UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1997

MEng Honours Degrees in Computing Part IV
MEng Honours Degree in Information Systems Engineering 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

PAPER 4.74 / I4.8

DISTRIBUTED ARTIFICIAL INTELLIGENCE

Monday, April 28th 1997, 10.00 - 12.00

Answer THREE questions

For admin. only: paper contains 4 questions

1a Briefly describe the key features of the contract net protocol for allocating tasks over a network of agents/problem solvers. What are its limitations, particularly with respect to time constrained contracts?

b Sketch how you might implement the role of a Contract Net Manager as an April process. Assume there is a skill server agent that can be consulted and that the Manager is able to determine what skills are relevant for each subtask for which the Manager wants to place a contract. Say what tuples of data the manager would maintain, what kinds of messages it would send and receive, and what auxilary agents/processes it would consult or fork. (You do *not* need to give the April program or to give type specifications of the data structures and messages. Brief descriptions will suffice.)

The two parts of this question carry equal marks.

2a Briefly explain the role of a *matchmaker* 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. It can handle (advertise ...ask_about...) and (recommend .... ask_about...) messages.

The program assumes that a KQML message of the form:

(performative_name :f1 v1 :f2 v2 .... :fk vk)

is represented as the April tuple:

(performative_name, [(f1,v1), (f2,v2),..., (fk,vk)])

```
kplist::=(symbol,any)[];  /*type definition*/
MM(){
(handle?advertiser,symbol?ontology,symbol[]?topics)[]?Ads:= [];

repeat {
  (advertise,kplist?adKeyPairs )::
        (content,(ask_about,kplist?aaKeyPairs)) in  adKeyPairs and
        (ontology,symbol?O) in aaKeyPairs and
        (content,symbol[]?Topics)in aaKeyPairs
                ->
                Ads :=  [(sender, O,Topics),..Ads]
  |
  (recommend,kplist?recKeyPairs ) ::
        (reply_with,symbol?Label) in recKeyPairs and
        (content,(ask_about,kplist?aaKeyPairs )) in recKeyPairs and
        (ontology,symbol?O) in aaKeyPairs and
        (content,symbol?QTopic) in aaKeyPairs
                ->
        if {(handle?A,  O, symbol[]?AdTopics) ::
                                QTopic in  AdTopics} in Ads
        then
                (reply,[(in_reply_to,Label), (content,[A])])   >> replyto
        else
                (reply,[(in_reply_to, Label),(content,[])])     >> replyto
 } until quit
};
```

i) Give the April message that an agent would 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 the topic: dps using the ontology: computing_dai.

iii) What changes would you 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, given as a symbol. However, the value of the content field of the embedded ask_about is *not* a symbol topic name, as it is for the recommend message it accepts. Instead, the content of the embedded ask_about is a function of type

{(symbol[])->logical}

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

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

3a AGENT0 implements a simple model of a communicating agent.

i) *Briefly* describe the mental components of an AGENT0 agent, indicating what role they have.

ii) *Briefly* describe the message formats that the agent can accept, the effect they have on the agent components, and the types of action that an agent can be requested to do.

iii)*Summarise* the execution cycle of the agent.

b i) Sketch how you might implement an AGENT0 style agent as one or more April processes.

ii) What simple generalisations of the AGENT0 model would be possible using April?

The two parts of this question carry equal marks.

Turn over ....

4 Consider the generalised condition-action rule

```
if      ask(Agent, self, Question) and
        friendly(Agent) and
        demo(kb, Question, Answer)
then
        tell(self, Agent, Answer)
```

where kb is the self agent's knowledge base, formulated as a set of sentences in some logical language, and

demo(kb, Question, Answer)

means that the self agent can derive the answer, Answer, to the question, Question, from the knowledge base, kb.

a Explain why an agent using this rule with a condition-action production rule interpreter might not be reactive.

b Without changing the production rule interpreter or changing the rule, how could the self agent's knowledge base be restricted to make the agent reactive?

c Without changing the production rule interpreter or restricting the knowledge base, but possibly at the expense of the agent being able to answer the question, how could the rule be changed to make the interpreter reactive?

d Without restricting the knowledge base or changing the rule, but possibily at the expense of the agent giving a timely answer to the question, how could the interpreter be changed to make the agent reactive?

e How does Agent0 obtain the effect of the generalised condition-action rule above? How does it deal with the problem of reactivity, while still obtaining the effect of using such a rule?

f How does the Rao-Georgeff BDI "practical" agent architecture obtain the effect of the generalised condition-action rule above? How does it deal with the problem of reactivity, while still obtaining the effect of using such a rule?

The six parts carry, respectively, 15%, 15%, 15%, 15%, 20% and 20% of the marks.

End of paper