COMPSYS 704 Project 1

Final GRP Report – Milestone 2

Automated Bottling System (ABS)

Group 13

Francis Cho

Xuhan Liu

Gayeon Kim

Abstract

This report details Group 13's final design and implementation of the ABS, along with 3 other chosen subsystems (POS, ECS, SACS). The SystemJ design approach, along with Java, was utilised to model and design the distributed embedded systems of an Automated Bottling System. This effectively allowed a full system functional design without the need of the whole platform, along with powerful abstraction tools to allow concurrency, reactivity, and synchronisation of software behaviours. With all components and subsystems of the ABS successfully implemented, the user can use the Product Order System (POS) GUI to place an order detailing the bottle quantity and liquid specifications which will automatically begin the bottling process at the ABS facility. This final design report showcases Group 13's integration and implementation of the Automated Bottling System (ABS), along with the subsystems.

Introduction

As part of the company Advantech Ltd.'s goal of building a new automated bottling facility that involves advanced monitoring and control systems, we (Group 13) have been tasked with designing and integrating new systems into the already existing process of producing customised liquids. These systems include the Automated Bottling System (ABS) which is the main manufacturing component as well as other control and monitoring systems such as: Environment Control System (ECS), Safety and Access and Control System (SACS), Energy Consumption Optimisation System (ECOS), and Product Order System (POS). This is accompanied by multiple, detailed Graphical User Interfaces (GUIs) that have been developed using Java in order to visualise each subsystem of the Automated Bottling System (ABS) providing the user with adequate information of the inner processes of the entire system as a whole. In addition to this, separate GUIs have been developed for the Group 13's chosen IRP subsystems (POS, SACS, ECS), which will also interface and interact with the ABS implementation.

Automated Bottling System (ABS) Final Overall Design

ABS Final Design

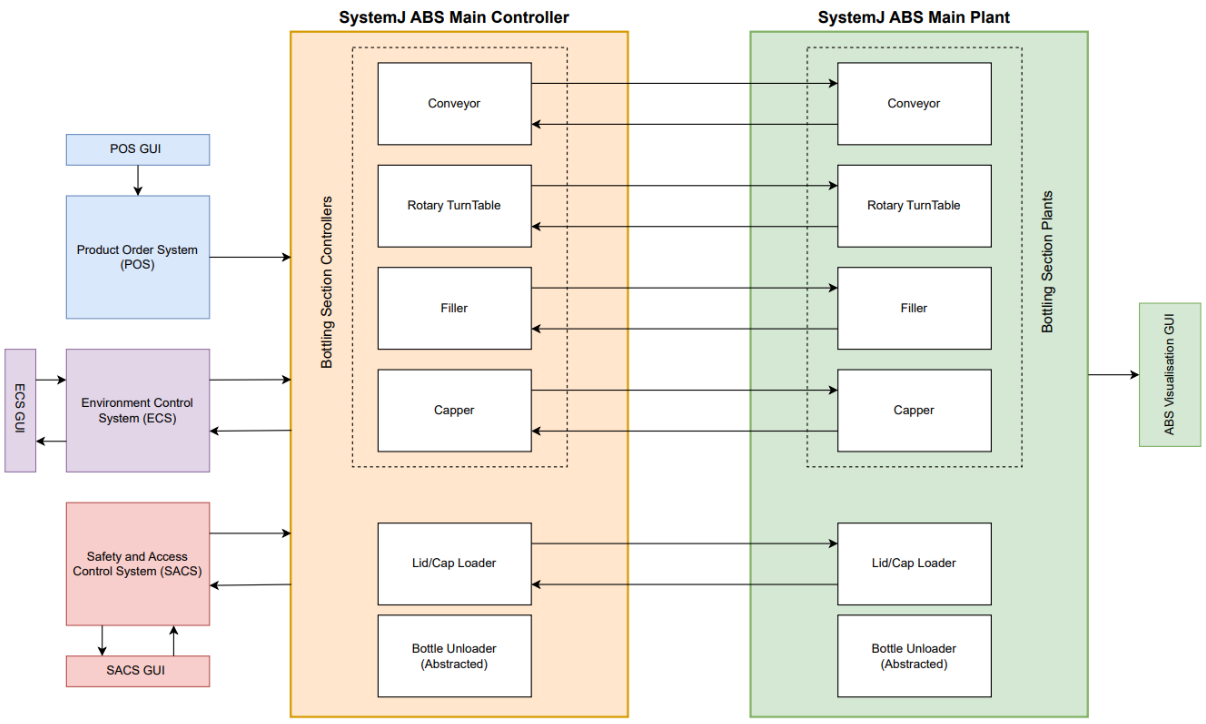
SystemJ ABS Main Controller
SystemJ ABS Main Plant
POS GUI
Product Order System (POS)
ECS GUI
Environment Control System (ECS)
Safety and Access Control System (SACS)
SACS GUI
Bottling Section Controllers
Conveyor
Rotary TurnTable
Filler
Capper
Lid/Cap Loader
Bottle Unloader (Abstracted)
Bottling Section Plants
ABS Visualisation GUI

Figure 1: Top level diagram of the entire Automated Bottling System (ABS)

There are a total of six different clock domains utilised in the main ABS design: Conveyor, Rotary Turntable, Filler, Capper, Cap/Lid Loader, MainController (top-level). In addition to this, the IRP subsystems ECS and SACS will have their own clock domains. The POS subsystem will utilise the clock domain of the ABS Main Controller. Each component uses a model-driven approach to model both the machine (also known as Plant), with its sensors and actuators (the controller) to generate proper control signals in order to bring the machine and its mechatronic components into the desired states.

It was decided that if all the clock-domains are implemented to run on a single machine (also known as centralised systems) usually require higher cost to build. Furthermore, a disruption in one subsystem may impact on the entire production process leading to delays due to maintenance downtimes. Therefore, we have divided the ABS clock-domains into their respective subsystem separations of Conveyor, Rotary Turntable, Filler, Capper, Cap/Lid Loader. This decentralised design effectively allows Advantech Ltd. engineers to maintain the facility easily, potentially avoiding entire system shutdowns.

In order for our SystemJ modelled controller and plant systems to communicate with each other, we utilised SystemJ Net API (as known as *sysjnetapi*). The *sysjnetapi* library effectively simplifies the process of TCP socket communication by abstracting away all the socket implementations and protocols giving us an easy to work with signal communication solution between our SystemJ clock-domain programs. Throughout our implementation, we utilised *SimpleClient* and *SimpleServer* classes from the IPC classes provided by SystemJ Runtime Environment (RTE/RTS) to handle signal communications. We have declared input and output signals (iSignal and oSignal) for each of our Plant's and Controller's clock domains. To send a signal from a subsystem (output), we would require the clock domain address (e.g. "*ConveyorControllerCD.bottleAtPos1*") and the relevant IP and port information (e.g. "*127.0.0.1*" and "*40000*"). *SimpleClient* class is used for outbound signal communications (e.g. *Class="com.systemj.ipc.SimpleClient"*), whereas for inbound signal communications the SimpleServer class was used (e.g. *Class="com.systemj.ipc.SimpleServer"*).

The overall high-level diagram shown in Figure 1. showcases the relationships between each controller and plant systems. The bottling section controllers (Conveyor, Rotary TurnTable, Filler, Capper) along with the lid/cap loader controller are encapsulated by the Main ABS controller. The top-level main controller communicates with the other subsystems in our design such as POS, ECS, and SACS, as well as our Unified ABS visualisation GUI. The bottle unloader (i.e. Baxter robot) is abstracted so connections are hidden.

Design of Automated Bottling System (ABS) Systems

Conveyor

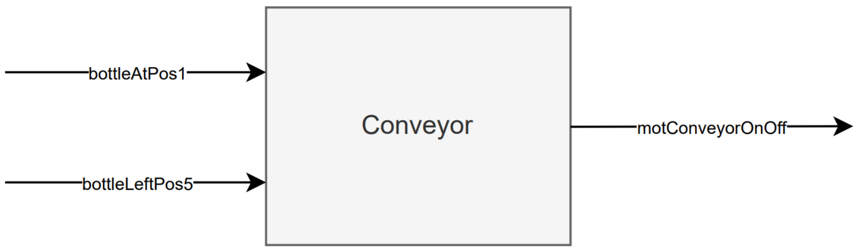

Figure 2: Input and output signals of the Conveyor clock-domain

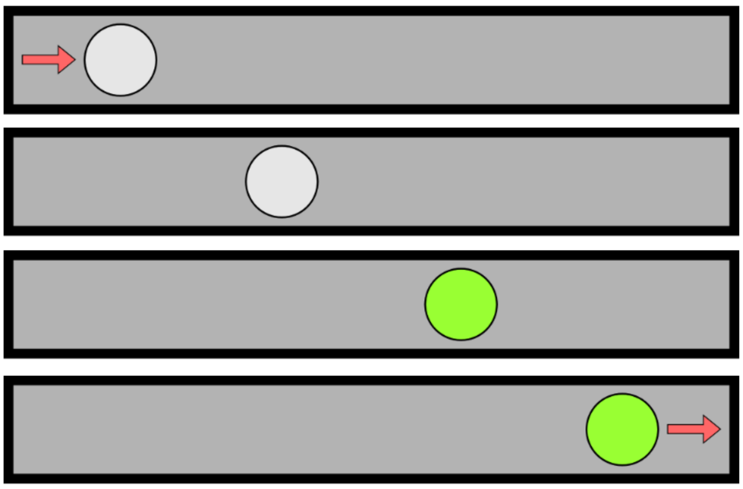

Figure 3: Conveyor system in sequence of events

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.

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.

Rotary Turntable

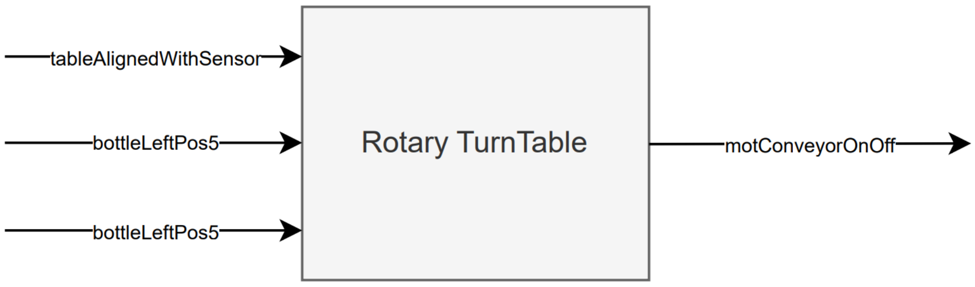

Figure 4: Input and output signals of the Rotary TurnTable clock-domain

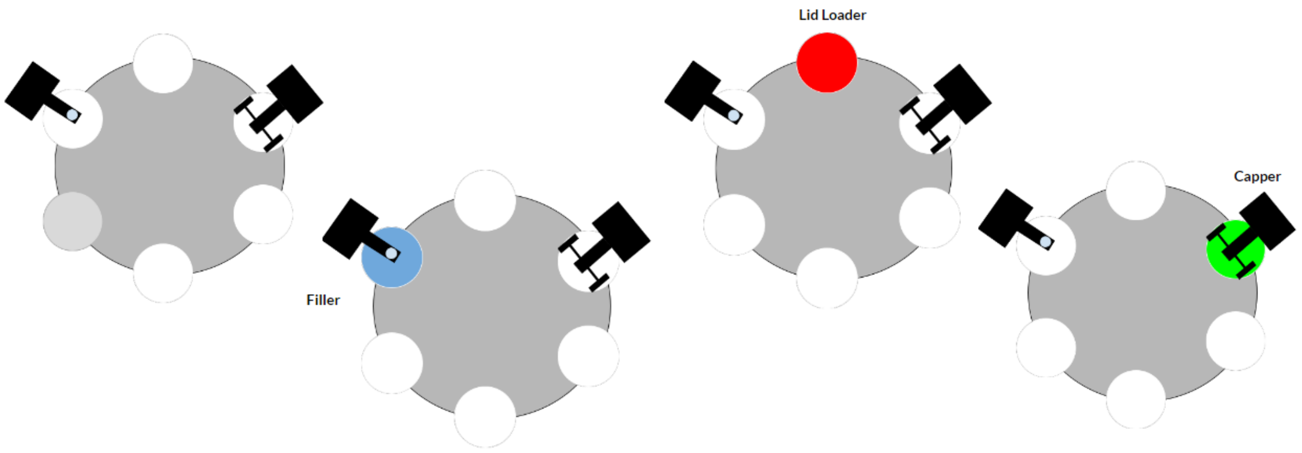

Figure 5: Rotary Turntable different sequence

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.

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

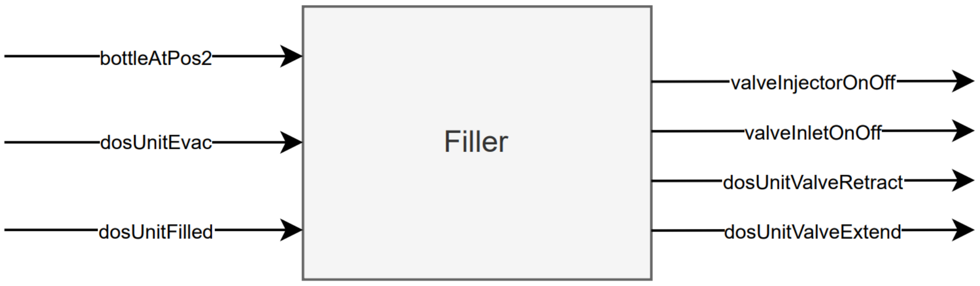

Figure 5: Input and output signals of the Filler clock-domain

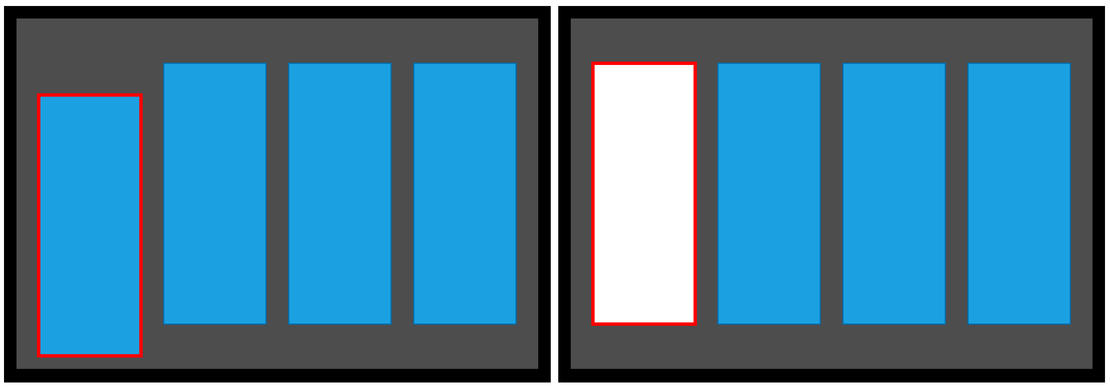

Figure 6. (Left) Selected Filler tube is extended (Right) Selected Filler tube is retracted after being emptied

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 which could be determined by the *bottleAtPos2* signal. By emitting the *valveInjectorOnOff* signal, the valve injector will be turned on. If the *dosUnitEvac* signal is present which means a pressure canister is at bottom, it should emit the *dosUnitValveRetract* signal to bring the pressure canister to the top to fill the bottle with liquid. Then it waits for the *dosUnitFilled* signal to be present which indicates that the cylinder is fully retracted (pressure canister has reached the top). Next, the injector is turned off which is controlled by the absence of the *valveInjectorOnOff* signal. The *valveInletOnOff* signal is emitted to open the inlet valve and *dosUnitValveExtend* signal is emitted to push the pressure canister down. The system should wait until *dosUnitEvac* signal to be present which indicates that the cylinder is fully extended, then close the inlet (absence of *valveInletOnOff* signal). It was assumed that only one bottle was filled at any given time. Some other assumptions included: the operations are sequential and cannot overlap, components of the Filler system work without any failure, a bottle is kept at the same position while being filled until all liquids are filled.

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

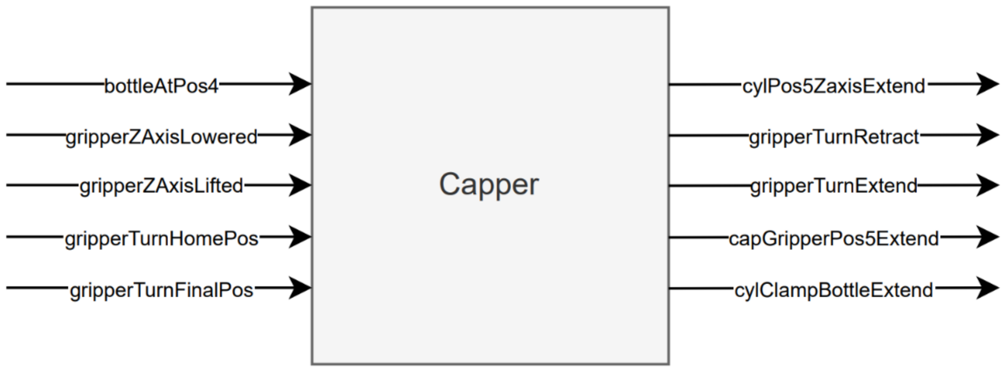

Figure 7: Input and output signals of the Capper clock-domain

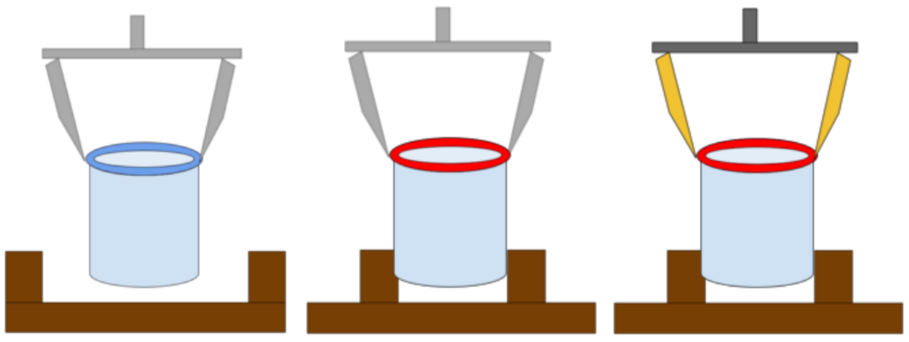

Figure 8. (Left) Capper gripper lowered (Middle) Capper bottle clamped and cap gripped (Right) Capper gripped twisted

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 which is indicated by the *bottleAtPos4* signal. The *cyClampBottleExtend* and *cylPos5ZaxisExtend* signals should be emitted in the presence of the *bottleAtPos4* signal in order to clamp the bottle and lower the gripper. When the gripper and capper are fully lowered, the *gripperZAxisLowered* signal will be present to indicate the lowered state. When the *gripperZAxisLowered* signal is present, the *capGripperPos5Extend* signal should be emitted in order to grip the cap. Furthermore, the *gripperTurnExtend* signal should be emitted to start twisting/turning the gripper. When the gripper has fully turned and the capping action completed, the *gripperTurnFinalPos* signal will be emitted. This will stop emitting *capGripperPos5Extend* to release the cap. In order to untwist the gripper, the gripperTurnRetract needs to be emitted by the system. When the gripper has returned to its initial position, the *gripperTurnHomePos* signal will be present. Finally, the gripper is raised by stopping the *cylPos5ZaxisExtend* signal emission. When the gripper is fully raised before proceeding to the next bottle, the *gripperZAxisLifted* signal will be present.

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 GUI for ABS visualisation

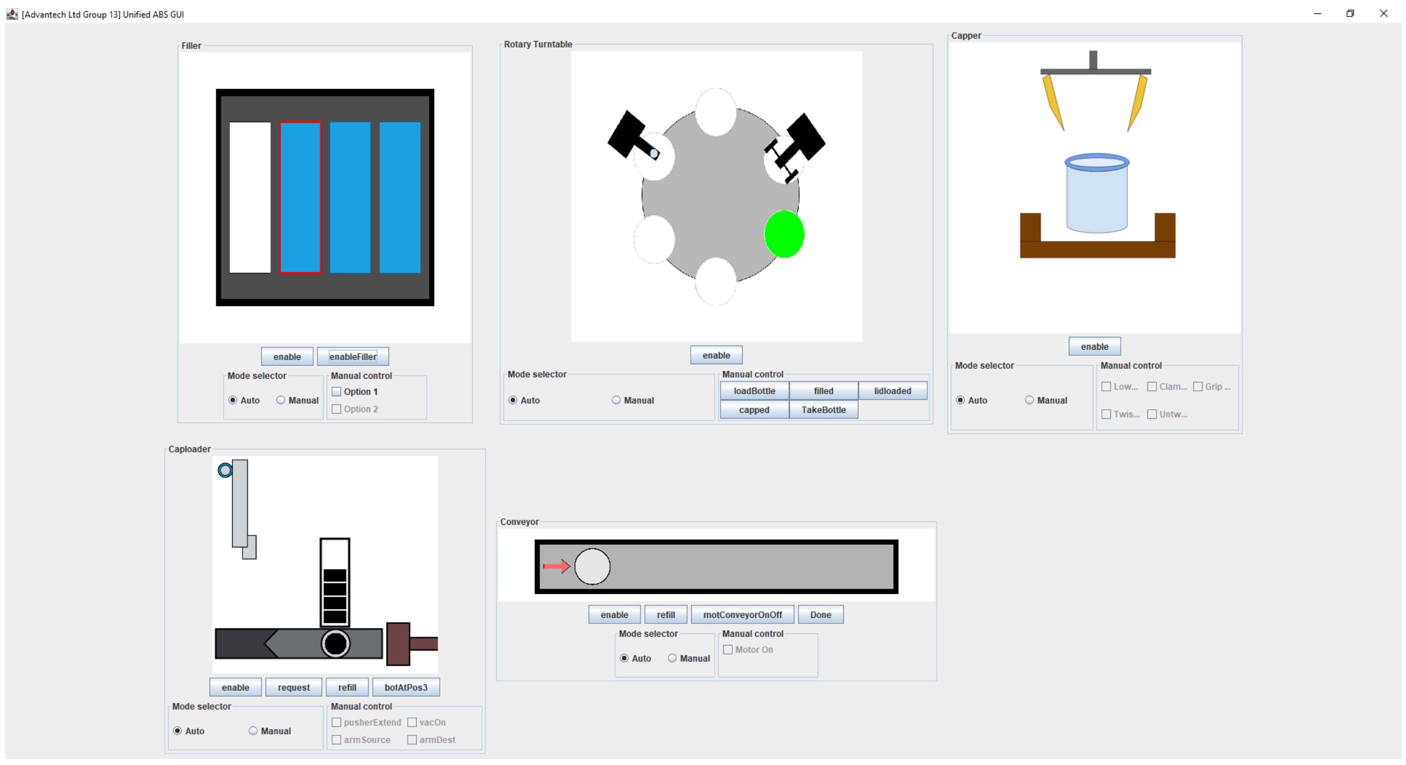

Figure 9. Automated Bottling System (ABS) Unified GUI Implementation

A Graphical User Interface (GUI) is an effective way to display the current status of the systems. The Unified ABS GUI, shown in Figure 9, takes into account the status of each component of the ABS - (1) Conveyor (2) Rotary Turntable (3) Filler (4) Capper (5) Cap Loader. The status of each position will update on the GUI depending on which signals it has been received from the other subsystems. 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. 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.

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.

The Unified ABS GUI is extremely beneficial as it allows the user to have a high level overview of all the ABS systems working in real time. Although there are both auto and manual modes present in each ABS GUI (along with multiple options), only the auto mode has been thoroughly tested to work seamlessly from start to finish. The manual modes were mainly used for debugging and testing. The usage of our implemented solution is very simple. When the program is launched, it will open both the Unified ABS GUI and the POS GUI. The user may log into the POS GUI and then place their order (with specified bottle quantity and liquid selection) and this will automatically signal the ABS facility to begin the bottling process until all the requested bottle quantities in the order are complete. The user may make a new order once their previous order has been completed by the ABS facility.

IRP Selected Subsystems

POS - Product Order System

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 POS will be connected to the top-level ABS controller where signals can be effectively distributed to the appropriate subsystem. More detail is provided in the IRP submission.

ECS - Environment Control System

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.

SACS - Safety and Access Control System

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.