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 1 (WE1)

February 5th, 2024

Last name, first name and Id number (matricola):
Number of paper sheets you are submitting as part of the exam:

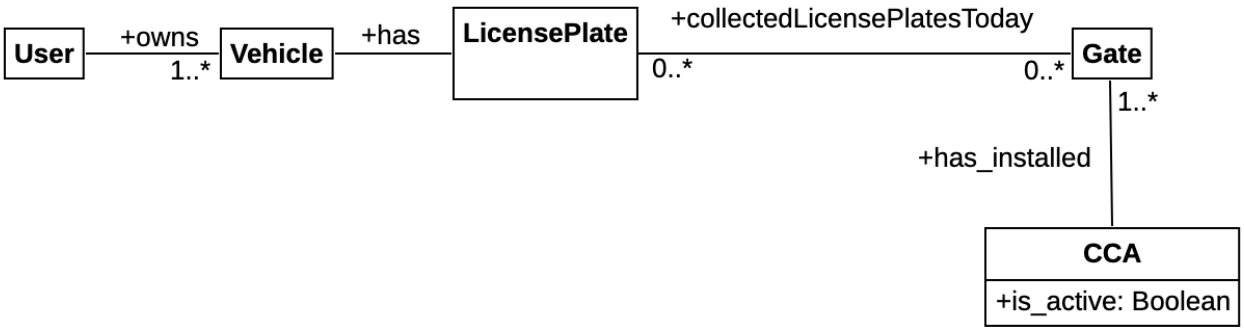
Notes

A. Write your name and Id number (matricola) on each piece of paper that you hand in.
B. You may use a pencil.
C. Unreadable handwriting is equivalent to not providing an answer.
D. The use of any electronic devices (computer, smartphone, camera, etc.) is strictly forbidden, except for an ebook reader or a plain calculator.
E. The exam is composed of three exercises. Read carefully all points in the text.
F. Total available time for WE1: 1h and 30 mins

Question 1 – Alloy (5 points)

Consider a system to monitor the accesses of vehicles to the center of a city (think "Area C" in Milano). The system detects vehicles entering the City Center Area (CCA for short) using "gates" installed on the streets through which vehicles can access the CCA. Each time a vehicle goes through a gate, the gate reads the license plate of the vehicle and provides the system with the corresponding information (license plate number and time of passage of the vehicle). The CCA is active only until a certain time of the day (e.g., until 7pm).
You are asked to use the features of **Alloy6** to capture some features of the system, focusing on the handling of the accesses **for a single day**.

In particular, consider the following (simplified) domain model for the system, represented through a UML Class Diagram.



Alloy_1 (3 points)
Define suitable signatures and constraints to capture the domain model shown above. In particular, identify the elements of the model that are mutable.

Note: Use the following signature for `Boolean`:

```
abstract sig Boolean {}
one sig True extends Boolean {}
one sig False extends Boolean {}
```

Alloy_2 (2 points)
Define a fact `activeCCAdef` that states that the CCA is initially active, then at some point it must become inactive; after becoming inactive, the CAA cannot become active again.

Solution

Alloy_1

```
sig User {
  owns : some Vehicle
}

sig Vehicle {
  has : one LicensePlate
}{
  one u : User | this in u.owns
}
```

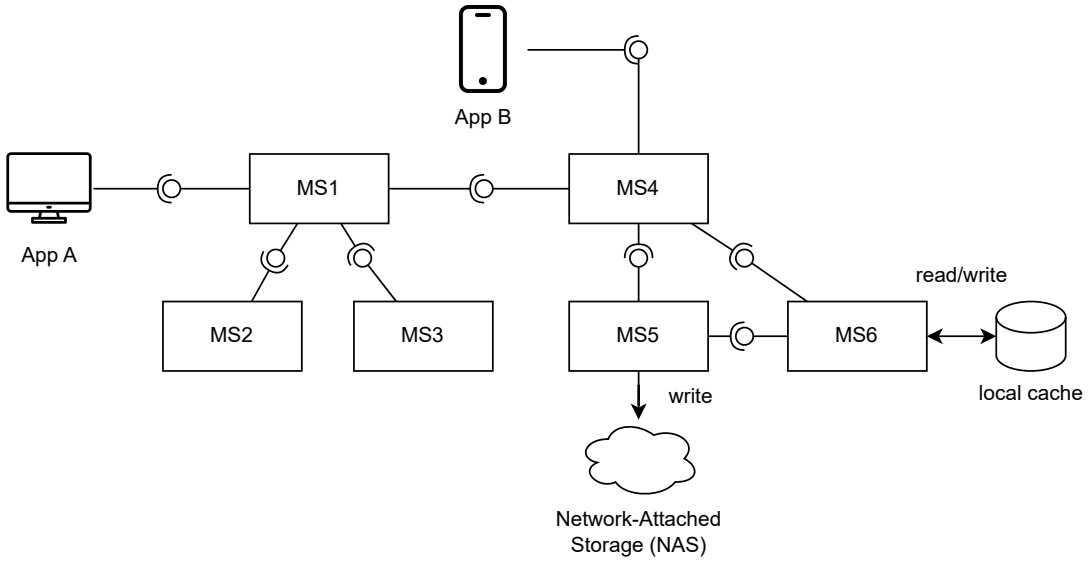
```
sig LicensePlate {}{
  one v : Vehicle | v.has = this
}

sig Gate {
  var collectedLicensePlatesDay : set LicensePlate
}{
  one cca : CCA | this in cca.has_installed
}

sig CCA {
  has_installed : some Gate
  var is_active : Boolean
}
```

Alloy_2

```
fact activeCCAdef {
  all cca : CCA | cca.is_active = True and
                  eventually cca.is_active = False and
                  always ( cca.is_active = False
                           implies
                           always cca.is_active = False )
}
```

Question 2 – Software Architecture (6 points)

Consider the following high-level component structure that is a static description of a possible microservices-based system implementing the business logic for two applications (App A, and App B). The UML Component Diagram below highlights the interfaces (both provided and required ones) of microservices MS1, …, MS6. Moreover, the diagram shows that MS5 writes data on a Network-Attached Storage (NAS) device and MS6 reads and write data on a local cache.



SA_Q1 (1 point): Based exclusively on the static information presented in the Component Diagram, enumerate all possible execution paths (in terms of sequences of microservices) triggered by requests generated by App A and App B, respectively. Notice that we are interested only in paths that terminate with a microservice that does not require any further interface.
Then, describe the ripple effect possibly generated by a sudden slowdown of the NAS on each of the identified paths. Assume every request to MS5 causes interactions with the NAS. Identify the application(s) affected by this disruptive event. Justify your answer.

SA_Q2 (2 points): Assume you have the following additional information about the runtime behavior of the system. Microservice MS5 writes data into the NAS and then propagates the same pieces of data to MS6, which writes them on the local cache. Remember that MS5 does not read data from the NAS. All reads are carried out by MS6 using the local cache only.

Describe how you can use the circuit breaker pattern to mitigate the disruption described above (SA_Q1), taking into account the following constraints:

- NAS must be avoided during the disruption.
- All write operations shall eventually take place, either on the NAS or on the local cache, or on both. Do not consider possible data reconciliation issues.

In your description, explicitly state:

- how many circuit breakers you would use;
- where you would place them;
- their behavior.

Justify your choices.

SA_Q3 (3 points): Assume now that the expected number of requests per second (workload) for the two applications is as follows:

Application	**Workload**
App A	100
App B	150

You can also assume the following behavior when all circuit breakers are closed:

- Requests directed to MS1 are always distributed evenly among all microservices used by MS1.
- 25% of the requests to MS4 are write operations and are transferred to MS5, which, in turn, executes them into the NAS and transfers them to MS6 for local cache updates.
- 75% of the requests to MS4 are read-only operations and are transferred to MS6.
- The number of requests generated by MS2 is always negligible.

Address the following points:

a. Determine the number of requests per second that reach each microservice.
b. Use the numbers you have computed to determine the availability of each individual microservice replica according to the following rules: if the total number of requests a microservice receives is less than 60, the availability estimate is 0.99, if the number of requests is between 60 and 80 the estimate is 0.98, 0.97 otherwise. These estimates apply to each of the considered microservices.
c. Focus now on the write operations generated by App B and on the corresponding path. What is the availability shown by App B for write operations when a single replica of each microservice is used?
d. If we want to satisfy the following nonfunctional requirement: "*The total availability of App B for write operations, shall be at least 0.999, under the condition that any circuit breaker is closed*", what is the required minimum number of replicas for each of the involved microservices?

Solution

SA_Q1

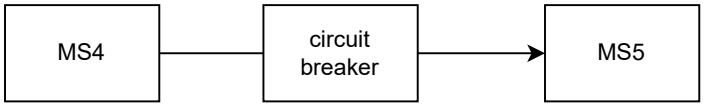
According to static information only, the possible paths triggered by either App A or App B are as follows.

App A → MS1 → MS3
App A → MS1 → MS4 → MS5 → MS6
App A → MS1 → MS4 → MS6
App B → MS4 → MS5 → MS6
App B → MS4 → MS6

A sudden slowdown of the NAS could cause a growing number of pending requests to MS5 due to high number of concurrent (slow) operations of the NAS. This could cause a saturation of the resources of MS5. Therefore, pending requests to MS4 also grow potentially saturating its available resources. Essentially the ripple effect follows in a backward manner all the paths listed above that include MS5 or MS4. Thus, according to the paths the effect potentially propagates back to both App A and App B.

SA_Q2

Only one circuit breaker is necessary to avoid interactions with the slow NAS. We can place the circuit breaker in between MS4 and MS5.



As soon as the number of failed requests grows above the threshold, the circuit breaker goes into state open. At this point the mechanism drops all requests directed to MS5 and runs a fallback procedure that interacts directly with MS6, thus writing the fresh data into the local cache (rather than the NAS). When the circuit breaker changes from half-open to closed, write requests can follow again the nominal flow (… → MS4 → MS5 → MS6).

SA_Q3
When the circuit is *closed,* given the defined workload, we have the following concurrent requests per each microservice:

- MS1 = 100 requests
- MS2 negligible
- MS3 = 50 requests
- MS4 = 150 + 50 = 200 requests
- MS5 = 200 * 0.25 = 50 requests
- MS6 = 200 * 0.75 + 50 = 200 requests

Under such conditions, individual availability estimates are as follows:

- MS1 = 0.97
- MS2 = 0.99
- MS3 = 0.99
- MS4 = 0.97
- MS5 = 0.99
- MS6 = 0.97

The path followed by write operations incoming from App B is MS4 → MS5 → MS6, which, for this purpose, are organized in a series. The availability is then: 0.97*0.99*0.97 = 0.93
To satisfy the non-functional requirement we need to introduce additional replicas for all the three involved microservices. In general, we should satisfy the following constraint:

$(1 - (1 - 0.97)^x) * (1 - (1 - 0.99)^y) * (1 - (1 - 0.97)^z) > 0.999$

With x, y, z, number of replicas for MS4, MS5, MS6, respectively.
Assignments satisfying the previous condition are:

- x=2, y=3, z=3 and x=3, y=3, z=2 that lead to Avail = 0,9991
- x=3, y=2, z=3 that leads to Avail = 0,9998

Instead, instantiating only two replicas per service leads to Avail = 0,9981, while two replicas for MS5 and MS6 or MS4 and three replicas for the remaining service leads to Avail = 0,99897312, which is still slightly lower than 0,999.

Question 3 – Testing (5 points)

Consider the following function `foo` written in C language, where `pp` is a black-box function and, as such, the sources are not available. While doing the exercise you can assume that `pp(x)` returns x^2.

```
0:  void foo(int x) {
1:      int i = 0, z, y;
2:      while (i < 2){
3:          y = pp(x);   // black-box function
4:          if (y > 3)
5:              i = i - y;
6:          else
7:              i = i + y;
8:          i++;
9:      }
10:     if (y*4 > 12){
11:         printf("y: %d", z);
12:         printf("Log: %d", y);
13:     }
14: }
```

Assume that function `foo` receives integer inputs in the range [-4, 5].

Testing_1 (2 points) Can you generate, using *concolic* execution, a test case that exercises a path that leads to statement 14? Justify your conclusion.

Testing_2 (2 points) Can you generate, using *concolic* execution, a test case that exercises a path that leads to blocks 11 and 12 (print statements)? Justify your conclusion.

Testing_3 (1 point) Assume you have a fuzzer that generates random inputs in the range [-4, 5] to test function *foo*. What is the probability that the function hangs (i.e., it enters in an infinite loop)?

Solution

Testing_1

<0>				
x	*z*	*y*	*i*	*π*
X				T
1				

<0,1,2>				
x	*z*	*y*	*i*	*π*
X				0<2
1			0	

<0,1,2,3>				
x	*z*	*y*	*i*	*π*
X		pp(X)		
1		1	0	

<0,1,2,3,4,7>				
x	*z*	*y*	*i*	*π*
X		pp(X)	0+pp(X)	pp(X)<=3
1		1	1	

<0,1,2,3,4,7,8,2>				
x	*z*	*y*	*i*	*π*
X		pp(X)	pp(X)+1	pp(X)+1>=2,
1		1	2	pp(X)<=3

<0,1,2,3,4,7,8,2,10,14>				
x	*z*	*y*	*i*	*π*
X		pp(X)	pp(X)+1	pp(X)>=1,
1		1	2	pp(X)<=3, 4pp(X)<=12

Path condition: pp(X)>=1 && pp(X)<=3 **SAT**

Test case (x=1) leads to statement 14.

Testing_2

Let's start a new execution from the previous leaf.

Partial negation: pp(X)>=1 && !(pp(X)<=3) ≡ pp(X)>=1 && pp(X)>3 ≡ pp(X)>3

Symbolic-to-concrete step (assume we sample x=3): 9>3 **SAT**

Execution:

<0>				
x	*z*	*y*	*i*	*π*
X				T
3				

<0,1,2>				
x	*z*	*y*	*i*	*π*
X				0<2
3			0	

<0,1,2,3>				
x	*z*	*y*	*i*	*π*
X		pp(X)		
1		9	0	

<0,1,2,3,4>				
x	*z*	*y*	*i*	*π*
X		pp(X)		pp(X)>3
3		9	0	

<0,1,2,3,4,5>				
x	*z*	*y*	*i*	*π*
X		pp(X)	0-pp(X)	pp(X)>3
3		9	-9	

<0,1,2,3,4,5,8,2>				
x	*z*	*y*	*i*	*π*
X		pp(X)	-pp(X)+1	-pp(X)+1<2, pp(X)>3
3		9	-8	

<0,1,2,3,4,5,8,2,3>				
x	*z*	*y*	*i*	*π*
X		pp(X)	-pp(X)+1	-pp(X)+1<2, pp(X)>3
3		9	-8	

<0,1,2,3,4,5,8,2,3,4>				
x	*z*	*y*	*i*	*π*
X		pp(X)	-pp(X)+1	-pp(X)<1, pp(X)>3
3		9	-8	

<0,1,2,3,4,5,8,2,3,4,5>				
x	*z*	*y*	*i*	*π*
X		pp(X)	-pp(X)+1-pp(X)	-2*pp(X)<1, pp(X)>3
3		9	-17	

…etc. Variable *i* decreases indefinitely. Therefore, the program enters in an infinite loop. Therefore, there are no test cases leading to location 11 and 12.

Testing_3
According to the previous analysis, the execution hangs if

pp(x) > 1/2 && pp(x)>3, that is, pp(x)>3

given that $pp(x)=x^2$ and x is integer, then pp(x)>3 if x >= 2 or x <= -2

The range for x is [-4, 5] => the probability foo hangs is 7/10 = 0.7