ADD | Group 14

Content

1. Use case

Comments:

- The robot is the raspberry pi.
- Board types: EV3 & Grove-pi

Build

Use case name	Build
Actors	BPjs program
Preconditions	The boards which mention in the param are connected to the robot. The sensors which mention in the param are connected to the appropriate board.
Post conditions	The system connected to the boards and sensors.
Parameters	Json with the data of the boards to build and be supported. Include the board type and data for each board. For EV3 board, the port name of the Bluetooth connection. For Grove-pi board, for each port the sensor name which connected to. If there many boards from the same type the index of the board need to be indicated.
Flow	The system gets the command build with the data that include the boards and sensors which connected to their ports. The system builds their instances in the system and connect them.
Positive acceptance scenario	The build command absorbed into the system. The system successfully connected to the boards and the appropriate sensors
Negative acceptance scenario	The build command didn't absorb into the system because the json structure was different from the excepted structure.

Subscribe

Use case name	Subscribe
Actors	BPjs program
Preconditions	Build command execute before, with the boards which mention in the param and sensors that connected to the ports which mention in the param.
Post conditions	The system starts to track after the updated values that back from the sensors in the ports which mention in the param.
Parameters	Json with the ports of the boards to track their values. Include the board type and ports name for each board. If there many boards from the same type the index of the board needs to be indicated.
Flow	The system gets the command subscribe with the data that include the ports which connected sensor to them. The system starts to track after the updated values that back from the sensors in those ports
Positive acceptance scenario	The subscribe command absorbed into the system. The system successfully tracks after the updated values that back from the sensors.
Negative acceptance scenario	The subscribe command didn't absorb into the system because the json structure was different from the excepted structure.

Unsubscribe

Use case name	Unsubscribe
Actors	BPjs program
Preconditions	Build command execute before, with the boards which mention in the param and sensors that connected to the ports which mention in the param. • Basically, this case intended to execute after subscribe command, but is not precondition because when the unsubscribe call before subscribe nothing will happened and the system continue as usual.
Post conditions	The system stops to track after the updated values that back from the sensors in the ports which mention in the param.
Parameters	Json with the ports of the boards to stop track their values. Include the board type and ports name for each board. If there many boards from the same type the index of the board needs to be indicated.
Flow	The system gets the command unsubscribe with the data that include the ports which connected sensor to them. The system stops to track after the values that back from the sensors in those ports
Positive acceptance scenario	The unsubscribe command absorbed into the system. The system successfully stops tracking after the values that back from the sensors.
Negative acceptance scenario	The unsubscribe command didn’t absorb into the system because (1) the json structure was different from the excepted structure. (2) subscribe command don’t call before so the system don’t even start to track after those ports.

Set sensor mode | Set actuator data

Use case name	Set sensor mode | Set actuator data
Actors	BPjs program
Preconditions	Subscribe command execute before, with the boards and ports which mention in the param.
Post conditions	The system set the modes or data of the sensors or actuators respectively, in the appropriate ports where they connected according to the input data.
Parameters	Json with the ports of the boards to set their values with the value for each port. Include the board type, ports name for each board and value for each port. If there many boards from the same type the index of the board needs to be indicated.
Flow	The system gets the command setSensorMode or setActuatorData with the data that include the ports which connected sensor or actuators to them and the values. The system set the modes or data of the sensors or actuators respectively, in the appropriate ports
Positive acceptance scenario	The setSensorMode or setActuatorData command absorbed into the system. The system successfully set modes or data of the sensors or actuators respectively, in the appropriate ports
Negative acceptance scenario	The setSensorMode or setActuatorData command did not absorb into the system because (1) the json structure was different from the excepted structure. (2) subscribe command don't call before so the system don't even start to track after the values of those ports.

Get sensors data

Use case name	Get sensors data
Actors	BPjs program
Preconditions	Subscribe command execute before, with the boards and ports which mention in the param.
Post conditions	The system gets the values of the sensors in the appropriate ports according to the input data.
Parameters	Board type, board index and port

Flow	The system gets the value that back from waitFor with var that contain eventSet of all the events their names are GetSensorData. For example:
	`var dataEventSet = bp.EventSet("", function (e) {`
	`    return e.name.equals("GetSensorsData");`
	`});`
	`var e = bp.sync({waitFor: dataEventSet});`
	from the data that return which contain the data of all the sensors the system exacts the value according to the board and port which mention in the param.
	As a follow-up to the example:
	`var data = JSON.parse(e.data);`
	`var distance = data.GrovePi._1.D4;`

Positive acceptance scenario	The GetSensorData command absorbed into the system. The system successfully exacts the value according to the board and port which mention in the param.
Negative acceptance scenario	The system can’t exact the value because (1) the syntax was given was incorrect. (2) subscribe command don’t call before so the system don’t even start to track after the values of those ports.

Drive

Use case name	Drive
Actors	BPjs program
Preconditions	Build command execute before, with the boards which mention in the param.
Post conditions	The system spins the motors of the robot in the ports with the speed which mention in the param.
Parameters	Json with the ports of the boards where the motors connected and speeds. Include the board type, ports name for each board and speed for each port. If there many boards from the same type the index of the board needs to be indicated.
Flow	The system gets the command drive with the data that include the ports which connected motors to them and speeds. The system spins the motors of the robot
Positive acceptance scenario	The drive command absorbed into the system. The system successfully spins the motors of the robot.
Negative acceptance scenario	The drive command didn't absorb into the system because (1) the json structure was different from the excepted structure. (2) subscribe command don't call before so the system don't even recognize the board.

Rotate

Use case name	Rotate
Actors	BPjs program
Preconditions	Build command execute before, with the boards which mention in the param.
Post conditions	The system spins the motors of the robot in the ports according to the speed and angle which mention in the param.
Parameters	Json with the ports of the boards where the motors connected and speeds and angels. Include the board type, ports name for each board and speed and angle for each port. If there many boards from the same type the index of the board needs to be indicated.
Flow	The system gets the command rotate with the data that include the ports which connected motors to them and speeds and angels. The system spins the motors of the robot
Positive acceptance scenario	The rotate command absorbed into the system. The system successfully spins the motors of the robot.
Negative acceptance scenario	The rotate command didn't absorb into the system because (1) the json structure was different from the excepted structure. (2) subscribe command don't call before so the system don't even recognize the board.

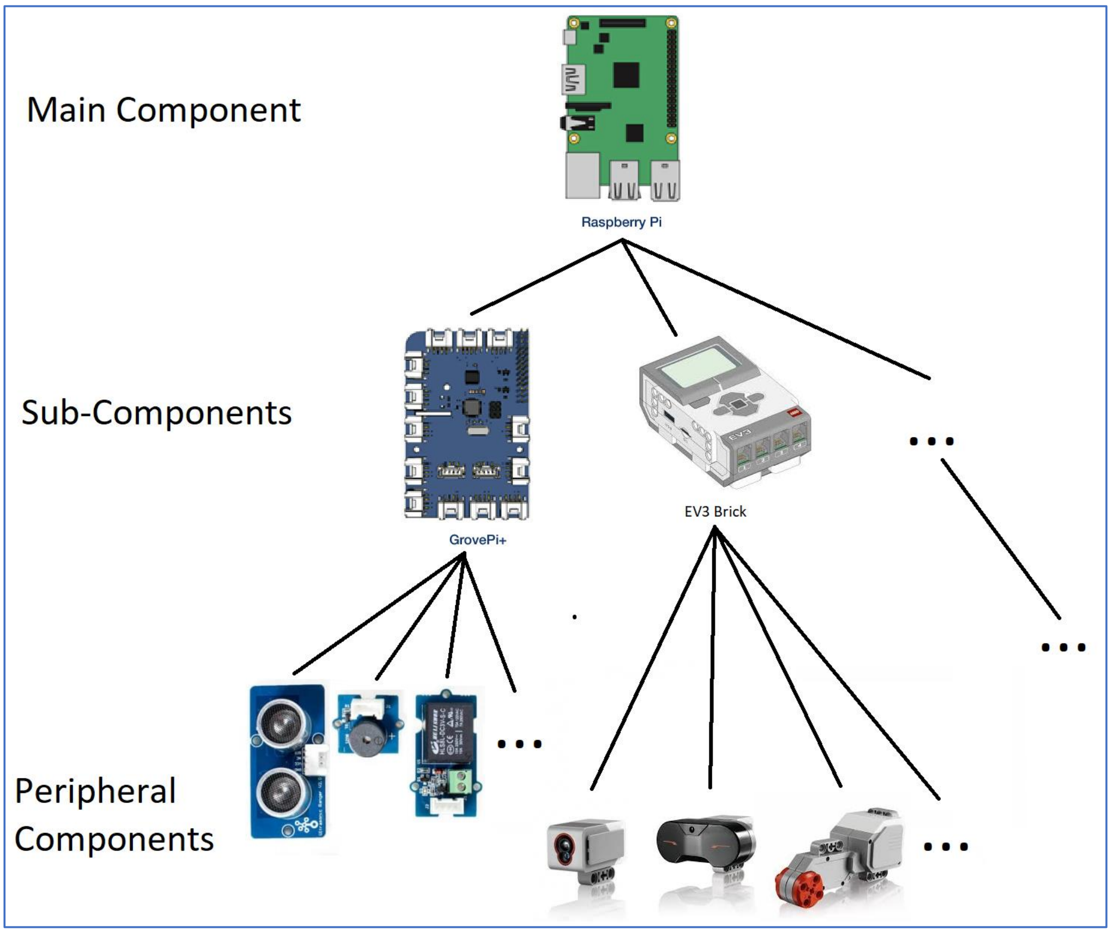
MyAlgorithm

Use case name	MyAlgorithm
Actors	BPjs program
Preconditions	Build command execute before, with the boards which mention in the params.
Post conditions	The system executes the algorithm the programmer defines with the params he gives.
Parameters	Json with the boards to run the algorithm belongs them and their indexes, and for each board specifies the parameters the programmer needs for the algorithm with free hand of the programmer.
Flow	The system gets the command myAlgorithm with the data that include the boards which their algorithm needs to execute with the appropriate params. The system executes the algorithms the programmer defines. The results return to the programmer if he defined it as part of the js file.
Positive acceptance scenario	The myAlgorithm command absorbed into the system. The system successfully executes the algorithm the programmer defines.
Negative acceptance scenario	The myAlgorithm command didn’t absorb into the system because (1) build command doesn’t call before so the system don’t recognize those ports. (2) the params the programmer gave and the params the algorithm need don’t match.

2. System architecture

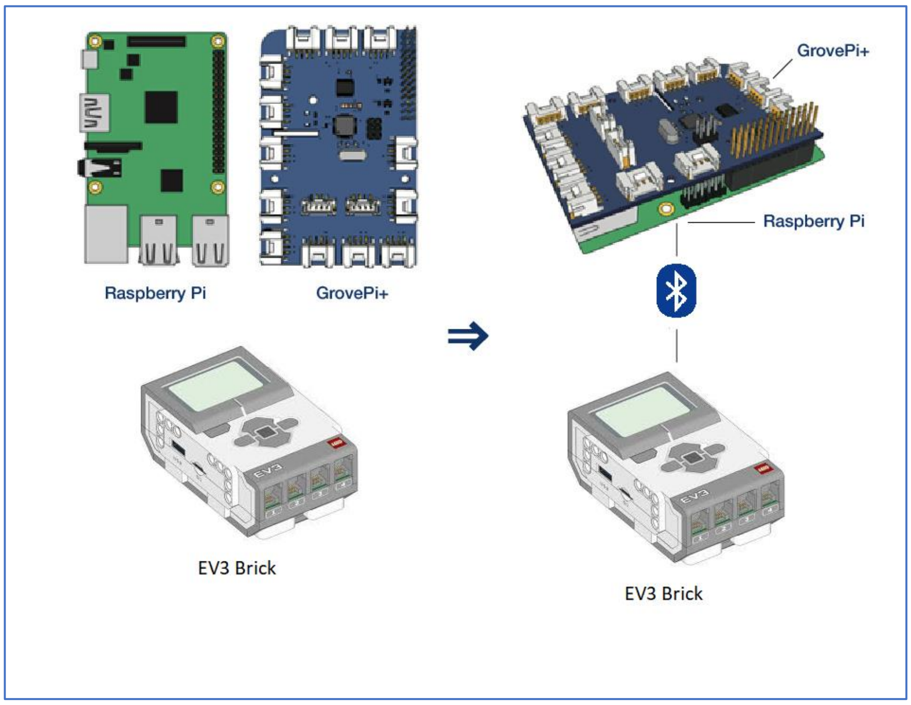
We will describe a layout of the physical components we will support

General Hardware Diagram



We will describe the connection made between the components in the system

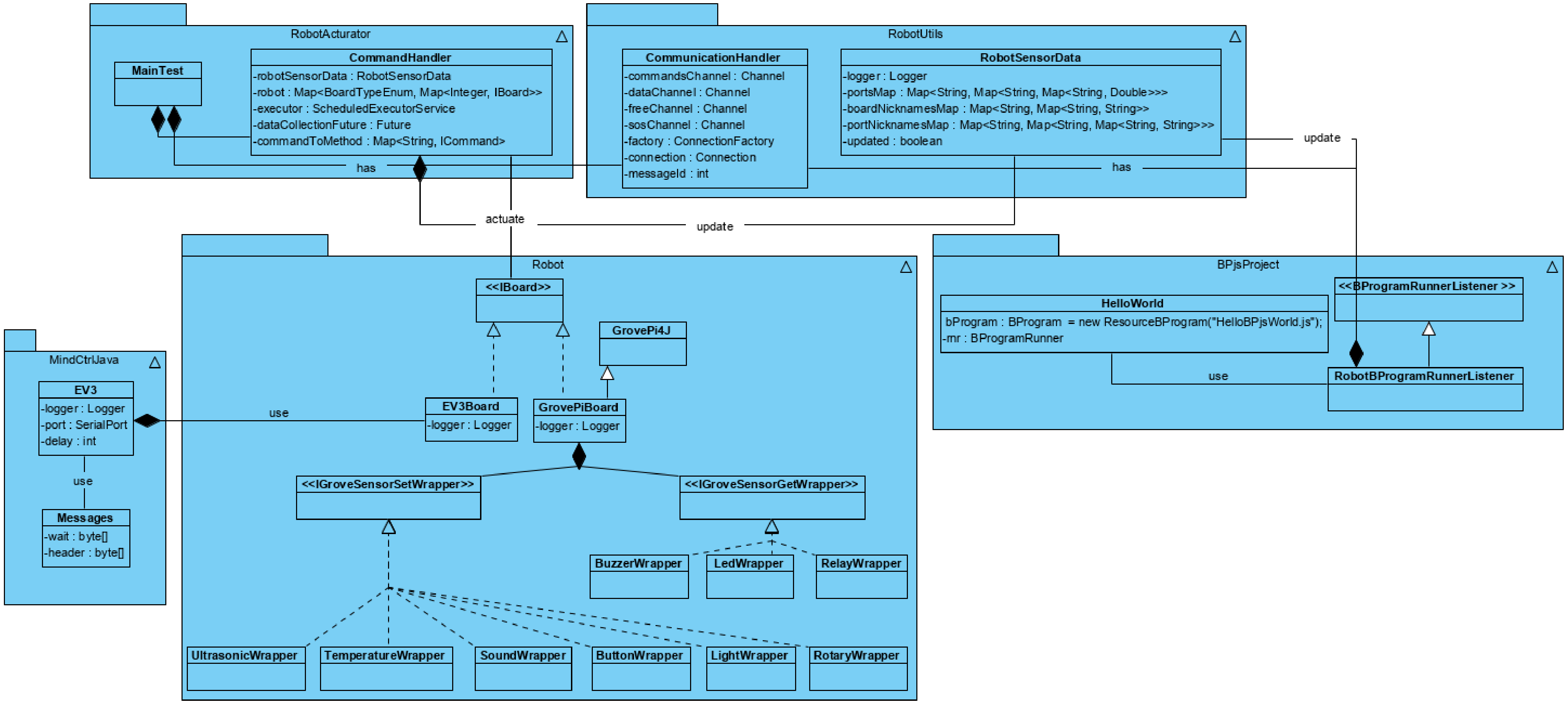
Hardware set up



System components

Component	Description
EV3	Responsible for transferring the commands directly to the robot (using byte messages)
Ev3Board	Responsible for performing operations related to EV3
GrovePiBoard	Responsible for performing operations related to Grove pi
RobotSensorData	Responsible for handling the tracking after the values that back from the robot's sensors.
CommunicationHandler	Responsible for communication between the commands that send to the robot from the user and the data that back from the robot.
CommandHandler	Responsible for handling the commands that gets from the program by the CommunicationHandler

3. Data Model


RobotActurator
MainTest
CommandHandler
-robotSensorData : RobotSensorData
-robot : Map<BoardTypeEnum, Map<Integer, IBoard>>
-executor : ScheduledExecutorService
-dataCollectionFuture : Future
-commandToMethod : Map<String, ICommand>
has
actuate
update
RobotUtils
CommunicationHandler
-commandsChannel : Channel
-dataChannel : Channel
-freeChannel : Channel
-sosChannel : Channel
-factory : ConnectionFactory
-connection : Connection
-messageId : int
RobotSensorData
-logger : Logger
-portsMap : Map<String, Map<String, Map<String, Double>>>
-boardNicknamesMap : Map<String, Map<String, String>>
-portNicknamesMap : Map<String, Map<String, Map<String, String>>>
-updated : boolean
update
has
Robot
<<IBoard>>
GrovePi4J
EV3Board
-logger : Logger
GrovePiBoard
-logger : Logger
<<IGroveSensorSetWrapper>>
<<IGroveSensorGetWrapper>>
BuzzerWrapper
LedWrapper
RelayWrapper
UltrasonicWrapper
TemperatureWrapper
SoundWrapper
ButtonWrapper
LightWrapper
RotaryWrapper
use
MindCtrlJava
EV3
-logger : Logger
-port : SerialPort
-delay : int
use
Messages
-wait : byte[]
-header : byte[]
BPjsProject
<<BProgramRunnerListener >>
HelloWorld
bProgram : BProgram = new ResourceBProgram("HelloBPjsWorld.js");
-mr : BProgramRunner
use
RobotBProgramRunnerListener

3.1. Description of Data Objects

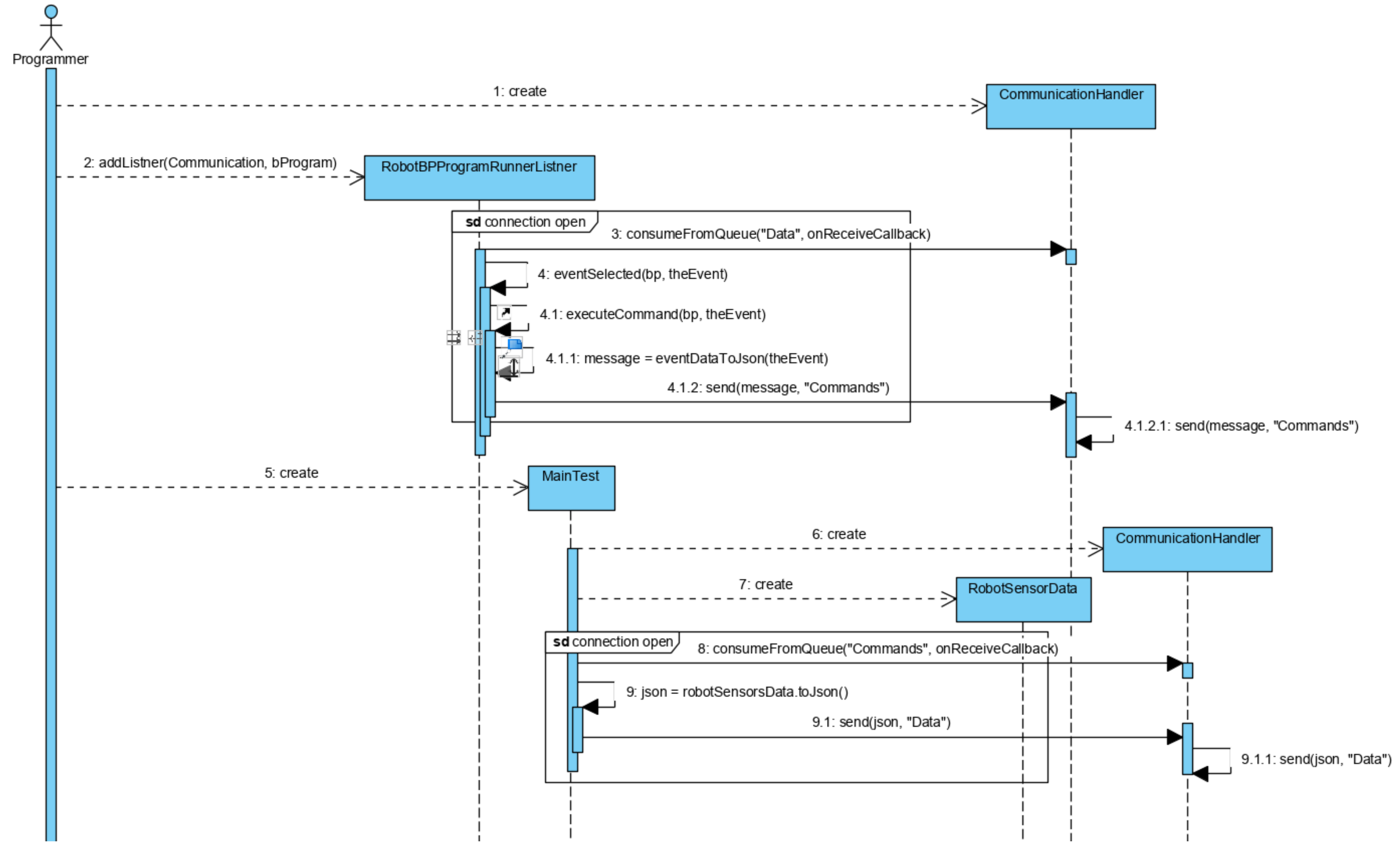
Data object	Description
Ev3Board	Represent the Ev3 Mindstorms and responsible for performing operations related to EV3
GrovePiBoard	Represent the Grove Pi and responsible for performing operations related to Grove pi
RobotSensorData	Responsible for handling the tracking after the values that back from the robot's sensors.
CommunicationHandler	Responsible for communication between the commands that send to the robot from the user and the data that back from the robot. Open different queues for different commands and send data over the appropriate queue.
CommandHandler	Responsible for handling the commands that gets from the program by the CommunicationHandler
RobotBPProgramRunnerListener	Get the updated values and send the command from the js file over the queues that define by the CommunicationHandler.
MainTest	Collect the data from the robot when the data is updated and send it by the CommunicationHandler. Receive commands and execute the appropriate function as define.

3.2. Data Objects Relationships

1. RobotSensorData: contain some Ev3Board and GrovePiBoard
2. CommandHandler: update the robotSensorData according to the commands it execute.
3. RobotBPProgramRunnerListener: update the robotSensorData within external event. Get the updated values and send the command from the js file over the queues that define by the CommunicationHandler.
4. MainTest: update the robotSensorData according to the commands it execute. Send the updated values and get the command from the queues that define by the CommunicationHandler.

4. Behavioral Analysis

4.1. Sequence diagram



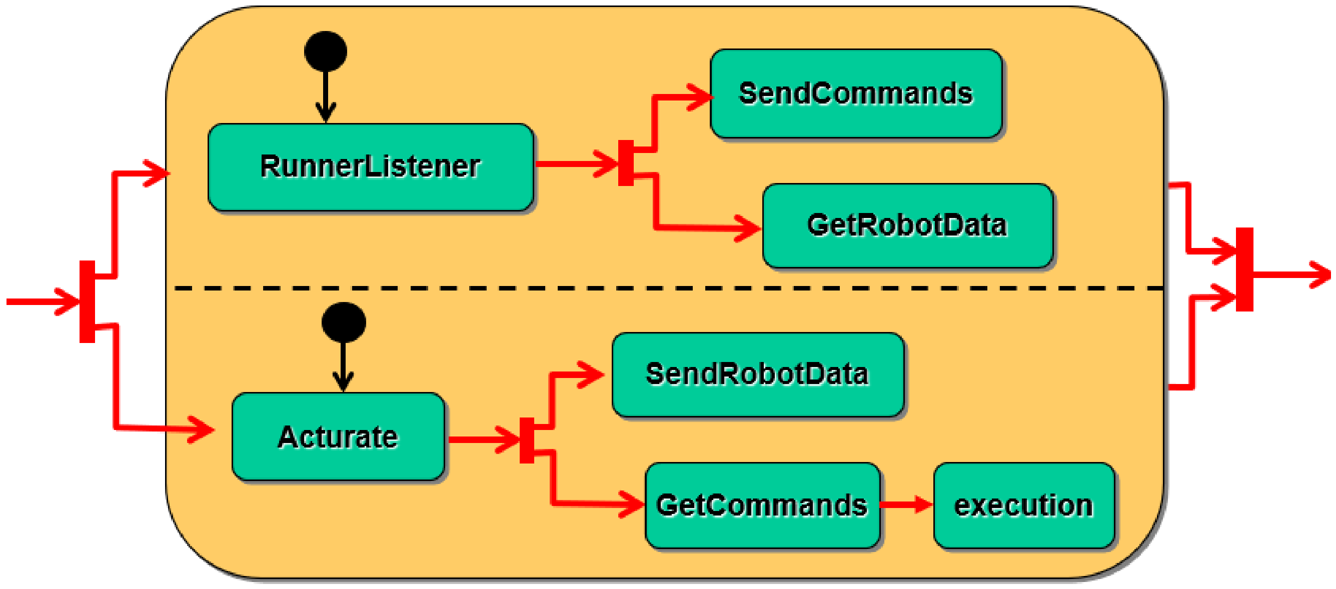
Programmer
1: create
CommunicationHandler
2: addListner(Communication, bProgram)
RobotBPProgramRunnerListner
sd connection open
3: consumeFromQueue("Data", onReceiveCallback)
4: eventSelected(bp, theEvent)
4.1: executeCommand(bp, theEvent)
4.1.1: message = eventDataToJson(theEvent)
4.1.2: send(message, "Commands")
4.1.2.1: send(message, "Commands")
5: create
MainTest
6: create
CommunicationHandler
7: create
RobotSensorData
sd connection open
8: consumeFromQueue("Commands", onReceiveCallback)
9: json = robotSensorsData.toJson()
9.1: send(json, "Data")
9.1.1: send(json, "Data")

4.2. Events

- Stop the run - when sent interrupt the system should stop the robot (EV3 / GrovePi), print to the logger that the run has stopped and close the port to which we are connected.

- Changing the environment in which the robot runs - when we change the location of the robot (and thus the environment in which it runs - physically) the system needs to re-perceive the robot's location with the help of sensors and recalculate the robot's position according to its environment (distance from the wall, colors

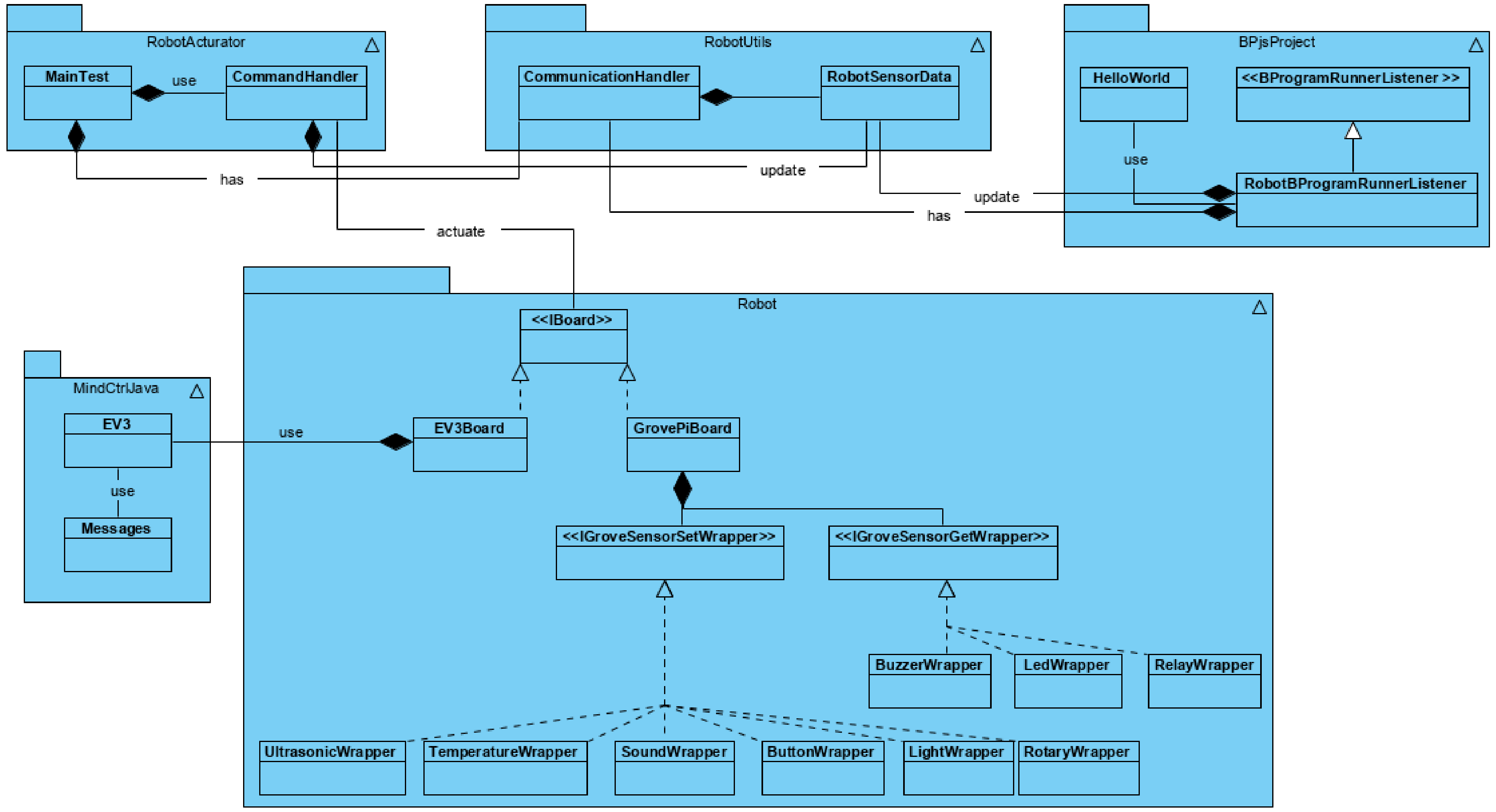
4.3. State chart

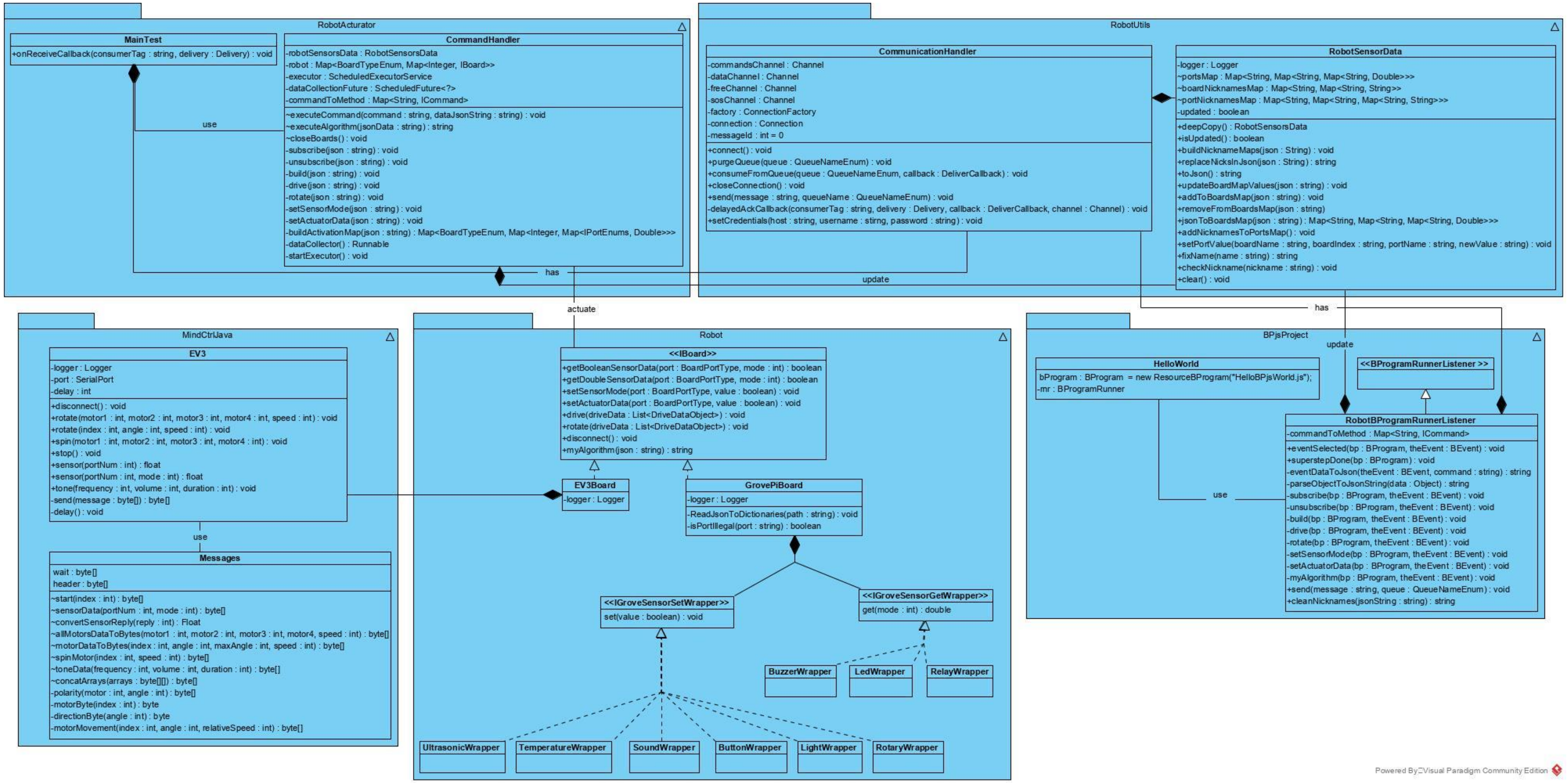


5. Object oriented analysis

5.1. class diagram

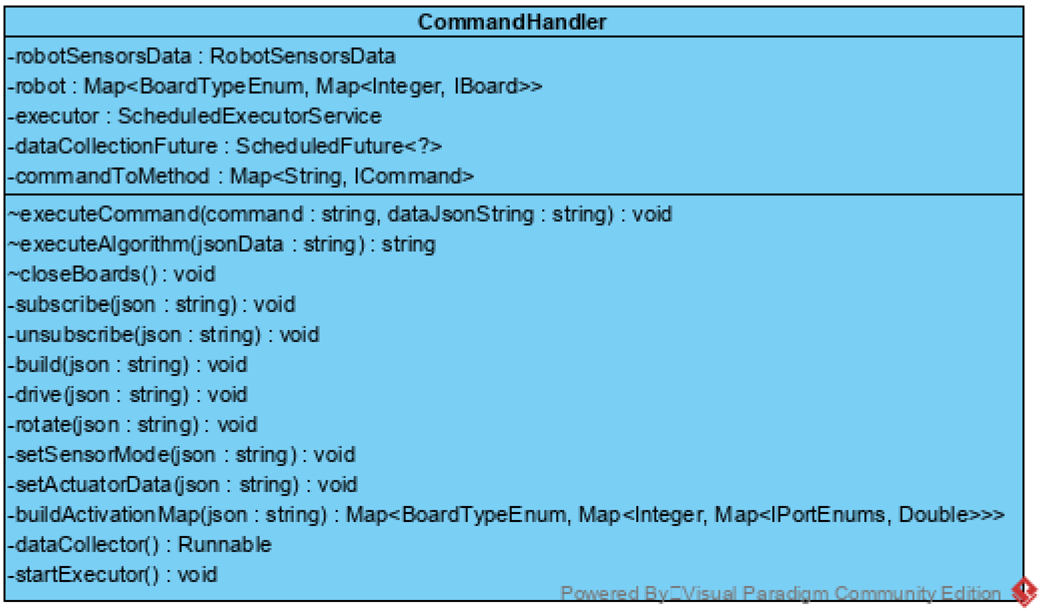
class diagram without attributes and methods:




RobotActurator
MainTest
+onReceiveCallback(consumerTag : string, delivery : Delivery) : void
CommandHandler
-robotSensorsData : RobotSensorsData
-robot : Map<BoardTypeEnum, Map<Integer, IBoard>>
-executor : ScheduledExecutorService
-dataCollectionFuture : ScheduledFuture<?>
-commandToMethod : Map<String, ICommand>
~executeCommand(command : string, dataJsonString : string) : void
~executeAlgorithm(jsonData : string) : string
~closeBoards() : void
-subscribe(json : string) : void
-unsubscribe(json : string) : void
-build(json : string) : void
-drive(json : string) : void
-rotate(json : string) : void
-setSensorMode(json : string) : void
-setActuatorData(json : string) : void
-buildActivationMap(json : string) : Map<BoardTypeEnum, Map<Integer, Map<IPortEnums, Double>>>
-dataCollector() : Runnable
-startExecutor() : void
use
has
update
actuate
RobotUtils
CommunicationHandler
-commandsChannel : Channel
-dataChannel : Channel
-freeChannel : Channel
-sosChannel : Channel
-factory : ConnectionFactory
-connection : Connection
-messageId : int = 0
+connect() : void
+purgeQueue(queue : QueueNameEnum) : void
+consumeFromQueue(queue : QueueNameEnum, callback : DeliverCallback) : void
+closeConnection() : void
+send(message : string, queueName : QueueNameEnum) : void
-delayedAckCallback(consumerTag : string, delivery : Delivery, callback : DeliverCallback, channel : Channel) : void
+setCredentials(host : string, username : stirng, password : string) : void
RobotSensorData
-logger : Logger
~portsMap : Map<String, Map<String, Map<String, Double>>>
~boardNicknamesMap : Map<String, Map<String, String>>
~portNicknamesMap : Map<String, Map<String, Map<String, String>>>
-updated : boolean
+deepCopy() : RobotSensorsData
+isUpdated() : boolean
+buildNicknameMaps(json : String) : void
+replaceNicksInJson(json : String) : string
+toJson() : string
+updateBoardMapValues(json : string) : void
+addToBoardsMap(json : string) : void
+removeFromBoardsMap(json : string)
+jsonToBoardsMap(json : string) : Map<String, Map<String, Map<String, Double>>>
+addNicknamesToPortsMap() : void
+setPortValue(boardName : string, boardIndex : string, portName : string, newValue : string) : void
+fixName(name : string) : string
+checkNickname(nickname : string) : void
+clear() : void
has
update
MindCtrlJava
EV3
-logger : Logger
-port : SerialPort
-delay : int
+disconnect() : void
+rotate(motor1 : int, motor2 : int, motor3 : int, motor4 : int, speed : int) : void
+rotate(index : int, angle : int, speed : int) : void
+spin(motor1 : int, motor2 : int, motor3 : int, motor4 : int) : void
+stop() : void
+sensor(portNum : int) : float
+sensor(portNum : int, mode : int) : float
+tone(frequency : int, volume : int, duration : int) : void
-send(message : byte[]) : byte[]
-delay() : void
use
Messages
wait : byte[]
header : byte[]
~start(index : int) : byte[]
~sensorData(portNum : int, mode : int) : byte[]
~convertSensorReply(reply : int) : Float
~allMotorsDataToBytes(motor1 : int, motor2 : int, motor3 : int, motor4, speed : int) : byte[]
~motorDataToBytes(index : int, angle : int, maxAngle : int, speed : int) : byte[]
~spinMotor(index : int, speed : int) : byte[]
~toneData(frequency : int, volume : int, duration : int) : byte[]
~concatArrays(arrays : byte[][]) : byte[]
-polarity(motor : int, angle : int) : byte[]
-motorByte(index : int) : byte
-directionByte(angle : int) : byte
-motorMovement(index : int, angle : int, relativeSpeed : int) : byte[]
Robot
<<IBoard>>
+getBooleanSensorData(port : BoardPortType, mode : int) : boolean
+getDoubleSensorData(port : BoardPortType, mode : int) : boolean
+setSensorMode(port : BoardPortType, value : boolean) : void
+setActuatorData(port : BoardPortType, value : boolean) : void
+drive(driveData : List<DriveDataObject>) : void
+rotate(driveData : List<DriveDataObject>) : void
+disconnect() : void
+myAlgorithm(json : string) : string
EV3Board
-logger : Logger
GrovePiBoard
-logger : Logger
-ReadJsonToDictionaries(path : string) : void
-isPortIllegal(port : string) : boolean
<<IGroveSensorSetWrapper>>
set(value : boolean) : void
<<IGroveSensorGetWrapper>>
get(mode : int) : double
BuzzerWrapper
LedWrapper
RelayWrapper
UltrasonicWrapper
TemperatureWrapper
SoundWrapper
ButtonWrapper
LightWrapper
RotaryWrapper
BPjsProject
update
HelloWorld
bProgram : BProgram = new ResourceBProgram("HelloBPjsWorld.js");
-mr : BProgramRunner
<<BProgramRunnerListener >>
use
RobotBProgramRunnerListener
-commandToMethod : Map<String, ICommand>
+eventSelected(bp : BProgram, theEvent : BEvent) : void
+superstepDone(bp : BProgram) : void
-eventDataToJson(theEvent : BEvent, command : string) : string
-parseObjectToJsonString(data : Object) : string
-subscribe(bp : BProgram, theEvent : BEvent) : void
-unsubscribe(bp : BProgram, theEvent : BEvent) : void
-build(bp : BProgram, theEvent : BEvent) : void
-drive(bp : BProgram, theEvent : BEvent) : void
-rotate(bp : BProgram, theEvent : BEvent) : void
-setSensorMode(bp : BProgram, theEvent : BEvent) : void
-setActuatorData(bp : BProgram, theEvent : BEvent) : void
-myAlgorithm(bp : BProgram, theEvent : BEvent) : void
+send(message : string, queue : QueueNameEnum) : void
+cleanNicknames(jsonString : string) : string
Powered By Visual Paradigm Community Edition

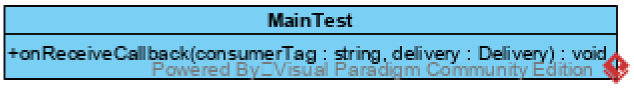
5.2. class description

RobotActuator



```
CommandHandler
-robotSensorsData : RobotSensorsData
-robot : Map<BoardTypeEnum, Map<Integer, IBoard>>
-executor : ScheduledExecutorService
-dataCollectionFuture : ScheduledFuture<?>
-commandToMethod : Map<String, ICommand>
~executeCommand(command : string, dataJsonString : string) : void
~executeAlgorithm(jsonData : string) : string
~closeBoards() : void
-subscribe(json : string) : void
-unsubscribe(json : string) : void
-build(json : string) : void
-drive(json : string) : void
-rotate(json : string) : void
-setSensorMode(json : string) : void
-setActuatorData(json : string) : void
-buildActivationMap(json : string) : Map<BoardTypeEnum, Map<Integer, Map<IPortEnums, Double>>>
-dataCollector() : Runnable
-startExecutor() : void
```

handling the commands that gets from the program by the CommunicationHandler and update the RobotSensorData.

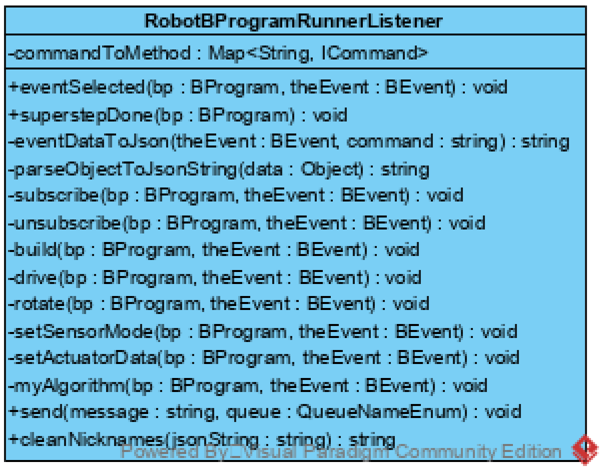


```
MainTest
+onReceiveCallback(consumerTag : string, delivery : Delivery) : void
```

Collect the data from the robot when the data is updated and send it by the CommunicationHandler.

Receive commands and execute the appropriate function as define by the CommandsHandler.

BPjsProject



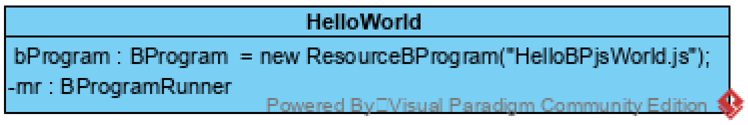
RobotBProgramRunnerListener

-commandToMethod : Map<String, ICommand>

+eventSelected(bp : BProgram, theEvent : BEvent) : void
+superstepDone(bp : BProgram) : void
-eventDataToJson(theEvent : BEvent, command : string) : string
-parseObjectToJsonString(data : Object) : string
-subscribe(bp : BProgram, theEvent : BEvent) : void
-unsubscribe(bp : BProgram, theEvent : BEvent) : void
-build(bp : BProgram, theEvent : BEvent) : void
-drive(bp : BProgram, theEvent : BEvent) : void
-rotate(bp : BProgram, theEvent : BEvent) : void
-setSensorMode(bp : BProgram, theEvent : BEvent) : void
-setActuatorData(bp : BProgram, theEvent : BEvent) : void
-myAlgorithm(bp : BProgram, theEvent : BEvent) : void
+send(message : string, queue : QueueNameEnum) : void
+cleanNicknames(jsonString : string) : string

Get the updated values from the queue that define by the CommunicationHandler and update the RobotSensorData.

Send the command from the js file over the queue that define by the CommunicationHandler

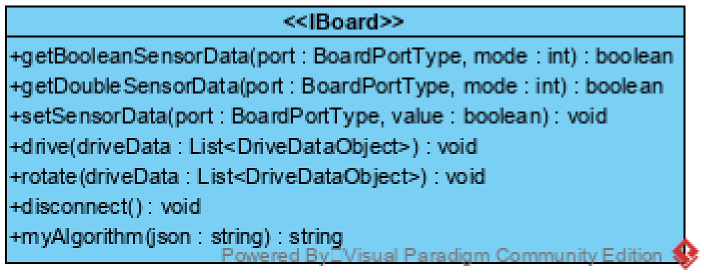


HelloWorld

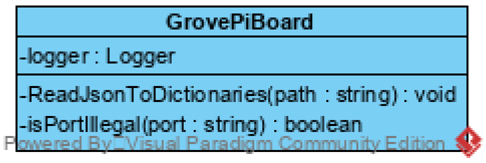
bProgram : BProgram = new ResourceBProgram("HelloBPjsWorld.js");
-mr : BProgramRunner

The main class that define the js file native and add the RobotBPProgramRunnerListener.

Robot

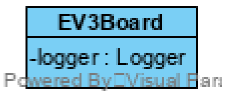

<<IBoard>>
+getBooleanSensorData(port : BoardPortType, mode : int) : boolean
+getDoubleSensorData(port : BoardPortType, mode : int) : boolean
+setSensorData(port : BoardPortType, value : boolean) : void
+drive(driveData : List<DriveDataObject>) : void
+rotate(driveData : List<DriveDataObject>) : void
+disconnect() : void
+myAlgorithm(json : string) : string
Powered By Visual Paradigm Community Edition

Interface for the boards in the system for the actions they can be supported.


GrovePiBoard
-logger : Logger
-ReadJsonToDictionaries(path : string) : void
-isPortIllegal(port : string) : boolean
Powered By Visual Paradigm Community Edition

Represent the Grove Pi and responsible for performing operations related to Grove Pi.

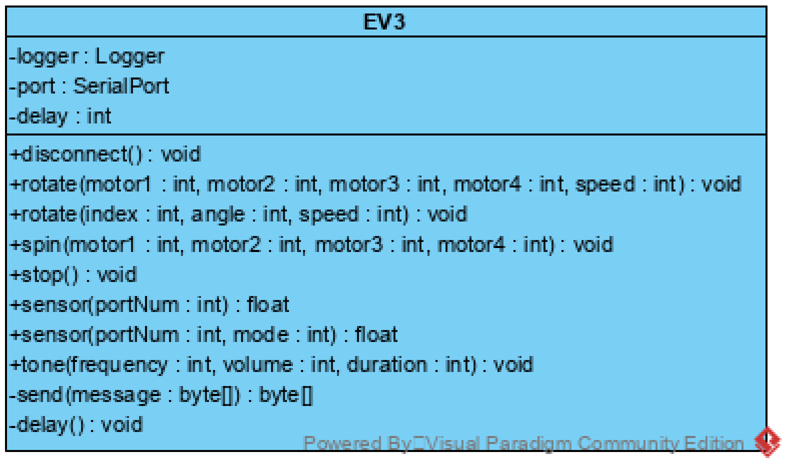
The GrovePi has analog(A0-A2) and digital(D3-D8) sensors that derived in the GrovePiBoard to sensors their values can be set and sensors their values only get.



Represent the Ev3 Mindstorms and responsible for performing operations related to EV3.

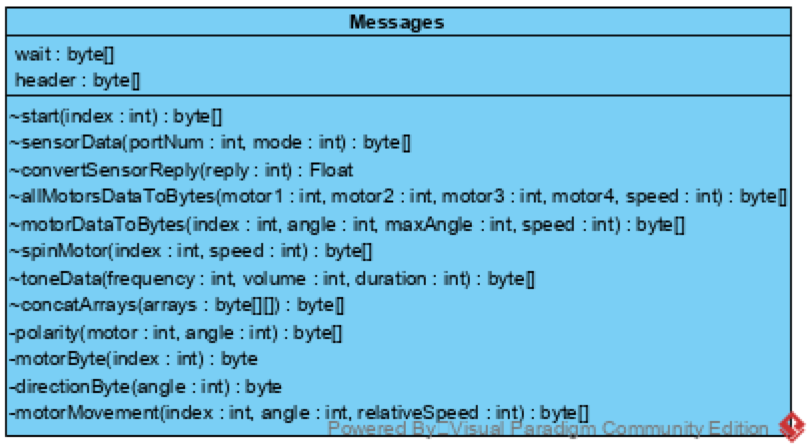
The EV3Board has EV3 instance such that the implementation of the functions in the IBoard interface calls the functions of the EV3.

MindCtrlJava



```
EV3
-logger : Logger
-port : SerialPort
-delay : int
+disconnect() : void
+rotate(motor1 : int, motor2 : int, motor3 : int, motor4 : int, speed : int) : void
+rotate(index : int, angle : int, speed : int) : void
+spin(motor1 : int, motor2 : int, motor3 : int, motor4 : int) : void
+stop() : void
+sensor(portNum : int) : float
+sensor(portNum : int, mode : int) : float
+tone(frequency : int, volume : int, duration : int) : void
-send(message : byte[]) : byte[]
-delay() : void
```

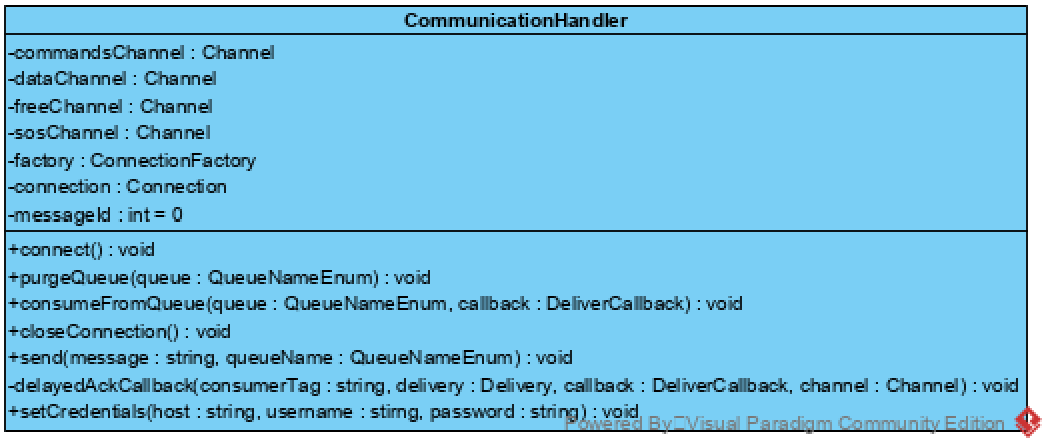
The EV3 send messages of bytes to the physical EV3 board which connected by the SerialPort.



```
Messages
wait : byte[]
header : byte[]
~start(index : int) : byte[]
~sensorData(portNum : int, mode : int) : byte[]
~convertSensorReply(reply : int) : Float
~allMotorsDataToBytes(motor1 : int, motor2 : int, motor3 : int, motor4, speed : int) : byte[]
~motorDataToBytes(index : int, angle : int, maxAngle : int, speed : int) : byte[]
~spinMotor(index : int, speed : int) : byte[]
~toneData(frequency : int, volume : int, duration : int) : byte[]
~concatArrays(arrays : byte[][]) : byte[]
-polarity(motor : int, angle : int) : byte[]
-motorByte(index : int) : byte
-directionByte(angle : int) : byte
-motorMovement(index : int, angle : int, relativeSpeed : int) : byte[]
```

Class for the messages data itself. Messages are split into smaller parts that can be easily constructed using this class.

RobotUtils



```
CommunicationHandler
-commandsChannel : Channel
-dataChannel : Channel
-freeChannel : Channel
-sosChannel : Channel
-factory : ConnectionFactory
-connection : Connection
-messageId : int = 0
+connect() : void
+purgeQueue(queue : QueueNameEnum) : void
+consumeFromQueue(queue : QueueNameEnum, callback : DeliverCallback) : void
+closeConnection() : void
+send(message : string, queueName : QueueNameEnum) : void
-delayedAckCallback(consumerTag : string, delivery : Delivery, callback : DeliverCallback, channel : Channel) : void
+setCredentials(host : string, username : stirng, password : string) : void
```

Handle the communication between the commands that send to the robot from the user and the data that back from the robot.

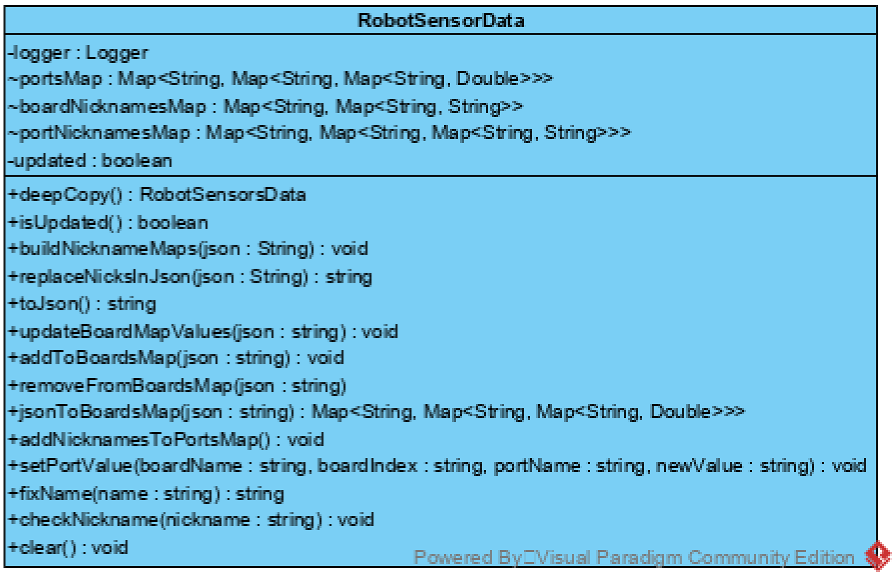
Open different queues for different commands and send and receive the data over the appropriate queue.

Commands queue: pass regular (not special) commands. Example: drive/rotate command.

SOS queue: pass special commands that must receive and can't be override for the speed communication of the system. Example: build command.

Data queue: pass the data that back from the robots in json which contain the value for each port.

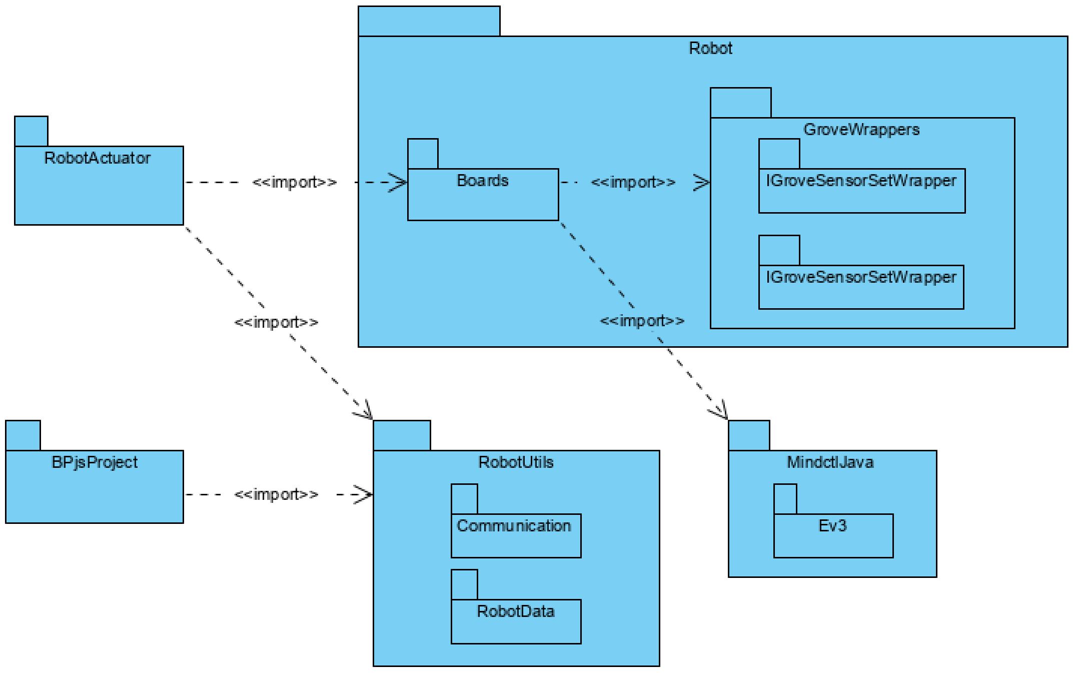
Free queue: pass json the programmer wants for his algorithm.



```
RobotSensorData
-logger : Logger
~portsMap : Map<String, Map<String, Map<String, Double>>>
~boardNicknamesMap : Map<String, Map<String, String>>
~portNicknamesMap : Map<String, Map<String, Map<String, String>>>
-updated : boolean
+deepCopy() : RobotSensorsData
+isUpdated() : boolean
+buildNicknameMaps(json : String) : void
+replaceNicksInJson(json : String) : string
+toJson() : string
+updateBoardMapValues(json : string) : void
+addToBoardsMap(json : string) : void
+removeFromBoardsMap(json : string)
+jsonToBoardsMap(json : string) : Map<String, Map<String, Map<String, Double>>>
+addNicknamesToPortsMap() : void
+setPortValue(boardName : string, boardIndex : string, portName : string, newValue : string) : void
+fixName(name : string) : string
+checkNickname(nickname : string) : void
+clear() : void
```

Hold the values that back from the robot's sensors in the portsMap. Support nicknames in addition to the indexes of the boards and ports.

5.3. Package diagram



5.4. Unit testing

In the implementing of the unit tests we will perform an assert for the values returning from the functions for each class. We want to make sure that the values returning from the functions match what is expected of them.

We will use Mockito to simulate the actions belonging to the robot and thus we will examine the required function on its own without internal calls to external factors, in particular we will ignore internal calls to actions belonging to the robot and not related to the functionality being tested.

The specification of the unit testing can be found in the Testing Document file in the ‘Testing functional requirements’ section.

6. Appendices

Hardware Components

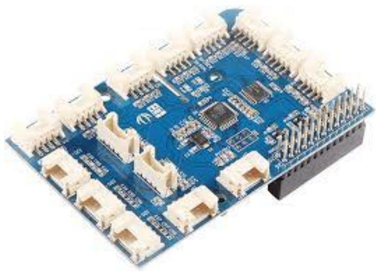
Raspberry Pi

Raspberry Pi is a series of small single-board computers.
Most boards include:

- USB ports
- HDMI port
- On board Wi-Fi and Bluetooth Connectivity.
- SD slot
- 40-pins General Purpose Input/Output (GPIO)

This is out main component.
We intend to use an RP board for computational power and to control peripheral components related to our project.
RP boards runs a Debian-based (32-bit) Linux distribution (other OS are possible).

GrovePi+



GrovePi+ is an add-on board that brings
Grove sensors to the Raspberry Pi.
It is compatible with almost all Raspberry Pi models.

GrovePi+ is easy-to-use and modular.
No need for soldering or breadboards,
just plug in your Grove sensors and start programming directly.

As a sub-component of our system, The GrovePI+ will provide a bridge to controlling I/O devices from our main Raspberry Pi component.
The main difference between the GrovePI+ to the RP (in scope of our project) is that the GrovePI+ is a very nice to have optional component.
The Raspberry Pi board is a must.

Lego Mindstorms EV3

Lego Mindstorms is a hardware and software structure which is produced by Lego for the development of programmable robots based on Lego building blocks.
Each version of the system includes a computer Lego brick that controls the system, a set of modular sensors and motors,
and Lego parts to create the mechanical systems.

Similar to the GrovePI+, we intent to use the the EV3 computer brick as a sub-component of our system.
The brick will “bridge” between our main program running on the Raspberry Pi and the Lego sensors and motors.
Again, this is a very nice to have optional component, that vastly expands the usability of the robot.
But this is not a must have, and our project should be able to work without it.

7. Testing

For the unit tests we will define mocks board to demonstrate the behavioral of the robot without connect it. In this way the code testing is not depending in the hardware.

We will enforce the non-functional requirements by the response of the users who will work with the product.

In this way we can make sure that the installation is simple and does not consume excessive resources and that the communication is efficient and fast, as described in the non-functional requirements contained in the ARD file.