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)

January 13th, 2023

Last Name, First Name

Id number (Matricola)

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

Notes

1. Remember to write your name and Id number (matricola) on each piece 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, with the exception of 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: PubCoReader - Cooperative reading and annotating

For a large course like Software Engineering 2 (with multiple instructors) we are asking you to design the **PubCoReader** tool. It allows groups of students to read (or watch – for simplicity, in the following the two terms are used as synonyms) and annotate publications (books, papers, reports, videos…) in a collaborative way. **PubCoReader** should support the creation of multiple groups of readers and the parallel independent reading of the same publication or of different publications.
For instance, at a certain point in time, we can observe the following scenario, where three groups of students are using the system. Group 1 is studying the book titled "Software Architectures" and is currently annotating Chapter 2 by complementing the book content with a picture from the lecture slides and some textual notes. Each group member sees the identity of the group mate that has added an annotation of any kind to the book, as well as the exact position where the annotation has been added. Group 2 is watching the video of the latest lecture. The group members are discussing through the chat implemented as part of **PubCoReader** a point that is not clear to any of them and add the following annotation to the specific point under discussion "@professor: please help us understand this point". Group 3 has run the "translate to audio" feature on a certain paper and is currently listening to the generated audio. Each time a group member stops the audio and inserts an annotation, this is associated with the corresponding part of text in the original paper. Two professors are monitoring the students' activities. One of them notices the @professor message and temporarily joins the corresponding group to discuss the misunderstanding with them.

Part 1 Requirements (7 points)

RASD_Q1 (2 points)
Define the goals for the **PubCoReader** system.

RASD_Q2 (2.5 points)
Select one of the goals defined in point RASD_Q1 and define in natural language suitable domain assumptions and requirements to guarantee that the **PubCoReader** system fulfills the selected goal.

RASD_Q3 (2.5 points)
Draw a UML Use Case Diagram describing the main actors and use cases of the **PubCoReader** system.

Part 2 Design (7 points)

DD_Q1+Q2 (3+2 points)
Assuming you need to implement system **PubCoReader** analyzed above:

1. Identify the most relevant components and interfaces and describe them through a UML Component diagram.
2. Provide a brief description of each identified component.
3. For each component, list the operations it provides through its interfaces.

NB: 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.

DD_Q3 (2 points)
Describe through one or more UML Sequence Diagrams the interactions that occur between the defined **PubCoReader** components when, referring to the scenario description above, a Group 2 member sends the @professor message, an active professor is notified and joins the group.

Solutions

RASD_Q1

G1: Students can create groups that independently work in collaboration (annotate and discuss) on publications (which can be multimedia artifacts).
G2: Students and teachers can interact through discussion groups.

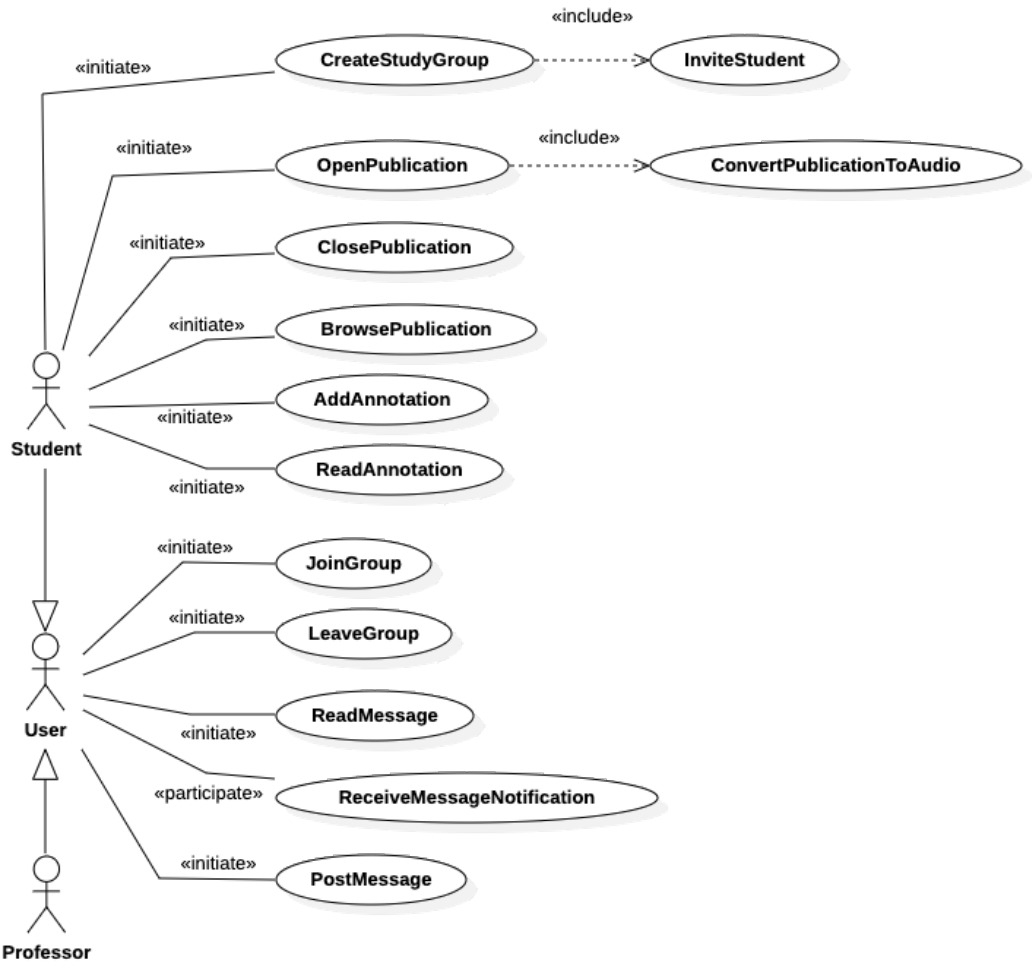
RASD_Q2

(For completeness' sake, the solution lists requirements and assumptions for each goal identified in question RASD_Q1; however, listing only the requirements and domain assumptions for one goal was enough to answer this question. Also, notice that requirements and assumptions that are related to both goals are duplicated to make each list self-contained. For the same reason, duplicated requirements are renumbered based on the goal under which they are listed.)

G1: Students can create groups that independently work in collaboration (annotate and discuss) on publications (which can be multimedia artifacts).
R1.1: *PubCoReader* allows students to create study groups
R1.2: *PubCoReader* allows students to join/leave study groups
R1.3: *PubCoReader* allows students to invite other students to study groups
R1.4: *PubCoReader* allows students in a study group to open one publication at a time
R1.5: *PubCoReader* allows students in a study group to close an open publication
R1.6: *PubCoReader* allows students in a study group to browse the open publication
R1.7: *PubCoReader* allows students in a study group to add an annotation to the open publication (the annotation is tagged with the name of the student who made it)
R1.8: *PubCoReader* allows students in a study group to post messages in the group chat
R1.9: *PubCoReader* notifies students in a group when a new message is posted in the group chat
R1.11: *PubCoReader* allows students to convert textual publications into audio format
A1.1: Students are registered in the system by school administrators when they enroll

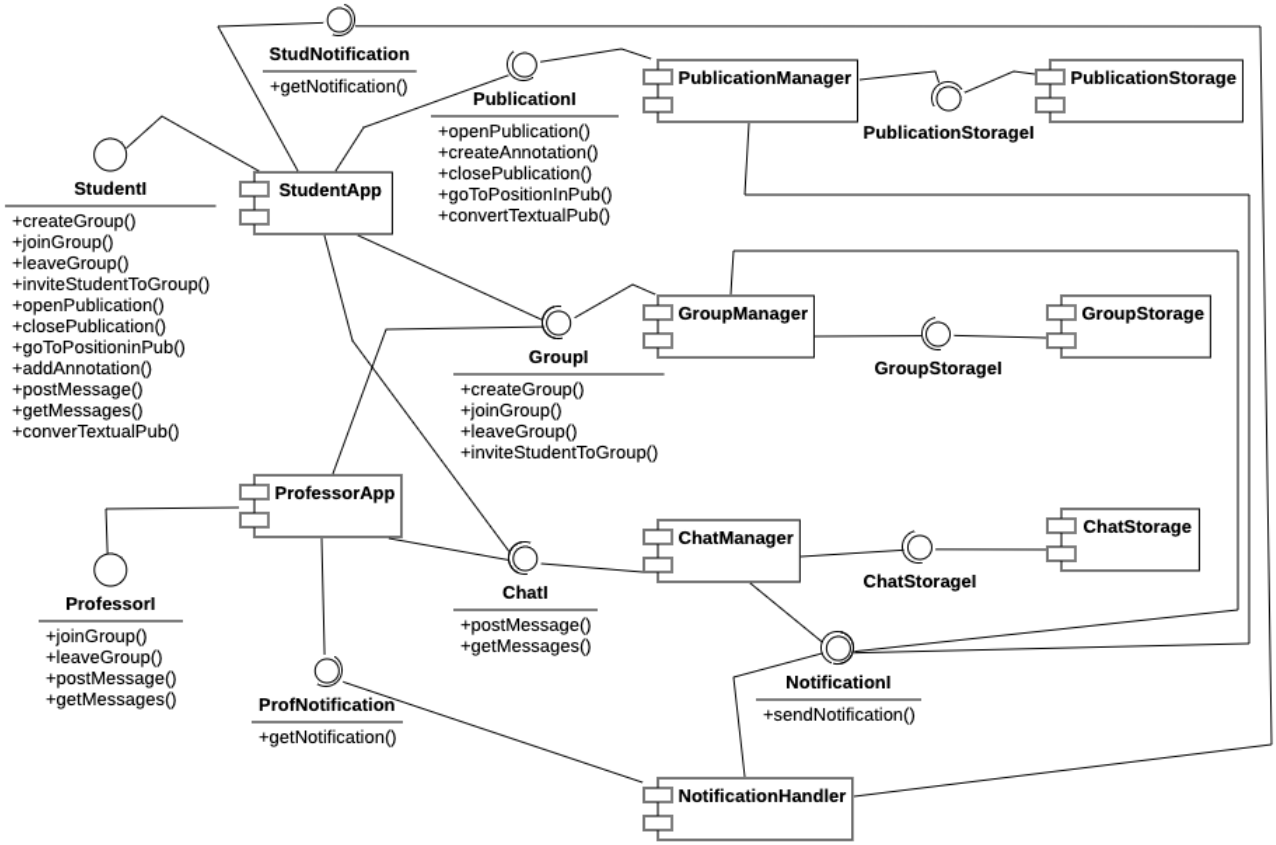
G2: Students and teachers can interact through discussion groups.
R2.1: *PubCoReader* allows students to create study groups
R2.2: *PubCoReader* allows students to join/leave study groups
R2.3: *PubCoReader* allows students to invite other students to study groups
R2.4: *PubCoReader* allows students in a study group to open one publication at a time
R2.5: *PubCoReader* allows students in a study group to close an open publication
R2.6: *PubCoReader* allows students in a study group to browse the open publication
R2.7: *PubCoReader* allows students in a study group to add an annotation to the open publication (the annotation is tagged with the name of the student who made it)
R2.8: *PubCoReader* allows students to tag professors in their annotations, to indicate that the annotation requires their attention
R2.9: *PubCoReader* notifies professors when they are tagged in students' annotations
R2.10: *PubCoReader* allows professors to join and leave students' study groups
R2.11: *PubCoReader* allows professors and students to post messages in the group chat
A1.1: Students are registered in the system by school administrators when they enroll
A2.1: Professors are registered in the system by school administrators when they start teaching at the school

RASD_Q3



DD_Q1+Q2

The system includes two applications as front-ends, one to be used by students (*StudentApp*) and one to be used by professors (*ProfessorApp*). Each application provides its users with the operations that the corresponding group of users can perform in the system (e.g., create groups for students, or join groups for both kinds of users). Each application interacts with several server-side modules. In particular, the *GroupManager* allows users (both students and professors) to realize the group operations (e.g., create group, join group, etc.); the *PublicationManager* component allows students to handle publications (open and close them, add annotations to them, etc.). The *ChatManager* allows both students and professors to interact with chats (post and read messages); finally, the *NotificationHandler* sends to the client applications (both the students' and the professors' one) notifications (e.g., regarding tags). Each component handling relevant data has its own storage (the interface of each storage allows the client modules to create/read/update/delete data). Indeed, this architecture has a microservices-oriented flavor. Naturally, different solutions would have been possible.

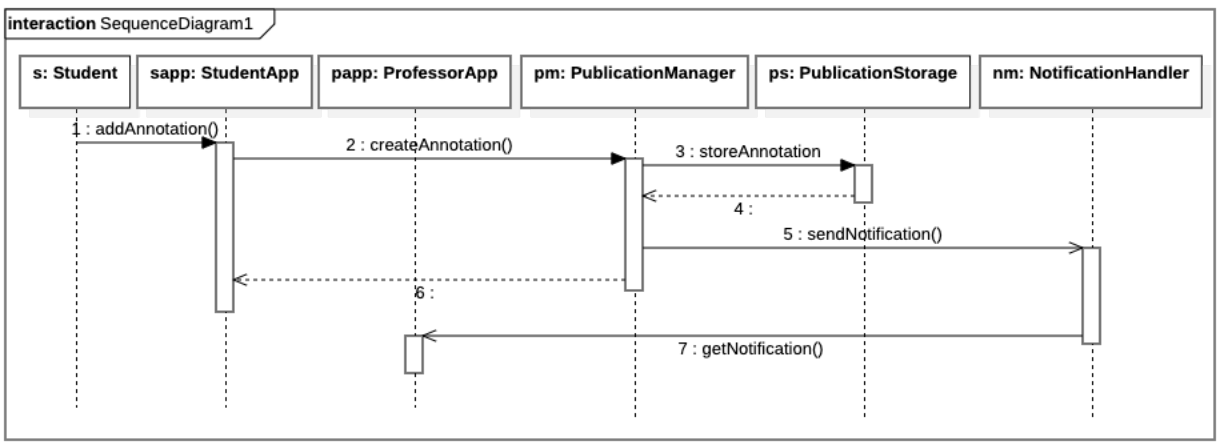


Notice that the *PublicationManager* component in this design handles all kinds of publications (books, articles, videos, etc.). In particular, operation *goToPositionInPub* is used to move to a point in the publication (could be a page in the case of a book, or a minute in the case of a video). One could have added other operations (or possibly also a separate component) to handle specific kinds of publications (e.g., operations to start/stop reproducing a video).

DD_Q3

For clarity, we separate the interaction in two diagrams, where the secondo interaction clearly follows the first.

The first part of the interaction concerns the students creating the annotation to ask the professor for help (and the professor receiving the corresponding notification):



The second part of the interaction concerns the professor joining the group to answer the request (notice that the actual response is not depicted, as the text of the exercise stopped at the point where the professor joined the group):