POLITECNICO
MILANO 1863

Exercise Lecture 3

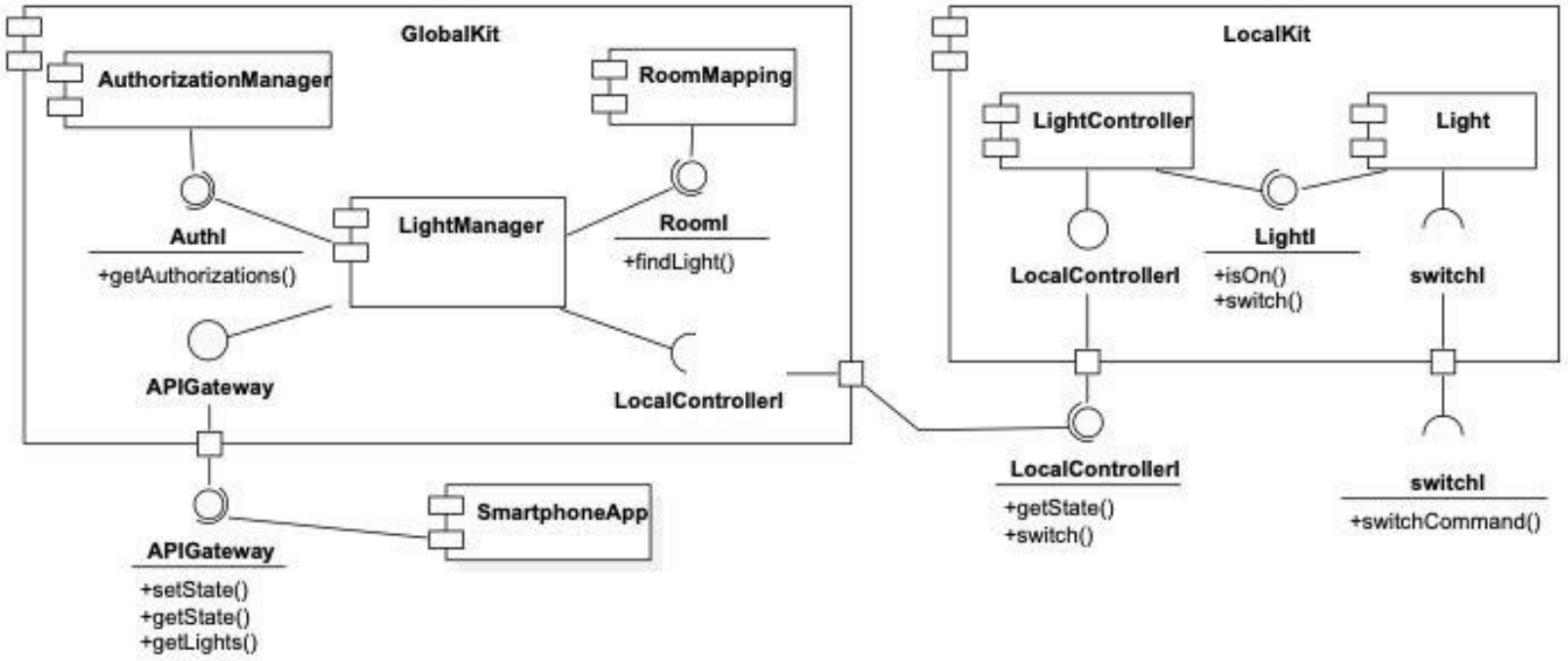
SmartLightingKit

POLITECNICO MILANO 1863

- 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.

SmartLightingKit

- This diagram describes the portion of the SmartLightingKit system supporting lights turning on/off and lights status checking.



SmartLightingKit – part 2

POLITECNICO MILANO 1863

- The diagram is complemented by the following description
  - `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:
    - `RoomMapping`, which keeps track, in a persistent way, of lights' locations within the building's rooms;
    - `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
    - `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.

SmartLightingKit – part 2 (cont.)

- Q1:
  Analyze the operations offered by the components shown in the diagram and identify proper input and output parameters for each of them.

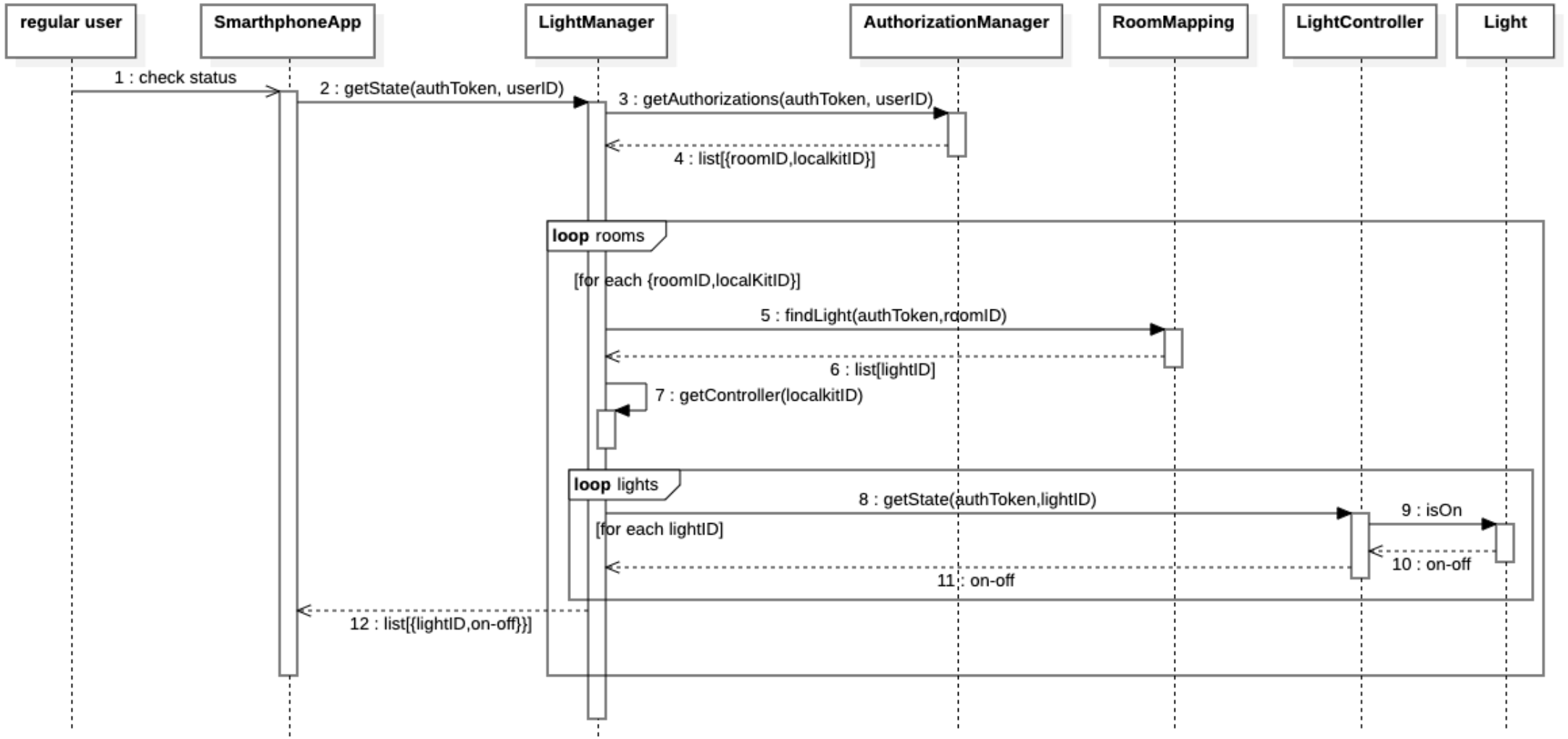
Operations

- **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.

- Q2: 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).

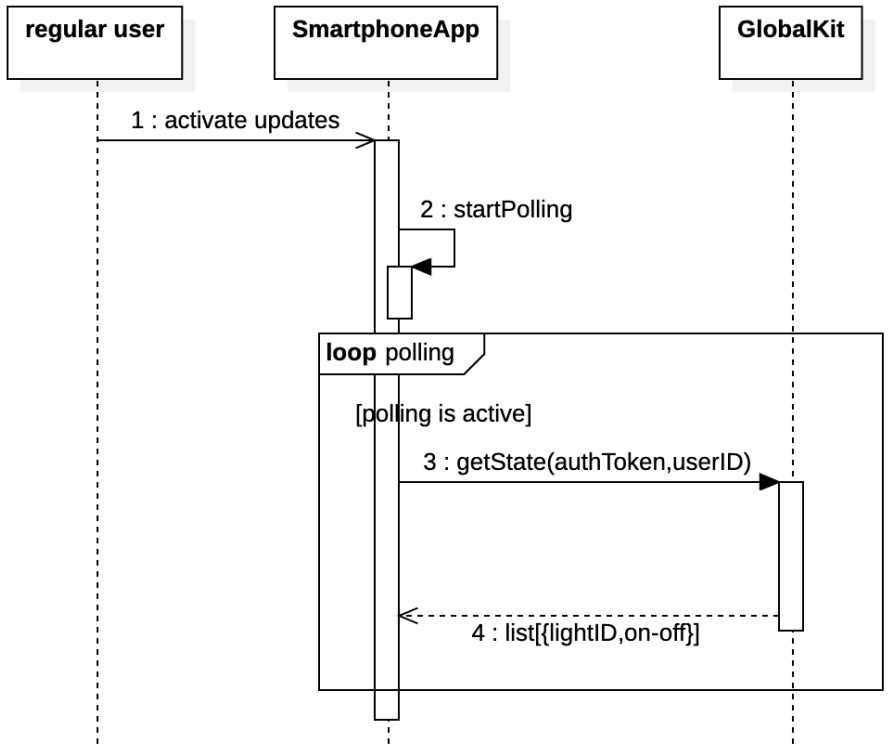
Sequence diagram



SmartLightingKit – part 3

- 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 real-time 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.

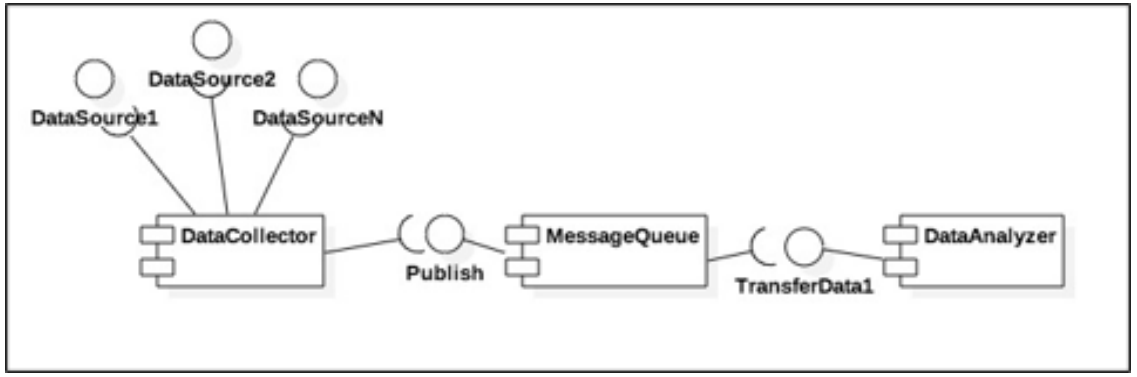
Realizing NewReq



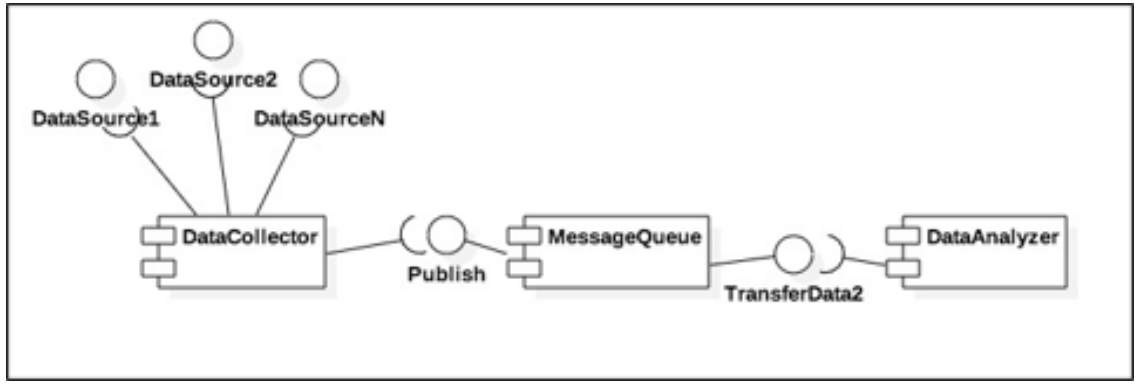
- Problem: the current architecture does not support updates in push mode.
- The `SmartphoneApp` carries out a continuous polling process to retrieve the status of all the lights even in case it does not change.
- This propagates also internally to `GlobalKit` and the involved `LocalKits`, thus resulting in a potentially significant communication overhead.

Data Collector

- Consider a software system that acquires and elaborates information from a number of different sources by polling them periodically.
- The two component diagrams below provide an overview of the types of components defined for this system, with two possible types of interaction between some of them.



Component Diagram 1



Component Diagram 2

Data Collector

- Describe the system in the two versions, focusing on the possible interactions between the various components, based on the information you can derive from the component diagrams.

Component Diagram 1:

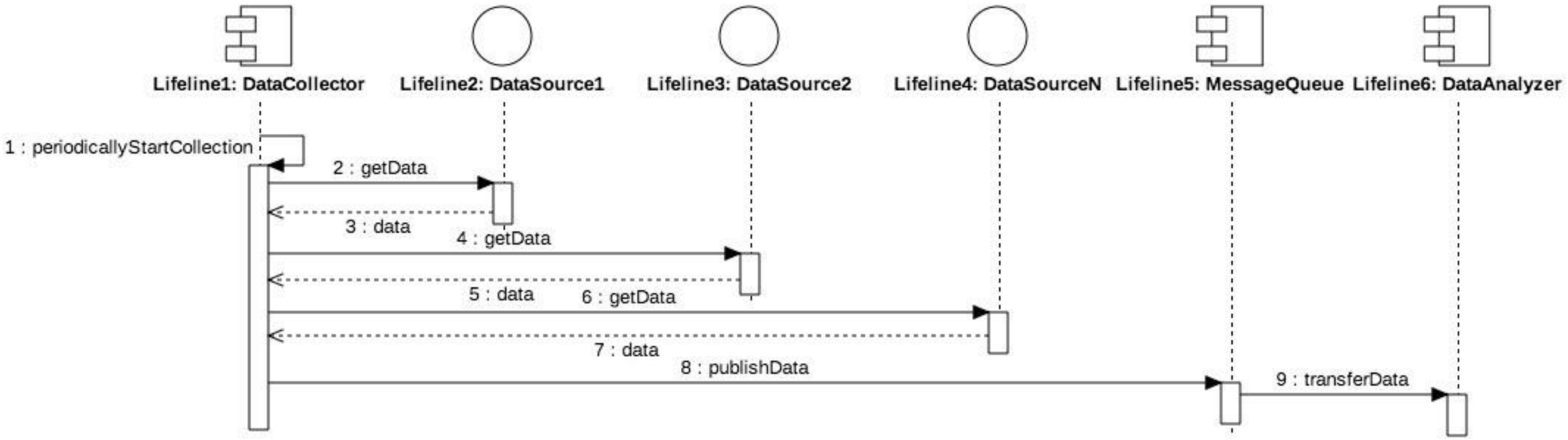
- The DataCollector component is exploiting the interfaces offered by some data sources to acquire data and the interface of the MessageQueue to pass collected data to the other components. Based on the way it interacts with the other elements, we can assume that the DataCollector is continuously or periodically polling the sources and is transferring the received data to the MessageQueue. We do not know if such transfer occurs in batches or not. The MessageQueue exploits the interface offered by the DataAnalyzer to pass the data to this component. Once again, we have no information on whether this transfer occurs in batches. We could derive such information from an analysis of the interfaces offered by MessageQueue and DataAnalyzer, respectively.

Component Diagram 2:

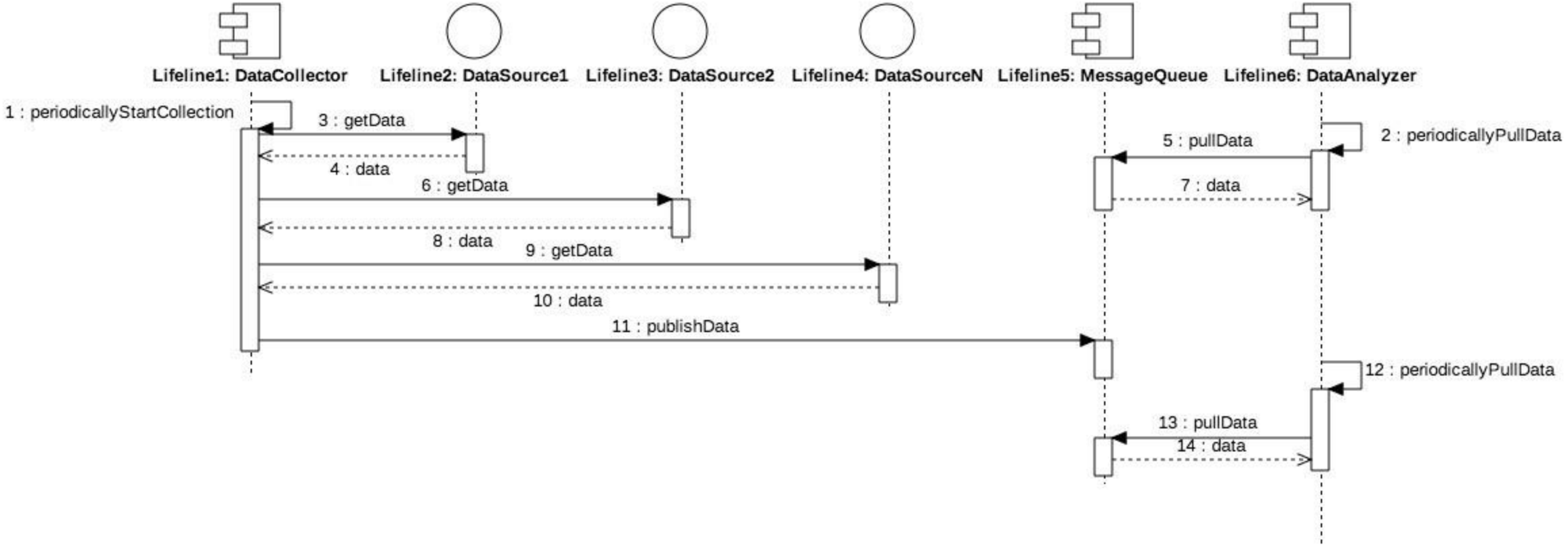
- In this case, the MessageQueue does not actively push the data to the DataAnalyzer, but it offers interface TransferData2 so that the DataAnalyzer can pull data as soon as it is ready to process them. Also in this case, both a batch or a per data approach is possible. The rest of the system behaves as already described above.

- With reference to the component diagram in Figure 1, define a sequence diagram that describes the way data flow through the system. How would this sequence diagram change in the case of the component diagram of Figure 2?

Option 1


Lifeline1: DataCollector
Lifeline2: DataSource1
Lifeline3: DataSource2
Lifeline4: DataSourceN
Lifeline5: MessageQueue
Lifeline6: DataAnalyzer
1 : periodicallyStartCollection
2 : getData
3 : data
4 : getData
5 : data
6 : getData
7 : data
8 : publishData
9 : transferData

Option 2


Lifeline1: DataCollector
Lifeline2: DataSource1
Lifeline3: DataSource2
Lifeline4: DataSourceN
Lifeline5: MessageQueue
Lifeline6: DataAnalyzer
1 : periodicallyStartCollection
3 : getData
4 : data
6 : getData
8 : data
9 : getData
10 : data
11 : publishData
5 : pullData
7 : data
2 : periodicallyPullData
12 : periodicallyPullData
13 : pullData
14 : data

Kitchen Designer

POLITECNICO MILANO 1863

- We want to build an application, KitchenDesigner, that allows users to define the layout of kitchens and to insert in such layouts the furniture and appliances (refrigerators, stoves, dishwashers, etc.) that go in them. For simplicity, we consider only rooms that have rectangular shape.

- Users can define the physical features of the room (length, width, height). Moreover, they can add pieces of furniture and appliances, and move them around. The position of each piece of furniture/appliance is given by the 3D coordinates of the lower left corner of the bottom side of the item, and by its orientation (which is the angle with respect to the x axis, and which can only be a multiple of 90 degrees).

- Users register with the application to be able to store and retrieve their designs. After a user finalizes his/her kitchen design, he/she can ask to have the kitchen delivered to a desired address; in this case, the kitchen is sent to production, and the user is given a probable date of delivery (producing a kitchen can take a few days, or even weeks). For simplicity, we do not consider payment. When the kitchen is ready to be delivered, the user is notified of the confirmed date of delivery.

- The system keeps track of the designs created by users, to identify the most common combinations of pieces of furniture and appliances. Hence, upon request by a user, given a draft layout for the kitchen, the system returns a list of possible pieces of furniture and appliances that might be added to that kitchen.

Goal and Requirements

Goal:

- G1: Users have their desired kitchens delivered to them

Requirements

- R1: The system must allow users to define the dimensions of the kitchen
- R2: The system must allow users to add items (pieces of furniture and appliances)
- R3: The system must allow users to remove items from a kitchen
- R4: The system must allow users to move items around a kitchen
- R5: The system must allow users to finalize a kitchen design
- R6: The system must allow users to order a finalized kitchen design
- R7: The system must notify the factory about the order
- R8: The system must inform the user about the forecasted and the actual delivery date

Additional requirements related to the system storing the kitchen designs and mining them as basis to provide suggestions to users, and also requirements related to the user registering and being able to ask for suggestions, are not listed here, because they are not necessary to meet the specific stated goal, though they are relevant for the application.

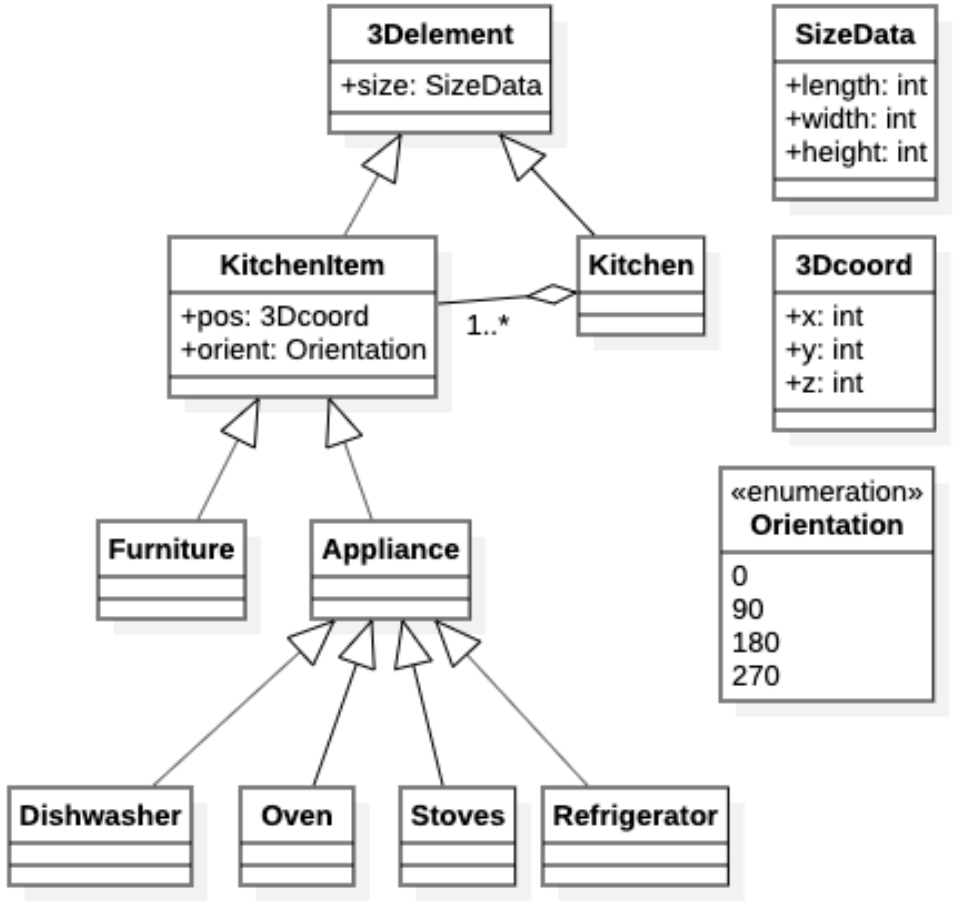
Domain Assumptions

- DA1: The information input by the user in the application (size of the room, pieces of furniture, appliances, etc.) is accurate
- DA2: After order is confirmed, the kitchen is prepared and delivered

DA1 and R1-R4 guarantee that the system knows the layout of the kitchen that the user wants. Then, R5-R8 and DA2 guarantee that the kitchen is delivered to the user.

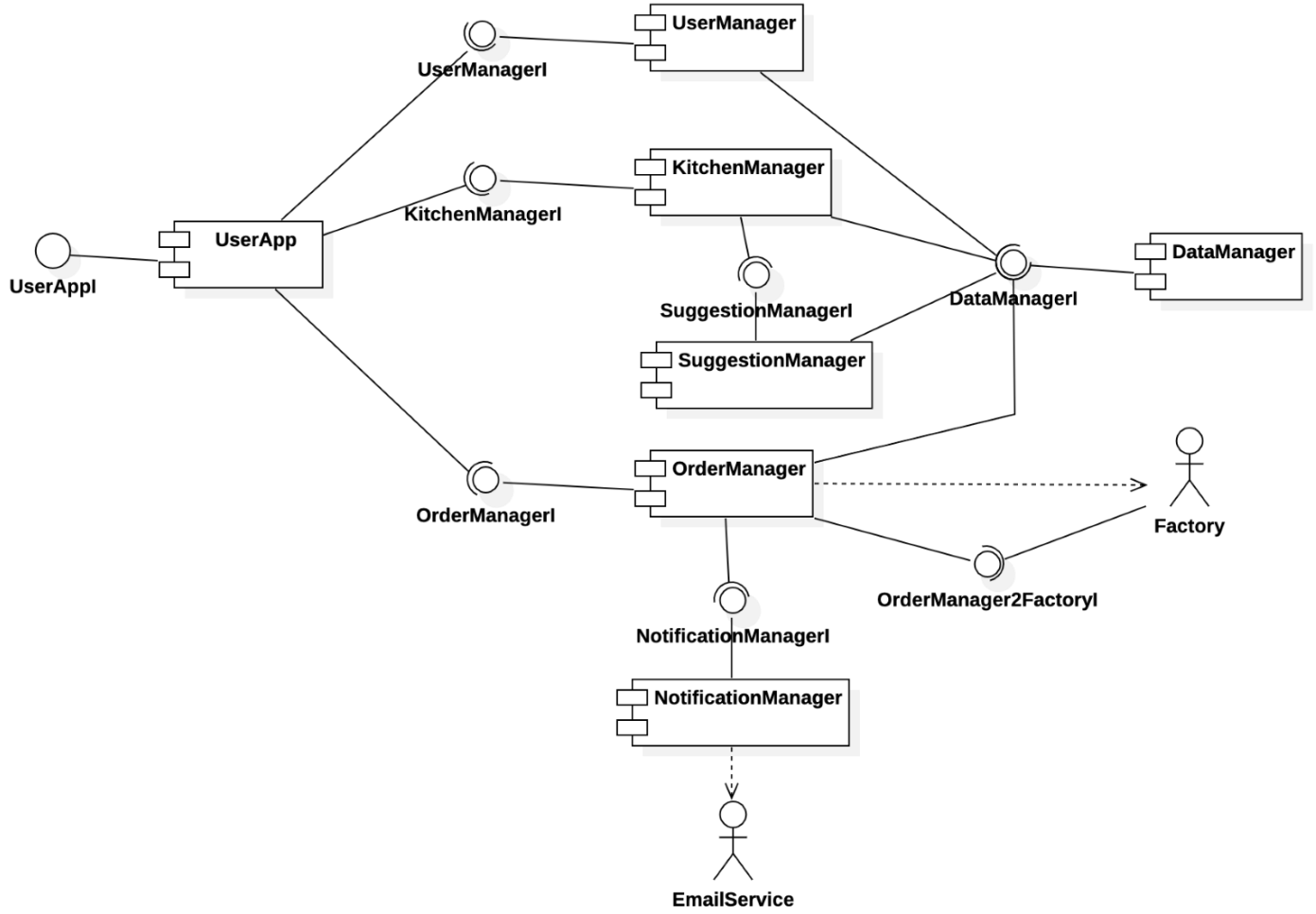
Describe the relevant elements of the domain of the KitchenDesigner system, use a UML Class Diagram for this purpose.

Class Diagram



- Assuming you need to implement system KitchenDesigner analyzed above, identify the most relevant components and interfaces and describe them through a UML Component diagram.
- Provide a brief description of each component.

Component Diagram



Interfaces

UserApp

This is the front-end for users. It allows them to interact with the system by offering the following functions through interface UserAppI:

- Register (input: user data)
- Login (input: userid and password)
- Create a new kitchen project (input: name)
- Set the dimensions of the kitchen (input: dimensions)
- Add an item to kitchen (input: item to be added)
- Move item in kitchen (input: item to be moved, new position/orientation)
- Remove item from kitchen (input: item to be removed)
- Ask for a suggestion (returns suggested elements)
- Finalize kitchen
- Place order

The module can keep track of the current kitchen being designed, so the user operates on the “open project”, and the information does not need to be included in the calls each time.

UserManager

This component offer, through interface *UserManagerI*, the basic functions for handling users:

- Register (input: user info)
- Login (input: user id and password)

Kitchen Manager

This component provides interface *KitchenManagerI*, which handles functions related to the management of kitchens (excluding placing the order, which is handled by another component):

- Create a new kitchen project (input: name)
- Set the dimensions of the kitchen (input: kitchen id, dimensions)
- Add an item to kitchen (input: kitchen id, item to be added)
- Move item in kitchen (input: kitchen id, item to be moved, new position/orientation)
- Remove item from kitchen (input: kitchen id, item to be removed)
- Ask for a suggestion (input: kitchen id, returns suggested elements)
- Finalize kitchen (input: kitchen id)

These operations are similar to those offered by the *UserApp*, but they also include the id of the kitchen which should be modified.

SuggestionManager

- This component provides, through interface *SuggestionsManagerI*, functions related to the retrieval of suggestions:
- Get suggestion (input: kitched id)
- The idea is that the component periodically retrieves kitchen designs from the *DataManager*, mines them, and identifies which combinations of items are most common. Hence, it only provides a single function, for getting the outcome of this mining. Hence, the computation is mostly offline, the “get suggestion” function compares what is present in the kitchen with what is most common, and suggests additional elements.

OrderManager

- This component provides, through interfaces *OrderManagerI* and *OrderManager2FactoryI* functions related to the management of orders. These are used by 2 different clients. Interface *OrderManagerI* is used by *UserApp*; it provides the following function
- Place order (input: kitchen id)
- which is used to start the process to produce a kitchen. To handle the order the *OrderManager* needs to notify the factory of the new order. The handling of the order, and in particular its creation, is outside of the scope of the *KitchenDesigner* application; this is represented in the diagram by the fact that there is an interaction with the factory. When the order is complete, the *OrderManager* is informed of this through interface *OrderManager2FactoryI* (to be used by the external actor “factory”) which provides the following operation:
- Kitchen completed (input: kitchen id)
- Which simply informs the *OrderManager* that the kitchen is indeed ready (hence the user can be notified of the date of its actual delivery).

NotificationManager
This component handles the notification to users, in particular updates on when the kitchen will be delivered.
It provides the following function through interface *NotificationManagerI*:

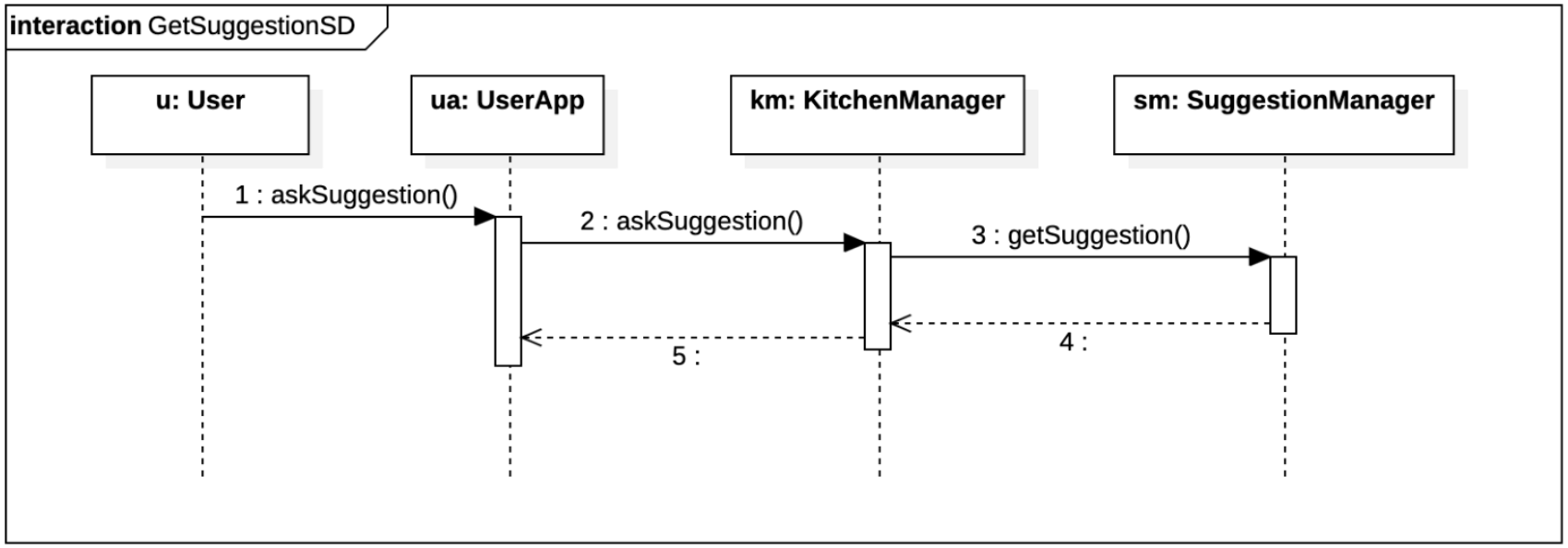
- Notify user (input: message to be sent, recipient)

The notification is sent as an email, and for this reason it goes through an email server, which is an external component, outside of the system.

Data Manager
This component handles the data of the system, which consists essentially of Users and Kitchens. It provides interface *DataManagerI*, which includes all necessary functions to handle CRUD operations on data.

- Define a Sequence Diagram describing the interaction that occurs among the KitchenDesigner components when the user asks for a list of suggested elements to be added to the kitchen.

Sequence Diagram



As explained above, the mining of the designs is done asynchronously, not when a suggestion is requested, but offline, so it is not represented in this interaction.