IN2013, Week 1 – Use Case Realization

Answers to Tutorial

Dr Peter Popov

3rd October 2018

Scenario

Consider the BAPERS case study and the specifications of the use cases given in Appendix 1 and the class analysis class diagram provided in Appendix 2.

Question 1 (Sequence diagrams with combined fragments)

Using your 'first cut' analysis class diagram (e.g. the one given in Appendix 2), draw a sequence diagram for the use case "PlaceOrder". Concentrate on the main flow and ignore the alternative flows.

Answer: The point of this exercise is for you to realize that an ***analysis class diagram*** *may change as a result of developing a realization of the chosen use cases- the initial (first cut) analysis class diagram will change and new operations may become needed. (UML tools will assist you keep the class and sequence diagram consistent. The tools 'see' both models – the class the sequence diagram).*

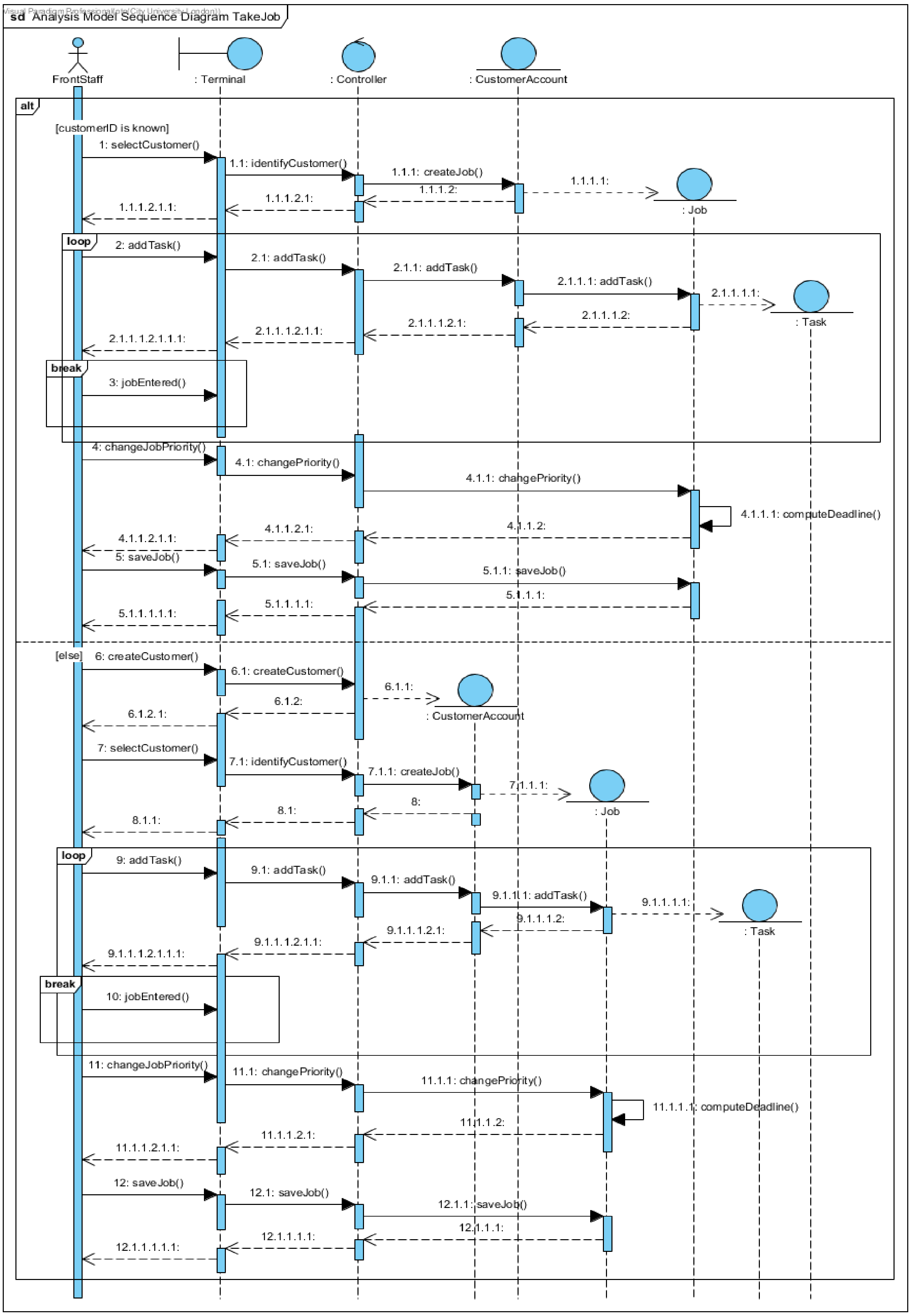
In this case, I would expect you to realize that quite a few new operations will be needed in the boundary class and in the control class – compare the class diagrams presented in this model answer with the class diagram provided in the assignment.

The sequence diagram for the use case will look as shown below. I intentionally have shown the return messages (which are optional) so that the flow of messages is clear and the flow of control between the lifelines (i.e. objects) is clear.

Note the use of combined fragments, ***loop*** *and* ***alt****. The life lines are shown using the built-in icons for class stereotypes (<<boundary>>, <<control>> and <<entity>>).*

Note also the use of the break combined fragment. It takes you out of the loop combined fragment. The loop is implicitly infinite and will be interrupted by the user selecting jobEntered(), which one would interpret as some sort of visual control in the GUI, e.g. pressing a button "Done". This message HAS to be used to interrupt the loop. An alternative would be to state explicitly the maximum number of tasks that one can add to a job, but according to the initial statement of requirement, BAPERS, the number of tasks which belong to a Job may vary. The important point here is to understand that jobEntered() MUST be used or the loop will continue indefinitely.

Notice also the syntax of the break combined fragment, which gets us outside the loop! This fragment should not be entirely inside the loop. Instead it should partially overlap with the loop combine fragment.


Visual Paradigm Professional(ate(City University London))
sd Analysis Model Sequence Diagram TakeJob
FrontStaff
: Terminal
: Controller
: CustomerAccount
alt
[customerID is known]
1: selectCustomer()
1.1: identifyCustomer()
1.1.1: createJob()
1.1.1.1:
: Job
1.1.1.2:
1.1.1.2.1:
1.1.1.2.1.1:
loop
2: addTask()
2.1: addTask()
2.1.1: addTask()
2.1.1.1: addTask()
2.1.1.1.1:
: Task
2.1.1.1.2:
2.1.1.1.2.1:
2.1.1.1.2.1.1:
2.1.1.1.2.1.1.1:
break
3: jobEntered()
4: changeJobPriority()
4.1: changePriority()
4.1.1: changePriority()
4.1.1.1: computeDeadline()
4.1.1.2:
4.1.1.2.1:
4.1.1.2.1.1:
5: saveJob()
5.1: saveJob()
5.1.1: saveJob()
5.1.1.1:
5.1.1.1.1:
5.1.1.1.1.1:
[else]
6: createCustomer()
6.1: createCustomer()
6.1.1:
: CustomerAccount
6.1.2:
6.1.2.1:
7: selectCustomer()
7.1: identifyCustomer()
7.1.1: createJob()
7.1.1.1:
: Job
8:
8.1:
8.1.1:
loop
9: addTask()
9.1: addTask()
9.1.1: addTask()
9.1.1.1: addTask()
9.1.1.1.1:
: Task
9.1.1.1.2:
9.1.1.1.2.1:
9.1.1.1.2.1.1:
9.1.1.1.2.1.1.1:
break
10: jobEntered()
11: changeJobPriority()
11.1: changePriority()
11.1.1: changePriority()
11.1.1.1: computeDeadline()
11.1.1.2:
11.1.1.2.1:
11.1.1.2.1.1:
12: saveJob()
12.1: saveJob()
12.1.1: saveJob()
12.1.1.1:
12.1.1.1.1:
12.1.1.1.1.1:

Question 2. Modular Sequence diagram (use of combined fragment 'ref')

You will notice that there are ***two important flows*** to address (handled by the extension use cases):

i) when the CustomerID is known at the time of placing an order and

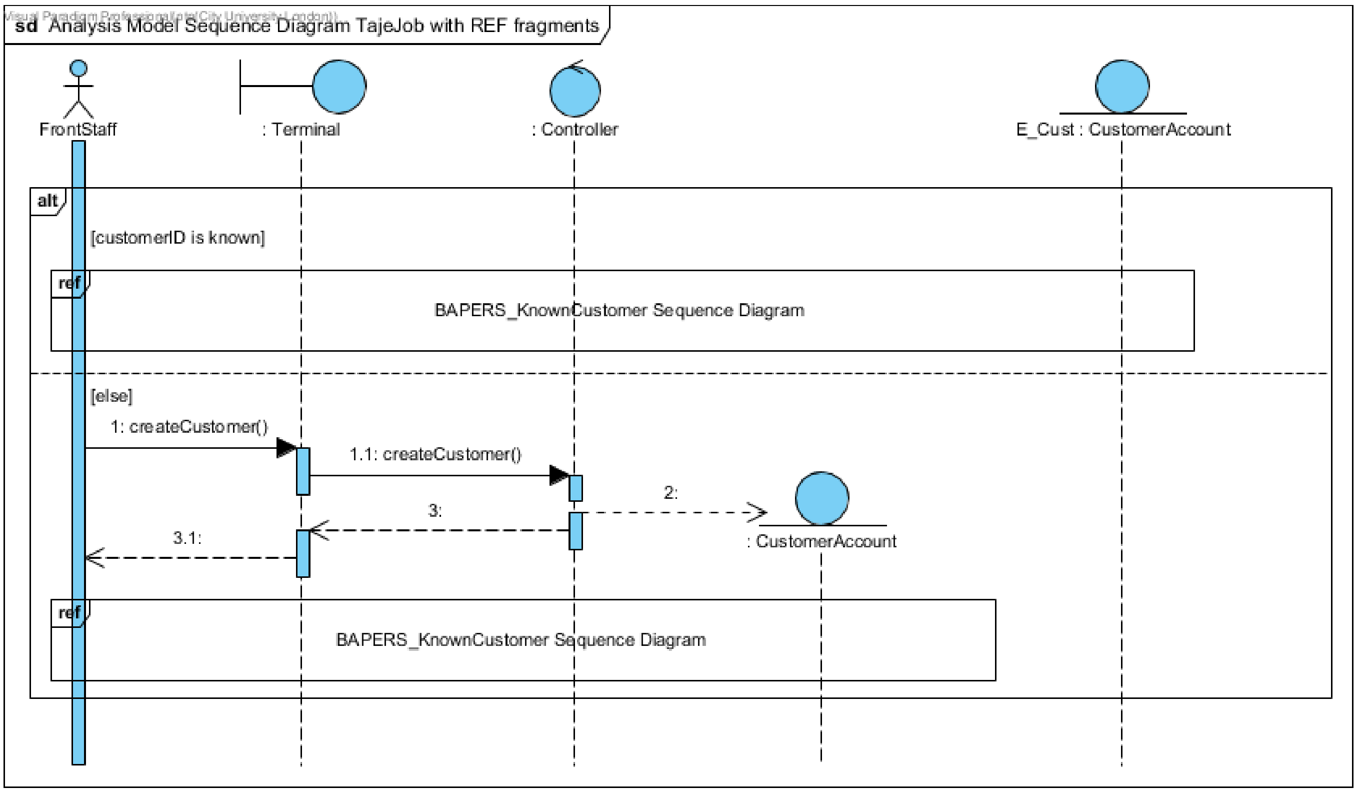
ii) when the Customer is not known and a new CustomerAccount must be created before the job can be created.

The two flows differ only in the part of creating a CustomerAccount. Once the customer is established the flow of creating a job is the same.

Redesign the sequence diagram to make use of the combined fragment **ref**, which allows one to separate out the common interaction.

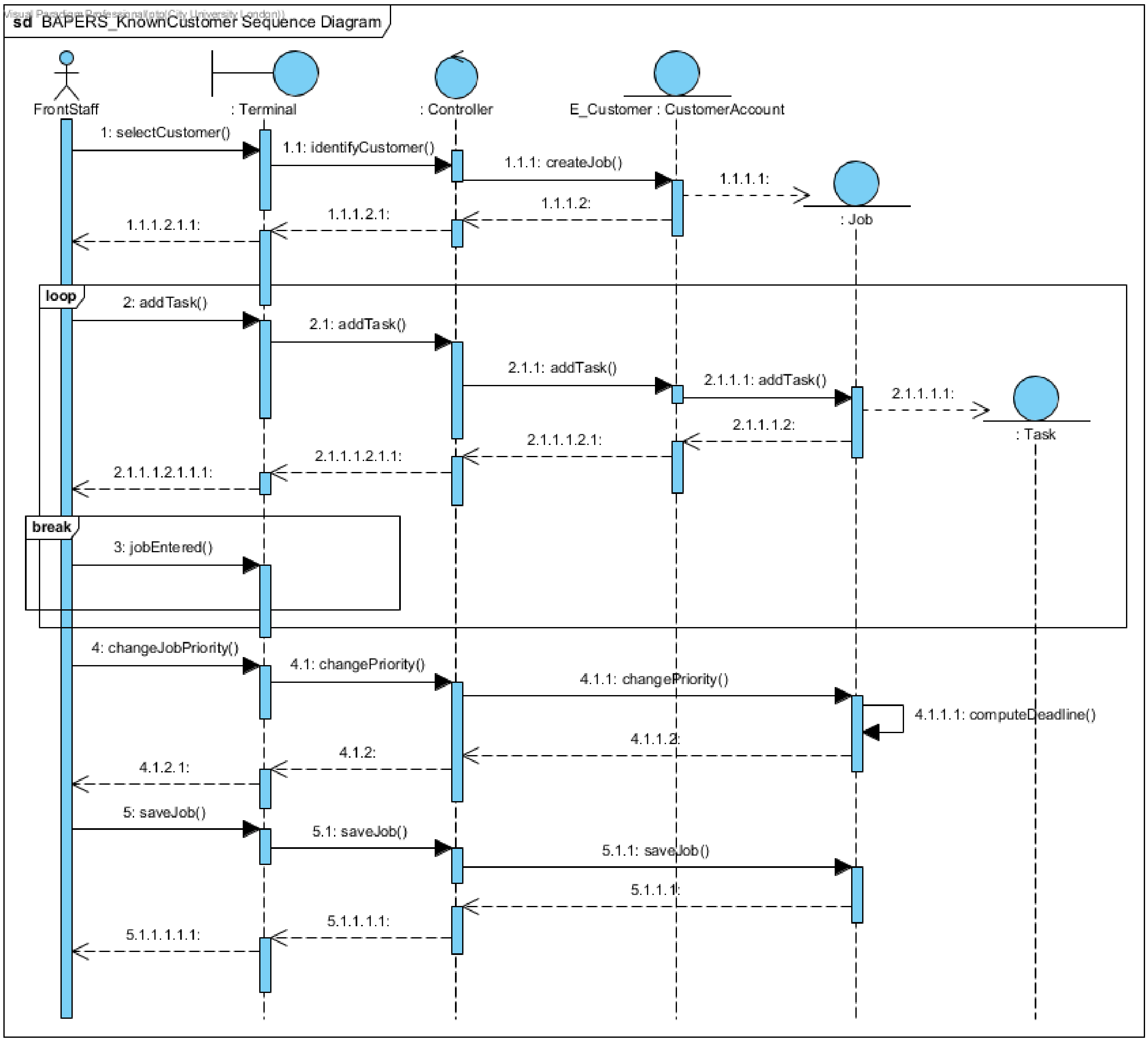
Answer:

The sequence diagram now is presented below.



Visual Paradigm Professional(Intel City University London)
sd Analysis Model Sequence Diagram TajeJob with REF fragments
FrontStaff
: Terminal
: Controller
E_Cust : CustomerAccount
alt
[customerID is known]
ref
BAPERS_KnownCustomer Sequence Diagram
[else]
1: createCustomer()
1.1: createCustomer()
2:
3:
3.1:
: CustomerAccount
ref
BAPERS_KnownCustomer Sequence Diagram

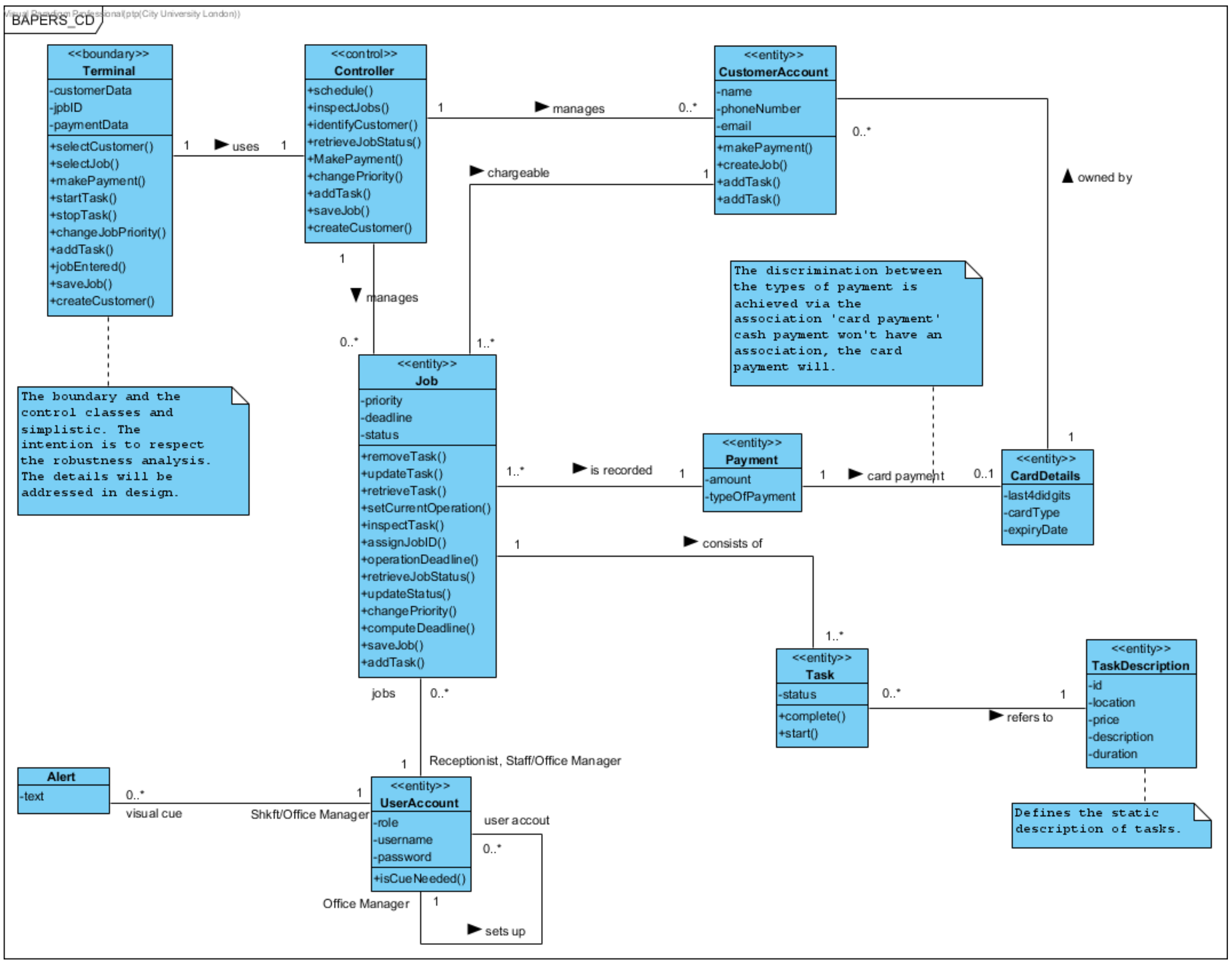
The common functionality is shown using a combined fragment ref (called in the VP tool – interaction Use). The interaction use, BAPERS_KnownCustomer is shown below.



As a result of the analysis and building the sequence diagrams, the <u>*class diagram*</u> ***has evolved too*** *and new operations were added to classes as needed****,*** *e.g. the operation CreateJob, which we used in the sequence diagram (message 1.1.1 in the first sequence diagram) has been added to the class CustomerAccount.*

Below we show the resulting class diagram. Compare the two versions – the initial version provided in Appendix 2 and the one provided here and make sure that you understand how they differ (new operations were added to the boundary class, Terminal, the control class, Controller, and the class Job).

The Visual Paradigm (VP) tool will help you keep the class and sequence diagram consistent: every time an operation is used in the sequence diagram which is not defined in the class diagram, VP will modify the class diagram by adding the new operation. This is an example of one of the benefits from using a UML tool.



The new class diagram also fixes a number of inconsistencies between the sequence and 1st cut class diagram (provided in the assignment). If two lifelines (objects) communicate by exchanging messages, links between these objects are needed, which in turn implies that the respective classes must be associated in the class diagram. We have two examples of inconsistency:

- *The lifelines :Terminal and :Controller exchange many messages and yet in the class diagram above, these two classes are not associated. The relationship between them is merely one of dependence, which is weaker than an association and merely implies that changes in the definition of the source class may require changes in the definition of the destination class. Given the fact that messages are exchanged between the lifelines :Terminal and :Controller, dependence has been replaced by an association.*
- *Similarly, the lifelines :Controller and :Job exchange message, hence these classes in the refined class diagram are now associated (i.e. an association between them was added).*

Unfortunately, VP ***does not help*** *with the last two problems. It is the modeler responsibility to enforce consistency by replacing manually the dependencies relationship with a simple association or adding associations between classes as required by the flow of messages in the sequence diagrams.*

In summary, I hope it is clear from this exercise that by developing sequence diagrams we are indeed undertaking an "analysis", which may lead to a refined class diagram.

Additional exercises:

1. The sequence diagram so far has provides a realization of the ***main flow*** only. Extend it to deal with the identified ***alternative flows***.
2. Develop a use case specification for ‘MakePayment’ and develop a sequence diagram for it.
3. Draw the communication diagram for the above use cases.

3rd October 2018

Appendix 1. Use case specifications

ID: TM02	**Use case:** PlaceOrder

Brief description:

The Customer brings in photographic material to be processed in “The Lab”. A new Job sheet is made out with a unique ID, which lists the tasks to be performed, sets the completion time. The job is linked to a CustomerAccount. If necessary a new Customer Account is created.

Primary actors:

FrontStaff (Receptionist, Shift or Office Manager)

Secondary actors:

None

Preconditions:

1. BAPERS is operational, the primary actor has logged in.

Flow of events:

1. Actor activated the functionality for creating a new job

   Extension point: SearchCustomer

   Extension point: CreateCustomerAccount

2. Actor selects the chosen CustomerAccount.
3. The system assigns a unique ID to the Job and links the job to the chosen Customer account.
4. Until more tasks are to be added:
   5.1. The actor selects a task from the list of available tasks and adds it to the job.
5. The actor selects the urgency of the job (either “urgent” or “normal”).
6. The system records the job completion time.
7. The system computes the price of the job.
8. The actor confirms that the job is completed.
9. The system prints a label (to be attached to the photographic material) which with the job ID and a collection note[1] (with the job ID, the Customer name and the job completion time).

Postconditions:

1. The system has recorded a new job with its ID, Customer who placed it, urgency, completion time, price and list of tasks to be completed.
2. The system printed a label and a collection note.

Alternative flow:

NoCustomerAccountProvided

PrintingFailure

[1] This note is handed over to the customer as a proof that the photographic material has been collected for processing.

Alternative flow: NoCustomerAccountProvided
ID: TM02.1
Brief description: No CustomerAccount is provided.
Primary actors: FrontStaff
Secondary actors: None
Preconditions: The session does not contain a valid CustomerAccount.
Alternative flow: 1. The system prompts the actor that no valid CustomerAccount has been chosen. 2. The actor acknowledges the notification 3. The system removes the session.
Postconditions: None.

Alternative flow: PrintingFailure
ID: TM02.2
Brief description: Printer fails to print the required document: either is out of paper or some other problem occurs.
Primary actors: FrontStaff
Secondary actors: None
Preconditions: 1. The printer is out of paper or a technical fault has accured.
Alternative flow: 4. The alternative flow starts after step 9 of the normal flow 5. BAPERS displays a message: “Printer fault”. 6. If further details are known about the problem   The system prints further details.
Postconditions: None

Use case ID: TM03	**Use case:** CreateCustomerAccount
Primary actors: FrontStaff	
Secondary actors: None	

Segment 1 Precondition (extension point: CreateCustomerAccount**)**
No customer account exists in BAPERS.

Segment 1 flow:

1) Actor activates the functionality for creating new customer account.
2) The system places an empty customer record in the session.
3) The system opens a form for entering the customer details such as Name, phone number and email address.
4) The actor types in the required details and confirms their validity.
5) The system creates a new customer record in BAPERS.
6) If the system succeeds
   it places the customer details in the session by adding a 'current customer' record in the session.
   Else
   Places an empty 'current customer' record in the session.

Postcondition:

1. CustomerAccount details are stored in the 'current customer' record in the session.

Use case ID: TM04	**Use case:** SearchCustomer

Primary actors:
FrontStaff

Secondary actors:
None

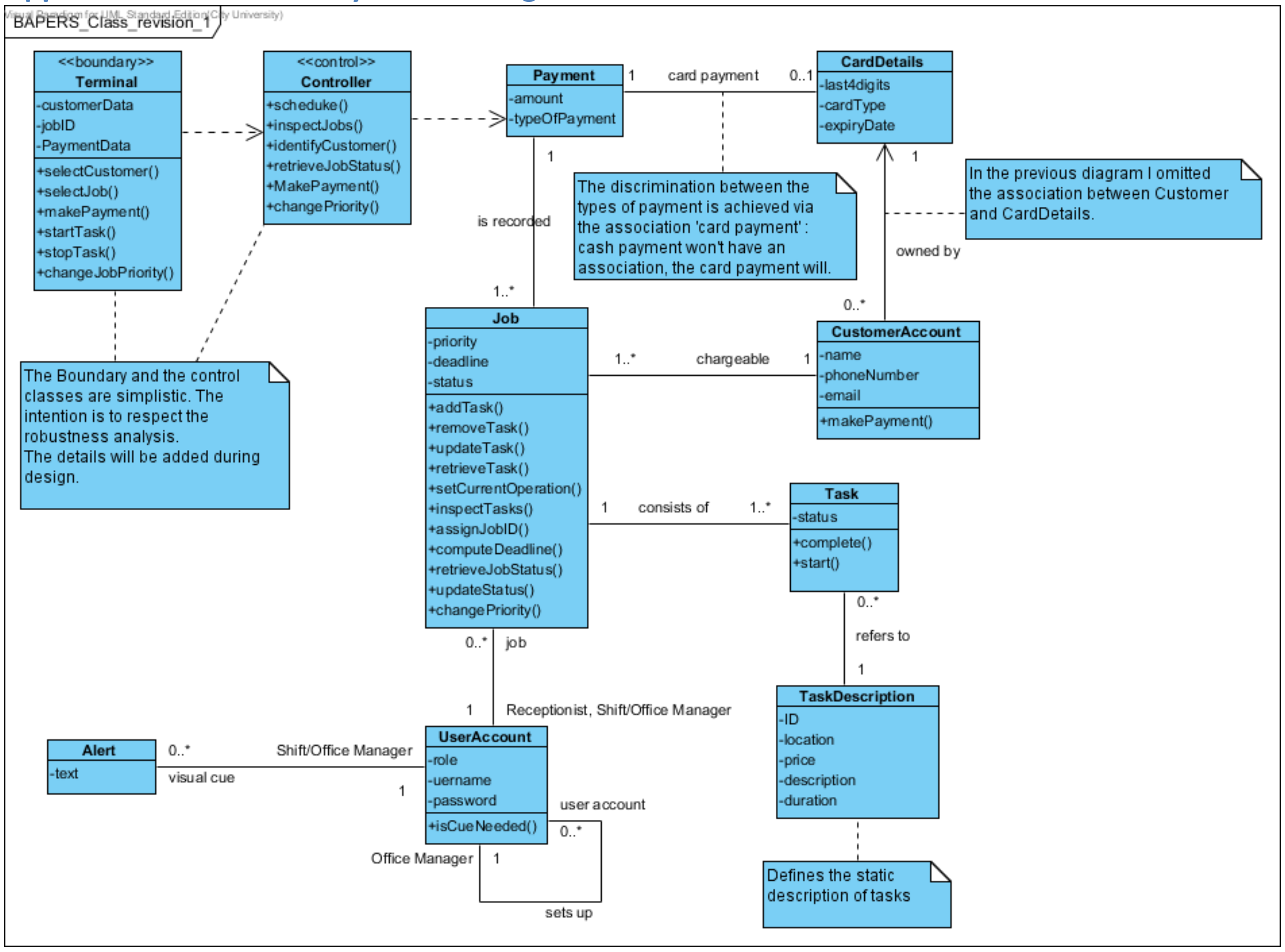
Segment 1 Precondition (extension point: SearchCustomer**)**
Customer chooses the CustomerAccount search functionality by activated it.

Segment 1 flow:

1) The system offers a search form for customer to specify the search criteria (e.g. name, email, phone number or any combination of these)
2) While no more than on record is found
   a. The actor specifies the search criteria (any combination of the three listed above will be acceptable) and confirms the validity of the search criteria.
   b. The system undertakes a search based on the search criteria.
3) If a single customer record is found
   it is placed in the 'current customer' record in the session.
   Else
   An empty record 'current customer' is placed in the session.

Postcondition:
CustomerAccount details are stored in the 'current customer' record in the session.

Appendix 2: BAPERS analysis class diagram



The Boundary and the control classes are simplistic. The intention is to respect the robustness analysis.
The details will be added during design.

The discrimination between the types of payment is achieved via the association 'card payment' : cash payment won't have an association, the card payment will.

In the previous diagram I omitted the association between Customer and CardDetails.

Defines the static description of tasks