ENPM663: RWA-1
University of Maryland, College Park

Team members:

Anirudh Krishnan Komaralingam (117446432)
Jayesh Jayashankar (117450101)
Kartikeya Mishra (118106755)
Neha Saini (117556292)

Professors:

Dr. Craig Schlenoff
Dr. Zeid Kootbally

Table of Contents

1. Assignment Objectives:

The objectives as part of RWA-1 are as follows:

- Creation a catkin package with the dependencies mentioned.
- Represent the architecture of the different functionalities of ARIAC through class and sequence diagrams using UML.
- Translating the architectures into a program by creating appropriate classes.
- Implementing a program to simulate the processing of an order.

2. Architecture

The goals are to represent the architecture of ARIAC through Class and sequence diagrams. Class diagram is used to represent the different components of a system and its attributes and the relationship between these components. A sequence diagram on the other hand is used to represent how the objects in a system interact with each other to execute different processes. The respective diagrams for ARIAC have been detailed in the following subsections.

2.1. Class Diagram

Figure 1 shows the class diagram for ARIAC. The class diagram represented here was a preliminary version of ARIAC based on our initial understanding of ARIAC. The classes as part of this diagram were based on the different components represented in the ARIAC wiki, the ARIAC example and the nist_gear package. The different classes in the diagram are as follows:

- **Brain:** This class consists of the Competition State and starts or ends the competition. This class would also consist of the world models for the AGVs and other components.
- **Plan:** The plan class would consist of the orders to process and the sequence in which it needs to be processed such as the handling of orders by priority.
- **Kitting Shipment:** This class would contain some information such as the station where the product needs to be delivered, shipment type, and methods to control the kitting robot and submit AGVs for kitting and assembly.
- **Assembly Shipment:** The Assembly class would be structured like the Kitting class and inherit the properties of the Plan class. This class should directly process assembly shipments or check if the kitting is completed to submit AGVs for assembly.
- **Sensors:** The sensor class contains the type of sensors for example breakbeam, laser scanner, depth camera, etc. along with their respective pose.
- **Products:** This class consists of the part type, faulty check flag and its location which would be the product bins or the AGV trays.
- **AGVs:** The AGV class would consist of information on the AGV ID and methods to obtain its location.

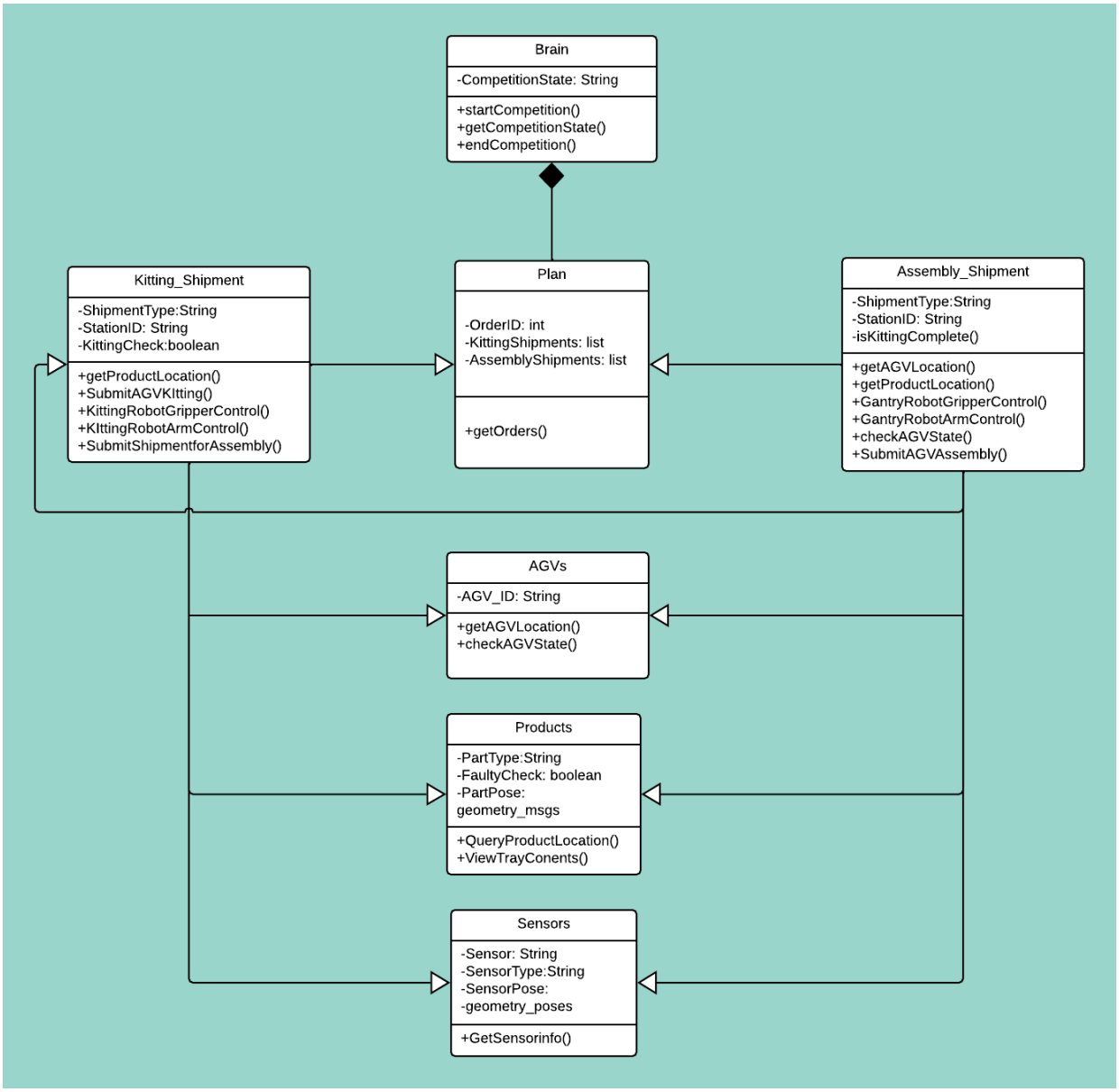

Figure 1: Class Diagram for ARIAC. The class diagram can be viewed here.

The above figure represents the classes which we came up with before starting to implement the tasks as part of RWA1. As we began the software implementation, we came up with easier ways to establish relationship between the classes and therefore some of the class names and methods would differ from the classes as part of the package.

2.2. Sequence Diagrams

The sequence for kitting and assembly are covered in the next subsections. Figure 2 represents a general sequence of events taking place in ARIAC. Based on the possible scenarios of assembly and kitting we developed the following generic sequence diagram, once the order is transferred; the number of shipments which vary between zero to three are transferred to supply location which has information of pose. This pose information might contain location to stationary bins, conveyor belt or AGVs. Once the

kitting robot which is actor here receives the pose, it will transfer the instruction for gripping the part and leave the part in the desired kit tray on the assembly station.

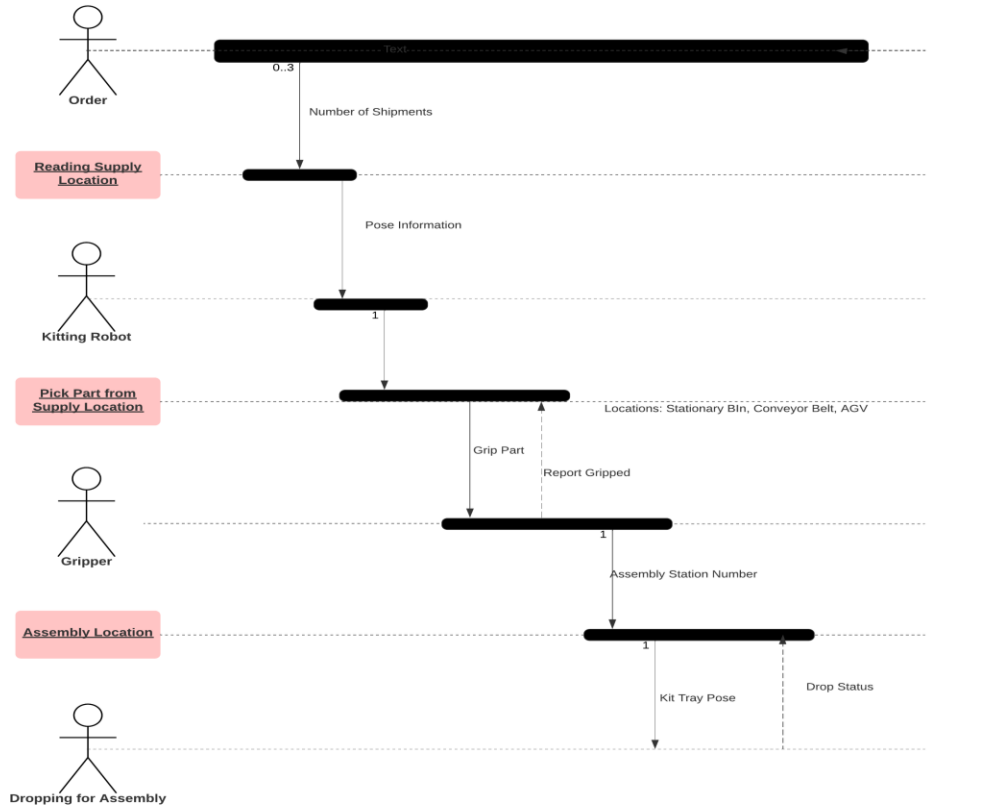

Figure 2: Sequence Diagram showing a general flow of the events in ARIAC. The diagram can be viewed here.

2.2.1. Sequence diagram for kitting

Figure 2 represents the sequence diagram for kitting. The different objects in the sequence diagram are main, competition, orders, products, AGVs and Kitting. The different components and the sequence of events are described as follows:

1. The main object would start the competition subscribing to the competition state and updating it.
2. Once the competition is started the next step would be to fetch the orders subscribing to the orders topic.
3. Before submitting the AGVs we need to obtain the product details such as its pose, the bin location, and checking if the part is faulty or not through quality control sensor. The sequence of events are as follows:
    a. If the part if not faulty then the part if fit to be collected by the AGV and it moves to the next sequence which is to fetch the AGV details to pick up the part.
    b. If not the next product in the order is processed or if there are no more products in the order, the next order is processed.
4. The AGV details are obtained such as the AGV ID, its location if it is in the kitting station or assembly station. The AGV sometimes might already contain a part. Therefore, it is essential to check for the faulty part in the tray. The sequence of events to handle this are as follows.
    a. If the part is not faulty. The AGV can be submitted for kitting.

b. If the part is faulty. The part Is moved to the faulty part collector

5. The next process in the sequence is to submit the AGV for kitting. Once the kitting is complete the next product in the order or the next order is processed. If there are no more orders, the control goes back to the main function and ends the competition.

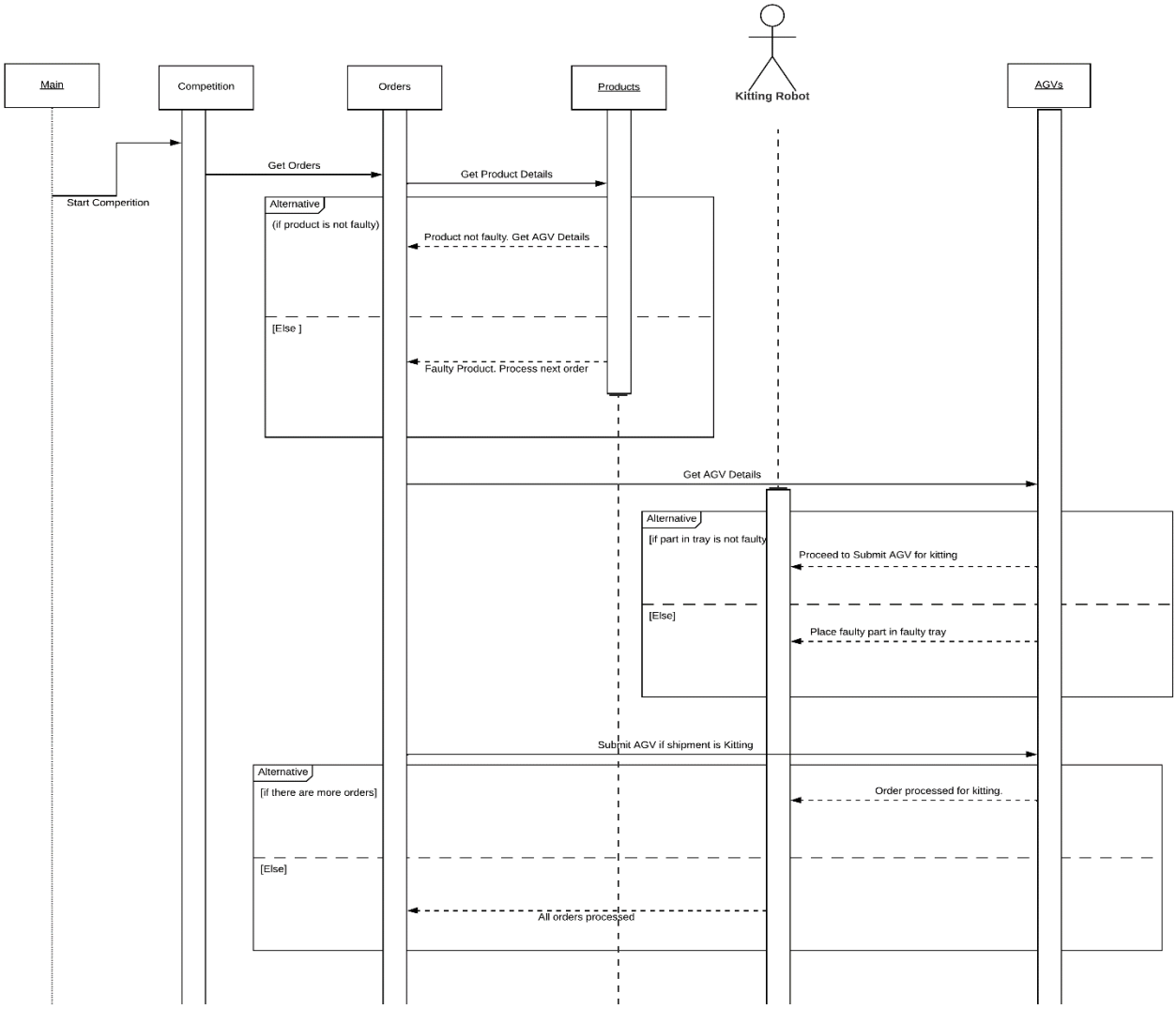

Figure 3: Sequence Diagram for Kitting. The diagram can be viewed here.

2.2.2. Sequence Diagram for Assembly

Figure 4 represents the assembly diagram. The sequence diagram for assembly is similar to that of kitting. The new object in the diagram would just be the assembly component.

The sequence of events for the assembly process can be described as follows:

1. The orders consisting of shipments for kitting would require the assembly to be performed after kitting. Therefore, once the kitting is complete, the AGV can be submitted for assembly. This can be done by using a flag to check if kitting is complete and change this flag indicating the product is ready for assembly.

2. The other consisting of shipments for just assembly are directly processed for assembly without going through the kitting process. But the same faulty product checks which are applied for kitting would apply for assembly as well.

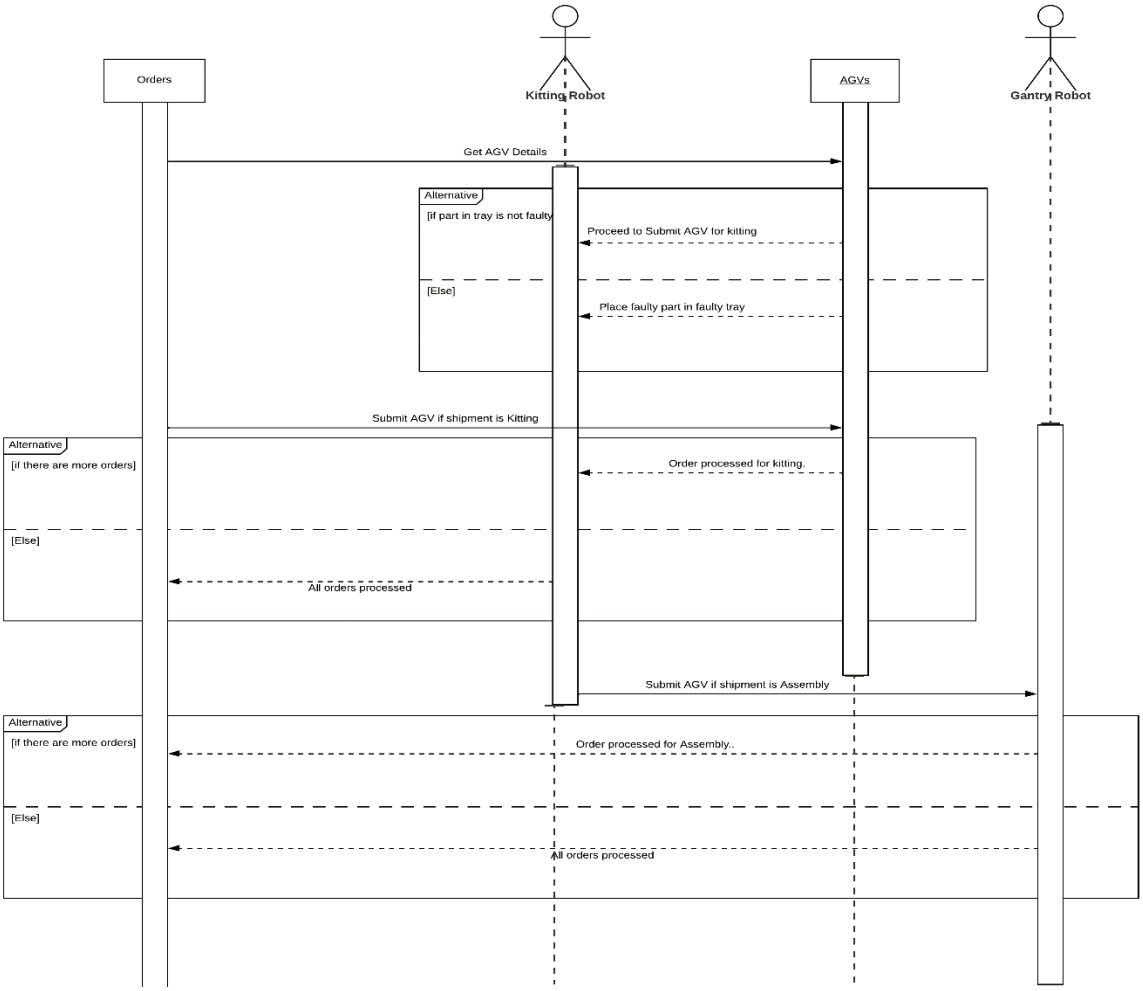

Figure 4: Sequence Diagram for Assembly. The diagram can be viewed here.

3. Program Structure and Flow

For out structure of program, we are following the ROS PyStyleGuide: http://wiki.ros.org/PyStyleGuide

In brief, we have segregated our code into two parts, ROS Nodes and Python Packages.

All nodes are kept in **nodes** folder and our regular OOP code is built into a Python Package and kept in **src**. For to integrate **python package** into ROS package, we supplied it with setup.py and __init__.py which gets called via catkin build

Our, main class is called Brain. By using Singleton Pattern, we instantiate a brain object once which can be used throughout the code which is called **StaticBrain**. StaticBrain makes sure that brain instance is created once and used everywhere.

Brain contains the **world model**, the **orders**, and the **plans** for completing those orders.

So, each plan is tailored towards a particular order. To fulfill each order, we create **tasks**. And this collection of tasks becomes a plan. Each Task is inherited from a base class to maintain the integrity of a task.

The **world model** inside the **Brain** is equivalent to knowledge representation inside our architecture. This consists of movable objects such as AGVs and other immovable objects. And each robot is a dummy robot containing only low level actions and services to perform those actions. Because **Brain** is handling the actual flow of tasks and ordering robots around and tell them what to do.

We have a **main_node** i.e., a ROS Node which has callbacks from various topics to update information into our architecture i.e., in **World Model**. So, we always have updated information when performing a particular task. And, in the **main_node** we are polling for any new plan to execute. Therefore, whenever an order is received, a plan is formed and executed.

And, we also have a separate **constant file** for the whole project for keeping the integrity of whole project intact.

4. Future Considerations and Conclusion

The report consists of creating the class and sequence diagrams based on our understanding of ARIAC. Even though most of the components have been represented, there are still a few more things to consider when it comes to the sequence diagram for kitting and assembly such as the handling of an event of higher priority order and avoiding collision with other objects. The diagrams represent the architecture of ARIAC as a whole. Therefore, this set up a good blueprint to be followed for not just this RWA but for future RWAs as well.

References

https://github.com/usnistgov/ARIAC/tree/master/wiki