Logica fuzzy, modelul FLETPN

1. Obiectivele laboratorului

- însușirea conceptelor:
  - fuzzyficarea şi defuzzyficarea unor mărimi,
  - determinarea setului de reguli fuzzy logic FLRS,
  - modelul FLETPN,
  - implementarea comparatorului în *Java* folosind utilitarul *FuzzyP*,
  - componente cu modele FLETPN
- dezvoltări şi teste.

2. Consideraţii teoretice

Avem setul fuzzy FS= {NL, NM, ZR, PM, PL}

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.

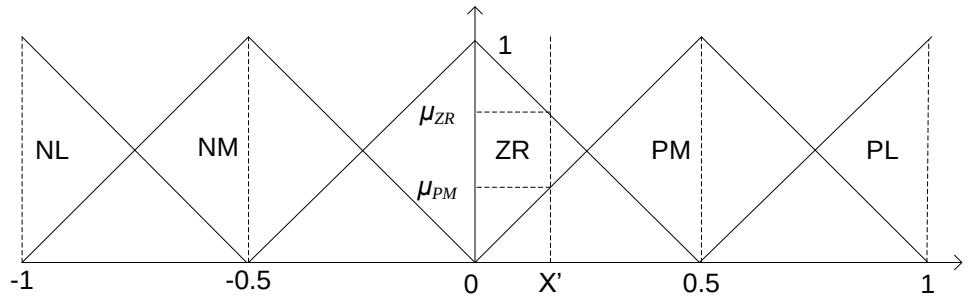


Figura 2.1 Funcția de fuzzyficare a mărimilor

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.

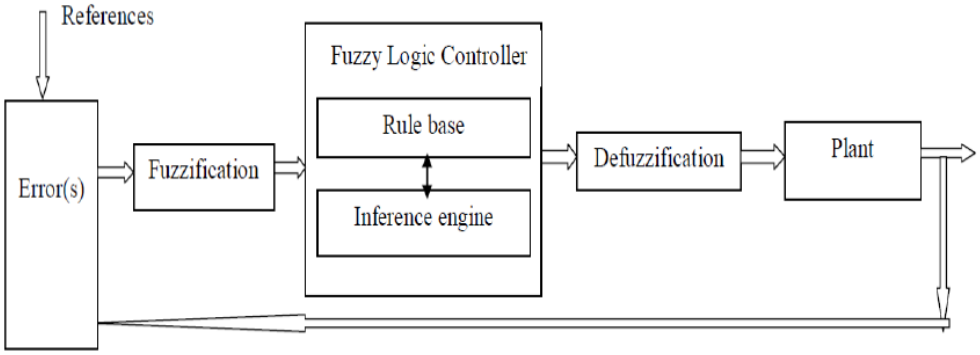


Figura 2.2 Arhitectura unui sistem de control bazat pe logica fuzzy

3. 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 ϵ *[-1,1]*, şi considerăm factorii de amplificare w_1, w_2 ϵ *[-10, 10]*. Amplificarea se foloseşte astfel încât valorile rezultate x'_1 şi x'_2 să fie într-un interval acceptabil.

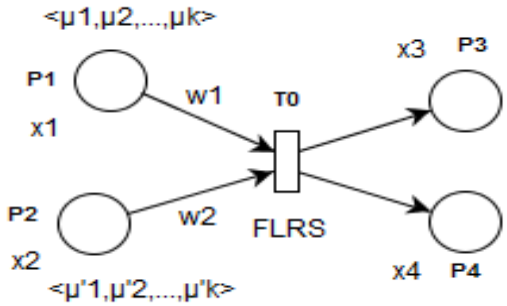


Figura 2.3 Exemplu FLETPN

Fiecare locaţie are asignată o variabilă x_i. Un jeton setat într-o locaţie p_i reprezintă gradele de apartenenţă a variabilei μ_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 și două ieșiri 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

4. Dezvoltări și teste

4.1. Aplicația 1

Specificații: se cere să se construiască un program folosind aplicatia All_Petri_Nets_Framework în care pentru mărimile de intrare x_1, x_2, w_1, w_2 este construit un inversor și care va calcula mărimile de ieșire x_3 și x_4.

In modelul matematic pentru inversor, considerăm:

$(x_1, x_2) \rightarrow (x_3, x_4)$ $x_3 = - x_1$; $x_4 = - x_2$; $w_1 = w_2 = 1$:

$x_3 = - w_1 \cdot x_1 \leftrightarrow w_1 \cdot x_1 \in [-1,1]$
$x_3 = 1; \leftrightarrow w_1 \cdot x_1 < -1;$
$x_3 = -1; \leftrightarrow w_1 \cdot x_1 > 1;$
$x_4 = - w_2 \cdot x_2 \leftrightarrow w_2 \cdot x_2 \in [-1,1]$
$x_4 = 1; \leftrightarrow w_2 \cdot x_2 < -1;$
$x_4 = -1; \leftrightarrow w_2 \cdot x_2 > 1;$

iar tabela FLRS:

$x_1 \setminus x_2$	**NL**	**NM**	**ZR**	**PM**	**PL**
NL	PL, PL	PL, PM	PL, ZR	PL, NM	PL, NL
NM	PM, PL	PM, PM	PM, ZR	PM, NM	PM, NL
ZR	ZR, PL	ZR, PM	ZR, ZR	ZR, NM	ZR, NL
PM	NM, PL	NM, PM	NM, ZR	NM, NM	NM, NL
PL	NL, PL	NL, PM	NL, ZR	NL, NM	NL, NL

In figura 4.1 avem reprezentata diagrama generala a claselor pentru aplicatia folosita pentru implementare. In figura 4.2 avem reprezentata diagrama claselor pentru cazul in care obiectele sunt de tip Fuzzy, reprezentate cu clasa DataFuzzy. Clasa Fuzzy are in campul Value date de tip float (jetonul nefuzzificat) si FuzzyVector este vectorul ce reprezinta fazzificarea marimii.

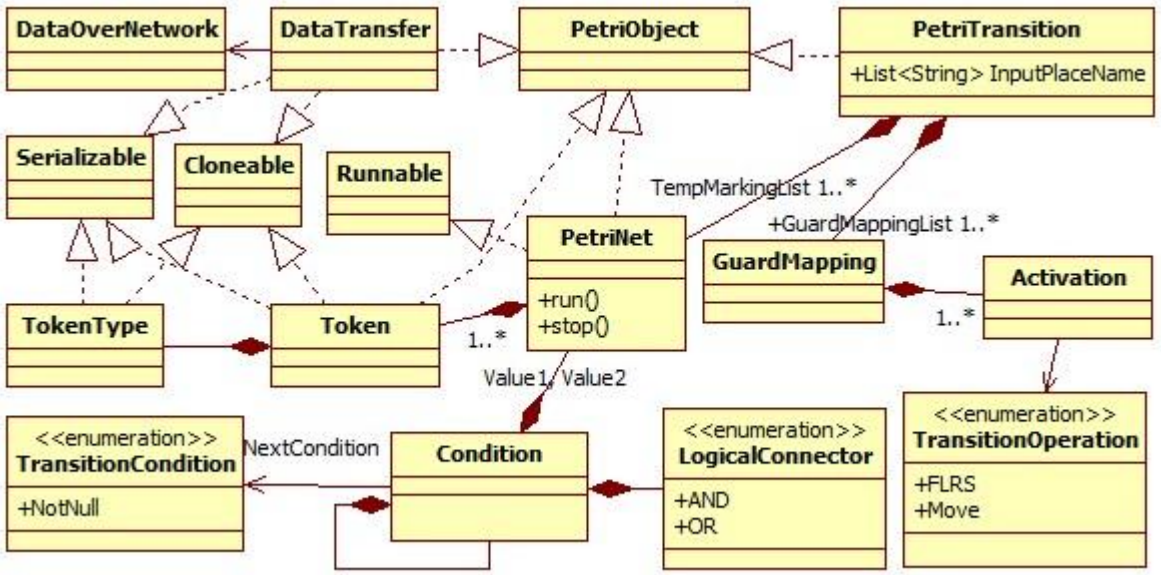


Figura 4.1 Diagrama claselor pentru aplicatia All Petri Framework – FLETPN

Setul FLRS (Fuzzy Logic Rule Set) este FLRSPart care are doi vectori: FV (FuzzyValue) si FZ (FuzzyValue).

Fiecare FLETPN este un obiect ce contine o lista cu locatii si tranzitii. Tranzitiile obligatoriu trebuie sa contina un obiect GuardMapping cu conditiile de garda care sunt o instanta a clasei Condition care este legata de urmatoarea conditie cu LogicalConnector (AND, OR).

Tranzitia este activata cand conditia tranzitiei este „true”. Cand tranzitia este activata se creaza o instanta a clasei Activation care contine operatiile ce trebuie executate aflate in TransitionOperations si indica unde pleaca jetoanele si ce informatie vor purta, calculand locatiile de iesire pe baza tabelei FLRS atasata tranzitiei sau doar o operatie simpla de tipul le Move. Cand iesirea tranzitiei este un canal de iesire, atunci tipul de data va fi DataTrasnfer.

Pentru implementare gasiti folderul All_Petri_Nets_Framework cu codul in Teams. si se va importa ca si o sursa existenta.

Eclipse: Files/Import Projects from File System or Archive/Directory/Select Folder

Intellij: File/Project from Existing Sources

https://github.com/dahliajanabi/All_Petri_Nets_Framework

Explicarea codului:

Se creaza un obiect *PetriNet* object si primeste numele Inversor. I se ataseaza un port, 1081 pentru conectarea la canalul de intrare.

```
PetriNet pn = new PetriNet();
pn.PetriNetName = "Inversor";
pn.NetworkPort = 1081;
```

Se creaza tabela FLRS respectand tabelul creat pentru inversor. Pe randuri avem valorile pentru x1 iar pe coloana valorile pentru x2. Celula [0,0] este [x1 NL, x2 NL], si are valloarea pentru x3 (PL) si valoarea pentru (PL).

```
FLRS flrs2x2 = new FLRS(new FV(FZ.PL, FZ.PL), new FV(FZ.PL, FZ.PM), new FV(FZ.PL, FZ.ZR), new FV(FZ.PL, FZ.NM),new FV(FZ.PL, FZ.NL),
                        new FV(FZ.PM, FZ.PL), new FV(FZ.PM, FZ.PM), new FV(FZ.PM, FZ.ZR), new FV(FZ.PM, FZ.NM),new FV(FZ.PM, FZ.NL),
                        new FV(FZ.ZR, FZ.PL), new FV(FZ.ZR, FZ.PM), new FV(FZ.ZR, FZ.ZR), new FV(FZ.ZR, FZ.NM),new FV(FZ.ZR, FZ.NL),
                        new FV(FZ.NM, FZ.PL), new FV(FZ.NM, FZ.PM), new FV(FZ.NM, FZ.ZR), new FV(FZ.NM, FZ.NM),new FV(FZ.NM, FZ.NL),
                        new FV(FZ.NL, FZ.PL), new FV(FZ.NL, FZ.PM), new FV(FZ.NL, FZ.ZR), new FV(FZ.NL, FZ.NM),new FV(FZ.NL, FZ.NL));
```

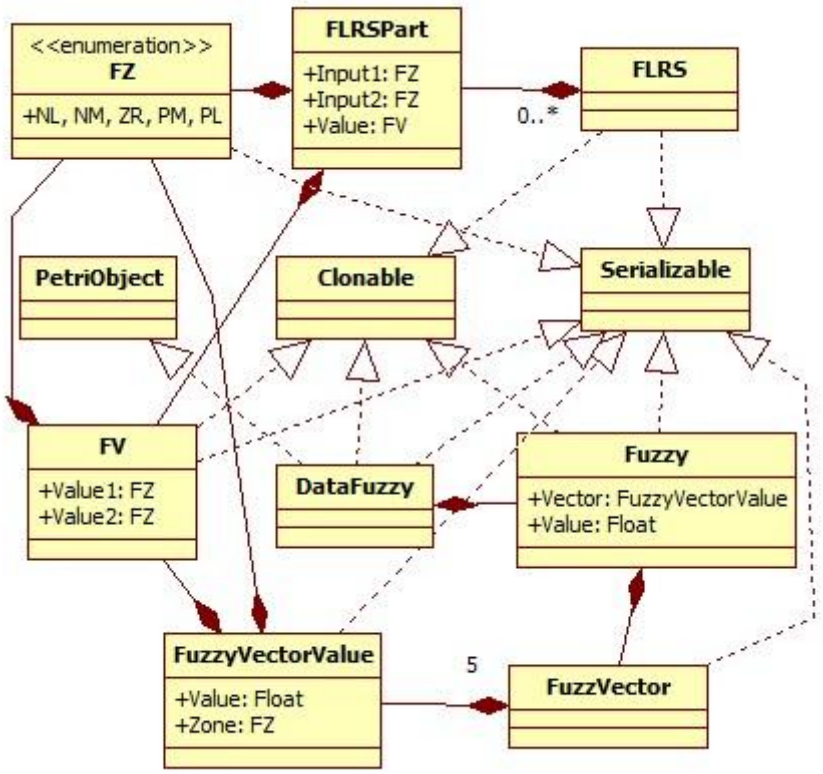


Figura 4.2 Tipurile de ietoane pentru FLETPN

Locatiile sunt create folosind obiecte DataFuzzy. Valoarea initiala este setata folosind metoda *setValue*(*value*) si apoi fiecare locatie este adaugata listei *PlaceList* din obiectul *PetriNet*.

```
DataFuzzy p1 = new DataFuzzy();
p1.SetName("P1");
p1.SetValue(new Fuzzy(0.1F));
pn.PlaceList.add(p1);
```

Tranzitiile sunt definite cu *PetriTransition*, numele este specificat cu *TransitionName.* Apoi se adauga locatiile de intrare.

```
PetriTransition t1 = new PetriTransition(pn);
            t1.TransitionName = "T1";
            t1.InputPlaceName.add("P1");
            t1.InputPlaceName.add("P2");
```

Pentru a defini conditiile de garda se creaza un obiect Condition. Acest obiect contine numele tranzitiei, locatia de intrare si operatia. Daca sunt mai multe obiecte acestea trebuie legate cuh *LogicConnector* (AND, OR).

Implementarea pentru conditia de garda a tranzitiei: $p1 \neq \phi$ AND $p2 \neq \phi$

Prima conditie este conectata cu a doua folosind AND, ca si la *LinkList*.

```
Condition T1Ct1 = new Condition(t1, "P1", TransitionCondition.NotNull);
Condition T1Ct2 = new Condition(t1, "P2", TransitionCondition.NotNull);
T1Ct1.SetNextCondition(LogicConnector.AND, T1Ct2);
```

De asemenea trebuie creat un obiect *GuardMapping* la care se adauga prima conditie si apoi se adauga la *Activations* (mappings).

```
GuardMapping grdT1 = new GuardMapping();
grdT1.condition = T1Ct1;
```

Obiectul *Activation* trebuie sa contina tranzitia, obiectul cu tabela FLRS si lista *Arraylist<PlaceNameWithWeight>* care contine numele, ponderea arcelor corespunzatoare locatiilor, tabela *FLRS*, si lista locatiilor de iesire *ArrayList <String>* care contine numele locatiilor de iesire. Obiectele *Activations* trebuie adaugate listei *GuardMapping*. La sfarsit, intreaga lista trebuie adaugata tranzitiei t1 pentru a stoca conditiile de garda.

```
ArrayList<PlaceNameWithWeight> input = new ArrayList<>();
                    input.add(new PlaceNameWithWeight("P1", 1F));
                    input.add(new PlaceNameWithWeight("P2", 2F));

                    ArrayList<String> twoOutput = new ArrayList<>();
                    twoOutput.add("P3");
                    twoOutput.add("P4");

                    grdT1.Activations.add(new Activation(t1, flrs2x2, input,
TransitionOperation.FLRS, twoOutput));

                    t1.GuardMappingList.add(grdT1);
```

Intarzierea tranzitiei trebuie de asemenea definita si tranzitia este adaugata listei de tranzitii *Transitions* corespunzatoare retelei Petri.

```
                    t1.Delay = 0;
                    pn.Transitions.add(t1);
```

cand se ruleaza reteaua Petri va aparea fereastra *GUI* corespunzatoare dupa cum se vede in figura 4.3.

```
pn.Delay = 3000;
PetriNetWindow frame = new PetriNetWindow(false);
frame.petriNet = pn;
frame.setVisible(true);
```

In randul de sus in figura 4.3 se vede jetonul fuzzy calculat cu ponderea corespunzatoare arcului care uneste locatia cu tranzitia.Restul vor avea ponderea 1.

In figura 4.4 se pot vedea intrarile Fuzzy, care se executa din pachetul GUIs si se pot trimite jetoane in timpul executiei prin canalul de intrare corespunzator canalului FLETPN, trebuie scris numele locatiei de intrare (cand aceasta este activa)., valoarea float si numrul portului pentru FLETPN.

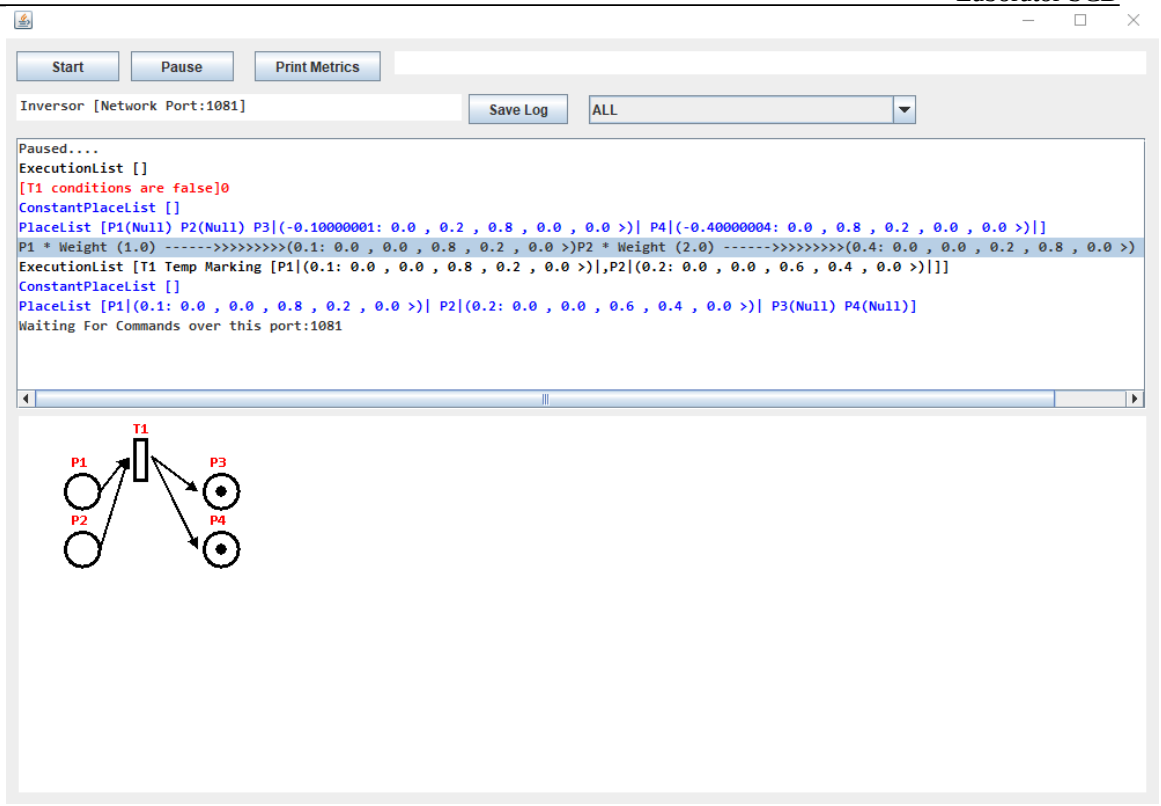
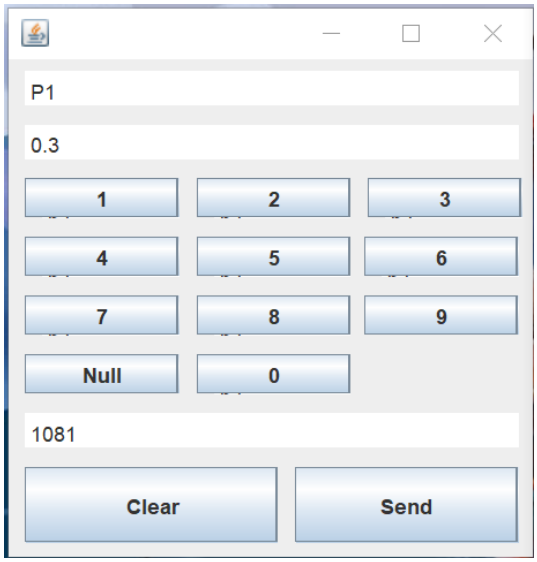


Figura 4.3 Interfata GUI pentru inversor



Figurea 4.4 Interfata pentru intrarile tFuzzy

Code listing 1: Inversor

```
package DCS_FuzzyLab1_2;

import java.util.ArrayList;
import Components.Activation;
import Components.Condition;
import Components.GuardMapping;
import Components.PetriNet;
import Components.PetriNetWindow;
import Components.PetriTransition;
```

```
import DataObjects.DataFuzzy;
import DataOnly.FLRS;
import DataOnly.FV;
import DataOnly.Fuzzy;
import DataOnly.FuzzyVector;
import DataOnly.PlaceNameWithWeight;
import Enumerations.FZ;
import Enumerations.LogicConnector;
import Enumerations.TransitionCondition;
import Enumerations.TransitionOperation;

public class Inversor {
	public static void main (String[]args) {
	PetriNet pn = new PetriNet();
	pn.PetriNetName = "Main Petri";
	pn.NetworkPort = 1081;

	FLRS flrs2x2 = new FLRS(new FV(FZ.PL, FZ.PL), new FV(FZ.PL, FZ.PM), new FV(FZ.PL,
FZ.ZR), new FV(FZ.PL, FZ.NM),new FV(FZ.PL, FZ.NL),
                                                new FV(FZ.PM, FZ.PL), new FV(FZ.PM,
FZ.PM), new FV(FZ.PM, FZ.ZR), new FV(FZ.PM, FZ.NM),new FV(FZ.PM, FZ.NL),
                                                new FV(FZ.ZR, FZ.PL), new FV(FZ.ZR,
FZ.PM), new FV(FZ.ZR, FZ.ZR), new FV(FZ.ZR, FZ.NM),new FV(FZ.ZR, FZ.NL),
                                                new FV(FZ.NM, FZ.PL), new FV(FZ.NM,
FZ.PM), new FV(FZ.NM, FZ.ZR), new FV(FZ.NM, FZ.NM),new FV(FZ.NM, FZ.NL),
                                                new FV(FZ.NL, FZ.PL), new FV(FZ.NL,
FZ.PM), new FV(FZ.NL, FZ.ZR), new FV(FZ.NL, FZ.NM),new FV(FZ.NL, FZ.NL));

	flrs2x2.Print();

	DataFuzzy p1 = new DataFuzzy();
	p1.SetName("P1");
	p1.SetValue(new Fuzzy(0.1F));
	pn.PlaceList.add(p1);

	DataFuzzy p2 = new DataFuzzy();
	p2.SetName("P2");
	p2.SetValue(new Fuzzy(0.2F));
	pn.PlaceList.add(p2);

	DataFuzzy p3 = new DataFuzzy();
	p3.SetName("P3");
	pn.PlaceList.add(p3);

	DataFuzzy p4 = new DataFuzzy();
	p4.SetName("P4");
	pn.PlaceList.add(p4);

	// T1 ------------------------------------------------
                PetriTransition t1 = new PetriTransition(pn);
                t1.TransitionName = "T1";
                t1.InputPlaceName.add("P1");
                t1.InputPlaceName.add("P2");

                Condition T1Ct1 = new Condition(t1, "P1",
TransitionCondition.NotNull);
                Condition T1Ct2 = new Condition(t1, "P2",
TransitionCondition.NotNull);
                T1Ct1.SetNextCondition(LogicConnector.AND, T1Ct2);

                GuardMapping grdT1 = new GuardMapping();
                grdT1.condition = T1Ct1;

                ArrayList<PlaceNameWithWeight> input = new ArrayList<>();
                input.add(new PlaceNameWithWeight("P1", 1F));
                input.add(new PlaceNameWithWeight("P2", 2F));

                ArrayList<String> twoOutput = new ArrayList<>();
                twoOutput.add("P3");
```

```
                        twoOutput.add("P4");

                        grdT1.Activations.add(new Activation(t1, flrs2x2, input,
TransitionOperation.FLRS, twoOutput));

                        t1.GuardMappingList.add(grdT1);

                        t1.Delay = 0;
                        pn.Transitions.add(t1);

                        // ----------------------------------------

                        System.out.println("Exp1 started \n ----------------------------");
                        pn.Delay = 3000;
                        // pn.Start();

                        PetriNetWindow frame = new PetriNetWindow(false);
                        frame.petriNet = pn;
                        frame.setVisible(true);
        }
}
```

Exercitii:

1. Testati aplicatia pentru diferite valori x_1, x_2, w_1 si w_2 apoi afisati x_3 si x_4.
2. Pornind de la aceasta aplicatie, construiti o aplicatie in care iesirile x_3 si x_4. Sa afiseze suma respective diferenta valorilor de intrare.

Modelul mathematic pentru sumator/differentiator este:

$(x_1, x_2) \rightarrow (x_3, x_4)$ $x_3 = x_1 + x_2$; $x_4 = x_1 - x_2$; $w_1 = w_2 = 1$:

$x_3 = w_1 \cdot x_1 + w_2 \cdot x_2$; $\leftrightarrow (w_1 \cdot x_1 + w_2 \cdot x_2) \in [-1,1]$

$x_3 = -1$; $\leftrightarrow (w_1 \cdot x_1 + w_2 \cdot x_2) < -1$

$x_3 = 1$; $\leftrightarrow (w_1 \cdot x_1 + w_2 \cdot x_2) > 1$

$x_4 = w_1 \cdot x_1 - w_2 \cdot x_2$; $\leftrightarrow (w_1 \cdot x_1 - w_2 \cdot x_2) \in [-1,1]$

$x_4 = -1$; $\leftrightarrow (w_1 \cdot x_1 - w_2 \cdot x_2) < -1$

$x_4 = 1$; $\leftrightarrow (w_1 \cdot x_1 - w_2 \cdot x_2) > 1$

si tabela FLRS:

$x_1 \setminus x_2$	NL	NM	ZR	PM	PL
NL	NL, ZR	NL, NM	NL, NL	NM, NL	ZR, NL
NM	NL, PM	NL, ZR	NM, NM	ZR, NL	PM, NL
ZR	NL, PL	NM, PM	ZR, ZR	PM, NM	PL, NL
PM	NM, PL	ZR, PL	PM, PM	PL, ZR	PL, NM
PL	ZR, PL	PM, PL	PL, PL	PL, PM	PL, ZR

Testati pentru diferite valori x_1 , x_2, w_1 si w_2.