UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1999

MEng Honours Degrees in Computing Part IV
MEng Honours Degree in Information Systems Engineering Part IV
MSci Honours Degree in Mathematics and Computer Science Part IV
MSc Degree in Advanced Computing
for Internal Students of the Imperial College of Science, Technology and Medicine

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

PAPER 4.74 / I 4.8

MULTI–AGENT SYSTEMS

Wednesday, May 12th 1999, 2.00 – 4.00

Answer THREE questions

For admin. only:
paper contains 4 questions

1a Briefly explain Russell and Norvig's PAGE classification of intelligent agents, indicating why each component of the classification can be deemed necessary for an intelligent agent. Illustrate your answer by giving the elements of each component for a part-sorting robot which can recognise and classify each individual part on a tray, grasp an individual object, and remove it from the tray onto a cube-like or cylinder-like pile. Indicate how the software and hardware ingredients of a vision system, robot arm control and stimulus-response design relate to the external PAGE classification.

b Briefly explain the internal architecture of Shoham's AGENT0, indicating how the part-sorting robot of *part a above* would be designed. Suggest a generalisation where the additional structure of AGENT0 would be justified.

Parts a and b carry, respectively, 60% and 40% of the marks.

2a Briefly describe the key features of the Contract Net protocol for distributing tasks over a network of agents or problem solvers. Pay particular attention to:

i) the roles of manager and contractor,
ii) the contents of a task announcement,
iii) the use of information messages.

b Explain how a skill server agent, whose identity is known to all the agents, could be used to allow agents to join and leave the net and to facilitate more focussed task announcements.

c If bids had to include a task completion time, and a contractor were able to execute more than one task at a time:

i) What extra capabilites and knowledge would a contractor need?
ii) What modifications would be needed in the content of the messages?

d Suppose a manager M divides a task T into subtasks T1 and T2 which are awarded to contractors C1, C2. Suppose that C1 and C2 have to co-operate in solving C1 and C2. Suggest how this might be achieved within the standard contract net protocol.

The four parts carry, respectively, 40%, 20%, 20% 20% of the marks.

3a Suppose we want to have an information gathering mobile agent such that:

(1) there will be no communication with its parent process until it completes its task,

(2) when it has completed its task it will simply send a suitable message to its parent process and then terminate,

(3) it will be sent directly by its parent process to the first mobile agent station at which it is to gather information.

Briefly describe how such a mobile agent could be implemented in the April language. What support processes will be needed on the mobile agent stations it is to visit? Describe the functionality of these support processes.

[Turn over.........

b Based on your answer to part a, give the April program for the mobile agent described below which gathers details of papers that are relevant to a given set of keywords. Give the message send statement that the parent process uses to launch the agent.

When launched the agent is provided with:

> a list of keywords, `K`,
> the handle, `P`, of the parent process that launches it
> the list [S], where S is the handle of the first station to which it is sent

At each station it visits, it first consults the local skills server agent to find a local agent, `L`, with the skill, `'librarian`. The local skills server is identified using the public handle: `handle{name="skills_server"}`)

The message it sends the skill server is: `('request_one,'librarian)`

and the reply it receives is of the form: `('consult_for,'librarian,?L)`

(You can assume the agent will always receive such a reply.)

It then further consults `L` with a query message:

```
('papers_on, K) >> L
```

The answer is returned as a message:

```
('papers_for,K,?Papers)
```

The mobile agent stores the answer `Papers`, in a list `D` of pairs of the form, `(L,Papers)`, that it builds up on its travels.

To find where to visit next, for further papers relevant to `K`, it again consults `L` with a query message:

```
('suggestions_please,Visited,K) >> L
```

where `Visited` is a list of stations it has already visited. The reply will be a message of the form:

```
('try_for,K,?NextStation)
```

or a message :

```
'no_suggestions
```

If the reply is a `'try_for` message the agent moves on to `NextStation`, to repeat what it has just done. If the reply is `'no_suggestions`, the agent sends a message:

```
('have_found,D)
```

to the agent which dispatched it, `D` being the list of pairs it has been accumulating. It then terminates.

The two parts of this question carry equal marks.

[Turn over.........

4a Briefly explain the role of a *matchmaker* or *facilitator* in a distributed information system.

b Explain the role of the KQML performatives: *ask-about*, *advertise*, *recommend*, *subscribe*.

c The program below is an April program which when forked as a process can act as a simple matchmaker facilitator. It can handle

```
('advertise ... ('content, ('ask_about...))...) and

('recommend .... ('content,('ask_about...))...)
```

messages.

The program assumes that a KQML message of the form:

(performative_name :attr1 v1 :attr2 v2 .... :attrk vk)

is represented as the April tuple:

```
('performative_name,
    [('attr1,v1), ('attr2,v2),..., ('attrk,vk)]).
```

The program:

```
MM(){
   AskAbouts:= [];

   repeat {
       ('advertise, ?AdvAVList)::
           ('content, (ask_about, ?AskAbtAVList))
                   in  AdvAVList
               ->
           AskAbouts :=
               [(sender,AskAbtAVList),..AskAbouts]
       |
       ('recommend, ?RecAVList) ::
           ('reply_with, ?Label) in  RecAVList and
           ('content, (ask_about, ?AskAbtAVList))
                       in RecAVList
               ->
           ('reply,[(in_reply_to,Label),
                   ('content,
                      bagof{A:
                              (?A,?AVList) in AskAbouts
                              and
                              match(AskAbtAVList,AVList)
                          })])
               >> replyto
       } until 'quit
};

match(AskAbtAVList,AVlist) =>
    ('ontology,?O) in AskAbtAVList and
    ('ontology,O) in AVlist and
    ('content,?T1) in AskAbtAVList and
    ('content,?T2) in AVlist and
    {forall ?T in T1 then T in T2};
```

[Turn over.........

i) Give an April message that an agent could send to this matchmaker if it wanted to advertise the fact that it could be asked about any topic on the topic list: `['agents,'dps,'KQML]` using the ontology: `'computing_dai`.

ii) Give the April message that an agent would send to this matchmaker if it wanted to find the identity of an agent that it could ask about each of the topics: `['agents,'KQML]` using the ontology: `'computing_dai`.

iii) Give the changes you would make to the program so that it can also accept KQML style subscribe messages which contain a `reply_with` label and a `content` which is an KQML style `ask_about` message. This embedded `ask_about` includes an `ontology`, O. However, the value of the `content` field of the embedded `ask_about` is *not* a list of topic names, as it is for the `recommend` messages. Instead, the content of the embedded `ask_about` is a function of type

```
{(symbol[])->logical}
```

When it subsequently receives an `advertise` message with an embedded `ask_about` that specifies the same ontology, O, given in the `subscribe` message, the matchmaker applies this function to the list of topics given as content of the embedded `ask_about.` If the result of this function application is **true**, a suitable message is sent to the agent that sent the `subscribe` message. Correct syntax for the change to the `MM` program is not important. Make up your own message format for messages sent out to subscribers by the matchmaker.

[*Hint*: For each subscription you should remember the sender, the reply_with label, the ontology and the test function given in the embedded `ask_about`.]

The three parts carry, respectively, 20%, 30%, 50% of the marks.

[End of Paper