Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

***Prof. Elisabetta Di Nitto, Matteo Rossi and Damian Tamburri***

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)

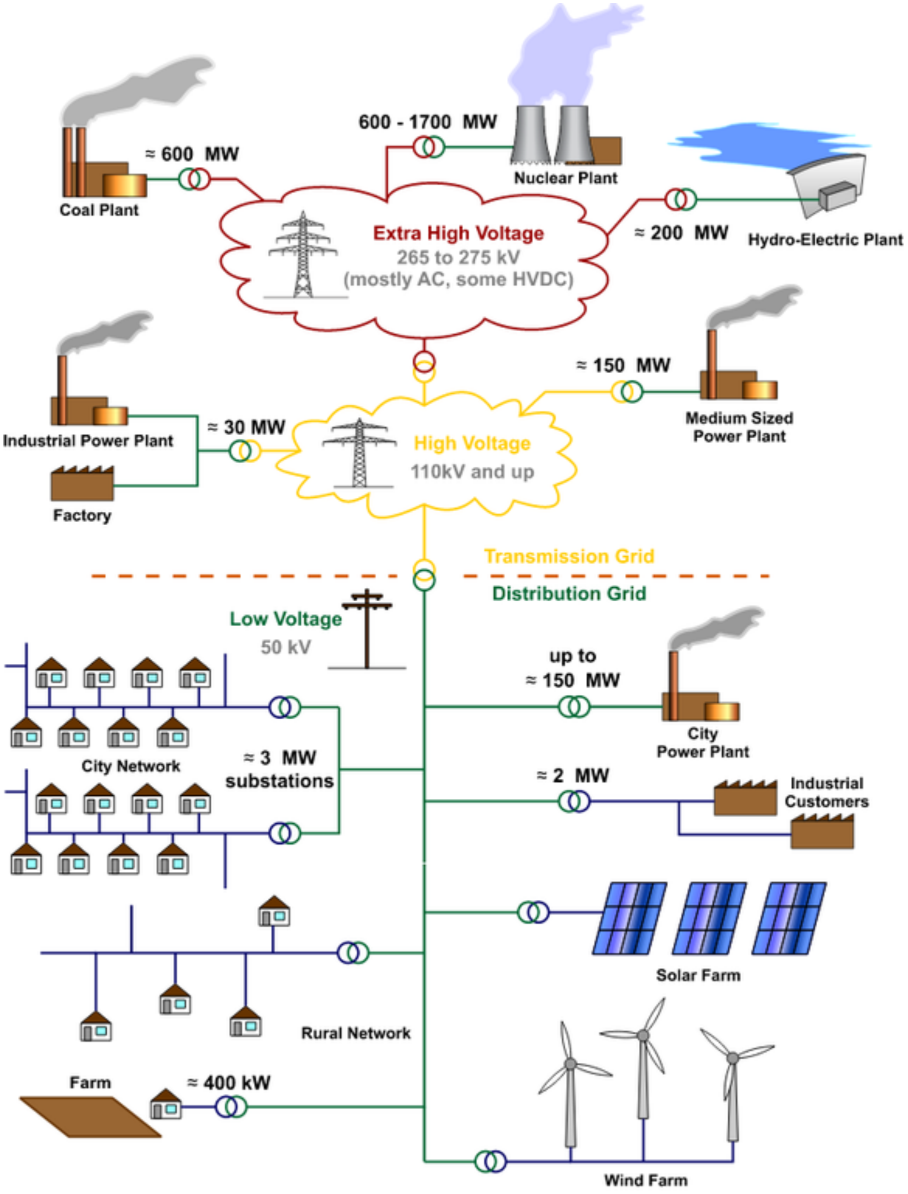
**February 14th, 2022**

### 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 three exercises. Read carefully all points in the text!
6. **Total available time for WE1: 1h and 30 mins**

## Question 1 Alloy (8 points)

The following figure describes an electrical grid[1]. According to the Energy Education organization[2]: "*The electrical grid is the intricate system designed to provide electricity all the way from its generation to the customers that use it for their daily needs. These systems have grown from small local designs, to stretching thousands of kilometers and connecting millions of homes and businesses today. The grid consists of countless complex interconnections, however there are three main sections: **electricity generation**, **transmission** and **distribution***".



Electricity generation can be obtained through either nonrenewable sources such as coal, or renewable ones such as water, wind and solar energy.

Electricity transmission can take place either over short distances or over long ones if high voltage power lines are used. In this case, transformers are used to increase voltage at the source (step-up transformers) and then decrease it at the destination (step-down transformers).

The distribution grid connects step-down transformers to customers that can range from industrial buildings to homes. Other transformers in the distribution grid help to lower the voltage further.

### Alloy_1 (3 points)

Given the following fragment of Alloy model:

```
abstract sig Voltage {}
one sig LowVoltage extends Voltage {}
one sig MediumVoltage extends Voltage {}
one sig HighVoltage extends Voltage {}

abstract sig Source {}
one sig RenewableSource extends Source {}
one sig NonRenewableSource extends Source {}

sig GenerationPlant {
```

[1] Wikimedia Commons [Online], Available: https://commons.wikimedia.org/wiki/File:Electricity_Grid_Schematic_English.svg

[2] https://energyeducation.ca/encyclopedia/Electrical_grid

```
   energySource: Source,
   sendTo: Transformer,
   generateVoltage: MediumVoltage
}
{generateVoltage = sendTo.inVoltage}
```

Complete it by modeling the following concepts:

- Transformers that have an incoming electricity flow at a certain voltage (`inVoltage`) and produce as output an outgoing flow at a different voltage (which can be higher or lower). Moreover, transformers can be connected to other transformers.
- Customers that have a required electricity voltage and receive it from a certain transformer which should provide a compatible outgoing voltage.

**Alloy_2 (1 point)**
Define the function `TransformerConnectedToPlants` that returns the set of transformers that directly receive electricity from a generation plant.

**Alloy_3 (2 points)**
Define a fact to ensure that all transformers are connected either directly (through the `sendTo` relation) or indirectly to a generation plan.

**Alloy_4 (2 points)**
Define a predicate that, given a customer, returns true if this customer receives electricity from a renewable generation plant (`energySource` for this plant should be `Renewable`).

**Solution**

**Alloy_1 (3 points)**

```
sig Transformer {
   inVoltage: Voltage,
   outVoltage: Voltage,
   connectedTo: set Transformer
}
{ not this in connectedTo and
  not(inVoltage=outVoltage) }

fact noCircularity {
  all t:Transformer | not t in t.^connectedTo
}


fact transformerProperlyConnected {
all   t:   Transformer   |   t.connectedTo   !=   none   implies   t.outVoltage   =
t.connectedTo.inVoltage
}

sig Customer {
   receiveFrom: Transformer,
   requiredVoltage: Voltage
}{ not (requiredVoltage = HighVoltage)  and requiredVoltage = receiveFrom.outVoltage}
```

**Alloy_2 (1 point)**

```
fun TransformerConnectedToPlants(): set Transformer {
  {GenerationPlant.sendTo}
}
```

**Alloy_3 (2 points)**

```
fact electricityDeriveFromPlant {
   all  t:  Transformer  |  t  not  in  TransformerConnectedToPlants  implies
(^connectedTo).t & TransformerConnectedToPlants != none
}
```

**Alloy_4 (2 points)**

```
pred getFromRenewable [c: Customer] {
   let originT = {(^connectedTo).(c.receiveFrom) & TransformerConnectedToPlants} |
   some g: GenerationPlant | g.sendTo & originT != none and
                             g.energySource = RenewableSource

}
```

**Question 2 JEE (5 points)**

DoReMi is a prestigious music school that has several branches in Lombardy, with thousands of students. A software company is implementing DoReMi's enterprise system using JEE and is currently focusing on the following functions:

F1. *Candidate student registration*: each candidate student must register to the system, entering email, name, surname, date of birth, his/her preferred instrument, and the corresponding proficiency level.

F2. *Enrolment request*: candidate students can request to enrol in a specific branch of the school. Each candidate can apply for more branches but can send one application at a time.

F3. *Evaluation of enrolment requests*: branch directors can accept or reject the registration requests.

F4. *Visualization of Enrolment Request status*: candidate students can visualize the status of their requests, which can be one of the following: accepted, pending, or rejected.

**JEE_1 (2 points)**

The software company has designed the beans listed in the following table. Referring to the multiplicity of the relationships highlighted in the table, explain, in the last column, whether they are correct or incorrect for the problem at hand and why.

| Entity | Attributes | Relationship With | Multiplicity | Your comment |
|---|---|---|---|---|
| Student | Email, Name Surname, Birth Proficiency level, instrument | EnrolmentReq | Many_to_Many | |
| SchoolBranch | Name | EnrolmentReq | Many_to_Many | |
| EnrolmentReq | Status | Student | Many_to_Many | |

| | | | | |
|---|---|---|---|---|
| | | | | |

**JEE_2 (3 points)**
Besides the entities described above, the software company has also designed the following beans:

- Student_manager: This is a stateless bean in charge of allowing users to register, issue enrolment requests and visualize the pending requests (F1, F2, F4).
- School_manager: This is a stateless bean in charge of allowing users (DoReMi directors) to evaluate requests (F3).

Currently, the company is focusing on the web tier and on ensuring that the workflow concerning the management of enrolment requests, from the time they are issued to the time they are accepted/rejected by the DoReMi branch, works properly. To this end, the company is building the following table. Help the company complete it with the proper elements. More specifically, fill in the empty cells and add as many rows as needed with the required additional components (the number of rows available in the table does not necessarily correspond to the number of components to be added).

| Web component/bean/Entity | Activating event/element | Type | Short description of the component/bean/entity behavior |
|---|---|---|---|
| Front end for EnrolmentReq form | A servlet component sending this form to the client | | The form will allow the candidate student to fill in the required information (email, name, surname, date of birth, his/her preferred instrument, and the corresponding proficiency level) |
| Student_ER_Servlet | The web server upon receival of the form content through an HTTP request | Servlet | This servlet will perform some syntactic check on the acquired data and will call the session bean in charge of handing the request |
| Student_manager | | Session bean | This stateless session bean will instantiate a new EnrolmentReq and call a student's entity method addNewRequest to add the request to the list of those associated with that student |

**Solution**
The added parts are highlighted in bold

**JEE_1 (2 points)**

| Entity | Attributes | Relationship With | Multiplicity | Your comment |
|---|---|---|---|---|
| Student | Email, Name Surname, Birth | EnrolmentReq | Many_to_Many | **The multiplicity is not correct.**<br>**The student can be associated with more than one request, but each request is associated with only one student. The correct multiplicity is One_to_Many** |

| | | | | |
|---|---|---|---|---|
| | Proficiency level, instrument | | | |
| SchoolBranch | Name | EnrolmentReq | Many_to_Many | **The multiplicity is not correct.<br>The student can enrol in only one branch at a time so, each request is associated with only one SchoolBranch. The correct multiplicity is One_to_Many** |
| EnrolmentReq | Status | Student | Many_to_Many | **The multiplicity is not correct.<br>The student can be associated with more than one request, but each request is associated with only one student. The correct multiplicity is Many_to_One** |

## JEE_2 (3 points)

| Web component/bean/Entity | Activating event/element | Type | Short description of the component/bean/entity behavior |
|---|---|---|---|
| Front end for EnrolmentReq form | A servlet component sending this form to the client | **JSP** | The form will allow the candidate student to fill in the required information (email, name, surname, date of birth, his/her preferred instrument, and the corresponding proficiency level). **The selected school branch should be included as well.** |
| Student_ER_Servlet | The web server upon receival of the form content through an HTTP request | Servlet | This servlet will perform some syntactic check on the acquired data and will call the session bean in charge of handing the request |
| Student_manager | **The Student_ER_Servlet that calls this bean to handle the request from the student prespective** | Session bean | This stateless session bean will instantiate a new EnrolmentReq and call a Student's entity method addNewRequest to add the request to the list of those associated with that student**, as well as a SchoolBranch entity method addNewRequest to add the request to the list of those associated with that SchoolBranch.** |
| **Student** | **Student_manager** | **Entity** | **Updates the list of requests associated with the student** |
| **School_Servlet** | **The web server upon receival of an "evaluate requests" message (wrapped into an HTTP get) coming from a director's browser** | **Servlet** | **This servlet will send a request to the School_manager to retrieve the list of requests to be evaluated. Upon receival of this list, it will create the form to be sent back to the browser.** |
| **Front end for request evaluation** | **School_Servlet** | **JPS** | **The form shows the list of pending requests and allows the directors of the school to decide whether to accept or reject** |
| **School_ER_Servlet** | **The web server upon receival of the form** | **Servlet** | **The servlet acquires the data (the choices of the director) and calls the School_manager bean to handle the operation** |

|  |  |  |  |
|---|---|---|---|
|  | **content through an HTTP request** |  |  |
| **School_manager** | **This is activated by two servlets for the purpose of the analysed functionality:**<br>**1. The School_Servlet to retrieve the list of requests the director must evaluate.**<br>**2. The School_ER_Servlet that calls this bean to handle the result of the request evaluation** | **Session bean** | **This stateless session bean will call:**<br>**The SchoolBranch' s entity method to retrieve the corresponding list of requests and send it back to the School_Servlet;**<br>**The EnrolmentReq' s entity method changeERStatus to change the status of the request.** |
| **SchoolBranch** | **School_manager** | **Entity** | **This entity offers the getListOfERRequests** |
| **EnrolmentReq** | **SchoolBranch** | **Entity** | **This entity offers the changeStatus method to change the status of the specific enrolment request** |

Note that, if we consider the visualization of the request status by the student, we need to foresee the possibility for Student_ER_Servlet and Student_manager to handle also a visualizeERStatus request coming from the frontend. The response sent back to the browser can be a new webpage describing the enrolment request status.

## Question 3 Symbolic Execution (3 points)

Consider the following fragment of code:

```
1  int computation(int a[], int n){
2    int i, count, prod;
3    if (n < 2)
4        return -1;
5    i = 0;
6    count = 0;
7    prod = 1;
8    while (i < n){
9        if (a[i] == 0)
10            count++;
11       prod = prod*a[i];
12       i++;   }
13   if (count < 2 || prod == 0)
14       return -1;
15   else return prod;
16 }
```

### SE_1 (1.5 points)

Derive the path conditions corresponding to the execution of path 1, 2, 3, 5, 6, 7, 8, 9, 11, 12, 8, 9, 10, 11, 12, 8, 13, 14.

### SE_2 (1.5 points)

Derive the path conditions corresponding to the execution of path 1, 2, 3, 5, 6, 7, 8, 13, 15.

**Solution**

**SE_1**

Symbolic execution for path 1, 2, 3, 5, 6, 7, 8, 9, 11, 12, 8, 9, 10, 11, 12, 8, 13, 14:

1 $a = A, n = N$
2
3 $N \geq 2$
5 $i = 0$
6 $count = 0$
7 $prod = 1$
8 $0 < N$
9 $A[0] \neq 0$
11 $prod = A[0]$
12 $i = 1$
8 $1 < N$
9 $A[1] = 0$
10 $count = 1$
11 $prod = A[0]*A[1] = 0$
12 $i = 2$
8 $2 \geq N$
13 $1 < 2$ or $0 = 0$

The path condition is then $N = 2$ and $A[0] \neq 0$ and $A[1] = 0$

**SE_2**

Symbolic execution for path 1, 2, 3, 5, 6, 7, 8, 13, 15:

1 $a = A, n = N$
2
3 $N \geq 2$
5 $i = 0$
6 $count = 0$
7 $prod = 1$
8 $0 \geq N$

We have already obtained a contradiction, because it cannot be at the same time $N \leq 0$ and $N \geq 2$, so this path is not feasible.