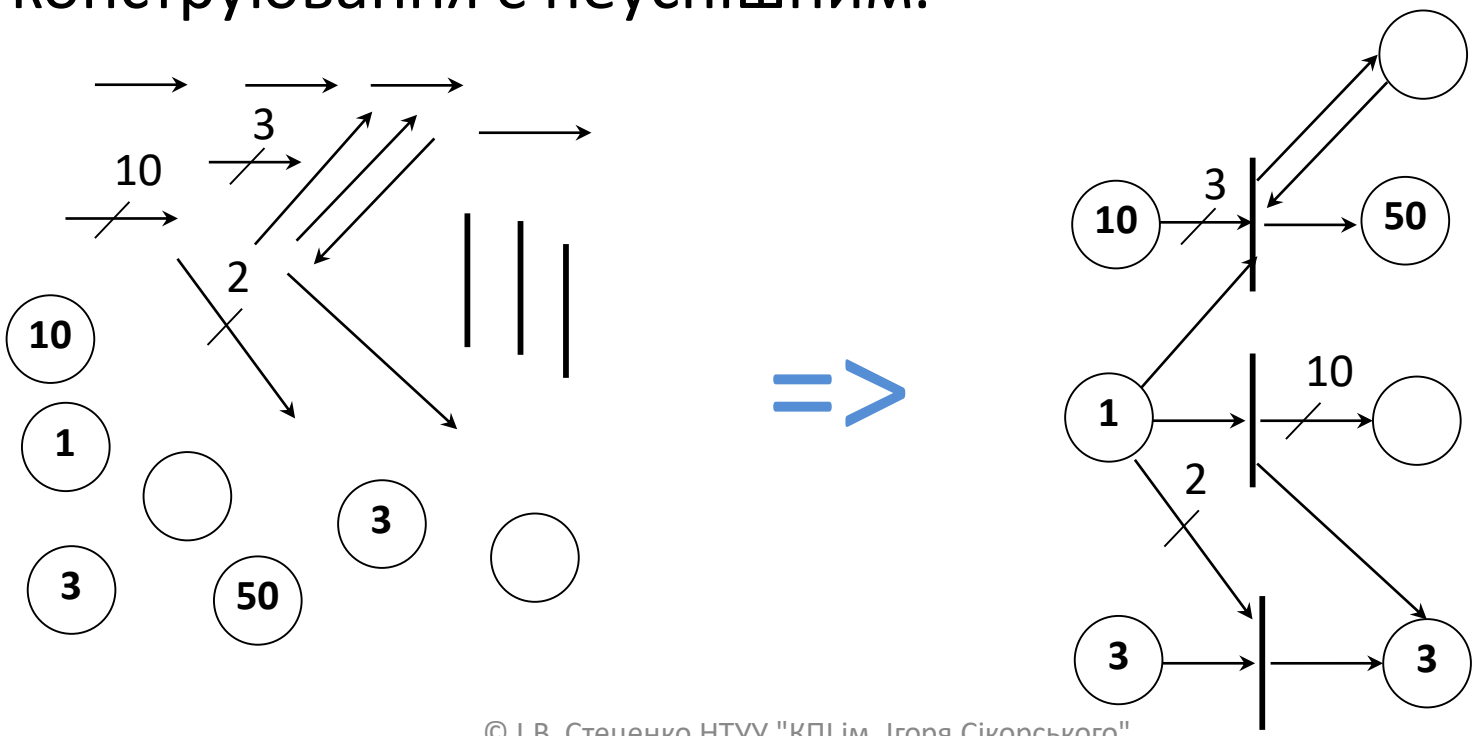
Алгоритм імітації стохастичної мережі Петрі

Допустима конструкція мережі Петрі

- ! Мережа Петрі повинна мати хоч один перехід
- ! Кожний перехід повинен мати хоч одну вхідну позицію і хоч одну вихідну позицію
- ! Вхідна дуга з’єднує позицію з переходом, вихідна дуга, навпаки, перехід з позицією
- ! Перехід з інформаційною вхідною дугою обов'язково повинен мати звичайну вхідну дугу
- ! Часова затримка в переході повинна приймати невід’ємні значення
- ! Мережа Петрі з часовими затримками повинна мати хоч один перехід з ненульовою часовою затримкою
- ✓ Початкове маркірування мережі Петрі повинно мати хоч одну позицію з ненульовим маркіруванням

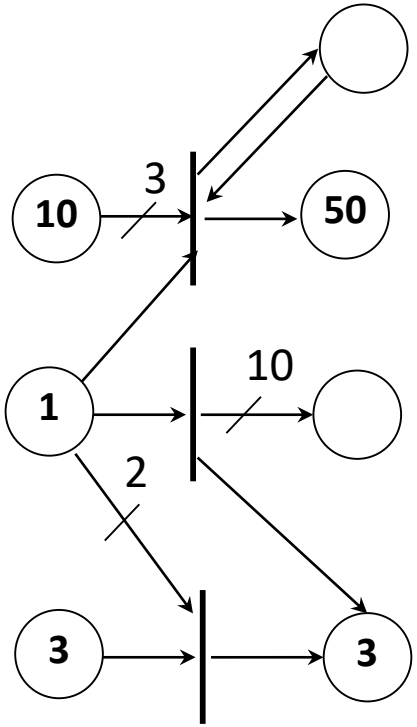
Конструювання мережі Петрі

При конструюванні мережі Петрі для кожного переходу встановлюється множина його вхідних позицій та множина його вихідних позицій. Якщо якась із цих множин виявилась порожньою, конструювання є неуспішним.



Умова запуску переходу

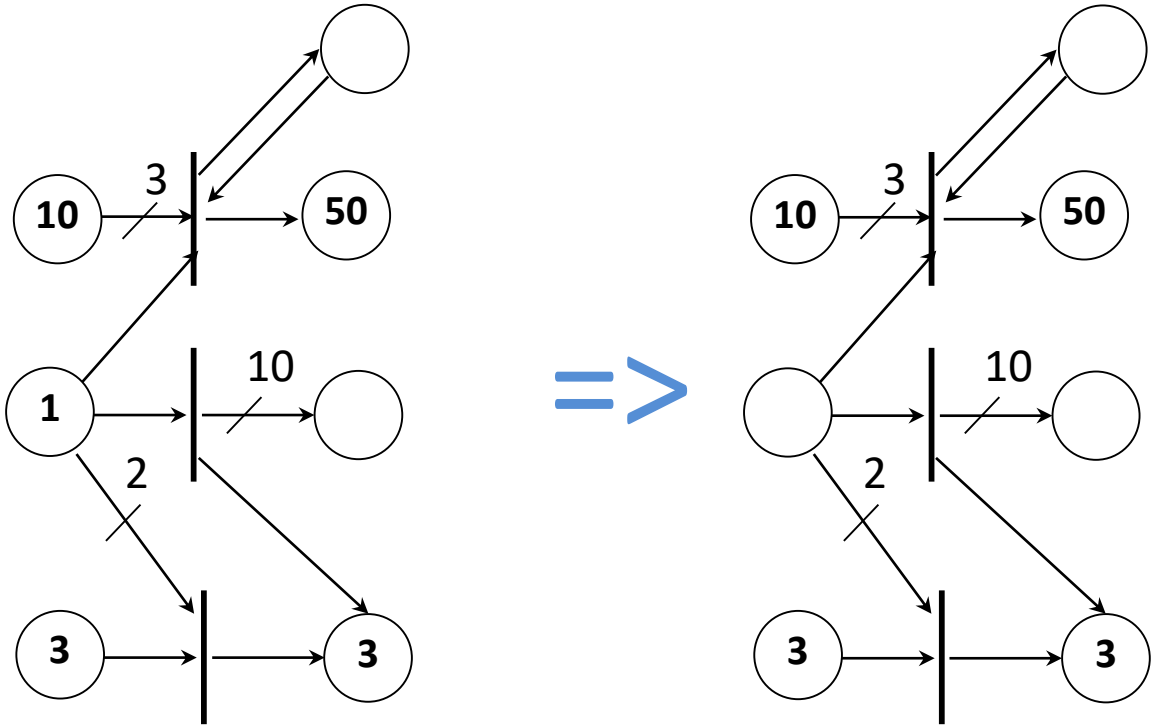
! Якщо у <u>всіх</u> вхідних позиціях переходу є маркери у кількості, рівній кратності дуги, то умова запуску переходу виконана.



!!! Як тільки умова запуску виконана, в цей же момент відбувається вхід маркерів в перехід

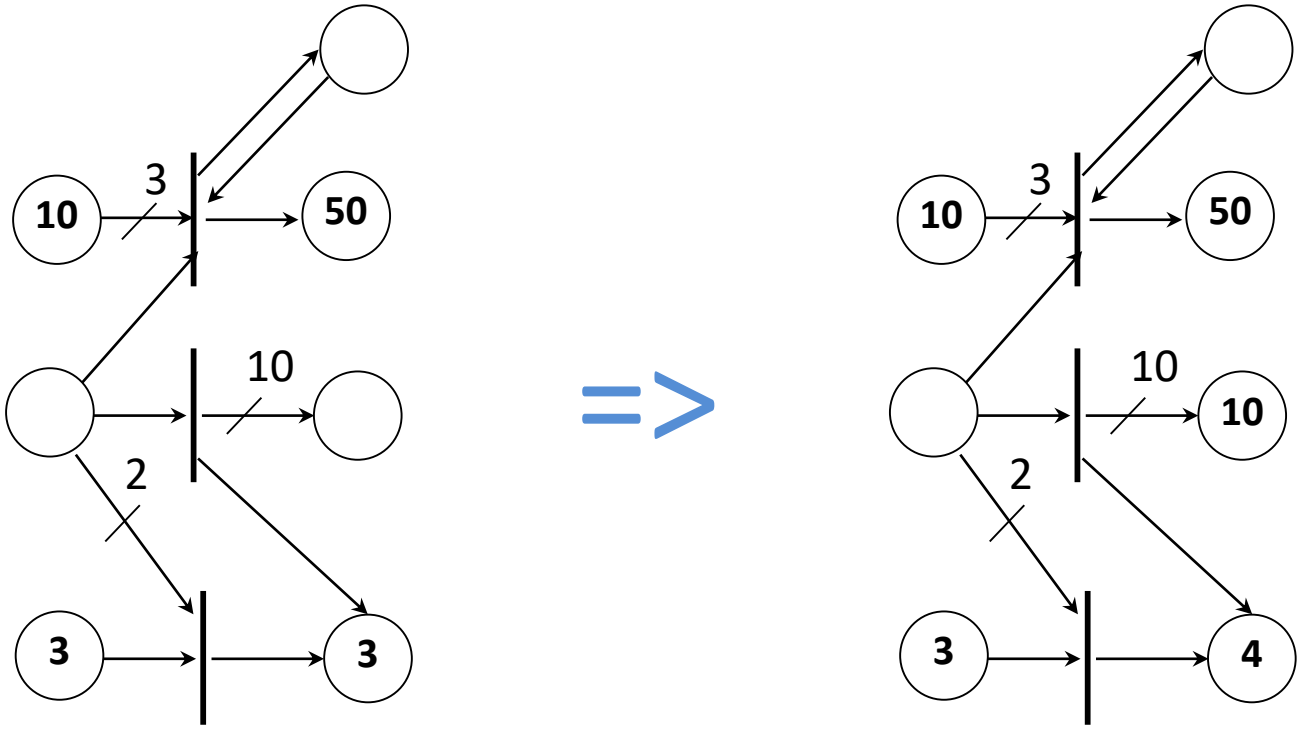
Вхід маркерів в перехід

При вході маркерів в перехід з кожної його вхідної позиції маркери видаляються в кількості, рівній кратності дуги, яка з'єднує цю позицію з цим переходом.



Вихід маркерів з переходу

При виході маркерів з переходу в кожну його вихідну позицію маркери видаляються в кількості, рівній кратності дуги, яка з'єднує цей перехід з цією позицією.



Алгоритм імітації класичної мережі Петрі з неявним пріоритетом переходів

Ввести елементи мережі Петрі, початковий стан маркірування, кількість кроків N.
Виконати конструювання мережі Петрі.
Для кожного переходу :
якщо умова запуску переходу виконана,
здійснити вхід маркерів в перехід,
запам'ятати стан переходу «активний».
Доки кількість кроків < N
для кожного переходу :
якщо перехід у стані «активний»,
здійснити вихід маркерів з переходу;
для кожного переходу :
якщо умова запуску переходу виконана,
здійснити вхід маркерів в перехід,
перерахувати статистику про функціонування моделі;
кількість кроків збільшити на 1 .
Вивести результати моделювання.
Кінець.

Алгоритм імітації класичної мережі Петрі з явним пріоритетом переходів

Для переходу задане значення пріоритету. За замовчуванням пріоритет = 0.

Ввести елементи мережі Петрі, початковий стан маркірування, кількість кроків N

Виконати конструювання мережі Петрі

Визначити список переходів з виконаною умовою запуску

Якщо список переходів з виконаною умовою запуску непорожній:

фільтрувати список переходів з виконаною умовою запуску так, щоб в ньому залишились тільки переходи з найбільшим пріоритетом;

якщо в списку переходів з виконаною умовою запуску після фільтрування залишилось більше ніж один, вибрати з них один з рівною ймовірністю, а інші відкинути;

виконати вхід маркерів в перехід, який залишився в списку переходів переходів з виконаною умовою запуску, запам'ятати стан «активний» для переходу

Інакше передчасне завершення імітації («стоп»)

Доки кількість кроків < N

Для всіх переходів: якщо перехід у стані «активний», то виконати вихід маркерів з переходу.

Визначити список переходів з виконаною умовою запуску

Якщо список переходів з виконаною умовою запуску непорожній:

фільтрувати список переходів з виконаною умовою запуску так, щоб в ньому залишились тільки переходи з найбільшим пріоритетом;

якщо в списку переходів з виконаною умовою запуску після фільтрування залишилось більше ніж один, вибрати з них один з рівною ймовірністю, а інші відкинути;

виконати вхід маркерів в перехід, який залишився в списку переходів переходів з виконаною умовою запуску.

зібрати статистику про функціонування моделі;

кількість кроків збільшити на 1.

Інакше передчасне завершення імітації («стоп»)

Кінець.

Алгоритм імітації класичної мережі Петрі з конфліктними переходами

Для переходу задане значення пріоритету та значення ймовірності запуску.

За замовчуванням пріоритет = 0, ймовірність запуску = 1.0

Ввести елементи мережі Петрі, початковий стан маркірування, кількість кроків N

Виконати конструювання мережі Петрі

Визначити список переходів з виконаною умовою запуску;

Якщо список переходів з виконаною умовою запуску непорожній:

фільтрувати список переходів з виконаною умовою запуску так, щоб в ньому залишились тільки переходи з найбільшим пріоритетом;

якщо в списку переходів з виконаною умовою запуску після фільтрації залишилось більше ніж один, визначити ймовірності запуску цих переходів на основі заданих значень та вибрати з них один з заданою ймовірністю, а інші відкинути;

виконати вхід маркерів в перехід, який залишився в списку переходів з виконаною умовою запуску, запам'ятати стан «активний» для переходу

Інакше передчасне завершення імітації («стоп»)

Доки кількість кроків < N

Для всіх переходів: якщо перехід у стані «активний», то виконати вихід маркерів з переходу.

Визначити список переходів з виконаною умовою запуску;

Якщо список переходів з виконаною умовою запуску непорожній:

фільтрувати список переходів з виконаною умовою запуску так, щоб в ньому залишились тільки переходи з найбільшим пріоритетом;

якщо в списку переходів з виконаною умовою запуску після фільтрації залишилось більше ніж один, визначити ймовірності запуску цих переходів на основі заданих значень та вибрати з них один з заданою ймовірністю, а інші відкинути;

виконати вхід маркерів в перехід, який залишився в списку переходів з виконаною умовою запуску;

зібрати статистику про функціонування моделі;

кількість кроків збільшити на 1;

інакше передчасне завершення імітації («стоп»)

Кінець.

Алгоритм імітації стохастичної мережі Петрі з конфліктними переходами

Вважається, що при вході маркерів в перехід він переходить в стан «зайнятий» і інші входи здійснюватись не можуть. Перехід, який в стані «зайнятий», не проходить перевірку на умову запуску переходу.

Алгоритм починається з входу маркерів в переходи мережі Петрі.

Доки t < Tmod

визначити момент найближчої події min;

зібрати статистику про функціонування моделі;

t = min;

якщо t < Tmod

виконати вихід маркерів з переходів мережі Петрі:

виконати вихід маркерів з переходу, що відповідає моменту найближчої події:

збільшити кількість маркерів в позиції на відповідне число

та запам'ятати момент виходу з переходу як рівний «нескінченності»

для кожного переходу:

якщо момент виходу маркерів з переходу співпадає з поточним часом,

виконати вихід маркерів з цього переходу:

збільшити кількість маркерів в позиції на відповідне число

та запам'ятати момент виходу з переходу як рівний «нескінченності»

виконати вхід маркерів в переходи мережі Петрі:

визначити список переходів з виконаною умовою запуску та вибрати з них один

(за заданими значеннями пріоритету та ймовірності запуску)

доки список переходів з виконаною умовою запуску непорожній

виконати вхід маркерів в перехід:

зменшити кількість маркерів у відповідних позиціях та запам'ятати нове значення

моменту виходу маркерів з переходу;

визначити список переходів з виконаною умовою запуску та вибрати з них один

(за заданими значеннями пріоритету та ймовірності запуску);

Кінець

Особливості розробки алгоритму імітації стохастичної мережі Петрі з конфліктними переходами, з багатоканальними переходами

Вважається, що кількість входів в перехід обмежується тільки кількістю маркерів у вхідних позиціях переходу.

В переході зберігається список значень моментів виходу з переходу. У списку зберігається не менше 1 значення. Це значення дорівнює «нескінченність», якщо найближчим часом не очікується вихід маркерів з переходу.

При виході маркерів з переходу виконується повторення виходу маркерів з переходу доки у списку моментів виходу з цього переходу є моменти часу, які дорівнюють t. Кожний вихід супроводжується видаленням відповідного моменту часу зі списку моментів виходу переходу. Останнє значення у списку не вилучається, а встановлюється в значення «нескінченність».

При вході маркерів в перехід виконується повторення входу маркерів в перехід доки список переходів з виконаною умовою запуску не порожній. При цьому в один і той самий перехід може здійснитись декілька входів маркерів, якщо для цього є достатня кількість маркерів у його вхідних позиціях.

Програмна реалізація конструювання мережі Петрі

```
public PetriNet(String s, PetriP[] pp, PetriT tt[], ArcIn[] in, ArcOut[] out) {
        name = s;
        numP = pp.length;
        numT = tt.length;
        numIn = in.length;
        numOut = out.length;
        listP = pp;
        listT = tt;
        listIn = in;
        listOut = out;
        for (PetriT transition : listT) {
          try {
               transition.createInP(listP, listIn);
               transition.createOutP(listP, listOut);
               if (transition.getInP().isEmpty()) {
                  throw new ExceptionInvalidNetStructure(
                          "Error: Transition " + transition.getName() +
                          " has empty list of input places ");
               }
               if (transition.getOutP().isEmpty()) {
                  throw new ExceptionInvalidNetStructure(
                          "Error: Transition " + transition.getName() +
                          " has empty list of output places"); }
            } catch (ExceptionInvalidNetStructure ex) {
                Logger.getLogger(
                  PetriNet.class.getName()).log(Level.SEVERE, null, ex);
            }
         }
    }
```

Приклад конструювання мережі Петрі

```
public static PetriNet createNetSMO(int numChannel, double timeMean, String name)
throws ExceptionInvalidTimeDelay, ExceptionInvalidNetStructure{
        ArrayList<PetriP> d_P = new ArrayList<PetriP>();
        ArrayList<PetriT> d_T = new ArrayList<PetriT>();
        ArrayList<ArcIn> d_In = new ArrayList<ArcIn>();
        ArrayList<ArcOut> d_Out = new ArrayList<ArcOut>();
        d_P.add(new PetriP("P1",0));
        d_P.add(new PetriP("P2",numChannel));
        d_P.add(new PetriP("P3",0));
        d_T.add(new PetriT("T1",timeMean,Double.MAX_VALUE));
        d_T.get(0).setDistribution("exp", d_T.get(0).getTimeServ());
        d_T.get(0).setParamDeviation(0.0);
        d_In.add(new ArcIn(d_P.get(0),d_T.get(0),1));
        d_In.add(new ArcIn(d_P.get(1),d_T.get(0),1));
        d_Out.add(new ArcOut(d_T.get(0),d_P.get(1),1));
        d_Out.add(new ArcOut(d_T.get(0),d_P.get(2),1));
        PetriNet d_Net = new PetriNet("SMOwithoutQueue"+name,d_P,d_T,d_In,d_Out);
        PetriP.initNext();
        PetriT.initNext();
        ArcIn.initNext();
        ArcOut.initNext();
        return d_Net;
}
```

Програмна реалізація умови запуску переходу (в класі PetriT)

```
public boolean condition(PetriP[] pp) {
//Нумерація позицій тут відносна!!!  inP.get(i) - номер позиції у списку позицій, який побудований при конструюванні мережі Петрі

        boolean a = true;
        boolean b = true;
        for (int i = 0; i < inP.size(); i++) {
            if (pp[inP.get(i)].getMark() < quantIn.get(i)) {
                a = false;
                break;
            }
        }
        for (int i = 0; i < inPwithInf.size(); i++) {
            if (pp[inPwithInf.get(i)].getMark() < quantInwithInf.get(i)) {
                b = false;
                break;
            }
        }
        return a == true && b == true;
    }
```

Програмна реалізація входу маркерів в перехід (в класі PetriT)

```
public void actIn(PetriP[] pp, double currentTime) {
        if (this.condition(pp) == true) {
            for (int i = 0; i < inP.size(); i++) {
                pp[inP.get(i)].decreaseMark(quantIn.get(i));
            }
            if (buffer == 0) {
                timeOut.set(0, currentTime + this.getTimeServ());
            } else {
                timeOut.add(currentTime + this.getTimeServ());
            }
            buffer++;
            if (observedMax < state) {
                observedMax = buffer;
            }
            this.minEvent();
        } else {
            //  System.out.println("Condition not true");
        }
    }
```

Програмна реалізація виходу маркерів з переходу (в класі PetriT)

```
public void actOut(PetriP[] pp) {
 //  num – номер каналу з найменшим значенням момену виходу маркерів
// buffer – кількість зайнятих каналів переходу
          if (buffer > 0) {
             for (int j = 0; j < outP.size(); j++) {
                 pp[outP.get(j)].increaseMark(quantOut.get(j));
             }
             if (num == 0 && (timeOut.size() == 1)) {
                 timeOut.set(0, Double.MAX_VALUE);
             } else {
                 timeOut.remove(num);
             }
             buffer--;
             if (observedMin > buffer) {
                 observedMin = buffer;
             }
          } else {
             // System.out.println("Buffer is null");
          }
}
```

Програмна реалізація входу маркерів в переходи мережі Петрі (в класі PetriSim)

```
public void input() {
        //формування списку активних переходів
       ArrayList<PetriT> activeT = this.findActiveT();
       if (activeT.isEmpty() && isBufferEmpty() == true) {
      //зупинка імітації за умови, що не має переходів, які запускаються

          timeMin = Double.MAX_VALUE;
       } else {
          while (activeT.size() > 0) { //запуск переходів доки можливо
              this.doConflikt(activeT).actIn(listP, getTimeCurr());
               activeT = this.findActiveT();
          }

          this.eventMin();//знайти найближчу подію та її час
       }
    }
```

Програмна реалізація виходу маркерів з переходів мережі Петрі (в класі PetriSim)

```
public void output() {
        if (getTimeCurr() <= getTimeMod()) {
            eventMin.actOut(listP);//здійснення події
            if (eventMin.getBuffer() > 0) {
                boolean u = true;
                while (u == true) {
                    eventMin.minEvent();
                    if (eventMin.getMinTime() == getTimeCurr()) {
                        eventMin.actOut(listP);
                    } else {
                        u = false;
                    }
                }
            }
// продовження на наступному слайді
```

Програмна реалізація виходу маркерів з переходів мережі Петрі (в класі PetriSim)

```
//Вихід з усіх переходів, що час виходу маркерів == поточний момент часу
     for (PetriT transition : listT) {
        if (transition.getState() > 0 &&
                            transition.getMinTime() == getTimeCurr()) {
                    transition.actOut(listP);
                    if (transition.getBuffer() > 0) {
                        boolean u = true;
                        while (u == true) {
                            transition.minEvent();
                            if (transition.getMinTime() == getTimeCurr()) {
                                transition.actOut(listP);
                            } else {
                                u = false;
                            }
                        }
                    }
                }
            }
        }
    }
```

Запуск моделі

```
ArrayList<PetriSim> list = new ArrayList<PetriSim>();
list.add(new PetriSim(NetLibrary.createNetSMO(2.0)));

PetriObjModel model = new PetriObjModel(list);
model.setIsProtocol(false);
double timeModeling = 1000000;

model.go(timeModeling);
```