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 II

July 13th, 2018

Last Name
First Name
Id number (Matricola)

Note

1. The exam is not valid if you do not fill in the above data.
2. Write your answers on these pages. Extra sheets will be ignored. You may use a pencil.
3. Incomprehensible hand-writing is equivalent to not providing an answer.
4. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden.
5. You cannot keep a copy of the exam when you leave the room.
6. The exam is composed of three exercises. Read carefully all points in the text!
7. **Total available time: 2h**

Scores of each question:

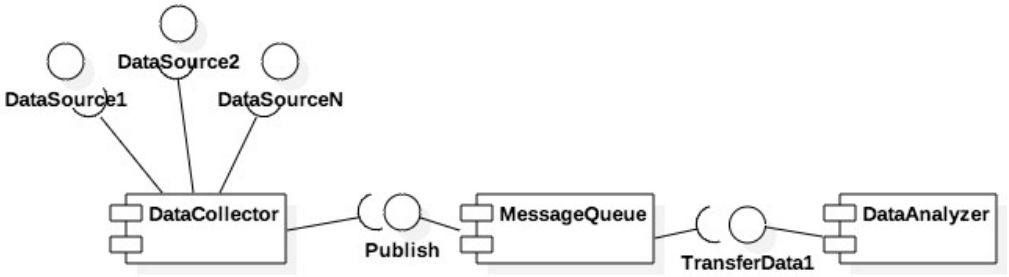
Question 1 (MAX 8) _____

Question 2 (MAX 6) _____

Question 3 (MAX 5) _____

Question 1 Alloy (8 points)

Consider the following UML component diagram.



This diagram describes a software system that acquires and elaborates information from a number of different sources by polling them periodically. The `DataCollector` component is exploiting the interfaces offered by some data sources to acquire data and the interface of the `MessageQueue` to pass collected data to the other components. The `MessageQueue` exploits the interface offered by the `DataAnalyzer` to pass the data to this component.

Write in Alloy the signatures that model a `DataSource`, a `DataCollector`, a `MessageQueue` and a `DataAnalyzer`. Make sure that you represent in the model the connections between components that are highlighted in the UML component diagram.

Assume that we decide to replicate the `DataCollector` component. Model in Alloy the following possible configurations of the system:

- **Configuration 1**: Each `DataCollector` replica is connected to a disjoint subset of `DataSource` components.
- **Configuration 2**: All `DataCollector` replicas are connected to all `DataSource` components.
- **Configuration 3**: `DataCollector` components are classified in *master* and *slaves*. There is always one `DataCollector` that acts as *master*.

Solution

A possible solution for the exercise is the following. An alternative one is proposed after it. Other ones are also possible, of course.

```
abstract sig Interface {}
sig DataSource extends Interface {}
sig Publish extends Interface {}
sig TransferData1 extends Interface {}

abstract sig Component {
  provides: set Interface,
  requires: set Interface
}

sig DataCollector extends Component {}
```

```
{ #provides = 0 and
one i: Interface | i in requires and i in Publish and
requires & TransferData1 = none }

sig MessageQueue extends Component {
}
{ #provides = 1 and
provides in Publish and
requires in TransferData1 }

sig DataAnalyzer extends Component {}
{ #provides = 1 and
provides in TransferData1 }

//The various configurations are modeled as predicates so they can coexist
//in the same model

pred configuration1 [] {
  all disj dc1, dc2: DataCollector | (dc1.requires - Publish) & (dc2.requires -
Publish) = none
  all ds: DataSource | ds in DataCollector.requires
  #DataCollector >=1 and #DataSource >=1
}

run configuration1 for 5

pred configuration2 [] {
   all dc: DataCollector | all ds: DataSource | ds in dc.requires
  #DataCollector > 1 and #DataSource > 1
}

run configuration2 for 5

abstract sig Bool {}
one sig True extends Bool {}
one sig False extends Bool {}

sig MSDataCollector extends DataCollector {
   master: one Bool
}

pred configuration3 [] {
   all dc: DataCollector | dc in MSDataCollector
   one dc: DataCollector | dc.master = True and (DataSource - (DataSource &
dc.requires) = none)
```

```
}

run configuration3 for 5
```

Alternative Solution

```
sig DataSource {}

sig DataCollector {
  sources: set DataSource,
  queue : MessageQueue
}

sig MessageQueue {
  analyzer: DataAnalyzer
}

sig DataAnalyzer {}

sig Configuration {
  sources: set DataSource,
  collectors: set DataCollector,
  queue: MessageQueue,
  analyzer: DataAnalyzer
}
{ // all DataCollector components are connected to the same MessageQueue,
  // which is connected to the DataAnalyzer of the configuration
  all coll : collectors | coll.queue = queue
  queue.analyzer = analyzer
  // also, the DataSource components used by the DataCollector ones are exactly
  // those of the configuration
  collectors.sources = sources
}

// We capture the different configurations through extensions of the Configuration
// signature above; they add the necessary constraints
sig Configuration1 extends Configuration{}
{ all disj coll1, coll2 : collectors | coll1.sources & coll2.sources = none }

sig Configuration2 extends Configuration{}
{ all coll : collectors | coll.sources = sources }

sig MasterDataCollector extends DataCollector {}
sig SlaveDataCollector extends DataCollector {}
```

```
sig Configuration3 extends Configuration{}
{ all coll : collectors | coll in (MasterDataCollector | SlaveDataCollector)
  one coll : collectors | coll in MasterDataCollector
}
```

Question 2 Requirements (6 points)

A startup is interested in positioning itself in the segment of outdoors sport events (marathons, bike races, etc.). In particular, it is interested in developing a new system called *RiderTrack*. The project main goals are to simplify the organization behind an event, to foster the participation of the athletes to whichever kind of race they could be interested in joining, and to allow spectators to see the status of the sport event and, in particular, the position of the potential winner on a map.

To reach this scope, *RiderTrack* allows event organizers to create an event, define the corresponding path and rules (registration date, date of the event, type of event) and make the event available to the athletes for online registration. In addition, during the sport event, *RiderTrack* is able to gather tracking data from multiple sources (e.g., smart phones, smart watches, specialized devices). Such tracking data concern the position of each athlete on the path defined for the event as well as his/her vital parameters. Thanks to such data, the system is also able to offer a web interface through which spectators can see the status of the event and visualize on a map the position of each athlete.

Q1: Given the system described above and referring to the Jackson-Zave distinction between the world and the machine, identify:

- At least two world phenomena that are not shared with the machine.
- At least two shared phenomena controlled by the world.
- At least two shared phenomena controlled by the machine.
- At least one machine phenomenon not shared with the world.

Please use the table below to answer this question (the number of rows does not necessarily correspond to the number of phenomena you can identify for this case).

Phenomenon	**Shared nor not**	**Who controls it**	**Short explanation (if the name you assign to the phenomenon is not self-explanatory, or you need to specify some conditions on the phenomenon)**

Q2: Define in natural language one specific goal for *RiderTrack*, one requirement and one domain assumption referring to the phenomena identified above.

Solution

Phenomena	**Shared nor not**	**Who controls them**	**Short explanation (if the name you assign to the phenomenon is not self-explanatory or you need to specify some conditions on the phenomenon)**
Organization of the sport event	N		World phenomenon
Position of athletes	N		World phenomenon
Path to be followed	N		World phenomenon
Vital parameters of athletes	N		World phenomenon
Competition rules	N		World phenomenon
Position of athletes measured by their devices	Y	World	
Vital parameters of athletes measured by their devices	Y	World	
Map of the path to be followed	Y	Machine	
Map of the status of all athletes shown to the spectators	Y	Machine	
Ordering of athletes shown to the spectators	Y	Machine	

Database with registration data by all athletes	N		Machine phenomenon
Database with status of all athletes during the race	N		Machine phenomenon

Goal: The spectators want to see who is leading in all parts of the path
Assumption: The error of devices in measuring the position of athletes is lower than 5 mt.
Requirement: RiderTrack must display the position of the leading athlete on the map

Question 3: Testing (5 points)
Consider the following fragment of C code.

```
int myprogram(int x, int y, int z)
1  if (x > 5 && y > 1) || (y > 1 && z < 10)
2    return x + y + z;
3  while (x > 5 || y < 1) {
4    if (x > 5) {
5      x = x − 1;
6      if (y > 1)
7        y = y + x;
8    }
9    z = z − y;
10 }
11 return y;
```

1. Using symbolic execution, show whether the statement on line 7 can be executed in one of the first 2 iterations of the `while` loop.

2. Is it possible to find an initial assignment to parameters `x`, `y`, `z` such that the `while` loop never terminates? Support your response with the help of symbolic execution.

Solution

1.
Assume that when `myprogram` is invoked it holds that
`x = A`
`y = B`
`z = C`
Then, to enter the while loop, the following conditions must hold:
`(i) (A ≤ 5 or B ≤ 1) and (B ≤ 1 or C ≥ 10)`
`(ii) (A > 5 or B < 1)`
To reach line 7 at the first iteration it must also hold that
`(iii) A > 5`
`(iv) x = A - 1`
`(v) B > 1`
However, from (i) and (iii) it must hold that `B ≤ 1`, hence condition (v) cannot be true.
Then, at the start of the next iteration of the `while` loop it holds that
`x = A - 1`
`y = B`
`z = C − B`
So, to execute line 7 both `A − 1 > 5` and `B > 1` should hold, which is still impossible.

We still have to consider the case that in the first iteration the condition at line 4 does not hold (in which case the statement at line 7 for sure is not executed).

In this case $A \leq 5$ holds (so $B < 1$ must hold, otherwise the loop is not entered), so x and y are not modified in the loop, and at the beginning of the second iteration we have $x = A$, $y = B$, $z = C - B$, and again the condition at line 4 is false, so line 7 is not executed.

2.

From the discussion of point 1, we can see that if it holds that $A \leq 5$, then at the next iteration it still holds that $x = A$, $y = B$, which means that the loop is executed in the same manner (only z is updated), so at the next iteration it will still hold that $x = A$, $y = B$, and so on. So, for any $A \leq 5$, $B < 1$, the program enters an infinite loop.