Programarea în reţea si prezentarea logicii fuzzy

1. Scopul lucrării

Scopul acestei lucrări este de insuşire a următoarelor mecanisme;

- trimiterea obiectelor prin socketuri
- comunicarea prin UDP
- logica fuzzy
- fuzzyficarea si defuzzyficarea unor marimi
- determinarea setului de reguli fuzzy logic FLRS
- modelul FLETPN.

2. Trimiterea obiectelor prin socket-uri

Mecanismul de serializare pune la dispoziţia programatorului o metodă prin care un obiect poate fi salvat şi restaurat atunci cand este nevoie. Tot prin acelaşi mecanism un obiect poate fi transmis la distanta catre o altă maşină utilizând socketurile.

Pentru a putea serializa un obiect acesta va trebui să implementeze interfaţa ***Serializable.***

Pentru scrierea şi citirea obiectelor serializate se utilizează fluxurile de intrare / ieşire : ***ObjectInputStream şi ObjectOutputStream.***

Listingul următor prezintă modul în care se poate serializa / deserializa un obiect.

```
import java.io.*;
import java.net.*;

public class SerialTest extends Thread{
      public void run(){
      try{
          ServerSocket ss = new ServerSocket(1977);
          Socket s = ss.accept();
          ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
          Pers p = (Pers) ois.readObject();
          System.out.println(p);
          s.close();
          ss.close();
          }catch(Exception e){e.printStackTrace();}
     }
  public static void main(String[] args) throws Exception{
       //trimite obiect prin socket
       (new SerialTest()).start();

       Socket s = new Socket(InetAddress.getByName("localhost"),1977);
       ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
       Pers p = new Pers("Alin",14);
```

```
        oos.writeObject(p);
        s.close();
    }
}

class Pers implements Serializable{
    String nume;
    int varsta;

    Pers(String n, int v){
      nume = n; varsta = v;
    }

    public String toString(){
      return "Persoana: "+nume+" varsta: "+varsta;
    }
}
```

Există situaţii în care atunci cand se salvează starea unui obiect prin serializare, nu se doreşte salvarea tuturor stărilor obiectului, respectiv nu se doreşte salvarea sau transmiterea anumitor parametri ai obiectului. Pentru a bloca serializarea unui atribut al unui obiect serializabil se utilizează cuvantul cheie ***transient***.

3. Server de timp (UDP)

Clienţii care comunică prin intermediul TCP, utilizând socket-uri, au un canal dedicat, iar transmisia datelor este sigură. Datele sunt recepţionate în ordinea în care acestea au fost trimise.

In contrast, când datele sunt transmise prin UDP, ajungerea acestora la destinaţie nu este garantată, de asemenea, ordinea de sosire la destinaţie a datagramelor poate să difere de ordinea în care acestea au fost transmise.

Avantajul lucrului cu datagrame este creşterea vitezei cu care pachetele (datagramele) ajung la destinaţie. Există cazuri în care viteza de transmisie a datelor este mai importantă decât garantarea 100% a ajungerii acestora la destinaţie. De exemplu în cazul transmiterii unui semnal audio în timp real, viteza de transmitere a acestuia este mai importantă decât garantarea ajungerii la destinaţie.

In java pentru implementarea protocolului UDP sunt utilizate clasele: **DatagramPacket** şi **DatagramSocket**. Spre deosebire de programarea TCP, în cazul UDP nu există conceptul de ServerSocket. Atât serverul cât şi clientul utilizează **DatagramSocket** pentru realizarea conexiunii. Pentru transmiterea şi recepţionarea datelor se utilizează clasa **DatagramPacket**.

In continuare este construit un server de timp care va trimite la cerere data curentă către clienţii care solicită acest lucru. De asemenea este construit şi clientul care accesează serviciile serverului de timp.

La nivelul serverului se creează un obiect **DatagramSocket**, care va primi ca parametru portul pe care serverul va începe ascultarea.

DatagramSocket socket = new DatagramSocket(1977);

In continuare se construieşte un obiect **DatagramPacket**, care va fi utilizat de către server pentru a recepţiona cererea de la client. O dată construit obiectul **DatagramPacket**, serverul va începe ascultarea portului 1977, prin invocarea metodei ***receive()***.

```
byte[ ] buf = new byte[256];
DatagramPacket packet = new DatagramPacket(buf,buf.length);
socket.receive(packet);
```

In momentul în care un client doreşte să apeleze la serviciile serverului, acesta va trimite un pachet către server. Serverul citeşte din cadrul pachetului portul şi adresa clientului, şi îi va trimite acestuia un pachet ce conţine data curentă.

```
InetAddress address = packet.getAddress();
int port = packet.getPort();
buf = ((new Date()).toString()).getBytes();
packet = new DatagramPacket(buf,buf.length,address,port);
socket.send(packet);
```

Un client, pentru a se conecta la server, trebuie să creeze un obiect **DatagramSocket**, şi să trimită un pachet către server. Spre deosebire de server, clientul nu este obligat să specifice nici un port în momentul creierii obiectului **DatagramSocket**, întrucât se atribuie automat un port liber respectivului obiect.

```
import java.io.*;
import java.net.*;
import java.util.*;

public class TimeServer extends Thread{
  boolean running=true;
  public TimeServer() { start();}
  public void run(){
    try{
       DatagramSocket socket = new DatagramSocket(1977);
       while(running)
  //asteapta client
            byte[] buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf,buf.length);
            socket.receive(packet);
    //citeste adresa si portul clientului
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
    //trimite un reply catre client
            buf = ((new Date()).toString()).getBytes();
            packet = new DatagramPacket(buf,buf.length,address,port);
            socket.send(packet);
            }
   }catch(Exception ex){ex.printStackTrace();}
 }
 public static void main(String[] args) {
   TimeServer timeServer1 = new TimeServer();
```

```
  }
}
```

```
 import java.io.*;
 import java.net.*;
 import java.util.*;

public class Client {

  public static void main(String[] args) {
  try{
      DatagramSocket socket = new DatagramSocket();
      byte[] buf = new byte[256];

      DatagramPacket packet = new DatagramPacket(buf,buf.length,
      InetAddress.getByName("localhost"),1977);
      socket.send(packet);
      packet = new DatagramPacket(buf,buf.length);
      socket.receive(packet);

      System.out.println(new String(packet.getData()));
  }catch(Exception ex){ex.printStackTrace();}
  }
}
```

4. Logica fuzzy, fuzzificarea si defuzificarea unor marimi

Consideratii teoretice

După cum se ştie, fiinţa umană lucrează după nişte reguli formulate prin educaţie şi instinct, folosind concepte şi măsuri nu întotdeauna exacte, ci mai degrabă mai ”flexibile” în cuantificare. Aceste aspecte au condus la logica fuzzy, teoretizată de Lotfi A. Zadeh în 1965. Logica fuzzy foloseşte variabile speciale, mărimi ”fuzzy”, formând astfel un spaţiu de lucru diferit de cel caracterizat prin valori numerice exacte.

Mărimile fuzzy reprezintă o clasă sau o mulţime de valori x_i, care sunt conţinute într-o anumită proporţie într-o mulţime, proporţia este dată de funcţia $\mu_i\ (x_i)$. Scopul este de a reduce complexitatea lumii exterioare la un număr finit de variabile cu care se poate opera în mod intuitiv. Modul de a opera cu aceste concepte sau variabile este sub forma unor reguli de implicaţie logică:

IF (*input **IS** {fuzzy_set1}*) **THEN** (*action **IS** {fuzzy_set2}*),

unde ”fuzzy_set1” este mulţimea valorilor pe care ”input” le poate lua, iar ”fuzzy_set2”, mulţimea valorilor posibile pentru ”action”.

Exemplu:

IF (x_1 is A_1 ^ x_2 is A_2 ^ … ^ x_m is A_m) **THEN** (y_1 is A_1 ^ y_2 is A_2 ^ … ^ y_n is A_n),

unde $x_i \in$ *fuzzy_set1* şi $y_i \in$ *fuzzy_set2*.

Mărimile fuzzy din mulţimile ”fuzzy_set” pot fi de tipul ”LOW”, ”MEDIUM”, ..., ”EXTREME”, ”HOT”, ”WARM”, ”CLOSE”, ”FAR”, etc.

In continuare vom folosi setul fuzzy FS= {NL, NM, ZR, PM, PL} cu valorile Negative Long, Negative Mediu, Zero, Pozitive Medium şi Pozitive Long pentru *fuzzy_set1* şi pentru *fuzzy_set2*.

Operaţia prin care o variabilă din lumea reală se transformă într-o variabilă fuzzy se numeşte ***fuzzyficare*** şi se realizează conform unei funcţii de apartenenţă, vezi Fig 2.1.

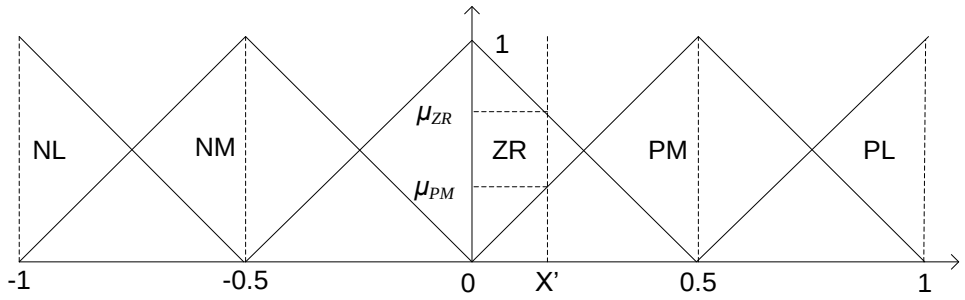

Figura 2.1 Funcția de fuzzyficare a mărimilor

Exemplu:

Pentru o variabila x‘, funcţia de apartenenţă la ZR: $\mu_{ZR}(x')$ = 0,65 iar pentru apartenenţa la PM: $\mu_{PM}(x')$ = 0.35. Observăm că: $\mu_{ZR}(x') + \mu_{PM}(x') = 1$.

Operaţia inversă, prin care o variabilă fuzzy este transformată într-o valoare numerică se numeste ***defuzzyficare*** şi se realizează similar operaţiei de fuzzyficare, calculând poziţia variabilei pe axa absciselor, în funcţie de gradele de apartenenţă la fiecare dintre mărimile fuzzy indicate. Aici sunt mai multe metode posibile, se utilizează de obicei metoda centrului de greutate:

$$x = \frac{\sum_i x_i \cdot \mu_i(x_i)}{\sum_i \mu_i(x_i)}$$

în care x_i sunt valorile nivelelor fuzzy din universul variabilei x, $\mu_i(x_i)$ sunt gradele de apartenenţă ale variabilei fuzzy la fiecare dintre nivelele fuzzy, iar x este rezultatul *crisp* al operaţiei de defuzzyficare.

O regulă poate fi îndeplinită într-o proporţie mai mare sau mai mică, după cum condiţia sa este îndeplinită. Astfel, mai multe reguli pot fi îndeplinite la un moment dat şi deci valoarea mărimii de comanda (”action”) ia mai multe valori fuzzy, în diverse proporţii.

In figura 2.2 este prezentată arhitectura unui sistem de control care foloseşte logica fuzzy. Mărimile de intrare, măsurate de senzori sunt fuzzyficate, se aplică algoritmul de control bazat pe logica fuzzy, apoi mărimile de ieşire sunt defuzzyficate şi aplicate sistemului, apoi sunt măsurate erorile şi ciclul se reia.

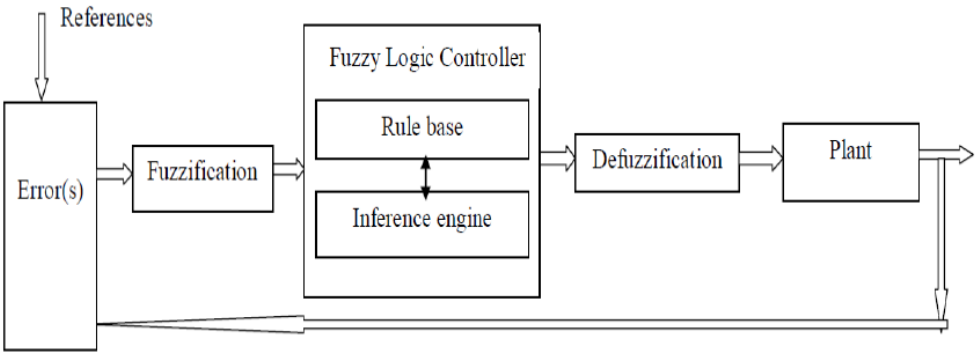

Figura 2.2 Arhitectura unui sistem de control bazat pe logica fuzzy

Fuzzy Logic Enhanced Time Petri Nets

Figura 2.3 reprezintă o componentă simplă cu modelul Fuzzy Logic Enhanced Time Petri Nets (FLETPN) cu două intrări şi două ieşiri. Domeniile x_1, x_2, x_3, $x_4 \in [-1,1]$, și considerăm factorii de amplificare w_1, $w_2 \in [-10, 10]$. Amplificarea se folosește astfel încât valorile rezultate x'_1 și x'_2 să fie într-un interval acceptabil.

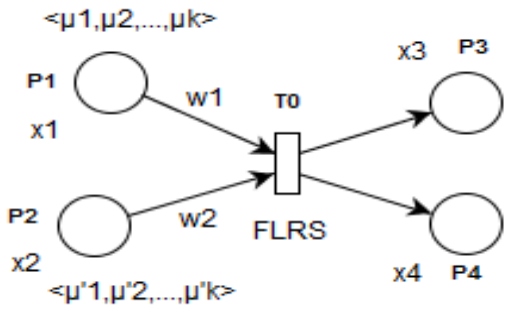

Figura 2.3 Exemplu FLETPN

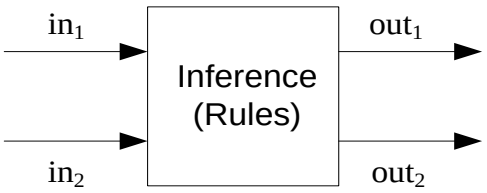

Figura 2.4 Modelul pentru FLETPN

Modelul este capabil să descrie reacția sincronă şi asincronă a unui sistem la modificări continue şi discrete. O locaţie poate conţine un jeton care reprezintă o reacţie sincronă sau asincronă respectiv o modificare continuă sau discretă.

Fiecare locaţie are asignată o variabilă x_i. Un jeton setat într-o locaţie p_i reprezintă gradele de apartenenţă a variabilei $\mu_i (x_i)$ la setul fuzzy FS. Pentru un jeton în exemplul nostru avem avem $n=5$: $<\mu_{NL}, \mu_{NM}, \mu_{ZR}\ \mu_{PM}, \mu_{PL}>$. Fiecare arc are un coeficient w_i.

O tranziție este executabilă dacă și numai dacă este cel puțin o regulă în tabela corespunzătoare care poate fi activată. Dacă mai multe reguli pot fi activate, la un moment dat, toate regulile sunt folosite, jetoanele rezultate fiind suma jetoanelor. Este utilizată o procedură de normalizare astfel încât locația de ieșire conține un jeton care îndeplinește condiția $\mu_{NL} + \mu_{NM} + \mu_{ZR} + \mu_{PM} + \mu_{PL} = 1$.

Execuția unei tranziții executabile t_i implică:

- extragerea jetoanelor din locațiile de intrare, notate ${}^{o}t_j$;
- defuzzyficarea tuturor variabilelor de intrare x_i;
- multiplicarea variabilelor cu coeficientul corespunzător w_i;
- fuzzyficarea variabilelor x_{ij};
- folosirea FLRS cu variabilele de intrare x_{ij} ;
- normalizarea operației care reduce consecințele la una singură și conduce la injectarea unui singur jeton în locația de ieșire;
- injectarea jetoanelor rezultate în locațiile de ieșire, notate t^{o}_j.

Un exemplu de soluții calculate pentru două intrări şi două ieşiri este cel din tabela 1:

Tabela 1: Exemplu de FLRS.

x_1/x_2	NL	NM	ZR	PM	PL
NL	NL,PL	PM,NL	ZR,ZR	NL,PL	ZR,PL
NM	PL,PM	NM,ZR	PL,NM	PL,NM	NM,PL
ZR	NL,PM	PL,NM	ZR,ZR	ZR,NM	PL,ZR
PM	ZR,PL	ZR,PM	NM,PM	PM,PM	NL,NL
PL	PM,ZR	PM,NM	ZR,ZR	NM,ZR	PL,NM

Fuzzyficarea şi defuzzyficarea mărimilor

Exemplu:

Se dau două mărimi de intrare x_1=0.3 şi x_2=0.7 ϵ [-1, 1] şi coeficienții w_1=2 şi w_2=3.

Se cere: să se calculeze ieșirile x_3 și x_4.

Înmulțim fiecare variabilă cu coeficientul corespunzător:

$x'_1=x_1*w_1=0.3*2=0.6$ şi

$x'_2=x_2*w_2=0.7*3=2.1=>x'_2=1$

Fuzzyficând cele două mărimi cu funcția din figura 2.1, avem pentru x'_1:

$\mu_{PL}=(0.6-0.5)*1/0.5=0.2$ *și* $\mu_{PM}=1*(0.5-0.1)/0.5=0.8$ deci locaţia p_1 va conţine valorile:

$\mu_{x'1}=<0, 0, 0, 0.8, 0.2>$

iar pentru x'_2 iar locaţia p_2:

$\mu_{x'2}=<0, 0, 0, 0, 1>$.

Considerând tabela FLRS din exemplul anterior, se activează regulile:

$r_1: <x_3,x_4> = <NL,NL>$ și $r_2:<x_3,x_4> = <PL,NM>$.

Defuzzyficând avem:

Puterea regulii s_i, în cazul în care avem conjuncție, este: $s_i = \mu_1 \cdot \mu_2$ sau $s_i = min\{\mu_1, \mu_2\}$.

In cazul nostru s_1=0.8*1=0.8 și s_2=0.2*1=0.2.

x_3=(NL*0.8+PL*0.2)/(0.8+0.2)=((-1)*0.8+(1)*0.2)/1=(-0.8+0.2)/1= -0.6.

x_4=(NL*0.8+NM*0.2)/(0.8+0.2)=((-1)*0.8+(-0.5)*0.2)/1=(-0.8-0.1)/1= -0.9.

Exercitii

1. Sa se construiasca o aplicatie folosind protocolul TCP/IP in care un client trimite un obiect de tip fir de executie serverului. Serverul il citeste si il lanseaza in executie.
2. Sa se construiasca o aplicatie in care clientul face schimb de mesaje cu serverul prin protocolul UDP. Mesajele sunt introduse de la tastatura iar raspunsurile primite sunt afisate la consola.
3. Sa se fuzzyfice mărimile x1= 0.4, x2= -0.1, w1=2, w2=3. Să se folosească tabela FLRS nr.1. Sa se calculeze iesirile x3 si x4.