COMPSYS 704 Project 1

GRP Compendium – Milestone 2

Automated Bottling System (ABS)

Group 13

Francis Cho

Xuhan Liu

Gayeon Kim

Project Components

The main components of these projects are:

GRP (ABS) Components

- Conveyor
- Rotary Turntable
- Filler
- Capper
- Cap Loader (Lid Loader)

IRP Components

- Product Order System (POS)
- Environment Control System (ECS)
- Safety and Access Control System (SACS)

The ABS was designed as a team while POS, ECS, and ACS were completed individually.

Work distributions

GRP Allocation	Group member
ABS (Conveyor)	Everyone
ABS (Filler)	Francis
ABS (Capper)	Gayeon
ABS (Rotary TurnTable)	Xuhan
IRP Allocation	Group member
POS	Francis
ECS	Gayeon
SACS	Xuhan

Conveyor

The conveyor system will transport the first (empty) bottle at position 1 in the rotary turntable and transport out the last (filled) bottle from position 5. There are four sensors that are used to identify the presence of a bottle in each of the 4 positions - bottleAtPos1, bottleAtPos2, bottleAtPos4, bottleAtPos5, bottleLeftPos5. These sensors could all be used as input signals in our conveyor system's clock domain to track the positions of each bottle and decide to either enable or disable the conveyor motor. However, we have decided that only two signals: bottleAtPos1 and bottleLeftPos5 are necessary for use cases.

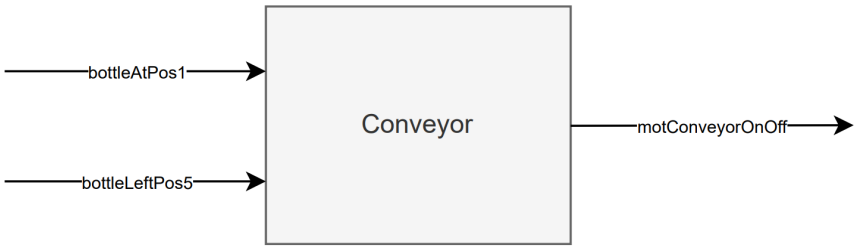

Figure 1: Basic Input and output signals of the Conveyor clock-domain

The conveyor checks if there is any remaining bottle at its far end before proceeding the new bottle in. This is identified by the *bottleLeftPos5* signal. If the *bottleLeftPos5* signal is absent, the conveyor will activate the motor by emitting *motConveyorOnOff* signal to move the new bottle to position 1. The presence of *bottleAtPos1* signal will indicate that the bottle has reached position 1 and the bottle will be transferred to the rotary turntable. Once the bottle has been processed and reached the end of the conveyor, the *bottleLeftPos5* signal should be emitted. The *motConveyorOnOff* signal is used to decide whether to deactivate the conveyor's motor when the bottle has successfully moved past position 5.

In order to help with testing and debugging, as well as providing more system capabilities we decided to introduce two modes of operation of the conveyor system: manual and automatic. To implement this, we had to add two additional internal signals in order to transition between manual and automatic mode. We have a function running concurrently that is checking for the presence of either (internal signal). In our design, present statements were used to check for the presence of both input and internal signals. We utilised the await statements to pre-empt the change to a different mode of operation signal. For example, if a manual signal is emitted, our system will check for the presence of the *motConveyorOnOff* signal in order to either turn on/off the conveyor motor. We used sustain statements when we needed the conveyor motor to run continuously (which required a constant *motConveyorOnOff* signal emission). This was useful for automatic mode where neither *bottleLeftPos5* or *bottleAtPos1* signals are present (indicating no bottle at position 1 or no bottle has left position 5) and the *motConveyorOnOff* signal can be sustained to start the conveyor. An assumption is that the success rate of the conveyor to transport the bottles is 100% so we do not have any systems in place to handle (mechanical) errors or failures. Unfortunately, the conveyor was not able to be seamlessly integrated with our entire ABS system so manual mode (enable) needs to be used in order to activate the conveyor system. Future improvements to our design may incorporate more input signals for a more robust system that tracks all bottles, the conveyor clock domain may utilise all the bottle sensors as input signals.

We decided to introduce 3 additional signals that serve a major purpose in the overall flow of the ABS design. These are input signals bottleLeftPos5EConveyor, enableConveyorMain and output signals bottleLeftPos5RotaryE. These allow communication with both the Rotary Turntable system and the Main controller top-level system that gives us more control.

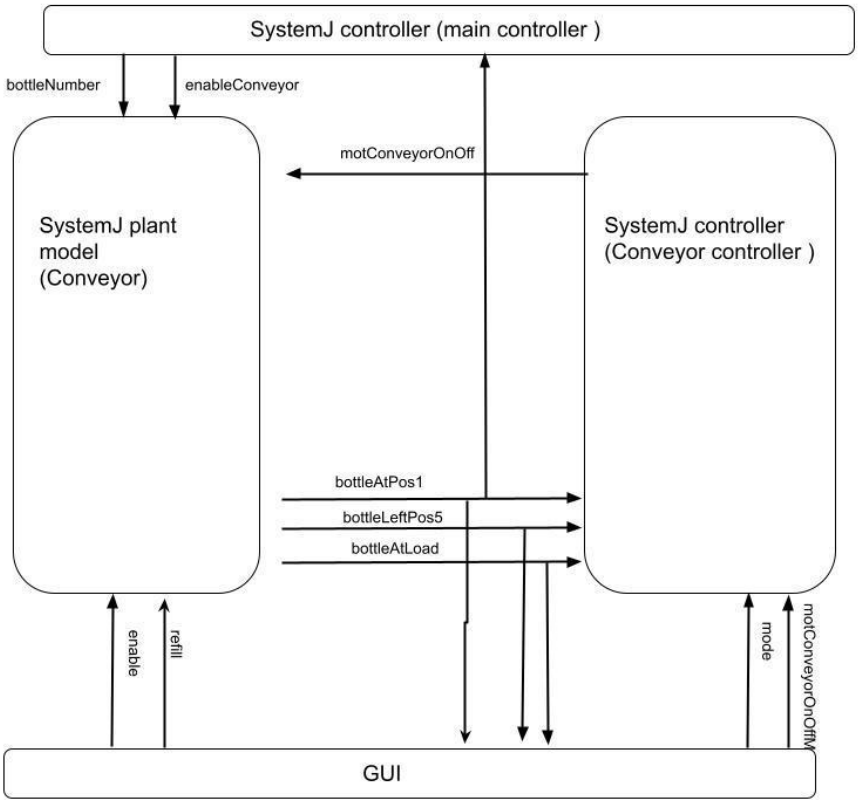

Figure 2: *Diagram of Conveyor System*

Rotary Turntable

The rotary turntable is in charge of transitioning bottles from the conveyor for further processing. It rotates and moves the bottle by 60 degrees with each rotation. The first rotation positions the bottle directly under the liquid filler for filling. The next rotation takes the bottle to the lid placement point where a lid can be placed. The turntable rotates once more which aligns the bottle with the capper, which then attaches and tightens the lid. Lastly, the finished bottle is returned to the conveyor which is then transported to the collection point at the right end of the conveyor.

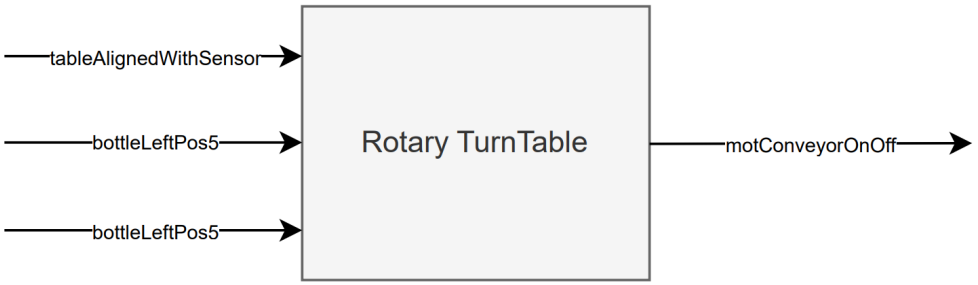

Figure 3: Basic Input and output signals of the Rotary TurnTable clock-domain

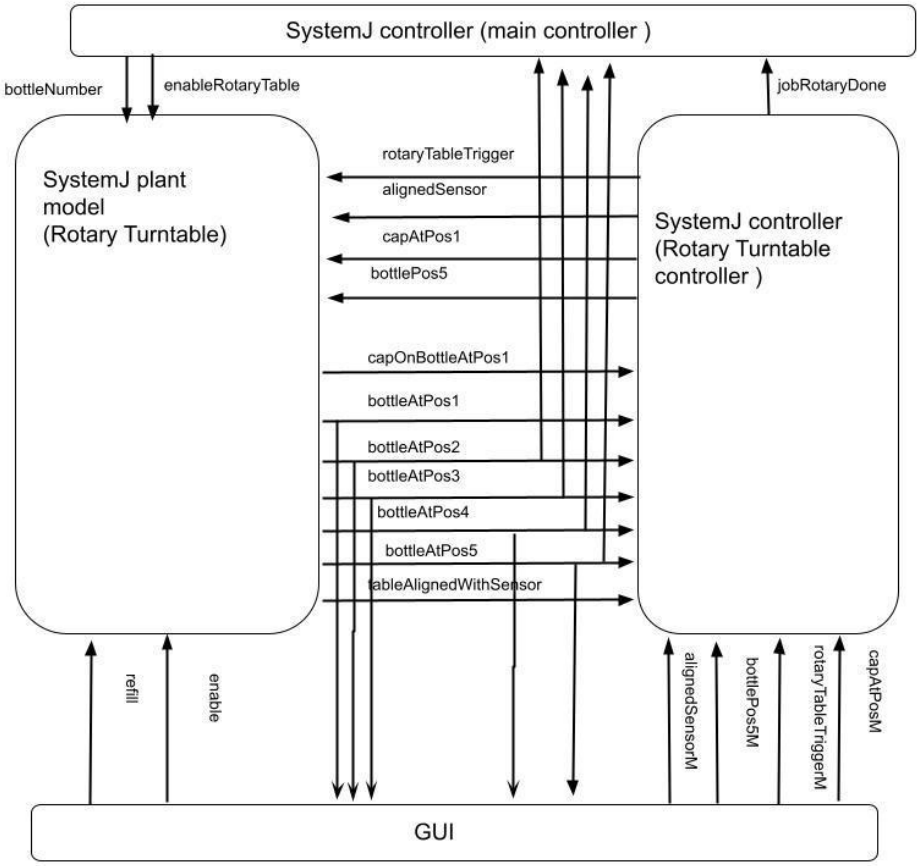

Figure 4: Diagram of Rotary Turntable System

The turntable works with other Automated Bottling System (ABS) subsystems such as the liquid filler and lid loader/capper in addition with the sensors that will be used to detect the position of the bottle. If the bottle at position 1 already contains a lid, the system will wait until the bottle is manually removed from the turntable. This will be indicated by the *capOnBottleAtPos1* signal and the table should only rotate when this signal is absent. The *bottleAtPos5* signal will be used to indicate when the bottle is at position 5. If *capOnBottleAtPos1*

signal is absent, emit the *rotaryTableTrigger* signal for about 0.5 seconds to turn the rotary table by 60 degrees, then proceed to the next step. After rotating the table, it should wait for the *tableAlignedWithSensor* signal which indicates that the table's positions are well aligned with the sensors. It is assumed that the rotary turntable will always rotate by exactly 60 degrees with each turn, only in one direction. Only one bottle is rotated and processed at a time on the turntable.

We refined the rotary turntable clock domain further into a plant clock domain and a controller clock domain. The controller CD handles signals from sensors or other parts of ABS, such as filler, capper, and lid loader. Based on these input signals, the controller decides what actions the rotary should take, and then it sends signals like rotary table triggers to the plant CD. During our initial implementation, we examined the behaviours of the rotary turntable: it cannot rotate until there is a bottle in position 1. When the bottle is in the filling, lid loading, and capping processes, it cannot rotate until those processes are finished. Therefore, we added several input signals to the rotary controller CD: *bottleDonePos1, bottleDonePos2, 3, 4, and 5*. These signals indicate that the bottle is loaded in position 1, the bottle is filled with liquid, the lid is placed on the bottle, the lid is screwed onto the bottle, and the bottle is transferred to the conveyor. When the rotary controller CD receives these signals, it sends rotary trigger signals to the plant CD to initiate the rotation. The controller CD has three main reactions: one for mode selection, one for manual mode reaction, and another for auto mode. The manual and auto main reactions are further divided into 5 sub-reactions, each of which handles reactions for 5 positions.

Three additional signals (*bottleLeftPos5RotaryE*, *bottleAtPos1R*, and *fillerReadyStart*) were introduced over our initial design in order to give more control and communication abilities between the ABS components. The signal *bottleLeftPos5RotaryE* is an input signal from the conveyor plant that indicates when a bottle was about to leave the rotary turntable. The signal bottleAtPos1R is another input signal that is used within an await statement at the beginning of the main loop. This signal is also from the conveyor plant to indicate when the bottle was at position 1, ready for the rotary turntable to initialise and begin turning. The signal fillerReadyStart is an output signal to the filler plant which effectively activates the filler when the rotary turntable was in a certain position. The usage of these extra signals gave more flexibility and higher degree of accuracy in how the implemented designs function and interact with each other.

Filler

The Filler is a system that fills liquid in the bottle at position 2 of the rotary turntable. As a maximum of 4 different liquids can be mixed to the final product, four liquid fillers are required. The Filler only operates if a bottle is detected at position 2 of the rotary turntable.

Figure 5: Input and output signals of the Filler clock-domain

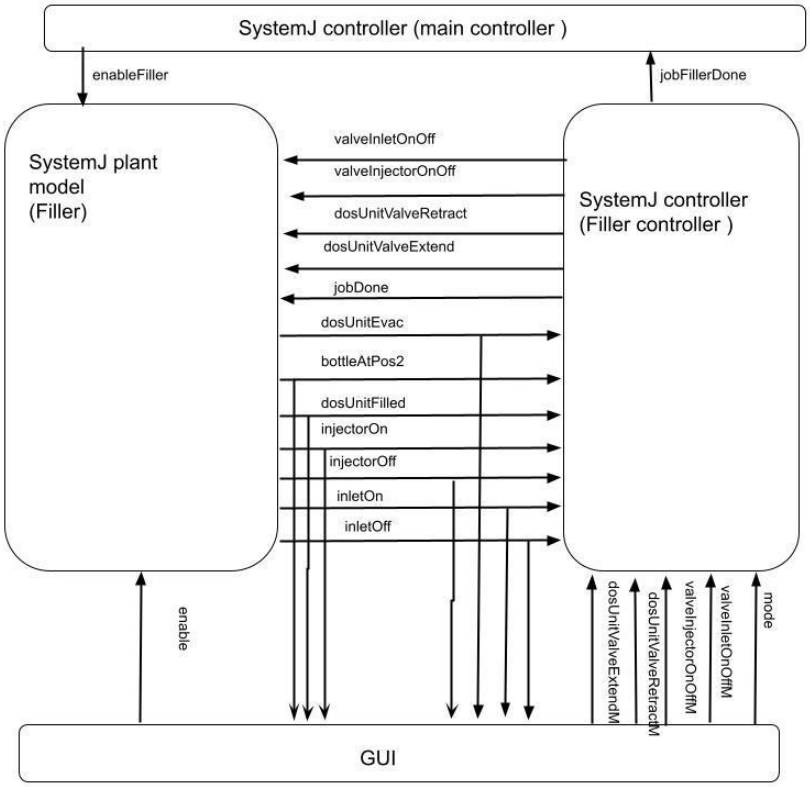

Figure 6: Diagram of Filler System

The Filler Clock Domain (Filler CD) is further refined into two distinct domains: the Plant Domain and the Controller Domain, providing a structured approach for managing the filler process. The filler operation commences upon the receipt of a *"bottleAtPos2"* signal, originating from the rotary turntable. Within the Filler Plant Clock Domain, there are four primary reactions. The first is dedicated to updating GUI states, ensuring real-time information is displayed to users. The second manages the actual filling process, overseeing the flow of liquid into containers. The third reaction handles various outputs, including commands like *"valve extend"* and *"valve extract,"* received from the Controller CD to control the valve mechanisms in the filler plant. The fourth reaction captures signals related to liquid ratio and bottle quantity from the POS GUI system, vital for maintaining precise measurements during filling. Upon successful completion of the filling process, the Filler CD emits a *"bottleDonePos2"* signal, which is communicated back to the rotary turntable, signifying the end of the filling operation at Position 2. This division and structured approach enhance the coordination and control of the filler system, ensuring its efficient and reliable operation.

Additional design decisions made to the Filler include introducing four additional signals to the Filler Plant (*bottleAtPos2F*, *fillerReadyStart*, *disableConveyorMain*, *systemEnableFiller*), as well as sending the relevant liquid amount and bottle quantity selection from the POS GUI directly to the Filler Plant through the Main Controller. The output *bottleAtPos2F* signal is from the Rotary controller and acts as a means of aborting multiple sustain statements within the Filler Plant. The input signal *fillerReadyStart* from the rotary turntable serves as an enable function to initialise and start all four fillers. The input signal *systemEnableFiller* encapsulates the global start signal from the POS GUI (*systemEnable*) and is received via the Main Controller. This allows an almost direct line of communication with the POS GUI to the Filler system and allows the signal to abort "sustain disableConveyorMain" which allows the conveyor to continue working. The purpose of this is to allow to user to submit another order once their previous one has been completed by the ABS facility.

Capper

The capper system essentially attaches and tightens the lid of the bottle at position 4 of the rotary turntable. The capper system only operates if a bottle is placed at position 4 of the rotary turntable.

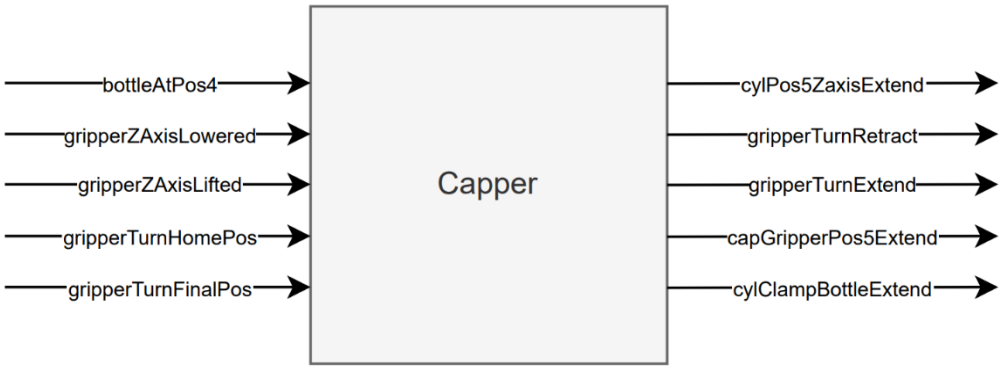

Figure 7: Input and output signals of the Capper clock-domain

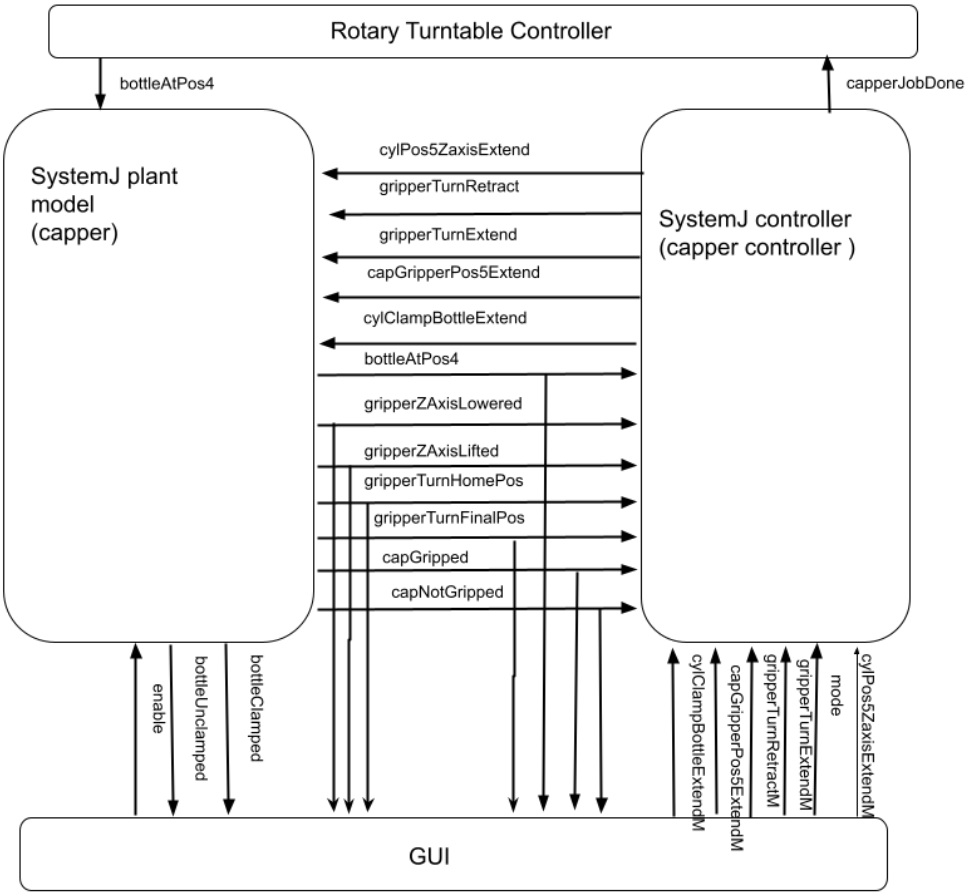

Figure 8: Diagram of capper System

Several new signals have been introduced to monitor and control the operational status. These include signals to indicate the status of the cap lid gripping (*capGripped* and *capUnGripped*) and signals to indicate the clamping status of the bottle (*bottleClamped* and *bottleUnclamped*). The capGripped signal is specifically used to ensure that the gripper twisting action can only be activated when the cap is gripped. In our design, the *await* statement can be used to check for certain signals and *sustain* statements to continuously emit signals until another input signal is detected by the system such as signals *cyClampBottleExtend and gripperTurnExtend* in order for the clamp and gripper to extend. Some assumptions include: The bottle remains stable during the capping process and that all components of capper work without any failure

We can refine the capper clock domain further into a capper plant domain and a capper controller domain. The capper process begins when there is a *'bottleAtPos4'* signal. In the capper controller CD, there are four main reactions: mode selection reaction, manual mode reactions, auto mode reactions, and a reaction that sends the *'bottleDonePos3'* signal to the rotary when the capper process is finished. In the capper plant CD, there are also four main reactions: the arm reaction, the pusher reaction, the vacuum reaction, and a reaction for updating states in the Unified ABS GUI.

Unified ABS GUI for Visualization

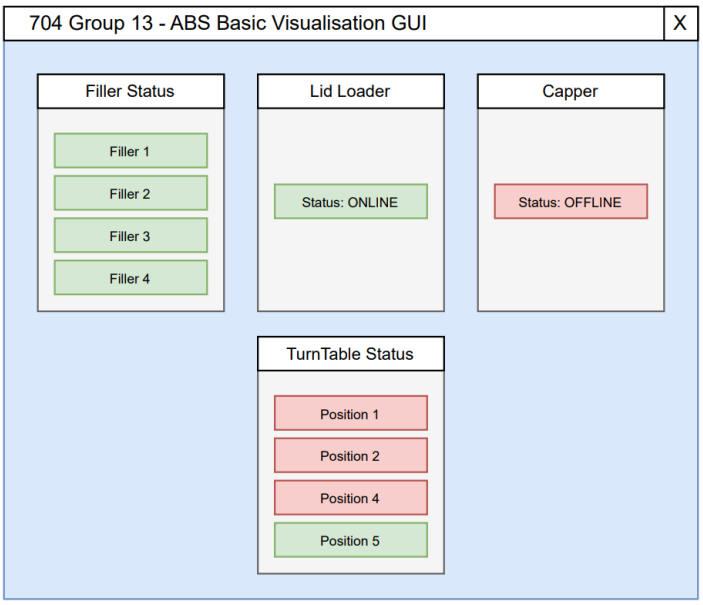

Figure 9: Mockup GUI

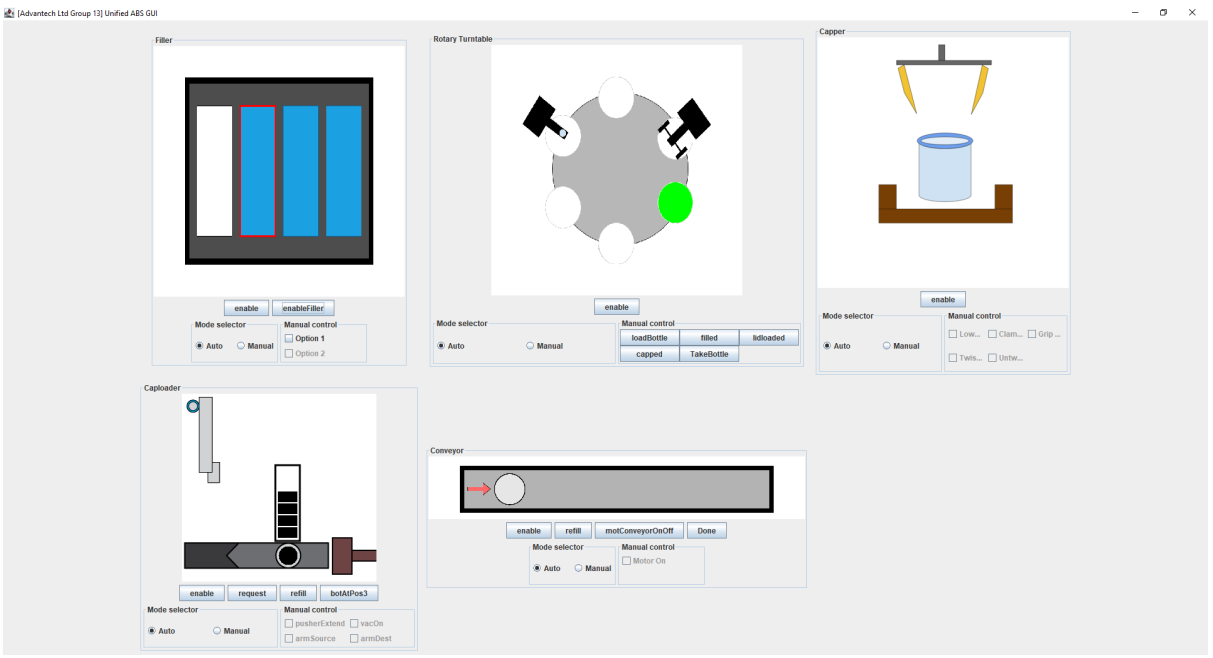

Figure 10: Actual GUI Implementation

We initially had a very basic GUI setup to only update the status of each ABS subsystem, but decided that this did not provide enough information. Therefore, we opted to design and develop a more detailed GUI for each ABS component and in the later stages of development to combine all the separate GUI components into a single, unified ABS GUI window.

Java Swing toolkit was used to develop this GUI - mainly using GridLayout structures with JPanels or JFrames. Regarding the implementation process, we initially designed and developed a separate GUI window for each ABS component subsystem. Each GUI window consists of a canvas class which handles repainting/updating with images based on each ABS component's state. We had multiple VizWorker classes that enabled us to capture the relevant signals from the ABS component clock domains and change the corresponding states. We have additionally added some degree of interaction through the use of buttons and checkboxes. By setting up a

connection with a signal client, we are able to manually generate signals for the particular ABS component when required (for example during debugging). We made use of await statements in the ABS clock domain to avoid missing signal inputs from the GUI. The Unified ABS GUI essentially embeds each separate instance of the individual GUIs of the subsystems. This means for signal communication a SignalServer for each VizWorker (Visual Worker) needed to be set up with the appropriate ports and initialised on a new thread. (E.g. *SignalServer<FillerVizWorker> server = **new** SignalServer<FillerVizWorker>(Ports.PORT_FILLER_VIZ, FillerVizWorker.**class**)*, note the ports are separately defined in a Ports.java class with the relevant address data.

```
// Start the signal servers for the GUIs
SignalServer<FillerVizWorker> server = new SignalServer<FillerVizWorker>(Ports.PORT_FILLER_VIZ, FillerVizWorker.class);
SignalServer<RotaryVizWorker> server2 = new SignalServer<RotaryVizWorker>(Ports.PORT_Rotary_VIZ, RotaryVizWorker.class);
SignalServer<CapperVizWorker> server3 = new SignalServer<CapperVizWorker>(Ports.PORT_CAPPER_VIZ, CapperVizWorker.class);
SignalServer<ConveyorVizWorker> server4 = new SignalServer<ConveyorVizWorker>(Ports.PORT_Conveyor_VIZ, ConveyorVizWorker.class);
SignalServer<LoaderVizWorker> server5 = new SignalServer<LoaderVizWorker>(Ports.PORT_LOADER_VIZ, LoaderVizWorker.class);

new Thread(server).start();
new Thread(server2).start();
new Thread(server3).start();
new Thread(server4).start();
new Thread(server5).start();
```

Figure 11. Setup of multiple threads for the Signal Servers

Because there are five separate embedded GUI windows (or JPanels) within the Unified ABS GUI, multiple signal servers were required to be running. This was achieved by simply initialising and creating a new instance of a signal server for each ABS component VizWorker and allocating them to a new thread. Furthermore, the Unified ABS GUI required a *repaint()* function used each time a new instance of each GUI is called in order to update the image state for each panel. These new instances depend on predefined image states, where states are set up in each VizWorker class through the use of switch-case statements.

```
case "bottleDoneE":
    States.BOTTLE_AT_POS_5 = false;
    States.BOTTLE_AT_POS_1 = false;
    States.BOTTLE_AT_LOADER = false;
    States.BOTTLE_AT_EMPTY = false;;
    States.BOTTLE_AT_DONE = true;
    break;
```

Figure 12. Example of a case from the ConveyorVizWorker class

These then allow the component canvas classes to update based on the most current updated states. For example, if the case is “bottleDoneE” then all states except BOTTLE_AT_DONE is set to true. In the corresponding ConveyorCanvas class, the statement for BOTTLE_AT_DONE is executed.

```
else if (States.BOTTLE_AT_DONE) { // done
    g.clearRect(0, 0, x, y);
    g.drawImage(conveyor4_resized, x1, y1, null);
}
```

Figure 13. Example from ConveyorCanvas class

The following figures showcase the different possible sequences displayed by each ABS component. Note, not every single sequence is shown.

Rotary Turntable

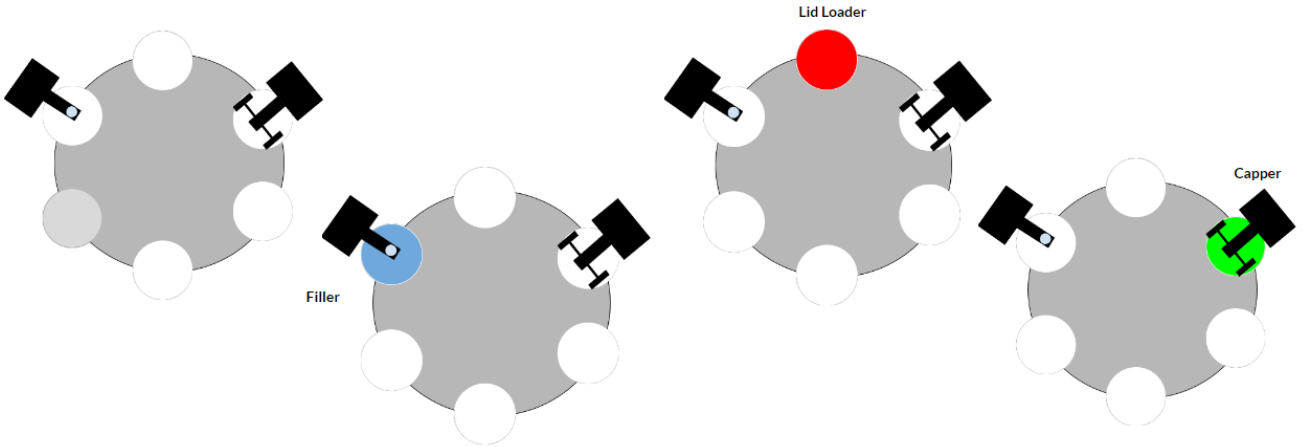

Figure 14. Rotary Turntable different sequence

Conveyor

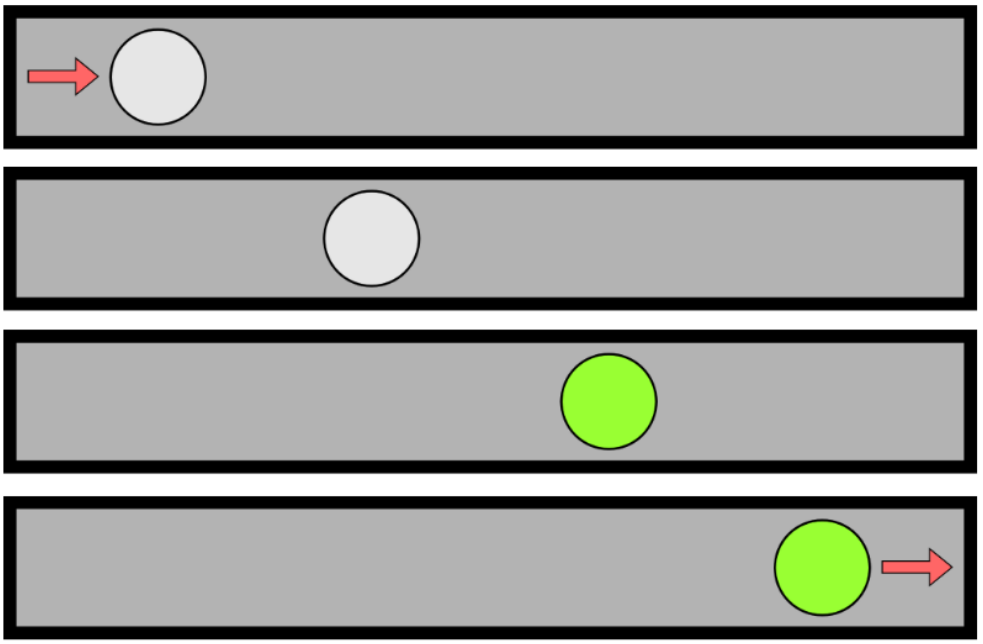

Figure 15. Conveyor system in sequence of events

Filler

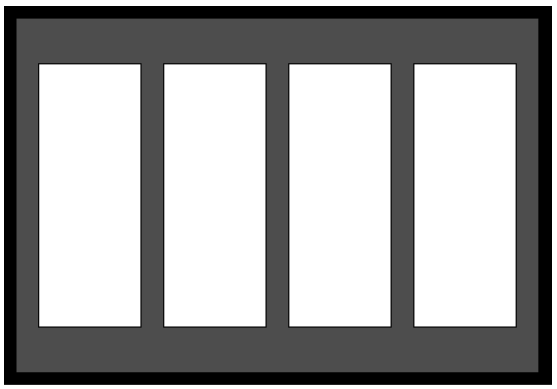

Figure 16. All four Fillers are Empty

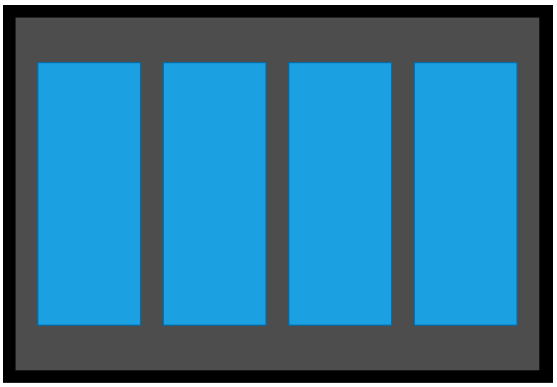

Figure 17. All four Fillers are full of specified liquid

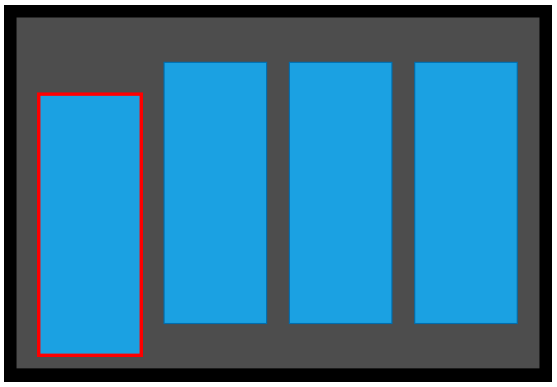

Figure 18. Selected Filler tube is extended

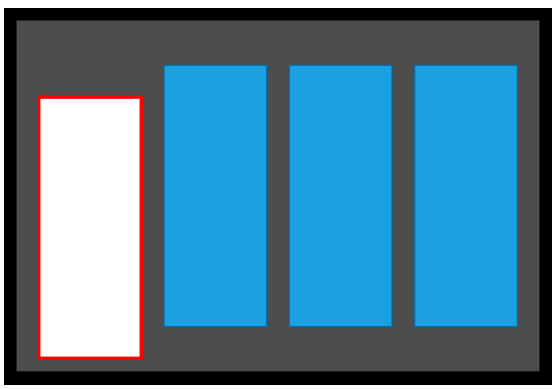

Figure 19. Selected Filler tube is emptied

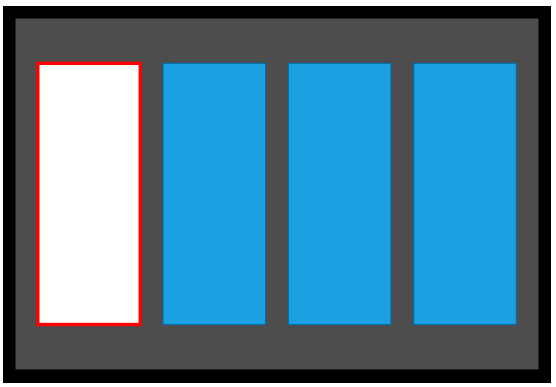

Figure 20. Selected Filler tube is retracted

Capper

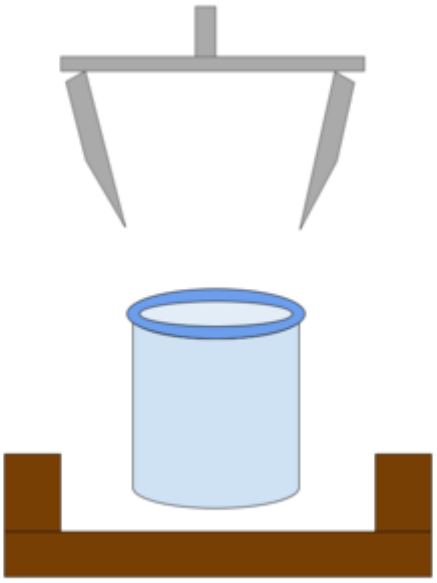

Figure 21. Capper idle

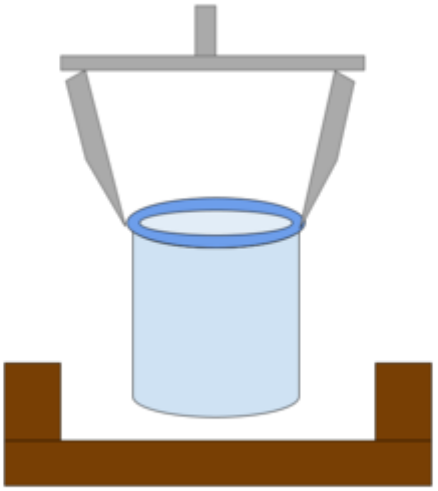

Figure 22. Capper gripper lowered

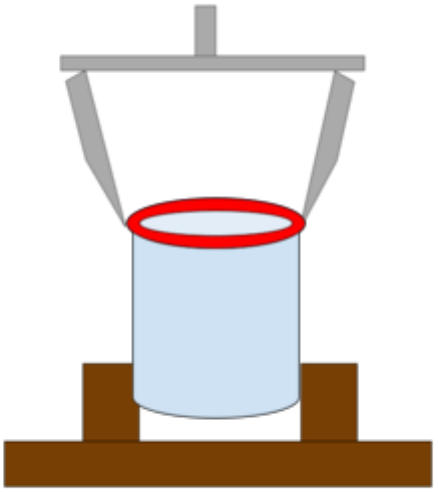

Figure 23. Capper bottle clamped and cap gripped

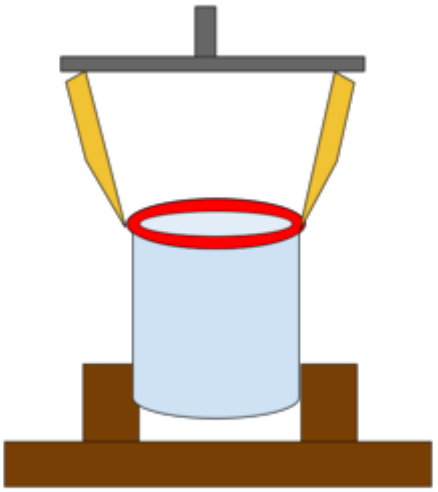

Figure 24. Capper gripped twisted

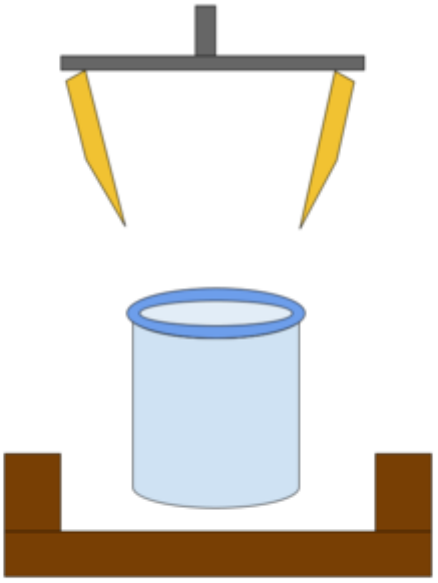

Figure 25. Capper bottle unclamped and gripper lifted

The Capper operates when the bottle reaches position 4 on the rotary turntable. It has a clamp that secures the bottle to ensure stable operation and a gripper that performs the cap twisting action. Once the capping is done, the signal capperJobDone is sent to the RotaryControllerCD indicating that the capping stage is successfully completed so that the rotary turntable can proceed to the next stage.

Capper/Lid loader

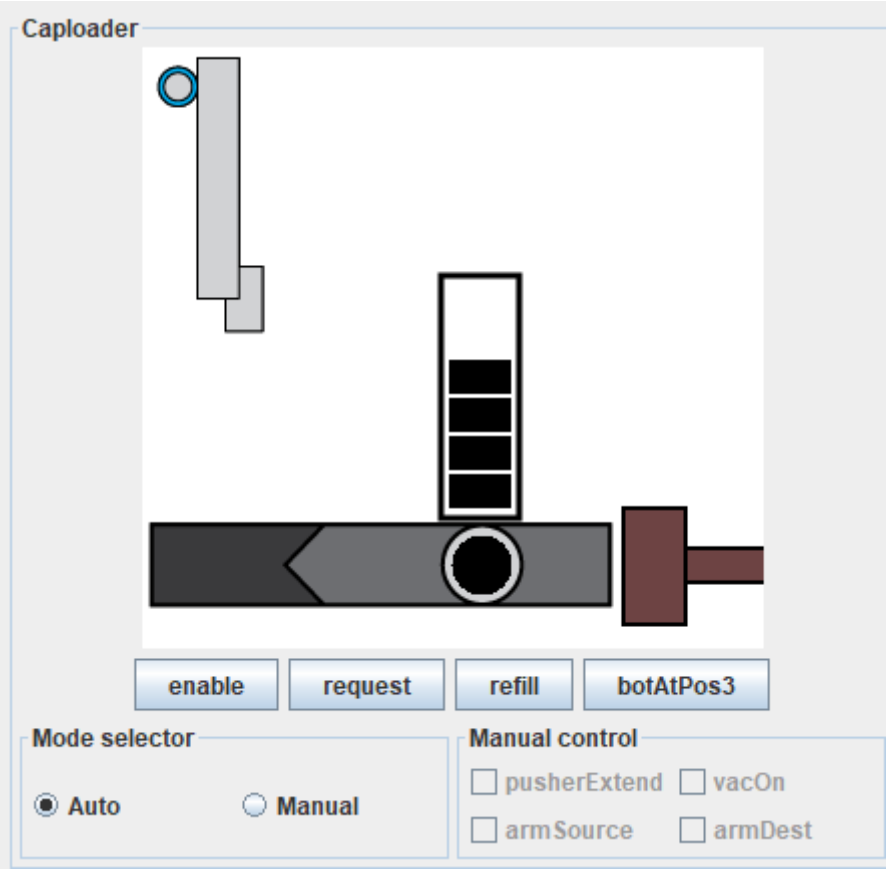

Figure 26. Caploader/LidLoader

The Capper/LidLoader implementation was based on the already implemented solution from COMPSYS 704 Lab 3.

IRP Selected Subsystems

Product Order System (POS)

The Product Order System (POS) forms the connection between the customers and the production system through a GUI. This allows authenticated customers to place custom orders which will be automatically sent to the production facility and processed automatically. The GUI will provide options of up to four different liquid options that can be selected by the customer to mix, as well as the number of bottles that are needed to be produced. The Product Order System (POS) is connected to the top-level main controller of the ABS (Automated Bottling System) which is able to pass the relevant signals to the subsystem controllers (such as the Filler Controller). In this way, the customer's order can be delivered directly to the manufacturing system. The transmission of signals is carried out by utilizing SimpleClient and SimpleServer classes.

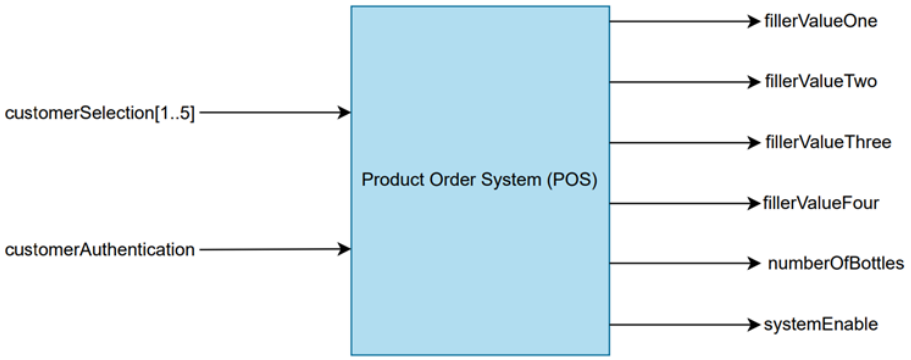

Figure 27. Input/Output POS signals

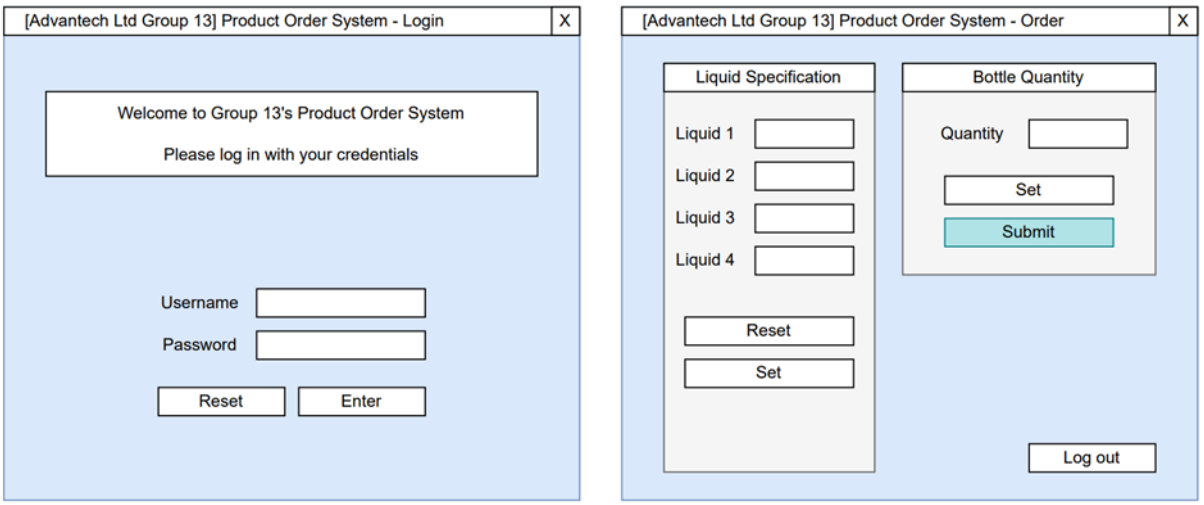

Figure 28. Mock POS GUI

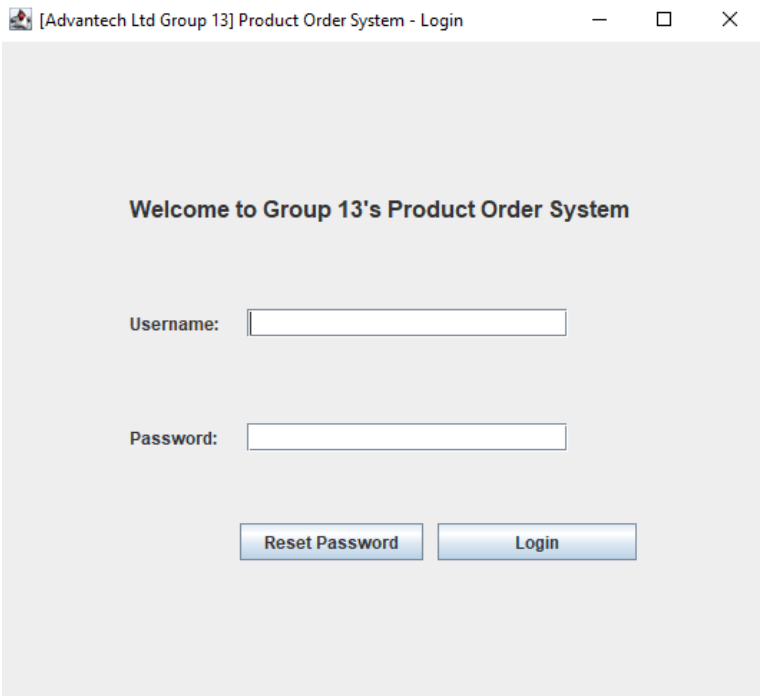

Figure 29. Implemented POS GUI: Login

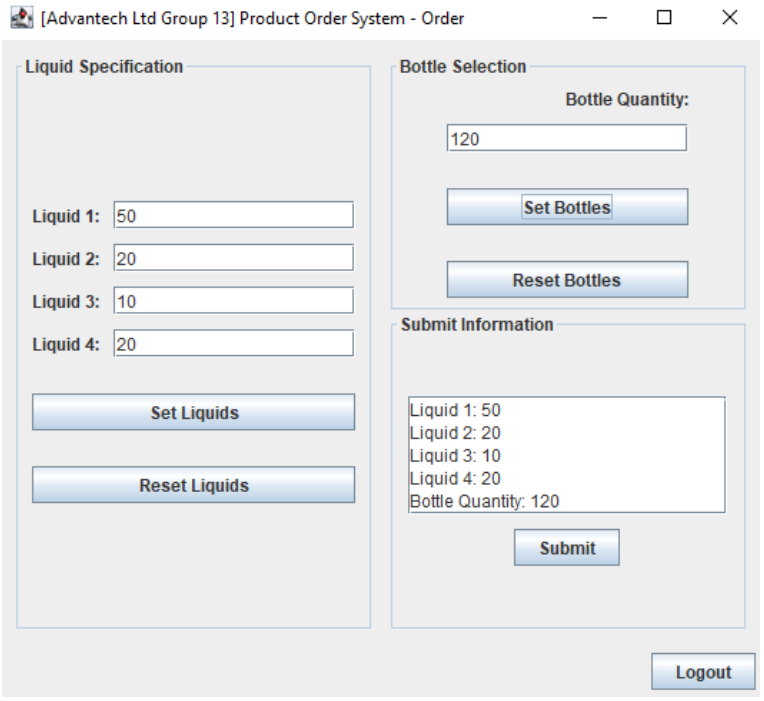

Figure 30. Implemented POS GUI: Order

```
------------Status------------
Liquid 1 Amount: 50
Liquid 2 Amount: 20
Liquid 3 Amount: 10
Liquid 4 Amount: 20

Bottle quantitiy left: 119
------------Status------------
```

Figure 31. Terminal print of received signals

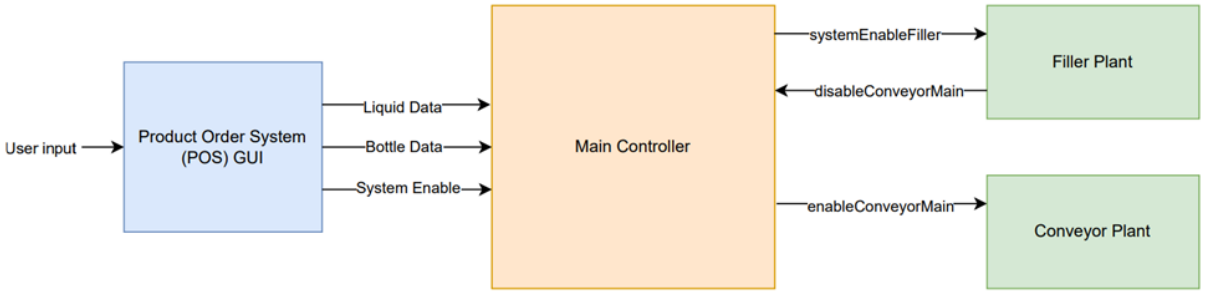

Figure 32: Signal connections between POS GUI and ABS

Environment Control System (ECS)

When designing an Automated Bottling System (ABS), it is essential to consider the environment control system to ensure an optimal working environment across different zones of a facility. ECS is in charge of maintaining temperature, humidity, and lighting at desired levels to provide a comfortable and safe environment. It is also responsible for detecting smoke and fire threats and triggering alarms when necessary. Please refer to appendices for detailed conceptual design diagrams in Figure A.6. More detail is provided in the IRP submission.

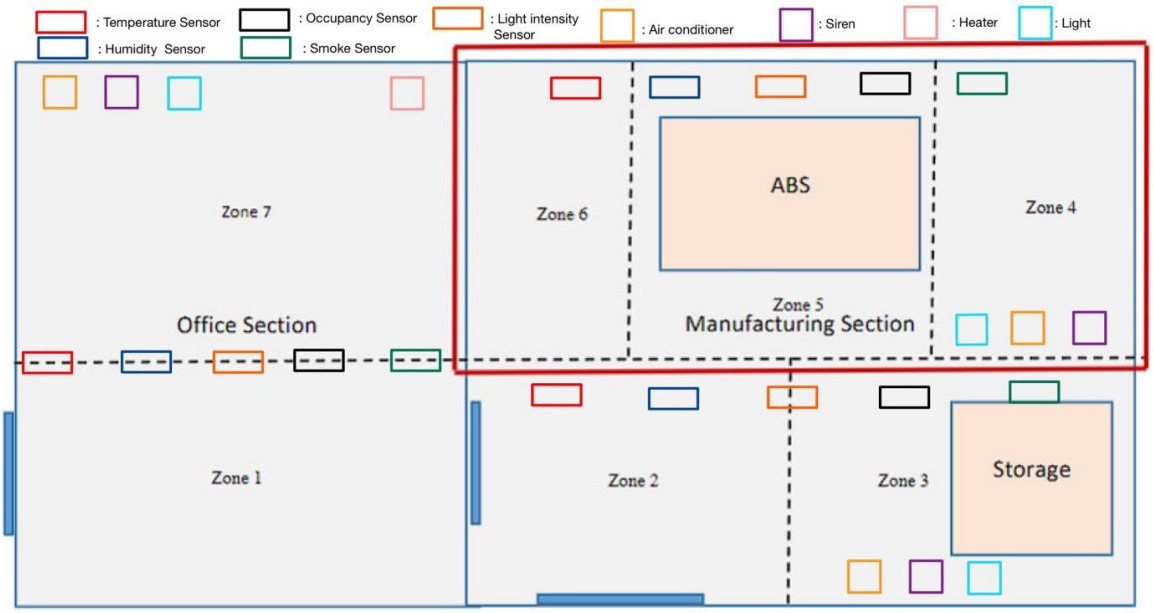

Figure 33. ECS different zones

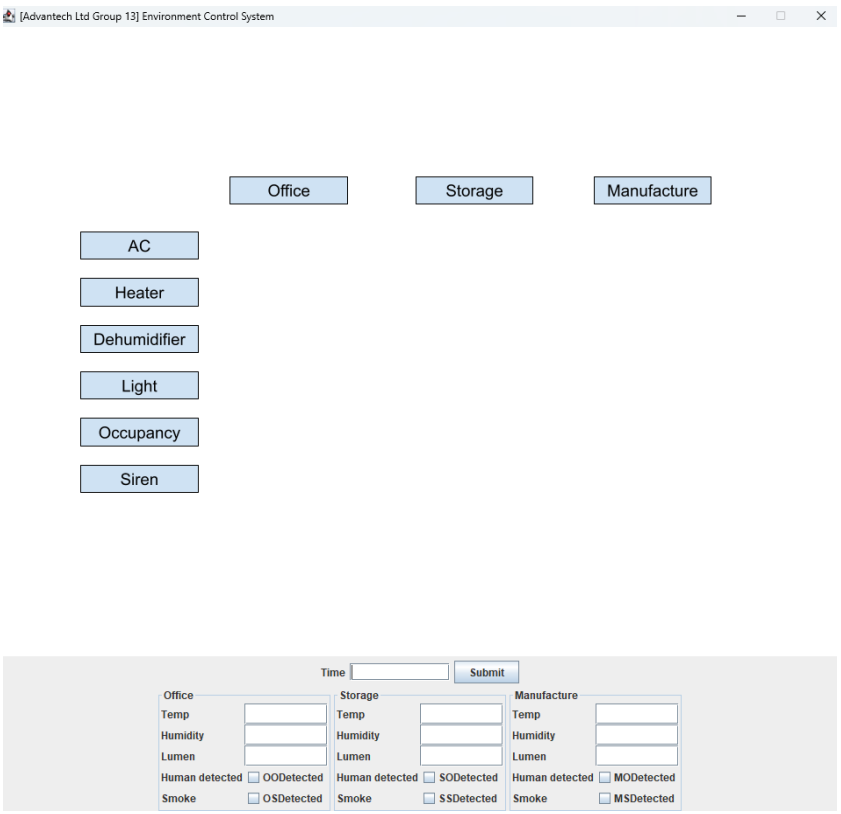

Figure 34. Implemented ECS GUI

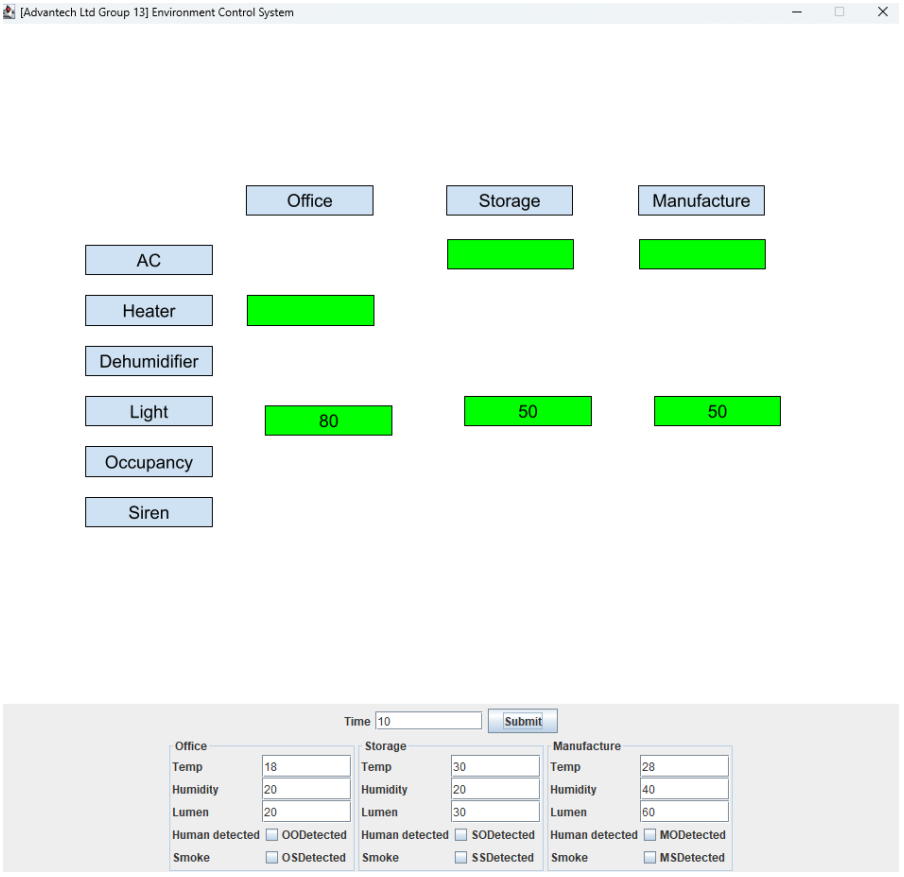

Figure 35. Implemented ECS GUI - office hour

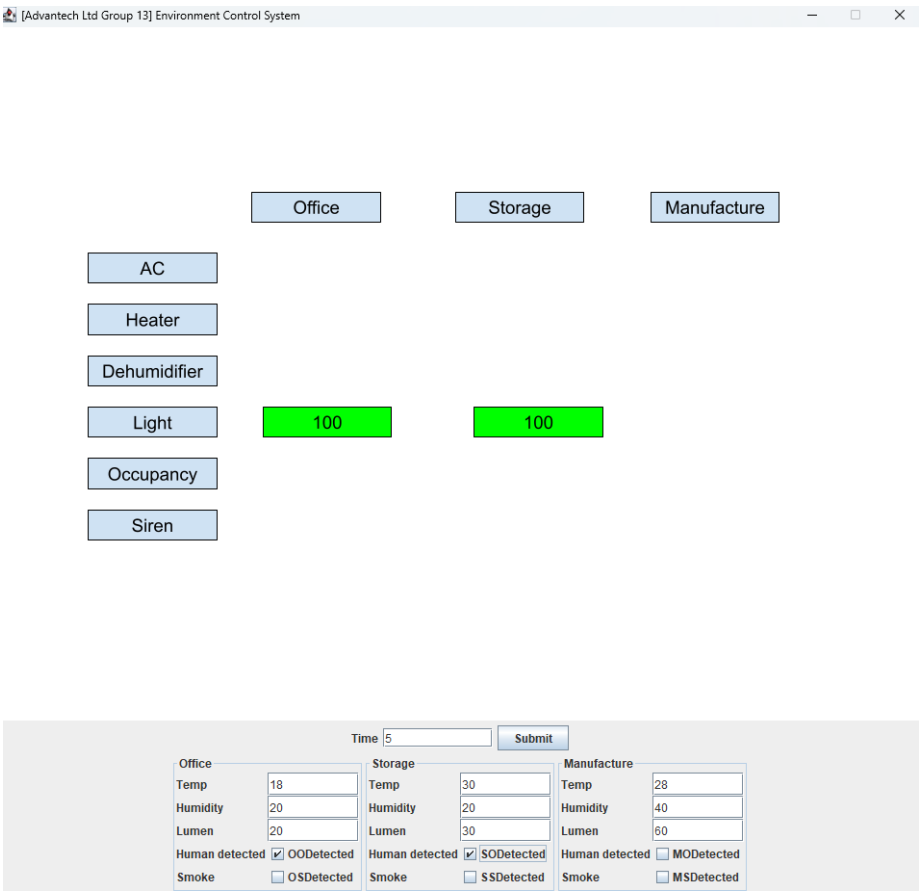

Figure 36. Implemented ECS GUI- non office hour

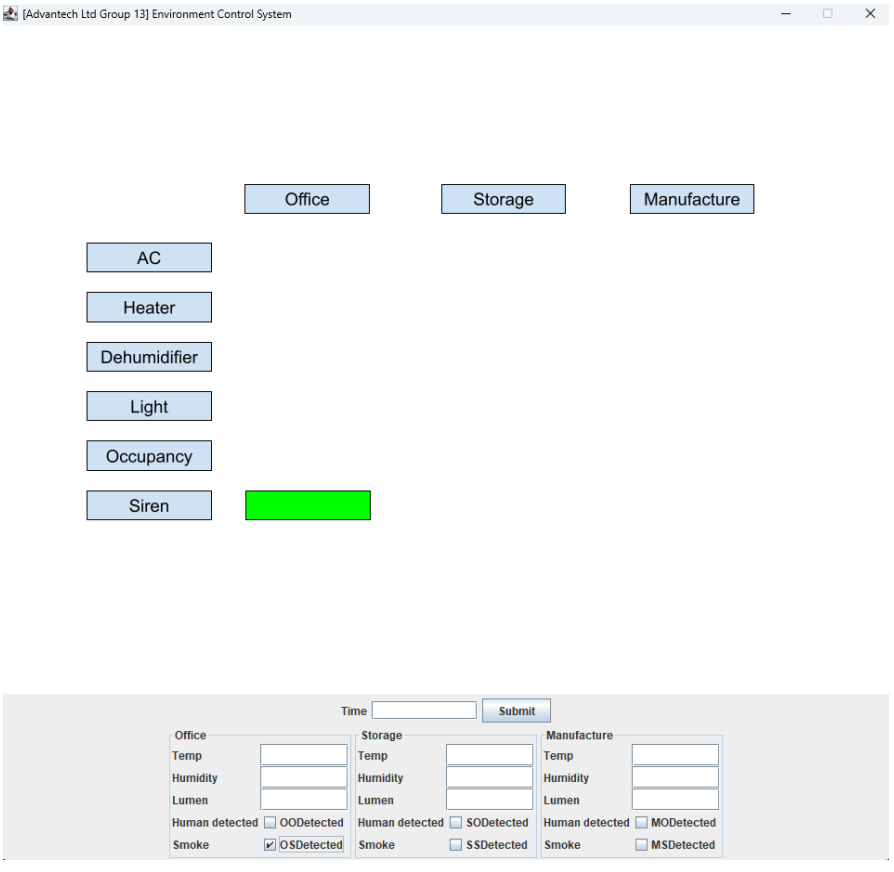

Figure 37. Implemented ECS GUI- smoke detection

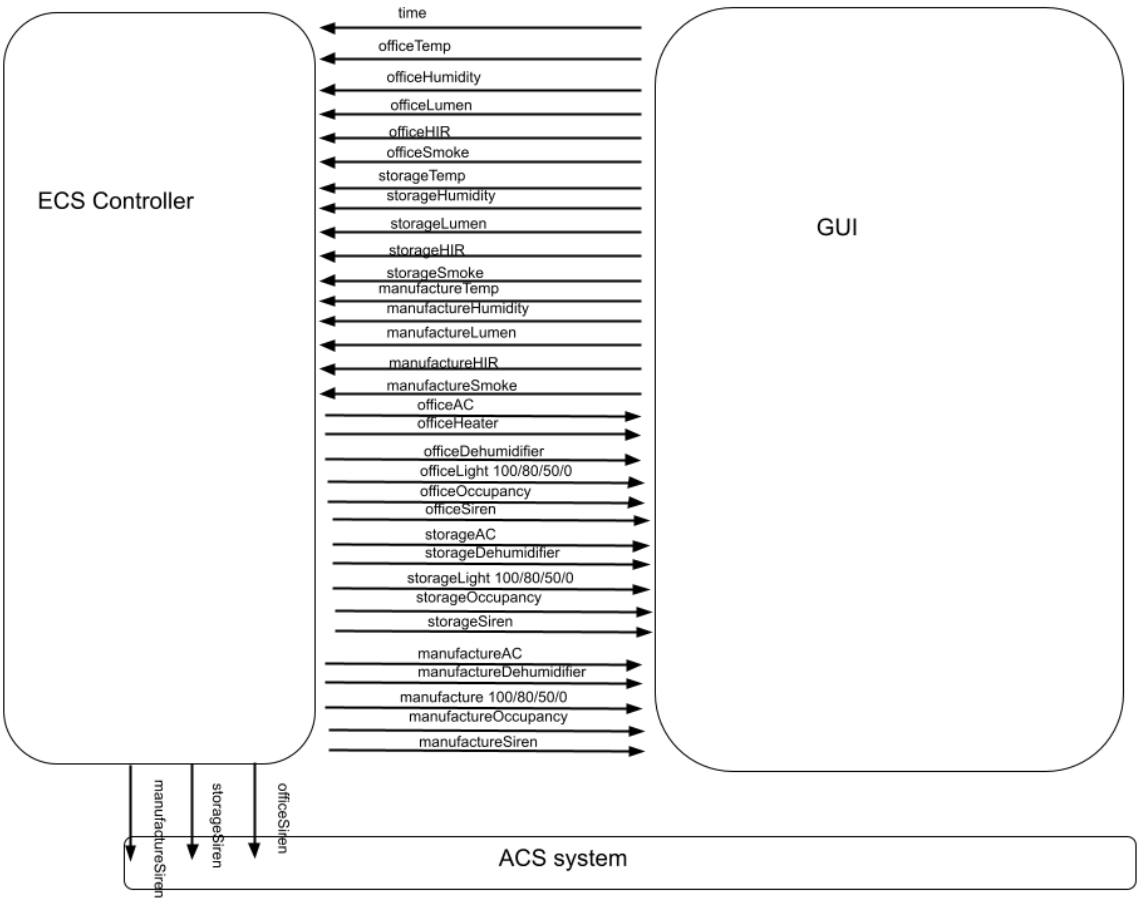

Figure 38. Diagram of ECS

Safety and Access Control System (ACS)

Safety and Access and Control System, SACS, that controls presence of people in both major sections of the facility and enforces both safety and security measures according to the adopted regulations and monitors safety of the environment in terms of presence of fire, humans in certain areas. The conceptual design is shown in Figure A.5. More detail is provided in the IRP submission.

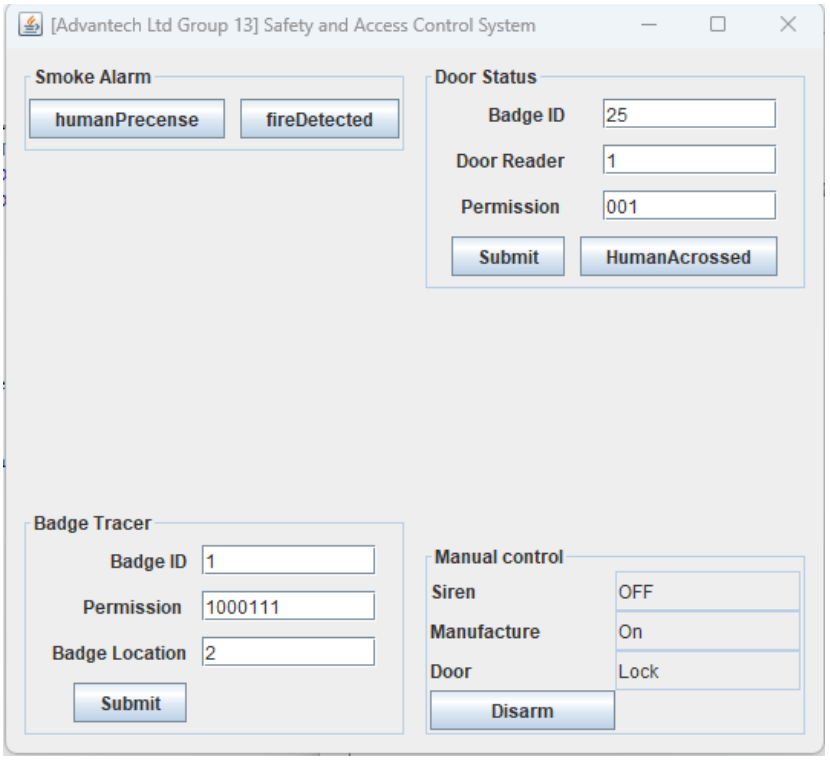

Figure 39. Implemented ACS GUI

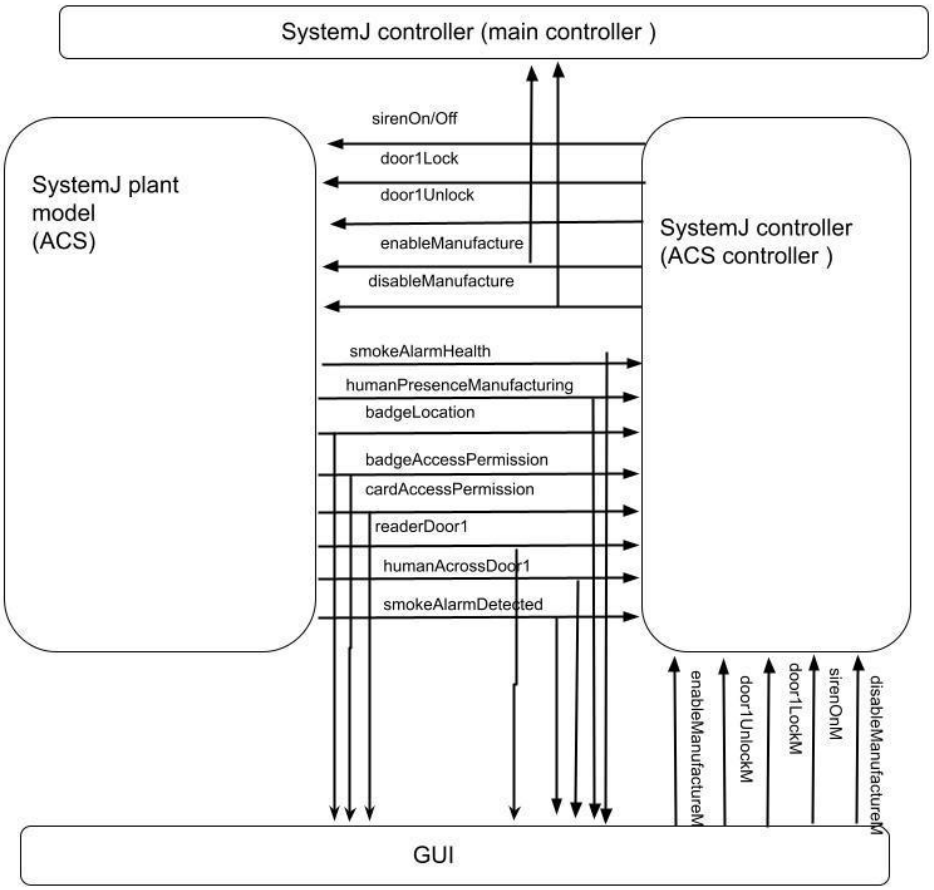

Figure 39. Diagram of SACS System