Monitoring the status of the garbage containers

Architecture & Design

Authors: Xavier Cerqueda Puig

Bernat Garcia Torrentsgeneros

Pierre Biojoux

Quentin Studeny

Junyoung Bang

Joonas Luukkanen

Supervisor: Torben Gregersen

Date: 12/12/2016

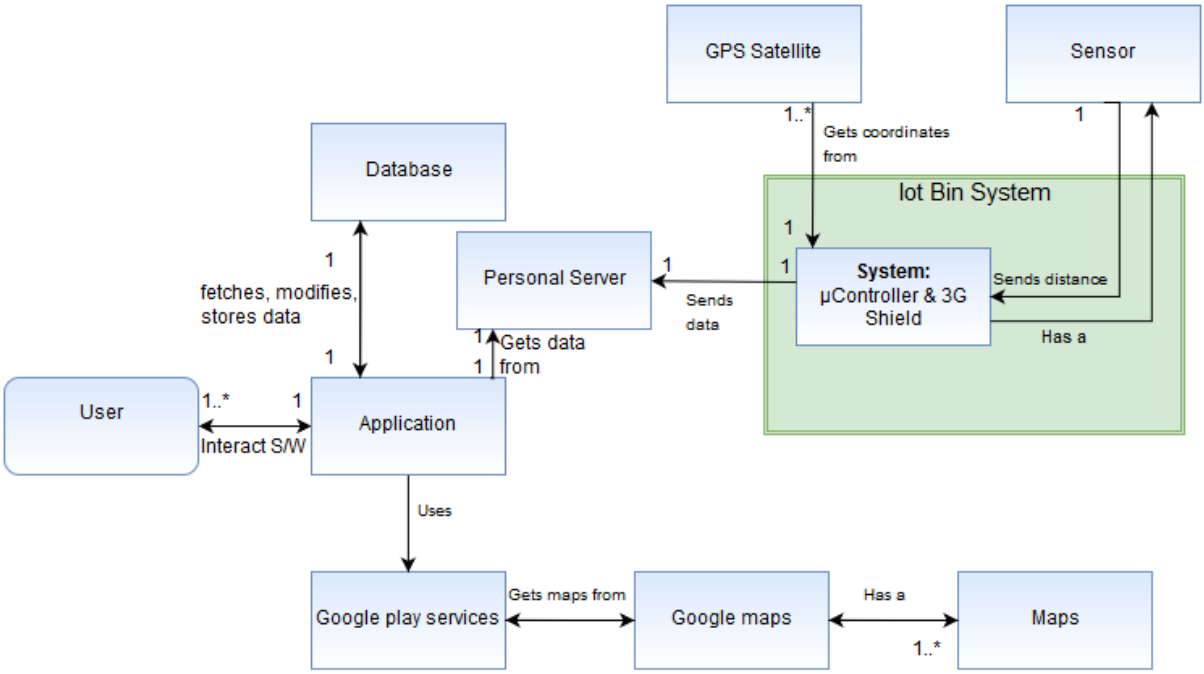
1. Table of contents

2. Introduction

After the requirement specification and pre-analysis documents, we can proceed with the description of the hardware:

This document contains all the information about the internal and external connections that conforms the system.

2.1 Domain Model



3. Hardware Architecture

3.1 Introduction

Now, we can proceed with the description of the hardware.

This part contains all information about the internal and external connections that conforms the system.

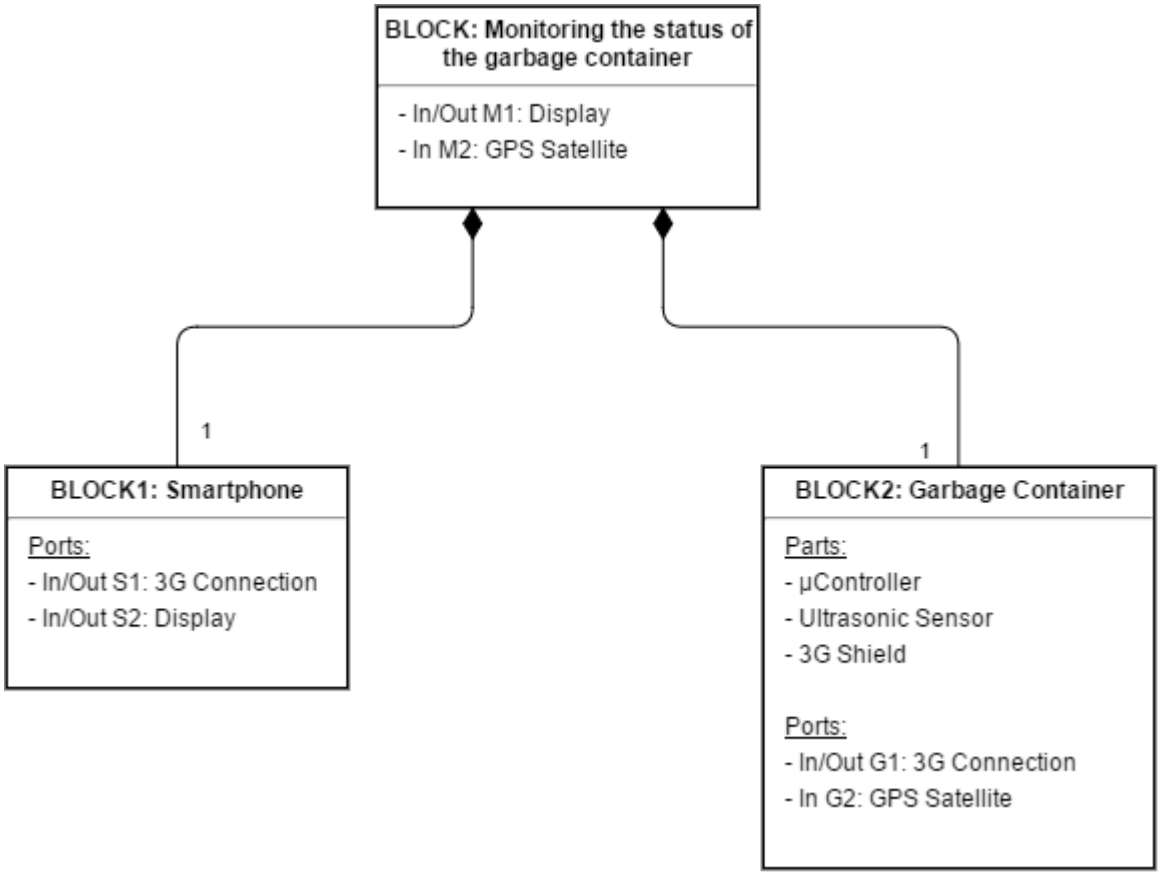
We will use two different types of sysml diagrams to explain the functionality: BDD (Block Diagram Description) and IBD (Internal Block Diagram).

3.2 Monitoring Garbage Container

3.2.1 BDD

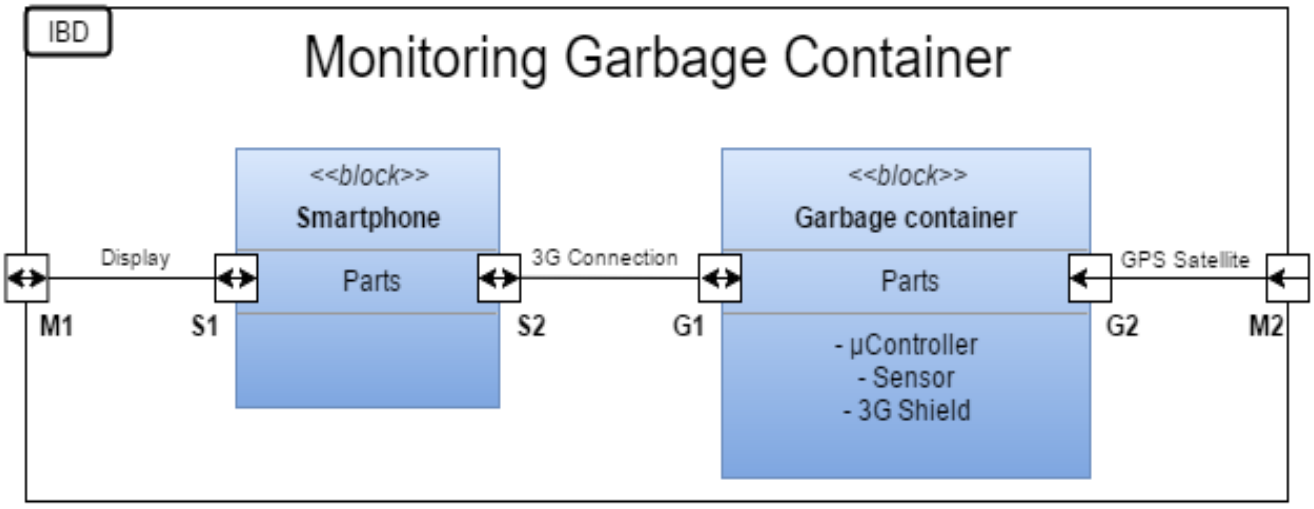
The next figure shows the main system blocks and their connections.

The Smartphone have the information for show to the user and the microcontroller gets the data from the sensor and send it to the smartphone via 3G.



Block Name	Function Description	Port Name	Direction	Comment
Monitoring the status of garbage container	This block shows the integration of the device	M1	In/Out	User interface with phone
		M2	In	Connection with the GPS Satellite
Smartphone	This Smartphone has the app to show the status of garbage container	S1	In/Out	The user controls the system through screen
		S2	In/Out	Through 3G internet, we receive the data from the Arduino
Garbage container	It consists in a microcontroller controlling the ultra-sonic sensor and communication with Phone	G1	In/Out	Connection between phone and garbage container through 3G
		G2	In	Connection between garbage container and GPS Satellite

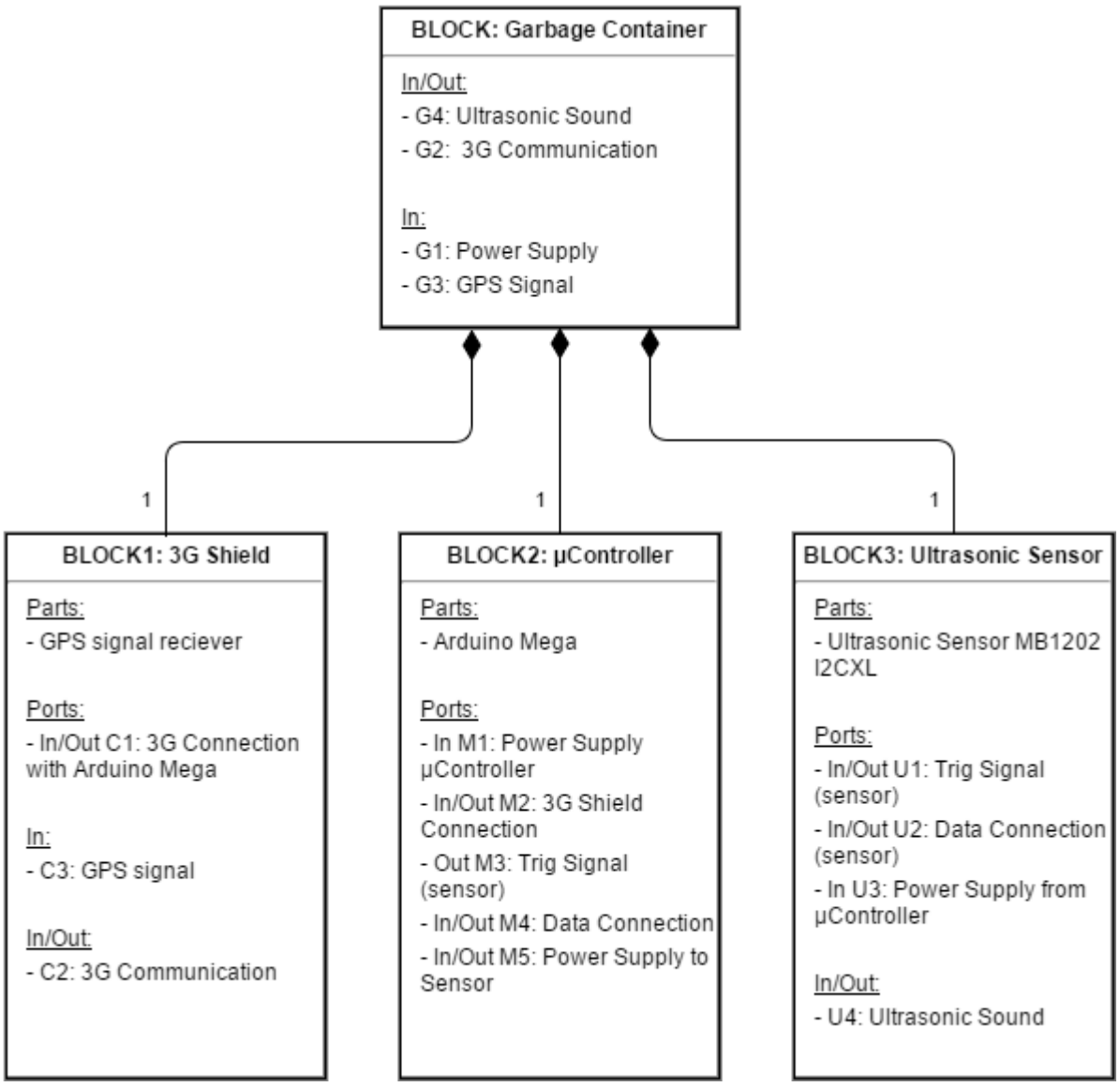
3.2.2 IBD



Name	Description	Port 1	Port 2
Display	User touch the screen to interact with the main App	M1	S1
3G Connection	Creates the connection between the User Smartphone and the µController to send and receive data	S2	G1
GPS Satellite	GPS Satellite sends data about the location to the µController	G2	M2

3.2 Garbage Container

3.2.1 BDD

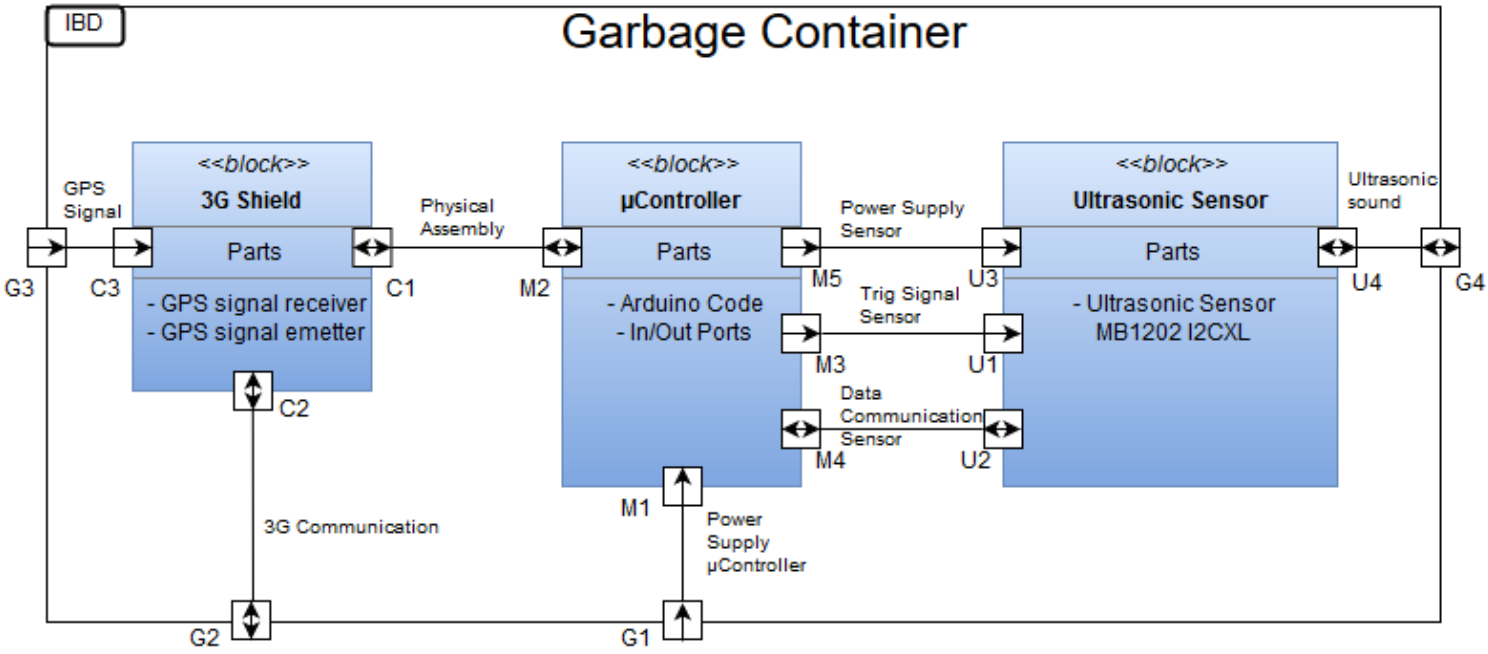


Block name	Function Description	Port Name	Direction	Comment
Garbage container	Provides the 3G connection to our Arduino board.	G1	IN	The garbage container has a battery that we use to supply all the external devices.
		G2	IN/OUT	Port to enable connection with garbage container and 3G shield.
		G3	IN	Garbage container received GPS signal through this port.
		G4	IN/OUT	Port to enable connection with garbage container and Ultrasonic sound.
µController	This block consists of all HW devices for our project.	M1	IN	Connection where we supply our` Arduino board. Directly connected with the battery.
		M2	IN/OUT	Direct connection to 3G shield using the port.
		M3	OUT	Port to enable the distance control
		M4	IN/OUT	Port where we receive data from the ultrasonic censor.
		M5	IN/OUT	Port to supply to sensor.
3G shield	Provides the 3G connection to our Arduino board.	C1	IN/OUT	Direct connection to 3G shield with µController using the port.
		C2	IN/OUT	Port to enable connection with garbage container and 3G shield.
		C3	IN	3G shield received GPS signal through this port.
Ultrasonic	We have one	U1	IN	Port to enable the distance control

sensor	ultrasonic sensor.	U2	IN/OUT	Port to activate the beginning of the sensor measurement
		U3	IN	Port to supply power from micro controller
		U4	IN/OUT	Port to enable connection with garbage container and Ultrasonic sound.

3.2.1 IBD

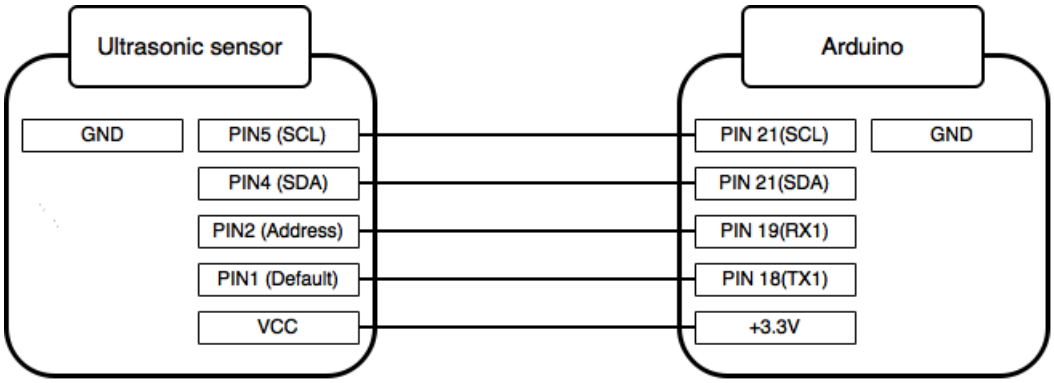
Name	Description	Port 1	Port 2
Power Supply µC	This connection is directly from the battery and it supplies the µController with 5 V the Arduino ATmega2560.	G1	M1
3G Communication	This connection makes garbage container available for using 3G.	G2	C2
GPS Signal	The 3G shield get GPS signal from this connection	G3	C3
Ultrasonic sound	Ultrasonic sensor emit ultrasonic waves to get information about distance.	G4	U4
Physical Assembly	The 3G shield is mounted above the µC and pins fit perfectly. With this connection the 3G shield is fed and communication can begin between the two devices.	C1	M2
Power Supply Sensor	The V_{DC} pin of the sensor is connected with one of the ports of the µC that can provide 5V.	M5	U3
Trig Signal Sensor	This signal allows the sensor to start measuring. Clock signal, that the µC sends to sensor.	M3	U1
Data Communications Sensor	Connection that permits the data communication between sensor and µC.	M4	U2



4. Hardware design

4.1 Ultrasonic sensor

Since the ultrasonic sensor is already pretty much in our ideal configuration, we won't need to modify it. It shall be used as such, connected to the microprocessor and aimed at the bottom of the garbage bin.



4.1.1 Performance

The sensor will always report a range between 20-cm and 765-cm. If a target is detected closer than 20-cm, the sensor will typically report the distance as 20-cm, and if no target is detected by the time a target at 765cm would have been found, the sensor will limit the maximum reported distance to 765cm.

This formula is the way how to calculate the distance with sensor and object.

$$\mathrm{DIstance(cm)} = \frac{\left(343\left(\frac{m}{s}\right)\cdot \text{high level time}(\mu s)\right)}{2}\cdot\frac{1}{10^6}\left(\frac{s}{\mu s}\right)\cdot\left(\frac{10^2 \text{cm}}{1\text{m}}\right)$$

To obtain real distance with centimetre we have to divide 2 and multiply with values.

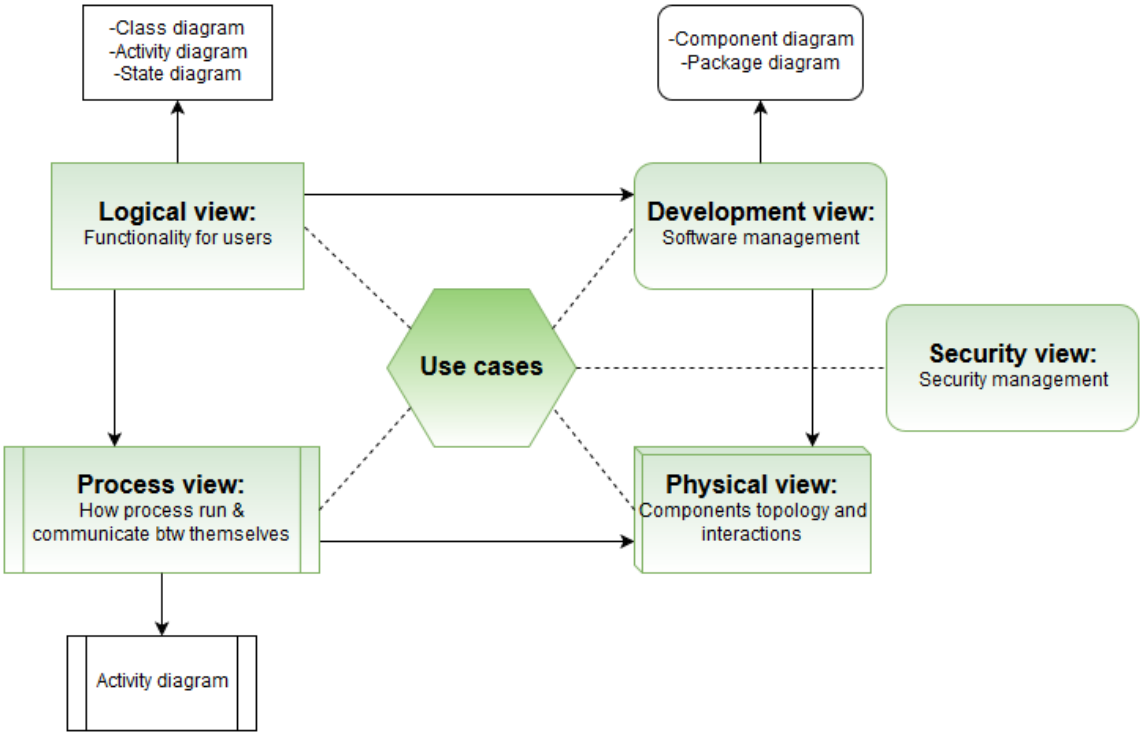
5. Software architecture

5.1 Introduction

We opted for the agile-style development, since it fits our requirements, as we're not a huge team, and we do not have the necessary knowledge to perfectly time our organizational planning.

We will use the N+1 model view as a structural basis for this following part of the document.

5.2 Document structure



Model view figure

5.2.1 Use case view

The **use case view** is an explanation of how the system is supposed to be handled: to showcase this, each use case can be explained with an example. As such, it is intended to carry the intention of the system, and give insight about the whole project.

5.2.2 Logical View

The **logical view** describes the structure design of the system, but as the users see and understand it. There is only one logical view, that illustrate the arborescence. This view would be refined every time the system is improved and changed. Its objective it to list all the desired functionalities, in order to ensure their presence and operability.

5.2.3 Process view

The **process view** depicts the process decomposition, and the exchange of information between those IT systems, or with the outside world. Like the logical view, it consists of one view, and would be refined in a similar manner.

5.2.5 Security view

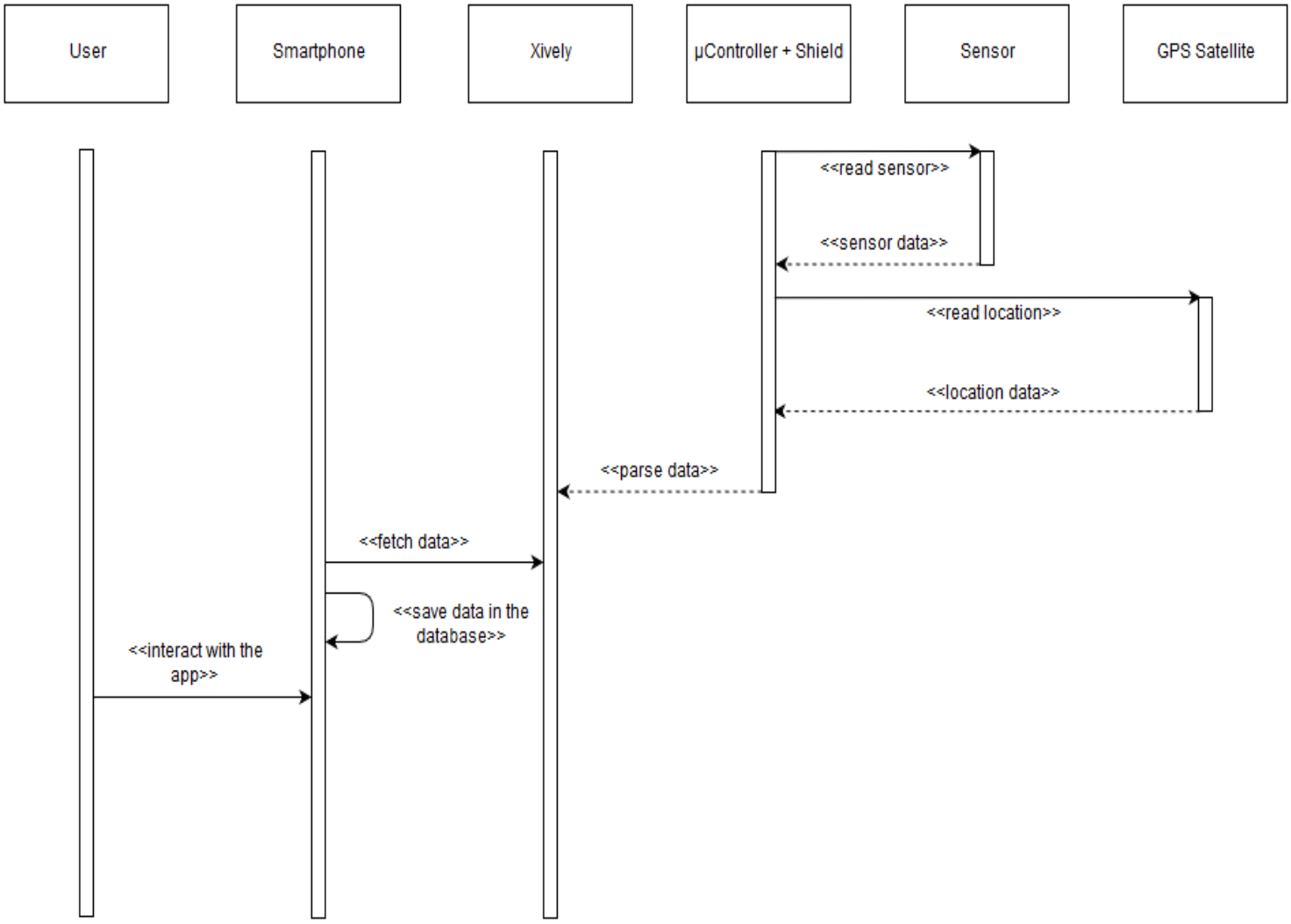
The **Security view** is about protection data. It examines the system to establish what information is stored and processed, how valuable it is, what threats exist, and how they can be addressed. Major concerns for this view are understanding how to assure that the system is available to only those that have permission, and how to protect the system from unauthorized tampering.

5.2.6 Development view

The **development view** is about the developer's view, the tools used and things like code structure, dependencies or standards and constraints, i.e.: relationships and dependencies between modules. (It is also known as the implementation view.) It mainly relies on the UML component view. Intended for developers working on different modules and subsystems.

5.3 Logical view

In this part, we describe how the software works and particularly the code. You can see the system sequence diagram below. Then we see in more details the smartphone application and the garbage container perspective.



System Sequence Diagram

5.3.1 Smartphone perspective

5.3.1.1 Use cases – Sequence Diagrams

5.3.1.1.1 Use case 1: Authenticate

This use case describes how the user can introduce the data to authenticate and stablish the connection with the server in order to access the application system.

Use case 1 – Involved classes

TrashCollector:

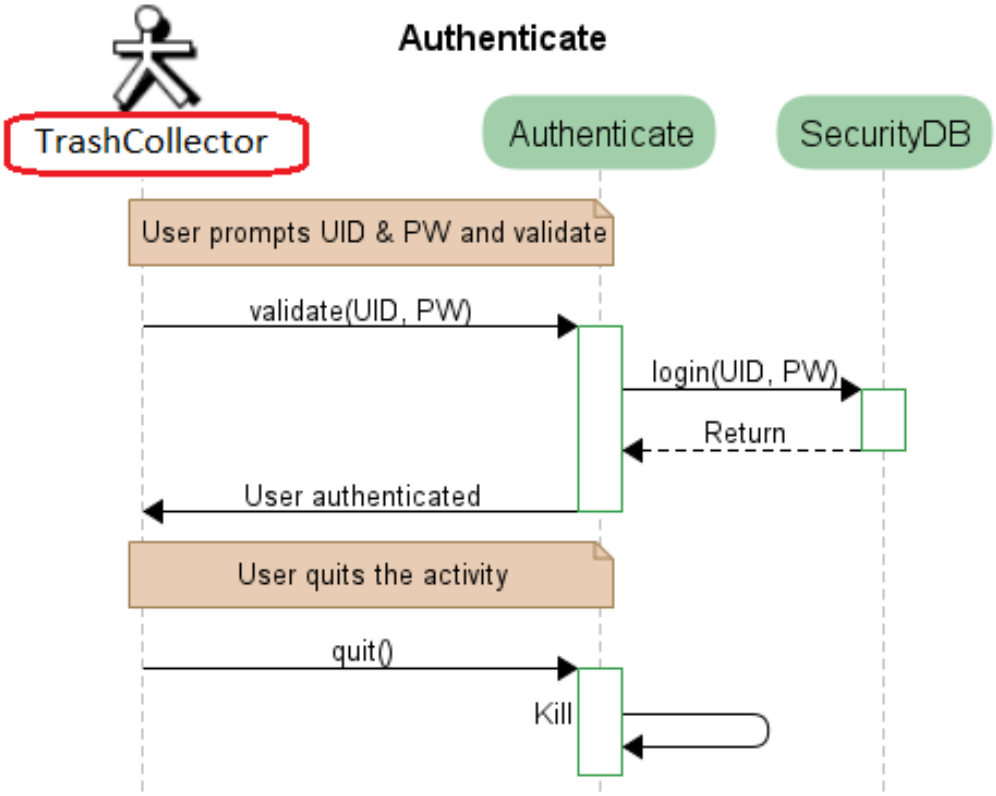
The trash collector is an actor. It's him who interacts with the app and see what is displaying on the screen. Therefore, it is not a class.

Authenticate:

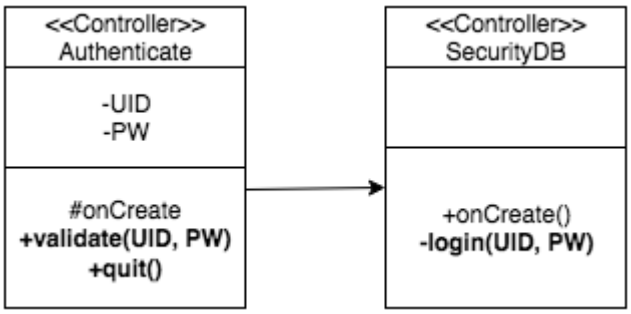
The controller-class Authenticate is the connection between the user and the smartphone. The user introduces the authentication data and then the smartphone sends a request to the database in order to validate this data.

SecurityDB:

The controller-class SecurityDB is the interface responsible for carrying out the task of checking if the data introduced by the user is correct or not. If the data is correct, allows the access to the App, and if it's not, ask again for new data and block the access.



Authenticate Sequence Diagram



Authenticate Class Diagram

5.3.1.1.2 Use case 2: Check status of garbage bins

This use case describes how the smartphone shows the user, the status of all garbage containers and see which ones need to be emptied using a list with all the containers.

Use case 2 – Involved classes

GarbageHandler:

The controller-class garbage handler is to control database of garbage bins inside the Android.

CheckStatus:

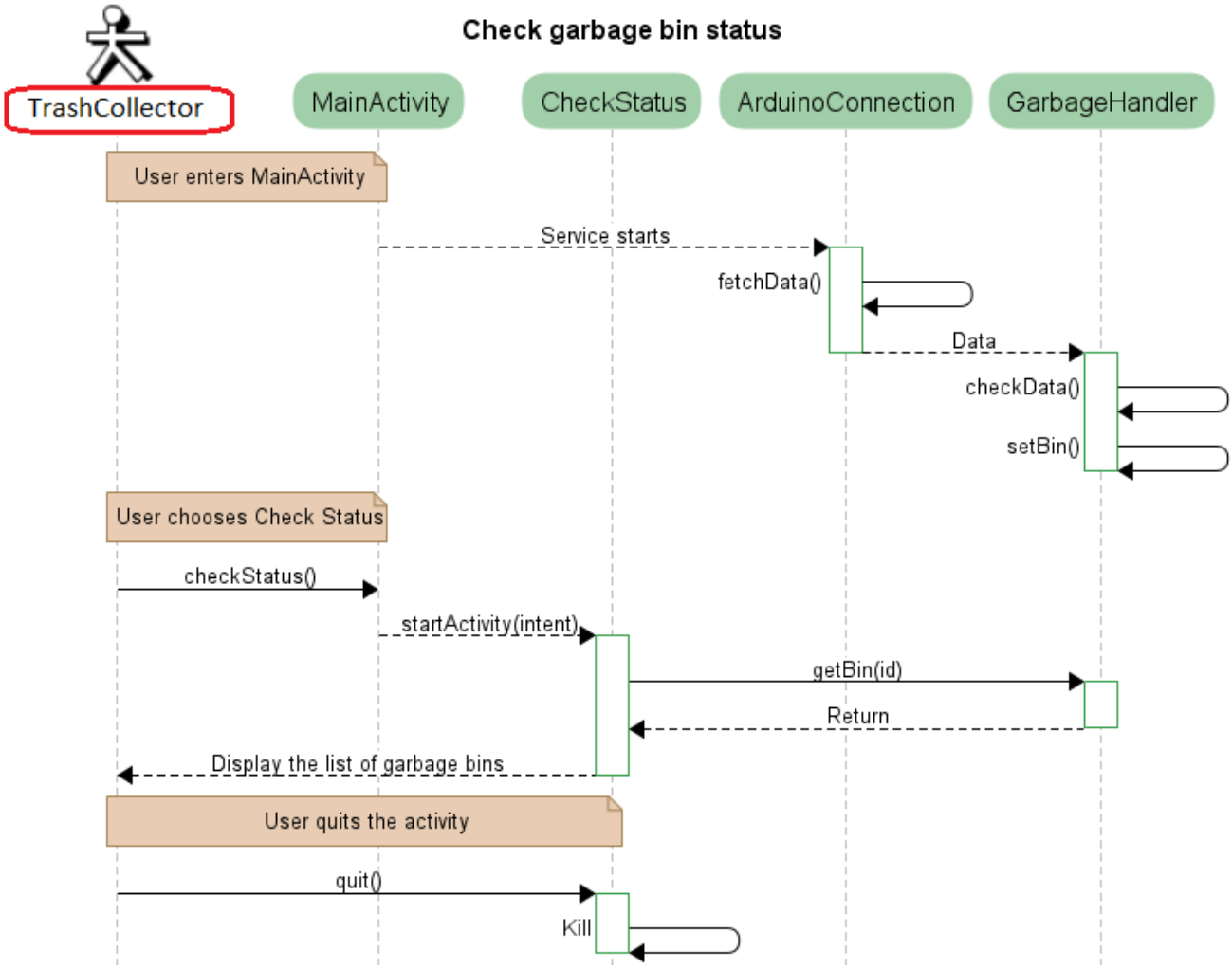
The controller-class check status is the connection between the user and the smartphone. That allows the user to see the status of all the garbage.

MainActivity:

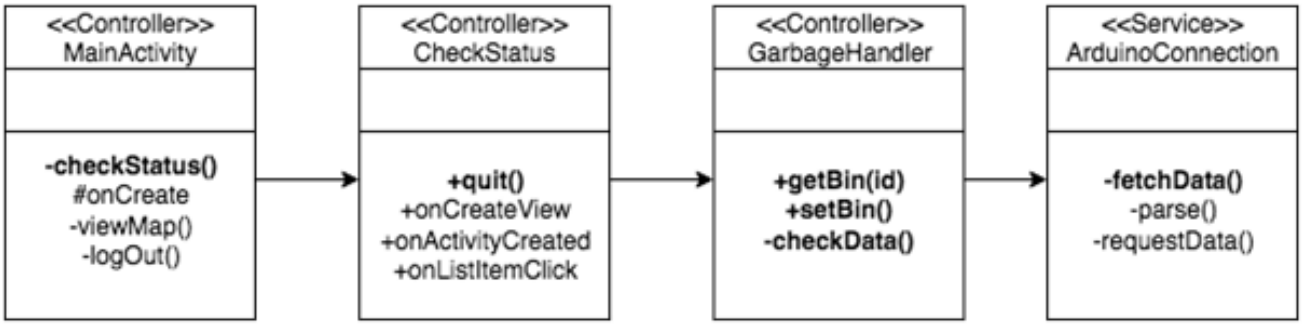
The controller-class main activity is the menu where you have two features: list of garbage bins and map where you can see the location of garbage bins.

ArduinoConnection:

The service-class Arduino connection allows the trespass of data between the database and Xively server.



Check Status Sequence Diagram



Check Status Class Diagram

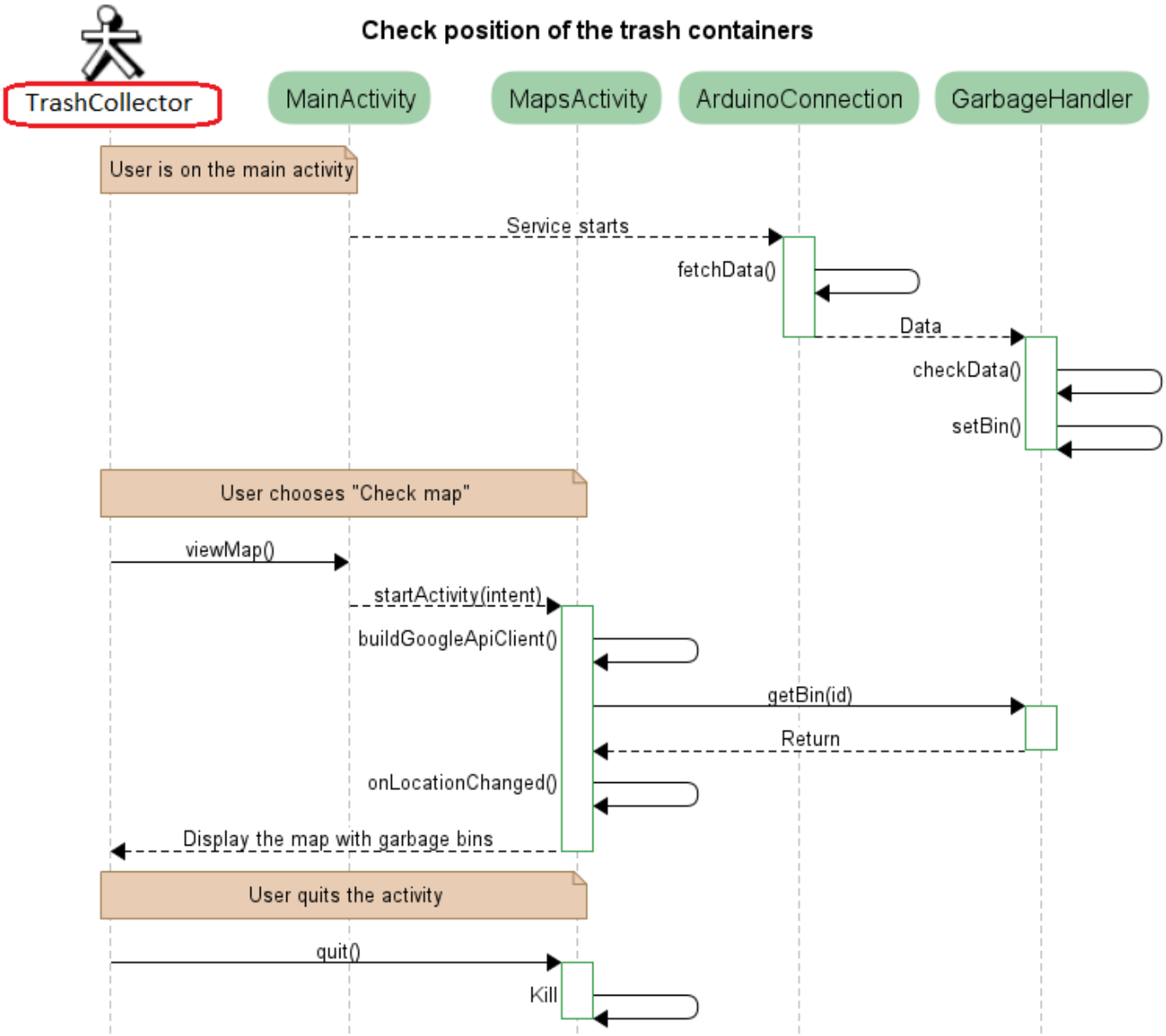
5.3.1.1.3 Use case 3: Check position of the trash containers

This use case describes how the smartphone shows the user, the location of all garbage containers on the map using different markers.

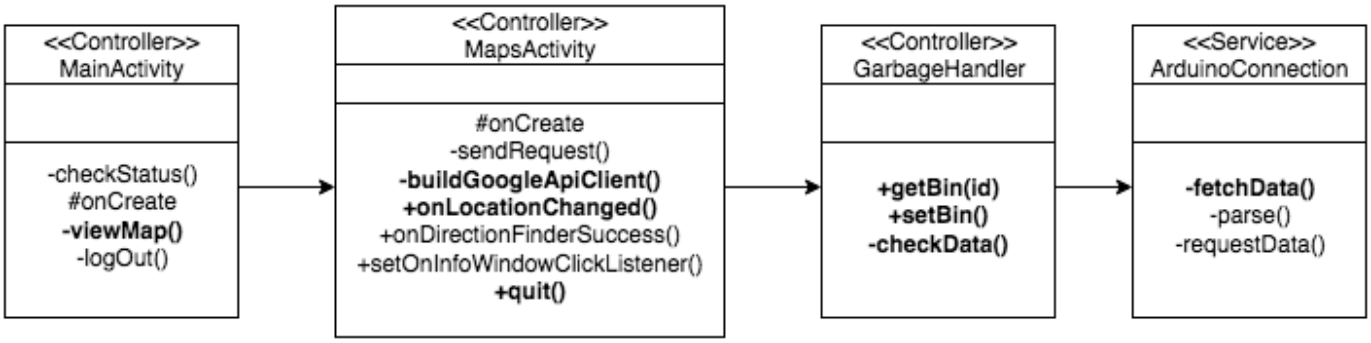
Use case 3 – Involved classes

MapsActivity:

The controller-class check position is the connection between the user and the smartphone. That allows the user to see on the map the location of all the garbage containers.



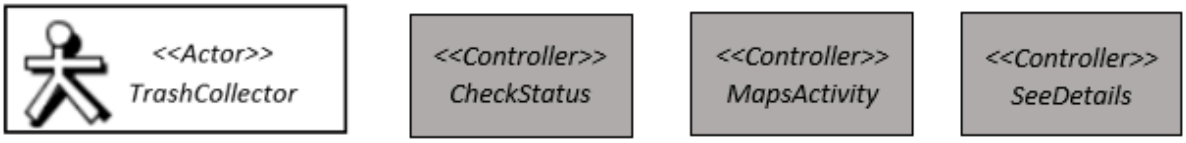
Check Position Sequence Diagram



Check Position Class Diagram

5.3.1.1.4 Use case 4: See details of a specific garbage container

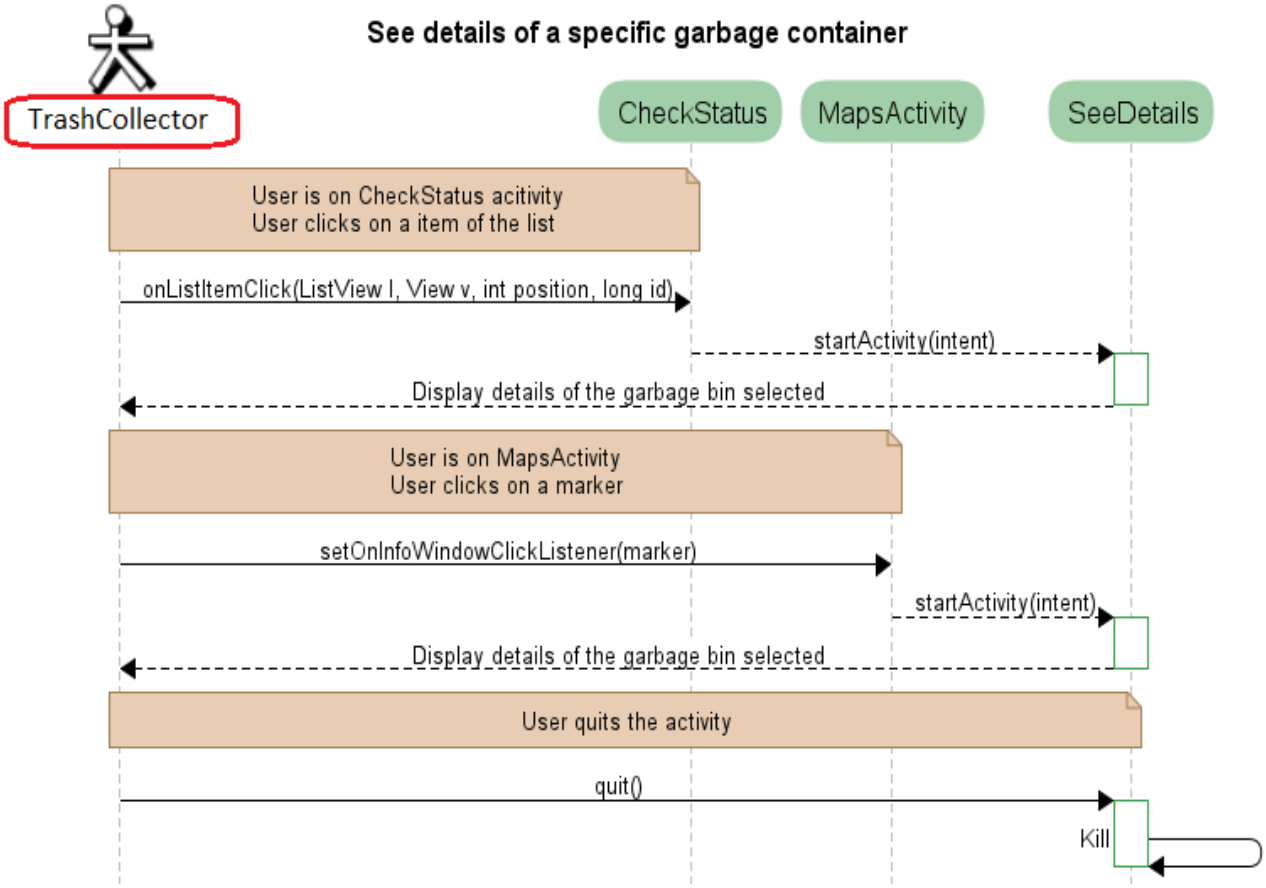
This use cases describes how the user can see a description of a specific garbage bin with many details. The App has three different states to show the information of the capacity of every garbage container. The first state is green and it contains any garbage container between 0% and 30% of full. The second state is yellow and it's for the containers between 30% and 70% of their capacity. The last state for the ones that are between 70% and 100% of their capacity.



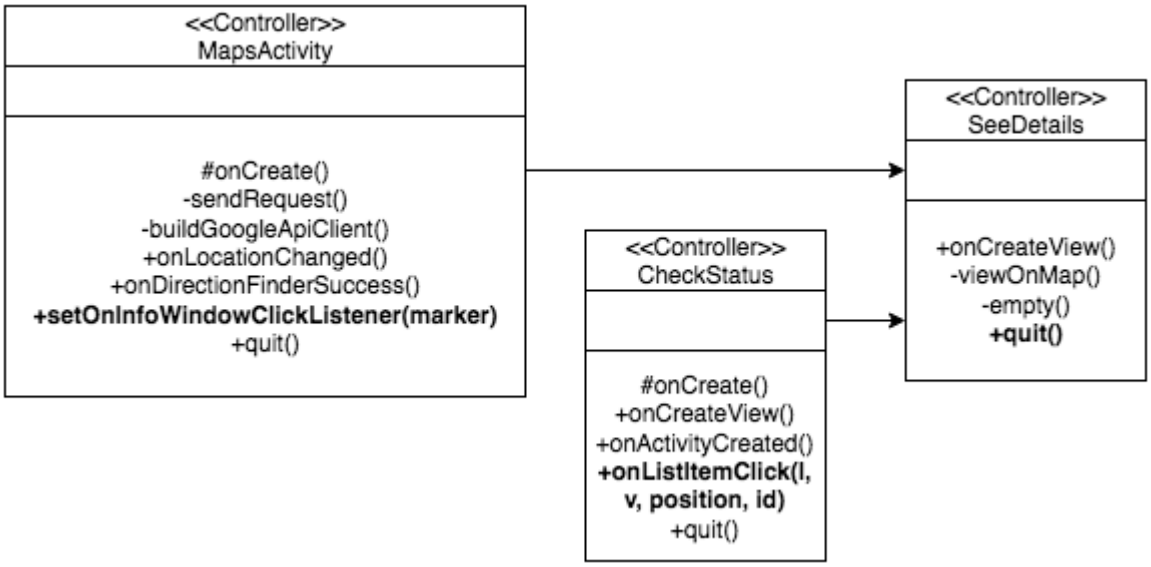
Use case 4 – Involved classes

SeeDetails:

The controller-class see details is the connection between the user and the smartphone. That allows the user to see specific details of the container selected.



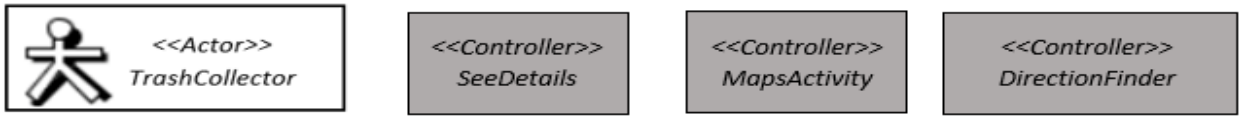
See Details Sequence Diagram



See Details Class Diagram

5.3.1.1.5 Use case 5: Find the best garbage collection route

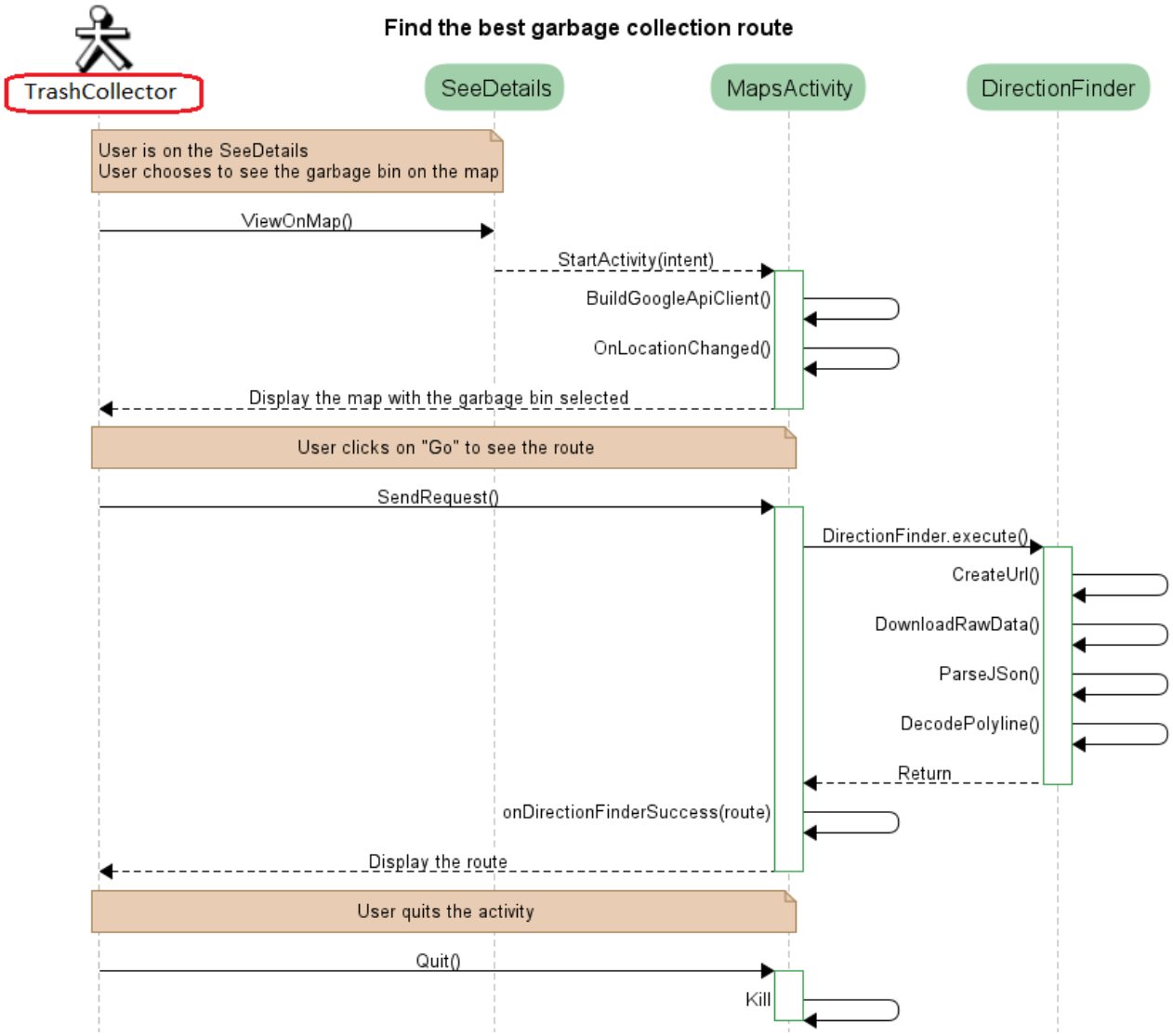
This use case describes how the App shows to user or garbage collector the best route to collect the garbage.



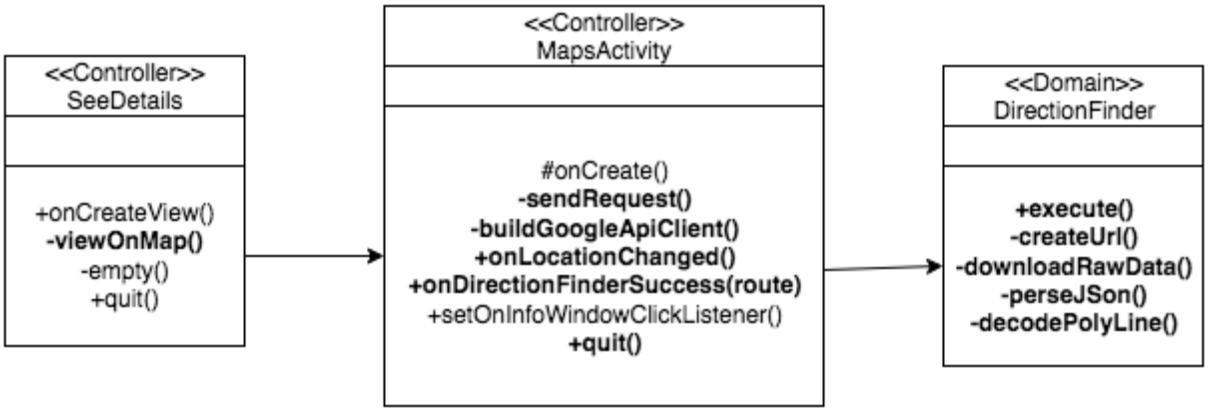
Use case 5 – Involved classes

DirectionFinder:

The domain-class Directionfinder is the class that allows to find the route and display a polyline thanks to an url that we have to download and decode.



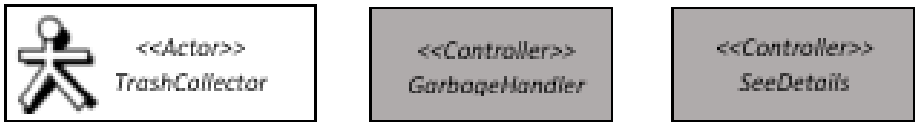
Collection Route Sequence Diagram



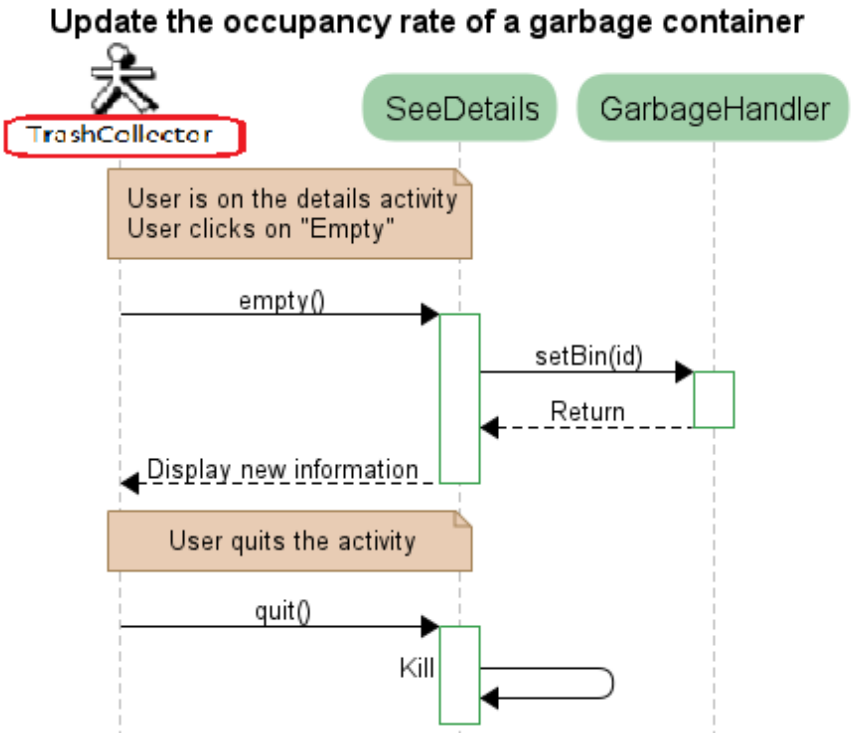
Collection Route Class Diagram

5.3.1.1.6 Use case 6: Update the occupancy rate of a garbage container

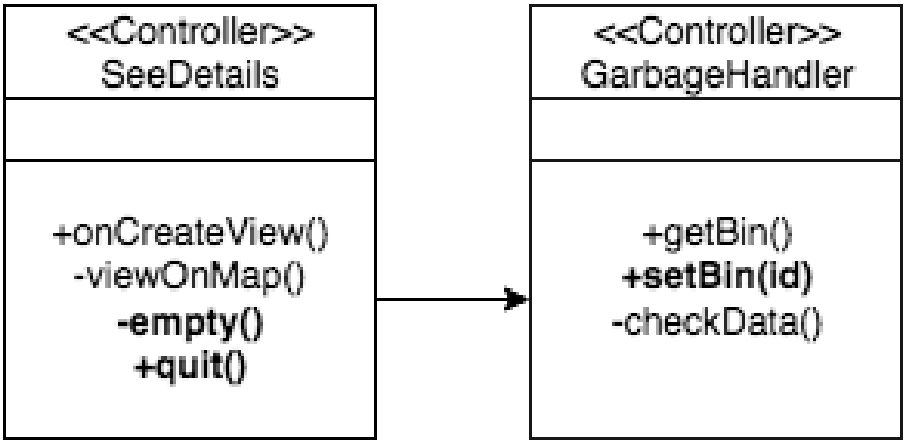
This use case describes how the App shows the user the last time the bin has been emptied and also he can click on a button to say that it's been collected in order to update the date of the last time. This helps to have an idea of how long it takes to full each bin.



Use case 6 – Involved classes

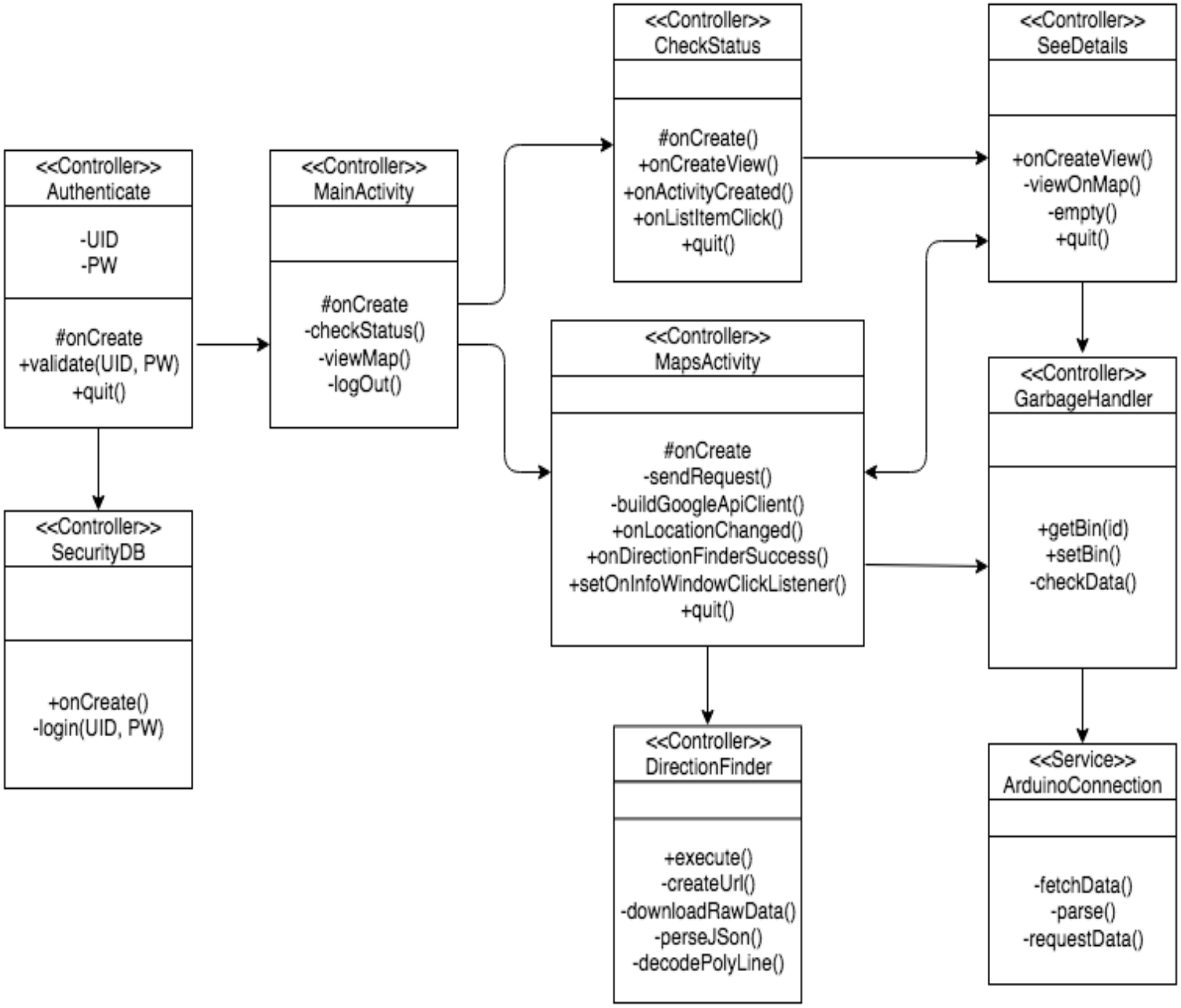


Update occupancy Sequence Diagram

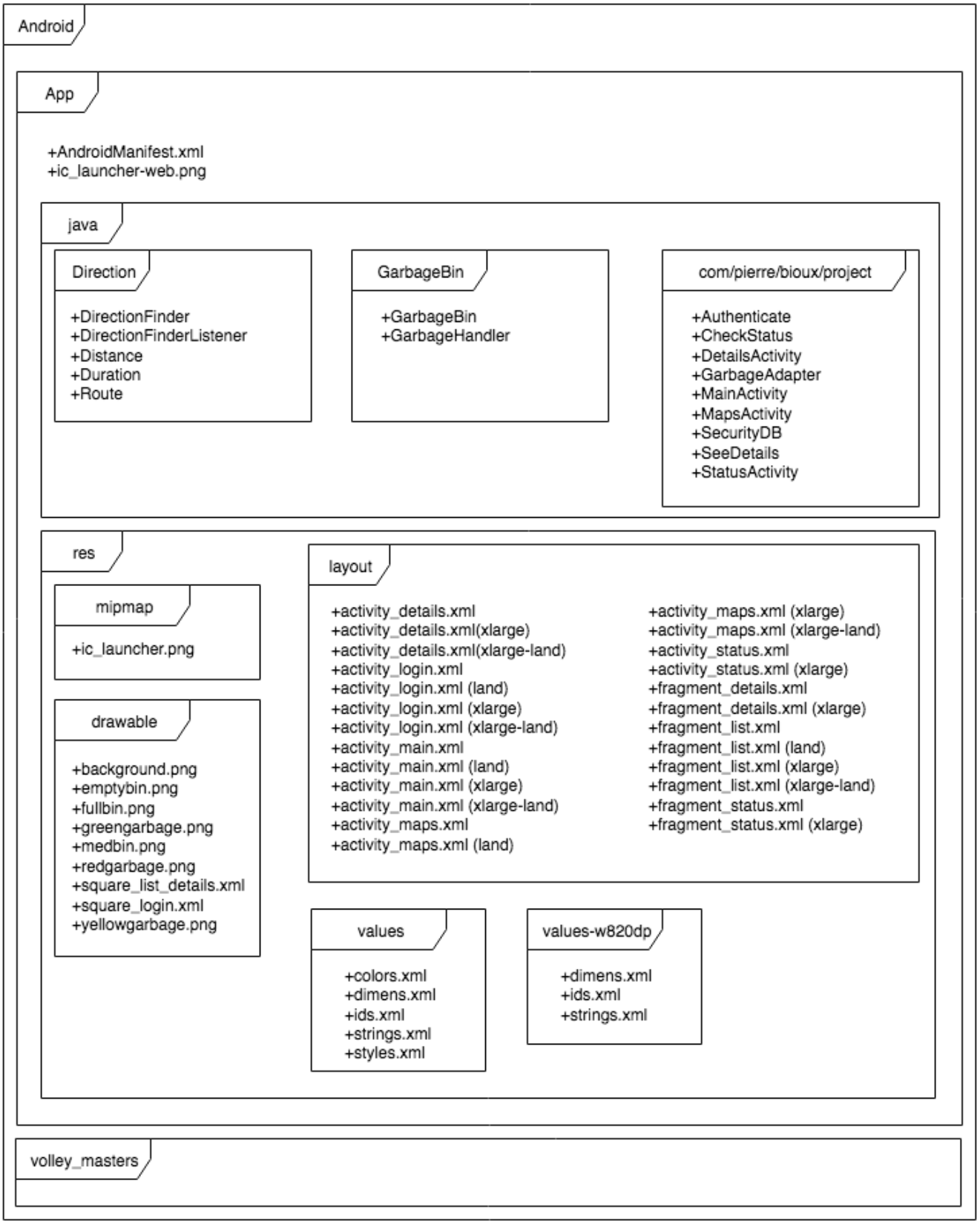


Update occupancy Class Diagram

5.3.1.2 Static Class Diagrams for Phone



Entire Class Diagram for Phone

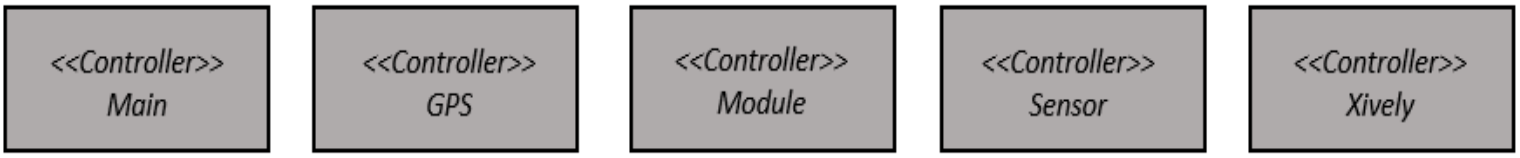


Mobile Application Package Diagram for Phone

5.3.2 Garbage container perspective

System inside garbage bin only needs to serve data on request from its point of view. System is going to be used as a webserver. Upon request sensor will be checked and location added to the final data structure.

All of this is done in the background each time that you go in the main activity. This service is useful for the use case #2, #3 and #4 because we need the range of bins and locations for these. We will go into more detail on this to see methods and the C-Class.



Involved classes

Main:

The controller-class main is responsible of the setup and the loop where everything is lunched.

GPS:

The controller-class GPS is the interface that collects the data about the location of the garbage bin.

Module:

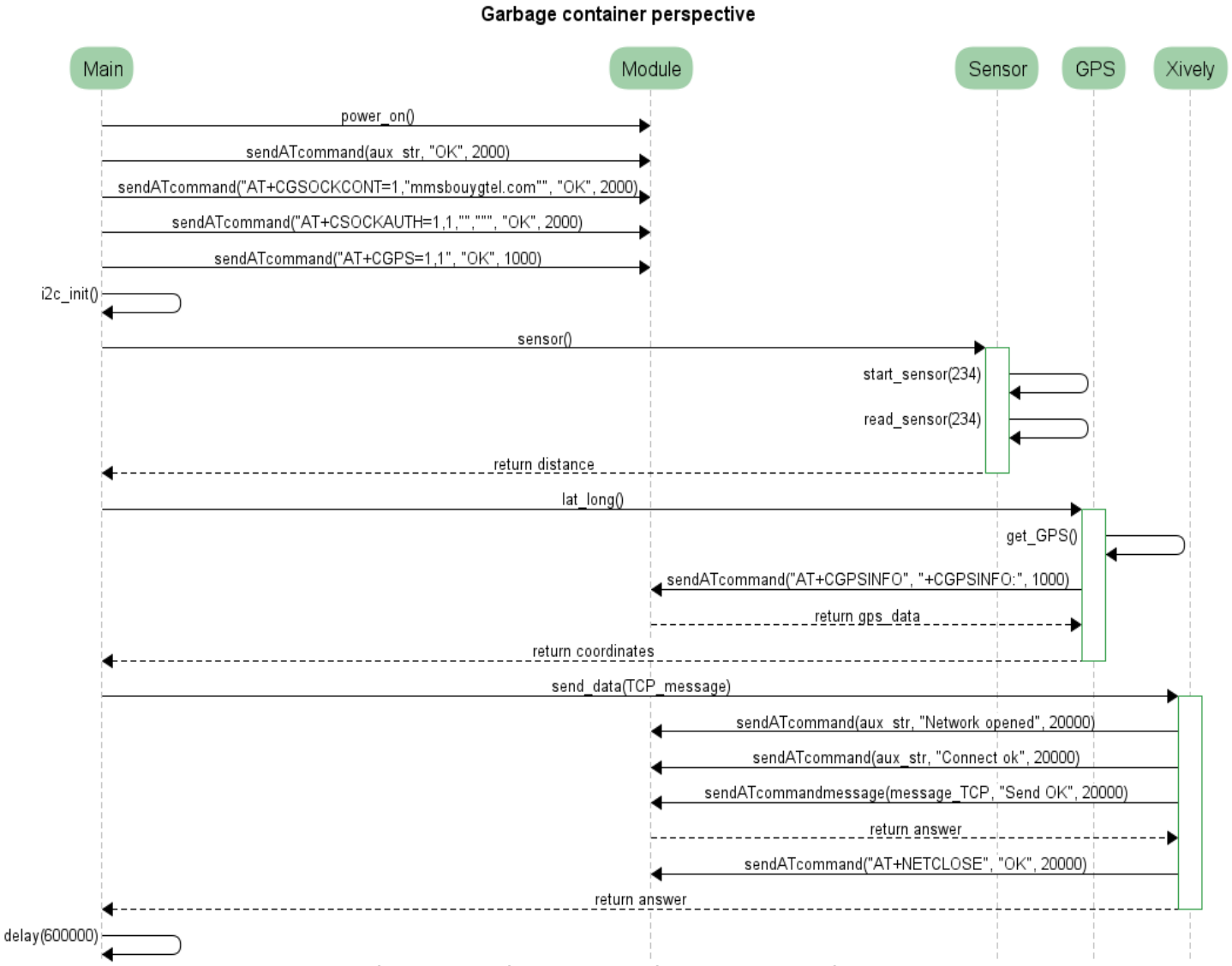
The controller-class module is the class which send command to the 3G Shield and start the module.

Sensor:

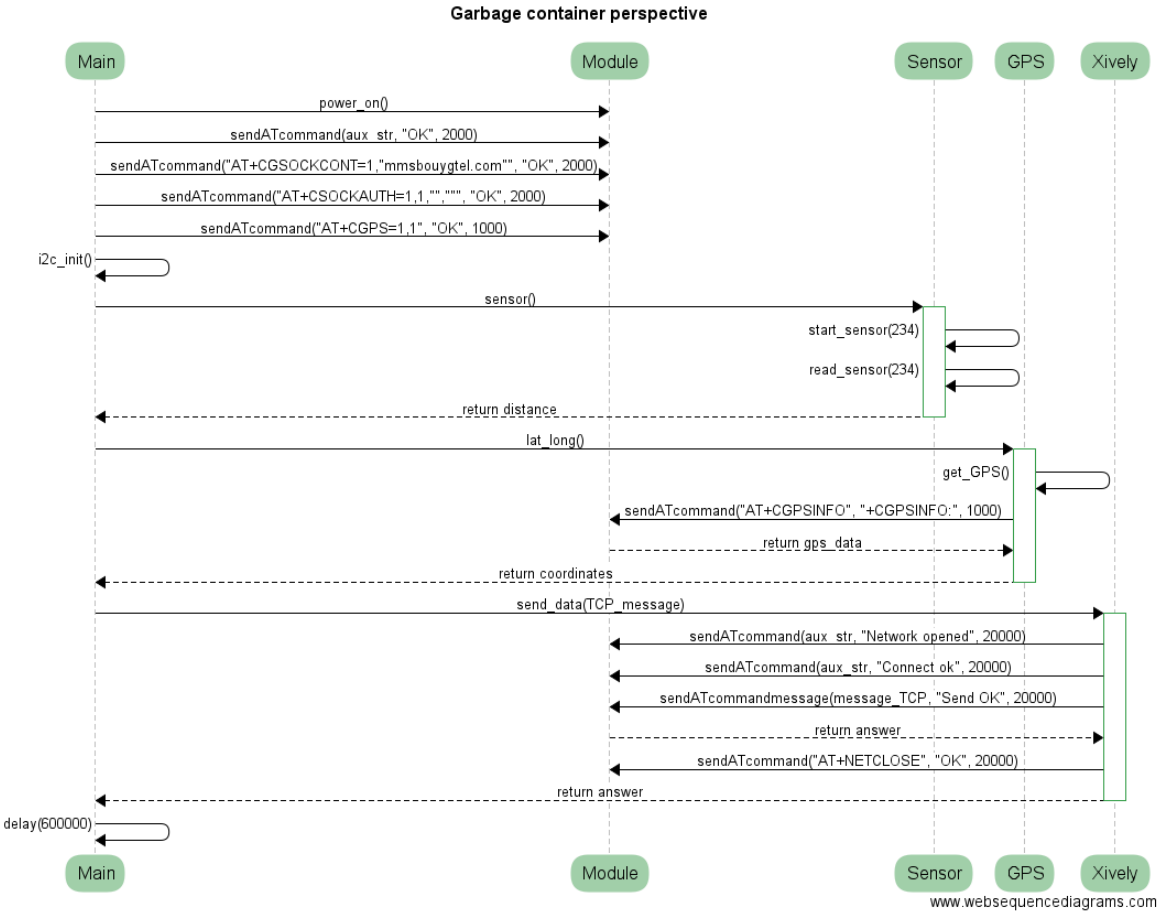
The controller-class sensor is the interface that collects the data about the distance.

Xively:

The controller-class Xively is the class which send data to the personal server.



Garbage Container Perspective Sequence Diagram



Garbage Container Perspective Class Diagram (C arduino - CODE)

5.4 Process view

Following section describes interaction between software components of the system. In this section, we'll describe how components interact with each other and which communication channels they need. We also discuss communication with third party actors.

5.4.1 Smartphone application

To proceed making process view we need to split project into two larger software components. First one to be inspected is smartphone application. Application has multiple task it need to accomplish and figure X gives us insight of this.

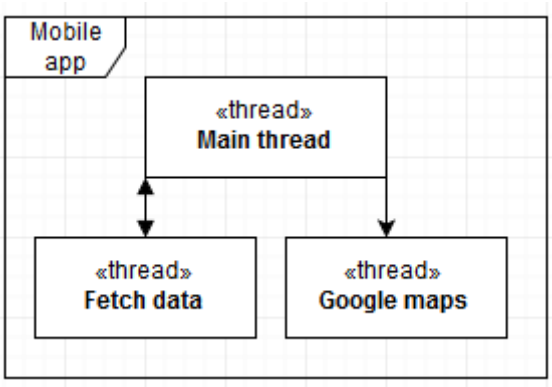


Figure 1 Smartphone application

Applications main thread handles UI elements and transition between activities. Fetch data thread is asynchronous task that runs in the background fetching data from sensor and weather API. Google maps is a thread that is passed coordinates by main activity and implements navigation.

5.4.2 Garbage container

Second large software component is our system inside garbage bin. Systems purpose is to measure and pass the data of sensor. CPU of the system is not capable of multi-threading so instead of threads we have to use interrupts. Figure X describes system inside garbage bin.

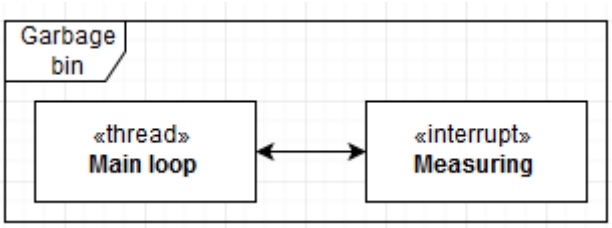


Figure 2 System inside garbage bin

Due to our microcontrollers looping main structure we are going to use main loop to host web server which serves data on requests, which are coming from smartphone. For measuring we are going to use timer interrupts. Timer interrupts makes our application scalable if we ever want to extend the scope of the program with trend lines for example.

5.4.3 Overall view

As we briefly touched overall view of the project in figure X we specify communication link between mobile application and garbage bin system.

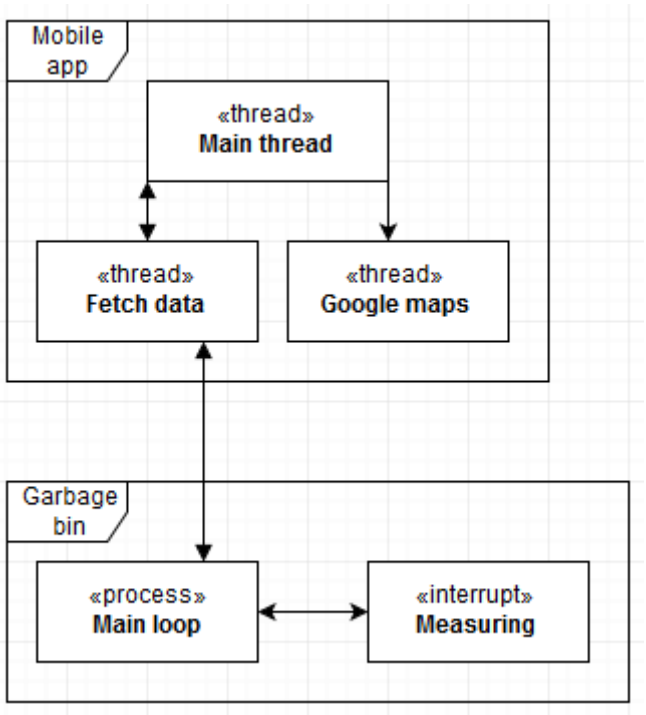


Figure 3 Overall system view

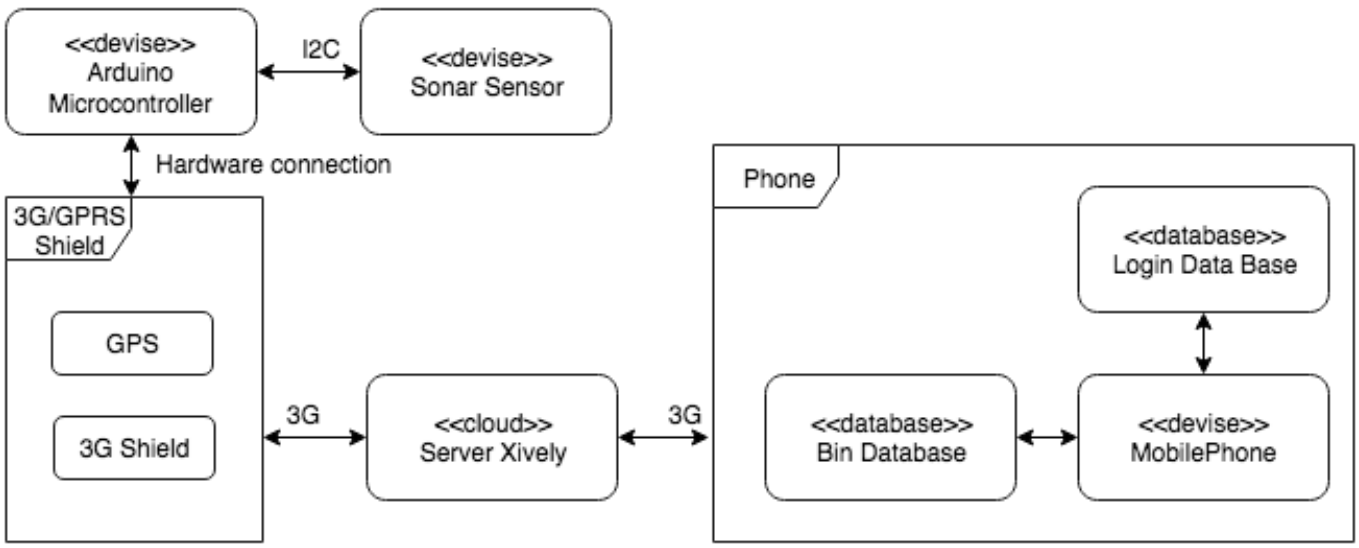
System communicates with TCP connection from smartphone application to system inside garbage bin which is hosting GSM server. Authentication to web server is done in smartphone application due to data storing limitations.

In our system there are two third party actors we communicate with, Google Maps and GPS-satellite. Communication to google maps is done within smartphone application passing it coordinates of garbage bins as an argument.

Second third party actor is GPS satellite that is connected during boot of our garbage bin system. GPS satellite response time might vary and checking it constantly might cause our system to be unstable. Project has possibility of extending GPS tracking to measurement interrupt but for our application garbage bin is assumed not to switch locations between measurements.

5.5 Deployment view

The deployment view is relatively straightforward, as the interactions between our components are, as we can see in this diagram, without extensive interrelation.



Deployment Diagram

As the diagram suggest, there are two main parts for deployment, which are the embedded system (sensor + µcontroller) and the cloud/user part (database & mobile app). Those two can be deployed separately.

We don't have complex runtime dependencies or environments, since the project is a prototype developed only one machine.

5.5.1 3G protocol

The 3G protocol is the way of communication between the µController, the server and the APP. The data is stored in a cloud server called Xively and we are using TCP to send and request the information about distance and coordinates.

The distance is requested to the sensor and it measure through sonar waves and sent the data using I2C communication.

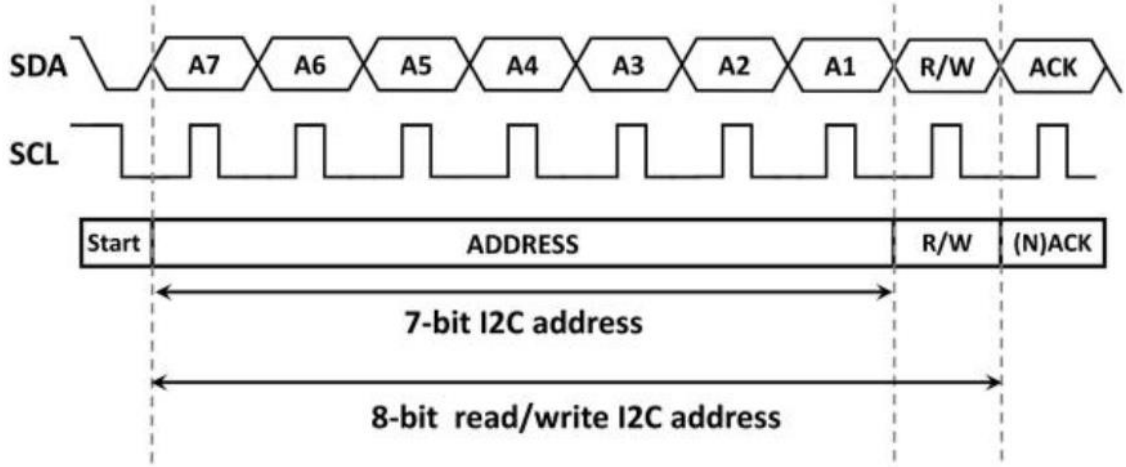
The GPS data is requested with the command AT+CGPSINFO, a special instruction of the cooking hacks command, the module bring directly latitude, longitude, date, UTC time, altitude and speed. Once we have the data, we just take the coordinates (latitude and longitude).

To write in the server, we use a TCP_message containing all the information about distance and coordinates and we send it to the server using the command AT+TCPWRITE.

5.5.2 I2C protocol

The sonar sensor that we are using for our prototype uses the I2C protocol to trespass data. This method is characterized because it only uses a few wires to do the communication. The sensor sends the data when it’s requested by writing a one in the address pin.

As we can see in the figure below, this type of protocol, has two main lines connected from master to slave. On one hand, we have the SDA that acts as a slave and is the responsible of trespassing the data bit by bit like a chain, and on the other hand, there is the SCL, the clock signal that allows the synchronization of the tasks.



This sensor supports I2C clock frequencies up to 400kHz provided clock stretching is supported by the master device. Without clock the sensor can run at speeds up to 50kHz.

Our sensor, sends two bytes to pass the distance measured: MSB and LSB. It has a resolution of 8 bits, that it's enough for our application.

As we get the two bytes separately, in order to put it together as a single variable, we have to made a simply bit operation and finally convert it to centimetres.

5.6 Security view

In the following section, we will describe measures to protect data integrity, availability and confidentiality. Threats and countermeasures will be handled case by case instead linking them to use cases. Linking threats to use cases may leave critical vulnerabilities undiscovered via unintended routes of use.

5.6.1 Integrity

In our project data that can be modified is stored inside a database. Database is handled by database handler inside smartphone application thus limiting attack vector to modify data. Each input that affects database is sanitized and validated. To limit physical modifying of data and circuit our system should be enclosed and locked inside garbage bin.

5.6.2 Availability

Systems inside garbage bin are out of scope to monitor and thus susceptible to denial of service attacks. Measures to protect data integrity also apply here since enclosed system is not as vulnerable to physical violence as disclosed system.

For signal jamming our system is currently unable to do anything. Possibility to avoid jamming is to make another communication channel. For our systems scope this

seems unnecessary. System that is not working properly would be noticed quickly and inspected.

5.6.3 Confidentiality

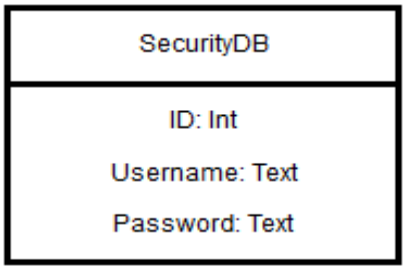
Our systems data confidentiality is handled inside smartphone application. For user to see status and location of garbage bin they need to authenticate. Currently authentication data is stored within the application which is bad practice. To use best practice and scalability we could use centralized authentication server with two factor authentication, for example password and identification card.

5.7 Data view

In our project, we can find two separate databases. To store it, we use a SQLiteDatabase.

First, we have the SecurityDB which stores user data to access to the app.
The attributes are:

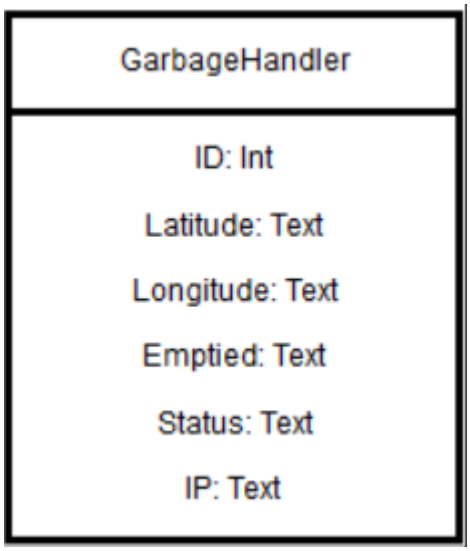
- ID: Integer which is auto incremented when a new content is added.
- Username: Text to identify the user
- Password: Text which is linked to the username to access to the app.



Then, we have the GarbageHandler which stores data about trashes specifics.
The attributes are:

- ID: Integer which is auto incremented when a new content is added.
- Latitude: Text which is the value of the latitude's bin thanks to the GPS.
- Longitude: Text which is the value of the longitude's bin thanks to the GPS.
- Emptied: Text which indicates the last time that the bin has been emptied.
- Status: Text which indicates the range of the bin

- IP: Text with a unique value



The microcontroller stores data in a memory and it sends it to a personal server (Xively) through the 3G shield. To get the data in the smartphone, we use JSON and Volley. It is an HTTP library that makes networking for Android apps easier and faster. Like this, the phone should be able to update Database.
Microcontroller only gives the status and coordinates.

Handling data will be straightforward, as the user database would be updated on user creation, and trash database would be updated on trash creation and edited on status change.

5.8 Development view

The application will have the organization of a standard android project, following the framework structure. Moreover, the application part is described in detailed earlier. Thus, we will mainly describe here the different modules and interactions.

Arduino files are planned to be as follow:

-Sensor reader and data handling

-Communication protocol (TCP)

-Data exchange to database

-Connection module for the phone

Critical setups are as following:

The process should first be able to read the sensor, then be able to communicate with 3G the data, and then the rest of the development can follow up independently.

No notable external tools or out of the ordinary math expressions are used at the moment.

Code snippets:
For the sensor, we picked an adapted one, thus no conversion or other alteration code was needed. Yet we still need a way to handle the collected data by the sensor:

```
String sensor() {

  Serial.println(F("\n********************* GET DISTANCE ********************"));
  Serial.println("Starting measure: ");
  boolean error = 0; //Create a bit to check for catch errors as needed.
  int range;
  error = start_sensor(234); //Start the sensor and collect any error codes.
  if (!error) { //If you had an error starting the sensor there is little point in reading it as you will get old data.

    //Do it 4 times to be sure of the mesure with a delay of 1.5s
    for (int i = 0; i < 4 ; i++) {
      delay(1500); //Take a measure every 1500 ms
      range = read_sensor(234); //Reading the sensor will return an integer value -- if this value is 0 there was an error
    }

    Serial.print("Distance: ");
    Serial.print(range);
    Serial.print("cm");

    if (range <= 60) {
      Serial.println("\tFull");
    }
    if ((60 < range) && (range <= 90)) {
      Serial.println("\tAlmost Full");
    }
    if (range > 90) {
      Serial.println("\tEmpty");
    }
  }
  String distance = String(range);
  Serial.println(F("***************** GET DISTANCE : DONE ****************"));
  return distance;
}
```

Sensor data handling code

For the communication protocol, here is an extract to send message to Xively (personal server):

```
void send_data(String TCP_message) {
  Serial.println(F("\n******************** SEND MESSAGE ********************"));
  Serial.println("Start send data");
  char message_TCP [500];
  TCP_message.toCharArray(message_TCP, 500); // converting TCP_message to char inorder for it to pass through AT command

  sprintf(aux_str, "AT+NETOPEN=\"TCP\",%s", port);
  Serial.println("Opening network");
  answer = sendATcommand(aux_str, "Network opened", 20000);

  if (answer == 1)
  {
    Serial.println("Network opened");
    sprintf(aux_str, "AT+TCPCONNECT=\"%s\",%s", server, port);
    Serial.println("Opening socket");
    answer = sendATcommand(aux_str, "Connect ok", 20000);
    if (answer == 1)
    {
      Serial.println("Socket opened");

      sprintf(aux_str, "AT+TCPWRITE=%d", strlen(message_TCP));

      answer = sendATcommand(aux_str, ">", 20000);
      if (answer == 1)
      {
        sendATcommandmessage(message_TCP, "Send OK", 20000);
        Serial.println("Message send");
      }

      sendATcommand("AT+NETCLOSE", "OK", 20000);
      Serial.println("Network closed");
      Serial.println(F("***************** SEND MESSAGE : DONE *****************"));
```

Communication server code

Here is as snippet to get the GPS info:

```
String get_GPS() {
  Serial.println(F("\n********** GET GPS COORDINATES ***************"));
  Serial.println("Get coordinates:");
  Serial.println("Start loop GPS");
  //Do it until get coordinates
  do {
    answer = sendATcommand("AT+CGPSINFO", "+CGPSINFO:", 1000);  // request info from GPS
    Serial.println("In the loop GPS");
    if (answer == 1)
    {

      counter = 0;
      do {
        while (Serial.available() == 0);
        gps_data[counter] = Serial.read();
        counter++;
      }
      while (gps_data[counter - 1] != '\r');
      gps_data[counter] = '\0';
      if (gps_data[0] == ',')
      {
        Serial.println("No GPS data available");
      }
      else
      {
        Serial.print("GPS data:");
        Serial.println(gps_data);
      }
    }
    else
    {
      Serial.println("Error");
    }

    delay(5000);
  } while (gps_data[0] == ',');

  Serial.println(F("************** GET GPS COORDINATES : DONE *************"));
  return gps_data;
}
```

GPS code