

MATLAB EXPO

2024.06.11 | 그랜드 인터컨티넨탈 서울 파르나스

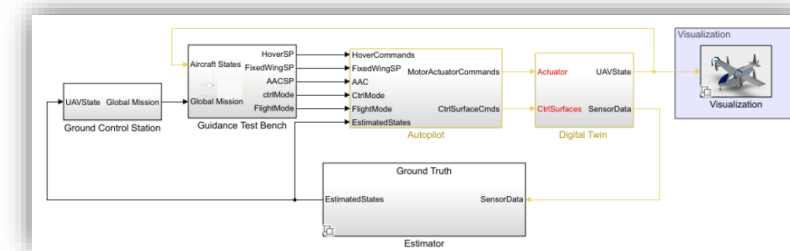
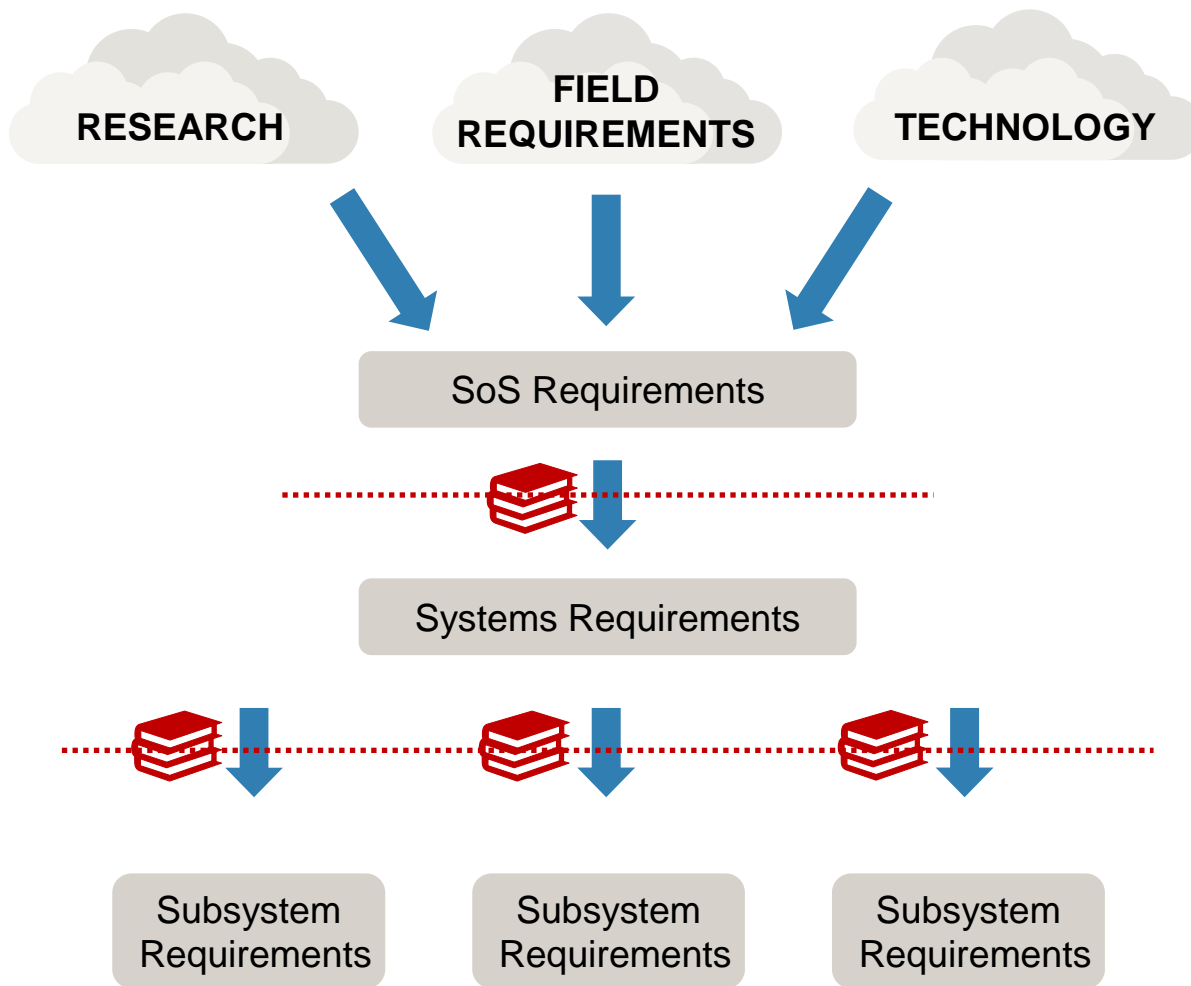
DDS Blockset 및 System Composer를 활용한 무인항공기 시스템의 분산 시뮬레이션과 아키텍처 설계

유성재, 매스웍스코리아

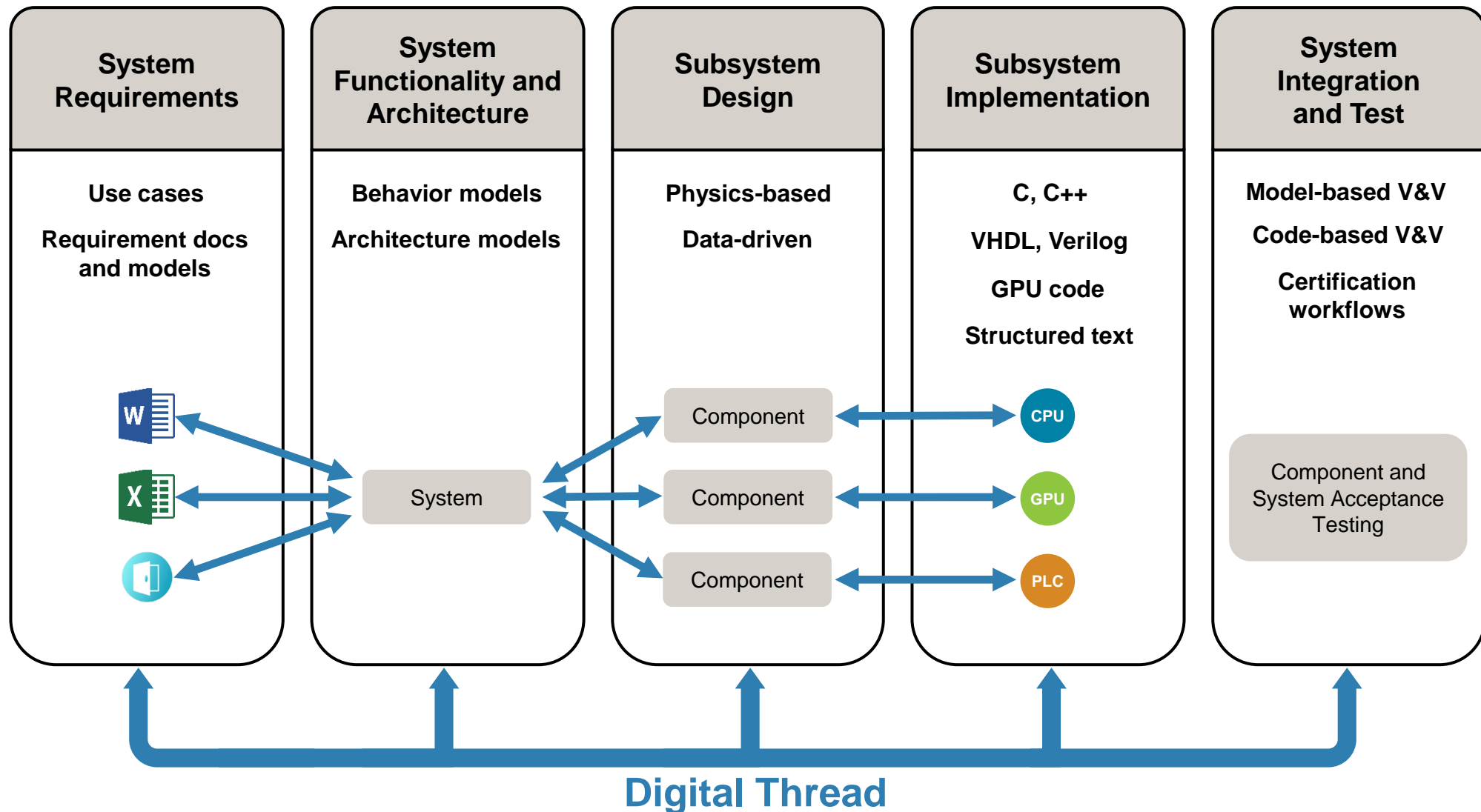


System of Systems

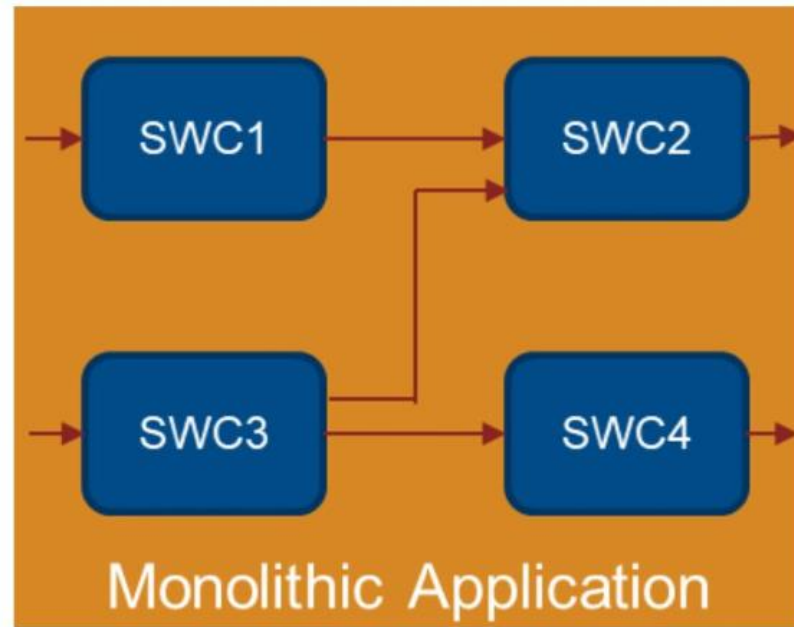
Challenges are bigger as the perspective grows



Model-Based Design

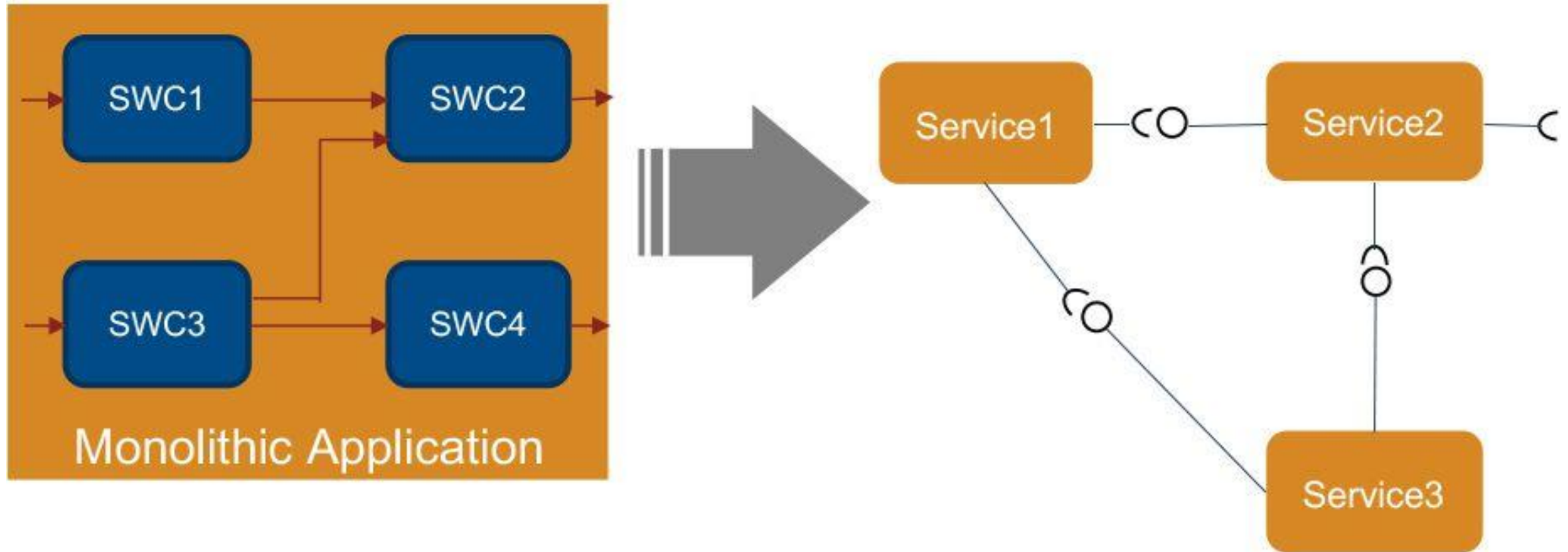


Monolithic Application

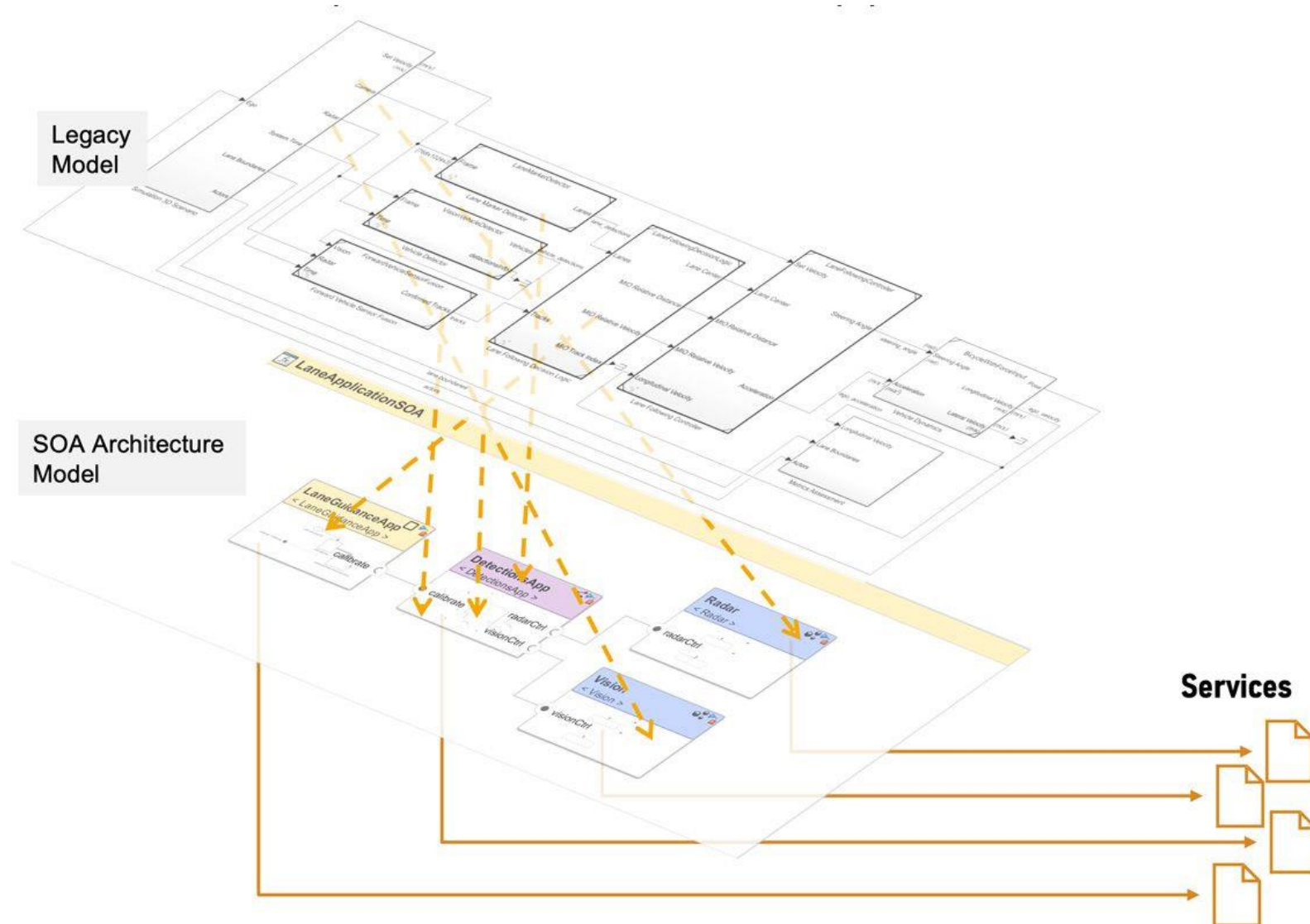


- No/minimal SW reuse
- High SW-HW coupling
- Monolithic update

SOA (Service Oriented Architecture)

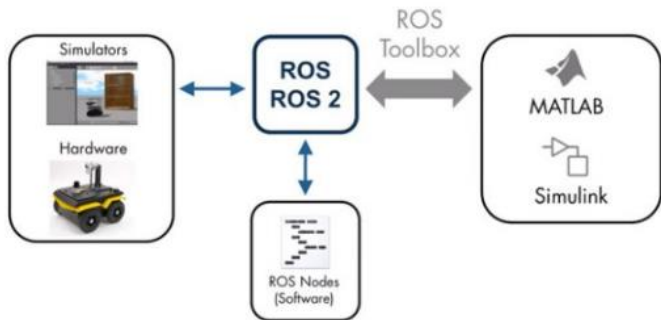


Migration from Legacy Model to SOA Architecture Model



Middleware Support

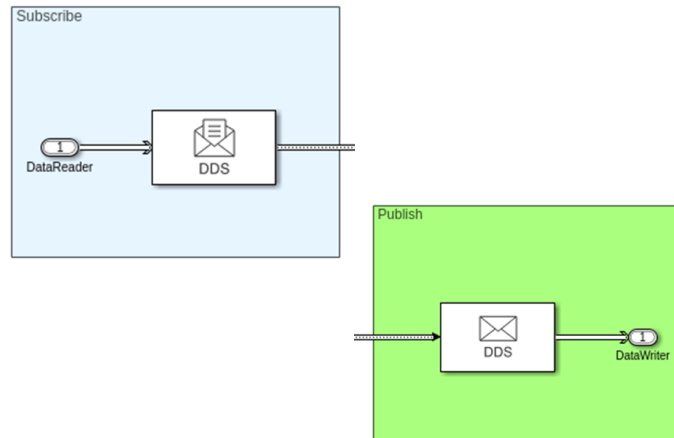
ROS / ROS2



[Get Started with ROS Toolbox](#)
ROS Toolbox

R2019b

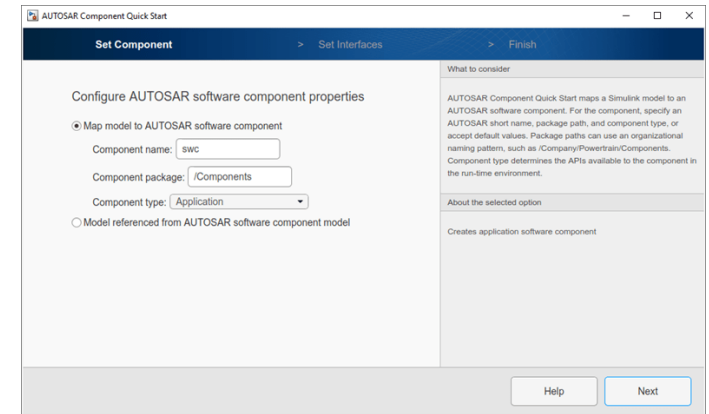
DDS



[DDS Blockset Shapes Demo](#)
DDS Blockset

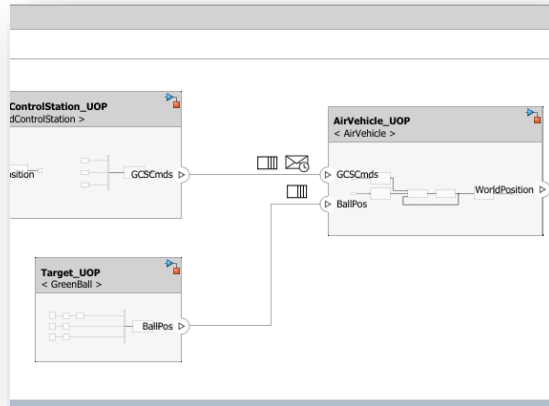
R2021a

AUTOSAR

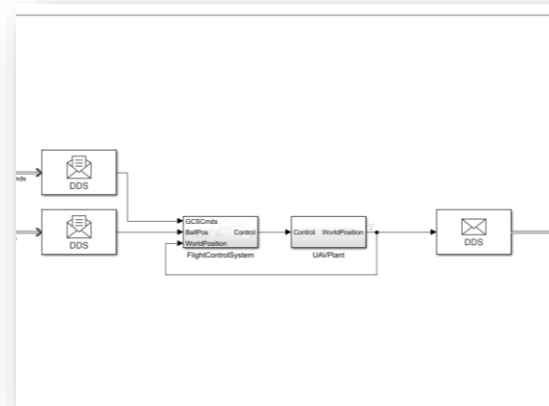


[Create and Configure AUTOSAR Software Component](#)
AUTOSAR Blockset

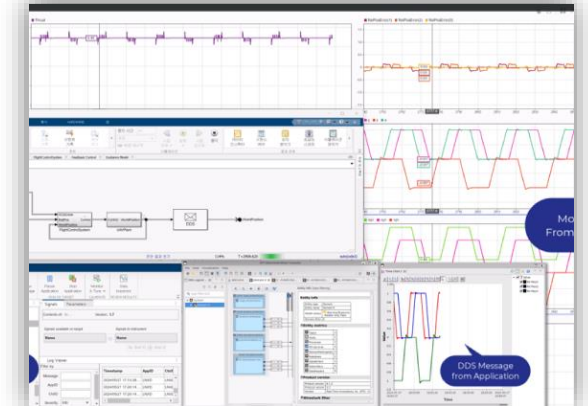
From System Architecture to deployment of DDS Application



Architecture



DDS Model & Simulation



Deployment & Monitoring

System Composer

System-Level Simulation - Quadcopter

The image displays the MATLAB Simulink environment for a Quadcopter simulation. The main workspace shows a system architecture diagram with components like GroundControlStation, Vehicle, and GreenBall. A blue arrow labeled "Mode" points to a "To Video Display" window showing a 3D rendering of the quadcopter. A green cloud labeled "Tracking View" points to a VR window showing a first-person view of the quadcopter. A purple cloud labeled "Quadcopter" points to the 3D rendering window. An orange cloud labeled "System Architecture" points to the main workspace diagram.

System Architecture

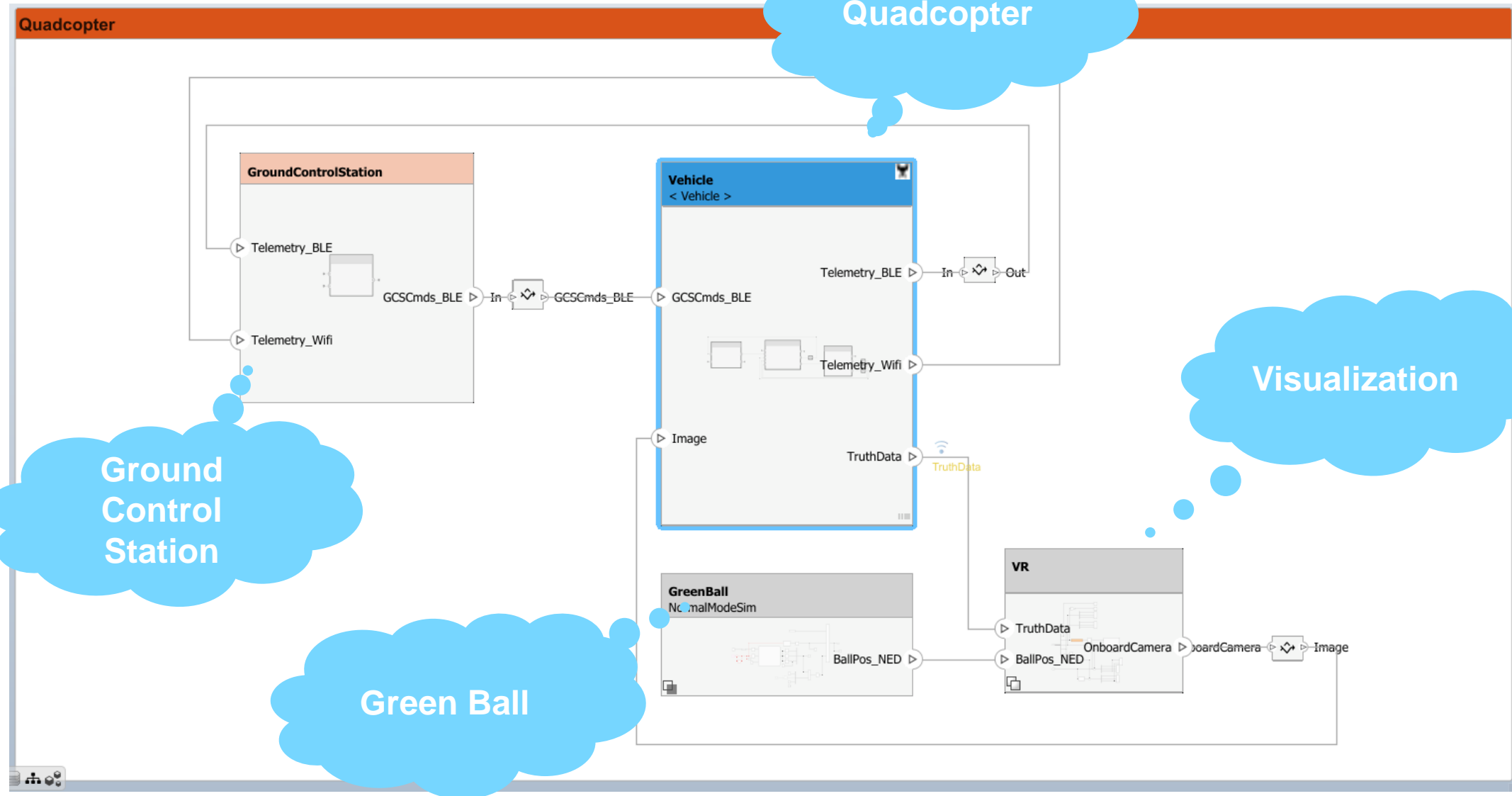
Quadcopter

Tracking View

Mode

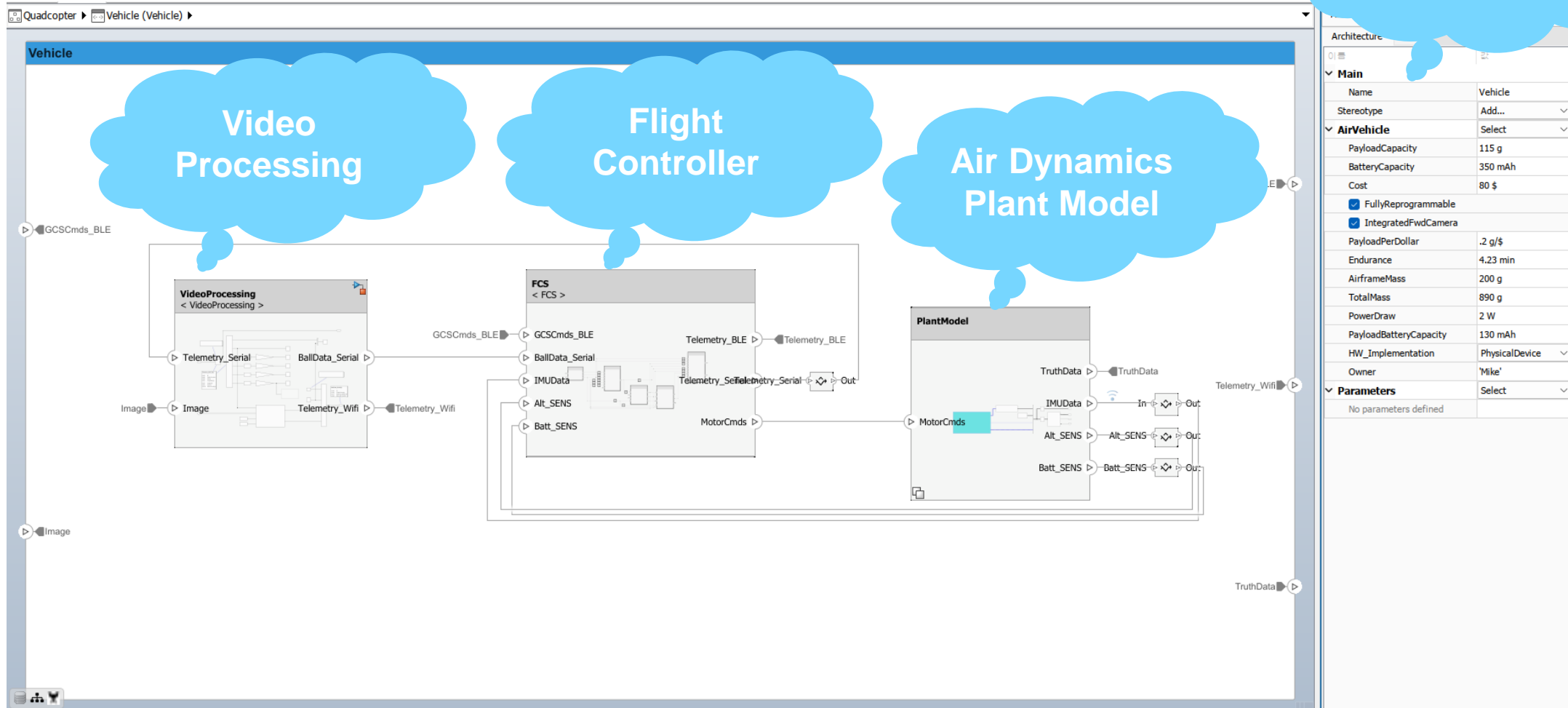
Quadcopter

System Architecture

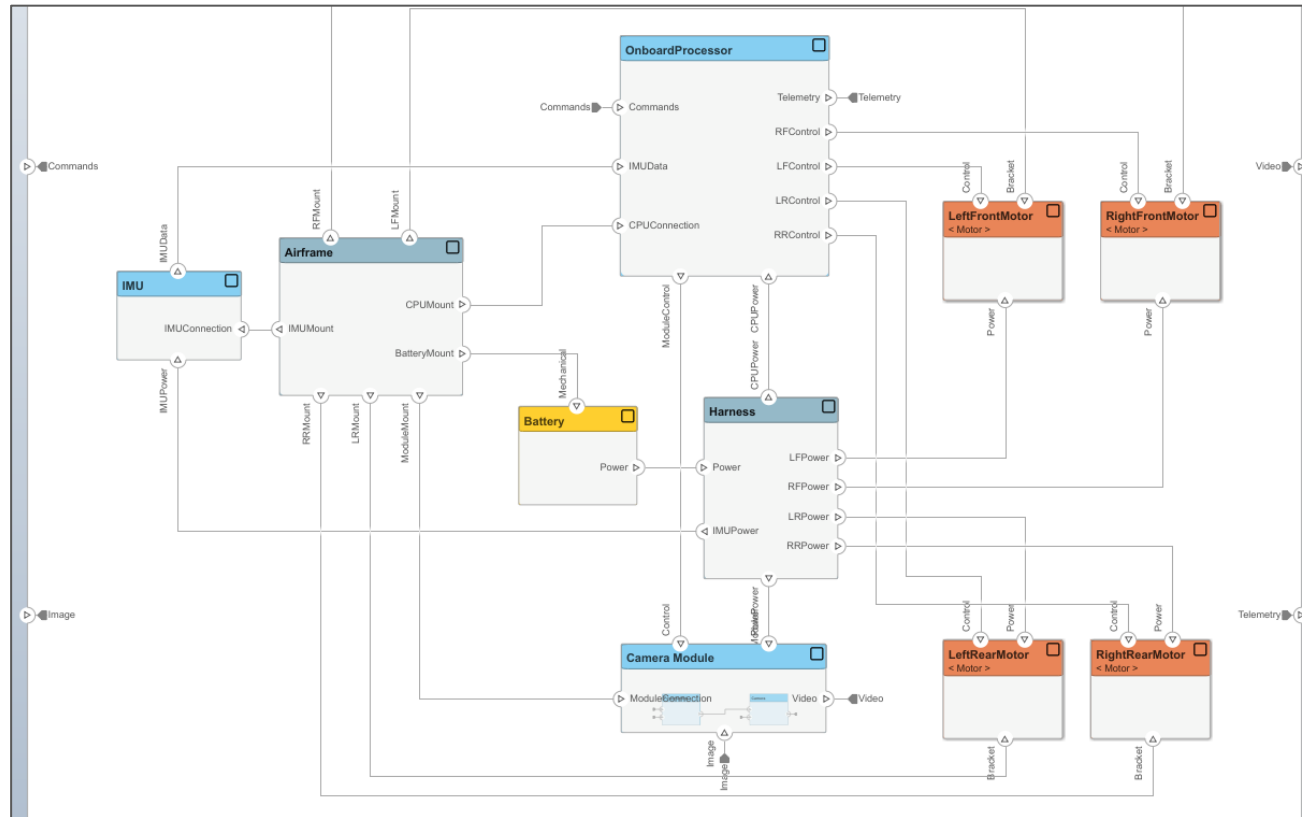


Sub-system Architecture - Vehicle

Component Property



Sketch Architecture & Elaborate Incrementally



Sketch system interfaces
and elaborate
incrementally

Sketch Architecture & Elaborate Incrementally

Trace to requirements and refine requirements alongside the architecture

The diagram shows a system architecture with the following components and their relationships:

- IMU**: Provides *IMUData* to **Airframe** and receives *IMUPower* from **Airframe**.
- Airframe**: Contains *IMUMount*, *RRMount*, *LRMount*, and *ModuleMount*. It provides *CPUPower* to **Harness** and receives *BatteryPower* from **Battery**.
- Battery**: Provides *Power* to **Harness** and receives *Mechanical* from **Airframe**.
- Harness**: Provides *CPUPower* to **Airframe** and receives *IMUPower* from **IMU**.

Requirements are implemented as follows:

- #9: Total volume** is implemented by the **IMU** component.
- #6: Power Source** is implemented by the **Battery** component.
- #10: Total Mass** is implemented by the **Airframe** component.

Requirements - Quadcopter

View: Requirements

Index	ID	Summary	Implemented
quadcopter			
1	#1	Aircraft Performance	<input type="checkbox"/>
1.1	#14	Aircraft horizontal velocity	<input type="checkbox"/>
1.2	#15	Aircraft vertical velocity	<input type="checkbox"/>
2	#2	Power System	<input type="checkbox"/>
2.1	#6	Power Source	<input type="checkbox"/>

Index	ID	Summary	Implemented	
>	4	25	Calibrate the Sensors	
>	10	50	Crash	
>	2	16	Establish Communications	
>	3	21	Initialization	
>	11	#56	Justifications	
>	9	46	Land	
>	9.1	43	Enter Land from Lost Ball	
>	9.2	47	Enter Land from Track Altitude	
>	9.3	47	Enter Land from Track Altitude and ...	
>	8	41	Lost Ball	
>	1	3	Modes of Operation	
>	5	29	Ready for Flight	
>	6	33	Track Altitude	
>	7	37	Track Altitude and Position	

Summary: Enter Land from Track Altitude and Position

Description Rationale

The Land mode shall be entered from the Track Altitude and Position mode when commanded by the ground station or if the battery voltage goes below 2.0 volts or communications with the Ground Control station are lost.

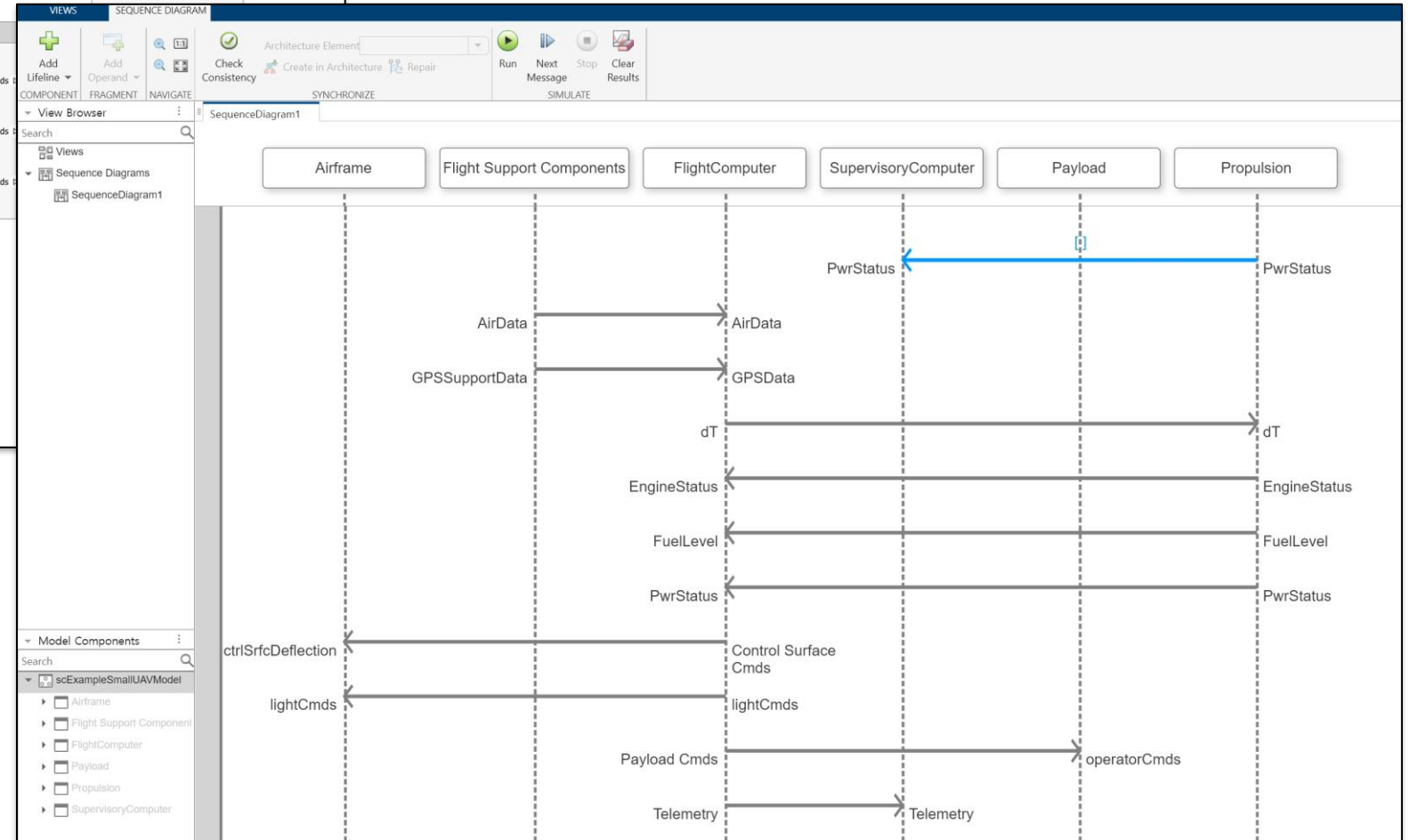
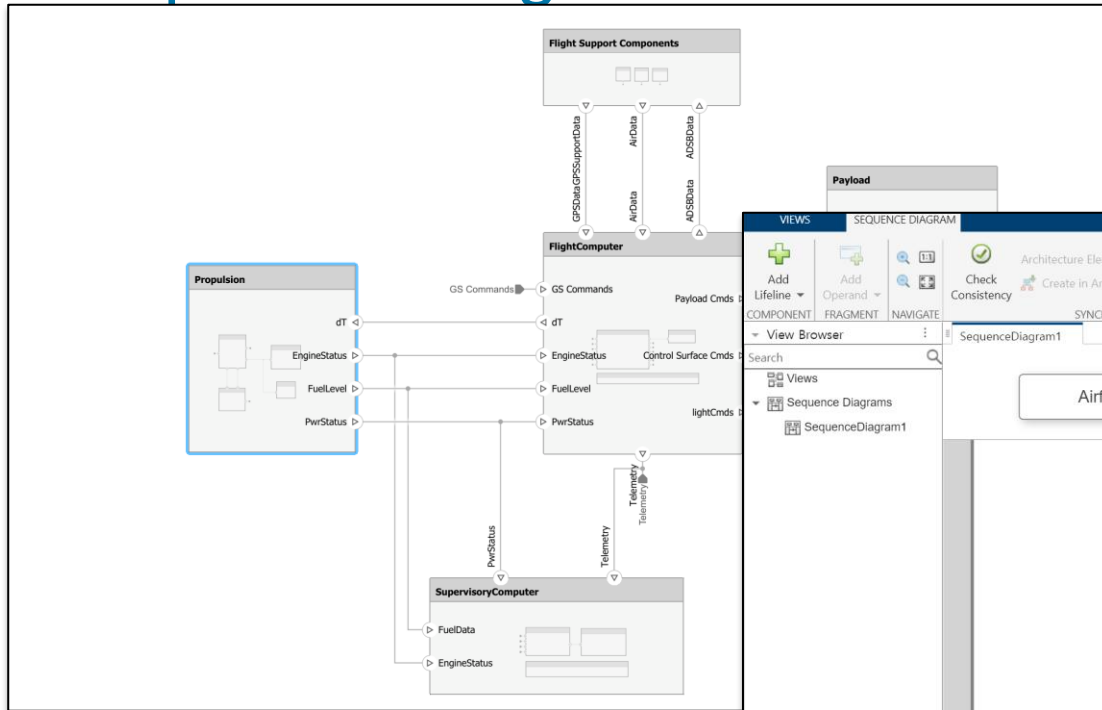
Keywords:

Revision information:

Links

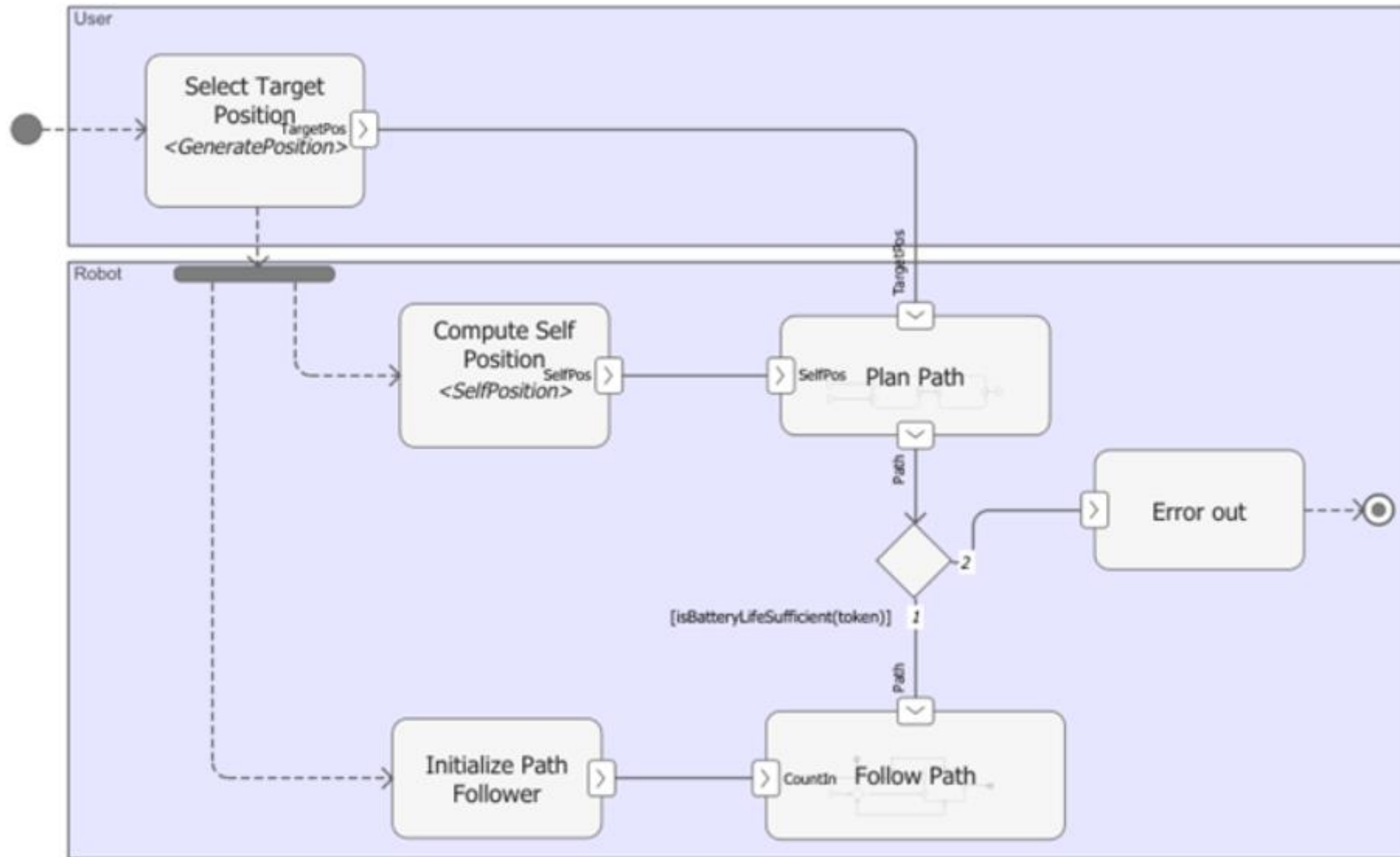
- Derived from: Autonomous Landing
- Implemented by: Land
- (GCSComds.GCS_MissionMode == vint8(2))
- (GoodComms)...
- Batt_SENS <= single(3.2)

Sequence Diagram



Activity Diagram

R2024a



DDS Blockset

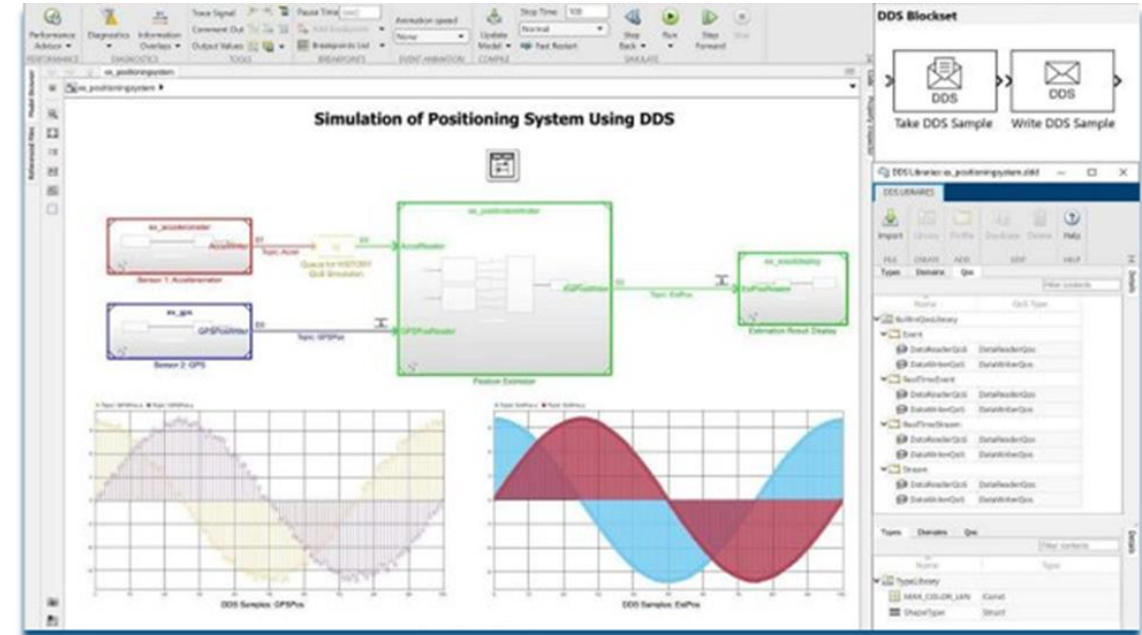
Data Distribution Services (DDS)



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.



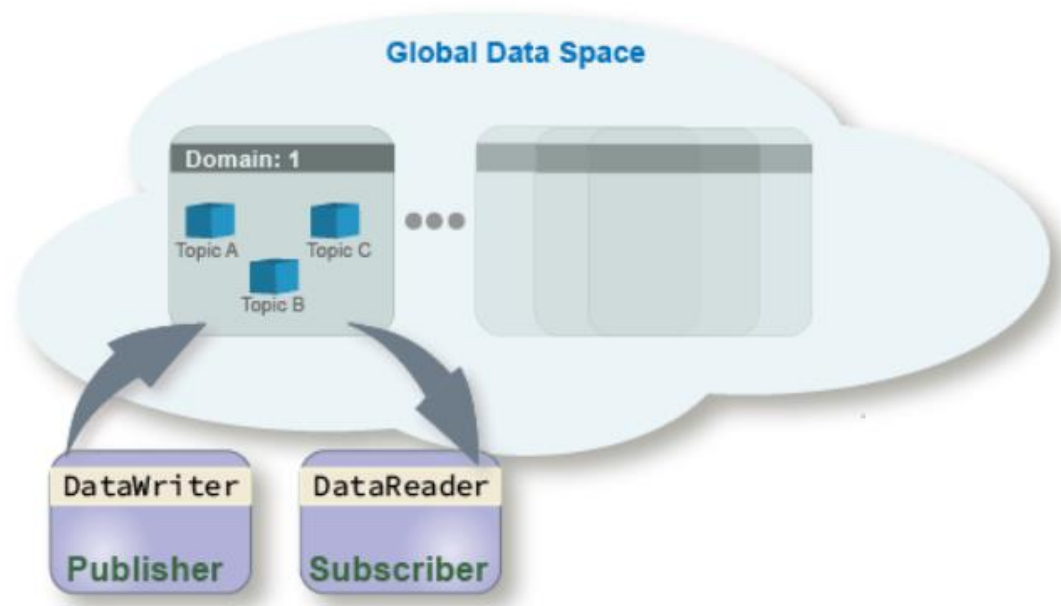
R2021a

DDS Blockset

Design and simulate DDS applications

[Request a trial](#)

DDS : Publishers and Subscribers



- Publishers : Applications that send data
- Subscribers : Applications that receive data

DDS Blockset

FOR	System and algorithm engineers
WHO	Develop software for DDS (Data Distribution Service) based embedded systems
Provides	<ul style="list-style-type: none"> • Apps and blocks to model and simulate DDS software applications • DDS dictionary to manage DDS definitions • API to Import and Export DDS definitions • C++ production code generation with DDS APIs (with Embedded Coder)
<p>DDS Blockset fully integrates with third-party DDS stacks including RTI Connex and eProsima Fast DDS</p>	

DDS Blockset is supported for all platforms - Mac, Windows and Linux

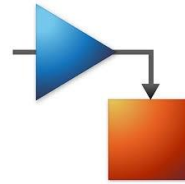
DDS Blockset

Design and simulate DDS applications

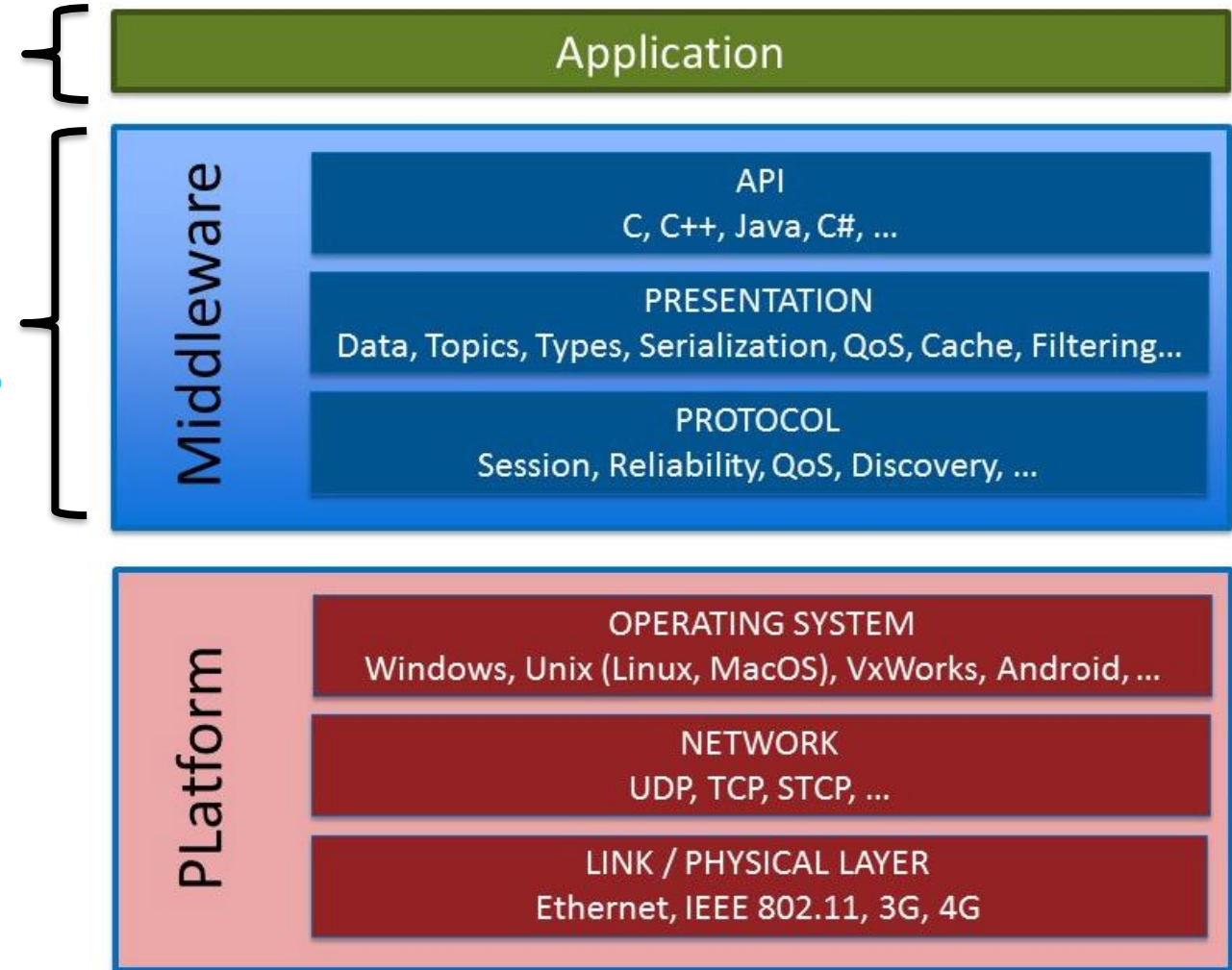
[Request a trial](#)

DDS Layered Architecture

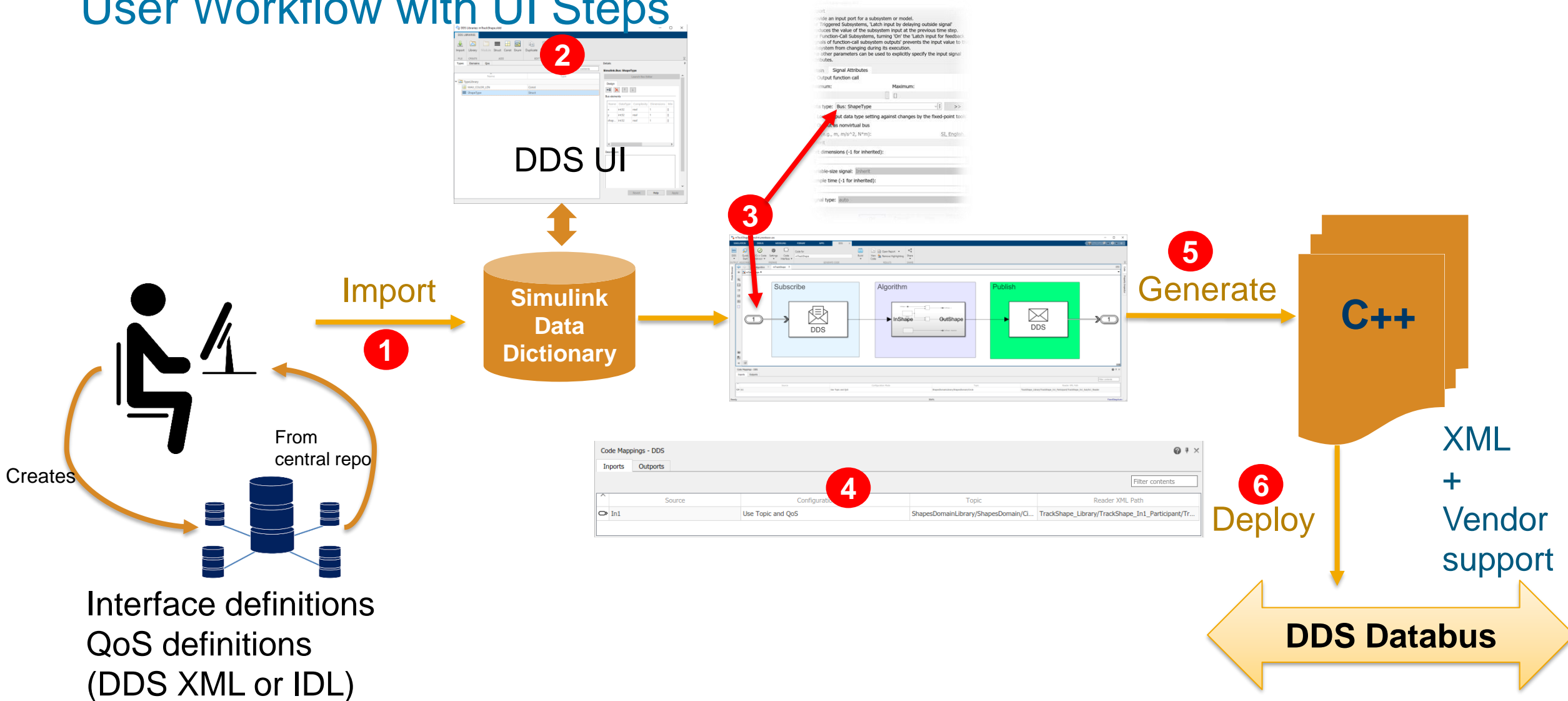
- DDS implemented in 3rd party libraries
- Application generated from Simulink model
- Code from Simulink model links to 3rd party DDS Libraries
- Open source and paid implementations



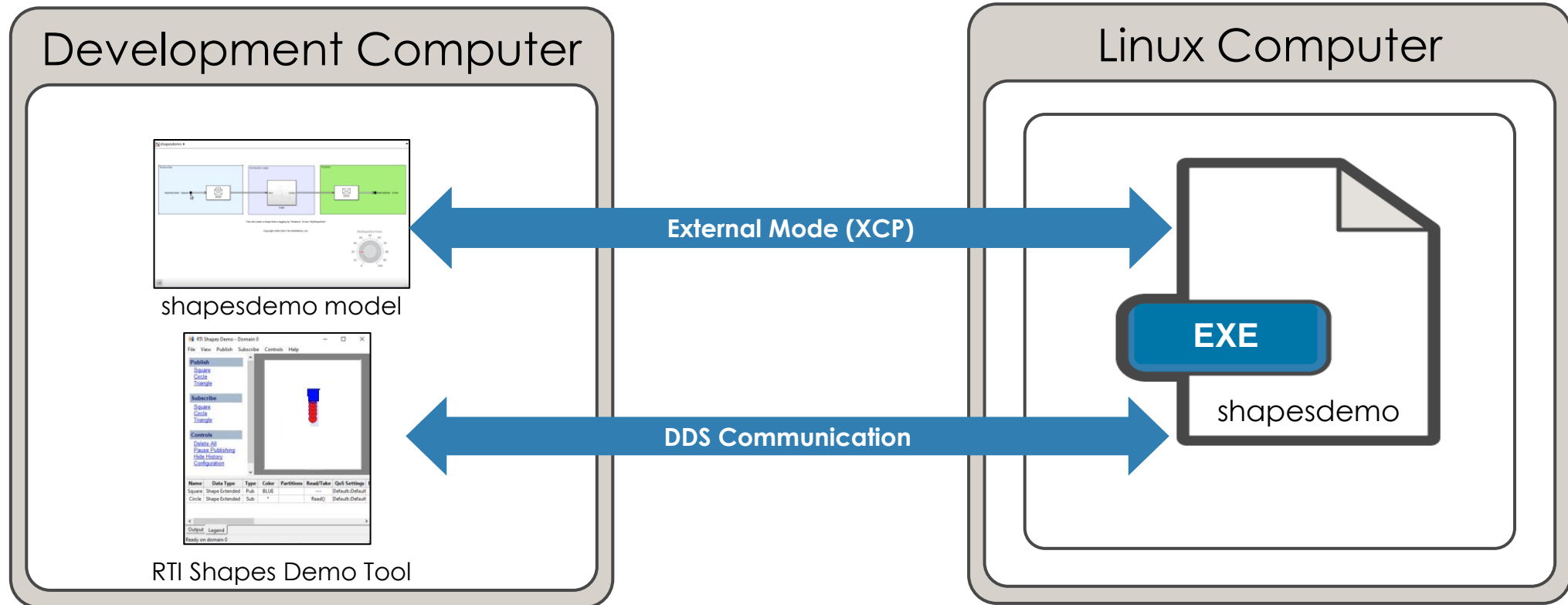
DDS
Libraries



User Workflow with UI Steps



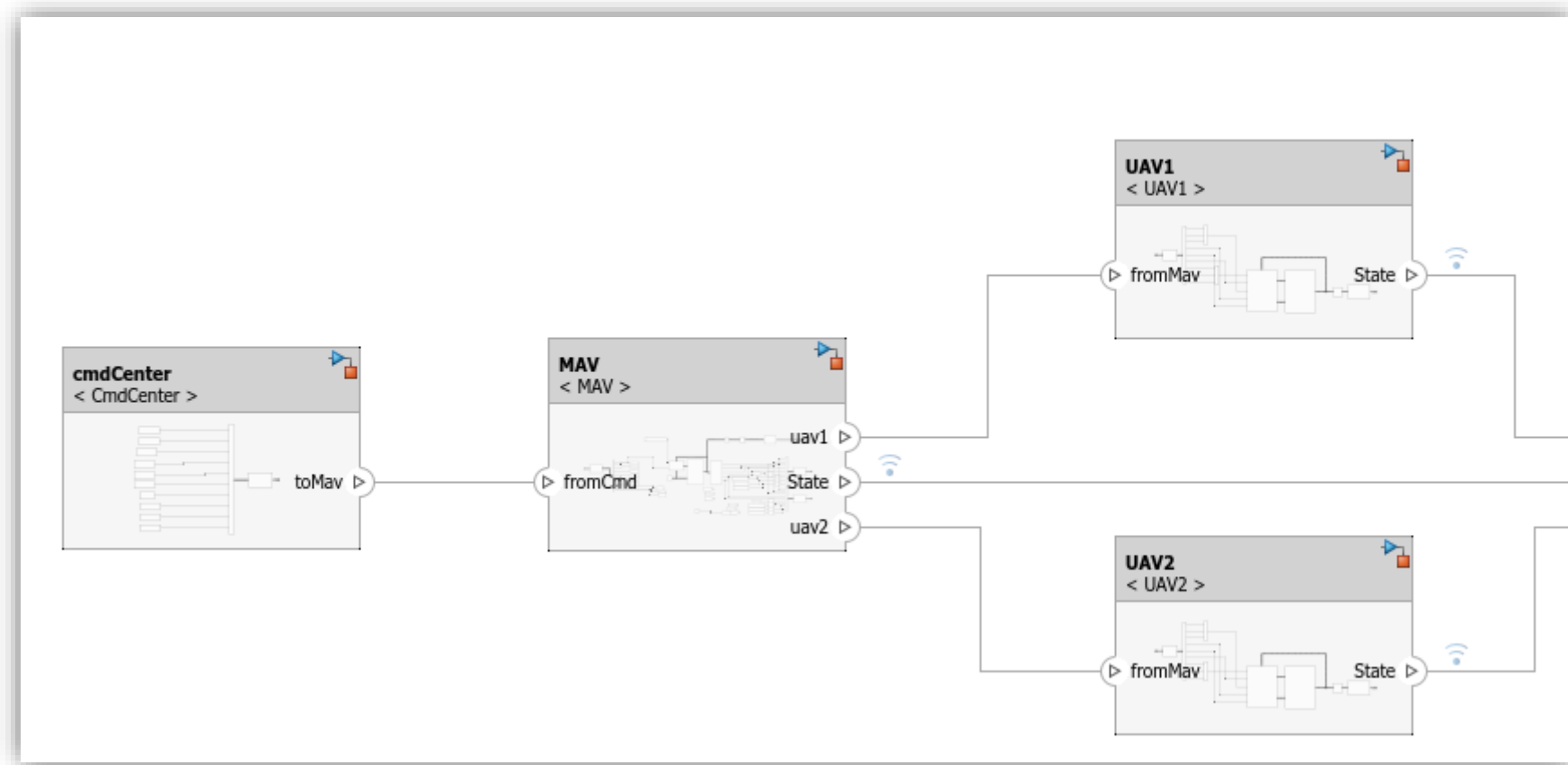
Deploy DDS Application to Linux Target



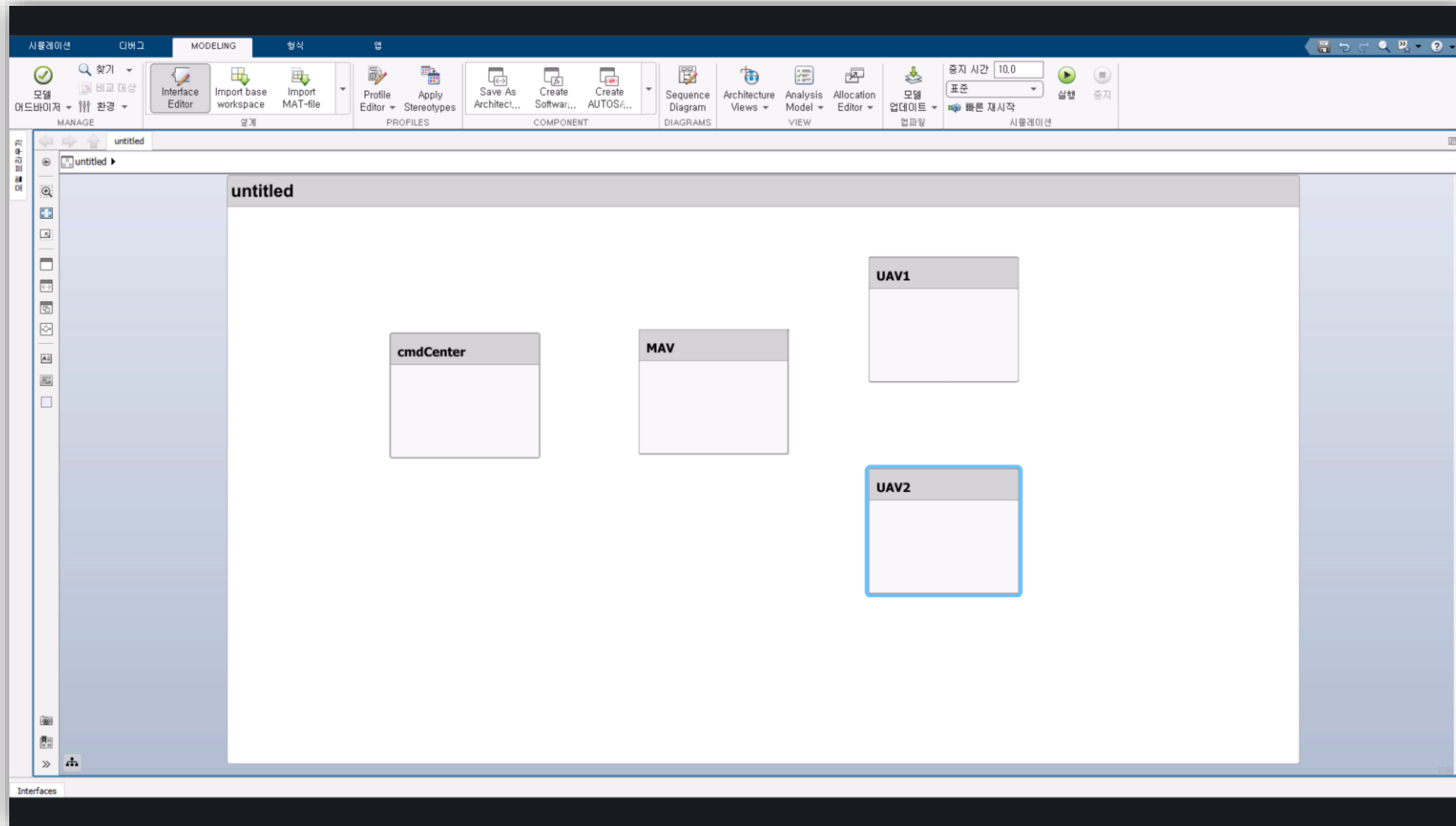
Embedded Coder Support Package for Linux Applications

DEMO

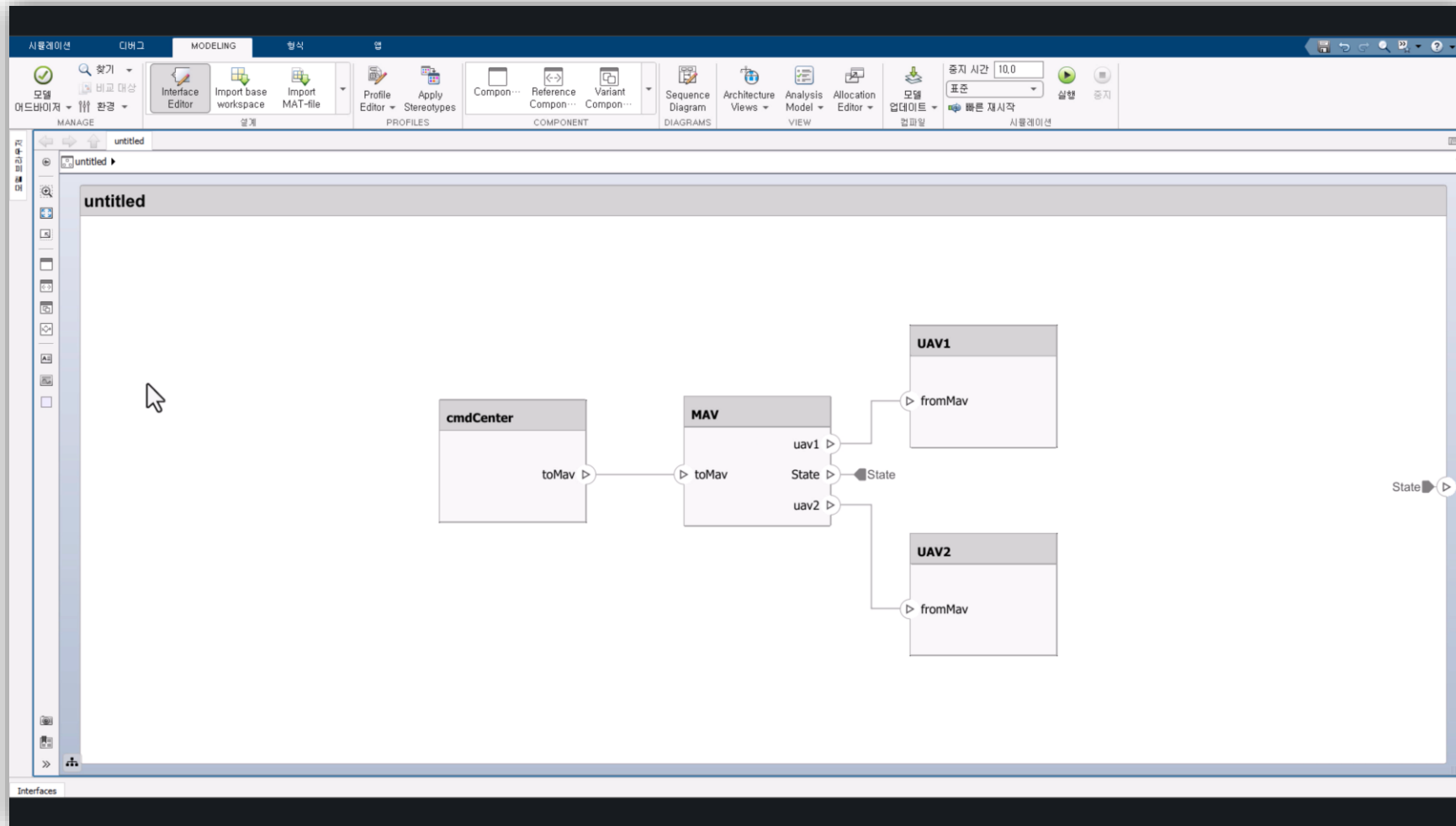
System Architecture of Multiple Aircraft Systems



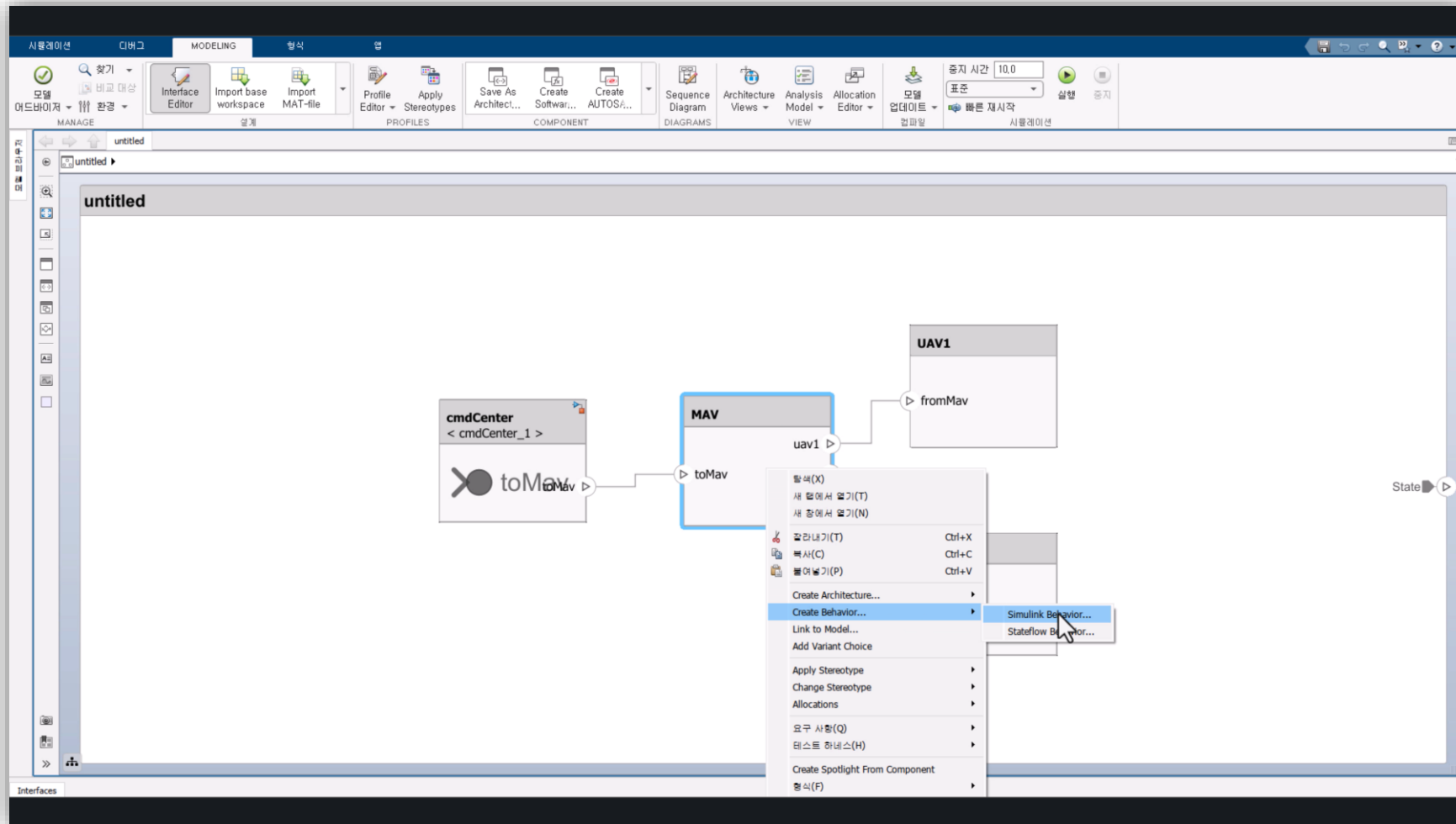
Architecture : Component Creation



Architecture : Port and Connection

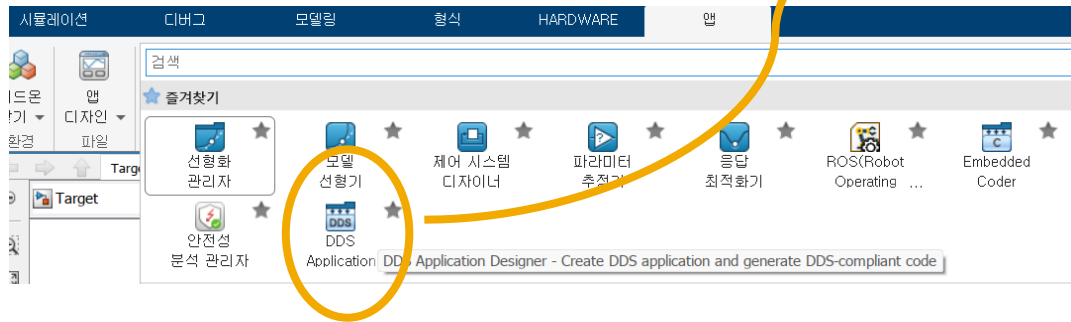


Component : Behavior Model Creation

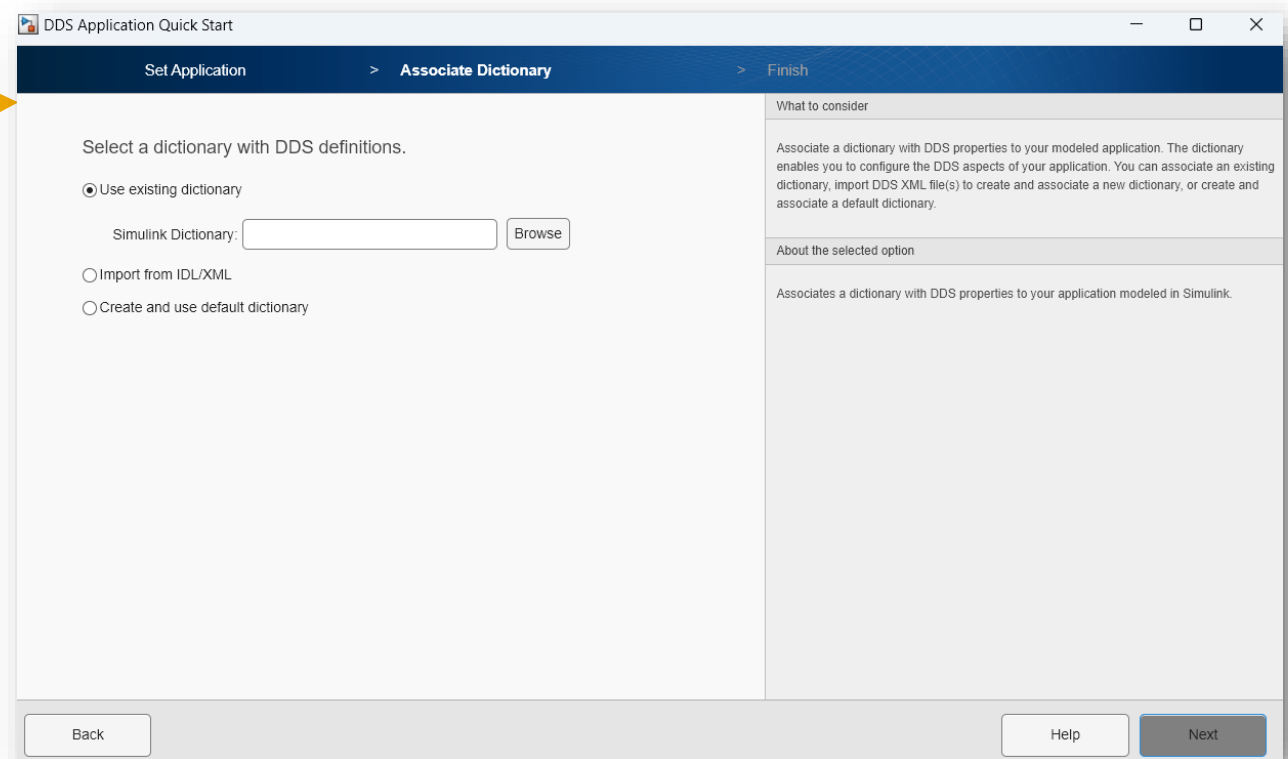


Component : Import DDS Interface definitions

- Import DDS definitions from XML or create new Definitions

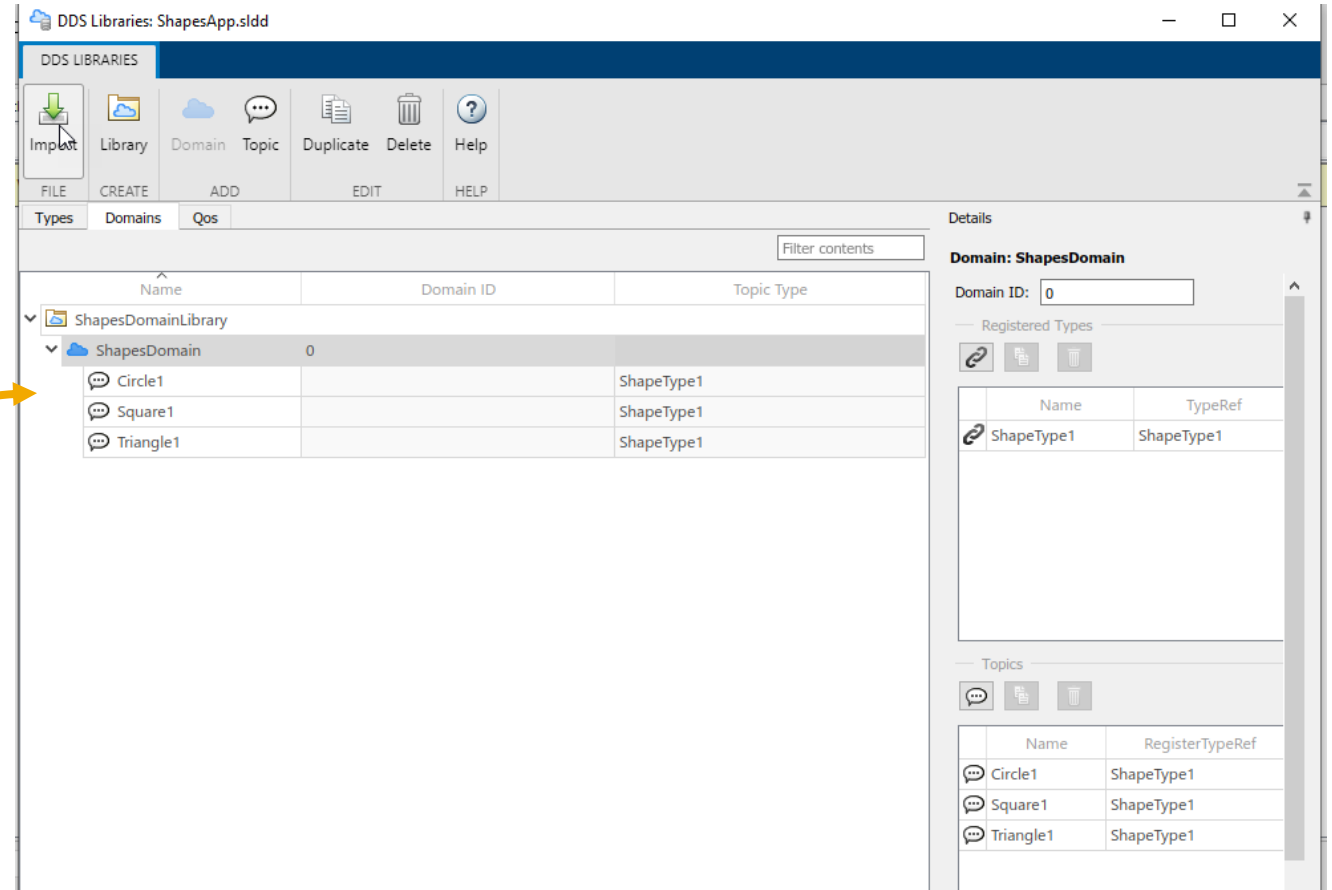
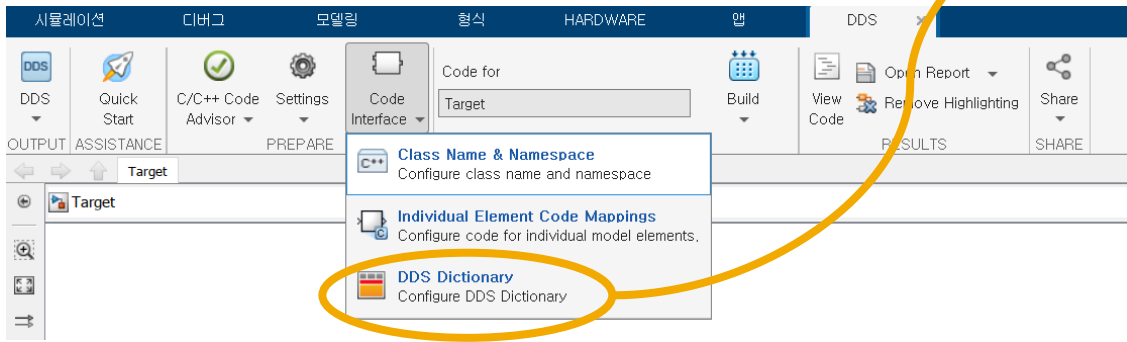


DDS Application Designer



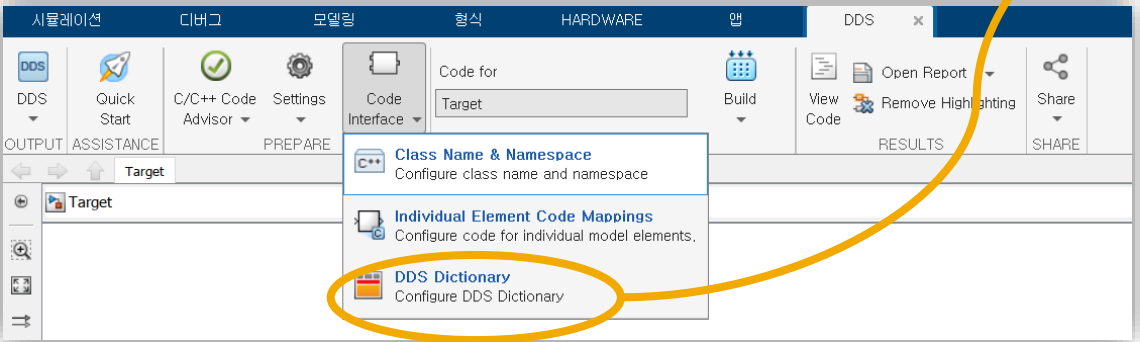
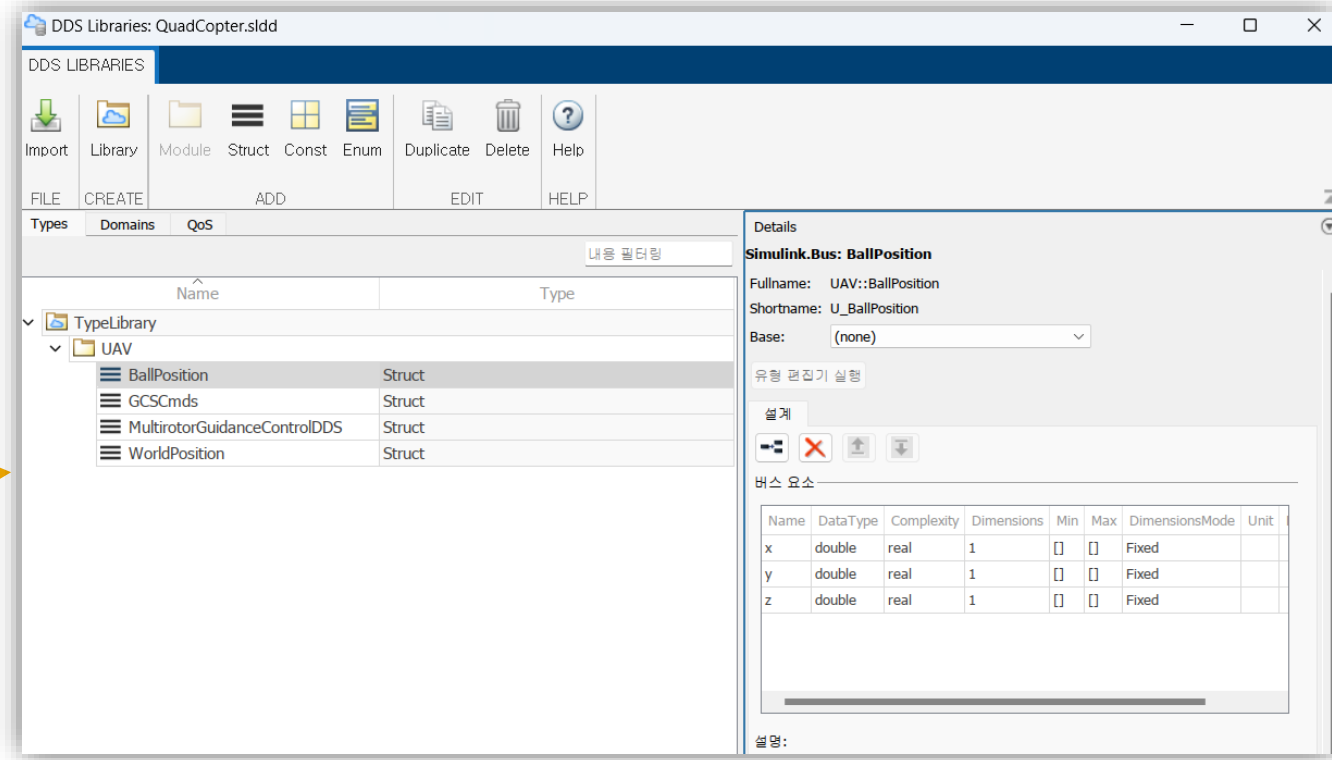
Component : Import DDS Interface definitions

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

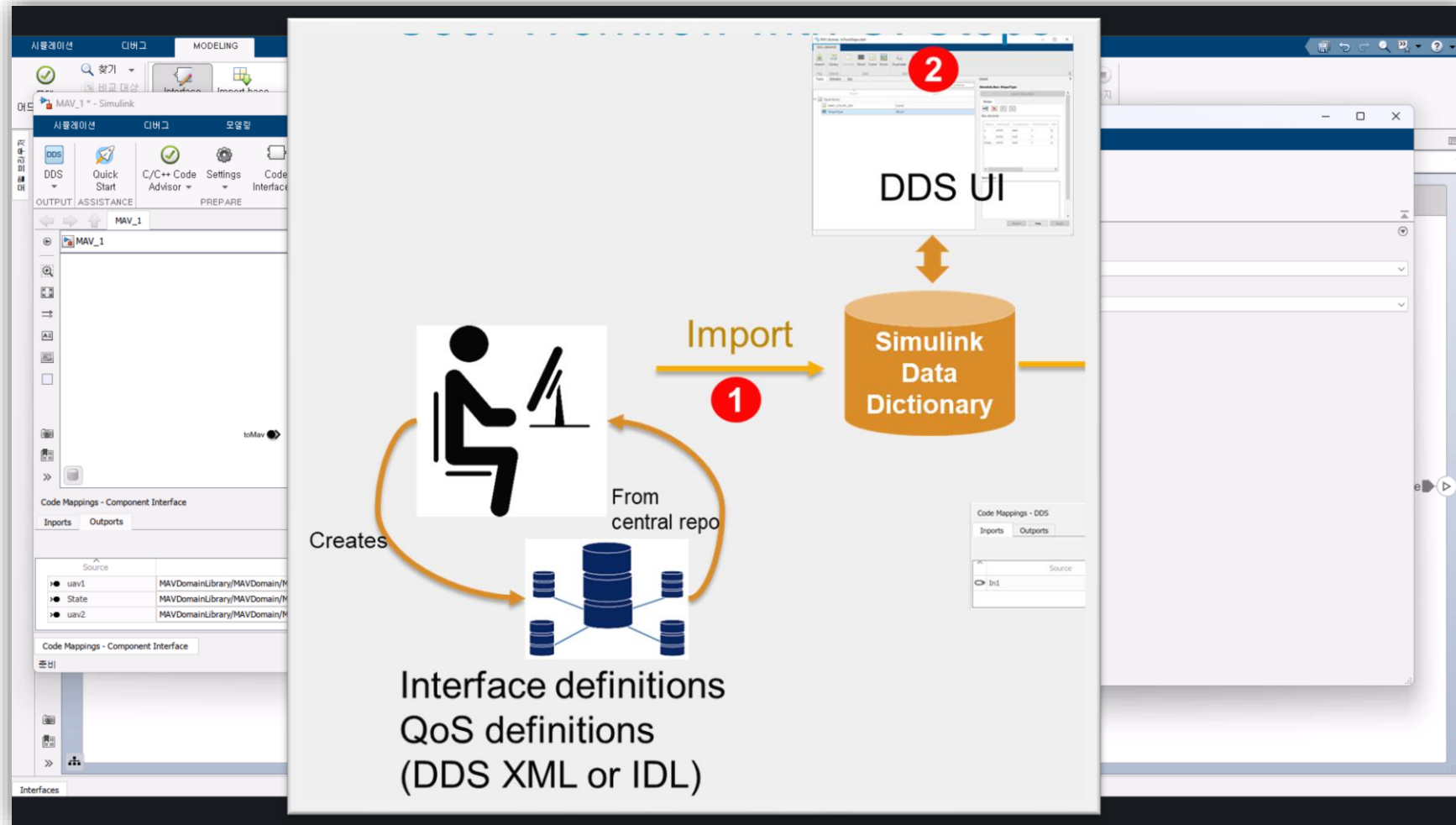


Component : Import DDS Interface definitions

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



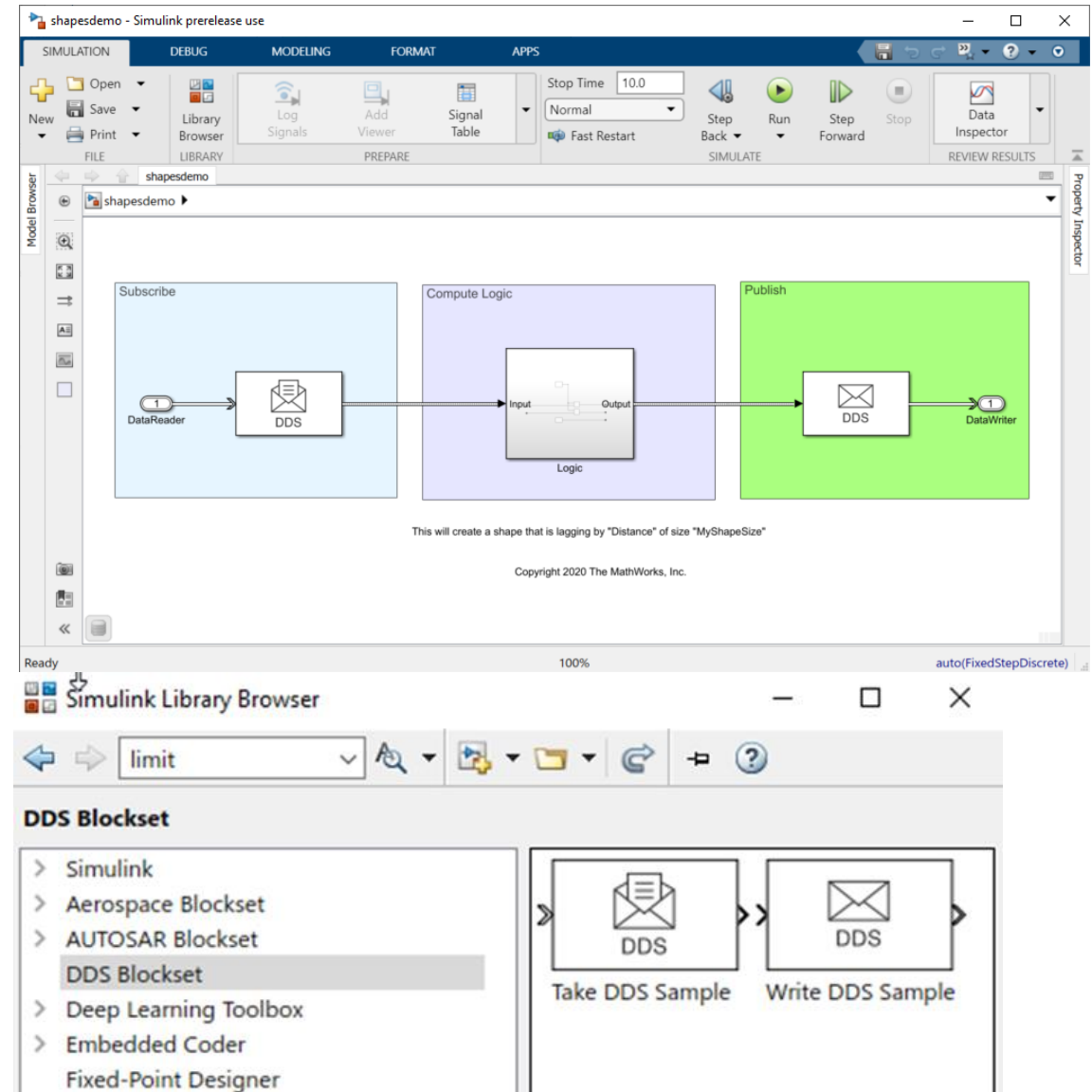
Component : Import DDS Interface definitions



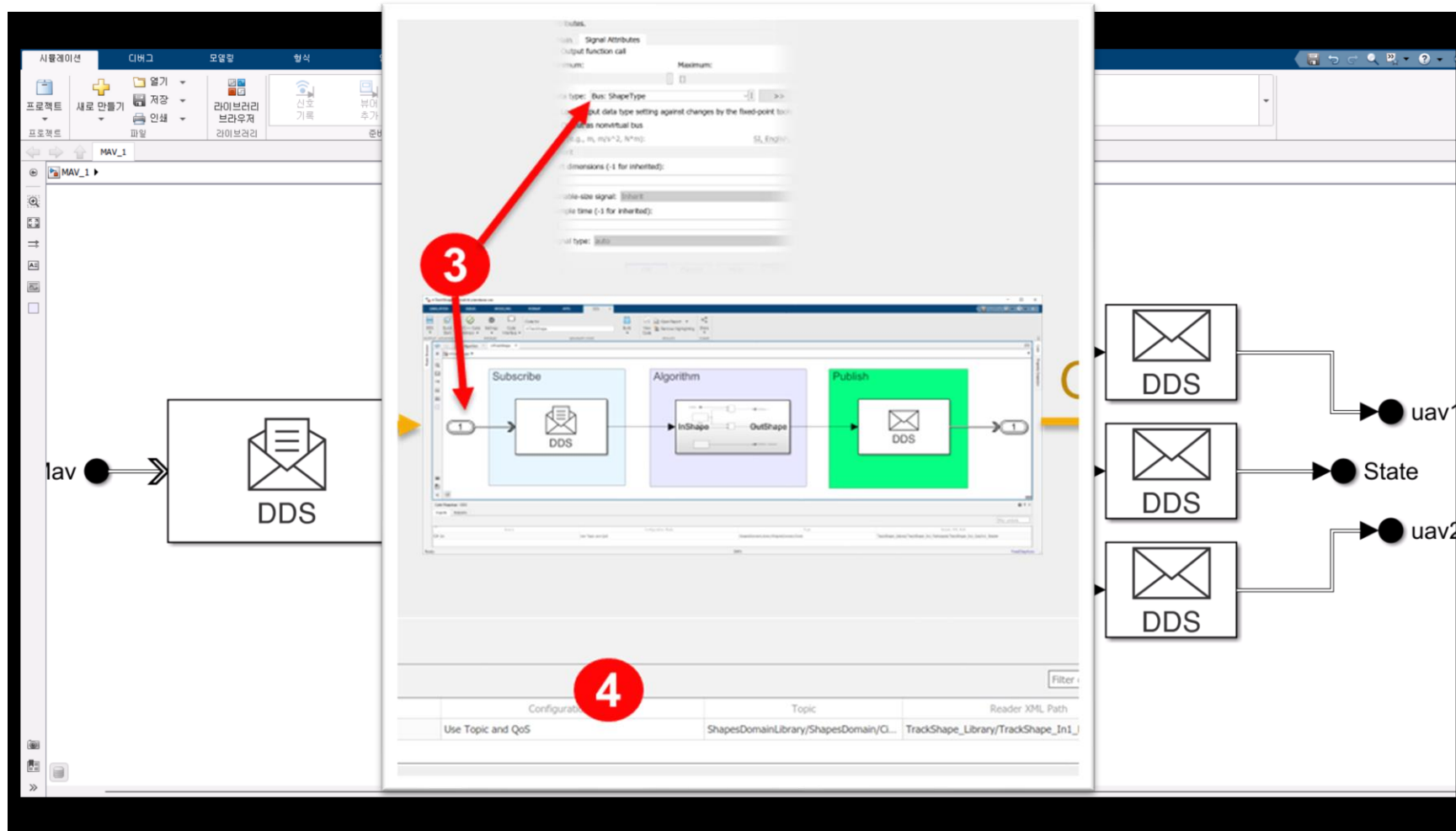
Component : Model DDS Application

- 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



Component : Model DDS Application



Component : Deployment of DDS Application

- 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

```
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);
};

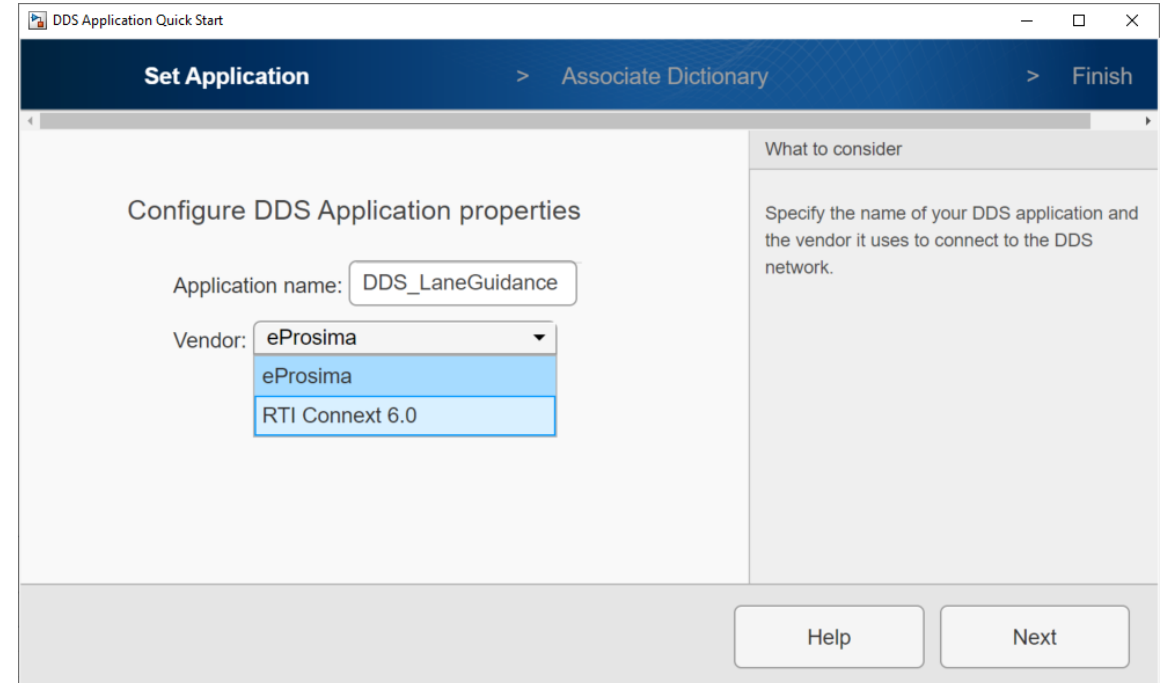
bool createParticipant(std::string participantName) {
    if (participants.find(participantName) == participants.end()) {
        DDS_DomainParticipant* participant =
            DDS_DomainParticipantFactory_create_participant_from_config(
                DDS_TheParticipantFactory, participantName.c_str());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

With Embedded coder, generate

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

Component : Deployment of DDS Application

- 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



Full integration with third-party DDS stacks including RTI Connnext and eProsima Fast DDS

Component : Deployment DDS Applications (Single Application)

5 Generate

C++

6 Deploy

XML + Vendor support

DDS Databus

Signals from Simulink

North	East	Height	publication_handle	instance_state
5561.258465814111	4454.477086612404	-16.041399999999943	01011fcd.fb5dc5f...	

Signal from DDS Message

Value

Time

In live mode

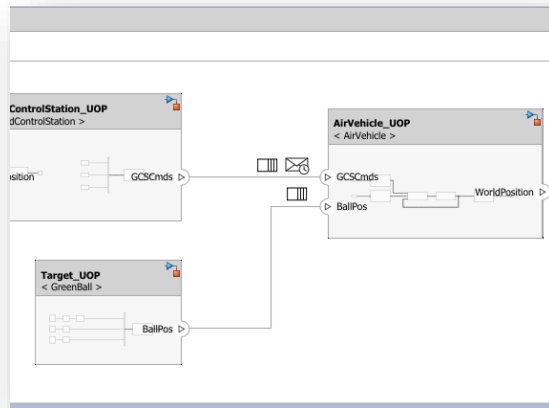
System : Deployment of DDS Applications

The screenshot displays the MATLAB Simulink environment for a system deployment. It is divided into several key sections:

- System Architecture:** A block diagram showing the interconnection of components like 'cmdCenter', 'MAV', 'UAV1', 'UAV2', and 'Visualization'.
- Signal from Simulink:** A plot showing time-series data for variables such as '<North>', '<East>', and '<Height>'.
- DDS Domain:** A window showing the configuration of a DDS domain with various topics and endpoints.
- Deployed DDS Applications:** A 'Linux Target' window showing the deployment of applications to a target system.
- Signal from DDS Message:** A plot showing a received DDS message value over time, with a value fluctuating around 36.1.

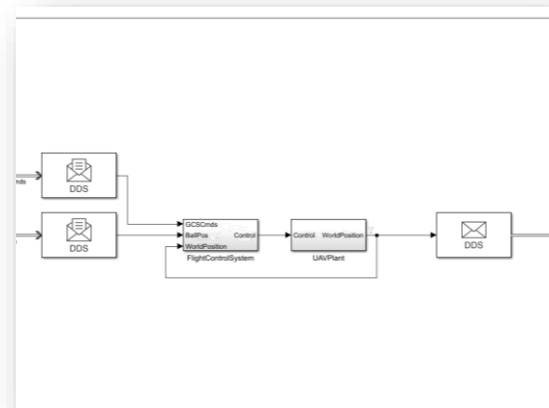
Timestamp	AppID	CtxID	Type	Message
2024/06/05 19:58:14	LNXD	LNXD	info	[ACTION: Terminating proc...
2024/06/05 20:00:50	LNXD	LNXD	info	[ACTION: Launching proce...
2024/06/05 20:00:50	LNXD	LNXD	info	[Process launch initiated: V...

Key Takeaways



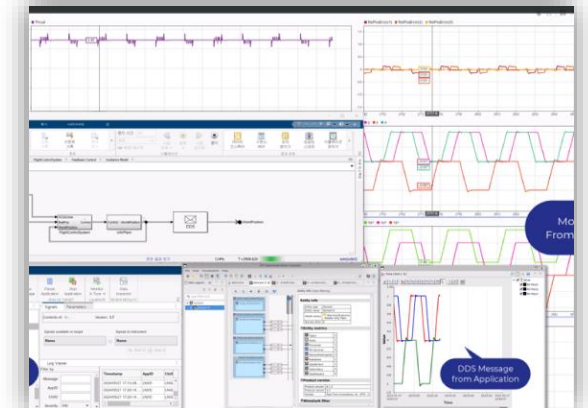
Architecture

System of Systems architectures are evolving, pushed by need for advanced, complex functions



DDS Model & Simulation

New, **service-oriented architectures** are required to **master complexity** and enable **frequent updates**



Deployment & Monitoring

You can **design, simulate and generate code** to deploy service-oriented applications in **Simulink, reusing your existing expertise and models**

DEMO Booth

MATLAB EXPO

DDS Blockset을 활용한 무인기
분산 시뮬레이션

유성재, 매스웍스코리아



MATLAB EXPO

고정익 UAV의 경로 추종 비행
시뮬레이션

한재훈, 매스웍스코리아



MATLAB EXPO



© 2024 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.

