



Z. Kootbally & C. Schlenoff
Spring 2023
UMD
College Park, MD

RWA1 (v0.0)


ENPM663: Building a Manufacturing Robot Software System

Due date: **Saturday, February 25, 2023, 11 pm**

Contents

1	Updates	3
2	Conventions	3
3	Prerequisites	3
4	Assignment Tasks	5
4.1	Competitor Manual Command	7
4.2	Starting the Competition	7
4.3	Retrieving Orders	8
4.4	Submitting Orders	8
4.5	Ending the Competition	9
5	Timer Callback	9
6	Contribution	9
7	Submission	10
8	Grading Rubric	10







1 Updates

This section describes updates added to this document since its first release. Updates include addition to the document and fixed typos. The version number seen on the cover page will be updated accordingly.  There may be some typos even after proofreading the document. If this is the case, please let me know.






- **v0.0**: Original release of the document.

2 Conventions


In this document, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

- This is a  file.txt
- This is a  folder
- This is a  package
- This is an important note 
- This is a warning 
- This is a [link](#)
- This is a  topic
- This is a  service

3 Prerequisites

- Your first task is to understand what an Order is in ARIAC. Start with the [documentation](#) on Orders.
- Retrieve the file  rwa1.yaml from Canvas and place it in the  config folder as shown in [Figure 3.1](#).  This file is different from the one shown in class, make sure you get the latest version from Canvas. This file is a trial configuration file and consists of 3 orders that will be announced at different time intervals. The first order consists of a kitting task and is announced when the competition is started. The second order consists of an assembly task and will be announced 5 s after the competition is started. The third order consists of a combined task and will be announced 10 s after the competition is started.  You can modify the time announcement for each order if you want some orders to be announced before others or if you want some orders to be announced later or earlier.
- Create a package using your group number as the package name. For instance, the package for group1 will be  group1. This package will be reused in future

assignments and more functionalities will be added to it. It is recommended to create this package in the same workspace as ARIAC (See Figure 3.2), this way you will be working with only one workspace at a time. You cannot create a ROS Python package for the course since you will need C++ later for motion planning. Therefore, you need to work with a ROS C++ package and it is up to you if you want to include a Python package within this ROS package.

- In this assignment you will need to include/import Messages and Services from ARIAC. The ARIAC package which hosts Messages and Services is  `ariac_msgs`. For example:

```
# python
from ariac_msgs.msg import Order, CompetitionState
from ariac_msgs.srv import SubmitOrder

// cpp
#include <ariac_msgs/msg/competition_state.hpp>
#include <ariac_msgs/msg/order.hpp>
#include <ariac_msgs/srv/submit_order.hpp>
```

Figure 3.1: Trial file location.

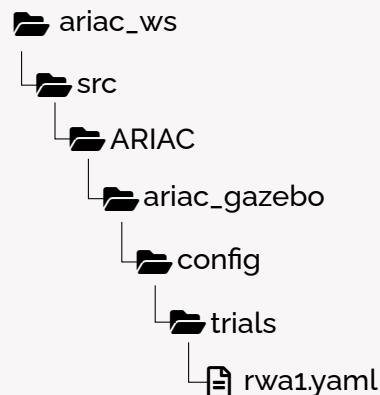
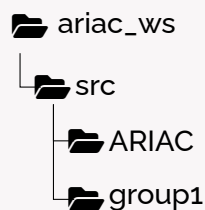



Figure 3.2: Package structure.

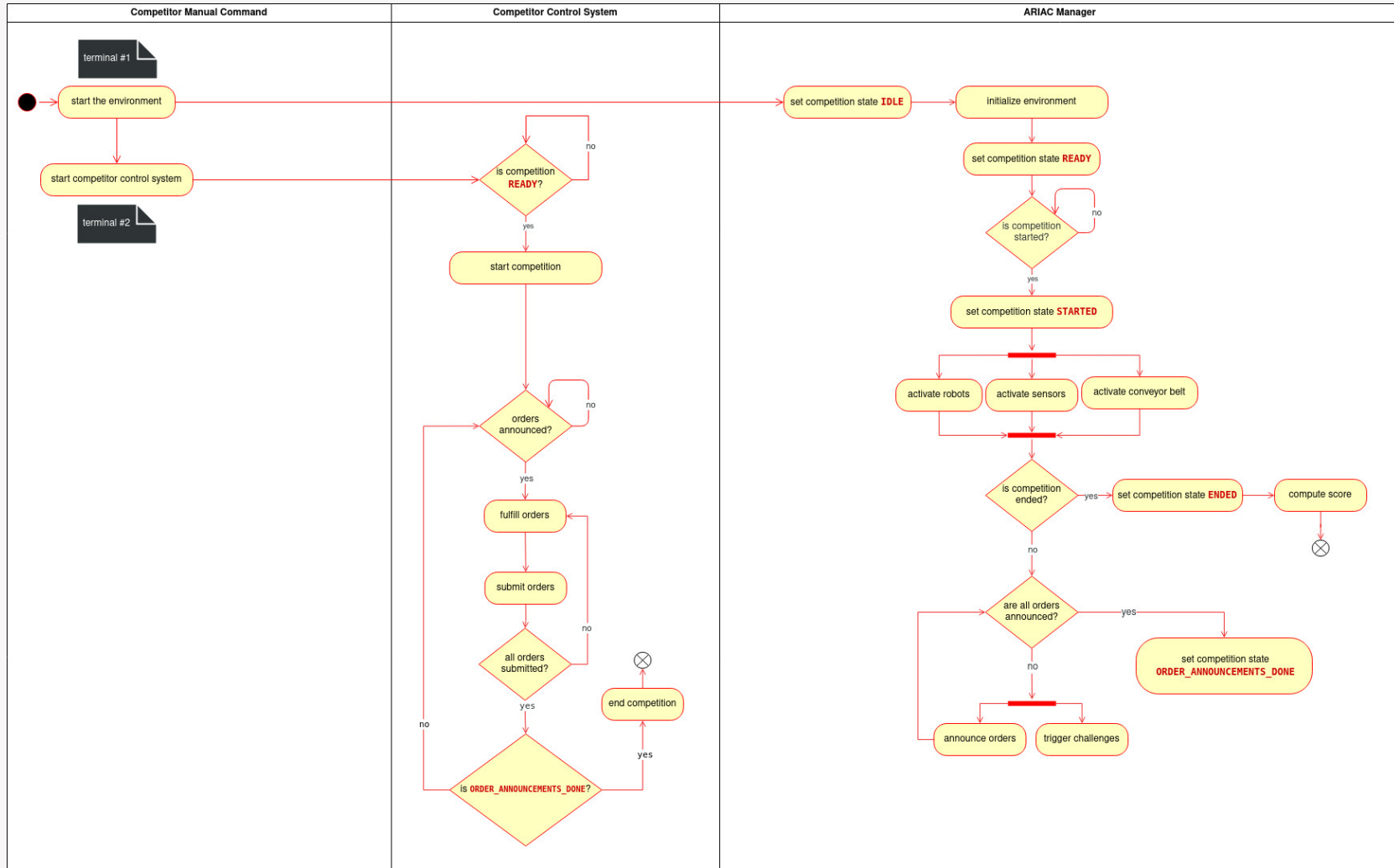


4 Assignment Tasks

-  This is a group assignment.
- Resources needed for the assignment can be found in the official [documentation](#).

The assignment consists of writing an application to 1) start the competition, 2) store orders published by the ARIAC manager, 3) submit these orders, and 4) end the competition. [Figure 4.1](#) shows the different states of the competitor's control system (CCS) and the ARIAC manager (AM). The left column shows what you have to do in terminals, the middle column shows what your control system (CCS) needs to do, and the right column shows what the ARIAC manager does. Your focus should be only on the first and second column. The third column is for your information only.

Figure 4.1: Competition Flowchart.



4.1 Competitor Manual Command

In one terminal, start the environment using `ros2 launch ariac_gazebo ariac.launch.py trial_name:=rwa1`

- `ros2 launch ariac_gazebo ariac.launch.py trial_name:=rwa1`

Start your Node(s) in a different terminal once you are done with the implementation.

- `ros2 run <package> <executable>` if you are running one Node.
- `ros2 launch <package> <launch file>` if you want to start multiple Nodes.

4.2 Starting the Competition

Once you start the environment, Gazebo will start but the competition is not started yet. This means that sensors are not publishing, robots cannot be controlled, and parts on the conveyor belt are not spawned. You need to start the competition with the service `/ariac/start_competition` which is a trigger Service (the request is empty). You can only start the competition when the competition is in the state READY (see Figure 4.1). The state of the competition is published on the Topic `/ariac/competition_state`. Messages published on this Topic have the structure described in Listing 4.1. The steps to start the competition are described in Tasks 4.1.

Listing 4.1: CompetitionState.msg

```
uint8 IDLE=0    # Competition cannot be started yet by the competitor
uint8 READY=1   # Competition can be started by the competitor
uint8 STARTED=2 # Competition has been started
uint8 ORDER_ANNOUNCEMENTS_DONE=3 # All order announcements have been made
uint8 ENDED=4   # Competition has ended

uint8 competition_state
```

Tasks 4.1: Tasks to start the competition.

- Write a Service client for starting the competition.
- Create a subscriber to the Topic `/ariac/competition_state`
- In the subscriber callback function check the value of the incoming Messages is READY before calling the Service client to start the competition. You can call the Service client directly from the callback function.
- In class we saw that we should use constant names directly and not their value. For instance, in the callback function for `/ariac/competition_state`, you would check the value of incoming Message as follows:
 - (C++) `if (msg->competition_state == ariac_msgs::msg::CompetitionState::READY)`
 - (Python) `if msg.competition_state == CompetitionState.READY`

4.3 Retrieving Orders

Once the competition is started, Orders are published to the Topic `t /ariac/orders`. Your task is to store each order in a data structure in your program. An order is a complex entity and it is recommended to write an Order class. Examples of classes can be found [here](#). Since a trial may contain multiple orders, it is recommended to push/append Order objects in a vector/map (C++) or list/dictionary (Python) so that they can be processed later. The tasks to retrieve Orders are described in [Tasks 4.2](#).

Tasks 4.2: Tasks to retrieve Orders.

- Create OOP class to represent Orders.
- Create a subscriber to the Topic `t /ariac/orders`
- In the callback of `t /ariac/orders`, parse the incoming Messages, create Order objects from the Messages, and add the Order objects to your data structure.

4.4 Submitting Orders

Before submitting an Order we need to complete it by performing pick-and-place. Since we are not doing pick-and-place yet, your next task is to read the data structure used in the previous task and submit each Order from the data structure. This data structure should contain 3 Order objects since `rwa1.yaml` has 3 Orders. ⚠ One of the three Orders ('LUFFY301') is a high-priority Order and must be submitted before the other Orders. The other two Orders can be submitted in any order.

To submit Orders, you need to use the Service `s /ariac/submit_order`, which has the definition seen in [Listing 4.2](#). This Service is called by passing a string, which is the ID of the Order. The tasks to submit Orders are provided in [Tasks 4.3](#).

Listing 4.2: SubmitOrder.srv

```
string order_id
---
bool success
string message
```

Tasks 4.3: Tasks to submit Orders.

- Create a Service client to submit Orders.
- For each Order object in your data structure, call the service client by passing the Order ID.
 - The Order with high priority must be submitted before non high-priority Orders.

4.5 Ending the Competition

Based on [Figure 4.1](#), the CCS should end the competition once all Orders have been submitted and when the state of the competition is `ORDER_ANNOUNCEMENTS_DONE`. To end the competition, the Service `s /ariac/end_competition` should be used. Just like the `s /ariac/start_competition` Service, it is a trigger Service, you need to call it with an empty request. The tasks to end the competition is provided in [Tasks 4.4](#).

Tasks 4.4: Tasks to end the competition.

1. Create a Service client to end the competition.
2. Check all Orders have been submitted (data structure is empty).
3. Check there are no more Orders to process (competition state is `ORDER_ANNOUNCEMENTS_DONE`).
4. If both 2. and 3. are satisfied, call the Service client to end the Order.

5 Timer Callback

In class we saw how to use a timer callback to publish Messages at a regular interval. A timer callback is often used in ROS to check the status of class attributes to perform some actions. For instance, to end the competition, all announced orders must be submitted and the state of the competition must be `ORDER_ANNOUNCEMENTS_DONE`. You can set up a timer callback which runs at 2 Hz for instance. In this timer callback you could call the end competition Service if both conditions are satisfied. These conditions will be checked regularly in the timer callback in the background so you do not need to check them in other parts of the code.

6 Contribution

This assignment is not difficult but requires the contribution of each student. Peer reviews are required for each assignment to ensure everyone is contributing. Using Github for assignments is highly recommended and using private repositories is a safe way to avoid plagiarism. If someone in your group has experience with Github, I would like this person to communicate to their teammates how to tackle each piece of this assignment.

Try to divide and conquer. The way this assignment is set up, everyone can work on a part of the assignment without waiting for the others. Here are some suggestions.

- One person should be responsible for creating the Service clients and the subscribers. These should be done first. Once these are implemented you can start working on the other tasks.

- **Starting the Competition:** Once the Service client for starting the competition and the callback function is implemented for the Topic `t /ariac/competition_state`, this task consists of just calling the client in the callback function. Maybe two lines of code are only needed here.
- **Retrieving Orders:** Students working on this task do not need to wait for the previous tasks to be done. You can start the competition from the terminal during the implementation of this task. This task is the most important one and may require at least 2 students.
- **Ending the competition.** The person working on this task should be responsible for creating the timer callback to check if the conditions are satisfied before calling the end competition Service. More conditions will be added later in future assignments in this timer callback.

7 Submission

You can either compress and submit your package on Canvas or invite us (anikk94 and zeidk) as collaborators to your private Github repository. If you submit your package, make sure you only submit your package and not the whole workspace. There is no deadline extension for this assignment and late submissions will be penalized. We often have situations where students forget to include a bug fix in the submission and they ask permission to submit again. We do not grant permission to resubmit the assignment passed the due date and time. Before you submit the assignment, delete `build`, `log`, and `install`, rebuild your package, and run your Node(s) again to make sure everything works.

8 Grading Rubric

- This assignment is worth **30 pts**.
 - **5 pts** for starting the competition based on the state of the competition.
 - **15 pts** for representing Orders using OOP and storing them in a data structure.
 - **8 pts** for submitting each order from your data structure.
 - **2 pts** for ending the competition based on the state of the competition and the data structure.