Distributed and Intelligent Automation Systems

Assignment 1

Valtteri Nikkanen **282688**

Lauri Nuutinen **283219**

Väinö Kurvi **274490**

Table of contents

Task 1 – Traffic Lights

Task 1a was to implement a traffic light system using a centralized and a decentralized approach. Task 1b was to add a wait button to the system. In this traffic light system, we have two semaphores that we need to control. In figure 1 we show a mockup of the problem scenario given in the problem description.

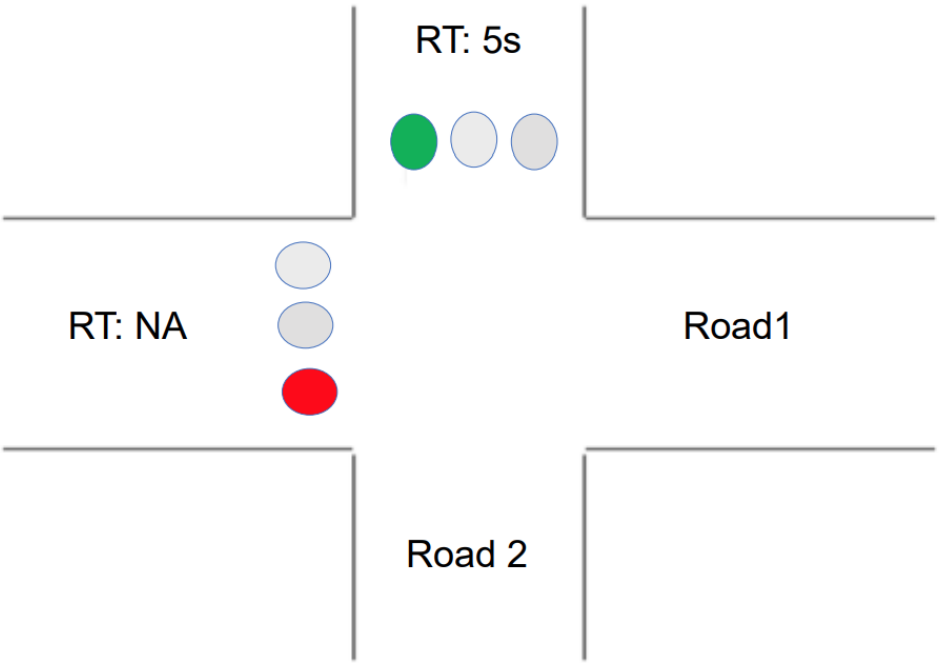

Figure 1: Traffic Lights

We know that only one of the lights can be green at a time. When the other one is green the other one needs to be red. And before turning from red to green the light needs to go through yellow light. For the green and yellow light there also needs to be a timer that shows how long the current light stays on before switching to the next one. The times that the green and yellow light stay on also need to be configurable by the user through an HMI.

Real traffic lights probably are in communication or are designed with the larger traffic network in mind to achieve the best possible traffic flow. Our traffic light controller however is just a simple system that can handle a singular intersection with no regard to anything outside of it. They also stay both in red for a small time before the next one changes to green to let everyone in the intersection to pass safely. Our proof-of-concept system is not doing this as it only needs to show that the logic works without the need for any unnecessary delays.

The system has some non-functional requirements in addition to the functional requirements that we have already seen. The traffic light system needs to be:

- Robust/Reliable,
- Usable,
- and if it would go to real use it would need to be secure and regulatory.

Scalability is also a good non-functional requirement which we can achieve with the decentralized versions of the system much easier than with the centralized version.

The functional requirements for the system are:

- User needs to be able to configure the times for the green and yellow lights for both semaphores.
- The view displays the time that the green or yellow light is going to stay on before changing.
- When one semaphore changes to red light the other changes to green light.
- The lights come on in the correct order every time.
- The user can see the states from the view layer of the system.

The rough use case diagram for the system is presented in figure 2. Some changes are made in the different versions of the system as some changes need to be made to get the desired functionality.

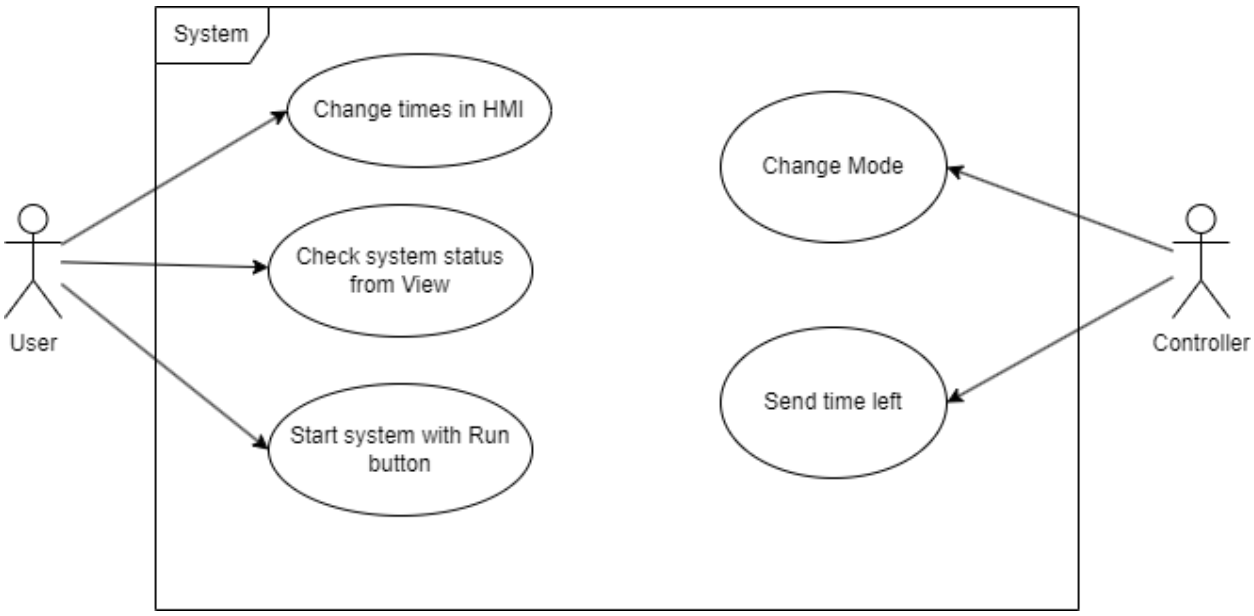

Figure 2: Use case diagram for the traffic light system

Centralized version

The centralized version uses a central controller that keeps track of the statuses of the traffic lights and gives appropriate commands to the semaphores. The controller cycles trough four states, green1, yellow1, green2 and yellow2. No states are needed for red lights as a red light is what is shown when no other state is active in a semaphore. In the centralized approach the semaphores have no other purpose than to display the light that the controller is currently ordering. In figure 3 is the architecture and components of our centralized system. In figure 4 we are presenting the class diagram of our centralized traffic light system. And in figure 5 is the sequence diagram for the centralized system. Figure 2 can be used as the use case diagram for this version of the system.

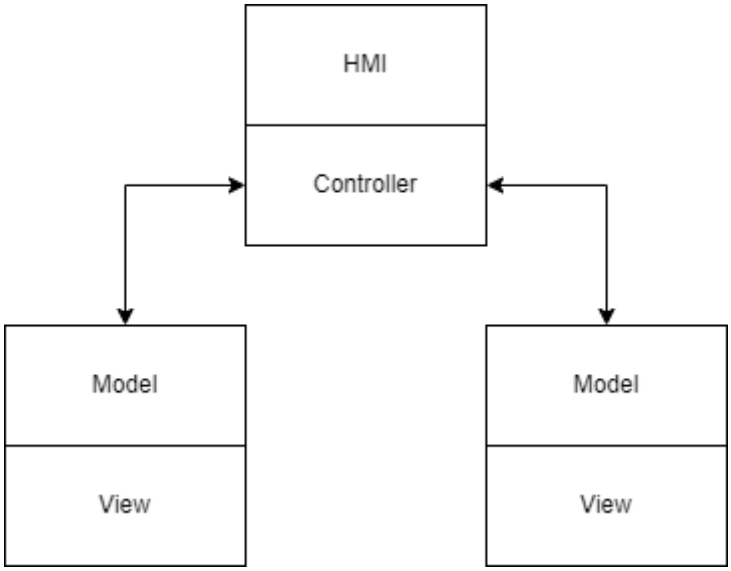

Figure 3: The diagram shows the components and architecture of the centralized solution.

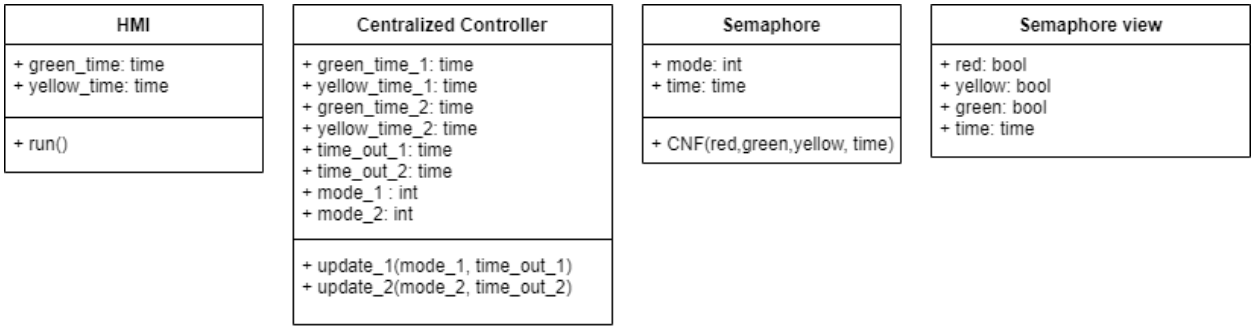

Figure 4: Class diagram of the centralized system.

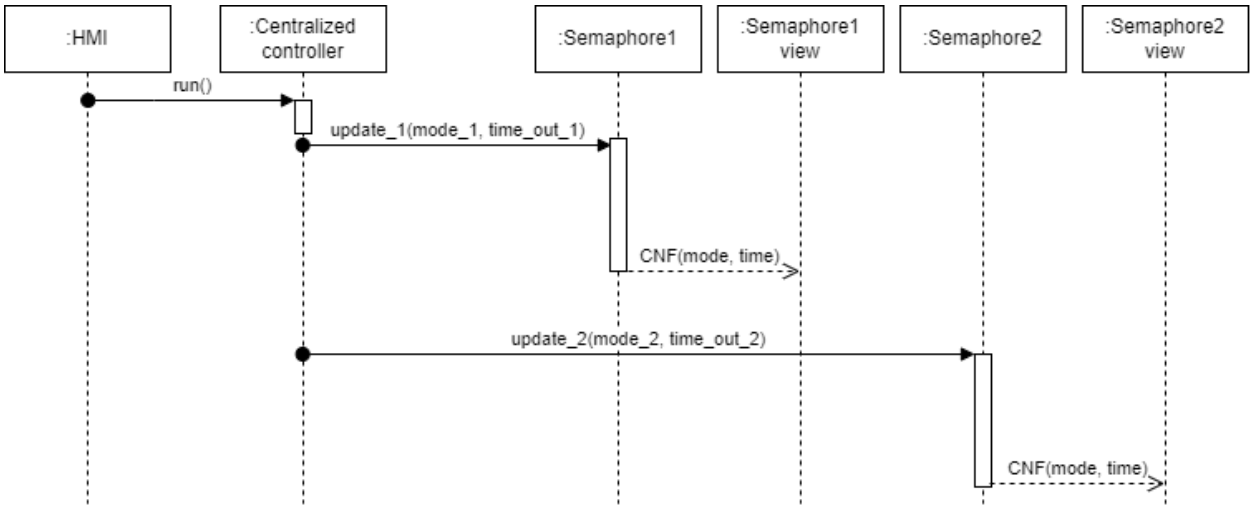

Figure 5: Sequence diagram of the centralized system.

These diagrams should give an adequate explanation about our design principles for the centralized version of the traffic light system.

Decentralized version

The decentralized solution has two identical traffic lights. They are connected to one another through their controllers from which they call one another to activate when they are done. This means that the solution is extremely scalable with minimal setup needed. The semaphore model is still the same as in the centralized version, but the controllers are different to facilitate the decentralized nature of the solution. In figure 6 the architecture and components of the decentralized system is presented. In figure 8 is the sequence diagram of the system and finally in figure 9 is the use case diagram of the system.

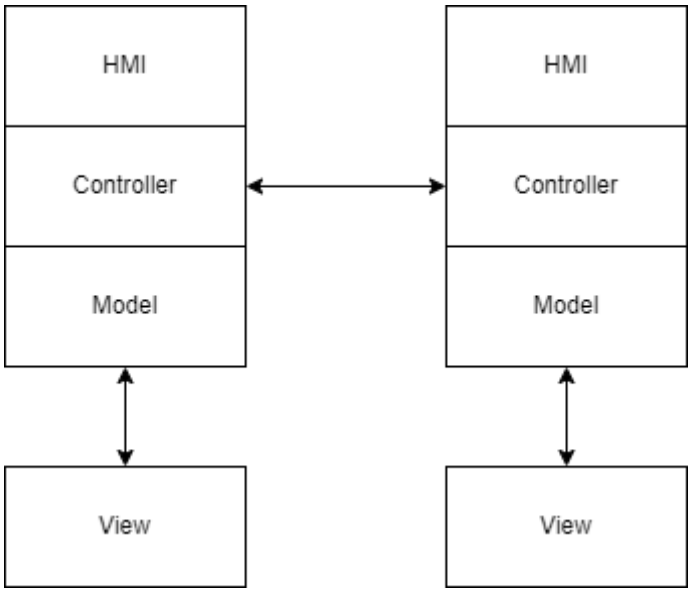

Figure 6: The diagram shows the components and architecture of the decentralized/distributed solution.

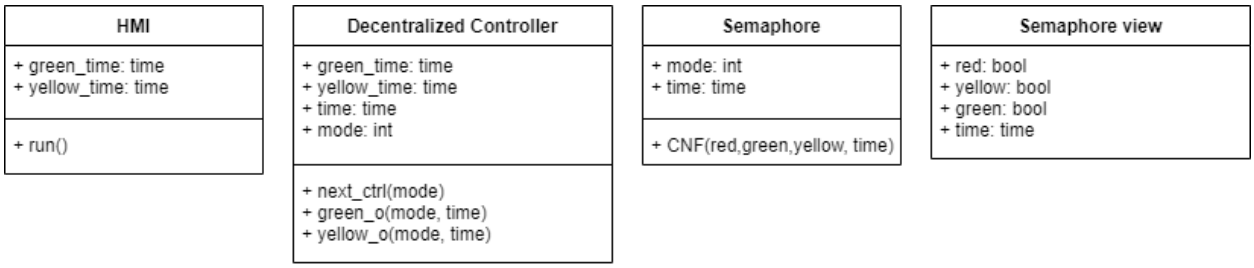

Figure 7: Class diagram of the decentralized system.

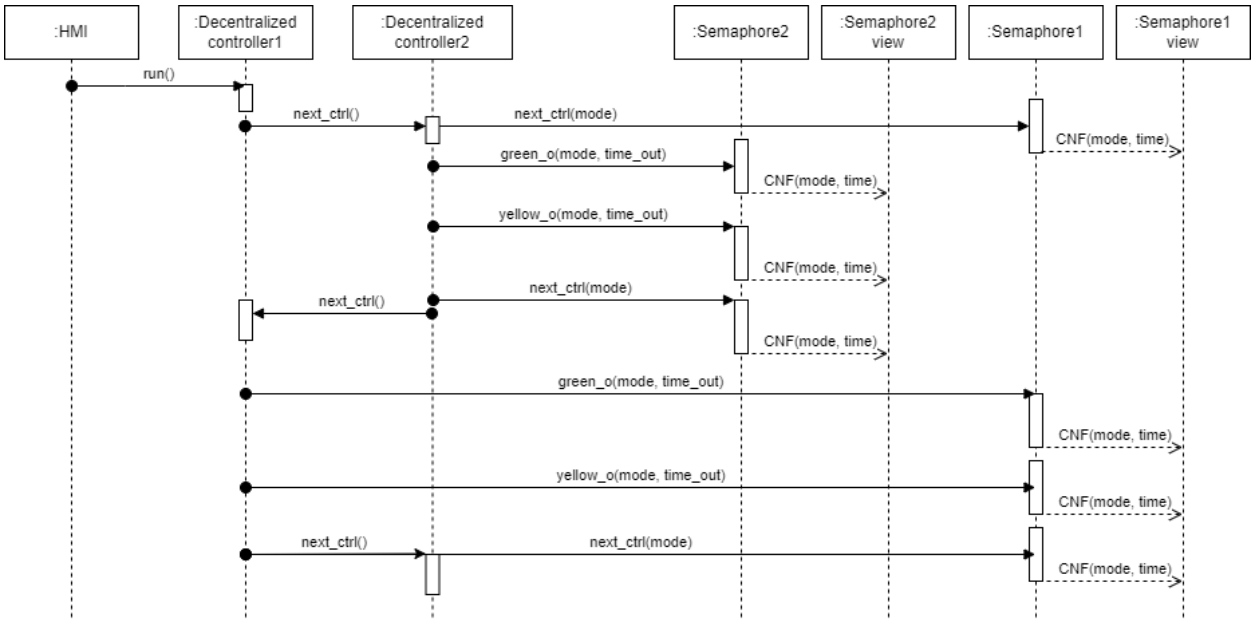

Figure 8: Sequence diagram of the decentralized system.

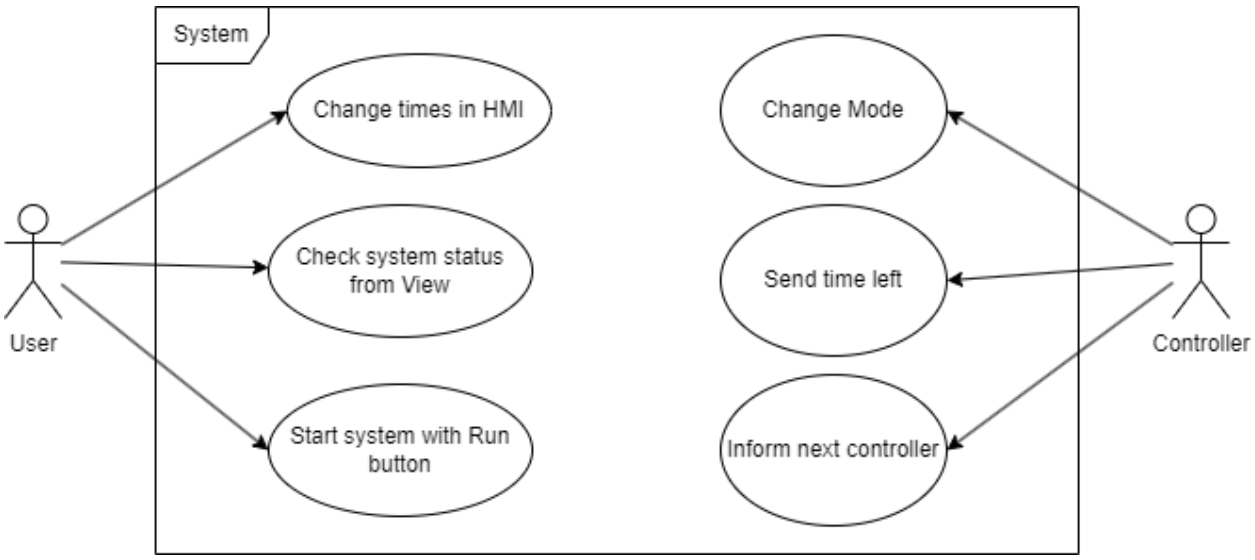

Figure 9: Use case diagram for the decentralized version of the system

Decentralized with wait button

The version with wait buttons works similarly to the previously presented decentralized version, but there is an extra button in the HMI which the user can press. If the button is pressed four times and the other semaphore has been on green light for at least four seconds the other semaphore changes directly to yellow light to make the current semaphore turn green quicker. It uses the same architecture and components as the version without wait buttons in figure 6. The updated use case diagram is in figure 10. The class diagram of the system is also unchanged except for the wait button in the HMI this is presented in figure 11. And finally, the sequence diagram of the wait button system is shown in figure 12.

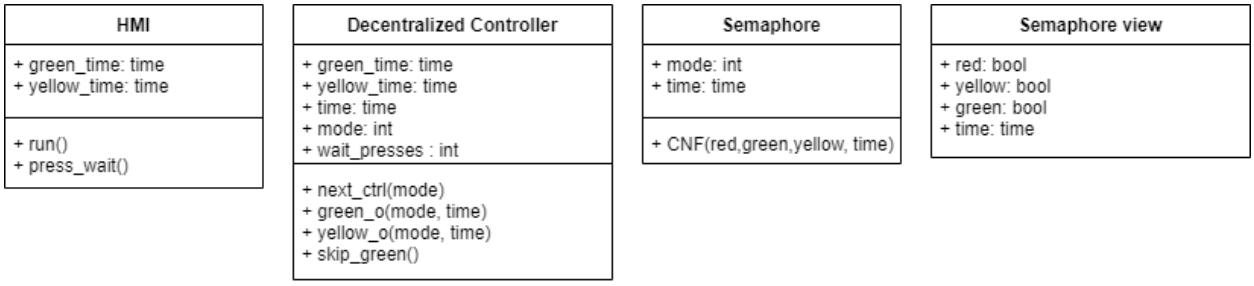

Figure 10: Class diagram of the decentralized system with button.

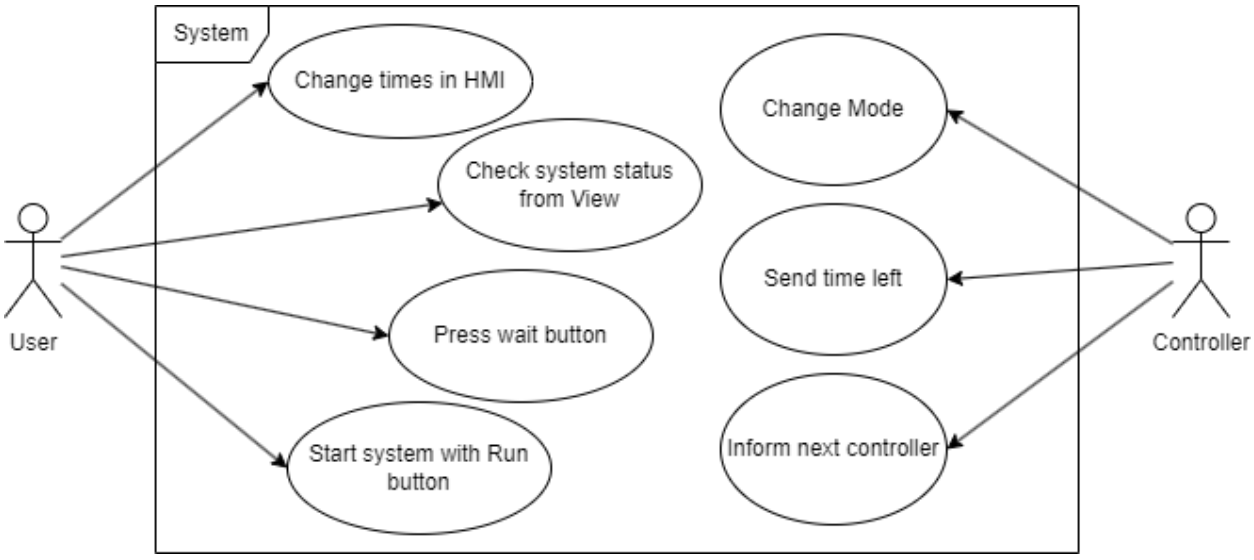

Figure 11: Use case diagram for the decentralized version with the wait button

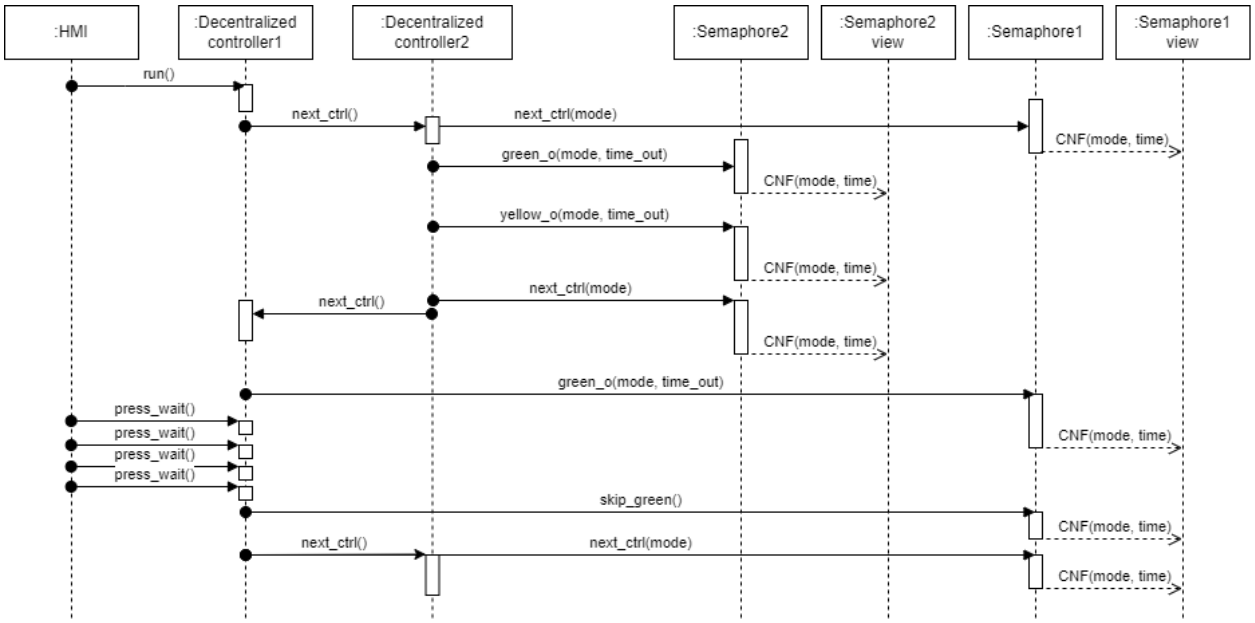

Figure 12: Sequence diagram of the decentralized system with button.

Task 2 – Conveyor

The task was to design a conveyor system that could sort the products on it based on their characteristics. The layout of the system was given in the task description and is shown in figure 13.

The items start at the beginning of conveyor 1. There can be at least three different coloured items that need to travel the system and be sorted correctly by it. The system then pushes the item to the end of the conveyor 1 where there is a turntable. If the turntable is empty it loads the item from the conveyor 1 on it and turn to face the conveyor 2. If conveyor 2 is running it unloads the item onto it and turns back to face conveyor 1 to load the next item.

There also needs to be automatic and manual modes for the system. In manual mode the control layer of the system should be disabled and the outputs of the system can be set manually by the user. In automatic mode the only thing the user needs to do is to select the colour of the item and loading it to the system.

The system also needs a view layer that shows the state of the system similarly as it is presented in figure 13. This view layer needs to be able to animate the products on the conveyors and their colour. It should also show the status of the turntable and the pushers.

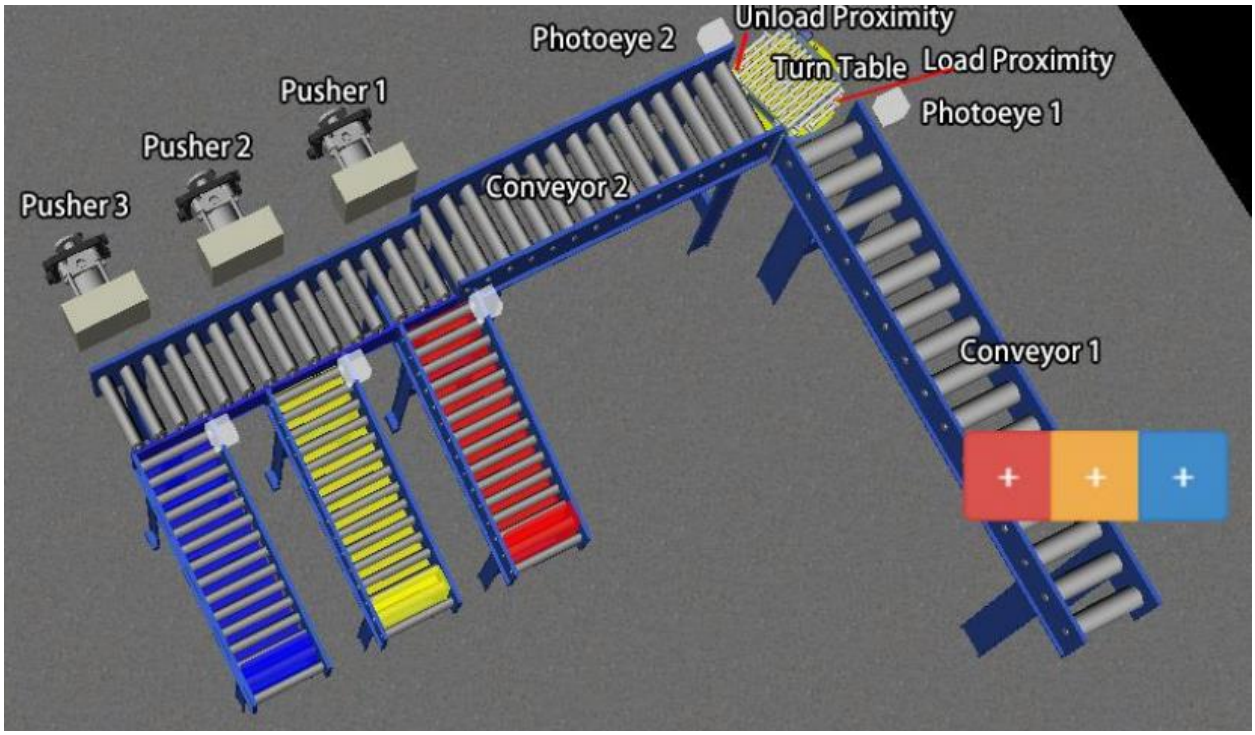

Figure 13: Layout of the system

The system has some non-functional requirements in addition to the functional ones that can be inferred from the problem definition. Similarly to the system in task 1 this one needs to be:

- Robust
- Usable

The functional requirements on the other hand are:

- The system should take inputs from the user through an HMI.
- It should also work automatically so that the only inputs that are required from the user is the loading of one of the three coloured items
- The view layer should visualize the state of the system.
- Turntable should only take one item at a time, it should only load from the first conveyor if the item is at the end position and it should only unload the item to the second conveyor if it is running.
- There should be an automatic and a manual mode. In manual mode the systems control layer should be disabled and the outputs be set manually by the user,
- A reset button should remove all the products from the system and restart the system.

The rough architecture and component breakdown is presented in figure 14. The aim was to create clearly separations for the HMI, Controller, Model and View layers. Communication

between models was allowed as well as communication between controllers, but communication between another components controller and another's model was disallowed.

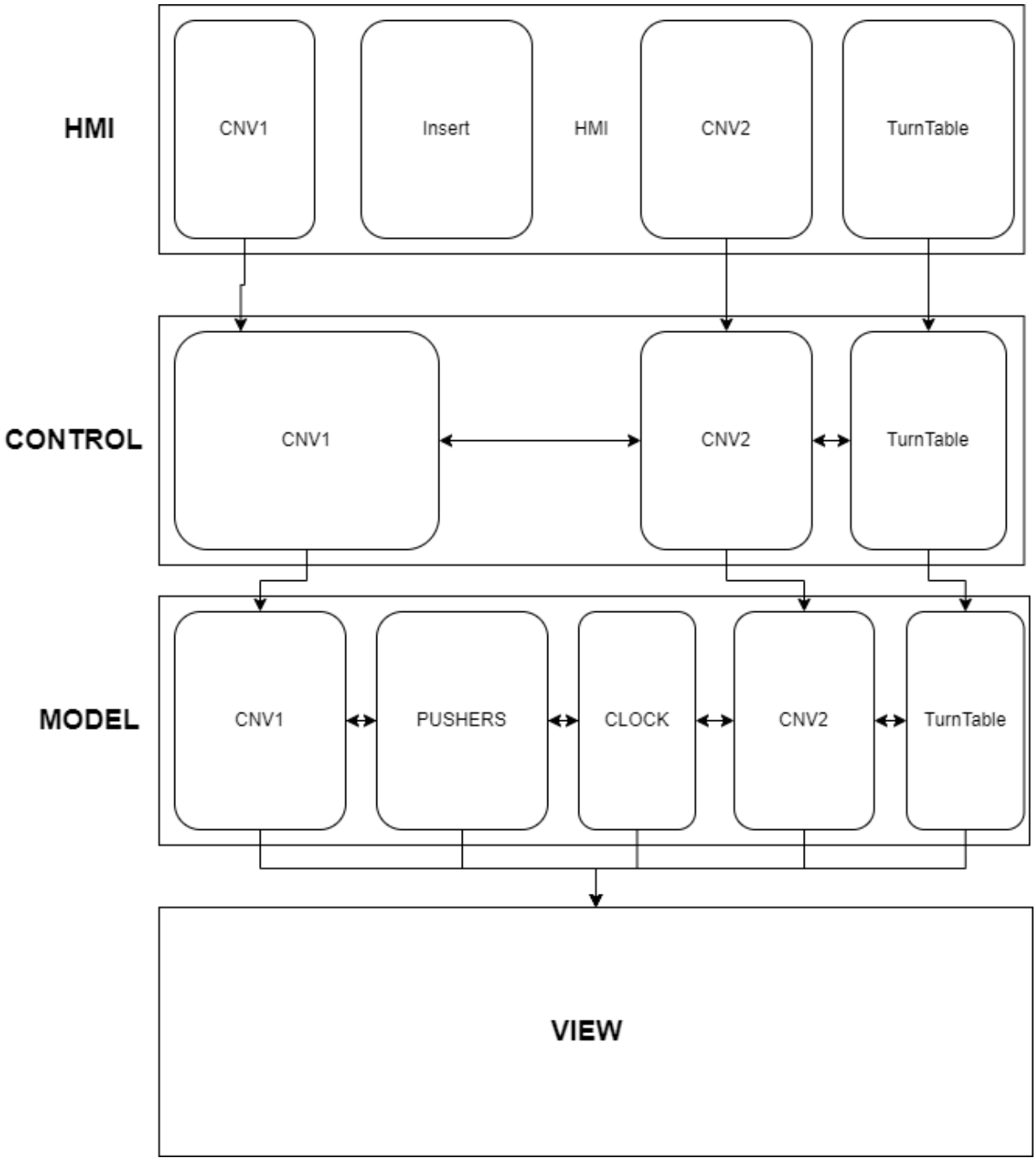

Figure 14: Architecture and components of the sorting system

Sequence diagram, class diagram and use case diagrams are presented in figures 15-17. The sequence diagram demonstrates how the different parts of the system reacts when a red work piece is inserted on to the conveyor 1.

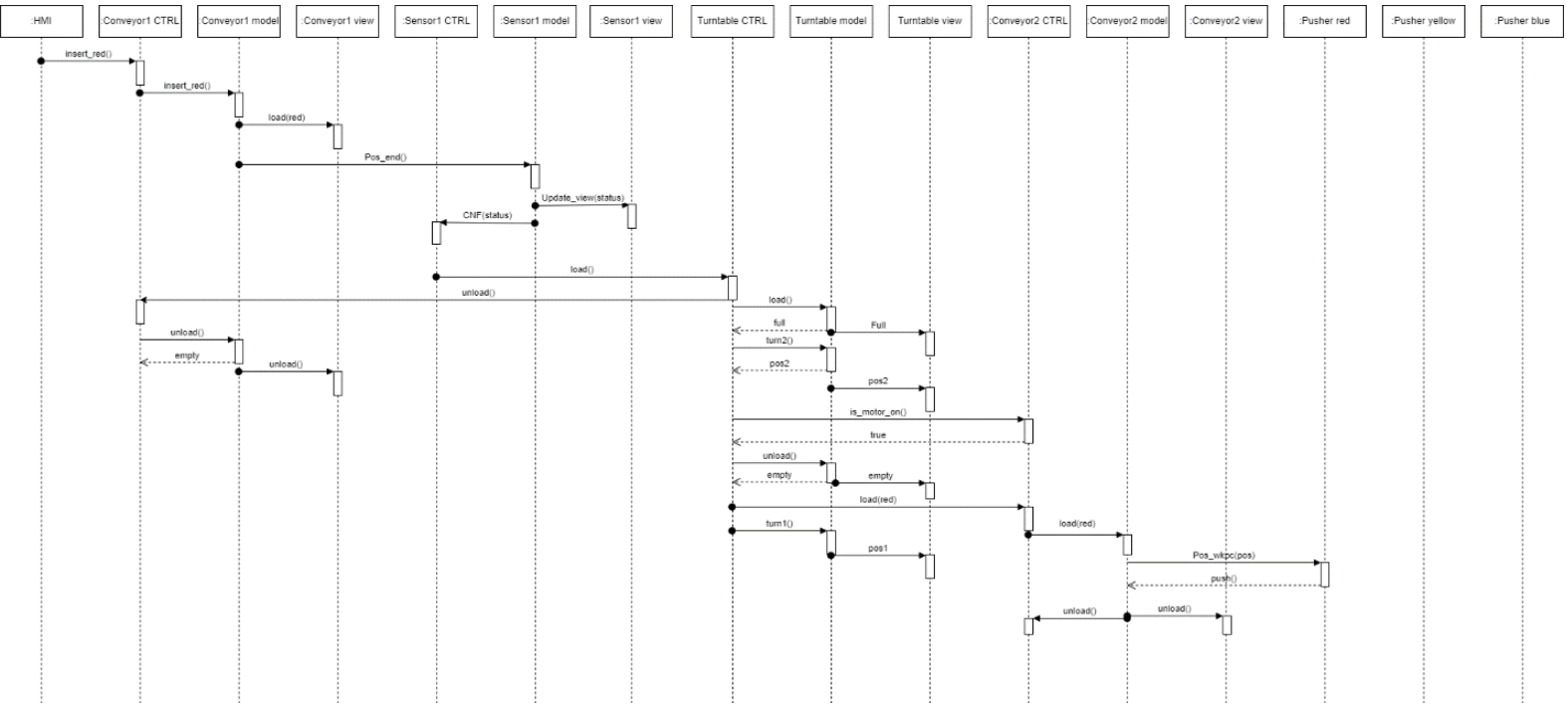

Figure 15: Sequence diagram

The class diagram tries to display the different properties of the systems recourses. Sometimes in FBDK the boundary between an event and a variable can feel tenuous, and naming the events and variables as precisely as possible does not feel as important on the sixth redesign of a block. This makes trying to accurately display the properties via a class diagram more difficult than one would assume.

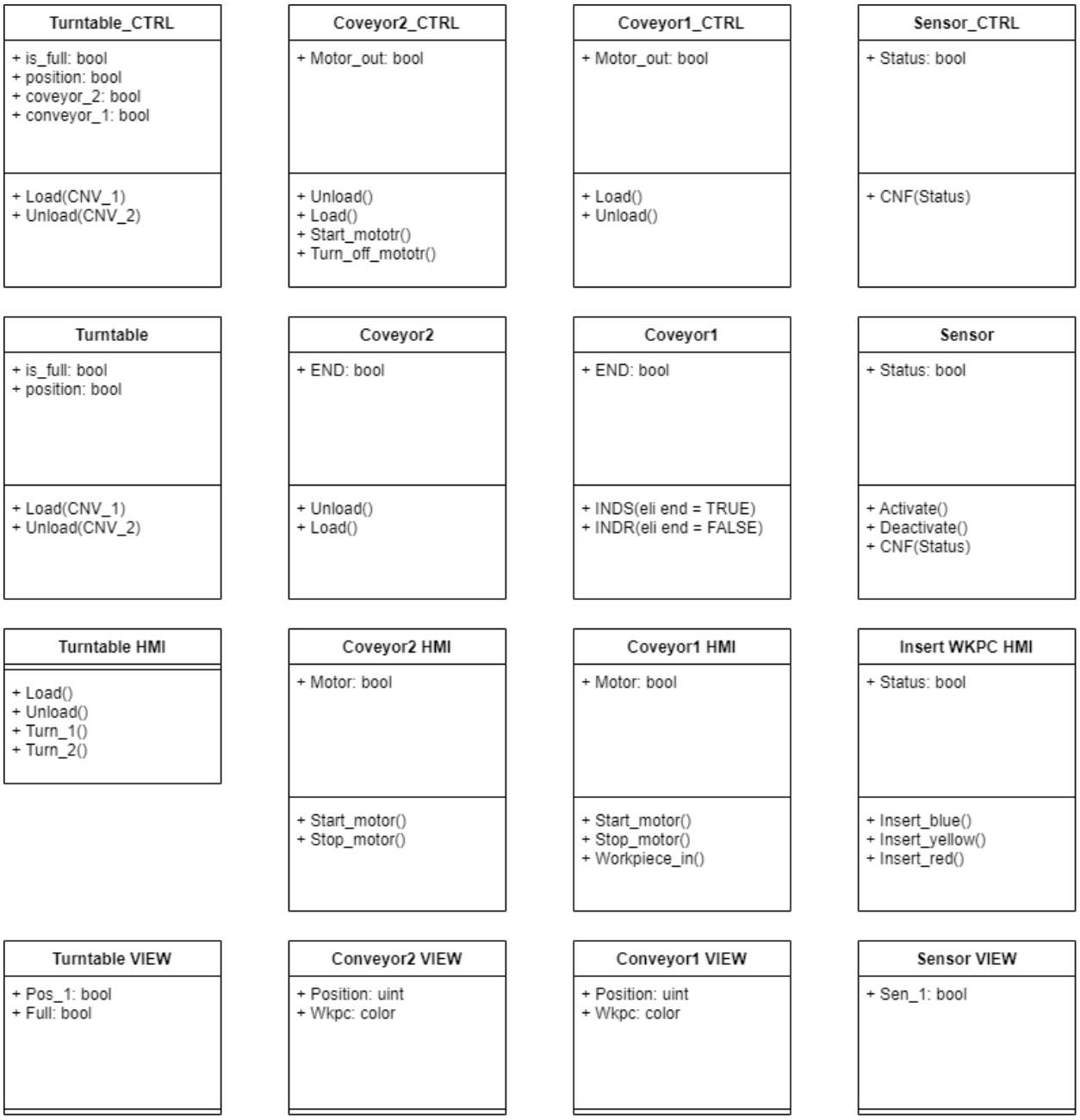

Figure 16: Conveyor system class diagram.

Use case diagram for the conveyor system is shown in figure 17. If automatic and manual modes were implemented the user could control the whole system in manual mode through the human-machine interface. And the controller could operate the turn table and conveyors in automatic mode.

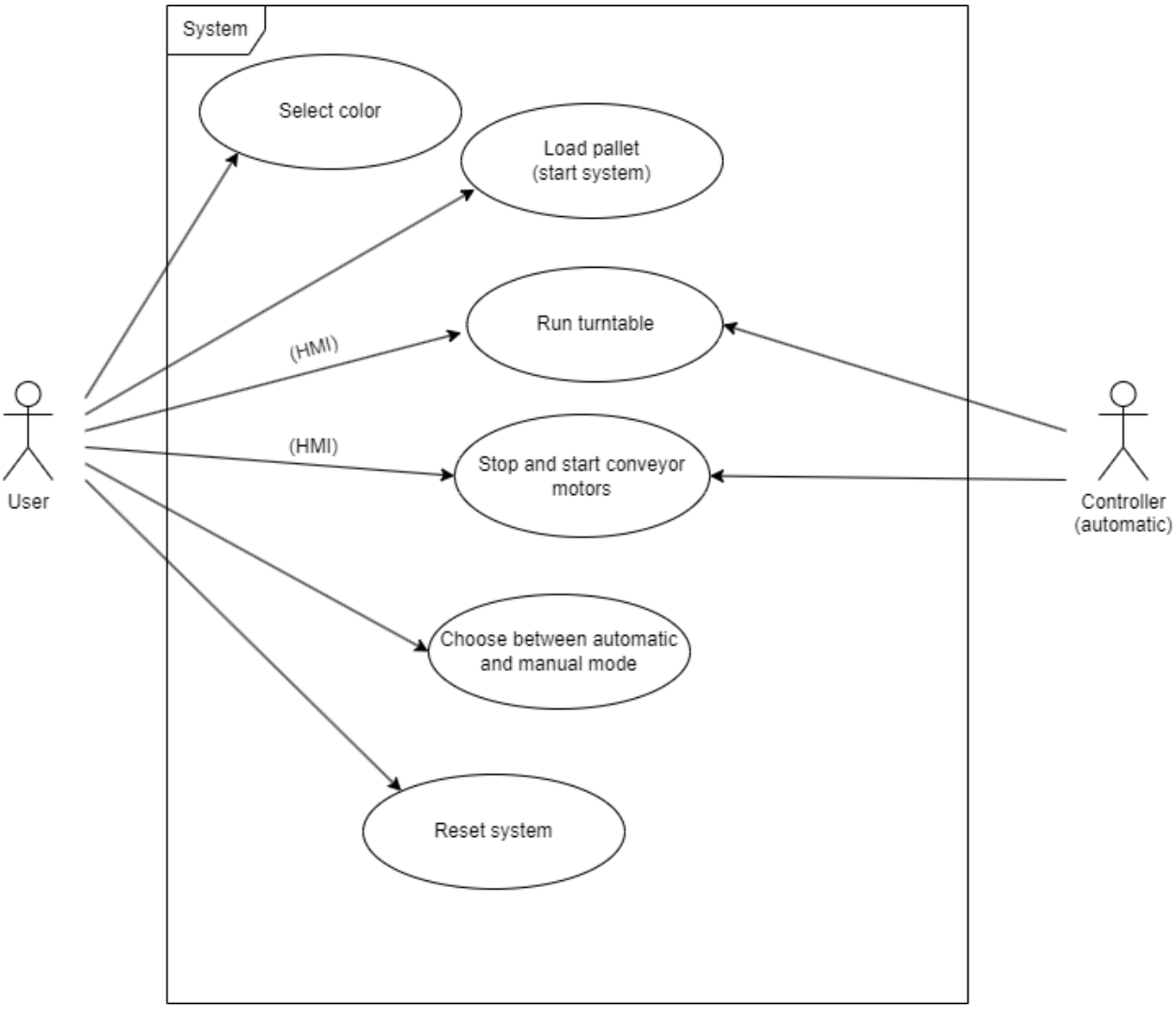

Figure 17: Use case diagram for conveyor system

Final thoughts

Function block development kit (FBDK11) is a clumsy old software (understatement) with no easily accessible documentation or discussion boards for help. It is extremely hard to debug or even understand what is happening at a given time. It also feels like many times the same system that just worked doesn't work anymore when the software is restarted on another day which cost us many hours during the development time just to catch up to where we were last time, we were developing the systems. All in all, we are not fans of the development environment even if function blocks themselves could be powerful for many applications.