MathWorks
AUTOMOTIVE
CONFERENCE 2024
North America

Migration of Monolithic Algorithm to Service Oriented Architecture

Mark Danielsen, MathWorks

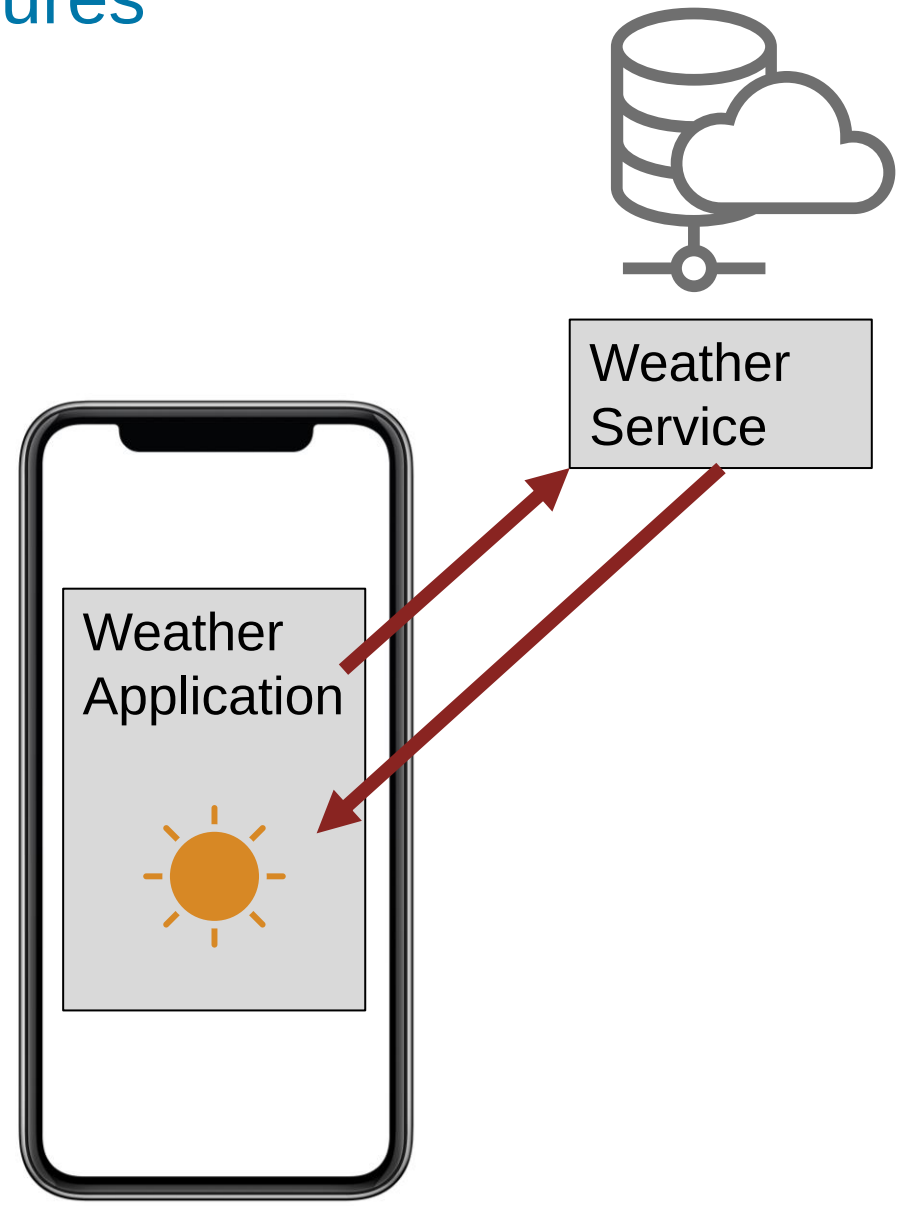
MathWorks

An Example of Service Oriented Architectures

Let's Talk About the Weather …

Mobile Weather Reports

- Weather App calls out to a Weather Service in the cloud
- Weather Service gets request and responds with a block a data that represents current weather info
- Weather App decodes and displays the information



An Example of Service Oriented Architectures

Let's Talk About the Weather …

Mobile Weather Reports

- Weather App calls out to a Weather Service in the cloud
- Weather Service gets request and responds with a block a data that represents current weather info
- Weather App decodes and displays the information

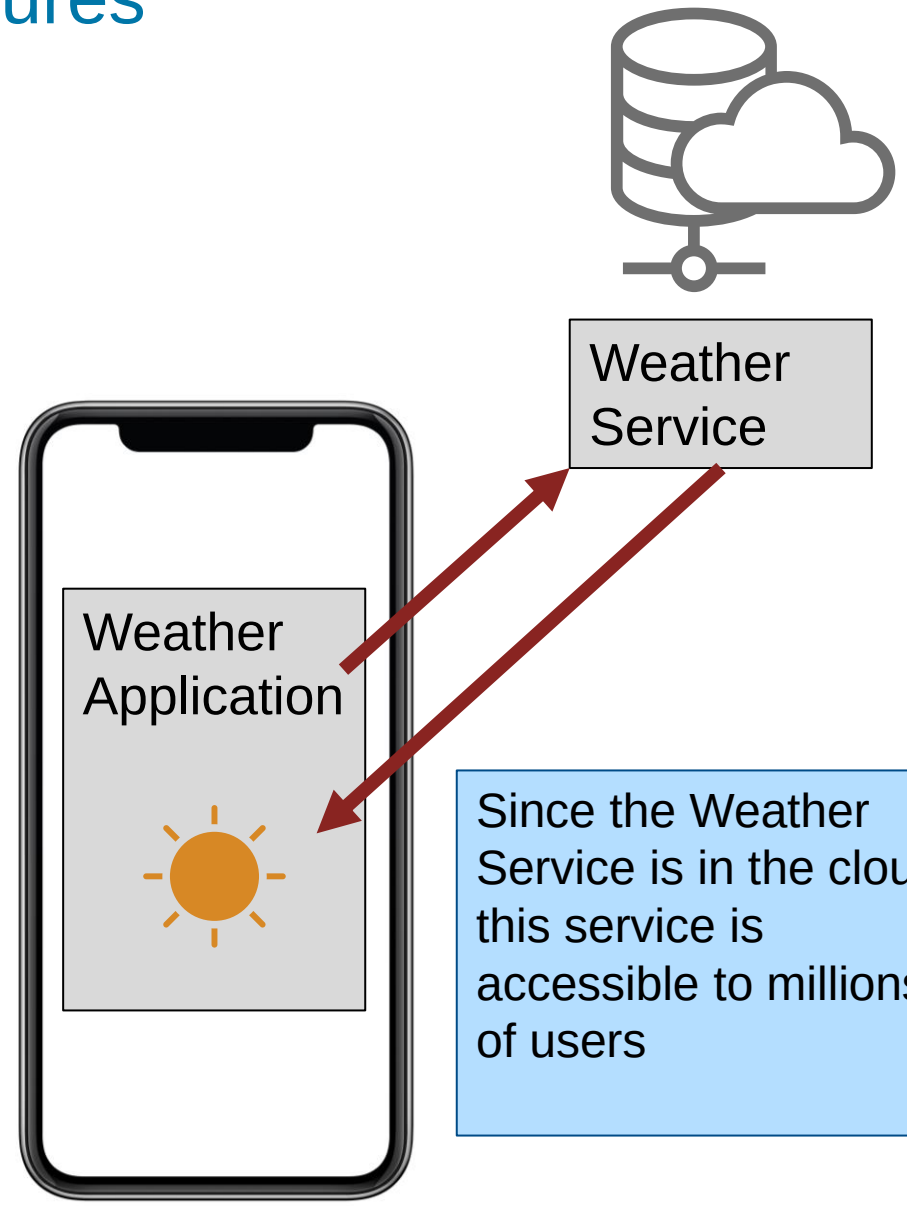
More Details:

Weather Application

Calling out to a function that is not part of the Weather App

Weather Service is providing some function or service

The Service may or may not respond depending on the function or application



Since the Weather Service is in the cloud this service is accessible to millions of users

Key Take Aways

- We have the tools to help move to Service Oriented Architectures (SOA)
- We can generate C++ code for SOA based components
- We are going to show you a lot of information in this presentation
- We are here to help so please engage with us in your SOA type projects

Challenge Statement and Project Benefits

Challenge:

- How to partition an algorithm into Services that allow for reusability, portability and still maintain functionality of the algorithm
- Repartitioning algorithm content into smaller pieces will consist of engineering design choices and requires engineering rigor

We will show:

- Once the partitioning decisions have been made, how to create the Service Oriented Architecture framework
- How to create interfaces to Service Components
- How to create Simulink behavioral models
- how to create agnostic C++ code
- how to apply Adaptive AUTOSAR

Previous projects that helped defined SOA (Service Oriented Architecture) with Models

We will build from these previous projects to show how to migrate a Monolithic Model to SOA

[Technical Article: Migrating Traditional Automotive Applications to SOA]

[Applying Model Based Design to SDV Development]

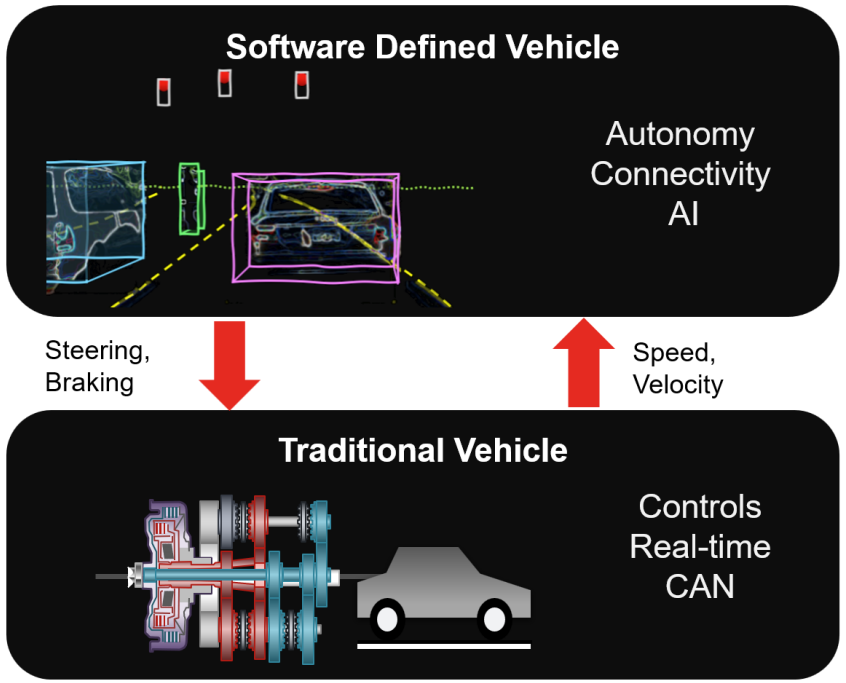
Trend: Automakers are embracing Software Oriented Architectures and Standards

What makes SOA so attractive to Automakers ??

By creating Service Components:

- Reusable Services that may be shared with many applications or components
- Potential for relocation of function
- Higher level of testability
- Potential for reduction in validation due to sharing of functions across application or components

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



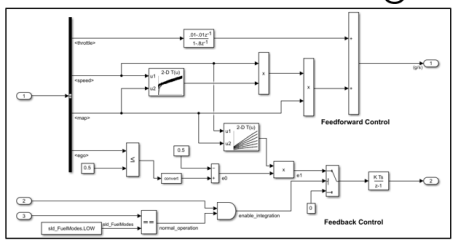
SOA based standards

AUTOSAR Adaptive Platform

ROS

DDS

Model-Based Design



Process of Decomposition to Composition

This is the process of pulling your legacy model apart …

and building in service components with service interfaces…

to create an SOA framework

How to decompose traditional application software compositions into services for Software Defined Vehicles applications?

Reference to Potential Guiding Principles

8.5.2.1 Single-responsibility Principle

The single-responsibility principle (SRP,SWEBOK3) [7] states that a component or class should be responsible for a single part of the overall functionality provided by the software. That responsibility should be encapsulate
provided by
interface(s))
responsibilit

The single-r
reasons (i.e
require a ch
impact on cl
more mainta

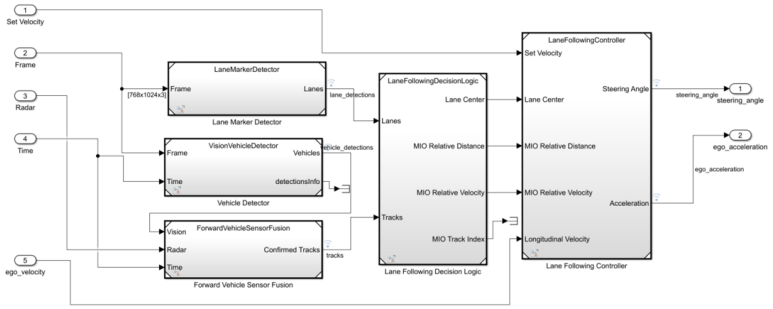
We will
working
partition

Requires Engineering Decisions and Choices

We will show you the Engineering Decisions that MathWorks made in order to separate out Service Components

Decompose the Monolithic Application

Some techniques can be found under Section 8.3 Decomposition Strategy and Section 8.5 Design Principles in the standard document - *Explanation of Adaptive Platform Software Architecture R22-11*



Few of the notable techniques from design principles

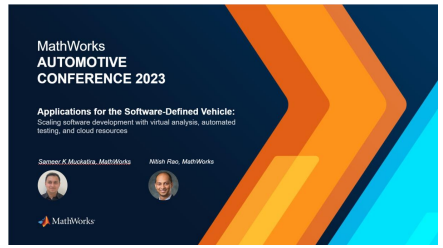
1. Single-responsibility Principle
2. Interface Segregation Principle
3. Dependency Inversion Principle
4. Acyclic Dependencies Principle

A Highway Lane Following application designed previously as a monolithic application composition can be converted to a single service or it can be decomposed into multiple services like camera service, vision service, radar service, lane detection service etc.

AUTOSAR MathWorks

Explanation of Adaptive Platform Software Architecture (autosar.org)

The Models that we are going to work on come from the second presentation that I referenced

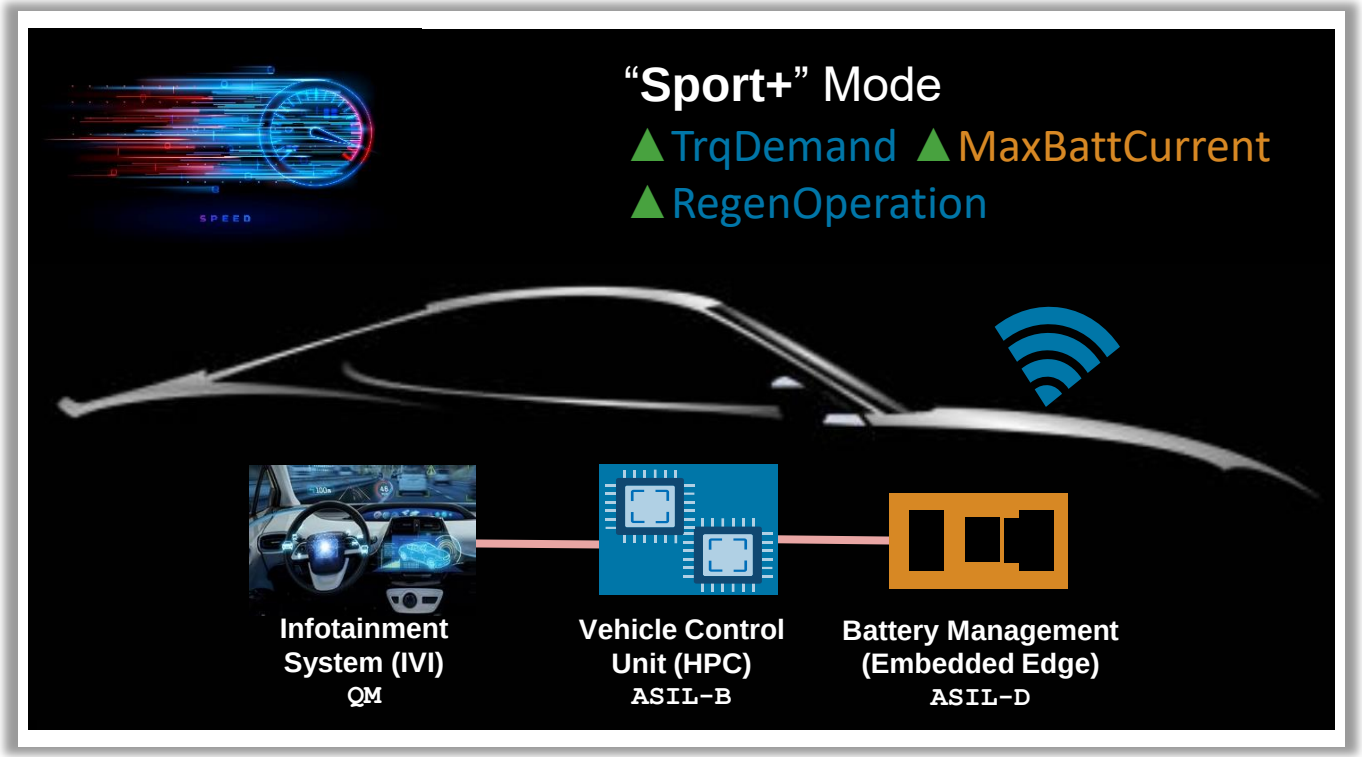


This project had models for two different ECUs working together

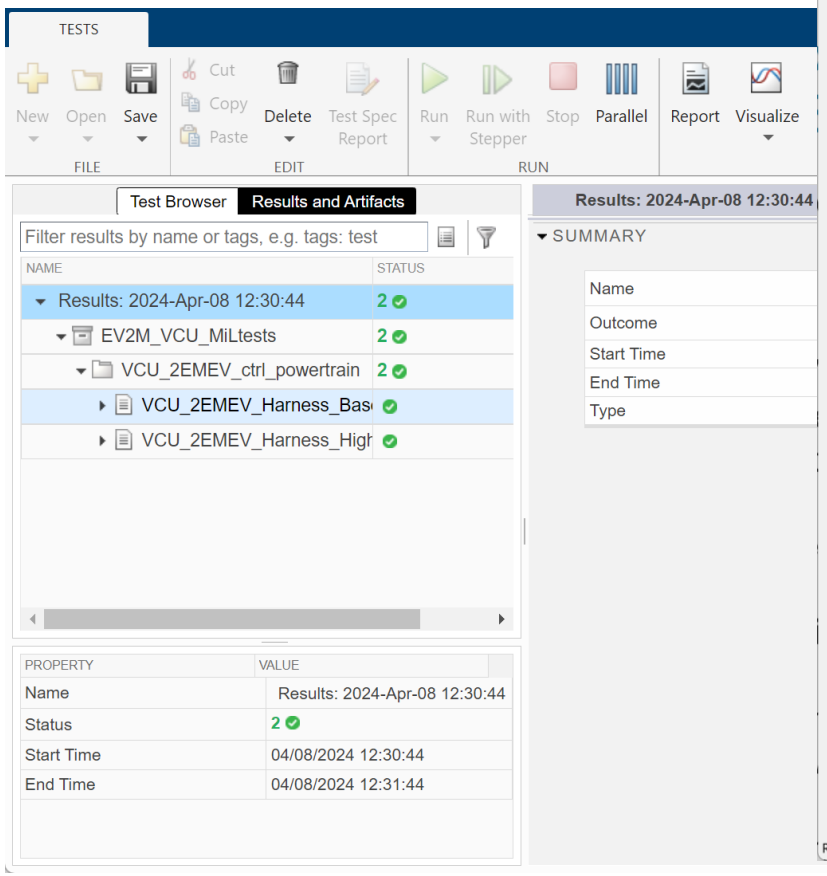
Battery Management – which consisted of AUTOSAR Classic components for Battery State Health and State of Charge

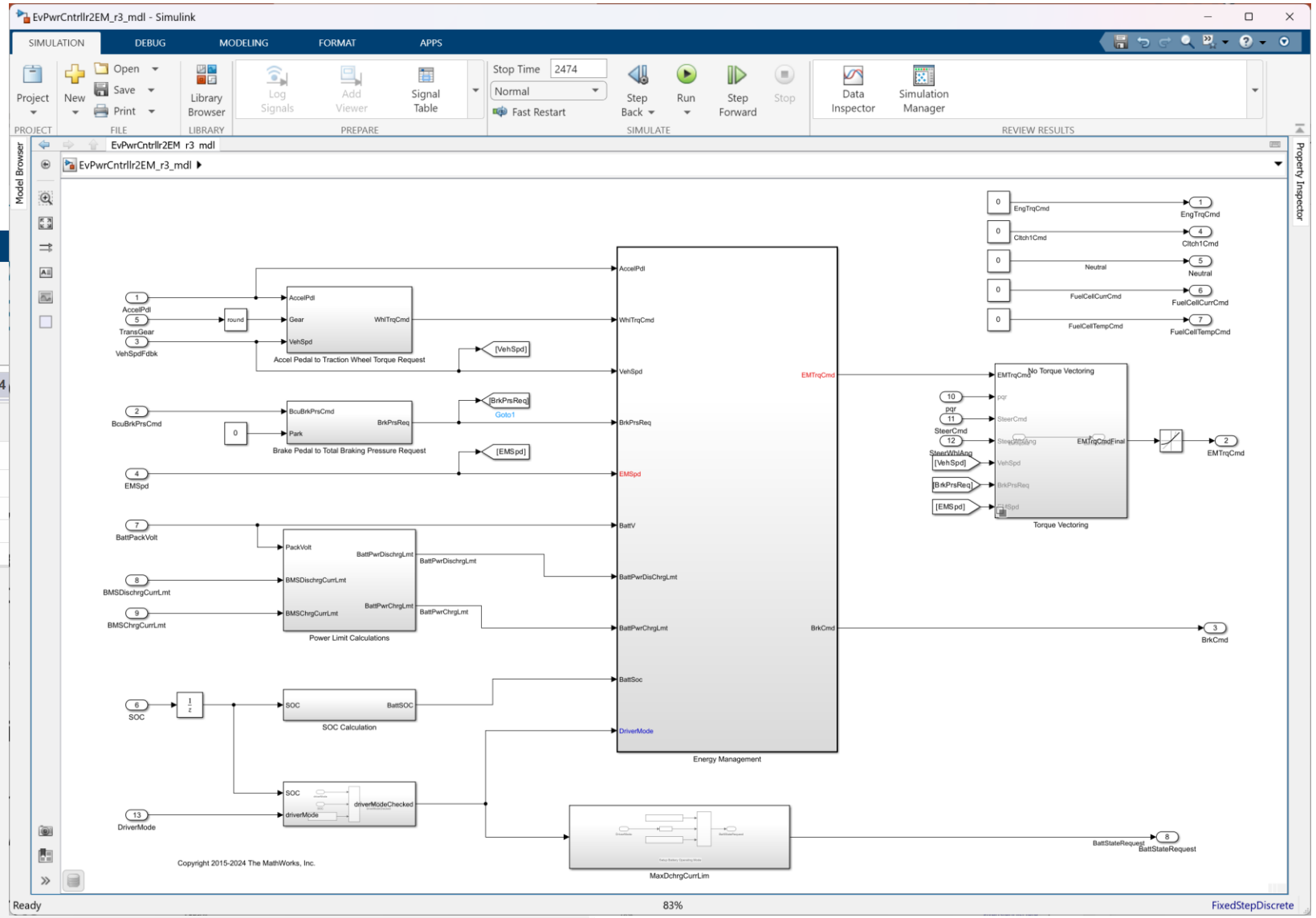
Vehicle Control Unit (HPC) – Which consisted of algorithms that are slated for an Adaptive Platform

We are going to use the models for the Vehicle Control Unit (HPC) for this demonstration



Start with a Baseline Model with tests that pass





Based on Model Functions decide what will be the main app and what will be Services

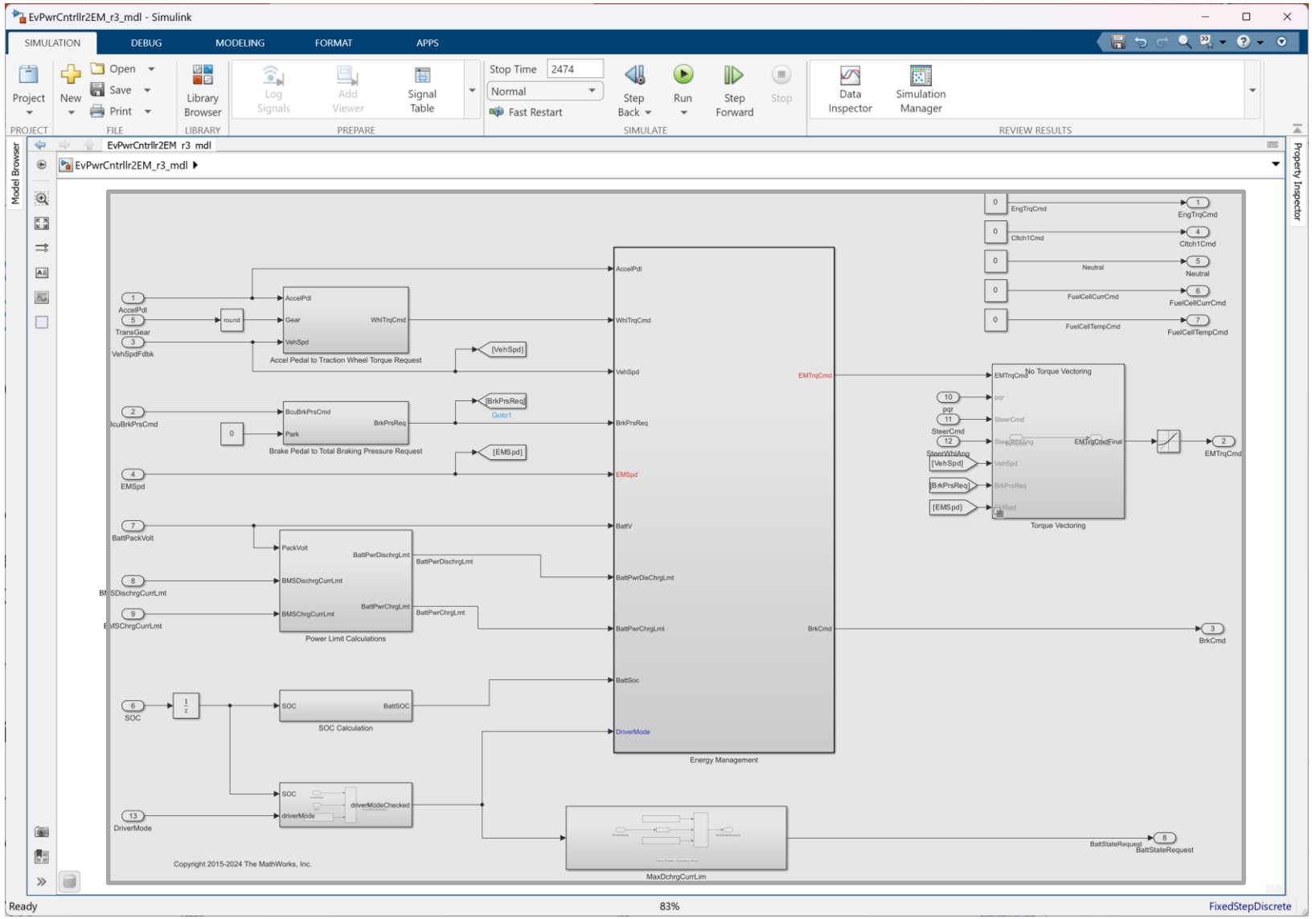
MathWorks Design Decisions

<u>Identify and Analyze Services</u>

Main app will retain full functionality

Our end model will look like the grey box in the model

However, the end refactored model will call out to services in place of some of the inline functionality



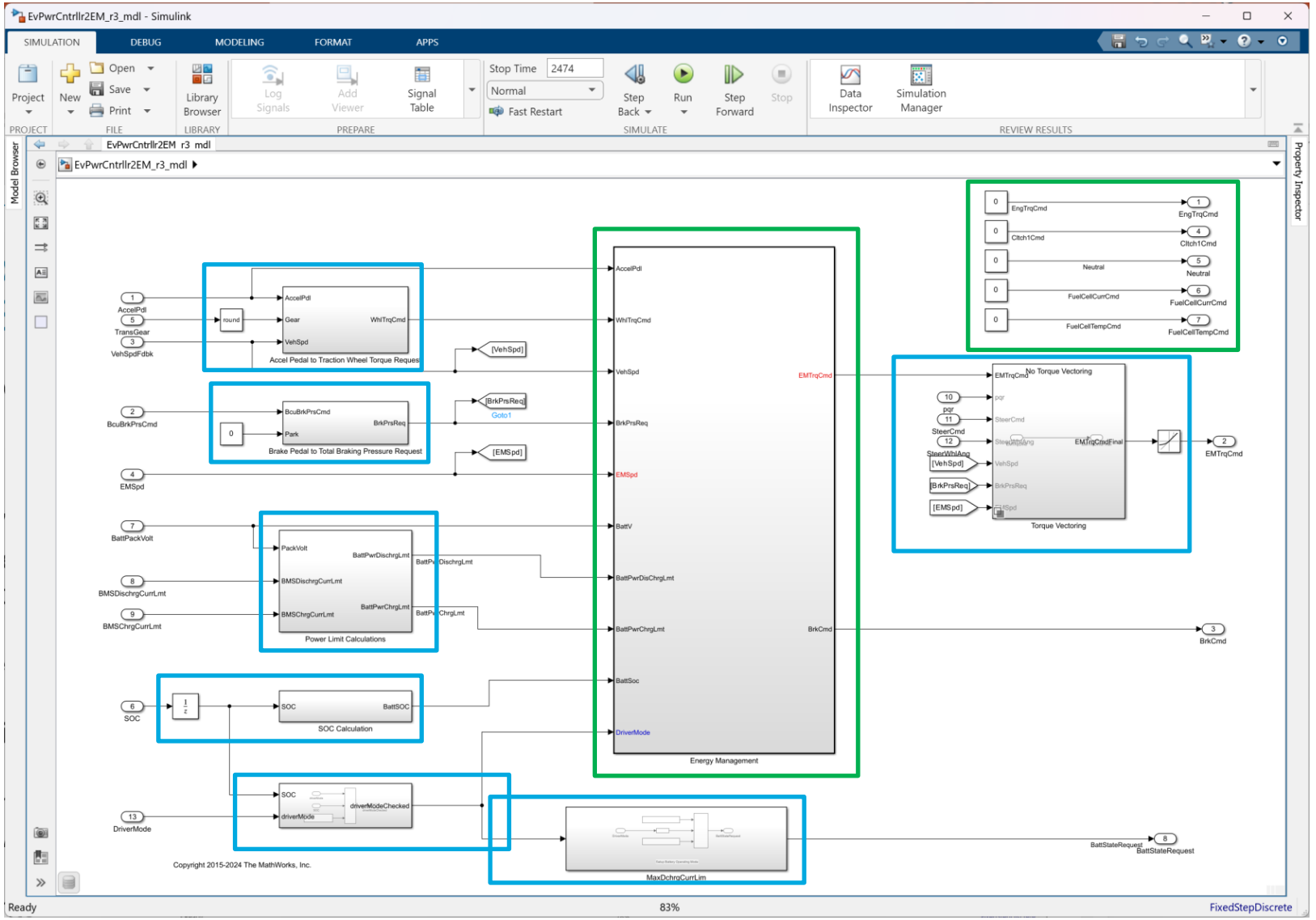
Based on Model Functions decide what will be the main app and what will be Services

MathWorks Design Decisions

<u>Identify and Analyze Services</u>
For this example, we are going to refactor all of the functions that are in blue boxes into service components

In a staged approach, goal is to make a few changes, and verify with tests that no functionality was lost.

Use System Composer to refactor our design to include service components

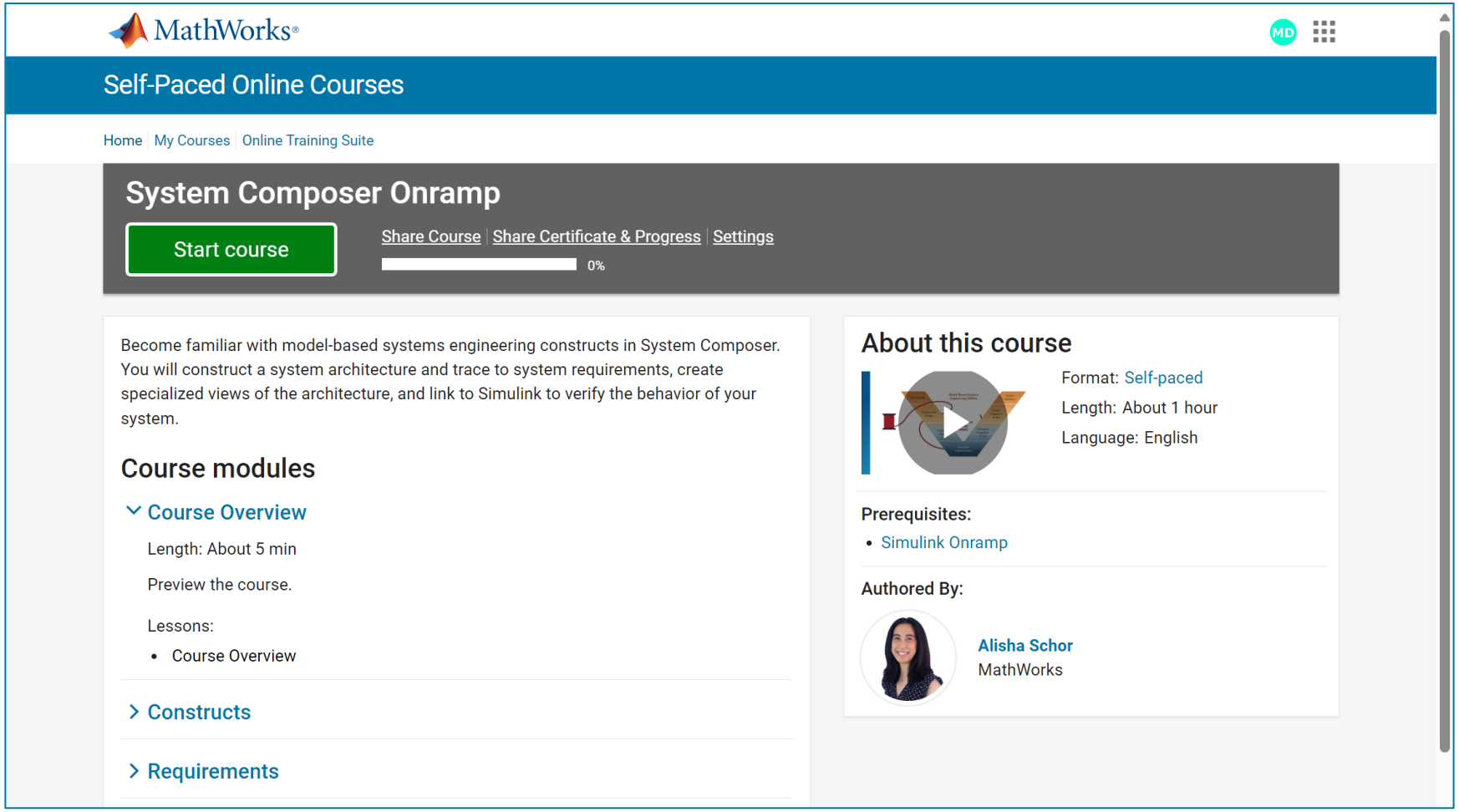


Identify and Analyze Services → Define Services and its interfaces → Define Service Contracts → Implement and deploy Services

New System Composer Onramp Training

Newly Release
March 2024 w/ R2024a

System Composer Onramp is free self paced, online training that will introduce you to System Composer features



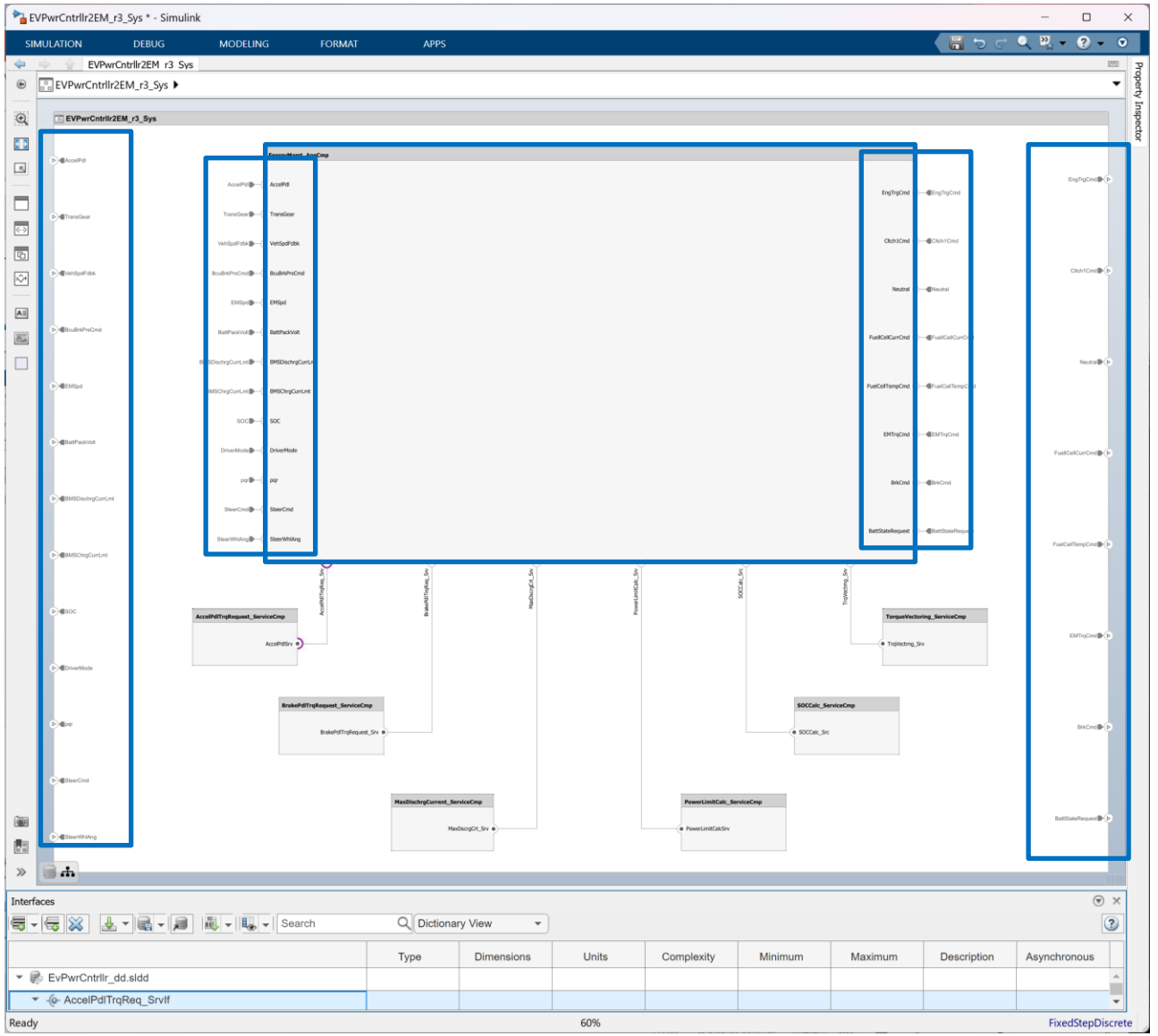
System Composer Onramp

Partition Algorithm Model Into Smaller parts with Service Interface

Define Services and Interfaces

In System Composer:

1) Created software architecture model
2) Create a software component box for our main application
3) Define all I/O in the main application and connect to Interface boundary of composition



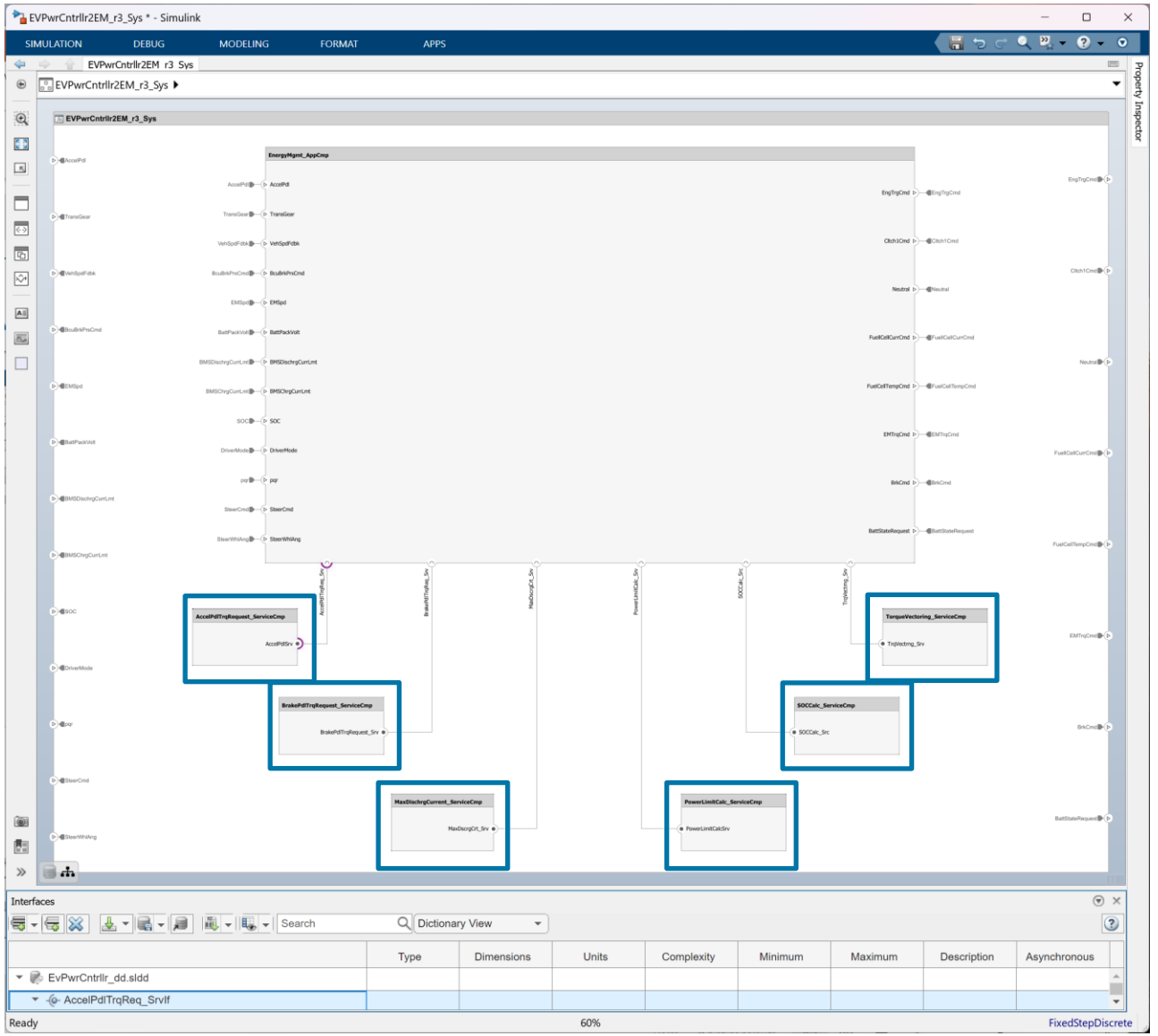
Identify and Analyze Services → **Define Services and its interfaces** → Define Service Contracts → Implement and deploy Services

Partition Algorithm Model Into Smaller parts with Service Interface

Define Services and Interfaces

In System Composer:

1) Created software architecture model
2) Create a software component box for our main application
3) Define all I/O in the main application and connect to Interface boundary of composition
4) Create a software component boxes for all new service components



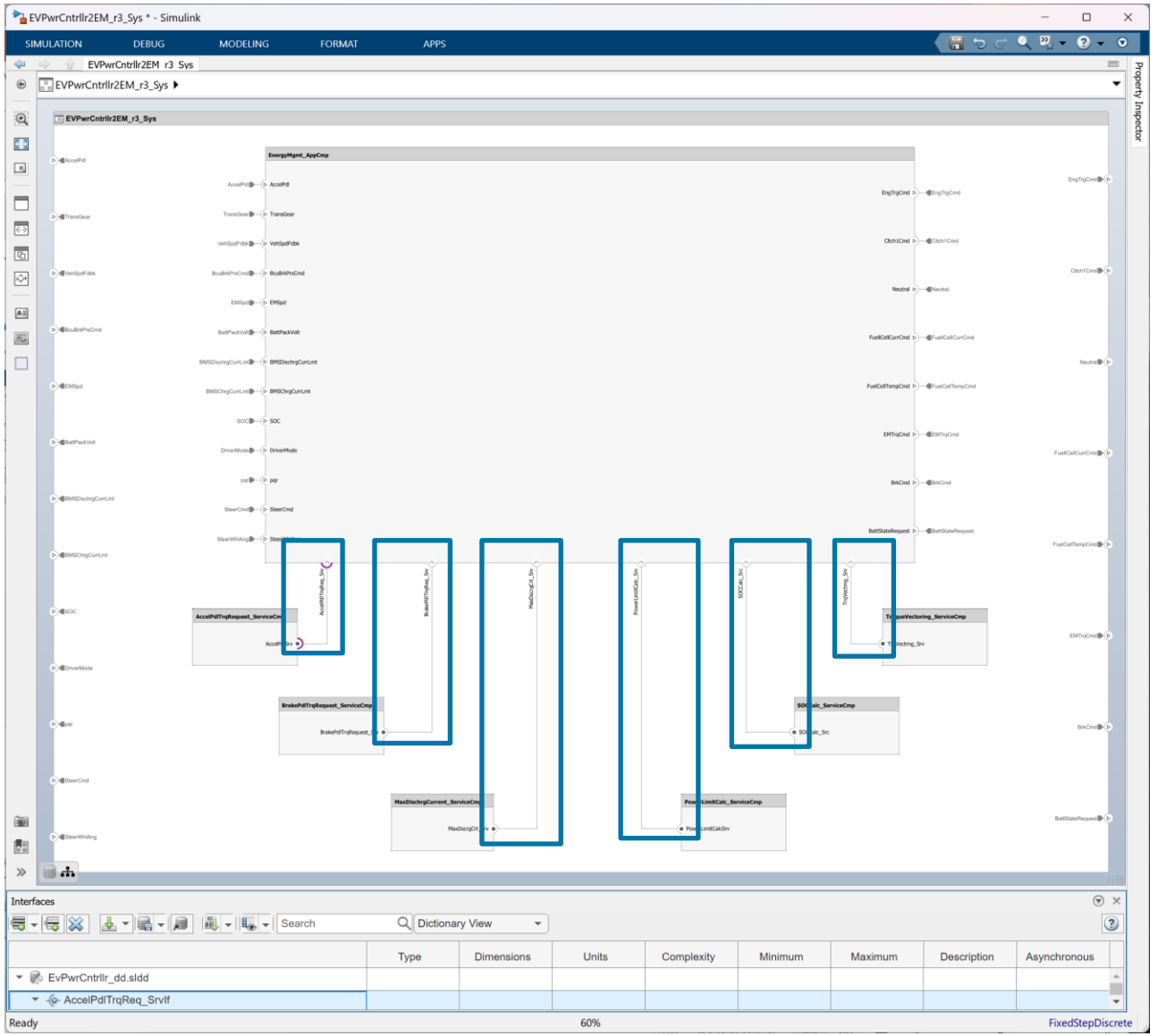
Identify and Analyze Services → **Define Services and its interfaces** → Define Service Contracts → Implement and deploy Services

Partition Algorithm Model Into Smaller parts with Service Interface

Define Services and Interfaces

In System Composer:

1) Created software architecture model
2) Create a software component box for our main application
3) Define all I/O in the main application and connect to Interface boundary of composition
4) Create a software component boxes for all new service components
5) Connect all service components to main application with client server connectors



Identify and Analyze Services → Define Services and its interfaces → Define Service Contracts → Implement and deploy Services

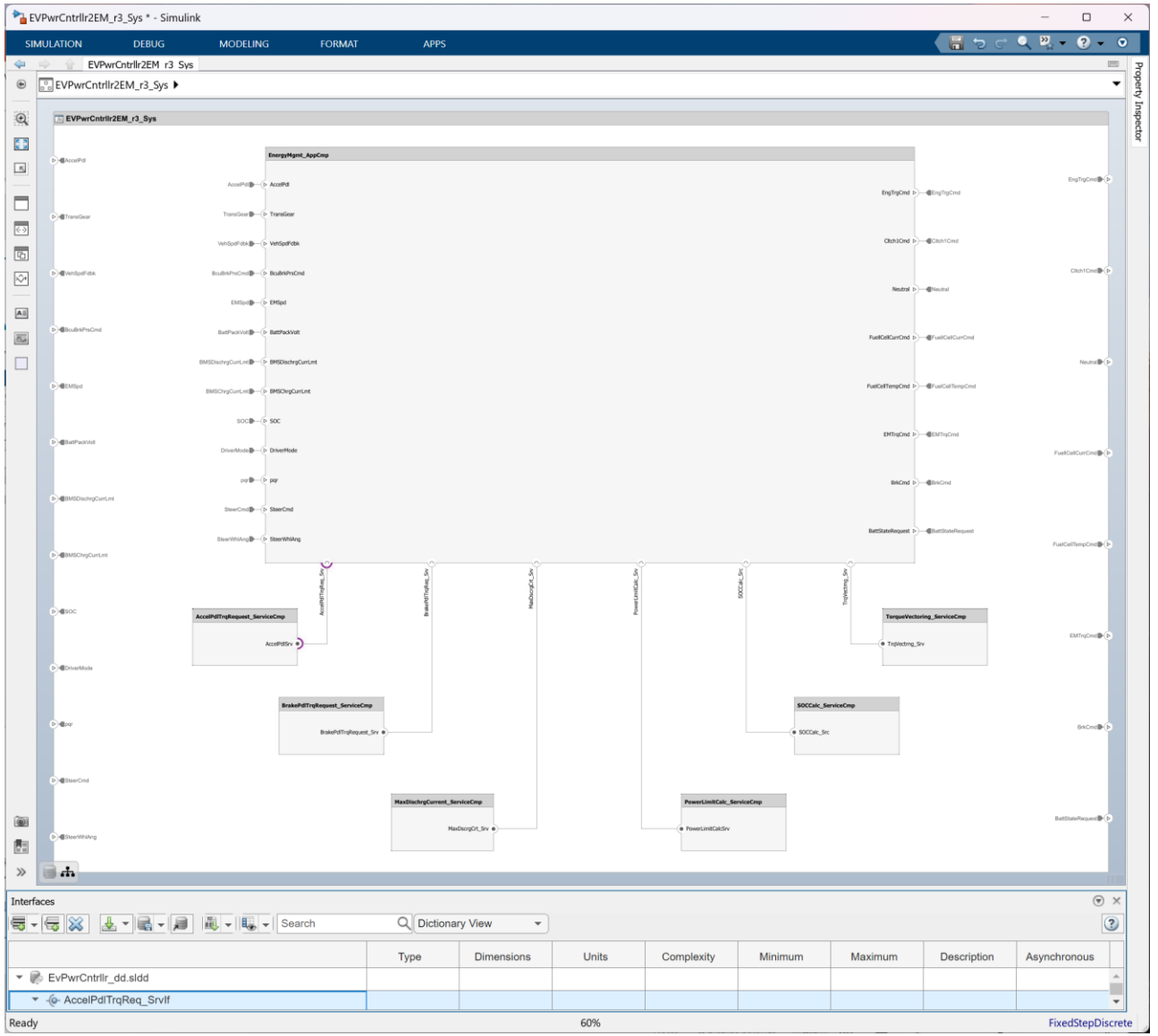
Partition Algorithm Model Into Smaller parts with Service Interface

Define Services and Interfaces

In System Composer:

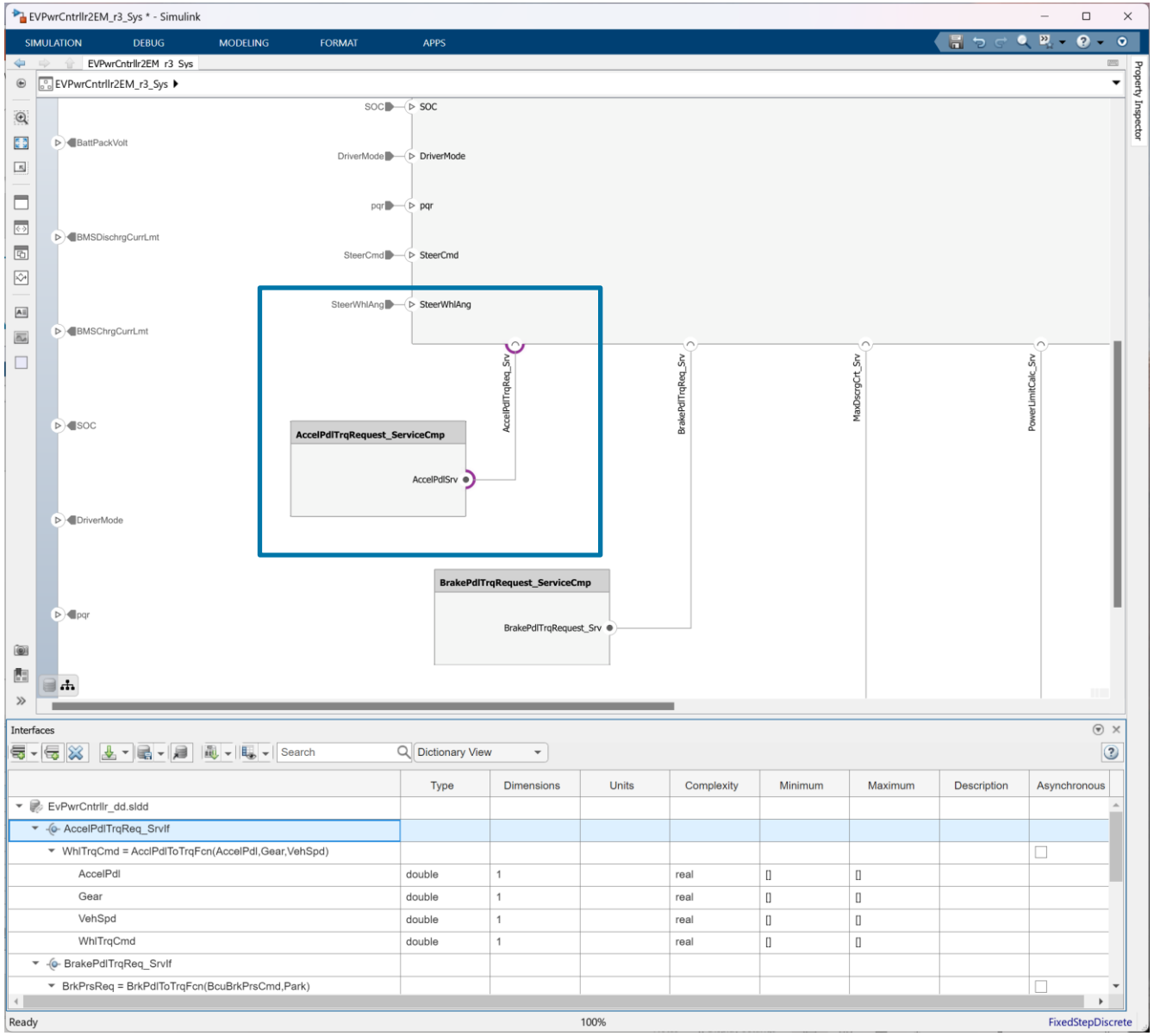
1) Created software architecture model
2) Create a software component box for our main application
3) Define all I/O in the main application and connect to Interface boundary of composition
4) Create a software component boxes for all new service components
5) Connect all service components to main application with client server connectors

Now we have our Architecture model drawn
Next step is to create Client Server Interfaces



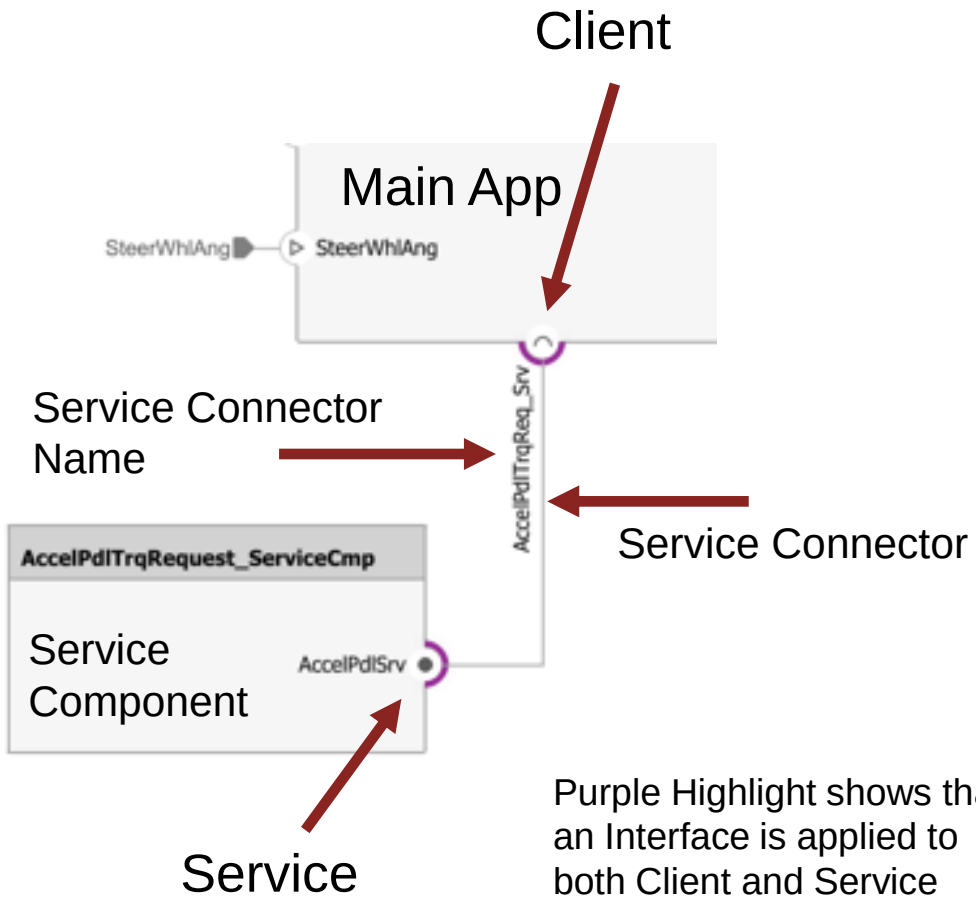
Identify and Analyze Services → Define Services and its interfaces → Define Service Contracts → Implement and deploy Services

Define Client Server Interfaces



Define Services and Interfaces

A Service Component is connected to the Main App through a Service Connector between the Client Port and a Service Port



Purple Highlight shows that an Interface is applied to both Client and Service Ports

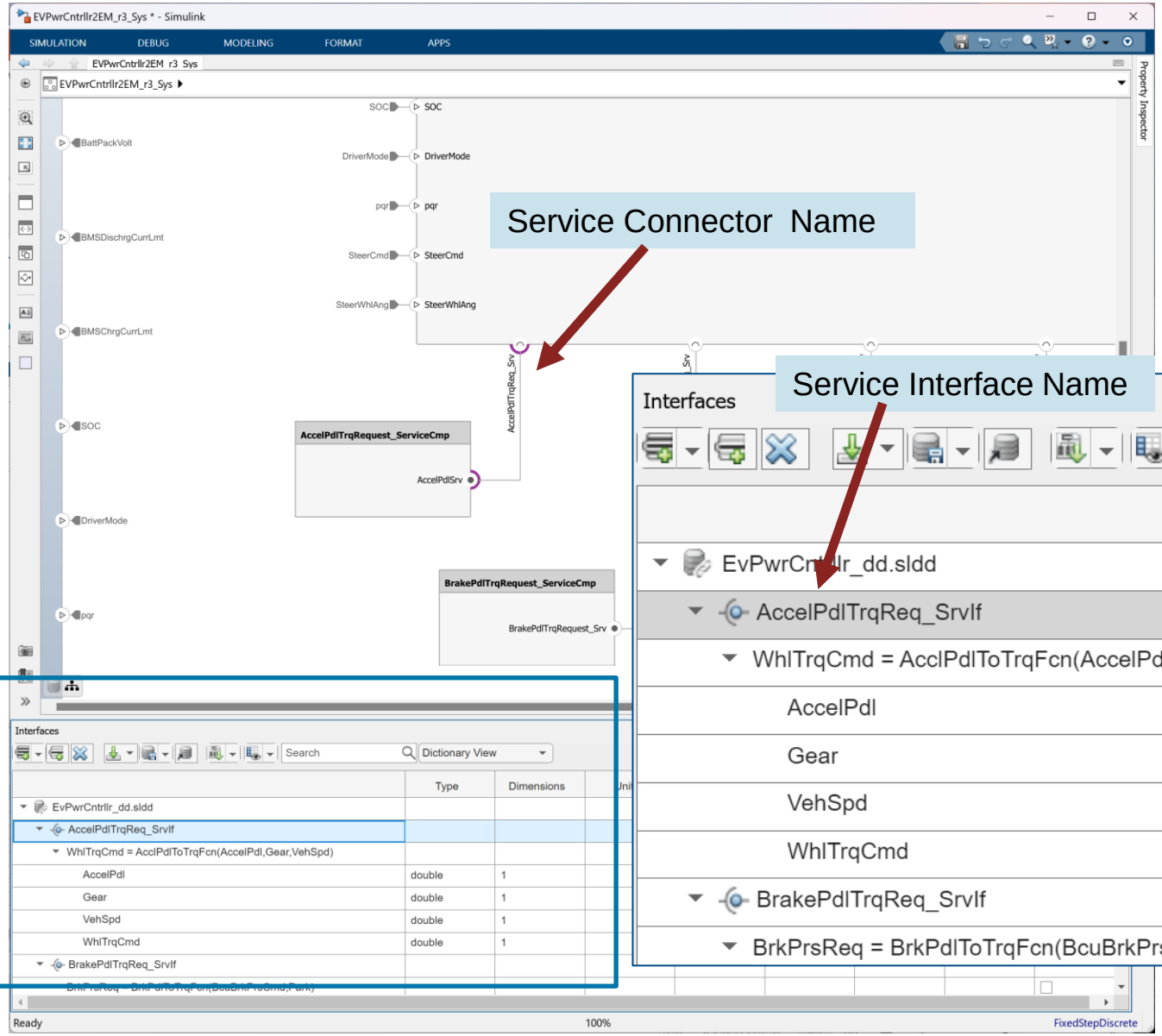
Define Client Server Interfaces

Define Services and Interfaces

Ability to define the full Service-Interface in terms of the function prototypes and arguments

This allows for the Service Interface to be used to create the model constructs for both the main app component and the Service Component

Define the Client / Service Interface for each of the connected Service Components



Service Connector Name

Service Interface Name

Function based Proto-Type shows Function Name and Input / Output Arguments

Define Inputs and Outputs to function in terms of datatypes, dimensions, units, min, max etc;

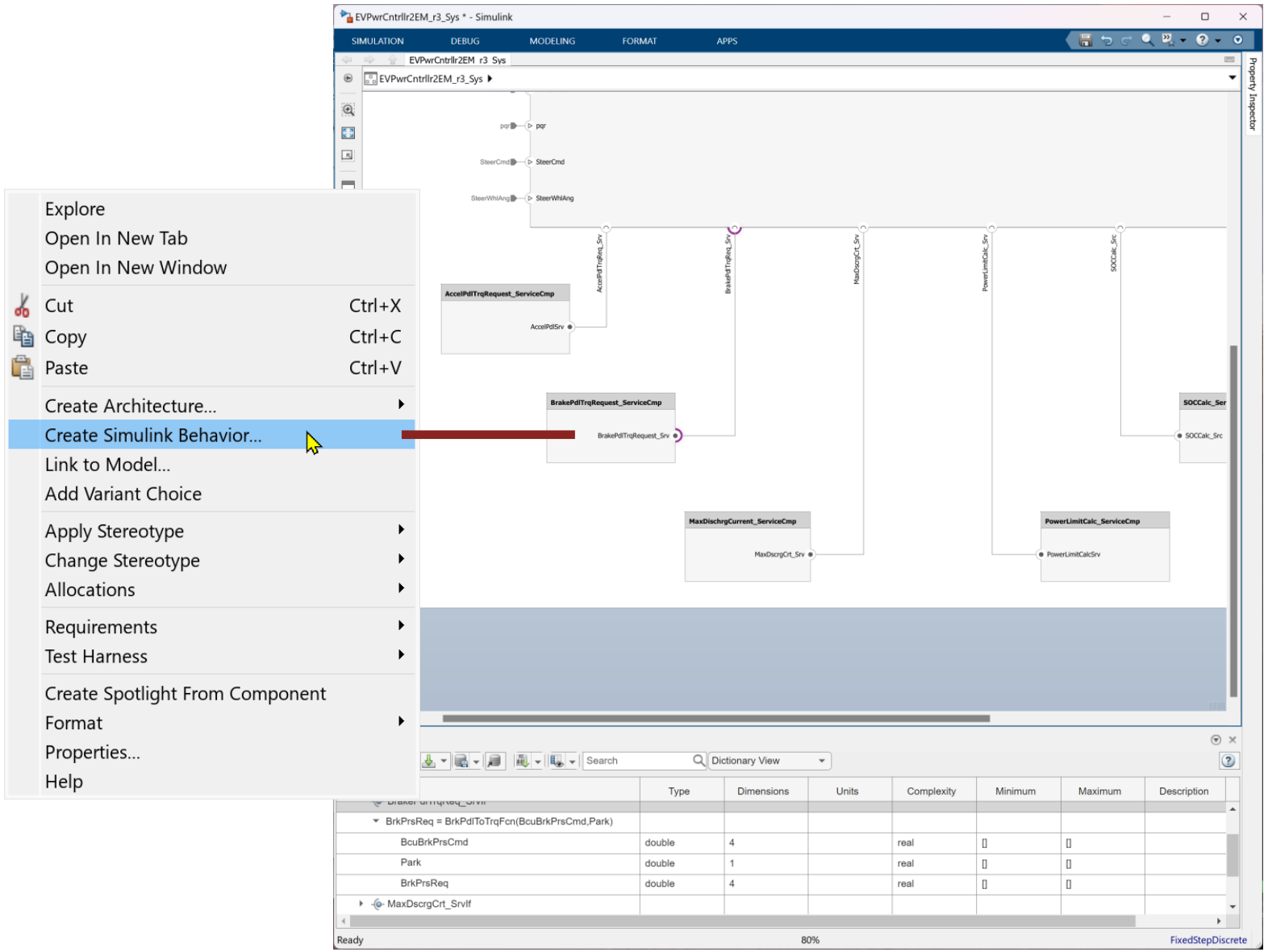
Create Simulink Behavior Model for Service Component

Creating the Simulink Behavior (empty skeleton model that will contain your algorithm)

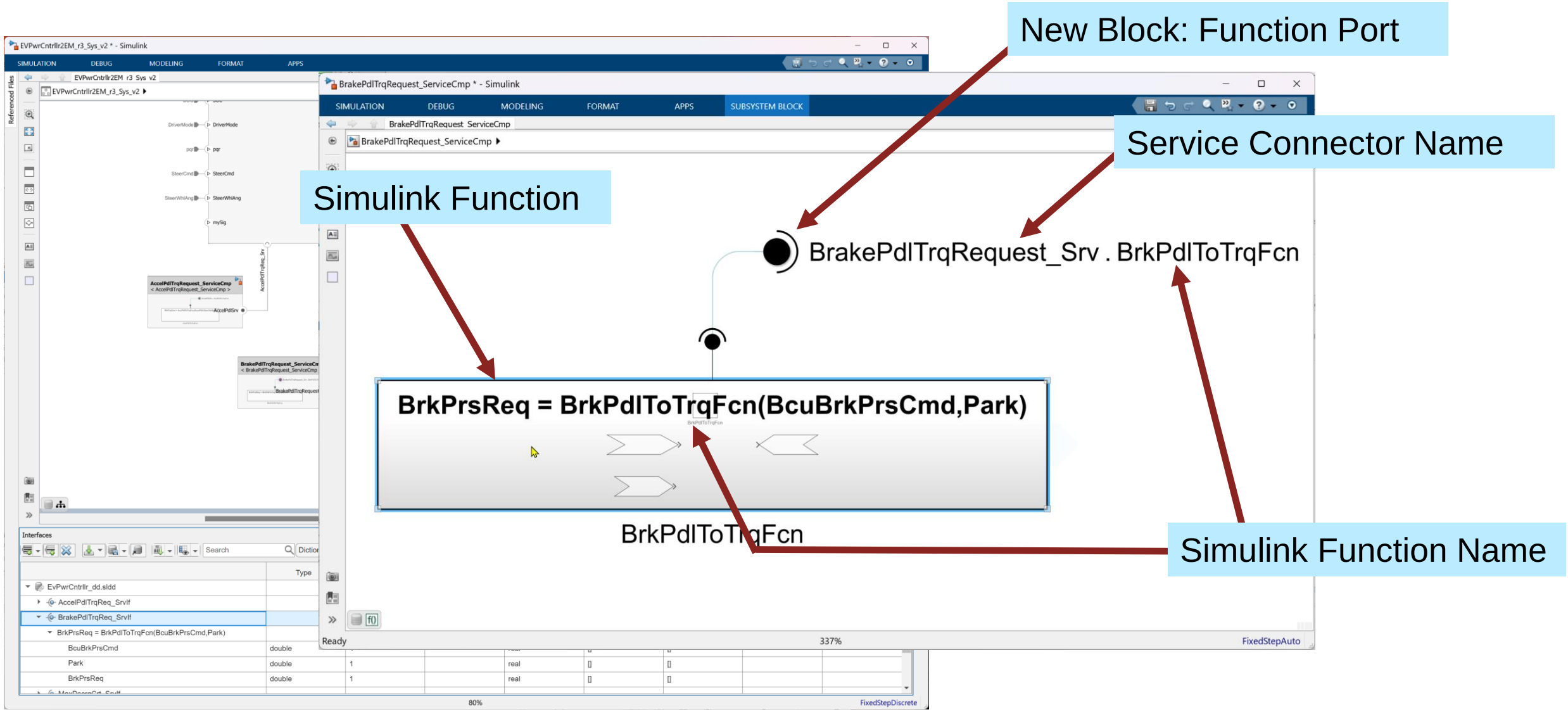
Right click on Component boxes to create the Simulink Behavior

This action creates a skeleton or shell model

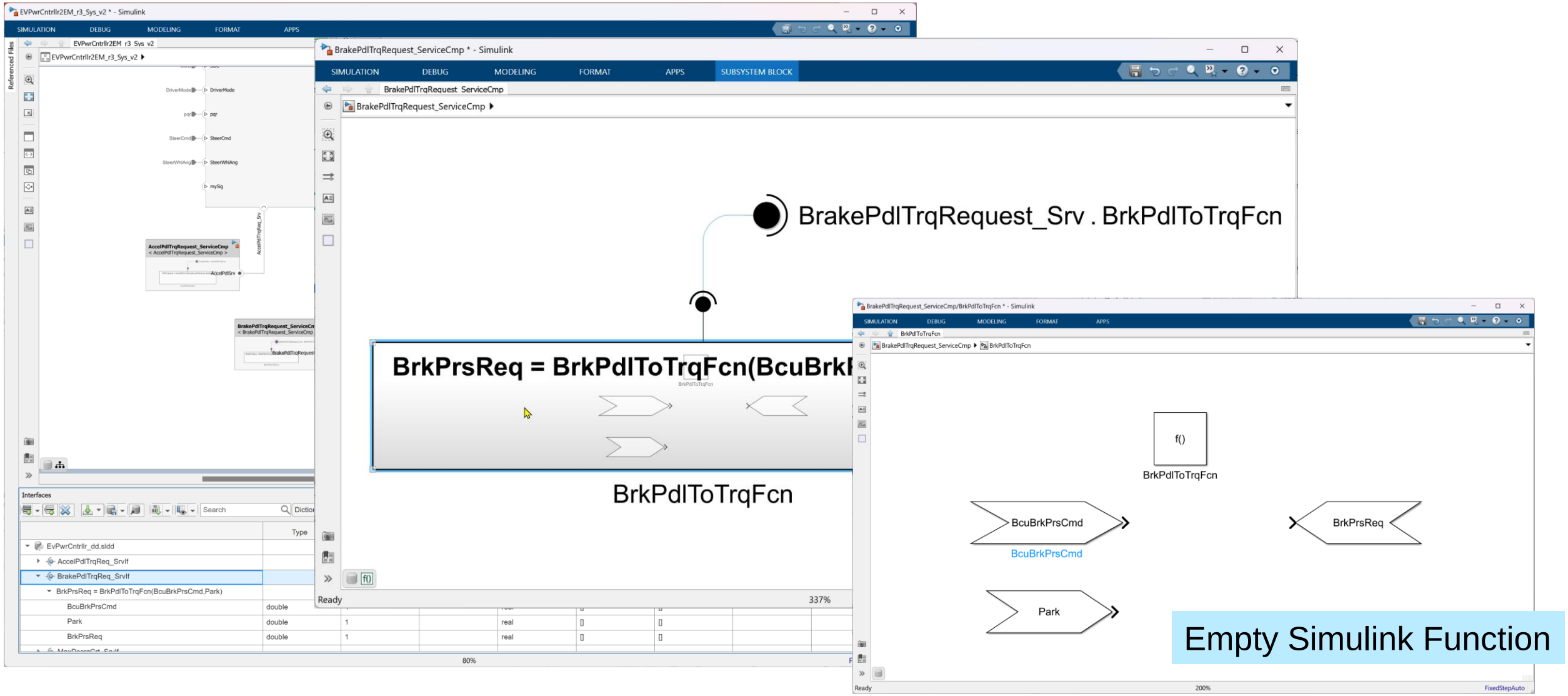
Also, this will create blocks that represent the Client / Service interfaces



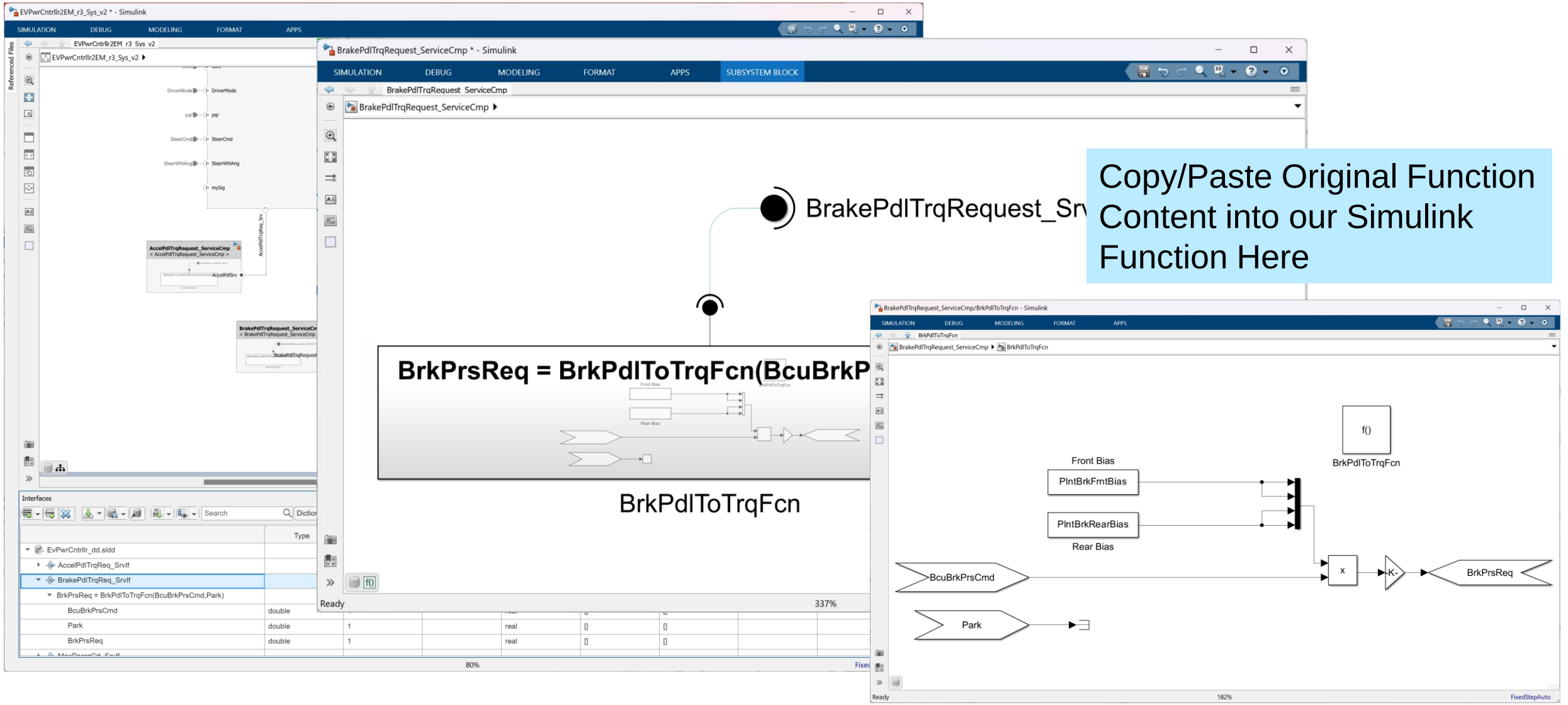
Create Simulink Behavior Model for Service Component



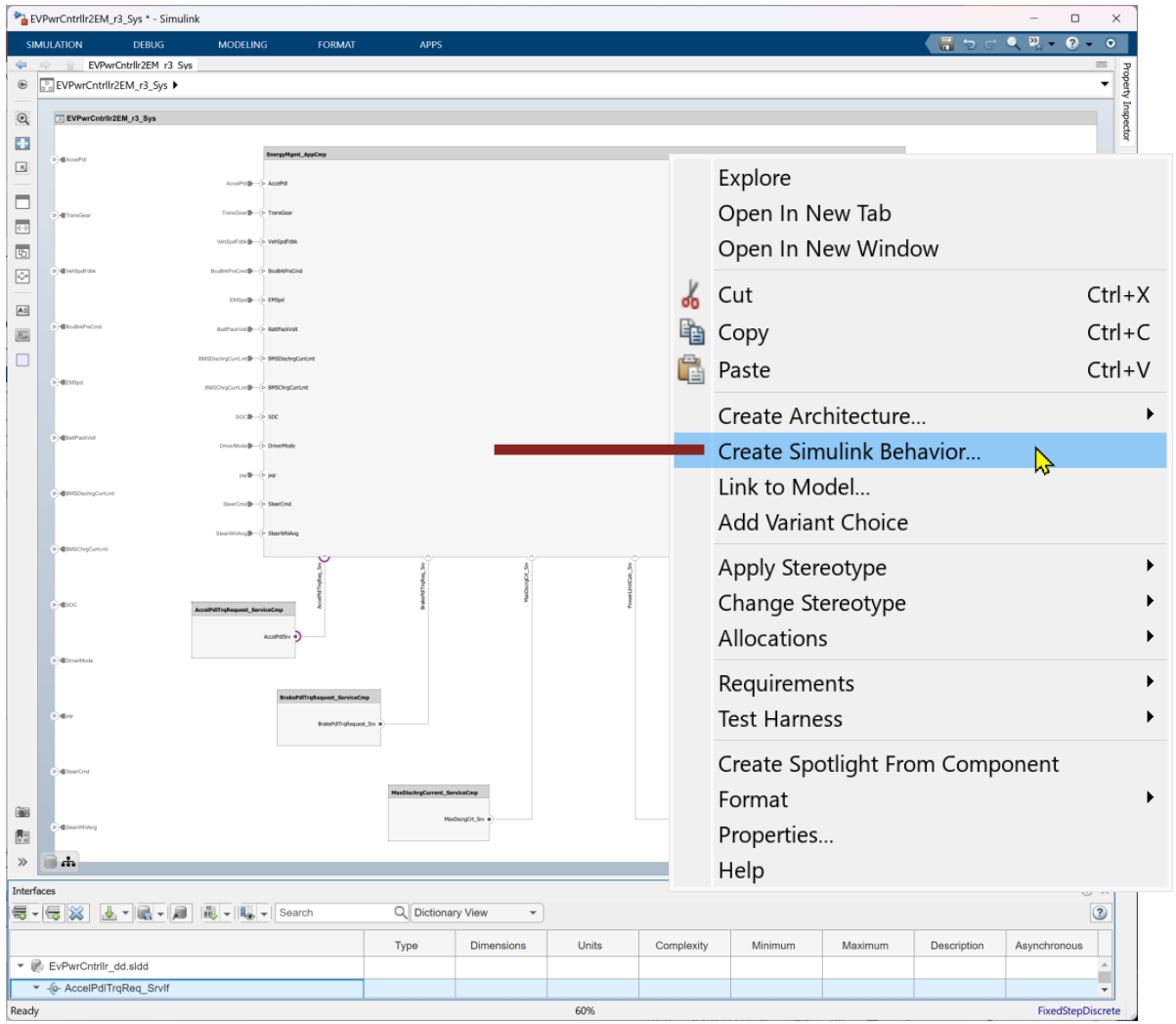
Create Simulink Behavior Model for Service Component



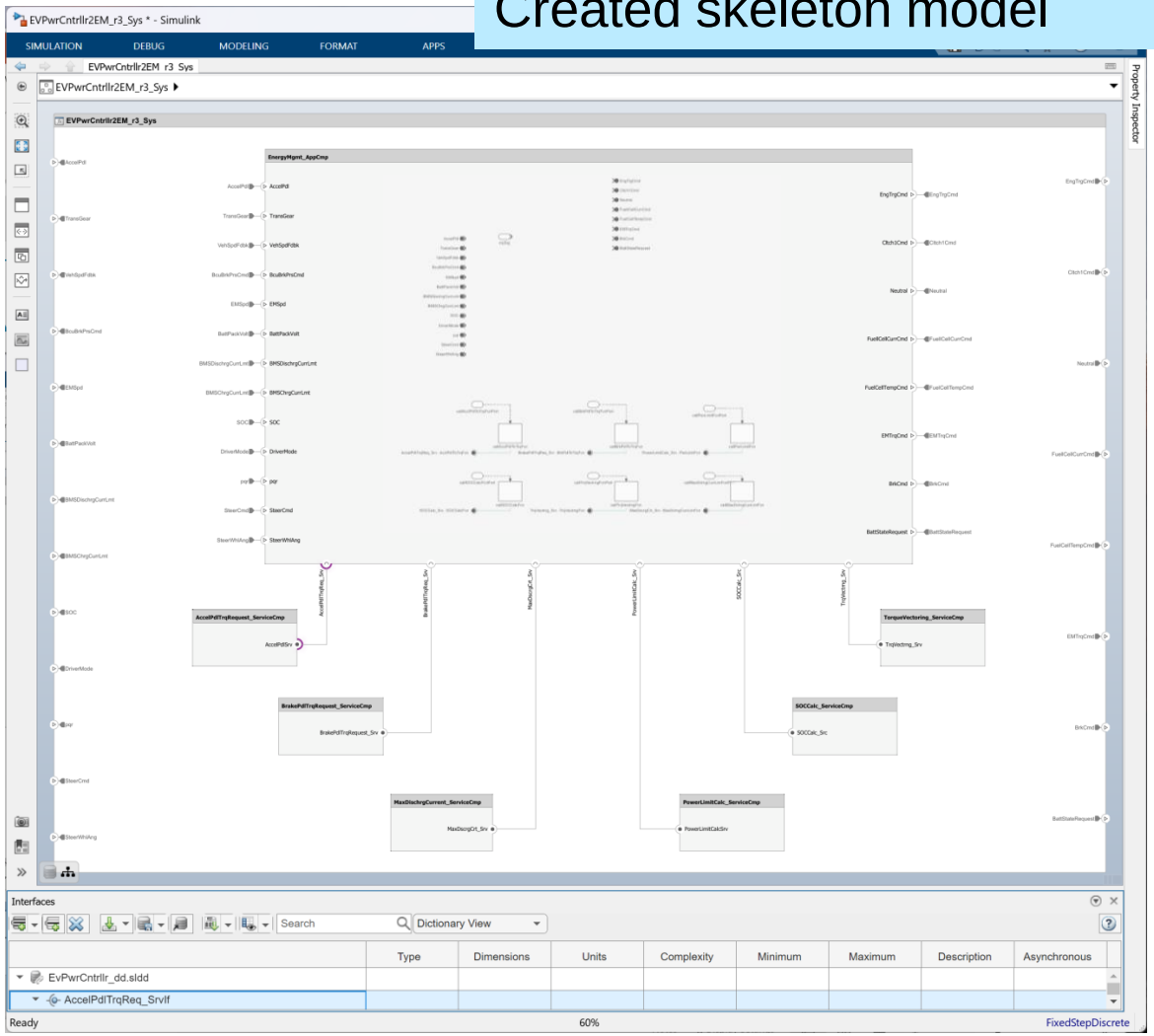
Create Simulink Behavior Model for Service Component



Create Simulink Behavior Model for Main Application Component



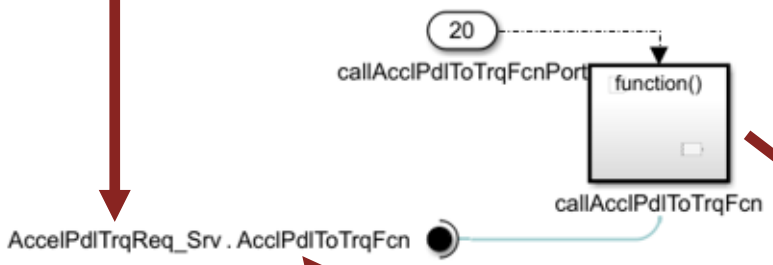
Created skeleton model



Create Simulink Behavior Model for Main Application Component

Service Connector Name

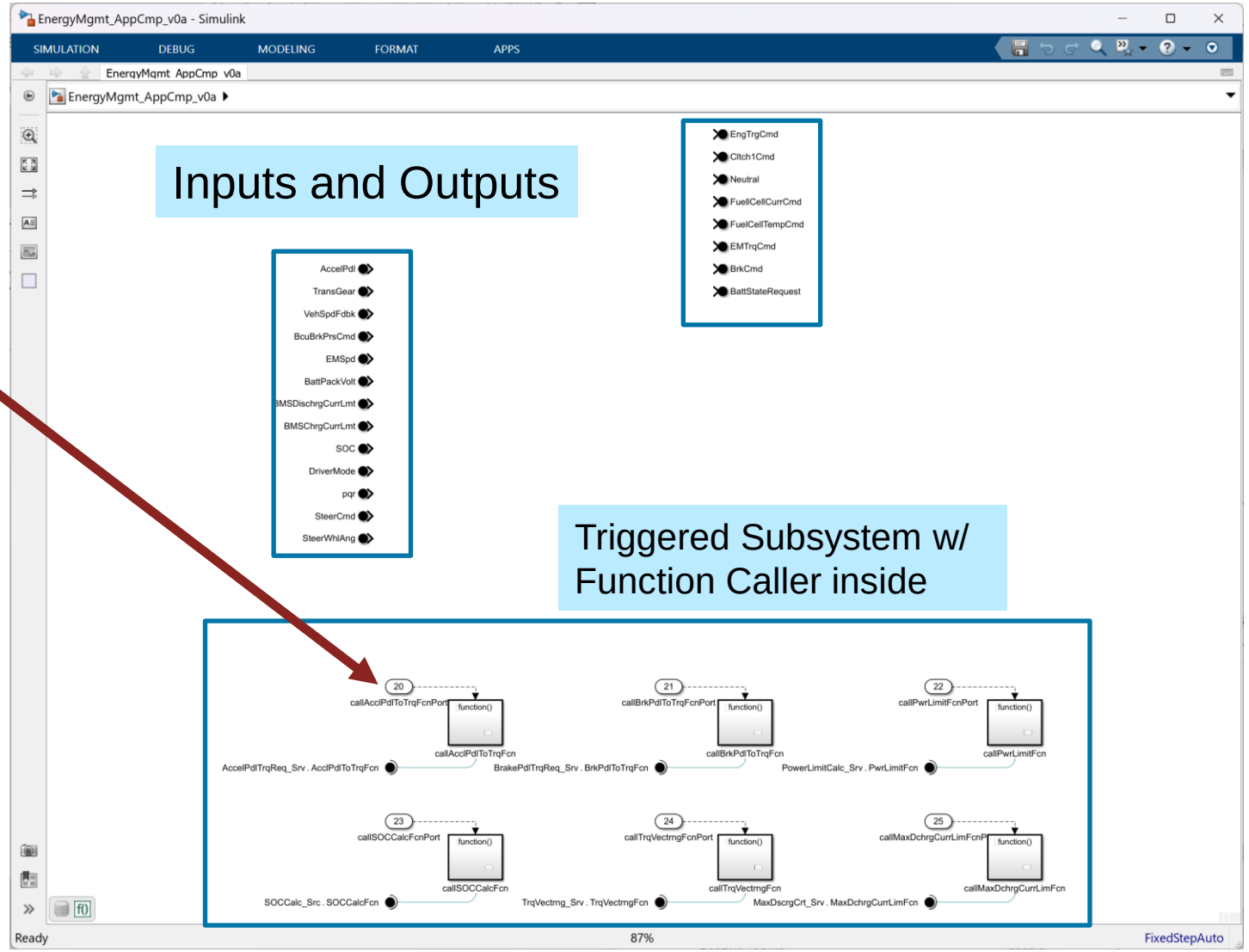
Triggered Subsystem



Function Port

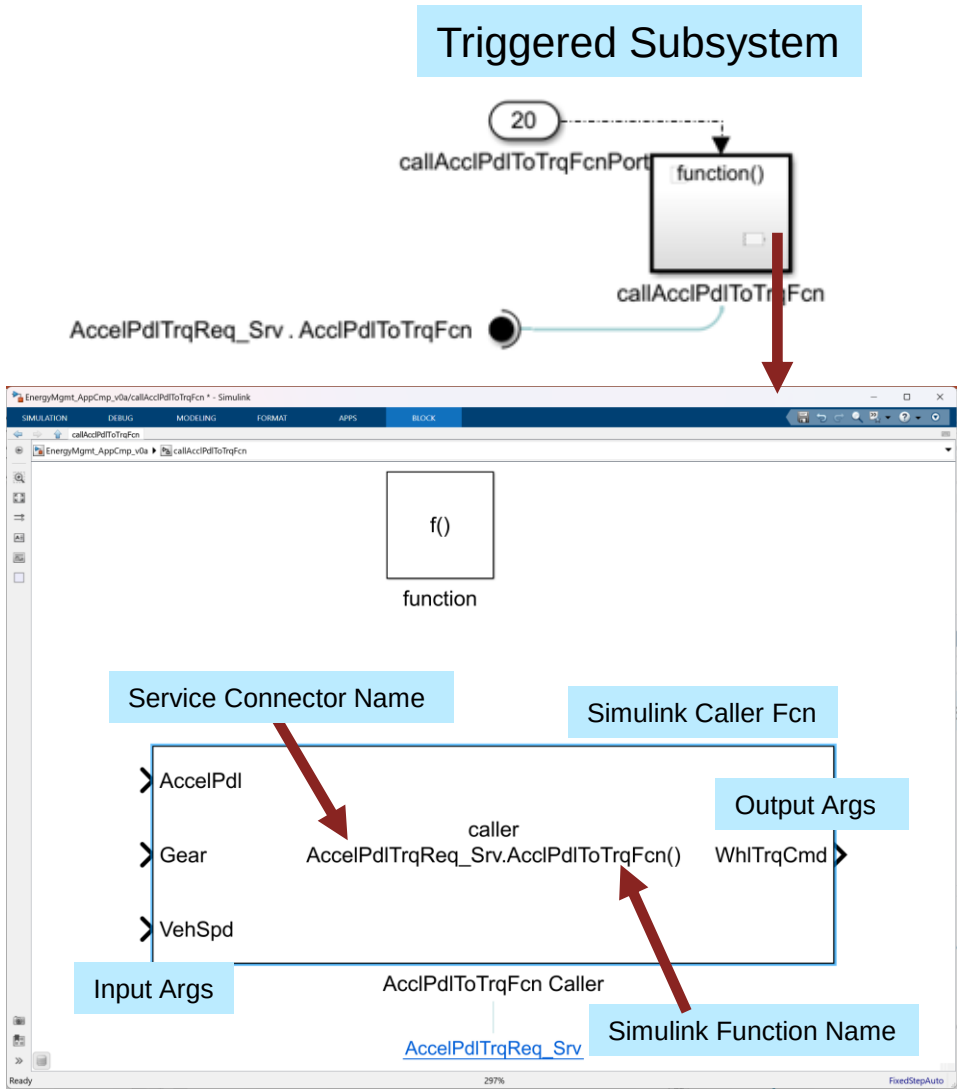
Simulink Function Name

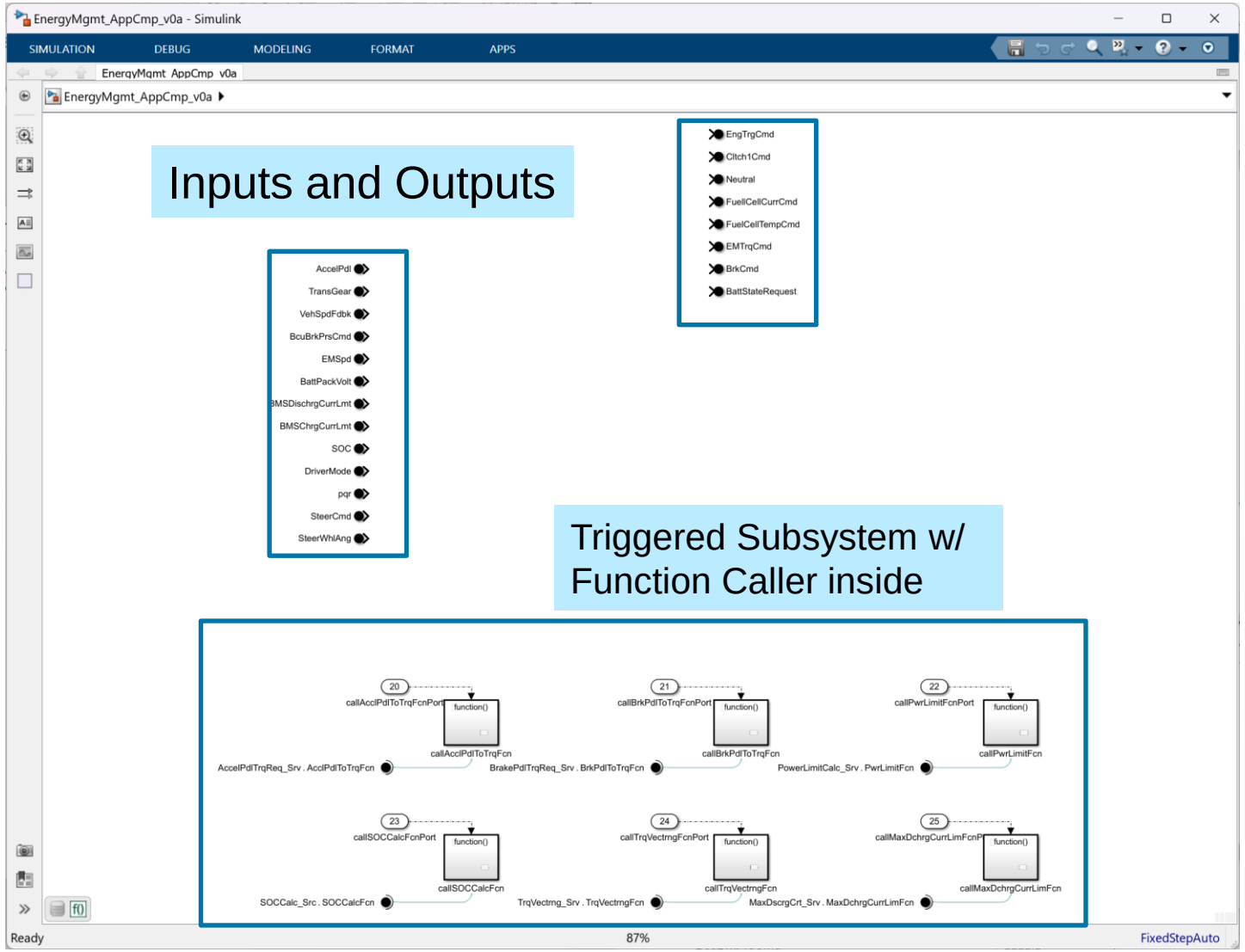
Inputs and Outputs



Triggered Subsystem w/ Function Caller inside

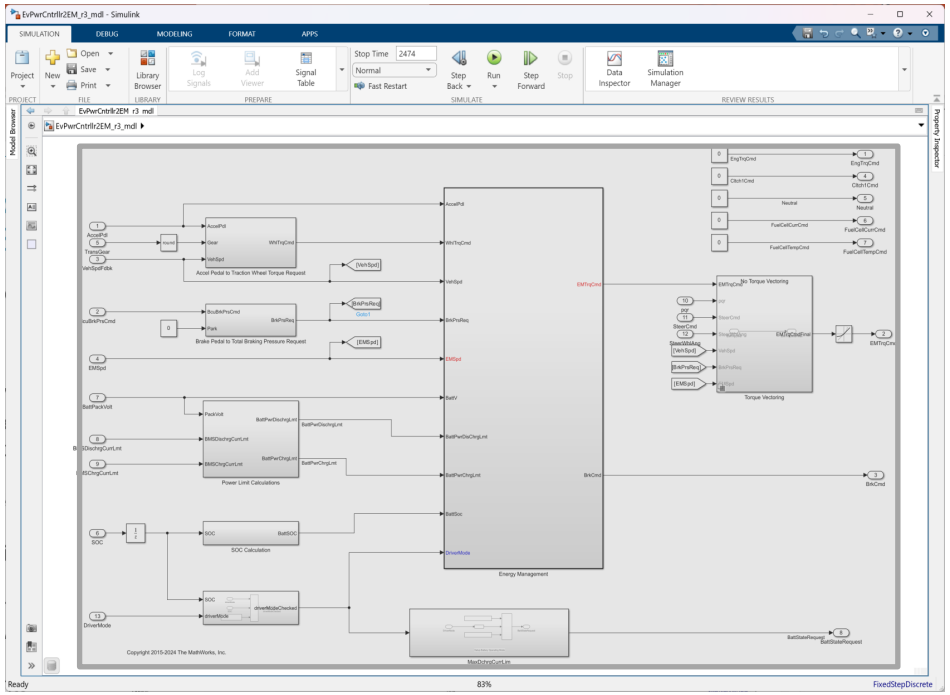
Create Simulink Behavior Model for Main Application Component



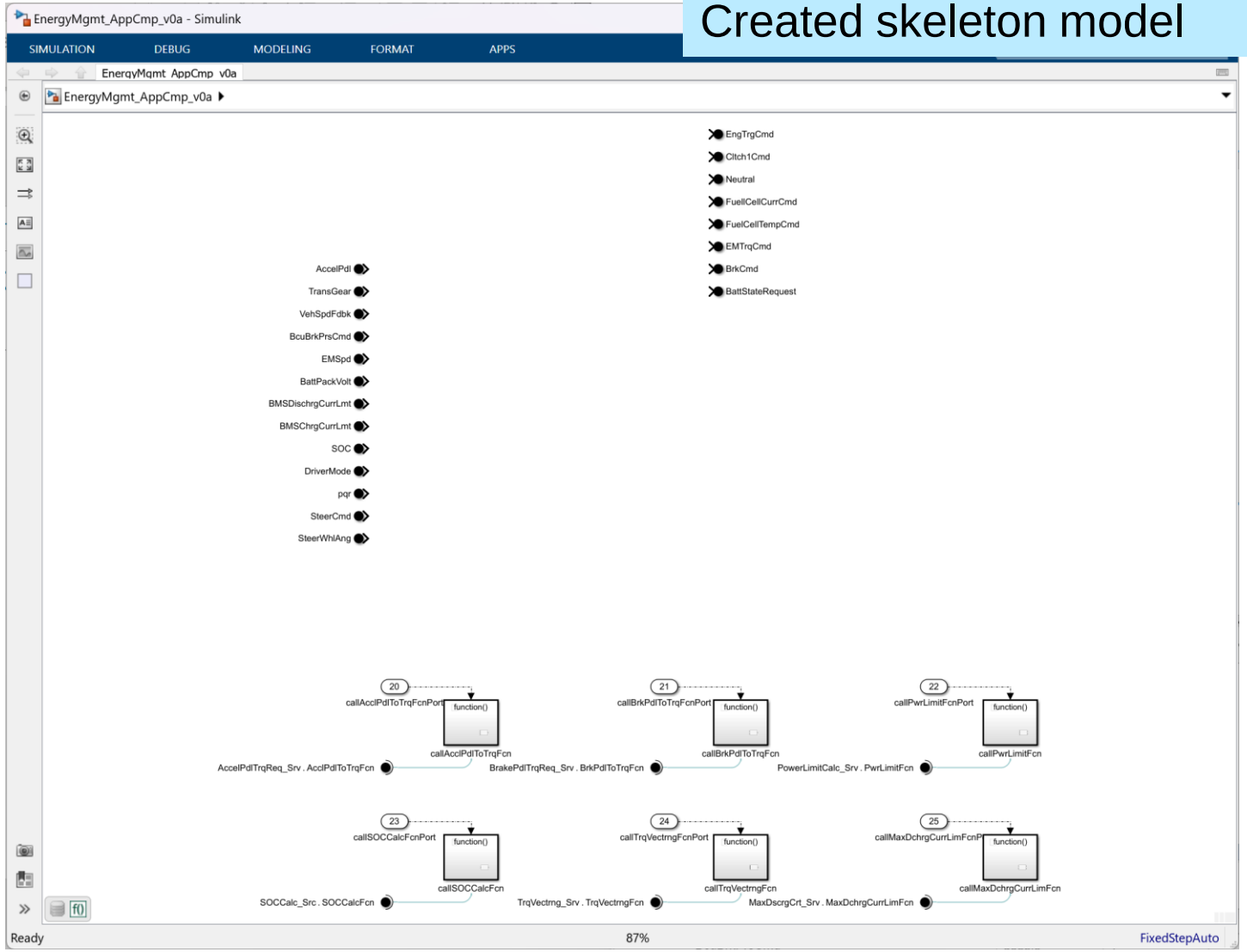


Create Simulink Behavior Model for Main Application Component

Original Model



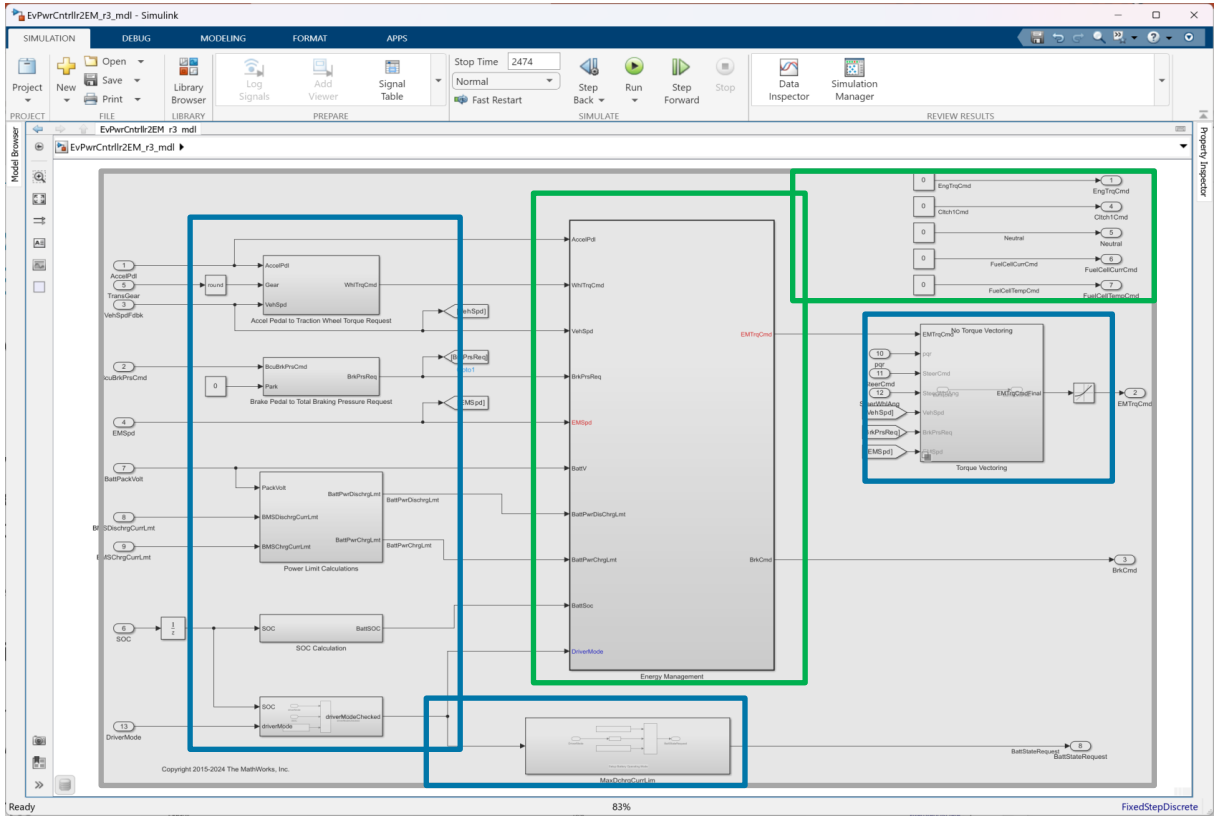
Created skeleton model



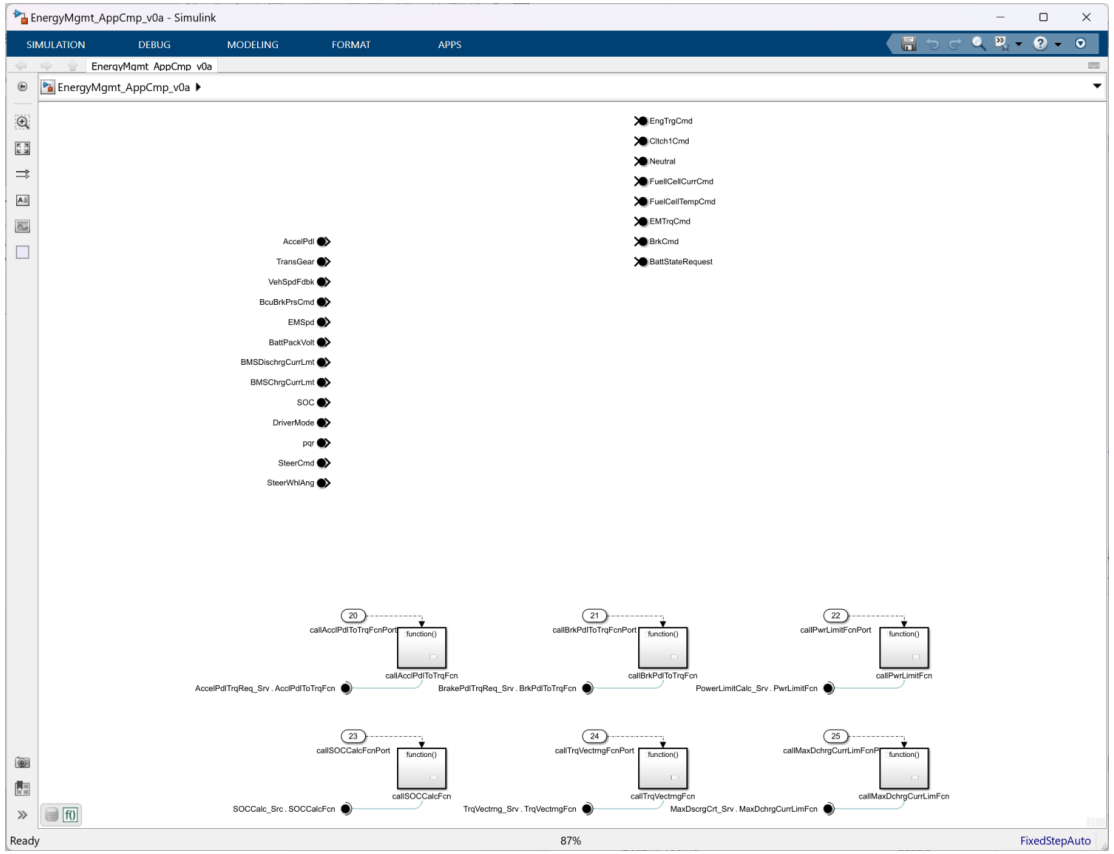
Process:

- Copy/paste parts from original model that we are keeping
- Alter parts that represent the client / caller functions

Create Simulink Behavior Model for Main Application Component

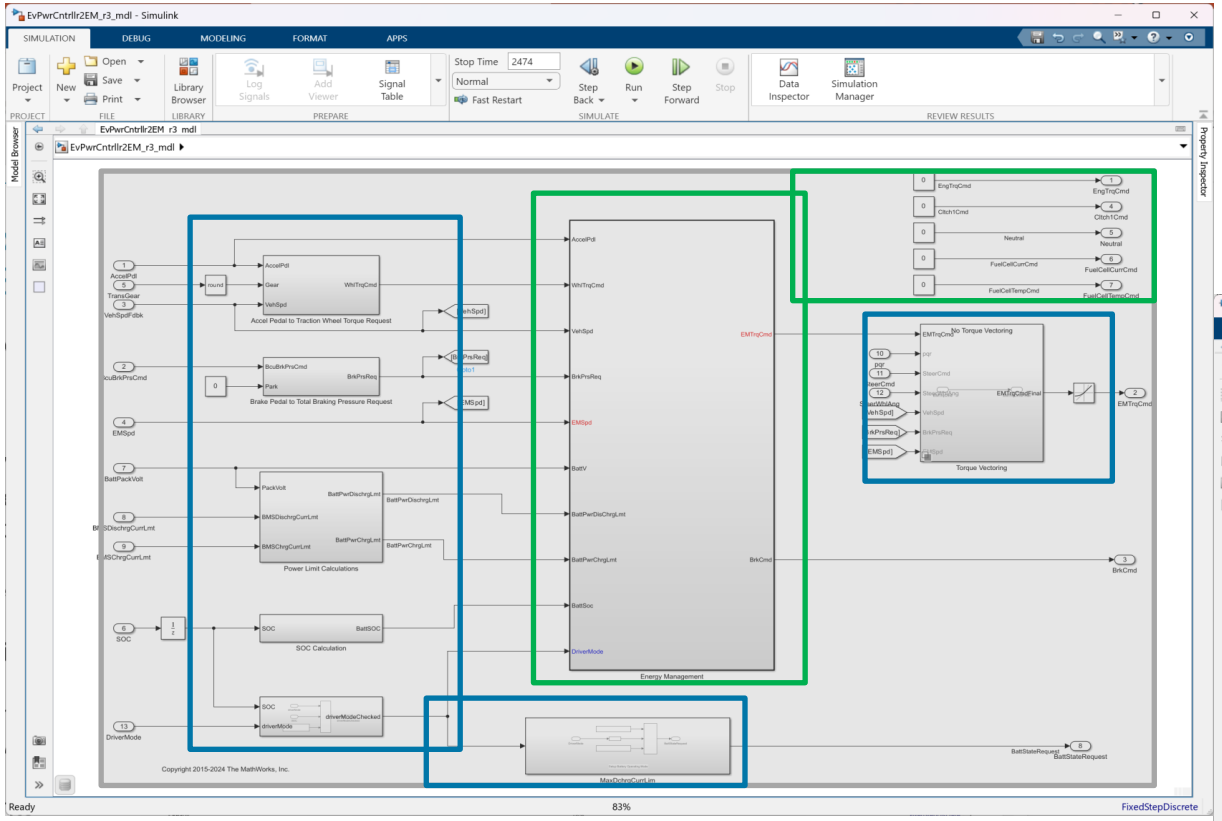


Original Model



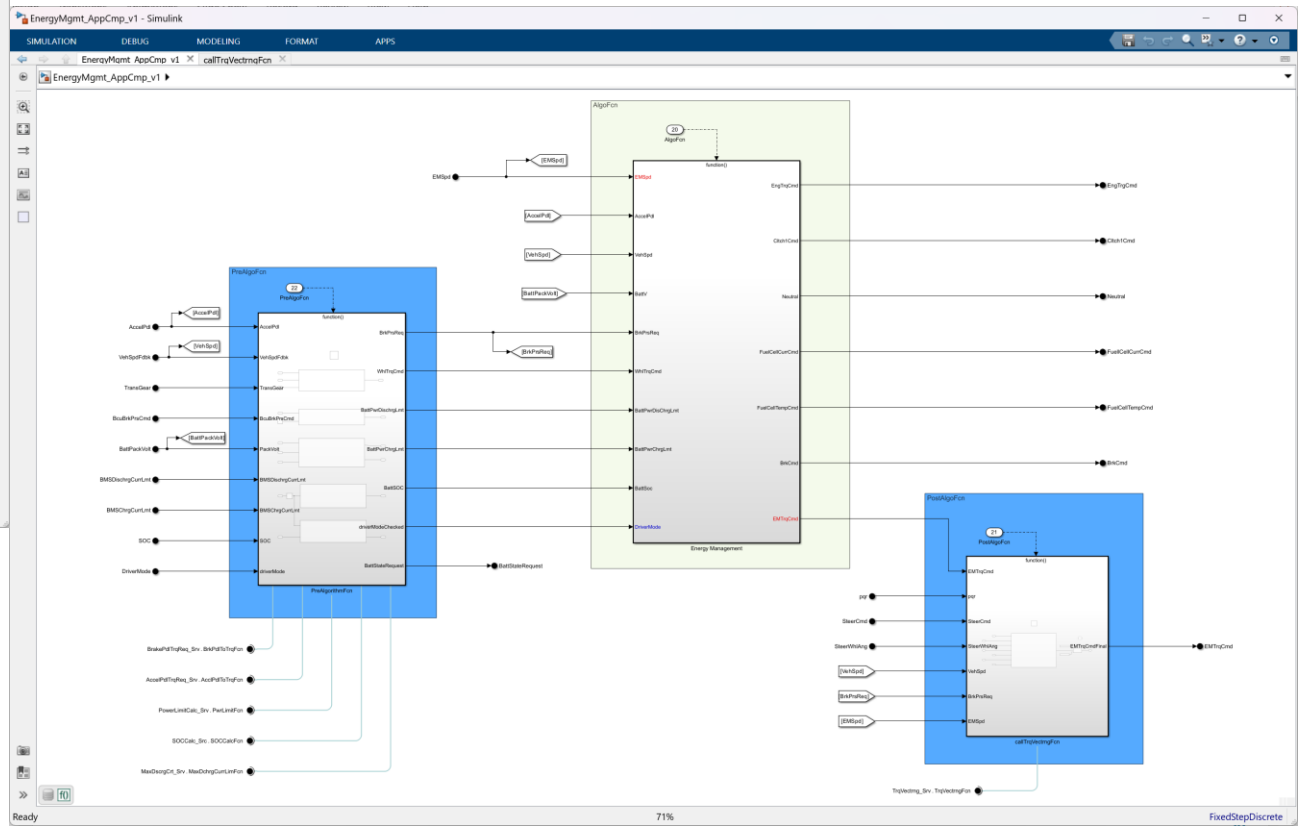
Transform this shell model to capture original intent

Create Simulink Behavior Model for Main Application Component



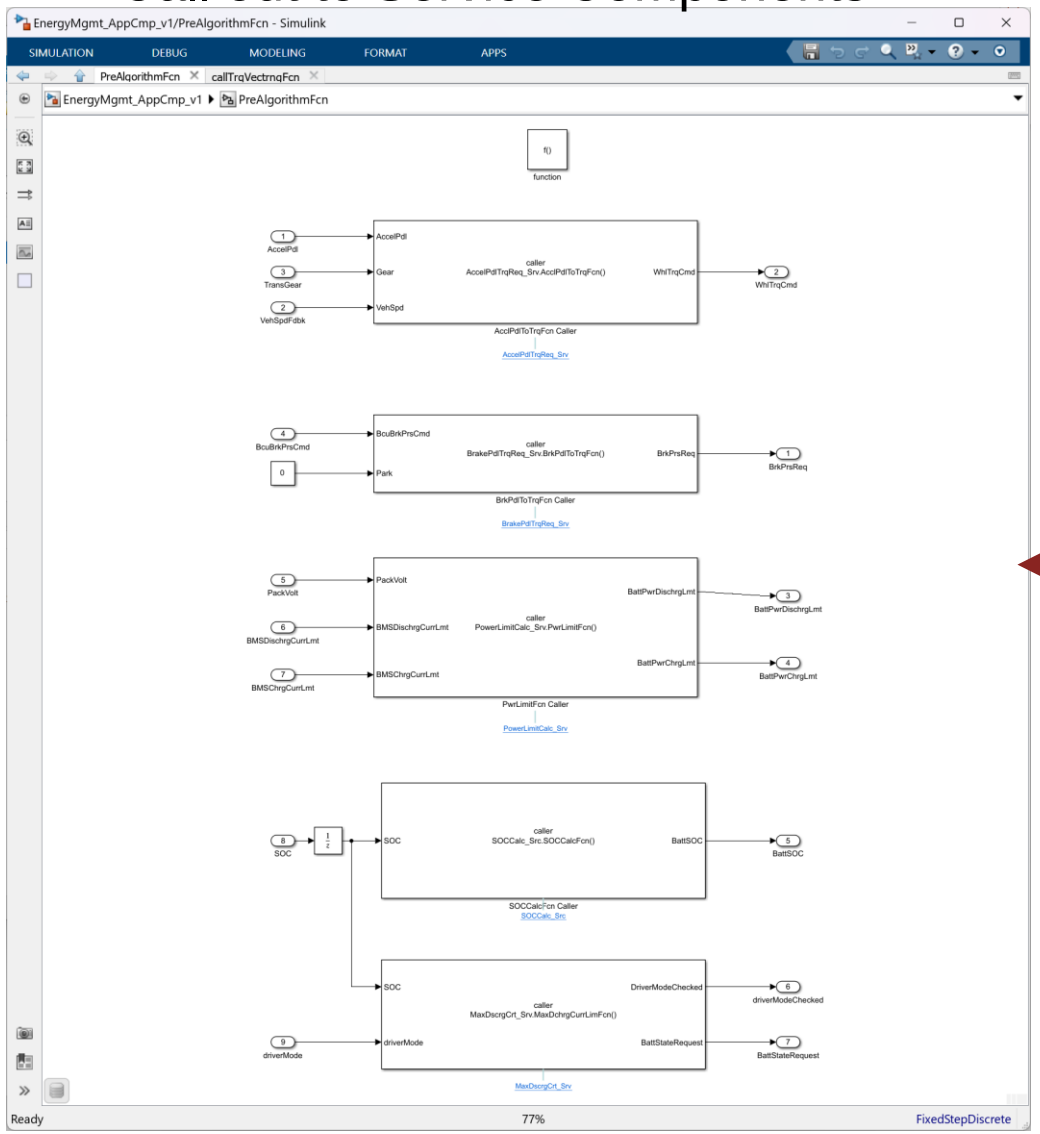
Original Model

New Simulink behavior model that captures Application logic and Service Call outs

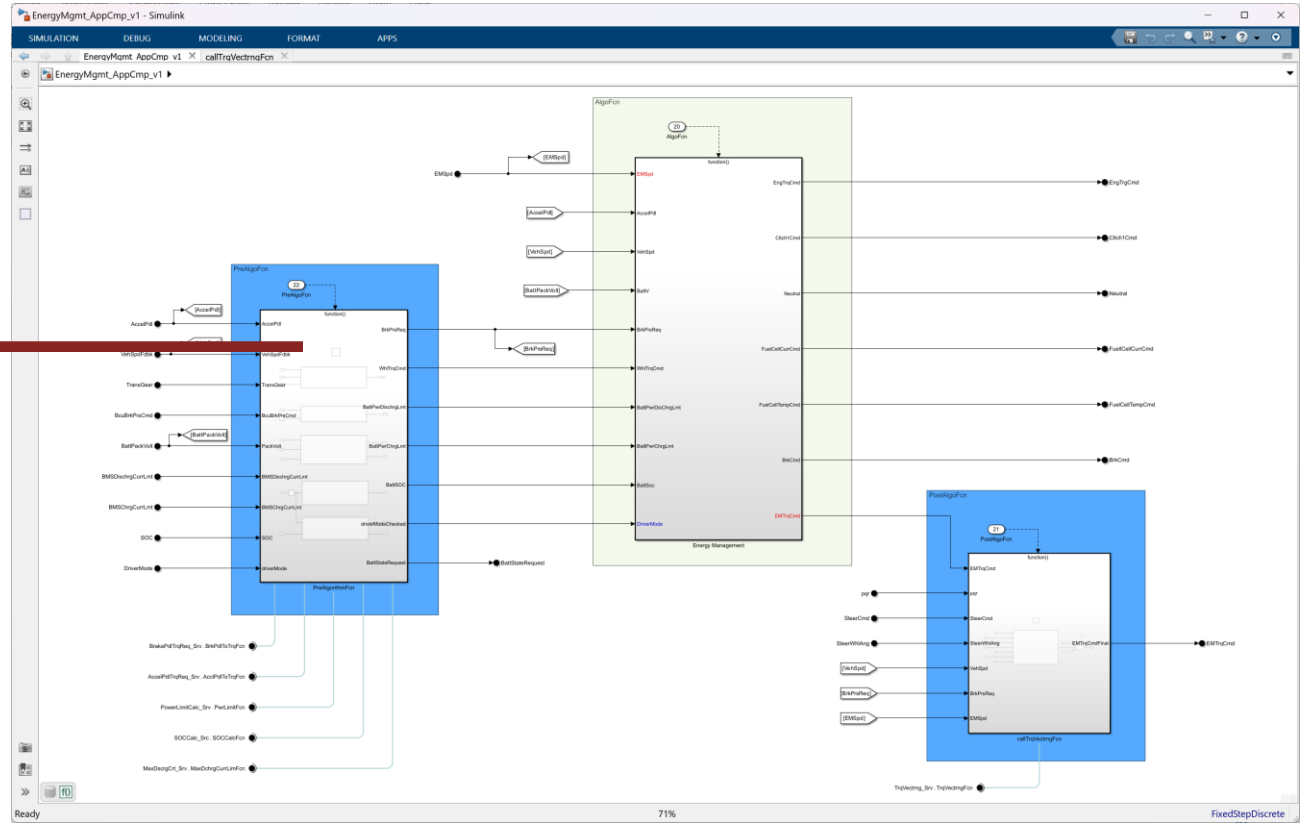


Create Simulink Behavior Model for Main Application Component

Call out to Service Components

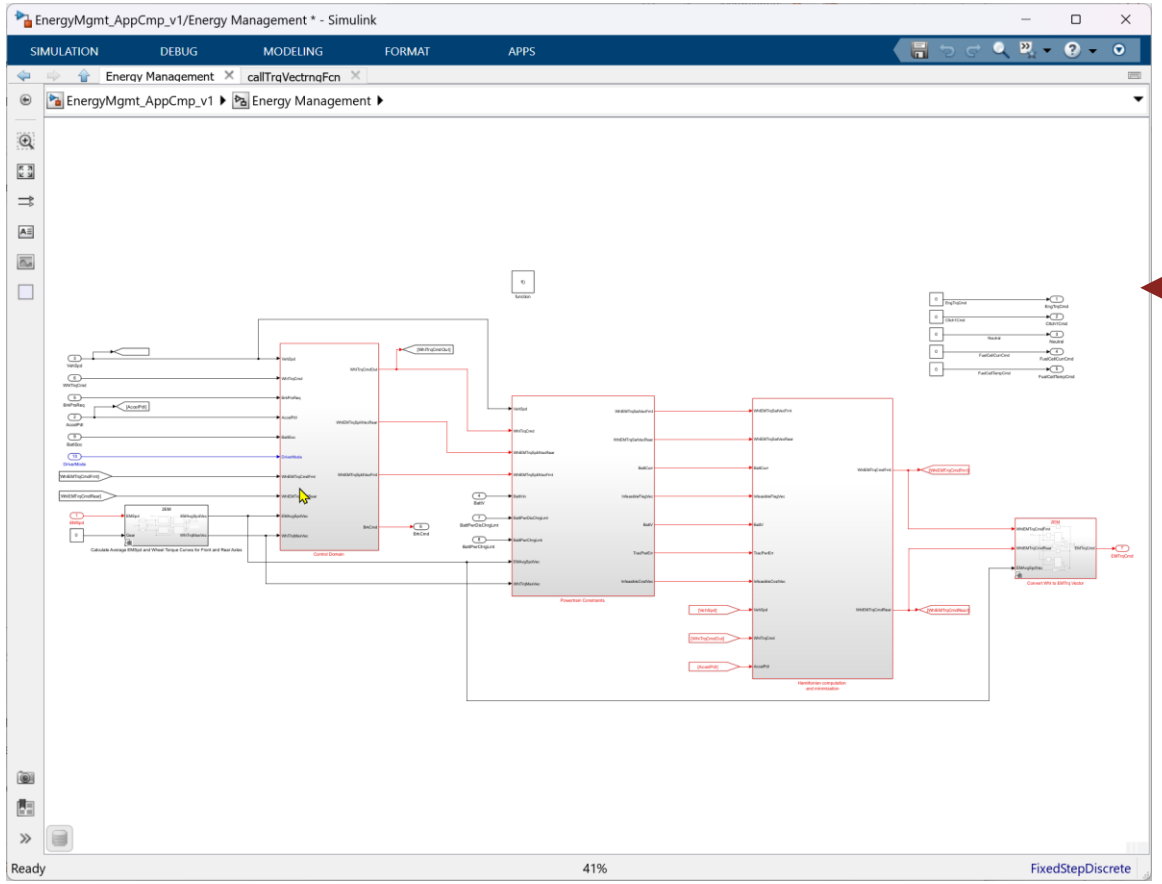


New Simulink behavior model that captures Application logic and Service Call outs

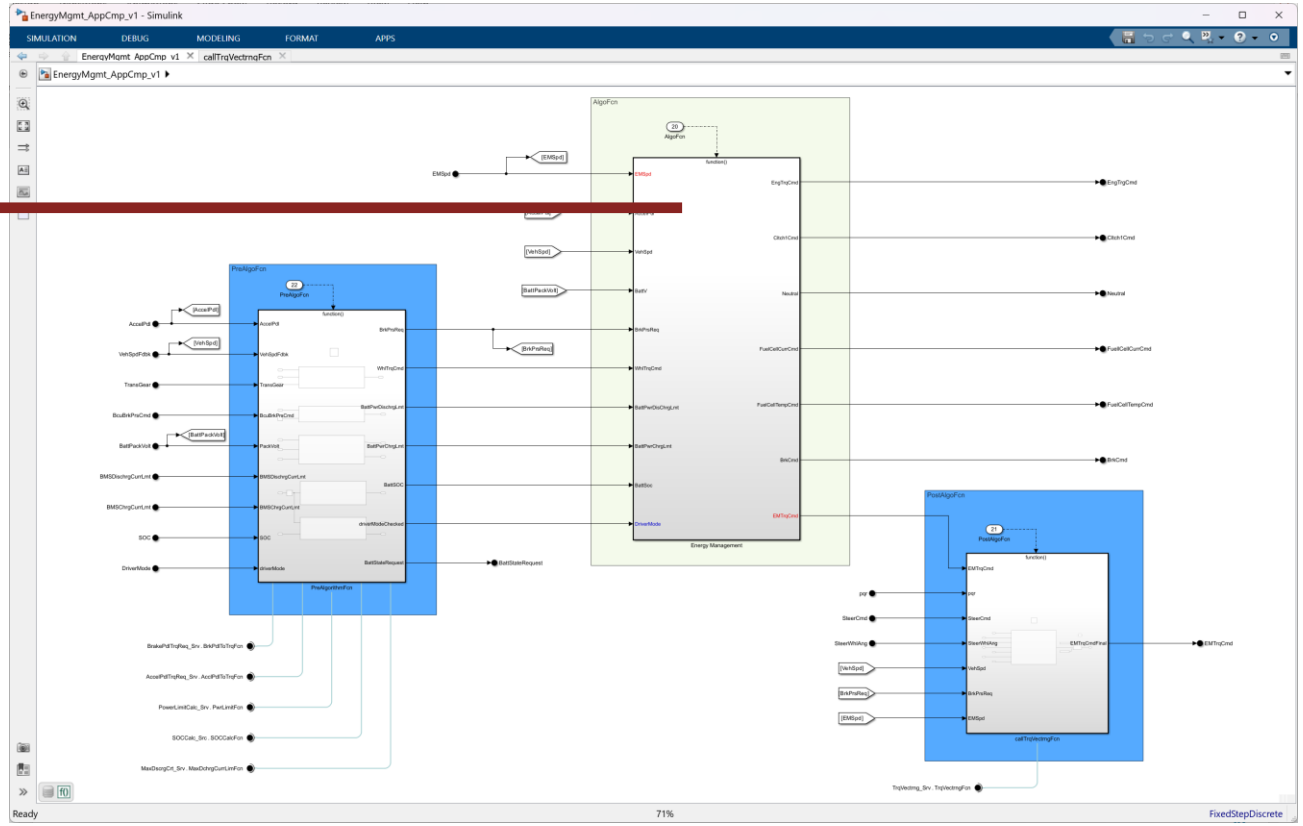


Create Simulink Behavior Model for Main Application Component

Original Logic that we left alone

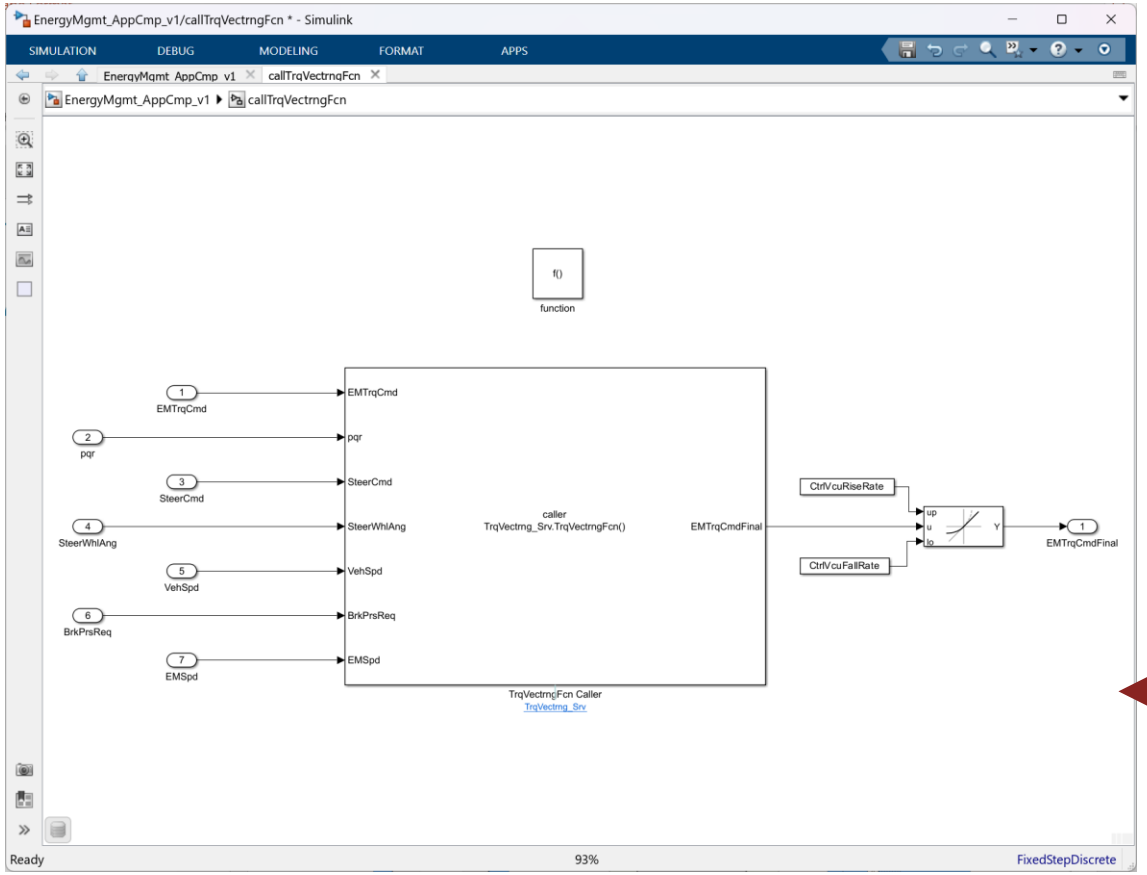


New Simulink behavior model that captures Application logic and Service Call outs

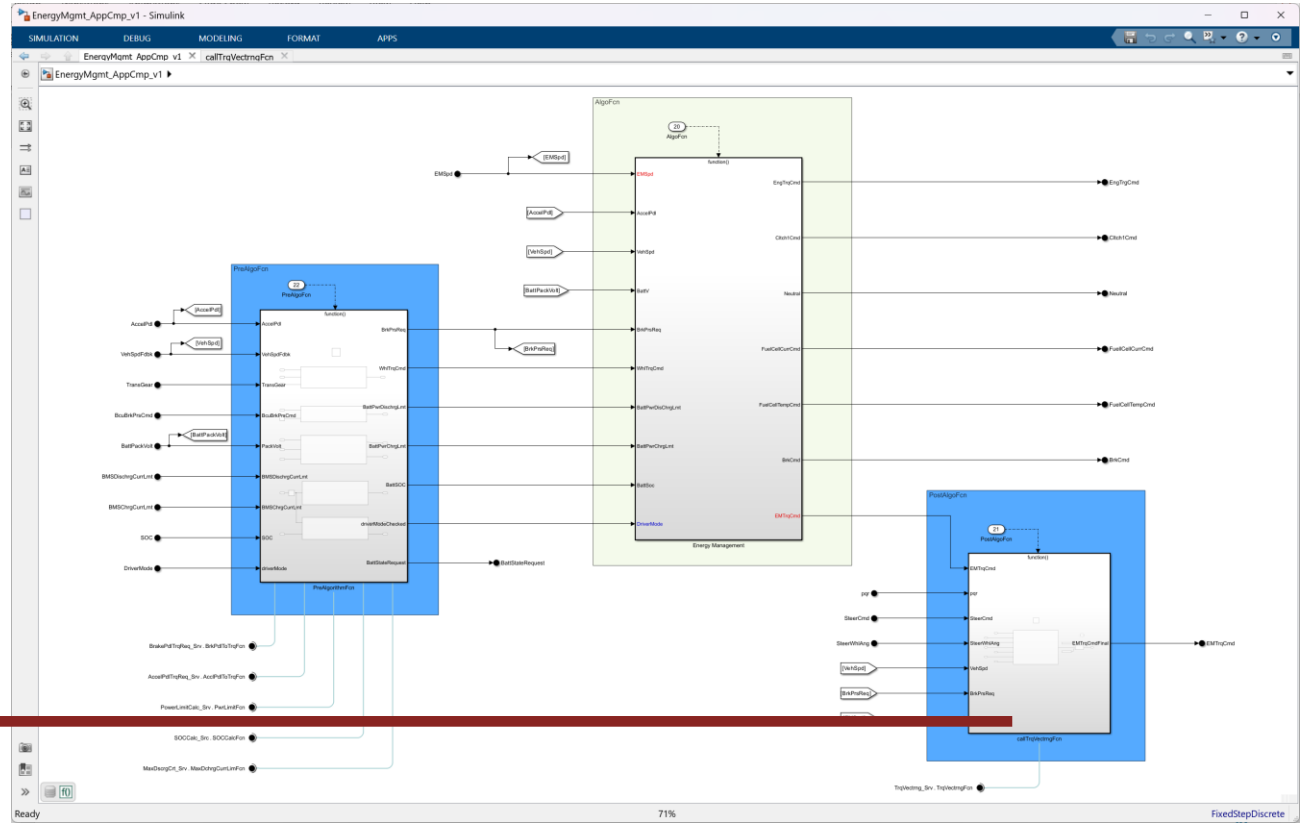


Create Simulink Behavior Model for Main Application Component

Call out to Service Component

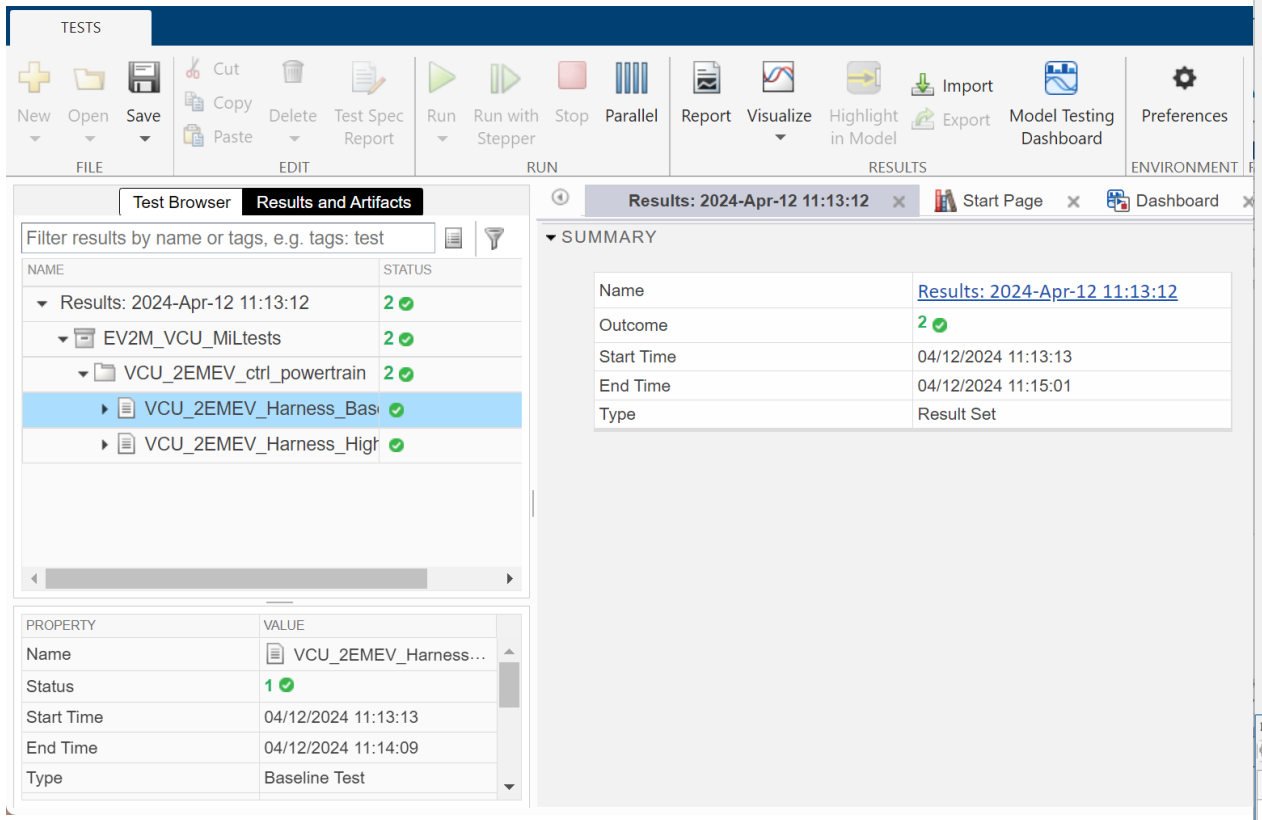


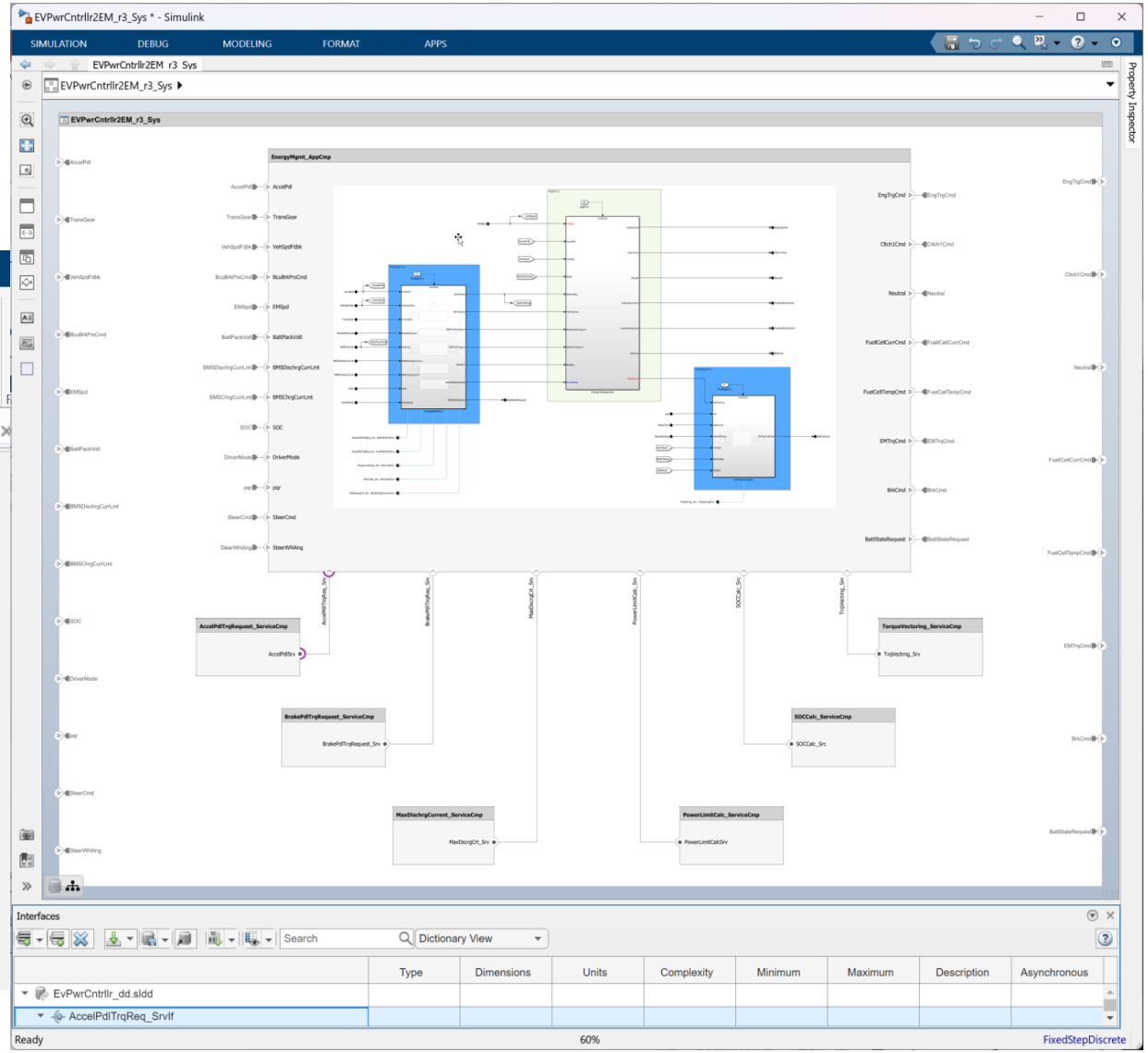
New Simulink behavior model that captures Application logic and Service Call outs



End refactoring process with Baseline Tests that pass

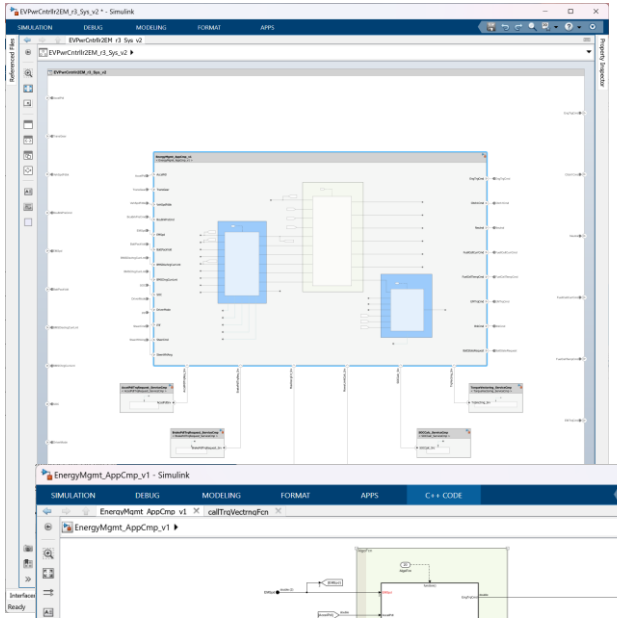
Very Important: Apply original model tests such as to prove out that functionality remains unchanged

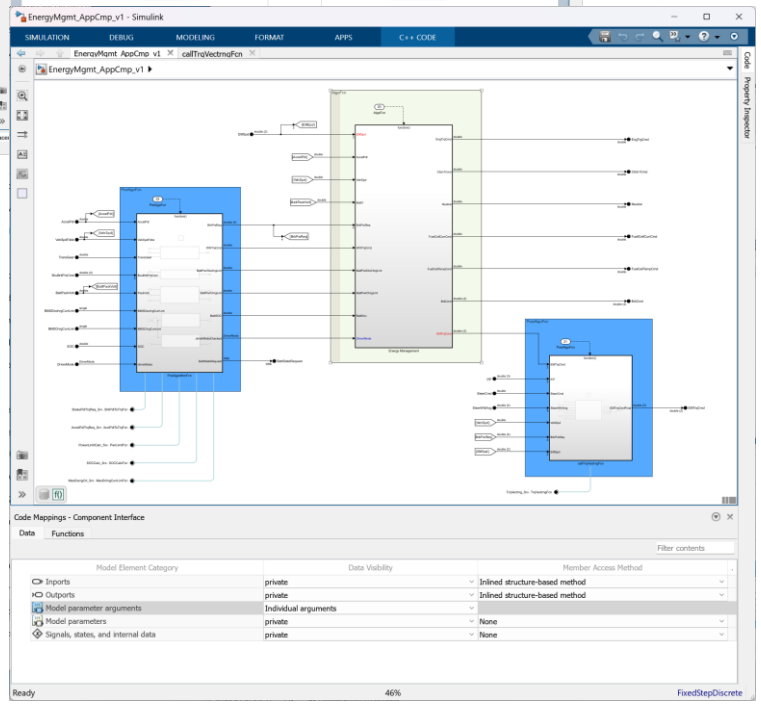




Tests Passed After Model Modifications

Generate C++ code that is agnostic to any platform



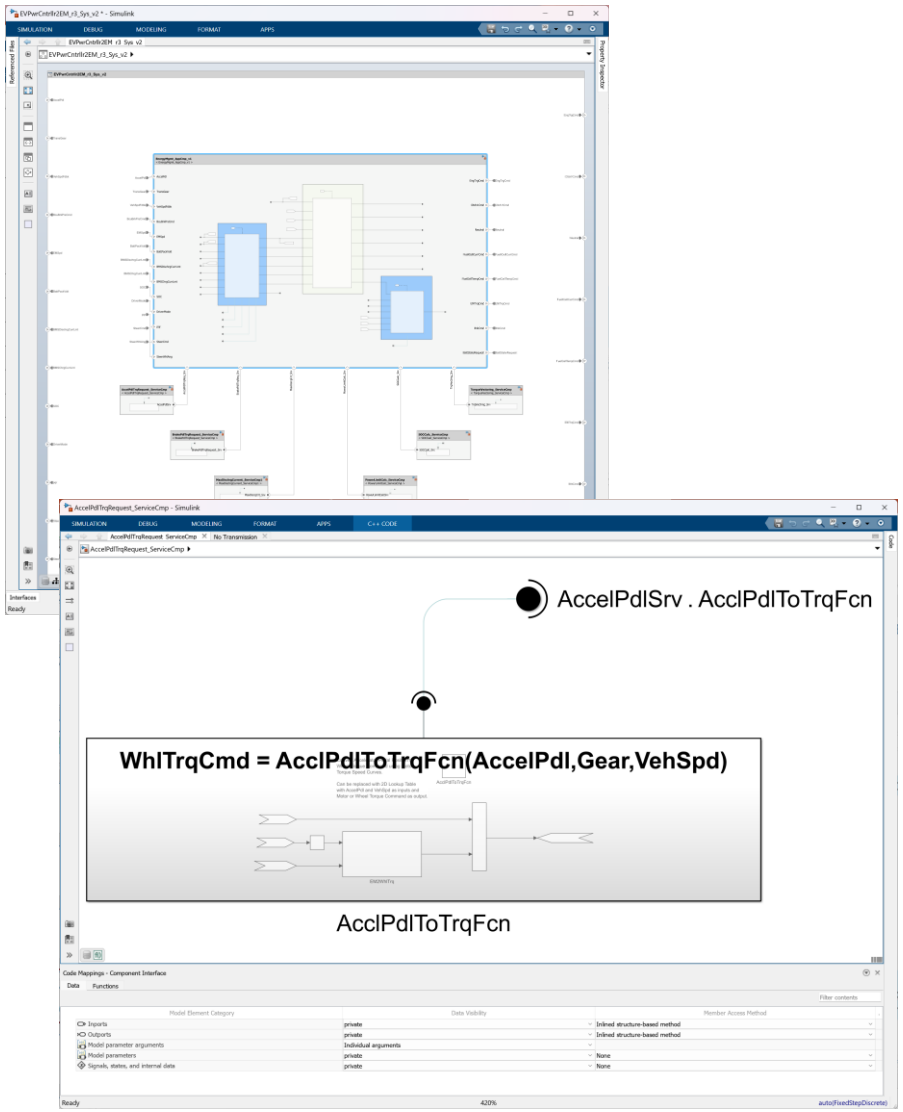


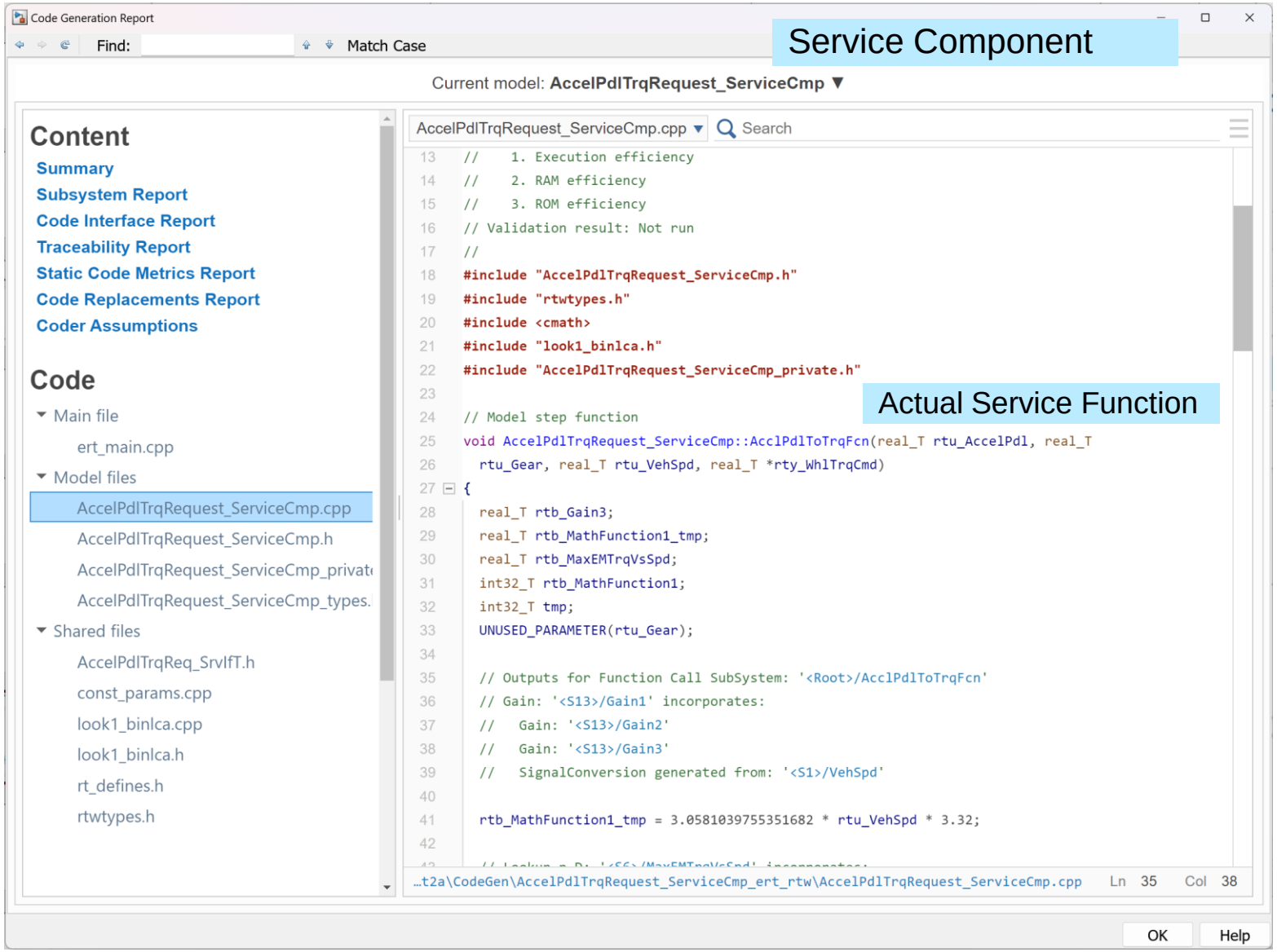
Application Component

PreAlgoFcn Body

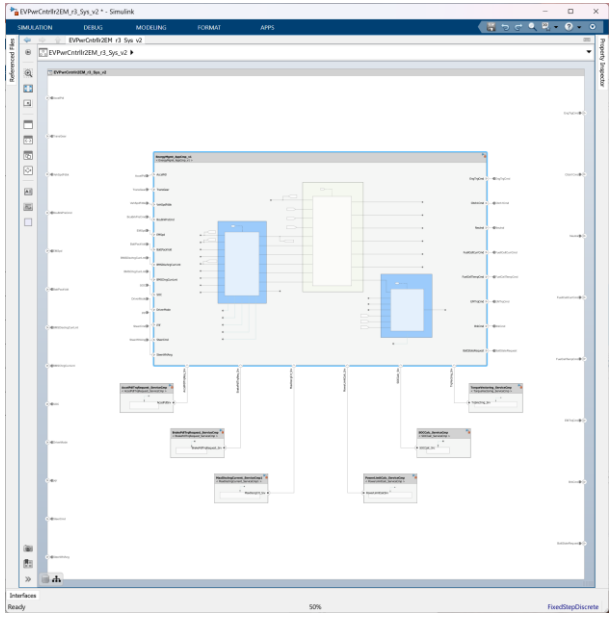
Call out to AccelPdlTrqReq_Srv Service Function

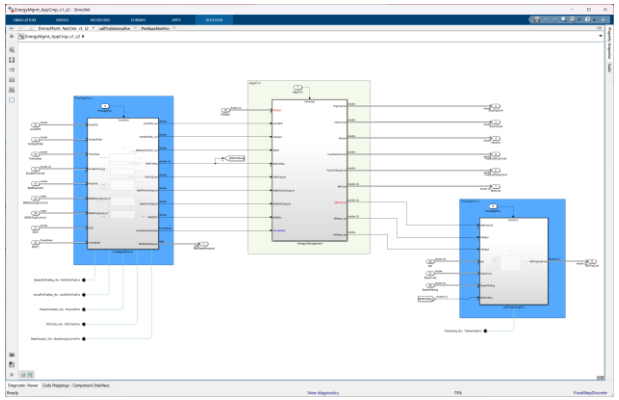
Generate C++ code that is agnostic to any platform





AP: Generate C++ Code for Adaptive AUTOSAR Platform





Application Component

Code Generation Report

Find: Match Case

Current model: **EnergyMgmt_AppCmp_v1_z2**

Content

- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Eliminated Blocks
- Code Replacements Report
- Coder Assumptions

Code

- Model files
  - EnergyMgmt_AppCmp_v1_z2.cpp
  - EnergyMgmt_AppCmp_v1_z2.h
  - EnergyMgmt_AppCmp_v1_z2_private.h
  - EnergyMgmt_AppCmp_v1_z2_types.h
- Shared files
  - binsearch_u32d_prevIdx.cpp
  - binsearch_u32d_prevIdx.h
  - complex_types.h
  - const_params.cpp
  - intrp2d_la.cpp
  - intrp2d_la.h
  - look1_binlca.cpp
  - look1_binlca.h

EnergyMgmt_AppCmp_v1_z2.cpp ▼ Search

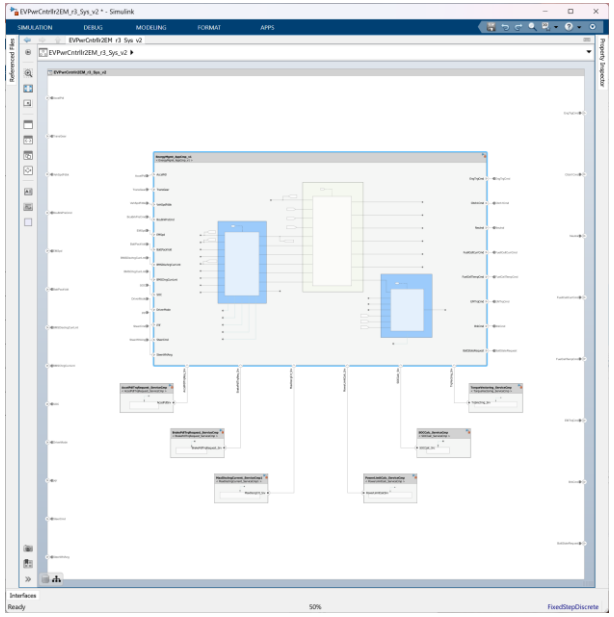
```
1589   ProvidedPort->EMTrqCmd.Send(arrMsgData);
1590 }
1591
1592 // Model step function for TID3
1593 void EnergyMgmt_AppCmp_v1_z2::PreAlgoFcn() // Explicit Task: PreAlgoFcn
1594 {
1595   ara::core::Array<double, 4 > arrayArgBrkPdlToTrqFcn;
1596   ara::core::Future<proxy::methods::AcclPdlToTrqFcn::Output>
1597     AcclPdlToTrqFcnFuture;
1598   ara::core::Future<proxy::methods::BrkPdlToTrqFcn::Output> BrkPdlToTrqFcnFuture;
1599   ara::core::Future<proxy::methods::MaxDchrgCurrLimFcn::Output>
1600     MaxDchrgCurrLimFcnFuture;
1601   ara::core::Future<proxy::methods::PwrLimitFcn::Output> PwrLimitFcnFuture;
1602   ara::core::Future<proxy::methods::SOCCalcFcn::Output> SOCCalcFcnFuture;
1603   proxy::methods::AcclPdlToTrqFcn::Output callOutput;
1604   proxy::methods::BrkPdlToTrqFcn::Output callOutput_1;
1605   proxy::methods::MaxDchrgCurrLimFcn::Output callOutput_2;
1606   proxy::methods::PwrLimitFcn::Output callOutput_0;
1607   proxy::methods::SOCCalcFcn::Output callOutput_3;
1608   std::shared_ptr<ara::core::Result<proxy::methods::AcclPdlToTrqFcn::Output>>
1609     AcclPdlToTrqFcnResultPtr;
1610   std::shared_ptr<ara::core::Result<proxy::methods::BrkPdlToTrqFcn::Output>>
1611     BrkPdlToTrqFcnResultPtr;
1612   std::shared_ptr<ara::core::Result<proxy::methods::MaxDchrgCurrLimFcn::Output>>
1613     MaxDchrgCurrLimFcnResultPtr;
1614   std::shared_ptr<ara::core::Result<proxy::methods::PwrLimitFcn::Output>>
1615     PwrLimitFcnResultPtr;
1616   std::shared_ptr<ara::core::Result<proxy::methods::SOCCalcFcn::Output>>
1617     SOCCalcFcnResultPtr;
```

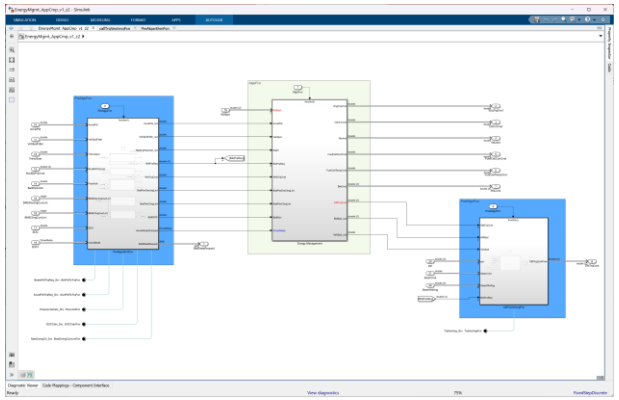
…MAC2024_DevShare\03_VCU Models_ExpPart2b\CodeGen\EnergyMgmt_AppCmp_v1_z2_autosar_adaptive\EnergyMgmt_AppCmp_v1_z2.cpp Ln 1596 Col 3

PreAlgoFcn Body

OK Help

AP: Generate C++ Code for Adaptive AUTOSAR Platform





Application Component

Code Generation Report

Find: Match Case

Current model: **EnergyMgmt_AppCmp_v1_z2**

Content

- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Eliminated Blocks
- Code Replacements Report
- Coder Assumptions

Code

- Model files
  - EnergyMgmt_AppCmp_v1_z2.cpp
  - EnergyMgmt_AppCmp_v1_z2.h
  - EnergyMgmt_AppCmp_v1_z2_private.h
  - EnergyMgmt_AppCmp_v1_z2_types.h
- Shared files
  - binsearch_u32d_prevIdx.cpp
  - binsearch_u32d_prevIdx.h
  - complex_types.h
  - const_params.cpp
  - intrp2d_la.cpp
  - intrp2d_la.h
  - look1_binlca.cpp
  - look1_binlca.h

EnergyMgmt_AppCmp_v1_z2.cpp ▾ Search

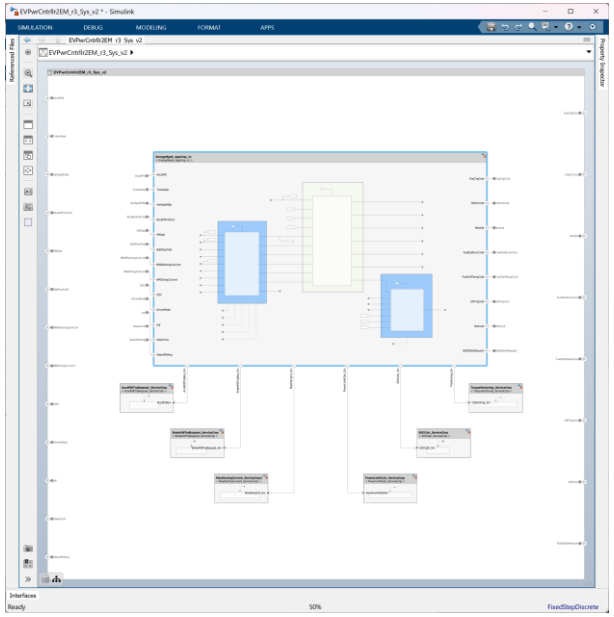
```
1650
1651  if (AccelPdlTrqReq_Srv) {
1652    // RootInportFunctionCallGenerator generated from: '<Root>/PreAlgoFcn' incorporates:
1653    //   SubSystem: '<Root>/PreAlgorithmFcn'
1654
1655    // FunctionCaller: '<S2>/AcclPdlToTrqFcn Caller'
1656    AcclPdlToTrqFcnFuture = AccelPdlTrqReq_Srv->AcclPdlToTrqFcn
1657      (EnergyMgmt_AppCmp_v1_z2_B.EventReceive,
1658       EnergyMgmt_AppCmp_v1_z2_B.EventReceive2,
1659       EnergyMgmt_AppCmp_v1_z2_B.EventReceive1);
1660
1661    // End of Outputs for RootInportFunctionCallGenerator generated from: '<Root>/PreAlgoFcn'
1662    // Retrieve result on method AcclPdlToTrqFcn's completion
1663    AcclPdlToTrqFcnResultPtr = std::make_shared< ara::core::Result<proxy::
1664      methods::AcclPdlToTrqFcn::Output> >(AcclPdlToTrqFcnFuture.GetResult());
1665
1666    // Check if method AcclPdlToTrqFcn completed successfully and returned valid results
1667    if (AcclPdlToTrqFcnResultPtr->HasValue()) {
1668      // Retrieve return arguments from method AcclPdlToTrqFcn's Result container
1669      callOutput = AcclPdlToTrqFcnResultPtr->Value();
1670
1671      // FunctionCaller: '<S2>/AcclPdlToTrqFcn Caller'
1672      EnergyMgmt_AppCmp_v1_z2_B.AcclPdlToTrqFcnCaller = callOutput.WhlTrqCmd;
1673    }
1674  }
1675
1676  // RootInportFunctionCallGenerator generated from: '<Root>/PreAlgoFcn' incorporates:
1677  //   SubSystem: '<Root>/PreAlgorithmFcn'
```

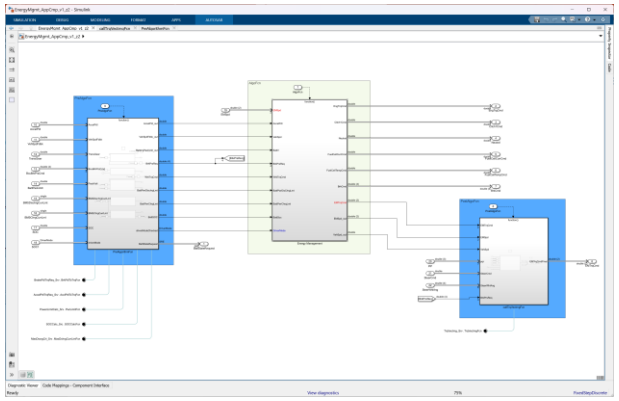
Call out to AccelPdlTrqReq_Srv Service Function

…MAC2024_DevShare\03_VCU Models_ExpPart2b\CodeGen\EnergyMgmt_AppCmp_v1_z2_autosar_adaptive\EnergyMgmt_AppCmp_v1_z2.cpp Ln 1669 Col 5

OK Help

AP: Generate C++ Code for Adaptive AUTOSAR Platform





Application Component

Main.cpp / Application

Code Generation Report

Find: Match Case

Current model: **EnergyMgmt_AppCmp_v1_z2**

Content

- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Eliminated Blocks
- Code Replacements Report
- Coder Assumptions

Code

- EnergyMgmt_AppCmp_v
- EnergyMgmt_AppCmp_v
- EnergyMgmt_AppCmp_v
- EnergyMgmt_AppCmp_v
- Other files
  - PosixExecutor.hpp
- Other files
  - main.cpp
- ARA files
  - MachineManifest.arxml
  - accelpdltrqreq_srvif_com
  - accelpdltrqreq_srvif_pro
  - brakepdltrqreq_srvif_con
  - brakepdltrqreq_srvif_pro
  - impl_type_drivermode.h
  - impl_type_rt_array_doub

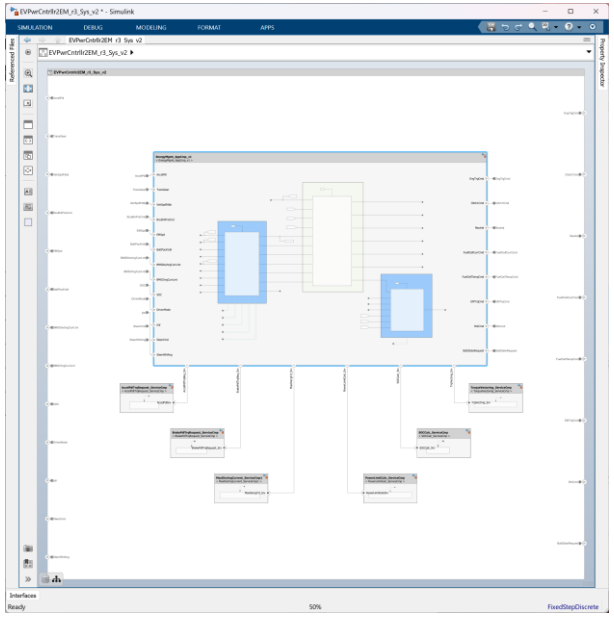
main.cpp ▼ Search

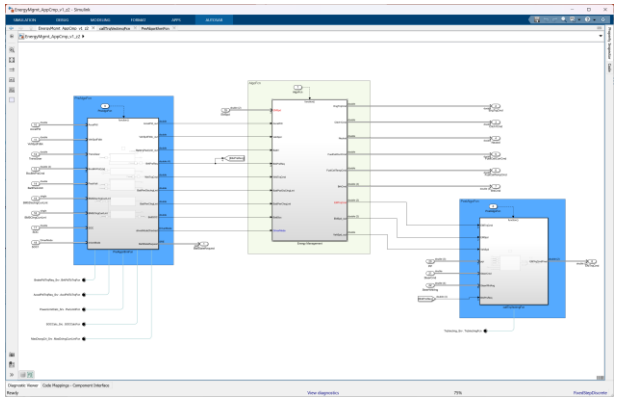
```
9  #include <ara/core/initialization.h>
10 #include <ara/core/result.h>
11 #include <ara/core/string_view.h>
12 #include <ara/exec/execution_client.h>
13 #include <ara/log/common.h>
14 #include <ara/log/logger.h>
15 #include <ara/log/log_stream.h>
16 #include "EnergyMgmt_AppCmp_v1_z2.h"
17
18 /* main() handles the following: */
19 /*  - Instantiates the model object and owns its memory. */
20 /*  - Reports the Execution state to ARA */
21 /*  - Calls the model's initialize and terminate functions. */
22 /*  - Creates an executor instance to schedule the periodic step functions */
23 /*        - A timer that is set to the base rate is created in the executor */
24 /*        - The step functions are added to the executor and run */
25 /*          based on their sample periods */
26 int32_t main() {
27   /* Used to control the flow in case of error in any api's used. */
28   bool bProceed{true};
29   /* Used to decide whether ara function clusters has been initialized. */
30   bool bAraInitialized{true};
31   /* ara function cluster init. */
32   const ara::core::Result<void> initStatus{ara::core::Initialize()};
33
34   if (!initStatus.HasValue()) {
35     bProceed = false;
36     bAraInitialized = false;
37   } /* if */
38
39   if (bAraInitialized) {
40     ara::log::Logger &araLog{ara::log::CreateLogger(
```

…orks\Documents\00_AE_Info_Content\AE_Share\MAC2024_DevShare\03_VCU Models_ExpPart2b\CodeGen\EnergyMgmt_AppCmp_v1_z2_autosar_adaptive\main.cpp Ln 27 Col 6

OK Help

AP: Generate C++ Code for Adaptive AUTOSAR Platform





Application Component

Code Generation Report

Find: Match Case

Current model: **EnergyMgmt_AppCmp_v1_z2**

Content

- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Eliminated Blocks
- Code Replacements Report
- Coder Assumptions

Code

- EnergyMgmt_AppCmp_v
- Other files
  - PosixExecutor.hpp
- Other files
  - main.cpp
- ARA files
  - MachineManifest.arxml
  - accelpdltrqreq_srvif_com
  - accelpdltrqreq_srvif_pro
  - brakepdltrqreq_srvif_con
  - brakepdltrqreq_srvif_pro
  - impl_type_drivermode.h
  - impl_type_rt_array_doub
  - impl_type_rt_array_doub
  - impl_type_rt_array_doub

main.cpp ▼ Search

```
82    fcnExecutor.setBaseRateInSeconds(std::chrono::duration<double>(baseRate));
83
84    /* Register periodic step functions in the executor. */
85    fcnExecutor.addPeriodicEvent(
86        [&EnergyMgmt_AppCmp_v1_z2_Obj, &araLog]() {
87          try {
88            EnergyMgmt_AppCmp_v1_z2_Obj.PreAlgoFcn();
89          } catch (std::exception const &e) {
90            araLog.LogError() << "Error executing step: " << e.what();
91          }
92        },
93        1);
94    fcnExecutor.addPeriodicEvent(
95        [&EnergyMgmt_AppCmp_v1_z2_Obj, &araLog]() {
96          try {
97            EnergyMgmt_AppCmp_v1_z2_Obj.AlgoFcn();
98          } catch (std::exception const &e) {
99            araLog.LogError() << "Error executing step: " << e.what();
100         }
101       },
102       1);
103   fcnExecutor.addPeriodicEvent(
104       [&EnergyMgmt_AppCmp_v1_z2_Obj, &araLog]() {
105         try {
106           EnergyMgmt_AppCmp_v1_z2_Obj.PostAlgoFcn();
107         } catch (std::exception const &e) {
108           araLog.LogError() << "Error executing step: " << e.what();
109         }
110       },
111       1);
```

…Share\MAC2024_DevShare\03_VCU Models_ExpPart2b\CodeGen\EnergyMgmt_AppCmp_v1_z2_autosar_adaptive\main.cpp Ln 107 Col 35

OK Help

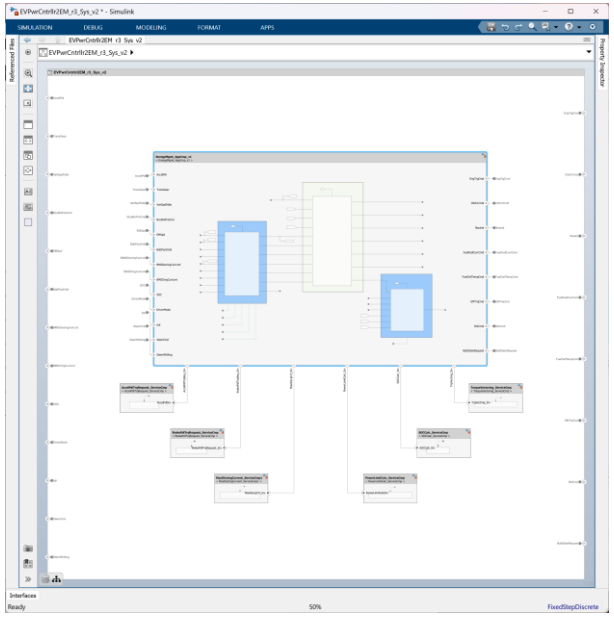
Main.cpp / Application

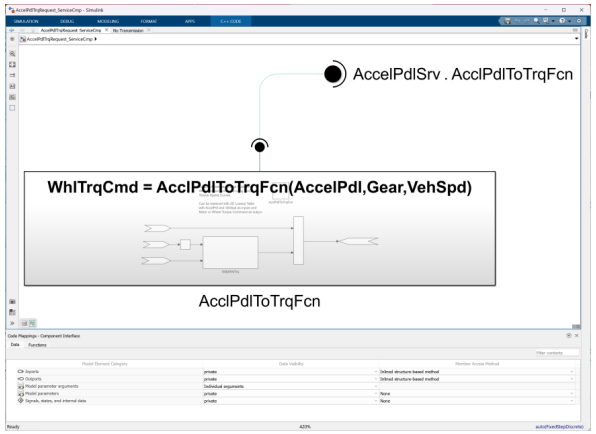
Adding tasks for Main Application executor

Ln 27 Col 6

OK Help

AP: Generate C++ Code for Adaptive AUTOSAR Platform





Code Generation Report

Find: Match Case

Service Component

Current model: **AccelPdlTrqRequest_ServiceCmp** ▼

Content

- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Code Replacements Report
- Coder Assumptions

Code

- Model files
  - AccelPdlTrqRequest_ServiceCmp.cpp
  - AccelPdlTrqRequest_ServiceCmp.h
  - AccelPdlTrqRequest_ServiceCmp_priv
  - AccelPdlTrqRequest_ServiceCmp_typ
- Shared files
  - complex_types.h
  - const_params.cpp
  - look1_binlcapw.cpp
  - look1_binlcapw.h
  - rt_defines.h
- Interface files
  - AccelPdlTrqRequest_ServiceCmp_Exe
  - AccelPdlTrqRequest_ServiceCmp_Ser
  - AccelPdlTrqRequest_ServiceCmp_cor

AccelPdlTrqRequest_ServiceCmp.cpp ▼ Search

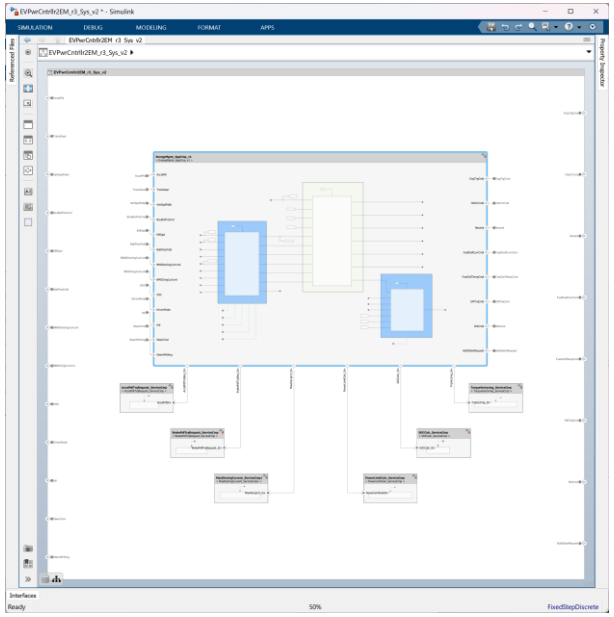
Actual Service Function

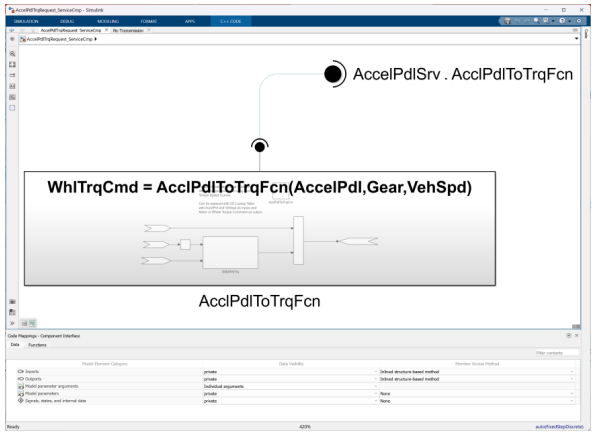
```
19  #include <stdint.h>
20  #include "look1_binlcapw.h"
21  #include "AccelPdlTrqRequest_ServiceCmp_private.h"
22
23  // Model step function
24  void AccelPdlTrqRequest_ServiceCmp::AcclPdlToTrqFcn(double AccelPdl, double Gear,
25    double VehSpd, double *WhlTrqCmd)
26  {
27    double rtb_Gain3;
28    double rtb_MathFunction1_tmp;
29    double rtb_MaxEMTrqVsSpd;
30    int32_t rtb_MathFunction1;
31    int32_t tmp;
32    UNUSED_PARAMETER(Gear);
33
34    // Outputs for Function Call SubSystem: '<Root>/AcclPdlToTrqFcn'
35    // Gain: '<S13>/Gain1' incorporates:
36    //   Gain: '<S13>/Gain2'
37    //   Gain: '<S13>/Gain3'
38    //   SignalConversion generated from: '<S1>/VehSpd'
39
40    rtb_MathFunction1_tmp = 3.0581039755351682 * VehSpd * 3.32;
41
42    // Lookup_n-D: '<S6>/MaxEMTrqVsSpd' incorporates:
43    //   Gain: '<S13>/Gain1'
44
45    rtb_MaxEMTrqVsSpd = look1_binlcapw(rtb_MathFunction1_tmp,
46      &rtCP_MaxEMTrqVsSpd_bp01Data[0], &rtCP_MaxEMTrqVsSpd_tableData[0], 13U);
47
```

…t2b\CodeGen\AccelPdlTrqRequest_ServiceCmp_autosar_adaptive\AccelPdlTrqRequest_ServiceCmp.cpp Ln 19 Col 3

OK Help

AP: Generate C++ Code for Adaptive AUTOSAR Platform





Service Component

Code Generation Report

Find: Match Case

Current model: **AccelPdlTrqRequest_ServiceCmp**

- AccelPdlTrqRequest_ServiceCmp.h
- AccelPdlTrqRequest_ServiceCmp_priv
- AccelPdlTrqRequest_ServiceCmp_typ
- Shared files
  - complex_types.h
  - const_params.cpp
  - look1_binlcapw.cpp
  - look1_binlcapw.h
  - rt_defines.h
- Interface files
  - AccelPdlTrqRequest_ServiceCmp_Exe
  - AccelPdlTrqRequest_ServiceCmp_Ser
  - AccelPdlTrqRequest_ServiceCmp_cor
  - AccelPdlTrqRequest_ServiceCmp_dat
  - AccelPdlTrqRequest_ServiceCmp_inte
- Other files
  - PosixExecutor.hpp
- Other files
  - main.cpp
- ARA files
  - MachineManifest.arxml
  - accelpdltrqreq_srvif_common.h
  - accelpdltrqreq_srvif_skeleton.h
  - accelpdltrqreq_srvif_skeleton_impl.h

main.cpp ▾ Search

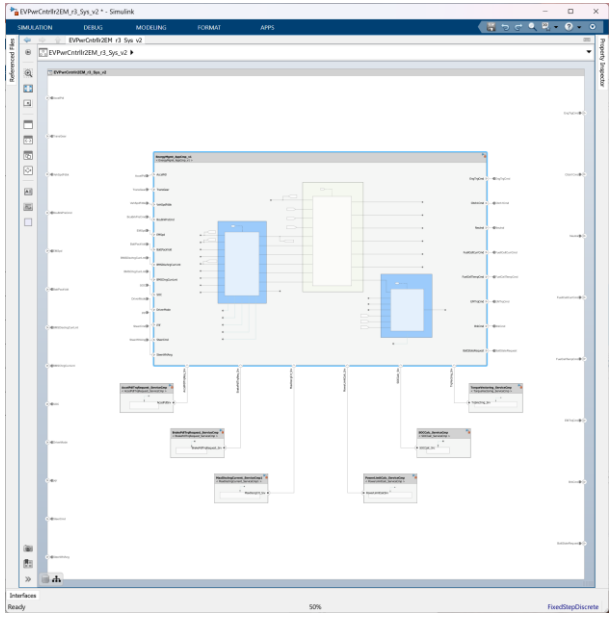
Main.cpp / Service

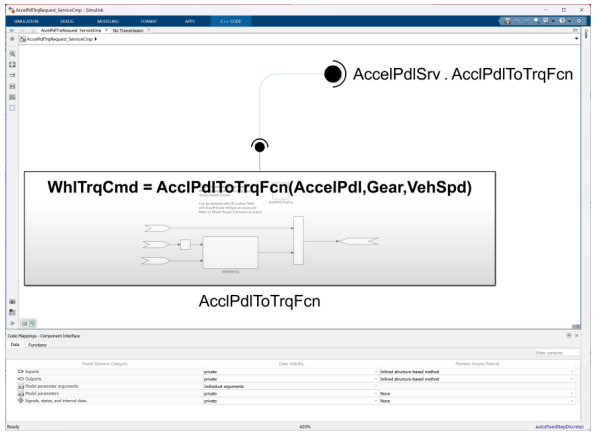
```
21  /*  - Calls the model's initialize and terminate functions. */
22  /*  - Creates an executor instance to schedule the periodic step functions */
23  /*       - A timer that is set to the base rate is created in the executor */
24  /*       - The step functions are added to the executor and run */
25  /*         based on their sample periods */
26  int32_t main() {
27    /* Used to control the flow in case of error in any api's used. */
28    bool bProceed{true};
29    /* Used to decide whether ara function clusters has been initialized. */
30    bool bAraInitialized{true};
31    /* ara function cluster init. */
32    const ara::core::Result<void> initStatus{ara::core::Initialize()};
33
34    if (!initStatus.HasValue()) {
35      bProceed = false;
36      bAraInitialized = false;
37    } /* if */
38
39    if (bAraInitialized) {
40      ara::log::Logger &araLog{ara::log::CreateLogger(
41          ara::core::StringView{"Acce"},
42          ara::core::StringView{
43              "Logger for AccelPdlTrqRequest_ServiceCmp's main function."},
44          ara::log::LogLevel::kWarn)};
45
46      /* Report Execution state */
47      const ara::exec::ExecutionClient exec_client;
48      try {
49        if (!exec_client.ReportExecutionState(
```

…hare\03_VCU Models_ExpPart2b\CodeGen\AccelPdlTrqRequest_ServiceCmp_autosar_adaptive\main.cpp Ln 30 Col 1

OK Help

AP: Generate C++ Code for Adaptive AUTOSAR Platform





Service Component

Code Generation Report

Find: Match Case

Current model: **AccelPdlTrqRequest_ServiceCmp**

AccelPdlTrqRequest_ServiceCmp.h
AccelPdlTrqRequest_ServiceCmp_priv
AccelPdlTrqRequest_ServiceCmp_typ
Shared files
complex_types.h
const_params.cpp
look1_binlcapw.cpp
look1_binlcapw.h
rt_defines.h
Interface files
AccelPdlTrqRequest_ServiceCmp_Exe
AccelPdlTrqRequest_ServiceCmp_Ser
AccelPdlTrqRequest_ServiceCmp_cor
AccelPdlTrqRequest_ServiceCmp_dat
AccelPdlTrqRequest_ServiceCmp_inte
Other files
PosixExecutor.hpp
Other files
main.cpp
ARA files
MachineManifest.arxml
accelpdltrqreq_srvif_common.h
accelpdltrqreq_srvif_skeleton.h
accelpdltrqreq_srvif_skeleton_impl.h

main.cpp ▾ Search

```
77      /* AccelPdlTrqRequest_ServiceCmp::AcclPdlToTrqFcn(); */
78      /* These asynchronous tasks are registered */
79      /* as callbacks in the model initialize function: */
80      /* AccelPdlTrqRequest_ServiceCmp::initialize() */
81
82      /* Create an executor instance to schedule the periodic step functions. */
83      /* Whenever the period of a step function passes, the executor */
84      /* schedules that function to be executed on a thread. */
85      platform::runtime::Executor fcnExecutor;
86
87      /* Base rate is the time unit of a tick. */
88      constexpr double baseRate{0.200000};
89      fcnExecutor.setBaseRateInSeconds(std::chrono::duration<double>(baseRate));
90
91      araLog.LogVerbose() << "Starting periodic execution of step functions.";
92  #if defined(rtmSetStopRequested) && defined(rtmGetStopRequested)
93      fcnExecutor.run(
94          [&AccelPdlTrqRequest_ServiceCmp_Obj]() {
95            return rtmGetStopRequested(
96              AccelPdlTrqRequest_ServiceCmp_Obj.getRTM());
97          },
98          araLog);
99  #else
100     fcnExecutor.run(araLog);
101 #endif
102   } /* if */
103
104   if (bProceed) {
105     try {
```

…hare\03_VCU Models_ExpPart2b\CodeGen\AccelPdlTrqRequest_ServiceCmp_autosar_adaptive\main.cpp Ln 85 Col 39

OK Help

Main.cpp / Service

Adding Service to executor

Ln 30 Col 1

OK Help

Key Take Aways

- We have the tools to help move to Service Oriented Architectures (SOA)
- We can generate C++ code for SOA based components
- We are going to show you a lot of information in this presentation
- We are here to help so please engage with us in your SOA type projects

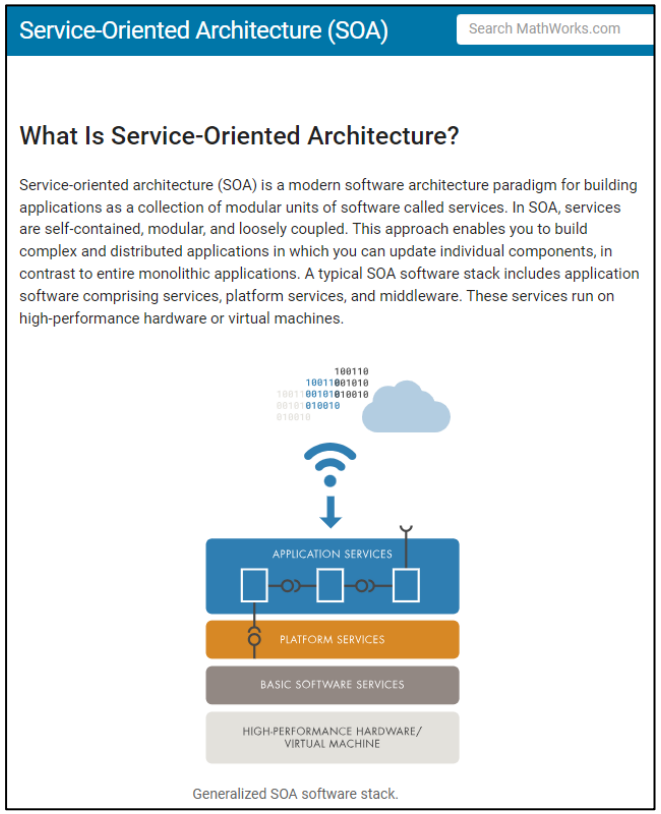
MathWorks Resources

Where can I find more information ?

[Solutions Page: Software-Defined Vehicle](#)

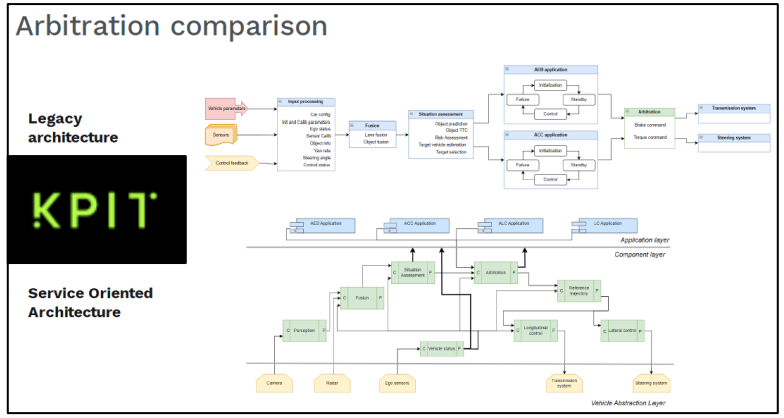
[Solutions Page: Software Architecture](#)

[What Is Service-Oriented Architecture (SOA)?](#)



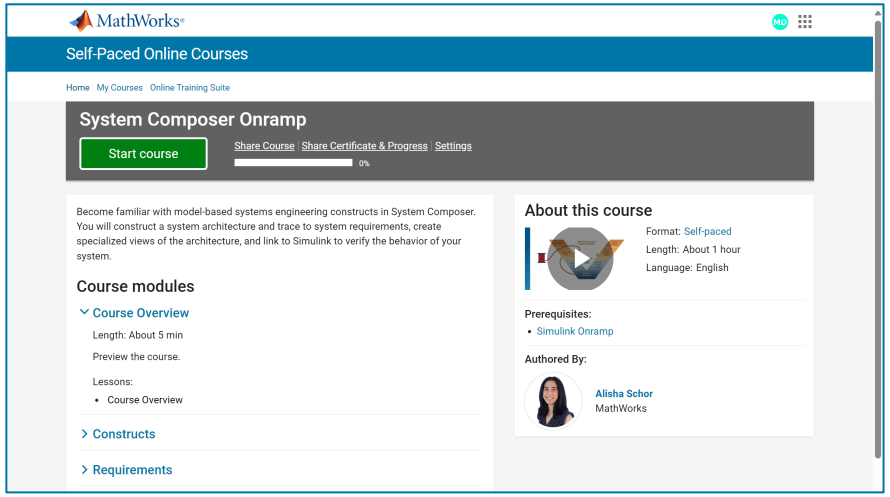
[Using Model-Based Design to Develop SOA Applications for In-Vehicle OS](#)

[Service-Oriented Arbitration of ADAS Features with Model-Based Design](#)

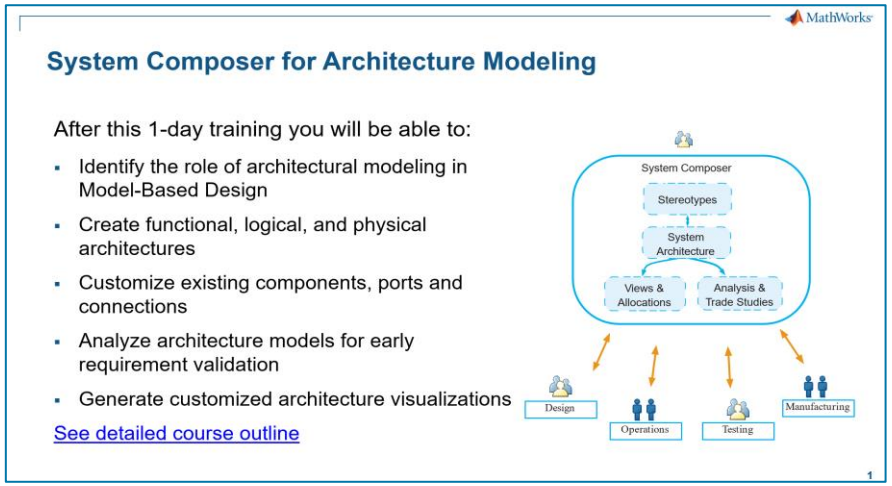


Training and Consulting Services

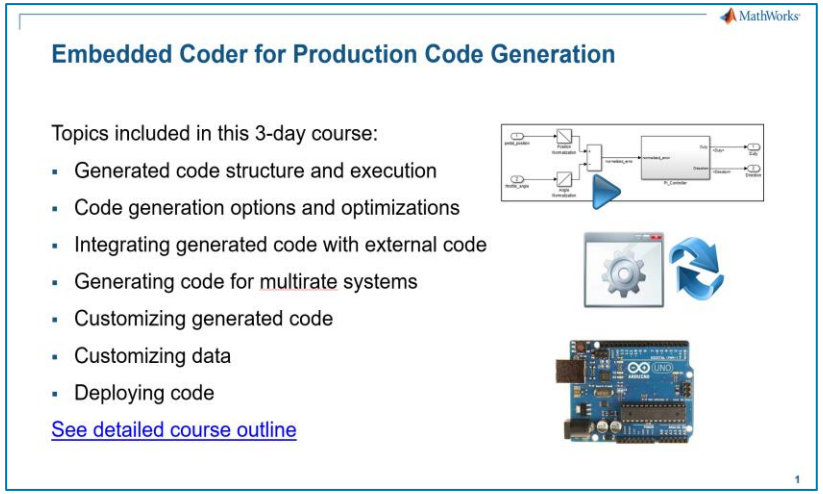
Where can I get more help ?



[System Composer Onramp]



[System Composer Course Details]



[Embedded Coder Course Details]

Training and Consulting Services

Where can I get more help ?

MathWorks

Code Generation for Classic AUTOSAR Software Components

After this two-day training you will be able to:

- Generate Simulink models from existing ARXML system descriptions
- Configure Simulink models for AUTOSAR compliant code generation
- Configure AUTOSAR communication elements in a Simulink model
- Model AUTOSAR events in Simulink
- Create calibration parameters
- Model AUTOSAR variation points within software components
- Import and export AUTOSAR compositions and software architectures
- Model calls to basic software services

- See detailed course outline

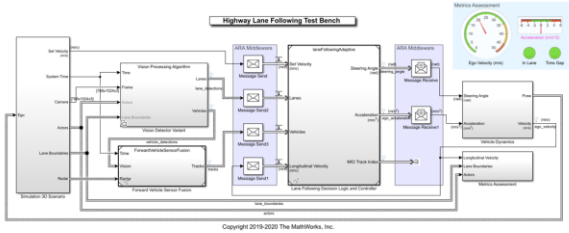
1

[Classic AUTOSAR Course Details]

MathWorks

AUTOSAR Adaptive Jumpstart Consulting Service

- Just enough Adaptive AUTOSAR
  - Learn Adaptive AUTOSAR elements relevant for MBD of application software
- Relevant exercises and workflows
  - Converting existing models to the Adaptive Platform
  - Choosing appropriate modeling constructs for Adaptive services
  - Developing, simulating, and testing Adaptive Applications
  - Generating and reviewing C++ AUTOSAR code
- Choose from 8-16hrs of coaching to fit your needs
  - AUTOSAR Adaptive Overview
  - Classic vs Adaptive
  - Adaptive platform architecture
  - Component modeling
  - Component code generation
  - Simulation, verification, and validation
  - Configuring for deployment
  - Measurement and calibration



AUTOSAR Adaptive Platform

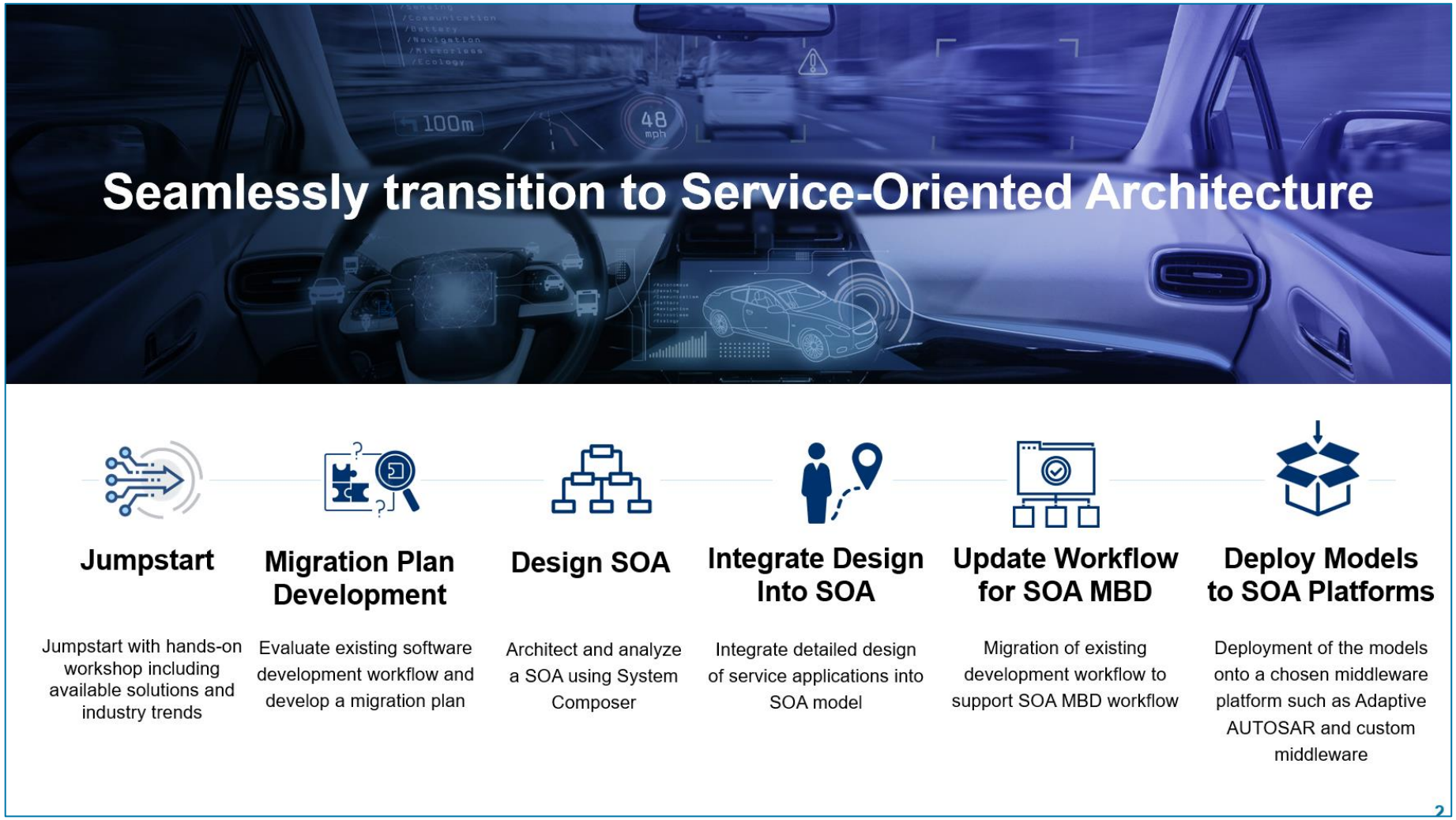
1-2 Day(s) / Onsite Instructor Led
Adaptive AUTOSAR Training

3

[AUTOSAR Adaptive Platform Jumpstart Course Details]

Training and Consulting Services

Where can I get more help ?



https://www.mathworks.com/services/consulting/contact.html

Technical Workshop on this topic – June 20th 1:30 to 3:30 PM EDT

If you have questions, you can reach out to me

Mark Danielsen
mdaniels@mathworks.com

Technical Workshop: Migration of a Monolithic Algorithm to Service-Oriented Architecture

Date & Time: June 20, 1:30PM – 3:30PM EDT

Overview: 2-hour interactive workshop at our Novi office to take a deep dive into the process of breaking apart a monolithic algorithm into services that can be reused.

MathWorks
AUTOMOTIVE CONFERENCE 2024
North America

Thank you

MathWorks®