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)

July 14th, 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: TMT - TennisMatchTracker

TennisMatchTracker (TMT) is a system that keeps track of scores of tennis matches. Every time that the chair umpire (i.e., the referee) of a tennis match updates the score of the match because a player won a point, the information is collected and stored by the system. In addition, the system gathers information from speed detectors that measure the speed of the serves hit for each point (each point starts with a serve; if the first serve is out, a second serve is hit; if a serve hits the net and stays in, it is repeated; if both the first and the second serve are out, the player who served loses the point through a "double fault"). The system also relies on human monitors, who can tag each point with information about how the point was won ("ace", "service winner", "forehand winner", "double fault", etc.), and about the serves in each point (e.g., "first serve out", "net first serve"); there is a standardized, finite set of tags that can be applied by human monitors. The information gathered by the system is made available to users, who can follow the status of matches in real time. Users can also indicate to TMT that they want to be notified when, after a point, certain matches (e.g., their favorite matches or the matches of their favorite players) they are interested in reach a crucial event (e.g., when a player is serving for the set, when there is set point, when there is match point).

Part 1 Requirements (7 points)

RASD_Q1 (2 points)
Define suitable world and machine phenomena for the TMT system.

RASD_Q3 (2.5 points)
Consider the following goal defined for the TMT system:

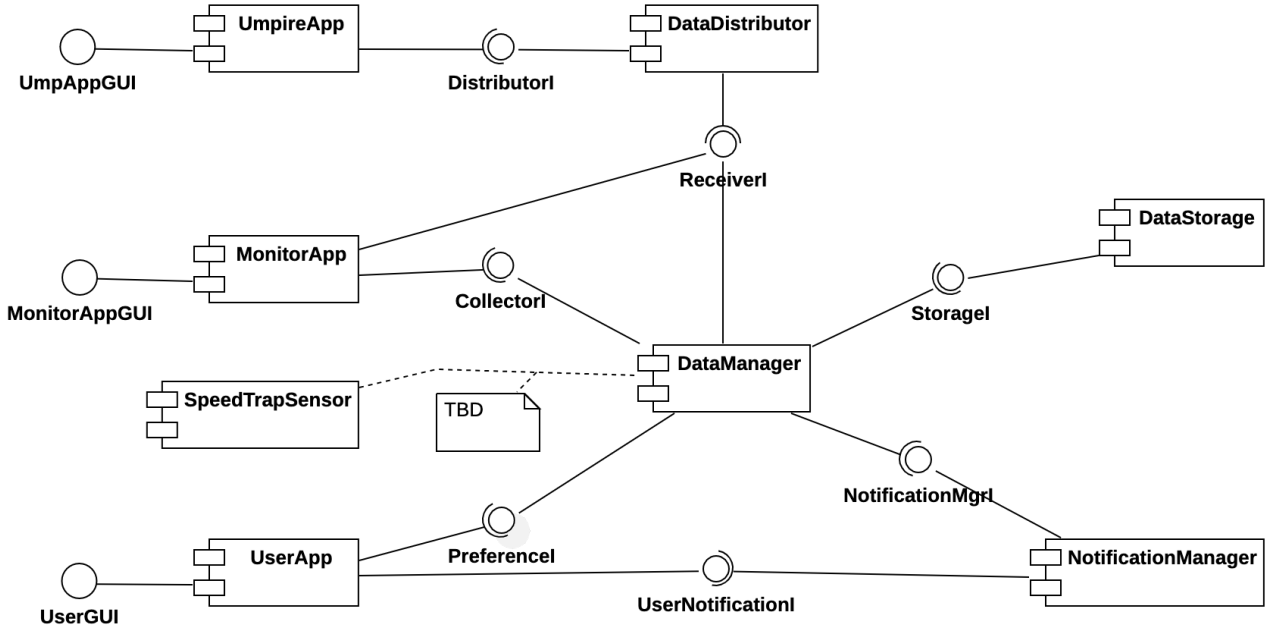
> **G1**: *Users want to be informed live when matches of their favorite players reach crucial events such as set point, match point, etc.*

Define in natural language suitable domain assumptions and requirements to guarantee that the TMT system fulfills goal G1.

RASD_Q3 (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 TMT system.

Part 2 Design (7 points)

Consider the following UML Component Diagram, which describes a possible architecture for the TMT system.



The system includes three front-end applications: one (*UmpireApp*) for chair umpires, which allows them (through the *UmpAppGUI* interface) to indicate that a new point has been won by a player; one (*MonitorApp*) for human monitors, which allows them (through *MonitorAppGUI*) to tag points; and one (*UserApp*) for general users, which allows them (through *UserGUI*) to view the results of tennis matches (both ongoing and already completed), indicate their preferred players, matches, etc. In addition, the *SpeedTrapSensor* component detects the speed of serves. The information about points inserted by chair umpires is sent (through the *DistributorI* interface) to a *DataDistributor* component, which uses the *ReceiverI* interface to notify both *MonitorApp* components and the *DataManager* component when a new point has been won (you can assume that the *DataDistributor* component is initialized with the

information about the components that need to be notified of the completion of a point). The tags added by human monitors are sent to the *DataManager* component through the *CollectorI* interface. The interaction and interface between *SpeedTrapSensor* and *DataManager*, on the other hand, is to be determined (see point DD_Q3). *DataManager* uses (through interface *StorageI*) the *DataStorage* component to persist information about points, tags, etc. *UserApp* uses interface *PreferenceI* to set user preferences (e.g., the matches to be followed, the favorite players). Finally, *DataManager* uses the *NotificationMgrI* interface of *NotificationManager* to distribute information about points (when a point is won, when tags are added, etc.) and crucial events (set points, match points, etc.); this information is relayed by *NotificationManager* to users through the *UserNotificationI* interface. Notice that *NotificationManager* also keeps track of the users who must be notified of the various events. To do so, *DataManager* uses interface *NotificationMgrI* also to inform *NotificationManager* when a user changes one of its notification preferences.

DD_Q1 (3 points)
Analyze the operations offered by the interfaces shown in the diagram and identify proper input and output parameters for each of them. You can skip the *StorageI* that exposes the usual CRUD (Create, Read, Update, Delete) operations. Additionally, skip the operations related to the interaction between *SpeedTrapSensor* and *DataManager,* as they are to be detailed in question DD_Q3. 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 a suitable UML Sequence Diagram illustrating the interactions that occur among the software components when the umpire signals that a new point has been won, in the case in which the new point leads to a crucial event, and there is a user who asked to be notified of relevant events of the match.

DD_Q3 (2 points)
Consider the interaction between *DataManager* and *SpeedTrapSensor* that occurs to retrieve the detected serve speeds. Consider that *SpeedTrapSensor* only stores the last 5 serve speeds, and it does not know when a new point is played, only when a new serve occurs (notice that there can be multiple serves for each point, and their number can vary; for example, in a point in which the first serve is good there is only one serve, whereas for a point in which the first serve touches the net, hence is repeated, the repetition of the first serve is out, and the second serve is in, there are three serves). Discuss and motivate – possibly with the help of a UML Sequence Diagram – what strategies the *DataManager* can use to retrieve the information about the speed, and define a suitable interface (and related operation(s)) between the two components.

Solutions

RASD_Q1

World phenomena:

(W1) Player serves.
(W2) Players play a point.
(W3) Speed serve is detected.
(W4) Serve hits the net
(W5) Serve is out

Shared phenomena controlled by world:

(SW1) Chair umpire inserts point.
(SW2) Human monitor tags point.
(SW3) User selects match to be followed.
(SW4) User selects match for which notifications are to be received.
(SW5) User selects player for whose matches notifications are to be received.

Note that in this solution we are assuming that users cannot select specific crucial events they are interested in, but only preferred matches and players. The possibility to specify particular crucial events could be seen as an extension.

Shared phenomena controlled by machine:

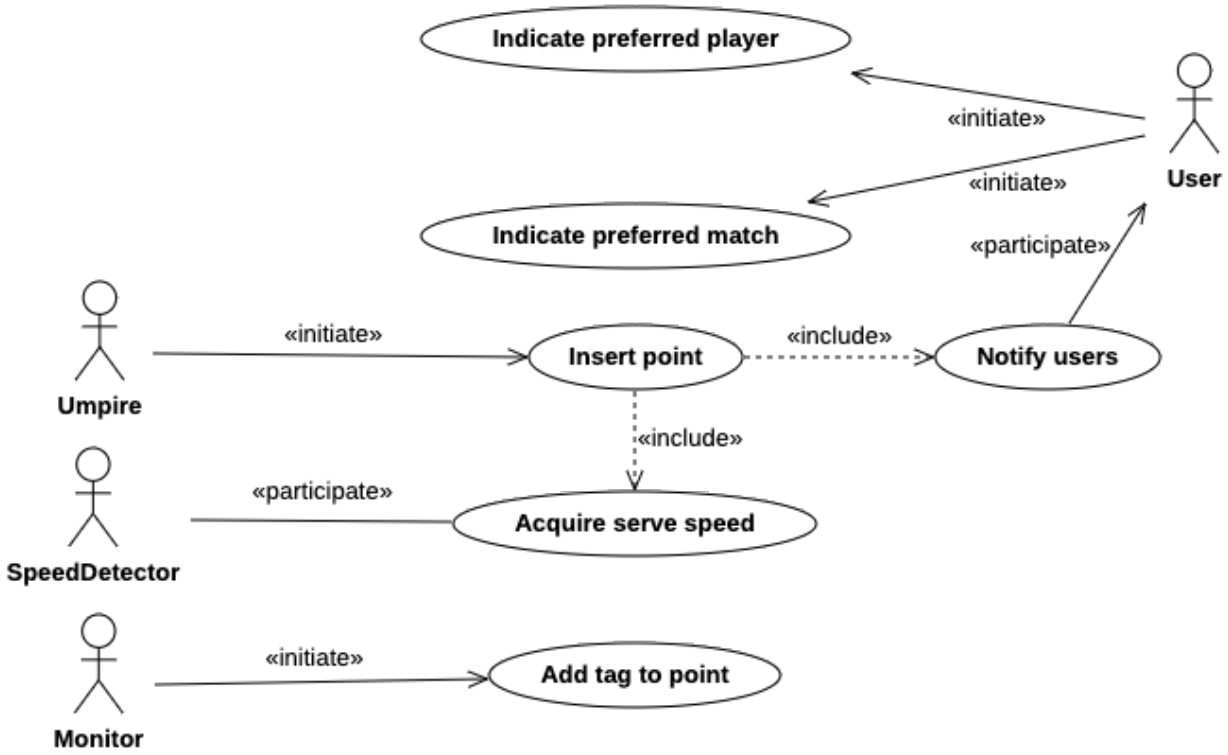
(SM1) System updates the status of match followed by user.
(SM2) System notifies user of crucial event.
(SM3) System gathers serve speed from speed detector.

RASD_Q2

R1. The system must allow chair umpires to input when a player wins a point.
R2. The system must allow human monitors to tag points of matches.
R3. The system must collect information from speed detectors regarding the speed of serves.
R4. The system must allow users to indicate which matches they want to follow live.
R5. The system must allow users to indicate the matches for which they want to receive notifications of relevant points.
R6. The system must allow users to indicate the players for whose matches they want to receive notifications of relevant points.
R7. When a new point is completed, the system must notify all users who are following the match live.
R8 When a new relevant point is completed, the system must notify all users who selected one of the players as one for which they want to be notified when he/she plays relevant points.

A1. Chair umpires input the correct information regarding points.
A2. Human monitors add the correct tags to points.
A3. Speed detectors detect the correct speeds of serves.
A4. A notification reaches the user before the next point is completed.

RASD_Q3



DD_Q1

UmpAppGUI offers the following operation:

```
void newPoint(MatchID mid, PlayerID, pid)
```

where `MatchID` and `PlayerID` are types that identify matches and players, respectively.

DistributorI offers the following operation:

```
void addNewPoint(Point p)
```

where `Point` represents the information related to a point (including who won it, the match in which it is played, the timestamp of when the point occurred).

ReceiverI offers the following operation:

```
void addNewPoint(Point p)
```

MonitorAppGUI offers the following operations:

```
void selectMatch(MatchID mid)
void tagPoint(Point p, Tag t)
```

where `Tag` is the enumeration listing the possible tags. Notice that operation `selectMatch` allows human monitors to select the match whose points they want to tag from now on.

CollectorI offers the following operations:

```
void updatePoint(Point p)
Point[] retrievePoints(MatchID mid)
```

Note that paremeter `p` of operation `updatePoint` contains the information about the point, amended with the corresponding tag.

PreferenceI offers the following operations:

```
void setPreferredPlayer(PlayerID pid, User u)
```

```
void setPreferredMatch(MatchID mid, User u)
void setLiveMatch(MatchID mid, User u)
void unsetPreferredPlayer(PlayerID pid, User u)
void unsetPreferredMatch(MatchID mid, User u)
void unsetLiveMatch(MatchID mid, User u)
```

where the `setPreferredX` operations are used to indicate the matches for which the user wants to be notified when relevant points occur, and `setLiveMatch` is used to indicate the match that the user is following live. The `unset` operations are used to indicate that the user no longer wants to be notified of points for those matches. `User` represents the information of the user who wants to be notified.

NotificationMgrI offers the following operations:

```
void notifyPoint(Point p)
void notifyRelevantPoint(Point p, EventType e)
void playerSubscribe (PlayerID mid, User u)
void liveMatchSubscribe (MatchID mid, User u)
void relevantPointMatchSubscribe (MatchID mid, User u)
void playerUnsubscribe (PlayerID mid, User u)
void liveMatchUnsubscribe (MatchID mid, User u)
void relevantPointMatchUnsubscribe (MatchID mid, User u)
```

where the `EventType` captures the kind of event that needs to be notified (e.g., set point, match point, etc.) as a consequence of point `p` having been played.

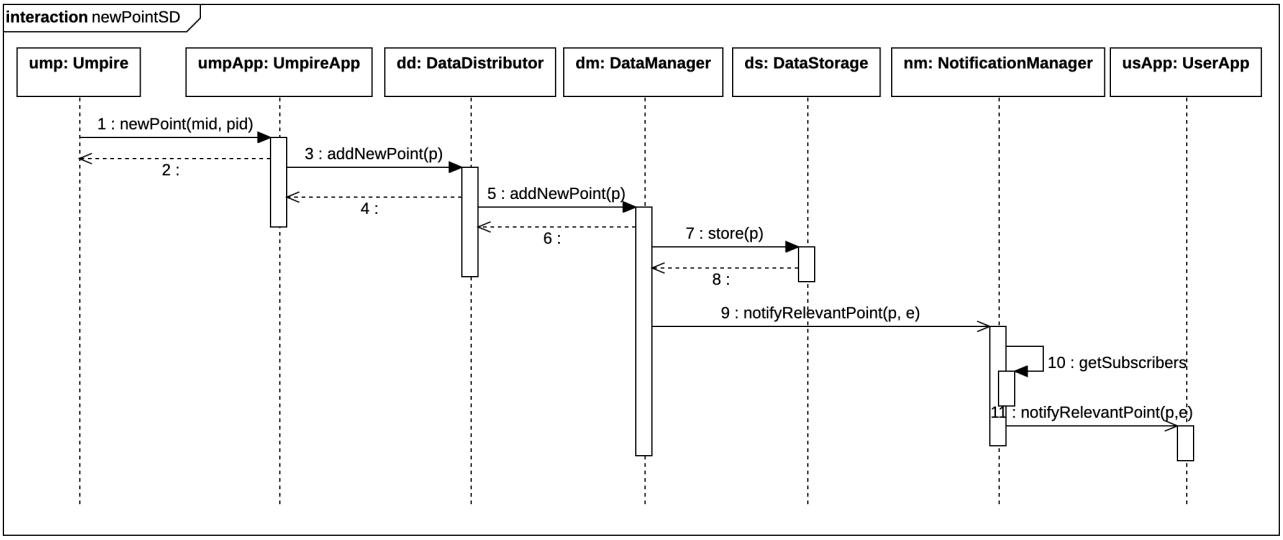
UserNotificationI offers the following operations:

```
void notifyPoint(Point p)
void notifyRelevantPoint(Point p, EventType e)
```

UserGUI offers the following operations*:*

```
void setPreferredPlayer(PlayerID pid)
void setPreferredMatch(MatchID mid)
void setLiveMatch(MatchID mid)
void unsetPreferredPlayer(PlayerID pid)
void unsetPreferredMatch(MatchID mid)
void unsetLiveMatch(MatchID mid)
```

DD_Q2



DD_Q3

To retrieve the serve speed there two main possible strategies: a "pull" strategy, in which the *DataManager* fetches the speed from the detector when a new point is completed, and a "push" strategy, in which *SpeedTrapSensor* sends the value of the speed to the *DataManager* when it detects a new one. In the first case, *SpeedTrapSensor* should provide an interface that allows *DataManager* to fetch the speed. In the second case, *DataManager* should offer an interface, to be used by *SpeedTrapSensor* to send the value to *DataManager*. Given that it is the *DataManager* that knows when a speed is needed (i.e., when a point is completed), in this case we opt for a "pull" strategy, as represented by the following snippet of Component Diagram and related Sequence Diagram.