Introduction to UniboDISI DDR-Robots (2018)

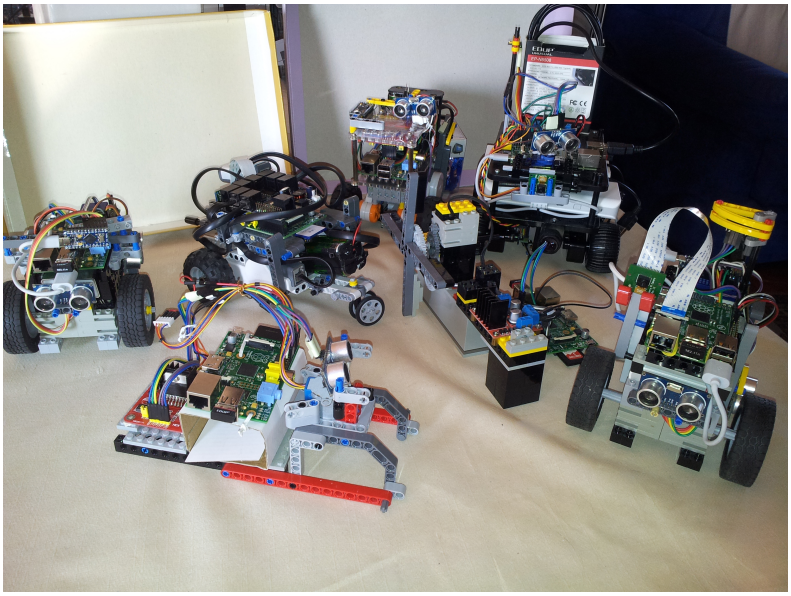
Antonio Natali

Alma Mater Studiorum – University of Bologna
Campus di Cesena - via Pavese 50, Cesena, Italy,
Viale Risorgimento 2,40136 Bologna, Italy
antonio.natali@studio.unibo.it

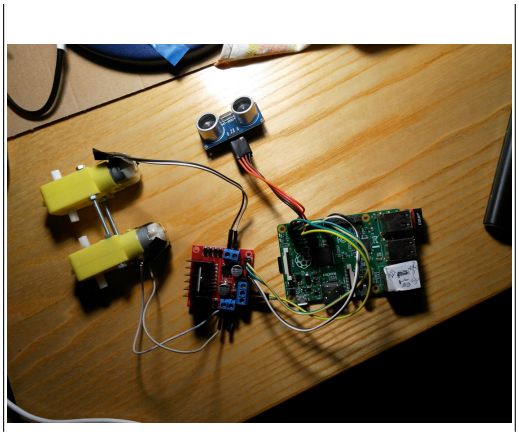
Table of Contents

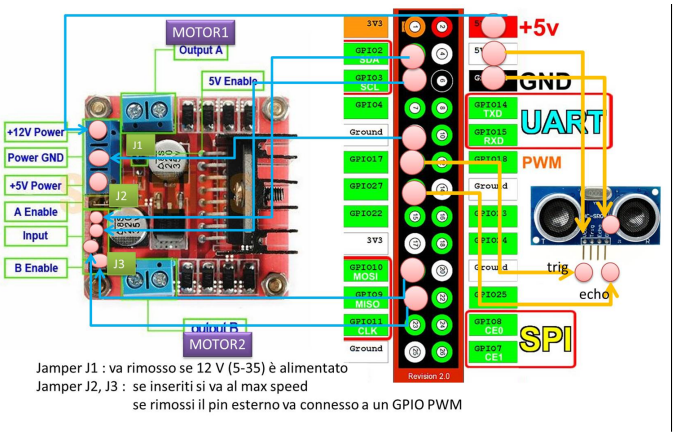
1 Introduction to UniboDisi DDR-Robots

From the physical point of view, a `BaseRobot` is a custom device built with low-costs components including sensors, actuators and processing units like *RaspberryPi* and *Arduino*. Examples are given in the following picture:



The hardware components of a `BaseRobot` and their configuration can be quite different from robot to robot. For example, a DDR-robot built around a RaspberryPi could have the basic hardware structure shown hereunder:





In this case, the correct configuration of the hardware connections can be tested by some simple bash code :

```
#!/bin/bash
# ------------------------------------------------------------------
# nanoMotorDriveA.sh
# test for nano0
# Key-point: we can manage a GPIO pin by using the GPIO library.
# On a PC, edit this file as UNIX
# ------------------------------------------------------------------

in1=2 #WPI 8 BCM 2 PHYSICAL 3
in2=3 #WPI 9 BCM 3 PHYSICAL 5
inwp1=8
inwp2=9

if [ -d /sys/class/gpio/gpio2 ]
then
 echo "in1 gpio${in1} exist"
 gpio export ${in1} out
else
 echo "creating in1 gpio${in1}"
 gpio export ${in1} out
fi

if [ -d /sys/class/gpio/gpio3 ]
then
 echo "in2 gpio${in2} exist"
 gpio export ${in2} out
else
 echo "creating in2 gpio${in2}"
 gpio export ${in2} out
fi

gpio readall

echo "run 1"
 gpio write ${inwp1} 0
 gpio write ${inwp2} 1
 sleep 1.5

echo "run 2"
 gpio write ${inwp1} 1
 gpio write ${inwp2} 0
 sleep 1.5

echo "stop"
 gpio write ${inwp1} 0
 gpio write ${inwp2} 0

gpio readall
```

Listing 1.1. `nanoMotorDriveA.sh`

1.1 Beyond the hardware level

However, some software layer is required to hide configuration differences as much as possible and to build a 'technology independent' layer, to be used by application designers.

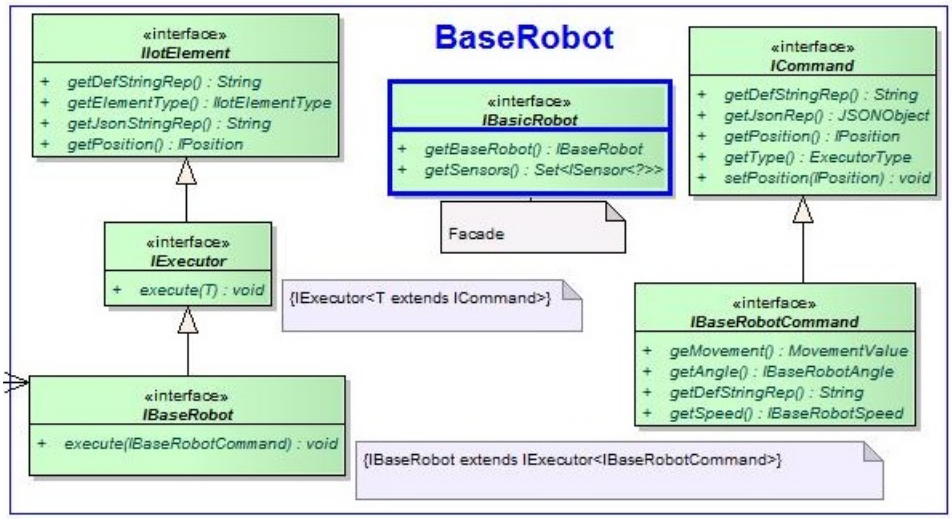
A software layer of this kind is provided by the library `labbaseRobotSam.jar`. More specifically:

it.unibo.lab.baseRobot	The basic software for differential drive robots that are able to move and to acquire sensor data. Library: *labbaseRobotSam.jar*.
it.unibo.robotRaspOnly.BasicRobotUsageNaive in project *it.unibo.mbot2018*	Example of the usage of the `API` of a BaseRobot.

1.2 A model for the BaseRobot

The main goal of the *labbaseRobotSam.jar* library is to simplify the work of an application designer by exposing at application level a very simple model of a `BaseRobot`:

- As regards the `structure`, a `BaseRobot` can be viewed as a entity composed of two main parts:
  - An executor (with interface `IBaseRobot`), able to move the robot according to a a prefixed set of movement commands (`IBaseRobotCommand`)
  - A set of `GOF` -observable sensors (each with interface `ISensor`), each working as an active source of data.
- As regards the `interaction`, a `BaseRobot` can be viewed as a `POJO` that implements the interface `IBaseRobot`, while providing a (possibly empty) set of observable sensors;
- As regards the `behavior`, a `BaseRobot` is an object able to execute `IBaseRobotCommand` and able to update sensor observers defined by the application designer.



The interface `IBasicRobot` is introduced as a (`GOF`) *Facade* for the model.

```
1 package it.unibo.iot.baseRobot.hlmodel;
2 import java.util.Set;
3 import it.unibo.iot.executors.baseRobot.IBaseRobot;
4 import it.unibo.iot.sensors.ISensor;
5
6 public interface IBasicRobot {
7     public IBaseRobot getBaseRobot();   //selector
8     public Set<ISensor<?>> getSensors(); //selector
9 }
```

Listing 1.2. `IBasicRobot.java`

1.3 The BasicRobot class

The class `BasicRobot` provides a factory method to create a `BaseRobot` and to select its main components.

```java
package it.unibo.iot.baseRobot.hlmodel;
import java.util.Set;
import it.unibo.iot.configurator.Configurator;
import it.unibo.iot.executors.baseRobot.IBaseRobot;
import it.unibo.iot.sensors.ISensor;

public class BasicRobot implements IBasicRobot{
private static IBasicRobot myself = null;
public static IBasicRobot getRobot(){
    if( myself == null ) myself = new BasicRobot();
    return myself;
}
public static IBaseRobot getTheBaseRobot(){
    if( myself == null ) myself = new BasicRobot();
    return myself.getBaseRobot();
}
//-------------------------------------------
private Configurator configurator;
private IBaseRobot robot ;
    //Hidden constructor
    protected BasicRobot() {
        init();
    }
    protected void init(){
        configurator = Configurator.getInstance();
        robot       = configurator.getBaseRobot();
    }
    public IBaseRobot getBaseRobot(){
        return robot;
    }
    public Set<ISensor<?>> getSensors(){
        Set<ISensor<?>> sensors = configurator.getSensors();
        return sensors;
    }
}
```

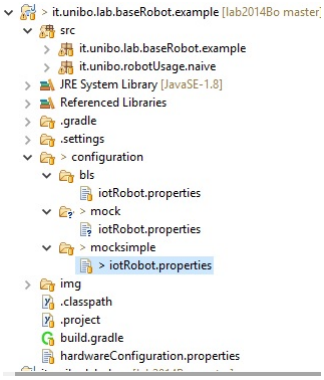
Listing 1.3. `BasicRobot.java`

This class shows that the work to set-up and access to the internal structure of a `BaseRobot` is delegated to a `Configurator`. This Configurator reads the specification of the robot structure written in a file named *iotRobot.properties* (see Subsection 1.5).

1.4 Using a BaseRobot

Let us introduce a robot with configuration named `mocksimple` (see Subsection 1.5), equipped with two motors and a distance sensor, all simulated.

1.4.1 The project workspace The application designer must organize its project workspace as shown in the following snapshot:

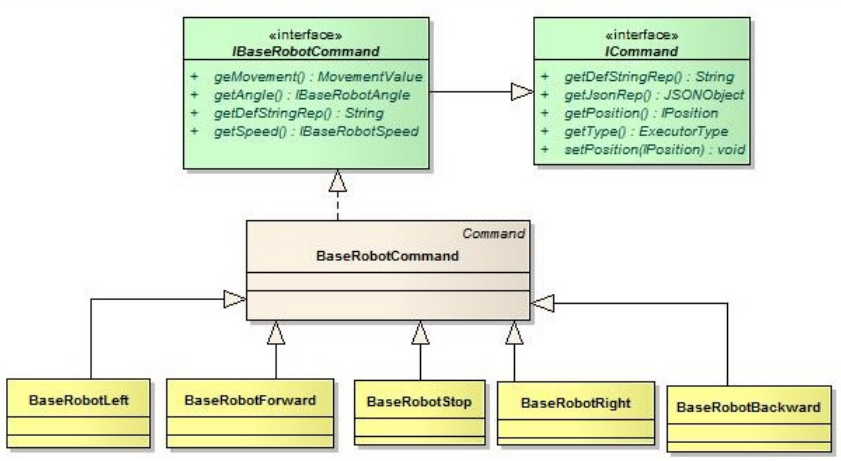


1.4.2 The code The application: *i)* first creates a sensor observer and adds it to all the sensors; *ii)* then it tells the robot to execute the commands (sent from an user console) it is able to understand.

```
1  public class BasicRobotUsageNaive{
2  private final IBaseRobotSpeed SPEED_LOW = new BaseRobotSpeed(BaseRobotSpeedValue.ROBOT_SPEED_LOW);
3  private final IBaseRobotSpeed SPEED_MEDIUM = new BaseRobotSpeed(BaseRobotSpeedValue.ROBOT_SPEED_MEDIUM);
4  private final IBaseRobotSpeed SPEED_HIGH = new BaseRobotSpeed(BaseRobotSpeedValue.ROBOT_SPEED_HIGH);
5  private IBaseRobot robot ;
6
7      public BasicRobotUsageNaive() {
8          configue();
9      }
10     protected void configue() {
11         IBasicRobot basicRobot = BasicRobot.getRobot();
12         robot                  = basicRobot.getBaseRobot();
13         addObserverToSensors(basicRobot);
14     }
15     public void handleUserCommands() {
16         try {
17             while(true) {
18                 int v    =  System.in.read();
19                 if( v == 13 || v == 10 ) continue;
20                 char cmd = (char)v;
21                 System.out.println( "INPUT= " + cmd + "(" + v +")");
22                 executeTheCommand( cmd );
23             }
24         } catch (IOException e) {
25             e.printStackTrace();
26         }
27     }
28     public void executeTheCommand( char cmd ) {
29         IBaseRobotCommand command = null;
30         switch( cmd ) {
31             case 'h' : command = new BaseRobotStop(SPEED_LOW );break;
32             case 'w' : command = new BaseRobotForward(SPEED_HIGH );break;
33             case 's' : command = new BaseRobotBackward(SPEED_HIGH );break;
34             case 'a' : command = new BaseRobotLeft(SPEED_MEDIUM );break;
35             case 'd' : command = new BaseRobotRight(SPEED_MEDIUM );break;
```

```
36              default: System.out.println( "Sorry, command not found");
37          }
38          if( command != null ) robot.execute(command);
39      }
40      protected void addObserverToSensors(IBasicRobot basicRobot){
41          ISensorObserver observer = new SensorObserver();
42          for (ISensor<?> sensor : basicRobot.getSensors()) {
43              System.out.println( "doJob sensor= " + sensor.getDefStringRep() + " class= " + sensor.getClass().getName()
                    );
44              sensor.addObserver(observer);
45          }
46      }
47
48   public static void main(String[] args) throws Exception{
49          new BasicRobotUsageNaive().handleUserCommands();
50   }
51  }
```

Listing 1.4. `BasicRobotUsageNaive.java`



1.5 The work of the Configurator

An object of class *it.unibo.iot.configurator.Configurator*:

1. first looks at the file `hardwareConfiguration.properties` to get the name of the robot (e.g. mock)
2. then, it consults the file `iotRobot.properties` into the directory `configuration/mock`

For each specification line, the `Configurator` calls (by using Java reflection) a factory method of the specific `DeviceConfigurator` class associated to the name of the robot.

For example, let us consider a Mock robot equipped with two motors and a distance sensor, all simulated: (file *configuration/mocksimple/iotRobot.properties*):

```
1  # ================================
2  # ioRobot.properties for mocksimple robot
3  # Pay attention to the spaces
4  # ================================
5  # --------- MOTORS ---------
6  motor.left=mock
7  motor.left.private=false
```

```
#
motor.right=mock
motor.right.private=false
# --------- SENSORS ---------
distance.front=mock
distance.front.private=false
# --------- COMPOSED COMPONENT ---------
actuators.bottom=ddmotorbased
actuators.bottom.name=motors
actuators.bottom.comp=motor.left,motor.right
actuators.bottom.private=true
# --------- MAIN ROBOT ---------
baserobot.bottom=differentialdrive
baserobot.bottom.name=mocksimple
baserobot.bottom.comp=actuators.bottom
baserobot.bottom.private=false
```

Listing 1.5. `configuration/mocksimple/iotRobot.properties`

The Configurator calls (using an object of class `IotComponentsFromConfiguration`):

- `getMotorDevice` of *it.unibo.iot.device.mock.DeviceConfigurator* (for **motor.left=mock**)
- `getBaseRobotDevice` of *it.unibo.iot.device.differentialdrive.DeviceConfigurator* (for **baserobot.bottom=differentialdrive**)
- `getDistanceSensorDevice` of *it.unibo.iot.device.mock.DeviceConfigurator* (for **distance.front=mock**)
- `getMotorDevice` of *it.unibo.iot.device.mock.DeviceConfigurator* (for **motor.right=mock**)
- `getActuatorsDevice` of *it.unibo.iot.device.ddmotorbased.DeviceConfigurator* (for **actuators.bottom=ddmotorbased**)

1.6 From mocks to real robots

In order to use a physical robot rather than a `Mock` robot, the software designer must simply change the specification of the robot configuration; the application code is unaffected. For example, to use our standard 'nano' robots, we have to include into the `configuration/nano` directory the following configuration file:

```
# ================================
# ioRobot.properties for nano robot
# Pay attention to the spaces
# ================================
# --------- MOTORS ---------
motor.left=gpio.motor
motor.left.pin.cw=8
motor.left.pin.ccw=9
motor.left.private=false
#
motor.right=gpio.motor
motor.right.pin.cw=12
motor.right.pin.ccw=13
motor.right.private=false
# --------- SENSORS ---------
distance.front_top=hcsr04
distance.front_top.trig=0
distance.front_top.echo=2
distance.front_top.private=false
# --------- COMPOSED COMPONENT ---------
actuators.bottom=ddmotorbased
actuators.bottom.name=motors
```

```
23 actuators.bottom.comp=motor.left,motor.right
24 actuators.bottom.private=true
25 # --------- MAIN ROBOT ---------
26 baserobot.bottom=differentialdrive
27 baserobot.bottom.name=nano
28 baserobot.bottom.comp=actuators.bottom
29 baserobot.bottom.private=false
```

Listing 1.6. `configuration/nano/iotRobot.properties`

2 Sensors and Sensor Data

The current version of the `BaseRobot` system implements the following sensors:

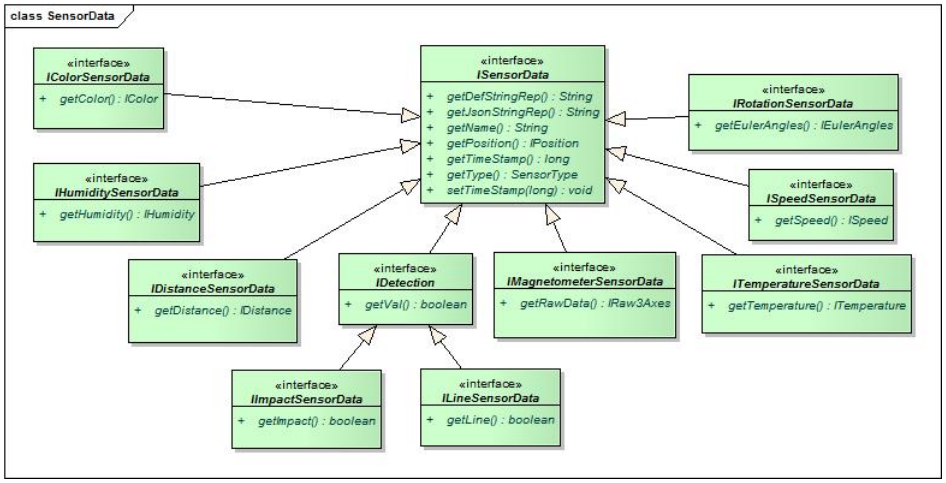
```
RobotSensorType: Line | Distance | Impact | Color | Magnetometer
```

Each sensor:

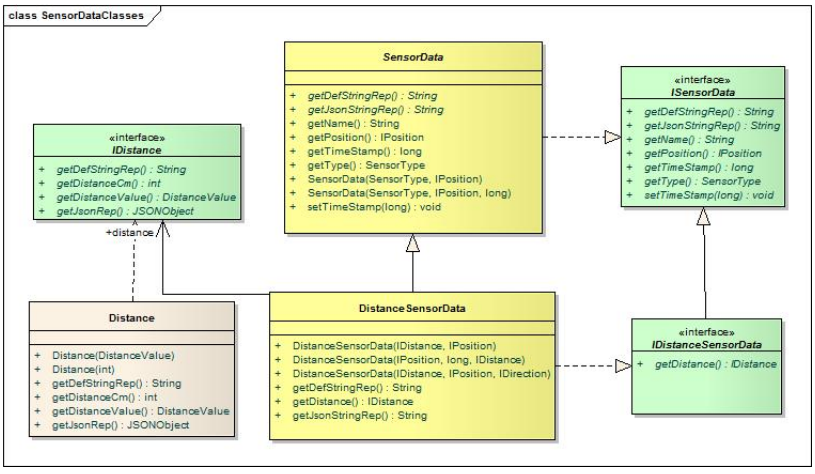
– is associated to a position that can assume one of the following values:

```
DONTCARE|
FRONT | RIGHT | LEFT | BACK | TOP | BOTTOM |
FRONT_RIGHT | FRONT_LEFT | BACK_RIGHT | BACK_LEFT |
TOP_RIGHT | TOP_LEFT | BOTTOM_RIGHT | BOTTOM_LEFT |
FRONT_TOP | BACK_TOP | FRONT_TOP_LEFT | FRONT_TOP_RIGHT |
FRONT_RIGHT_TOP | FRONT_LEFT_TOP | BACK_RIGHT_TOP | BACK_LEFT_TOP
```

– is a source of data, each associated to a specific class and interface:



Each class related to sensor data inherits from a base class `SensorData`. For example:



The class `SensorData` provides operations to represent data as strings in two main formats: *i)* in `Prolog` syntax and *ii)* in `Json` syntax. For example:

2.0.1 Sensor data representation in Prolog (high level)

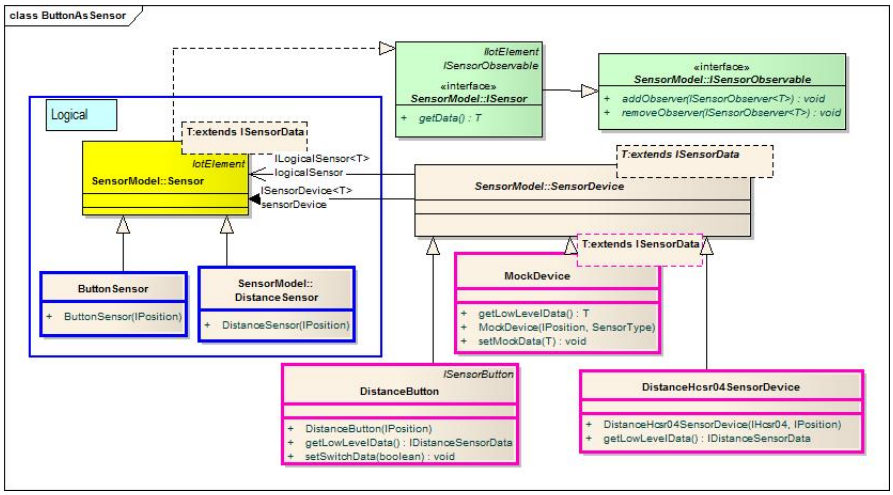
COLOR	`color(255 255 255, front)`
DISTANCE	`distance(43,forward, front)`
IMPACT	`impact(touch/loss, front)`
LINE	`line(lineLeft/lineDetected, bottom)`
MAGNETOMETER	`magnetometer(x(50),y(100),z(0), front)`

2.0.2 Sensor data representation in Json (low level)

COLOR	`"p":"f","t":"c","d":"color":"r":255,"b":255,"g":255,"tm":148...`
DISTANCE	`"p":"f","t":"d","d":"cm":43,"tm":14...`
IMPACT	`"p":"f","t":"i","d":"detection":"touch","tm":14...`
LINE	`"p":"b","t":"l","d":"detection":"lineDetected","tm":14...`
MAGNETOMETER	`"p":"f","t":"m","d":"raw3axes":"x":50,"y":100,"z":0,"tm":14...`

2.1 Sensor model

The sensor subsystem of the `BaseRobot` is based on the class `Sensor` that represents a sensor from the logical point of view. Each sensor is associated to a class that inherits form `Sensor`.



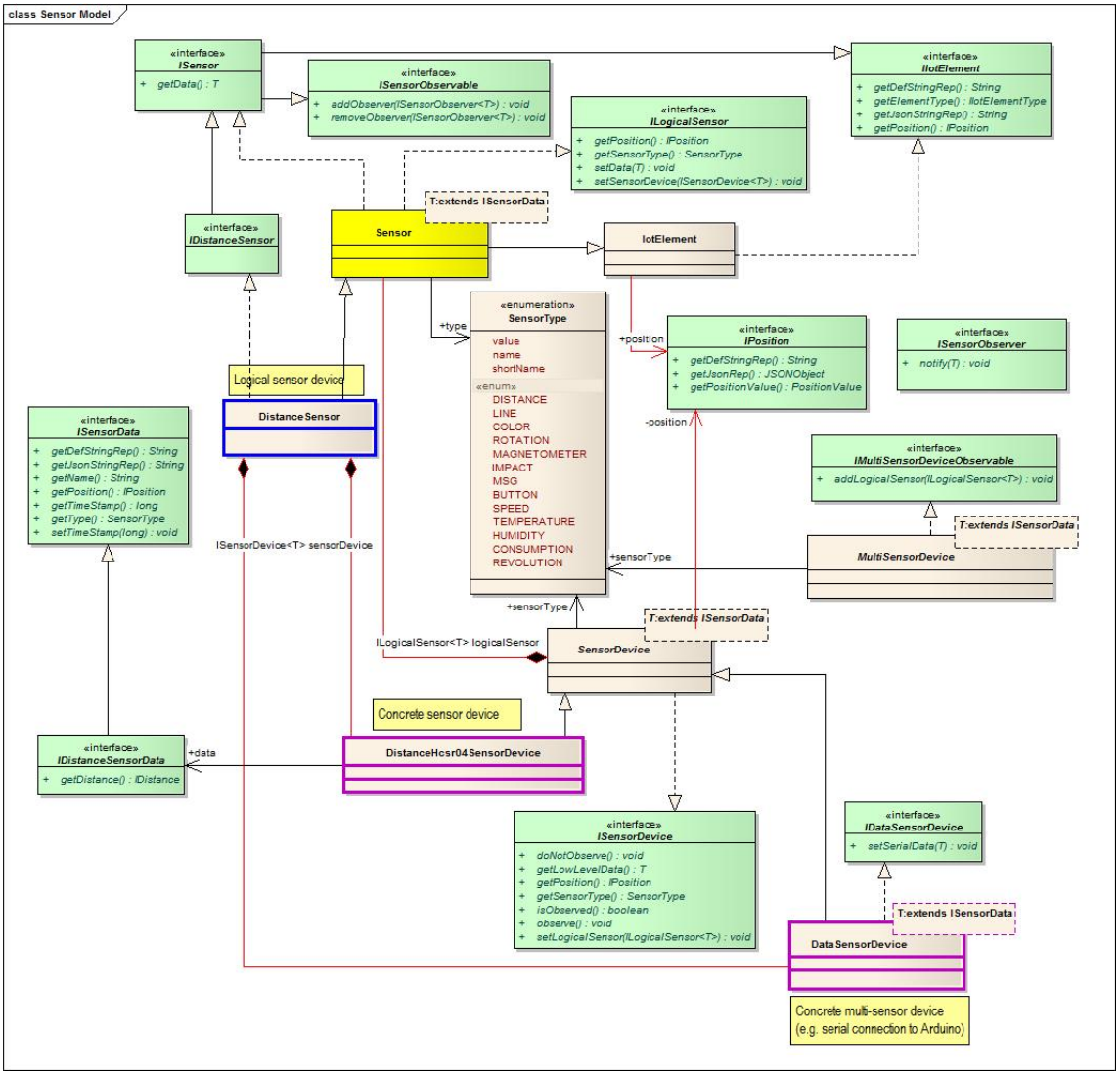
The model reported in the picture above shows that:

- A `DistanceSensor` is a logical `Sensor` associated (by the *Configurator*) to a concrete device (e.g. `DistanceHcsr04SensorDevice`). The same is true for a `ButtonSensor` (impact).
- A `DistanceHcsr04SensorDevice` is a concrete `SensorDevice` that updates its logical sensor when it produces a value. The same is true for a `DistanceButton` (impact). The diagrma shows also a `MockDevice` that can be used to simulate the behavior of the supported sensors.

– Any `Sensor` is an observable entity that, when updated from its concrete device, updates the registered application observers.

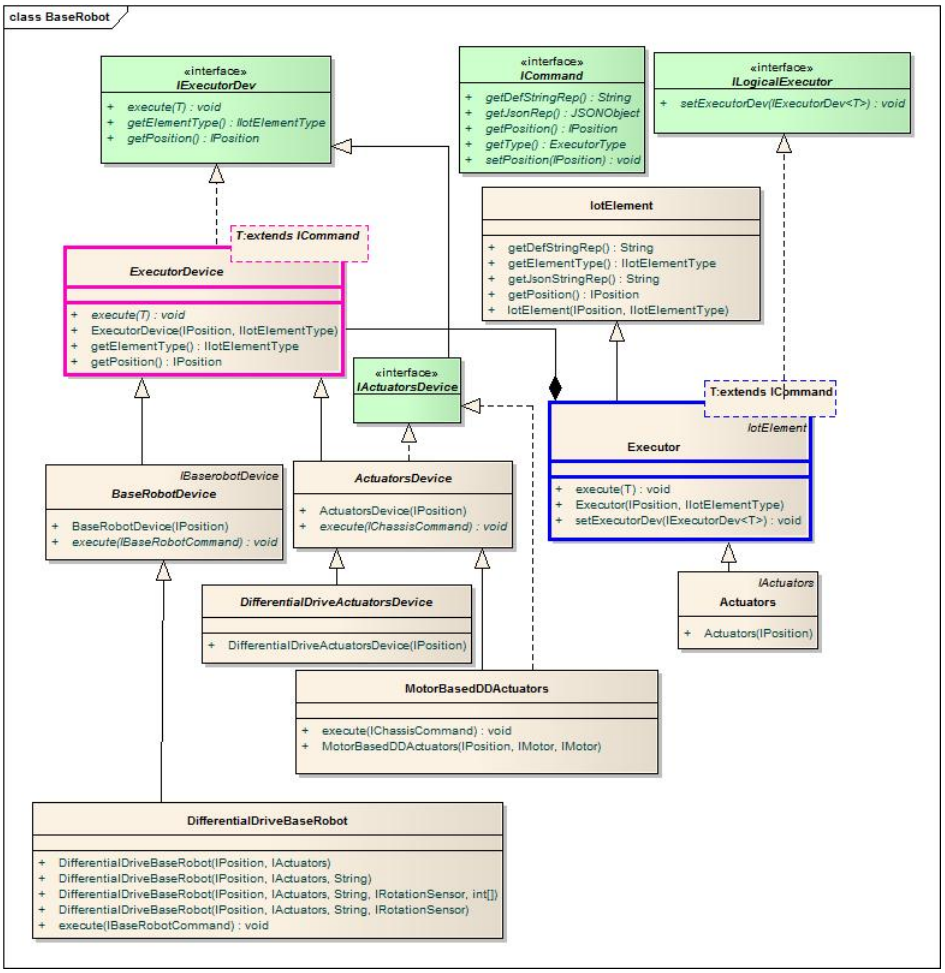
In this way, according to the `GOF` pattern *Bridge*, the `Sensor` abstraction hierarchy is decoupled from the hierarchy of `SensorDevice` implementation.

A more detailed picture is reported hereunder:



3 Actuators and Executors

The `GOF` Bridge pattern has been adopted also to model the 'motors' and the more general concept of 'executor'.



However this part of the `BaseRobot` can be ignored by the application designer and it is no more discussed here.