MATLAB EXPO

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

Rajat Arora, MathWorks

Nukul Sehgal, MathWorks

MathWorks®

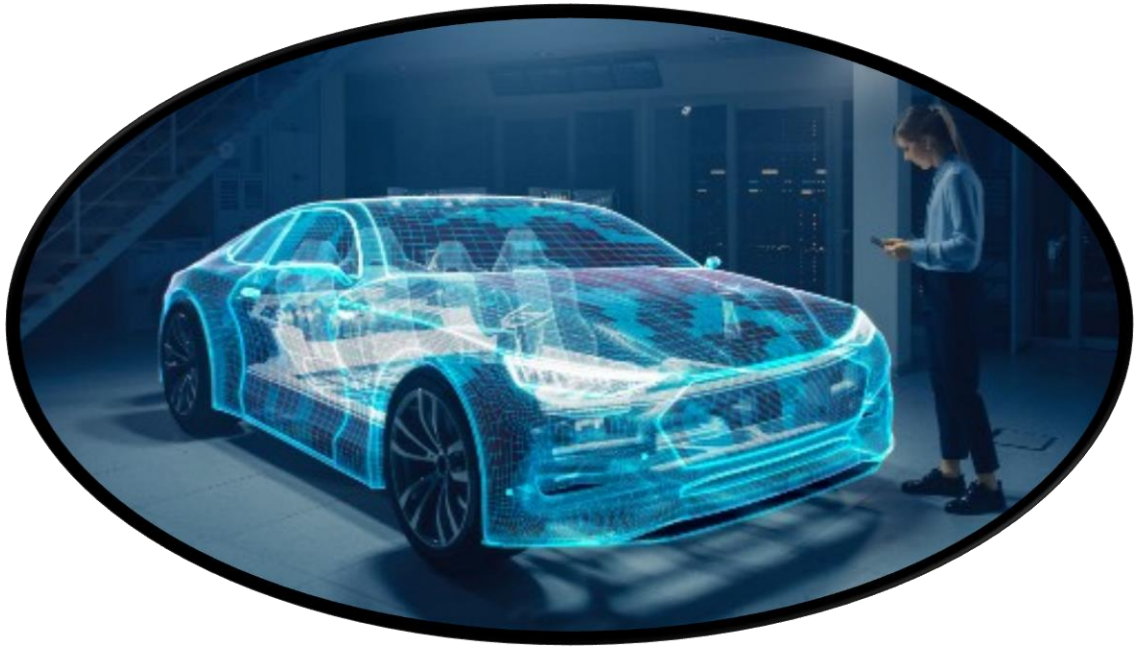
Agenda

- 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

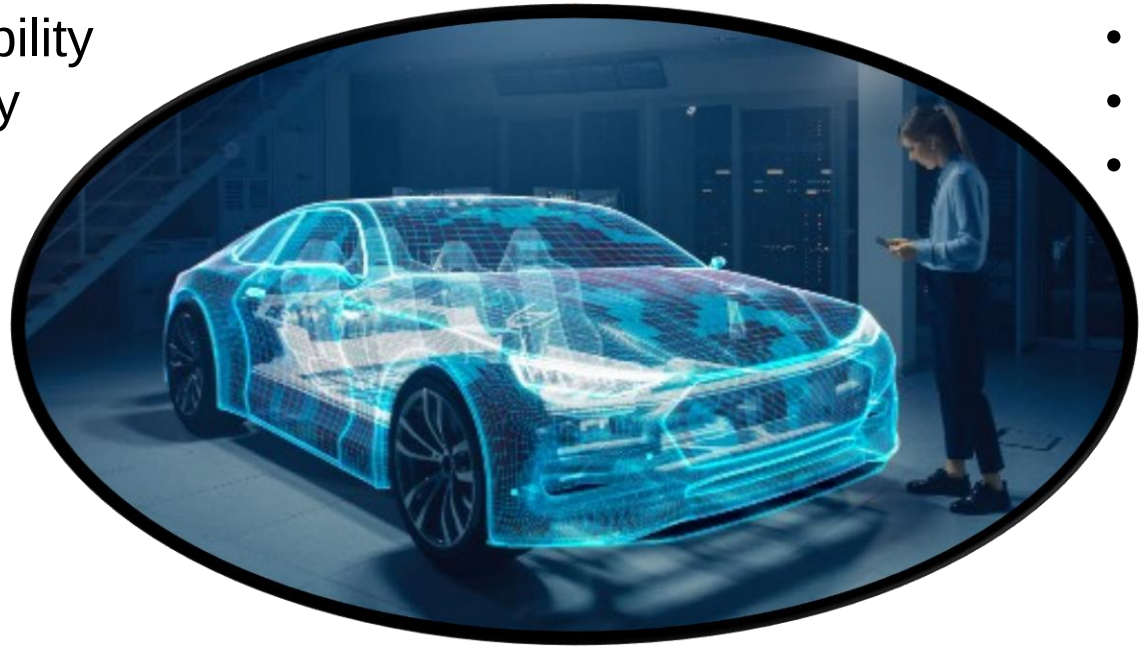
demand

Customer expectations

- Clean and Safe mobility
- Digital Life continuity

Technology & Innovation

- Electrification
- Autonomy
- Connectivity



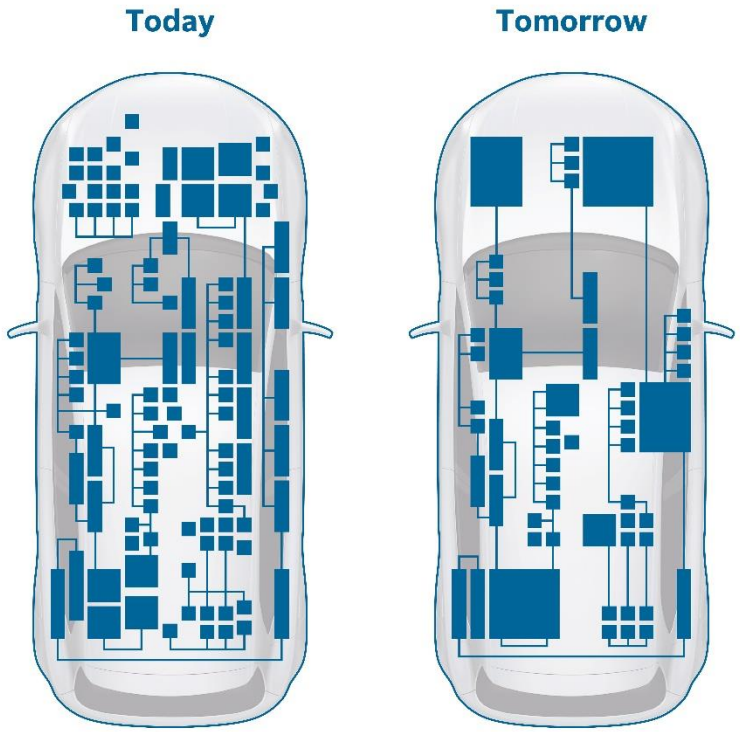
monetize

invest

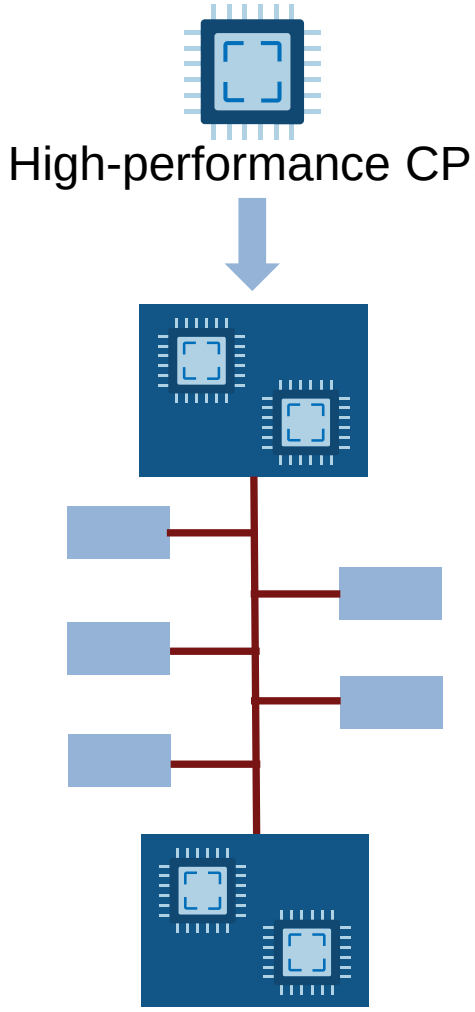
Business opportunity

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

Centralization of computing and SOA



Consolidation and centralization of computing

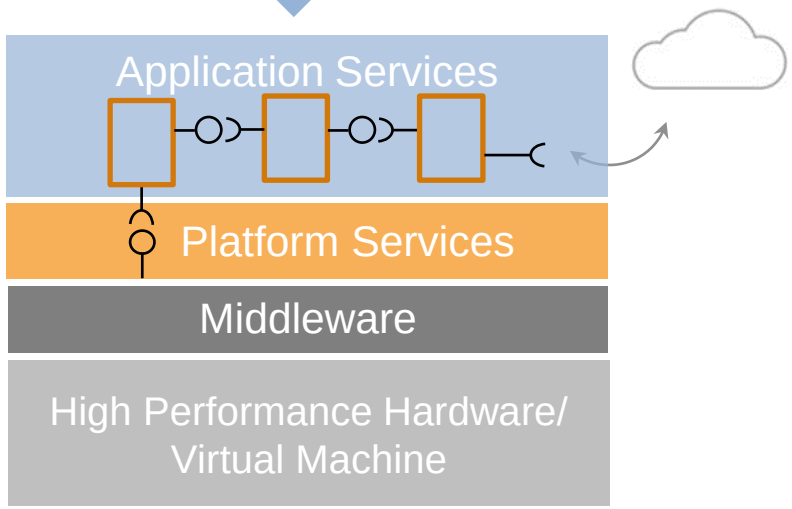


High-performance CPU/GPU

New E/E zonal architectures

SW updates

- Frequent
- Selective
- Over-the-air



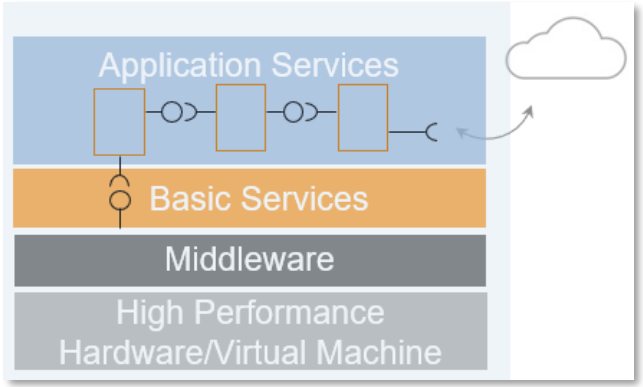
Higher HW abstraction:
Service-oriented architectures

Agenda

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

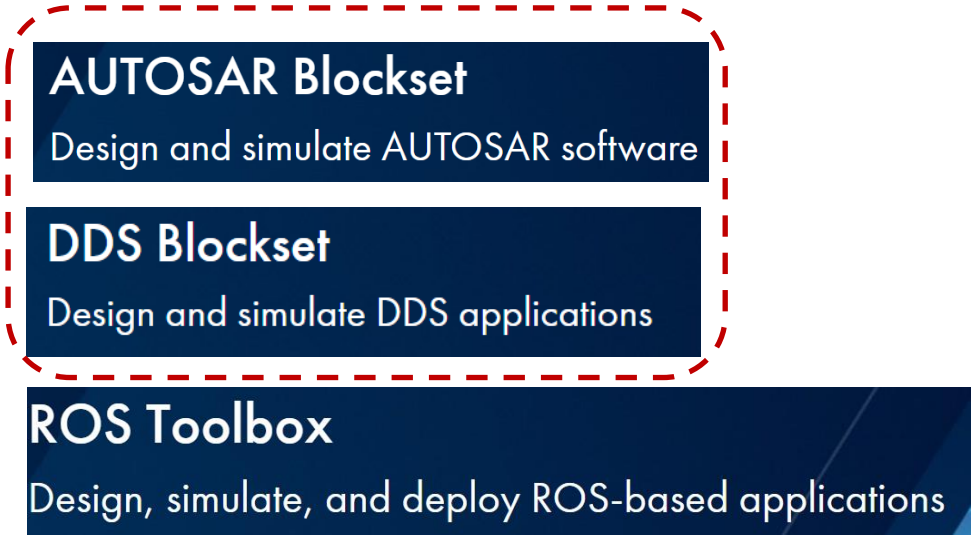
SOA – What's it all about?

- With SOA, applications are standalone processes that provide and/or require services distributed across the vehicle computing platform and the cloud

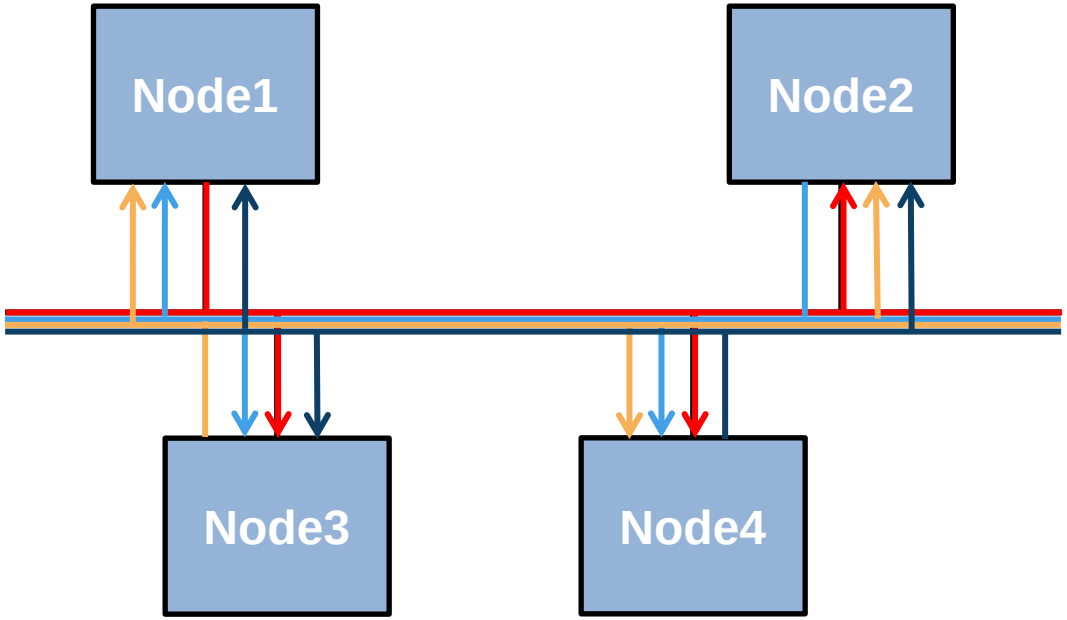


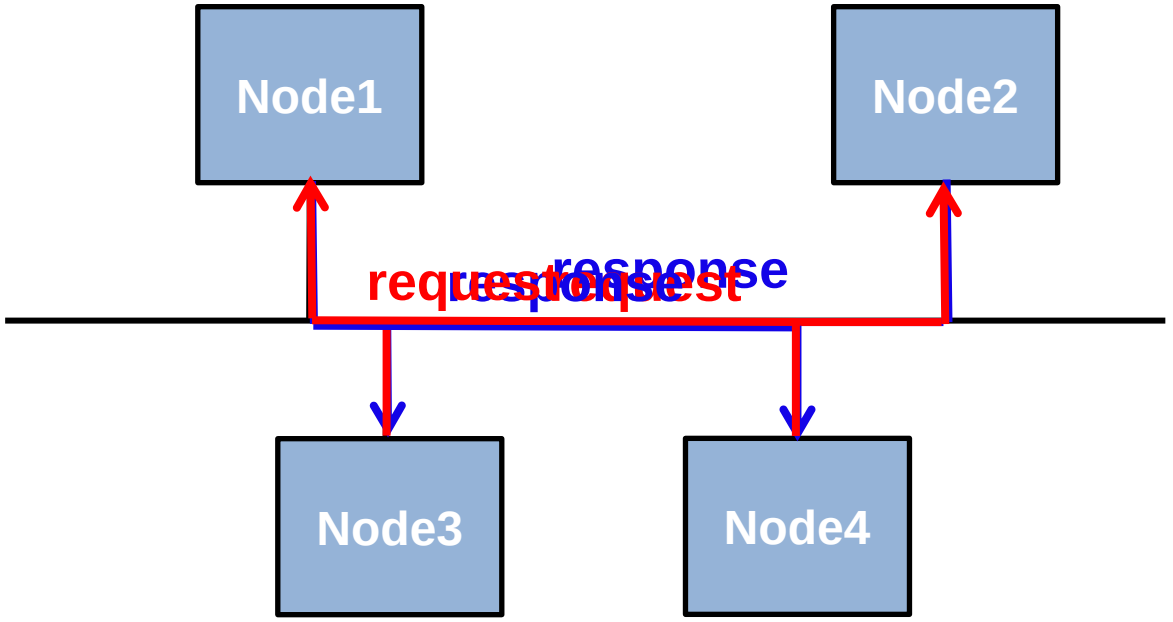
- SOA provides flexibility to add, remove, or update applications without impacting the entire, typically large, software system

- SOA is used by multiple industrial standards:
  - AUTOSAR Adaptive Platform
  - DDS (Data Distribution Services)
  - ROS (Robot Operating System)



SOC (Service Oriented Communication)





signal-oriented communication

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

service-oriented communication

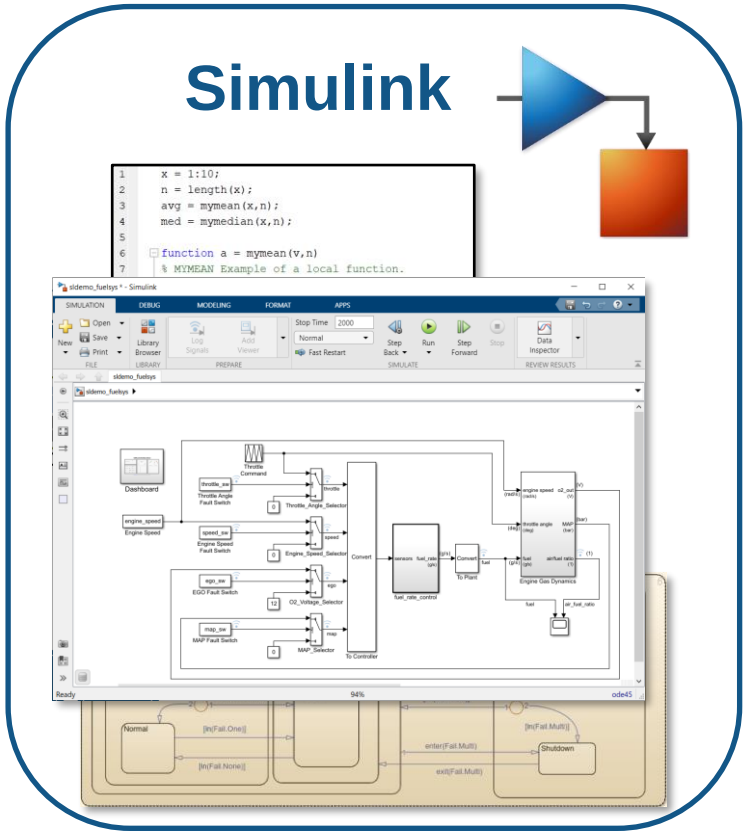
- send data dependent of needs
- low bus load
- more efficient

Key Challenges

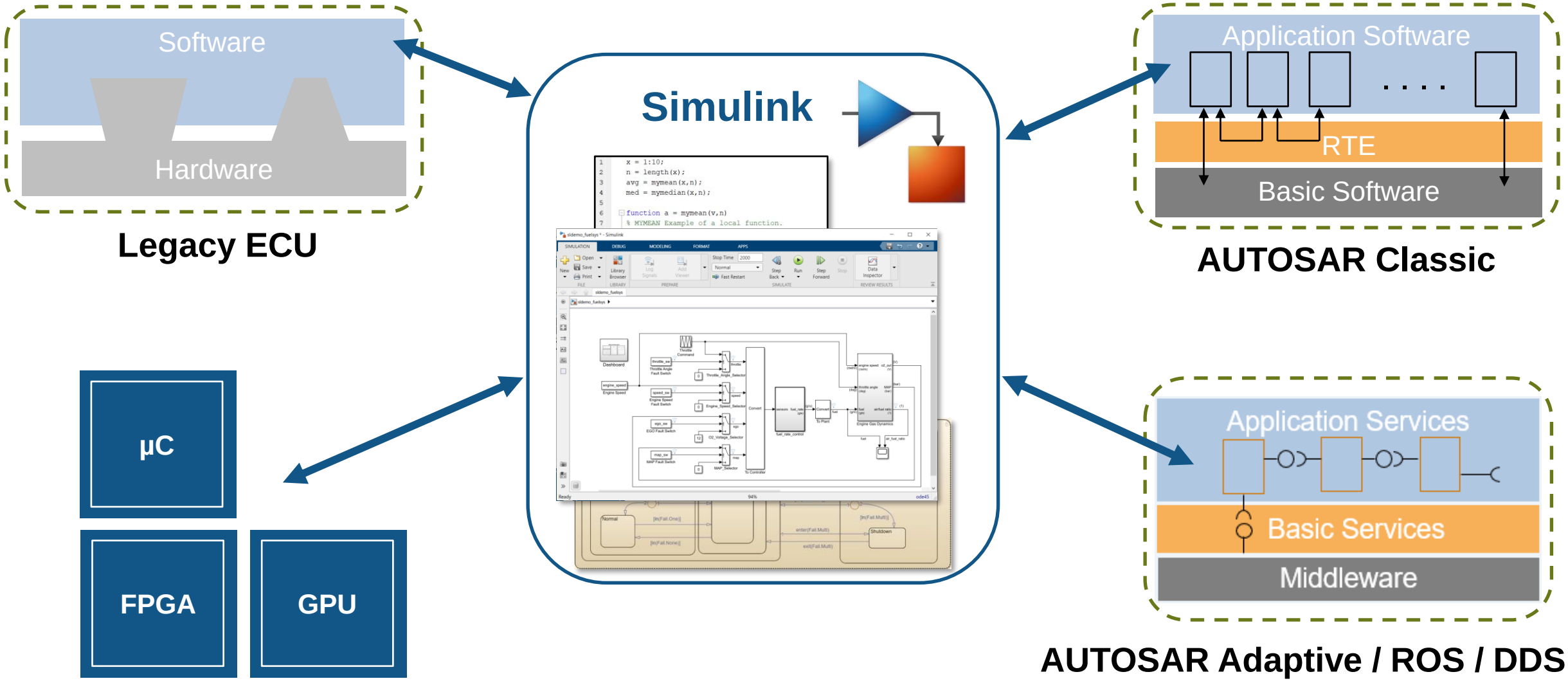
- Service-oriented architectures require a **change of mindset**
  - Shift from time-driven to event-driven execution
- **Centralize, re-architect** existing applications and partition in processes and services
  - e.g. Centralize energy management and path planning
- **Reuse of existing expertise**, 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

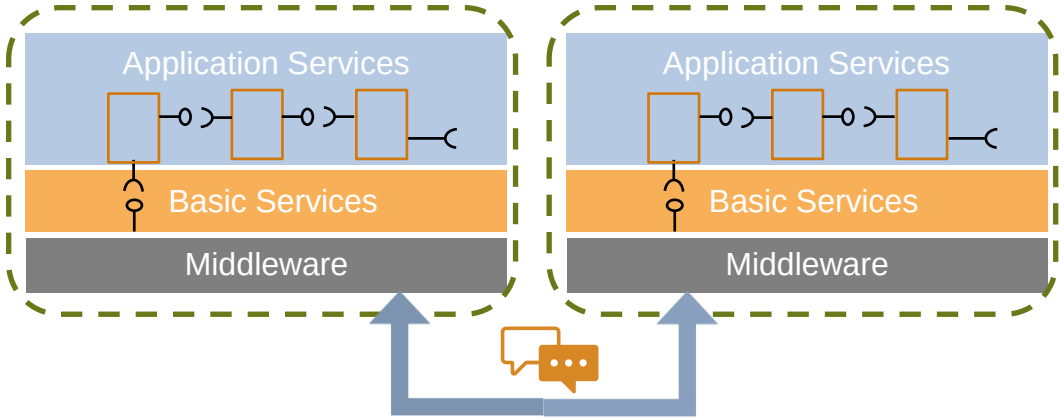
Simulink: Deploy software to different targets and standards



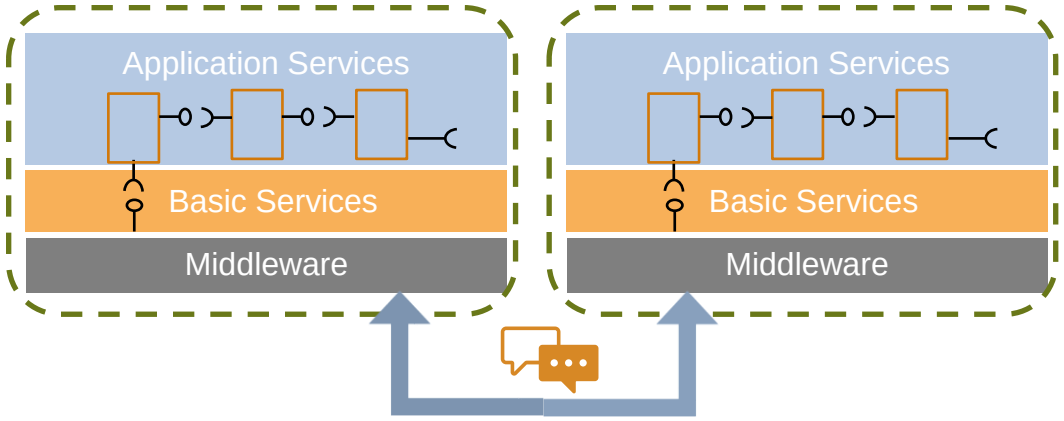
Simulink: Deploy software to different targets and standards

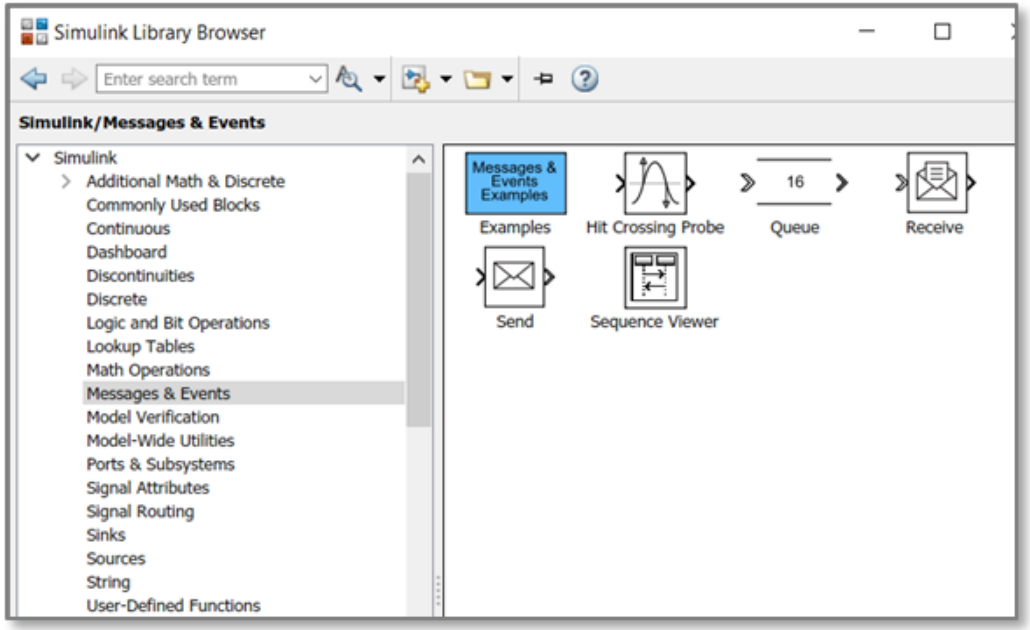


Simulink Messages for Service-oriented communication



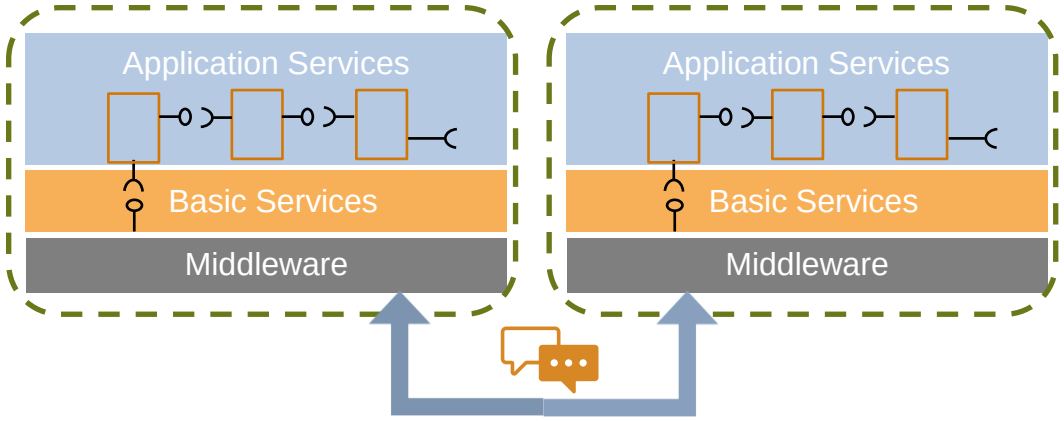
Simulink Messages for Service-oriented communication

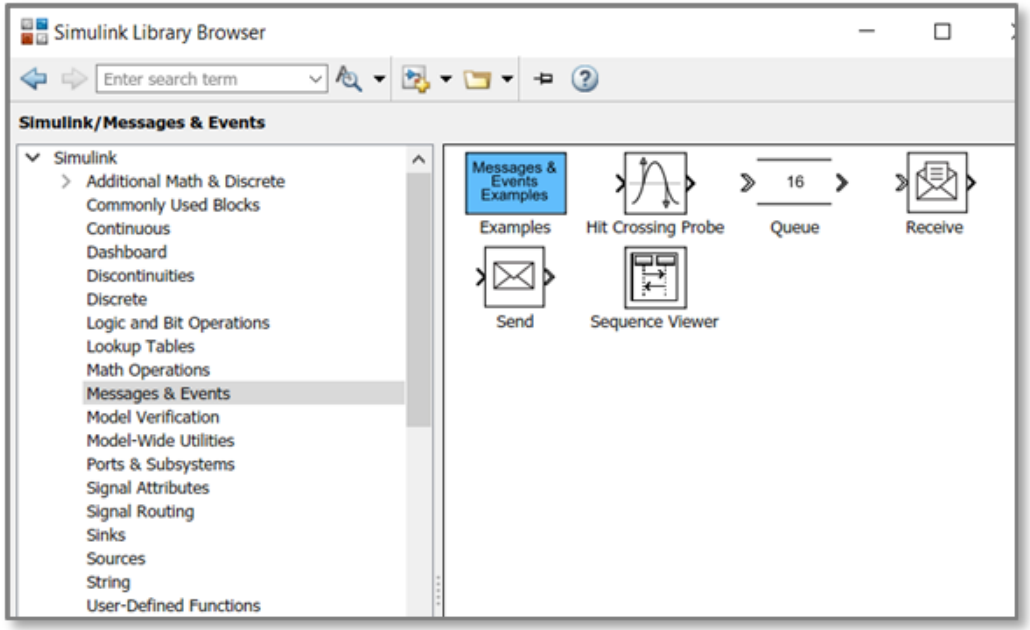


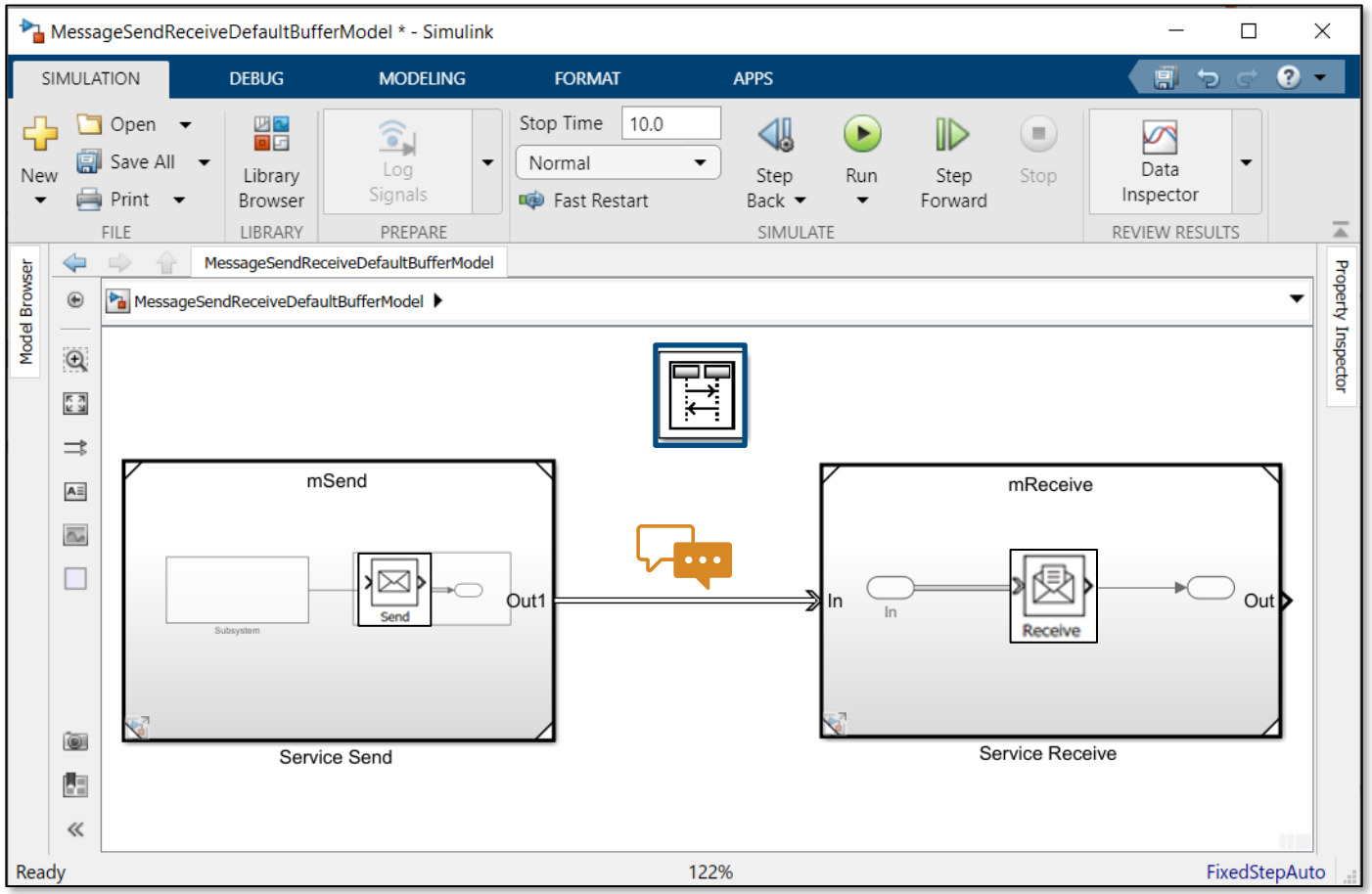


You can model service-oriented communication using messages (Send/Receive).

Simulink Messages for Service-oriented communication

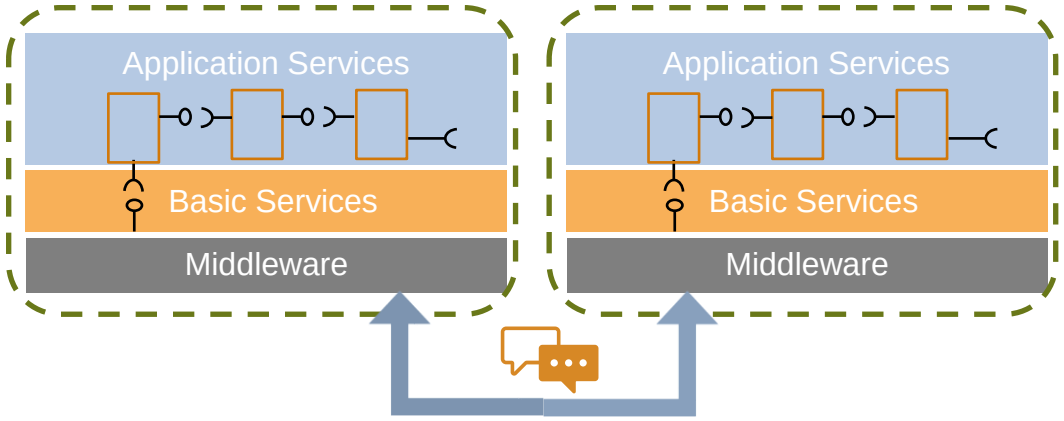


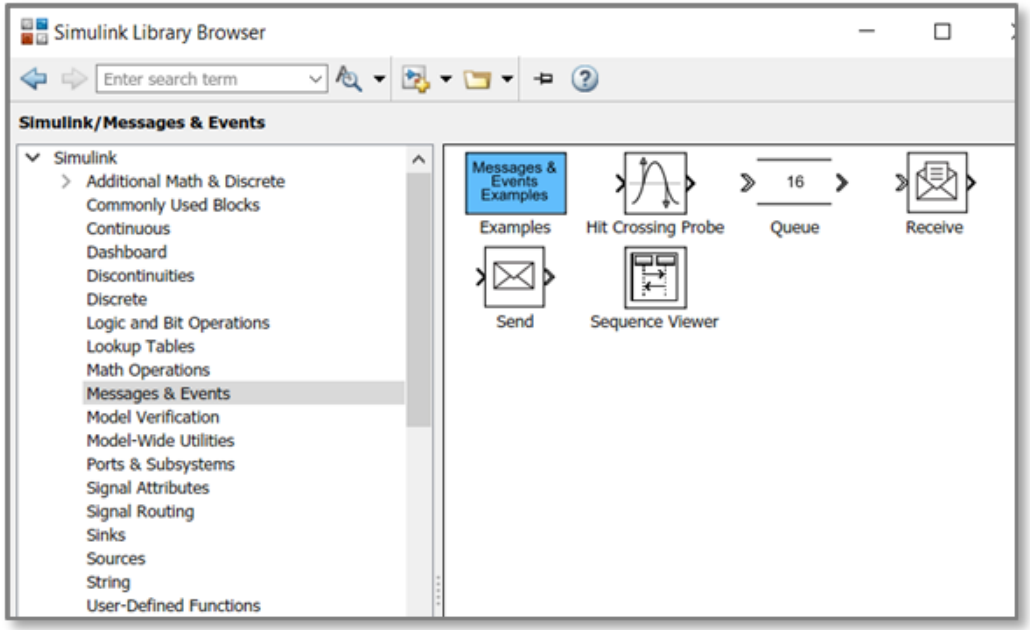


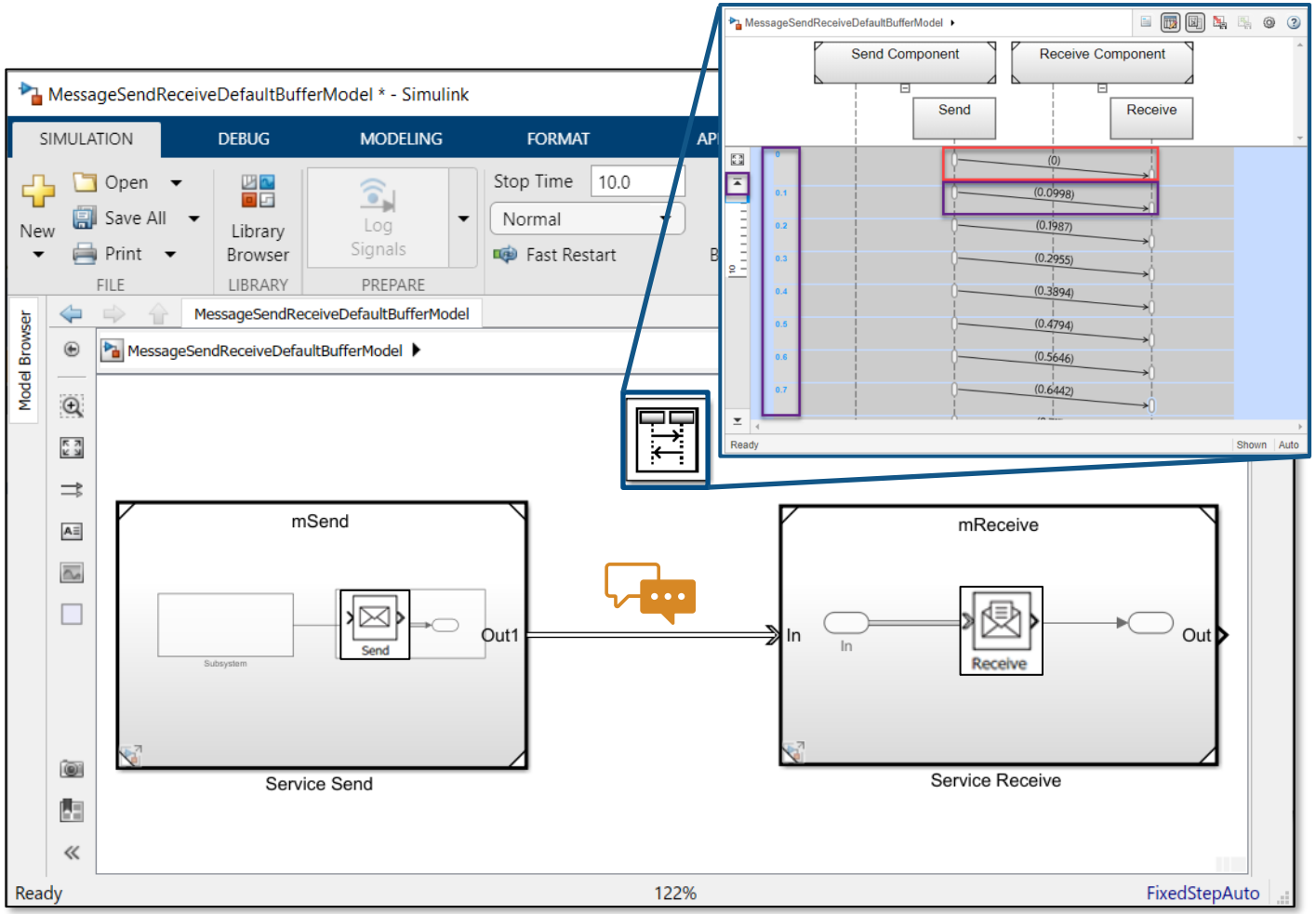


You can model service-oriented communication using messages (Send/Receive).

Simulink Messages for Service-oriented communication

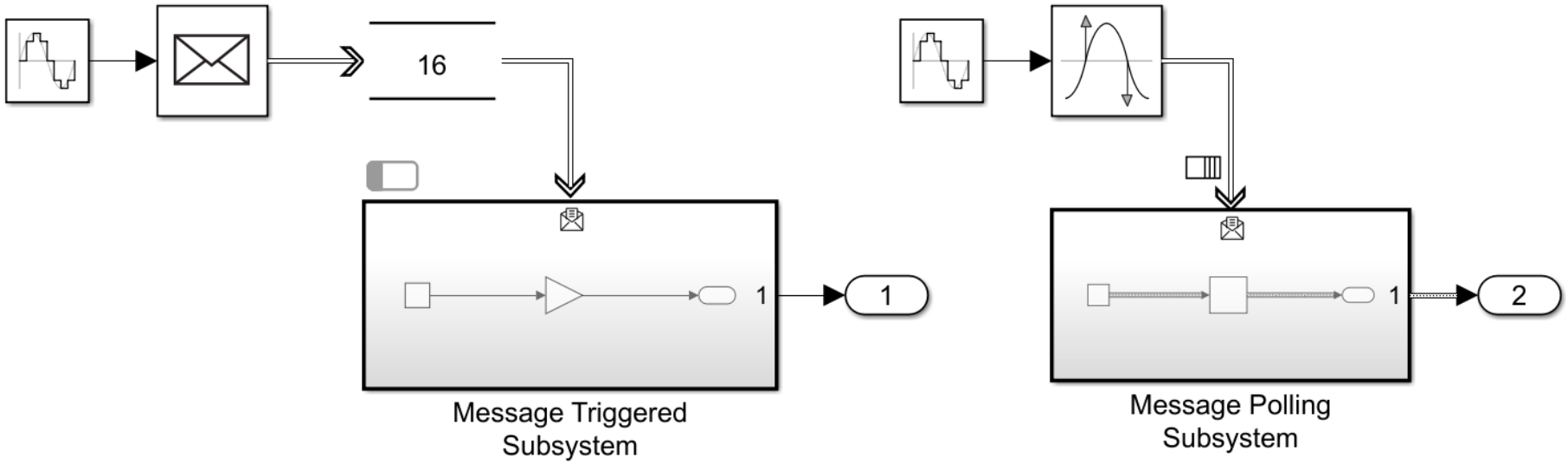






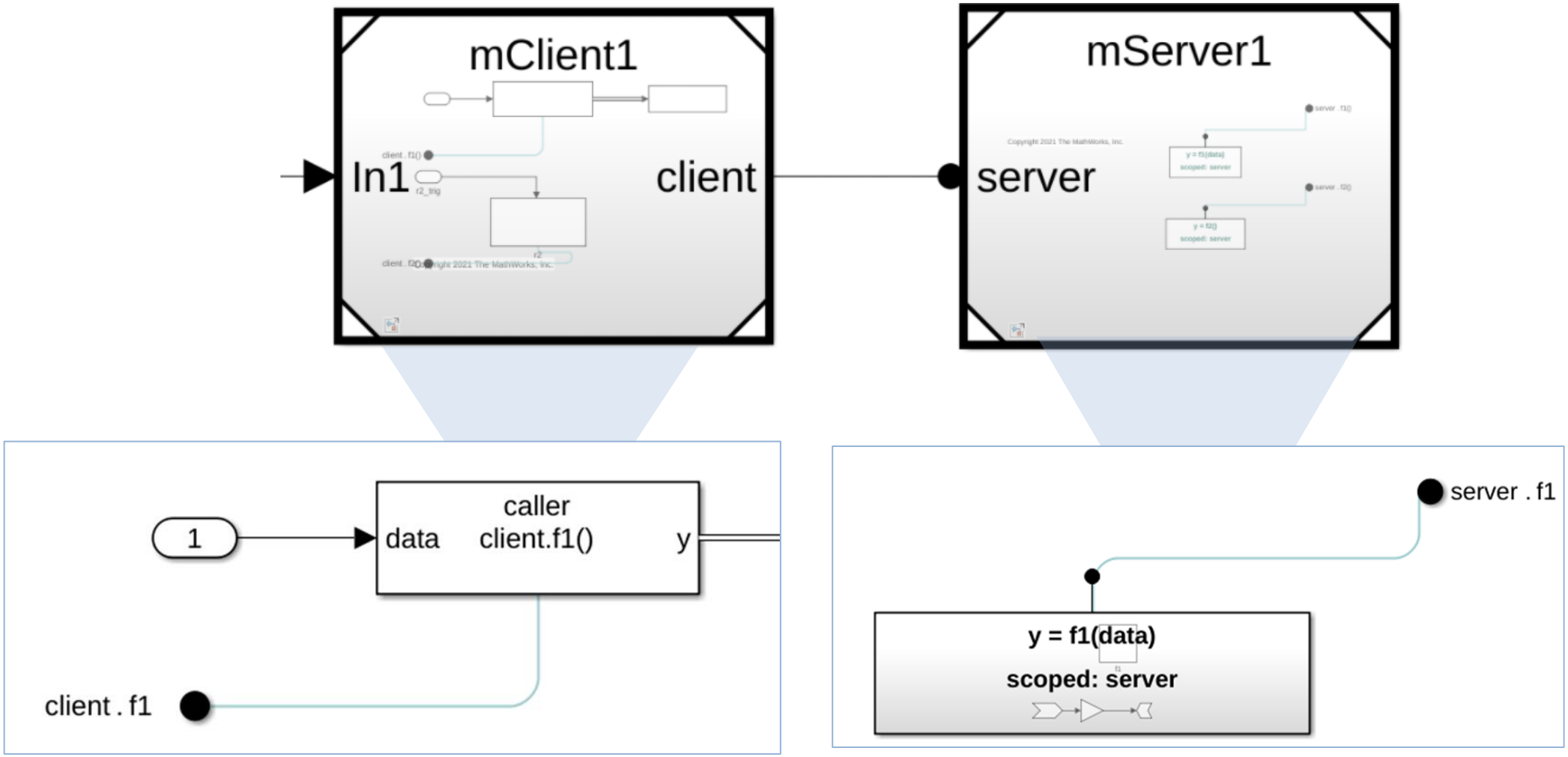
You can model service-oriented communication using messages (Send/Receive).

Message Triggered/Polling Subsystem for SOA



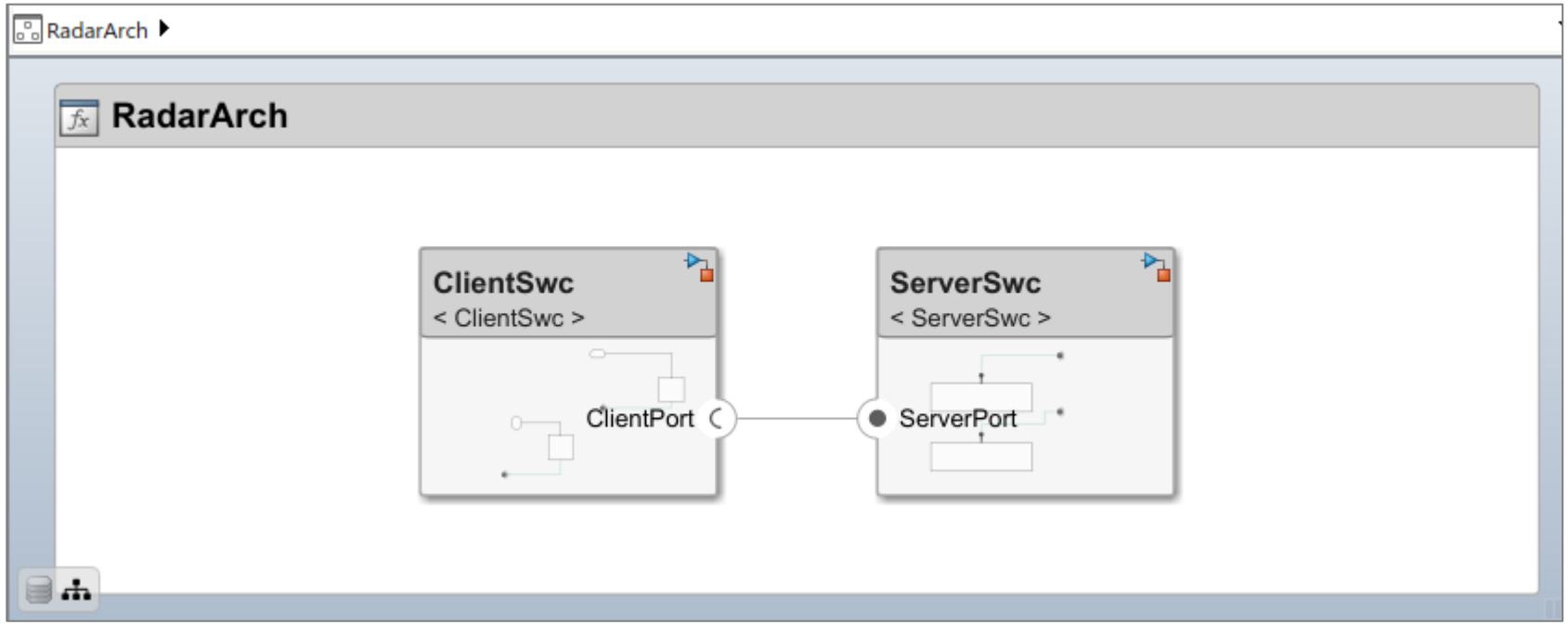
- New blocks to process messages by executing subsystem when message is available
- Model and generate code for components that are executed on message arrival

Function Ports for SOA



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

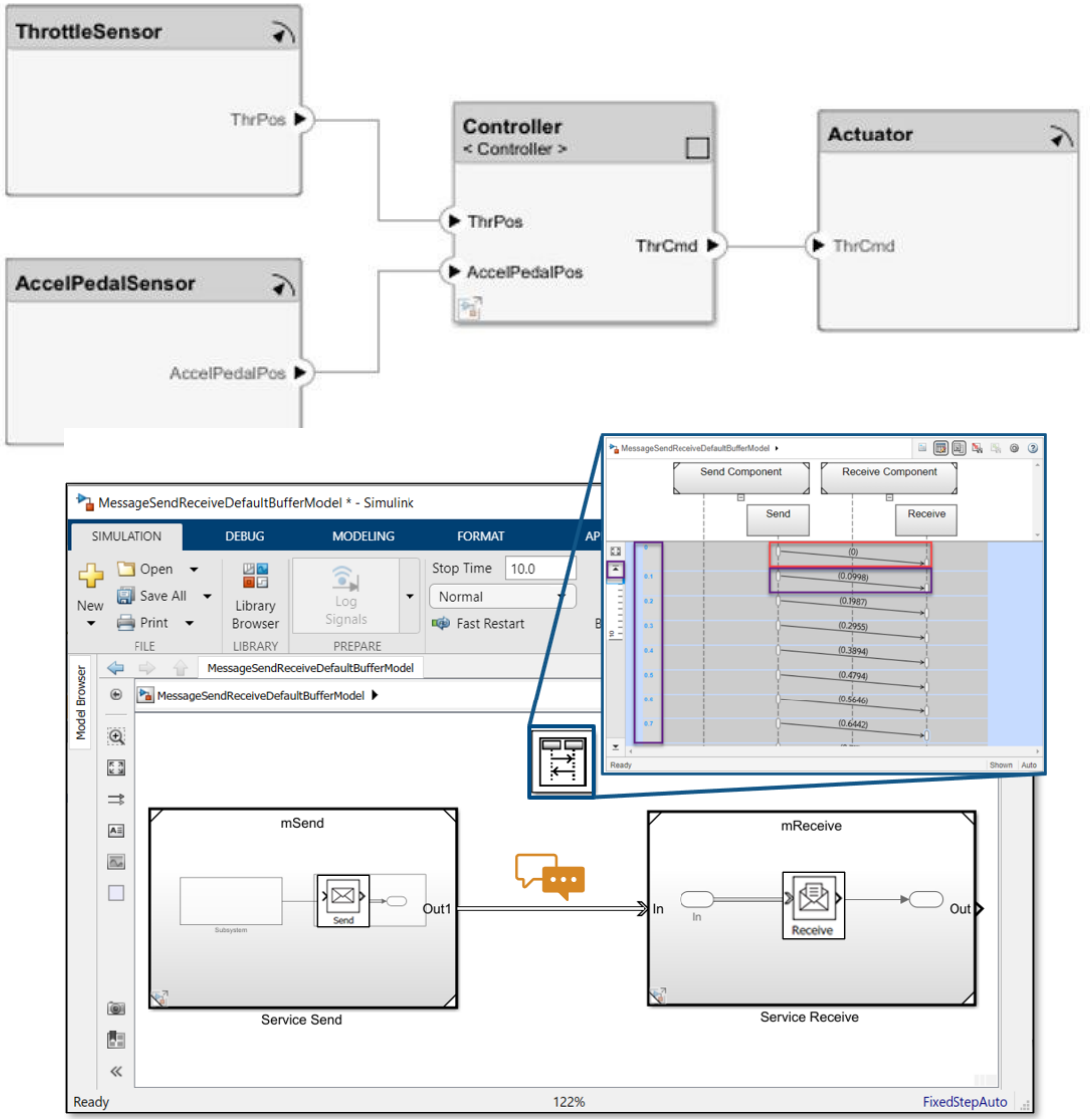
Author SOA applications in software architecture models



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

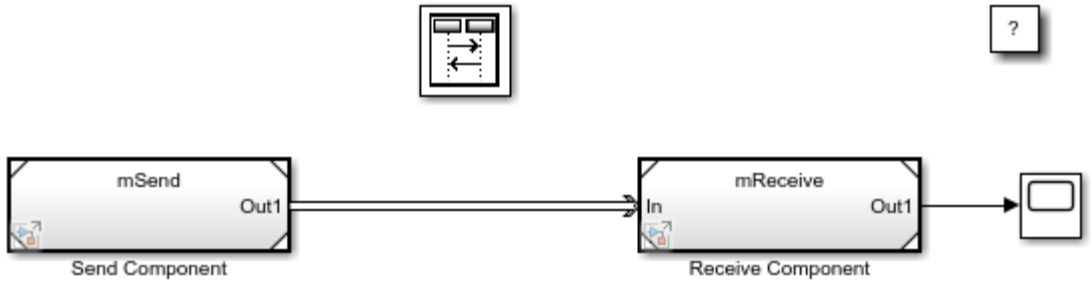
MathWorks investments in **SW architecture design** and **simulation**

- 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.




```
// 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);
};
```


```
// 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.
}
```

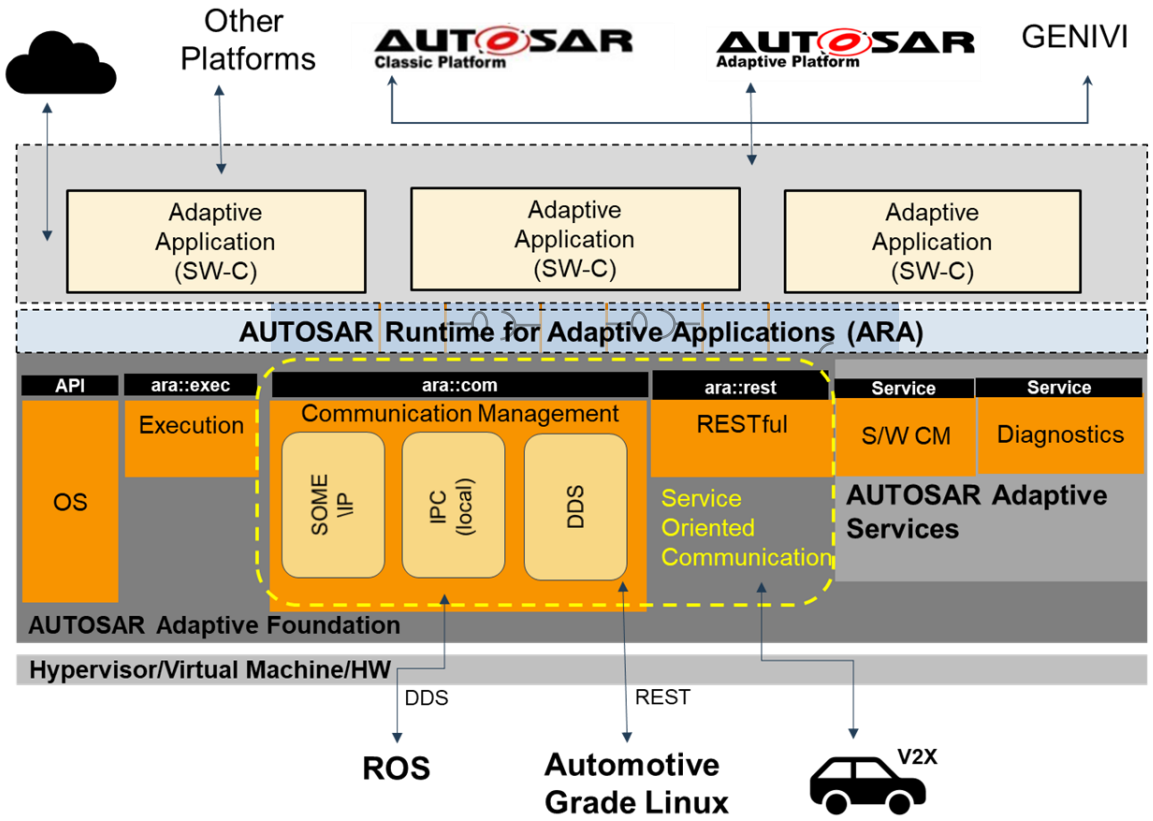
[>> Generate C++ Messages to Communicate Data Between Simulink Components]

Agenda

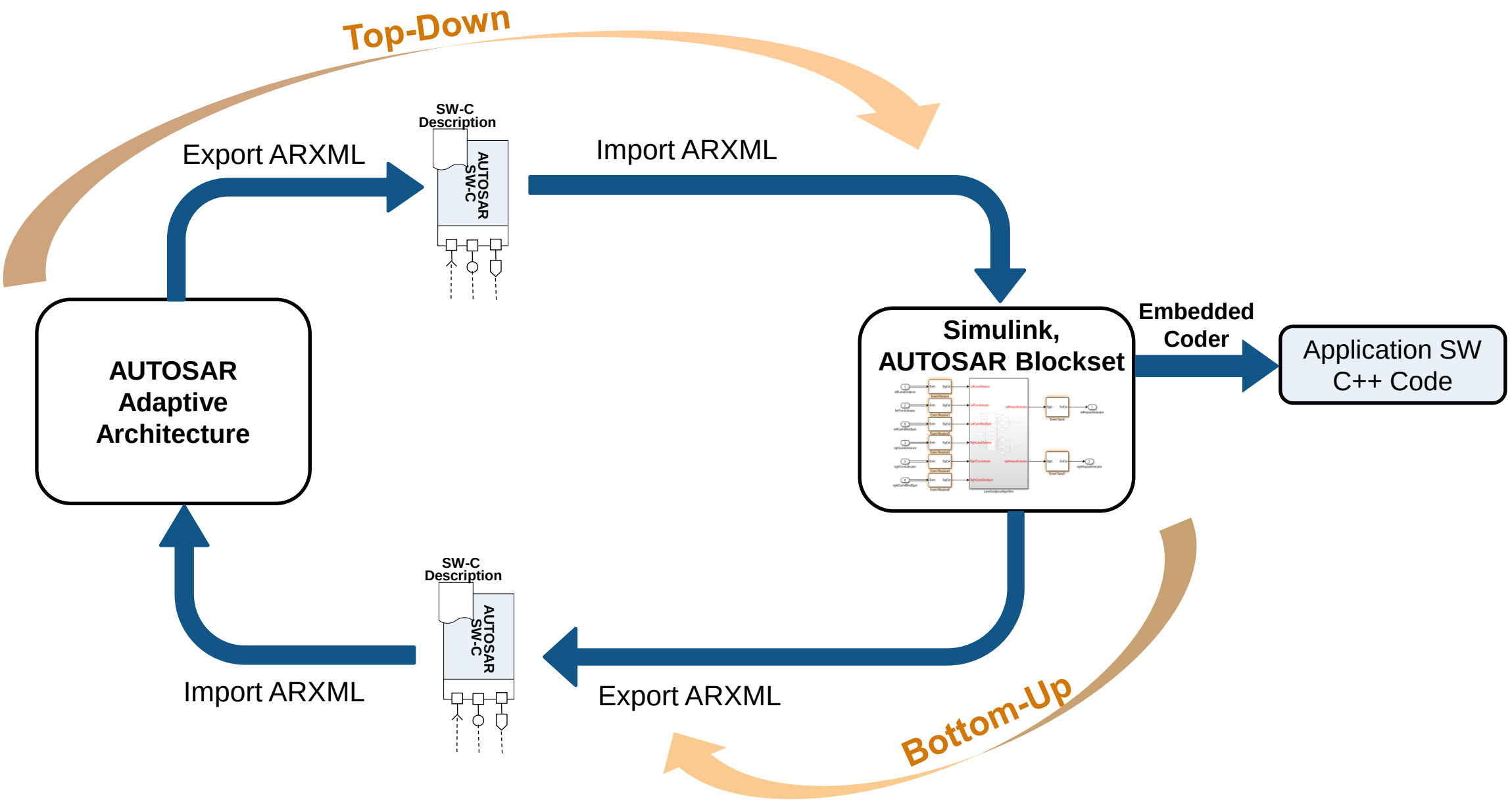
- Software-defined vehicles and new architectures (SOA)
- SOA Concepts
- MathWorks Solutions for SOA
  - Adaptive AUTOSAR
  - DDS/ROS
- Conclusions and key takeaways

AUTOSAR Adaptive

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

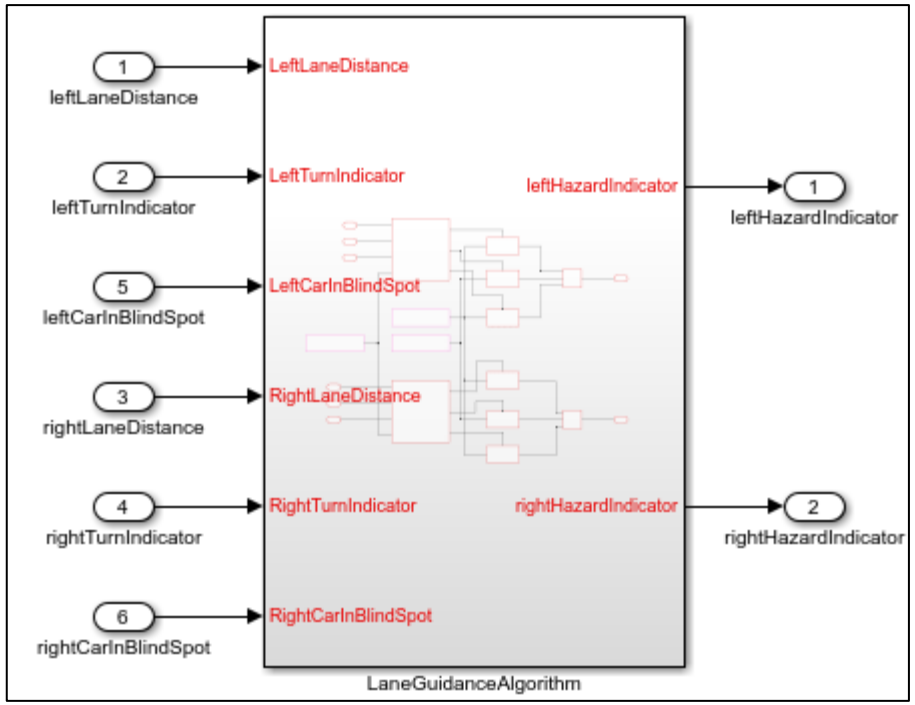


AUTOSAR Adaptive workflows



AUTOSAR Adaptive in action

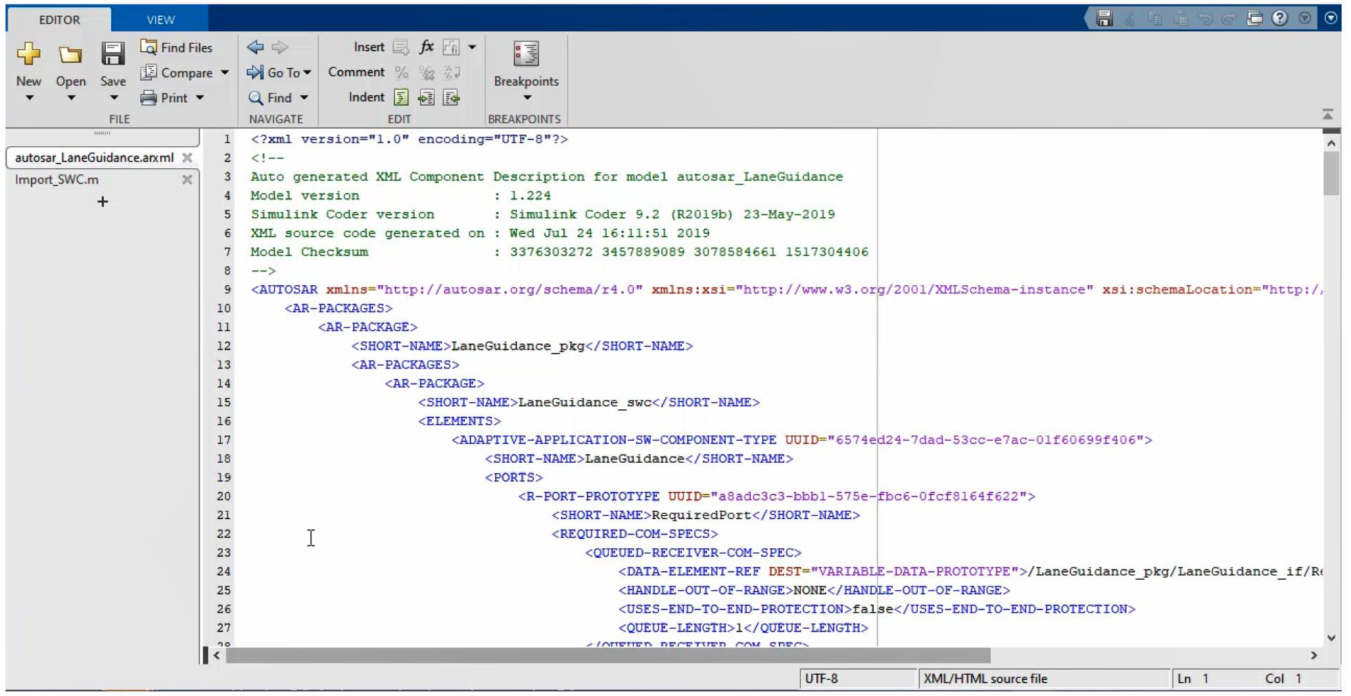
Legacy Simulink model



Bottom-Up

OR

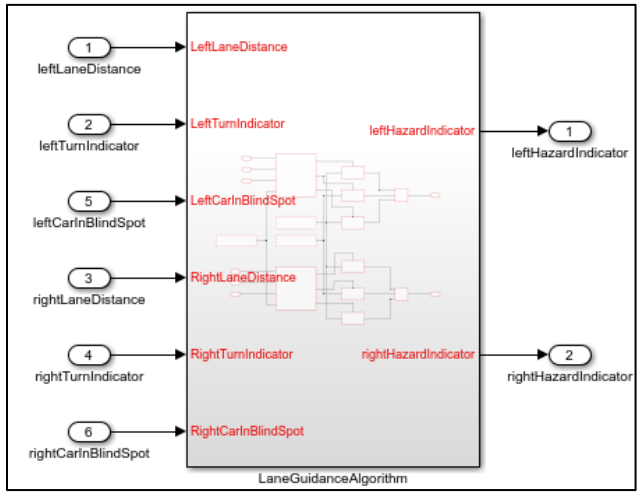
Start from an AUTOSAR Adaptive ARXML

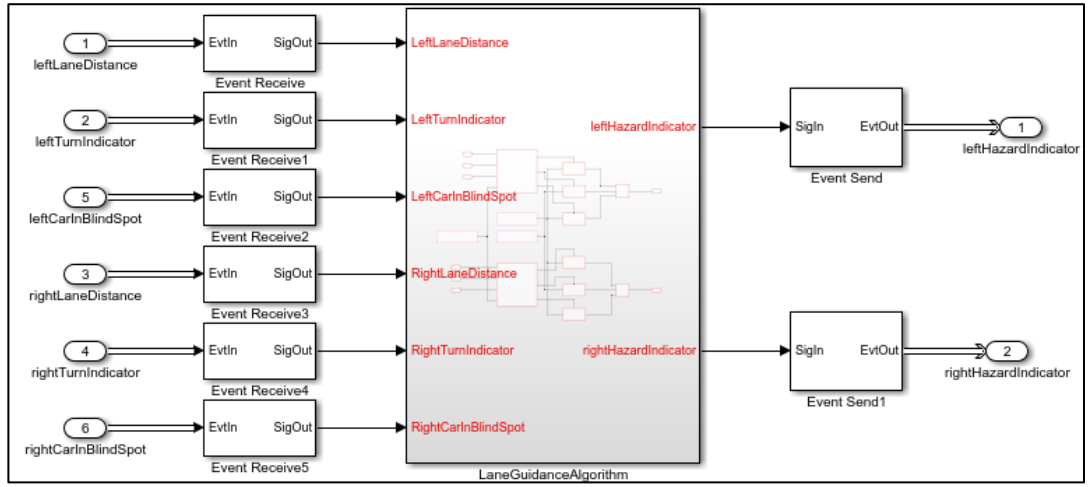


Top-Down

AUTOSAR Adaptive in action: bottom-up

Add blocks to make the necessary event and signal connections





Add Message ports and Message-Signal conversion blocks

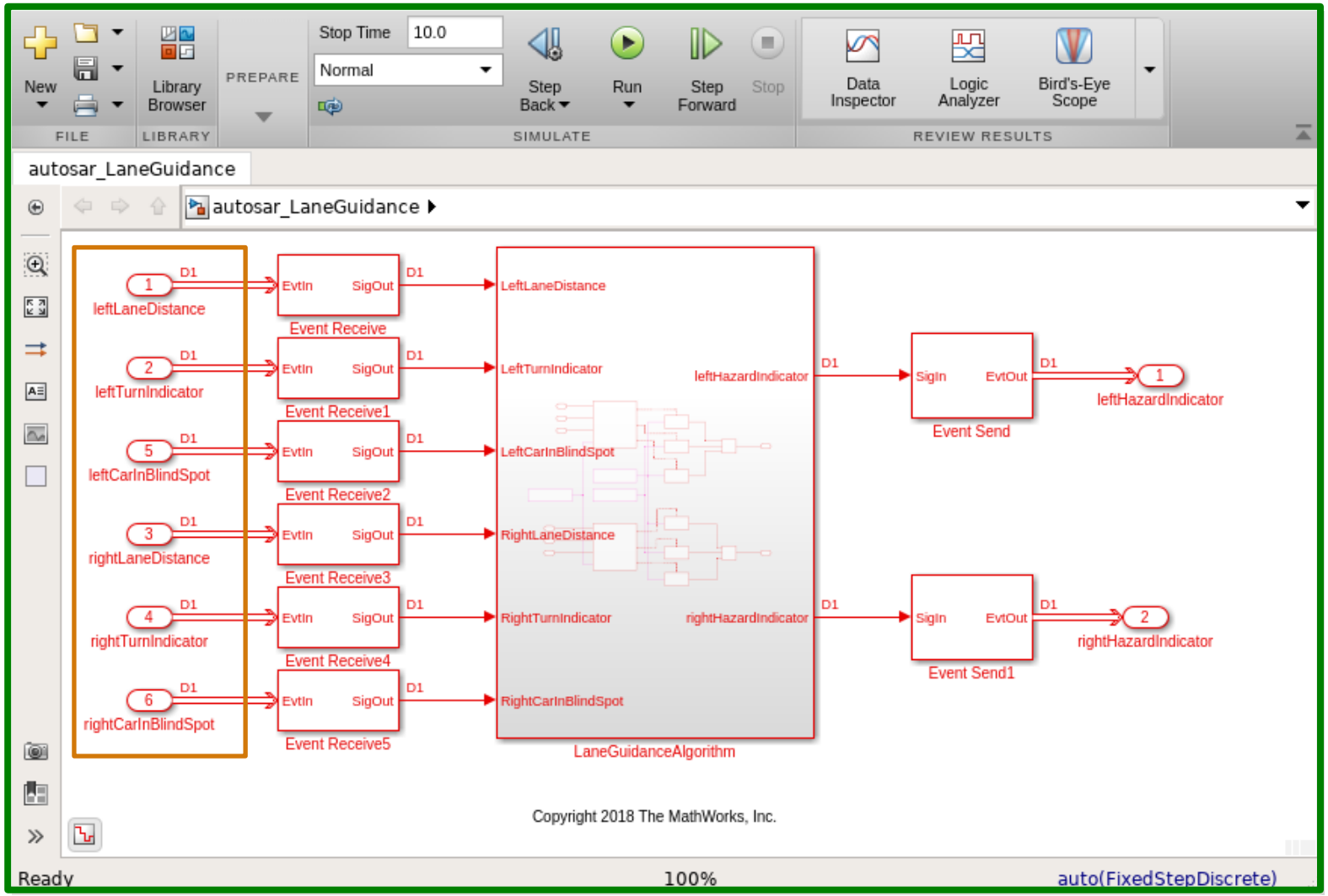
Modelling an AUTOSAR Adaptive application in Simulink

AUTOSAR

Adaptive Application

RequiredPort

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



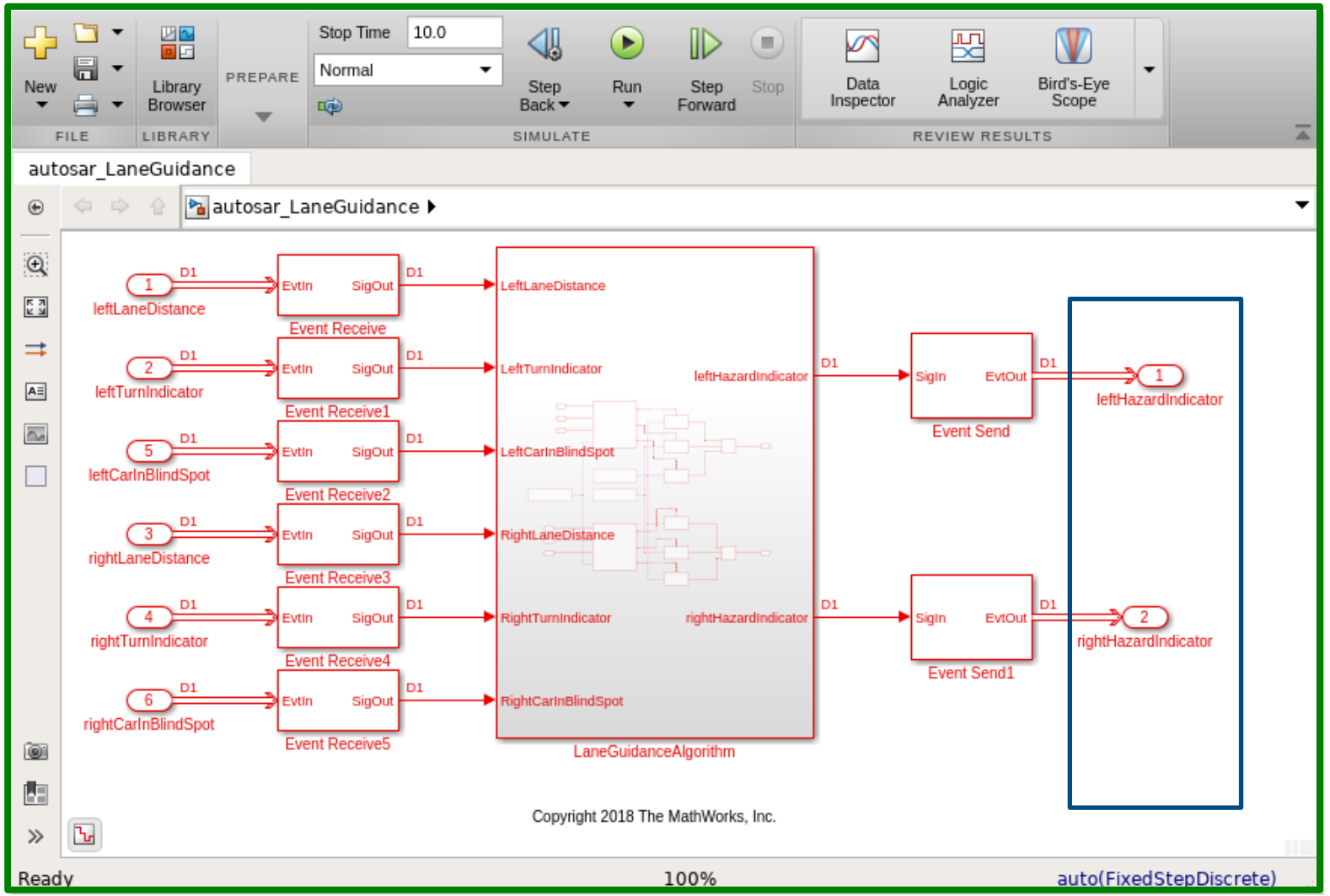
Modelling an AUTOSAR Adaptive application in Simulink

AUTOSAR

Adaptive Application

ProvidedPort

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



Modelling an AUTOSAR Adaptive application in Simulink

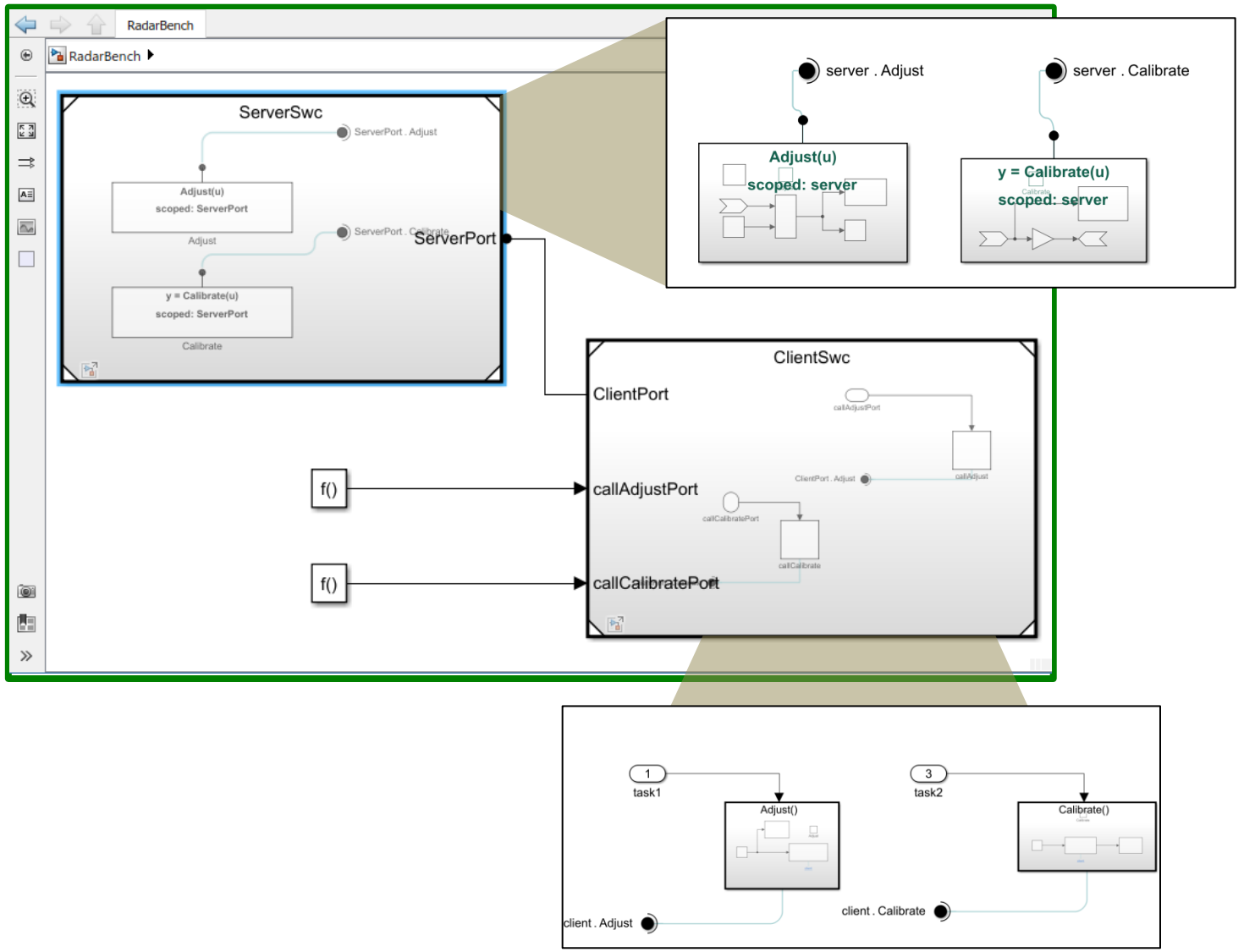
AUTOSAR

Client

Adaptive Application

Server

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

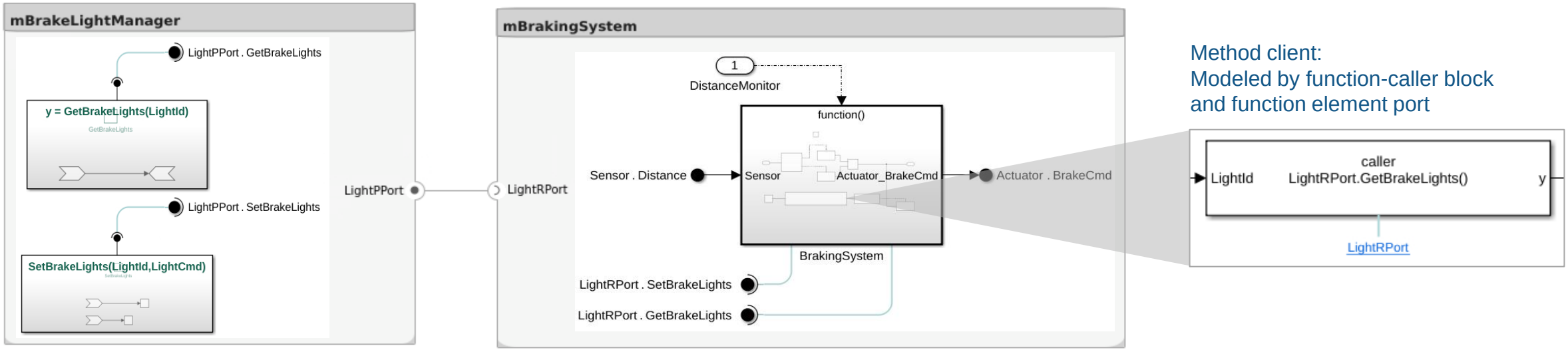


Model synchronous/blocking methods

- 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;
  }
  ...
}
```



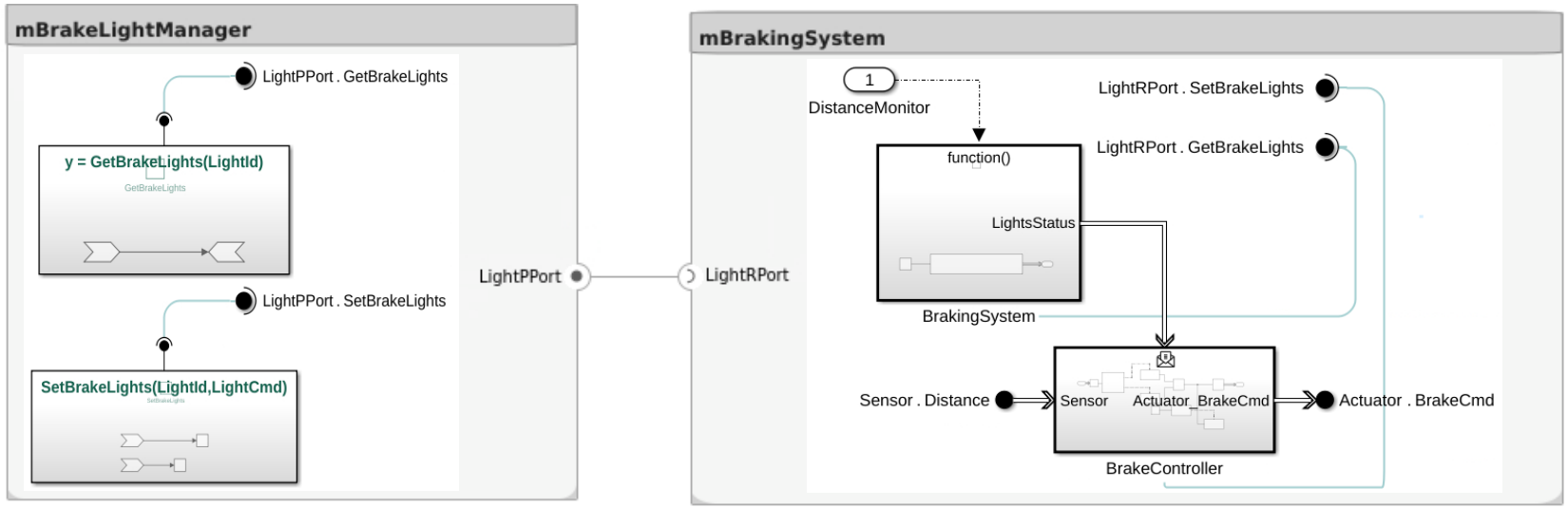
Model asynchronous/non-blocking methods

- 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

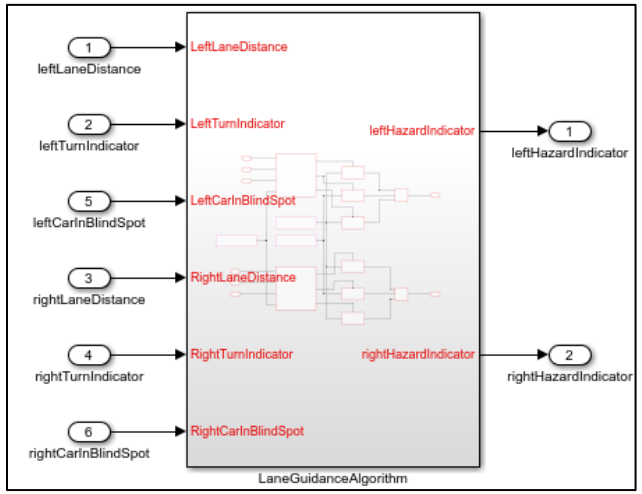


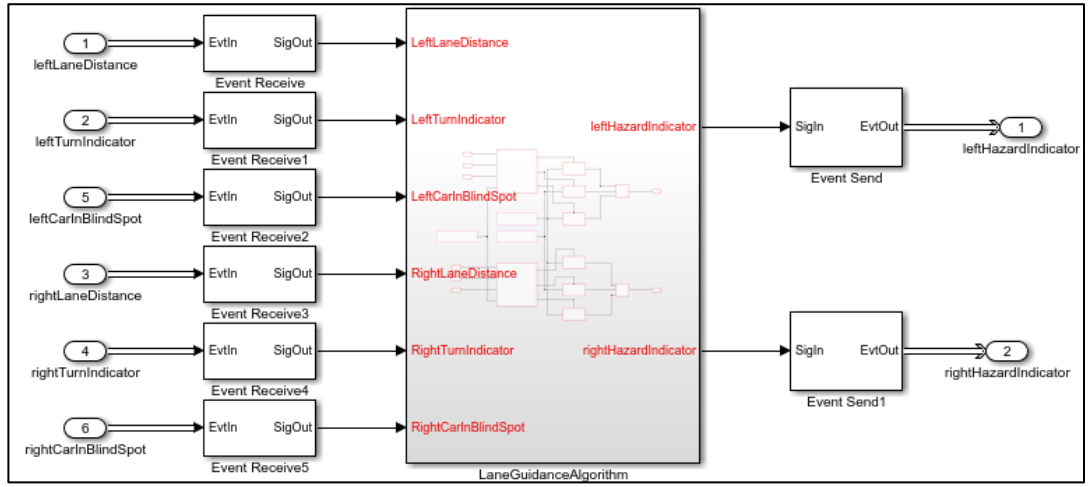

```
void mAsyncBrakingSystem::GetBrakeLightsCallback(
  ara::core::Future<GetBrakeLights::Output>
futureObj)
{
  auto GetBrakeLightsResult = futureObj.GetResult();
  if (GetBrakeLightsResult.HasValue()) {
    GetBrakeLights::Output callOutput =
        GetBrakeLightsResult.Value();
    BrakeController(callOutput.y);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

Method callback body

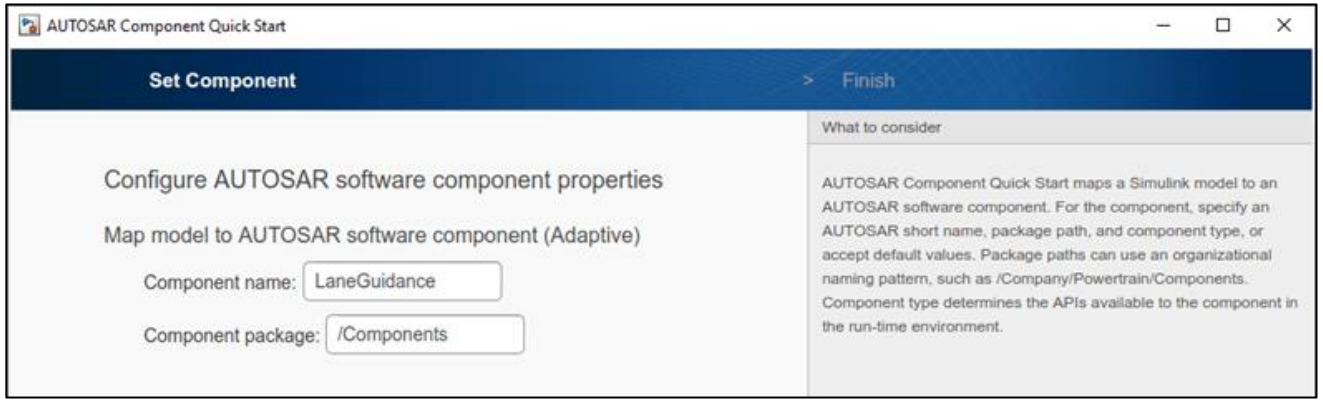
AUTOSAR Adaptive in action: bottom-up

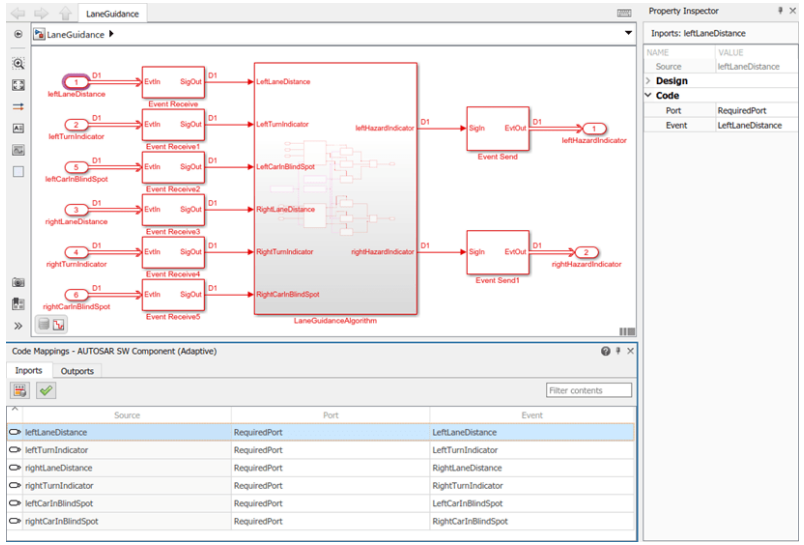
Add blocks to make the necessary event and signal connections





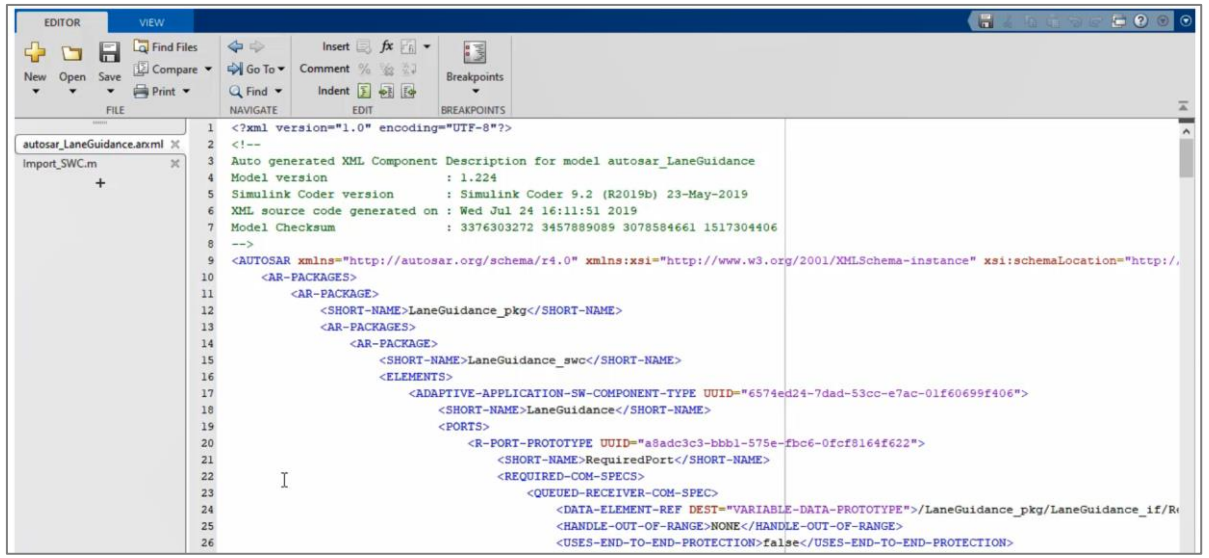
AUTOSAR Quick Start





AUTOSAR Adaptive in action: top-down

- Create model from ARXML

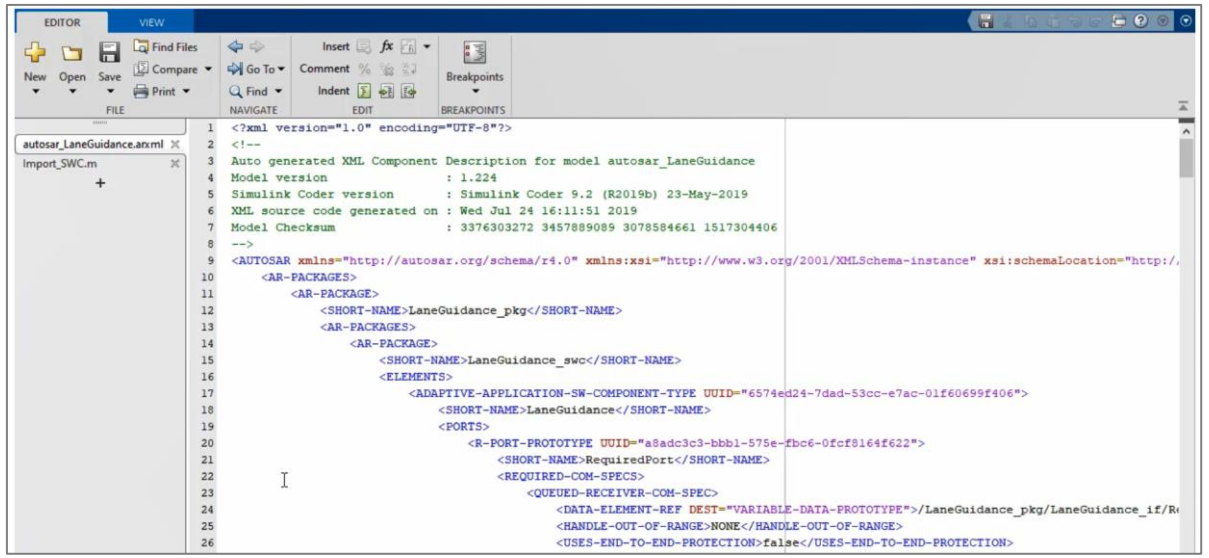


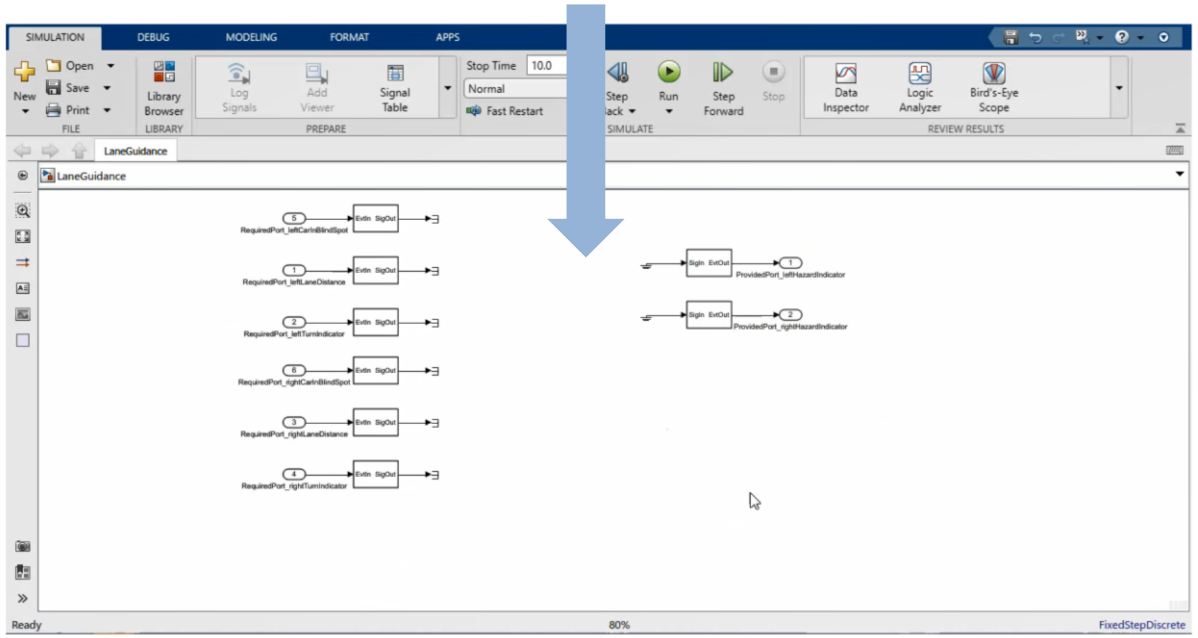

```
ar = arxml.importer({'fusion_app.arxml','radarService_app_mod.arxml','radar_svc_mod.arxml','stdtypes_mod.arxml'});
names = getComponentNames(ar)
```

AUTOSAR Adaptive in action: top-down

- Create model from ARXML

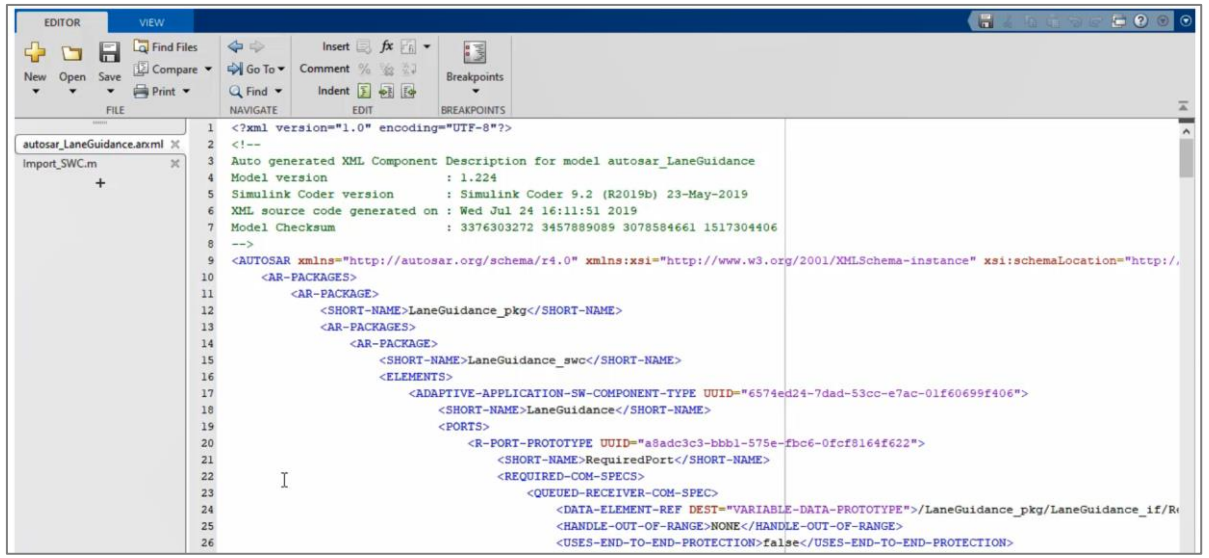
```
createComponentAsModel(ar,'/RadarFusion/fusion');
```

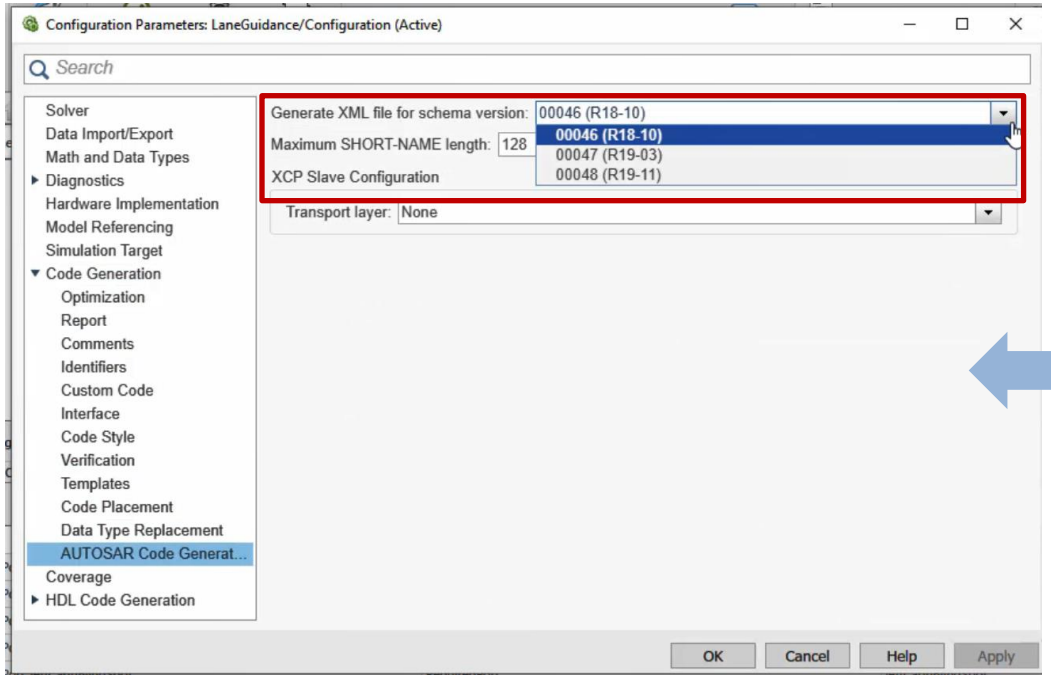


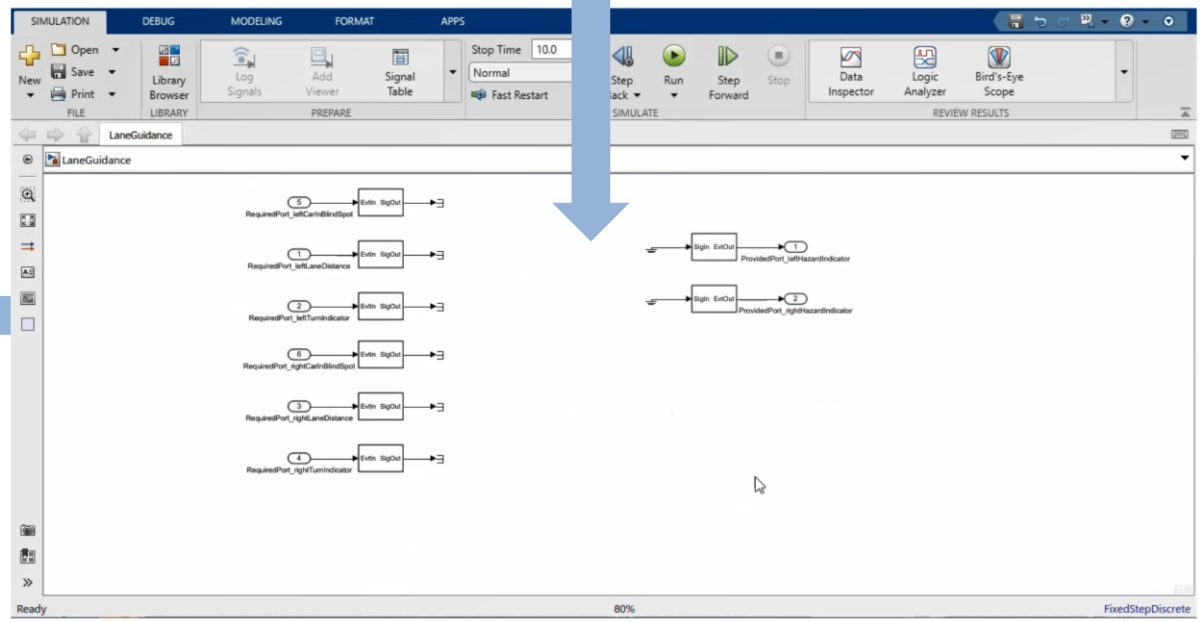


AUTOSAR Adaptive in action: top-down

- Create model from ARXML

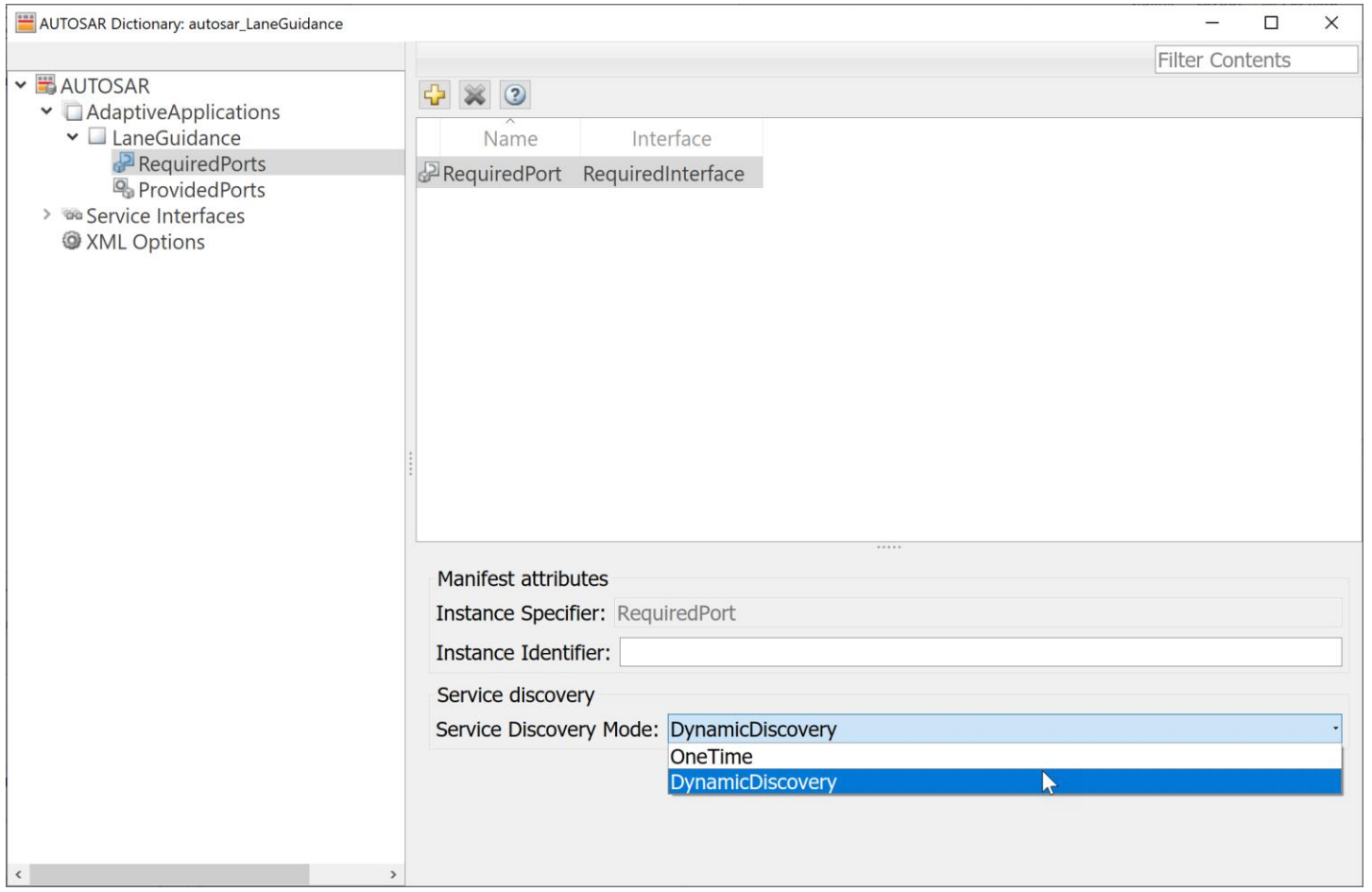






AUTOSAR Adaptive in action: top-down

- Create model from ARXML
- Configure Service Discovery



AUTOSAR Adaptive in action: top-down

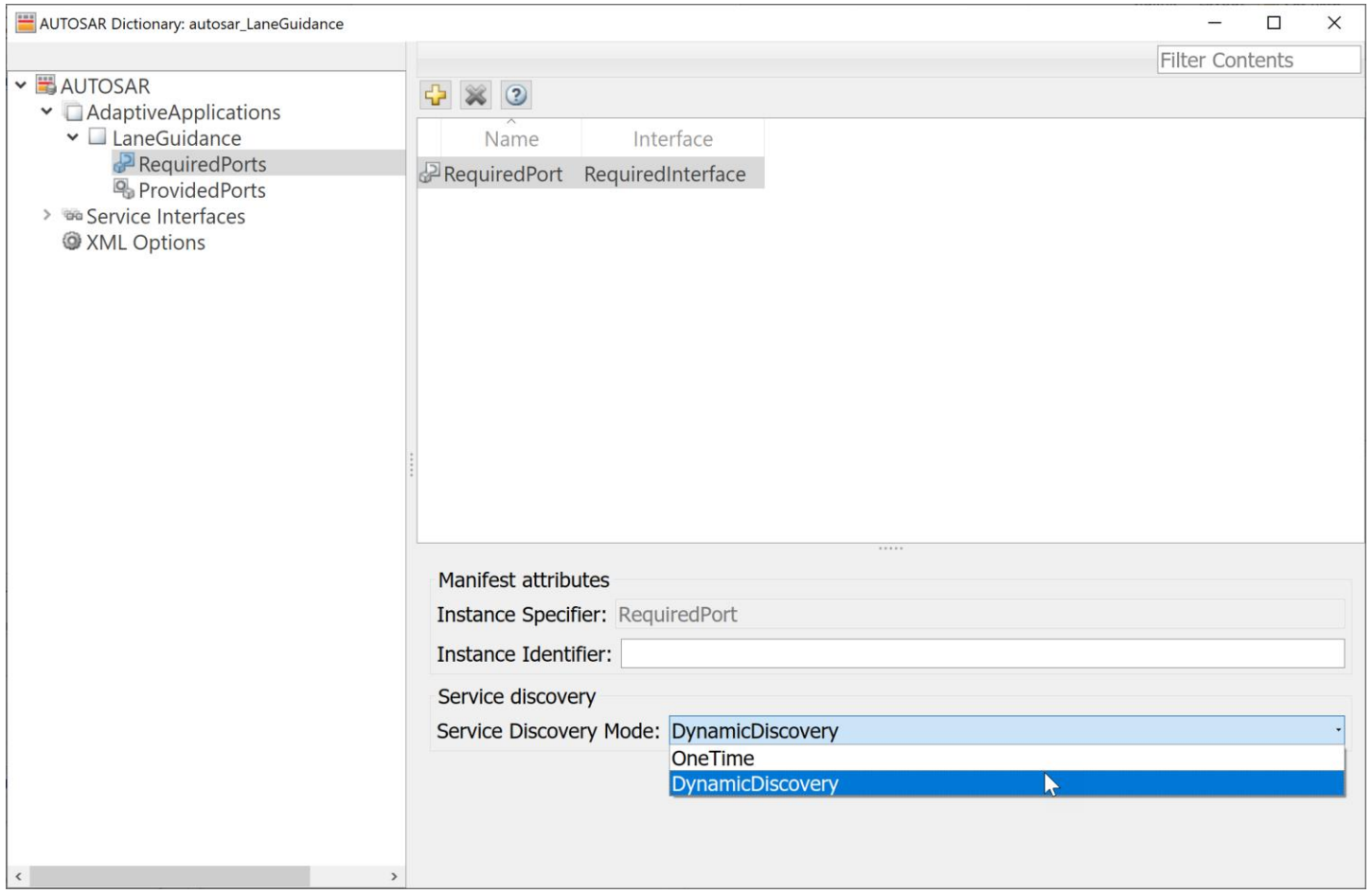
- Create model from ARXML

- Configure Service Discovery

  Subscribe to adaptive services

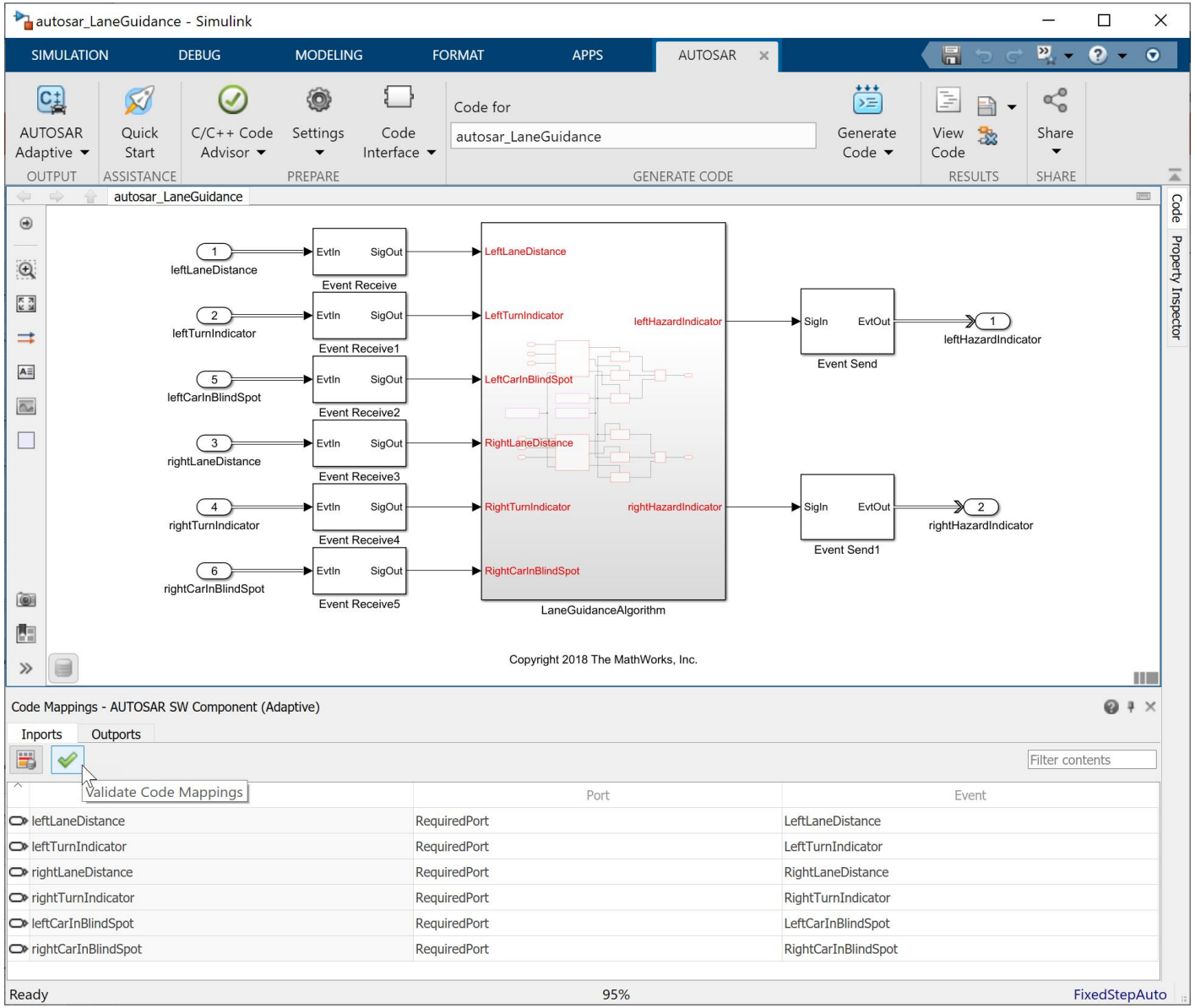
  – Only at startup, or

  – Dynamically, as they become available



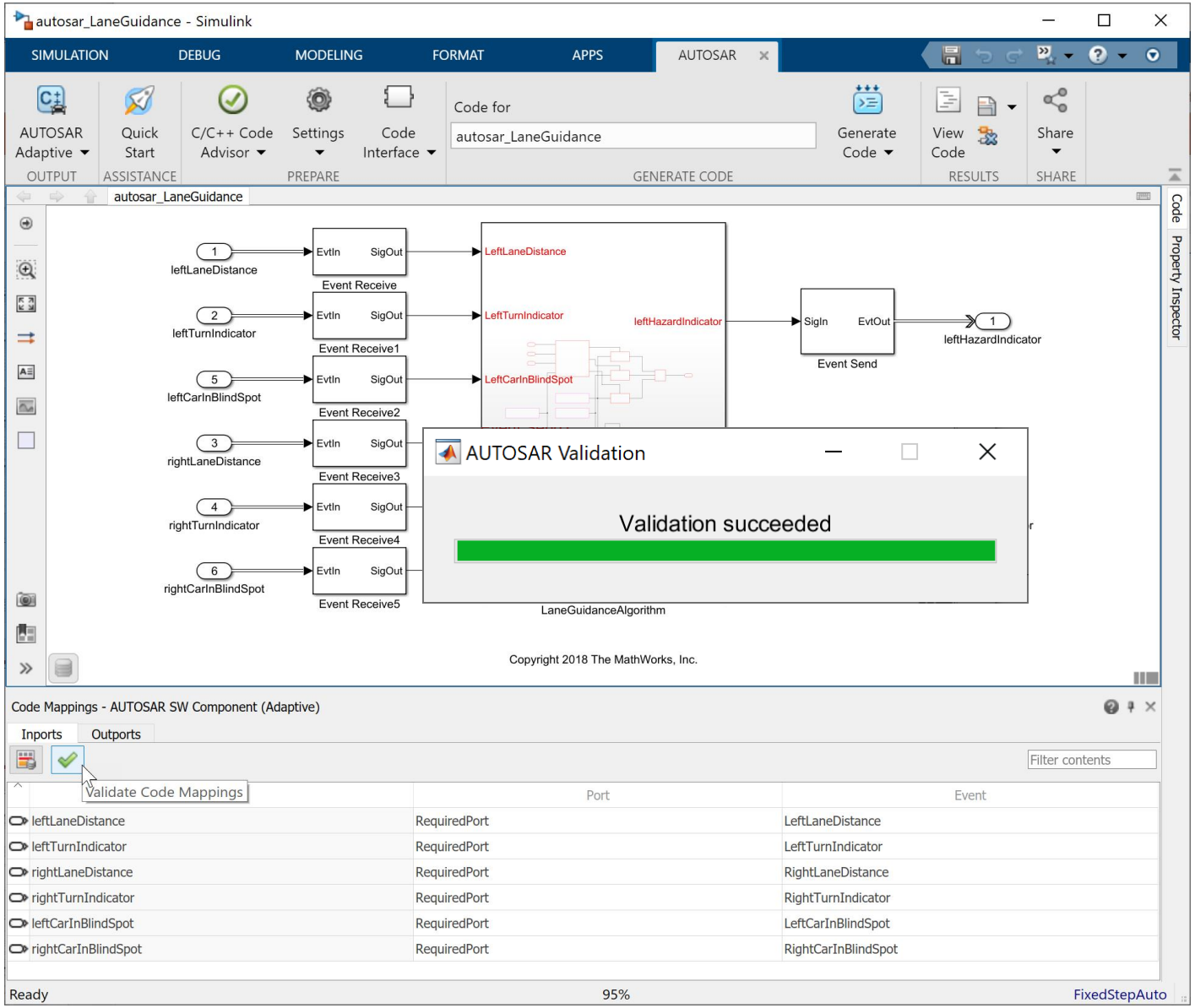
AUTOSAR Adaptive in action: top-down

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



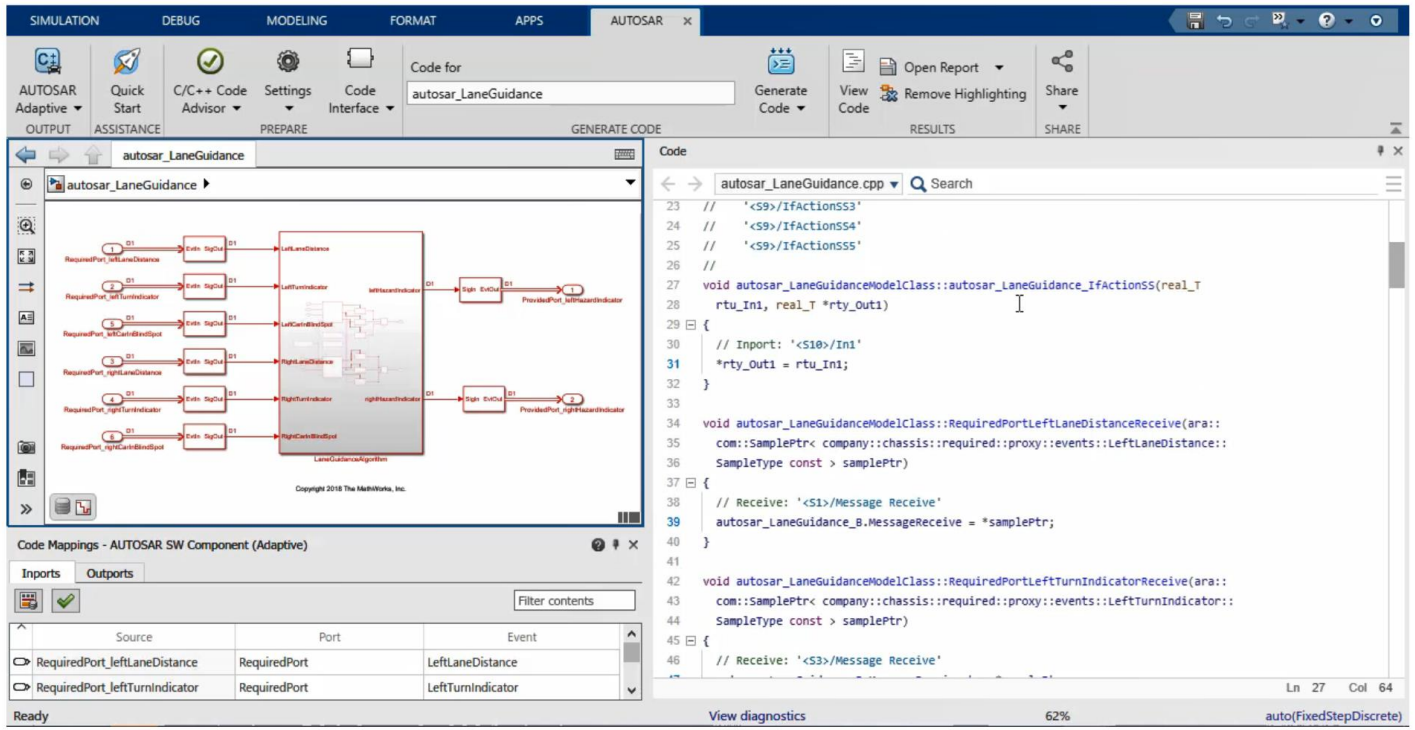
AUTOSAR Adaptive in action: top-down

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



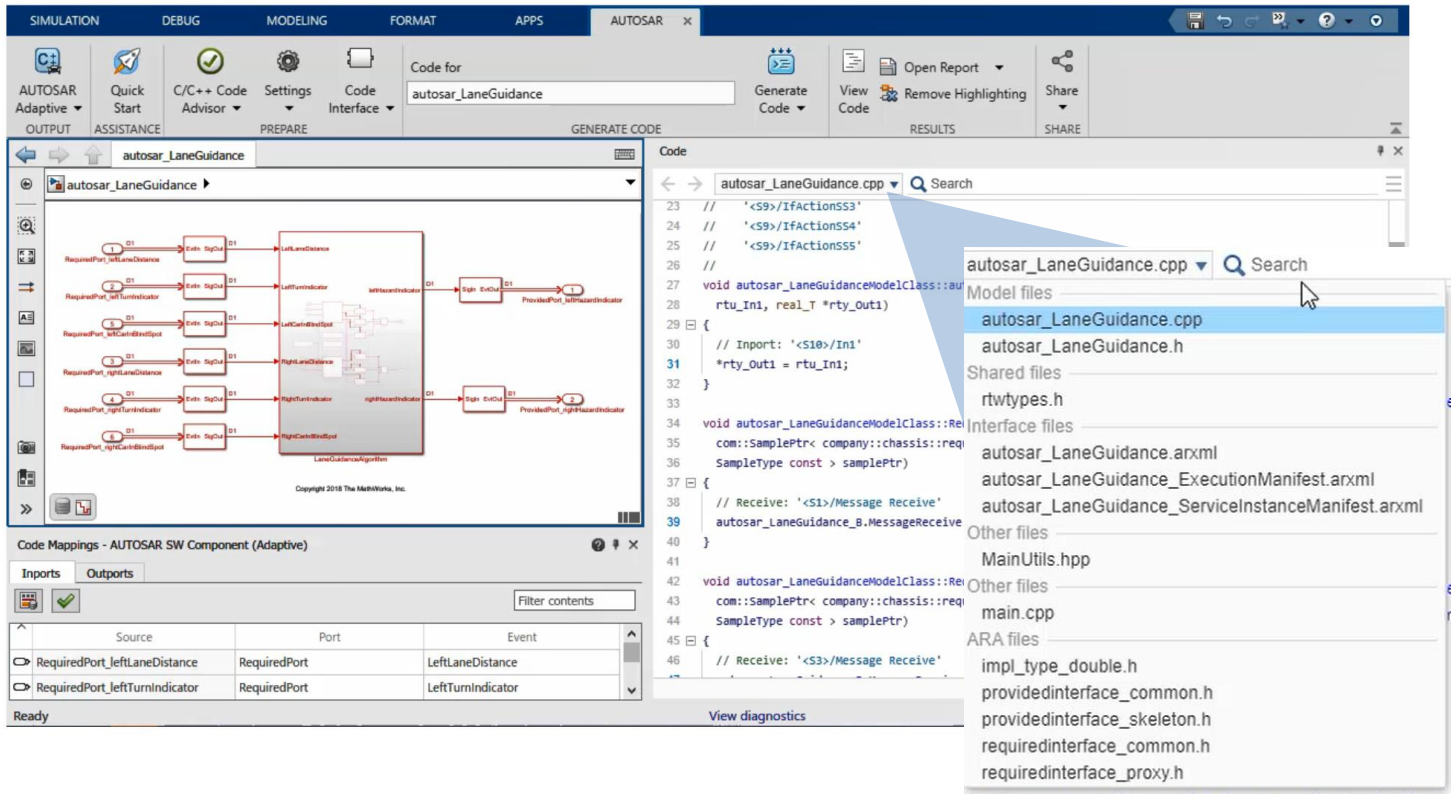
AUTOSAR Adaptive in action: top-down

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



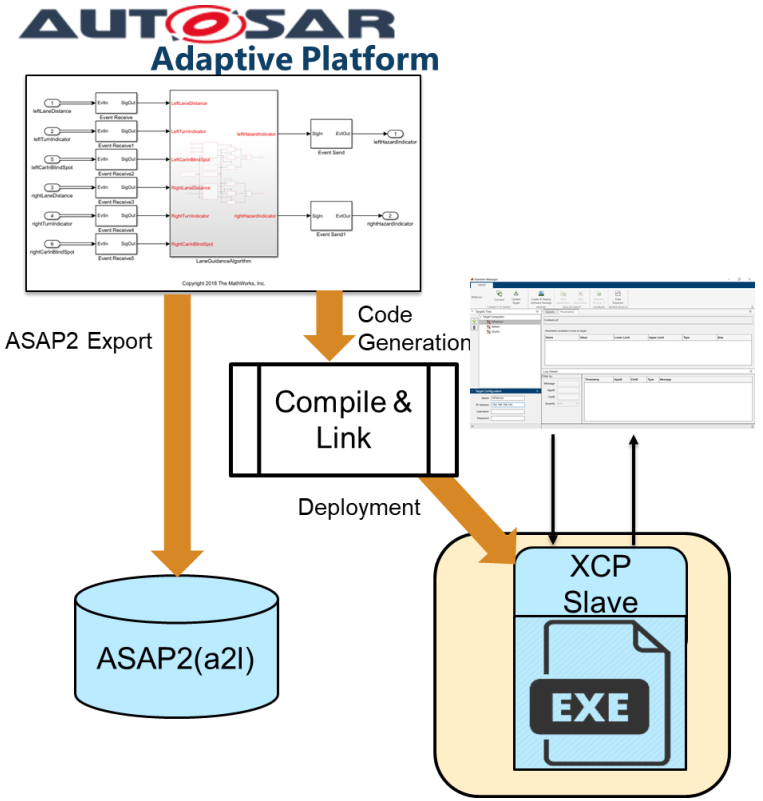
AUTOSAR Adaptive in action

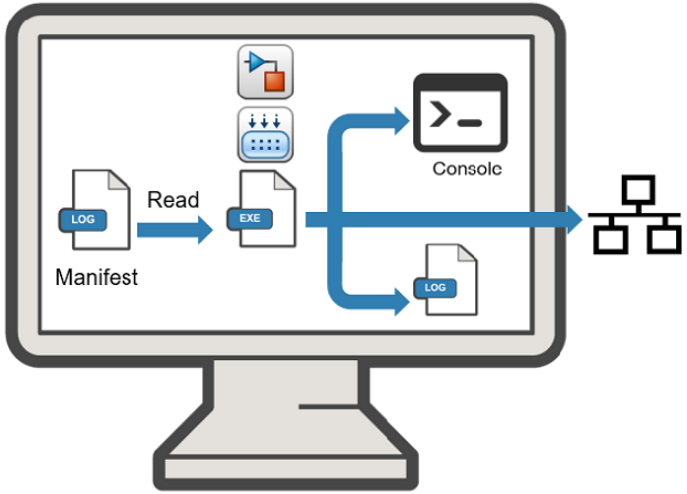
- Create model from ARXML
- Configure Service Discovery
- Verify AUTOSAR properties
- Generate code
  - Integrate Applications with third-party Adaptive stack
  - Create Linux executables for calibration and monitoring



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





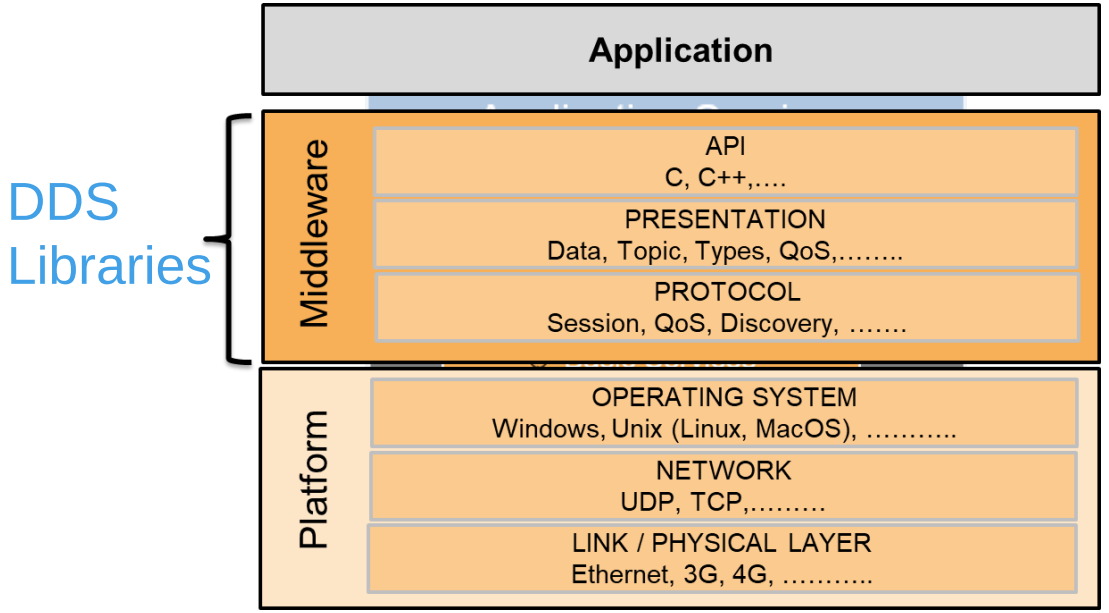
Agenda

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

Simulink for DDS

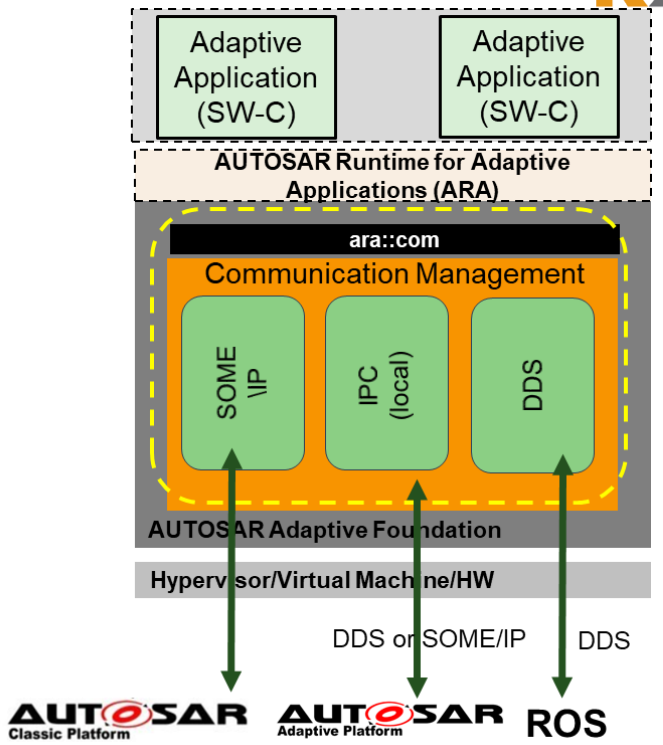
Data Distribution Services (DDS) uses SOA methodology, and directly addresses publish and subscribe communications for real-time and embedded systems.

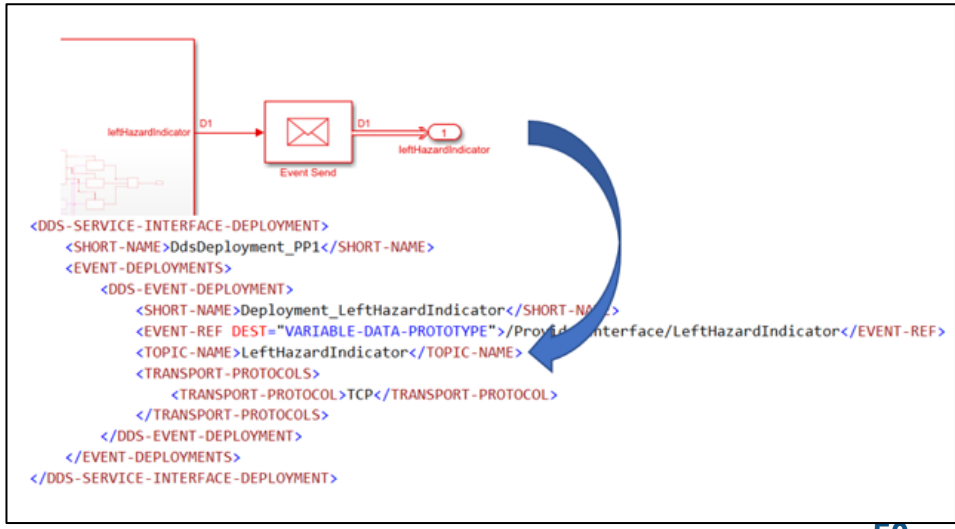
DDS addresses the needs of applications that require real-time data exchange in industries like aerospace and defense, automotive, and robotics.



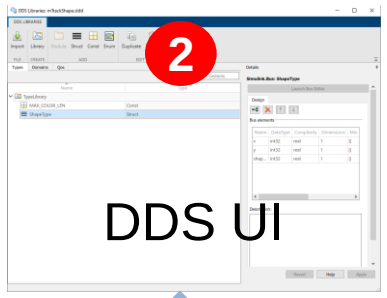
DDS (Data Distribution Services) is part of AUTOSAR Adaptive Deployment

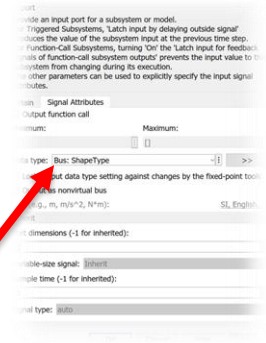
- Supports DDS binding for ara::com enabling communication between adaptive AUTOSAR applications
  - Generated `ServiceInstanceManifest.arxml` contains DDS deployment artifacts

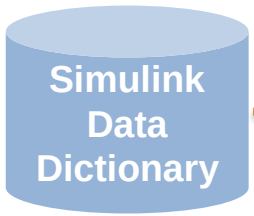




User Workflow with DDS Blockset

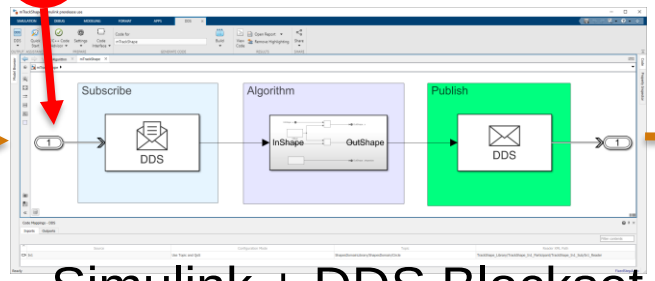






Import

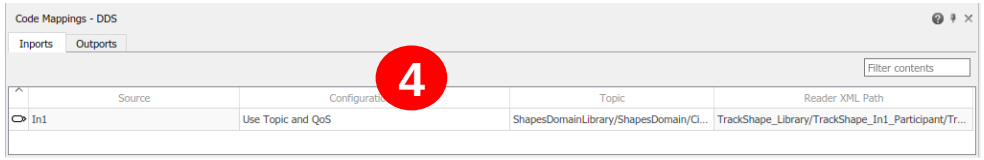
1



Simulink + DDS Blockset

5

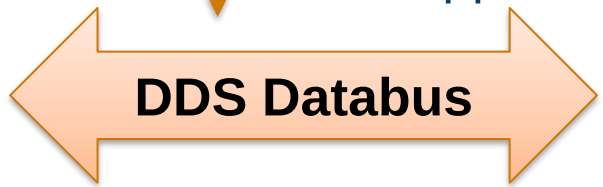
Generate

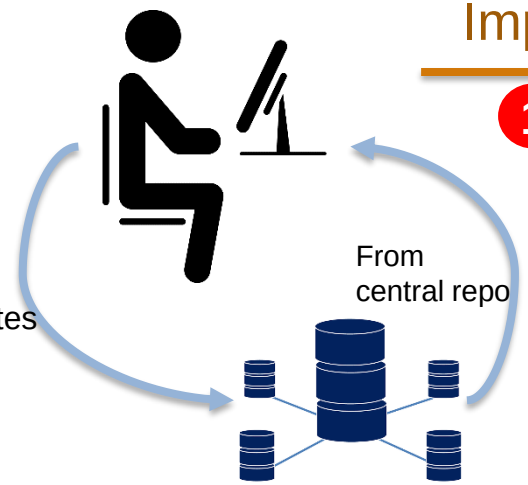


6

Deploy

IDL/XML
+
Vendor
support

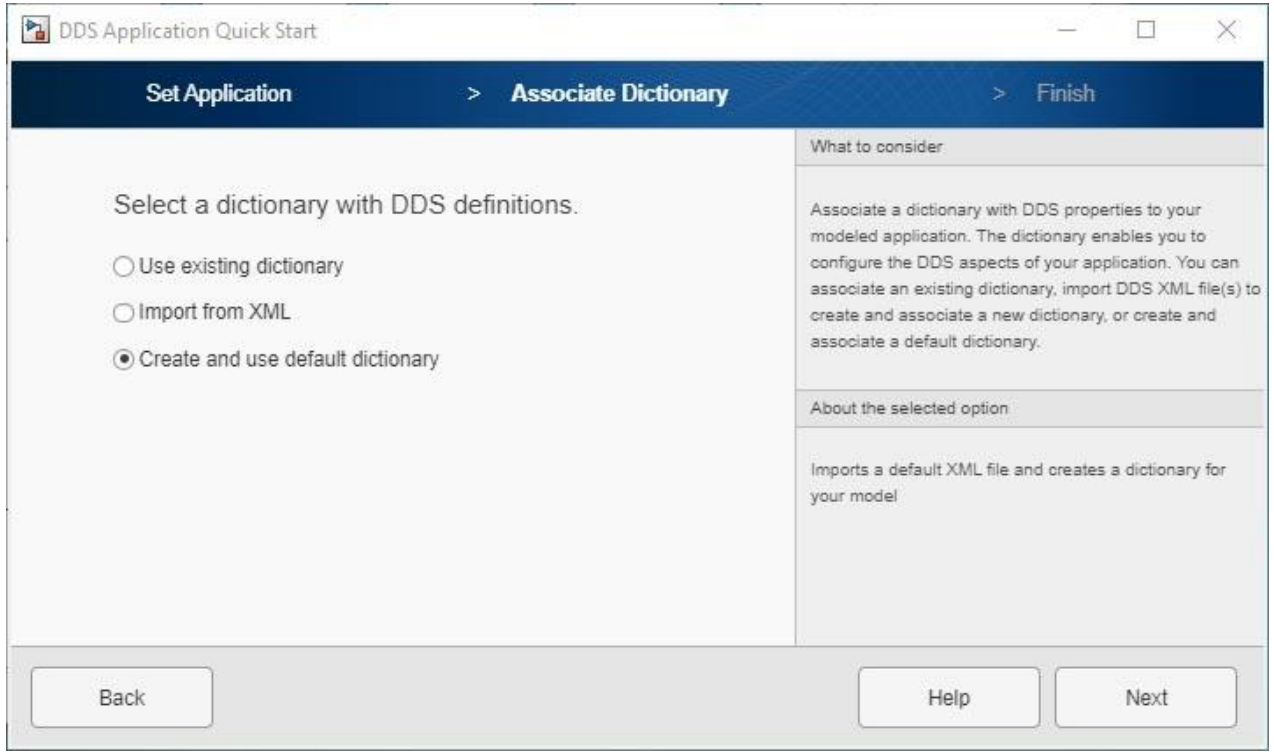




Interface definitions
QoS definitions
(XML)

DDS Blockset in action

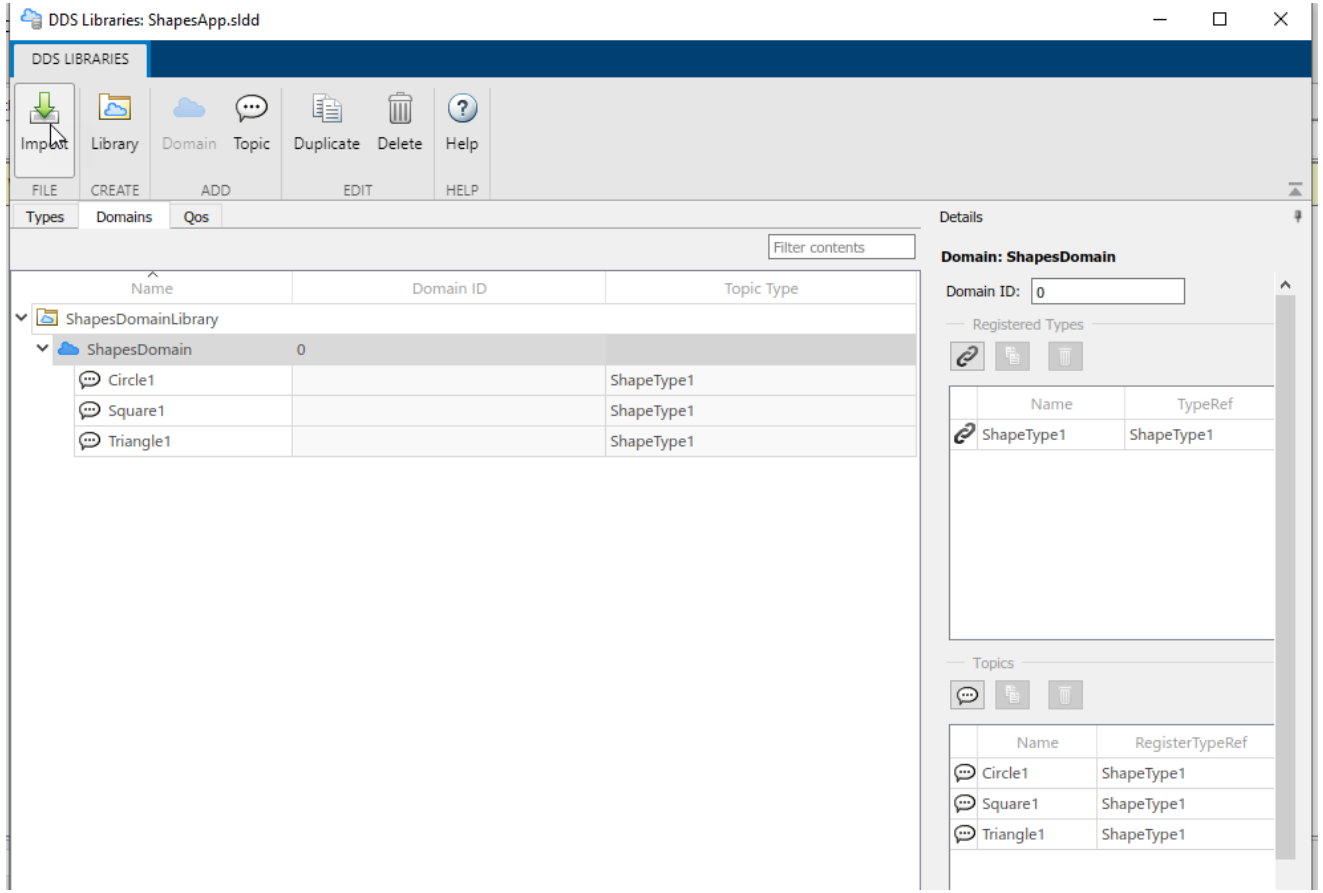
- Import DDS definitions from XML or create new Definitions



DDS Blockset in action

- Import DDS definitions from XML or create new Definitions

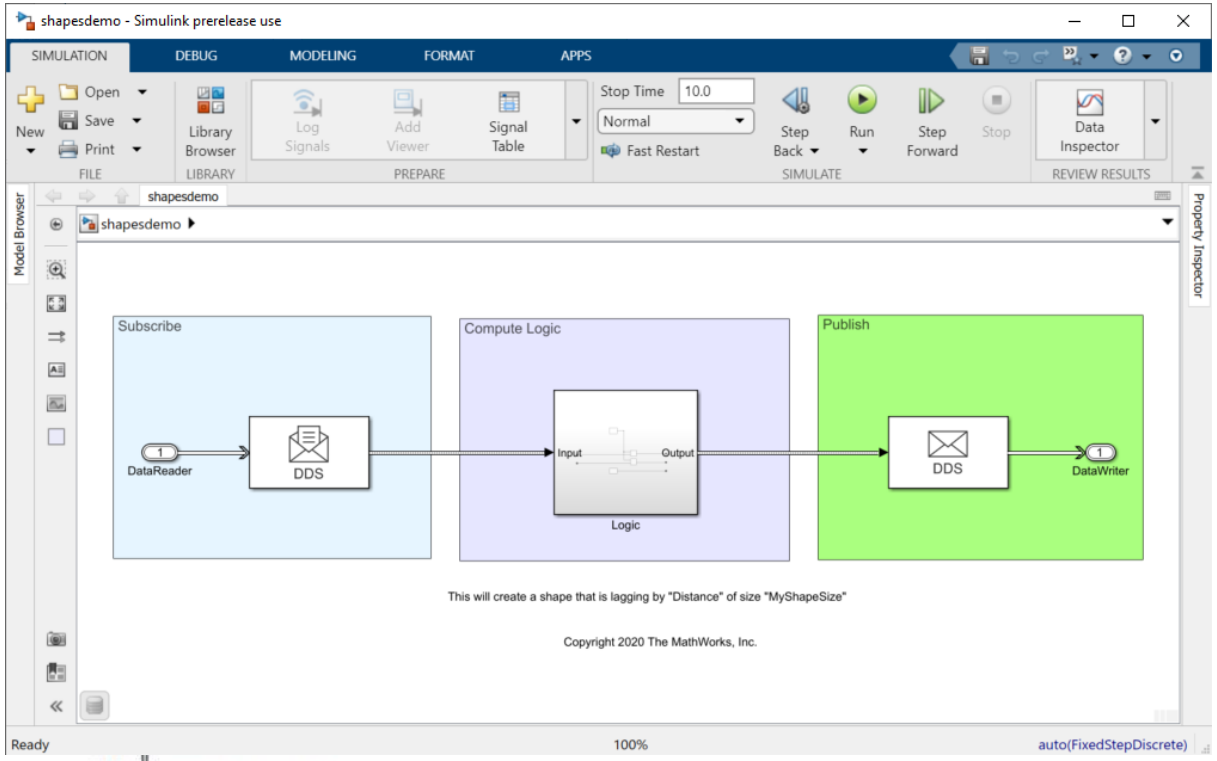
- Define/Modify DDS definitions in DDS Dictionary
  - Topic Types
  - Domains
  - QoS

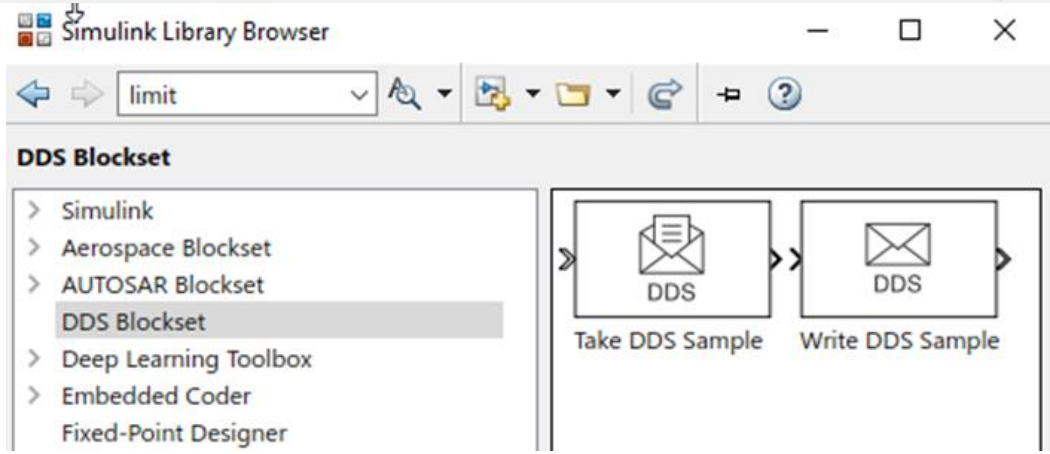


DDS Blockset in action

- 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

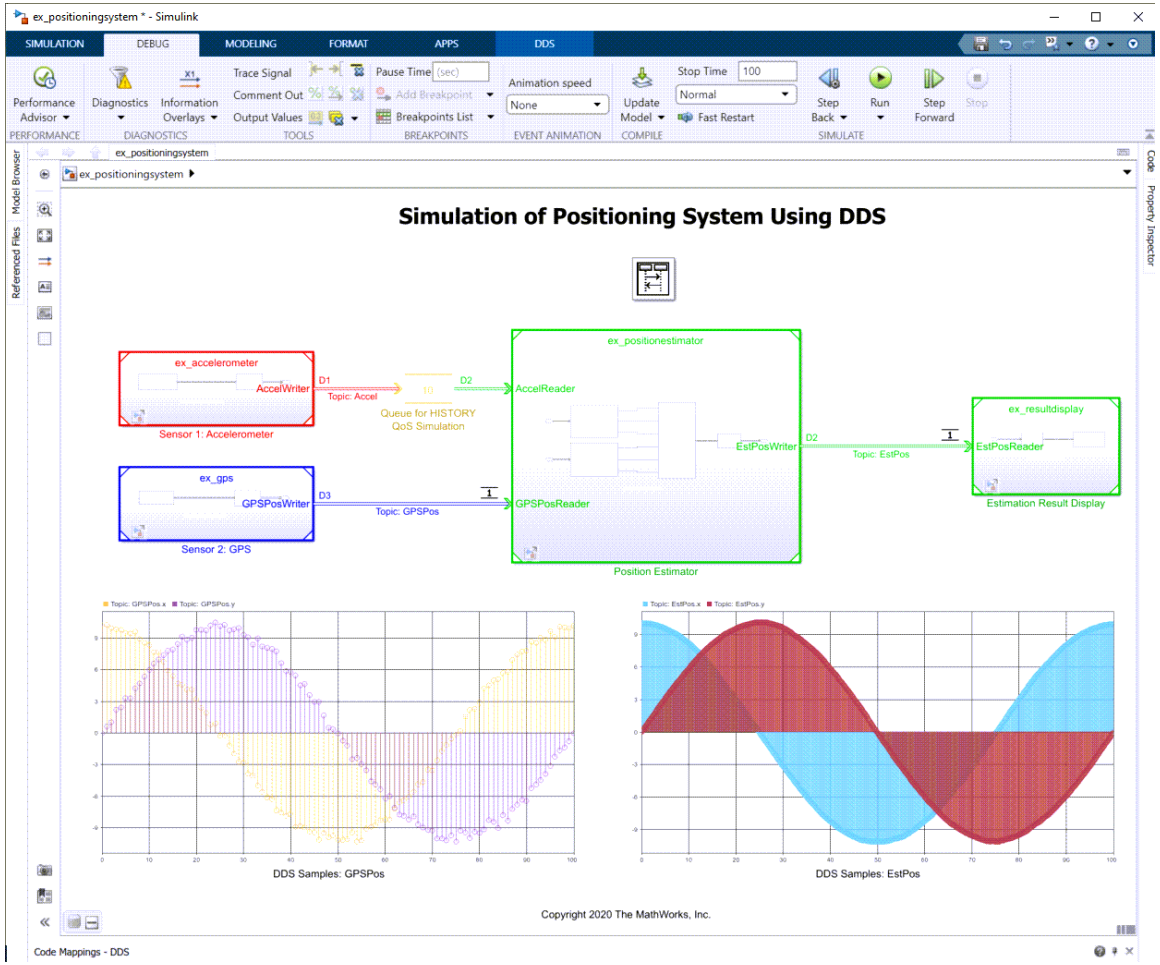




DDS Blockset in action

- 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

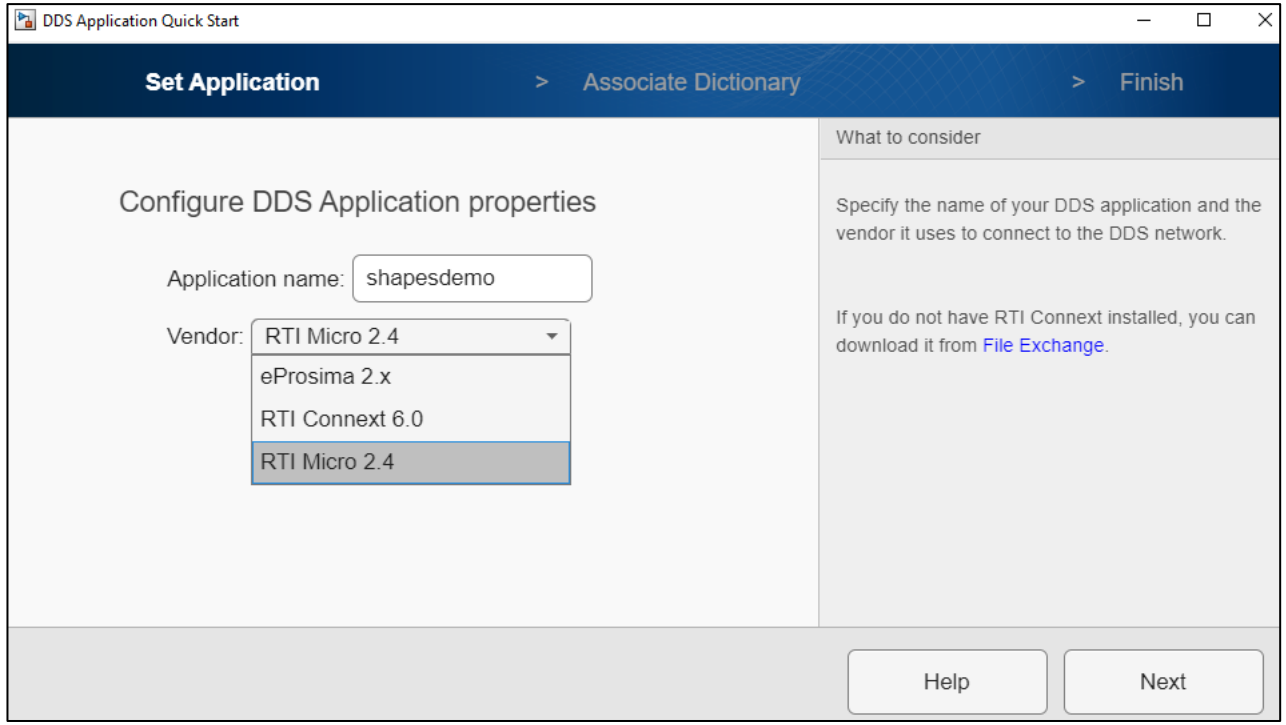
- Import DDS definitions from XML or create new Definitions
- Define/Modify DDS definitions in DDS Dictionary
- Model applications
- Simulate DDS models including QoS
- Generate DDS executables and deploy on a DDS network

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 wr
    DDS_DataWriter* writer = getWriter(writerName, participantName);
    PosTypeDataWriter* fooWriter = PosTypeDataWriter_narrow(writer);
    if(!fooWriter) {
        return false;
    }
    const DDS_ReturnCode_t ret = PosTypeDataWriter_write((PosTypeDataWriter*)writer,
    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 Connext, 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**

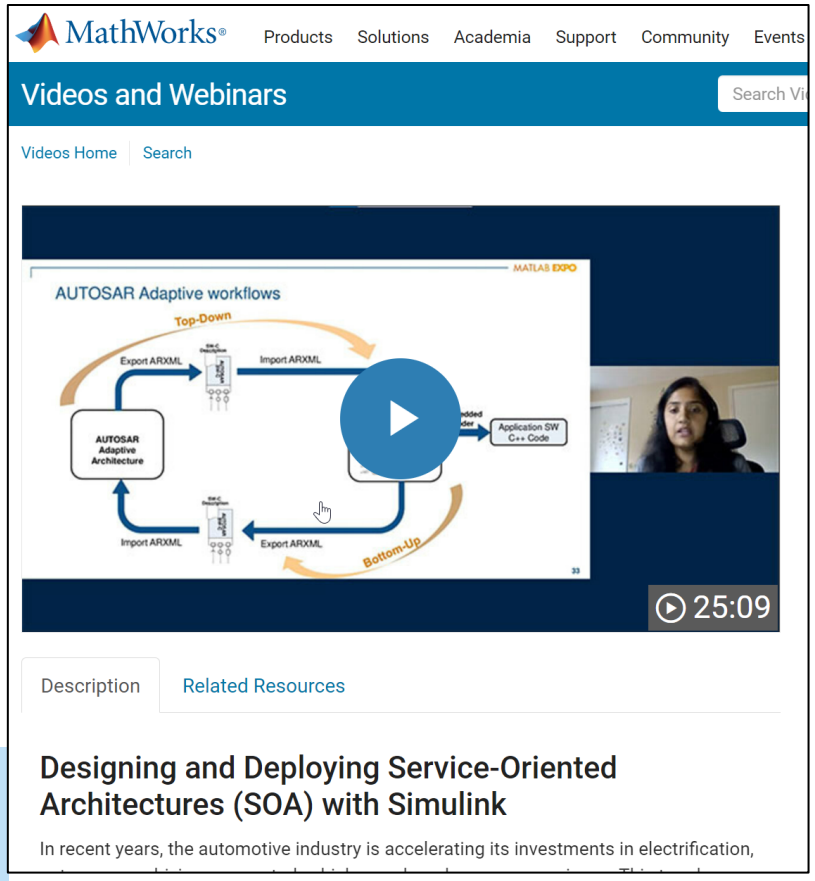
Solutions

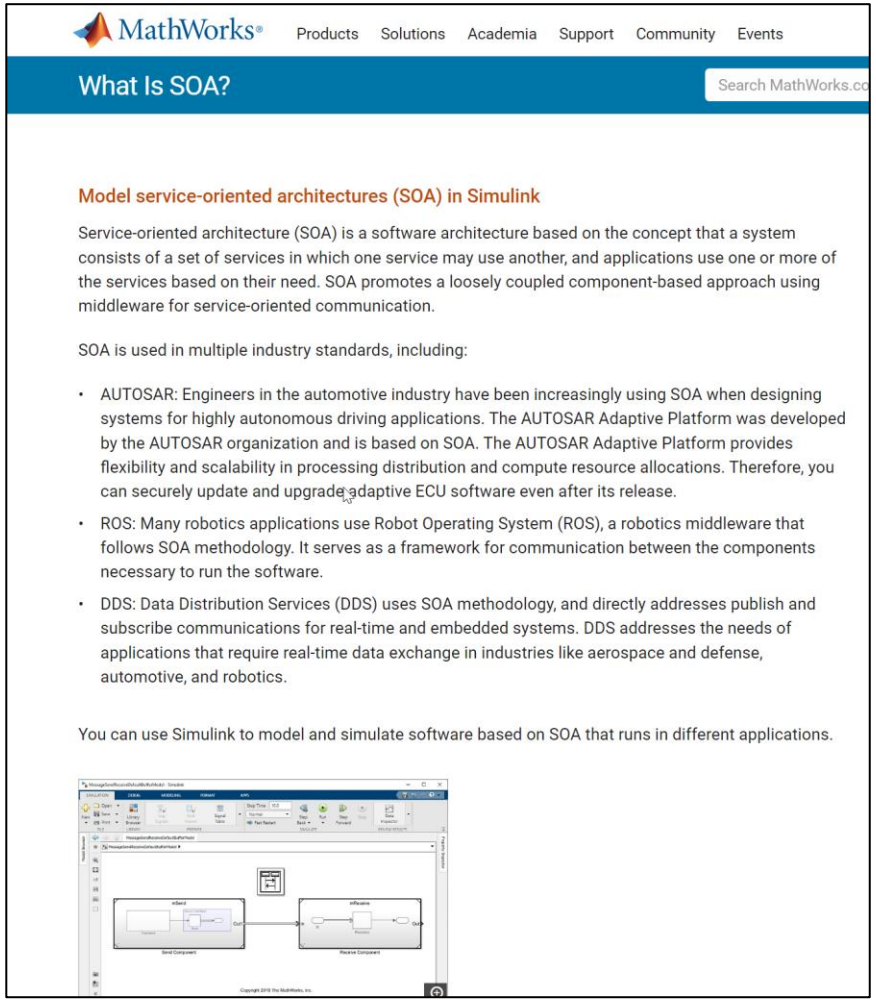
- You can **design, simulate and generate** code to deploy service-oriented applications (including AUTOSAR Adaptive and DDS) in **Simulink**
- You can **reuse your existing expertise and models** to mitigate the risk of migration to SOA applications

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)]





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

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

MATLAB EXPO

Thank you