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)

February 5th, 2021

Last Name
First Name
Id number (Matricola)

Notes

This exam is handled online. Rules

1. Only use a computer, NOT a tablet, NOR a smartphone
2. Activate the feed of your webcam
3. Share the screen of your computer
4. Keep the microphone on
5. No dual screens
6. No virtual machines
7. When you upload a file through the form, make sure to include your "person id" (the 8-digit number that starts with "10") in the name of the file, AT THE BEGINNING OF THE NAME (so the name of the file should be, say, "10143828_etc.", if your person id is "10143828").
8. The exam is open book, so you can check the course materials (notes, slides, books, past exams, etc.), which can be in paper form, or in electronic form. If the materials are in electronic form, you MUST use the same computer on which you are taking the exam to display them.
9. You cannot interact with other people during the exam.
10. **Total available time: 1h and 30 mins**

Scores of each question:

Part 1 (MAX 6) _____

Part 2 (MAX 8) _____

System Description: VirtualBookClub

We want to build an application, *VirtualBookClub*, that creates communities of book readers who share books and discuss them.
The system allows for the creation of "neighborhood communities" of people living nearby, who share hardcopies of books. A user can set up a virtual community, which entails creating the physical space (e.g., a bookshelf) where books are stored when they are available. Access to the community can be free, or regulated by the community administrator (e.g., the user who created the community). Access to the community can also be granted on an area basis; that is, only users who live in a certain area can join the community (subject to the approval of the administrator).
Users can make books available to their community (by putting them on the shelf), and they can borrow available books, much like in a library. Whenever is book is made available/borrowed/returned, users signal this through the system. Users can also take a book from the library permanently; in this case, unless they own the book, so they are simply removing it from the sharing, they must also, at the same time, "release" to the community another book that they own (not necessarily another copy of the book that they took, it could be a totally separate book).
Users who have read a book can start or participate to a discussion on the book. There can be different discussions ongoing on a book, on different topics. A reader can decide which discussions are more interesting for them and participate only to them.

Part 1 Requirements (6 points)

Q1 (1 point)
With reference to the Jackson-Zave distinction between the world and the machine, identify the relevant world and machine phenomena for *VirtualBookClub*, 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.

Q2 (2 points)
Referring to the phenomena identified above, define in natural language one specific goal for *VirtualBookClub*, together with the associated domain assumptions and requirements. Explain why the identified requirements and assumptions are relevant to the fulfillment of the goal.

Q3 (2 points)
Define a UML Use Case Diagram describing the relevant actors and use cases.

Q4 (1 point)
Identify which is, according to you, the most relevant use case among those identified in the Use Case Diagram; define the use case using the structure seen in class.

Solution

Q1

World phenomena:
User sets up bookshelf for community
User takes book from bookshelf
User puts book back in bookshelf

Shared phenomena, world-controlled
User creates virtual community
User requests to join virtual community
Administrator accepts/rejects request to join community
User adds book

User borrows/returns book
User starts online discussion
User posts comment
User subscribes to/unsubscribes from discussion
User lists available books

Shared phenomena machine-controlled
System returns list of available books
System notifies user of new post on subscribed discussion

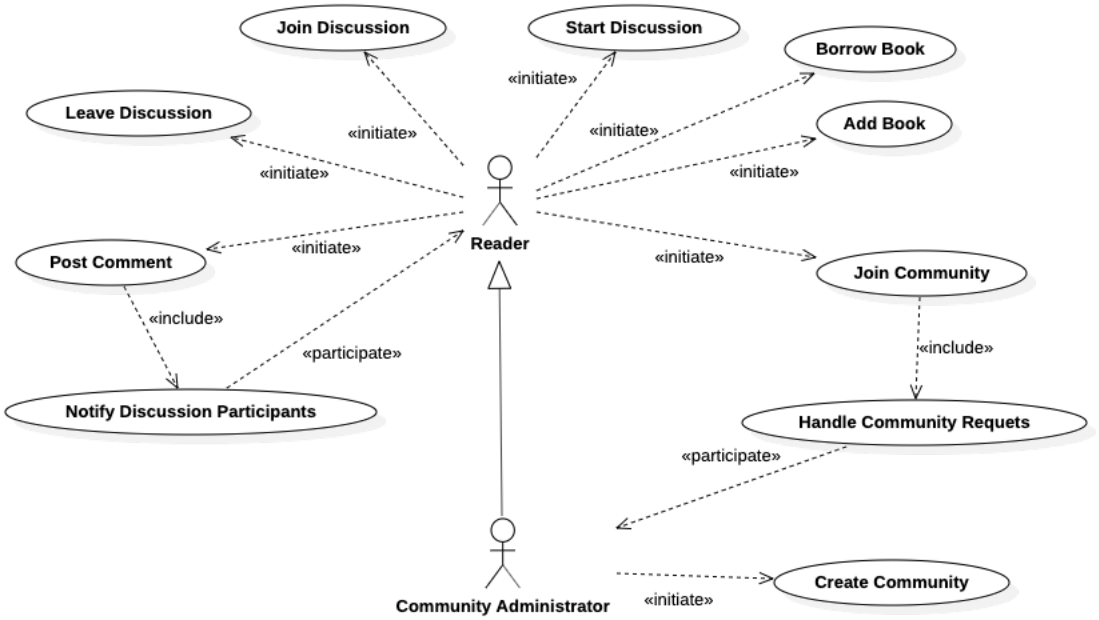
Q2
Goal
G1: Allow readers to discuss books they have read with people in their book club.

Requirements
R1: The system must allow users to indicate that they take a book from the shelf
R2: The system must allow users to indicate that they return a book to the shelf
R3: The system must allow users to post comment on book they have read
R4: The system must allow users to subscribe to the discussion on a book they have read
R5: The system must allow users to unsubscribe to the discussion on a book they have read
R6: The system must allow users to be notified of a new comment on a book whose discussion they are following
R7: The system must allow users to request to create a community
R8: The system must allow users to request to join a community
R9: The system must allow community administrators accept/reject a request to join the community they are administering

Domain assumptions
DA1: users actually take/put back books when they say they do so through the system
DA2: users actually set up bookshelves when they start a community
DA3: users have actually read the book when they return it
DA4: users live where they say they live

Assumptions DA1 and DA2 make sure that the books that are present in the bookshelf are known by the system (thanks to R1 and R2), so users can actually borrow and read them. DA3 ensures that the action of having returned a book (R2) corresponds to having read it. Hence, R4-R6 ensure that users participate to discussions on books they have read with their community.
R7-R9 ensure the building of the community, which, by DA4, is the right one.

Q3



Q4

Use case name: Post comment
Participating actors: Reader
Entry condition: Reader has a comment to leave on book they have read
Flow of events:

1. Reader inserts comment on book
2. System checks that Reader had indeed read book
3. System posts comment in discussion
4. System checks what Readers have subscribed to the discussion on book
5. System notifies selected Readers of the new comment

Exit condition: Readers who have read the book and who are interested in discussing it receive the new comment
Exceptions: If the Reader has not reads the book, the System will notify them that the comment cannot be posted

Part 2 Design (8 points)

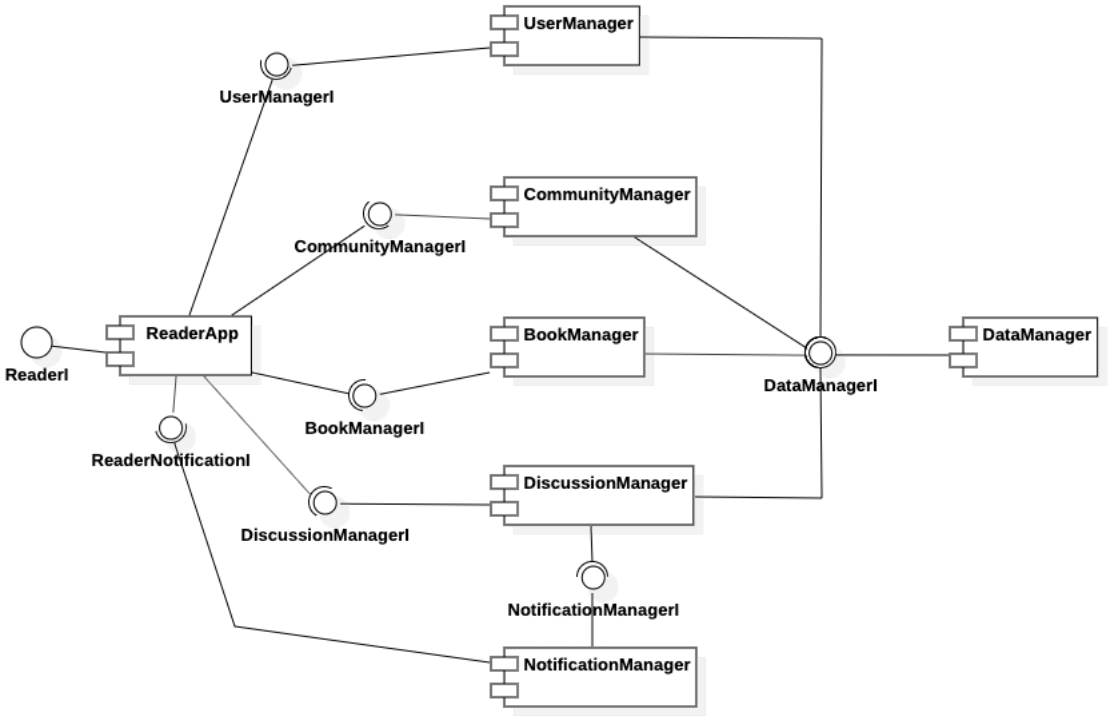
Q5 (3 points)
Assuming you need to implement system *VirtualBookClub* analyzed above, identify the most relevant components and interfaces describing them through UML Component or Class Diagrams.
Provide a brief description of each component.

Q6 (3 points)
For each interface identified in Q5, list the operations that it provides.
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 they use/produce.

Q7 (2 points)
Define a runtime-level Sequence Diagram describing the interaction among the components for what concerns the use case described in Q4.
If useful (optional), provide a brief description of the defined Sequence Diagram.

Solution

Q5-Q6



ReaderApp
This is the front-end both for Readers and for Administrators. It allows users to interact with the system by offering the following functions through interface *ReaderI*:

- Register (input: user data, including address, which is crucial for joining community)
- Login
- Set up a community (input: name, location of community)
- Request to join a community (input: user data, community to join)
- Request handling (accept/reject requests to join community)
- Add a book (input: data about the book, including an identifier, and the community)
- Search for books in community (input: search parameters)
- Borrow book (input: book id)
- Return book (input: book id)
- Take book permanently (input: id of book to take and of book to leave)
- Create discussion (input: book id)
- Subscribe/unsubscribe to discussion (input: book id)
- Post comment (input: book id, comment)
- Visualize received comment
- Visualize acceptance/rejection of community join requests

In addition, it provides interface *ReaderNotificationI*, through which it can receive notification messages sent by the *NotificationManager*.

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

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

Community Manager
This component provides interface *CommunityManagerI*, which handles functions related to the management of communities:

- Create community (input: administrator user id, community info)
- Receive join request (input: requesting user id, community id)
- Visualize join requests (assuming the Administrator logs in regularly, and checks the outstanding requests) (input: community id)
- Accept/reject requests (input: requesting user id, community id)

BookManager
This component provides, through interface *BookManagerI*, functions related to the management of books:

- Add book (input: user id, book id)
- Borrow book (input: user id, book id)
- Take book permanently (input: user id, taken book id, left book id)
- Return book (input: user id, book id)

DiscussionManager
This component provides, through interface *DiscussionManagerI*, functions related to the management of comments:

- Create discussion (input: creating user id, book id)
- Subscribe/unsubscribe to discussion (input: subscribing user id, book id)
- Post comment on discussion (input: posting user id, book id, comment)

This component keeps track of who follows discussion, and it invokes the *NotificationManager* component (through the required interface *NotificationManagerI*) when a new post is created to distribute the comment to subscribing users.

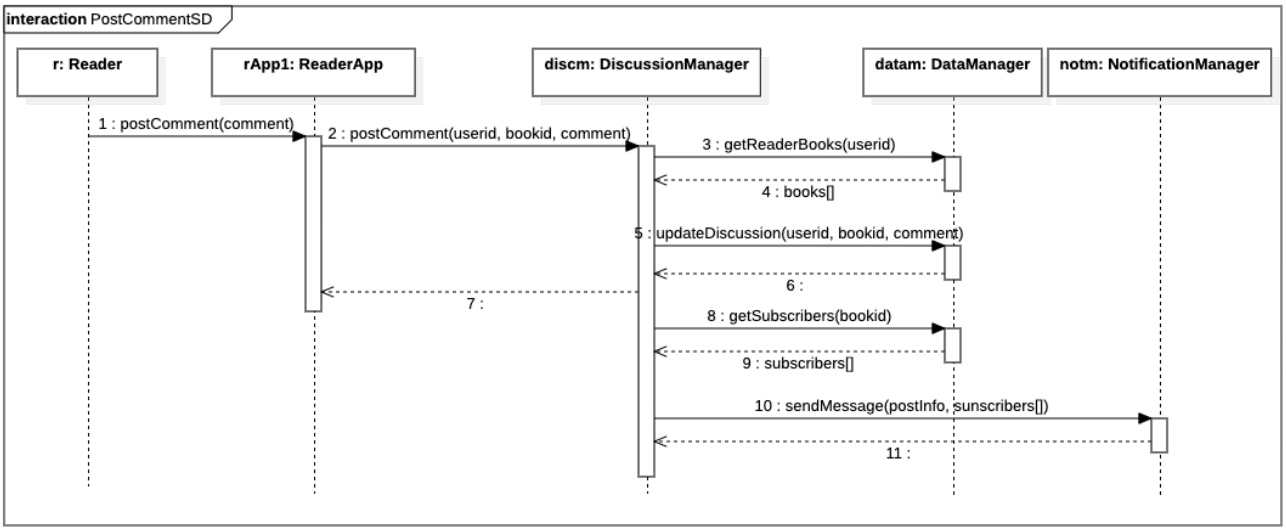
NotificationManager
This component handles the notification to users, in particular of new posts. It provides the following function through interface *NotificationManagerI*:

- Send message (input: message to be sent, list of recipients)

In addition, it uses interface *ReaderNotificationI* provided by *ReaderApp* to deliver messages to readers.

Data Manager
This component handles the data of the system: Users, Communities, Book, Discussions (including subscriptions). It provides interface *DataManagerI*, which includes all necessary functions to handle CRUD operations on data.
The only backend component that does not interact with *DataManager* is *NotificationManager*, which receives all necessary information to handle notifications from *DiscussionManager*.

Q7


interaction PostCommentSD
r: Reader
rApp1: ReaderApp
discm: DiscussionManager
datam: DataManager
notm: NotificationManager
1 : postComment(comment)
2 : postComment(userid, bookid, comment)
3 : getReaderBooks(userid)
4 : books[]
5 : updateDiscussion(userid, bookid, comment)
6 :
7 :
8 : getSubscribers(bookid)
9 : subscribers[]
10 : sendMessage(postInfo, sunscribers[])
11 :