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)

July 26th, 2021

Last Name
First Name
Id number (Matricola)

Notes

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 handwriting 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: 1h and 30 mins**

Scores of each question:

Question 1 (MAX 8) _____

Question 2 (MAX 5) _____

Question 3 (MAX 3) _____

Question 1 Alloy (8 points)

Consider a framework that manages the deployment of software modules on devices with different features. Each software module is characterized by an author and a packaging technology (e.g., JAR, APK, Debian package, Docker image, etc.). Each device is characterized by its location, it can handle one or more packaging technologies and it is able to host software modules built using those technologies.

Point A (2 points)
Define suitable Alloy signatures – and any related constraints – to describe software modules and devices.

Point B (1 point)
Define a signature capturing the configurations of a deployed system, where a configuration is represented by a set of devices and by the modules that can be hosted (i.e., that can be deployed) on them. Notice that a software module can be replicated across different devices, but there cannot be multiple instances of the same software module on a device.

Point C (2 points)
Define an Alloy function that returns, given a system configuration and a software module, the set of devices which host that module in that configuration.

Point D (3 points)
Formalize in Alloy the behavior of an operation that takes as input a system configuration, a software module, and a device of the system, and deploys the module on the device; that is, if the module is not yet deployed on the device, the operation adds it to the list of modules that can be deployed.

Solution

Point A

```
abstract sig PackagingTechnology {}
one sig JAR extends PackagingTechnology {}
one sig APK extends PackagingTechnology {}
one sig DEB extends PackagingTechnology {}
one sig DOCKER extends PackagingTechnology {}
// Others could be added

sig Location {}
sig Author {}

sig SoftwareModule {
    tech: PackagingTechnology,
    auth: Author
}

sig Device {
    loc: Location,
    canHandle: set PackagingTechnology,
    deployed: set SoftwareModule,
}{ deployed.tech in canHandle }
```

Point B

```
sig DeployedSystem {
    deployable : set SoftwareModule,
    devices: set Device,
}{ devices.deployed in deployable }
```

Point C

```
fun deployedOn [c : DeployedSystem, sm : SoftwareModule ] : set Device {
     let res = {d: Device | d in c.devices and sm in d.deployed} |
     res
}
```

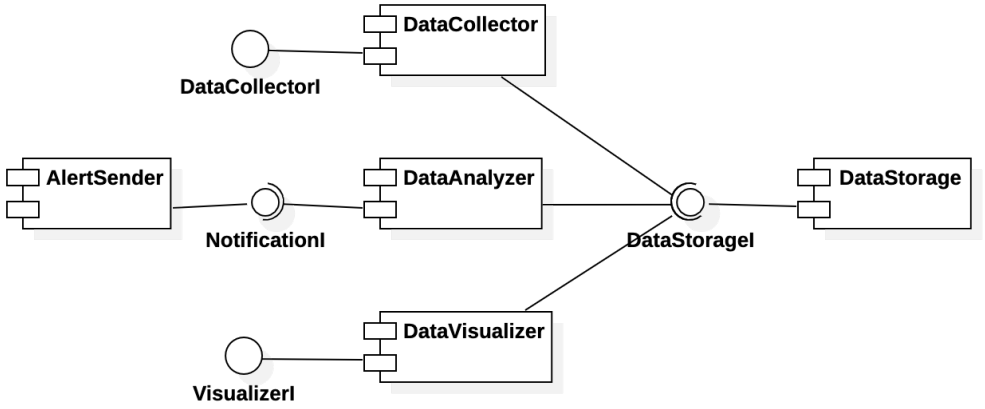
Point D

```
pred deployModule[c, c' : DeployedSystem, sm : SoftwareModule, d : Device] {
  // pre-conditions
  d in c.devices
  sm.tech in d.canHandle
  not sm in d.deployed   // this could be left out

  // post-conditions
  c'.deployable = c.deployable + sm
  some d' : Device |
       d'.loc = d.loc and d'.canHandle = d.canHandle and
       d'.deployed = d.deployed + sm and
       c'.devices = c.devices – d + d' )
}
```

Question 2 Architecture (5 points)

Consider the following architecture of a system for the monitoring of a safety-critical plant.



The *DataCollector* module receives data (in a push manner) from the plant and stores it in the *DataStorage* component. The *DataAnalyzer* module periodically reads data from the *DataStorage* module and elaborates it; if it detects anomalies, it uses the *AlertSender* module to alert the plant (human)

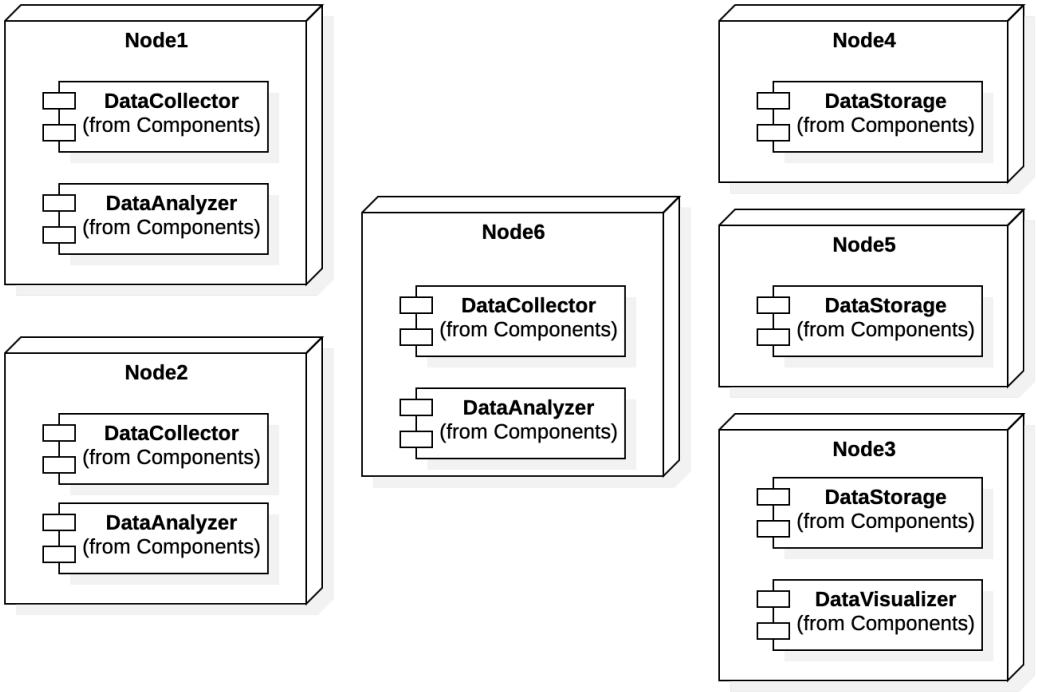
supervisor. Finally, the *DataVisualizer* module is used by the supervisor to see statistics concerning the plant.

The *AlertSender* module is deployed on a very reliable server, so it has 99.999% availability. We want to find a suitable, economical deployment of the other modules that guarantees a sufficient level of availability. Imagine that we have at our disposal a server that can host all modules, that guarantees 99.999% availability, and that costs 20Keuro. We also have cheaper servers at our disposal, but which guarantee lower availability, and which can host fewer modules. In particular, we have servers that cost 2keuro, which can host 2 modules each, and which guarantee 95% availability.

Define a suitable deployment of modules so that the data collection and data analysis functions have a 99.9% availability, while the visualization function has a 90% availability. The cheaper the deployment, the better.

Solution

Consider the following deployment, where the *DataCollector*, *DataAnalyzer*, and *DataStorage* components are triplicated. The data collection chain is composed of the *DataCollector* and *DataStorage* components, while the data analysis chain is composed of the *DataAnalyzer* and *DataStorage* components. The triplicated components have availability $1-(1-0.95)^3 = 0.999875$. Their series has availability $0.999875*0.999875 = 0.99975$, which higher than the desired one.



The data visualization chain, instead, has one instance of *DataVisualizer*, which is hosted on a server that also hosts an instance of *DataStorage*. Hence, if the server with *DataVisualizer* fails, also one of the instances of *DataStorage* fails. Then, we can consider the visualization chain to be made of one instance of *DataVisualizer*, in series with 2 parallel instances of *DataStorage*. As a consequence, the data visualization chain has availability $0.95*(1-(1-0.95)^2) = 0.947625$, which is sufficient.
The 6 nodes in total cost 12Keuro, which is less than the cost of the single, highly reliable server on which they could all be deployed.

Question 3 Testing (3 points)

Consider the following C function:

```
0 int my_func (int x, int y){
1   int z, k, res;
2   if (x <= 0 || y <= 0)
3       return -1;
4   z = x;
5   k = y;
6   while (x != y){
7       if (x > y)
8           x = x - y;
9       else y = y - x;  }
10  if (z % x == 0){
11      res = 1;
12      z = 1;     }
13  else z = 0;
14  return res;    }
```

Identify the def-use pairs for program `my_func` and say whether they highlight anything unusual.

Solution

x: <0,2>, <0,4>, <0,6>, <0,7>, <0,8>, <0,9>, <0,10>
<8,6>, <8,7>,<8,8>, <8,9>, <8,10>
y: <0,2>, <0,5>, <0,6>, <0,7>, <0,8>, <0,9>
<9,6>, <9,7>,<9,8>, <9,9>
z: <4, 10>, <12,?>, <13,?>
k: <5, ?>
res: <?,14>, <11,14>

Variable *z* is set at lines 12 and 13, but it is then never used, which is not necessarily a problem, though it is not ideal. Similarly, variable *k* is set at line 5, but it is never used. Variable *res* is potentially used at line 14 without being initialized. However, an inspection of the paths that lead to line 14 show that line 13 is actually never executed (i.e., the condition at line 10 is always true), so variable *res* will actually be always initialized when it is used.