DPS Project 2023

Distributed and Pervasive Systems Lab Course

April 2023

1 Project description

In a smart city named Greenfield, a fleet of robots moves around the districts of the city to clean their streets. Figure 1 shows how Greenfield is organized.

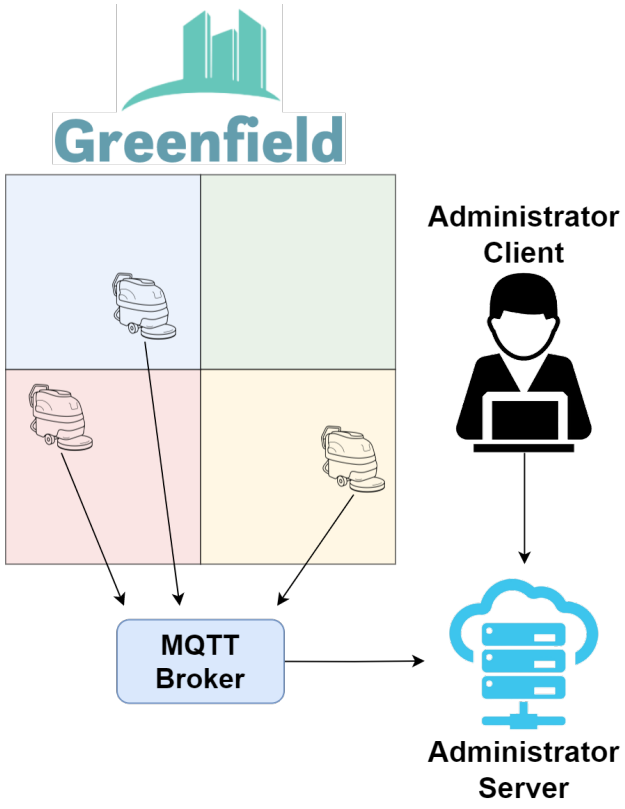

Figura 1: Overall architecture of Greenfield

The cleaning robots move around the four smart city districts. Occasionally, such robots need to go for maintenance issues to the mechanic of Greenfield, which can handle only a single robot at a time. Each robot is also equipped with a sensor that periodically detects the air pollution levels of Greenfield.

Such pollution measurements are periodically transmitted from the robots of each district to an Administrator Server through MQTT. The Administrator Server is in charge of dynamically registering and removing cleaning robots from the system. Moreover, it collects and analyses the air pollution levels of Greenfield in order to provide pollution information to the experts (Administrator Client) of the environmental department of Greenfield.

The goal of the project is to implement the Administrator Server, the Administrator Client, and a peer-to-peer system of cleaning robots that periodically send pollution measurements to the Administrator Server through MQTT, and autonomously organize themselves through gRPC when they concurrently need to go to the mechanic of the smart city.

1.1 Greenfield internal representation

The smart city of Greenfield is represented as a 10×10 grid (see Figure 2), divided into four 5×5 districts. Each cell of the grid represents a square

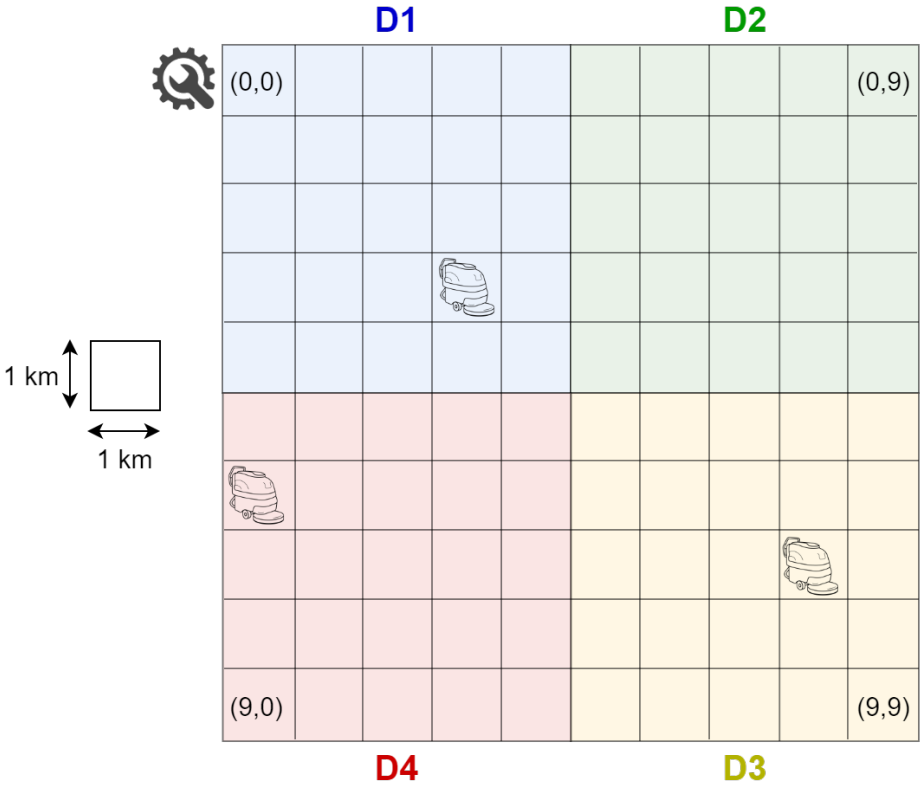

Figura 2: Greenfield representation

kilometer of Greenfield. When a cleaning robot asks the Administrator

Server to be registered in the smart city, it will be placed in a cell of one of the districts.

2 Applications to be implemented

For this project, you are required to develop the following applications:

- *MQTT Broker*: the *MQTT* broker on which the cleaning robots publish air pollution measurements
- *Cleaning Robot*: a cleaning robot of Greenfield
- *Administrator Server*: REST server that dynamically adds/removes cleaning robots from the network and computes statistics about the air pollution measurements received by subscribing to the MQTT Broker
- *Administrator Client*: a client that queries the *Administrator Server* to obtain information about the statistics of the air pollution levels of Greenfield

Please, note that each *Cleaning Robot* is a stand-alone **process** and, so, it **must not** be implemented as a thread.

In the following, we provide more details about the applications that must be developed.

3 MQTT Broker

The *MQTT Broker* on which Greenfield relies is online at the following address: `tcp://localhost:1883`.

The cleaning robots of Greenfield use this broker to periodically communicate the air pollution measurements to the Administrator Server. As will be described in the next section, each robot publishes such measurements to the MQTT topic dedicated to the district in which the robot operates. The Administrator Server subscribes to all the topics to receive the air pollution measurements from each district.

4 Cleaning robot

Each Cleaning robot is simulated through a **process**, which is responsible for:

- periodically sending the air pollution measurements it collected to the Administrator Server through **MQTT**
- coordinating with the other robots by using **gRPC** to distributively decide which robot is allowed to reach the mechanic of Greenfield for maintenance issues

4.1 Initialization

A cleaning robot is initialized by specifying

- ID
- listening port for communications with the other robots
- Administrator Server's address

Once it is launched, the cleaning robot process must register itself to the system through the Administrator Server. If its insertion is successful (i.e., there are no other robots with the same ID), the cleaning robot receives from the Administrator Server:

- its starting position in one of the smart city districts
- the list of the other robots already present in Greenfield (i.e., ID, address, and port number of each robot)

Once the cleaning robot receives this information, it starts acquiring data from its pollution sensor. Then, if there are other robots in Greenfield, the cleaning robot presents itself to the other ones by sending them its position in the grid. Finally, the cleaning robot connects as a publisher to the *MQTT* topic of its district.

4.2 Sending air pollution levels to the server

Every 15 seconds, each cleaning robot has to communicate to the Administrator Server the list of the averages of the air pollution measurements computed after the last communication with the server. This list of averages must be sent to the server associated with

- The ID of the cleaning robot
- The timestamp in which the list was computed

As already anticipated, the communication of the air pollution measurements must be handled through MQTT. In particular, a cleaning robot that operates in the district i will publish such data on the following *MQTT* topic:

`greenfield/pollution/district{i}`

4.3 Pollution sensors

Each cleaning robot is equipped with a sensor that periodically detects the air pollution level of Greenfield. Each pollution sensor periodically produces measurements of the level of fine particles in the air (PM10). Every single measurement is characterized by:

- PM10 value
- Timestamp of the measurement, expressed in milliseconds

The generation of such measurements is produced by a simulator. In order to simplify the project implementation, it is possible to download the code of the simulator directly from the page of the course on *Moodle*, under the section *Projects*. Each simulator assigns the number of seconds after midnight as the timestamp associated with a measurement. The code of the simulator must be added as a package to the project, and it **must not** be modified. During the initialization step, each cleaning robot launches the simulator thread that will generate the measurements for the air pollution sensor.

Each simulator is a thread that consists of an infinite loop that periodically generates (with a pre-defined frequency) the simulated measurements. Such measurements are added to a proper data structure. We only provide the interface (*Buffer*) of this data structure that exposes two methods:

- $void\ add(Measurement\ m)$
- $List <Measurement>\ readAllAndClean()$

Thus, it is necessary to create a class that implements this interface. Note that each cleaning robot is equipped with a single sensor.

The simulation thread uses the method $addMeasurement$ to fill the data structure. Instead, the method $readAllAndClean$, must be used to obtain the measurements stored in the data structure. At the end of a read operation, $readAllAndClean$ makes room for new measurements in the buffer. Specifically, you must process sensor data through the *sliding window* technique that was introduced in the theory lessons. You must consider a buffer of 8 measurements, with an overlap factor of 50%. When the dimension of the buffer is equal to 8 measurements, you must compute the average of these 8 measurements. A cleaning robot will periodically send these averages to the Administrator Server (as explained in Section 4.2).

4.4 Handling maintenance issues through mutual exclusion

Every 10 seconds, each cleaning robot has a chance of 10% to be subject to malfunctions. In this case, the cleaning robot must go to the mechanic of Greenfield (for simplicity, it is not necessary to simulate that the robot actually reaches the mechanic in the smart city grid). The mechanic may be accessed only by a single cleaning robot at a time. It is also possible to explicitly ask a cleaning robot to go to the mechanic through a specific command (i.e., *fix*) on the command line. In both cases, you have to implement one of the *distributed* algorithms of *mutual exclusion* introduced in the theory lessons in order to coordinate the maintenance operations of the robots

of Greenfield. You have to handle critical issues like the insertion/removal of a robot in the smart city during the execution of the mutual exclusion algorithm.

For the sake of simplicity, you can assume that the clocks of the robots are properly synchronized and that the timestamps of their requests will never be the same (like Lamport total order can ensure). Note that, all the communications between the robots must be handled through *gRPC*.

The maintenance operation is simulated through a `Thread.sleep()` of 10 seconds.

4.5 Leaving Greenfield

Cleaning robots can terminate in a controlled way. Specifically, only when the message *"quit"* is inserted into the command line of a robot process, it will leave the system. At the same time, you must handle also those cases in which a robot unexpectedly leaves the system (e.g., for a crash simulated by stopping the robot process).

When a robot wants to leave the system in a controlled way, it must follow the next steps:

- complete any operation at the mechanic
- notify the other robots of Greenfield
- request the *Administrator Server* to leave Greenfield

When a robot unexpectedly leaves the system, the other robots must have a mechanism that allows them to detect this event in order to inform the Administrator Server.

5 Administrator server

The Administrator Server collects the IDs of the cleaning robots registered to the system and also receives from them (through MQTT) the air pollution levels of Greenfield. This information will be then queried by the administrators of the system (Administrator Client). Thus, this server has to provide different REST interfaces for:

- managing the robot network
- enabling the administrators to execute queries on the air pollution levels

5.1 Statistics

The Administrator Server has to collect through MQTT the air pollution measurements sent by the cleaning robots of Greenfield. More specifically, the Administrator Server assumes the role of the subscriber for the following four *MQTT* topics:

- `greenfield/pollution/district1`
- `greenfield/pollution/district2`
- `greenfield/pollution/district3`
- `greenfield/pollution/district4`

The air pollution measurements have to be stored in proper data structures that will be used to perform subsequent analyses.

5.2 Robots REST interface

5.2.1 Insertion

The server has to store the following information for each robot joining Greenfield:

- ID
- IP address (i.e., localhost)
- The port number on which it is available to handle communications with the other robots

Moreover, the server is in charge of assigning to each joining robot a random position in one of the districts of Greenfield (positions in Greenfield are expressed as the Cartesian coordinates of a smart city's grid cell). Note that, there can be more robots in the same grid cell of the smart city. The district must be chosen so that the cleaning robots are uniformly distributed among the districts. For instance, if there are 2 robots in District 1, 1 robot in Districts 2 and 3, and no robots in District 4, the next robot joining Greenfield should be placed in District 4. A robot can be added to the network only if there are no other robots with the same identifier. If the insertion succeeds, the Administrator Server returns to the cleaning robot

- the starting position in Greenfield of the robot
- the list of robots already located in the smart city, specifying for each of them the related ID, the IP address, and the port number for communication

5.2.2 Removal

Whenever a cleaning robot asks the Administrator Server to leave the system, the server has to remove it from the data structure representing the smart city. Similarly, when one of the robots informs the Administrator Server that a certain robot left the system in an uncontrolled way (e.g., for a crash), the server has to remove such a robot from its internal data structure.

5.3 Administrator Client REST interface

When requested by the Administrator Client through the interface described in Section 6, the Administrator Server must be able to compute the following statistics:

- The list of the cleaning robots currently located in Greenfield
- The average of the last n air pollution levels sent to the server by a given robot
- The average of the air pollution levels sent by all the robots to the server and occurred from timestamps *t1* and *t2*

6 Administrator Client

The Administrator Client consists of a simple command-line interface that enables interacting with the *REST* interface provided by the Administrator Server. Hence, this application prints a straightforward menu to select one of the services offered by the administrator server described in Section 5.3 (e.g., the list of the smart city robots), and to enter possible required parameters.

7 Simplifications and restrictions

It is important to recall that the scope of this project is to prove the ability to design and build distributed and pervasive applications. Therefore, all the aspects that are not strictly related to the communication protocol, concurrency, and sensory data management are secondary. Moreover, it is possible to assume that no nodes behave maliciously. On the contrary, you should handle possible errors in data entered by the user. Furthermore, the code must be robust: all possible exceptions must be handled correctly.

Although the Java libraries provide multiple classes for handling concurrency situations, for educational purposes, it is mandatory to **use only the methods and classes explained during the laboratory course.** Therefore, any necessary synchronization data structures (such as lock, semaphores, or shared buffers) should be implemented from scratch and will be

discussed during the project presentation. Considering the communication between the robots' processes, it is necessary to use the *gRPC* framework. If broadcast communications are required among robots, these must be carried out in parallel and not sequentially.

8 Project presentation

You must develop the project individually. During the evaluation of the project, we will ask you to discuss some parts of the source code and we will also check if it runs correctly considering some tests. Moreover, we will ask you one or more theoretical questions about the theoretical content of the lab lessons. You will run your code on your machine, while the source code will be discussed on the tutor's machine. You must submit the source code before discussing the project. For the submission, you should archive your code in a *.zip* file, renamed with your university code (i.e., the "matricola"). For instance, if your university code is 760936, the file should be named 760936.zip. Then, the zip file should be uploaded at *http://upload.di.unimi.it*. It will be possible to submit your project a week before the exam date. You must complete the submission (strictly!) two days before the exam date at 11:59 PM (e.g., if the exam is on the morning of the 15th, you must complete the delivery before 11:59 PM of the 13th).

We suggest the students to use during their project discussion the `Thread.sleep()` instructions to show how the synchronization problems have been correctly handled. We recommend analyzing all the synchronization issues and creating schemes to illustrate how the components of your project communicate. These schemes may represent the format of the messages and the sequence of the communications that occur among the components during the different operations involved in the project. Such schemes can be very helpful during the project discussion. It is not necessary to show the structure of the classes of your project, but only the main aspects regarding the synchronization and the communication protocols you have implemented.

During the presentation of the project, we will ask you to test your code by launching at least 4 or 5 robots, the Administrator Server, and a single Administrator Client.

9 Plagiarism

The reuse of code written by other students will not be tolerated. In such a case, we will apply the sanctions described course website, in the section *Student Plagiarism* of *General Information*.

The code developed for the project **must not** be published (e.g., GitHub) at least until March of the next year.

10 Optional parts

In order to encourage the presentation of the project in the first exams sessions, the development of the first optional part becomes mandatory from September's session (included), while both the first and the second optional parts are mandatory from January's session (included).

10.1 First part

For this optional part, when one or more robots detect that a certain robot left Greenfield in an uncontrolled way, all the remaining robots in the smart city must organize themselves through gRPC so that the number of robots in each district is balanced. For instance, if there are 2 robots in District 1, 1 robot in Districts 2 and 3, and no robots in District 4, one of the robots of District 1 should reach District 4. This operation must be implemented in a distributed way, without the help of the Administrator Server.

When a robot changes its district, it has also to change its subscription to the MQTT topic of its new district.

10.2 Second part

In a distributed system, it is very likely that the clocks of the different devices are not synchronized. For this optional part, the assumption that the clocks are synchronized is no longer valid. Thus, you must implement the *Lamport* algorithm to solve this problem when robots must coordinate to decide which robot will be allowed to reach the mechanic of Greenfield. The timestamps of the messages that the robots use to implement the distributed algorithm you chose cannot be generated only through the method `System.currentTimeMillis()`. During the initialization steps, each cleaning robot must also generate a random offset that is added to every timestamp to simulate the mismatch between the clocks of the different robots.

11 Updates

If needed, the current text of the project will be updated. Changes in the text will be highlighted. Please, regularly check if new versions of the project have been published on *Moodle*.