UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2004

MSc in Computing Science
for Internal Students of the Imperial College of Science, Technology and Medicine

PAPER M225

SOFTWARE ENGINEERING

Friday 30 April 2004, 14:30
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

Section A *(Use a separate answer book for this Section)*

1 Consider the following aspects of a conference management system.

A conference selects a number of position papers and full papers for presentation from a number of submissions it receives by a set deadline. Each submitted paper has one or several co-authors. The conference has a program chair that appoints several program committee members (PC-members), who review the submitted papers. Each PC-member submits a separate review for each paper assigned to him/her by the program chair. Position papers have a maximum length of 4 pages and must be reviewed by at least 3 different PC-members. Full papers have a maximum length of 10 pages and must be reviewed by at least 4 PC-members. Reviews comprise a mark and comments. Once a review is entered, other PC-members may add additional comments to that review. The name of the PC member who has reviewed a paper should not be mentioned explicitly in the review. Instead, each PC-member is given an anonymous identifier which is used in all the reviews and comments that he/she enters. When all the reviews for a paper are completed, the program chair can decide whether the paper is accepted or rejected. For each user (author, PC-member, etc.) the system must record their affiliation and email address.

a i) Draw a use-case diagram for the conference management system. Indicate actors and overall use-cases.

ii) Draw a class diagram representing a software design model for the system described above. Indicate classes, their attributes, relationships, cardinalities and role names. Methods are **not** required. Briefly explain your model.

b i) PC-members assigned to review a paper cannot be from the same institution as any of the authors. Draw a sequence diagram indicating the invocations that occur when the `AssignReviewer(PCmember m,Paper p)` method is invoked to assign a PC-member as a reviewer to a paper.

ii) Briefly explain the attribution of responsibilities in your answer to bi). State clearly any assumptions you make.

c Briefly explain how the following would be implemented in your system:

- Submission of a new paper.
- Identification of *conflicts* (i.e., where the reviews diverge by more than 4 points in their marks) and notifying all the reviewers assigned to that paper.

The three parts a, b and c carry, respectively, 45%, 40%, 15% of the marks.

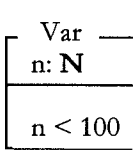
Section B (Use a separate answer book for this Section)

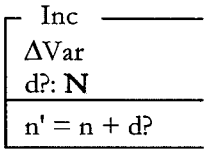
2 This question is about state-based specifications

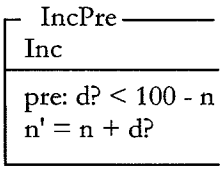
a i) What is a *model* of a schema? (Assume for simplicity that the schema has no base sort, i.e. no symbols in square brackets after the schema name.)

ii) What is the distinction between *state schemas* and *operation schemas*? Explain how the three symbols ', Δ and Ξ are used in operation schemas.

iii) Consider the following schemas. Write the operation *IncPre* in full, without using any schema inclusion. Using the defensive approach, write a disjunctive operation schema *IncDef* that captures the same models as *IncPre*. The schema *IncDef* should be written in full and without using schema inclusions







b A system is used to handle accounts in a bank. New accounts can be added to the bank or deleted from the bank. Each account is associated with a *number*, a *balance* and a *type*. For each account, the number gives a sequence of no more than 10 digits, the balance does not exceed £500,000, and the type can be either "checking" or "saving". The bank can have no more than 2500 accounts open. The bank is said to be in the state *OnlySavings* if all the open accounts are saving accounts, otherwise it is said to be in the state *Various*. New saving accounts can be added to the bank only when the bank is in state "Various", whereas new checking accounts can be added whatever state the bank is in. When a new account is added, its balance is assumed to be equal to 0.

Assume the existence of a sort "Accounts" and a sort "Type" = {checking, saving}.

i) Write a state schema *AccountsInfo* that specifies for each account the number, balance and type and related constraints as described above.

ii) Write a state schema *System* for the overall state of the system, covering the two cases when the bank is in state *OnlySavings* or in state *Various*. In both cases the schema includes *AccountsInfo*, and uses the variable *accounts* to keep track of the number of accounts open in the bank. Include appropriate constraints.

iii) Write operations schemas for the following operations using a defensive approach. The first two operations include three input variables *newaccount*, *accnum*, and *acctype*, and the third operation includes one input variable *account* and an output variable *balance*. Marks will be awarded for correct use of Δ and Ξ.

AddCheckingAccount adds a new checking account to the bank when possible. Otherwise it returns an error.

AddSavingAccount adds a new saving account to the bank when possible. Otherwise it returns an error.

QueryBalance returns the balance of an account given in input, if it exists in the bank. Otherwise it returns an error.

iv) Consider the operation *AddAccount* = *AddCheckingAccount* ∨ *AddSavingAccount*. Is this operation total? Explain your answer.

The two parts carry, respectively, 40% and 60% of the marks.

3 This question is about object-oriented specifications using ObjectZ

a i) Define what the components of a class schema are and discuss their role.

ii) Define the operators for objects interaction and inter-object communication in ObjectZ and describe their semantics. Describe the differences between the *parallel* and the *sequential composition* operators.

b) The control tower of a local airport uses a system to control the landing and taking off of planes at the airport. The airport has 10 runways. The system uses a class *Plane* and a class *Runway*. Each plane includes a maximal *capacity* of seats, a *status*, which can be either full or empty, and the number of *seats* occupied. A runway can be either occupied or free, and has a *maxwaiting* number of planes allowed to wait to land on that runway. The system includes a collection of planes and 10 runways. Each runway has a collection of planes waiting to land, called *waitinglanding*, and a collection of planes waiting to take off called *waitingtakingoff*. Initially, all runways are free, the collections *waitinglanding* and *waitingtakingoff* are empty, and all planes are empty. The *RequestLanding* and *RequestTakingOff* operations of the system have as input a plane and a runway. The landing is granted when the runway is free and there are no planes waiting to land on that runway. When the runway is busy, either the plane requesting to land is added to the collection of planes waiting to land or the request landing is denied. The *RequestTakingOff* operation, instead, allows planes to take off only after they have become full and the runway is then free. Otherwise the request is denied.

Write an ObjectZ class schema for each of the following classes, using a local control approach to specify the associations between a runway and the planes waiting to land to and to take off from that runway. Each class schema has to include a visibility list, a state schema, an initial schema, and the operations schemas described below:

i) The class *Plane* includes one operation *HasBecomeFull* that changes the status of the plane from empty to full. It should also include an operation for providing in output the "self" identity.

ii) The class *Runway* includes the following operations. *LandingGranted* to check there are no more planes waiting to land; *WaitingAuthorised* and *TakeOffAuthorised* to handle, respectively, the associations *waitinglanding* for a plane requesting to land, and *waitingtakingoff* for a plane requesting to take off. The operations *WaitingDenied* and *TakeOffDenied* to handle, respectively, the cases when *WaitingAuthorised* and *TakeOffAuthorised* are not allowed. The two operations *IsBusy* and *IsFree* to check whether the runway is busy or free.

iii) The class *ControlTower* includes two operations, called *RequestLanding* and *RequestTakingOff*, defined as described above. Specify these operation using appropriate ObjectZ operators. Do not specify them as full operation schemas.

The two parts carry, respectively, 40% and 60% of the marks.

4. **This question is about object-oriented specification and program verification.**

a i) Describe briefly the meaning of class invariants and post-conditions and how they are used when reasoning about the correctness of a program.

ii) Describe briefly the meaning of loop invariant and loop variant and how they differ. For what types of loop can a loop invariant be equivalently expressed as pre- and post-conditions?

b A home heating simulation program uses a module called *TemperatureAdjustment* to monit the temperature of a room. The module includes a fixed desired temperature, called *desiredtemp*, equal to 18, a fixed room size, called *roomsize*, equal to 6 (square meters), a *furnace* that is either on or off, and a current *temperature* value, which should be no more than 7 degrees below and not higher than 22 degrees above the desired temperature.

At the beginning the temperature is equal to the desired temperature and the furnace is off. The module uses an operation called *ComputeTemperature*, which measures and updates the current room temperature. The operation takes as input the *heatgained* (i.e. heat coming from the furnace) and the *heatloss* (i.e. room heat lost to the surrounding environment) and gives as output the new current temperature of the room, called *newtemp*. This is the sum of the current temperature and the difference between the heat gained and heat lost, divided by four times the room size. The operation called *FurnaceHandling* takes as input the temperature of the room and does one of the following actions. If the temperature is lower than the desired temperature and the furnace is off then the furnace is turned on. If the temperature is 3 degrees higher or more than the desired temperature and the furnace is currently on, then the furnace is turned off. Otherwise the module does nothing.

i) Write a class schema *TemperatureAdjustment* in ObjectZ that specifies the description given above. Include appropriate attributes and the operations *ComputeTemperature, FurnaceHandling*. Both operations are visible.

ii) Map the Object-Z class schema *TemperatureAdjustment* you have given in part (i) into a Java class *TemperatureAdjustment*, by including the declaration of the data fields of this Java class, the method headers, the class invariants, and pre- and post-conditions.

iii) Implement the method *FurnaceHandling* of the Java class *TemperatureAdjustment* from part (ii), including suitable mid-conditions and argue that the code you give satisfies the post-conditions of the method.

The two parts carry, respectively, 35% and 65% of the marks.