LAB BLIA – Comunicare intre agenti in LISP

Pentru ca doi sau mai multi sa fie capabili sa comunice unul cu celalalt, ei trebuie sa vorbeasca acelasi limbaj. Desi in general fiecare creator al unui sistem multi-agent foloseste propriul limbaj de comunicare, doua limbaje s-au impus ca fiind cele mai folosite: KQML si FIPA-ACL. *Limbajul de comunicare* (pe scurt ACL – Agent Communication Language) folosit in acest laborator este o varianta (mult) simplificata a FIPA-ACL.

In cazul nostru, un mesaj schimbat intre 2 agenti va trebui sa contina 5 campuri: <sender> - ce agent a trimis mesajul, <receiver> - ce agent(i) sunt destinatarii mesajului, <speechAct> - actul de limbaj al mesajului, <content> - continutul mesajului si <protocol> - numele protocolului de comunicare folosit. Actele de limbaj sunt folosite pentru a exprima intentia agentului care trimite mesajul, care este sensul pe care acel agent il asociaza cu acest mesaj. Doi agenti care vorbesc acelasi limbaj sunt capabili sa 'descifreze' un mesaj si sa inteleaga acelasi lucru. Pentru simplitate, presupunem ca ambii agenti inteleg continutul mesajului, nu este nevoie de o ontologie pentru a ii ajuta sa inteleaga acelasi lucru din continutul mesajului.

Pentru un astfel de mesaj propunem urmatoarea reprezentare LISP:

(*message* <sender> <receiver> <speechAct> <content> <protocol>)

Un exemplu de mesaj poate fi:

(*message* agentA agentB propose (object1 100) simpleProtocol),

semnificand ca agentul A ii propune agentului B obiectul 1 la pretul de 100 folosind protocolul simpleProtocol prezentat mai jos.

Dar pentru ca doi agenti sa poata comunica (deci sa poarte o conversatie) nu este de ajuns doar ca ei sa inteleaga mesajele primite. Este important ca ei sa respecte anumite reguli intr-o conversatie, in special reguli ce definesc ce mesaje pot fi trimise la un moment dat. Aceste reguli formeaza ceea ce se numeste un *protocol de comunicare*. Agentii despre care discutam in acest laborator comunica folosind protocolul prezentat in Figura 1. Pentru reprezentarea protocolului am ales A-UML care este o reprezentare semi-formala derivata din diagramele de interactiune UML si care reprezinta propunerea FIPA pentru reprezentarea protocoalelor de interactiune intre agenti.

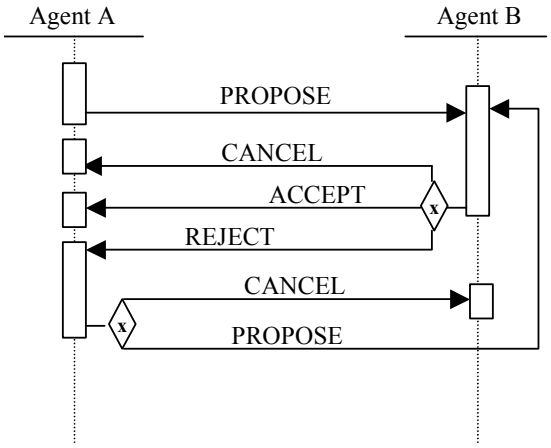

Figura 1. SimpleProtocol – un protocol de comunicare intre doi agenti

Semnificatia acestui protocol este urmatoarea: un agent (A) vrea sa cumpere un obiect de la un alt agent (B), asa ca ii face o oferta pentru acel obiect. B poate sa raspunda fie cu CANCEL si sa incheie conversatia fara succes (fara a vinde obiectul), cu ACCEPT si sa incheie conversatia cu succes (vinde obiectul), fie cu REJECT – semnificand ca a refuzat oferta lui A si ca asteapta alta oferta. In acest ultim caz, A poate sa faca o noua oferta (si se reia), fie sa incheie conversatia fara succes (cu mesajul CANCEL).

Acest laborator propune simularea in LISP a comunicarii intre 2 agenti urmand acest protocol. Este vorba de o *simulare,* intrucat in mod normal agentii sunt fire de executie separate, aflati probabil pe calculatoare diferite. In acel caz, functia de trimitere a unui mesaj (*send* de mai jos) ar trebui sa transmita

efectiv mesajul intre cele doua calculatoare (cei doi agenti), in timp ce in cazul nostru este un simplu apel de functie. Iata mai jos implementarea in LISP a comunicarii intre cei doi agenti:

```
    ;construieste un nou mesaj avand campurile specificate
    (defun newMessage (sender receiver speechAct content protocol)
      (list '*message* sender receiver speechAct content protocol))

    ;construieste un nou mesaj ca raspuns la unul vechi, modificand speechAct si content
    (defun replyTo (msg speechAct content)
      (list '*message* (third msg) (second msg) speechAct content (sixth msg
    ;aceasta este functia de initiere a conversatiei: A face o propunere lui B
    (defun start ()
      (print "start communication")
      (send (newMessage 'agentA 'agentB 'propose (makeProposal nil) 'simpleProtocol))
      "end of communication")

    ;functia care 'trimite' un mesaj. De fapt, apeleaza o functie corespunzatoare: ce agent a
primit
    ;mesajul
    (defun send (msg)
      (print msg)
      (if (eq 'agentA (third msg))
       (agentA_Receives msg)
       (agentB_Receives msg)))



    ;;;;;;;;;;;;;;;;;;;;;;;;
    ;;; AGENTUL A ;;;;;;;;;;
    ;;;;;;;;;;;;;;;;;;;;;;;;


    ;functie apelata cand agentul A a primit un mesaj. Conform protocolului, acest mesaj poate
fi:
    ;CANCEL - conversatia se incheie fara succes
    ;ACCEPT - agentul B a acceptat oferta, conversatia se incheie
    ;REJECT - agentul B a refuzat oferta, A poate sa raspunda cu un CANCEL sau cu PROPOSE (caz
in
    ;    care trebuie sa faca o noua oferta)

    (defun agentA_Receives (msg)
      (cond ((eq 'cancel (fourth msg))
             (print "Agent A failed to buy the object"))

       ((eq 'accept (fourth msg))
             (progn
                    (print "Agent A bought the object at the price:")
                    (print (second (fifth msg)))))

       ((eq 'reject (fourth msg))
             (if (eq 'cancel (agentA_ChooseAnswer msg '(propose cancel)))
                    (send (replyTo msg 'cancel nil))
                    (send (replyTo msg 'propose (makeProposal (fifth msg))))))))


    ;functie folosita de agentul A pentru a face o noua oferta, tinand cont de cea veche
(continutul
    ;mesajului anterior) -> de implementat!

    (defun makeProposal (content)
      (list 'object1 100))

    ;functie folosita de agentul A pentru a alege un raspuns din lista de raspunsuri posibile
in
    ;functie de mesajul primit -> de implementat!

    (defun agentA_ChooseAnswer (msg answers)
      (nth (random (length answers)) answers))
```

```
    ;;;;;;;;;;;;;;;;;;;;;;
    ;;; AGENTUL B ;;;;;;;;;;
    ;;;;;;;;;;;;;;;;;;;;;;


    ;functie apelata cand agentul B a primit un mesaj. Conform protocolului, acest mesaj poate
fi:
    ;CANCEL - conversatia se incheie fara succes
    ;PROPOSE - agentul A i-a facut o oferta. Agentul B trebuie sa aleaga un raspuns intre
ACCEPT,
    ;     REJECT sau CANCEL

    (defun agentB_Receives (msg)
      (cond ((eq 'cancel (fourth msg))
             (print "Agent B failed to sell the object"))

       ((eq 'propose (fourth msg))
             (send (replyTo msg (agentB_ChooseAnswer msg '(accept reject cancel))
                         (fifth msg))))))


    ;functie folosita de agentul B pentru a alege un raspuns din lista de raspunsuri posibile
in
    ;functie de mesajul primit -> de implementat!

    (defun agentB_ChooseAnswer (msg answers)
      (nth (random (length answers)) answers))
```

Se observa ca acest protocol este de fapt un foarte simplu protocol de negociere intre cei 2 agenti. Cu toate acestea, din implementare lipsesc functiile de decizie ale celor doi agenti: ce mesaj sa aleaga ca raspuns fiecare agent si ce oferta sa faca primul agent atunci cand trimite un mesaj PROPOSE. De fapt, pentru a putea executa codul, functiile sunt scrise mai sus, dar intr-o maniera foarte naiva (alegere aleatoare intre raspunsuri, intotdeauna aceeasi oferta).

TEMA

Se propune ca tema scrirerea acestor functii de decizie intr-o maniera mai ‘inteligenta’. Cu alte cuvinte, modificati codul pentru a obtine niste agenti rationali, care fac cea mai buna decizie pentru ei. La prezentarea temei va trebui sa justificati de ce considerati ca alegerea facuta este cea mai buna pentru agenti.

Pentru cei interesati, protocolul de comunicare poate fi modificat, puteti propune si implementa ce protocol doriti. Observati cum multe din cunostiintele de comunicare/negociere ale agentilor sunt codate direct. Discutie despre cat de dificila este realizarea unor agenti ‘generali’, care pot comunica folosind orice protocol sau limbaj de comunicare.