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)

June 26th, 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: SmartDaC – Smart City Data Collector
SmartDaC is a software system used to collect data from multiple sources in a smart city and to offer such data to interested parties. SmartDaC works following a publish-subscribe approach and, at the same time, keeps track of acquired data in a repository. It offers two APIs, one for the publish-subscribe operations and another for data extraction. Moreover, it features a GUI that allows users to register new sources in the system and to exploit the operations offered by the two APIs, visualizing on the screen the corresponding results. The following scenarios give an idea of how the system works.

Scenario 1: Mary wants to register a traffic sensor board as a new source of data. She uses the GUI offered by SmartDaC and selects the source registration operation. She is asked to select the type of source from a list or to add a new type in case none of those in the list characterizes her source. She sees that the currently available types are "traffic camera", "surveillance camera", "pollution sensor", and "public transport sensor". Since none of these types represents the characteristics of her source, she decides to define the new type "traffic sensor". She then inserts information about the position of the traffic sensor as GPS coordinates. SmartDaC acknowledges the acquisition of data and returns a URI for the publication API, together with a link to the human-readable documentation of such API. Mary then configures the traffic sensor to use the specified URI and publish data according to the format required by the API. As soon as she restarts the sensor, it starts to periodically send data to SmartDaC.

Scenario 2: John is creating a data analysis dashboard aiming at highlighting the correlation between traffic levels and pollution. He decides to exploit the data extraction capabilities offered by SmartDaC. He studies the API offered by this system and creates a software component that every week extracts from SmartDaC the following information: all pollution records produced through the week and all data produced by traffic sensors in the same timeframe. Moreover, in order to make precise correlations, he extracts also information about the positions of the two types of data sources.

Scenario 3: Jerry and Anna are the maintainers for the software developed by John (see the previous scenario) and realize that they can exploit the subscription mechanism offered by SmartDaC to acquire the relevant data on-the-fly and not only every week. They decide to modify their system so that it contacts SmartDaC to subscribe to the data generated by the pollution sensors and by the traffic sensors. As soon as these subscriptions are concluded, their application starts receiving data while they are being produced.

Part 1 Requirements (7 points)

RASD_Q1 (2.5 points)
Consider the following goal defined for the SmartDaC system:

> **G1**: *Users want to be able to collect data from multiple sources and to get access to such data both in real time (that is, as soon as they are produced by the sources) and in an off-line manner by extracting data by source type, specific source and/or specific time interval.*

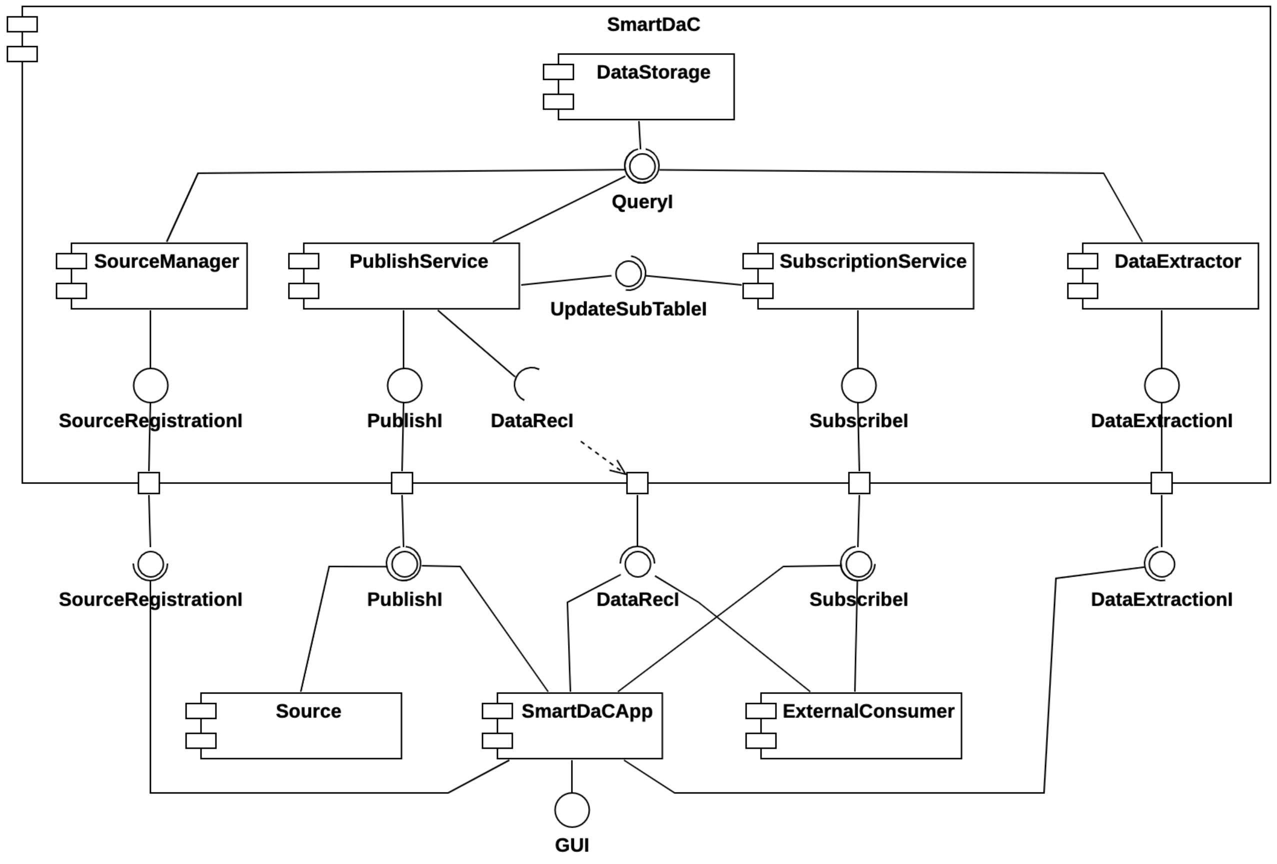
Define in natural language suitable domain assumptions and requirements to guarantee that the SmartDaC system fulfills the goal.

RASD_Q2 (2.5 points)
Considering goal G1 and the corresponding requirements, draw a UML Use Case Diagram describing the actors and the use cases of the SmartDaC 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 G1 and motivate your choice.

Part 2 Design (7 points)

The UML Component Diagram below describes the SmartDaC architecture. The system exposes the APIs for registering sources (*SourceRegistrationI*), to allow subscription for incoming data (*SubscribeI*), publication of new pieces of data (*PublishI*), and extraction of stored data (*DataExtractionI*). Each of these APIs is offered by a specific component internal to SmartDaC. *SourceManager* and *PublishService* rely on *DataStorage* for data recording concerning the sources and the published data, respectively. *DataExtractor* uses *DataStorage* to query data. *SubscriptionService* interacts with *PublishService* to update its subscription table as soon as a new subscription request comes. When data is published, *PublishService* uses the *DataRecI* interface of all corresponding subscribers for sending them data. *SmartDaCApp* is the application used by human users to interact with the system. *Source* and *ExternalConsumer* represent a generic source and a generic data consumer, respectively.



DD_Q1 (3 points)
Analyze the operations offered by the components shown in the diagram and identify proper input and output parameters for each of them. You can skip the *DataStorage* component that offers the usual CRUD (Create, Read, Update, Delete) operations. If necessary/useful, you may use a UML Class Diagram to describe the types of elements handled by the various operations.

DD_Q2 (2 points)
Write suitable UML Sequence Diagrams illustrating the interactions that occur among the software components in the following two cases:

- A user wants to extract data produced by all traffic sensors in the last 60 minutes.
- A source publishes a piece of data for which two subscriptions are active.

DD_Q3 (2 points)
You quickly realize that the number of sources and the amount of data they produce can be quite high. What are the parts of the architecture that could be negatively impacted by this problem? How could you mitigate it?

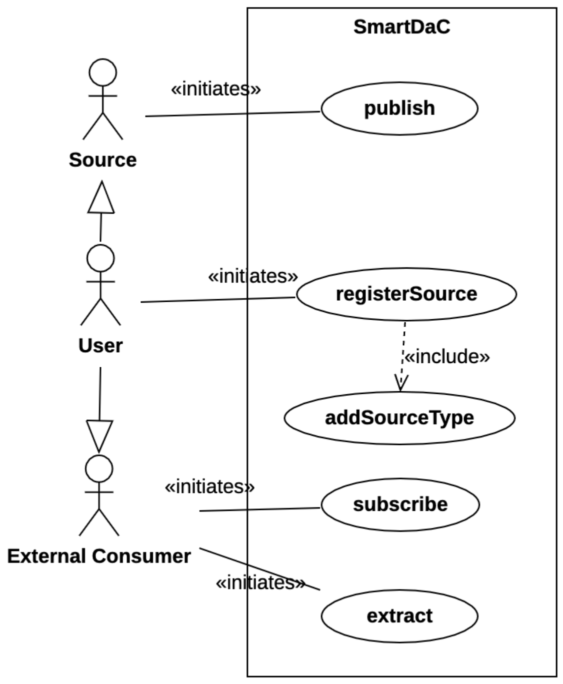
Solutions

RASD_Q1

R1 The system should allow users to register a new source by providing its position and source type and should send back to the user the URI to be used to contact the publish interface.
R2 The system should allow users to define a new source type.
R3 The system should offer a publish operation. When such operation is called, the system should record the received data and should send it to their current subscribers.
R4 The system should allow other components or users to subscribe to receive data. Subscriptions can be issued for a specific source, or for a source type.
R5 The system should allow other components or users to issue queries to extract data according to one of the following criteria at a time: their source type within a specified timeframe, their specific source within a specified timeframe, an interval to which their timestamps must belong.

A1 Sources are correctly configured and restarted by users once registered.

RASD_Q2



RASD_Q3

registerSource and *publish* are probably the most important use cases to achieve the goal. In fact, if the process for collecting data does not work properly, then the other operations do not lead to effective results. In this solution we decide to describe in detail the *publish* use case as it is the most articulated.

Use case	publish
Actors	Source
Entry condition	The source is registered in the system
Flow of events	1. The source issues the publish operation passing to SmartDaC some data. 2. SmartDaC performs the following actions: a. Acknowledges the operation.

	b. Stores the data in some repository, together with the source, source type and timestamp information. c. Identifies the subscribers for such data. d. Sends the data to all subscribers.
Exit condition	The published data has been recorded into the system and sent to the subscribers.
Exceptions	• SmartDaC is not able to send the data to one of the subscribers. In this case, the system will retry after a certain time interval. • SmartDaC fails while data is being sent to subscribers. In this case, when SmartDaC will be restarted, it will resume the sending process from the point in which it has been interrupted.

DD_Q1

SourceRegistrationI offers the following operations:

```
    SType[] getAvailableSourceTypes()
    boolean registerNewSourceType(SType s)
    SourceID registerSource(Coordinates c, SType s)
    PubAPIinfo getAPIinfo(SourceID s)
```

where `PubAPIinfo` is a data structure including the URI and the documentation of the publication API.

SubscribeI offers the following operations:

```
    boolean subscribe(SubscriptionType t, DataRecI d)
    boolean unsubscribe(SubscriptionType t, DataRecI d)
```

where `SubscriptionType` can be a superclass that is specialized in the various possible types of subscription (by source, or by source type), and `DataRecI` is passed as argument to allow the system to contact back the subscriber.

PublishI offers the following operation:

```
    boolean publish(Data d, SourceID s)
```

DataExtractionI offers the following operations:

```
    Source[] extractSources()
    Data[] extractDataBySource(SourceID s, Time start, Time end)
    Data[] extractDataBySourceType(SType s, Time start, Time end)
    Data[] extractDataByTimeInterval(Time start, Time end)
```

DataRecI is offered by the data consumers of SmarDaC and provides the following operation:

```
    boolean receiveData(Data d)
```

UpdateSubTableI is offered by the *PublishService* component and provides the following operation:

```
    boolean addSubscription (SubscriptionType t, DataRecI d)
    boolean removeSubscription(SubscriptionType t, DataRecI d)
```

GUI is the interface offered by SmartDaCApp to the users. It offers the following operations:

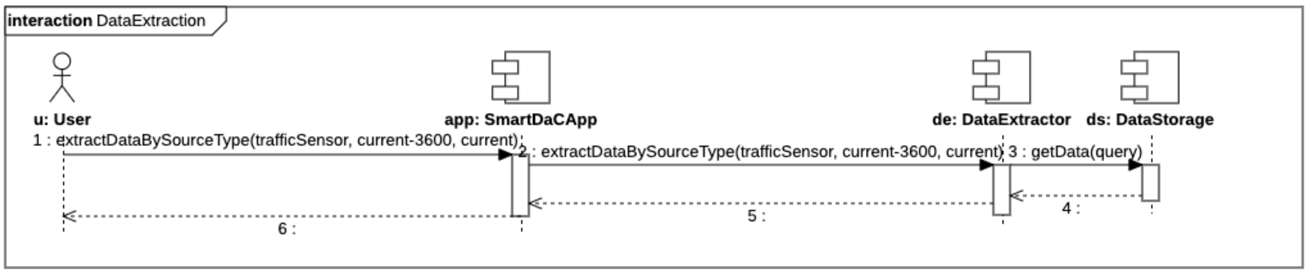
```
    visualizeSourceType
    registerNewSourceType
    registerSource
    subscribe
    unsubscribe
```

```
publish
extractSources
extractDataBySources
extractDataBySourceType
extractDataByTimeInterval
visualizeReceivedData
```

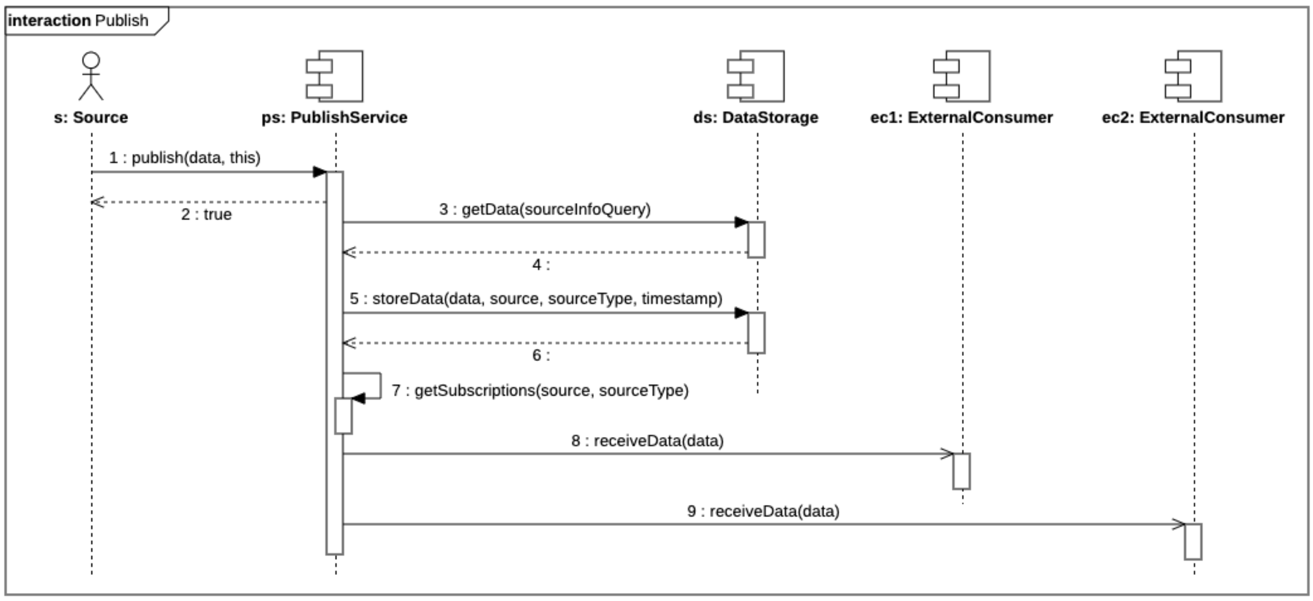
Note that all operations except the last one correspond to one of the operations offered by the SmartDaC system through the interfaces exposed for external use. These operations manipulate the same data structures mentioned above. The actual values to be assigned to such pieces of data are provided by the users, typically, through an iterative and software-guided procedure.
The last operation, *visualizeReceivedData*, is executed when the user wants to visualize the pieces of data that have been received by the app through the *DataRecI* interface.

DD_Q2

First Sequence Diagram: A user wants to extract data produced by all traffic sensors in the last 60 mins.

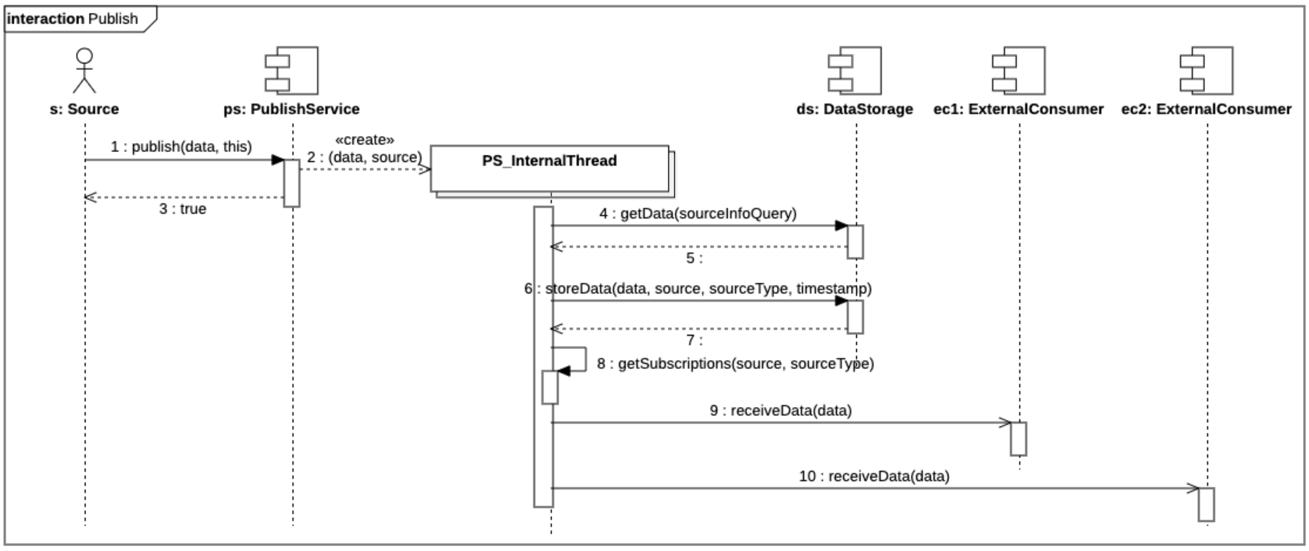

interaction DataExtraction
u: User
app: SmartDaCApp
de: DataExtractor
ds: DataStorage
1 : extractDataBySourceType(trafficSensor, current-3600, current)
2 : extractDataBySourceType(trafficSensor, current-3600, current)
3 : getData(query)
4 :
5 :
6 :

Second Sequence Diagram: A source publishes a piece of data for which two subscriptions are active:



Notice that the `getData` operation retrieves from the `DataStorage` component information about the type of source. This information is then stored as part of the data record and is also used in the `getSubscriptions` operation to identify the subscribers to which data need to be sent.

Notice that, to reduce the time required to handle an invocation of the `publish` operation, hence to allow the system to scale better (see also the answer to question DD_Q3), the `PublishService` component could delegate each interaction with the `DataStorage` and the `ExternalConsumer` components to a separate worker (e.g., a thread), to parallelize the sending of each data, as depicted in the following detailed Sequence Diagram.



DD_Q3

When the number of sources and amount of produced data grows, the component that is most affected is *PublishService*.

To mitigate the problem, we can replicate *PublishService* component and run the replicas on multiple computing nodes. In this case, a specific care should be put in the synchronization of the subscription tables at the various replicas so that every time a *PublishService* replica receives a piece of data, it is able to forward it to its subscribers.

Notice that the *DataStorage* is also affected by the increase of produced data. Several solutions can be used in practice to make the storage more responsive. Some of them are architectural in nature (e.g., distributing data across nodes). Others rely on suitable technological choices. For example, a NoSQL key-value database could be a good choice as it offers a very simple data model that allows for very fast insertion operations. It offers also the advantage of being available in a distributed version, thus enabling the possibility to scale. The main drawback is that the implementation of queries that require joining multiple datasets can become cumbersome. This would have an impact on the data extraction operations. They, however, can be assumed to be performed occasionally as those consumers interested in acquiring data in real time can still use the publish-subscribe approach. Therefore, lower performance in data extraction could be considered acceptable.