Software Engineering Lab 4 Report

1 Document Information

1.1 List of Authors

Group/Team	Names
<Your Group>	<Hilary Ogalagu>
<Your team name>	<Akter Rokaia>

1.2 Versions

Version	Date	Author	Notes
0.1	07.05.2021	<All>	First draft, we had a team meeting and discussed on which UC each team partner wants to work on. And after that we also tried to figure out which of the design patterns fits best in.
0.2	08.05.2021	<Hilary Ogalagu>	More detailed look into the design patterns to be sure it fits in and then finally making my decision on which UC and design pattern to go for.
0.3	08.05.2021	<Hilary Ogalagu>	Started the sketch of the Class Diagram in respect to the Design pattern I choosed, then the Table of responsibility and finally the sequence diagram of the class diagram(walk through).
0.4	08.05.2021	<Rokaia Akter>	Chose the design patterns to work with UC and design pattern to go for.
0.5	09.05.2021	<Rokaia Akter>	Sketch of Class Diagram and Table of class responsibilities, sketch of walk-through
0.6	09.05.2021	<Hilary Ogalagu>	Final editing and modification of lab report
0.7	15.05.2021	<Hilary Ogalagu>	Further detailed modification of my Observer Pattern in response to the correction gotten from the lab section.
0.8	16.05.2021	<Hilary Ogalagu>	Refined sequence diagram of the work flow of my design pattern demonstrating the observer pattern and the description of responsibility of the classes of the design pattern.
0.9	09.05.2021	<Rokaia Akter>	Final editing and modification of lab report
0.10	15.05.2021	<Rokaia Akter>	Final modification Iterator pattern : UC1: InitialPairing and use the description of responsibility and Refined sequence diagram.
0.11	16.05.2021	<Rokaia Akter>	Final modification Decorator pattern : UC1: InitialPairing and use the description of responsibility and Refined sequence diagram.

Content

2 Glossary

The following domain-specific terms are used throughout the document.

Term	Abbreviation	Description	Reference
Use Case	UC	Describes the use case of our system we working on which includes the functionalities of the system	

3 Prerequisites

<Prep task #1: Table of your four selected design patterns together with your UCs, including UC no., name and if it is app or robot and lab 3 or lab 2, and which team partner takes it.>

UC	**DESIGN PATTERNS**	**APP/SYSTEM**
UC5 DisplayStatus	**Composite**: Supports composition of object trees so that clients can treat individual objects and compositions of objects uniformly. In regards of my UC displStatus, the viewer treats individual objects uniformly as well as the composition of the objects.	App
	Observer: Here one object stores a list of observers that are updated when the state of the object is changed. So we define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and get an update call automatically and for our UC, the observer patterns works better and that's the reason we choose to work	APP

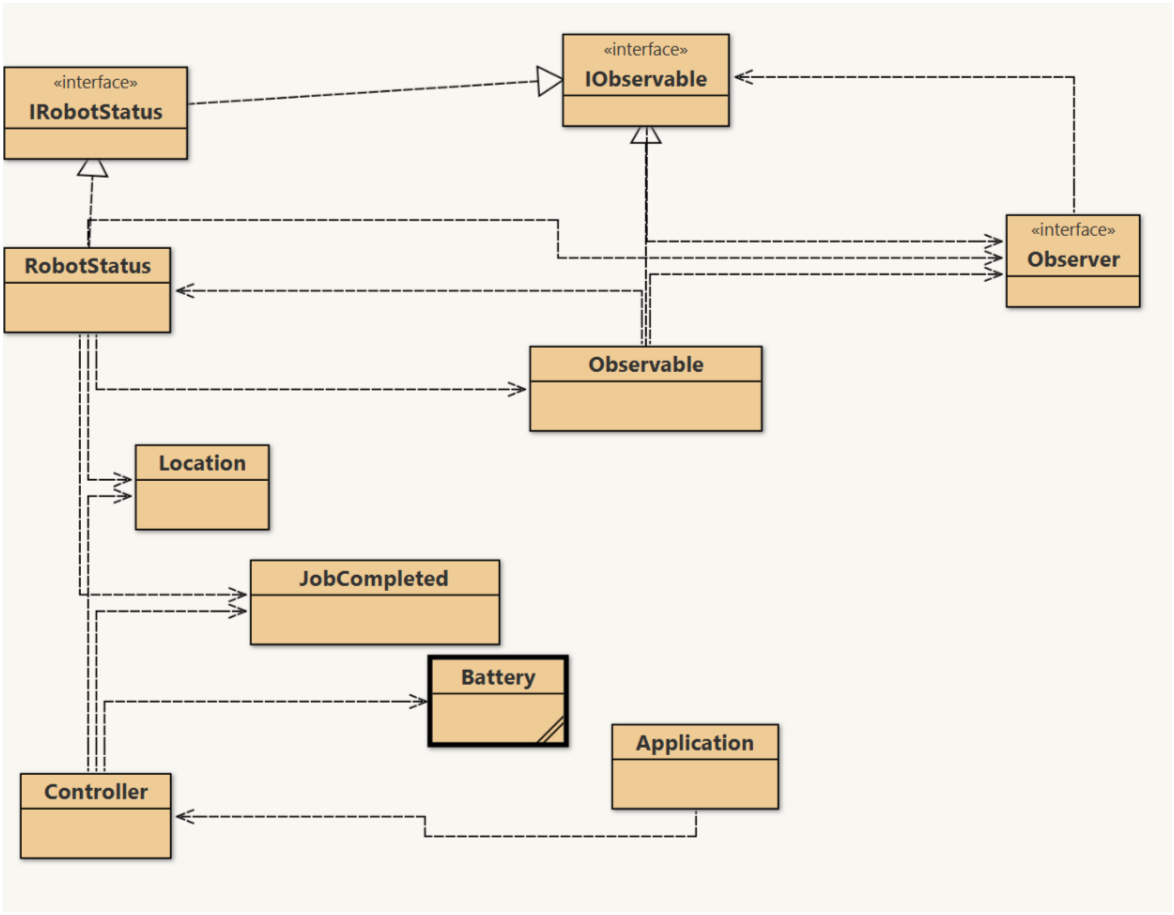
	specifically with this pattern. Because we every other objects to be updated as soon as the updater gets update.	
Author	<Hilary Ogalagu>	PARTNER A

User case number	**Design patterns**		**Use case**
UC1: InitialPairing	Iterator	Decorator	Apps
	Iterator pattern is used to get a way to access the elements of a collection object in sequential manner without any need to know its underlying representation. Send message to InitializePairing to initiate one time pairing when the user selects pairing and authorize the system to going next step.	Decorator – A means to attach additional responsibilities to an existing object dynamically. Decorators provide a flexible alternative to sub-classing to extend flexibility, when extending classes is impractical. The decorator conforms to the interface of the .	
AUTHOR	<Rokaia Akter>	<Rokaia Akter>	PARTNER B

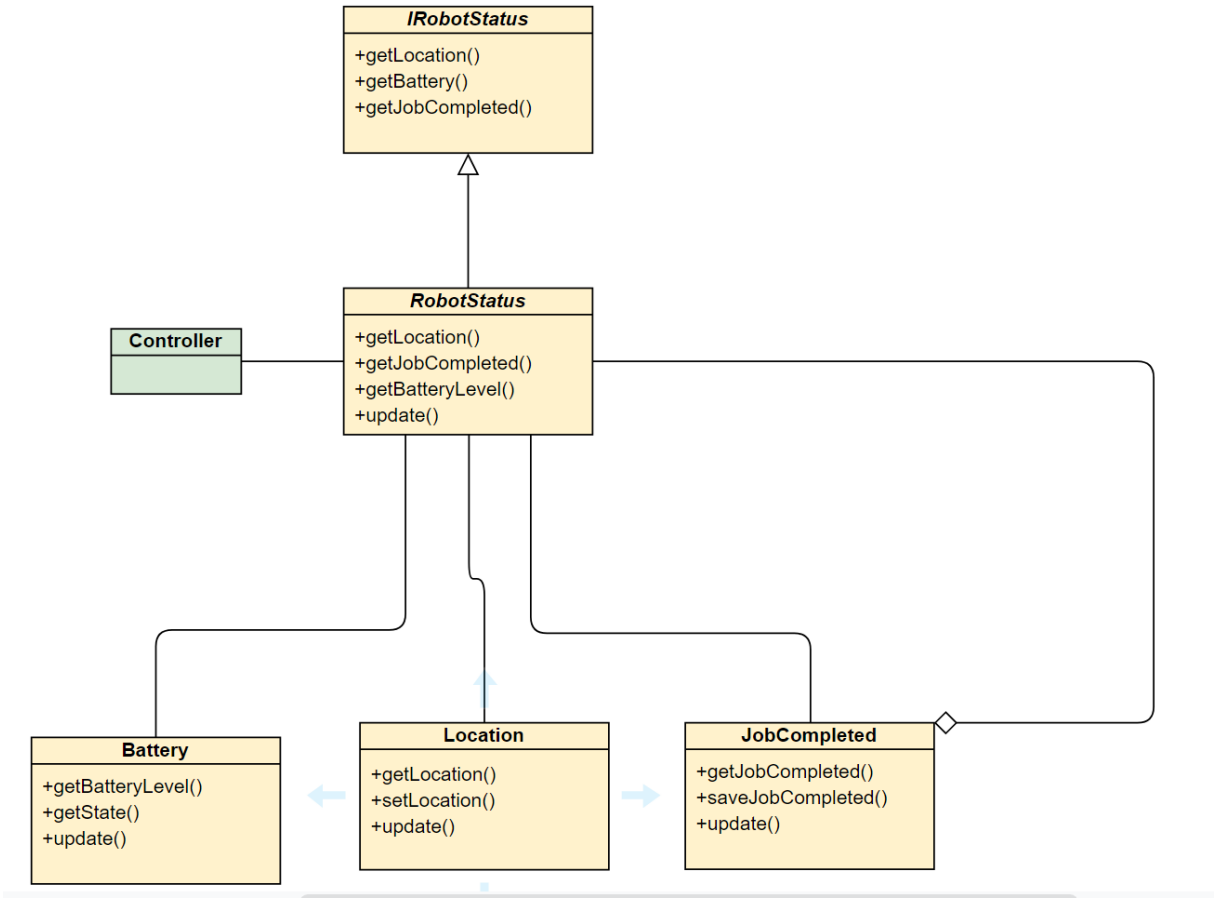
4 Class Design (Partner A)

< Prep task #2 and #3, Sketch of Class Diagram, Table of class responsibilities, sketch of walk-through>

Observer Pattern: UC5: DisplStatus



Composite Pattern: UC5 DisplStatus

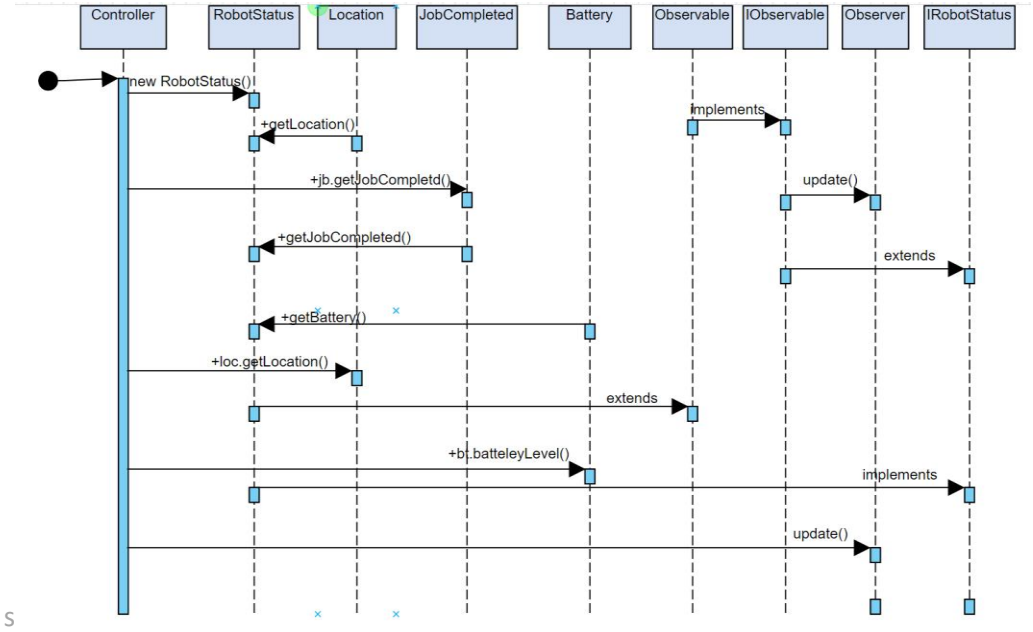


	Class Responsibility: Observer Pattern
Controller	Gets the robot state and sends message to the client with the status of the robot information gotten.
RobotStatus	RobotStatus is the state of the Robot, gets the state of the other classes by a method of get…(). And this class implement IRobotstatus.
IRobotStatus	Calls the method of the subclass it is implementing.
Observeable	Receives message from the RobotStatus which is the observable.
Battery	Gets updated by the robot so in others words the battery updates itself. It's also the state of the Robot.

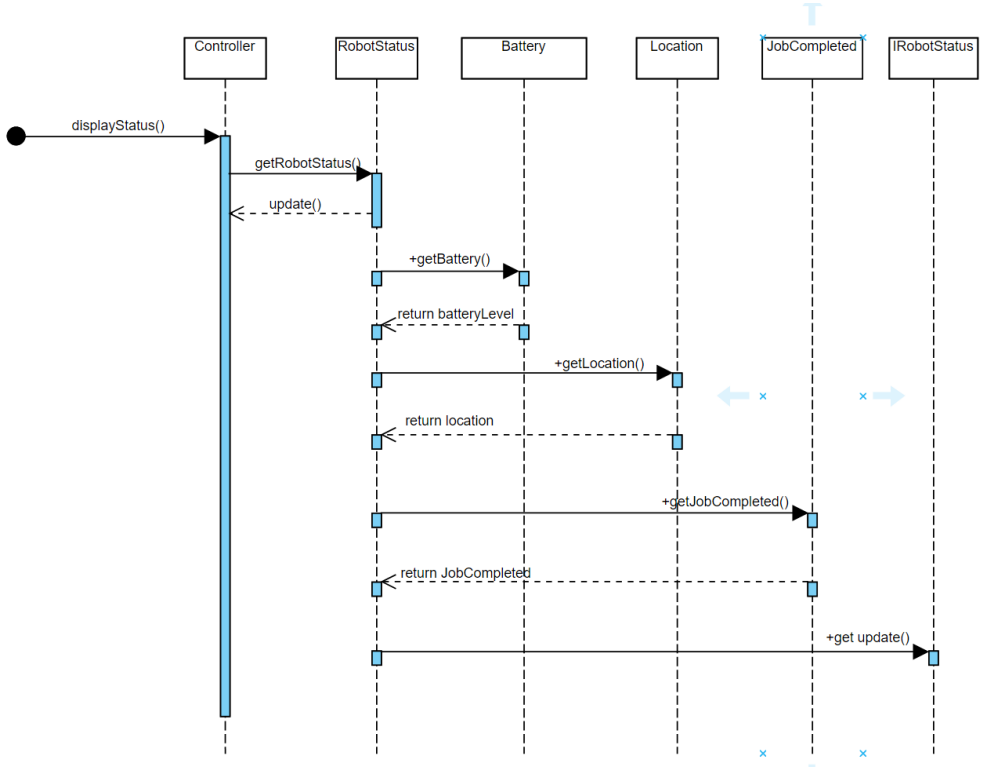
Observer	Observes get message from the Observable through update call.
Location	Gets the currentLocation of the robot. It also keeps the state of the robot
JobCompleted	Gets the resent job completed and as well as saves it for further references.

	Class Responsibility: Composite Pattern
Controller	Gets instructions from the user-interface through the displStatus command and sends message to RobotStatus to return the update of the command call given.The controller also sends and receives message RobotStatus.
Battery	This is the state of the Robot as it updates itself and sends the update to the RobotStatus where the client by method call receives the update.
Location	This is the current location of the robot system. It updates itself as well. And sends message to the RobotStatus of any recent update and with this the client can have a view of the currentLocation of the robot at any given time.
JobCompleted	Updates itself and as well keeping updates of all the task completed and return nothing when no information has been updated. Sends message to the RobotStatus of any recent update.
RobotStatus	Gets request from the client through a method call to retrieve update and send message to the battery class, location and JobCompleted accordingly. And implements IRobotStatus interface.
IRobotStatus	Recieves message from the Robotstatus. Get the method call of the robotstatus.

Sequence Diagram: Observer Pattern



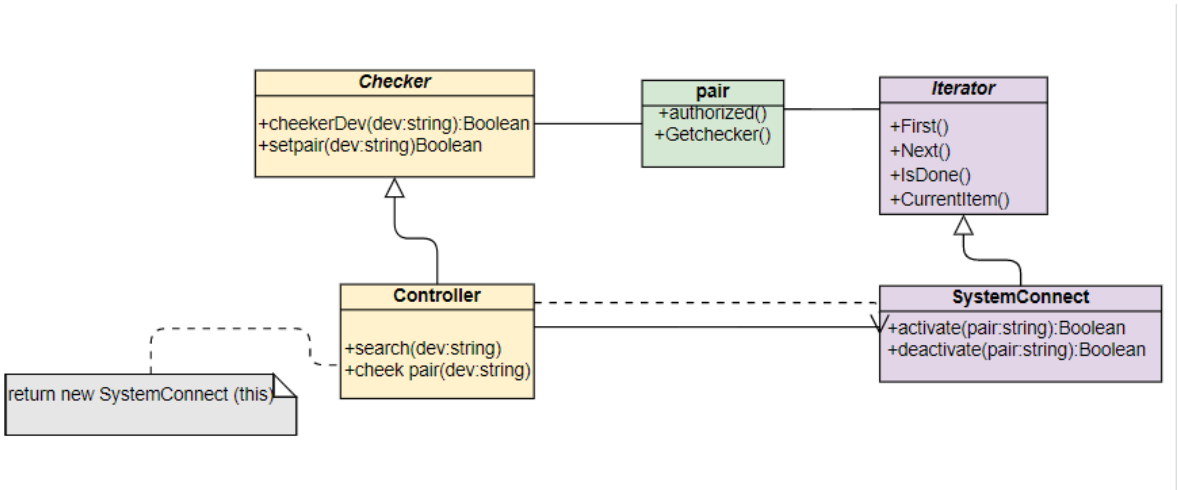
Sequence Diagram: Composite Pattern



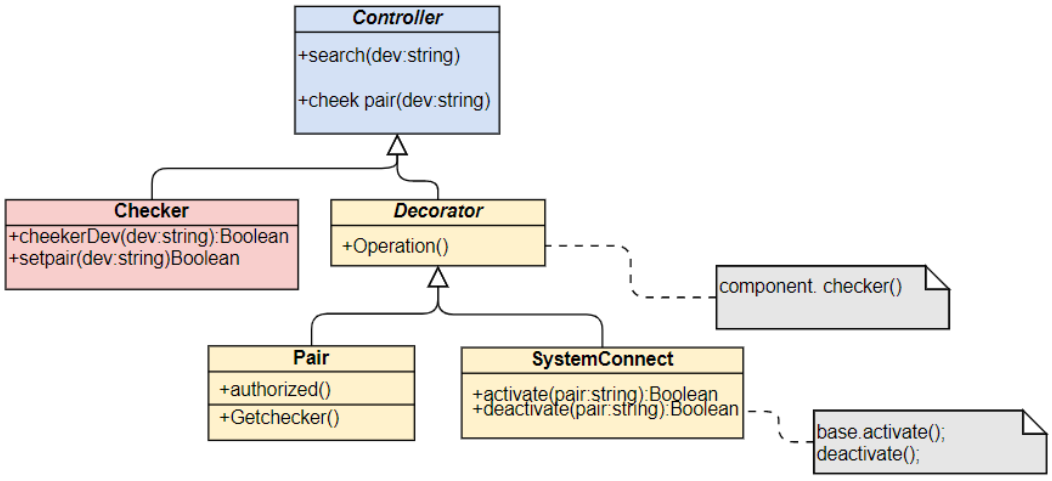
5 Class Design (Partner B)

< Prep task #2 and #3, Sketch of Class Diagram and Table of class responsibilities, sketch of walk-through>

Iterator Pattern: UC1: InitialPairing



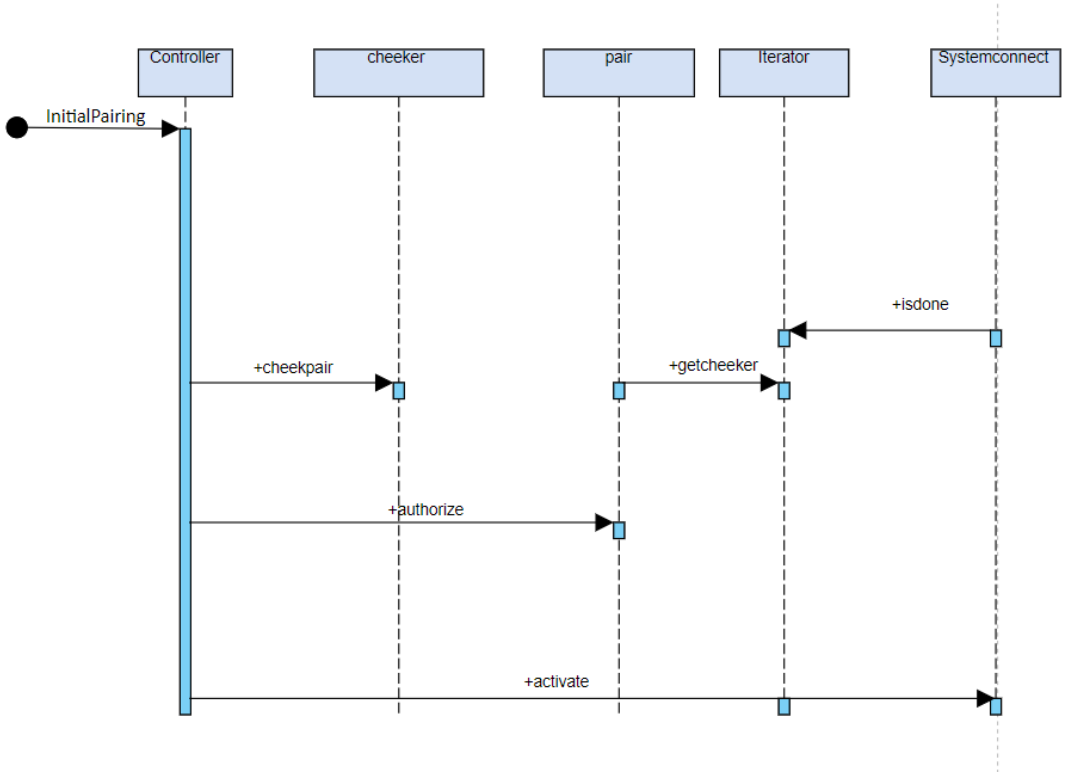
Decorator Pattern: UC1: InitialPairing



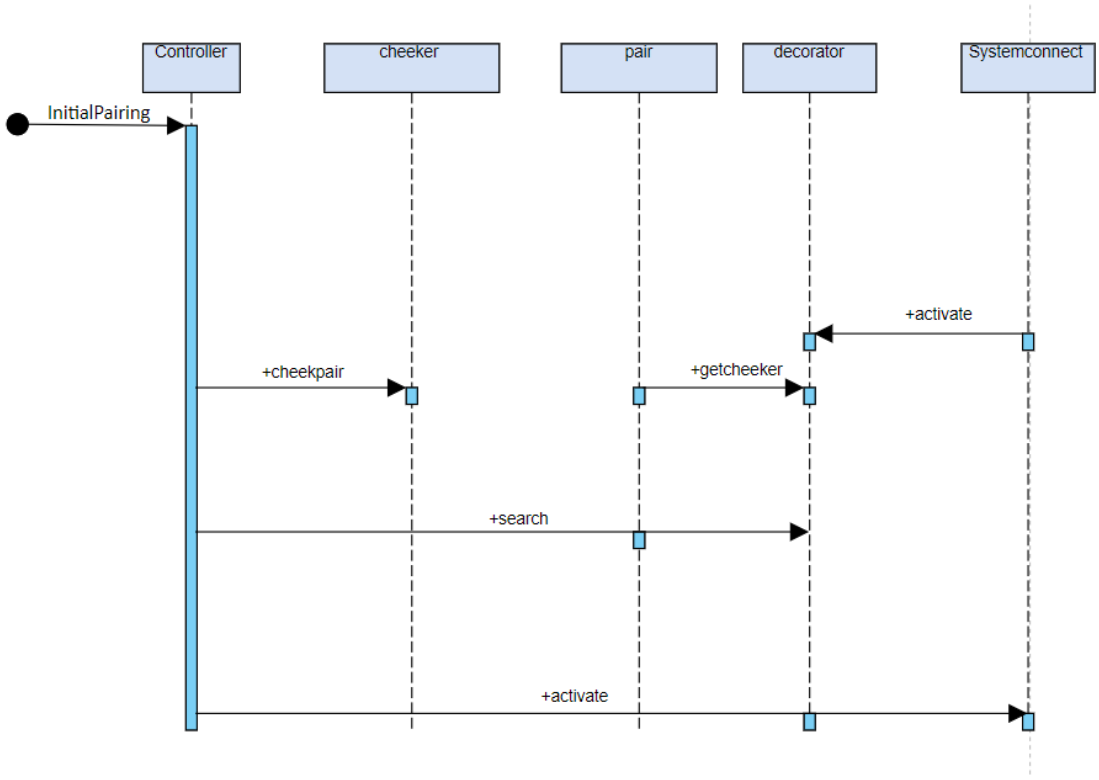
Class Responsibility: Iterator Pattern	
Controller	Gets the robot state and sends message to to Checker to check if robot is within the range of connection.
Checker	Gets updates from the controller and also sends message to the pair of the updates received .
pair	Recieves message from the controller for InitializePairing to initiate one time pairing when the user selects pairing.
System connect	Recieves message from the controller and Send message to Iterator to activate pairing.
Iterator	Pair and systemconnect send to massage to iterator for InitializePairing and active condition implement.

Class Responsibility: Decorator Pattern	
Controller	Controller Send message to WifiCtrl to connect robot with wifi. Gets instructions from the user-interface through the decorator and sends message to activateChecker to activate.
Checker	Gets updates from the controller and also sends message to the pair of the updates received .
pair	Pair insure authorize and Getchecker Recieves message from the controller for InitializePairing to initiate one time pairing when the user selects pairing.
System connect	Recieves message from the decorator and Send message to decorator to activate or deactivate connection .
Decorator	Decorator recived the information controller and make the communication with pair and System Connect .

Sequence Diagram: Iterator Pattern



Sequence Diagram : Decorator Pattern



6 Demonstrator (Partner A)

< Description, Lab task #1>

7 Demonstrator (Partner B)

< Description, Lab task #1>

8 Design comparison

<Generated class diagram, IntelliJ or BlueJ, Lab task #2 and #3>

9 Shortcomings and drawbacks

< Lab task #4>

References

<all used ressources>

[1] SE_09_Renz_DesignPatterns_2.pdf

[2] SE_07-08_Renz_DesignPatterns_1.pdf