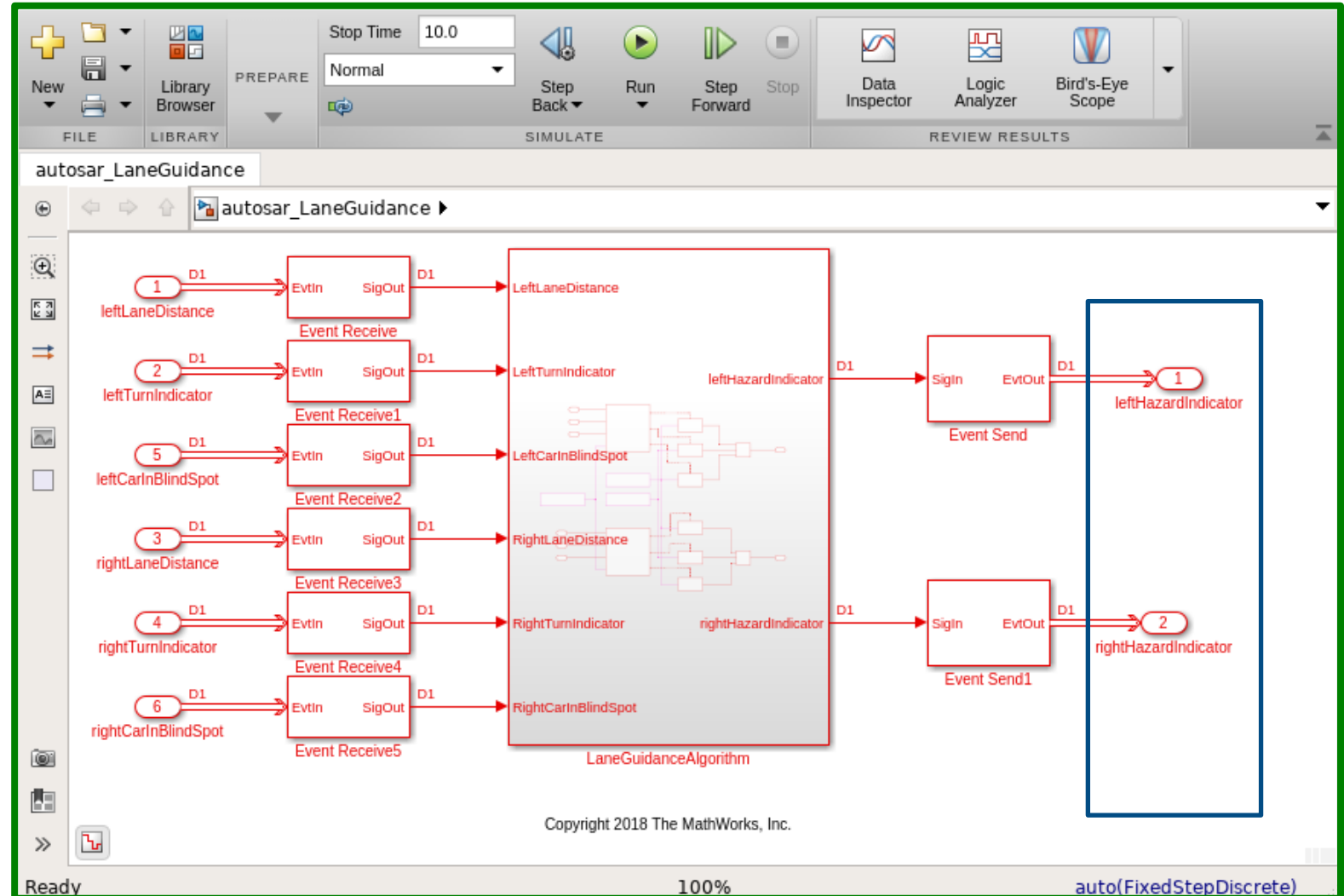


Adaptive Application

ProvidedPort

```

"Hazard" : {
  // events
  "event" : {
    "leftHazardIndicator"
    "rightHazardIndicator"
  },
  // methods
  "method" : { },
  // fields
  "field" : { }
}
    
```



Modelling an AUTOSAR Adaptive application in Simulink



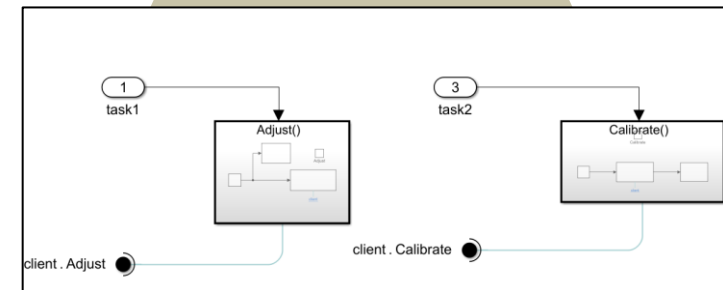
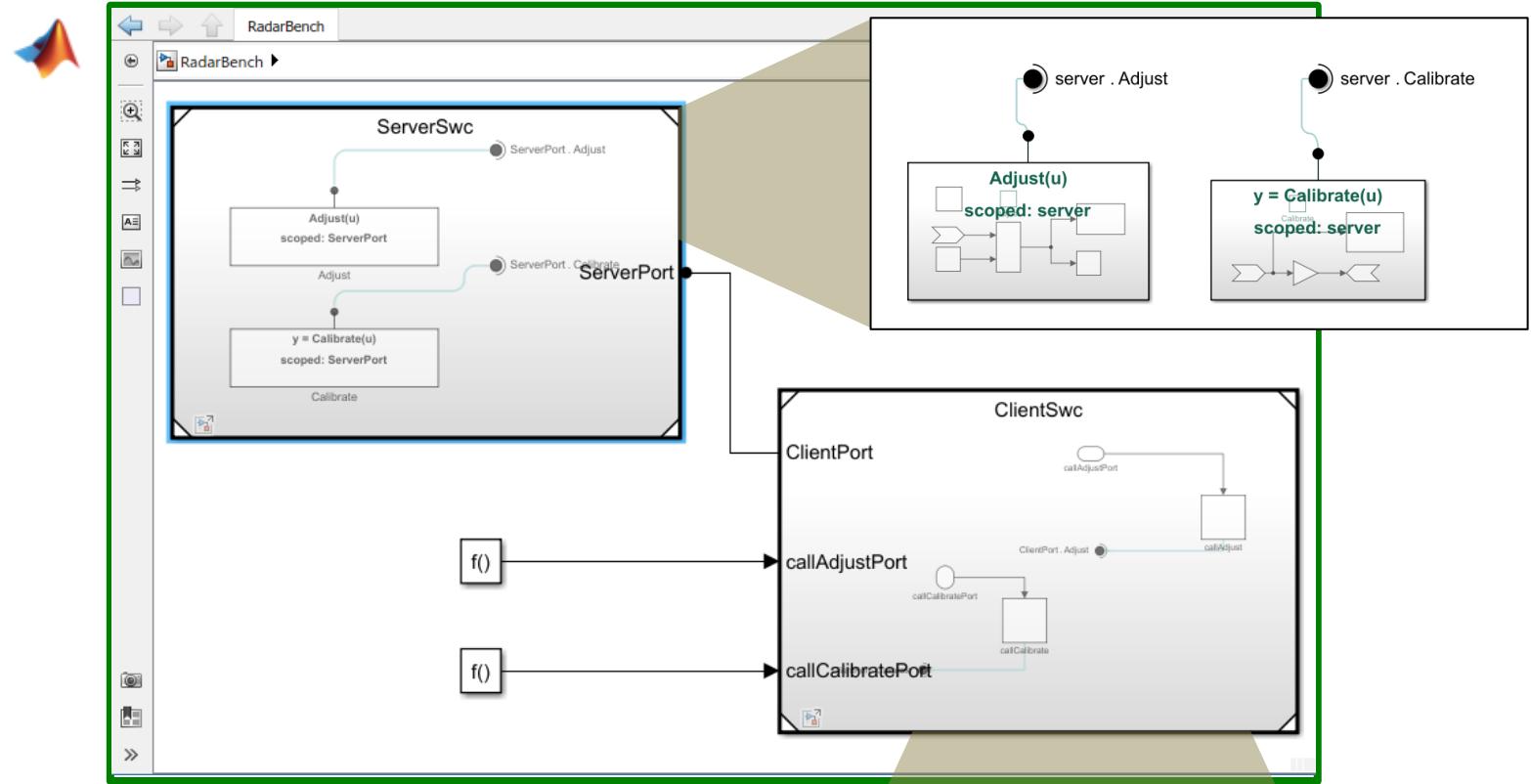
Client

Server



```

"Hazard" : {
  // events
  "event" : { {} },
  // methods
  "method" : {
    "calibrate"
    "adjust"
  },
  // fields
  "field" : { {} }
}
  
```

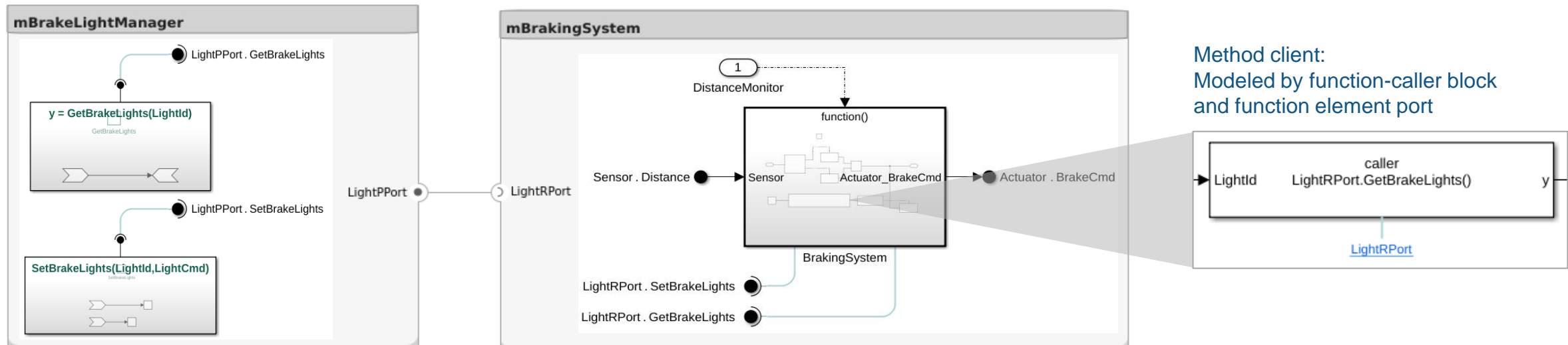


Model synchronous/blocking methods

- Use case
 - Client needs method results to proceed
 - Blocks on method call
- Simulink supports modelling of methods
 - Blocking Request-Response methods
 - Fire-Forget Methods

Client Pseudo-code:

```
void mBrakingSystem::DistanceMonitor ()
{
  ...
  auto GetBrakeLightsFuture = LightRPort->GetBrakeLights(1);
  auto GetBrakeLightsResult = GetBrakeLightsFuture.GetResult();
  if (GetBrakeLightsResult.HasValue()) {
    GetBrakeLights::Output callOutput = GetBrakeLightsResult.Value();
    rtb_FunctionCaller1 = callOutput.y;
  }
  ...
}
```



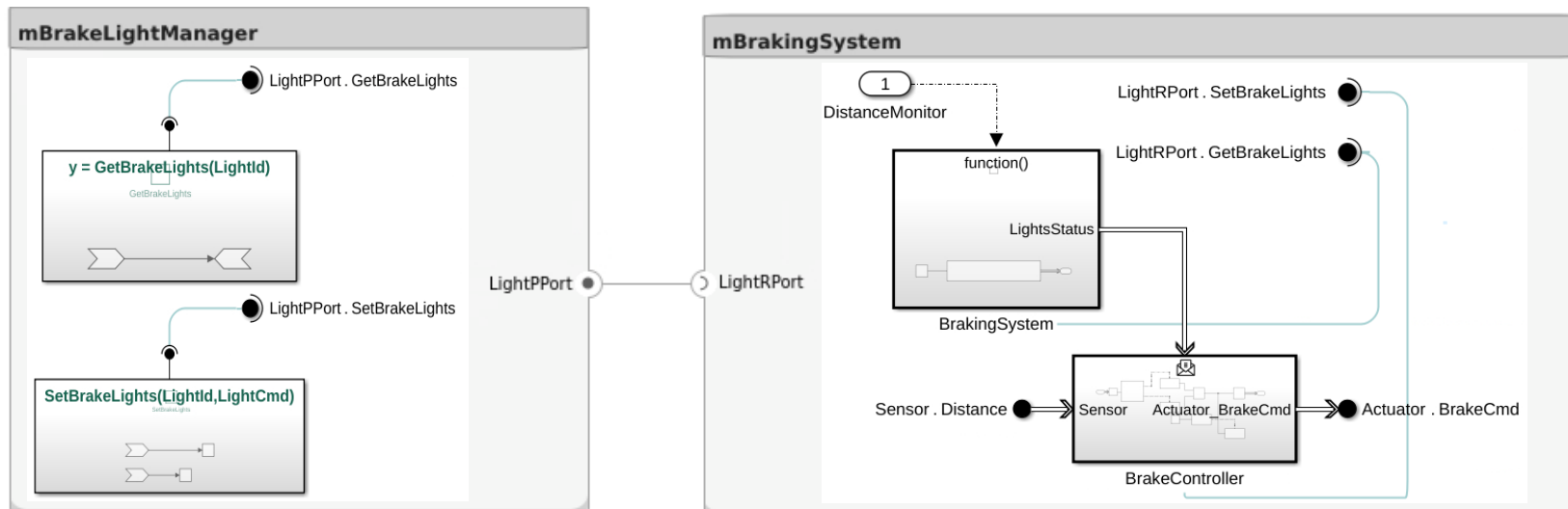
Model asynchronous/non-blocking methods

- Use case
 - Clients need not wait for method results
 - Register a call-back to process method output
- Simulink supports modelling of methods
 - Non-blocking Request-Response methods

Client Pseudo-code:

```
void mBrakingSystem::DistanceMonitor ()
{
  ...
  auto GetBrakeLightsFuture = LightRPort->GetBrakeLights(1);
  auto callbackGetBrakeLightsFuture = GetBrakeLightsFuture.then(
    &mAsyncBrakingSystem::GetBrakeLightsCallback);
  ...
}
```

Method call and callback registration

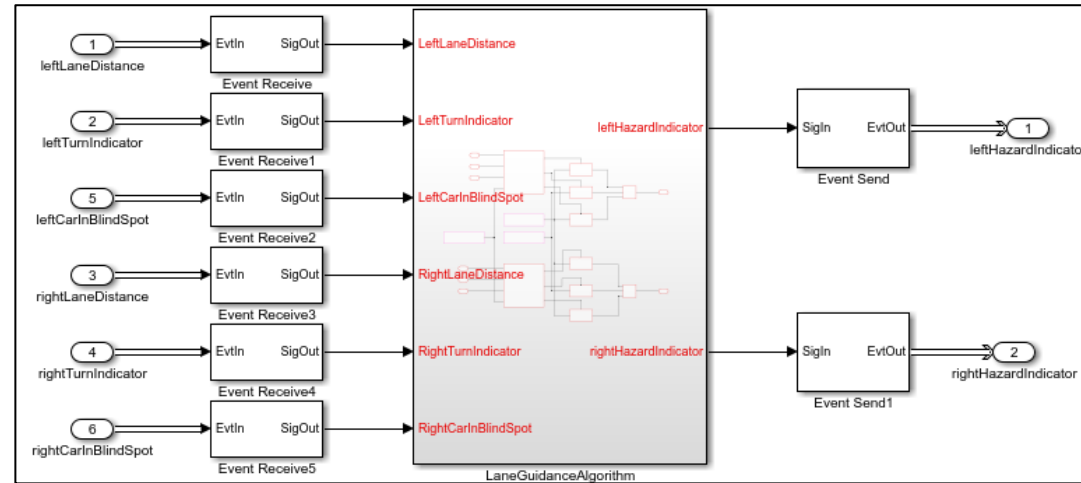
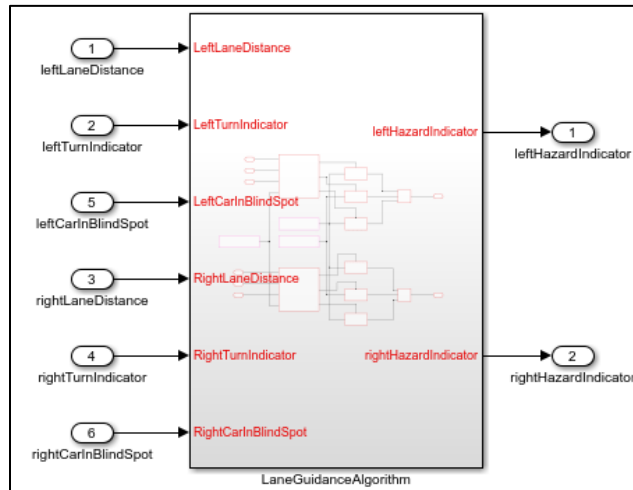


```
void mAsyncBrakingSystem::GetBrakeLightsCallback (
  ara::core::Future<GetBrakeLights::Output>
  futureObj)
{
  auto GetBrakeLightsResult = futureObj.GetResult ();
  if (GetBrakeLightsResult.HasValue ()) {
    GetBrakeLights::Output callOutput =
      GetBrakeLightsResult.Value ();
    BrakeController (callOutput.y);
  }
}
```

Method callback body

AUTOSAR Adaptive in action: bottom-up

Add blocks to make the necessary event and signal connections



AUTOSAR Quick Start

The screenshot shows the 'AUTOSAR Component Quick Start' dialog box. The 'Set Component' tab is active. The 'Finish' button is visible. The 'What to consider' section contains the following text: 'AUTOSAR Component Quick Start maps a Simulink model to an AUTOSAR software component. For the component, specify an AUTOSAR short name, package path, and component type, or accept default values. Package paths can use an organizational naming pattern, such as /Company/Powertrain/Components. Component type determines the APIs available to the component in the run-time environment.'

The 'Configure AUTOSAR software component properties' section is expanded, showing the following configuration:

- Map model to AUTOSAR software component (Adaptive)
- Component name:
- Component package:

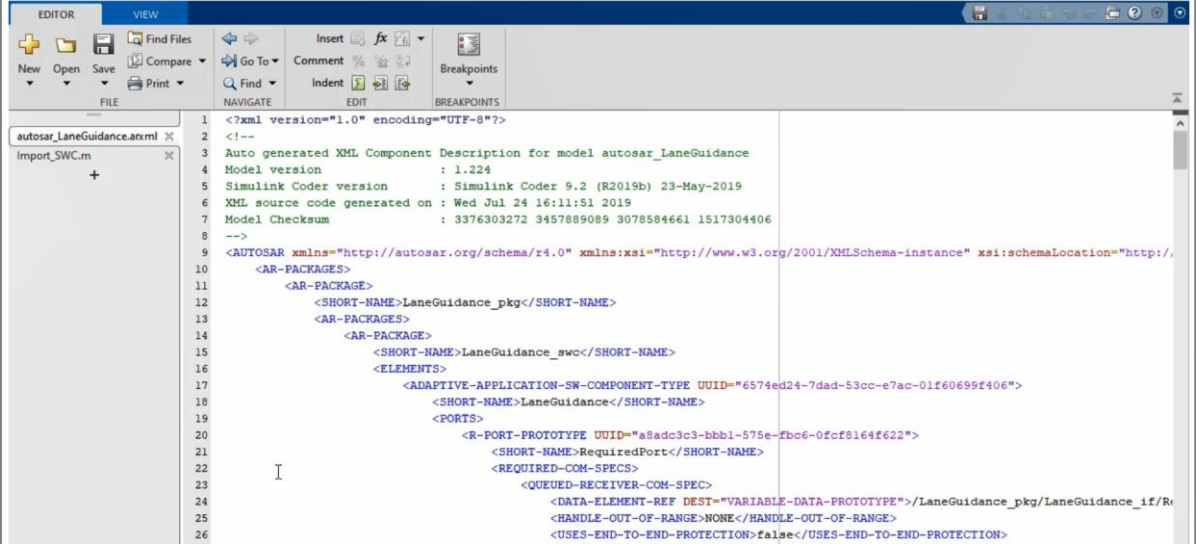
The screenshot shows the Simulink model with the 'Property Inspector' and 'Code Mappings' windows open. The 'Property Inspector' shows the 'Design' and 'Code' tabs. The 'Code Mappings' window shows the following table:

Imports	Source	Port	Event
leftLaneDistance	leftLaneDistance	RequiredPort	LeftLaneDistance
leftTurnIndicator	leftTurnIndicator	RequiredPort	LeftTurnIndicator
rightLaneDistance	rightLaneDistance	RequiredPort	RightLaneDistance
rightTurnIndicator	rightTurnIndicator	RequiredPort	RightTurnIndicator
leftCarInBlindSpot	leftCarInBlindSpot	RequiredPort	LeftCarInBlindSpot
rightCarInBlindSpot	rightCarInBlindSpot	RequiredPort	RightCarInBlindSpot



AUTOSAR Adaptive in action: top-down

- Create model from ARXML



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 Auto generated XML Component Description for model autosar_LaneGuidance
4 Model version : 1.224
5 Simulink Coder version : Simulink Coder 9.2 (R2019b) 23-May-2019
6 XML source code generated on : Wed Jul 24 16:11:51 2019
7 Model Checksum : 3376303272 3457889089 3078584661 1517304406
8 -->
9 <AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
10 <AR-PACKAGES>
11 <AR-PACKAGE>
12 <SHORT-NAME>LaneGuidance_pkg</SHORT-NAME>
13 <AR-PACKAGES>
14 <AR-PACKAGE>
15 <SHORT-NAME>LaneGuidance_sw</SHORT-NAME>
16 <ELEMENTS>
17 <ADAPTIVE-APPLICATION-SW-COMPONENT-TYPE UUID="6574ed24-7dad-53cc-e7ac-01f60699f406">
18 <SHORT-NAME>LaneGuidance</SHORT-NAME>
19 <PORTS>
20 <R-PORT-PROTOTYPE UUID="a8adc3c3-bbb1-575e-fbc6-0f0f8164f622">
21 <SHORT-NAME>RequiredPort</SHORT-NAME>
22 <REQUIRED-COM-SPECS>
23 <QUEUED-RECEIVER-COM-SPEC>
24 <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE"/>LaneGuidance_pkg/LaneGuidance_if/R
25 <HANDLE-OUT-OF-RANGE>NONE</HANDLE-OUT-OF-RANGE>
26 <USES-END-TO-END-PROTECTION>false</USES-END-TO-END-PROTECTION>
```

```
ar = arxml.importer({'fusion_app.arxml', 'radarService_app_mod.arxml', 'radar_svc_mod.arxml', 'stdtypes_mod.arxml'});
names = getComponentNames(ar)
```

AUTOSAR Adaptive in action: top-down

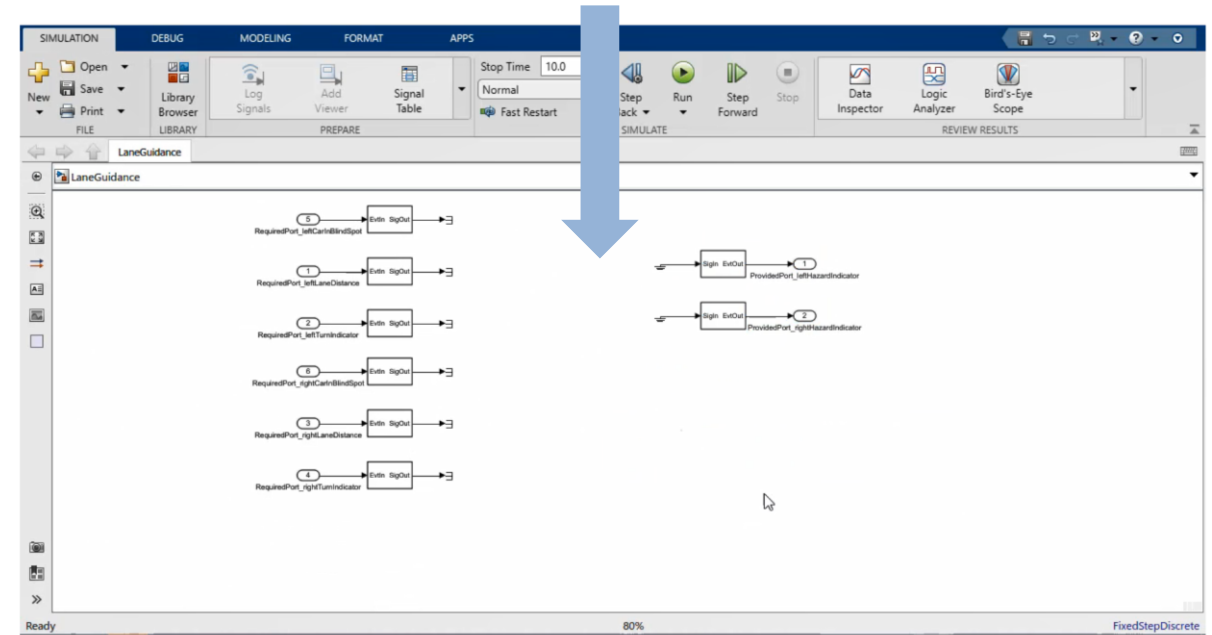
- Create model from ARXML

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 Auto generated XML Component Description for model autosar_LaneGuidance
4 Model version : 1.224
5 Simulink Coder version : Simulink Coder 9.2 (R2019b) 23-May-2019
6 XML source code generated on : Wed Jul 24 16:11:51 2019
7 Model Checksum : 3376303272 3457889089 3078584661 1517304406
8 -->
9 <AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
10 <AR-PACKAGES>
11 <AR-PACKAGE>
12 <SHORT-NAME>LaneGuidance_pkg</SHORT-NAME>
13 <AR-PACKAGES>
14 <AR-PACKAGE>
15 <SHORT-NAME>LaneGuidance_sw</SHORT-NAME>
16 <ELEMENTS>
17 <ADAPTIVE-APPLICATION-SW-COMPONENT-TYPE UUID="6574ed24-7dad-53cc-e7ac-01f60699f406">
18 <SHORT-NAME>LaneGuidance</SHORT-NAME>
19 <PORTS>
20 <R-PORT-PROTOTYPE UUID="a8adc3c3-bbb1-575e-fbc6-0fcf8164f622">
21 <SHORT-NAME>RequiredPort</SHORT-NAME>
22 <REQUIRED-COM-SPECS>
23 <QUEUED-RECEIVER-COM-SPEC>
24 <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE"/>LaneGuidance_pkg/LaneGuidance_if/R
25 <HANDLE-OUT-OF-RANGE>NONE</HANDLE-OUT-OF-RANGE>
26 <USES-END-TO-END-PROTECTION>false</USES-END-TO-END-PROTECTION>

```

```
createComponentAsModel(ar, '/RadarFusion/fusion');
```



AUTOSAR Adaptive in action: top-down

- Create model from ARXML

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 Auto generated XML Component Description for model autosar_LaneGuidance
4 Model version : 1.224
5 Simulink Coder version : Simulink Coder 9.2 (R2019b) 23-May-2019
6 XML source code generated on : Wed Jul 24 16:11:51 2019
7 Model Checksum : 3376303272 3457889089 3078584661 1517304406
8 -->
9 <AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
10 <AR-PACKAGES>
11 <AR-PACKAGE>
12 <SHORT-NAME>LaneGuidance_pkg</SHORT-NAME>
13 <AR-PACKAGES>
14 <AR-PACKAGE>
15 <SHORT-NAME>LaneGuidance_sw</SHORT-NAME>
16 <ELEMENTS>
17 <ADAPTIVE-APPLICATION-SW-COMPONENT-TYPE UUID="6574ed24-7dad-53cc-e7ac-01f60699f406">
18 <SHORT-NAME>LaneGuidance</SHORT-NAME>
19 <PORTS>
20 <R-PORT-PROTOTYPE UUID="a8adc3c3-bbb1-575e-fbc6-0f0f8164f622">
21 <SHORT-NAME>RequiredPort</SHORT-NAME>
22 <REQUIRED-COM-SPECS>
23 <QUEUED-RECEIVER-COM-SPEC>
24 <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE"/>LaneGuidance_pkg/LaneGuidance_if/R
25 <HANDLE-OUT-OF-RANGE>NONE</HANDLE-OUT-OF-RANGE>
26 <USES-END-TO-END-PROTECTION>false</USES-END-TO-END-PROTECTION>
    
```

Configuration Parameters: LaneGuidance/Configuration (Active)

Search

Generate XML file for schema version: 00046 (R18-10)

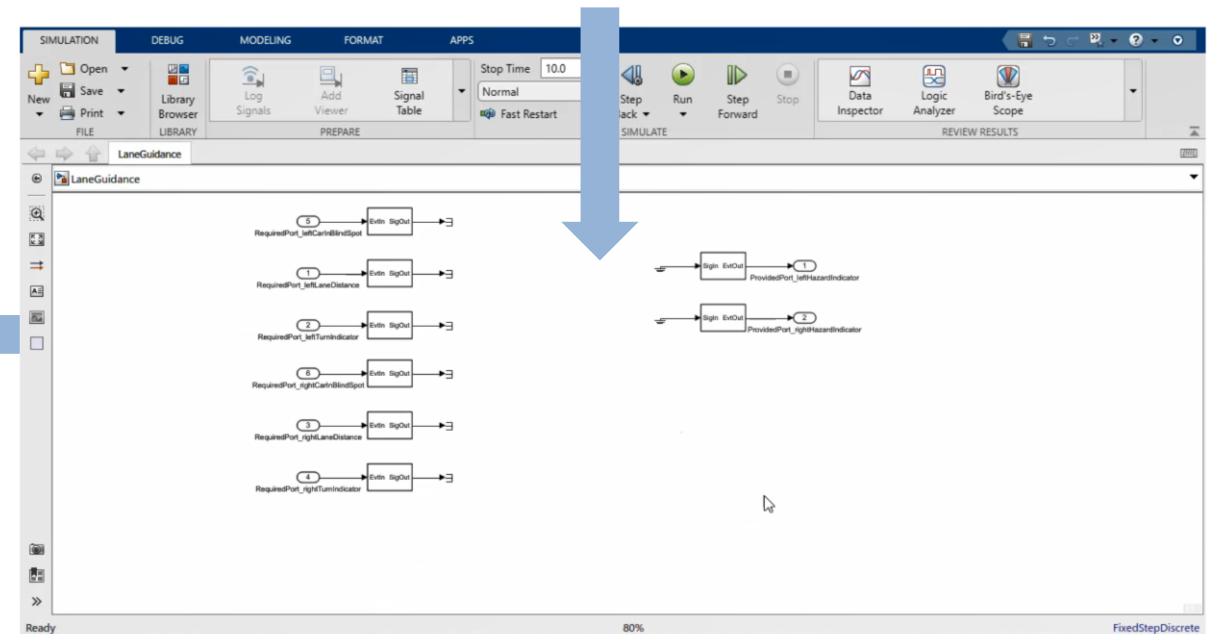
Maximum SHORT-NAME length: 128 00046 (R18-10)

XCP Slave Configuration 00047 (R19-03)

00048 (R19-11)

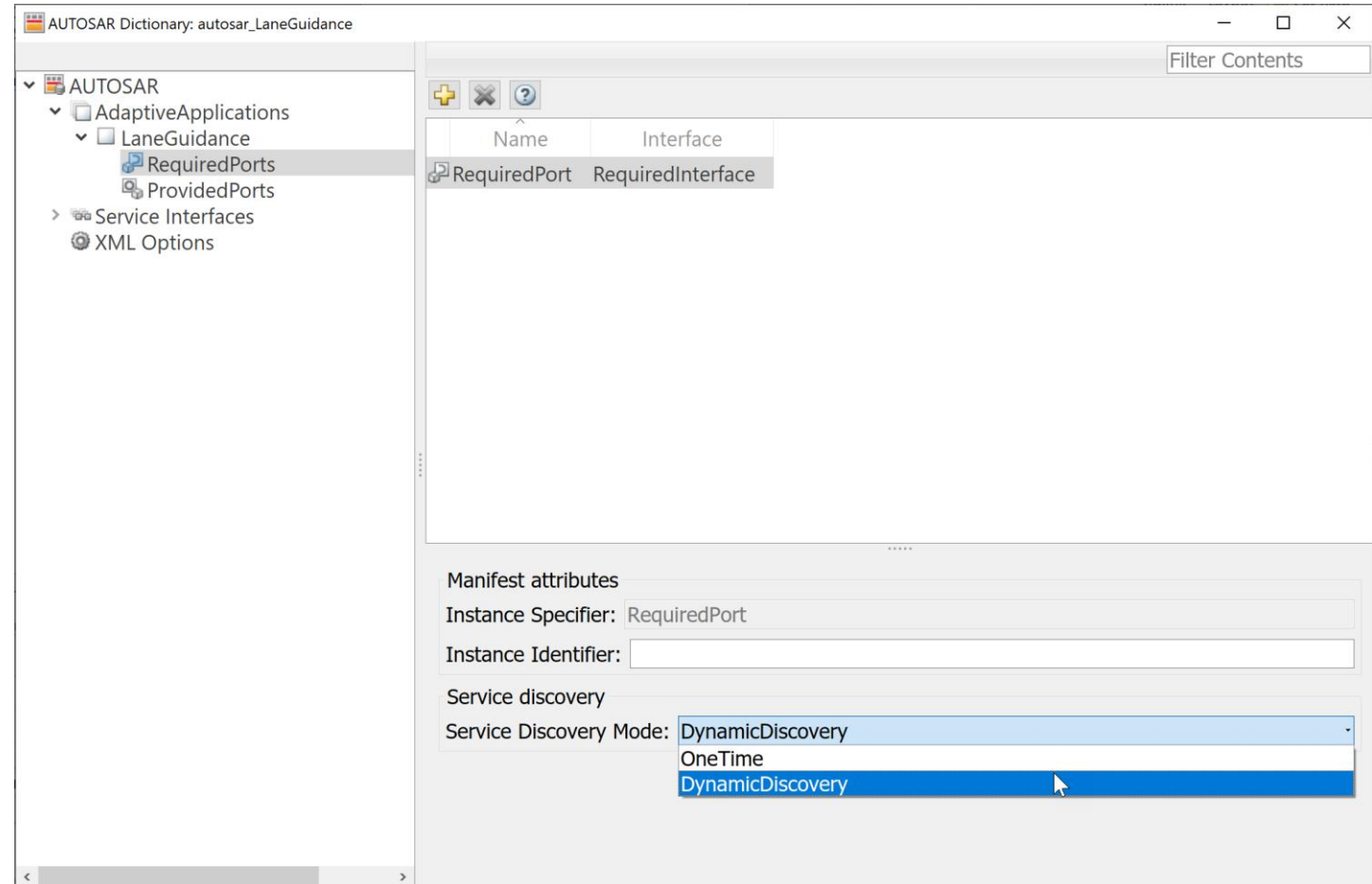
Transport layer: None

OK Cancel Help Apply



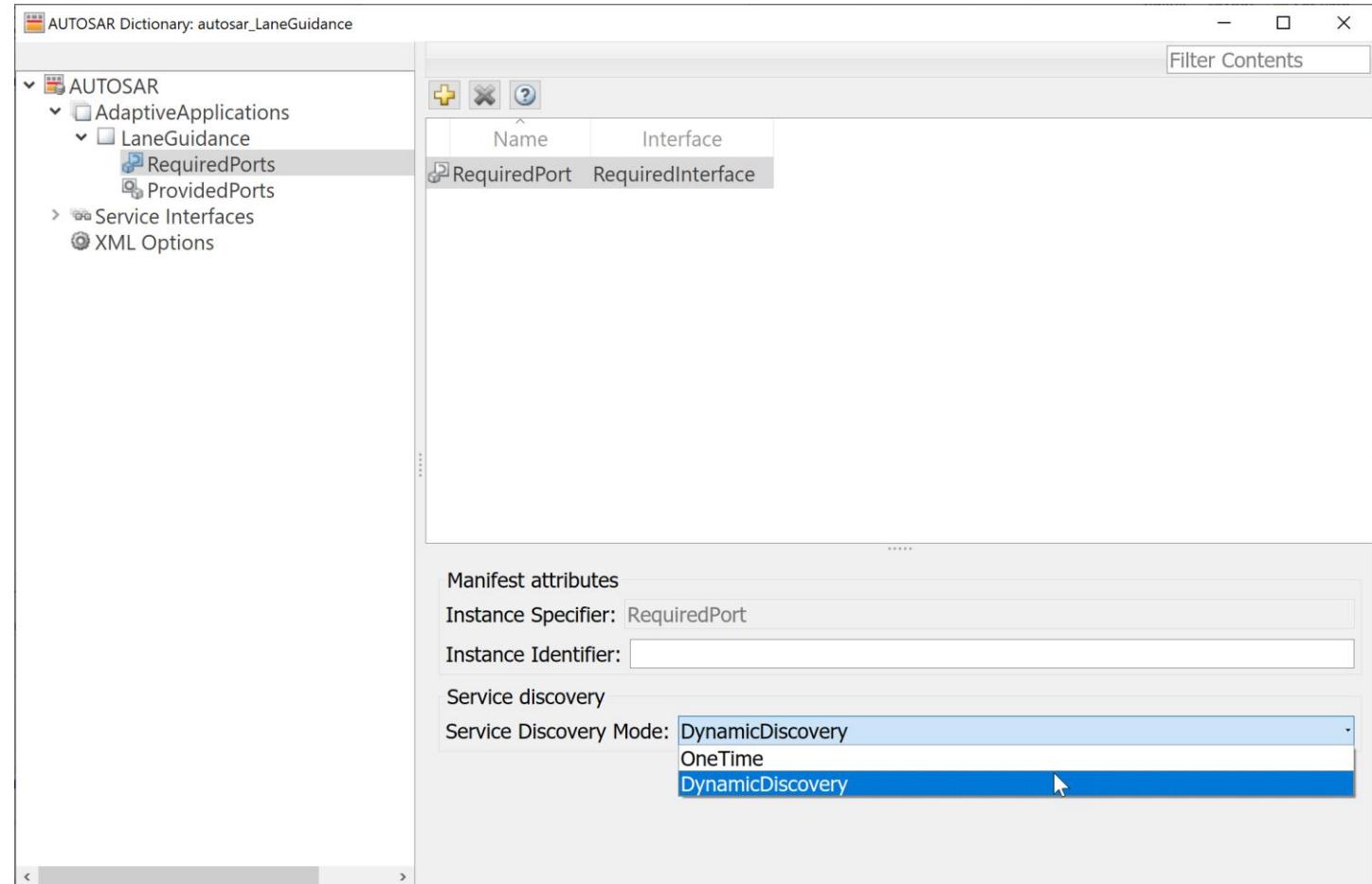
AUTOSAR Adaptive in action: top-down

- Create model from ARXML
- Configure Service Discovery



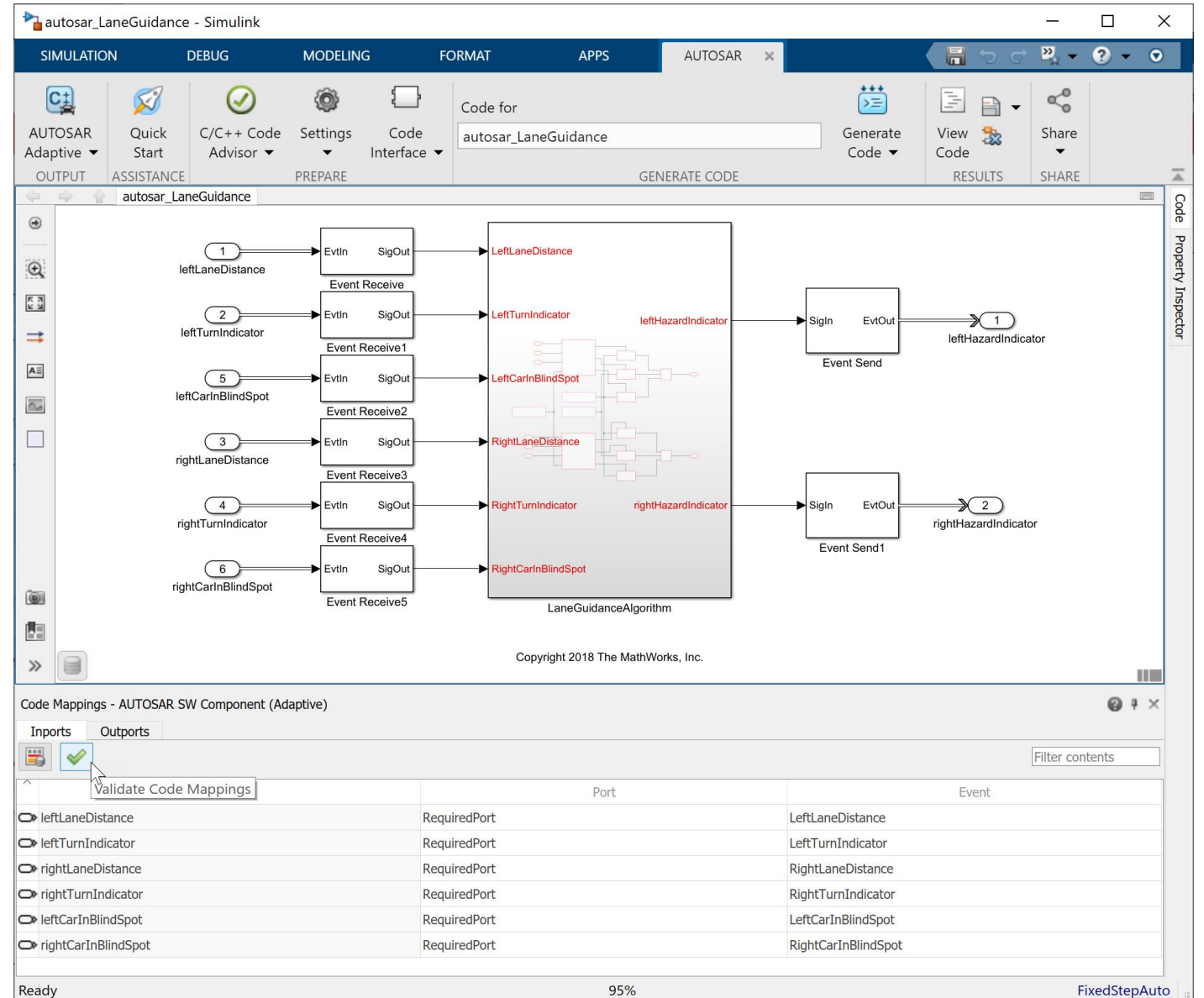
AUTOSAR Adaptive in action: top-down

- Create model from ARXML
- Configure Service Discovery
 - Subscribe to adaptive services
 - Only at startup, or
 - Dynamically, as they become available



AUTOSAR Adaptive in action: top-down

- Create model from ARXML
- Configure Service Discovery
- Verify AUTOSAR properties



AUTOSAR Adaptive in action: top-down

- Create model from ARXML
- Configure Service Discovery
- Verify AUTOSAR properties

The screenshot displays the MATLAB Simulink AUTOSAR Adaptive environment. The main workspace shows a block diagram for 'autosar_LaneGuidance'. On the left, there are six input ports: leftLaneDistance (1), leftTurnIndicator (2), leftCarInBlindSpot (5), rightLaneDistance (3), rightTurnIndicator (4), and rightCarInBlindSpot (6). Each port is connected to an 'EvtIn' block, which then feeds into a central 'LaneGuidanceAlgorithm' block. The algorithm outputs signals: LeftLaneDistance, LeftTurnIndicator, LeftCarInBlindSpot, and leftHazardIndicator. The leftHazardIndicator signal is processed by an 'Event Send' block, which outputs to a 'leftHazardIndicator' port (1).

An 'AUTOSAR Validation' window is overlaid on the diagram, displaying 'Validation succeeded' with a green progress bar. Below the diagram, the 'Code Mappings - AUTOSAR SW Component (Adaptive)' window is open, showing a table of port mappings.

Inports	Outports	Port	Event
leftLaneDistance	Validate Code Mappings	RequiredPort	LeftLaneDistance
leftTurnIndicator		RequiredPort	LeftTurnIndicator
rightLaneDistance		RequiredPort	RightLaneDistance
rightTurnIndicator		RequiredPort	RightTurnIndicator
leftCarInBlindSpot		RequiredPort	LeftCarInBlindSpot
rightCarInBlindSpot		RequiredPort	RightCarInBlindSpot

Copyright 2018 The MathWorks, Inc.

AUTOSAR Adaptive in action: top-down

- Create model from ARXML
- Configure Service Discovery
- Verify AUTOSAR properties
- Generate code

The screenshot displays the MATLAB AUTOSAR Adaptive tool interface. The top menu bar includes SIMULATION, DEBUG, MODELING, FORMAT, APPS, and AUTOSAR. The main workspace shows a model diagram for 'autosar_LaneGuidance' with various components and connections. Below the diagram is a 'Code Mappings' table for the AUTOSAR SW Component (Adaptive).

Source	Port	Event
RequiredPort_leftLaneDistance	RequiredPort	LeftLaneDistance
RequiredPort_leftTurnIndicator	RequiredPort	LeftTurnIndicator

The right pane shows the generated C++ code for 'autosar_LaneGuidance.cpp'. The code includes comments for service discovery and function definitions for message reception.

```

23 // '<S9>/IfActionS53'
24 // '<S9>/IfActionS54'
25 // '<S9>/IfActionS55'
26 //
27 void autosar_LaneGuidanceModelClass::autosar_LaneGuidance>IfActionS5(real_T
28   rtu_In1, real_T *rtu_Out1)
29 {
30   // Inport: '<S10>/In1'
31   *rtu_Out1 = rtu_In1;
32 }
33
34 void autosar_LaneGuidanceModelClass::RequiredPortLeftLaneDistanceReceive(ara::
35   com::SamplePtr< company::chassis::required::proxy::events::LeftLaneDistance::
36   SampleType const > samplePtr)
37 {
38   // Receive: '<S1>/Message Receive'
39   autosar_LaneGuidance_B.MessageReceive = *samplePtr;
40 }
41
42 void autosar_LaneGuidanceModelClass::RequiredPortLeftTurnIndicatorReceive(ara::
43   com::SamplePtr< company::chassis::required::proxy::events::LeftTurnIndicator::
44   SampleType const > samplePtr)
45 {
46   // Receive: '<S3>/Message Receive'
  
```


AUTOSAR Adaptive in action

- Create model from ARXML
- Configure Service Discovery
- Verify AUTOSAR properties
- Generate code
 - Integrate Applications with third-party Adaptive stack
 - Create Linux executables for calibration and monitoring

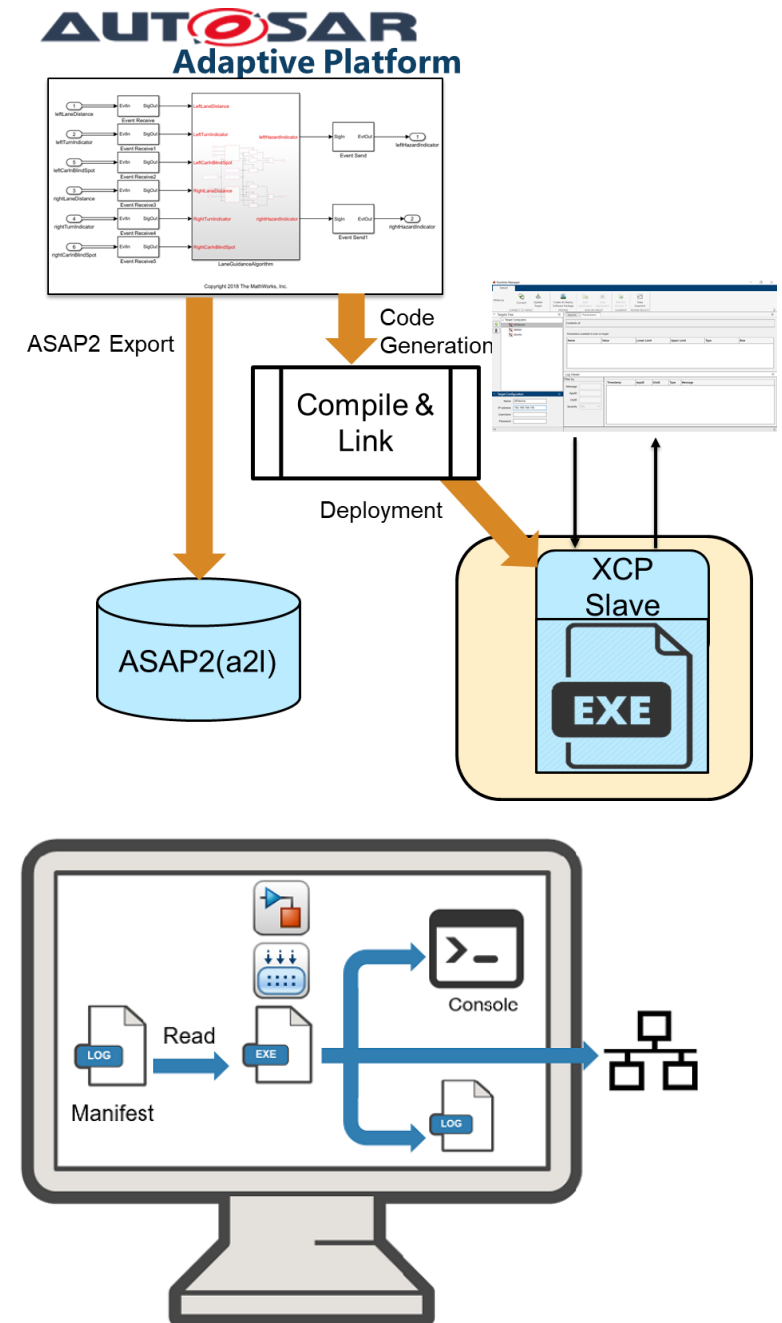
The screenshot displays the MATLAB AUTOSAR Adaptive tool interface. The top menu bar includes SIMULATION, DEBUG, MODELING, FORMAT, APPS, and AUTOSAR. The main workspace shows a block diagram of the 'autosar_LaneGuidance' model with various ports and internal components. Below the diagram is a 'Code Mappings' table for the AUTOSAR SW Component (Adaptive).

Source	Port	Event
RequiredPort_leftLaneDistance	RequiredPort	LeftLaneDistance
RequiredPort_leftTurnIndicator	RequiredPort	LeftTurnIndicator

The right side of the interface shows the 'Code' window for 'autosar_LaneGuidance.cpp'. A search bar is visible, and a dropdown menu lists files including 'autosar_LaneGuidance.cpp', 'autosar_LaneGuidance.h', and various interface and ARA files.

AUTOSAR Adaptive Deployment

- Create Linux executables for Run-Time Calibration and Measurement
- Run-time logging (ara::log) for adaptive executables
 - Forward event logging information to a console, file, or network, as defined in the AUTOSAR Diagnostic Log and Trace specification



Agenda

- Software-defined vehicles and new architectures (SOA)
- SOA Concepts
- **MathWorks Solutions for SOA**
 - Adaptive AUTOSAR
 - **DDS/ROS**
- Conclusions and key takeaways

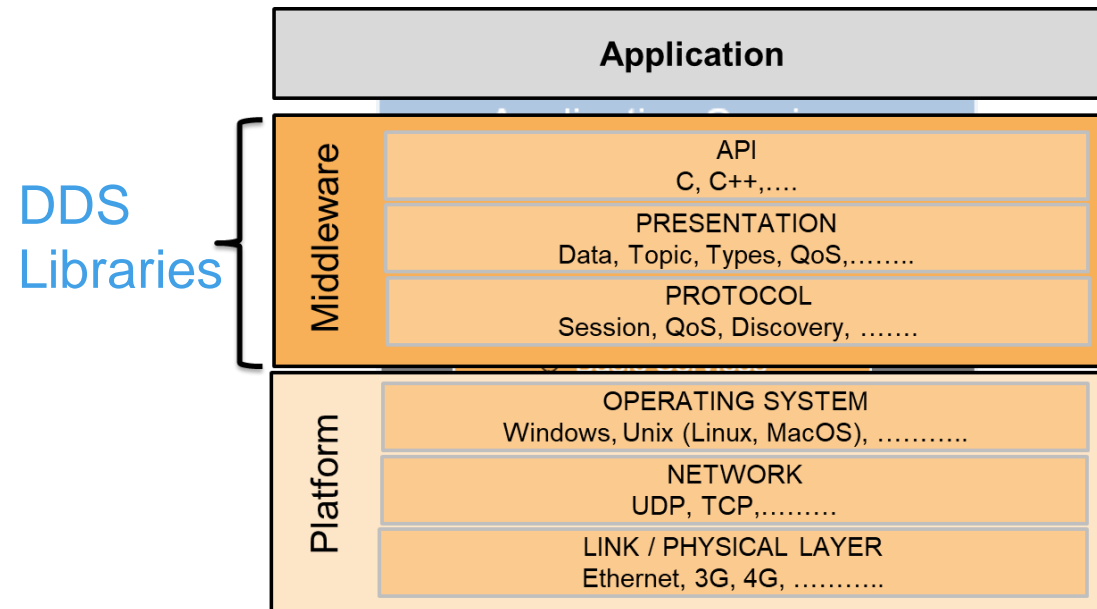
Simulink for DDS



Data Distribution Services (DDS) uses SOA methodology, and directly addresses publish and subscribe communications for real-time and embedded systems.

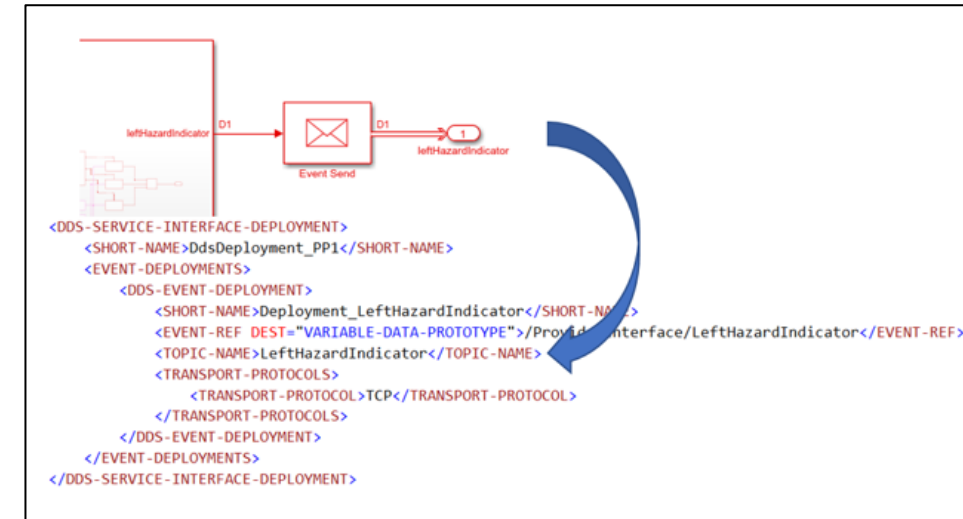
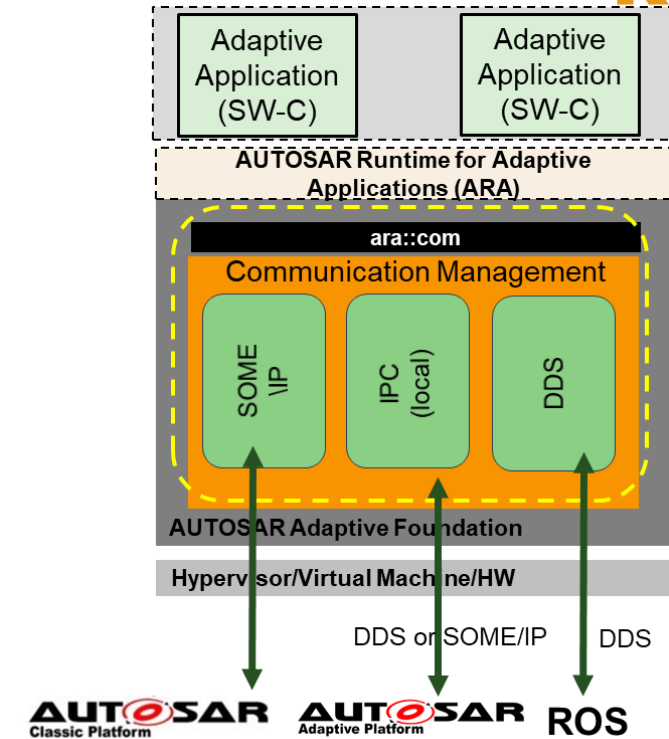


DDS addresses the needs of applications that require real-time data exchange in industries like aerospace and defense, automotive, and robotics.

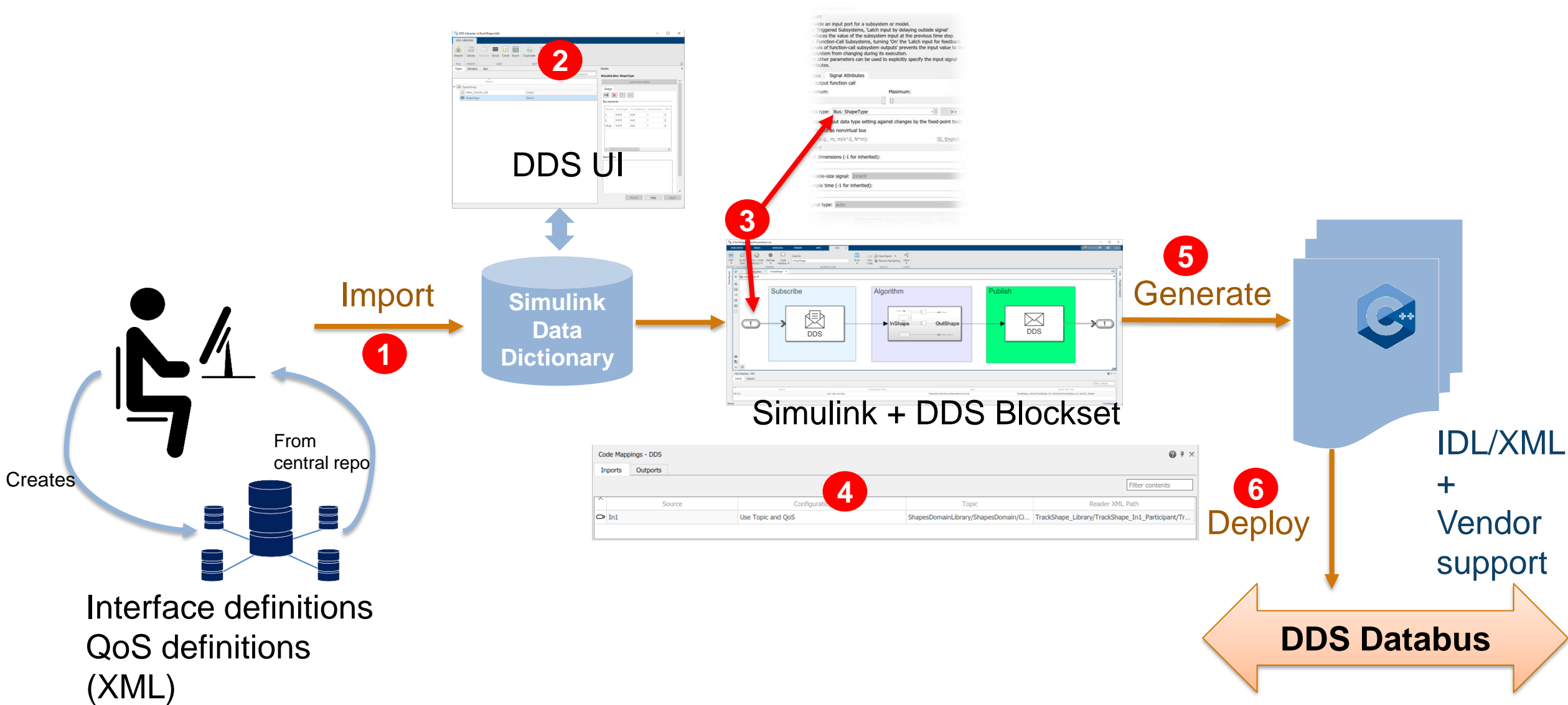


DDS (Data Distribution Services) is part of AUTOSAR Adaptive Deployment

- Supports DDS binding for ara::com enabling communication between adaptive AUTOSAR applications
 - Generated `ServiceInstanceManifest.arxml` contains DDS deployment artifacts

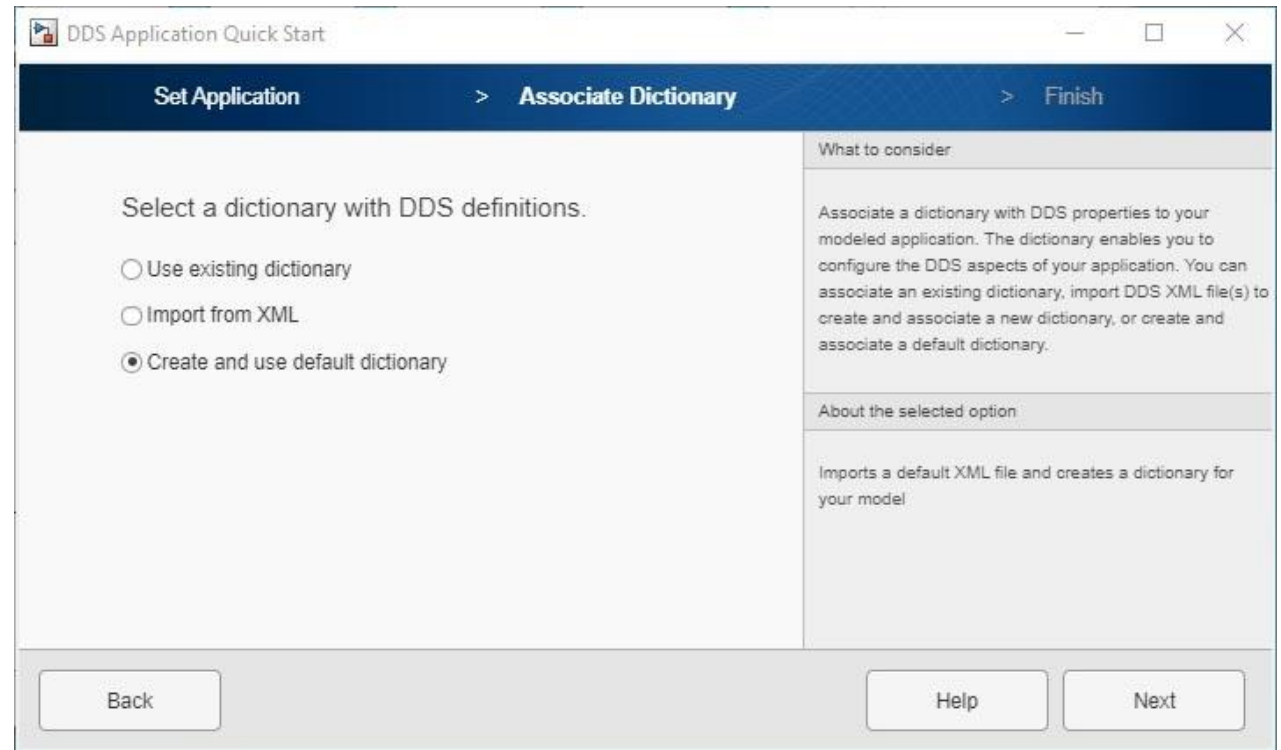


User Workflow with DDS Blockset



DDS Blockset in action

- Import DDS definitions from XML or create new Definitions



DDS Blockset in action

- Import DDS definitions from XML or create new Definitions
- Define/Modify DDS definitions in DDS Dictionary
 - Topic Types
 - Domains
 - QoS

The screenshot displays the 'DDS Libraries: ShapesApp.sidd' window. The main area shows a tree view of the 'ShapesDomainLibrary' containing a 'ShapesDomain' with ID 0. Below this, a table lists three topic types: Circle1, Square1, and Triangle1, all associated with the ShapeType1 type.

Name	Domain ID	Topic Type
ShapesDomainLibrary		
ShapesDomain	0	
Circle1		ShapeType1
Square1		ShapeType1
Triangle1		ShapeType1

The right-hand 'Details' panel shows the 'Domain: ShapesDomain' with ID 0. It includes a 'Registered Types' section with a table:

Name	TypeRef
ShapeType1	ShapeType1

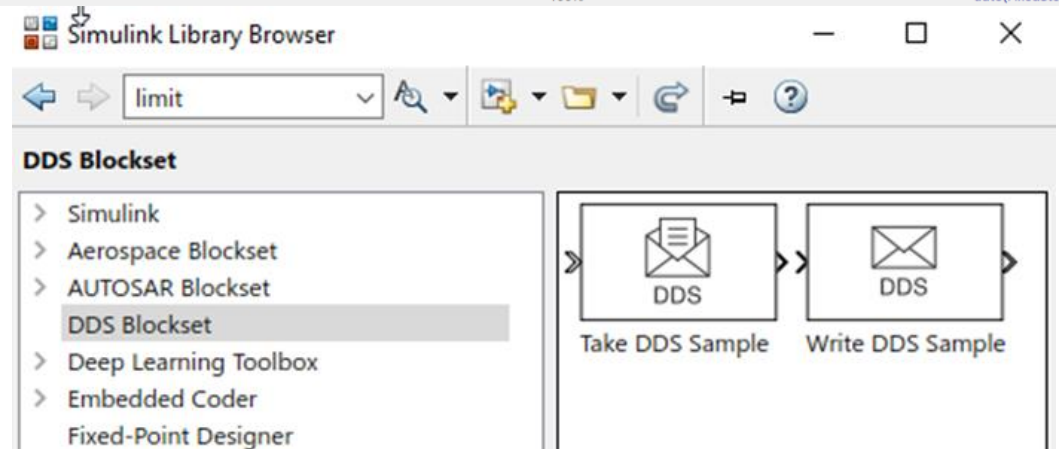
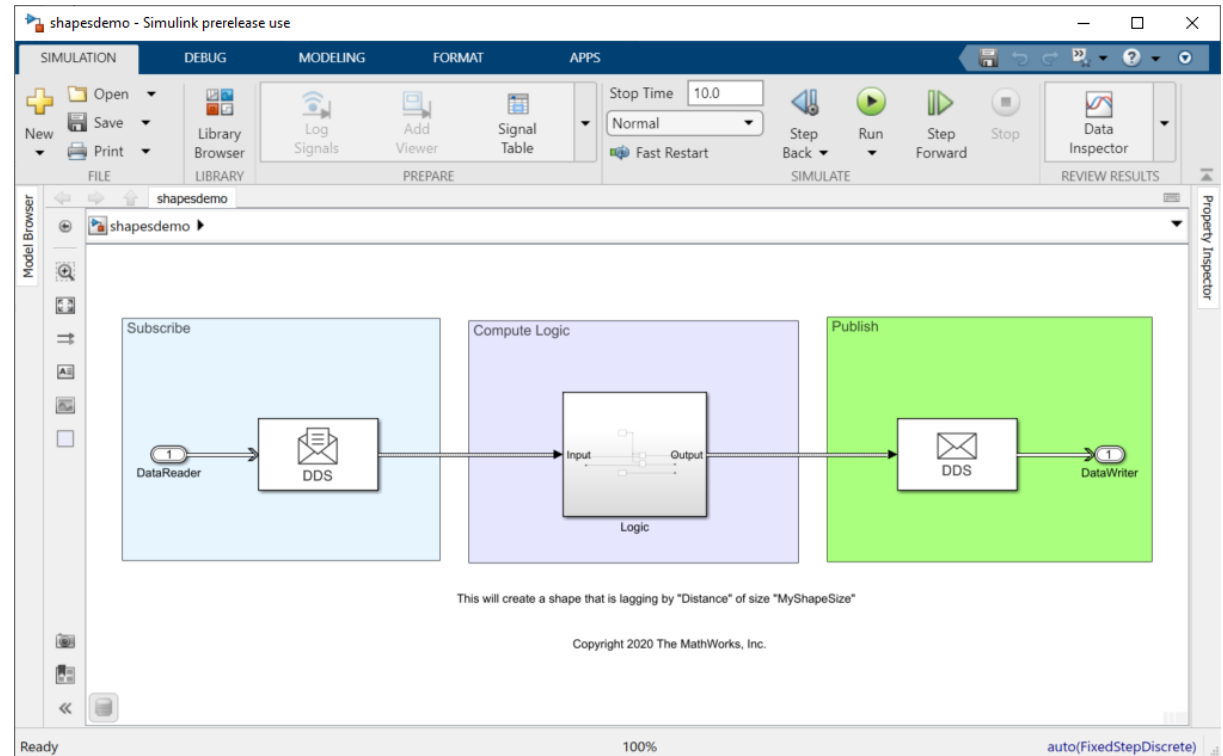
Below this is a 'Topics' section with another table:

Name	RegisterTypeRef
Circle1	ShapeType1
Square1	ShapeType1
Triangle1	ShapeType1

DDS Blockset in action

- Import DDS definitions from XML or create new Definitions
- Define/Modify DDS definitions in DDS Dictionary
- Model applications

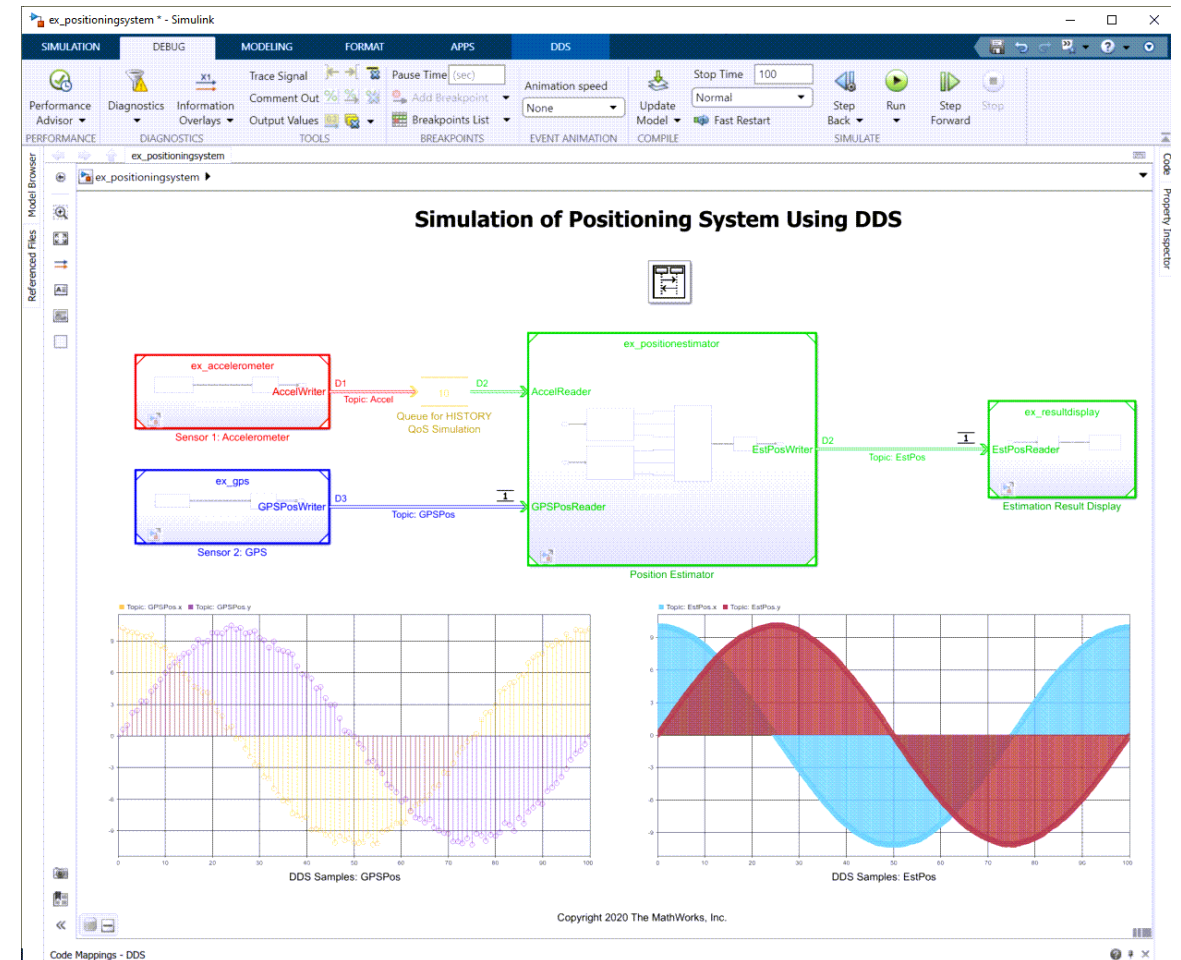
Use DDS Blocks to model a Publisher or Subscriber



DDS Blockset in action

- Import DDS definitions from XML or create new Definitions
- Define/Modify DDS definitions in DDS Dictionary
- Model applications
- Simulate DDS models including QoS

Use Simulink to model and simulation Quality of Services (QoS) policies including **history** to verify the runtime behavior.



DDS Blockset in action

- Import DDS definitions from XML or create new Definitions
- Define/Modify DDS definitions in DDS Dictionary
- Model applications
- Simulate DDS models including QoS
- Generate DDS executables and deploy on a DDS network

With Embedded coder, generate

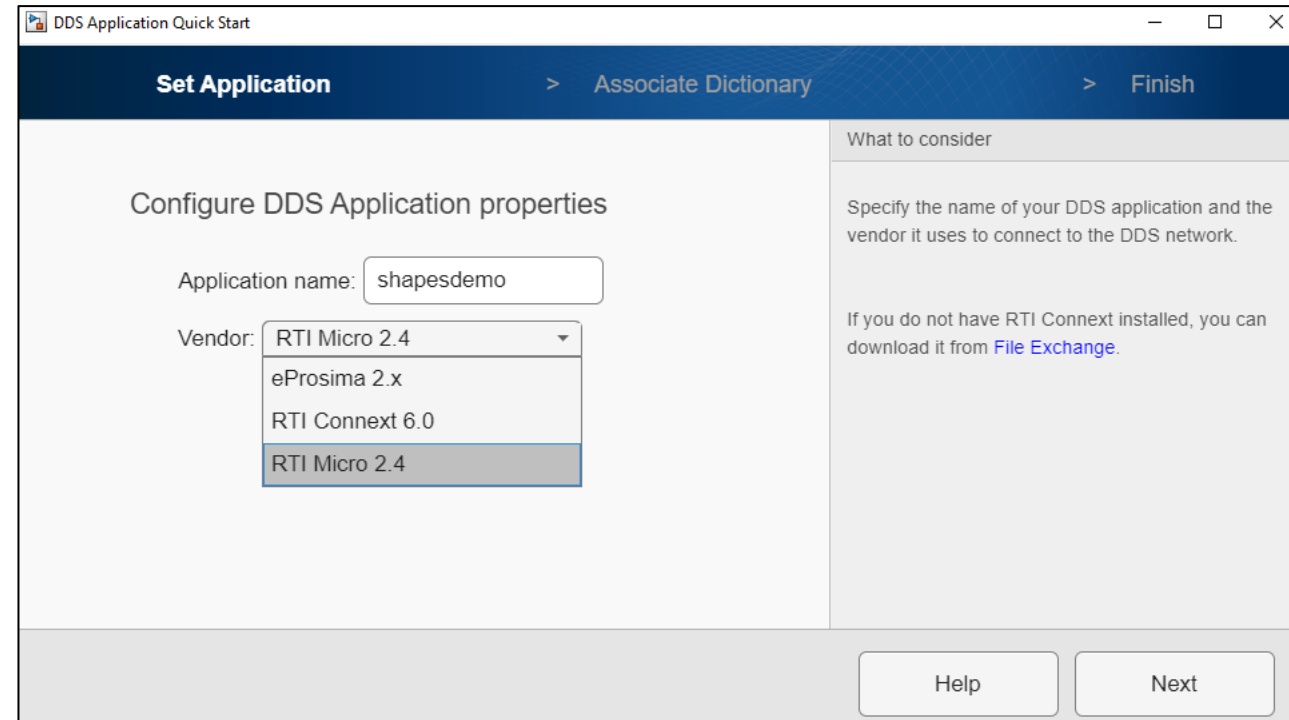
- C++ production code with DDS APIs
- XML or IDL files from Simulink models to deploy

```
bool writeWithWriter(const PosType* data, std::string participantName, std::string writerName) {
    DDS_DataWriter* writer = getWriter(writerName, participantName);
    PosTypeDataWriter* foowriter = PosTypeDataWriter_narrow(writer);
    if(!foowriter) {
        return false;
    }
    const DDS_ReturnCode_t ret = PosTypeDataWriter_write((PosTypeDataWriter*)writer, data);
    return (ret == DDS_ReturnCode_t::DDS_RETCODE_OK);
};

bool createParticipant(std::string participantName) {
    if (participants.find(participantName) == participants.end()) {
        DDS_DomainParticipant* participant =
            DDS_DomainParticipantFactory_create_participant_from_config(
                DDS_TheParticipantFactory, participantName.c_str());
        if(!participant) {
            return false;
        }
        participants[participantName] = participant;
    }
    return true;
};
```

Model DDS application

- Import DDS definitions from XML or create new Definitions
- Define/Modify DDS definitions in DDS Dictionary
- Model application algorithms
- Simulate DDS models including QoS
- Generate DDS executables and deploy on a DDS network



Full integration with third-party DDS stacks including RTI Connex, RTI Micro and eProsima Fast DDS

Agenda

- Software-defined vehicles and new architectures (SOA)
- SOA Concepts
- MathWorks Solutions for SOA
 - Adaptive AUTOSAR
 - DDS/ROS
- **Conclusions and Key Takeaways**

Conclusions

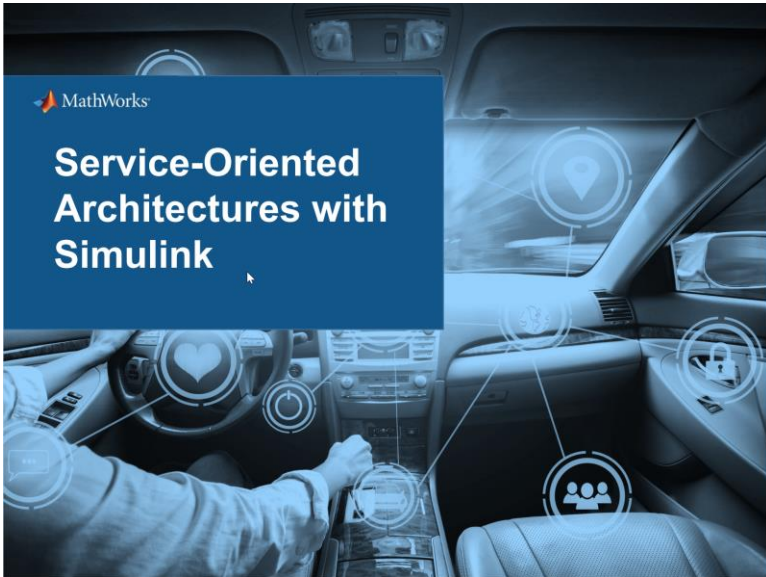
Challenges

- **Automotive E/E and SW architecture are evolving**, pushed by need for advanced, complex functions
- New, **service-oriented architectures** are required to **master complexity** and enable **frequent updates**

Solutions

- You can **design, simulate and generate** code to deploy service-oriented applications (including AUTOSAR Adaptive and DDS) in **Simulink**
- You can **reuse your existing expertise and models** to mitigate the risk of migration to SOA applications

To find more Info:



[Service-Oriented Architectures with Simulink Ebook \(mathworks.com\)](#)

[Designing and Deploying Service-Oriented Architectures \(SOA\) with Simulink Video - MATLAB & Simulink \(mathworks.com\)](#)

The screenshot shows the MathWorks website interface. At the top, there's a navigation bar with 'Products', 'Solutions', 'Academia', 'Support', 'Community', and 'Events'. Below that is a search bar. The main content area is titled 'Videos and Webinars' and features a video player. The video is titled 'AUTOSAR Adaptive workflows' and has a duration of 25:09. The video content shows a diagram of the AUTOSAR Adaptive workflow, which is a circular process involving 'Export ARXML' and 'Import ARXML' steps, leading to 'Application SW C++ Code'. The diagram is labeled 'Top-Down' and 'Bottom-Up'. Below the video player, there are tabs for 'Description' and 'Related Resources'. The video description starts with 'In recent years, the automotive industry is accelerating its investments in electrification,'.

The screenshot shows the 'What Is SOA?' page on the MathWorks website. The page has a blue header with the MathWorks logo and navigation links. The main content area is titled 'Model service-oriented architectures (SOA) in Simulink'. It defines SOA as a software architecture based on the concept that a system consists of a set of services in which one service may use another, and applications use one or more of the services based on their need. SOA promotes a loosely coupled component-based approach using middleware for service-oriented communication. It lists industry standards where SOA is used: AUTOSAR, ROS, and DDS. At the bottom, it states 'You can use Simulink to model and simulate software based on SOA that runs in different applications.' Below this text is a screenshot of a Simulink model showing a block diagram with 'Service Component' and 'Network Component' blocks.

[SOA - MATLAB & Simulink \(mathworks.com\)](#)

Reach out to us
 Rajat Arora rarora@mathworks.com
 Nukul Sehgal nsehgal@mathworks.com

MATLAB EXPO

Thank you



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