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 1 (WE1)

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. The exam is composed of three exercises. Read carefully all points in the text!
11. **Total available time: 1h and 30 mins**

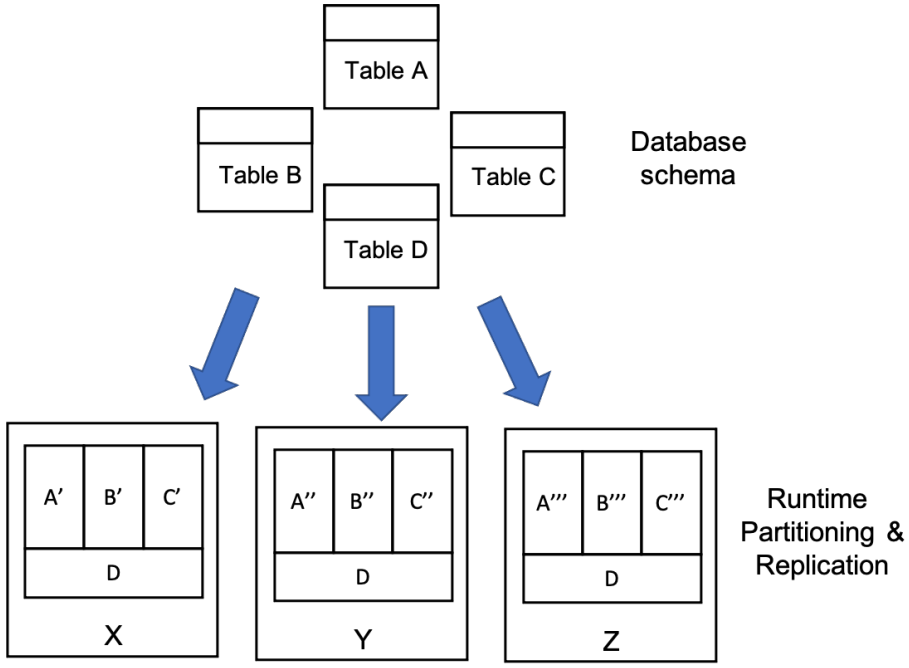
Scores of each question:

Question 1 (MAX 7) _____

Question 2 (MAX 6) _____

Question 3 (MAX 3) _____

Question 1 Alloy (7 points)
Consider the following figure that describes the status of a replicated and partitioned database management system (DBMS). The figure shows that database tables are deployed on three "nodes". More specifically, in the example, tables A, B, and C are partitioned and each partition is deployed on a different node (for instance, the whole table A is composed of A', A'' and A''', where A', A'' and A''' are sets of disjoint tuples belonging to table A). Table D, instead, is replicated on all subsystems, which means that all its tuples occur in all nodes.



Point A (4 points).
Define in Alloy the signatures to represent the concepts of node, tuple, replicated tables, and partitioned tables, together with any other concept you may need.

Point B (1.5 points).
Define a fact to ensure that each replicated table is deployed on at least two nodes.

Point C (1.5 points).
Define a fact to ensure that each partitioned table does not contain tuples that are replicated on multiple nodes.

Solution

Point A

```
sig Tuple {}

sig Node {}

abstract sig Table {
  tuples : set Tuple
}
```

```
sig Partition extends Table {
  node : Node
}

sig PartitionedTable extends Table {
  parts: set Partition
}{
  // in case we wanted to add that a partitioned table must have more than one partition,
  // this would be the constraint
  #parts > 1
}

sig ReplicatedTable extends Table{
  nodes: set Node
}{
  // this solves point B
  #nodes > 1
}

// this guarantees that the partitions of a Table include all tuples of the latter, and that a
// tuple can only appear in a partition
fact partitionsHaveAllTuples {
  (all pt: PartitionedTable, t : Tuple | t in pt.tuples
                                         implies
                                         (one p : Partition | p in pt.parts and t in p.tuples))
  (all pt: PartitionedTable, p : Partition, t : Tuple | p in pt.parts and t in p.tuples
                                                        implies
                                                        t in pt.tuples)
}

// the text is a bit ambiguous on whether a node can host multiple partitions of the same table;
// the constraint above leaves open the possibility that different partitions of the same table
// are hosted on the same node
// if this is forbidden, the following constraint should be added

fact differentPartitionsOnDifferentNodes {
  (all pt: PartitionedTable, disj p1, p2 : Partition | p1 in pt.parts and p2 in pt.parts
                                                       implies
                                                       p1.node & p2.node = none )
}
```

Point B

This has already been covered by the constraint within signature ReplicatedTable.

Point C

```
fact noDuplicateTuplesInNodes {
  all pt: PartitionedTable, disj p1, p2: Partition |
       p1 in pt.parts  and  p2 in pt.parts  and  p1.node & p2.node = none
       implies
       p1.tuples & p2.tuples = none
}
```

// if different partitions of the same table must be on different nodes (i.e., if we introduce
// constraint differentPartitionsOnDifferentNodes), then the required constraint is
// already guaranteed, and we do not need to introduce constraint noDuplicateTuplesInNodes

Question 2 JEE (6 points)
The Municipality of Milan wants to create a web-based system to let parents register their children in a public kindergarten in various regions of the city. The Municipality would like to have a robust architecture to manage this system.
Parents use the system to enter an Enrollment Request (ER) where they specify the information about their child, including the area they live in.
Kindergartens receive through the system the ER, evaluate it and contact the applicants to inform them if they are rejected/accepted and to further communicate with them.

In particular, the system offers the following functions to parents:

- F1 - Child registration: The parent should be able to register his/her child in the system by giving information such as name/surname, birthdate of the child, working status, phone number, and e-mail address of the parents. As a result of the registration, parents will be able to log in to the system.
- F2 - Enrollment request: When logged in, parents issue the ER by giving the following data in addition to the data in their profile: city area of interest, and if the child needs any special medical care.
- F3 - Kindergarten rating: Parents can rate the kindergarten of their child.
- F4 - Visualization of ER status: This can be one of the following: pending or accepted (for simplicity, we assume that there is enough availability for all children of a certain area, so no application is rejected).

Moreover, the system offers the following functions to kindergartens' staff:

- F5 - Kindergarten registration: This is performed by providing at least the following data: kindergarten name, city area, email, capacity.
- F6 - Profile update: Kindergartens can update their capacity. They will stop receiving ERs whenever their capacity is full.
- F7 - ER evaluation: The kindergarten's staff can accept/reject an application.

Finally, the system offers the following support functions:

- F8 - Upon submission, the system sends ERs to all available kindergartens of the desired city area. Notice that a parent cannot apply for a specific kindergarten, but only for a region.

Notice that when one of the kindergartens accepts an ER, then the ER status is changed to accepted. Also, the system keeps track of who is enrolled in each kindergarten, based on the acceptance information.

A. (2 points)
Identify at least two different Components of the **Web Tier**, their main **responsibility** and, for each Component, list the requirements that it satisfies.

B. (3 points)
Given the entities defined in the following table, focus on the business layer part of your system, and in particular on the definition of the Beans. More precisely, identify the required Beans, specify their types and, for each Bean, list the functions it satisfies (F1-F8) and at least one method (the most important one).

Entity	Attributes	Relationship With	Multiplicity
Child	String userID, *@Id, @GeneratedValue* String userName, *@NotNull* String password, *@NotNull* String childName, *@NotNull* String parentName, *@NotNull* Date birthDate, *@NotNull* String residenceAddress, *@NotNull* ---------- ER Collection/Set< ER >, *@ManyToMany*	ER	*ManyToMany*
Kindergarten	String kindergartenID, *@Id, @GeneratedValue* String kindergartenName, *@NotNull* String password, *@NotNull* Integer regionID, *@NotNull* Boolean availability, *@NotNull* Integer overallRate ---------- ER Collection/Set< ER >, @ManyToMany	ER	*ManyToMany*
ER	String er_ID, *@Id, @GeneratedValue* String userName, *@NotNull* Boolean specialCareRequired, String specialCare, ---------- Kindergarten Collection/Set< kindergarten>, @ManyToMany Child Collection/Set< Child >, @ManyToMany	Child Kindergarten	*ManyToMany* *ManyToMany*

C. (1 point)
Reflect on the possibility to adopt a Message-Based Communication model. Which part of the business layer is most suited to be addressed through such model? What is the JEE API to use in this case? Explain how you would incorporate such JEE API in your system.

Solution

Point A:
The web tier of the application is composed of various **JSPs** containing HTML forms and **Servlets** to manage them. For example, below are listed some of the needed components:

Table 1 Web Tier Elements

Web Tier		
Component	Responsibility	Requirement
JSP	Front-end for Child Registration	F1
JSP	Front-end for kindergarten Registration	F5
JSP	Front-end for ER form	F8
ChildRegistration_Servlet	To forward the forms data to corresponding Bean	F5
KindergartenRegistration_Servlet	To forward the forms data to corresponding Bean	F1
ER_Servlet	To forward the ER-Form data to corresponding Bean	F8

Point B:

Table 2 Business Tier Elements

Business Tier		
Bean	Type	Requirement
Kindergarten_Manager	Stateless Bean	F5, F6, F7
ER_Manager	Stateless Bean	F4, F8
Child_Manager	Stateless Bean	F1,F2,F3

Kindergarten_Manager, Child_Manager are stateless beans to manage the operations related to the Kindergarten and Child entities such as registration, profile updating and so on.

ER management could be implemented as a stateless bean, the ER_Manager. To address F8, when a new request is generated by filing the ER form, the main task of the ER_Manager bean is to first extract the requested region in the ER, then query the database to find all the available kindergartens in that region and then send them the ER.

Some methods of Child_Manager, Kindergarten_Manager, and ER_Manager are the following:

Child_Manager:

```
public Integer updatePassword(String userID, Integer rate) {}
public Integer updateResidenceAddress(String addr) {}
```

Kindergarten_Manager:

```
public void SetAvailability(Boolean availability) {}
public Integer updateRating(String userID, Integer rate) {}
```

ER_Manager:

```
public Collection<Kindergarten> findKindergartenByRegion(String RegionName){}

public void sendERs(Collection<Kindergarten> kindergartensInRegion) {
    for (kindergarten k : kindergartensInRegion) {
        // send the ER to k
    }
}
```

Point C:
The best communication model for this scenario is **Message Based Communication** and accordingly, the JEE API to implement it is the **Java Messaging System (JMS)**. The JMS style that could be used here is the **publisher-subscriber model**, so each region would be a Topic and upon ER submission the Publisher notifies the Subscriber of the respected region. The main advantage of having JMS versus Stateless bean is that here the load of the ER management task is distributed over different Beans, each one responsible for single region.
For completeness' sake, the discussion below provides some more details about the modifications to be applied to the previous design to accommodate JMS. Figure 1 shows some details about the architecture.

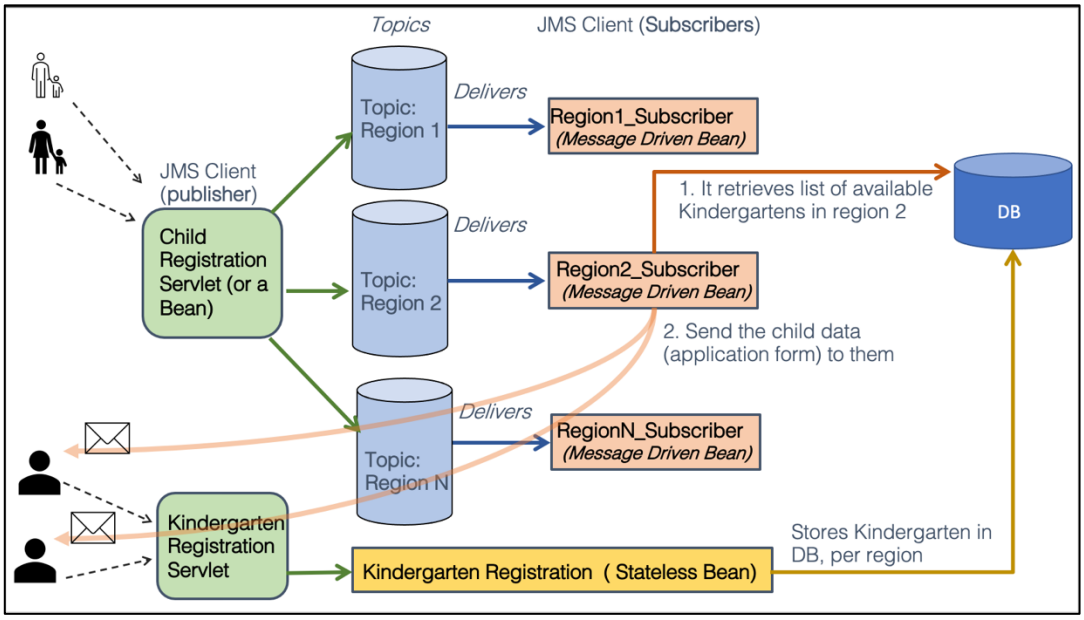


Figure 1 - Kindergarten Registration Management with JMS Architecture

Web Tier (with JMS)

To incorporate JMS we need a Publisher, which could be the ER_Servlet, for the elements shown in Table 1, we need to change ER_Servlet to act as the message producer, and the rest would be not be changed:

Web Tier		
Component	Responsibility	Requirement
ER_Servlet_Publisher	Message Producer	F2,F8

The Servlet here obtains the application form upon the submission of the ER form though a doGet (or doPut, or doPost) method. As depicted in Figure 1, another important job of the Servlet here is to implement the JMS Publisher, where Each Region is a Topic. Based on the selected Region in the ER Form, the Servlet sends a Message (i.e., the data in ER form) to the corresponding Topic Subscriber. *[Alternatively, one can create a Bean as the JMS Publisher. So, the Servlet can first send the form data to that Bean and the Bean generates and publishes the message for each topic.]*

Business Tier (with JMS)

With this design the ER_Manager addresses only F4, and we could have Region1 to RegionN subscribers to address F8. So, the additional/changed elements with respect to those on Table 2 are stated in the following table:

Business Tier		
Bean	Type	Requirement
ER-Manager	Stateless Bean	F4
Region1_Subscriber	Message Driven Bean	F8
Region2_Subscriber	Message Driven Bean	F8
RegionN_Subscriber	Message Driven Bean	F8

Region_Subscribers 1 to N, hence, are Message Driven Beans, i.e., the subscribers to each topic (here the Region 1 to N). So, as soon as a new ER for a particular region (say, Region 2) is generated, the topic

publisher (here the ER_Servlet_Publisher) sends a message to the respected Subscriber (here, the Region2_Subscriber)

Region_Subscribers 1 to N:

```
public void onMessage(Message message){}
```

In each subscriber, the onMessage Method is waiting to act upon receiving the message from the **publisher** (i.e., ER_Servlet_Publisher). For example, when a new ER for Region 2 is submitted, the **publisher** sends a message to **Topic: Region2**, hence the Region2_Subscriber that listens to Topic Region 2 would be immediately notified and it would trigger the onMessage method. This last one would then fetch the email address of all the "available" kindergartens in the Region 2 (by injecting the Entity Manager and querying the DB), and ultimately it would send those kindergartens the new registration request via email.

Question 3 Availability (3 points)

Consider again the replicated and partitioned DBMS of Question 1. Assume that nodes are coordinated by a frontend component that is in charge of acting as an intermediary between the external clients and the nodes. In particular, this frontend accepts two types of requests coming from clients: *QueryOnA*, and *QueryOnD*. The frontend computes *QueryOnA* by passing it to all three subsystems and merging the results obtained by the three. It computes *QueryOnD*, instead, by passing it to only one subsystem. If this one does not answer before a fixed timeout, then the query is passed to another subsystem.
Knowing that the availability of the frontend component is 99.99% and that the availability of each node is 90%, compute the availability of the whole DBMS for what concerns *QueryOnA* and *QueryOnD*.

Solution
The system behaves differently when executing *QueryOnA* and *QueryOnD*.
More specifically, the execution of *QueryOnA* requires the execution of all system components (the frontend and all nodes). As such, it is to be considered as a series from the availability perspective. The resulting availability, therefore, will be smaller than 90%, being the nodes the weakest links in the system ($0.9999 * 0.9^3 = 0.7289$).
The execution of *QueryOnD* needs the execution of the frontend, followed by the parallel constituted by the three nodes. The resulting availability is therefore obtained as: $0.9999 * (1 - (1-0.9)^3)$, which is 99.89%.