UNIVERSITÀ DI PISA

Daniele Giaquinta

Internet of Things Project

Smart Parking Application

A.Y. 2022/2023 AIDE

Contents

1 Smart Parking Introduction

In today's interconnected world, the Internet of Things (IoT) has opened up new possibilities for optimizing and improving various aspects of our daily lives. One domain that has greatly benefited from IoT advancements is smart parking systems. This project focuses on developing an innovative IoT solution specifically designed to enhance parking management by integrating a set of intelligent sensors and actuators.

The project incorporates four key components to create a comprehensive smart parking system: an occupancy sensor, a distance and proximity sensor, an actuator for the automatic barrier gate, and a display for parking slot occupancy information; all of the behaviors were simulated and the actual sensors used are Nordic rF562840.

The occupancy sensor plays a crucial role in monitoring the availability of parking spaces in real-time. By accurately detecting whether a parking slot is occupied or vacant, it provides valuable data for efficient parking management. This information can be utilized to guide drivers towards available spaces, optimize parking allocation, and reduce congestion within the parking area.

The distance and proximity sensor complements the occupancy sensor by monitoring the proximity of vehicles to the automatic barrier gate. It detects when a car is approaching the gate, allowing for seamless and automated access control. This ensures a smooth flow of vehicles and enhances the overall user experience by eliminating the need for manual intervention.

To actively control the barrier gate, an actuator is integrated into the system. The actuator receives instructions from the central control unit based on the data collected by the sensors. It responds by opening or closing the barrier gate accordingly, providing secure and convenient access to authorized vehicles. To provide real-time feedback to drivers, a display unit is incorporated into the smart parking system. The display showcases the occupancy status of each parking slot, indicating whether it is available or occupied. This information assists drivers in quickly identifying available spaces, reducing the time spent searching for parking and improving overall efficiency.

By leveraging the power of IoT technologies and intelligent integration of sensors and actuators, this smart parking system enhances parking management, optimizes space utilization, and improves the overall user experience. With real-time data, automated access control, and clear occupancy information, drivers can enjoy a seamless parking experience, while parking operators benefit from improved efficiency and resource allocation.

2 Structure and Implementation

As anticipated the architecture is composed of 2 sensors and 2 actuators. A java application coordinates the environment and sends commands to the actuators through the CoAP protocol based on the information published by the sensors to the broker exploiting the MQTT protocol.

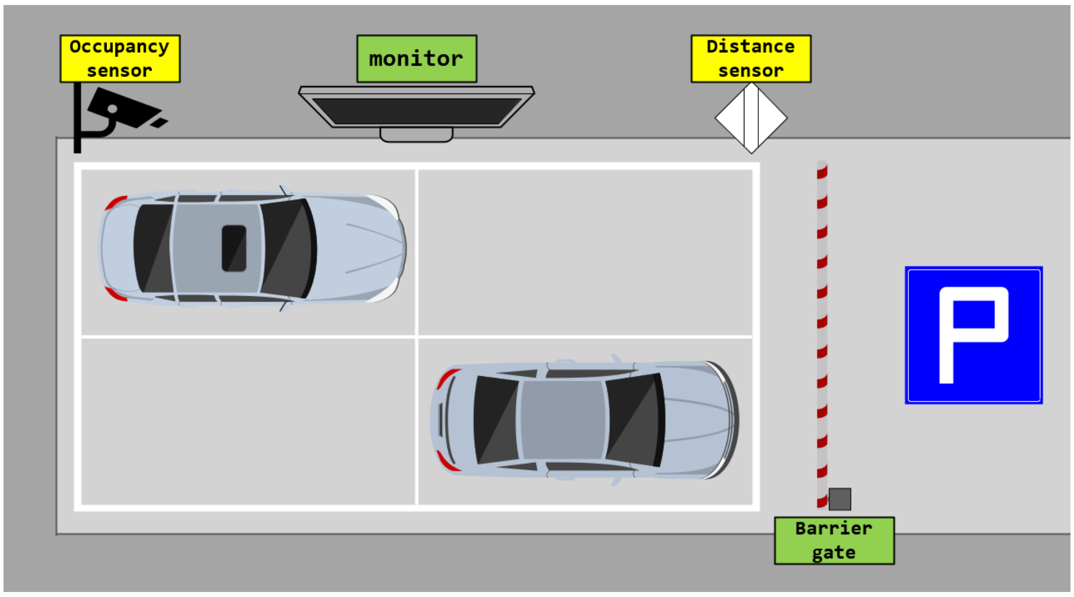


Figure 1: Smart Parking Schema

The elements involved are:

- The **automatic barrier gate actuator** which lifts and lower the barrier gate based on the feedback provided by the **distance sensor**.
- The **monitor actuator** which displays the occupancy of the parking slots based on the feedback provided by the **occupancy sensor**.

2.1 Application

The application is divided into 4 java files:

- **App.java** which is the main class and defines the structures of the user interface;
- **SmartParkingDB.java** which handles the connections and the inserts into the database;
- **CoAPClient.java** which defines the CoAP client and the put requests to the actuators

- **CollectorMqttClient.java** which receives data from the MQTT broker and sends CoAP requsts accordingly.

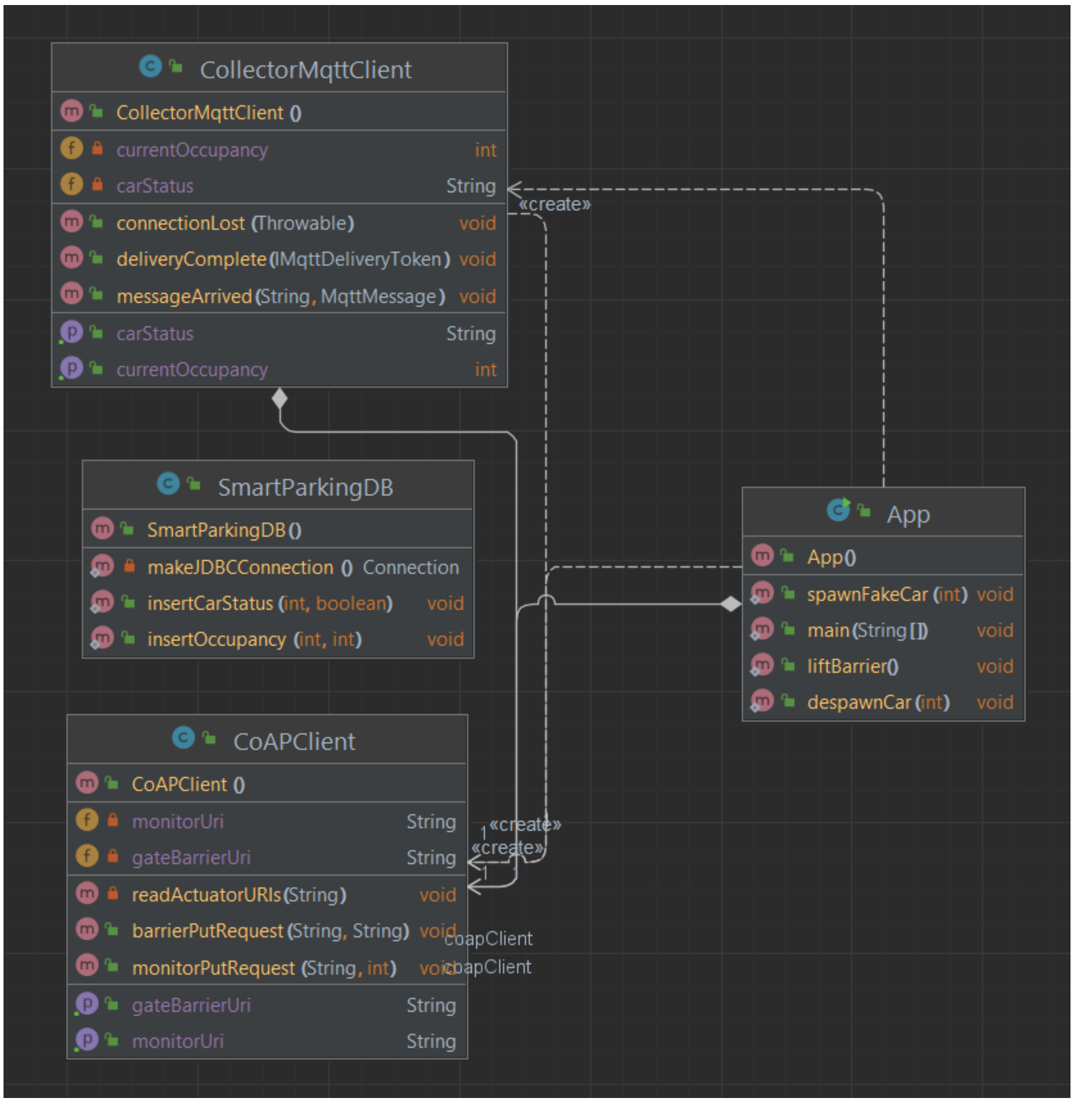


Figure 2: Application UML Class Diagram

The application only sends put requests to the actuators because this type of request is sufficient to perform the required actions on the resources (which are: lower or raise the barrier gate and update the occupancy information on the monitor).

The actuators are not registered to the application dynamically; the addresses of the actuators are instead stored on a textual configuration file which is read in the CoAPClient class.

2.1.1 Data Encoding

In this resource-constrained sensor network, I decided to use JSON as the preferred data format for transmitting data generated by the sensors to the collector. JSON's simplicity and text-based nature align well with the limited capabilities of the sensor devices. This choice ensures compatibility and ease of implementation within this specific environment (the JSON message format is hard coded), despite other potentially more efficient but less available formats like XML or CBOR.

3 Managing the Data

All of the data sampled by the sensors is gathered by the application through the MQTT broker and before doing anything else, the data is stored into a MySQL database. There are two tables, one for each sensor:

carDetected	**occupancy**
id	id
timestamp	timestamp
node	node
carPresent	occupancy

Table 1: Structure of the database tables

The timestamp provides temporal information, enabling analysis of data using time series views. The id is automatically incremented in each table, allowing for easy access to values in a clear order that is more immediately readable than the timestamp. The node id becomes useful when expanding the application with additional sensors or actuators of the same type. Lastly, the actual information is stored in the last attribute. In the first case, it indicates the presence of a car in front of the automatic barrier gate with a boolean value, while in the occupancy table, it represents the number of cars in the parking.

3.1 Grafana Views

This project exploits Grafana to create customized views of the data; Grafana is an open-source data visualization and monitoring tool used to analyze and display metrics from various data sources. It offers a user-friendly interface that allows users to create customizable dashboards and charts to visualize data in real-time. It supports a wide range of data sources, including databases, time series databases, cloud platforms, and IoT devices. It is widely used in industries such as IT operations, DevOps, and IoT to monitor system performance, track key metrics, and gain insights from data.

For the project four different views have been created: there are two time series which show the evolution of the status of both the occupancy of the parking and the presence or absence of cars in front of the automatic barrier gate; The other two sections in the dashboard show the real-time presence of car in front of the gate and occupancy of the parking slot.

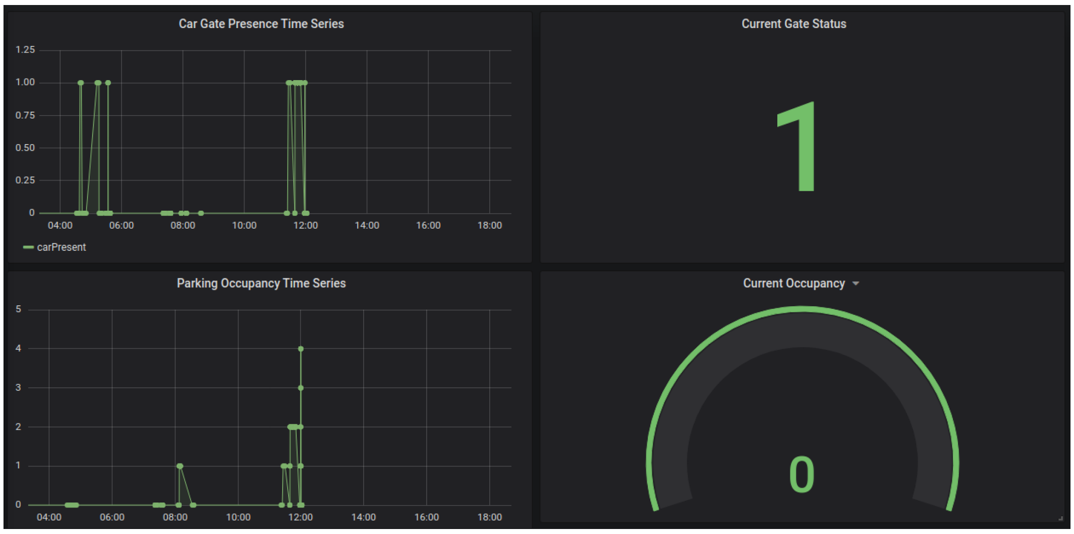
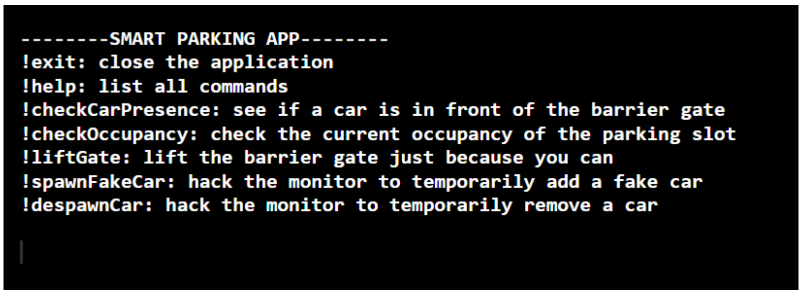


Figure 3: Grafana Dashboard for the project

4 User Interface

A simple user interface has been implemented in order to allow for some limited user interaction:



```
--------SMART PARKING APP--------
!exit: close the application
!help: list all commands
!checkCarPresence: see if a car is in front of the barrier gate
!checkOccupancy: check the current occupancy of the parking slot
!liftGate: lift the barrier gate just because you can
!spawnFakeCar: hack the monitor to temporarily add a fake car
!despawnCar: hack the monitor to temporarily remove a car
```

Figure 4: User Interface

The description of each command is displayed after sending the "!help" command. However, it's important to note that the last three commands, although they send a CoAP put request, do not permanently modify the parking status as monitored by the sensors. When the automatic barrier gate is lifted or lowered, the change will only be temporary and last for 3 seconds. Similarly, when cars are added or removed from the parking, the change will persist until a new status is sampled by the occupancy sensor, which will then accurately reflect the current occupancy status once again.