SE322 - Software Architecture

Project - Phase 1

2021-22 Spring

MEA
19244710069 – Mert Eren DAĞISTAN
18244710043 – Muhammed Alperen HARMANBAŞI
24.03.2022

TABLE OF CONTENTS

1 The Example System

1.1 Project name and description

UAV Technologies. This modelled system is an unmanned aerial vehicle model (UAV). This system includes both software and hardware. An unmanned aerial vehicle is a type of flying vehicle that does not physically contain humans. The most important and indispensable component of UAVs is a communication system between a ground-based controller and the aircraft.

1.2 List of components

1.2.1 Execution Platform Components

Camera – Device: It receives and transmits a data connection. So, it transmits snapshots.

GPS Receiver – Device: It shares the coordinate information (position) of the UAV, in short, its instantaneous position.

Radio – Device: It receives and transmits audio data. It provides voice communication.

UART – Device: It is a computer hardware device for serial communication. It needs the bus.

Wi-Fi – Device: It provides communication in UAVs.

MC_Proc – Processor: Mc Proc is the mission computer processor. It requires bus access.

GS_Proc – Processor: It is the processor used for the Ground Station communication system. And it needs a bus.

MC_Mem – Memory: Mission Computer's random-access memory. Also, it requires bus access.

GS_Mem – Memory: Memory for Ground Station communication system.

GS_Bus – Bus: Provides data transmission to UAVs in ground station communication system.

MC_Bus – Bus: It is the system that provides data flow with the mission computer in the UAVs.

RF_Bus – Bus: These are the buses that provide the transmission of the radio frequencies of the UAVs.

Serial_Bus – Bus: UAV onboard serial bus.

Wifi_Bus – Bus: The device that provides the wifi connection of UAVs.

1.2.2 Application System Components

GS_SW – Process: It is the software that enables the communication of the Ground Station.

SW – Process: The Mission Computer Software process runs all threads. These are data transmitted from and sent to components such as radio, wifi, uart.

WifiDriver – Thread: Wifi Driver allows other software components to communicate with other systems with WiFi connection.

RadioDriver – Thread: RadioDriver allows other software components to communicate with other systems with Radio frequencies.

FlightPlanner – Thread: The flight planner creates a map and flight pattern of the UAVs along with commands and missions. In addition, it can access the locations of regions that do not have a flight permit.

NoFlyZoneDatabase – Thread: It is the software that enables the determination of the regions where the UAV does not have a flight permit. This allows it to avoid positions. It transmits and receives data about maps and regions.

WaypointManager – Thread: FlightController can only handle a small number of waypoints at a time, WaypointManager creates these task windows in response to the current position of the UAV provided by FlightController GPS.

CameraManager – Thread: CameraManager is the component that allows transmitting and receiving data according to the coordinate information, flight plan and gimbal commands of the UAV.

UARTDriver – Thread: A thread that acts as a mediator between software components and other systems to let them interact via serial connection.

Coordinate – Data: It is the component that contains the latitude, longitude, and altitude coordinate information for the UAV.

Map – Data: It is the structure that contains the coordinates of a region.

MapArray – Data: It is the component that holds more than one map.

FlightPattern – Data: It defines how the UAV will fly in its defined region.

Command – Data: It is the component that defines the commands transmitted to the UAV.

RF_Msg – Data: Messages transmitted via radio frequencies.

Mission – Data: The Mission is a list of waypoints that is generated by the Flight Planner based on a Map and Flight Pattern.

MissionWindow – Data: It is a list of waypoints created for the UAV.

1.3 List of data exchanged by components

Inspect data, event data or event ports, and identify the data, which software elements exchanged this data and its structure if it is provided with an implementation.

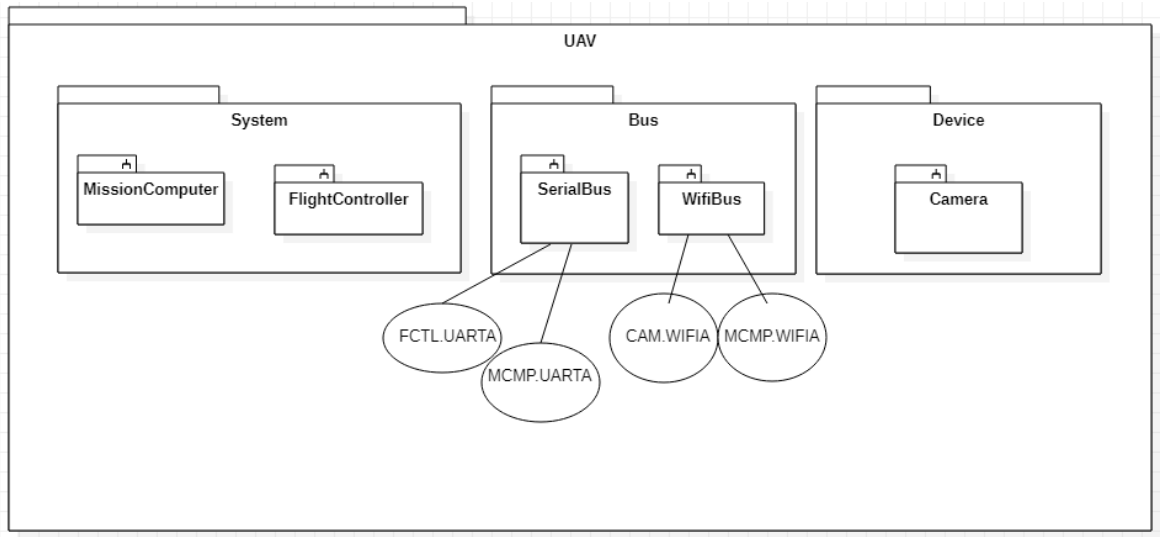
An example is given below:

- Coordinate data is shared between UARTDriver thread and WaypointManager, FlightPlanner and RadioDriver threads. It consists of latitude, longitude, and altitude in integer format.
- Map data is shared between FlightPlanner and NoFlyZoneDatabase, it consists of Coordinate elements since the basic data type is array and dimension 4.
- MapArray data is shared between FlightPlanner and NoFlyZoneDatabase. It consists of Map elements as basic type; data type is array and dimension 5.
- Mission data is shared between FlightPlanner and WaypointManager, CameraManager and WaypointManager. It consists of Coordinate as classifier, data type is array and dimension is 10.
- MissionWindow data is shared between WaypointManager and Camera Manager. It consists of Coordinate as classifier, data type is array and dimension is 4.
- Command data consists of Map data as a map and FlightPattern as a pattern.
- RF_Msg data is shared between FlightPlanner and Radio Driver. Consists of refined payload from command.

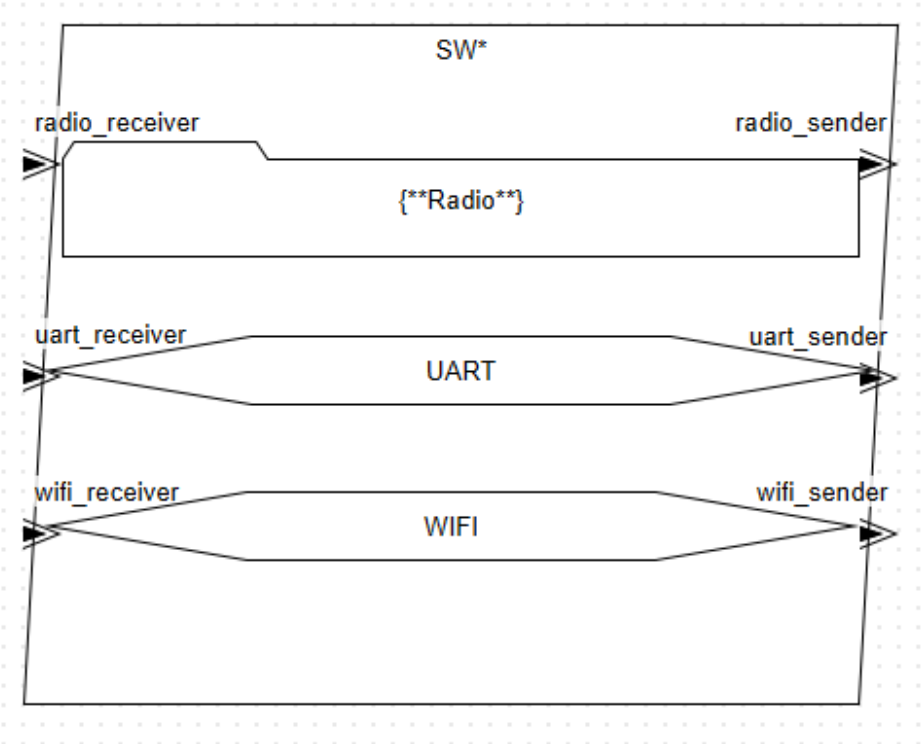
1.4 Views

1.4.1 Module View

Pick a module structure, refer to “week2 - Understanding Software Architecture” slide on moodle, and draw a diagram for **system UAV** (refer to UAV.aadl)

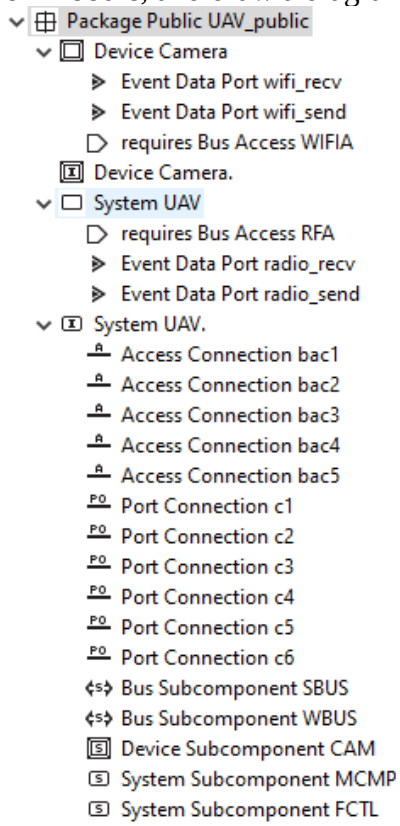


1.4.2 C&C View



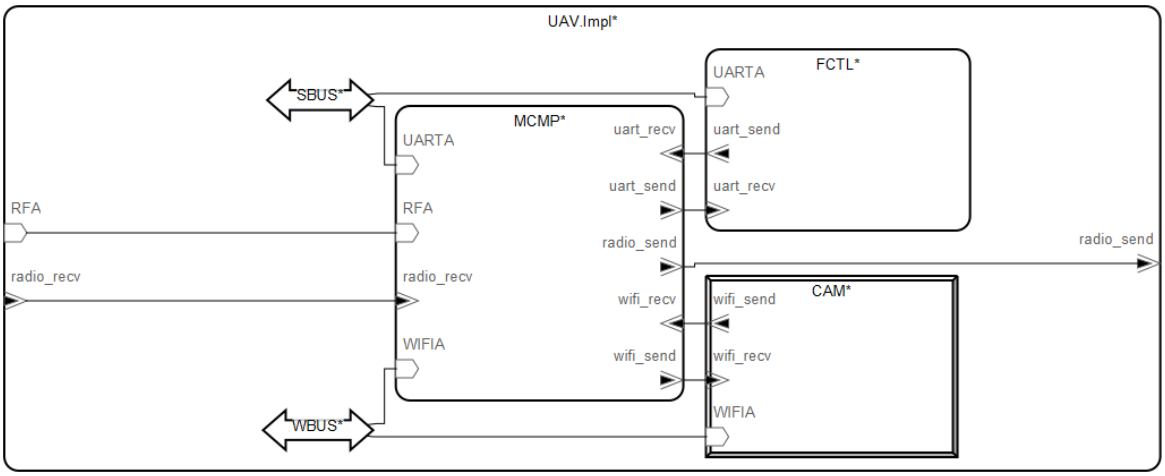
1.4.3 Allocation View

Pick a allocation structure, refer to “week2 - Understanding Software Architecture” slide on moodle, and draw a diagram for **system UAV** (refer to UAV.aadl)

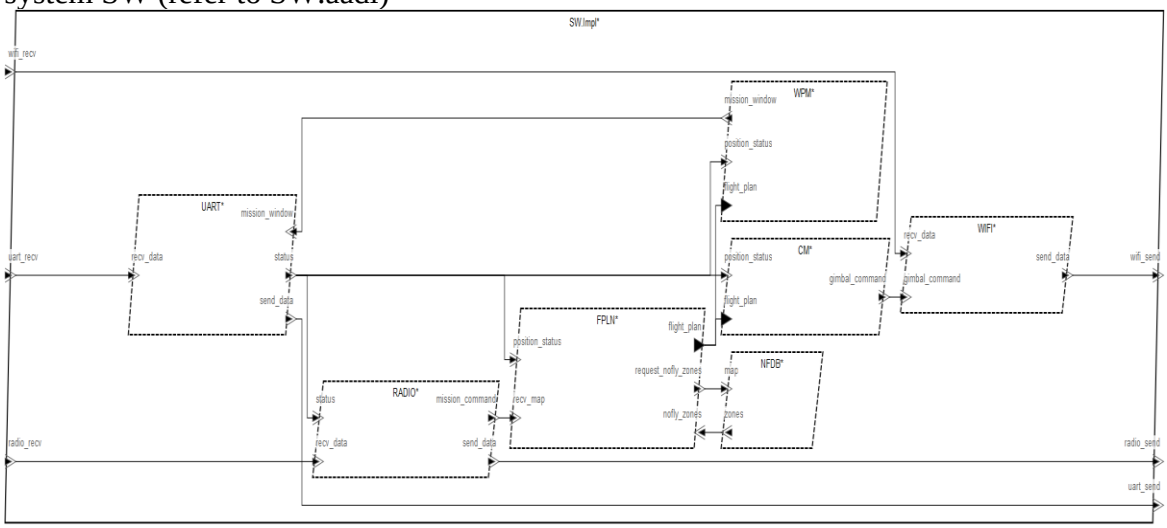


1.5 Architectural Diagrams by OSATE

(i) system UAV (refer to UAV.aadl)



(ii) system SW (refer to SW.aadl)



2 Github Repo

https://github.com/erendagstan/SE322-MEA

3 Discussion

3.1 Usability of the tool

We think OSATE is a very useful tool. And besides that, it is easy and understandable to use. We took help from sources such as YouTube and GitHub to learn how to use this tool. Also, we had no problems installing OSATE. We think it's quite easy and quick to install.

3.2 Learnability of AADL language

We think AADL has a very understandable syntax. Because when we look at the codes, we can easily understand which code snippet fulfils which task with proper naming.
In addition, there is information on many internet resources to understand and learn this language. Sites such as GitHub, YouTube, Udemy can be used.

3.3 Complexity of the given system

We consider the architecture of this given system to be uncomplicated. Of course, even if the code snippets seem complicated at first glance, when you are prone to the AADL language and examine the architecture a little, we conclude that it is not a complex structure. The reason why this system seems complicated is because there are so many elements of the system. UAVs are vehicles with very complex structures. And with that, they have many components. Therefore, the architecture of this system seems complicated. Naming, defining, describing the properties of the elements in the system and the communication between the elements are written using a simple and descriptive syntax. This makes the system understandable. (For example, we can see that the code line "position_status: in event data port Coordinate.Impl" in the thread FlightPlanner component receives data from the Coordinate data of the FlightPlanner thread.) Thanks to the naming we mentioned in this example, the readability and understandability of the code increases. This reduces the complexity of the system.