UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2004

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
BSc Honours Degree in Mathematics and Computer Science Part II
MSci Honours Degree in Mathematics and Computer Science Part II
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute
This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science

PAPER C220=MC220

SOFTWARE ENGINEERING - DESIGN I

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.

2a Briefly state what is meant by composite states in UML state charts, what kind of composite states there are, when they are used and what advantages they confer.

b The following is a description of a coin-operated public telephone which is connected to the telephone network through an exchange. The telephone allows users to make the calls but not to receive them.

The phone is initially hung-up and returns to this state whenever the handset is replaced. If the phone is not used for 2 minutes all credit remaining is reset. Whenever an error, insufficient credit or timeout are encountered the phone continuously sounds an error tone. The minimum cost of a call is 20 pence and the phone will take no action unless the minimum credit is reached.

After inserting the minimum credit required the phone connects to the exchange and rings a dial tone. The user has then 1 minute to enter the telephone number to call, otherwise a timeout will be received from the exchange. The telephone requests the exchange to dial the number when 3 seconds have lapsed and no digit has been entered. The exchange will notify the telephone if the number is valid or invalid. If the number is valid the call will be attempted. If the line is engaged, the telephone sounds the engaged sound and terminates the call, otherwise it sounds the ringing tone. If the other party has not answered within 30 seconds a timeout will be generated by the exchange and the call disconnected. When the other party answers, the call is connected and voice is transmitted. The call terminates when either party hangs up or the credit is 0.

At any point in time, users may insert additional coins to increase their credit. The exchange notifies the telephone whenever a unit has been spent through a UT event. If the current credit is 0, the telephone will disconnect the call otherwise it will decrement the user's credit.

You may assume that events can be generated using the following clauses:

`when(credit > 20)`

`after(time)` where time is expressed in seconds.

Draw a state chart modelling the behaviour of the coin-operated telephone. State explicitly any additional assumptions that you make.

c Considering the telephone described in 2b) as a single object draw an overview of the interactions with the user and the telephone exchange. Use the general representation of a sequence diagram but you may consider all messages to be asynchronous.

The three parts a, b and c carry, respectively, 25%, 50%, 25% of the marks.

3a Briefly state the purpose of the visitor pattern and explain how it works.

b A universal messaging centre connects a company's network to the outside world. It accepts general messages of the format below and can perform various actions such as routing the message, checking and removing viruses, or modifying the source or destination addresses. Note that the order in which the actions are performed is important.

```
class Message {
    String srcAdd;          // source address
    String destAdd;         // destination address
    String data;            // message data
}
```

New actions can be added and existing ones removed dynamically.

i) Explain which design pattern you would use to implement the messaging system described above and why would you choose it.

ii) Give a Java outline implementation which can perform the following actions in order:

- All messages originating from "hotmail.com" are dropped without further processing.
- All messages sent to `admin@imperial.ac.uk` should be sent to `tbll@imperial.ac.uk` and all the following actions are applied.
- All messages are delivered to destination and no further processing occurs. Assume a method `deliver(Message m)` exists.

c Consider the messaging system described in question 3b) above. Since defining the various actions to handle messages is difficult and error prone, there is a need for a tool which enables an administrator to dynamically manipulate the actions e.g., add a new action, remove an existing one or move an action to be evaluated on a message earlier or later than other actions (i.e., move it to a new position). The administrator must also be able to undo any of the configuration changes an unlimited number of times.

i) Explain which design pattern(s) you would use to implement this functionality and why you would use it (them). List their general advantages.

ii) Provide an outline Java implementation of the classes which would allow an administrator to:

```
remove x      // remove the xth action
undo          // undo the last action performed
```

You may assume the existence of any classes and functions you find useful but you must state clearly your assumptions.

The three parts a, b and c carry, respectively, 20% 35%, 45% of the marks.

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

4a 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.

```
┌ Var ──
│ n: N
├────────
│ n < 100
└────────
```

```
┌ Inc ────────
│ ΔVar
│ d?: N
├─────────────
│ n' = n + d?
└─────────────
```

```
┌ IncPre ─────────
│ Inc
├─────────────────
│ pre: d? < 100 - n
│ n' = n + d?
└─────────────────
```

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.