Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto and Matteo Rossi

20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering 2 – Written Exam 2 (WE2)

September 3rd, 2021

Notes

1. Remember to write your name and Id number (matricola) on the pieces of paper that you hand in.
2. You may use a pencil.
3. Incomprehensible handwriting is equivalent to not providing an answer.
4. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden.
5. The exam is composed of 2 parts, one focusing on requirements, and one focusing on design. Read carefully all points in the text!
6. **Total available time: 1h and 30 mins**

System Description: StreetParkHelper

We want to build an application, *StreetParkHelper*, that helps users find available parking spaces in a city. Each car is equipped with an on-board computer, with its own operating system. The on-board computer runs factory-installed software that is able to interact with the vehicle, and it is also able to run custom software developed by third parties. In particular, the factory-installed software running on the on-board computer allows custom software to interact with the GPS sensor installed on the vehicle, and to retrieve the status of the vehicle (e.g., whether the engine is running or not). The on-board computer also has a touchscreen that can be used to show information to the driver, but also to allow the driver to send inputs to custom applications.

StreetParkHelper periodically collects the information from the car (through the custom software installed on the on-board computer) and determines whether the car is parked or not. If the car has just switched from parking to moving, the system considers that a parking place is available, at least for a few minutes. Since not all cars will be equipped with the necessary electronics or software, information about available parking spaces is necessarily incomplete, but we can consider that a parking place is most likely to be available in the immediate future after a car has just started moving. Also, we do not consider that there are parking limitations in the city (e.g., "blue", "yellow", "pink" parking spaces). The system keeps track of the parking places that it considers available, and it shows them on a map on the display of the car. Through the touchscreen of the on-board computer, a driver can ask for the closest known available parking place, or to be notified when a parking place nearby becomes available. He/she can also inform the system that a parking space that it considered available is now occupied (that can happen if the car that occupies the parking place is not equipped with the necessary HW/SW, so the system is not informed of the change in the occupation status).

Part 1 Requirements (7 points)

RASD_Q1 (2 points)

With reference to the Jackson-Zave distinction between the world and the machine, identify the relevant world and machine phenomena for *StreetParkHelper*, including the shared ones, providing a short description if necessary. For shared phenomena specify whether they are controlled by the world or the machine. Focus in particular on phenomena that are relevant to describe the requirements of the system.

RASD_Q2 (3 points)

Referring to the phenomena identified above, define in natural language one specific goal for *StreetParkHelper*, together with the associated domain assumptions and requirements. Explain why the identified requirements and assumptions are relevant to the fulfillment of the goal.

RASD_Q3-Q4 (2 points)

Define a UML Use Case Diagram describing the relevant actors and use cases for *StreetParkHelper*.
(Optional) Provide a brief description of use cases if the name is not self-explaining.

Part 2 Design (7 points)

DD_Q1 (4 points)

Assuming you need to implement system *StreetParkHelper* analyzed above, identify the most relevant components and interfaces describing them through UML Component or Class Diagrams.

DD_Q2 (3 points)

Provide a brief description of each component identified in DD_Q1 and list the operations provided by the corresponding interfaces.

For each operation, you do not need to precisely specify its parameters; however, you should give each operation a meaningful enough name to understand what it does; you can also briefly describe what information operations use/produce.

Solutions

RASD_Q1

World phenomena:
The car stops moving
The car moves around
The engine stops/starts
A car that is not part of the system occupies a parking space
A car that is not part of the system frees a parking space

Shared phenomena, world-controlled
The user asks the system to see available parking spaces nearby
The user asks the system to be notified of when nearby parking spaces become available
The user informs the system of an available parking space nearby
The user informs the system that a supposedly available parking space is no longer available
The user sets the radius for showing available parking places

Shared phenomena machine-controlled
The system reads the status of the engine
The system reads the GPS position of the car
The system displays the nearby available parking places
The system notifies the user that a parking place nearby has become available

RASD_Q2
Goal
G1: The user knows when there are available parking spaces nearby

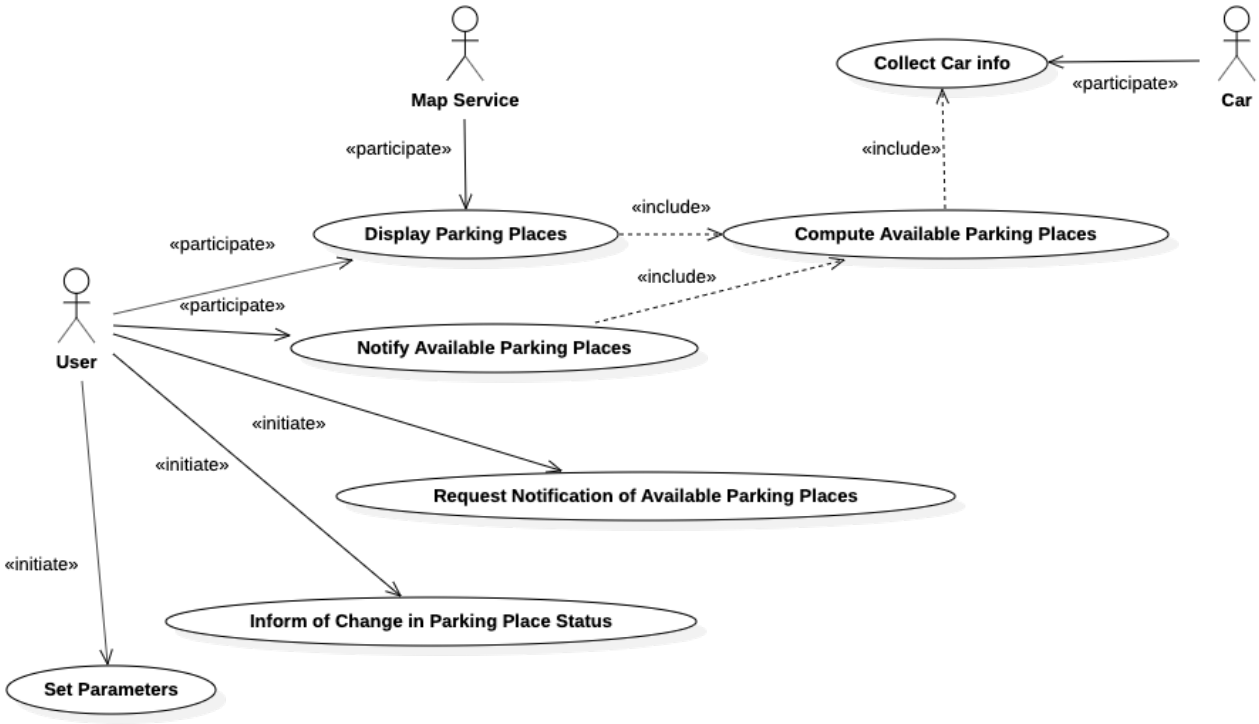
Requirements
R1: The system shall periodically collect the status of the engine of each monitored car
R2: The system shall periodically collect the position of each monitored car
R3: The system shall update the status of a monitored car to "parked" when the car stops moving and the engine is stopped for more than 1 minute
R4: The system shall update the status of a monitored car to "not parked" when the car starts moving after being "parked"
R5: When a car becomes "not parked", the system shall mark the position of the car (before leaving) as an available parking space.
R6: When a car becomes "parked", the system shall mark the position of the car (before leaving) as an available parking space.
R7: X minutes (where X is a configuration parameter of the system) after a parking space becomes available, the system shall remove it from the list of available parking places.
R8: The system shall display one each car a map with the position of the available parking places in a radius (from the current position) set by the user
R9: The system shall allow the user to request to be notified of available nearby parking places
R10: If the user has asked to be notified of available nearby parking places, when a new parking place becomes available in the radius set by the user, the system shall send a notification to the user through the on-board computer
R11: The system shall allow the user to set the radius of interest for available nearby parking places

Domain assumptions
DA1: Sensors (e.g., the engine sensor) provide correct information about the status of the car

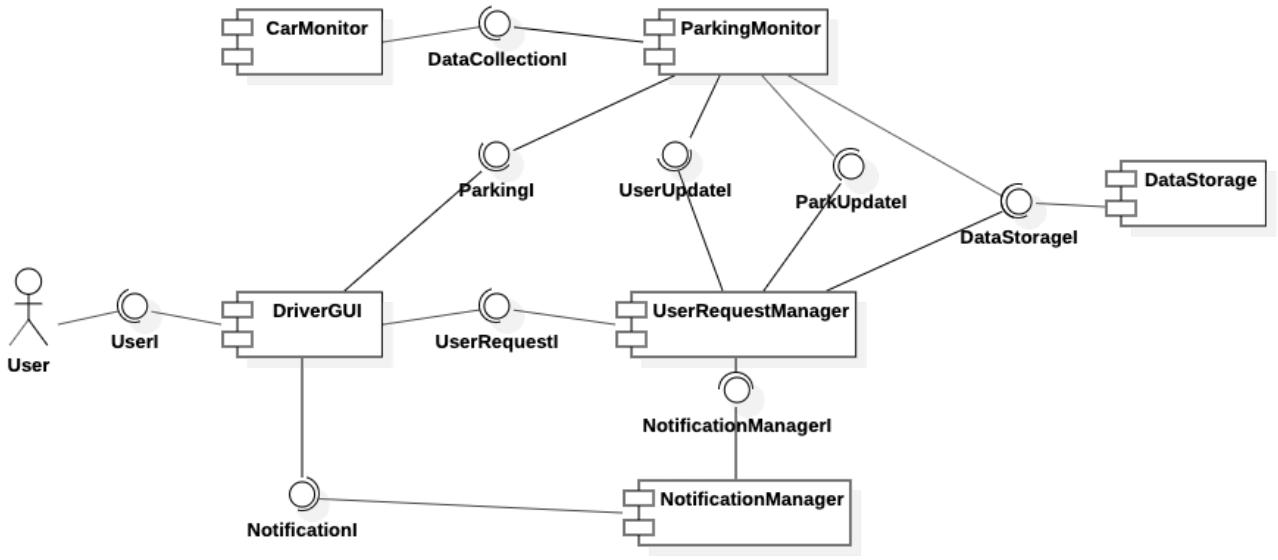
DA2: GPS sensors have an error of no more than 5 meters
DA3: If a car is stopped with the engine off, it is parked
DA4: The fleet of monitored cars is big enough that, if there is a parking place available in an area of the city that has the radius set by the user, there is at least one parking place that was freed by a monitored car
DA5: A parking place will remain available for at least X minutes, unless it is filled by another car that is part of the monitored fleet

DA1-DA5 and R1-R7, R11 ensure that the system knows when there are at least some parking places available in the radius of interest of the user (it is not necessarily aware of all of them, but at least of some of them). R8-R10 ensure that the information about available nearby parking places reaches the user.

RASD_Q3



DD_Q1



DD_Q2

DriverGUI
This is the front end to the user, which allows them to interact with the system. In particular, it provides interface *UserI* to allow users to send requests to the system. It provides interface *NotificationI* to allow the other components of the system to notify the user of fact such as when a parking becomes available. It uses interface *ParkingI* to retrieve available parking places around the user (and then show them on the display).
More precisely, interface *UserI* provides the following operations:

- *informParkingAvailable*, which takes as input the number of parking places that the user sees
- *informParkingNotAvailable*, which takes as input the position of a parking place that is no longer available
- *requestNotifications*, *requestStopNotifications*, which do not take any inputs
- *setParameter*, which take as input the paramenter to be set, and the desired value.

Interface *NotificationI*, instead, provides the following operation:

- *getNotification*, which takes as input the information about the notification to be shown to the user.

CarMonitor
This is the module, which resides on the on-board computer, which retrieves the information from the car, and sends it to the *ParkingMonitor* component through interface *DataCollectionI*.

DataStorage
This is the component that permanently stores the data handled by the system, and in particular the information about cars (positions, status) as various time instants, the information about the available parking places, and the values of the parameters set by each user. Interface *DataStorageI* provides CRUD operations for the data handled by the system.

ParkingMonitor
This component is in charge of receiving the data from monitored cars, and determine when parking places become available and are occupied. It uses the *DataStorage* to store information about cars and parking places, it receives requests from the user application concerning the list of available parking

places, it receives manual updates about available parking places from the *UserRequestManager* component, to which it sends updates when a new parking place becomes available. It provides the following three interfaces:
DataCollectionI:

- *getCarPosition*, which takes as input the information about the id and current position of a monitored car
- *getCarStatus*, which takes as input the information about the id and current status of a monitored car

ParkingI:

- *gerParking*, which takes as input a position, and returns the list of available places in a radius that is defined by a suitable system parameter (an available place is simply represented by its GPS coordinates)

UserUpdateI:

- *parkingAvailable*, which takes as input the GPS coordinates of an available parking signaled by the user
- *parkingNotAvailable*, which takes as input the GPS coordinates of a parking that is no longer available, as signaled by the user

UserRequestManager
This component handles requests coming from the user. In particular, it handles requests to receive (or stop receiving) notifications when a new nearby parking becomes available (and it stores internally the list of users who asked to be notified), requests to set parameters, and information about parking places that are available and that have not been detected by the system, and about parking places that the system considers available, but which are not so. It relays information about available/not available parking places to the *ParkingMonitor*. It receives information about newly available parking places from the *ParkingMonitor*; upon receiving the information about a newly available parking place, the component checks if there are users nearby who requested to be notified of newly available parking places and sends them a notification through the *NotificationManager*. To determine which users are near the newly created parking place, it retrieves the position of users from the *DataStorage*.
The component provides two interfaces, *UserRequestI*, which is used to receive requests from users, and *ParkUpdateI*, which is used to receive information about newly created parking places. The interfaces offer the following operations:
UserRequestI:

- *parkingAvailable*, which takes as input the GPS coordinates of an available parking place
- *parkingNotAvailable*, which takes as input the GPS coordinates of a parking place that is no longer available
- *requestNotifications*, which takes as input the id of the user who requested to be notified of newly available nearby parking places
- *requestStopNotifications*, which takes as input the id of the user who requested to stop being notified of newly available nearby parking places
- *setParameter*, which takes as input the id of the user requests the parameter change, the name of the parameter to be changed, and the new value

ParkUpdateI:

- *newParking*, which takes as input the GPS coordinates of a parking that newly became available.

NotificationManager
This component handles notifications sent to users. It uses interface *NotificationI* to deliver the information to the user GUI, and it provides an interface, *NotificationManagerI*, that is used by the *UserRequestManager* component to communicate the information that must be sent to a user.
Interface *NotificationManagerI* provides the following operation:

- *sendNotification*, which takes as input the id of the user who should be notified and the information that must be sent