Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Matteo Camilli, 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)

February 10th, 2023

Last Name, First Name

Id number (Matricola)

Number of paper sheets you are submitting as part of the exam

Notes

1. You must write your name and student ID (matricola) on each piece of paper 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, except for an ebook reader.
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 for WE2: 1h and 30 mins**

System Description: SmartLightingKit

SmartLightingKit is a system expected to manage the lights of a potentially big building composed of many rooms (e.g., an office space). Each room of the building, may have one or more lights. The system shall allow the lights to be controlled – either locally or remotely – by authorized users. Local control is achieved through terminals installed in the rooms (one terminal per room). Remote control is realized through a smartphone application, or through a central terminal installed in the control room of the building. While controlling the lights of a room, the user can execute one or more of the following actions: turn a light on/off at the current time, at a specified time, or when certain events happen (e.g., a person enters the building or a specific room). Moreover, users can create routines, that is, scripts containing a set of actions. **SmartLightingKit** manages remote control by adopting a fine-grained authorization mechanism. This means there are multiple levels of permissions that may even change dynamically. System administrators can control the lights of every room, install new lights, and remove existing ones. Administrators can also change the authorizations assigned to regular users. These last ones can control the lights of specific rooms. When an administrator installs a new light in a room, he/she triggers a pairing process between the light and the terminal located in that room. After pairing, the light can be managed by the system.

Part 1 Requirements (7 points)

RASD_Q1 (2.5 points)
Consider the following goals of the **SmartLightingKit** system:

> **G1**: *System administrators want to install new lights.*
>
> **G2**: *Regular users want to set up routines to control (i.e., turn on/off), in a certain time window, the lights of the rooms they are authorized to manage.*

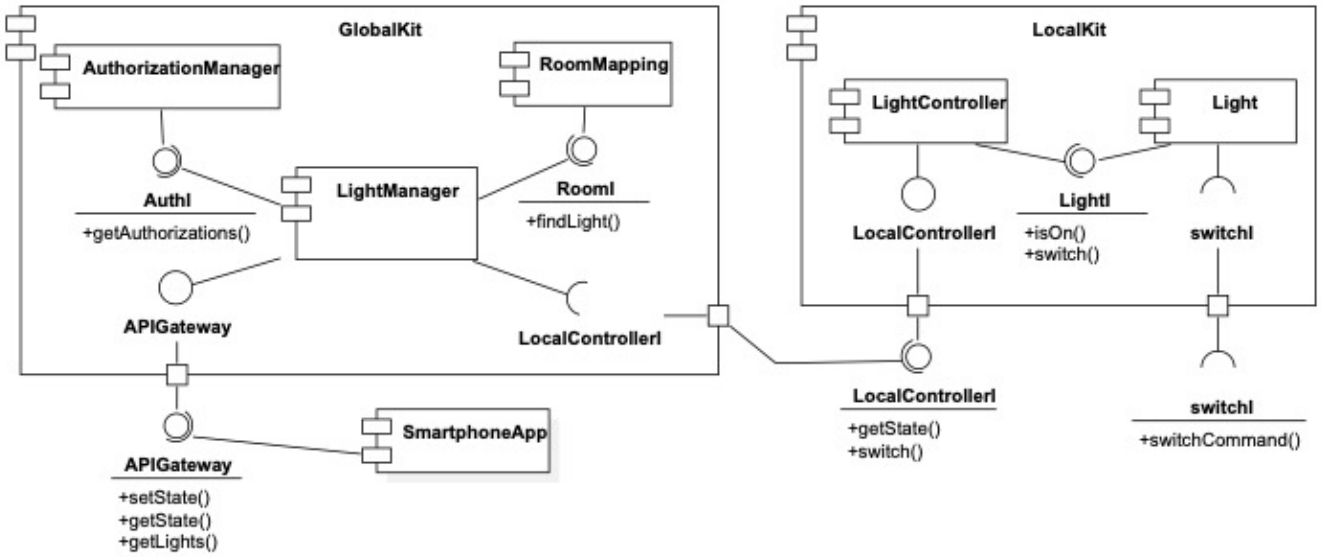
Define in natural language suitable domain assumptions and requirements to guarantee that the **SmartLightingKit** system fulfills the two goals.

RASD_Q2 (2.5 points)
Considering goals G1, G2, and the corresponding requirements, draw a UML Use Case Diagram describing the two actors (system administrators and regular users) and the use cases of the **SmartLightingKit** system.

RASD_Q3 (2 points)
Starting from the UML Use Case Diagram defined in the previous point, describe in detail (using the structure seen in class) the use case you consider most important to the achievement of goal G2 and motivate your choice.

Part 2 Design (7 points)

DD_Q1 (3 points)
The following UML Component Diagram describes the portion of the **SmartLightingKit** system in charge of supporting the turning on/off of the lights and the checking of the lights status in the building.



GlobalKit is the component installed in the central terminal. It can be contacted through the *APIGateway* interface by the *SmartphoneApp* component representing the application used for remote control. *GlobalKit* includes: (i) *RoomMapping*, which keeps track, in a persistent way, of lights' locations within the building's rooms; (ii) *AuthorizationManager*, which keeps track, in a persistent way, of the authorizations associated with each user (for simplicity, we assume that users exploiting the operations offered by the *APIGateway* are already authenticated through an external system and include in their operation calls a proper token that identifies them univocally); and (iii) *LightManager*, which coordinates the interaction with all *LocalKit* components. Each *LocalKit* component runs on top of a local terminal. Also, each *LocalKit* exposes the *LocalControllerI* interface that is implemented by its internal component *LightController*, which controls the lights in the room. Each light is represented in the system by a *Light* component which interacts with the external system operating the light through the *switchCommand* operation offered by the latter.

1. Analyze the operations offered by the components shown in the diagram and identify proper input and output parameters for each of them. If necessary/useful, you may use a UML Class Diagram to describe the types of elements handled by the various operations.
2. Write a UML Sequence Diagram illustrating the interaction between the software components when a regular user wants to use the smartphone application to check whether he/she left some lights on (among the lights he/she can control).

DD_Q2 (2 points)
Assume that, at a certain point, the following new requirement is defined:
NewReq*: The smartphone application should allow users to activate/deactivate the receival of on-the-fly updates about the state (on/off) of all the lights they can control.*
Define a high-level UML Sequence Diagram to describe how the current architecture could accommodate this requirement.
Highlight the main disadvantage emerging from the sequence diagram.

DD_Q3 (2 points)
Modify the high-level architecture of **SmartLightingKit** by producing a new version of the UML Component Diagram to solve the major issue(s) identified in the previous point. Then, draw a UML Sequence Diagram showing how the interaction changes compared to the previous version.

Solutions

RASD_Q1

G1: *System administrators want to install new lights.*

Requirements

- **R1_1**: *the system shall allow a system administrator to log into the system from a room using the corresponding room terminal.*
- **R1_2**: *the system shall allow a system administrator to trigger the pairing process in a room if he/she is authenticated in that room.*
- **R1_3**: *when the pairing process of a room is active, the system shall detect any new light in that room if the communication protocol of the light is X.*
- **R1_4**: *when a new light is detected, the system shall record the mapping between the light and the corresponding room.*

Assumptions

- **A1_1**: *system administrator accounts exist and are pre-loaded into the system.*
- **A1_2**: *the system administration (or a delegated actor) has activated the pairing process on the new light.*
- **A1_3**: *the new light uses the communication protocol X.*

A new light is successfully installed if the light has been detected and the mapping between the light and the corresponding room is recorded into the system. R1_4 ensures that such a mapping eventually exists if the pairing in that room is active (due to R1_3). It is worth noting that the system administrator can trigger the pairing because of R1_1 and R1_2. Furthermore, the system can detect the new light because the pairing of the light is on (assumption A1_2) and the communication protocol of the light is the one adopted by the system (assumption A1_3). Assumption A1_1 ensures that there exists at least one system administrator that can log into the system and trigger the pairing.

G2: *Regular users want to set up routines to control (i.e., turn on/off), in a certain time window, the lights of the rooms they are authorized to manage.*

Requirements

- **R2_1**: *the system shall allow system administrators to register new regular users for a given room.*
- **R2_2**: *the system shall grant control authorization for all lights of a room to each regular user registered for that room.*
- **R2_3**: *the system shall allow regular users to authenticate themselves.*
- **R2_4**: *the system shall receive as input a routine from an authenticated regular user for a given room.*
- **R2_5**: *the system shall parse a given routine and record each valid entry (i.e., a pair ≪on/off command, timestamp or event≫) for the given room if the author of the routine can control the room.*
- **R2_6**: *for all recorded entries, the system shall execute the existing commands at the corresponding timestamp.*

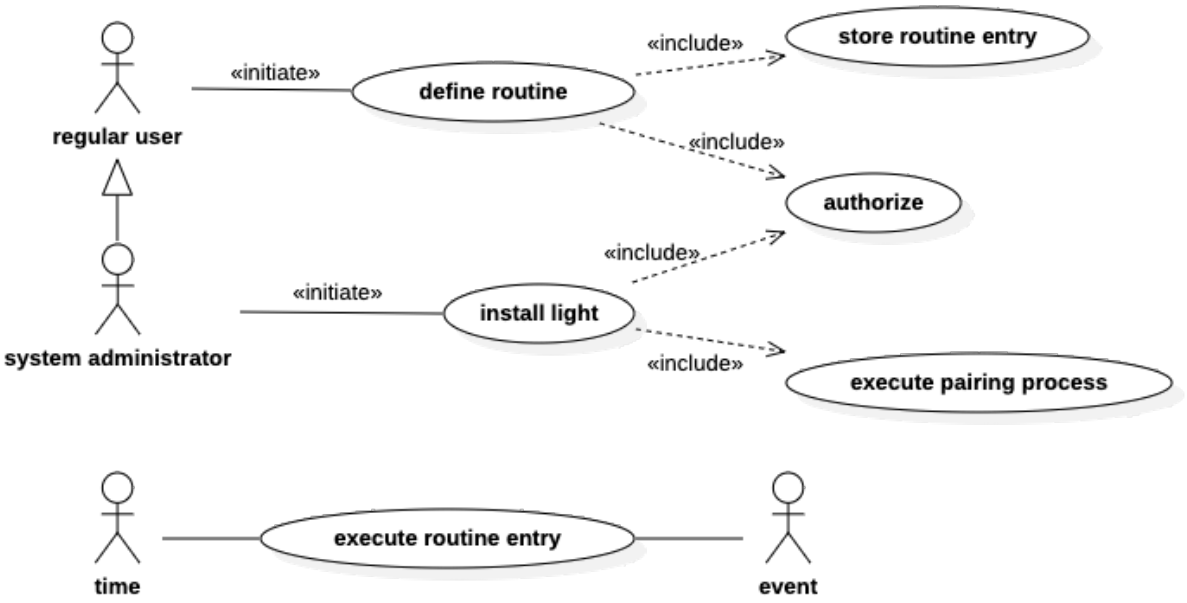
Assumptions

- **A2_1**: *the regular user defines a routine that captures correctly his/her intentions.*

The goal is achieved if a given routine is recorded in the system and the system triggers the commands of the routine on due time. Because of R2_4, the system can receive new routines from regular users that have been registered by an existing admin for a given room A (R2_1). The routine is accepted since the regular user can log into the system (R2_3) and send a new routine for A (R2_4). Note that the user can control all the lights of A because of R2_2. Assumption A2_1 ensures the routine fulfills the expectations

of its author. Thus, at least one entry eventually exists in the system due to R2_5. Commands of all the entries (including the new one) are then executed at the corresponding timestamp due to R2_6.

RASD_Q2



The execution of routine entries is activated depending on the time passing or based on other events such as the entrance of a person in the room. This is modeled through the time and event actors.

RASD_Q3

The following detailed description refers to the "define routine" use case that realizes some of the requirements to achieve goal G2: requirements R2_4, R2_5, R2_6.

Use case	Define routine
Actors	Regular user (initiates the use case)
Entry condition	- The user exists and his/her account is recorded into the system - The user is authenticated
Flow of events	1. From the UI of the application, the user selects "new routine" entering the new routine setup activity. 2. The user selects a room through a drop-down menu. 3. The user selects the type of routine between time-based and event-based. 4. If time-based, the user selects one or more days of the week and then inserts one or more time windows by selecting the starting and ending time, up to minutes precision, from a list box in order to avoid wrong formats. 5. If event-based, the user selects the type of events that can activate the lights from a drop-down menu. 6. For each time window and/or type of event, the user selects an action between "lights on" and "lights off". 7. The user confirms the routine using the corresponding button in the UI. 8. The system extracts the full list of entries *≪on/off command, timestamp or event≫* from the routine.

	9. The system checks if the user can control the selected room by activating the use case *authorize*. 10. The system stores each routine entry into the system by activating use case *store routine entry*. 11. The system confirms to the user the routine has been recorded by showing a success message in the UI.
Exit condition	The routine entries are recorded into the system using persistent storage. A success message is shown to the user.
Exceptions	The user is not authorized to control the selected room. In this case, the system shows an error message and asks the user to select another room.

DD_Q1

APIGateway
- *getLights*: input = userID; output = list[lightID]
- *getState*: input = userID; output = list[(lightID, state)]
- *setState*: input = userID, lightID, state; output = none

AuthI
- *getAuthorizations*: input = userID; output = list[(roomID, localKitID)]

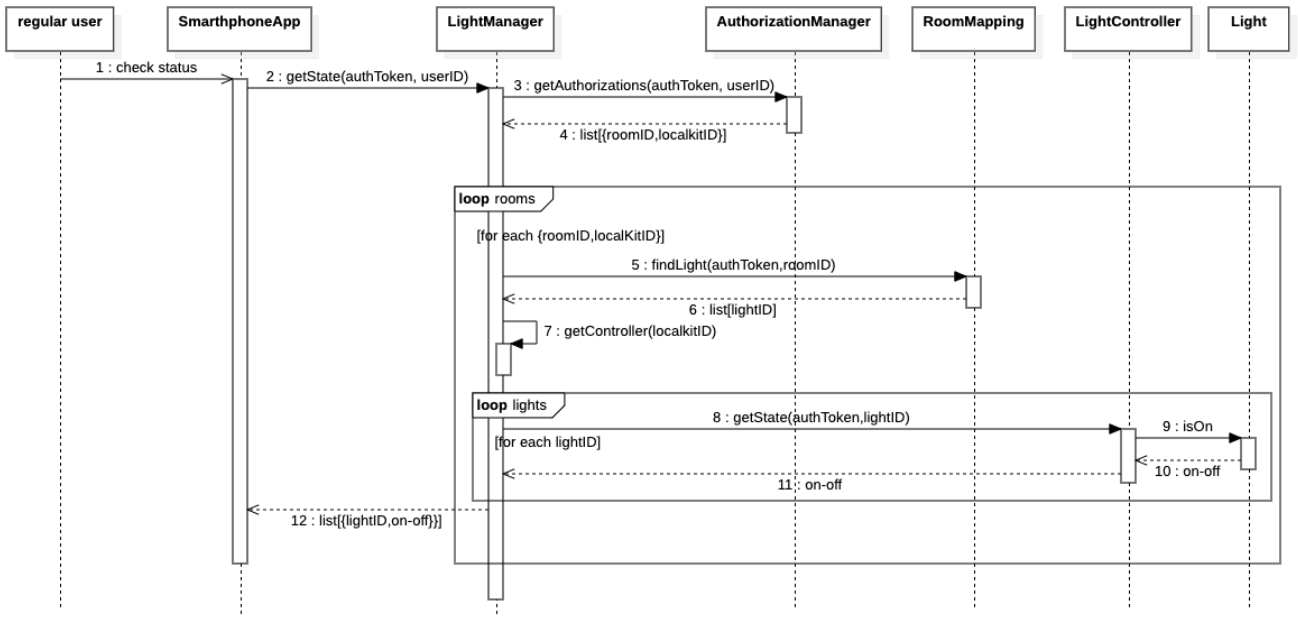
RoomI
- *findLight*: input = roomID; output = list[lightID]

LocalControllerI
- *switch*: input = lightID; output = none
- *getState*: input = lightID; output = state

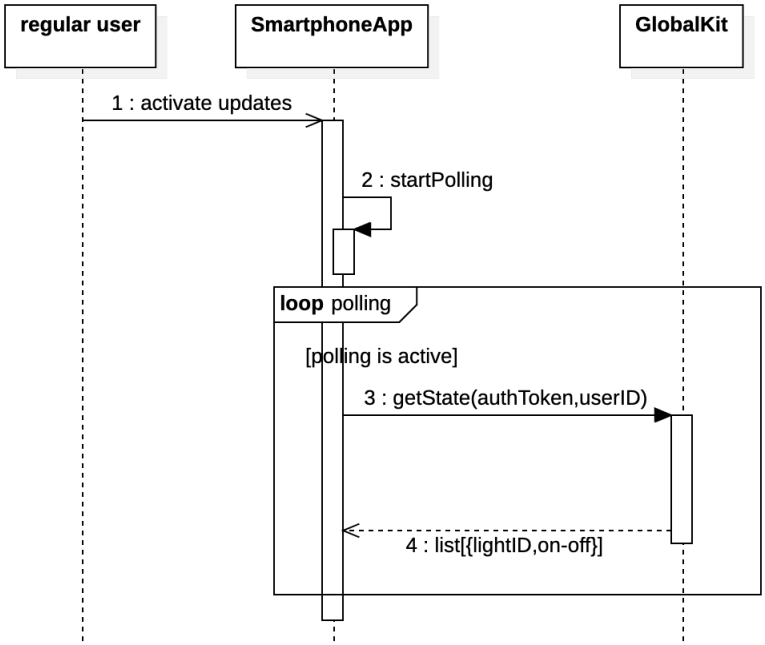
LightI
- *isOn*: input = none, output = True/False
- *switch*: input = none, output = none

SwitchI
- *switchCommand*: input = none; output = none

Note
- State = On/Off;
- All the operations of *APIGateway*, *AuthI*, *RoomI*, *LocalKitI* receive as input also the authentication token.



DD_Q2

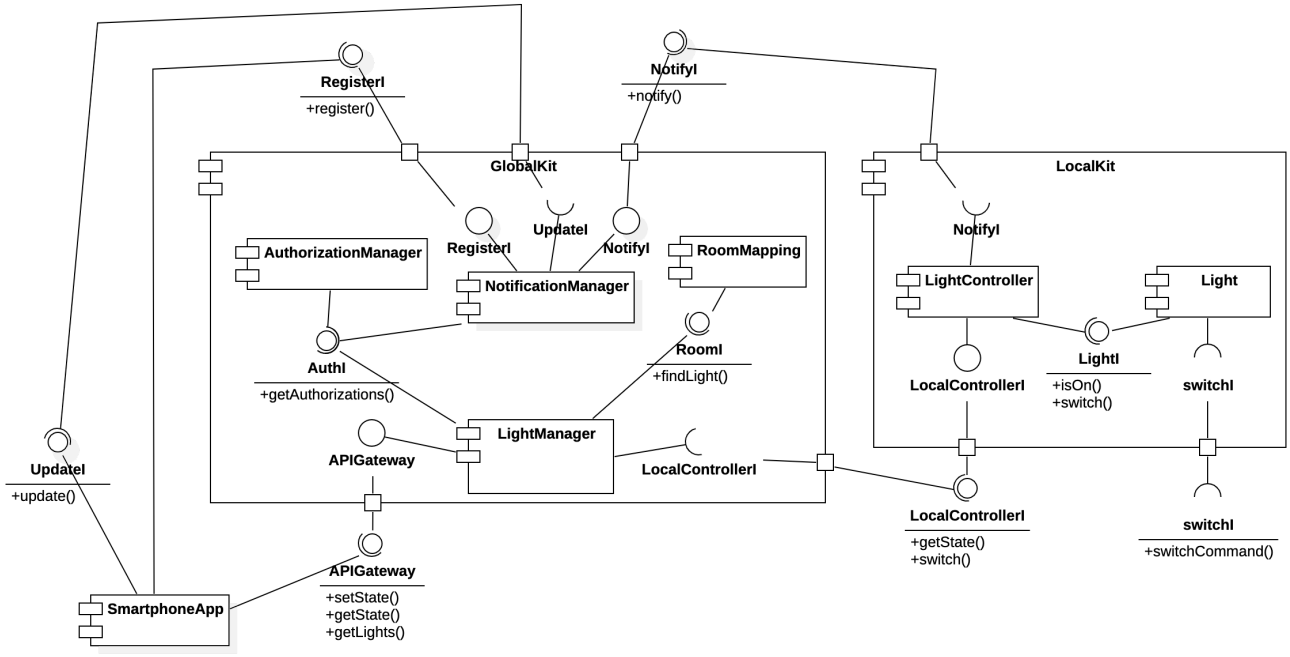


Note that this diagram is high level and focuses on the interaction between the *SmartphoneApp* and *GlobalKit* for the specific purpose we are analysing. The previous sequence diagram describes the communication between the internal components when the *getState* operation is called on *GlobalKit*.

The main disadvantage of the current architecture is that it does not support updates in *push* mode. Therefore, the *SmartphoneApp* carries out a continuous polling process to retrieve the status of all the lights even in case it does not change. This continuous polling propagates also internally to *GlobalKit* and the involved *LocalKit*s, thus resulting in a potentially significant communication overhead.

DD_Q3

A notification mechanism can solve the problem highlighted in the previous point. The new version of the UML Component Diagram follows.



In this case, *LightController* in each *LocalKit* can inform the *NotificationManager* about any state change in the lights it is controlling. Note that the *LocalControllerI* is supposed to be used not only by *GlobalKit* when a control command is issued from remote, but also by the local terminal that allows users to switch the lights from the room itself. In both cases, *LightController* is informed of the operation.
Furthermore, note that *NotificationManager* must know whether a subscribed users is authorized to be informed of the state changes of a specific light. This can be achieved in multiple ways. For example, the *NotificationManager* can use the *AuthI* interface offered by *AuthorizationManager* to retrieve the authorizations of a user. This can happen in two different moments, when the user registers for updates, or before updates are sent. The first case is more effective if registration for updates are less frequent than the occurrence of new updates and if authorizations assigned to users do not change or change rarely (when this happens, we would need to re-check again all registrations for updates associated with the corresponding user). The second case requires that the *NotificationManager* interacts with the *AuthorizationManager* each time a new update is to be sent to a specific user.
The following UML Sequence Diagram shows the new *push* mode interaction. The triggering event is the activation of the switch operation on *LightController*.