GitBerto:
The automated robot guide

Luca Landolfi   Luca Marchegiani

July 9, 2022

Abstract

In this paper we propose a small project for the *Mobile Systems* course. The project basically consists of a small robot that is piloted by a mobile device: the user sets a destination and then follows the robot that takes him to the desired location.

We underline that this work is only a very simplified description of the main operation of the system that can be extended in future developments. In this work we only develop e very simple system that shows the basic idea of this project without covering all the aspects of a real *sealable* product, but only those related with the contents of the *Mobile Systems* course with some elements derived by the course of *Ingegneria dei Sistemi Software*.

Contents

1 Introduction and fast Requirement Analysis

Since this is a small project with the aim to experiment something in *Mobile Systems*, we never provide a detailed project with exhaustive analysis. However, we will give some important details by making a fast analysis.

1.1 Main user story

This *user story* shows the main description of the operation of the system:

User Story 1.1: Main User Story

The user:

1. uses his device and **opens the `GitBerto` application** on his mobile that **automatically connects with the paired physical robot**;

2. inserts a destination using the classical *address searching* and the application shows the founded possibilities; then, **selects the desired target** to arrive to and the application calculates the route to get the destination;

3. **clicks on the *GO*** button;

4. **follows the physical `GitBerto` robot that started to move** guiding him towards the destination.
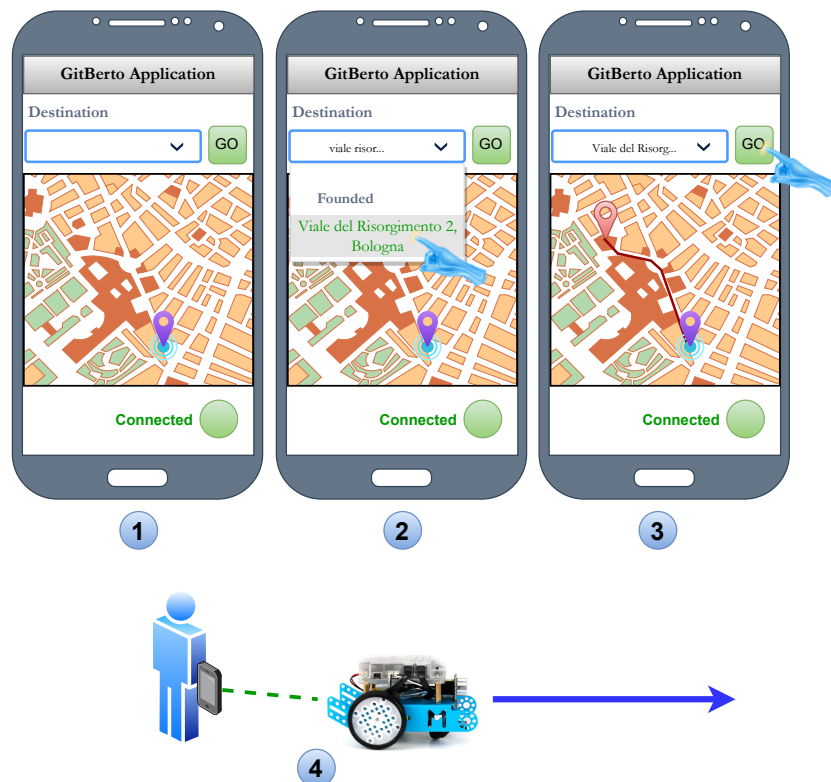

Figure 1: Main user story representation

The figure 1 shows a graphical representation of the steps presented in the main user

story.

1.2 Main overview of the architecture of the system

In order to simplify the work, we also make some **assumptions**:

1. angle between routes are only right (90°);

2. routes are all passable only by pedestrians;

3. all others pedestrian shows the robot and avoid collisions with him;

4. the position retrieved by the mobile device is exact without error.

These assumptions let us concentrate with the main focus of this work without caring about others aspects that are mainly related with Artificial Intelligence

As suggested by the assumption 3, as a requirement **all the computation of positions and routes must be performed by the user device**. So, as expected, **the mobile device and the robot must communicate**, exchanging commands and data.
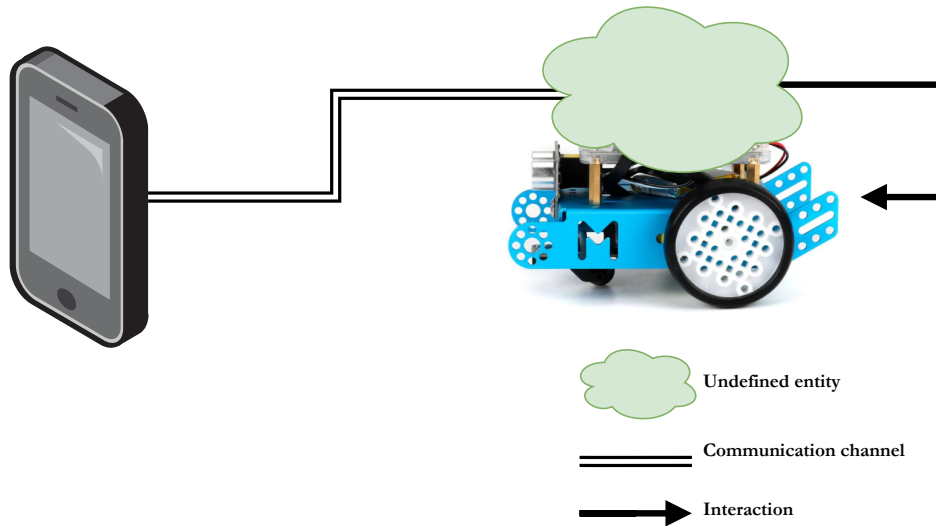

Figure 2: *Coarse* architecture of the system

The first *coarse* architecture is shown by the figure 2 that only say that:

1. there are two *undefined entities* that are *executable*: one on the device and the other on the robot;

2. this two entities has a *communication channel* that can be used for **communication**;

3. the entity on the robot can *interact* with the physical device.

About the communication, since one of the node is **mobile**, we must choose a mechanism that is supported by this type of node. So, we have restricted possibilities:

- **Bluetooth** that is fully supported from mobile devices (at least from both `iOS` and `Android`);

- **Wi-Fi Direct** that is supported too from the main mobile devices;

- **Internet protocol suite over Wi-Fi**, also supported from all devices.

Because the nodes are mobile, we exclude the set of *Internet protocol suite*. Indeed, this type of communication needs a router in order to be established. With the modern devices that now all people have, communication over Internet protocol suite is not because user might use his phone as a Wi-Fi Hotspost, but this implies possible additional costs and resource consumption (especially for battery).

About the mobile devices, is useless to say that the main options are two: `iOS` and `Android`. Since in the course of *Mobile Systems* we have studied `Android`, in this project we build the system considering only it but leaving the possibility to use the competitor in others development.

Finally, about the robot we will consider a physical robot controlled by `Arduino` or Raspberry. We give two main options, but the only requirement is that **the controller of the robot has to support the chosen communication mechanism**. The options we suggest are:

- **mBot**, a robot controlled by an Arduino board;

- **Nano Robot**, intended as a robot kit (like *alphabot*) with a Raspberry Pi mounted as controller.

1.3 Code availability

In the previous *fast requirement analysis* we analysed the devices and the available communication mechanisms. Now, we want to give a fast overview of the code already available. We only consider `Android` for the mobile device.

- about **communication**, Android provides a large set of components and documentation for `Bluetooth` and Wi-Fi Direct; we have also found some Bluetooth libraries that also work for Linux, like `PyBluez` with `Bluez` but no supports for `Wi-Fi Direct`;

- about the **robot** we have some complete supports developed by the professor Antonio Natali for the course of *Ingegneria dei Sistemi Software*, in particular we have the `BasicRobot` system.

We underline that `BasicRobot` is written by using the `QA-System` (also known as `QAK`) that is an infrastructure for **actor meta-modeling and programming**. We have full access to use this infrastructure.

2 Fast problem analysis

First, we have to resolve the main problem that comes from the requirement analysis: we said that the robot has a controller board such as `Arduino` or `Raspberry` and that it must communicate with `Android`.

Notice that even if the robot is a `mBot`, we can easily mount a `Raspberry` over `Arduino`, so we can use all the framework and the technologies that works with `Linux`. Then **we consider that the robot is controlled by a `Raspberry` that is well configured**.

> **Since we have the `BasicRobot` already developed and working, we decide to use it in order to build a very simple demo of the system with a `mBot` with a `Raspberry` installed over `Arduino`.**

2.1 Connection between `Android` and `Arduino`

As shown in the fast requirement analysis, from the `Android` point of view we have large set of instruments to develop using Bluetooth or Wi-Fi Direct, but we can't say the same thing from the `Raspberry` (Linux) point of view.

> **We choose to use Bluetooth for communication because we have found *poor* support for `Wi-Fi Direct` and also because we think that Bluetooth can consume less.**

Even if there are a working `Python` implementation of Bluetooth for Linux, we decided to make a fast porting of it because the `QAK` is written in `Kotlin`.

The entire ported library, called `KBluez` is fully accessible at this `GitHub` repository. Basically, there is a `Python` script (`pybluezwrapper.py`) launched by the `Kotlin` library as a system process using `Java Runtime`: `stdin`, `stdout` and `stderr` of the `Python` script are captured and handled by `Kotlin` that is able to send commands to `PyBluez` and read response by using a proper protocol.

Indeed, `pybluezwrapper.py` uses the main thread for reading commands from `stdin` that are sent by `Kotlin` as json strings. When the `Python` script receives the command to open a *Bluetooth Socket*, so `pybluezwrapper.py` launches another thread and entrust the new socket to it associating an unique identifier that `Kotlin` uses to refer to. **In this way, concurrency is safely guaranteed**, and it's possible to open multiple sockets without blocking the `Python` script.

The figure 3 shows a general overview of the architecture of our `KBluez`.

The figure 4 shows the class diagram of the `BluetoothSocket` interface that is the main key point of our library. Other classes will not be shown at the moment. This interface is implemented by PyBluezSocket class but the implementation will not be exposed in deep.

At the operational point of view, the developer can obtain a `BluetoothSocket` with RFCOMMby invoking:

```
val btSock = KBLUEZ.requestNewSocket(BluetoothServiceProtocol.RFCOMM)
```

Then, this socket can be used exactly as the internet protocol sockets in order to build a client or a server.

> **We have used `BluetoothSocket` implementation in order to create a bluetooth server on `Raspberry` that make it able to communicate with the device of the user. No library or third-party components are needed in `Android`.**
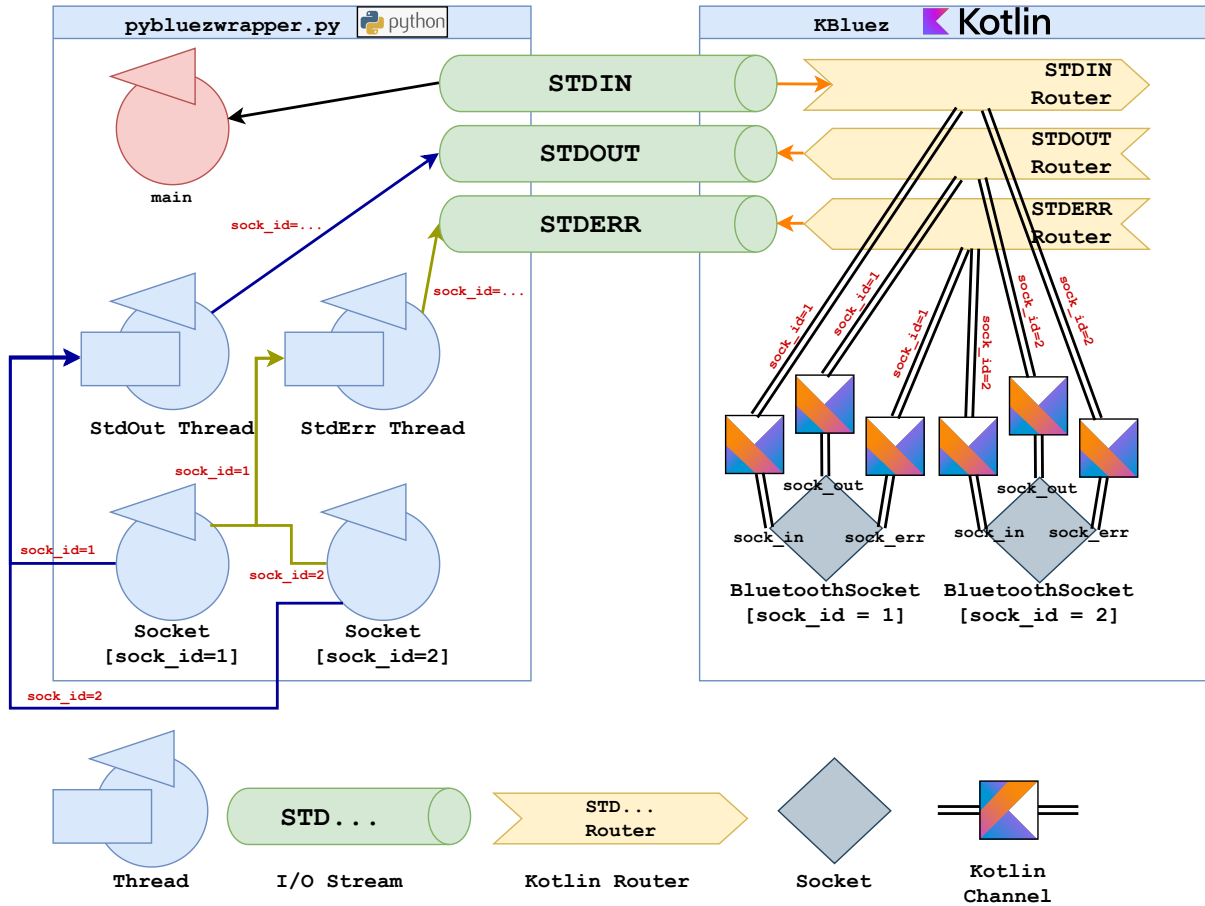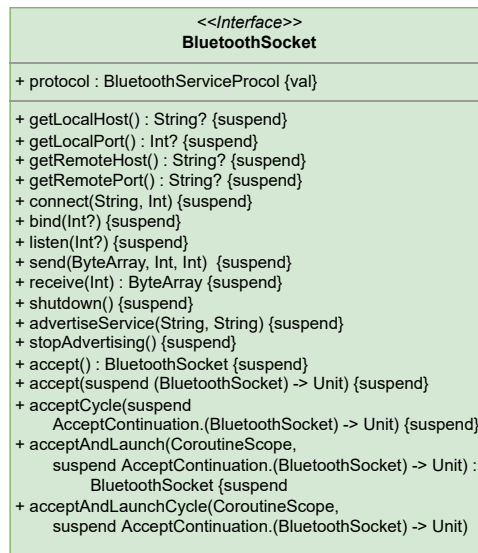
Figure 3: General architecture of the KBluez library

Figure 4: Class diagram of it.unibo.kBluez.socket.BluetoothSocket

2.2 Actor modeling

In the *Mobile System* course we study the fundamentals of Android developing. We deeply analysed all components offered to the Android developer such as Activity, Services, Intent, Broadcast Received, AsyncTask and so on and so forth...

Now, we want to merge all of these concepts with the already mentioned `actor meta-modeling` and programming. For these purposes, as anticipated we have the `QA-System` that is well documented at the `Github repository` of the corse of *Ingegneria dei Sistemi Software*.

Just to give a fast overview, **an actor is an active entity that receives messages thanks to a channel that it owns**. So, when an actor receives a message, it performs some action depending on the contents. The `QAK` system make easy to define and handle actors that are mapped into `Kotlin` coroutine