Exam

Course on UML

2007

Ludovic Apvrille

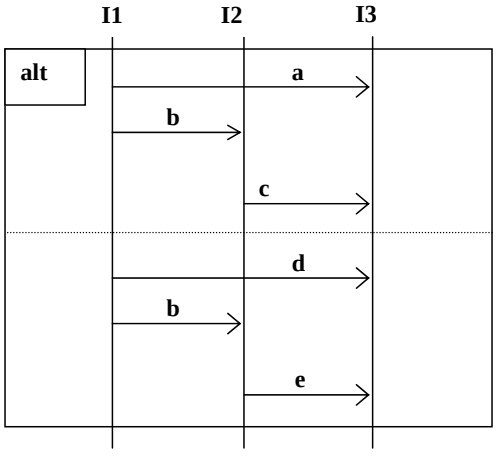
Ludovic.Apvrille@telecom-paris.fr

Authorized documents: courses' slides, notes you've taken during lectures, lab session documents.

Grading is explained for each question. 1 additional point is given for general appreciation (spelling, etc.).
Also, do not spend more than 40 minutes on *exercise I* since the modeling exercise is long.

I. Reasoning Exercise (7 points) ~40mn

We've seen during the last lecture that the language associated to sequence diagrams (or scenarios) has a very large expression power. Thus, an analysis may model a very large set of execution traces, in a very concise way.
This expression power is so large that it is sometimes not possible to make a design that contains exactly the same execution traces. For example, let us analyze the following scenario:

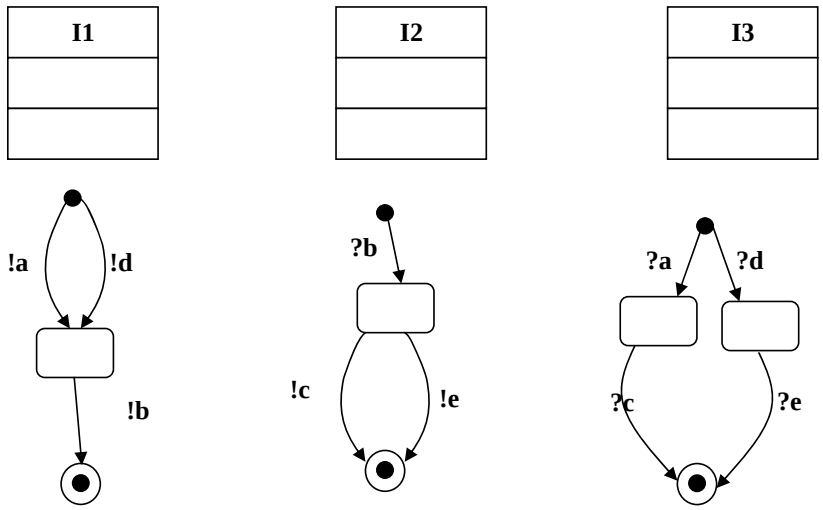


This scenario represents several possible traces, including: (we denote by *!a* the sending of a and by *?a* the receiving of a)
!a !b ?a ?b !c ?c
which means :
I1 sends a, then I1 sends b, I3 gets a, I2 gets b, then I2 sends c and I3 gets c.

1) Modeling an equivalent design

Let's now assume that we're looking for a process P that can generate a design equivalent to an analysis. By "equivalent", we mean that the design has exactly the same traces as the analysis i.e. no more, and no less traces. Process P may be as follows:

- For each instance of the considered scenario, a different class is used. For I1, a class I1 is generated, and so on.
- For each class, a state machine is generated. It represents the execution flow of the related scenario instance. For instance, for I2, the state machine should represent that I2 first gets message b, and then sends either message c or message e.



a) This design has several traces not specified on the scenario. Identify two of them. (1.5 points)

!a?a!b?b!e .. deadlock. This trace is not included in the scenarios.
!d?d!b?b!c ... deadlock. Once again, this trace is not included in the scenarios.

The « equivalent » design having, by process P, more traces than the scenario, and assuming that no other state machines could be found to implement those traces, the scenario is said to be « not implementable ». All design built using process P has at least all traces of the scenario (we won't demonstrate it). In the case of non implementability, it has more traces.

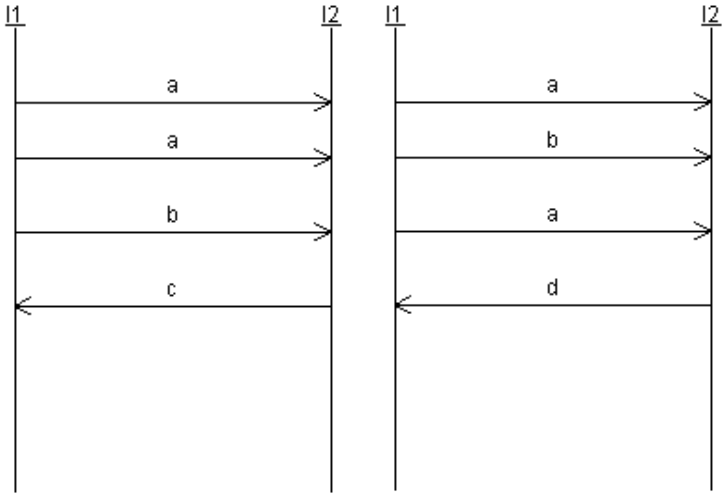
b) Briefly explain why, in your opinion, this scenario is not implementable i.e. the fundamentals behind the fact that process P does not generate an equivalent design. (1.5 points).

I2 has no way to detect which scenario has been selected by I1. Indeed, the scenario selection is based on the sending, by I1 of a or d. But I2 has no way to know whether a or d has been sent, because in both cases it gets b. Because the answer of I2 shall be different in both scenarios, and because there is a deadlock if I2 doesn't send the right message, the scenario is not implementable.

2) Implementable or not?

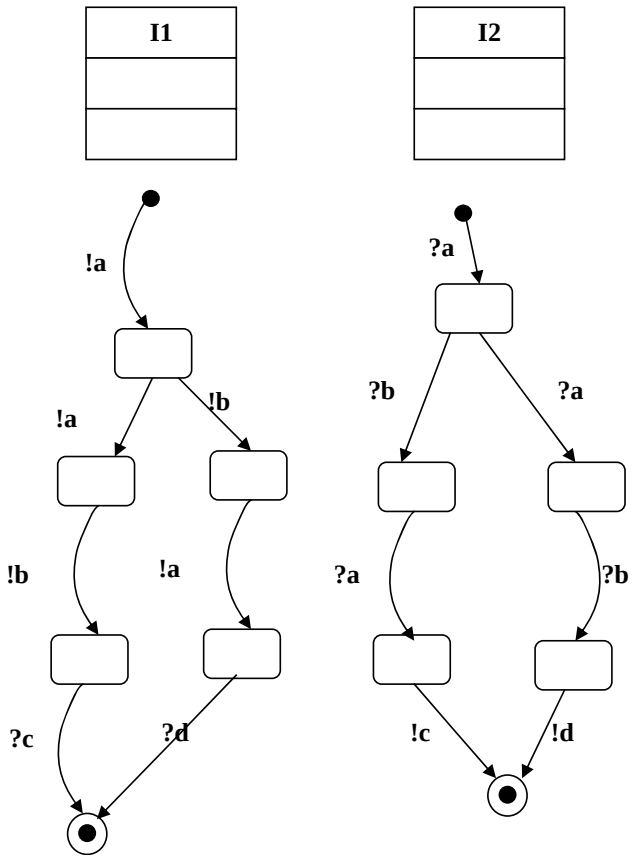
Find out whether each of the following scenario *a)* and *b)* are implementable using process P. It is assumed that there is an alternative between M1 and M2. Grading takes into account the answer only if it is clearly justified, and proved. **No point is given is the answer is not proved,** so, do not try to guess. A good proof should contain the design built using process P, and traces this design has with regards to the ones of the scenario. (2 points for each proof).

a)



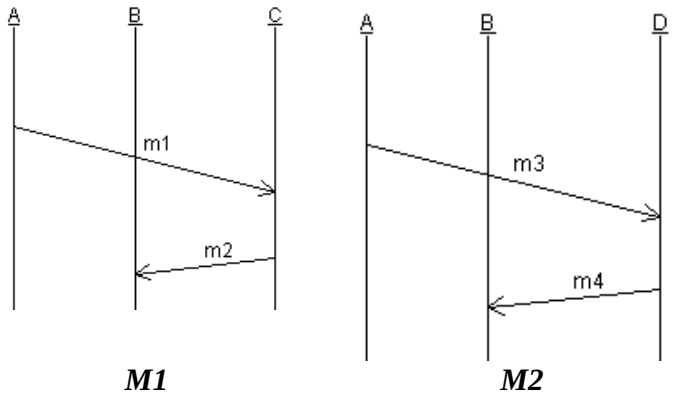
M1 ***M2***

States machines of the corresponding design are as follows:



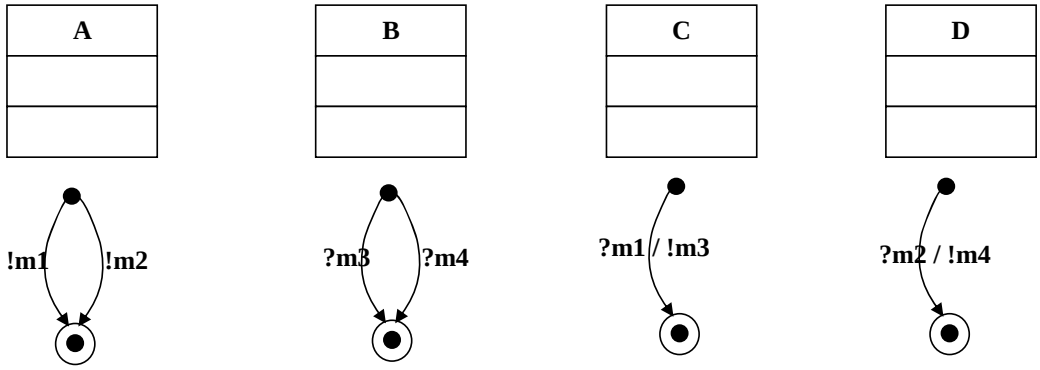
I2 knows which scenario has been selected according to message ordering (a and b).
Hypothesis one: channels do preserve message ordering. In that case, I2 is able to detect the scenario: it is implementable.
Hypothesis two: channels do not preserve message ordering. I2 is not able to detect the right scenario, for example on that trace:
!a!a!b?a?a?b!d deadlock.
Therefore, for that hypothesis, the scenario is not implementable.

b)



M1 ***M2***

States machines of the corresponding design are as follows:



Traces(scenarios) = { (!m1?m1!m3?m3), (!m2?m2!m4?m4)}
Traces(design) = { (!m1?m1!m3?m3), (!m2?m2!m4?m4)}

Since traces are equal, the analysis is implementable.

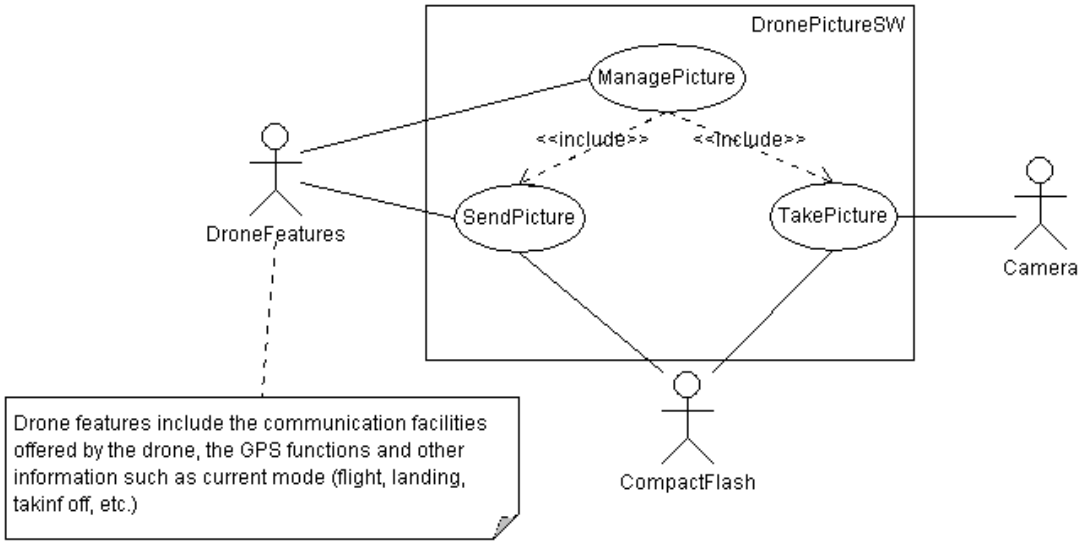
III. Modeling Exercise (12 points) ~80mn

The goal is to model the software part of a drone system. The time being short to carry out this modeling, you may omit some modeling details. But if you do so, please, do clearly mention which details you have omitted, and why you have skipped them. At last, do not forget to add comments to your diagrams. Grading takes equally into account both diagrams and comments. If the requirement description of the system is not clear, **do clearly mention your hypothesis**.

A drone is an aircraft with no pilot. It can autonomously take off, fly and land. The drone to model works as follows. It has a remote control system that makes it possible to take pictures. Only the software related to pictures has to be modeled. Pictures can be taken only when the drone is flying. A picture-order is remotely sent to the drone. This order contains the GPS position of the picture to be taken. To know its current position, a drone has an integrated GPS. When a GPS point is reached, with regards to a given margin, the picture is taken, and then stored on a CompactFlash removable storage system. The system needs 2seconds to take a picture, and between 4 and 5 seconds to store it in on the memory card. Pictures may be remotely downloaded, read– taking out the CompactFlash - once the drone has come back from its mission.

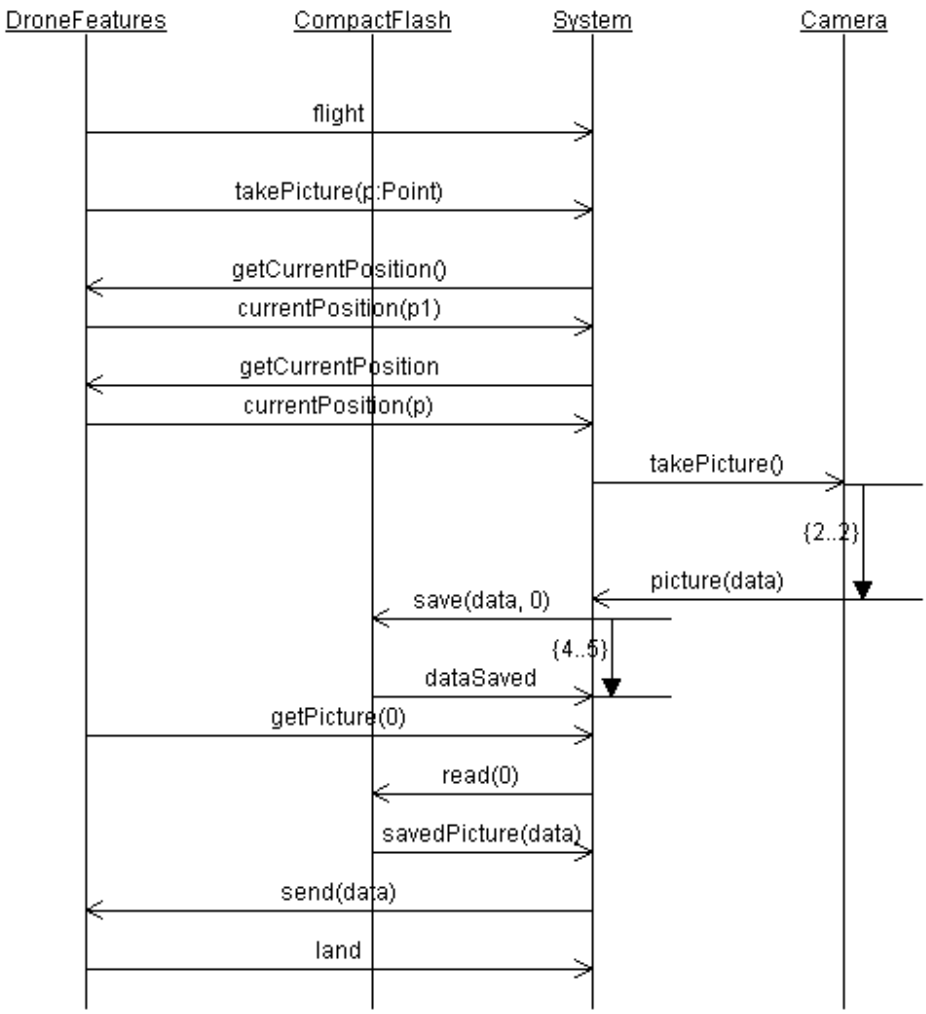
1) Analysis ~45mn

a) Make the use case diagram of this machine. (2 points)



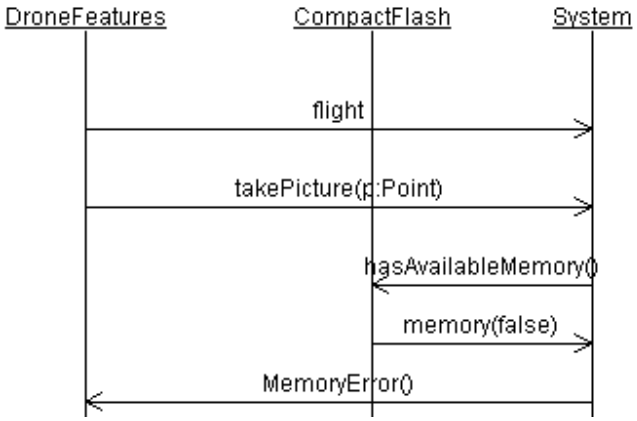
Since it is asked to model the SW managing pictures of the drone, main functions are "managing pictures" (accepting orders on pictures) which includes taking pictures and sending pictures. To take a picture, the system is in interaction with an external *Camera* and a *CompactFlash* system. To send a picture the system interacts with the C*ompactFlash* system, and the *DroneFeatures*. *DroneFeatures* include transmitting and receiving data from and to the ground, managing the GPS (the drone uses its GPS for other purpose, obviously).

b) Make two scenarios, one for the nominal case, and one for a non-regular case. (1 point)



On this scenario, we show a regular trace that consists first in the activation of the picture management (flight message), then, in taking a picture. Then, the picture is saved on card. When requested, data are first fetched from card, and then transferred to the drone. Once the drone has landed, picture management is no more functional. Also, this scenario doesn't show what happens if several concurrent orders are received: this issue is addressed at state machines diagrams level.

Error scenario may be based on a memory full error. I have slightly modified previous scenario to show how this could be taken into account:



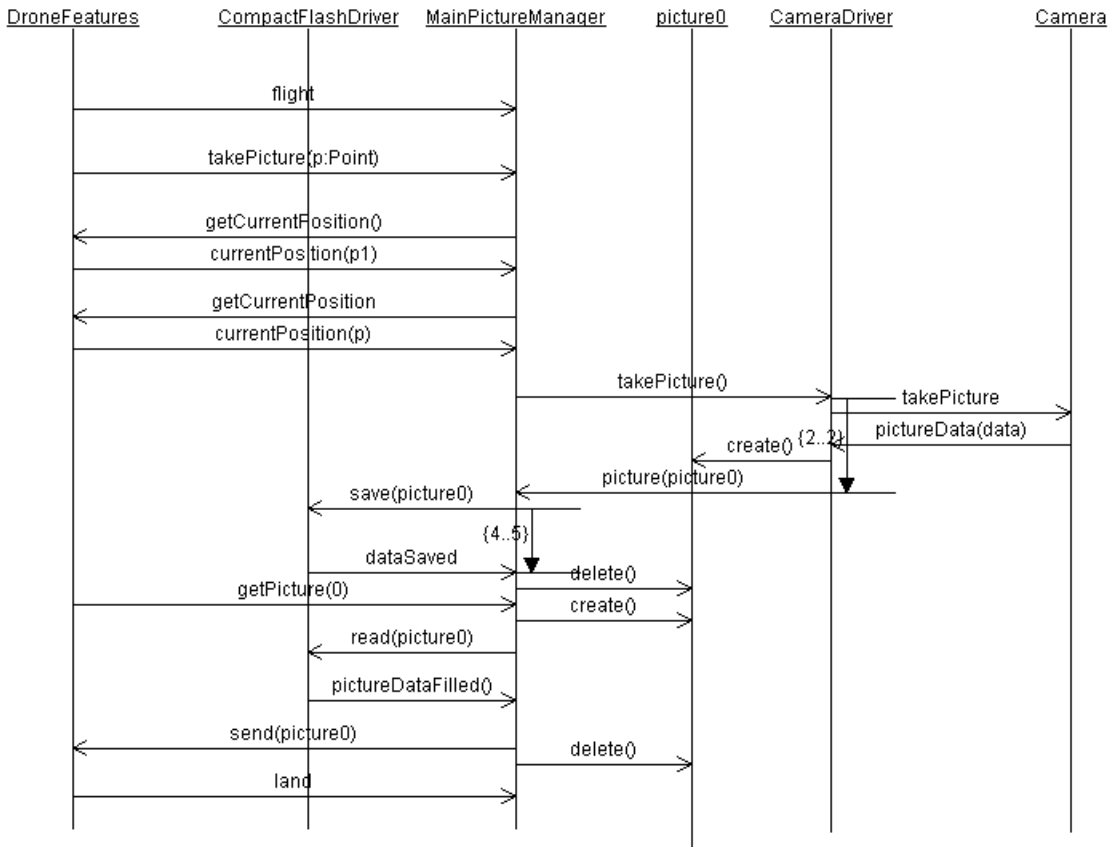
c) Using the technique of "words in the text", propose a collection of classes and objects for this system. (2 points)

drone. aircraft. pilot. take off. fly. land. Remote control. to take pictures. Pictures. picture-order. GPS position of the picture. Current position. Integrated GPS. GPS point is reached, given margin, stored. CompactFlash removable storage system.	Drone features : drone, aircraft, take off, land, remote control, current position, integrated GPS, GPS point is reached Compact flash: CompactFlash removable storage system Picture: Picture management function: 2 seconds to take a picture. Between 4 and 5 seconds to store.
Drone features, compact Flash	

From actors identified at first analysis step, and from words, I have identified four classes (we may add a GPSPoint class, that I have omitted to simplify the design):

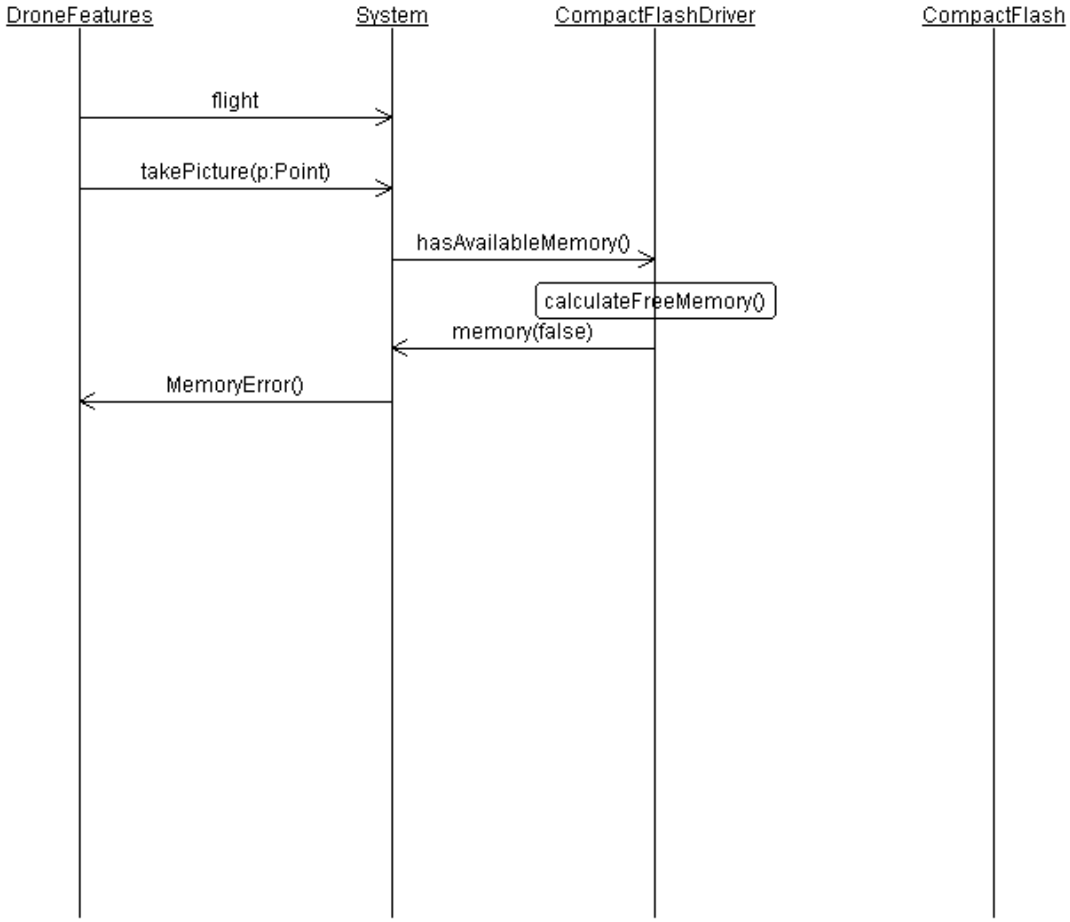
- PictureMainManager. attributes : Picture; Methods: takePicture(Point P), save Picture, and readPicture, and methods related to GPS : getCurrentPosition():p
- CompactFlashDriver : save(int, data), read(in):data
- CameraDriver : take():data
- Picture : attributes : id, data

d) Refine the two previously performed scenarios. (2 points)



A few comments:

- I have skipped the CompactFlash actor since messages between the driver and the real hardware are equivalent – on this diagram -.
- When a picture is taken, a picture object is created, then stored on compact flash, and then deleted. When a request for a picture is performed, a picture object is created, data are read from compact flash and put into that object, and the picture is sent to the requester. Once sent, the picture object may be deleted.

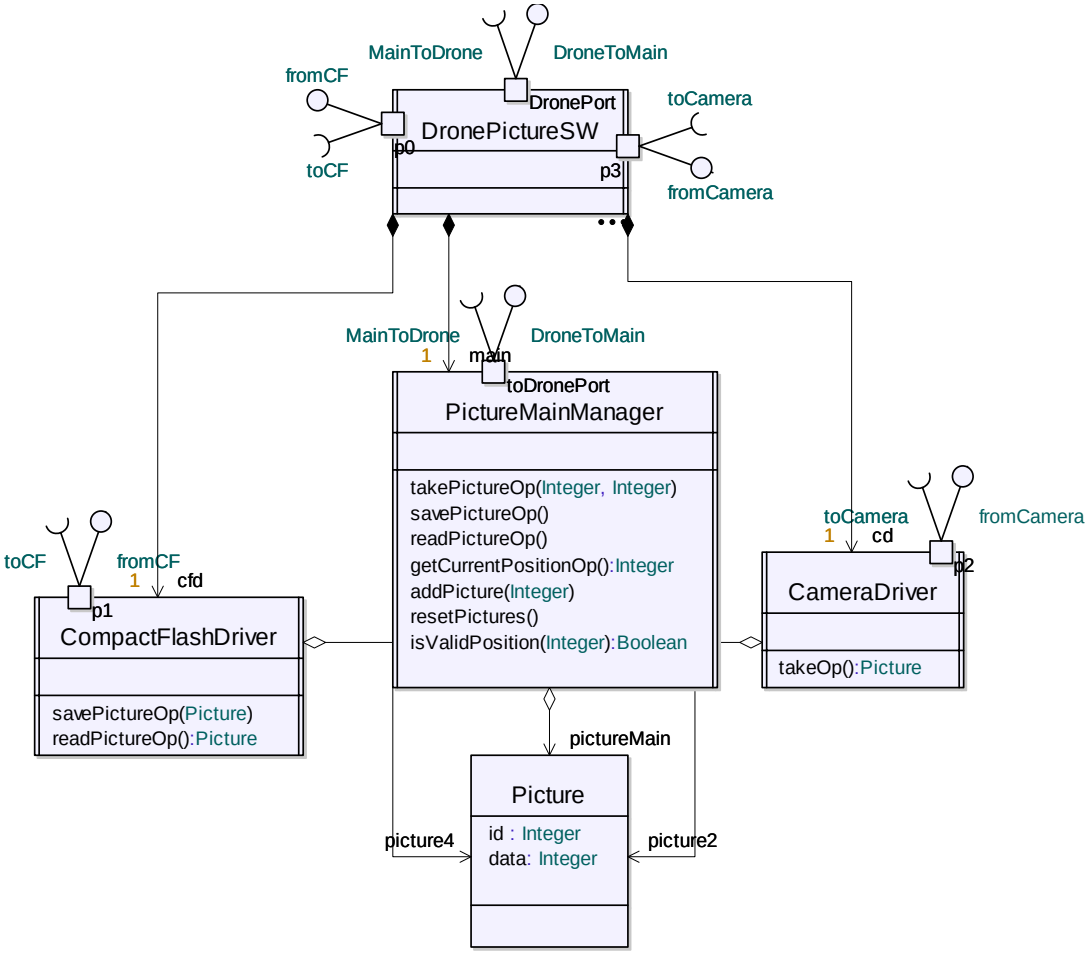


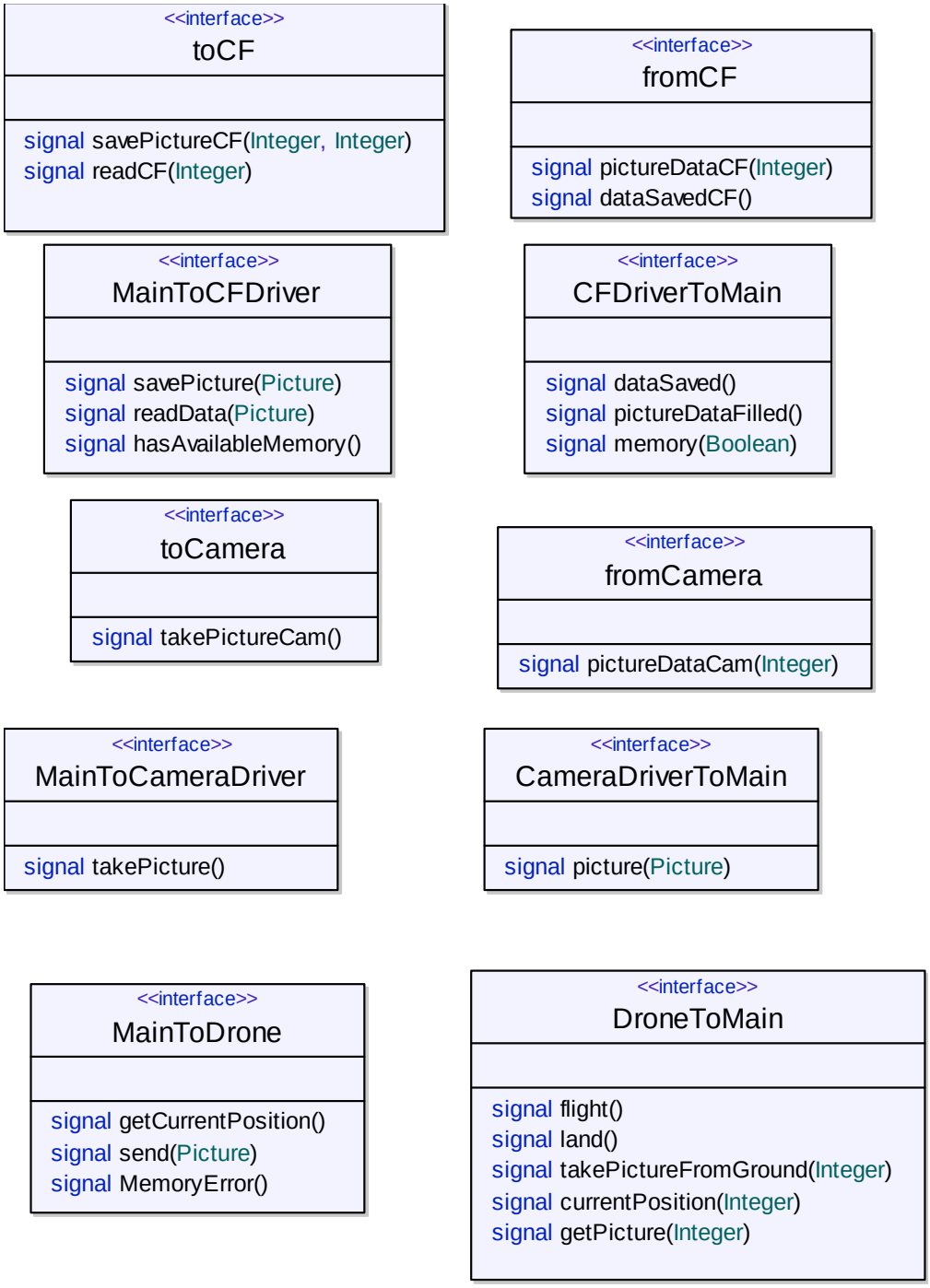
In this error scenario, one can notice that the hardware part of the CompactFlash system is not involved in the free memory management.

2) Design **~35mn**

a) From your analysis diagrams, propose a class diagram containing class relations such as associations, aggregations and so on, and also multiplicity. (1.5 points)

For simplicity reason, in the exam, you may omit the part related to saving pictures in design only (not in analysis).
On the class diagram is displayed ports related to environment. Also, we have added a class called "DronePictureSW". It is facultative. We have added operations to PictureMainManager, to deal with several picture orders at the same time (see corresponding state machine).

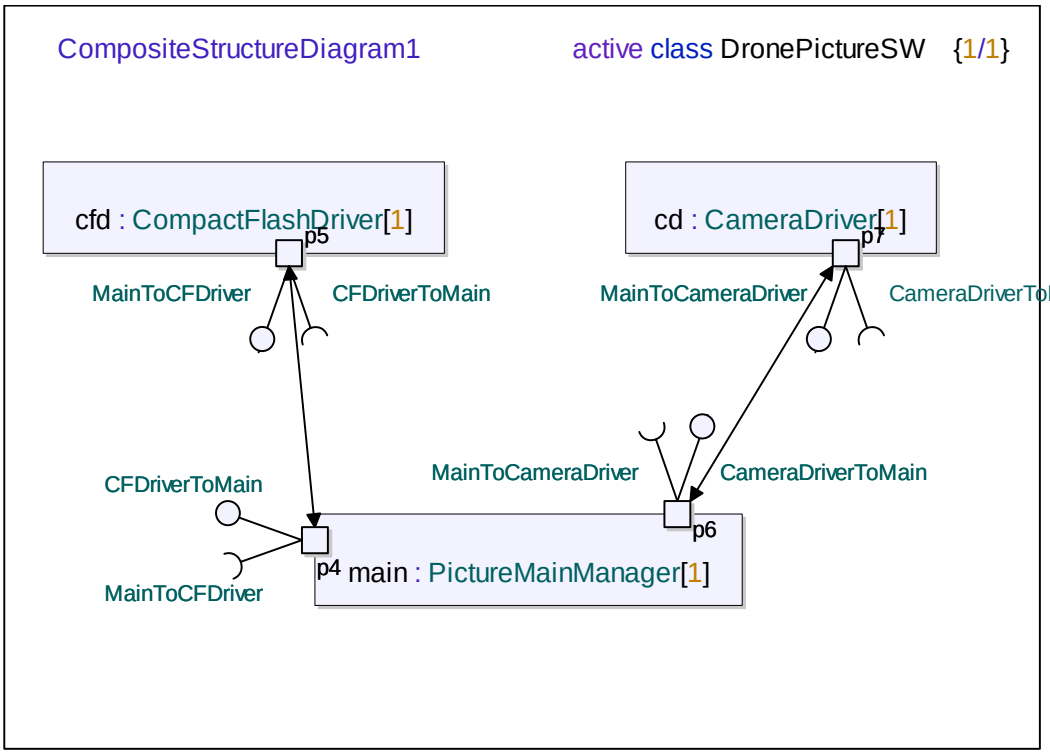

MainToDrone
DroneToMain
fromCF
DronePort
toCamera
DronePictureSW
p0
p3
toCF
fromCamera
MainToDrone
1
DroneToMain
main
toDronePort
PictureMainManager
takePictureOp(Integer, Integer)
savePictureOp()
readPictureOp()
getCurrentPositionOp():Integer
addPicture(Integer)
resetPictures()
isValidPosition(Integer):Boolean
toCamera
1
cd
fromCamera
p2
CameraDriver
takeOp():Picture
toCF
fromCF
1
cfd
p1
CompactFlashDriver
savePictureOp(Picture)
readPictureOp():Picture
pictureMain
Picture
id : Integer
data: Integer
picture4
picture2



Some signal names have been changed with regards to analysis to ensure non-duplicate names.

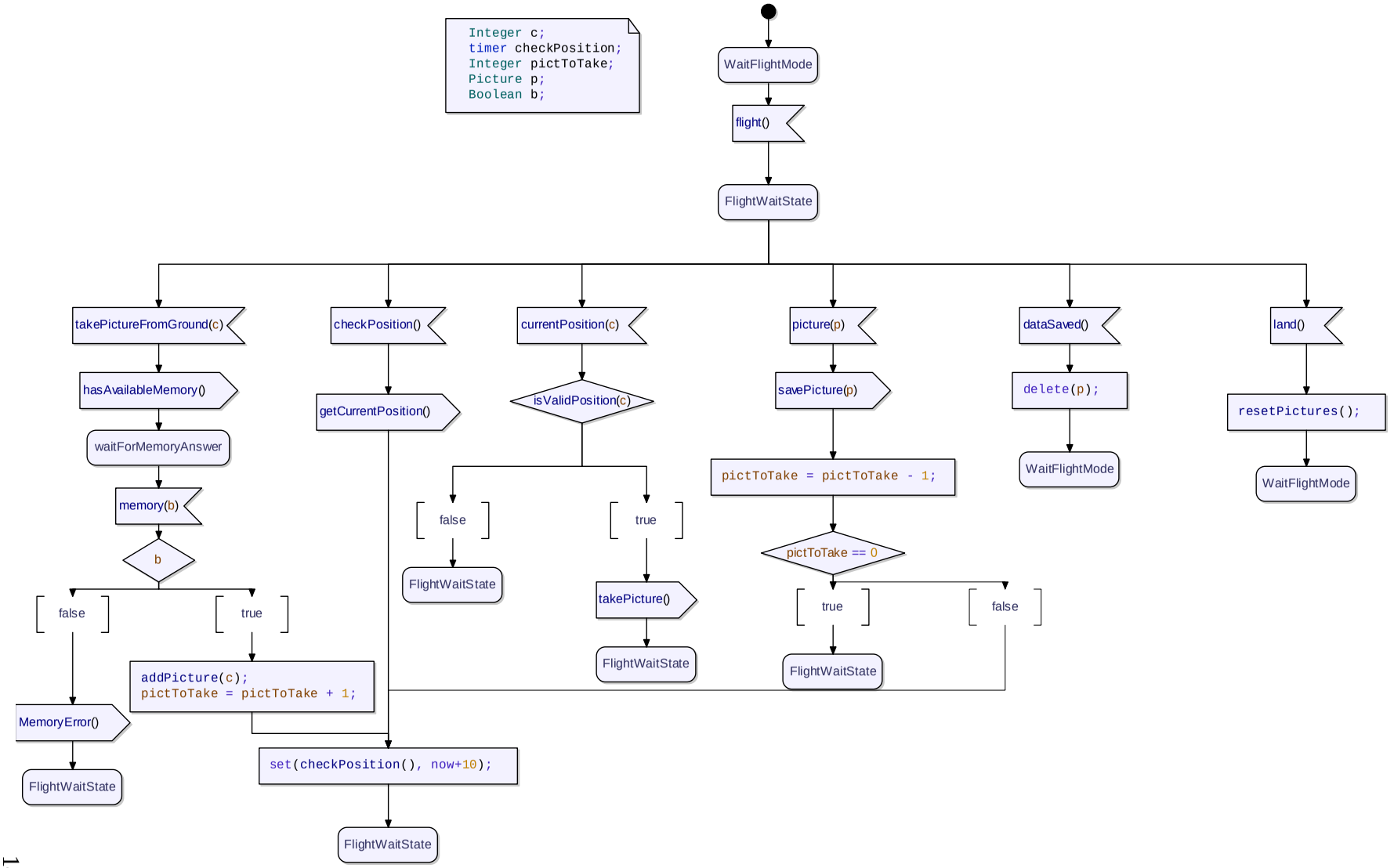
b) Perform the composite structure diagram of this system. Your diagram should model communication channels between system entities. (1.5 points)

The composite structure diagram is the one of *DronePictureSW*. It shows only internal ports.



c) Make the state diagram of the most important / complex class of your system. (2 points)

Hyp: no pictures can be saved at the same time. We have made a state machine based on one important state called FlightWaitState. Once again, for simplicity reason, in the exam, you may omit the part related to saving pictures.


Integer c;
timer checkPosition;
Integer pictToTake;
Picture p;
Boolean b;
WaitFlightMode
flight()
FlightWaitState
takePictureFromGround(c)
hasAvailableMemory()
waitForMemoryAnswer
memory(b)
b
false
true
MemoryError()
FlightWaitState
addPicture(c);
pictToTake = pictToTake + 1;
checkPosition()
getCurrentPosition()
set(checkPosition(), now+10);
FlightWaitState
currentPosition(c)
isValidPosition(c)
false
FlightWaitState
true
takePicture()
FlightWaitState
picture(p)
savePicture(p)
pictToTake = pictToTake - 1;
pictToTake == 0
true
FlightWaitState
false
dataSaved()
delete(p);
WaitFlightMode
land()
resetPictures();
WaitFlightMode