

ЛЕКЦІЯ 4

Формалізація процесів функціонування дискретно- подійних систем мережею масового обслуговування

Інна Вячеславівна Стеценко
д.т.н., проф., професор кафедри ІПІ
НТУУ «КПІ ім. Ігоря Сікорського»

Термінологія

- У моделях масового обслуговування об'єкти, що підлягають обслуговуванню, називають замовленнями.
- Замовлення проходять одну або декілька операцій (фаз) обслуговування у певній послідовності.
- Послідовність, в якій об'єкти проходять фази обслуговування, називають маршрутом слідування замовлення. Маршрут слідування може бути заданий детерміновано або із заданими ймовірностями у місцях розгалуження.
- Система масового обслуговування (СМО) – це один чи декілька однакових пристрій обслуговування та черга перед ними (одна!). Одна СМО виконує одну операцію (фазу) обслуговування замовлення.
- Мережа масового обслуговування - це сукупність систем масового обслуговування із заданим маршрутом слідування.
- Розрізняють мережі МО замкнуті і розімкнуті. У замкнутих, замовлення проходять обслуговування за заданим маршрутом (детермінованим чи ймовірнісним) нескінченно, повертаючись знову і знову до операцій обслуговування, які вже проходили. У розімкнутих, замовлення надходять ззовні з заданою часовою затримкою та з заданим маршрутом слідування в одну із СМО мережі масового обслуговування.
- Одне з розширень мережі МО передбачає використання різних типів замовлень в одній моделі. Кожний тип замовлень обслуговується по заданих для нього параметрах.

Параметри та вхідні змінні мережі МО

- Кількість СМО.
- Для кожної СМО:
 - кількість пристройв,
 - обмеження на чергу (або його відсутність),
 - параметри часу обслуговування.
- Для замкнутої мережі МО - кількість замовлень, які циркулюють в мережі
- Для розімкнутої мережі МО – параметри інтервалу надходження нового замовлення
- Для маршруту слідування
 - ймовірності слідування з однієї СМО в іншу,
 - для розімкнутої мережі, ймовірності надходження замовлення ззовні до однієї зі СМО,
 - наявність блокування маршруту між СМО та умова блокування

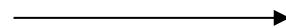
Вихідні характеристики

- Ймовірність відмови
- Середнє завантаження пристройв
- Середня довжина черги
- Середнє очікування в черзі

?? Якими змінними можна описати поточний стан мережі масового обслуговування

Елементи мережі масового обслуговування

Надходження запитів ззовні



$$t=\exp(2)$$

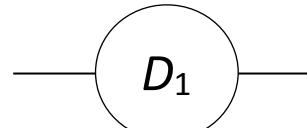
Надходження запитів на обслуговування з заданою часовою затримкою. Якщо надходження запитів ззовні немає, то цей елемент може бути відсутнім.

Параметри:

t - часова затримка в умовних одиницях часу, між послідовними надходженнями запитів на обслуговування, яка може бути вказана детермінованим значенням або випадковою величиною з заданим законом розподілу. Для мережі без зовнішнього надходження має бути вказана кількість запитів, що циркулюють в ній.

Елементи мережі масового обслуговування

Пристрій обслуговування



$$t = \text{unif}(5, 2)$$

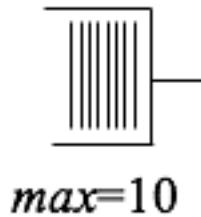
Пристрій, названий D1, обробляє запити по одному з часовою затримкою t.

Параметри:

t - часова затримка в умовних одиницях часу, необхідна для обробки одного запиту, яка може бути вказана детермінованим значенням або випадковою величиною з заданим законом розподілу.

Елементи мережі масового обслуговування

Черга



max=10

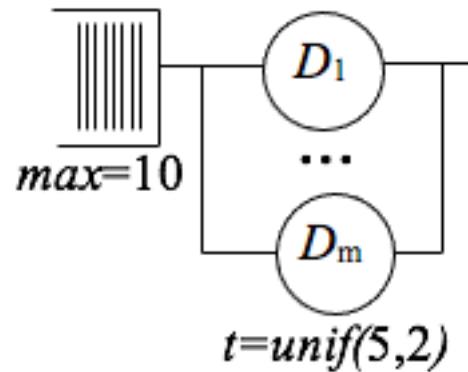
Накопичувач запитів, що очікують звільнення пристроя обслуговування.

Параметри:

max – обмеження на кількість місць в черзі, яке може бути задане цілим числом. За замовчуванням (тобто якщо параметр не вказаний), кількість місць в черзі не обмежена.

Елементи мережі масового обслуговування

Система масового обслуговування
(СМО)



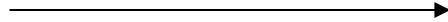
Один або кілька ідентичних пристройів з чергою перед ними. Черга є складовою частиною СМО, тому з'єднання між чергою та пристроями не дуга, а відрізок.

Параметри:

m - кількість пристройів у СМО

Елементи мережі масового обслуговування

Дуга



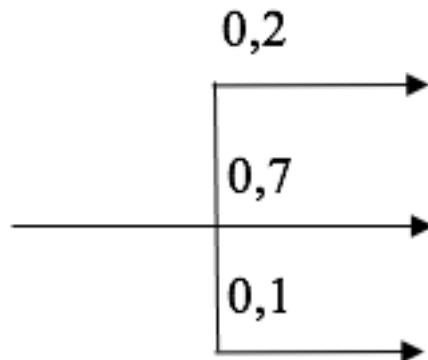
Маршрут слідування запиту до наступної СМО

Параметри:

За замовчуванням ймовірність слідування по маршруту 1.

Елементи мережі масового обслуговування

Розгалуження маршруту



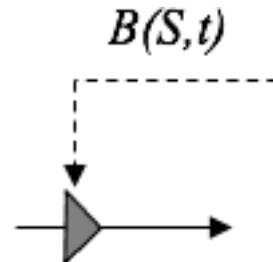
Маршрут слідування запиту до однієї з наступних СМО, вибір якої здійснюється за заданими ймовірностями.

Параметри:

Ймовірності вибору кожного маршруту слідування, які у сумі складають 1.

Елементи мережі масового обслуговування

Блокування маршруту



Встановлює блокування маршруту слідування у наступну СМО, якщо $B(S, t)=1$. Запит залишається у пристрої, з якого намагається вийти. Пристрій переходить у стан блокований (час обробки запиту завершився, але залишити пристрій немає можливості). Маршрут відкритий тільки за умови $B(S, t) = 0$.

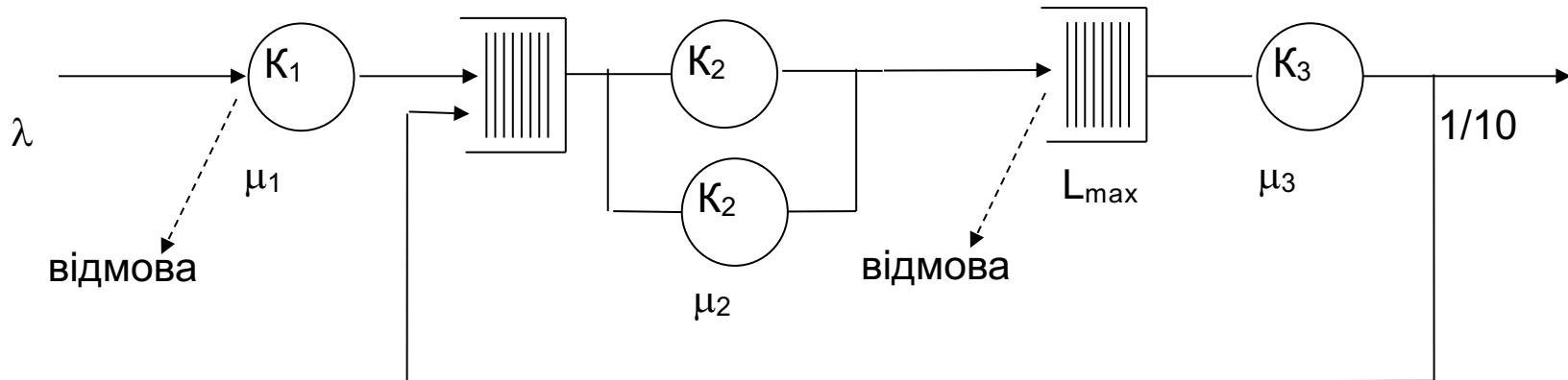
Параметри:

$B(S, t)$ – предикат, який в найбільш загальному випадку залежить від стану S моделі та/або поточному моменту часу t .

Наприклад, якщо умовою блокування маршруту є досягнення черги свого обмеження L_{max} , то предикат має вигляд:

$$B(S, t) = (L_j(t) = L_{max})$$

Приклад формалізованого представлення мережі масового обслуговування



?? Скільки тут СМО? Які вони – багатоканальні чи одноканальні?

?? Чи є тут розгалуження маршруту? Якщо так, то які ймовірності визначені для нього?

Послідовність дій, виконуваних для формалізації системи засобами мережі масового обслуговування

Для того, щоб представити систему мережею масового обслуговування потрібно:

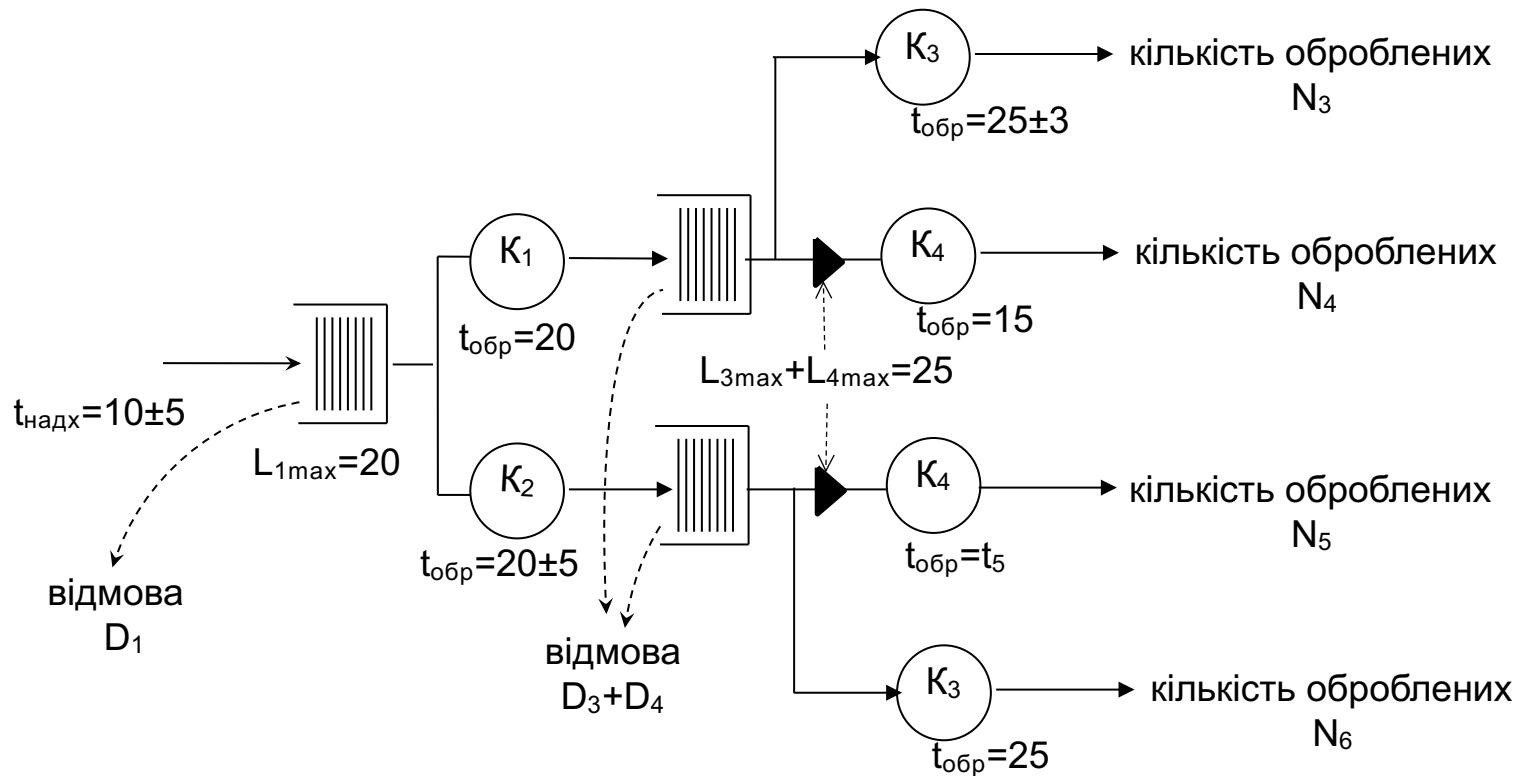
- з'ясувати, що в системі є об'єктом обслуговування;
- виділити елементи процесу обслуговування об'єктів і кожному елементу поставити у відповідність СМО;
- дляожної СМО визначити кількість пристроїв та наявність черги;
- з'єднати СМО у відповідності до процесу обслуговування;
- визначити маршрут проходження об'єкту обслуговування від однієї СМО до іншої;
- визначити умови надходження в кожну СМО (ймовірність вибору маршруту та інші);
- визначити наявність блокування маршруту та умови блокування;
- визначити числові значення параметрівожної СМО;
- визначити числові значення параметрів зовнішнього потоку на обслуговування;
- визначити стан мережі масового обслуговування на початку моделювання.

Приклад: Система передачі даних

Система передачі даних забезпечує передачу пакетів даних із пункту *A* в пункт *C* через транзитний пункт *B*. У пункті *A* пакети надходять через 10 ± 5 мс. Тут вони буферизуються в накопичувачі ємністю 20 пакетів і передаються по будь-якій із двох ліній *AB1*-за час 20 мс або *AB2*-за час 20 ± 5 мс. У пункті *B* вони знову буферизуються в накопичувачі ємністю 25 пакетів і далі передаються по лініях *BC1* (за 25 ± 3 мс) і *BC2* (за 25 мс). При цьому пакети з *AB1* надходять у *BC1*, а з *AB2* - у *BC2*. Щоб не було переповнення накопичувача, у пункті *B* вводиться граничне значення його ємності - 20 пакетів. При досягненні чергою граничного значення відбувається підключення резервної апаратури і час передачі знижується для ліній *BC1* і *BC2* до 15 мс.

Метою моделювання є визначення ймовірності підключення резервної апаратури, відсотку пакетів, які не передались через завантаження ліній зв'язку, та статистичних характеристик черг пакетів у пункті *A* та у пункті *B*.

Формалізована модель системи передачі даними



Статистичні х-ки черг пакетів у пункті А та у пункті В =

статистичні х-ки черг L_1 , L_3 , L_4 (середнє, відхилення, максимальне та мінімальне значення)

Відсоток пакетів, які не передались через завантаження ліній зв'язку =

$$(D_1 + D_3 + D_4) / (D_1 + D_3 + D_4 + N_3 + N_4 + N_5 + N_6)$$

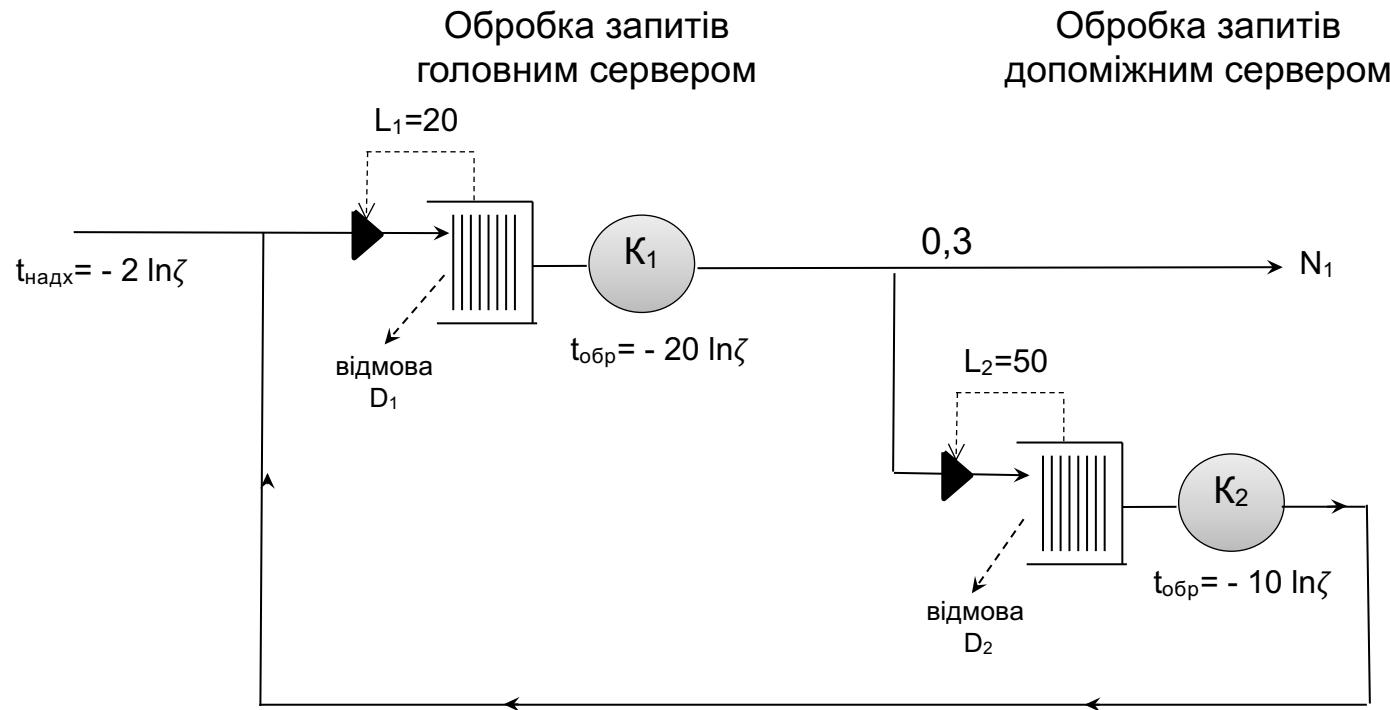
Ймовірності підключення резервної апаратури = $(N_4 + N_5) / (N_3 + N_4 + N_5 + N_6)$

Приклад. Інформаційна система

Інформаційна система надає інформацію про динаміку продажів торгової компанії, що має філії у різних країнах світу. Інформація агрегується та зберігається на двох серверах. З інформаційною системою працюють аналітики, які намагаються скласти прогнози та вчасно передбачити падіння попиту на певні види товарів. Запити від аналітиків надходять в середньому через 2 хвилини, з імовірністю 0,3 необхідна інформація знаходиться на головному сервері. У противному випадку потрібна додаткова інформація, щоб отримати яку головний сервер виконує запит на інший сервер. Час обробки запиту головним сервером складає в середньому 20 хвилин. Обробка запиту на іншому сервері триває в середньому 10 хвилин. Одночасно на головному сервері можуть опрацьовуватись 20 запитів, на допоміжному – 50 запитів. Після одержання інформації з допоміжного сервера головний сервер знову намагається обробити запит.

Метою моделювання є визначення середнього та максимального часу очікування відповіді на запит аналітиком.

Формалізована модель обробки запитів інформаційною системою



Середній час очікування відповіді на запит =
(загальний час очікування в черзі L_1 + загальний час очікування в черзі L_2 +
загальний час роботи пристрою K_1 + загальний час роботи пристрою K_2)/ N_1

Приклад. Модель обслуговування клієнтів у відділенні банку

Розглядається модель банку, у якому два касири сидять у приміщенні, а два обслуговують клієнтів, що під'їжджають на автомобілях. Частина клієнтів, що надходять у банк, намагається спочатку обслугуватись у автомобільних касирів. Час між надходженнями клієнтів цих клієнтів має експоненціальний закон розподілу з математичним сподіванням 0,75 хвилини.

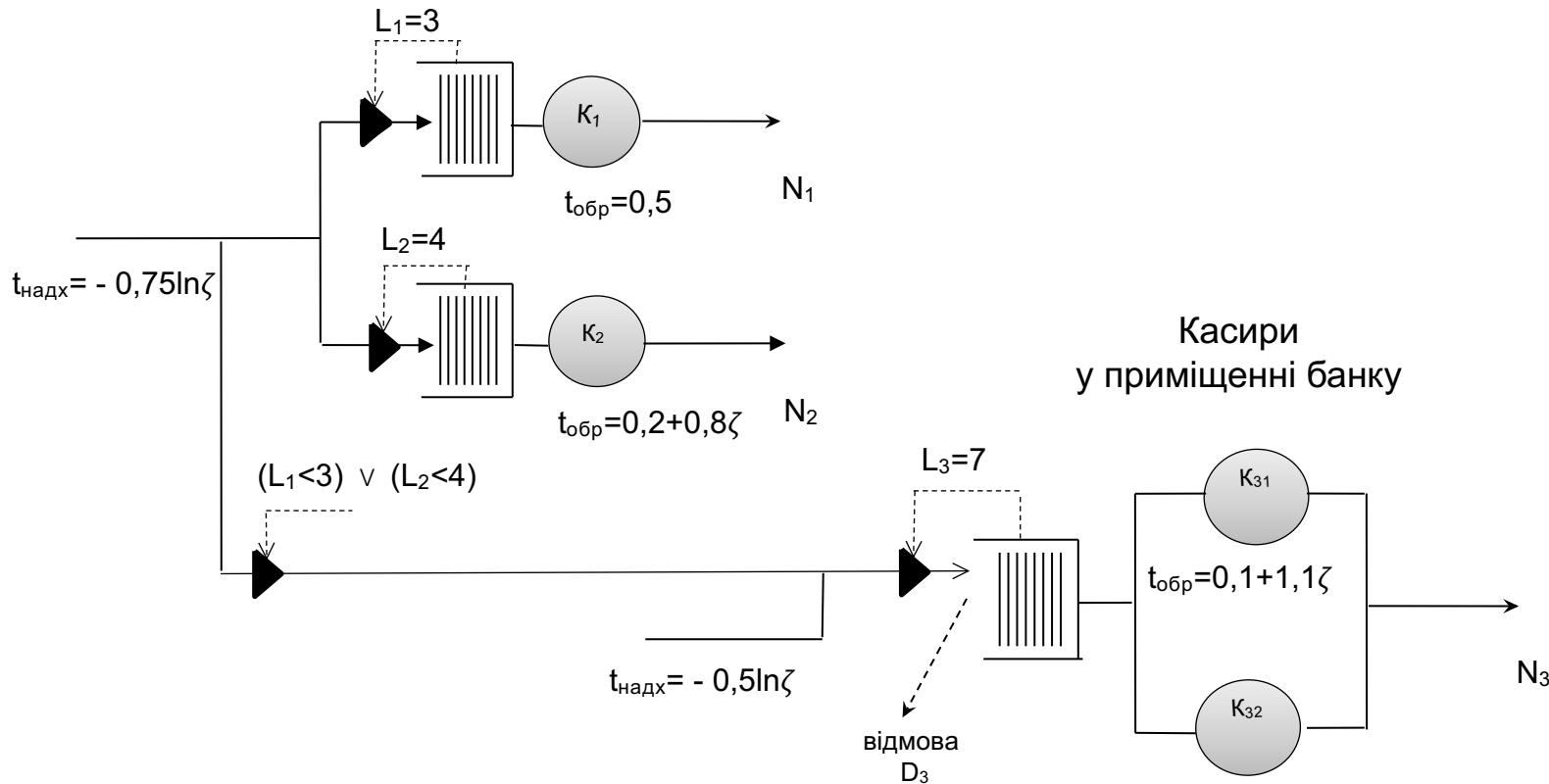
У черзі до першого касира можуть знаходитись три автомобілі, а в черзі до другого – чотири. Час обслуговування першим автомобільним касиром нормально розподілено з математичним сподіванням 0,5 хвилини і середньоквадратичним відхиленням 0,25 хвилини. Тривалість обслуговування другим касиром розподілена рівномірно на інтервалі $0,2 \div 1,0$ хвилини. Якщо клієнт, який приїхав на автомобілі, не може стати в чергу до автомобільних касирів через відсутність вільного місця, він залишає машину на стоянці і йде до касирів, що сидять у приміщенні банку.

Інша частина клієнтів йде зразу на обслуговування до касирів у приміщенні банку і стають в одну чергу з клієнтами, що прибули на автомобілях. Інтервал між їхніми прибуттями розподілений за експоненціальним законом з математичним сподіванням 0,5 хвилини. До обох касирів стоїть одна черга. У черзі не може стояти більше 7 клієнтів. Клієнти, що прийшли в банк, коли черга заповнена повністю, не обслуговуються і залишають банк. Час обслуговування в обох касирів у помешканні банку має рівномірний розподіл на інтервалі $0,1 \div 1,2$ хвилини.

Метою моделювання є визначення завантаження автомобільних касирів і касирів у приміщенні банку, середніх довжини черг, а також ймовірності того, що клієнт піде з банку не обслугованим.

Формалізована модель обслуговування клієнтів у відділенні банку

Автомобільні касири



Завантаження автомобільних касирів визначається завантаженням пристройів K_1 та K_2

Завантаження касирів у приміщенні банку визначається завантаженням пристройів K_{31} та K_{32}

Середні довжини черг визначаються середніми довжинами черг L_1 , L_2 , L_3

Ймовірність того, що клієнт залишить банк необслугованим $= D_3/(N_1+N_2+N_3)$

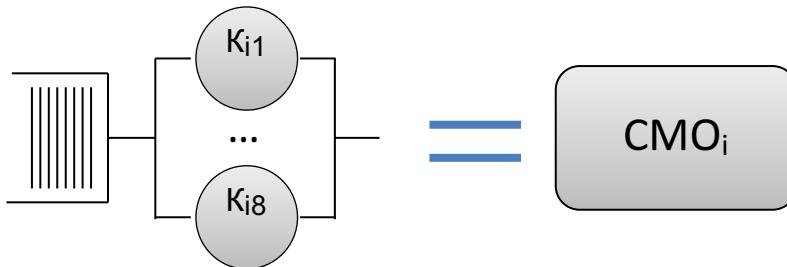
Приклад. Модель транспортної системи

Транспортний цех об'єднання обслуговує три філії A , B і C . Вантажівки везуть вироби з A в B і з B в C , повертаючись потім в A без вантажу. Навантаження займає в середньому 20 хвилин, переїзд з A в B триває в середньому 30 хвилин, навантаження в B - 20 хвилин, переїзд в C - 40 хвилин, і переїзд в A - 10 хвилин. Якщо до моменту навантаження в A і B відсутні вироби, вантажівки йдуть далі по маршруту.

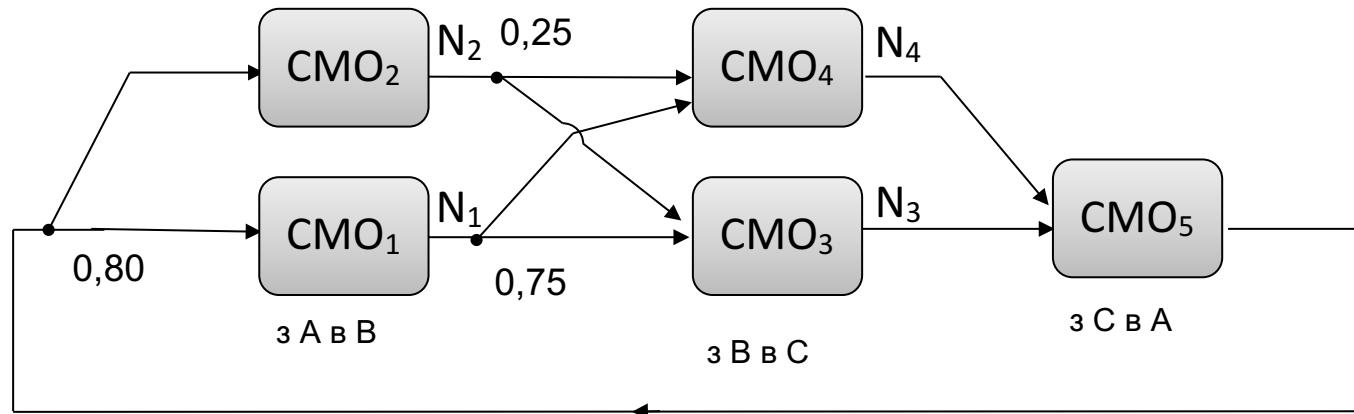
На лінії працює 8 вантажівок. Ймовірність того, що вироби на момент навантаження відсутні в A – 0,2, відсутні в B – 0,25.

Метою моделювання є визначення частоти порожніх перегонів вантажівок з A в B та з B в C .

Формалізована модель транспортної системи



8-канальна система масового обслуговування з необмеженою чергою



$$t_1 = - 20 \ln \zeta - 30 \ln \zeta$$

$$t_3 = - 20 \ln \zeta - 40 \ln \zeta$$

$$t_5 = - 10 \ln \zeta$$

$$\text{state}_{CMO5} = 8$$

$$t_2 = - 30 \ln \zeta$$

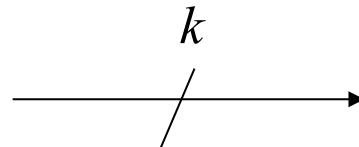
$$t_4 = - 40 \ln \zeta$$

Частота порожніх перегонів вантажівок з А в В = $N_2 / (N_2 + N_1)$

Частота порожніх перегонів вантажівок з В в С = $N_4 / (N_4 + N_3)$

Розширення формалізму мереж масового обслуговування: групування та розгрупування замовлень

Дуга з числовим параметром k



Групування елементів у вказаній кількості

Параметри:

Якщо на дузі вказаний параметр k , то k запитів перетворюються на дузі в 1 запит (групуються), що спрямовується далі по маршруту.

Якщо на дузі вказаний параметр $1/k$, то 1 запит перетворюється у k запитів (розгруповується), що спрямовуються далі по маршруту.

Обмеження на область застосування формалізму мереж масового обслуговування

- Процеси обслуговування, в яких є ресурси для обслуговування і об'єкти, які обслуговуються.
- Всі події в системі відбуваються виключно з об'єктами (а не з ресурсами).
- Об'єкти під час обслуговування **повністю** займають ресурс.
- Перетворення, які відбуваються з об'єктом при обслуговуванні, не враховуються
- Кількість об'єктів за результатами обслуговування не змінюється (об'єкти не діляться, не зникають, не знищуються).

Для задачі, що не вкладається у формалізм мережі масового обслуговування, слід розглянути можливість застосування іншого формалізму (мережа Петрі, Петрі-об'єктна модель, або дискретно-подійна система).

Для будь-якої дискретно-подійної системи можна побудувати алгоритм імітації з просуванням часу до найближчої події та змінюванням стану моделі за подійним підходом

Приклад задачі, що не вкладається у формалізм мережі масового обслуговування

Розглянемо систему керування запасами товарів одного типу деякого торгового підприємства.

Відомо, що попит на товари виникає через випадкові інтервали часу із середнім значенням $t_{\text{надх}}$. При наявності товару в запасі покупець, що надійшов, здійснює покупку, інакше підраховується невдоволений попит покупців на товар. Максимальний рівень запасу товарів, що зберігається, складає N товарів. Стратегія прийняття рішень про поповнення запасів складається у періодичному перегляді стану запасі з визначенням часом $t_{\text{контролю}}$. Якщо при перегляді стану запасів товару виявилось, що кількість товарів у запасі менша за m штук, то приймається рішення про поповнення запасу товарів і здійснюється замовлення на доставку товарів. Доставка товарів здійснюється протягом відомого часу $t_{\text{доставки}}$. Кількість товарів, що доставляються, доводить запас до максимального рівня запасів.

Метою моделювання є визначення такої стратегії прийняття рішень, що забезпечує найбільш ефективне функціонування торгового підприємства.

Формалізм дискретно-подійної системи: виділення подій

Виділимо події, які відбуваються в підсистемі обслуговування покупців:

- надійшов покупець;
- покупка одиниці товару;
- товар відсутній.

Виділимо події, які відбуваються в підсистемі прийняття рішень про поповнення запасу:

- контроль стану запасу;
- прийняття рішення про достатній стан запасу;
- прийняття рішення про недостатній стан запасу;
- поповнення запасу;
- періодичність контролю запасів.

Лекція 5

Універсальний алгоритм імітації мережі масового обслуговування

Універсальний алгоритм імітації мережі масового обслуговування

- Використовуємо способи:
 - просування часу за принципом до найближчої події,
 - орієнтований на події спосіб просування моделі в часі.
- Об'єктно-орієнтований підхід
- Розгалуження маршруту: за заданою ймовірністю або за пріоритетом
- Багатоканальність обслуговування
- Можливість вибору правила вибору замовлення з черги: FIFO або LIFO
- Блокування маршрутів за умовою, що враховує стан мережі або окремих її елементів
- Емпіричний закон розподілу

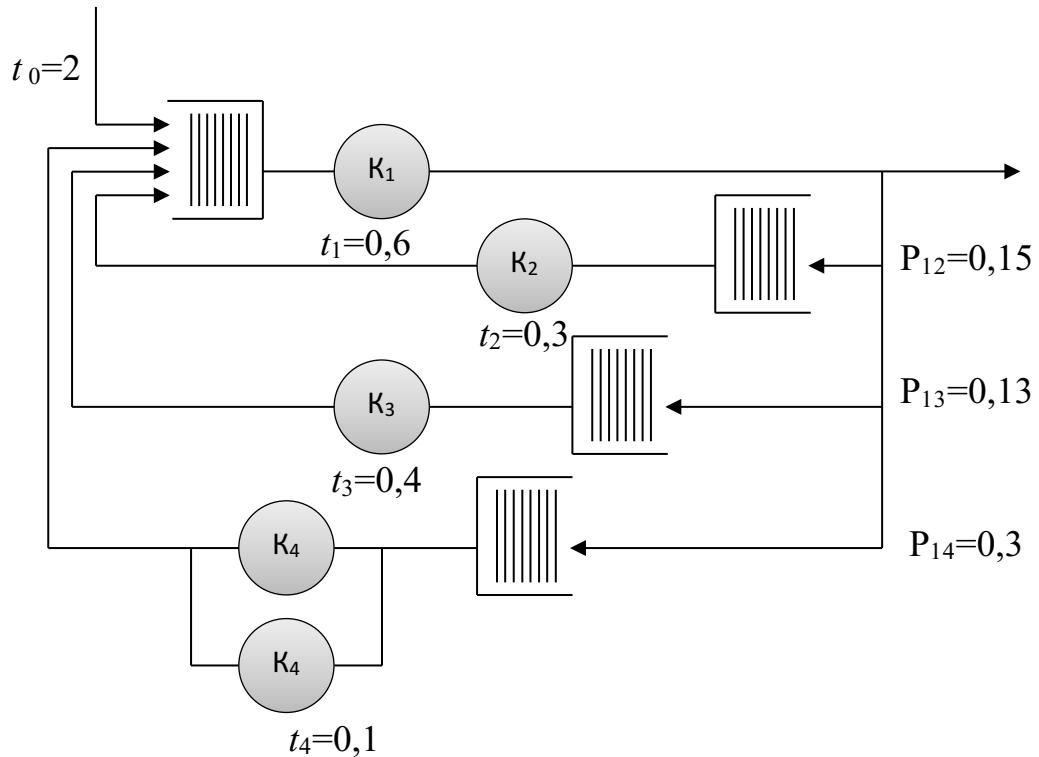
Оцінка точності алгоритму імітації мережі масового обслуговування

- Виконується порівнянням з результатами аналітичного розрахунку.
- Аналітичний розрахунок можливий за таких умов:
 - усі черги необмеженої довжини
 - усі часові затримки задані випадковими величинами з експоненціальним законом розподілу
 - вибір маршруту виключно за заданими ймовірностями
 - блокування маршрутів відсутні

Аналітичний розрахунок мережі масового обслуговування

- Теорія масового обслуговування ґрунтується на теорії марковських процесів.
- Система диференційних рівнянь будується за графом переходів з одного стану в інший.
- Рівняння стаціонарного стану описують функціонування системи в умовах стаціонарності (перехідний період завершився). Можуть бути безпосередньо отримані з графу переходів.
- Формули аналітичного розрахунку та приклади застосування див. Стеценко І.В. «Моделювання систем»

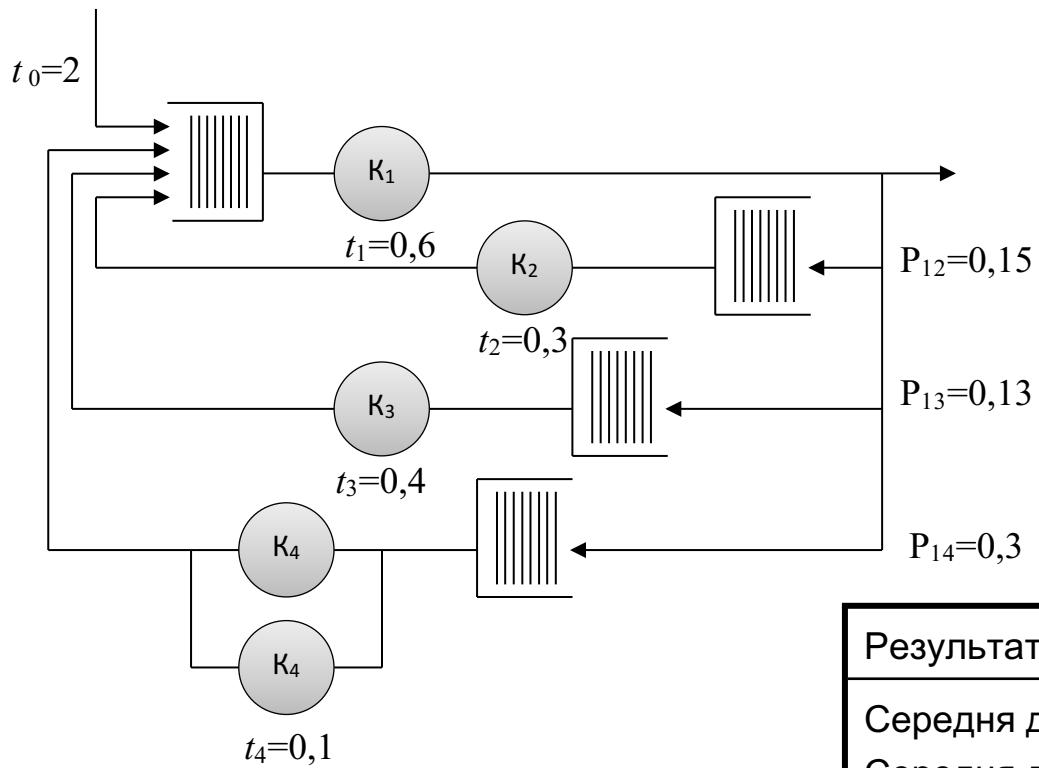
Мережа масового обслуговування з результатами аналітичного розрахунку для тестування алгоритму імітації



?? Спробуйте на основі власних міркувань передбачити:

- Яка з черг буде найбільшої довжини?
- Який ресурс буде найменш завантаженим?

Мережа масового обслуговування з результатами аналітичного розрахунку для тестування алгоритму імітації



Результати аналітичного моделювання

- Середня довжина черги СМО1 = 1,786
- Середня довжина черги СМО2 = 0,003
- Середня довжина черги СМО3 = 0,004
- Середня досжина черги СМО4 = 0,00001
- Середня зайнятість пристройв СМО1 = 0,714
- Середня зайнятість пристройв СМО2 = 0,054
- Середня зайнятість пристройв СМО3 = 0,062
- Середня зайнятість пристройв СМО4 = 0,036

Оцінка складності алгоритму імітації для мережі масового обслуговування

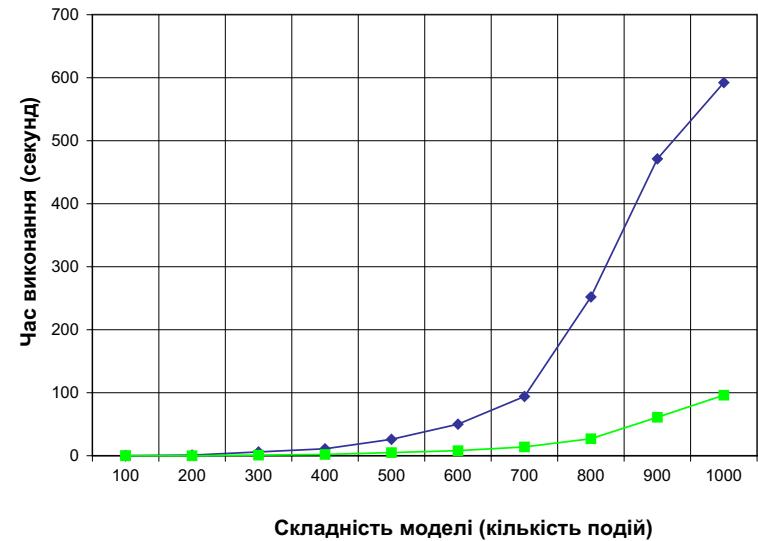
- Експериментальна оцінка складності виконується побудовою залежності часу виконання алгоритму в залежності від складності моделі
- Теоретична оцінка складності виконується підрахунком кількості елементарних операцій в алгоритмі в залежності від складності моделі:

$$O(v \cdot timeMod \cdot k)$$

Інтенсивність подій

Час моделювання

Середня (або максимальна) кількість елементарних операцій для обробки однієї події



Термінологія англійською

- Queueing theory – Теорія масового обслуговування
- Simulation of Queueing Systems – Задачі та програмні продукти для імітаційного моделювання масового обслуговування

Лекція 7

Формалізм мережі Петрі

Переваги формалізації мережею масового обслуговування:

Простота

Можливість аналітичного розрахунку (при певних обмеженнях)

Наочність представлення процесу функціонування системи

Недоліки формалізації мережею масового обслуговування:

Обмеженість застосування для процесів управління

Обмеженість зв'язків між елементами моделі

Недостатня гнучкість розробки моделі дискретно-подійної системи (через налаштованість виключно на процеси обслуговування)

Переваги формалізації мережею Петрі:

Гнучкість формалізації подій

Універсальність алгоритму імітації

Візуалізація процесу функціонування

Пристосованість до представлення паралельних процесів

Недоліки формалізації мережі Петрі:

Велика кількість елементів

Мережі Петрі у стандартах з інженерії програмного забезпечення

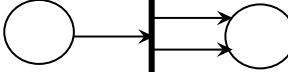
- ISO/IEC 15909-1:2004 SYSTEMS AND SOFTWARE ENGINEERING -- HIGH-LEVEL PETRI NETS -- PART 1: CONCEPTS, DEFINITIONS AND GRAPHICAL NOTATION
- ISO/IEC 15909-1:2004 defines a semi-graphical modelling language for the specification, design and analysis of discrete event systems, including software and in particular distributed and parallel systems where concurrency is an important characteristic. The technique, High-level Petri Nets, is mathematically defined and may thus be used to provide unambiguous specifications and descriptions of applications. The graphical nature of the technique allows information, or resource flow, and control flow to be visualised, providing a powerful aid to understanding system behaviour. It is also an executable technique, allowing specification prototypes to be developed to test ideas at the earliest and cheapest opportunity. Specifications written in the technique may be subjected to analysis methods to prove properties about the specifications, before implementation commences, thus saving on testing and maintenance time. The field of application encompasses a wide range of systems from technical systems such as manufacturing, business processes, computer software and hardware, telecommunication networks and signalling systems, defence systems, mechatronics, postal services and avionics to biological and sociotechnical systems.
- ISO/IEC 15909-1:2019 SYSTEMS AND SOFTWARE ENGINEERING -- HIGH-LEVEL PETRI NETS -- PART 1: CONCEPTS, DEFINITIONS AND GRAPHICAL NOTATION
- “The technique is particularly suited to parallel and distributed systems development as it supports concurrency. The technique is able to specify systems at a level that is independent of the choice of implementation (i.e. by software, hardware (electronic and/or mechanical) or humans or a combination). This document may be cited in contracts for the development of systems (particularly distributed systems) or used by application developers or Petri net tool vendors or users.”
- В українські ДСТУ цей стандарт теж увійшов (без змін)!
- ДСТУ ISO/IEC 15909-1:2016 (ISO/IEC 15909-1:2004, IDT)

Елементи мережі Петрі

ЕЛЕМЕНТИ МЕРЕЖІ ПЕТРІ

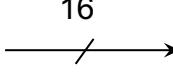
Перехід | позначає подію

Позиція ○ позначає умову

Дуга  позначає зв'язки між подіями та умовами

**Маркер(один)
(один)** ● позначає виконання
(або не виконання) умови

Багато фішок  позначає багатократне виконання умови

Багато дуг  позначає велику кількість зв'язків

Розробка мережі Петрі для дискретно-подійної системи

Для того, щоб представити систему засобами мережі Петрі потрібно:

- виділити події, що виникають в системі, і поставити у відповідність кожній події перехід мережі Петрі;
- з'ясувати умови, при яких виникає кожна з подій, і поставити у відповідність кожній умові позицію мережі Петрі;
- визначити кількість фішок у позиції мережі Петрі, що символізує виконання умови;
- з'єднати позиції та Переходи відповідно до логіки виникнення подій у системі: якщо умова передує виконанню події, то з'єднати в мережі Петрі відповідну позицію з відповідним Переходом; якщо умова являється наслідком виконання події, то з'єднати в мережі Петрі відповідний перехід з відповідною позицією;
- з'ясувати зміни, які відбуваються в системі при здійсненніожної події, і поставити у відповідність змінам переміщення визначеного кількості фішок із позицій в Переходи та з Переходів у позиції;
- визначити числові значення часових затримок в Переходах мережі Петрі;
- визначити стан мережі Петрі на початку моделювання.

Формальне означення класичної мережі Петрі

$$N = (\mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{W})$$

$\mathbf{P} = \{P\}$ - множина позицій;

$\mathbf{T} = \{T\}$ - множина переходів;

$$\mathbf{P} \cap \mathbf{T} = \emptyset$$

$\mathbf{A} \subseteq (\mathbf{P} \times \mathbf{T} \cup \mathbf{T} \times \mathbf{P})$ - множина дуг;

$\mathbf{W}: \mathbf{A} \rightarrow \mathbb{N}$ - множина натуральних чисел, що задають кратності дуг (кількість зв'язків);

Правило запуску переходу класичної мережі Петрі

- Якщо в усіх вхідних позиціях переходу є маркери у кількості, рівній кратності дуги, то умова запуску переходу виконана
- Якщо умова запуску переходу виконана, то з усіх вхідних позицій переходу маркери видаляються у кількості, рівній кратності дуги, а в усі вихідні позиції переходу маркери додаються у кількості, рівній кратності дуги. Таким чином, в класичній мережі Петрі запуск переходу здійснюється миттєво.

Формальний опис запуску переходу

Множина вхідних позицій переходу:

$$\bullet T = \{P \in P | (P, T) \in A\}$$

Множина вихідних позицій переходу:

$$T^\bullet = \{P \in P | (T, P) \in A\}$$

Умова запуску переходу:

$$\forall P \in \bullet T \quad M_P \geq W_{P,T} \rightarrow I(T) = 1$$

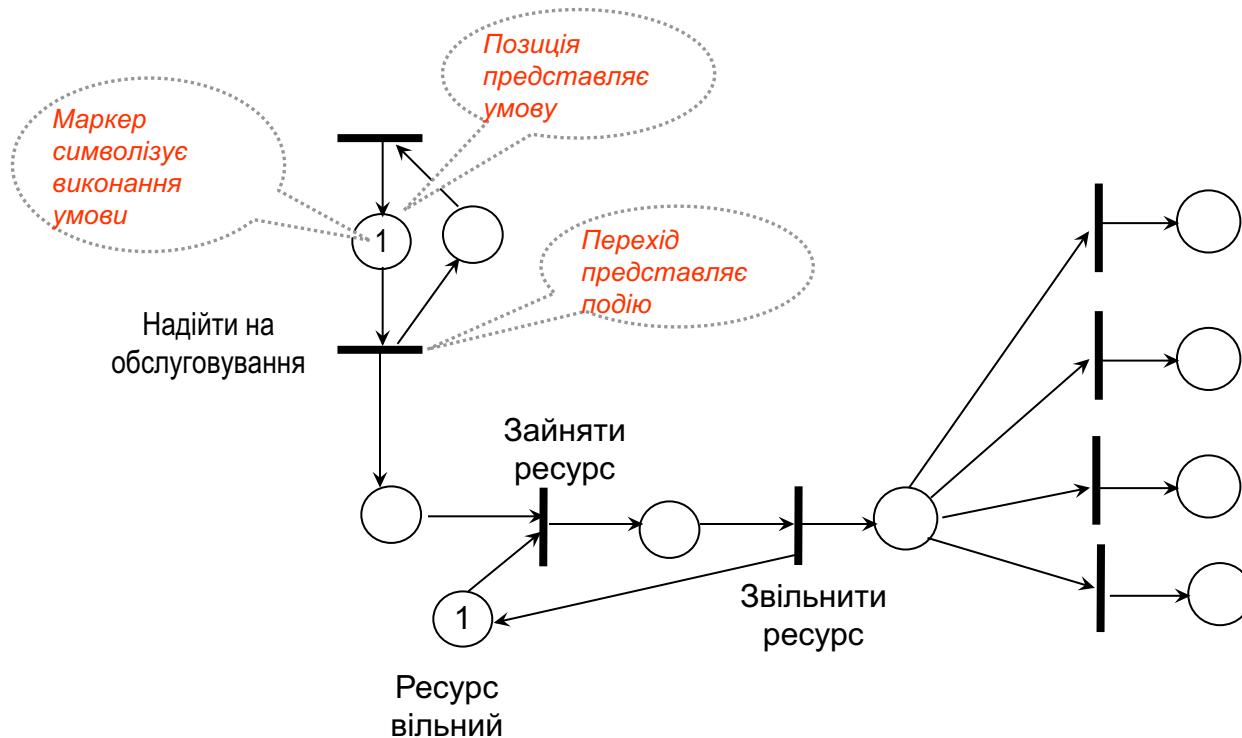
$$\exists P \in \bullet T \quad M_P < W_{P,T} \rightarrow I(T) = 0$$

Запуск переходу

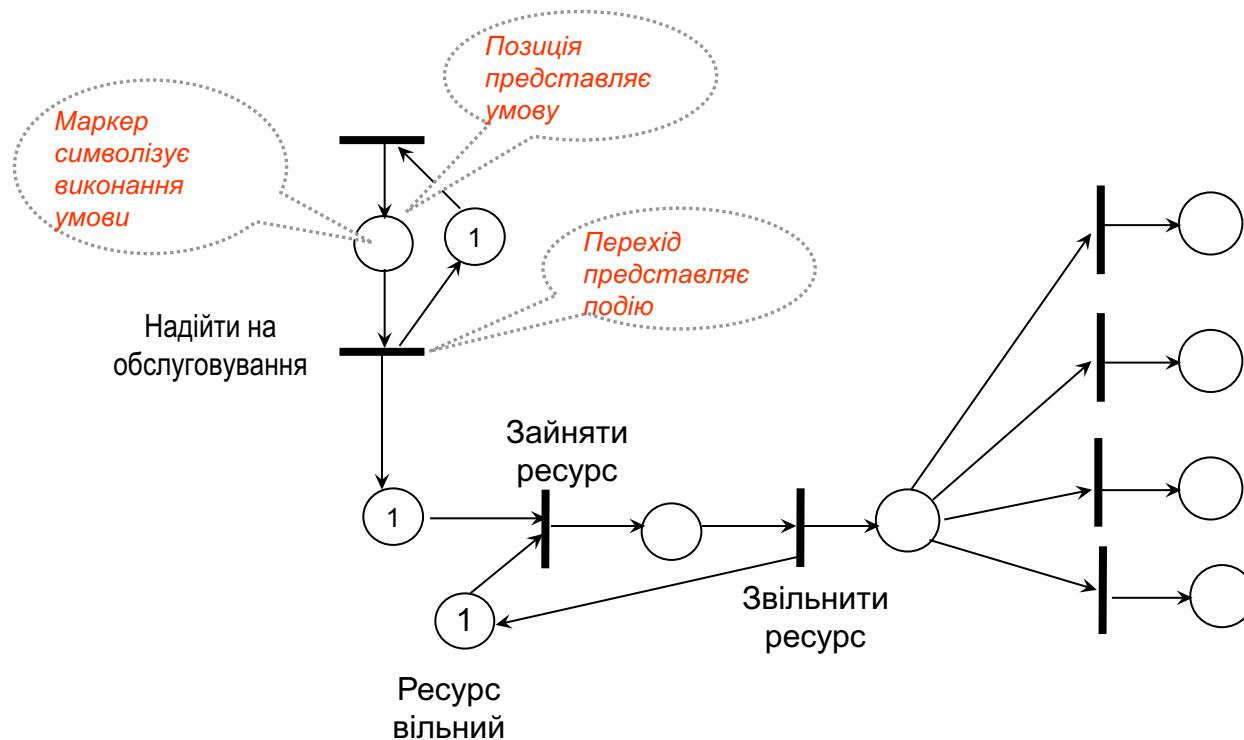
$$I(T) = 1 \rightarrow \begin{aligned} \forall P \in \bullet T \quad M'_P &= M_P - W_{P,T} \\ \forall P \in T^\bullet \quad M''_P &= M'_P + W_{P,T} \end{aligned}$$

Класичні мережі Петрі

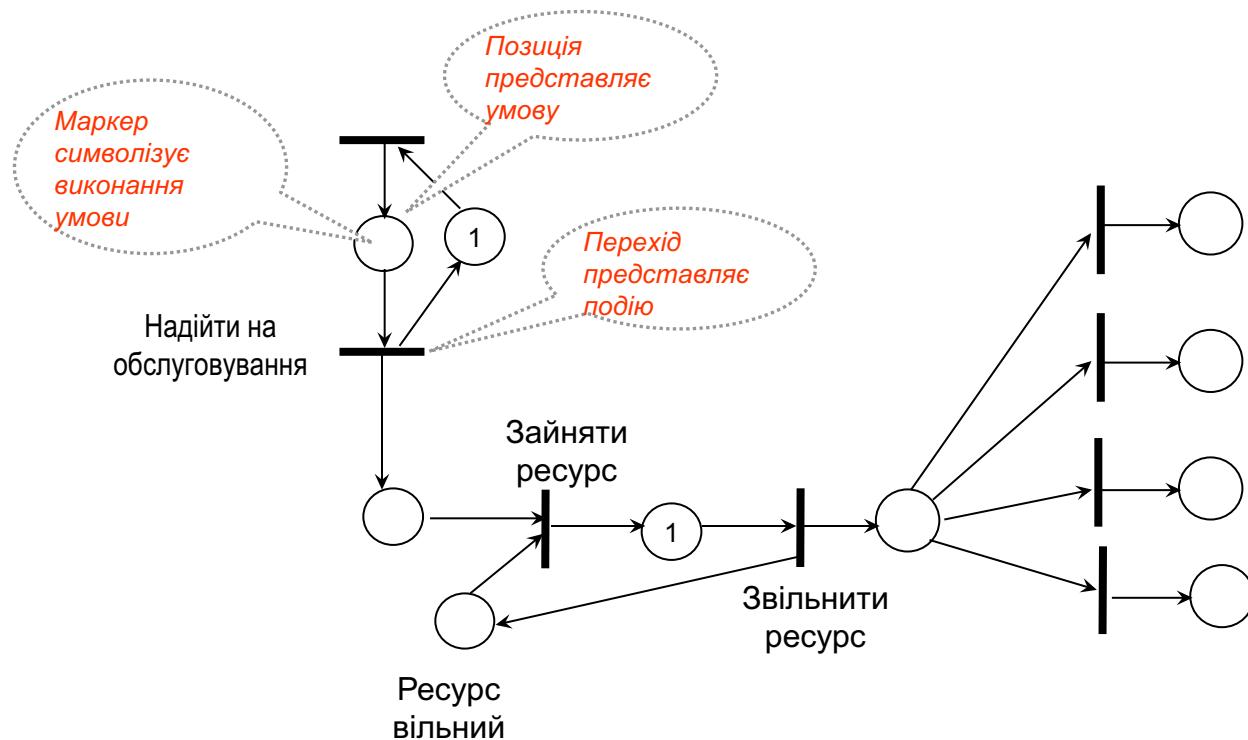
Правило запуску перехідів. Конфліктні переходи



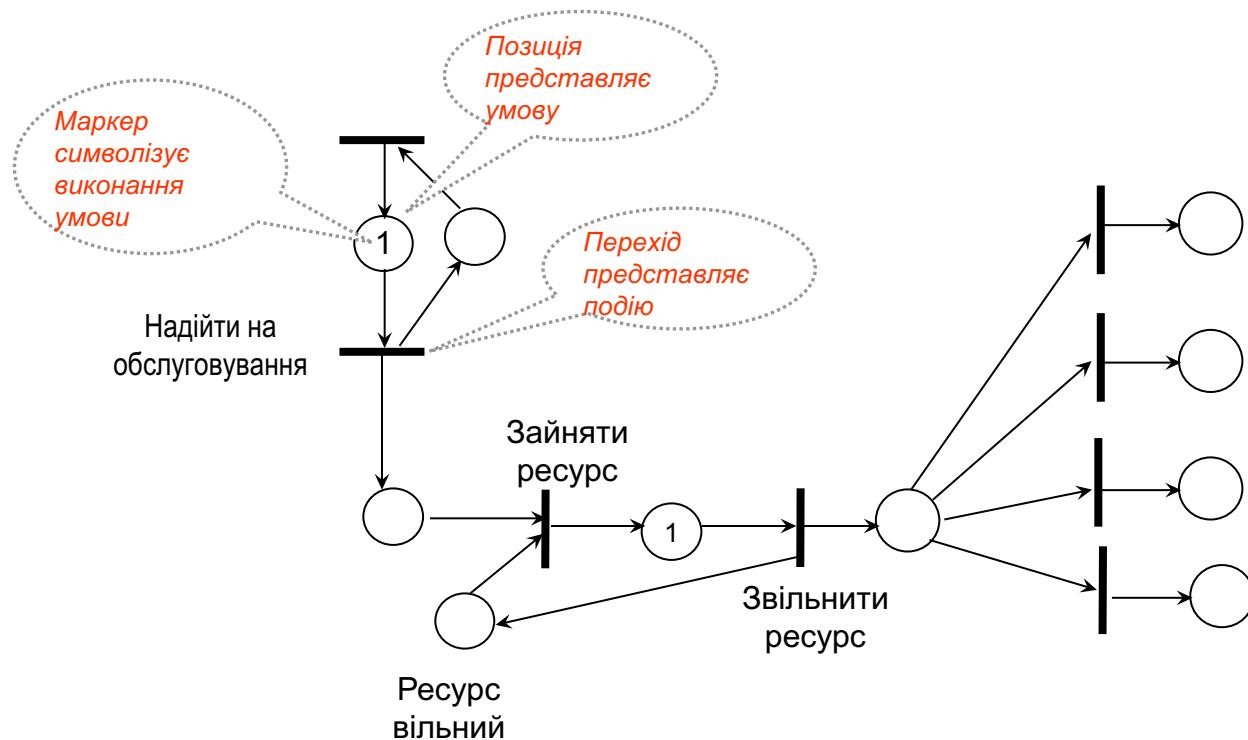
Класичні мережі Петрі



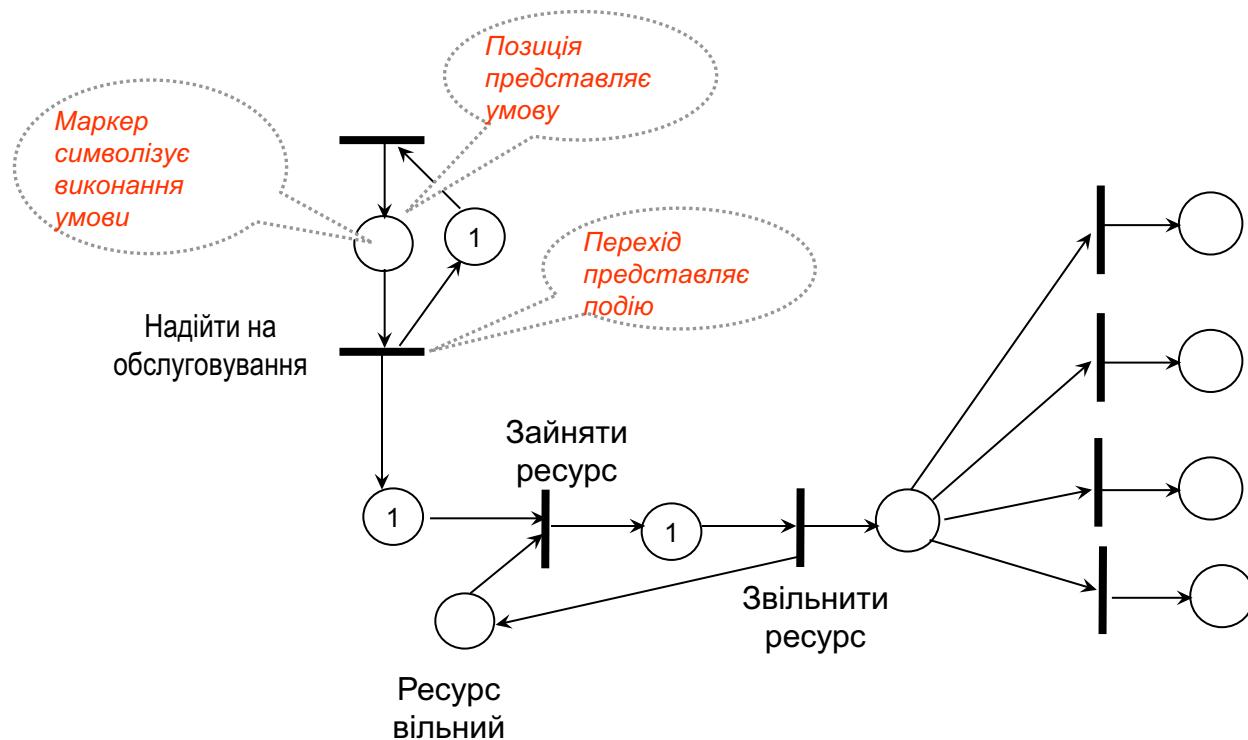
Класичні мережі Петрі



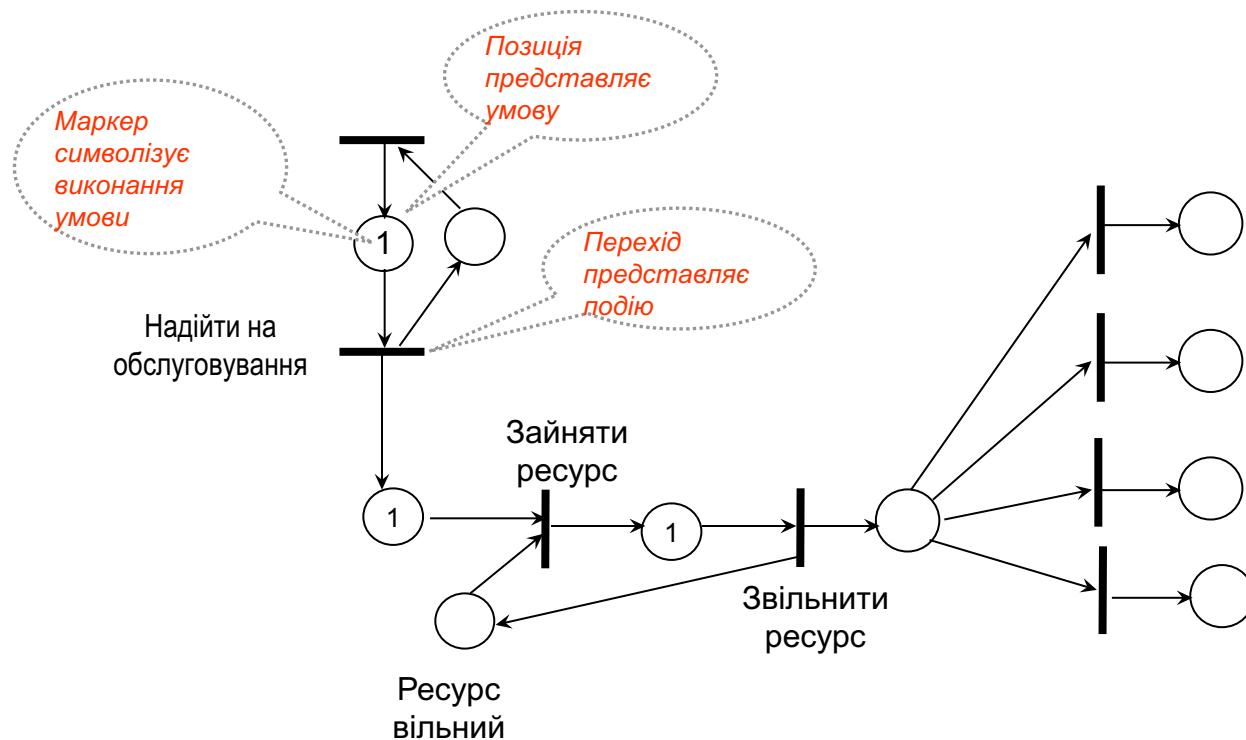
Класичні мережі Петрі



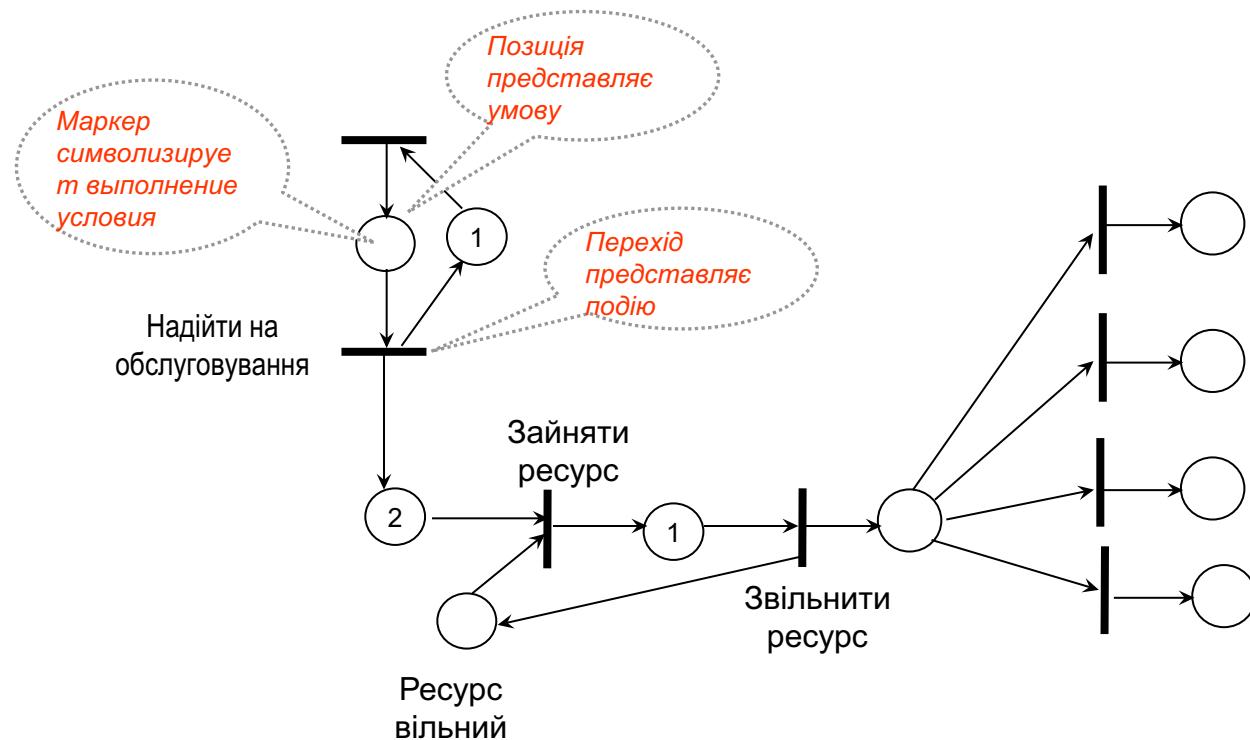
Класичні мережі Петрі



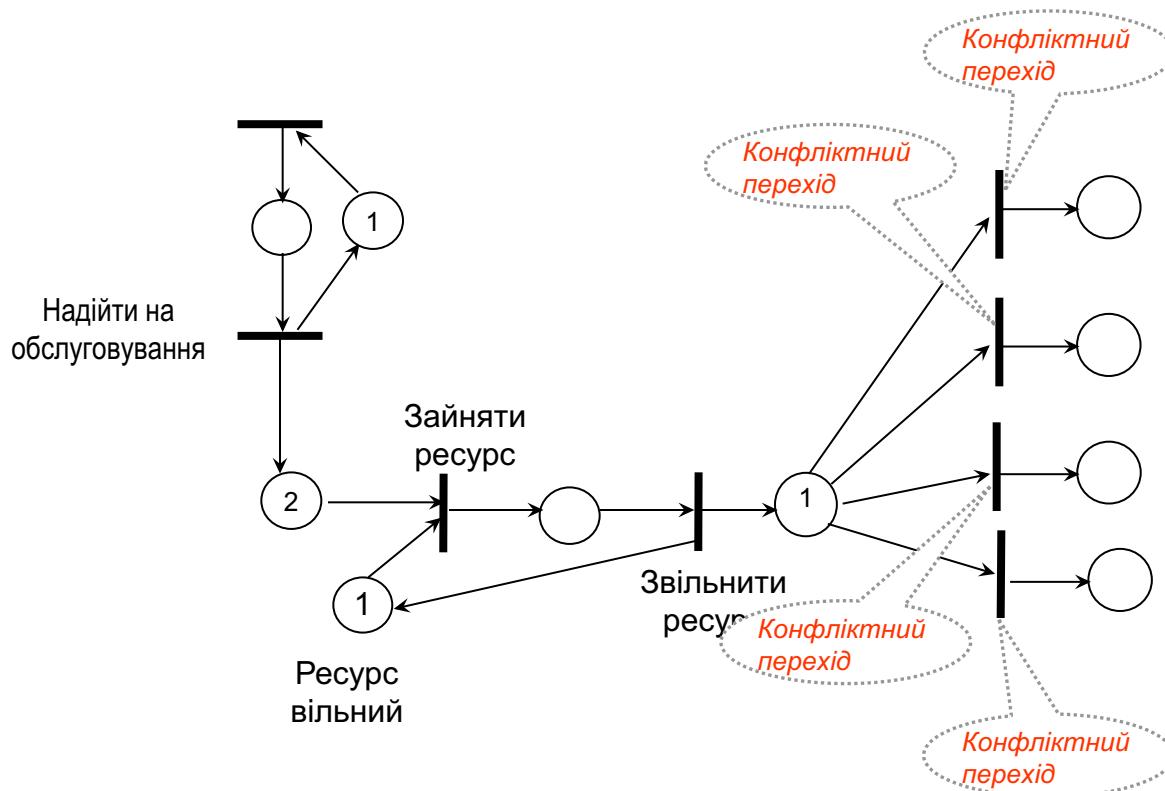
Класичні мережі Петрі



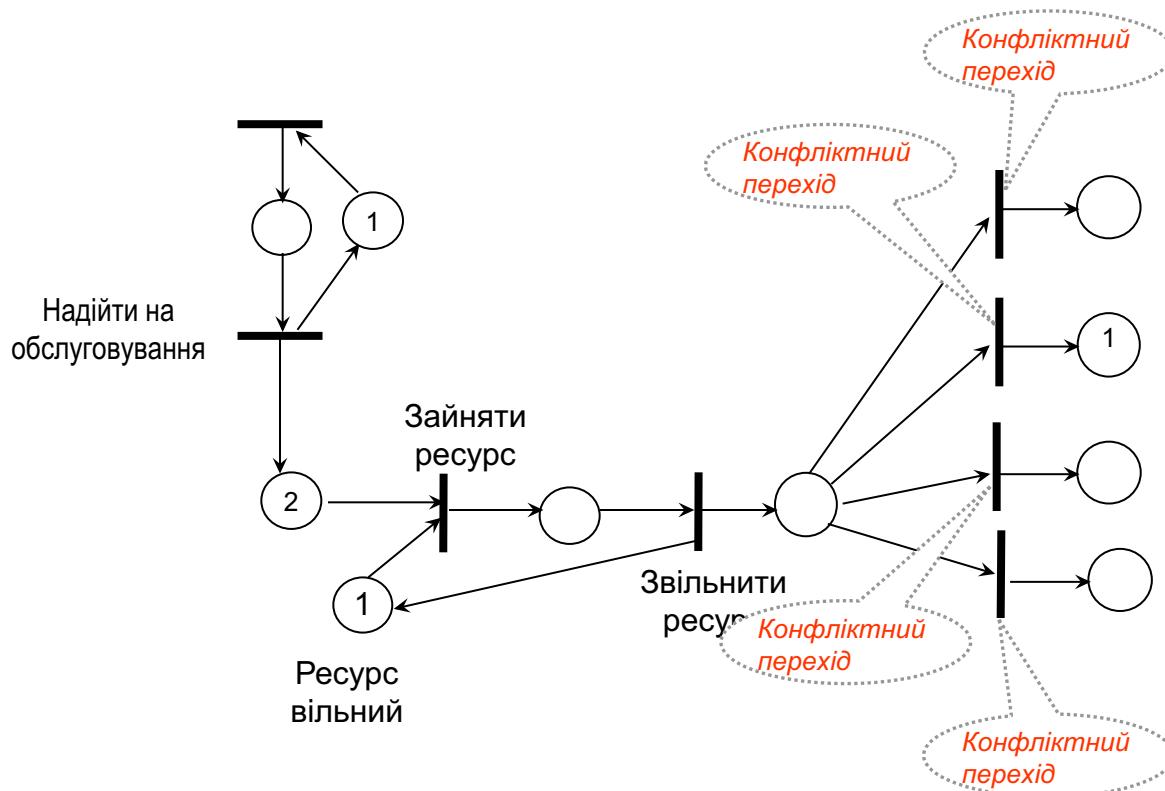
Класичні мережі Петрі



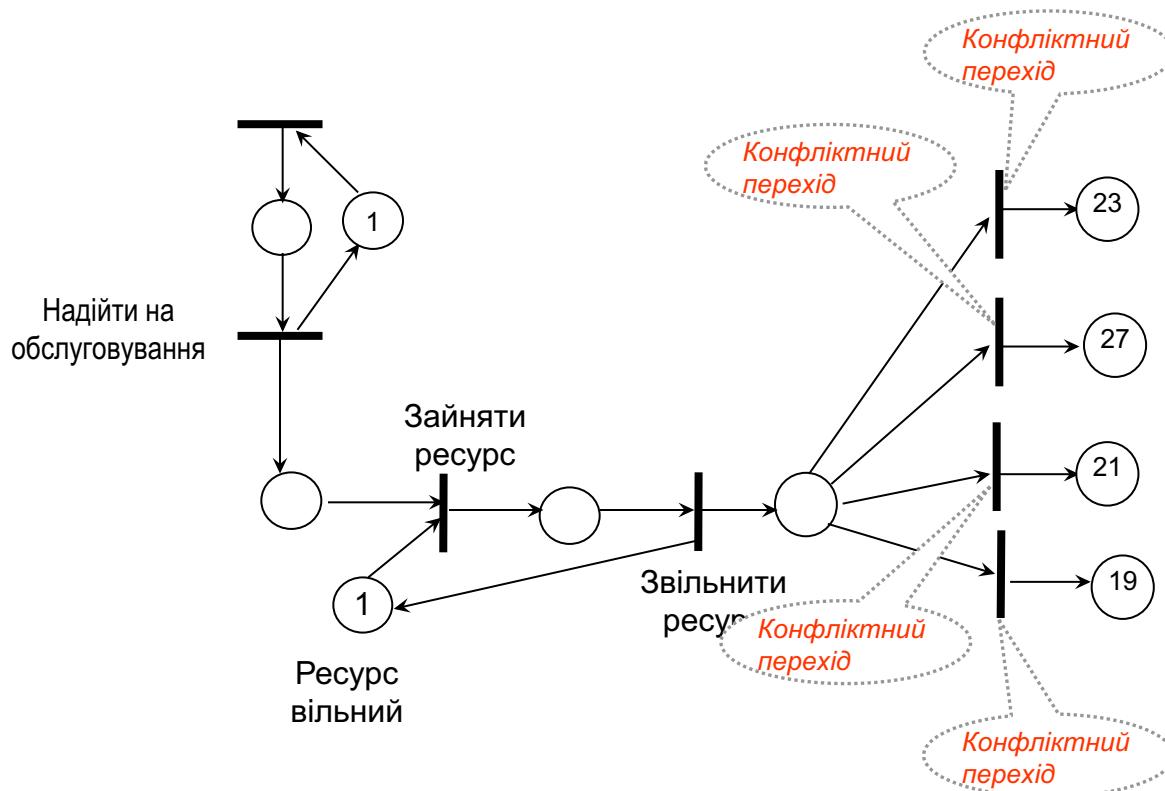
Класичні мережі Петрі



Класичні мережі Петрі



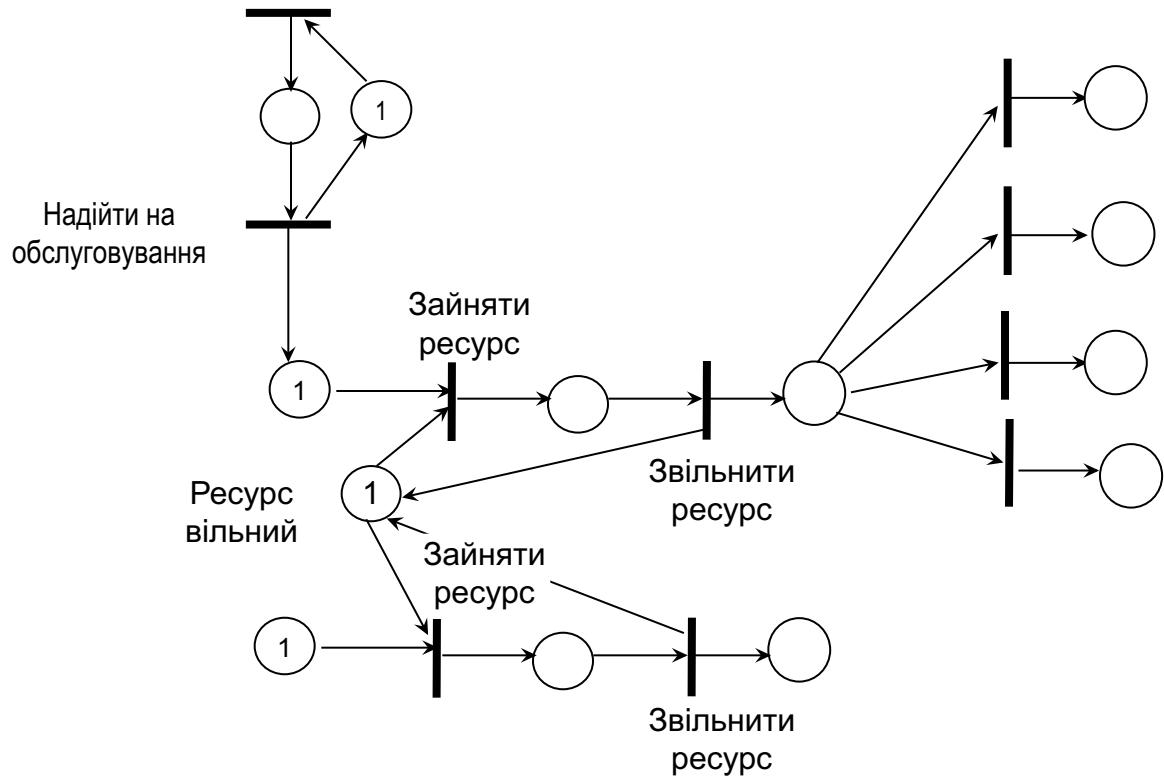
Класичні мережі Пєтрі



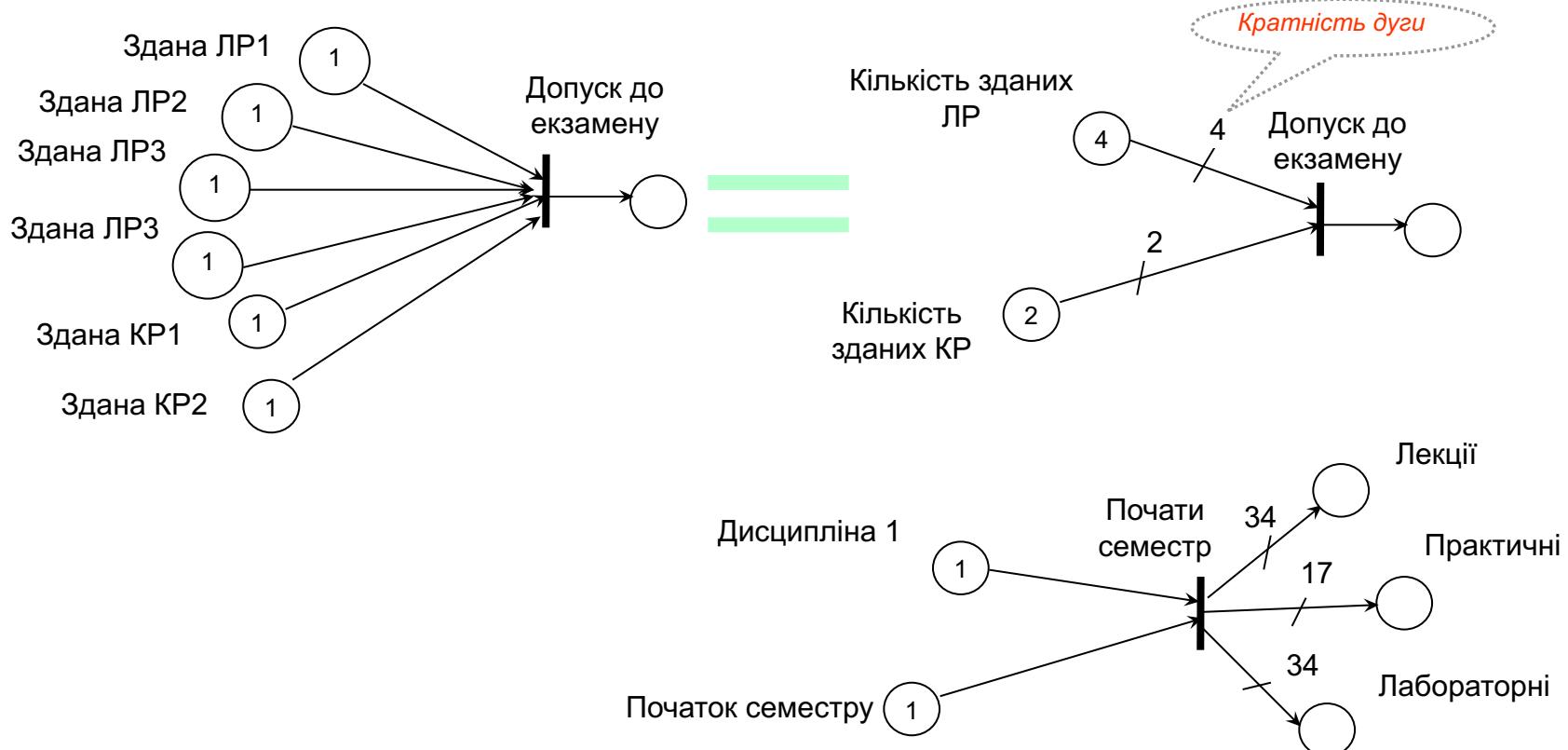
!! Будь-яку мережу масового обслуговування можна представити мережею Петрі

!! Будь-який цифровий автомат можна представити мережею Петрі

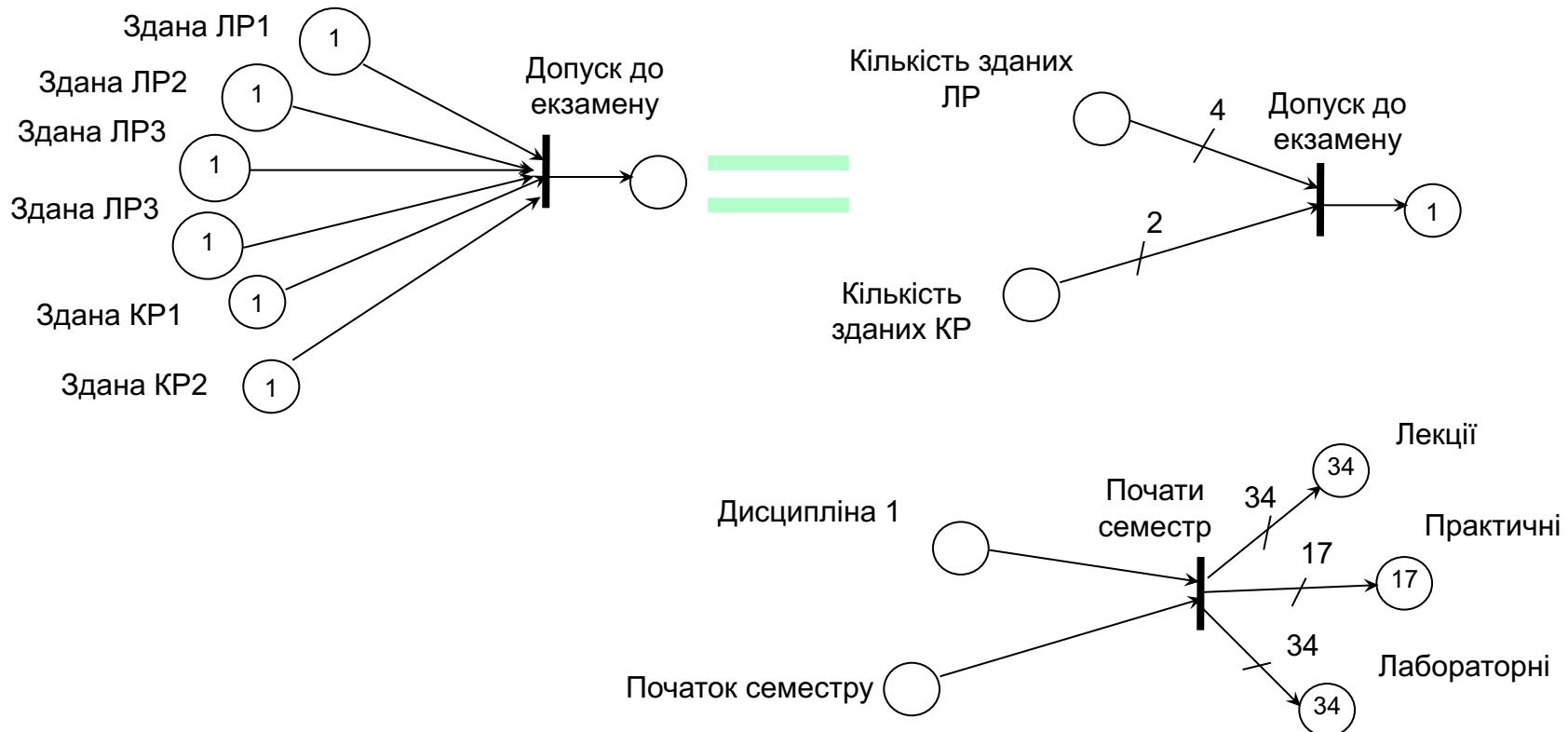
Класичні мережі Петрі: використання спільногого ресурсу



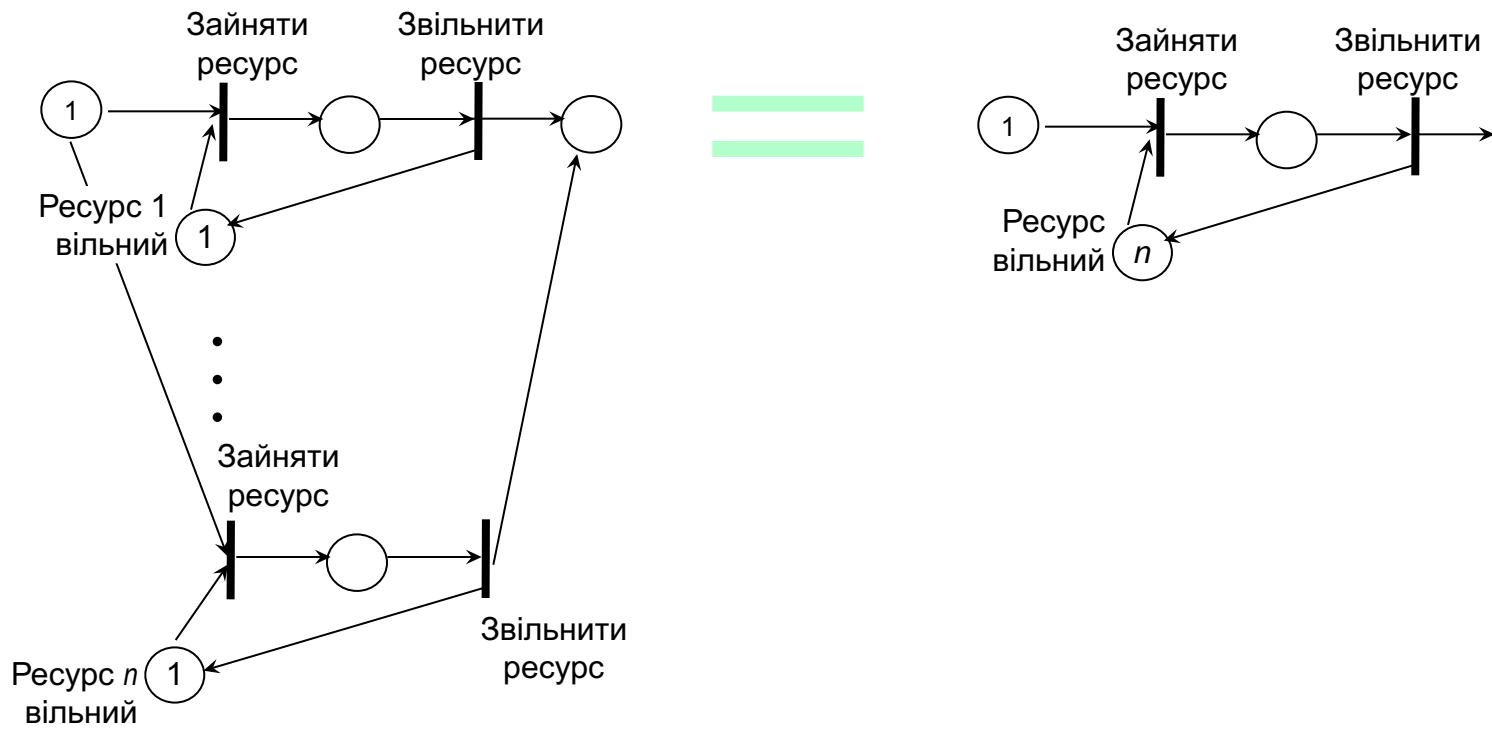
Класичні мережі Петрі з кратними зв'язками



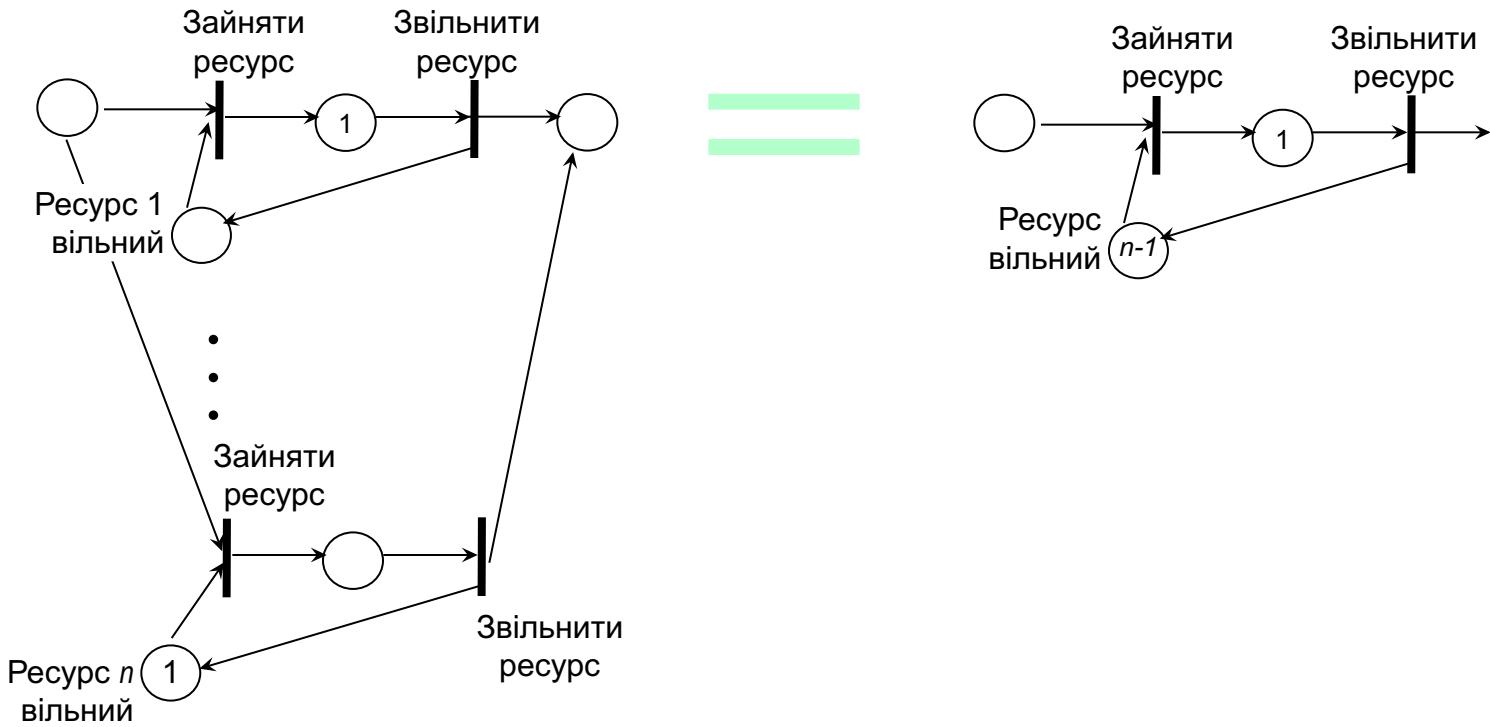
Класичні мережі Петрі з кратними зв'язками



Класичні мережі Петрі з багатоканальними переходами



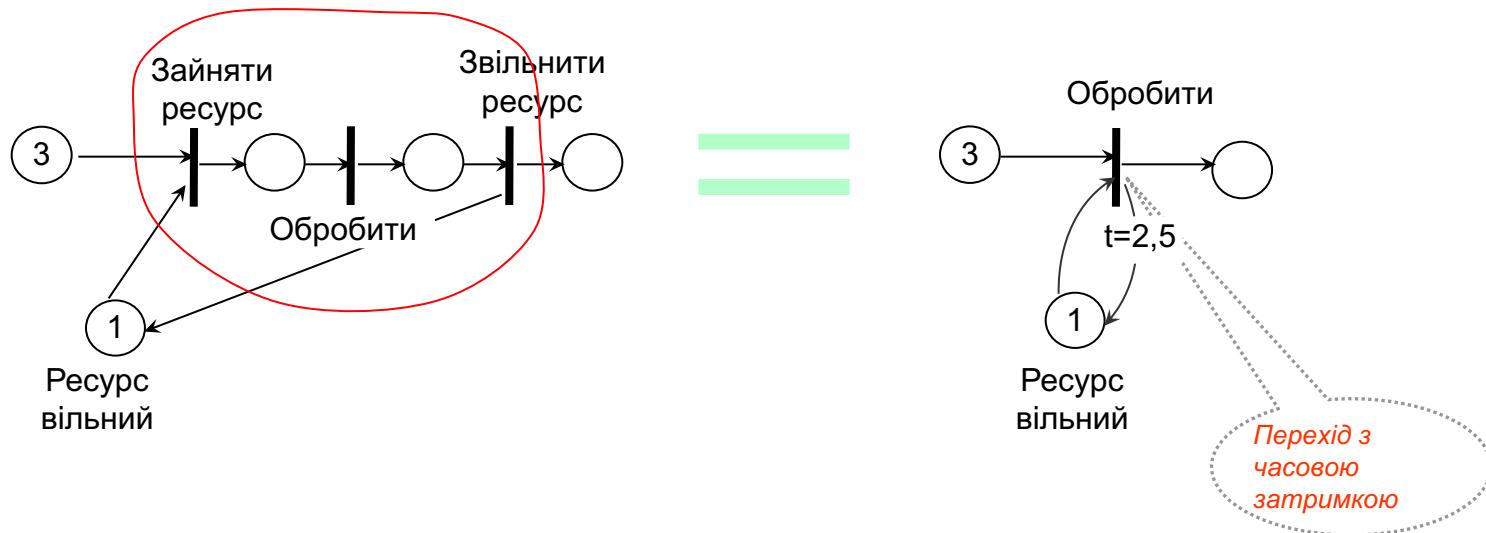
Класичні мережі Петрі з багатоканальними переходами



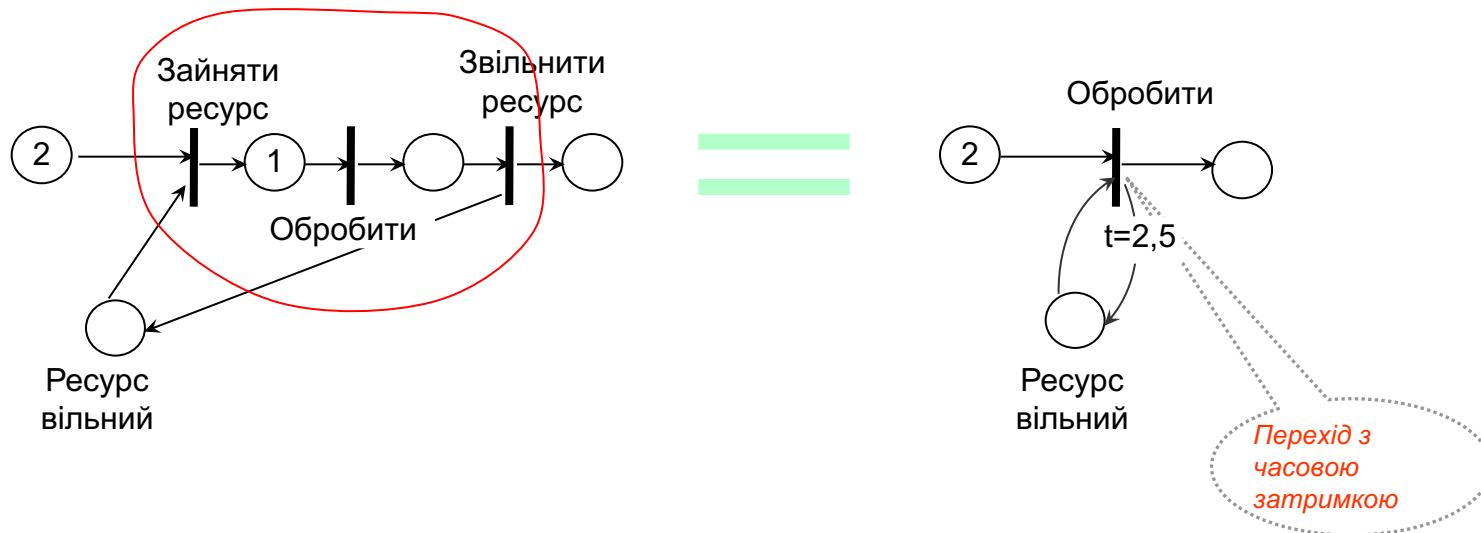
Правило запуску переходу мережі Петрі з багатоканальними переходами

- Якщо в усіх вхідних позиціях переходу є маркери у кількості, рівній кратності дуги, то умова запуску переходу виконана
- Повторювати доки виконана умова запуску переходу: з усіх вхідних позицій переходу маркери видаляються у кількості, рівній кратності дуги, збільшити кількість зайнятих каналів переходу на 1. Отже, багатоканальність переходу означає багатократність входу маркерів в переход.
- При виході маркерів з переходу доки лічильник зайнятих переходів більше нуля виконувати: в усі вихідні позиції переходу маркери додаються у кількості, рівній кратності дуги, і зменшити лічильник зайнятих каналів на 1.

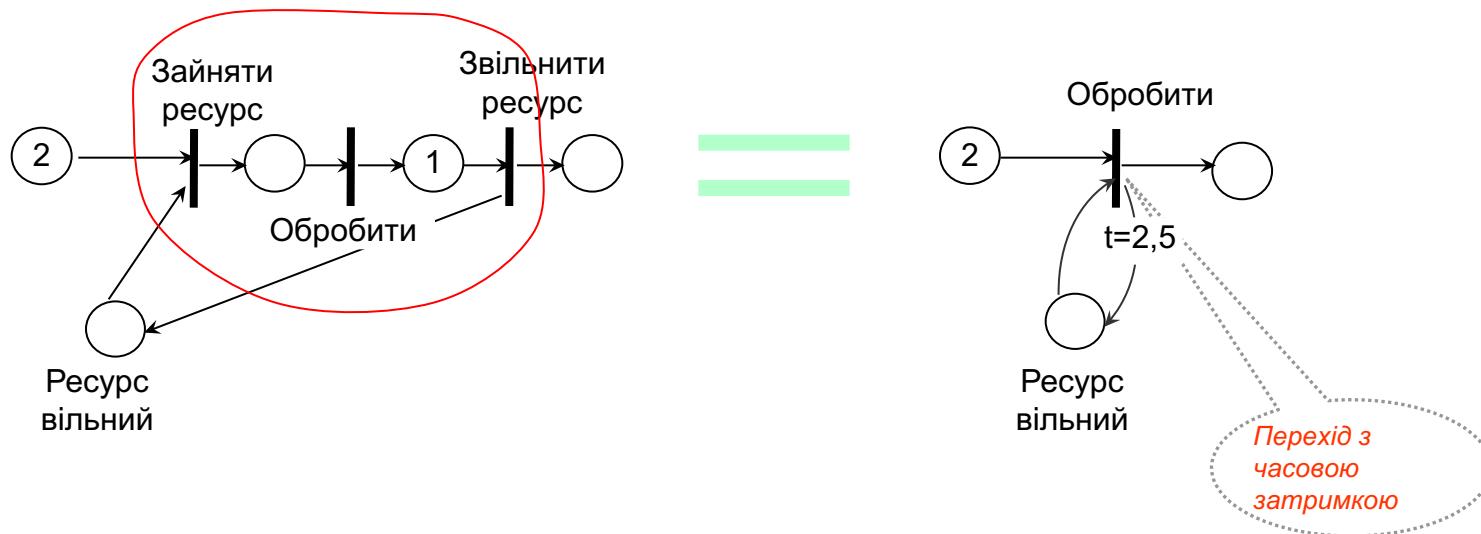
Мережі Петрі з часовими затримками



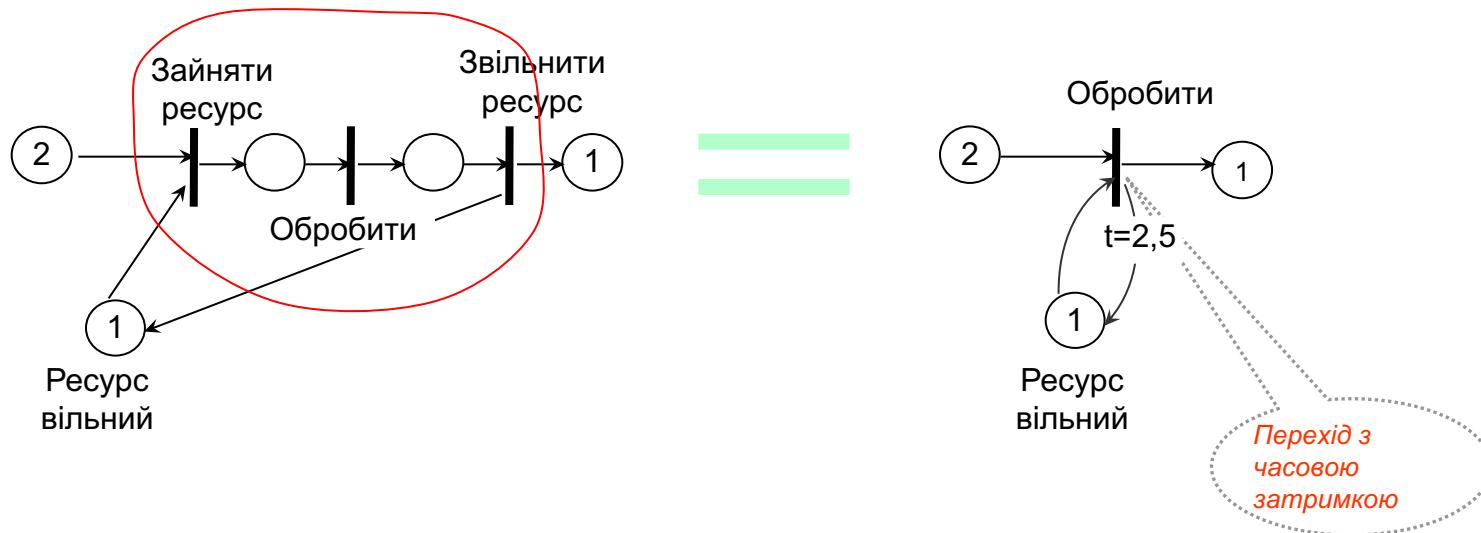
Мережі Петрі з часовими затримками



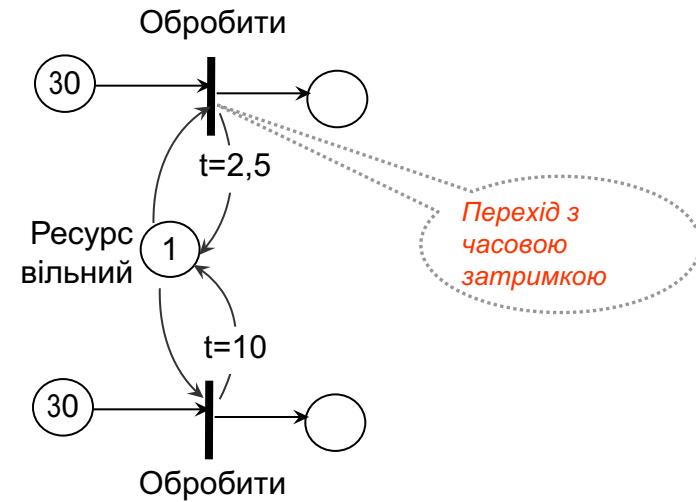
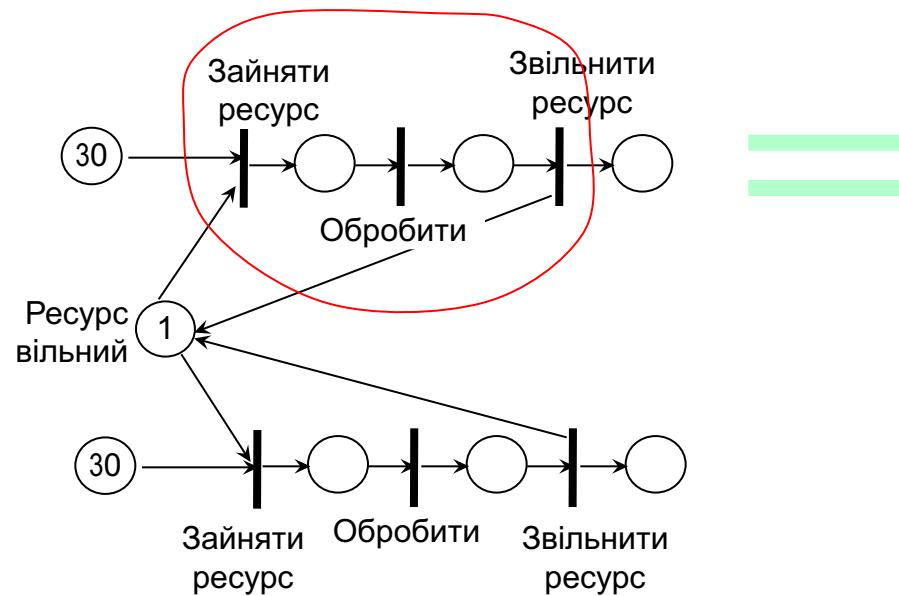
Мережі Петрі з часовими затримками



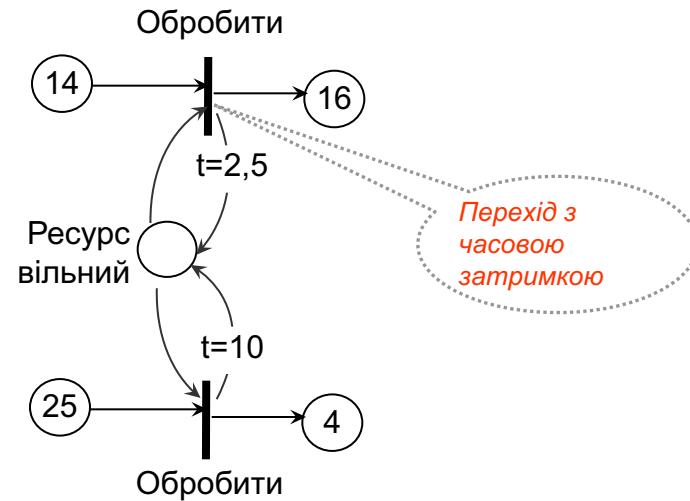
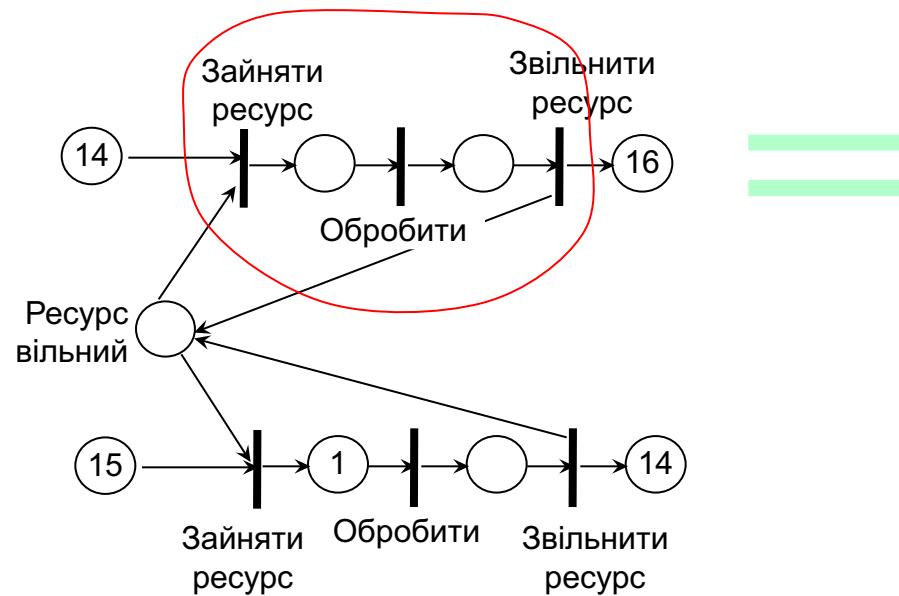
Мережі Петрі з часовими затримками



Мережі Петрі з часовими затримками



Мережі Петрі з часовими затримками



Запуск переходу мережі Петрі з часовими затримками

- Якщо момент виходу з переходу співпадає з поточним моментом часу, то виконати вихід маркерів з переходу: в усі вихідні позиції переходу маркери додаються у кількості, рівній кратності дуги, а момент виходу з переходу змінюється на нескінченність.
- Якщо в усіх вхідних позиціях переходу є маркери у кількості, рівній кратності дуги, то умова запуску переходу виконана. Якщо умова запуску переходу виконана, то з усіх вхідних позицій переходу маркери видаляються у кількості, рівній кратності дуги.

Таким чином, в мережі Петрі запуск переходу здійснюється у дві дії – вихід та вхід маркерів, між якими відбувається часова затримка.

Запуск переходу мережі Петрі з часовими затримками та багатоканальними переходами

- Якщо момент виходу з каналу переходу співпадає з поточним моментом часу, то виконати вихід маркерів з переходу: в усі вихідні позиції переходу маркери додаються у кількості, рівній кратності дуги, а момент виходу з каналу переходу змінюється на нескінченність.
- Повторювати доки виконана умова запуску переходу: з усіх вхідних позицій переходу маркери видаляються у кількості, рівній кратності дуги, а до моментів виходу з переходу додати новий, що дорівнює поточному моменту часу плюс часова затримка (або замінити нескінченність на новий момент виходу).

Таким чином, в мережі Петрі запуск переходу здійснюється у дві дії – вихід та багатократний вхід маркерів, між якими відбувається часова затримка.

Формальне означення стохастичної мережі Петрі

$$N = (\mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{W}, \mathbf{K}, \mathbf{R})$$

$$\mathbf{P} = \{P\}$$

- множина позицій;

$$\mathbf{T} = \{T\}$$

- множина переходів;

$$\mathbf{P} \cap \mathbf{T} = \emptyset$$

$$\mathbf{A} \subseteq (\mathbf{P} \times \mathbf{T} \cup \mathbf{T} \times \mathbf{P})$$

- множина дуг;

$$\mathbf{W}: \mathbf{A} \rightarrow \mathbb{N}$$

- множина натуральних чисел, що задають кратності дуг (кількість зв'язків);

$$\mathbf{K} = \{(c_T, b_T) / T \in \mathbf{T}, c_T \in \mathbb{N}, b_T \in [0;1]\}$$

- множина пар значень, що задають пріоритет та ймовірність запуску переходів;

$$\mathbf{R}: \mathbf{T} \rightarrow \mathcal{R}_+$$

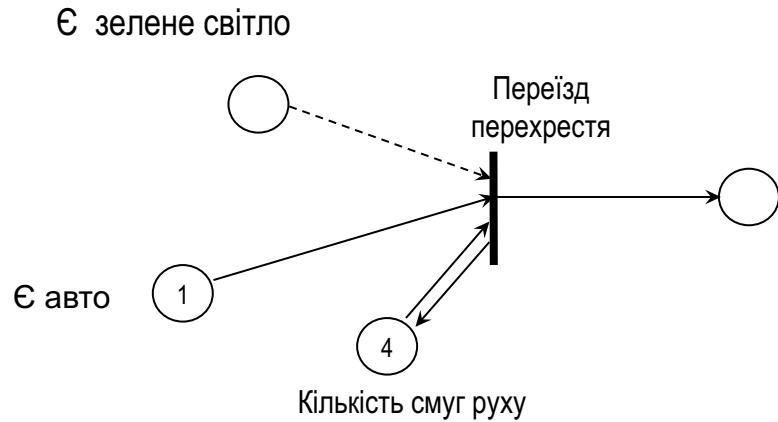
- множина невід'ємних чисел, що характеризують часові затримки

Стохастична мережа Петрі 1) містить часові затримки переходів, що можуть бути визначені випадковою величиною (з заданим законом розподілом), 2) містить розв'язання конфлікту переходів з заданою ймовірністю

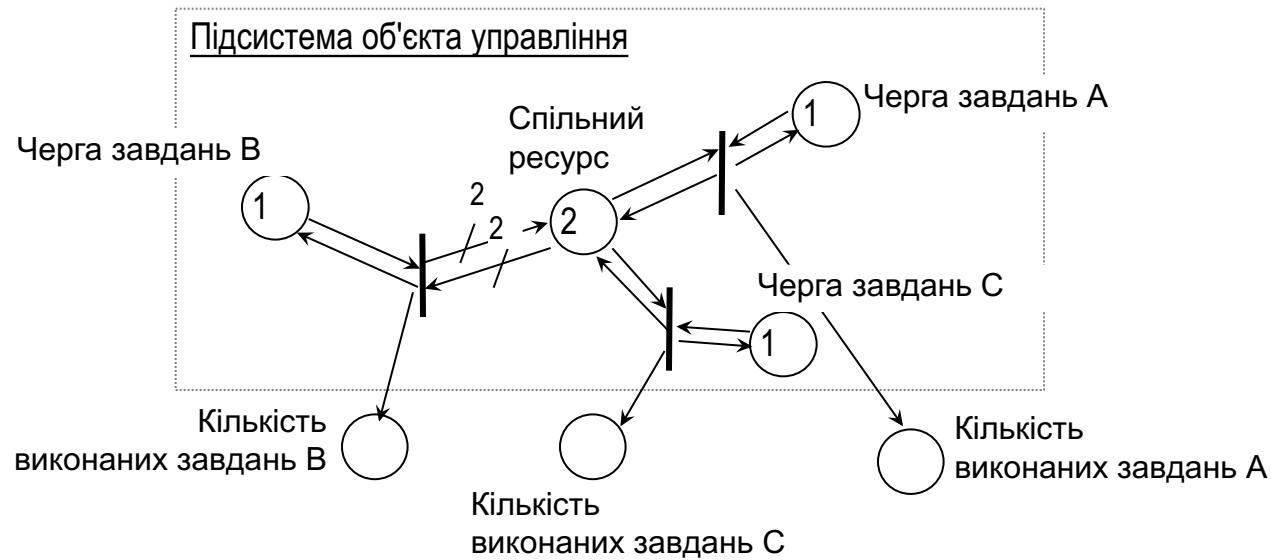
Стохастична мережа Петрі є узагальненням мереж Петрі:

- При нульових часових затримках та рівноймовірнісному способі розв'язання конфліктів вона еквівалентна класичній мережі Петрі,
- При детермінованих затримках та рівноймовірнісному способі розв'язання конфліктів вона еквівалентна детермінованій мережі Петрі з часовими затримками

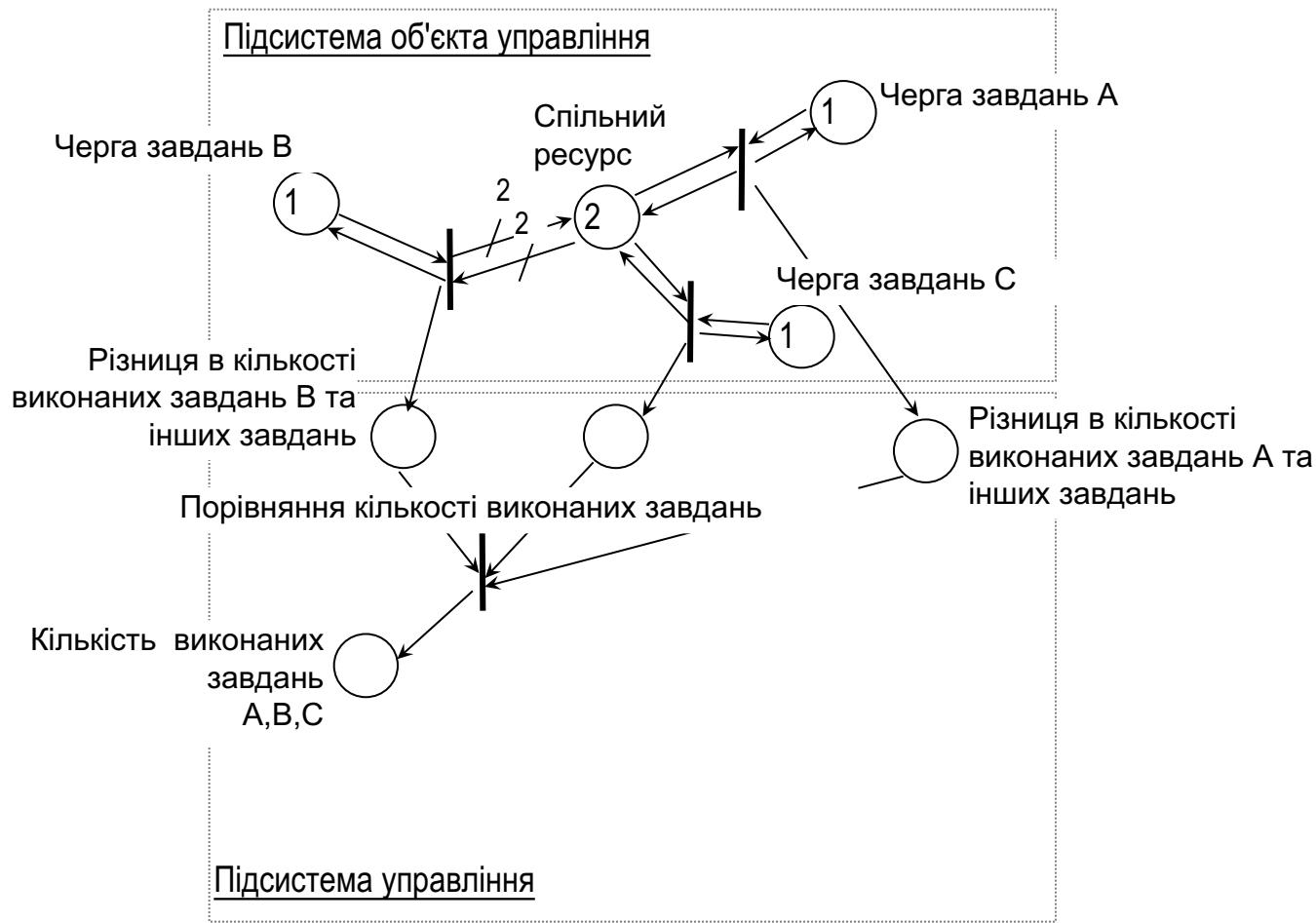
Мережі Петрі з інформаційними зв'язками



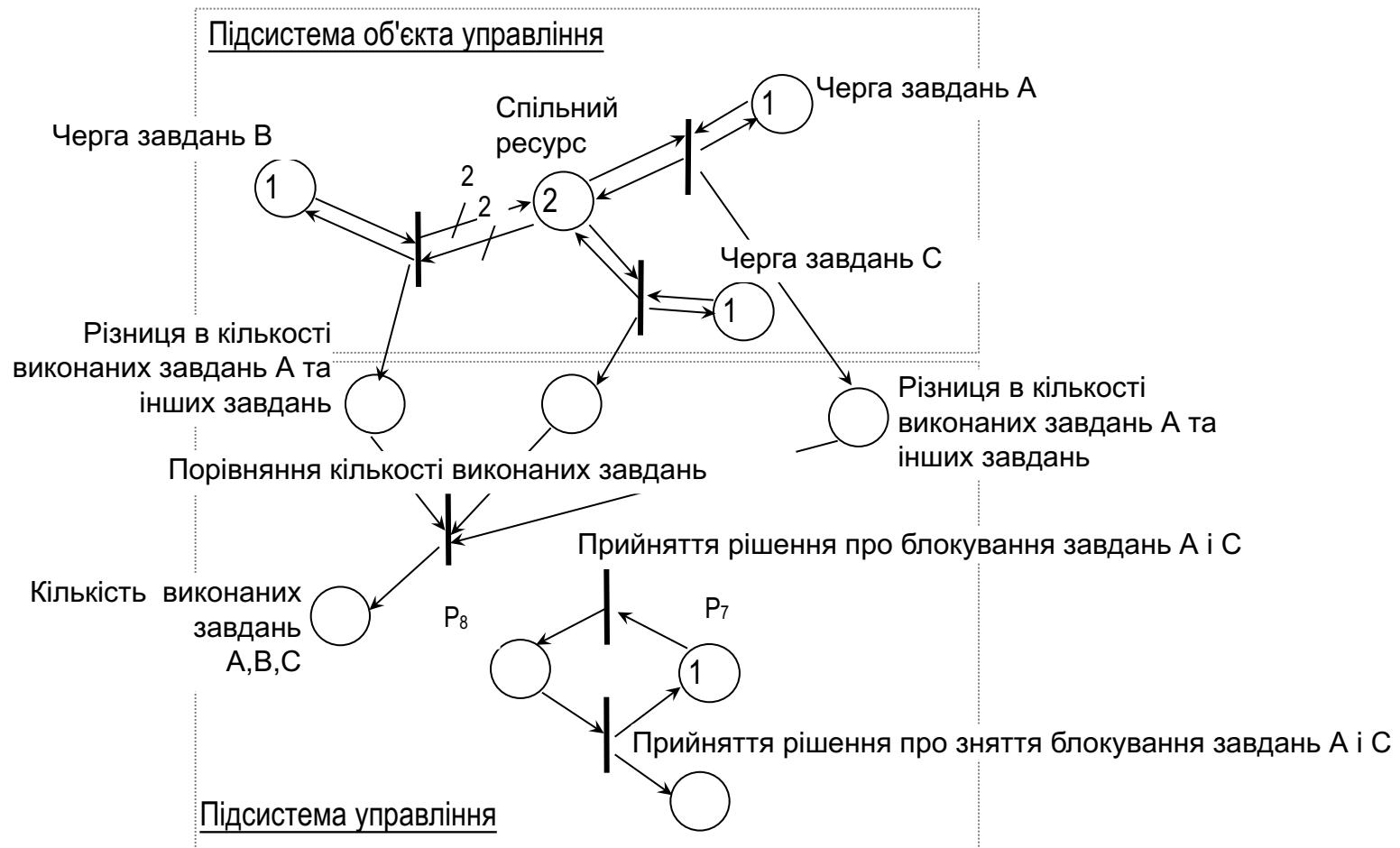
Приклад моделювання стохастичною мережею Петрі динамічного управління розподілом ресурсів



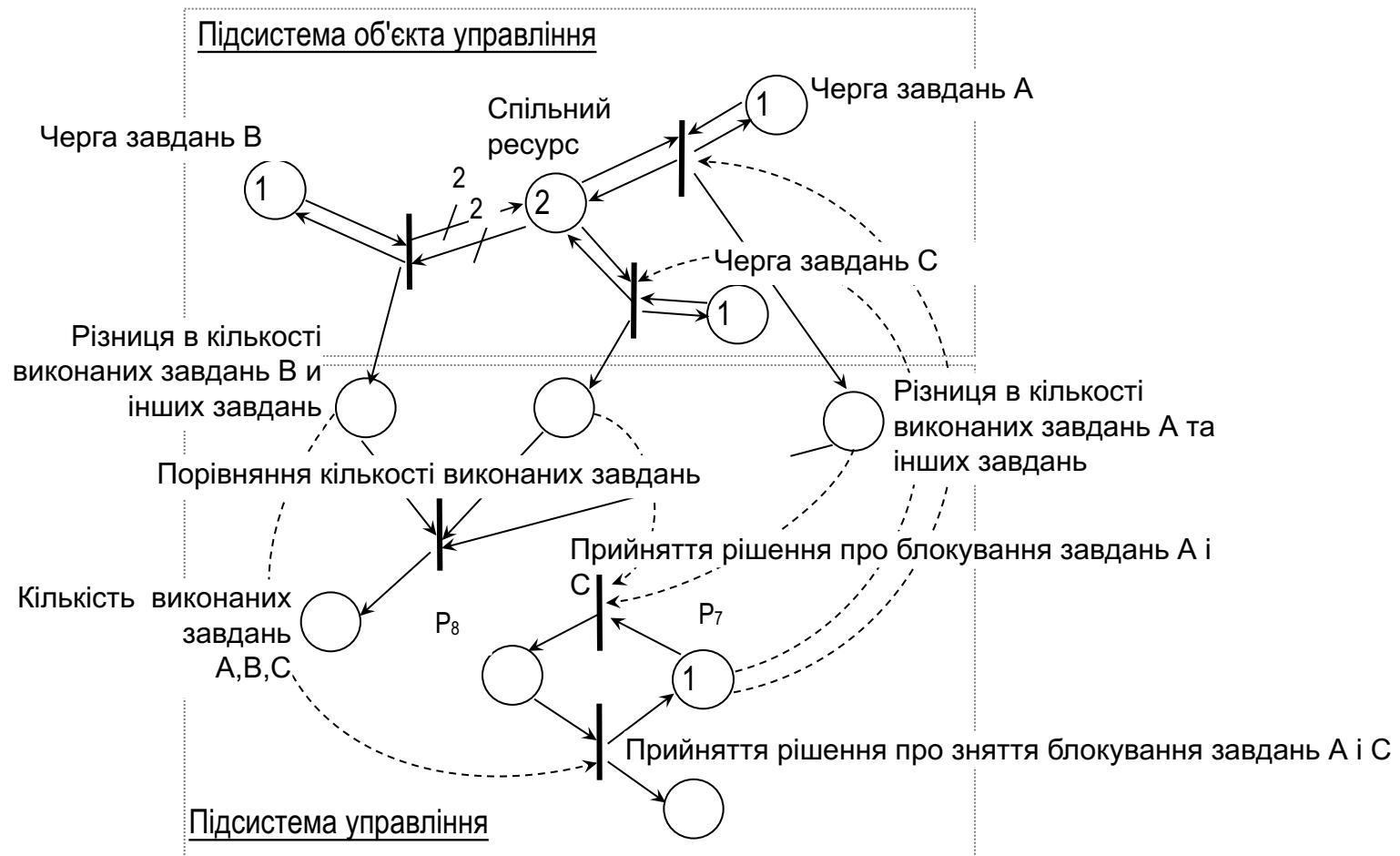
Приклад моделювання стохастичною мережею Петрі динамічного управління розподілом ресурсів



Приклад моделювання стохастичною мережею Петрі динамічного управління розподілом ресурсів



Приклад моделювання стохастичною мережею Петрі динамічного управління розподілом ресурсів



Формальне означення мережі Петрі з інформаційними зв'язками

$$N = (\mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{W}, \mathbf{I})$$

$\mathbf{P} = \{P\}$ - множина позицій;

$\mathbf{T} = \{T\}$ - множина переходів;

$$\mathbf{P} \cap \mathbf{T} = \emptyset$$

$\mathbf{A} \subseteq (\mathbf{P} \times \mathbf{T} \cup \mathbf{T} \times \mathbf{P})$ - множина дуг;

$\mathbf{I} \subseteq (\mathbf{P} \times \mathbf{T})$ - множина інформаційних дуг;

$\mathbf{W}: \mathbf{A} \cup \mathbf{I} \rightarrow \mathbb{N}$ - множина натуральних чисел, що задають кратності дуг (кількість зв'язків);

Формальне означення стохастичної мережі Петрі з інформаційними зв'язками

$$N = (\mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{W}, \mathbf{I}, \mathbf{K}, \mathbf{R})$$

$$\mathbf{P} = \{P\}$$

- множина позицій;

$$\mathbf{T} = \{T\}$$

- множина переходів;

$$\mathbf{P} \cap \mathbf{T} = \emptyset$$

$$\mathbf{A} \subseteq (\mathbf{P} \times \mathbf{T} \cup \mathbf{T} \times \mathbf{P})$$

- множина дуг;

$$\mathbf{I} \subseteq (\mathbf{P} \times \mathbf{T})$$

- множина інформаційних дуг;

$$\mathbf{W}: \mathbf{A} \cup \mathbf{I} \rightarrow \mathbb{N}$$

- множина натуральних чисел, що задають кратності дуг (кількість зв'язків);

$$\mathbf{K} = \{(c_T, b_T) / T \in \mathbf{T}, c_T \in \mathbb{N}, b_T \in [0;1]\}$$

- множина пар значень, що задають пріоритет та ймовірність запуску переходів;

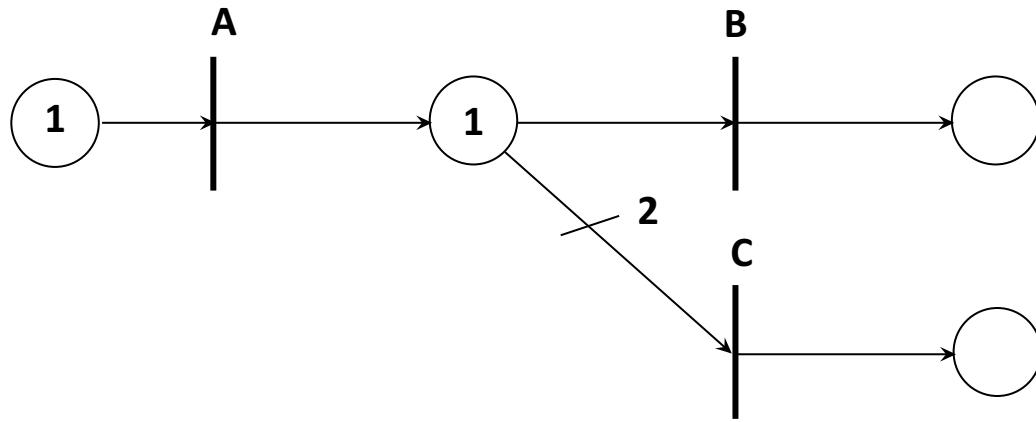
$$\mathbf{R}: \mathbf{T} \rightarrow \mathcal{R}_+$$

- множина невід'ємних чисел, що характеризують часові затримки

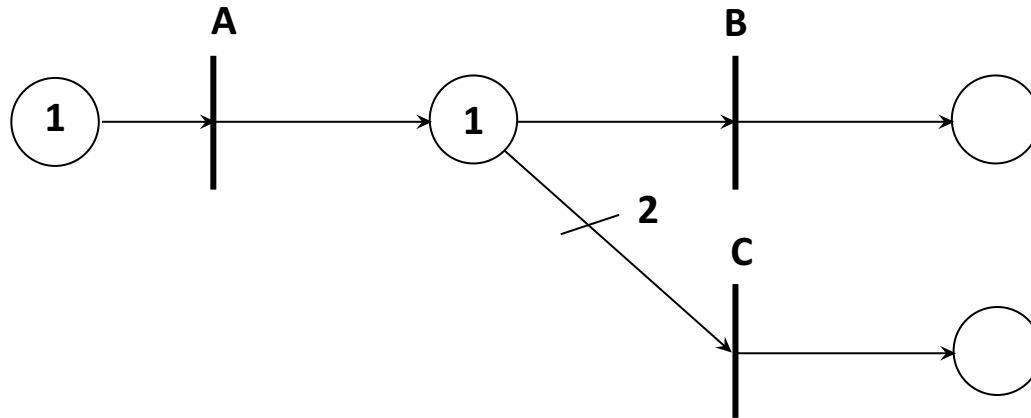
Лекція 7

Формалізм стохастичної мережі
Петрі: приклади

Приклад



Зміна стану мережі Петрі залежить від реалізованого в алгоритмі правила спрацьовування переходів



Якщо переходи А, В, С миттєві (*класична* мережа Петрі), то буде обрано один з двох, для яких виконана умова запуску. Тобто можливі варіанти А або В. Якщо станеться подія А, то на наступному кроці можливий запуск переходу С або В. Конфлікт між ними вирішується випадково, але у близько половині випадків переход С спрацює. Якщо переходи А, В, С з часовою затримкою (*стохастична* мережа Петрі), то здійснюється вхід маркерів в переходи А і В, в результаті якого досягається маркування, за якого для жодного з переходів мережі Петрі не виконана умова запуску. Після цього виконується вихід маркерів з переходів. Буде обрано один з переходів як найближча подія (випадково з двох) і виконано вихід з нього, за наступним кроком буде виконано інший переход як найближчу подію і виконано вихід з нього. Отже, у другому випадку немає можливості появи 2 маркерів в позиції, яка є умовою запуску переходу С.

Порівняння зміни стану мережі Петрі

Зауваження щодо позначення переходу мережі Петрі

Різні програмні засоби з імітації моделей мереж Петрі використовують позначення переходу у вигляді планки або прямокутника, щоб підкреслити його специфіку. Планку використовують для миттєвого переходу, прямокутник – для переходу з часовою затримкою. Оскільки в нашому курсі спираємось на формалізм мережі Петрі з часовими затримками та багатоканальними переходами, то можете зустріти в наших останніх роботах позначення переходу у вигляді прямокутника із заокругленими кутами, яким хочемо підкреслити саме особливість багатоканального переходу.

У програмних засобах для розробки та імітації мереж Петрі можна зустріти використання одночасно і планок, і прямокутників, що говорить про те, що програма імітації окремо опрацьовує переходи з затримками. У нашему програмному забезпеченні, яке пропонується для виконання практикуму, імітаційний алгоритм обробляє однаково усі переходи мережі Петрі. Переход з нульовою затримкою вважається таким, що еквівалентний переходу без часової затримки.

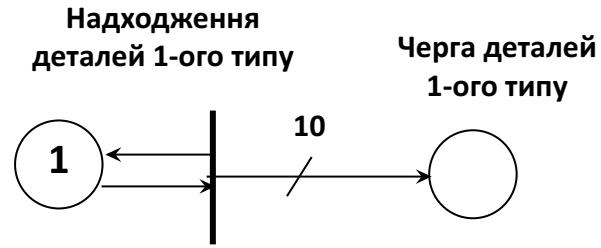
У нашему програмному забезпеченні [<https://github.com/Stetsenkolnna/PetriObjModelPaint>] використовуються планки, ширина яких залежить від пріоритету переходу: більш широка планка має більш високий пріоритет. Поступово переходимо до позначення у вигляді прямокутника із заокругленими кутами (слідкуйте за оновленнями). У навчальному посібнику з курсової роботи використано позначення у вигляді прямокутника із заокругленими кутами.

Приклад «Комплектувальний конвеєр»

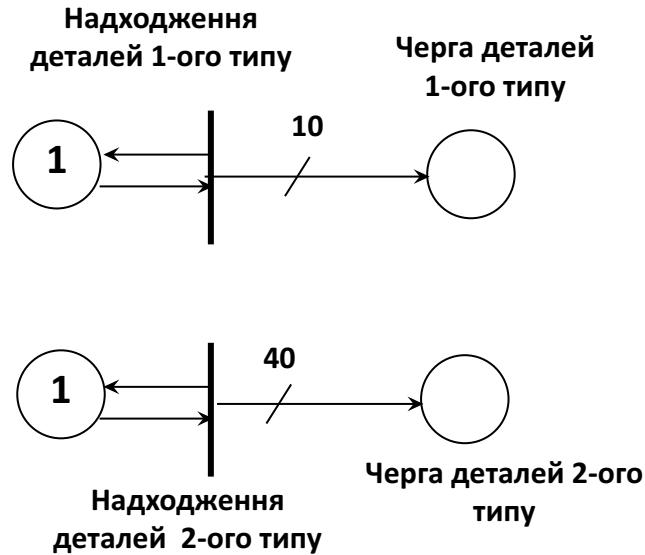
На комплектувальний конвеєр збирального цеху надходять: в середньому через 10 хвилин 10 деталей 1-го типу, в середньому через 40 хвилин 40 деталей 2-го типу. Конвеєр складається з секцій. Комплектація починається тільки за наявності 20 деталей кожного типу і продовжується 20 хвилин. У разі нестачі деталей секція конвеєру залишається порожньою.

Метою моделювання є визначення ймовірності порожньої секції, а також характеристик накопичення деталей по кожному типу.

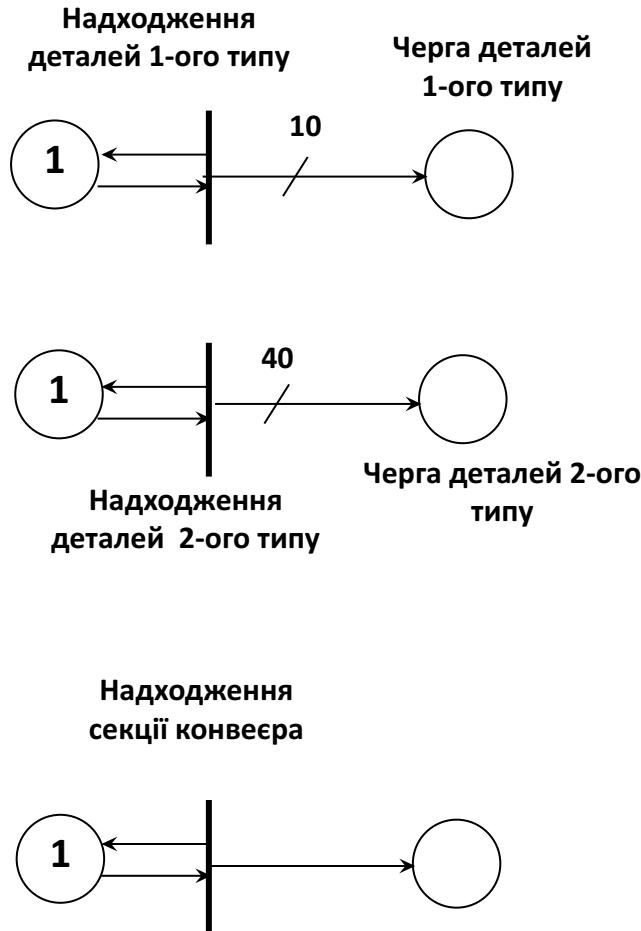
Приклад «Комплектувальний конвеєр»



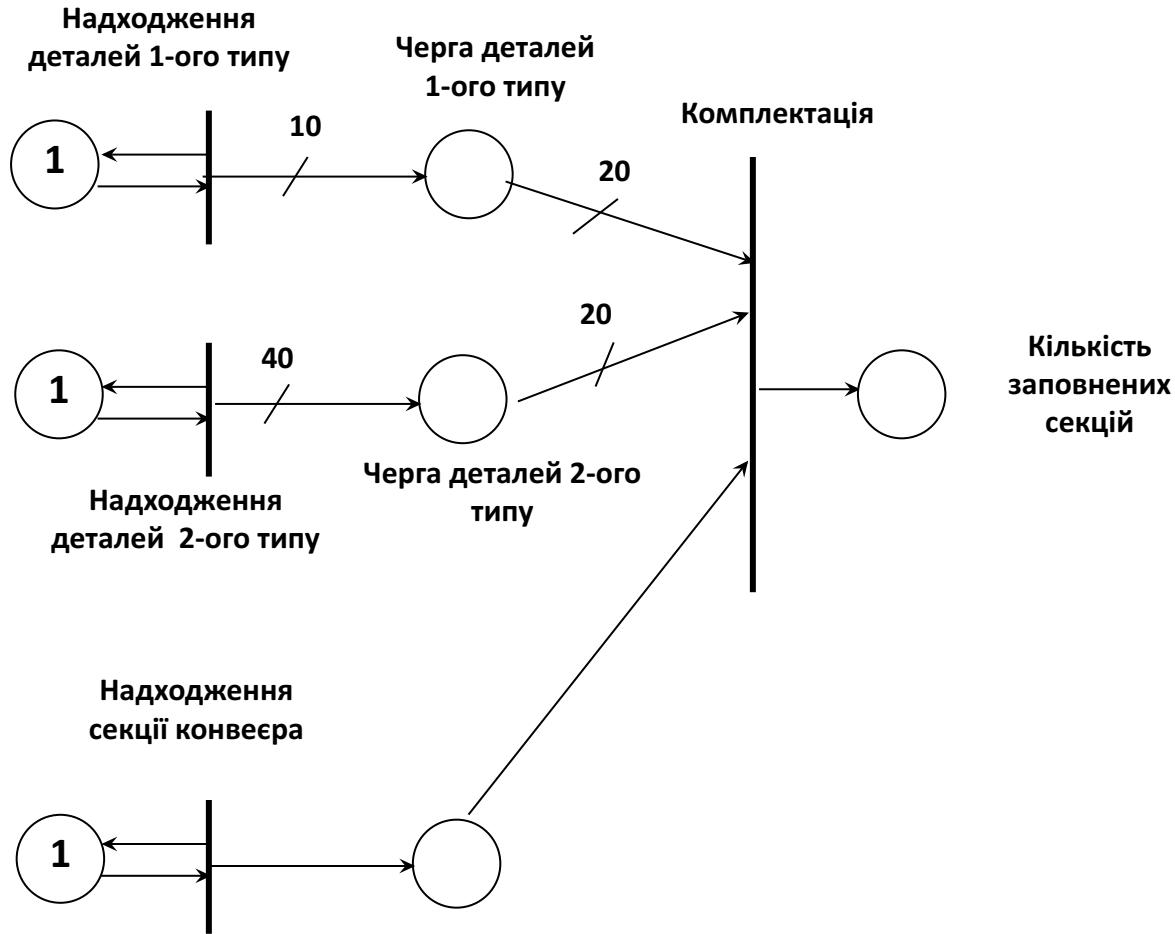
Приклад «Комплектувальний конвеєр»



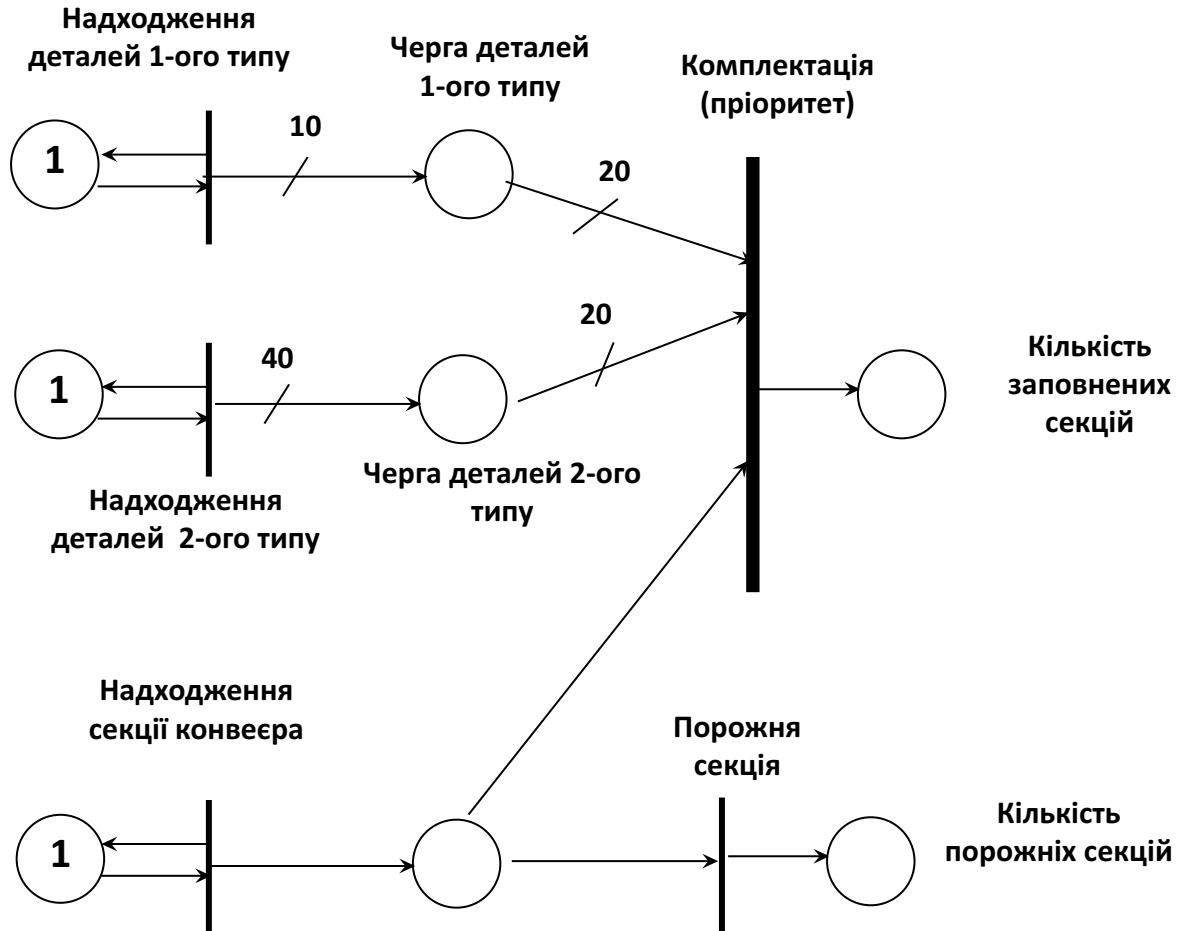
Приклад «Комплектувальний конвеєр»



Приклад «Комплектувальний конвеєр»



Приклад «Комплектувальний конвеєр»



Часові затримки переходів

Перехід	Пріоритет	Часова затримка
Надходження деталей 1-ого типу	0	$t = -10 \cdot \ln \zeta$
Надходження деталей 2-ого типу	0	$t = -40 \cdot \ln \zeta$
Надходження секції конвеєра	0	$t=20$
Комплектація	1	$t=20$
Пропуск секції	0	$t=0$

Визначення вихідних характеристик моделі

Ймовірність пропуску секції

$$P = \frac{N_0}{N_0 + N_1}$$

де N_0 - кількість порожніх секцій, N_1 - кількість заповнених секцій

Середня довжина черги

$$Q = \frac{\sum_{k=1}^n q_k \cdot \Delta t_k}{T_{sim}}$$

де q_k - значення довжини черги, що спостерігалось в інтервалі Δt_k

$$T_{sim} = \sum_{k=1}^n \Delta t_k \text{ - час імітації.}$$

Приклад «Вантажний аеропорт»

Вантажі прибувають для відправлення в аеропорт в контейнерах з інтенсивністю два контейнери за 1 хвилину. Вантажний аеропорт не має фіксованого розкладу, а літаки відправляються, коли вони повністю завантажені.

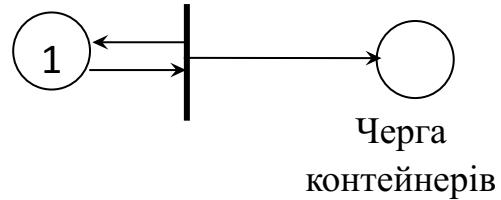
У розпорядженні є два типи літаків для перевезення вантажів: три літаки з вантажністю 80 контейнерів і два літаки з вантажністю 140 контейнерів. Час польоту кожного літака туди й назад розподілено нормально з математичним сподіванням 3 години, середньоквадратичним відхиленням 1 година, мінімумом 2 години, максимумом 4 години.

Управлючий аеропортом намагається як найчастіше використовувати літаки меншої вантажності. Літаки, що піднімають 140 контейнерів, використовуються тільки тоді, коли інших немає в наявності. Припускається, що часом вантаження можна знехтувати.

Метою моделювання є визначення 1) середнього часу очікування контейнерів із вантажами, 2) середнього завантаження літаків обох типів.

Приклад «Вантажний аеропорт»

Надходження
контейнерів

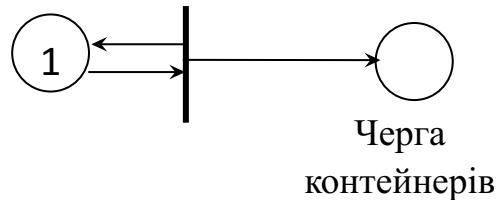


Приклад «Вантажний аеропорт»

Три маленьких літака

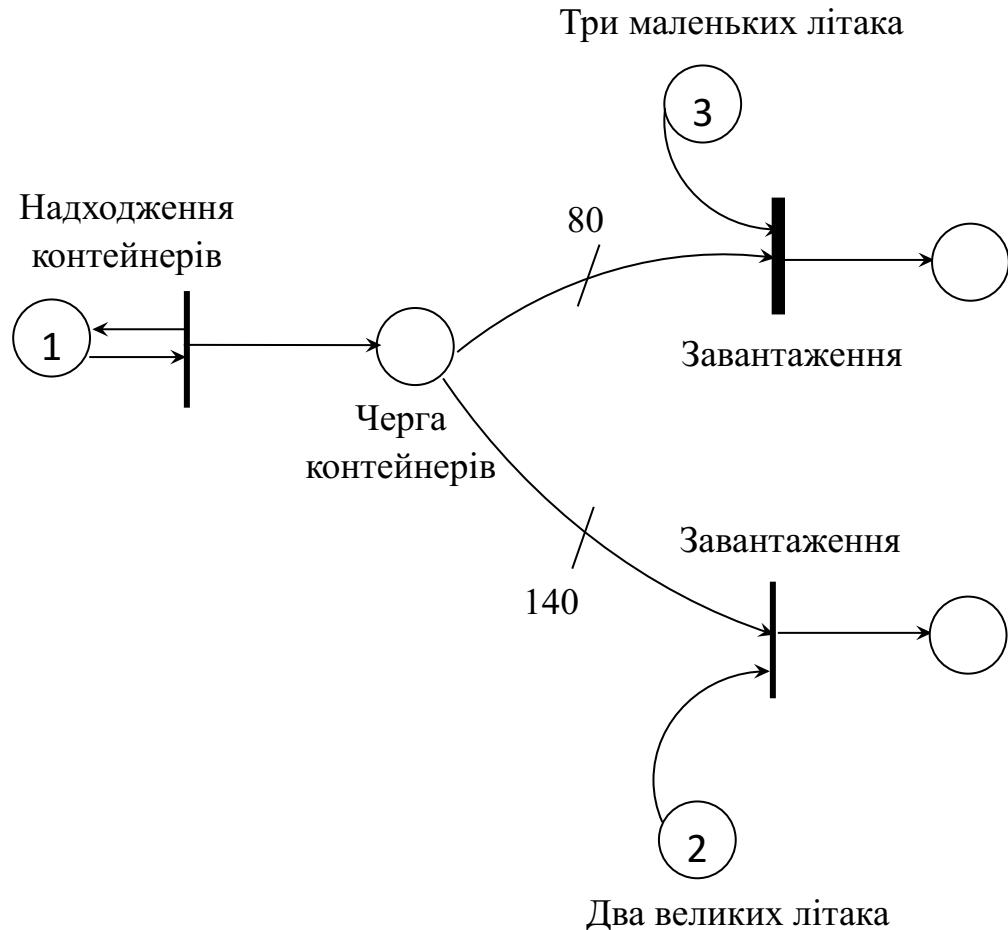


Надходження
контейнерів

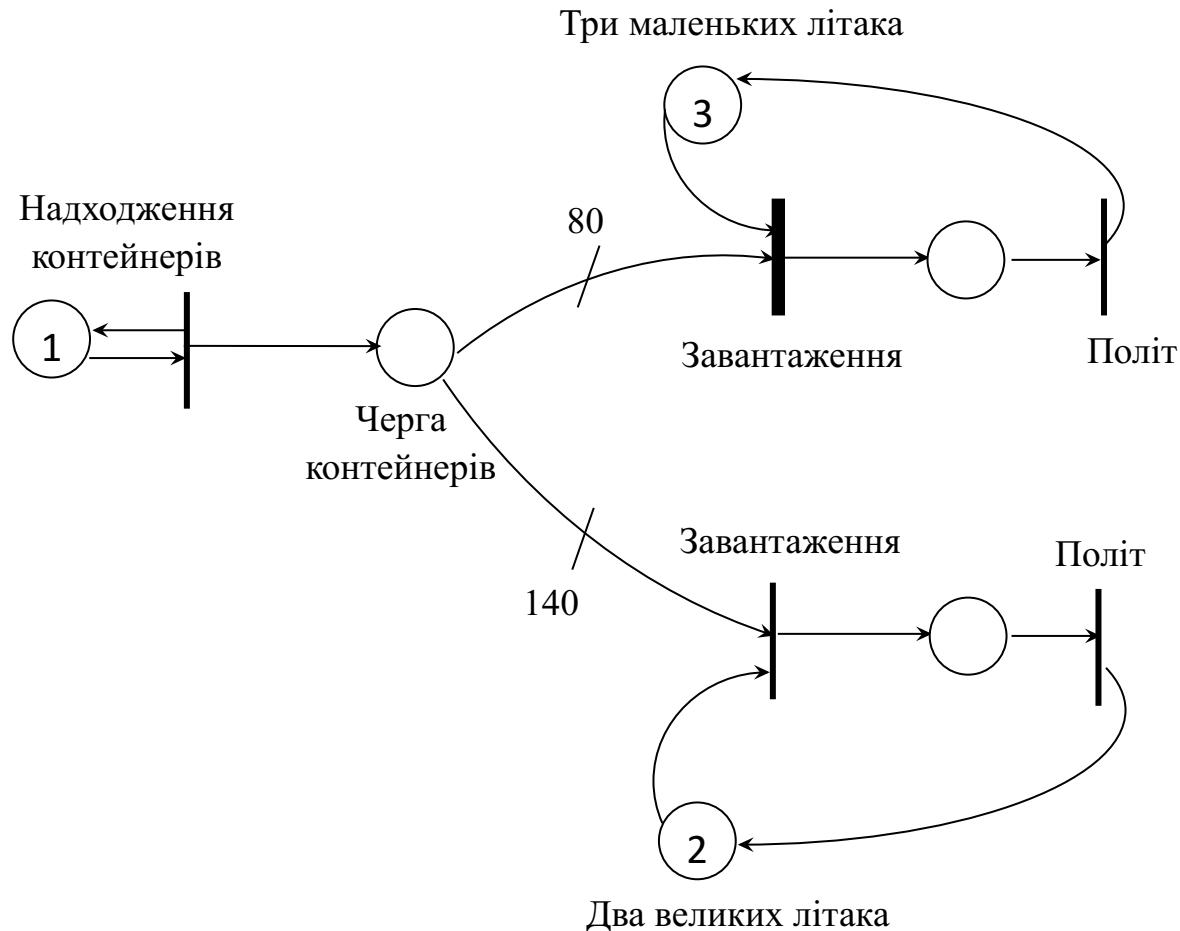


Два великих літака

Приклад «Вантажний аеропорт»



Приклад «Вантажний аеропорт»



Параметри переходів

Перехід	Пріоритет	Часова затримка
Надходження контейнерів	0	$t = 0,5$
Завантаження маленького літака	1	$t = 0$
Завантаження великого літака	0	$t = 0$
Політ літака	0	$t = \begin{cases} 120 \text{ хв, якщо } r < 120 \\ r \text{ хв, якщо } 120 \leq r \leq 240 \\ 240 \text{ хв, якщо } r > 240 \end{cases}$ $r = \left(\sum_{i=1}^{12} \zeta_i - 6 \right) + 180.$

Визначення вихідних характеристик моделі

Середнє завантаження маленьких літаків

$$L = 3 - \frac{\sum_{k=1}^n M_{small} \cdot \Delta t_k}{T_{sim}}$$

де M_{small} - значення маркірування позиції «Три маленьких літака», що спостерігалось протягом інтервалу Δt_k ,

Середнє завантаження великих літаків

$$L = 2 - \frac{\sum_{k=1}^n M_{big} \cdot \Delta t_k}{T_{sim}}$$

де M_{big} - значення маркірування позиції «Три маленьких літака», що спостерігалось протягом інтервалу Δt_k ,

$$T_{sim} = \sum_{k=1}^n \Delta t_k - \text{час імітації.}$$

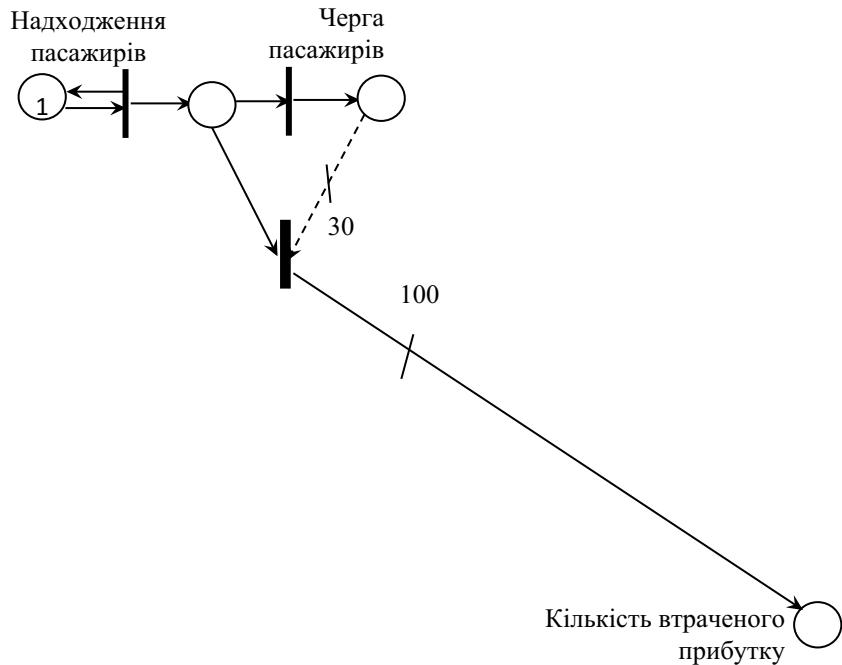
Приклад «Маршрутки»

На маршруті приміського сполучення працюють два мікроавтобуси (А і В), кожний з яких має 25 місць. Мікроавтобус А користується більшою популярністю, ніж автобус В, оскільки водій мікроавтобуса А їздить акуратніше і швидше. Тому пасажир, який підійшов до зупинки, сідає в мікроавтобус В тільки у випадку, коли автобуса А немає. Мікроавтобус відправляється на маршрут, якщо всі місця в ньому зайняті. Пасажири підходять до зупинки через $0,5\pm0,2$ хвилин і, якщо немає мікроавтобусів, утворюють чергу. Якщо черга більша, ніж 30 осіб, то пасажир обирає інший маршрут. Для спрощення моделі вважаємо, що всі пасажири їдуть до кінця маршруту. На проходження маршруту мікроавтобус А витрачає 80 ± 20 хвилин, а мікроавтобус В – 120 ± 30 хвилин,. Після висадки пасажирів автобус відправляється у зворотному напрямку таким же чином. Посадка та висадка пасажирів триває в середньому 5 ± 1 хвилини. Мікроавтобуси працюють 10 годин на добу.

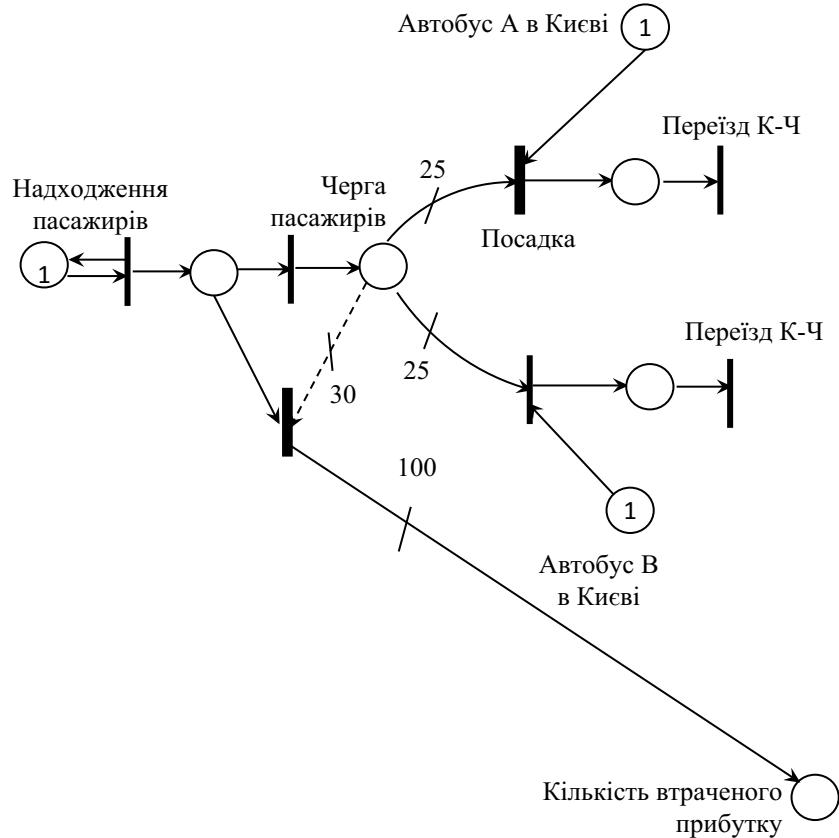
Плата за проїзд складає 100 гривень. Автопідприємство стільки ж втрачає (недоотримує), якщо пасажир, прийшовши на зупинку, не стає в чергу і обирає інший маршрут.

Метою моделювання є визначення часу очікування пасажира у черзі та виручки автопідприємства за день від маршруту.

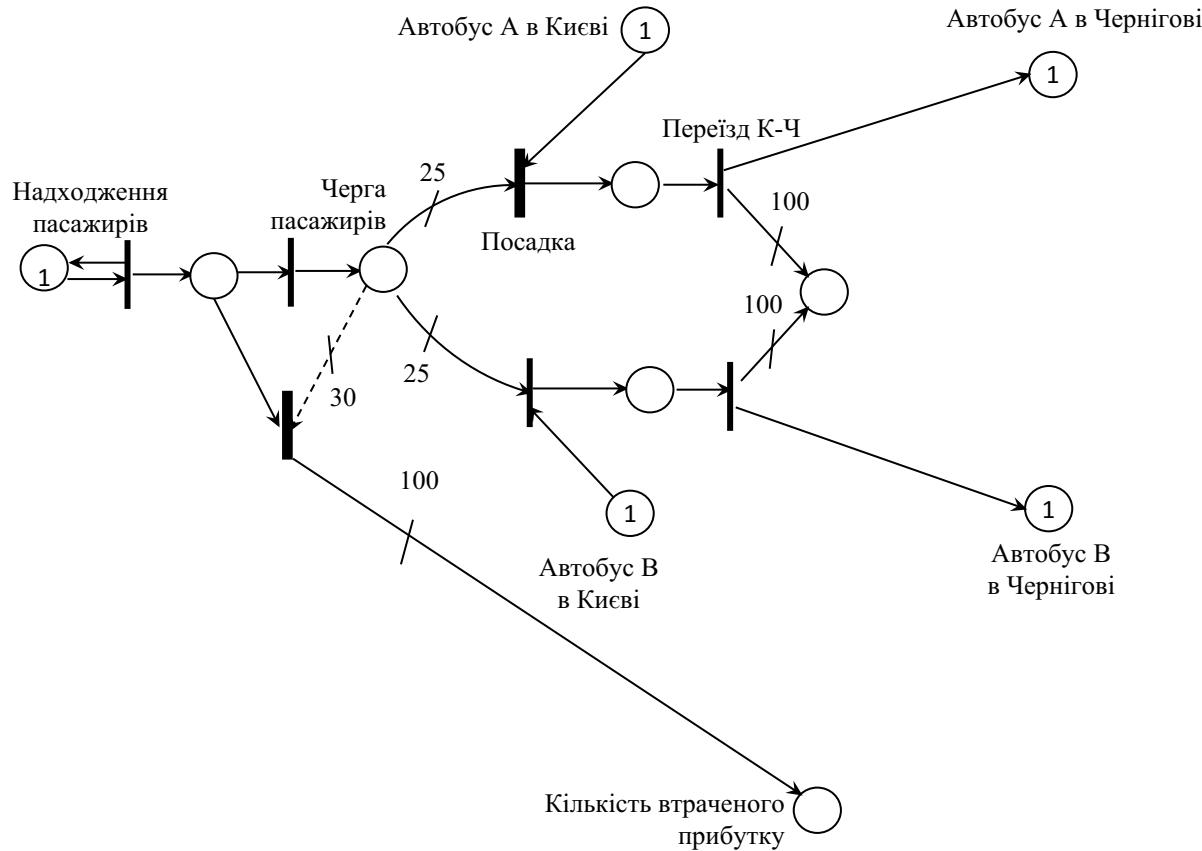
Приклад «Маршрутки»



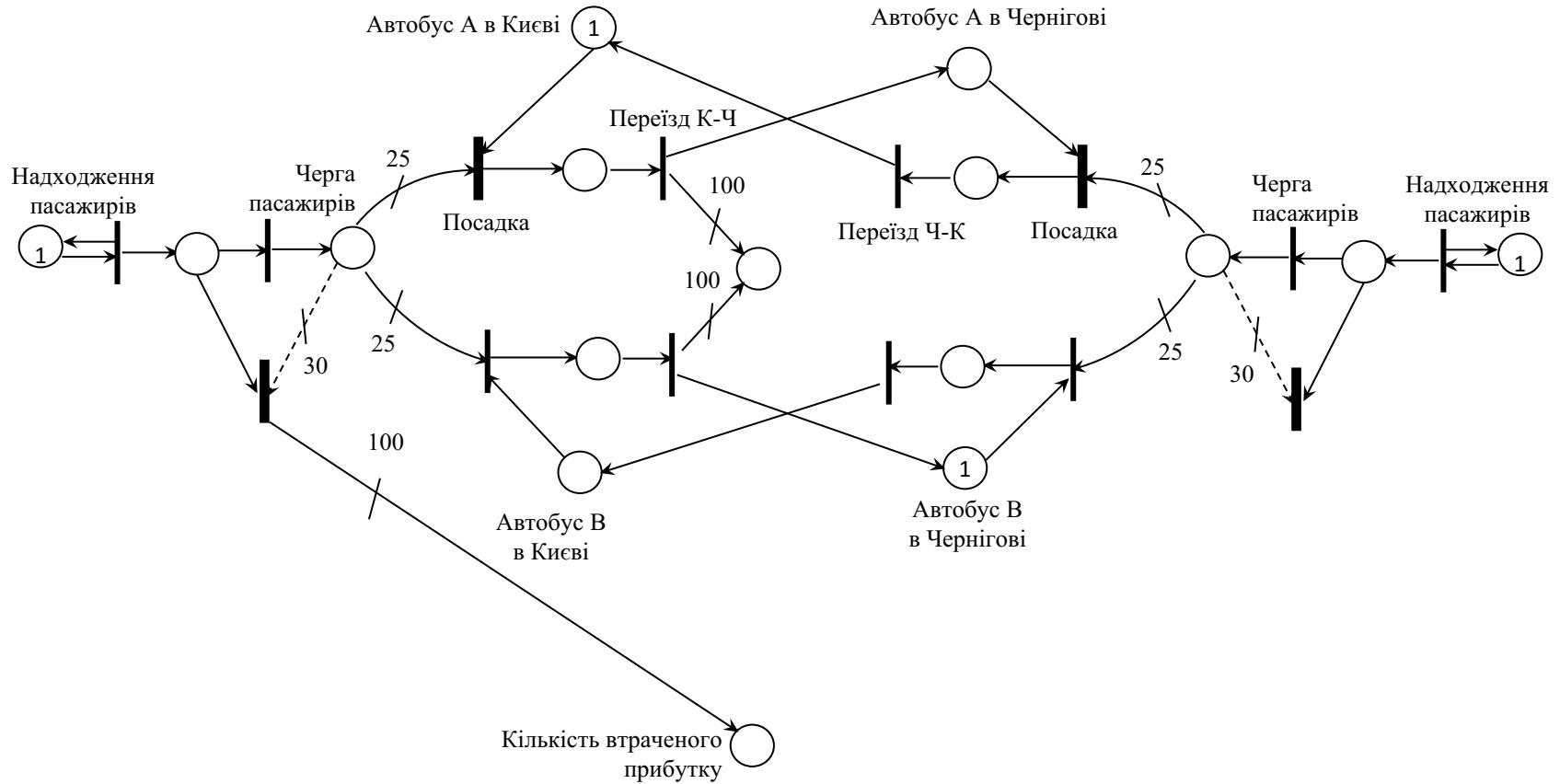
Приклад «Маршрутки»



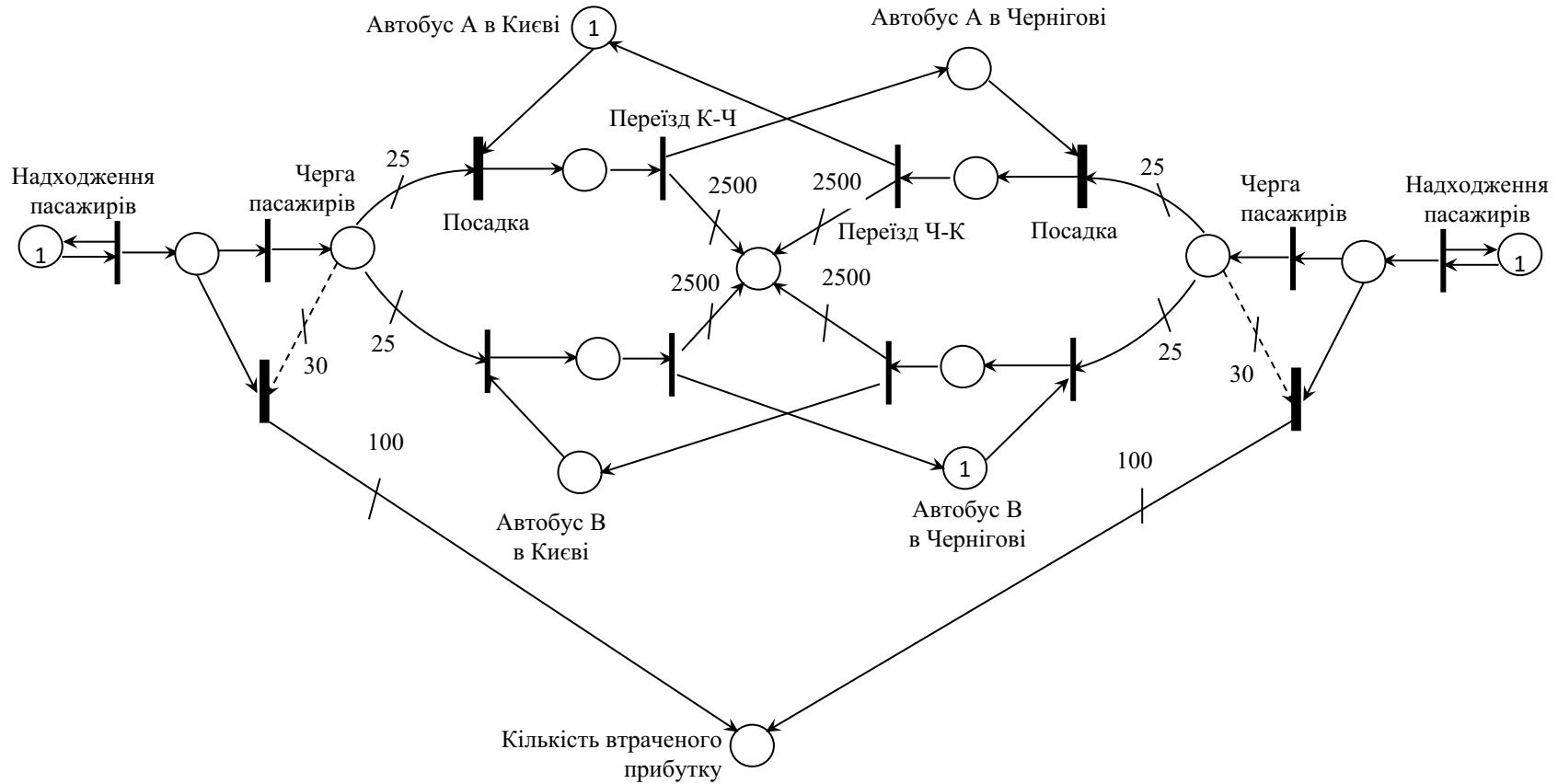
Приклад «Маршрутки»



Приклад «Маршрутки»



Приклад «Маршрутки»



Параметри переходів

Визначення вихідних характеристик моделі

Середній час очікування пасажира у черзі

$$W = \frac{\sum_{k=1}^n M_{queue} \cdot \Delta t_k}{N}$$

де M_{queue} - значення маркірування позиції «Черга пасажирів», що спостерігалось протягом інтервалу Δt_k ,

Вигучка кількості пасажирів за деякий проміжок часу.

$$M_{profit} - M_{loss}$$

де M_{profit} - значення маркірування позиції «Прибуток» наприкінці імітації,

M_{loss} - значення маркірування позиції «Кількість втраченого прибутку» наприкінці імітації.

Приклад «Оптовий магазин»

Фірма має 6 точок роздрібного продажу. Попит на товари у цих точках має розподіл Пуассона з математичним сподіванням 10 одиниць товару в день.

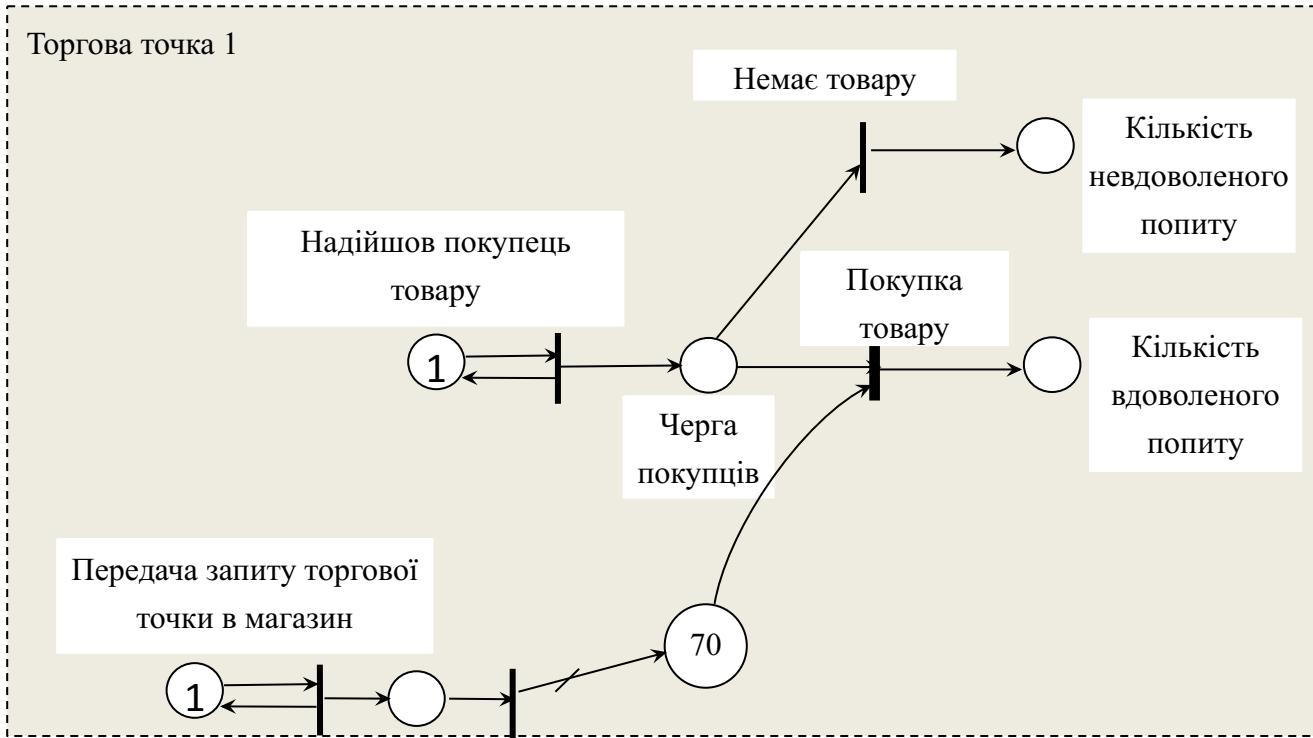
Торгові точки обслуговуються оптовим магазином. На передачу запиту торгової точки в магазин витрачається 1 день. Товари за запитом надходять з оптового магазина в торгову точку в середньому через 5 днів після одержання запиту. Ця величина має нормальній розподіл з дисперсією 1.

Оптовий магазин кожні 14 днів розміщує замовлення на фабриці. Час, протягом якого магазин одержує вантаж з фабрики, розподілено нормально з очікуванням 90 днів, середньоквадратичним відхиленням 10 днів, мінімумом 60 днів, максимумом 120 днів.

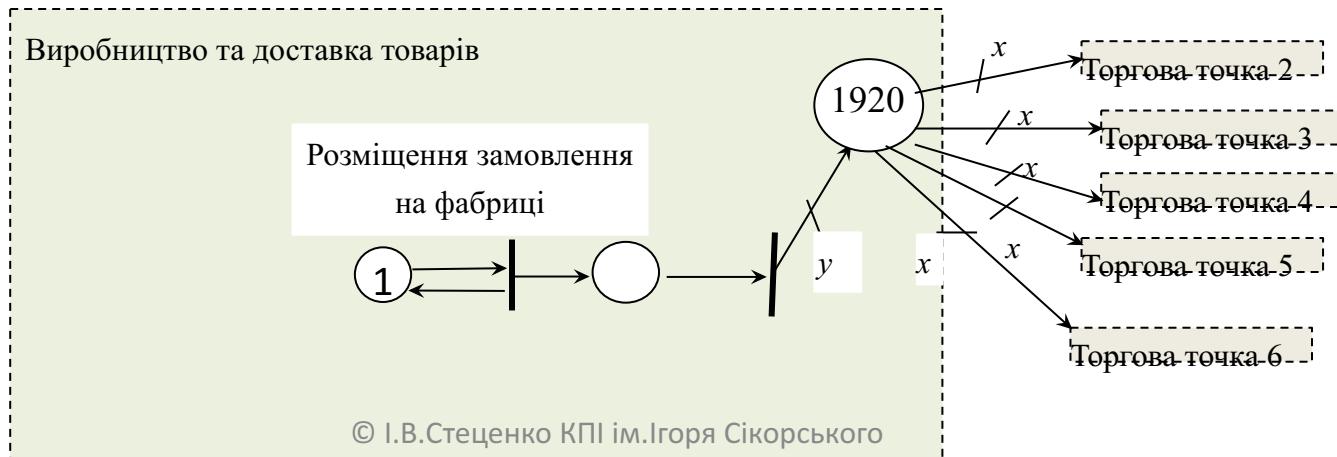
Метою моделювання є визначення таких величин: 1) рівень запасу в оптовому магазині, 2) ймовірність невдоволеного запиту торгової точки.

Приклад «Оптовий магазин»

Торгова точка 1

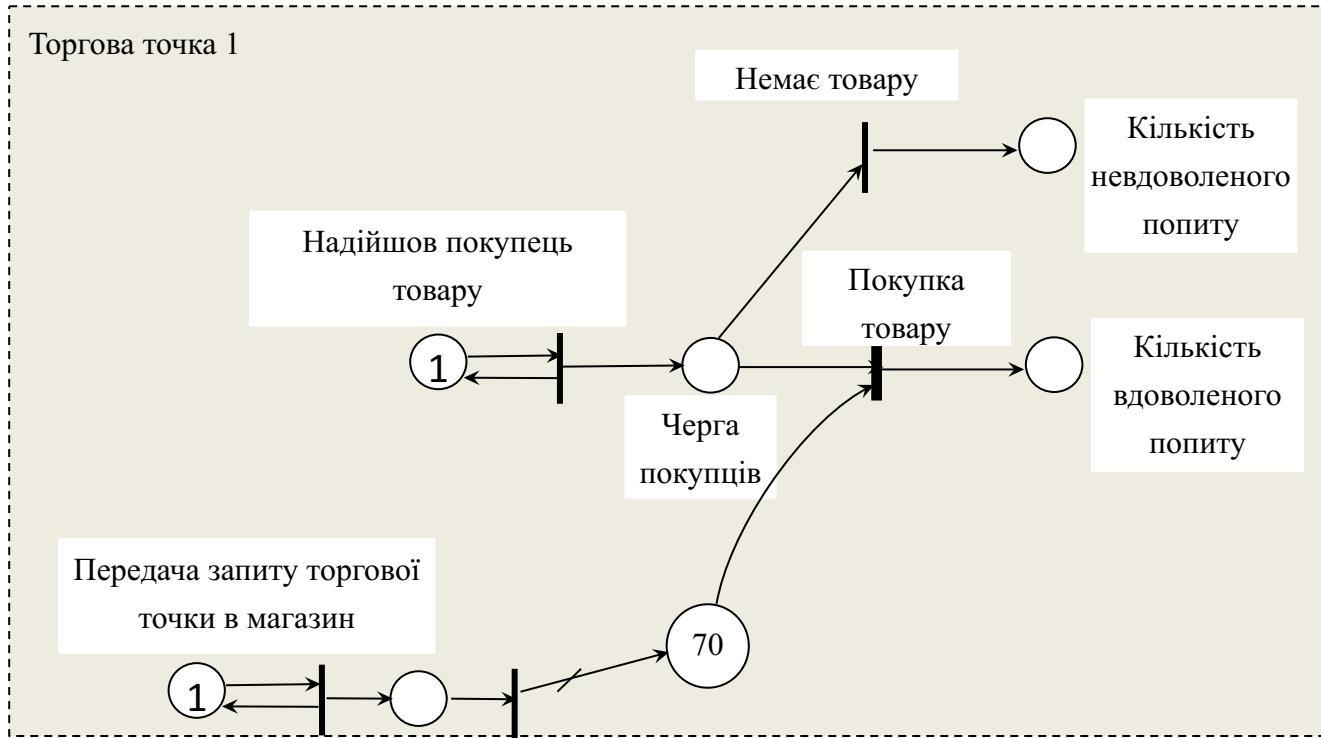


Приклад «Оптовий магазин»

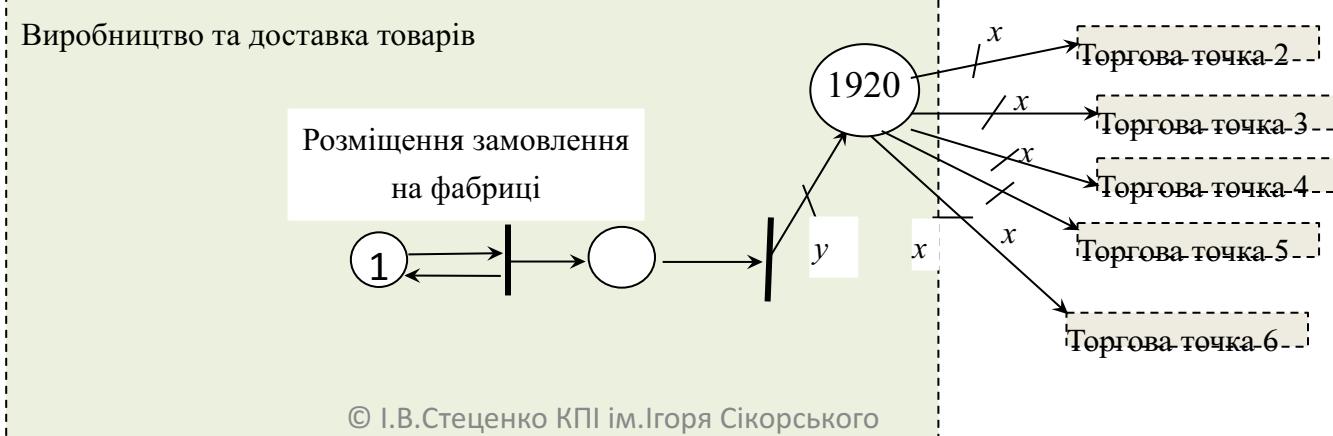


Приклад «Оптовий магазин»

Торгова точка 1

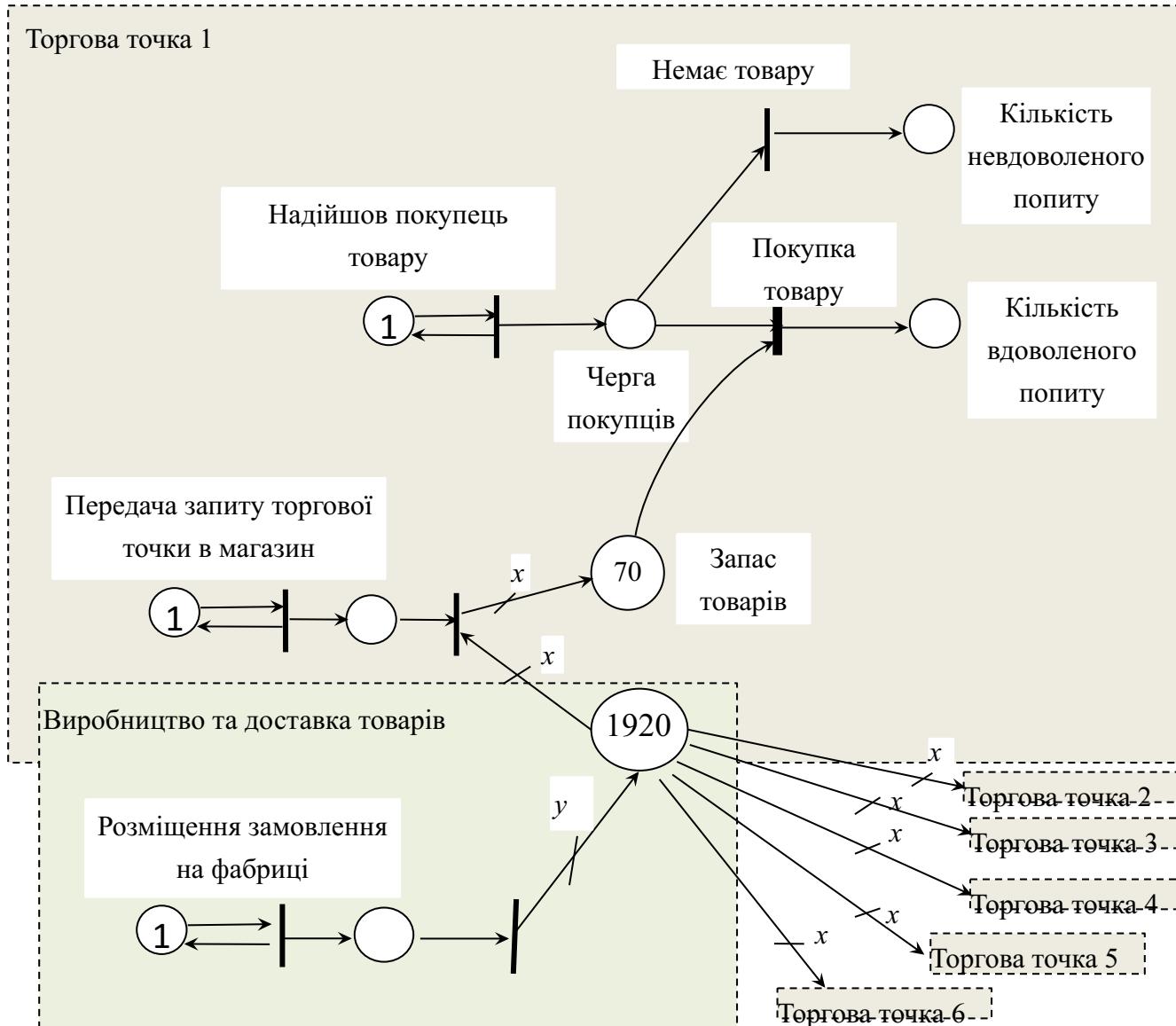


Виробництво та доставка товарів



Приклад «Оптовий магазин»

Торгова точка 1



Параметри переходів

Визначення вихідних характеристик моделі

Середній запас

$$\sum_{k=1}^n y_k \cdot \Delta t_k$$

де Y – середнє значення ~~запасу товарів~~,

y_k - значення запасу, що спостерігалось в інтервалі Δt_k

$$T_{sim} = \sum_{k=1}^n \Delta t_k$$

- час імітації.

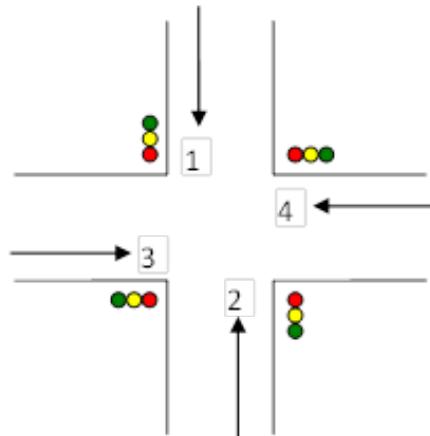
Ймовірність невдоволеного попиту

$$P = \frac{N_0}{N_0 + N_1}$$

де N_0 - кількість невдоволеного попиту на товари,

N_1 - кількість вдоволеного попиту на товари,

Приклад «Регульоване перехрестя»

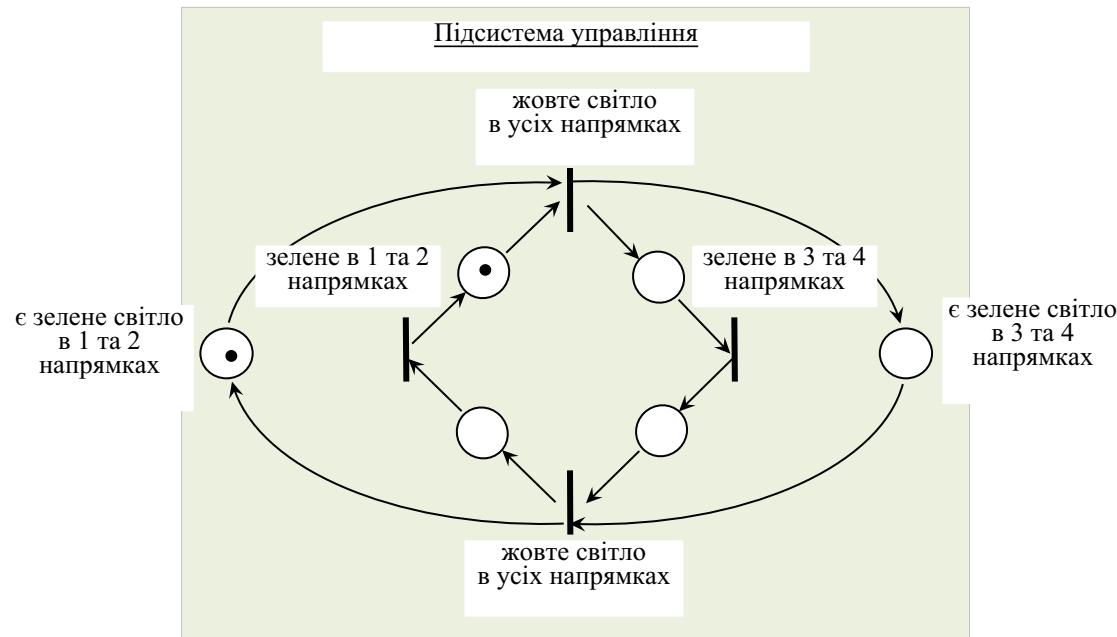


Напрямок руху	Фаза світлофора							
	I		II		III		IV	
	світло	час	світло	час	світло	час	світло	час
1	зелений	20	жовтий	10	червоний	30	жовтий	10
2	зелений	20	жовтий	10	червоний	30	жовтий	10
3	червоний	20	жовтий	10	зелений	30	жовтий	10
4	червоний	20	жовтий	10	зелений	30	жовтий	10

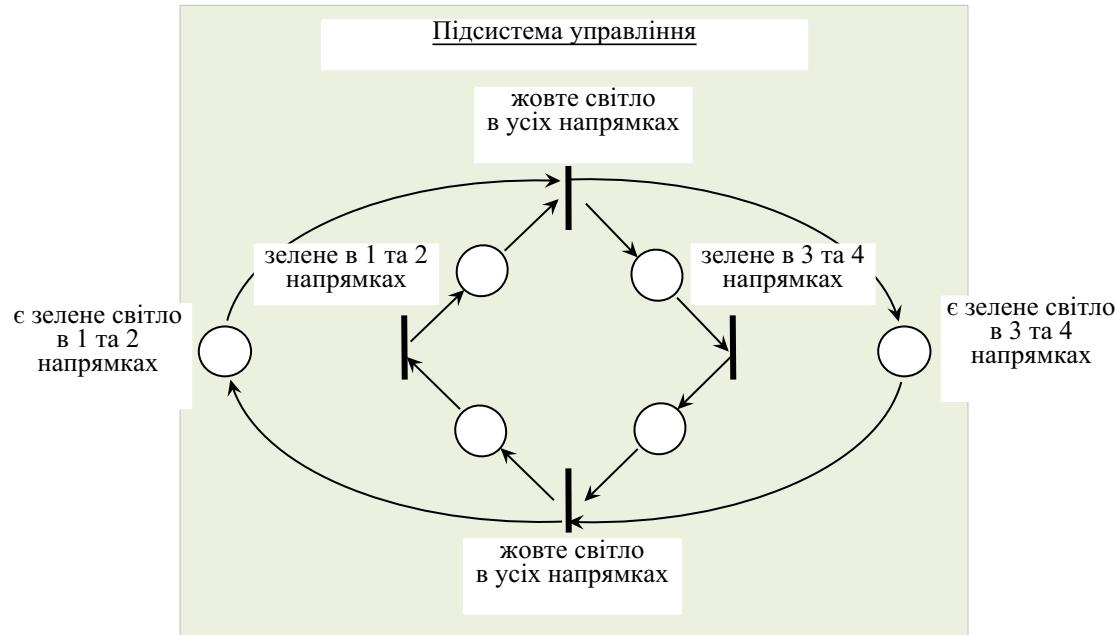
Регулювання транспортним рухом на перехресті здійснюється світлофорами так, що протягом певного часу горить зелене світло в першому та другому напрямках руху, а в третьому та четвертому напрямках горить червоне світло. Потім горить жовте світло в усіх напрямках протягом часу, що дозволяє автомобілям, які виїхали на перехрестях, залишити його до початку руху автомобілів з іншого напрямку. Далі вмикається зелене світло в третьому та четвертому напрямках руху, а в першому та другому напрямках горить червоне світло. Потім знову вмикається жовте світло в усіх напрямках і так далі. Тривалості горіння зеленого та червоного світла задаються так, як у таблиці.

Метою моделювання є визначення параметрів управління, при яких максимум середньої кількості автомобілів, що очікують переїзду в різних напрямках, досягає свого найменше значення: $L = \max \{L_1, L_2, L_3, L_4\} \rightarrow \min$

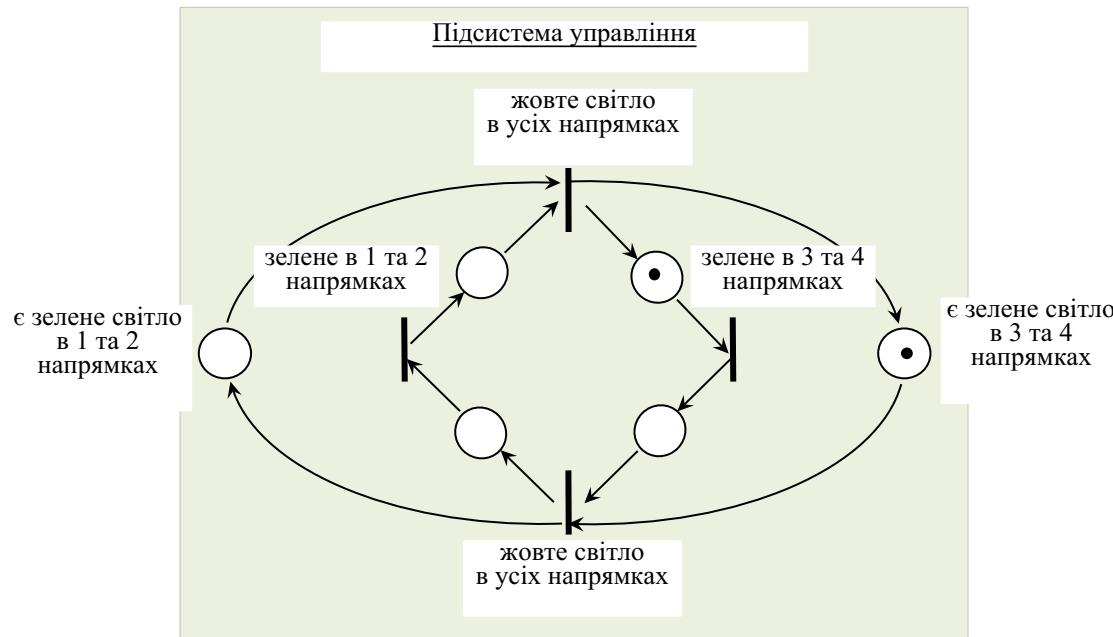
Приклад «Регульоване перехрестя»



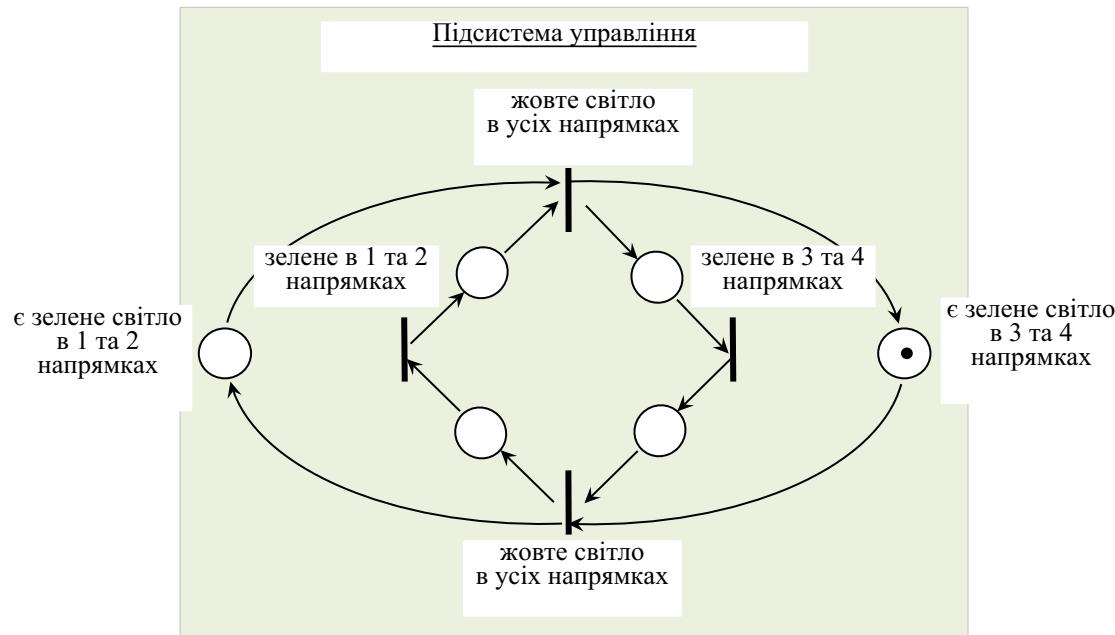
Приклад «Регульоване перехрестя»



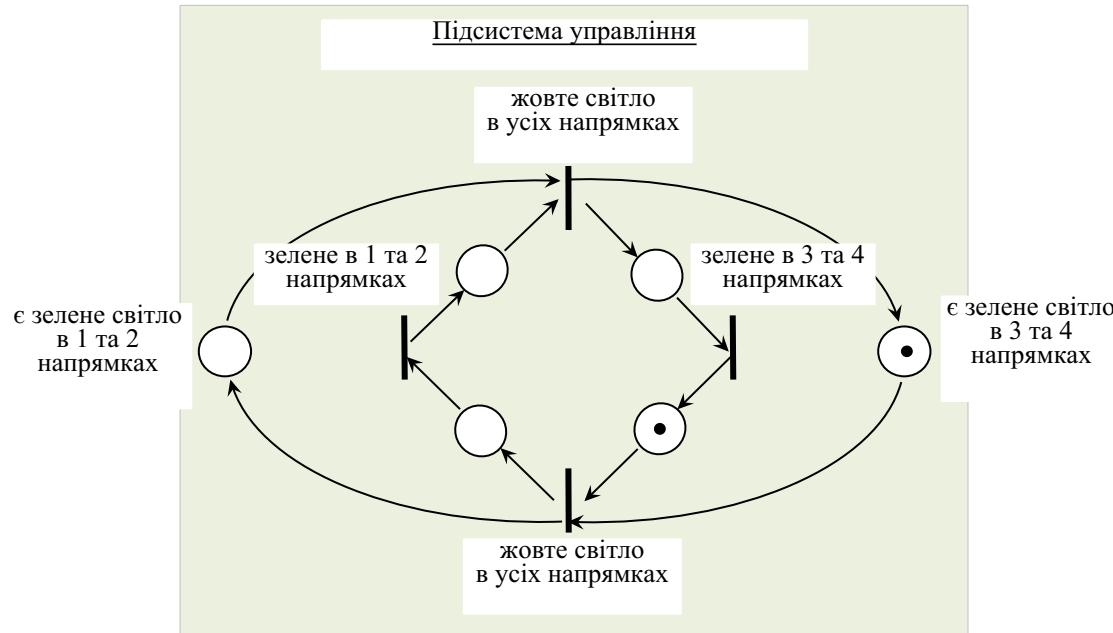
Приклад «Регульоване перехрестя»



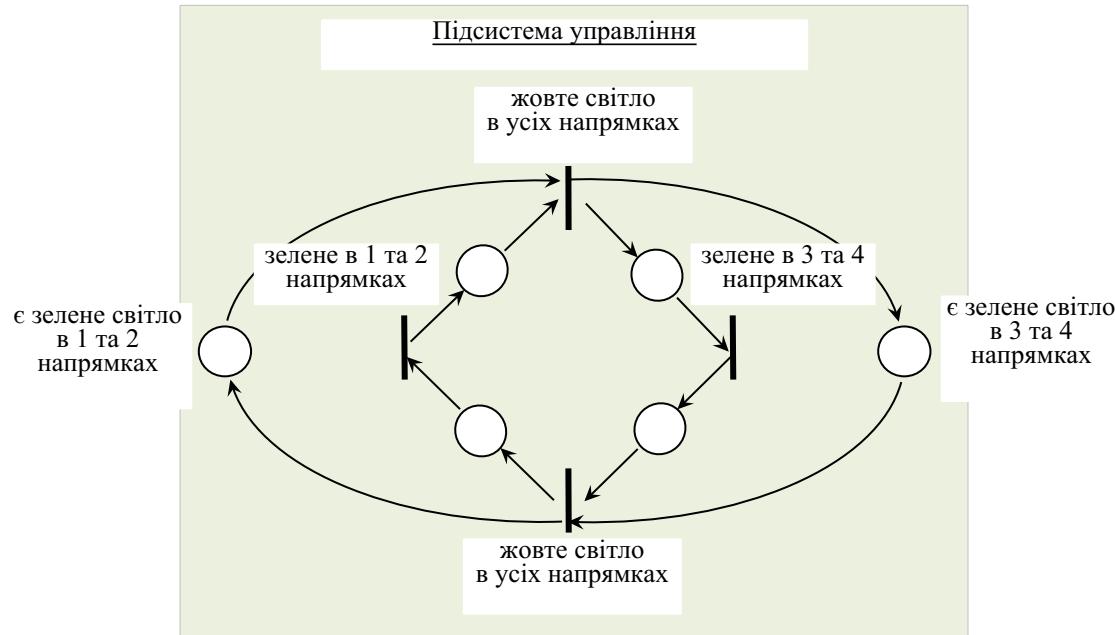
Приклад «Регульоване перехрестя»



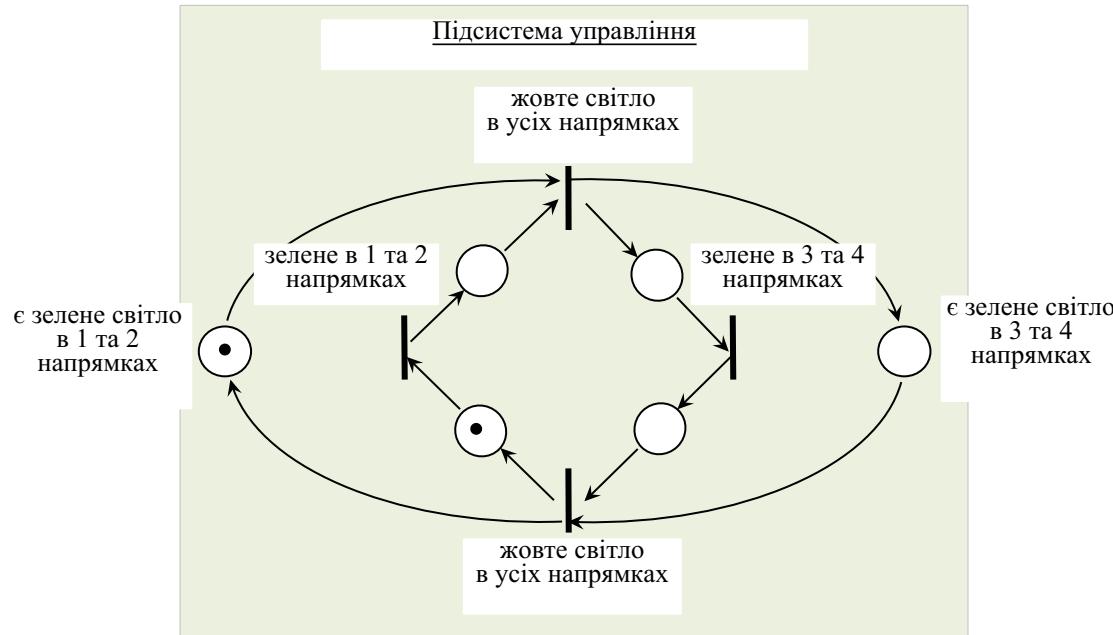
Приклад «Регульоване перехрестя»



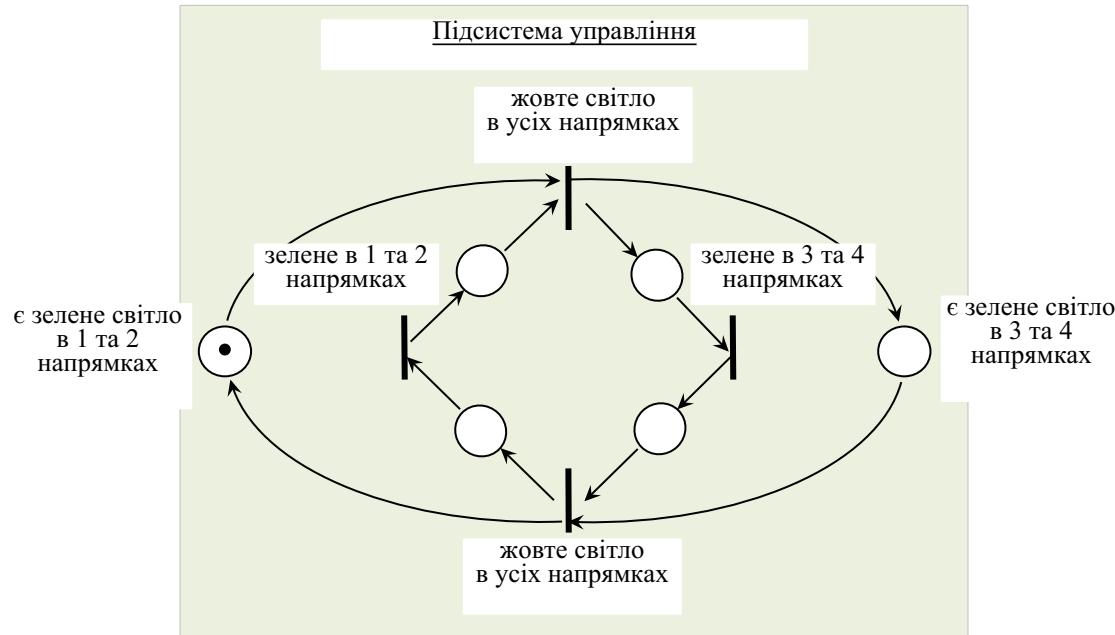
Приклад «Регульоване перехрестя»



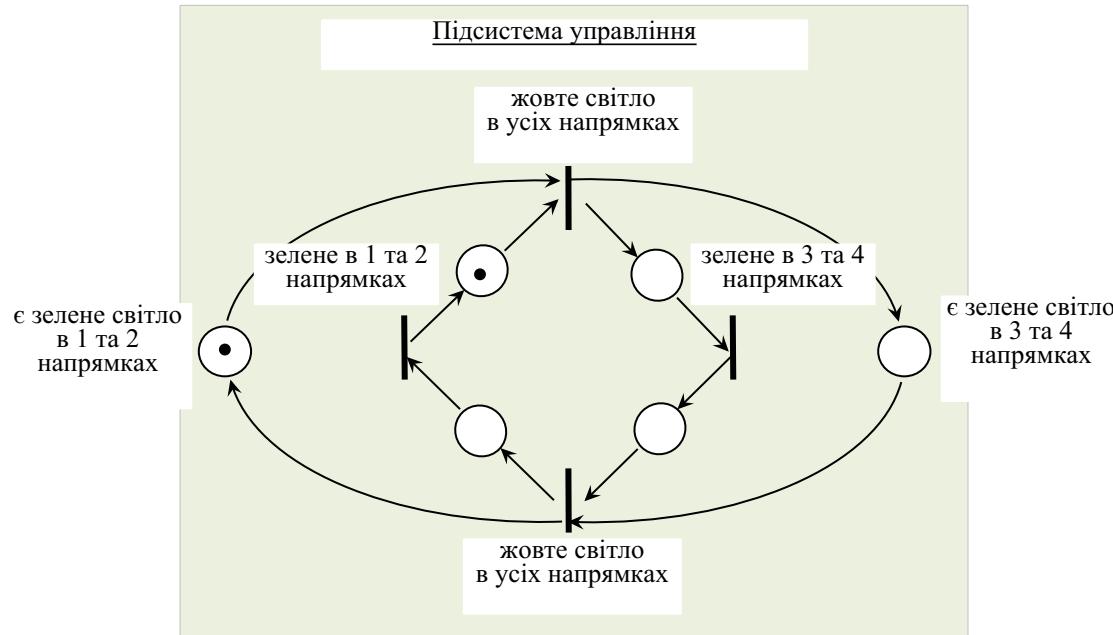
Приклад «Регульоване перехрестя»



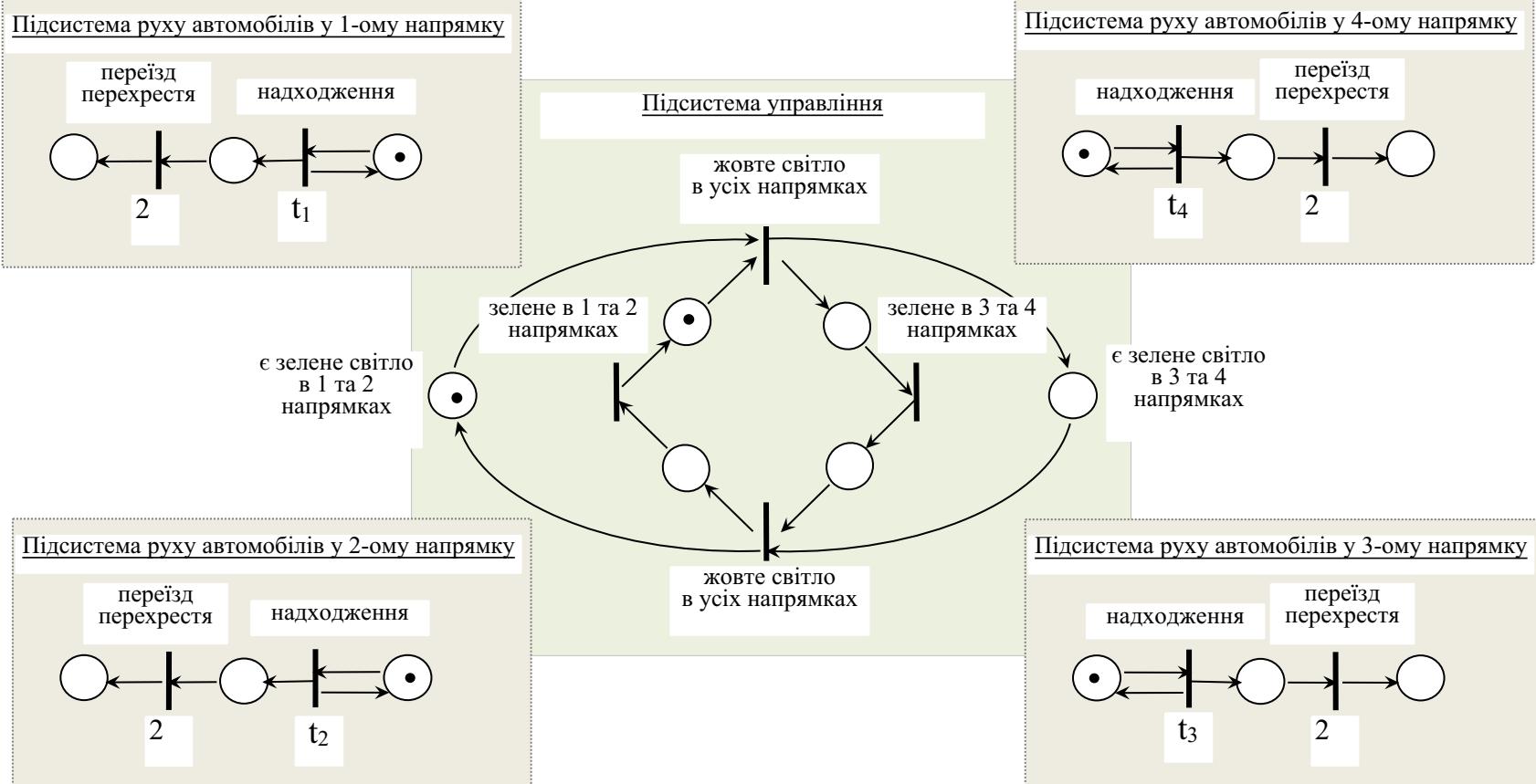
Приклад «Регульоване перехрестя»



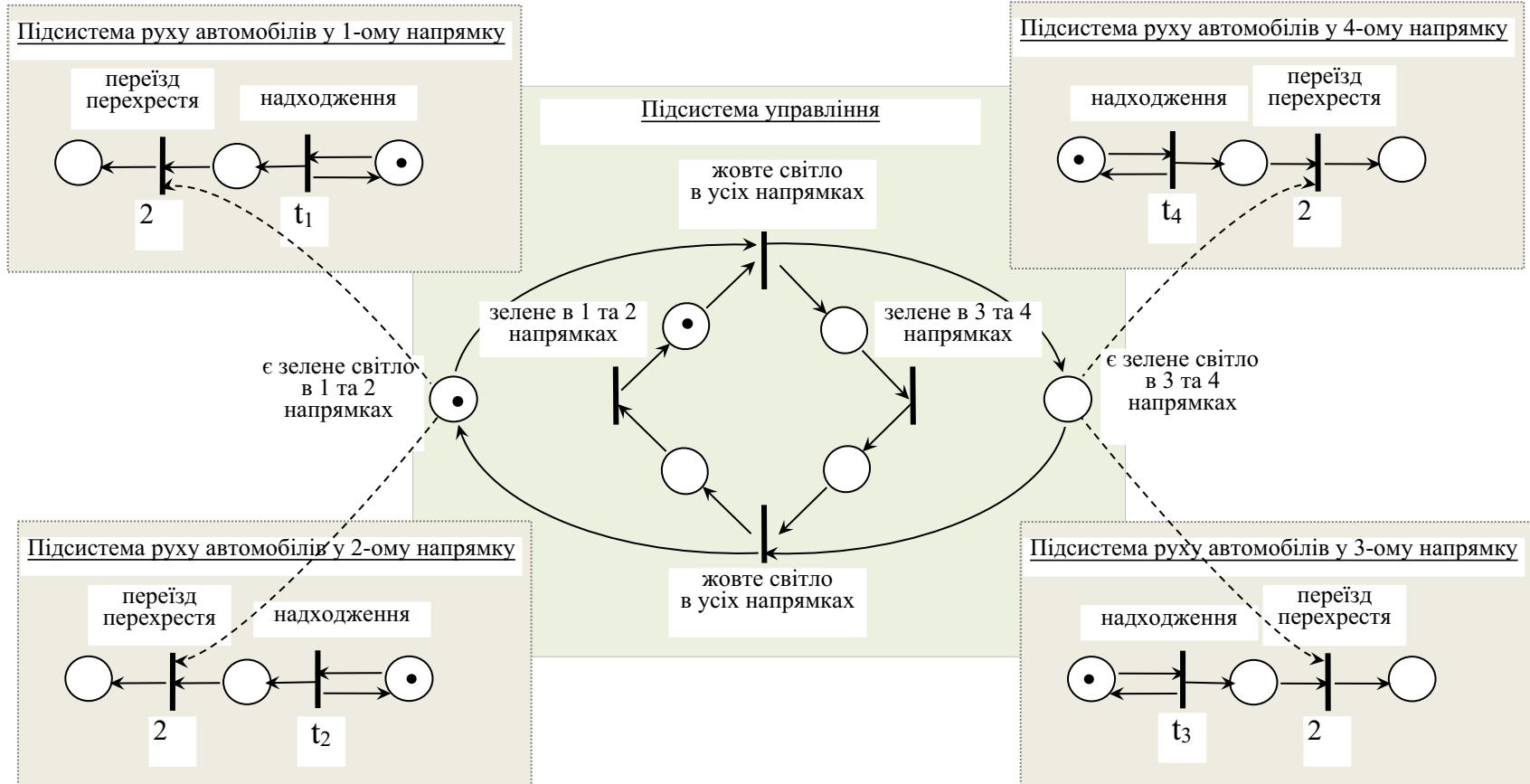
Приклад «Регульоване перехрестя»



Приклад «Регульоване перехрестя»



Приклад «Регульоване перехрестя»



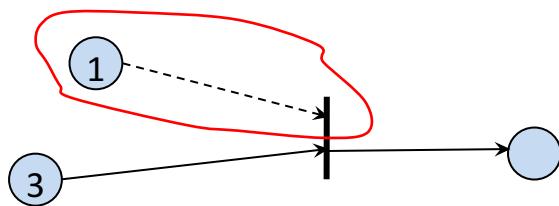
Зауваження 1: Інформаційною може бути тільки **вхідна** дуга

Зауваження 2. Перехід, який має інформаційну дугу, **обов'язково** повинен мати звичайну (неінформаційну) дугу

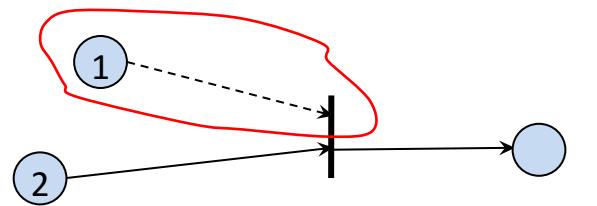
Інформаційна дуга

Інформаційна дуга - це дуга, вздовж якої маркери під час входу в перехід не видаляються.

Наявність маркерів у вхідній позиції, яка з'єднана з переходом інформаційною дугою, перевіряється, але при вході маркерів в перехід маркери з такої позиції не віднімаються.



До входу маркерів



Після входу маркерів

Зauważення 1. Інформаційною може бути тільки **вхідна** дуга

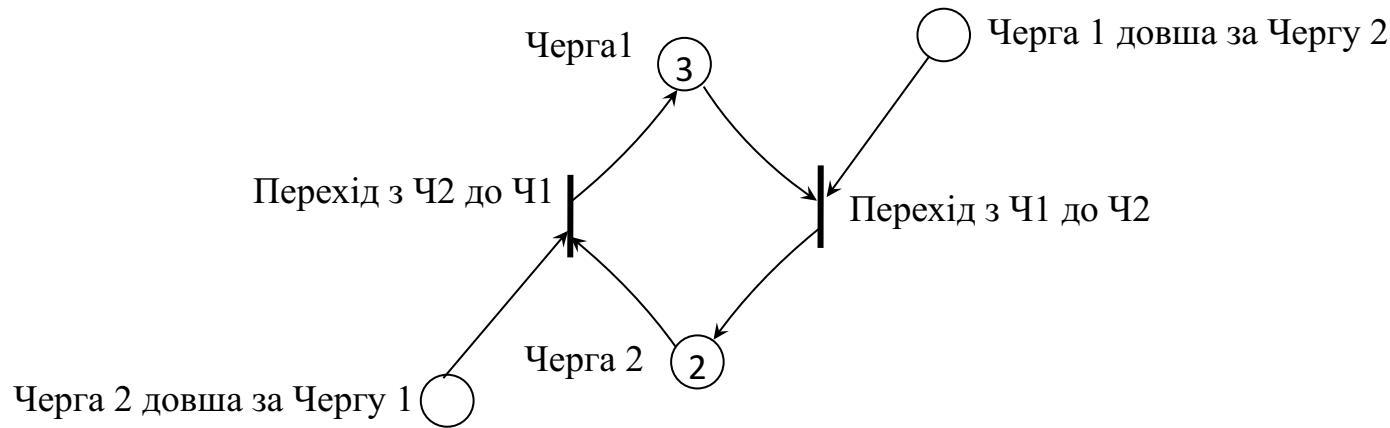
Зauważення 2. Перехід, який має інформаційну дугу, **обов'язково** повинен мати звичайну (неінформаційну) дугу

Приклад «Управління чергами»

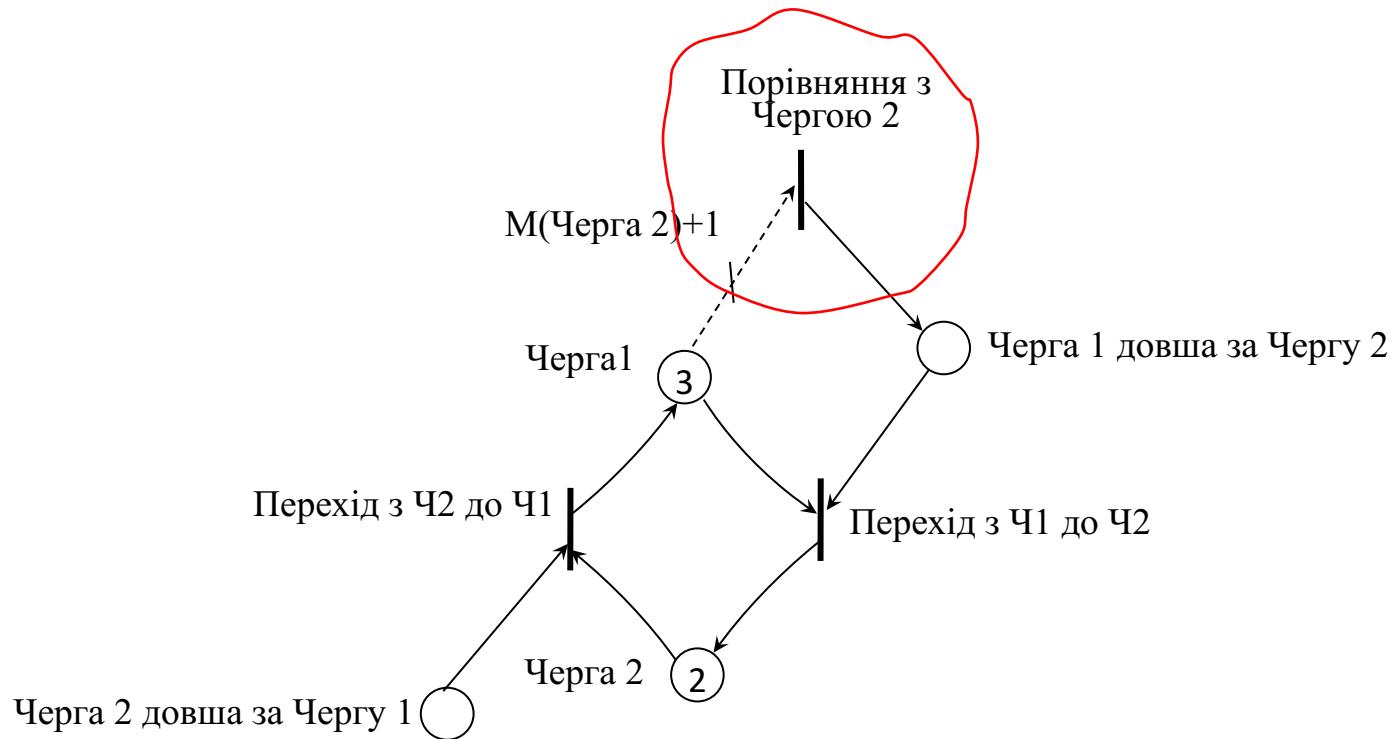
Черга1
③

Черга 2
②

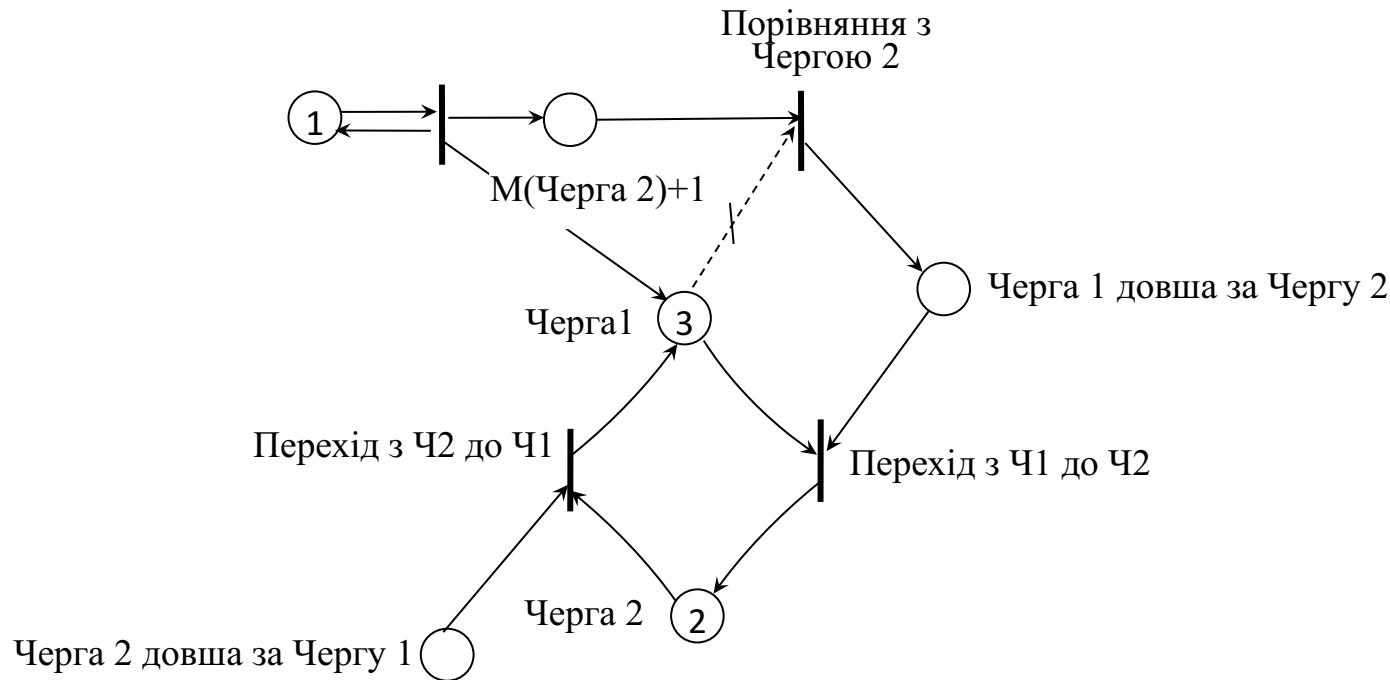
Приклад «Управління чергами»



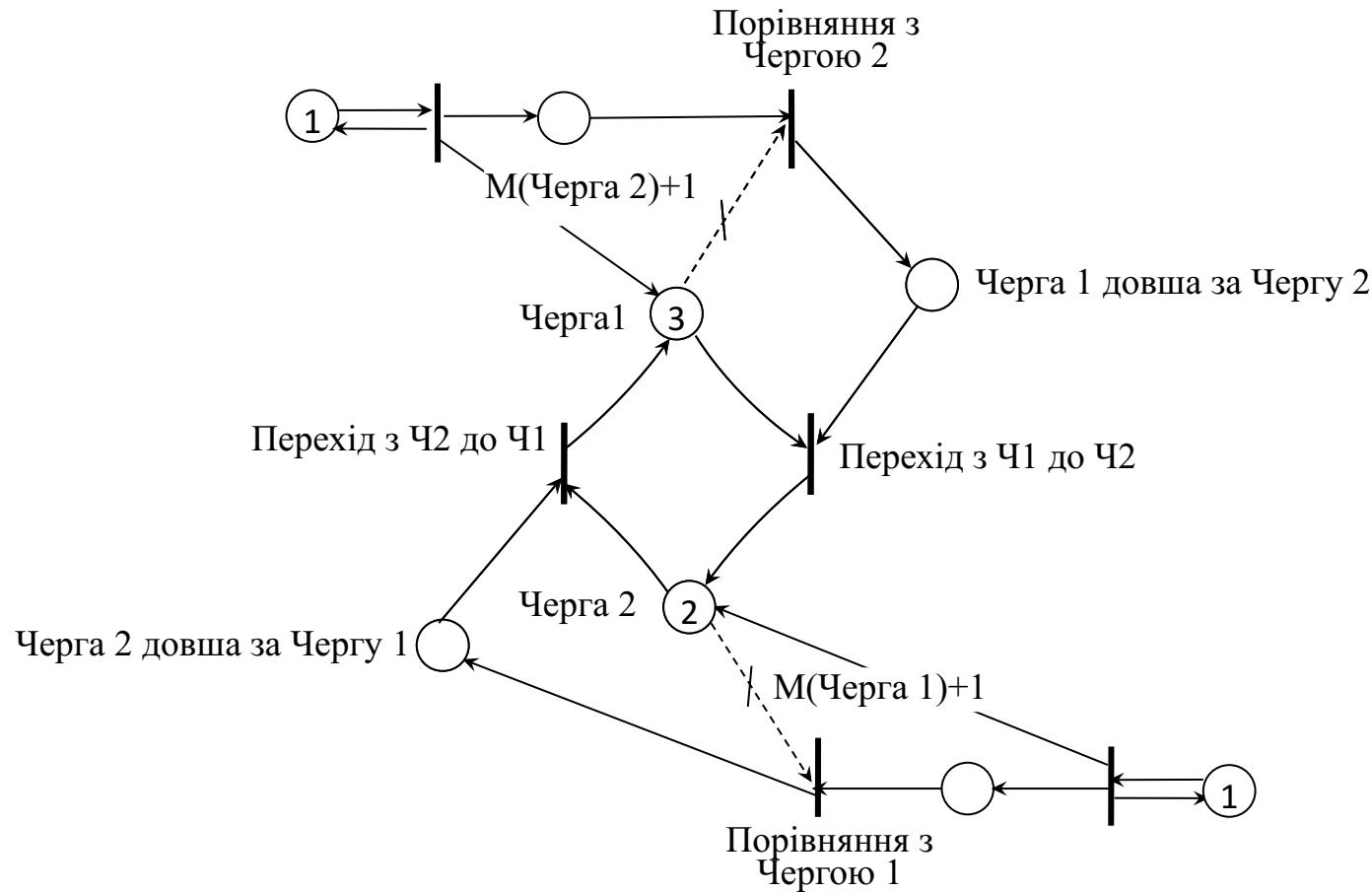
Приклад «Управління чергами»



Приклад «Управління чергами»



Приклад «Управління чергами»

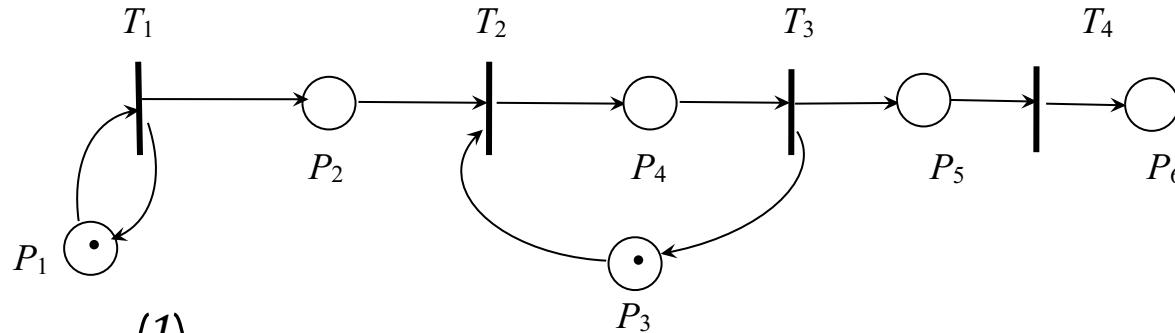


Математична теорія класичних мереж Петрі

Базові публікації

Murata T. Petri Nets: Properties, Analysis and Applications. // Proceedings of IEEE. – 1989.
- Vol.77, No.4. - P.541-580.

Матричний опис базової мережі Петрі



Маркірування $M = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Кількість маркерів в P_3

Кількість зв'язків з T_1 в P_2

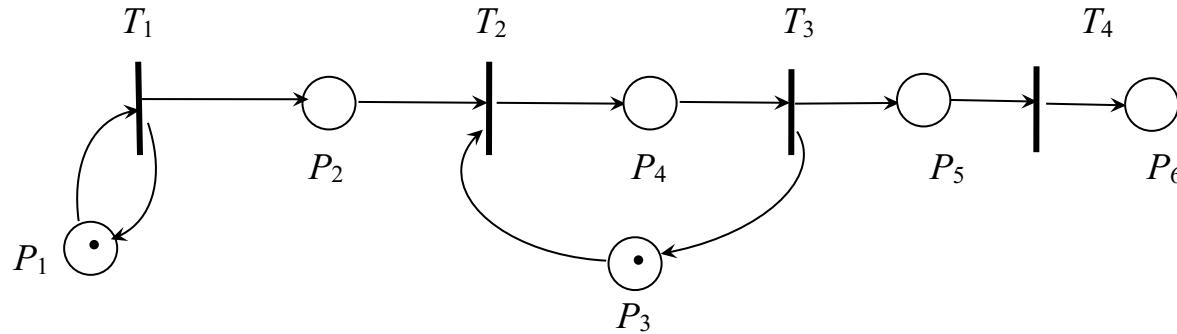
Кількість зв'язків з P_3 в T_2

Кількість зв'язків з P_5 в T_4

$$\text{Матриця входів } a^- = \begin{pmatrix} 1000 \\ 0100 \\ 0100 \\ 0010 \\ 0001 \\ 0000 \end{pmatrix}$$

$$\text{Матриця виходів } a^+ = \begin{pmatrix} 1000 \\ 1000 \\ 0010 \\ 0100 \\ 0010 \\ 0001 \end{pmatrix}$$

Матричний опис базової мережі Петрі



$$\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{T_1} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{T_2} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{T_1} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{T_3} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \xrightarrow{T_1} \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \xrightarrow{T_4} \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \xrightarrow{T_2} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \xrightarrow{T_3} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \xrightarrow{T_4} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 2 \end{pmatrix} \xrightarrow{T_2}$$
$$\xrightarrow{T_2} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 2 \end{pmatrix} \xrightarrow{T_3} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 2 \end{pmatrix} \xrightarrow{T_4} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 3 \end{pmatrix}$$

Матричний опис базової мережі Петрі

Вектор запуску
переходу

$$v = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Запуск переходу T_3

Умова запуску
переходу

$$\forall i M_i \geq (a^- \cdot v)_i$$

Результат запуску
переходу

$$M' = M - a^- \cdot v + a^+ \cdot v$$

$$M' = M + (a^+ - a^-) \cdot v$$

Результат запуску
послідовності $T_1 - T_2 - T_1$

$$M' = M + a \cdot v^{(1)}$$

$$M'' = M' + a \cdot v^{(2)}$$

$$M''' = M'' + a \cdot v^{(3)}$$

Матриця
змінювань

$$a = a^+ - a^-$$

$$M''' = M + a \cdot (v^{(1)} + v^{(2)} + v^{(3)})$$

$$M''' = M + a \cdot v$$

Вектор запуску переходів

$$v = \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Кількість запусків
переходу T_1

Матричне рівняння станів базової мережі Петрі

Результатує маркірування
(після запуску переходів)

$$M' = M + a \cdot v$$

Початкове маркірування

Матриця змінювань

Вектор запуску переходів

Змінювання маркірування

$$\Delta M = a \cdot v$$

Аналітичне дослідження властивостей мереж Петрі

- k -обмеженість,
- досяжність,
- збереження,
- активність.

Аналітичне дослідження властивостей мереж Петрі

Властивість	Визначення
k – обмеженість	Якщо кількість маркерів в будь-якій позиції мережі Петрі не перевищує k маркерів, то мережа являється k – обмеженою.
досяжність	Досяжністю мережі Петрі називається множина досяжних маркірувань.
збереження	Якщо в мережі Петрі неможливе виникнення і знищення ресурсів, то мережа володіє властивістю збереження.
активність	Якщо з будь-якого досяжного початкового стану можливий перехід в будь-який інший досяжний стан, то мережа Петрі володіє властивістю активності.

Збереження (консервативність)

Означення. Якщо існує вектор w , компоненти якого цілі додатні числа, такий що $w^T \cdot M = w^T \cdot M'$ для будь – якого досяжного з початкового маркування, то мережа Петрі володіє властивістю збереження

$$w^T \cdot M = w^T \cdot M' \quad \forall M' (\text{досяжного})$$

$$w^T \cdot M = w^T \cdot (M + a \cdot v), \quad \forall v$$

$$0 = w^T \cdot a \cdot v, \quad \forall v$$

$$0 = w^T \cdot a \quad 0^T = (w^T \cdot a)^T$$

$$0 = a^T \cdot w$$

Твердження. Мережа Петрі володіє властивістю зберігання тоді і тільки тоді, коли існує вектор w , компоненти якого цілі додатні числа, такий, що $a^T \cdot w = 0, w_i \in Z_+$

S – інваріант мережі Петрі

Розв'язки рівняння

$$a^T \cdot w = 0,$$

де w – невідомий вектор розміру $|P| \times 1$,
називають S – інваріантом мережі Петрі.

S -інваріант, або інваріант стану, дозволяє досліджувати консервативність системи.

Консервативність означає, що існує зважена сума маркувань позицій мережі Петрі, яка для будь-якого досяжного маркування залишається незмінною.

Рівняння, які формулюються і розв'язуються в термінах цілих чисел, називають діофантовими.

Циклічність

Означення. Якщо існує послідовність запусків переходів, така що мережа повертається в початкове маркірування, то функціонування мережі Петрі є циклічним

$$M' = M$$

$$M + a \cdot v = M, \exists v$$

$$0 = a \cdot v,$$

$$\exists v$$

Твердження. Функціонування мережі Петрі є циклічним тоді і тільки тоді, коли існує вектор v , компоненти якого цілі невід'ємні числа, такий, що $a \cdot v = 0, v_i \in Z_+$

T – інваріант мережі Петрі

Розв'язки рівняння

$$a \cdot v = 0,$$

де v – невідомий вектор розміру $|T| \times 1$,
називають T -інваріантом мережі Петрі.

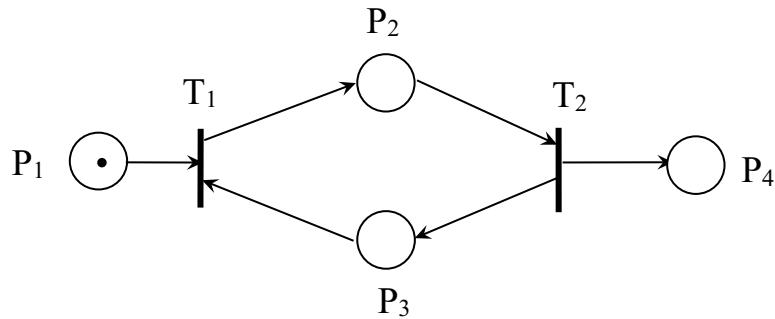
T -інваріант, або інваріант функціонування, означає досяжність початкового маркірування.

Цей інваріант є важливим для дослідження циклічності процесів функціонування.

Циклічність означає існування такої послідовності запусків переходів, що мережа Петрі повертається в початкове маркірування. Наявність T -інваріантів гарантує циклічність функціонування системи.

Досяжність

Існування невід'ємного цілого вектора запуску переходів, що задовольняє рівнянню $M' = M + a \cdot v$, є тільки необхідною, але не достатньою умовою



$$a = \begin{pmatrix} -1 & 0 \\ 1 & -1 \\ -1 & 1 \\ 0 & 1 \end{pmatrix}$$

$$M' = M + a \cdot v \Rightarrow \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 & 0 \\ 1 & -1 \\ -1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \Rightarrow \begin{pmatrix} -1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -v_1 \\ v_1 - v_2 \\ -v_1 + v_2 \\ v_2 \end{pmatrix}$$

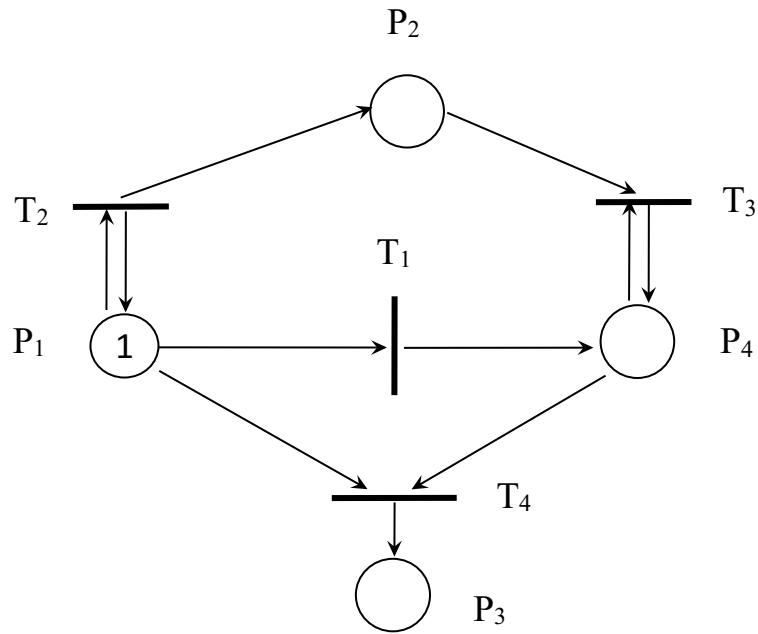
$$\Rightarrow \begin{cases} v_1 = 1 \\ v_2 = 1 \end{cases} \Rightarrow v = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Проте запуск переходів неможливий оскільки умова запуску не виконана.

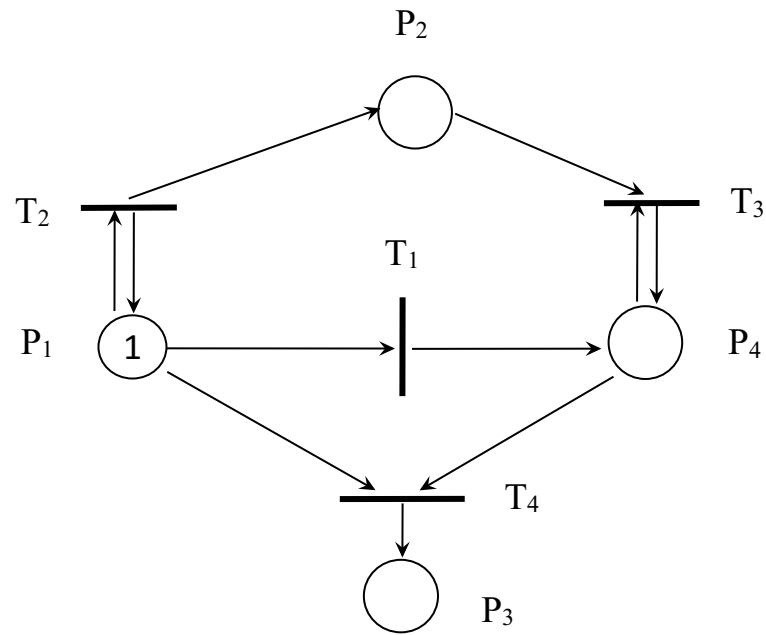
Активність

Рівень активності	Перехід T має рівень активності A, якщо
0	він ніколи не може бути запущений
1	існує маркірування (досяжне з початкового) , яке дозволяє запуск цього переходу T
2	для довільного цілого числа n існує послідовність запусків переходів, в якій переход T присутній принаймні n раз
3	існує нескінчена послідовність запусків переходів, в якій переход T присутній необмежено багато разів
4	якщо для довільного маркірування M, що є досяжним з початкового маркірування, існує послідовність запусків переходів, яка призводить до маркірування, що дозволяє запуск переходу T

Приклад визначення активності

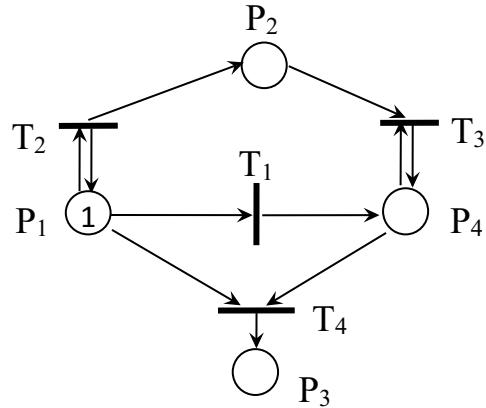


Приклад визначення активності

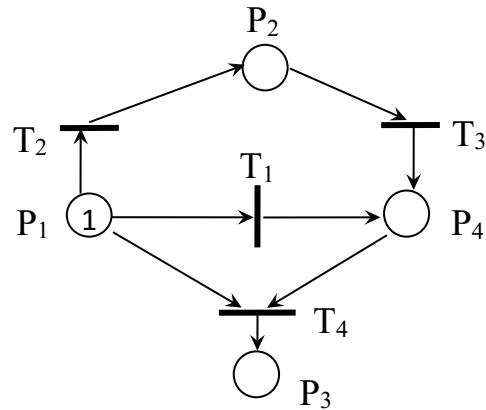


Рівень активності	Перехід
0	T_4
1	T_1
2	T_3
3	T_2

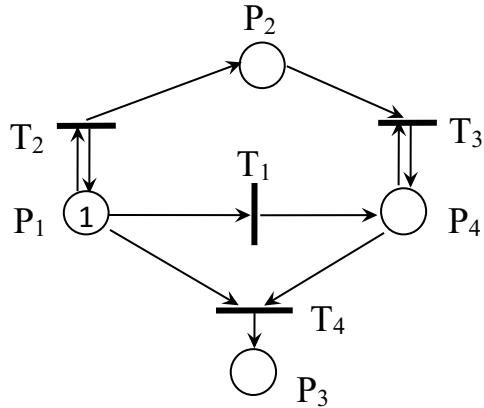
Приклад дослідження Т-інваріантів



$$a = \begin{pmatrix} -1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \end{pmatrix}$$



Приклад дослідження Т-інваріантів

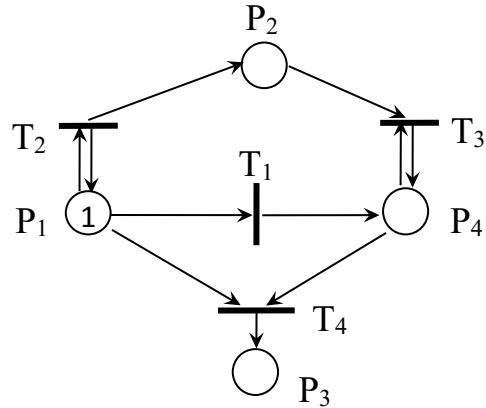


$$a = \begin{pmatrix} -1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \end{pmatrix}$$

$$a \cdot v = \mathbf{0} \Rightarrow \begin{pmatrix} -1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} -v_1 - v_4 \\ v_2 - v_3 \\ v_4 \\ v_1 - v_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow$$

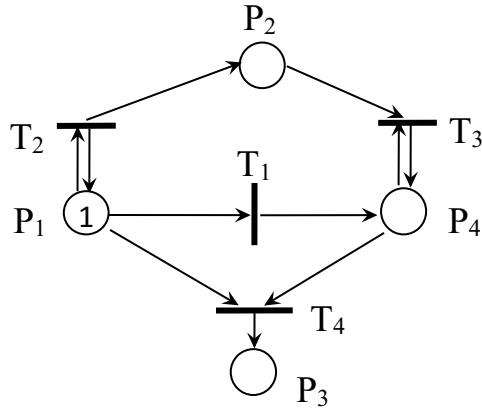
$$\Rightarrow \begin{cases} v_1 = 0 \\ v_2 = k \\ v_3 = k \\ v_4 = 0 \end{cases} \Rightarrow T\text{-інваріантів не існує. Отже, циклічність не гарантується.}$$

Приклад дослідження S-інваріантів



$$a = \begin{pmatrix} -1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \end{pmatrix}$$

Приклад дослідження S-інваріантів

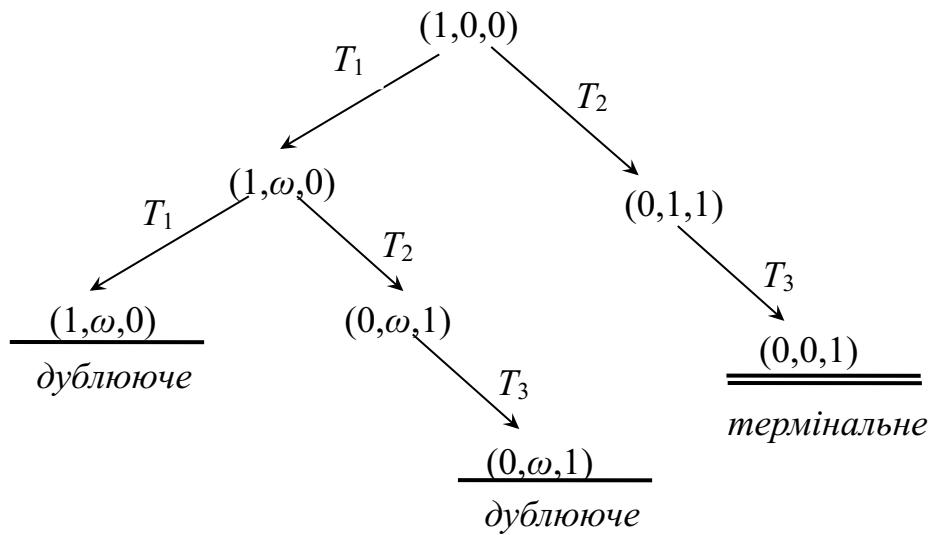
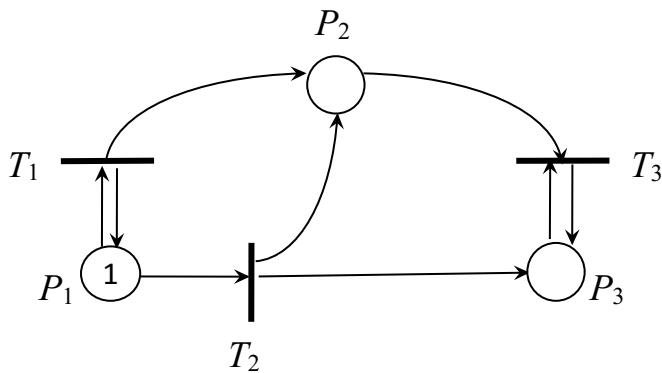


$$a = \begin{pmatrix} -1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \end{pmatrix}$$

$$a^T \cdot w = 0 \Rightarrow \begin{pmatrix} -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & -10 & 0 & 0 \\ -1 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} -w_1 + w_4 \\ w_2 \\ -w_3 \\ -w_1 + w_3 - w_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow$$

$$\Rightarrow \begin{cases} w_1 = 0 \\ w_2 = 0 \\ w_3 = 0 \\ w_4 = 0 \end{cases} \Rightarrow S - \text{інварінтів не існує. Отже, консерватіvnість відсутня.}$$

Дерево досяжності



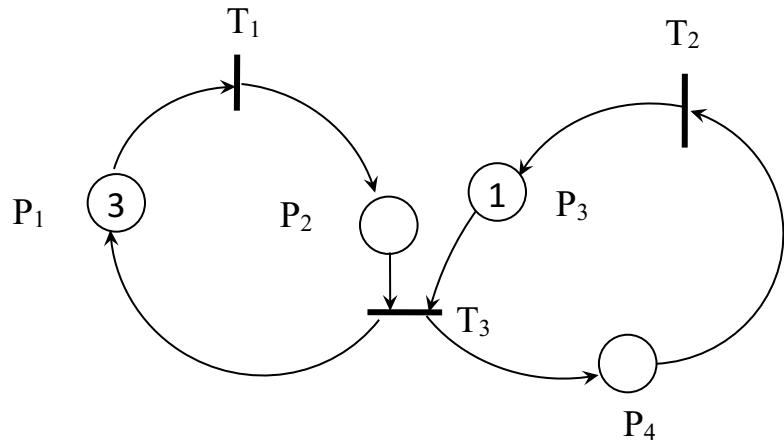
Дерево досяжності представляє множину досяжних маркірувань мережі Петрі. Дерево досяжності розпочинається з початкового маркірування, а закінчується термінальним або дублюючим маркіруванням.

Термінальним маркіруванням називається маркірування, в якому жоден з переходів мережі Петрі не запускається.

Дублюючим маркіруванням називається маркірування, що раніше зустрічалося в дереві досяжності

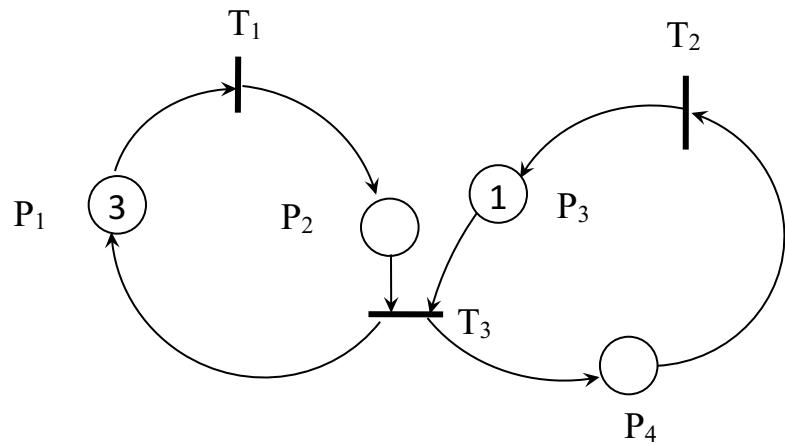
Символ ω в позиції M_j маркірування M з'являється тоді, коли на шляху до маркірування M' спостерігається маркірування M' , в якому всі значення, крім j -ого, не перевищують значення маркірування M , а j -е значення є меншим. Одного разу з'явившись, символ ω уже не змінюється і не зникає в дереві досяжності: додавання або віднімання від нескінченності є нескінченність.

Приклад «Три основні та один резервний пристрій»



$$a = \begin{pmatrix} -1 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

Приклад «Три основні та один резервний пристрій»

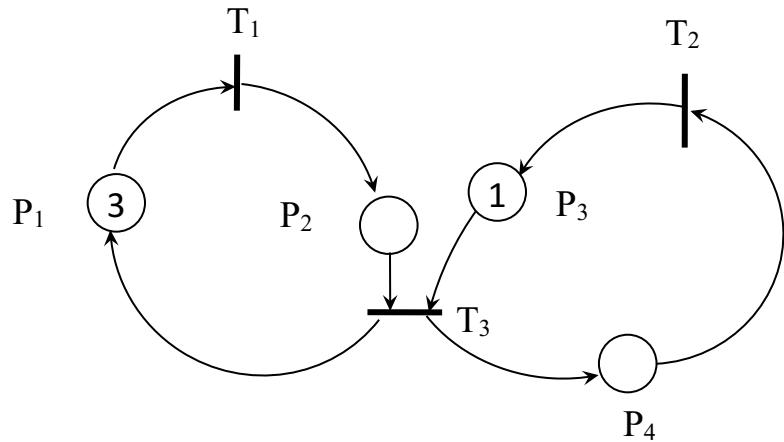


$$a = \begin{pmatrix} -1 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

$$a^T \cdot w = 0 \Rightarrow \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} -w_1 + w_2 \\ w_3 - w_4 \\ w_1 - w_2 - w_3 + w_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{cases} w_1 = 1 \\ w_2 = 1 \\ w_3 = 1 \\ w_4 = 1 \end{cases}$$

$\Rightarrow S$ – інварінт існує. Отже, консерватівність присутня.

Приклад «Три основні та один резервний пристрій»



$$a = \begin{pmatrix} -1 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

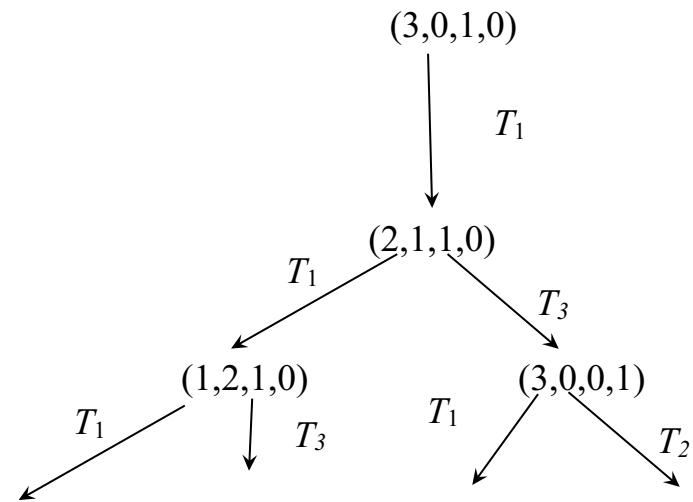
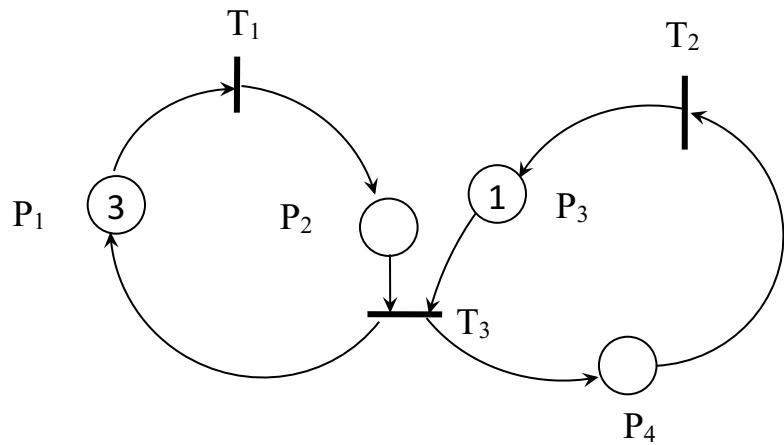
$$a^T \cdot w = 0 \Rightarrow \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} -w_1 + w_2 \\ w_3 - w_4 \\ w_1 - w_2 - w_3 + w_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{cases} w_1 = 1 \\ w_2 = 1 \\ w_3 = 1 \\ w_4 = 1 \end{cases}$$

$\Rightarrow S$ – інварінт існує. Отже, консерватівність присутня.

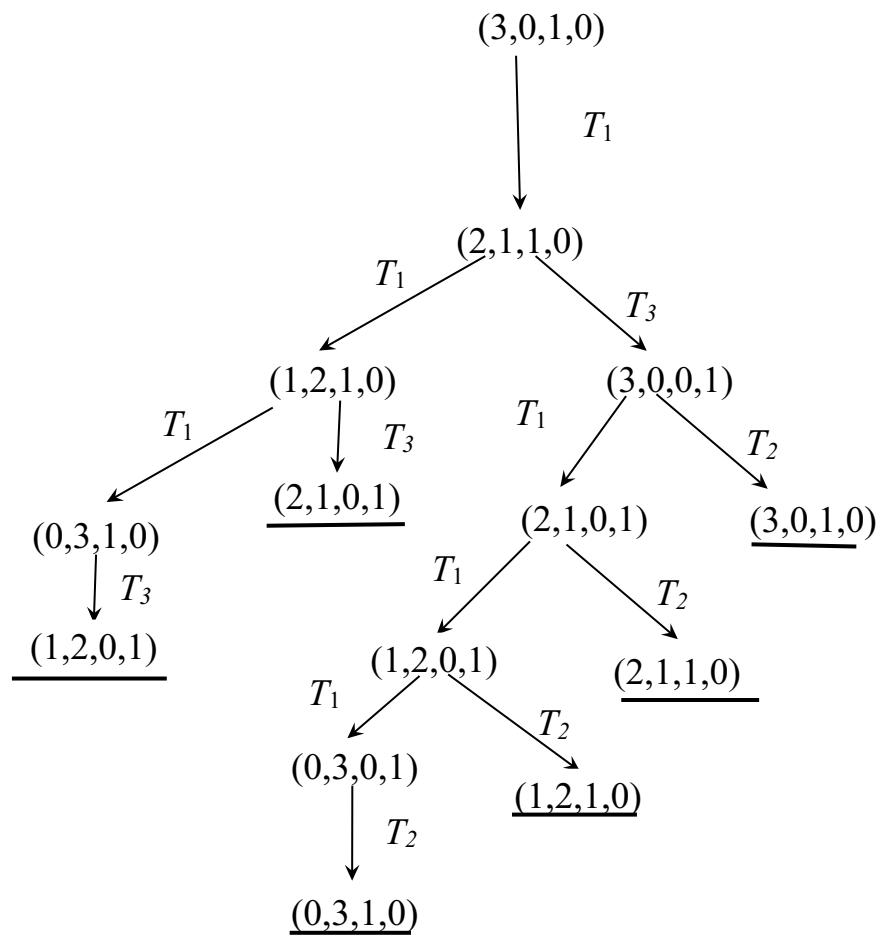
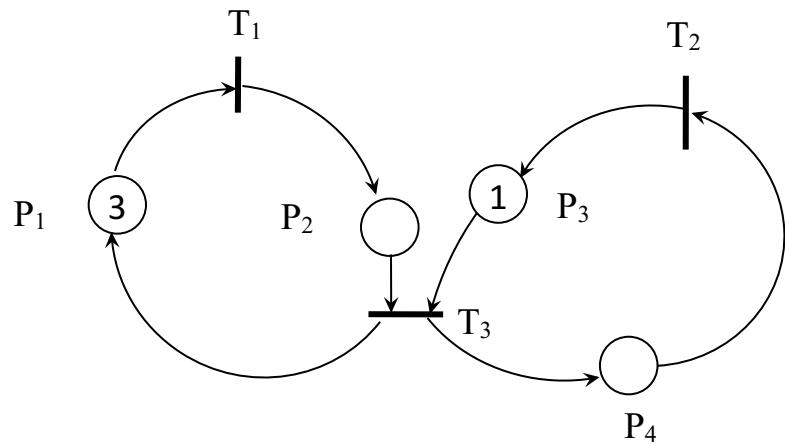
$$a \cdot v = 0 \Rightarrow \begin{pmatrix} -1 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} -v_1 + v_3 \\ v_1 - v_3 \\ v_2 - v_3 \\ -v_2 + v_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{cases} v_1 = 1 \\ v_2 = 1 \\ v_3 = 1 \end{cases}$$

$\Rightarrow T$ – інварінт існує. Отже, цикличність гарантується.

Приклад «Три основні та один резервний пристрій»



Приклад «Три основні та один резервний пристрій»



Порівняння способів аналітичного дослідження властивостей мережі Петрі

Властивість	Спосіб дослідження	
	Матричний підхід	Дерево досяжності
k-обмеженість	не досліджується	необхідна і достатня умова
зберігання	необхідна і достатня умова	необхідна і достатня умова
досяжність	тільки необхідна умова	послідовність переходів залишається невідомою
активність	не досліджується	не досліджується

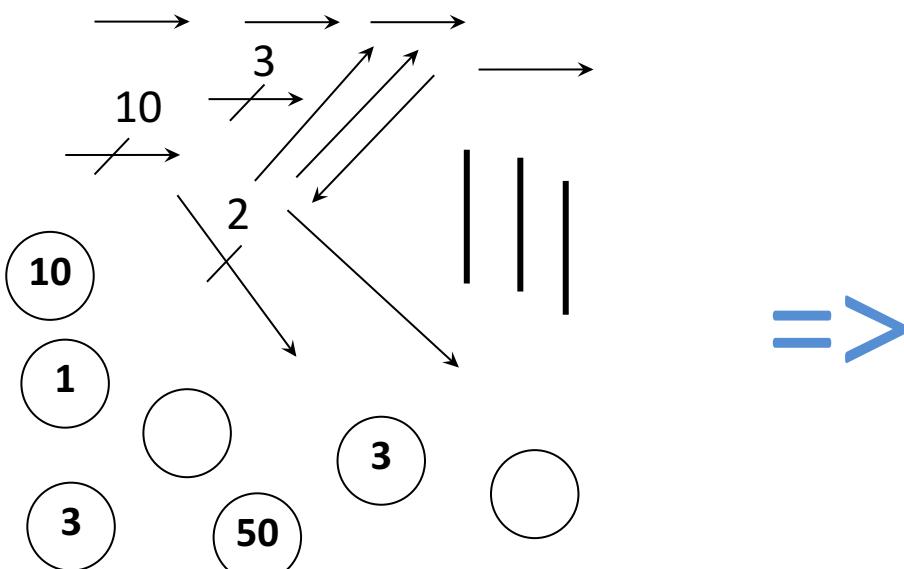
Алгоритм імітації стохастичної мережі Петрі

Допустима конструкція мережі Петрі

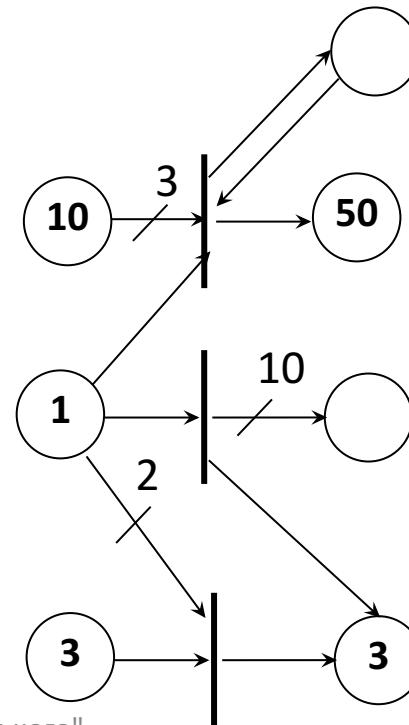
- ! Мережа Петрі повинна мати хоч один перехід
- ! Кожний перехід повинен мати хоч одну вхідну позицію і хоч одну вихідну позицію
- ! Вхідна дуга з'єднує позицію з переходом, вихідна дуга, навпаки, перехід з позицією
- ! Перехід з інформаційною вхідною дугою обов'язково повинен мати звичайну вхідну дугу
- ! 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.

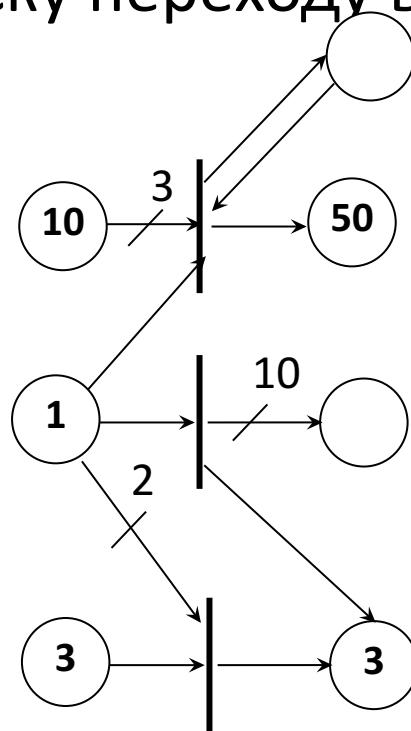


=>



Умова запуску переходу

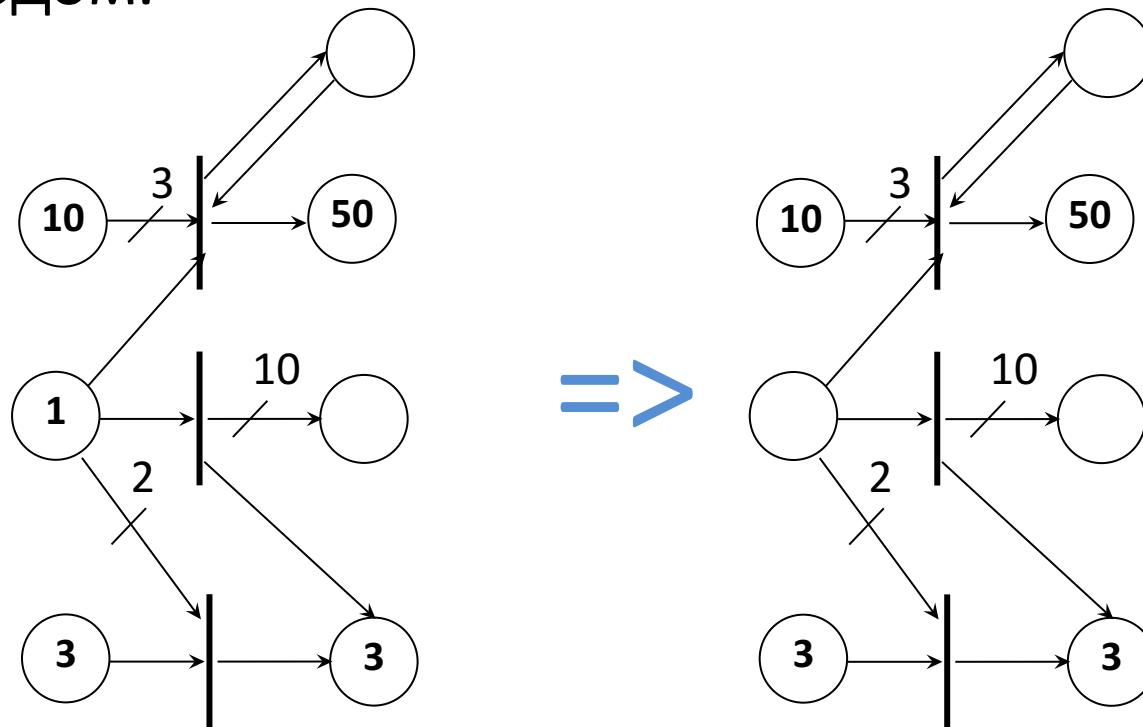
! Якщо у всіх вхідних позиціях переходу є маркери у кількості, рівній кратності дуги, то умова запуску переходу виконана.



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

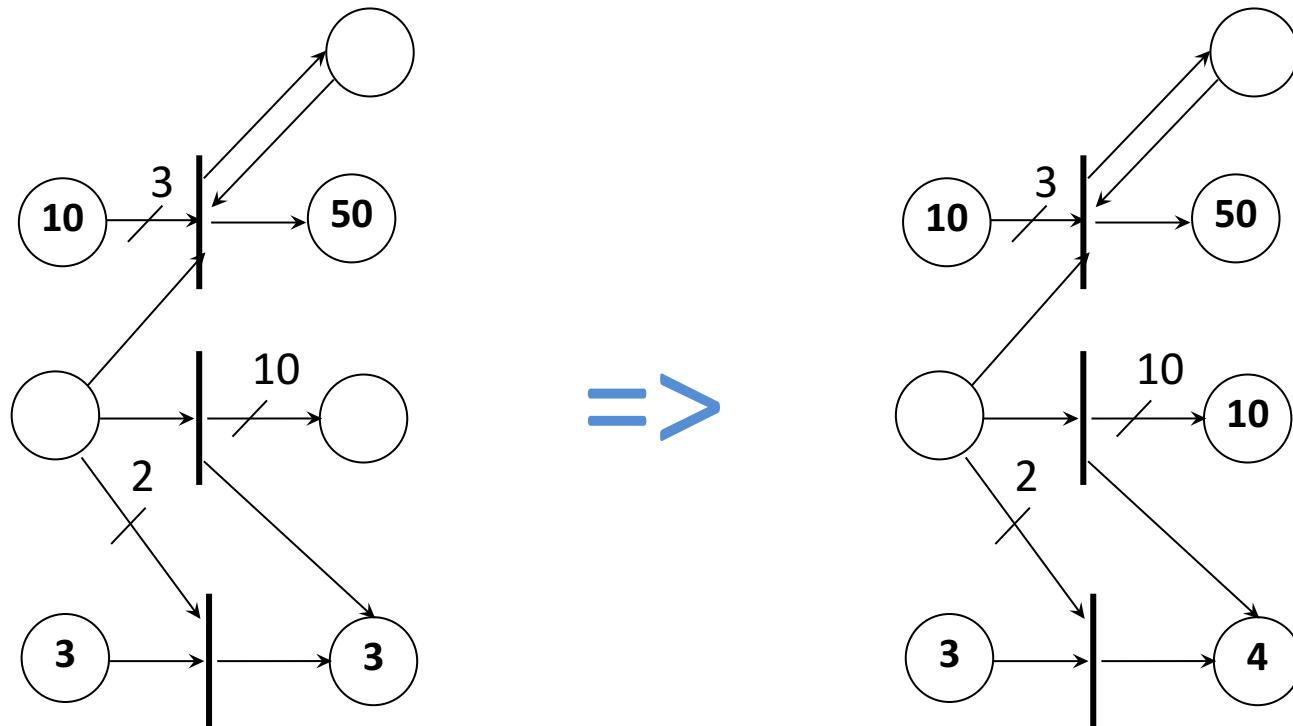
Вхід маркерів в перехід

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, вибрati з них один з рівною ймовірністю, а інші відкинути;

виконати вхід маркерів в переході, 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 переход 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, вибрati з них один з рівною ймовірністю, а інші відкинути;

виконати вхід маркерів в переході, 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 вибрati 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 вибрati 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 переход він переходить в стан «зайнятий» і інші входи здійснюватись не можуть. Переход, який в стані «зайнятий», не проходить перевірку на умову запуску переходу.

Алгоритм починається з входу маркерів в переходи мережі Петрі.

Доки $t < T_{mod}$

 визначити момент найближчої події min ;

 зібрати статистику про функціонування моделі;

$t = min$;

 якщо $t < T_{mod}$

 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 вихід маркерів в переходи мережі Петрі:

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

 (за заданими значеннями пріоритету та ймовірності запуску)

 доки список переходів з виконаною умовою запуску непорожній

 виконати вхід маркерів в переході:

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

 моменту виходу маркерів з переходу;

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

 (за заданими значеннями пріоритету та ймовірності запуску);

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

Вважається, що кількість входів в переход 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 «некінченність», 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 «некінченність».

При вході маркерів в переход виконується повторення входу маркерів в переход доки список переходів з виконаною умовою запуску не порожній. При цьому в один і той самий переход 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 ії 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);
```

Рівняння стохастичної мережі Петрі

Формальний опис стохастичної мережі Петрі

- [Murata T. (1989). Petri Nets: Properties, Analysis and Applications, *Proceedings of IEEE*, vol.77 (4), 541-580.]
- [Zaitsev D. A. (2004). Invariants of Timed Petri Nets, *Cybernetics and Systems Analysis*, vol. 40, pages226–237 (2004)]
- [Stetsenko I.V. (2012) State equations of stochastic timed Petri nets with informational relations. *Cybernetics and Systems Analysis*, vol. 48(5), 784–797]

Мережа Петрі:

$$N = (\mathbf{P}_N, \mathbf{T}_N, \mathbf{A}_N, \mathbf{W}_N, \mathbf{K}_N, \mathbf{I}_N, \mathbf{R}_N)$$

- множина позицій
- множина переходів
 - множина дуг
 - кратності дуг
 - пріоритети та ймовірності запуску переходів
- часові затримки в переходах
- множина вхідних та множина вихідних позицій перехода T
- множина вхідних і множина вихідних переходів позиції P

Стан стохастичної мережі Петрі

Stetsenko I.V. (2012) State equations of stochastic timed petri nets with informational relations. Cybernetics and Systems Analysis. Vol. 48, no. 5, 784–797.

$\mathbf{S}(t) = (\mathbf{M}(t), \mathbf{E}(t))$ - стан стохастичної мережі Петрі в момент часу t

$\mathbf{M}(t) = \{M_P(t) \mid M_P(t) \in Z_+, P \in \mathbf{P}\}$ - стан позицій

- стан переходів

- стан переходу T ,

q – номер запланованої події виходу маркерів з переходу
в момент часу t

якщо найближчим часом не очікується вихід маркерів з переходу

$\mathbf{S}(t) \in \{\mathbf{S}(t) \mid (M_P(t) \geq 0 \forall P \in \mathbf{P}) \wedge ([E_T(t)]_q \geq 0 \forall T \in \mathbf{T}, \forall q = 1, \dots, |E_T(t)|)\}$

Визначення моменту найближчої події

$\tau_T(t) = \min_q [E_T(t)]_q$, - момент запланованої найближчої події для переходу Т на поточний момент часу

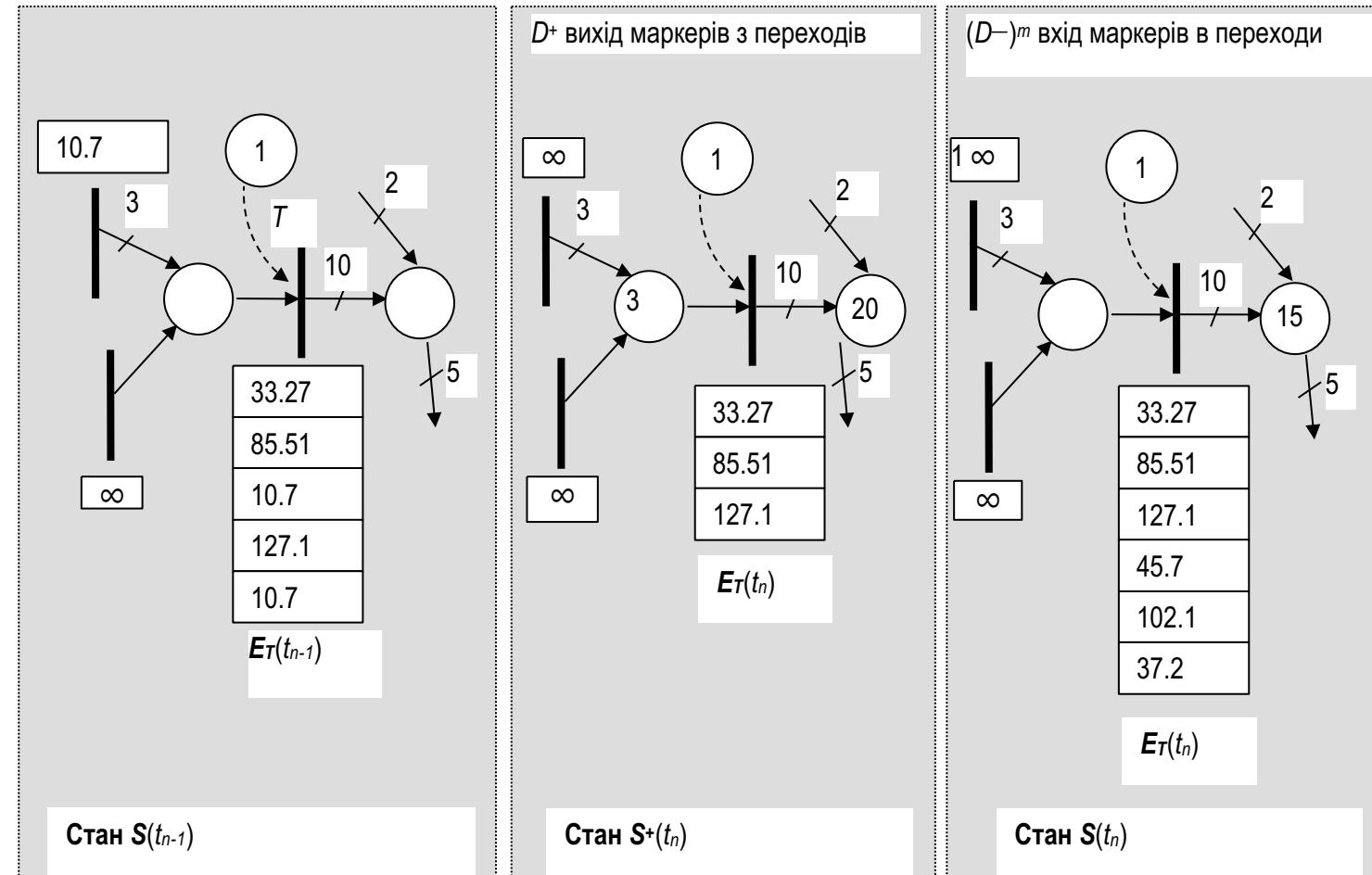
$t' = \min_T \tau_T(t), t' \geq t$ - момент запланованої найближчої події для мережі Петрі на поточний момент часу

$t_n = \min_T \left(\min_q [E_T(t_{n-1})]_q \right), t_n \geq t_{n-1}$

- визначення моменту найближчої події для мережі Петрі на поточний момент часу

Змінювання стану стохастичної мережі Петрі з часовими затримками

Змінювання стану мережі Петрі в момент часу $t_n=10,7$



$$S(t_{n-1}) \rightarrow S^+(t_n) \rightarrow S(t_n)$$

Вихід маркерів з переходів

$Y : \mathbf{T} \times \mathfrak{R} \rightarrow \{0;1\}$ - предикат, що визначає для кожного переходу співпадіння моменту найближчої події з поточним моментом часу

$$(\tau_T(t) = t') \Rightarrow Y(T, t') = 1,$$

$$(\tau_T(t) \neq t') \Rightarrow Y(T, t') = 0.$$

$$\forall P \in \mathbf{P} \quad M_P^+(t') = M_P(t) + \sum_{T \in \bullet^P} Y(T, t') \cdot W_{T,P} |s_T(t)|, \quad \text{- змінювання стану позиції } P$$

$$\forall T \in \mathbf{T} \mid Y(T, t') = 1 \quad E_T^+(t') = \begin{cases} \{\infty\} \leftarrow |s_T(t)| = |E_T(t)|, \\ E'_T(t') = E_T(t) \setminus \{E_T(t)\}_q \mid q \in s_T(t) \} \leftarrow |s_T(t)| \neq |E_T(t)|, \end{cases}$$

- змінювання стану переходу T

$D^+ : \mathbf{E}(t) \times \mathbf{M}(t) \rightarrow \mathbf{E}(t') \times \mathbf{M}(t')$ - перетворення стану стохастичної мережі Петрі

$\mathbf{S}(t') = D^+(\mathbf{S}(t))$ - змінювання стану стохастичної мережі Петрі

Вхід маркерів в переходи

$Z : T \times \mathfrak{R} \rightarrow \{0;1\}$ - предикат, що визначає для кожного переходу умову виконання запуску

$$(\forall P \in {}^\bullet T \quad M_P^+(t') \geq W_{P,T}) \Rightarrow Z(T, t') = 1$$

$$(\exists P \in {}^\bullet T \quad M_P^+(t') < W_{P,T}) \Rightarrow Z(T, t') = 0$$

$\Psi(t') = \{T \mid Z(T, t') = 1, T \in \mathbf{T},\}$ - множина переходів, для яких виконана умова запуску

$\Psi'(t') = \overline{\mathbf{T}_\Psi} \cup \widetilde{\mathbf{T}}_\Psi$ - множина переходів, що не містять спільні позиції з іншими, та переходів, вибраних в результаті вирішення конфлікту

$X : \mathbf{T} \times \mathfrak{R} \rightarrow \{0;1\}$ - предикат, що визначає для кожного переходу приналежність до множини переходів, вибраних в результаті вирішення конфлікту

$$T \in \Psi'(t') \Rightarrow X(T, t') = 1$$

$$T \notin \Psi'(t') \Rightarrow X(T, t') = 0$$

Вхід маркерів в переходи

$$\forall P \in \mathbf{P} \quad M_P(t') = M_P^+(t') - \sum_{T \in P^\bullet \setminus P^\circ} W_{P,T} \cdot X(T, t'), \quad - \text{змінювання стану позиції } P$$

$$\forall T \in \mathbf{T} \mid X(T, t') = 1 \quad E_T(t') = \begin{cases} \{t' + R_T\} \leftarrow \tau_T = \infty \\ E_T^+(t') \cup \{t' + R_T\} \leftarrow \tau_T < \infty \end{cases}$$

- змінювання стану переходу T

$D^- : \mathbf{E}(t') \times \mathbf{M}(t') \rightarrow \mathbf{E}(t') \times \mathbf{M}(t')$ - перетворення стану стохастичної мережі Петрі

$D^-(\mathbf{S}(t'))$ - змінювання стану стохастичної мережі Петрі

Багатократний вхід маркерів в переходи

$$(D^-)^n = D^- \circ D^- \circ D^- \dots \circ D^-$$

$$m : (D^-)^n (\mathbf{S}(t')) : \bigvee_T Z(T, t') = 0$$

$$M_P(t') = M_P^+(t') - \sum_{T \in {}^\bullet P \setminus {}^\circ P} W_{P,T} \cdot u_T(t'),$$

$$\{u_T(t')\} : \left(u_T(t') \leq \min_{P \in {}^\bullet T} \left\{ \frac{M_P^+(t')}{W_{P,T}} \right\} \right) \wedge \left(\exists P \in {}^\bullet T : \sum_{T \in {}^\bullet P \setminus {}^\circ P} W_{P,T} \cdot u_T(t') > \max_{T \in {}^\bullet P \setminus {}^\circ P} (M_P^+(t') - W_{P,T}) \right)$$

Багатократний вхід маркерів в переходи

$$\forall P \in \mathbf{P} \quad M_P(t') = M_P^+(t') - \sum_{T \in {}^\bullet P \setminus {}^\circ P} W_{P,T} \cdot u_T(t'),$$

$$\forall T \in \mathbf{T} \quad E_T(t') = \begin{cases} \underbrace{\{t' + R_T\} \cup \dots \cup \{t' + R_T\}}_{u_T(t')} \leftarrow \tau_T(t') = \infty \\ E_T^+(t') \cup \underbrace{\{t' + R_T\} \cup \dots \cup \{t' + R_T\}}_{u_T(t')} \leftarrow \tau_T(t') < \infty \end{cases}$$

Рівняння станів стохастичної мережі Петрі з часовими затримками, з конфліктними та багатоканальними переходами



- найближчий момент виходу маркерів з переходу
- перетворення стану мережі Петрі, пов'язане з виходом маркерів з переходів,
- перетворення стану мережі Петрі, пов'язане з входом маркерів в переходи
- кількість входів маркерів в переходи, за якої досягається стан мережі Петрі при якому жоден з переходів мережі Петрі не запускається

Аналіз обчислюваної складності

Stetsenko I.V., Dychyn A., Dorosh V.I. Petri-Object Simulation: Software Package and Complexity. Proceedings of the 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2015). Warsaw (Poland), 2015. P.381-385.

$$O\left(\left|T\right|^2 \cdot V \cdot timeMod \cdot \left(\underset{T \in T}{mean} \left|T^\bullet\right| + V + V \cdot \left|T\right| \cdot \underset{T \in T}{mean} \left|\bullet T\right| + V^2 \cdot \left|T\right| + V \cdot K^2 \right)$$

$$V = \underset{T \in T}{mean} v_T$$
 - середня кількість активних каналів переходу

$$K = mean \left| T_\Psi \right|$$
 - середня кількість конфліктних переходів

Матричний опис стану стохастичної мережі Петрі

- матриця виходів

$$\mathbf{a}^+ = \left\| a_{T,P}^+ \right\|$$

$$\mathbf{a} = \mathbf{a}^+ - \mathbf{a}^-$$

- матриця входів

$$\mathbf{a}^- = \left\| a_{T,P}^- \right\|$$

- вектор кількості активних каналів переходів

$$v_T(t) = \begin{cases} |E_T(t)|, & \tau_T < \infty, \\ 0, & \tau_T = \infty. \end{cases}$$

- вектор кількості входів в переходи за період часу $[t_0, t]$

$$\sum_{j=1}^n Z(T, t_j) \cdot u_T(t_j) = \gamma_T(t_n)$$

- вектор кількості виходів з переходів за період часу $[t_0, t]$

$$\sum_{j=1}^n Y(T, t_j) \cdot |s_T(t_{j-1})| = \eta_T(t_n)$$

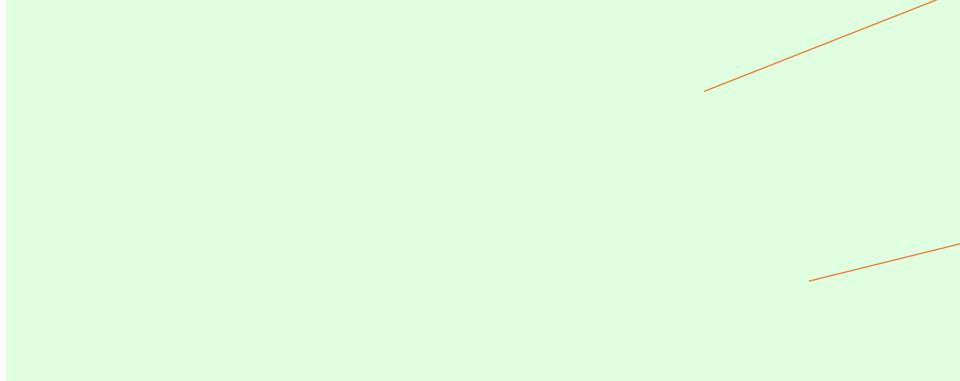
$\mathbf{M}(t) = M_p(t)$ - вектор маркірування

$\mathbf{i}(t) = \mathbf{M}(t) + \mathbf{a}^+ \cdot \mathbf{v}(t)$ - вектор розширеного маркірування

!!!! враховує не тільки маркери, які в позиціях, але й маркери, які очікують виходу з переходів

Матричні рівняння станів стохастичної мережі Петрі з часовими затримками, з багатоканальними та конфліктними переходами, з інформаційними зв'язками

В термінах вектора розширеного маркірування:



Аналогічне базовому, але сформульованого для розширеного маркірування та вектора кількості входів в переход

Додаткове рівняння формулює залежність між кількістю запусків, кількістю входів в переходи та кількістю виходів з переходів

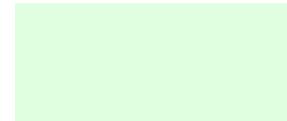
Порівняння матричних рівнянь станів стохастичної мережі Петрі з часовими затримками з відомими рівняннями станів мереж Петрі

\Rightarrow Матричні рівняння станів стохастичної мережі Петрі з часовими затримками без інформаційних зв'язків

- кількість завершених запусків переходу

\Rightarrow

\Rightarrow

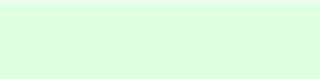


\Rightarrow

\Rightarrow

\Rightarrow

\Rightarrow



\Rightarrow

\Rightarrow Фундаментальне рівняння станів базової мережі Петрі

$$R_T = \text{const} \Rightarrow t_n = t_0 + n \cdot \Delta t$$

$$\Rightarrow t_n = n$$

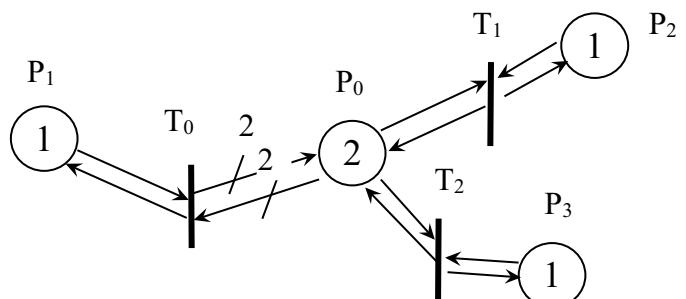
\Rightarrow

\Rightarrow

\Rightarrow

Фундаментальне рівняння станів детермінованої мережі Петрі з часовими затримками

Приклад



$$P_0 \geq 2, P_1 \geq 1 \Rightarrow Z(T_0, 0.0) = 1$$

$$P_0 \geq 1, P_2 \geq 1 \Rightarrow Z(T_1, 0.0) = 1$$

$$P_0 \geq 1, P_3 \geq 1 \Rightarrow Z(T_2, 0.0) = 1$$

$D^- :$ $M_{P_0}(0.0) = 2 - (1 \cdot 1 + 1 \cdot 1) = 0$

$$M_{P_1}(0.0) = 1 - 0 = 1$$

$$M_{P_2}(0.0) = 1 - 1 \cdot 1 = 0$$

$$M_{P_3}(0.0) = 1 - 1 \cdot 1 = 0$$

$$\mathbf{T} = \{T_0, T_1, T_2\}$$

$$\mathbf{P} = \{P_0, P_1, P_2, P_3\}$$

$$\mathbf{A} = \{(P_0, T_0), (T_0, P_0), (P_0, T_1), (T_1, P_0), (P_0, T_2), (T_2, P_0), (P_1, T_0), (T_0, P_1), (P_2, T_1), (T_1, P_2), (P_3, T_2), (T_2, P_3)\}$$

$\Psi = \{T_0, T_1, T_2\}$ P_0 - конфліктна позиція, можливі варіанти T_0 або $\{T_1, T_2\}$

Припустимо вибір пав на $\{T_1, T_2\}$

$$\Psi' = \{T_1, T_2\} \Rightarrow X(T_1) = 1, X(T_2) = 1$$

$$E_{T_0}(0.0) = \{$$

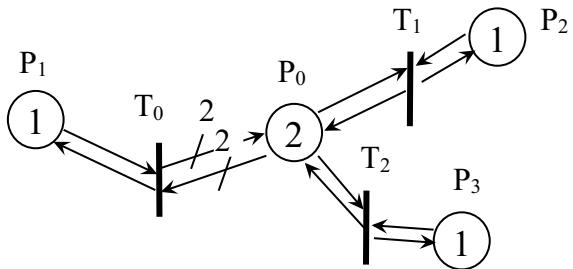
$$E_{T_1}^\infty(0.0) = \{0.0 + 0.7\}$$

$$E_{T_2}(0.0) = \{0.0 + 1.2\}$$

$$S(0.0) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \{\infty\} \\ \{0.7\} \\ \{1.2\} \end{pmatrix}$$

$$Z(T_0, 0.0) = 0$$

$$Z(T_1, 0.0) = 0$$



$$t_1 = \min \{ \infty, 0.7, 1.2 \} = 0.7$$

$$\underline{D^+ :} \quad \begin{aligned} Y(T_0, 0.7) &= 0 \\ Y(T_1, 0.7) &= 1 \\ Y(T_2, 0.7) &= 0 \end{aligned}$$

$$M_{P_0}(0.7) = 0 + 1 \cdot 1 = 1$$

$$M_{P_1}(0.7) = 1 + 0 = 1$$

$$M_{P_2}(0.7) = 0 + 1 \cdot 1 = 1$$

$$M_{P_3}(0.7) = 0 + 0 = 0$$

$$E_{T_0}(0.7) = \{$$

$$E_{T_1}^\infty(0.7) = \{$$

$$E_{T_2}^\infty(0.7) = \{1.2\}$$

$$S^+(0.7) = \left\{ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \{\infty\} \\ \{\infty\} \\ \{1.2\} \end{pmatrix} \right\}$$

$$P_0 < 2, P_1 \geq 1 \Rightarrow Z(T_0, 0.0) = 0$$

$$\Psi = \{T_1\} \Rightarrow X(T_1) = 1$$

$$P_0 \geq 1, P_2 \geq 1 \Rightarrow Z(T_1, 0.0) = 1$$

$$P_0 < 1, P_3 \geq 1 \Rightarrow Z(T_2, 0.0) = 0$$

$$\underline{D^- :} \quad M_{P_0}(0.7) = 1 - 1 \cdot 1 = 0$$

$$M_{P_1}(0.7) = 1 - 0 = 1$$

$$M_{P_2}(0.7) = 1 - 1 \cdot 1 = 0$$

$$M_{P_3}(0.7) = 0 - 0 = 0$$

$$E_{T_0}(0.7) = \{$$

$$E_{T_1}^\infty(0.7) = \{0.7 + 0.9\}$$

$$E_{T_2}(0.7) = \{1.2\}$$

$$S(0.7) = \left\{ \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \{\infty\} \\ \{1.6\} \\ \{1.2\} \end{pmatrix} \right\}$$

$$Z(T_0, 0.7) = 0$$

$$Z(T_1, 0.7) = 0$$

Формалізм Петрі-об'єктної моделі

Взаємне співвідношення класів мереж Петрі



Еквівалентності переходів різних класів мереж Петрі

Перехід класичної мережі Петрі



Перехід з нульовою часовою затримкою мережі
Петрі з часовими затримками



Одноканальний перехід з нульовою часовою
затримкою мережі Петрі з часовими затримками та з
багатоканальними переходами

Перехід мережі Петрі з часовою затримкою



Одноканальний перехід мережі Петрі з часовими
затримками та з багатоканальними переходами

Теоретичні основи стохастичної мережі Петрі з багатоканальними переходами

- Логіко-алгебраїчні рівняння станів
- Матричні рівняння станів
- Дослідження властивостей аналітичними методами
- Алгоритм імітації (реалізує обчислення за рівняннями станів)
- Обчислювальна складність моделі мереж Петрі

Рівняння станів стохастичної мережі Петрі з багатоканальними переходами та з інформаційними зв'язками

$$\begin{cases} t_n = \min_T \tau_T(t_{n-1}), t_n \geq t_{n-1}, \\ \mathbf{S}(t_1) = (D^-)^n (\mathbf{S}(t_0)), \\ \mathbf{S}(t_n) = (D^-)^n (D^+ (\mathbf{S}(t_{n-1}))) \\ n = 2, 3, \dots \end{cases}$$

де $\tau_T(t) = \min_q [E_T(t)]_q$ - найближчий момент виходу маркері з переходу

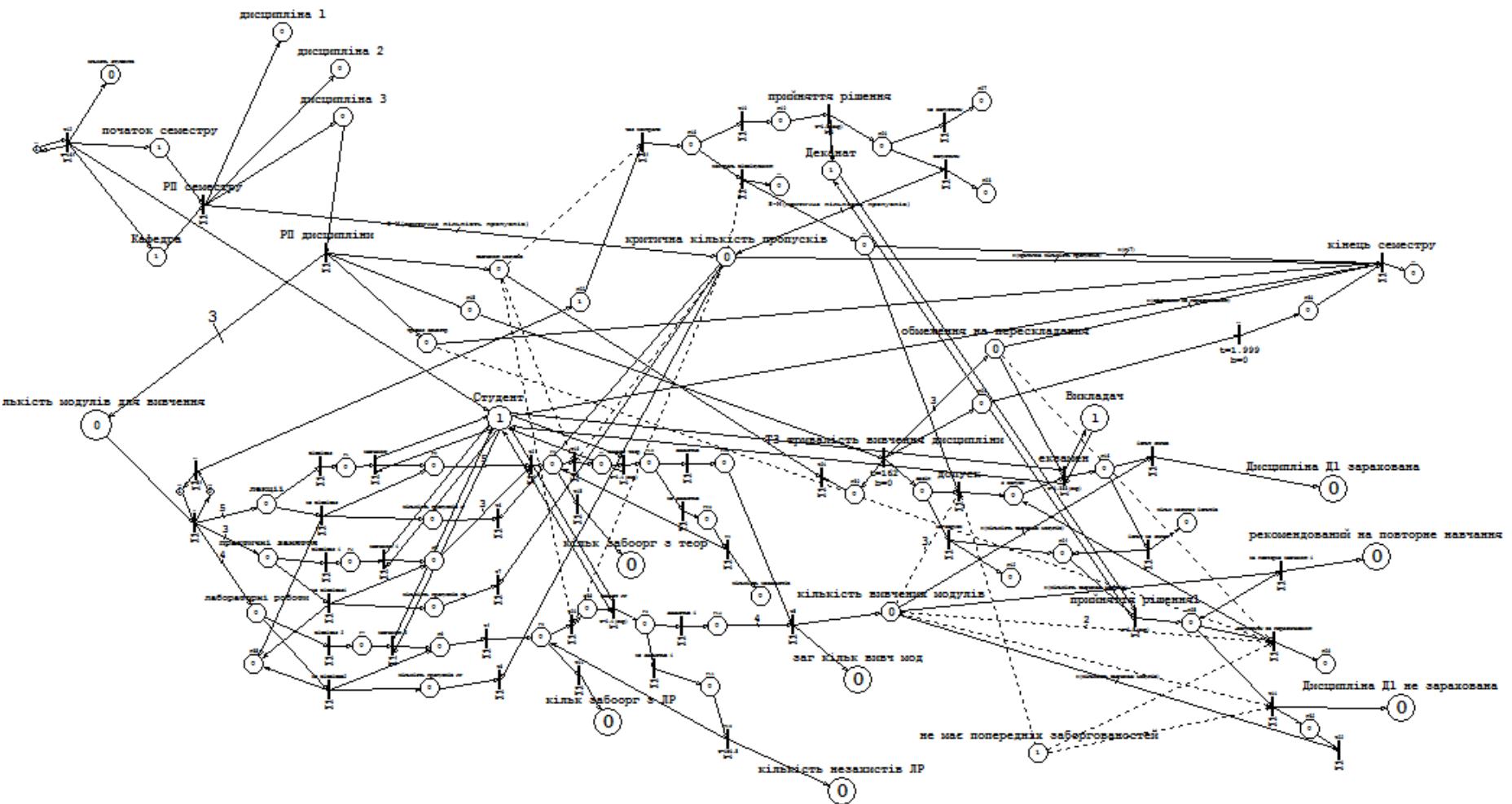
$m : (D^-)^n (\mathbf{S}(t_n)) : \bigvee_T Z(T, t_n) = 0$ - досягається стан, при якому жоден з переходів мережі Петрі не запускається

$Z(T, t_n)$ - предикат, який визначає виконання умови запуску переходу T в момент часу t_n

$$(\forall P \in {}^\bullet T \quad M_P^+(t_n) \geq W_{P,T}) \Rightarrow Z(T, t_n) = 1$$

$$(\exists P \in {}^\bullet T \quad M_P^+(t_n) < W_{P,T}) \Rightarrow Z(T, t_n) = 0$$

Недолік мережі Петрі



“При проектуванні складної програмної системі необхідно складати її з невеликих підсистем, кожну з яких можна відлагодити незалежно від інших.”

Граді Буч

ООП і мережі Петрі

Блочна структура мережі Петрі

[Ямпольський Л.С., Лавров О.А. Штучний інтелект у плануванні та управлінні виробництвом. – К.:Вища шк., 1995. - 255с.]

[Стеценко І.В., Бойко О.В. Система імітаційного моделювання засобами сіток Петрі // Математичні машини і системи. – Київ, 2009. – №1. – С.117-124.]

Функціональні підмережі

[Dmitriy A. Zaitsev Functional Petri net // Universite Paris Paris-Dauphine. - Cahier N 224. – mars 2005. – P.1-62.]

Об'єктно-орієнтовані мережі Петрі

[Lakos C. Object Oriented Modeling with Object Petri Nets // Concurrent Object-Oriented Programming and Petri Nets. - 2001. - P. 1-37.]

[Lakos C., Keen C. LOOPN++: a new language for object-oriented Petri nets, Technical Report R94-4, Networking Research Group, University of Tasmania,Australia, April 1994.]

Ієрархічна об'єктно-орієнтована мережа Петрі

[Hue Xu Timed Hierarchical object-oriented Petri net // Petri Net, Theory and Applications, Book edited by: Vedran Kordic. - I-Tech Education and Publishing, Vienna, Austria. - 2008. - P.253-280.]

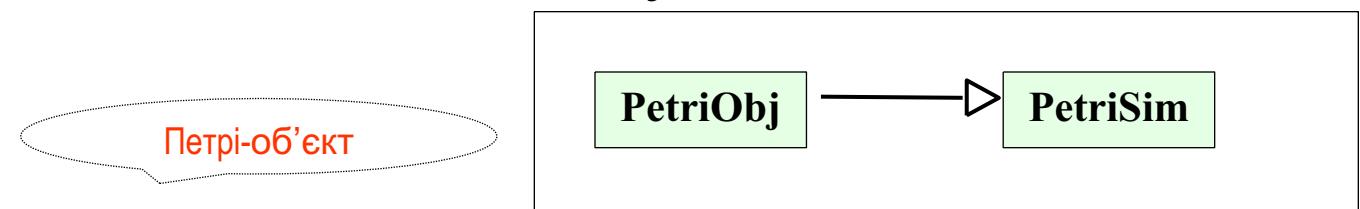
Високорівнева мережа Петрі для опису ООП

[Hong,J.E., Bae D.H. High-level Petri net for incremental specification of object-oriented system requirements // Institution of Engineering and Technology, IEEE Proceedings – Software. - 2001. - Vol. 148, No.1 - P.11-18.]

Поняття Петрі-об'єкта



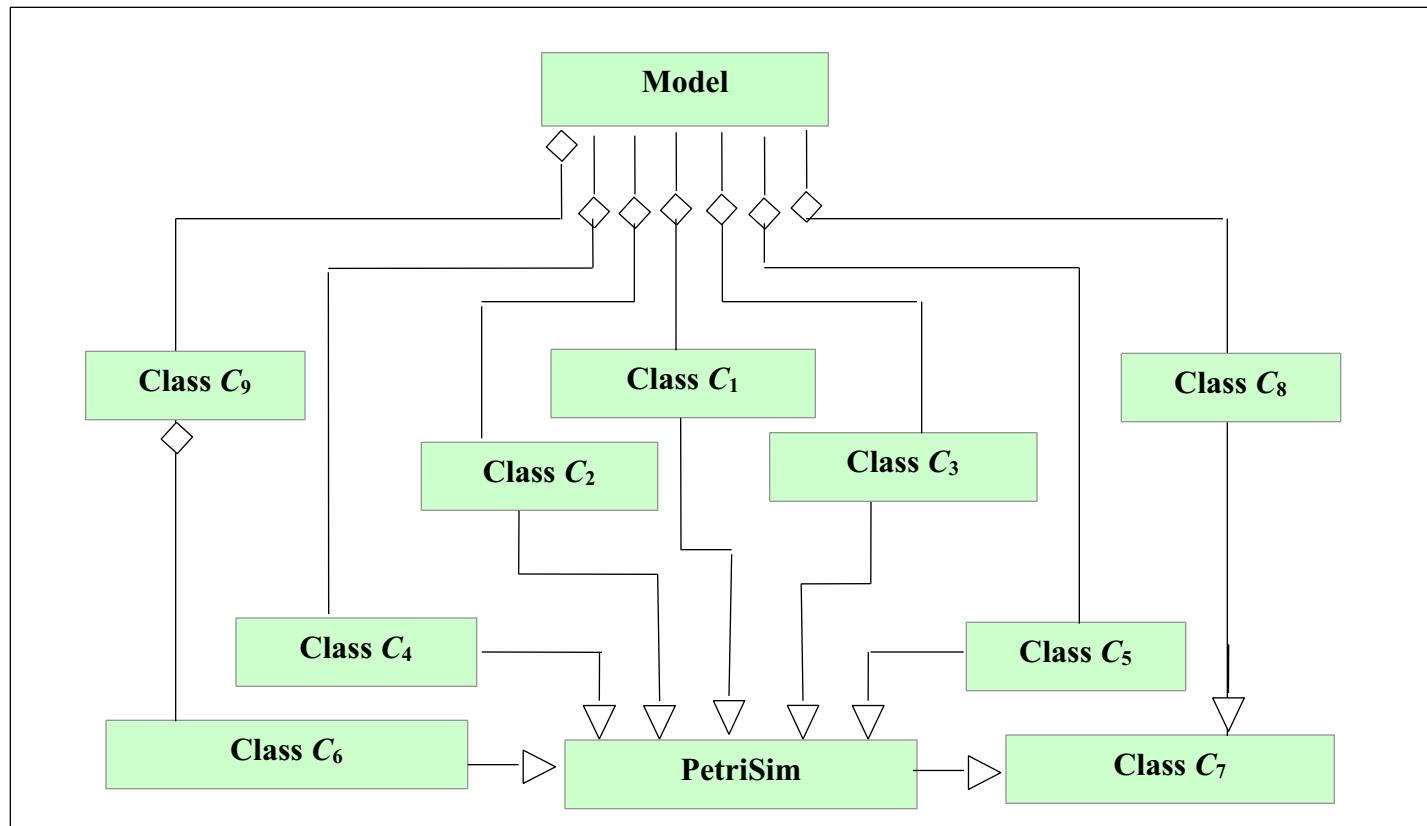
Означення. **Петрі-об'єктом** (PetriObj) називатимемо об'єкт, що є нащадком об'єкта Петрі-імітатор (PetriSim): PetriObj $\xrightarrow{inherit}$ PetriSim



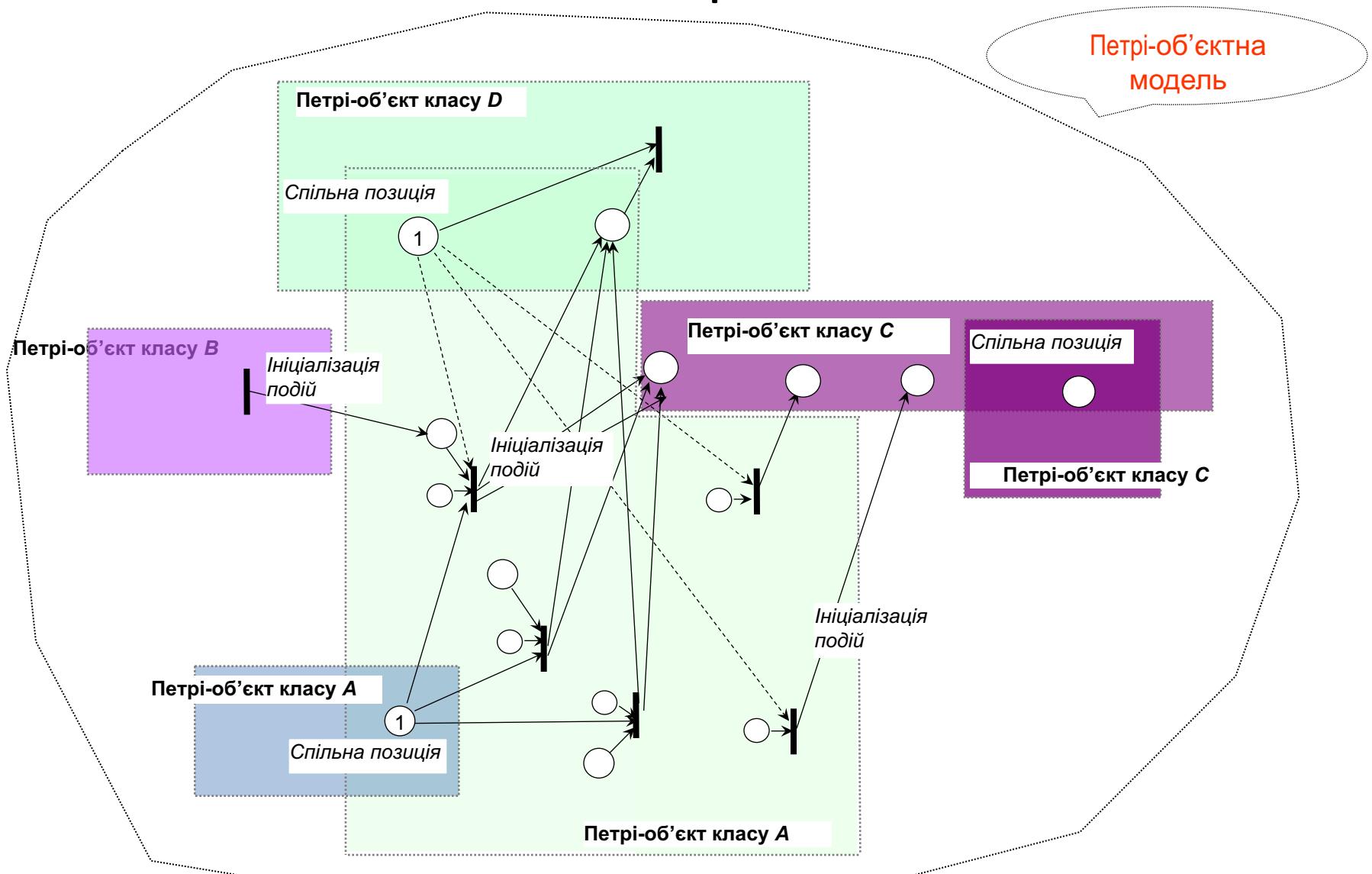
Поняття Петрі-об'єктної моделі

Означення. Петрі-об'єктною моделлю називатимемо модель, що отримана в результаті агрегування Петрі-об'єктів:

$$Model = \bigcup_N O_N \quad , \text{ де } O_N \xrightarrow{\text{inherit}} \text{PetriSim}$$

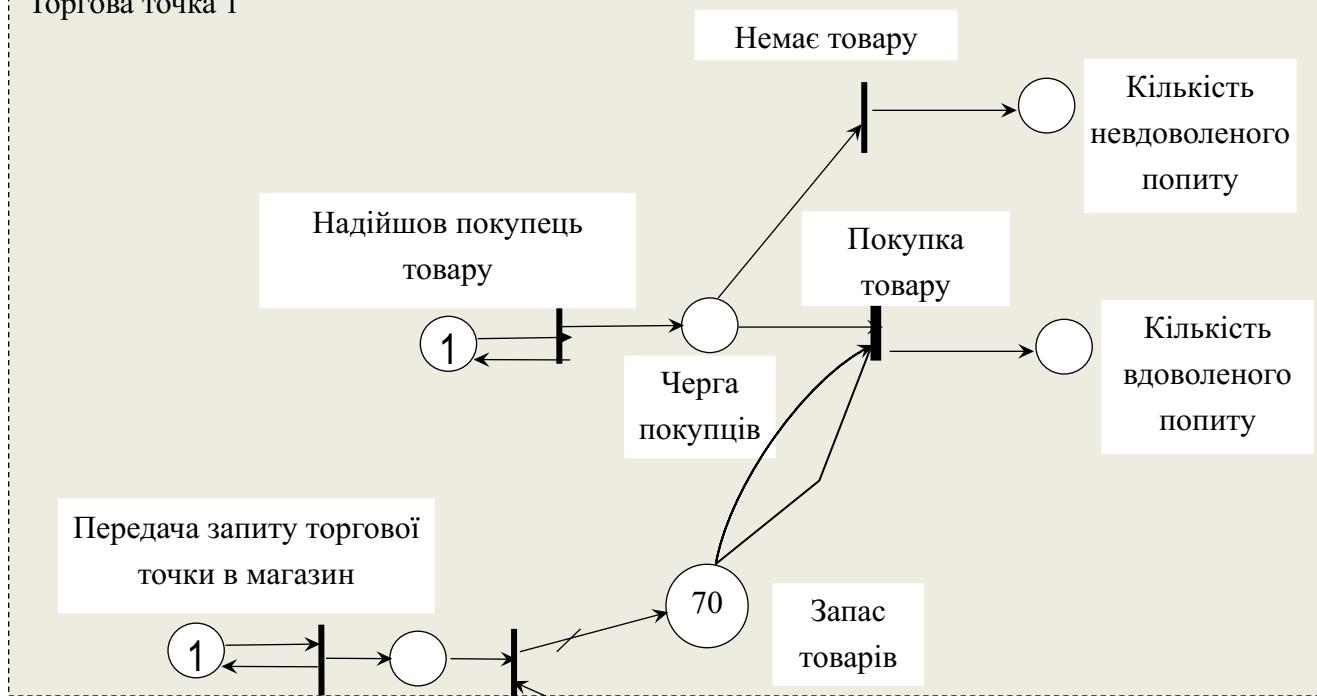


Зв'язки між Петрі-об'єктами



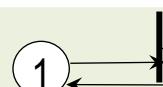
Приклад «Оптовий магазин»

Торгова точка 1



Виробництво та доставка товарів

Розміщення замовлення на фабриці



Приклад «Оптовий магазин»

Петрі-об'єкт “Торгова точка”

Немає товару

Надійшов покупець
товару

1

Покупка
товару

Кількість
невдоволеного
попиту

Черга
покупців

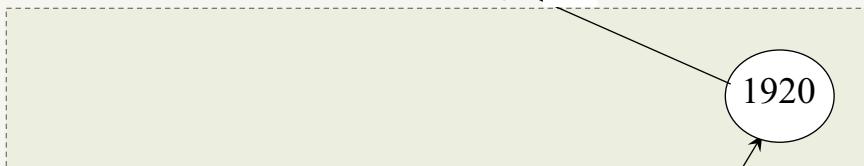
Кількість
вдоволеного
попиту

Передача запиту торгової
точки в магазин

1

Запас
товарів

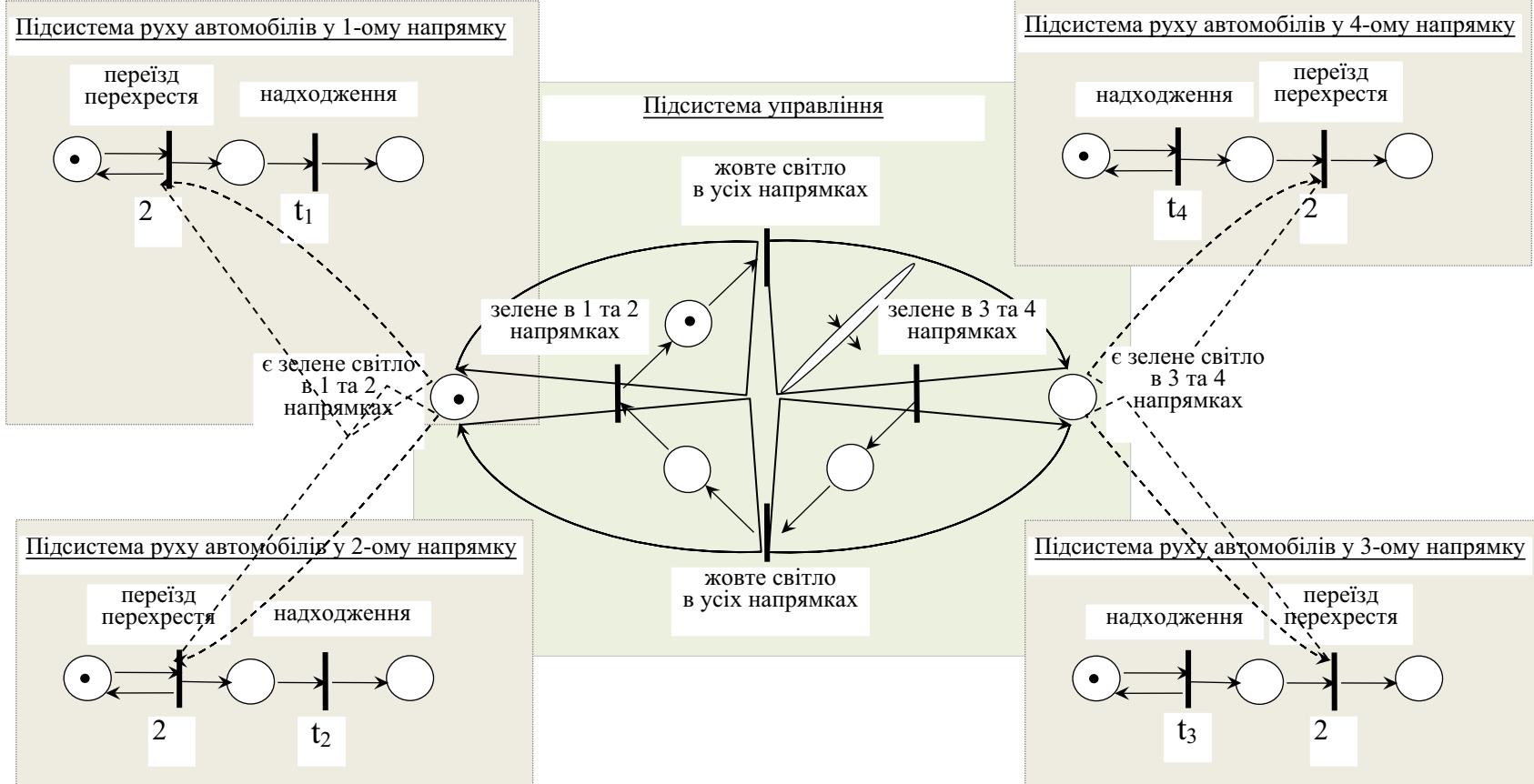
70



Петрі-об'єкт “Виробництво
та доставка товарів”

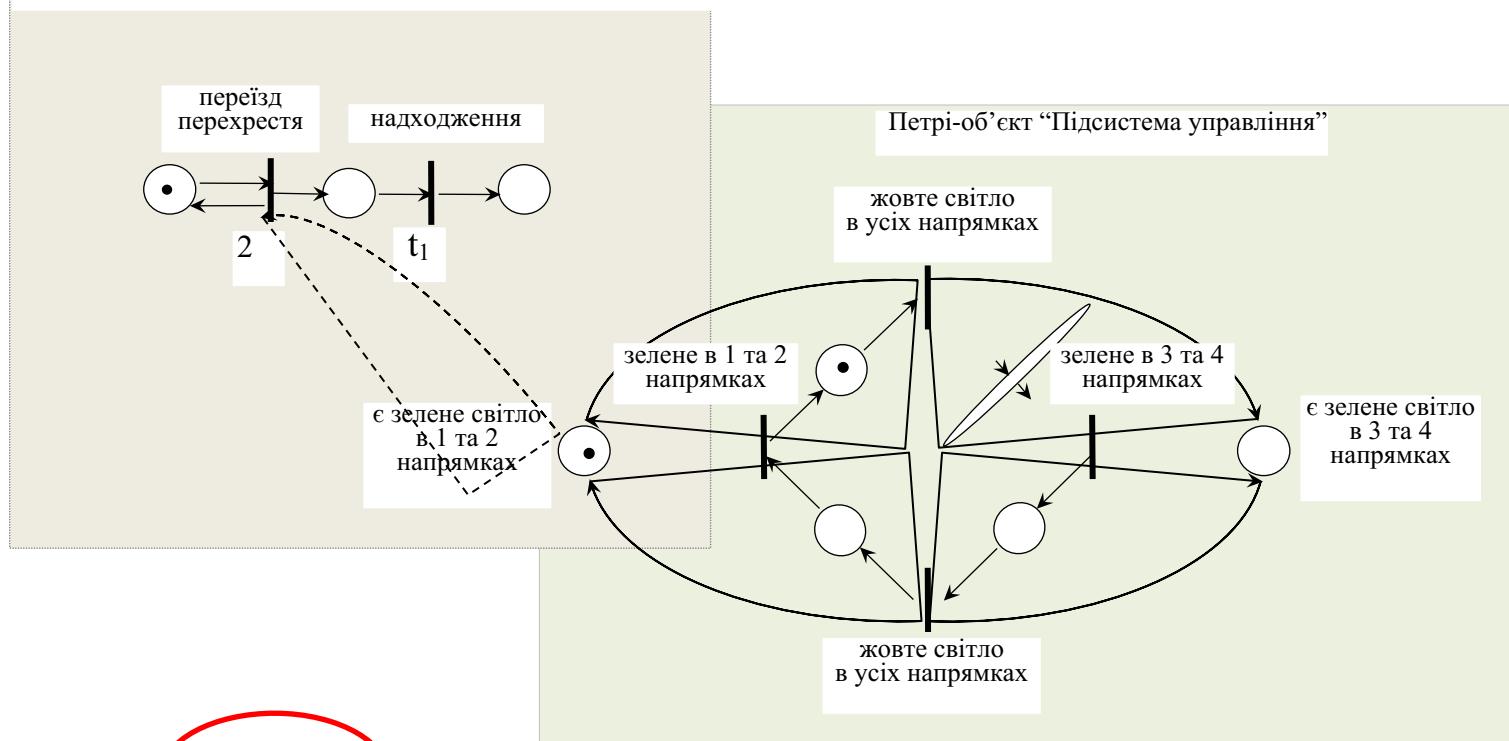


Приклад «Регульоване перехрестя»



Приклад «Регульоване перехрестя»

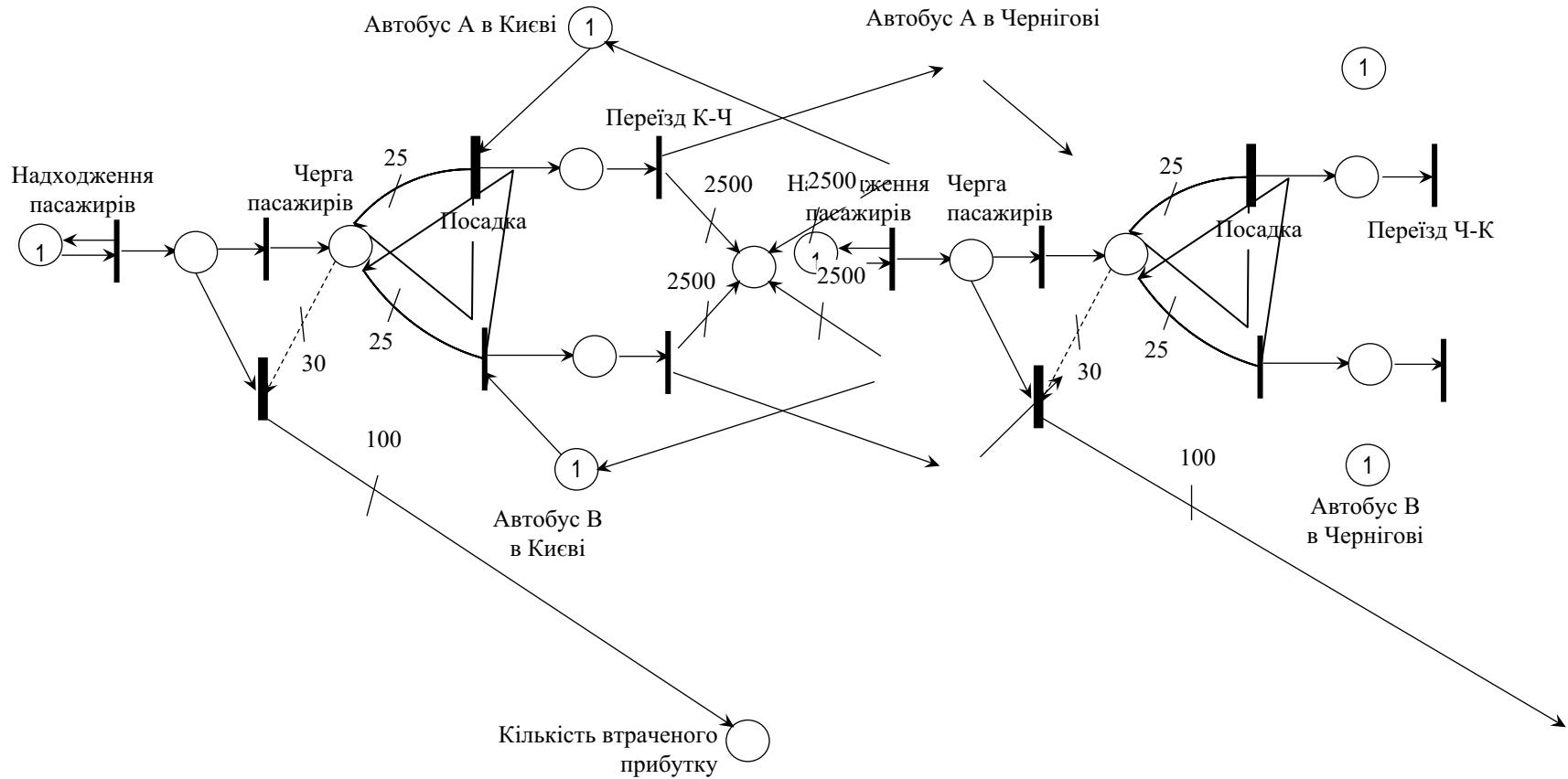
Петрі-об'єкт “Підсистема руху автомобілів у j-ому напрямку”



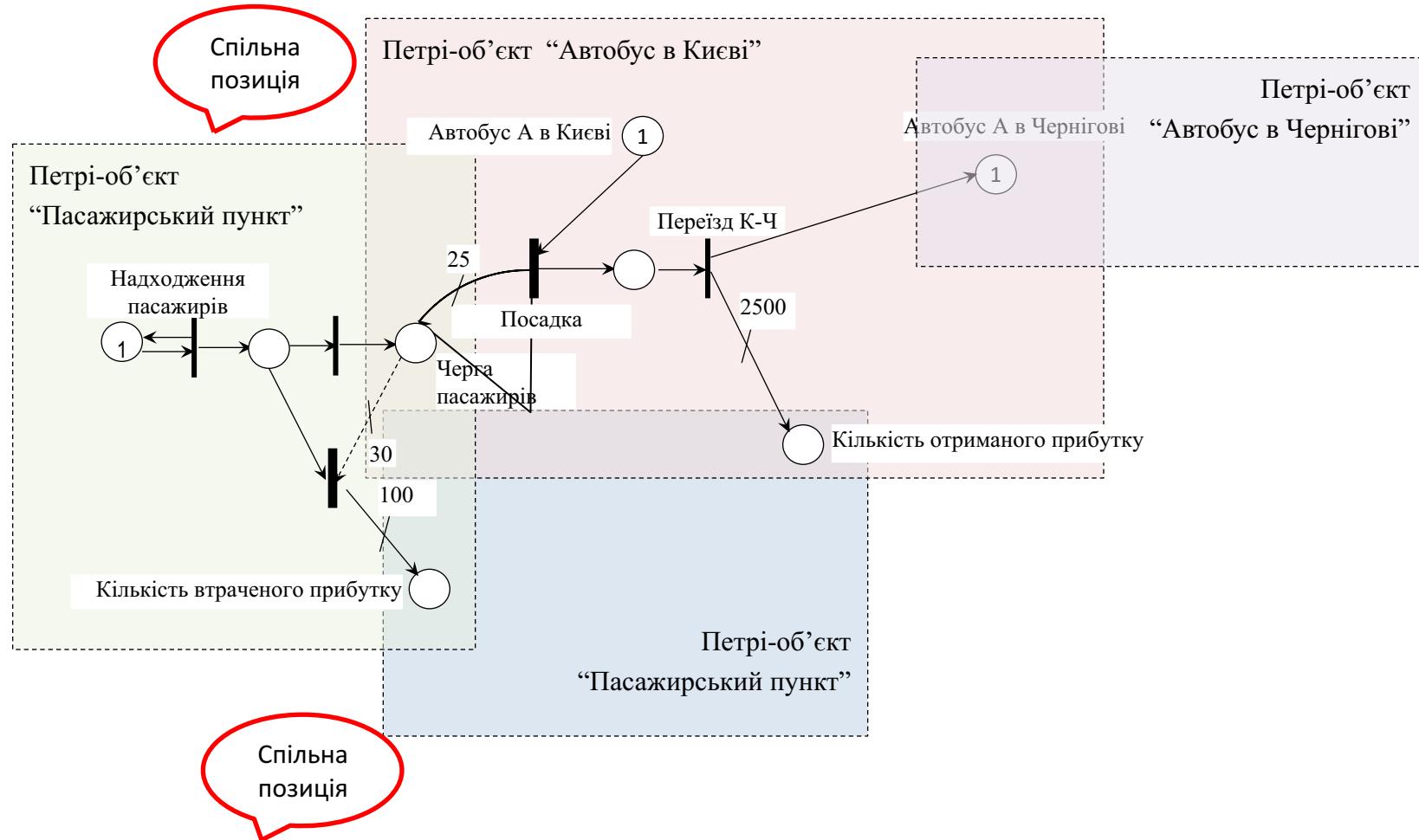
Спільна
позиція

Спільна
позиція

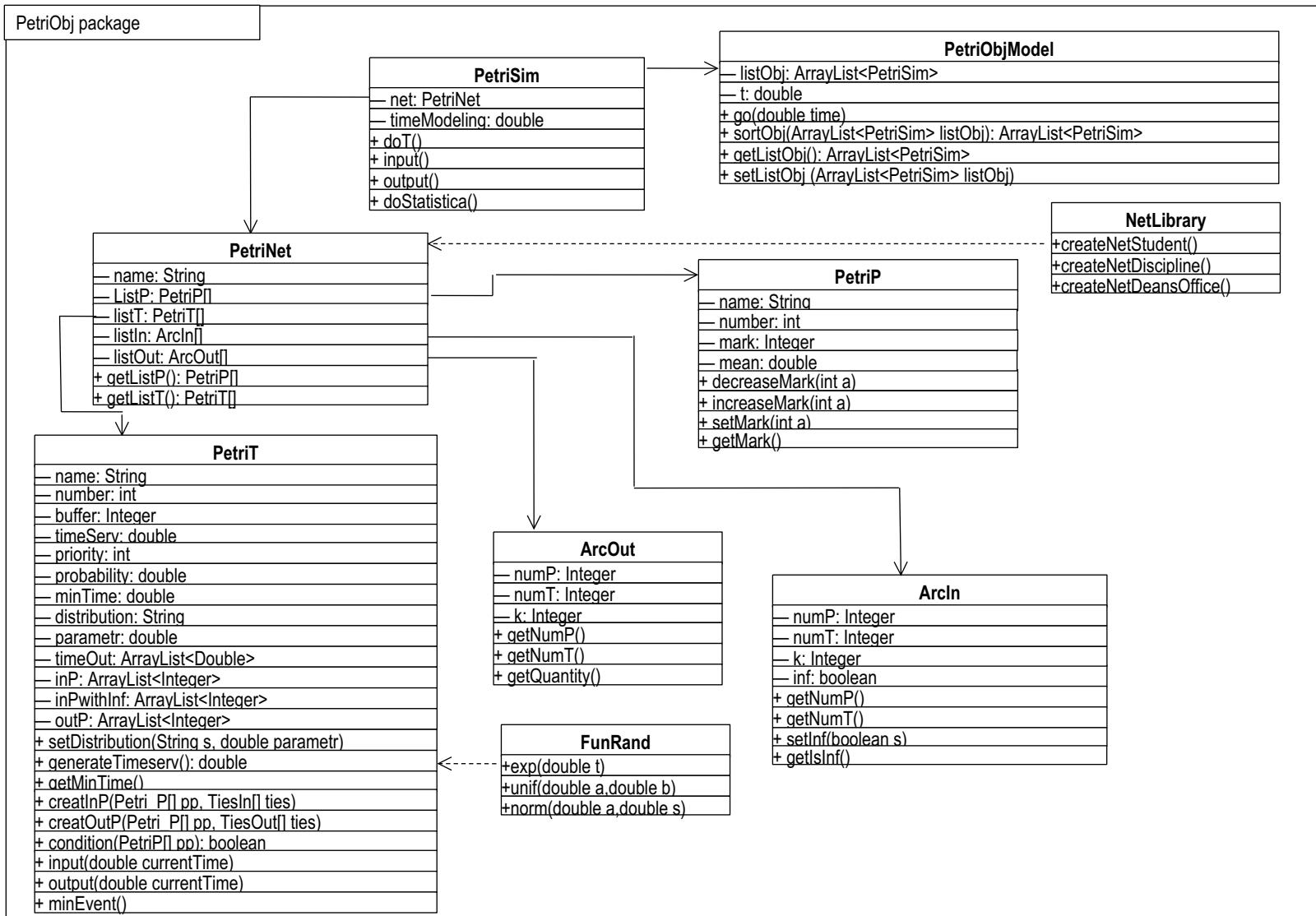
Приклад «Маршрутки»



Приклад «Маршрутки»



Бібліотека класів Петрі-об'єктного моделювання



Програмне забезпечення Петрі-об'єктного моделювання

<https://github.com/Stetsenkola/PetriObjModelPaint>

ЛЕКЦІЯ

Теоретичні основи Петрі-об'єктного моделювання систем

Стеценко И.В. Теоретические основы Петри-объектного моделирования систем / И.В. Стеценко // Математичні машини і системи.– Київ, 2011. - №4. – С.136-148.

Твердження 1

Петрі-об'єктна модель описується стохастичною мережею Петрі, що є об'єднанням мереж Петрі-об'єктів, з яких вона складається:

$$ModelNet = \bigcup_{\tilde{N}} \tilde{N}$$

де $\tilde{N} = (\mathbf{P}_N, \mathbf{T}_N, \tilde{\mathbf{A}}_N, \tilde{\mathbf{W}}_N, \mathbf{K}_N, \mathbf{I}_N, \mathbf{R}_N)$

$$\mathbf{P}_N$$

$$\mathbf{P}_N = \bigcup_N (\cdot \mathbf{T}_N \cup \mathbf{T}_N \cdot)$$

$$\mathbf{T}_N^\bullet = \bigcup_{T \in \mathbf{T}_N} T^\bullet = \{P \in \mathbf{P} \mid \exists T \in \mathbf{T} : \exists(T, P) \in \tilde{\mathbf{A}}_N\}$$

$$\tilde{\mathbf{A}}_N = \mathbf{A}_N \cup \mathbf{U}_N \quad \tilde{\mathbf{W}}_N = \mathbf{W}_N \cup \mathbf{w}_N$$

$$U_N = \{(T, P) \mid T \in \mathbf{T}_k, P \in \mathbf{P}_l, w_{k,l} > 0\}$$

- множина дуг Петрі-об'єкта, що з'єднує його з іншими об'єктами через ініціалізацію подій

Наслідок. Петрі-об'єктная модель є обчислюваною.

Тверждення 2

Перетворення D^+ мережі Петрі-об'єктної моделі $\bigcup_{\tilde{N}} \tilde{N}$

еквівалентно перетворенню D^+ мереж Петрі-об'єктів

$$\tilde{N} = (\mathbf{T}_N^\bullet, \mathbf{T}_N, \tilde{\mathbf{A}}_N, \tilde{\mathbf{W}}_N, \mathbf{K}_N, \mathbf{I}_N, \mathbf{R}_N)$$

Наслідок. Стан Петрі-об'єктної моделі, який є результатом виходу маркерів з переходів мережі Петрі-об'єктної моделі, описується станом її Петрі-об'єктів:

$$\mathbf{S}^+(t_n) = D^+(\mathbf{S}(t_{n-1})) = \begin{pmatrix} D^+(\tilde{\mathbf{S}}_1(t_{n-1})) \\ \dots \\ D^+(\tilde{\mathbf{S}}_N(t_{n-1})) \\ \dots \\ D^+(\tilde{\mathbf{S}}_L(t_{n-1})) \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{S}}_1^+(t_n) \\ \dots \\ \tilde{\mathbf{S}}_N^+(t_n) \\ \dots \\ \tilde{\mathbf{S}}_L^+(t_n) \end{pmatrix}$$

Твердження 3

Перетворення D^- мережі Петрі-об'єктної моделі $\bigcup_{\tilde{N}} \tilde{N}$

еквивалентно перетворенню D^- мережі Петрі-об'єктів

$$\tilde{N} = (\mathbf{T}_N^\bullet, \mathbf{T}_N, \tilde{\mathbf{A}}_N, \tilde{\mathbf{W}}_N, \mathbf{K}_N, \mathbf{I}_N, \mathbf{R}_N),$$

**для яких у випадку існування спільних позицій Петрі-об'єктів
вирішений конфлікт**

Наслідок. Стан Петрі-об'єктної моделі, який є результатом входу маркерів в переходи мережі Петрі-об'єктної моделі, описується станом її Петрі-об'єктів.

$$\mathbf{S}(t_n) = D^-(\mathbf{S}^+(t_n)) = \begin{pmatrix} D^-(\tilde{\mathbf{S}}_1^+(t_n)) \\ \dots \\ D^-(\tilde{\mathbf{S}}_N^+(t_n)) \\ \dots \\ D^-(\tilde{\mathbf{S}}_L^+(t_n))) \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{S}}_1(t_n) \\ \dots \\ \tilde{\mathbf{S}}_N(t_n) \\ \dots \\ \tilde{\mathbf{S}}_L(t_n) \end{pmatrix}$$

Рівняння станів Петрі-об'єктної моделі

Наслідок. Стан Петрі-об'єктної моделі в кожний момент часу описується станом її Петрі-об'єктів:

$$\mathbf{S}(t_n) = (D^-)^n (D^+(\mathbf{S}(t_{n-1}))) = (D^-)^n \begin{pmatrix} D^+(\tilde{\mathbf{S}}_1(t_{n-1})) \\ \dots \\ D^+(\tilde{\mathbf{S}}_N(t_{n-1})) \\ \dots \\ D^+(\tilde{\mathbf{S}}_L(t_{n-1})) \end{pmatrix} = \begin{pmatrix} (D^-)^n (D^+(\tilde{\mathbf{S}}_1(t_{n-1}))) \\ \dots \\ (D^-)^n (D^+(\tilde{\mathbf{S}}_N(t_{n-1}))) \\ \dots \\ (D^-)^n (D^+(\tilde{\mathbf{S}}_L(t_{n-1}))) \end{pmatrix}$$

**Рівняння станів
Петрі-об'єктної
моделі**

Алгоритм імітації Петрі-об'єктної моделі

- Формувати список Петрі-об'єктів;
- Виконати перетворення (метод `input()`);
- Доки не досягнутий момент завершення моделювання
 - Просунути час в момент найближчої події;
 - визначити список конфліктних об'єктів та вибрати об'єкт із списку конфліктних об'єктів;
 - для вибраного об'єкта виконати перетворення (методи `output()`, `input()`, `doT()`) ;
 - для всіх інших об'єктів виконати перетворення (метод `input()`);
- Вивести результати моделювання.

Матричні рівняння станів Петрі-об'єктної моделі

Матричні рівняння Петрі-об'єктів $\tilde{N} = (\mathbf{T}_N^+, \mathbf{T}_N, \tilde{\mathbf{A}}_N, \tilde{\mathbf{W}}_N, \mathbf{K}_N, \mathbf{I}_N, \mathbf{R}_N)$

$$\mathbf{i}_N(t_n) = \mathbf{i}_N(t_0) + \mathbf{a}_N \cdot \tilde{\mathbf{a}}_N(t_n)$$

$$\mathbf{c}_N(t_n) = -\mathbf{v}_N(t_n) + \mathbf{v}_N(t_0) + \tilde{\mathbf{a}}_N(t_n)$$

$\mathbf{i}_N(t) = \tilde{\mathbf{M}}(t) + \mathbf{a}_N^+ \cdot \mathbf{v}_N(t)$ - вектор розширеного маркірування Петрі-об'єкта

$\mathbf{a}_N = \mathbf{a}_N^+ - \mathbf{a}_N^-$ - матриця інцидентності

$\mathbf{v}_N(t)$ - вектор кількості активних каналів переходів Петрі-об'єкта

$\tilde{\mathbf{a}}_N(t_n)$ - вектор кількості входів в переход Петрі-об'єкта за період часу $[t_0, t]$

$\mathbf{c}_N(t_n)$ вектор кількості виходів з переходів Петрі-об'єкта за період часу $[t_0, t]$

Матричні рівняння станів Петрі-об'єктної моделі

$$\dot{\mathbf{i}}(t) = \|\dot{\mathbf{i}}_P(t)\| \quad \forall P \in \mathbf{P}_N \quad [\dot{\mathbf{i}}(t)]_P = [\dot{\mathbf{i}}_N(t)]_P$$

$$\mathbf{v}(t) = \|v_T(t)\| \quad \forall T \in \mathbf{T}_N \quad [v(t)]_T = [v_N(t)]_T \quad \mathbf{a}^- = \|a_{P,T}^-\| \quad \forall P \in \mathbf{P}_N \quad \forall T \in \mathbf{T}_N \quad [\mathbf{a}^-]_{P,T} = [\mathbf{a}_N^-]_{P,T}$$

$$\tilde{\mathbf{a}}(t) = \|\gamma_T(t)\|: \quad \forall T \in \mathbf{T}_N \quad [\tilde{\mathbf{a}}(t)]_T = [\tilde{\mathbf{a}}_N(t)]_T \quad \mathbf{a}^+ = \|a_{P,T}^+\| \quad \forall P \in \mathbf{P}_N \quad \forall T \in \mathbf{T}_N \quad [\mathbf{a}^+]_{P,T} = [\mathbf{a}_N^+]_{P,T}$$

$$\dot{\mathbf{c}}(t) = \|\mathbf{h}_T(t)\|: \quad \forall T \in \mathbf{T}_N \quad [\dot{\mathbf{c}}(t)]_T = [\dot{\mathbf{c}}_N(t)]_T$$

Фундаментальні рівняння Петрі-об'єктної моделі:

$$\dot{\mathbf{i}}(t_n) = \dot{\mathbf{i}}(t_0) + \mathbf{a} \cdot \tilde{\mathbf{a}}(t_n)$$

$$\dot{\mathbf{c}}(t_n) = -\mathbf{v}(t_n) + \mathbf{v}(t_0) + \tilde{\mathbf{a}}(t_n)$$

Дослідження властивостей Петрі-об'єктної моделі

Наслідок Мережа Петрі

$$ModelNet = \bigcup_{\tilde{N}} \tilde{N}$$

Петрі-об'єктної моделі володіє властивістю обмеженості, якщо мережі Петрі

$$\tilde{N} = (\mathbf{T}_N^\bullet, \mathbf{T}_N, \tilde{\mathbf{A}}_N, \tilde{\mathbf{W}}_N, \mathbf{K}_N, \mathbf{I}_N, \mathbf{R}_N)$$

з усіх її Петрі-об'єктів володіють властивістю обмеженості.

Наслідок. Мережа Петрі

$$ModelNet = \bigcup_{\tilde{N}} \tilde{N}$$

Петрі-об'єктної моделі є активною, якщо для довільного

$$t \geq t_0$$

хоч одна мережа Петрі

$$\tilde{N} = (\mathbf{T}_N^\bullet, \mathbf{T}_N, \tilde{\mathbf{A}}_N, \tilde{\mathbf{W}}_N, \mathbf{K}_N, \mathbf{I}_N, \mathbf{R}_N)$$

з усіх її Петрі-об'єктів є активною.

Аналіз обчислюваної складності алгоритму

Обчислювальна складність
стохастичної мережі Петрі

$$O(V^2 |\mathbf{T}| \cdot \text{time} \cdot (|\mathbf{T}| \cdot \left(\underset{T \in \mathbf{T}}{\text{mean}} |T^\bullet| + \underset{T \in \mathbf{T}}{\text{mean}} |T| \right) + V) + K^2 (\mathbf{T}) + K(\mathbf{T}) + \underset{T \in \mathbf{T}}{\text{mean}} |T| + V))$$

Обчислювальна складність Петрі-
об'єктної моделі

$$O(V^2 |\mathbf{T}| \cdot \text{time} \cdot (|\mathbf{T}|/q \cdot \left(\underset{T \in \mathbf{T}}{\text{mean}} |T^\bullet| + \underset{T \in \mathbf{T}}{\text{mean}} |T| \right) + V) + K^2 (\mathbf{T}/q) + K(\mathbf{T}/q) + \underset{T \in \mathbf{T}}{\text{mean}} |T| + V + \frac{1}{V} (q^2 + q)))$$

q - кількість об'єктів

$$\begin{aligned}\underset{T \in \mathbf{T}}{\text{mean}} |T| &= O(1) \\ \underset{T \in \mathbf{T}}{\text{mean}} |T^\bullet| &= O(1) \\ V &= O(1)\end{aligned}$$



$$O(V^2 |\mathbf{T}| \cdot \text{time} \cdot (|\mathbf{T}| + K^2 (\mathbf{T}) + K(\mathbf{T})))$$

для стохастичної
мережі Петрі

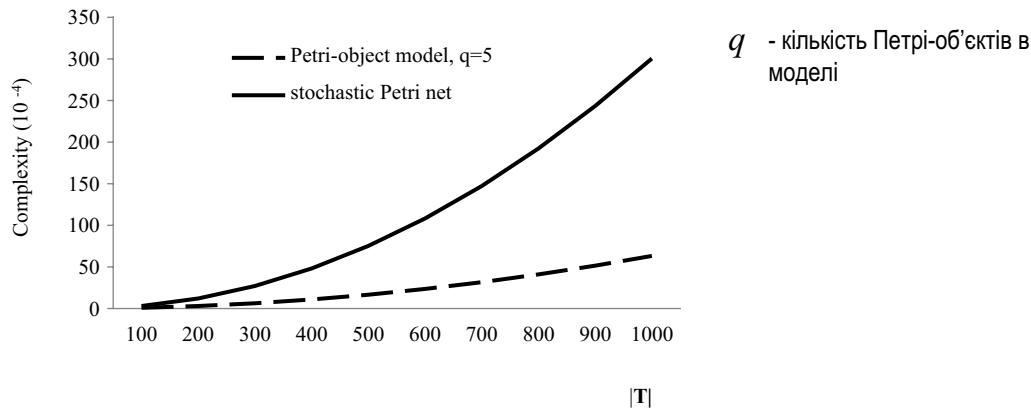
$$O(V^2 |\mathbf{T}| \cdot \text{time} \cdot (|\mathbf{T}|/q + K^2 (\mathbf{T}/q) + K(\mathbf{T}/q) + q^2 + q))$$

для Петрі-
об'єктної
моделі

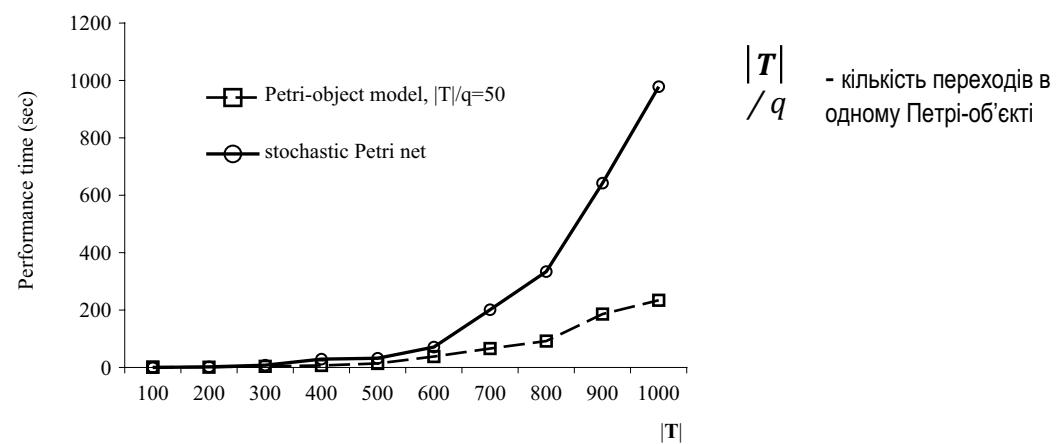
Обчислювальна складність Петрі-об'єктної моделі

Stetsenko I.V., Dyfuchyn A., Dorosh V.I. Petri-Object Simulation: Software Package and Complexity. Proceedings of the 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2015). Warsaw (Poland), 2015. P.381-385.

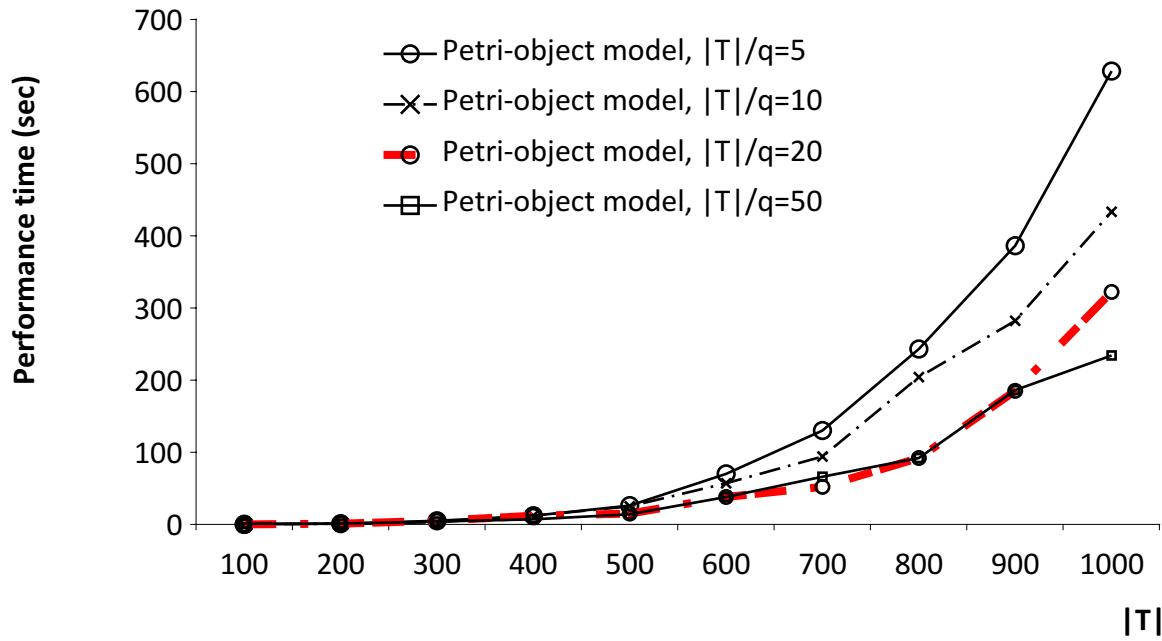
Обчислювальна складність Петрі-об'єктної моделі та стохастичної мережі Петрі, отримана за математичними формулами



Обчислювальна складність Петрі-об'єктної моделі та стохастичної мережі Петрі, отримана за результатами експериментальних досліджень



Експериментальне дослідження складності алгоритму в залежності від складності одного об'єкта



ЛЕКЦІЯ 11

**Програмне забезпечення
Петрі-об'єктного моделювання систем**

Алгоритм імітації Петрі-об'єктної моделі

- Формувати список Петрі-об'єктів;
- Виконати перетворення (метод `input()`);
- Доки не досягнутий момент завершення моделювання
 - Просунути час в момент найближчої події;
 - визначити список конфліктних об'єктів та вибрати об'єкт із списку конфліктних об'єктів;
 - для вибраного об'єкта виконати перетворення (методи `output()`, `input()`, `doT()`) ;
 - для всіх інших об'єктів виконати перетворення (метод `input()`);
- Вивести результати моделювання.

Аналіз обчислювальної складності алгоритму:

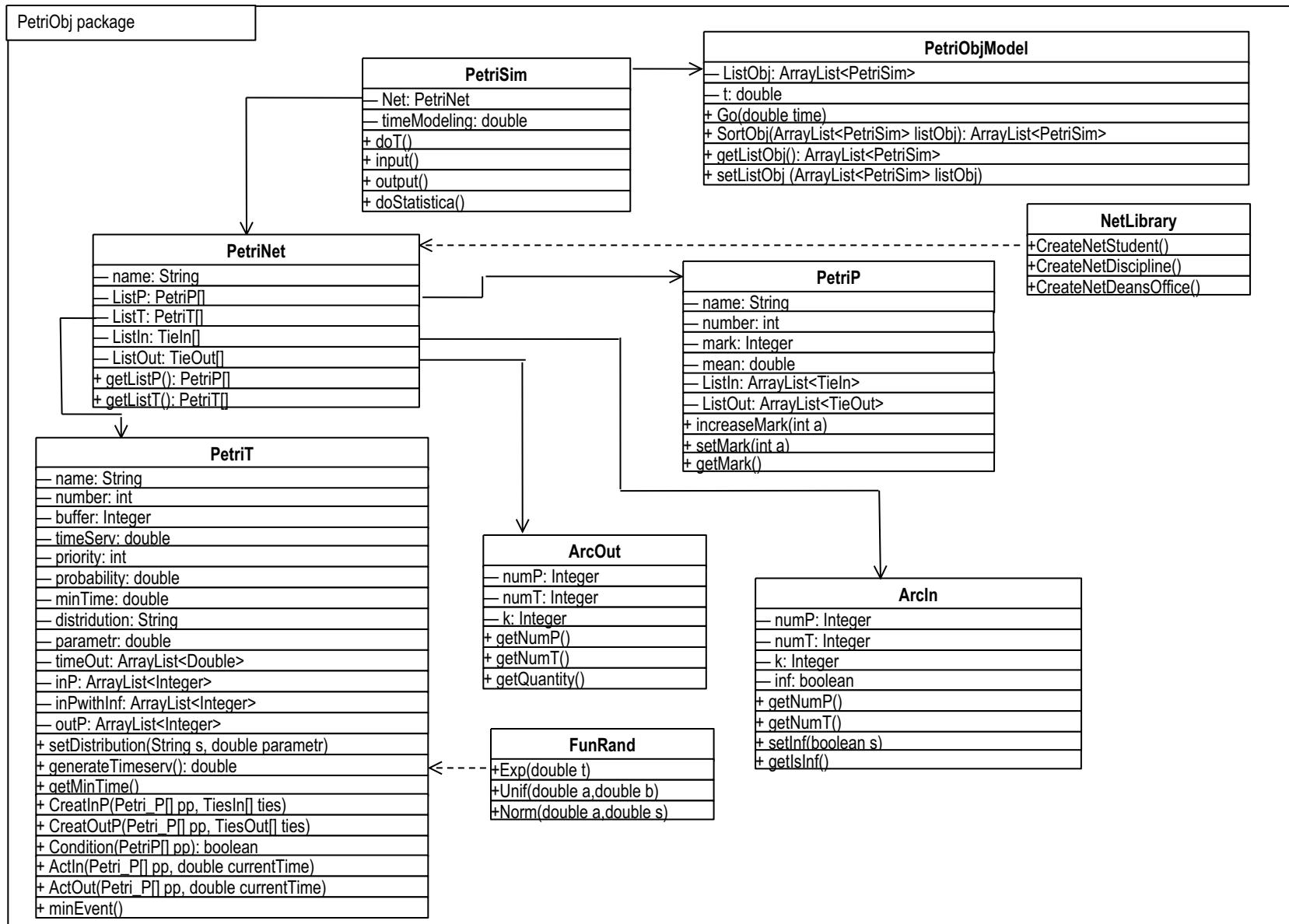
$$O(|\mathbf{T}|^2 \cdot V \cdot timeMod \cdot \left(\underset{T \in \mathbf{T}}{mean} |T^\bullet| + V + V \cdot |\mathbf{T}| \cdot \underset{T \in \mathbf{T}}{mean} |T| + V^2 \cdot |\mathbf{T}| + V \cdot K^2 \right))$$

Середня кількість активних каналів переходів

Середня кількість конфліктних переходів

Бібліотека класів PetriObj

<https://github.com/Stetsenkola/NetLibrary>



Документація

All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type			Method and Description
PetriSim			clone()
PetriT			doConflikt(java.util.ArrayList<PetriT> transitions) This method solves conflict between transitions given in parametr transitions
void			doStatistics() Calculates mean value of quantity of markers in places and quantity of active channels of transitions
void			doStatistics(double dt)
void			do() This method uses for describing other actions associated with transition markers output. Such as the output markers into the other Petri-object. The method is overridden in subclass.
void			eventMin() Determines the next event and its moment.
java.util.ArrayList<PetriT>			findActiveT() Finds the set of transitions for which the firing condition is true and sorts it on priority value
static java.util.Comparator<PetriSim>			getComparatorByNum()
static java.util.Comparator<PetriSim>			getComparatorByPriority()
double			getCurrentTime()
PetriT			getEventMin()
java.lang.String			getId()
java.util.ArrayList<PetriP>			getListPositionsForStatistica()
java.lang.String			getName()
PetriNet			getNet()

Метод go() об'єкту PetriObjModel

```
public void go(double timeModeling) {  
    double min;  
    this.setSimulationTime(timeModeling);  
    this.setCurrentTime(0.0);  
  
    getListObj().sort(PetriSim.getComparatorByPriority());  
    for (PetriSim e : getListObj()) {  
        e.input();  
    }  
    ArrayList<PetriSim> conflictObj = new ArrayList<>();  
    Random r = new Random();  
  
    while (this.getCurrentTime() < this.getSimulationTime()) {  
        conflictObj.clear();  
        min = getListObj().get(0).getTimeMin();  
  
        for (PetriSim e : getListObj()) {  
            if (e.getTimeMin() < min) {  
                min = e.getTimeMin();  
            }  
        }  
    }  
}
```

Метод go() об'єкту PetriObjModel

```
if (isStatistics() == true) {  
    for (PetriSim e : getListObj()) {  
        if (min > 0) {  
            if(min<this.getSimulationTime())  
                e.doStatistics((min - this.getCurrentTime()) / min);  
            else  
                e.doStatistics(  
                    (this.getSimulationTime() - this.getCurrentTime()) /  
                    this.getSimulationTime());  
        }  
    }  
    this.setCurrentTime(min);  
}
```

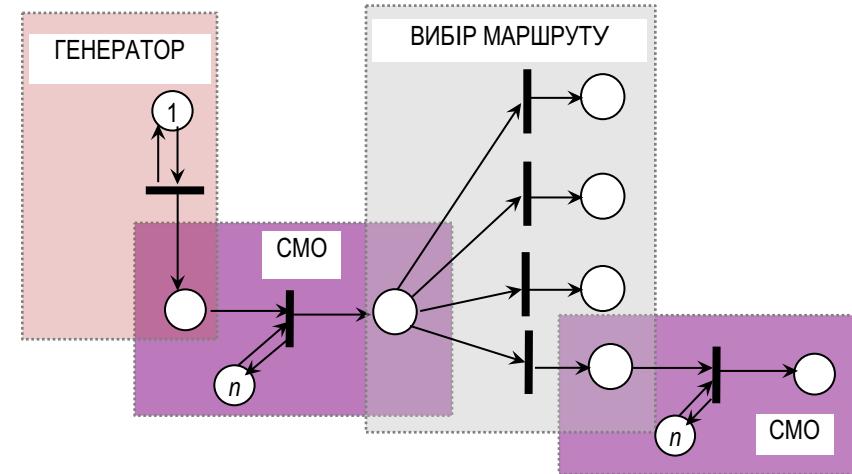
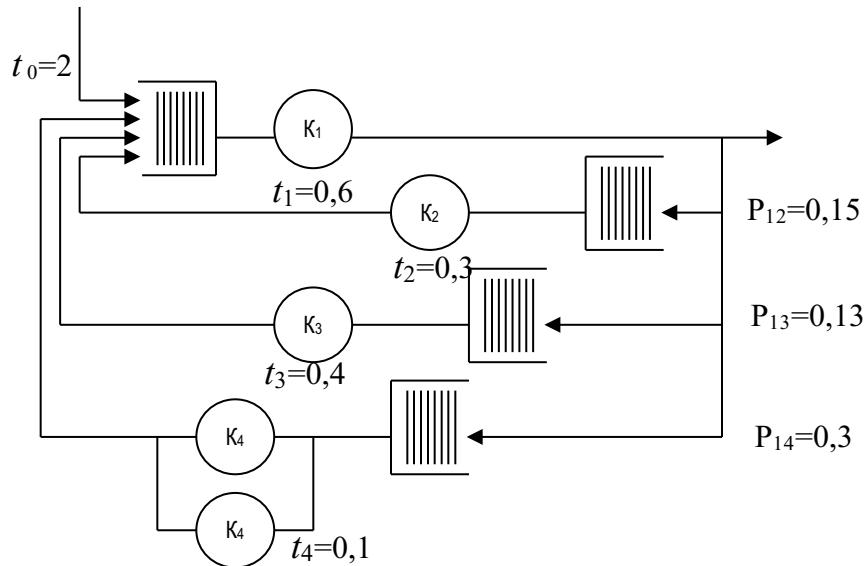
Метод go() об'єкту PetriObjModel

```
if (this.getCurrentTime() <= this.getSimulationTime()) {  
    for (PetriSim sim : getListObj()) {  
        if (this.getCurrentTime() == sim.getTimeMin()) {  
            conflictObj.add(sim);  
        }  
    }  
    int num;  
    int max;  
    if (conflictObj.size() > 1) { //вибір об'єкта, що запускається  
        max = conflictObj.size();  
        conflictObj.sort(PetriSim.getComparatorByPriority());  
        for (int i = 1; i < conflictObj.size(); i++) {  
            if (conflictObj.get(i).getPriority() <  
                conflictObj.get(i - 1).getPriority()) {  
                max = i - 1;  
                break;  
            }  
        }  
        if (max == 0) { . . . num = 0; }  
        else { . . . num = r.nextInt(max); }  
    } else { . . . num = 0; }  
}
```

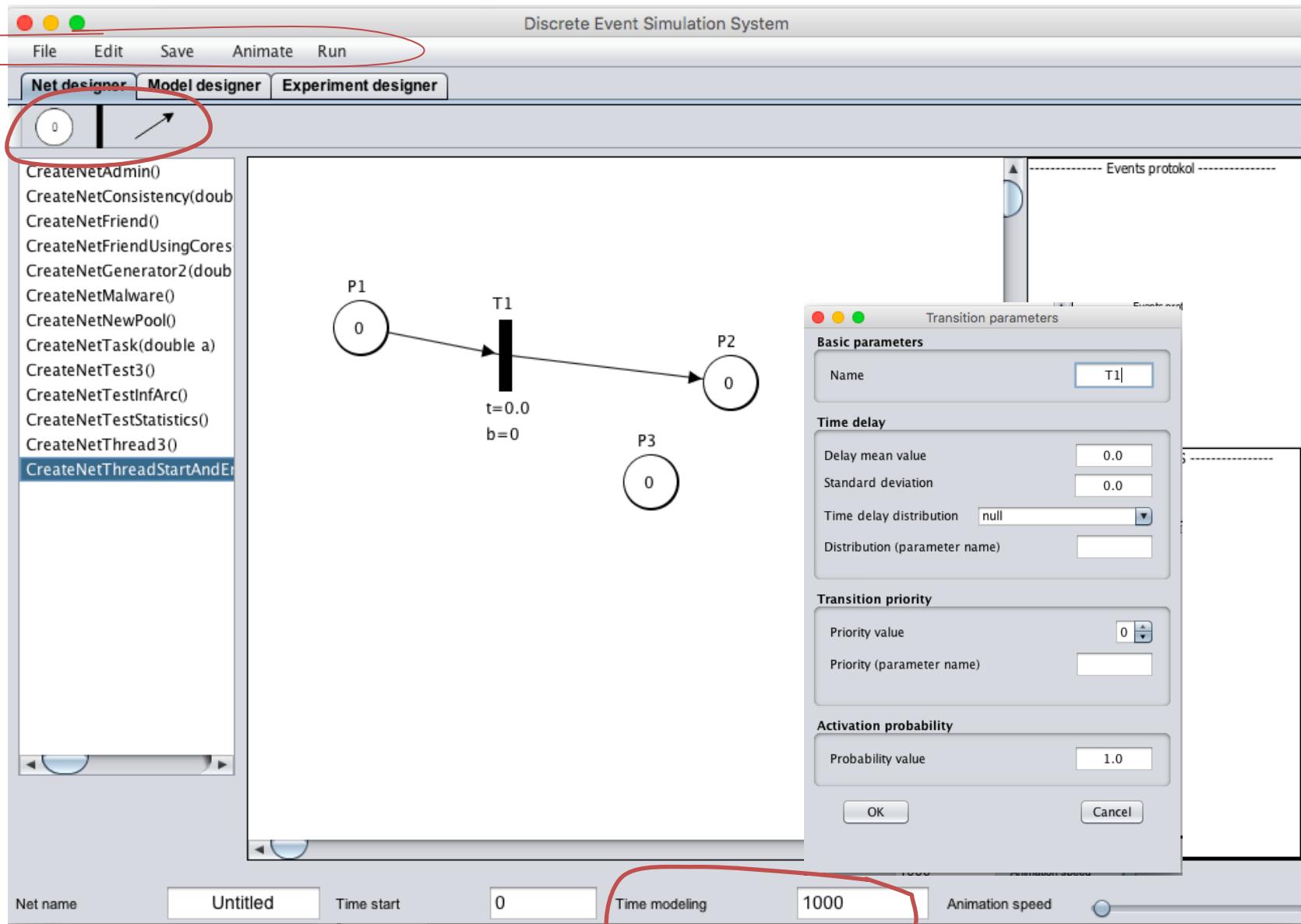
Метод go() об'єкту PetriObjModel

```
for (PetriSim sim: getListObj()) {  
    if (sim.getNumObj() == conflictObj.get(num).getNumObj()) {  
        sim.doT();  
        sim.output();  
    }  
}  
  
Collections.shuffle(getListObj());  
  
getListObj().sort(PetriSim.getComparatorByPriority());  
  
for (PetriSim e : getListObj()) {  
    e.input();  
}  
}  
}  
getListObj().sort(PetriSim.getComparatorByNum());  
}
```

Приклад розробки Петрі-об'єктної моделі з використанням бібліотеки PetriObjLib



Використання графічного редактора для розробки мережі Петрі



Конструювання мережі Петрі

```
public static PetriNet CreateNetSMOwithoutQueue(int numChannel, double timeMean,
String name) throws ExceptionInvalidTimeDelay, ExceptionInvalidNetStructure{
    ArrayList<PetriP> d_P = new ArrayList<>();
    ArrayList<PetriT> d_T = new ArrayList<>();
    ArrayList<ArcIn> d_In = new ArrayList<>();
    ArrayList<ArcOut> d_Out = new ArrayList<>();
    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;
}
```

Конструювання Петрі-об'єктів

```
ArrayList<PetriSim> list = new ArrayList<>();
list.add(new PetriSim(NetLibrary.CreateNetGenerator(2.0)));
list.add(new PetriSim(NetLibrary.CreateNetSMOwithoutQueue(1, 0.6, "First")));
list.add(new PetriSim(NetLibrary.CreateNetSMOwithoutQueue(1, 0.3, "Second")));
list.add(new PetriSim(NetLibrary.CreateNetSMOwithoutQueue(1, 0.4, "Third")));
list.add(new PetriSim(NetLibrary.CreateNetSMOwithoutQueue(2, 0.1, "Forth")));
list.add(new PetriSim(NetLibrary.CreateNetFork(0.15, 0.13, 0.3)));
```

Графічний редактор Петрі- об'єктної моделі

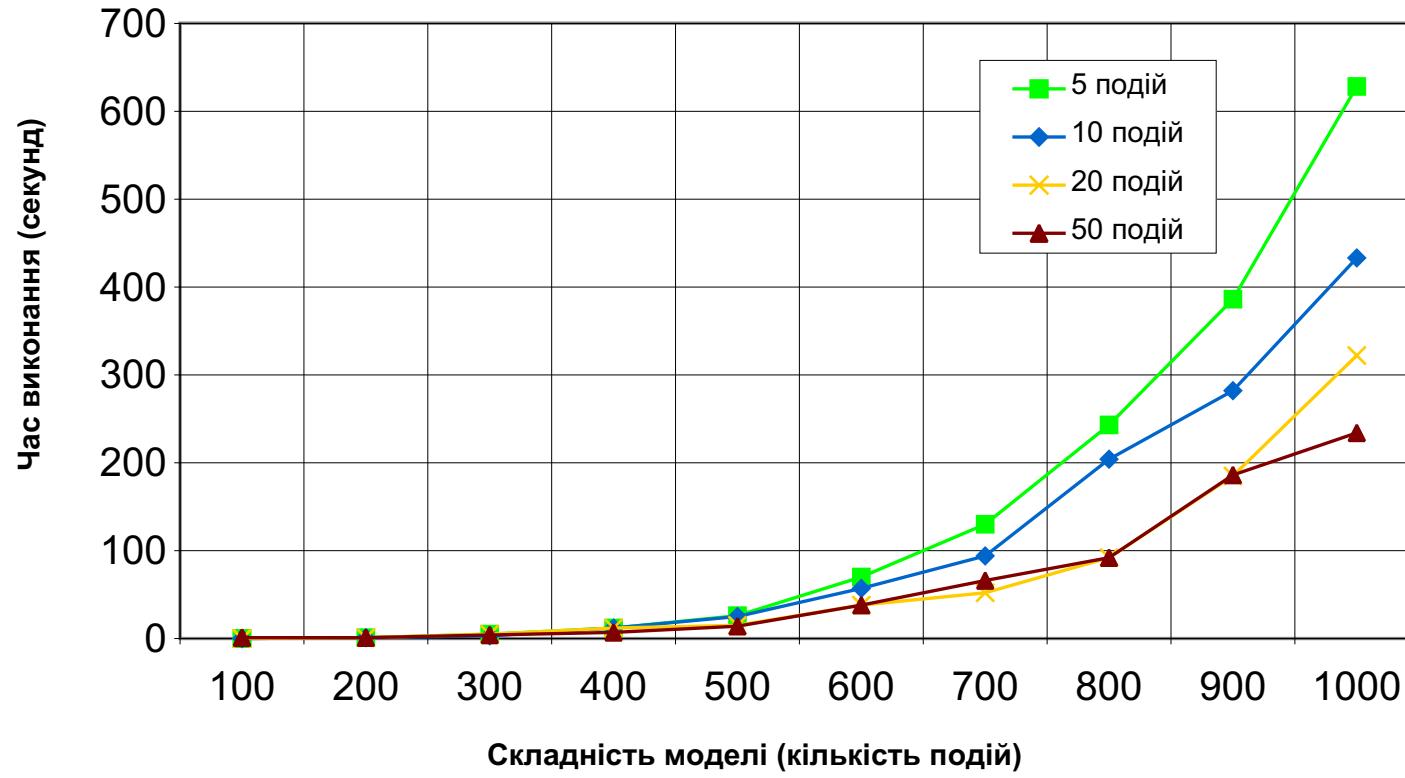
Конструювання Петрі-об'єктної моделі

```
public static PetriObjModel getModel() throws ExceptionInvalidTimeDelay,  
ExceptionInvalidNetStructure{  
    ArrayList<PetriSim> list = new ArrayList<>();  
    list.add(new PetriSim(NetLibrary.CreateNetGenerator(2.0)));  
    list.add(new PetriSim(NetLibrary.CreateNetSMOwithoutQueue(1, 0.6, "First")));  
    list.add(new PetriSim(NetLibrary.CreateNetSMOwithoutQueue(1, 0.3,  
"Second")));  
    list.add(new PetriSim(NetLibrary.CreateNetSMOwithoutQueue(1, 0.4, "Third")));  
    list.add(new PetriSim(NetLibrary.CreateNetSMOwithoutQueue(2, 0.1, "Forth")));  
    list.add(new PetriSim(NetLibrary.CreateNetFork(0.15, 0.13, 0.3)));  
  
    list.get(0).getNet().getListP()[1] = list.get(1).getNet().getListP()[0];  
    list.get(1).getNet().getListP()[2] = list.get(5).getNet().getListP()[0];  
    list.get(5).getNet().getListP()[1] = list.get(2).getNet().getListP()[0];  
    list.get(5).getNet().getListP()[2] = list.get(3).getNet().getListP()[0];  
    list.get(5).getNet().getListP()[3] = list.get(4).getNet().getListP()[0];  
    list.get(2).getNet().getListP()[2] = list.get(1).getNet().getListP()[0];  
    list.get(3).getNet().getListP()[2] = list.get(1).getNet().getListP()[0];  
    list.get(4).getNet().getListP()[2] = list.get(1).getNet().getListP()[0];  
    PetriObjModel model = new PetriObjModel(list);  
    return model;
```

Точність результатів моделювання

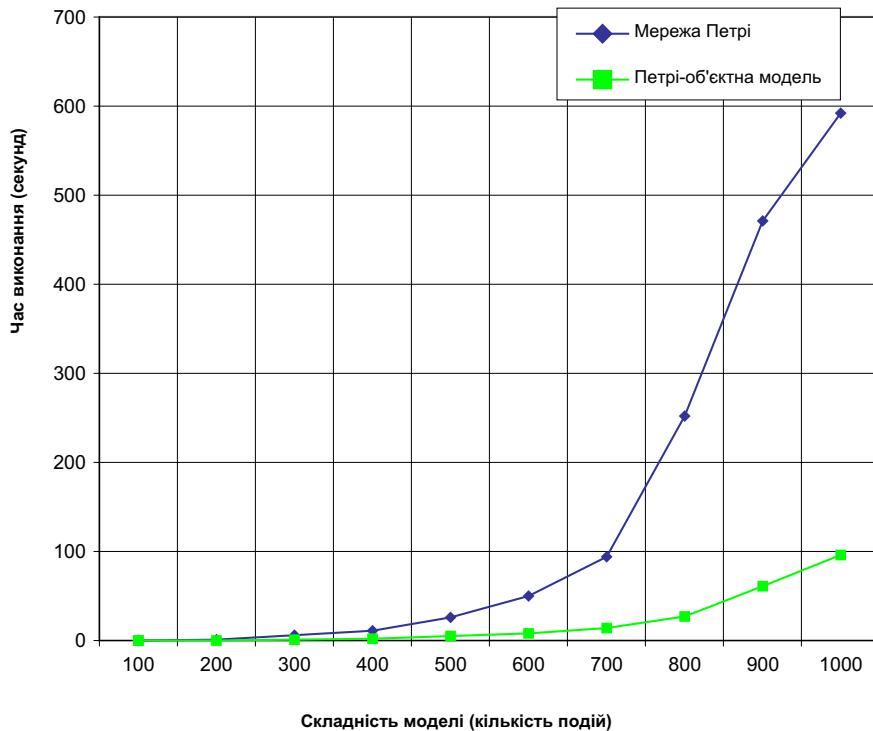
Результати аналітичного моделювання	Результати Петрі-об'єктного моделювання
Середня довжина черги СМО1 = 1,786	Середня довжина черги СМО1 = 1,766
Середня довжина черги СМО2 = 0,003	Середня довжина черги СМО2 = 0,0041
Середня довжина черги СМО3 = 0,004	Середня довжина черги СМО3 = 0,0035
Середня довжина черги СМО4 = 0,00001	Середня довжина черги СМО4 = 0,00001
Середня зайнятість пристройів СМО1 = 0,714	Середня зайнятість пристройів СМО1 = 0,714
Середня зайнятість пристройів СМО2 = 0,054	Середня зайнятість пристройів СМО2 = 0,054
Середня зайнятість пристройів СМО3 = 0,062	Середня зайнятість пристройів СМО3 = 0,065
Середня зайнятість пристройів СМО4 = 0,036	Середня зайнятість пристройів СМО4 = 0,035

Дослідження ефективності алгоритму імітації Петрі-об'єктної моделі при зростанні складності об'єктів

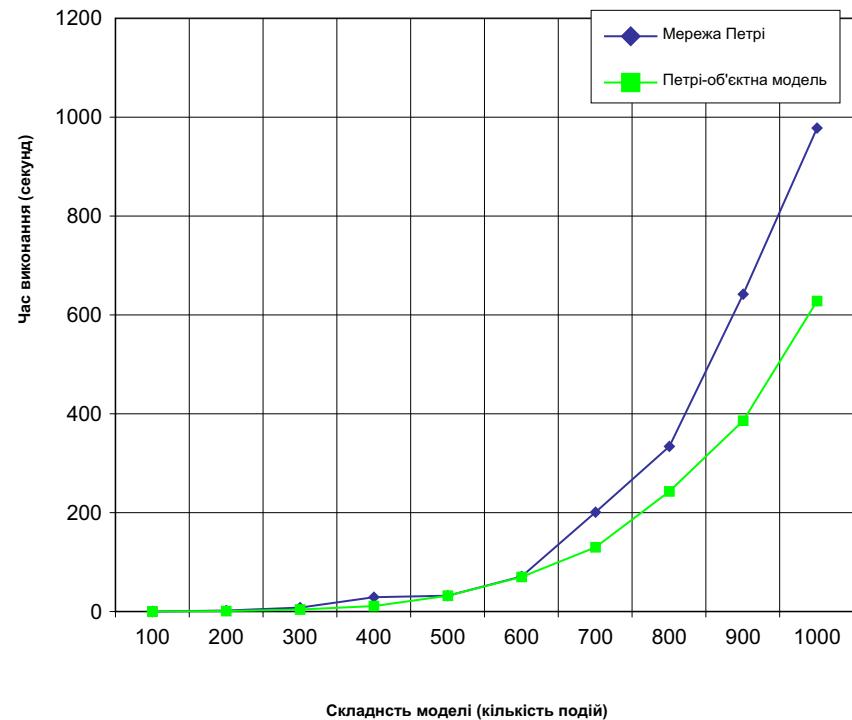


Дослідження ефективності алгоритму імітації Петрі-об'єктної моделі

а) переходи з детермінованими затримками



б) переходи зі стохастичними затримками



Петрі-об'єктна модель

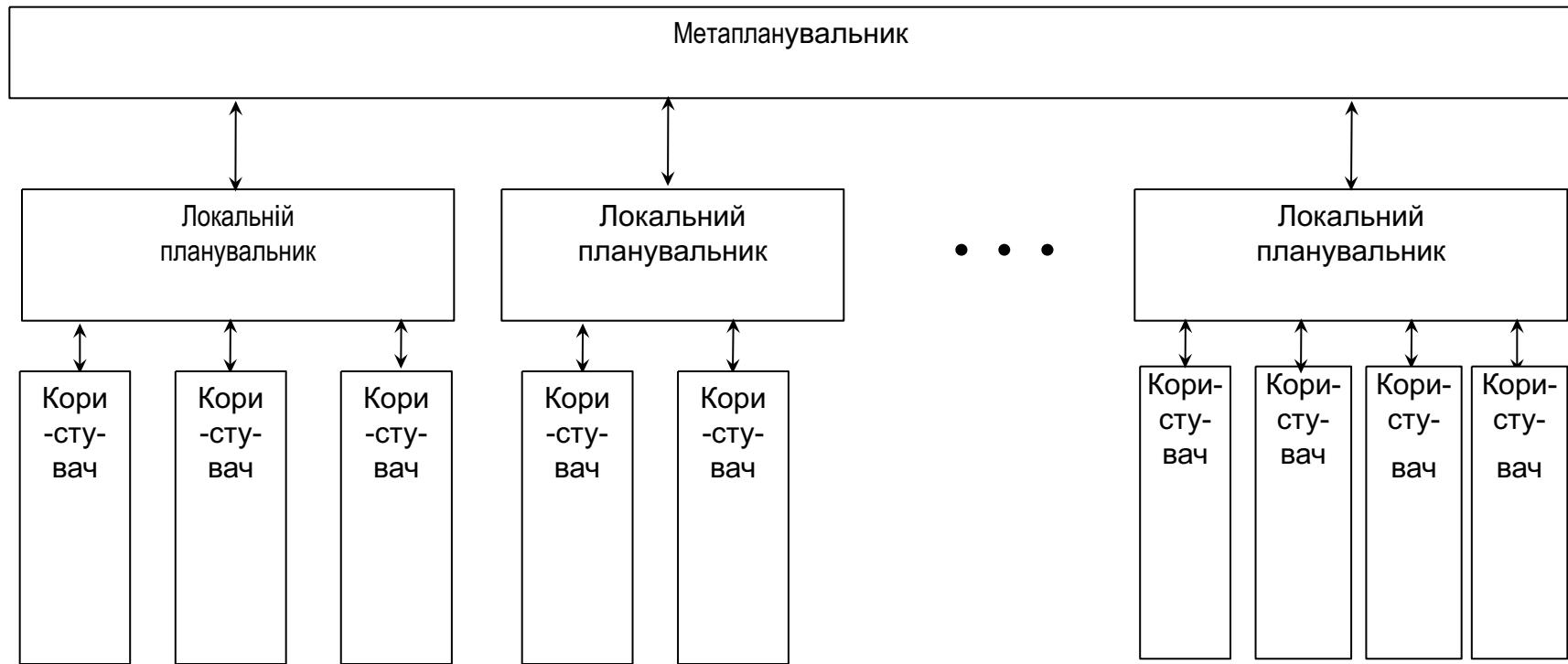
- це засіб формального опису систем, який:
- 1) має математичний опис, а значить, має високу ступінь абстракції та найбільш формалізований опис алгоритму імітації;
- 2) допускає використання не тільки імітаційних методів дослідження, але й аналітичні методи;
- 3) Надає можливість створювати моделі великих і складних систем з найменьшими витратами часу та зусиль;
- 4) основывается на ОРеменной стохастической сети Петри, а значит, допускает наиболее детализированное описание дискретно-событийных процессов функционирования;
- 5) основывается на объектно-ориентированной технологии, а значит, допускает моделирование структуры больших систем и совместимость с другими информационными технологиями.

Практичне застосування Петрі-об'єктного моделювання

- Моделювання систем управління (навчальний процес, транспортні системи, грід-системи)
- Моделювання паралельних обчислень (багатопоточність, синхронізація, дослідження: дедлоків, конфліктів доступу до спільної пам'яті)
- Моделювання процесів управління організаціями та підприємствами (процесно-орієнтований підхід до управління)
- Розподілене моделювання (модель великої системи будується за участю колективу розробників). Це ідея і концепція, якої притримувались при розробці ООП
- Петрі-процесор (Петрі-машина) і «нова парадигма обчислень»

Петрі-об'єктна модель системи управління розподіленими обчислювальними ресурсами

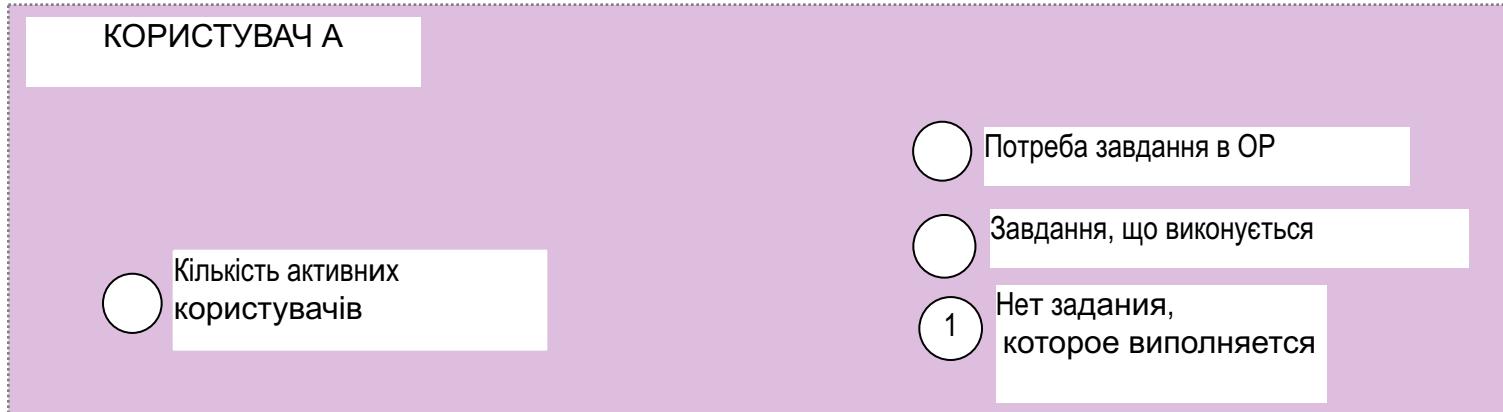
Архітектура двухрівневої грід-системи



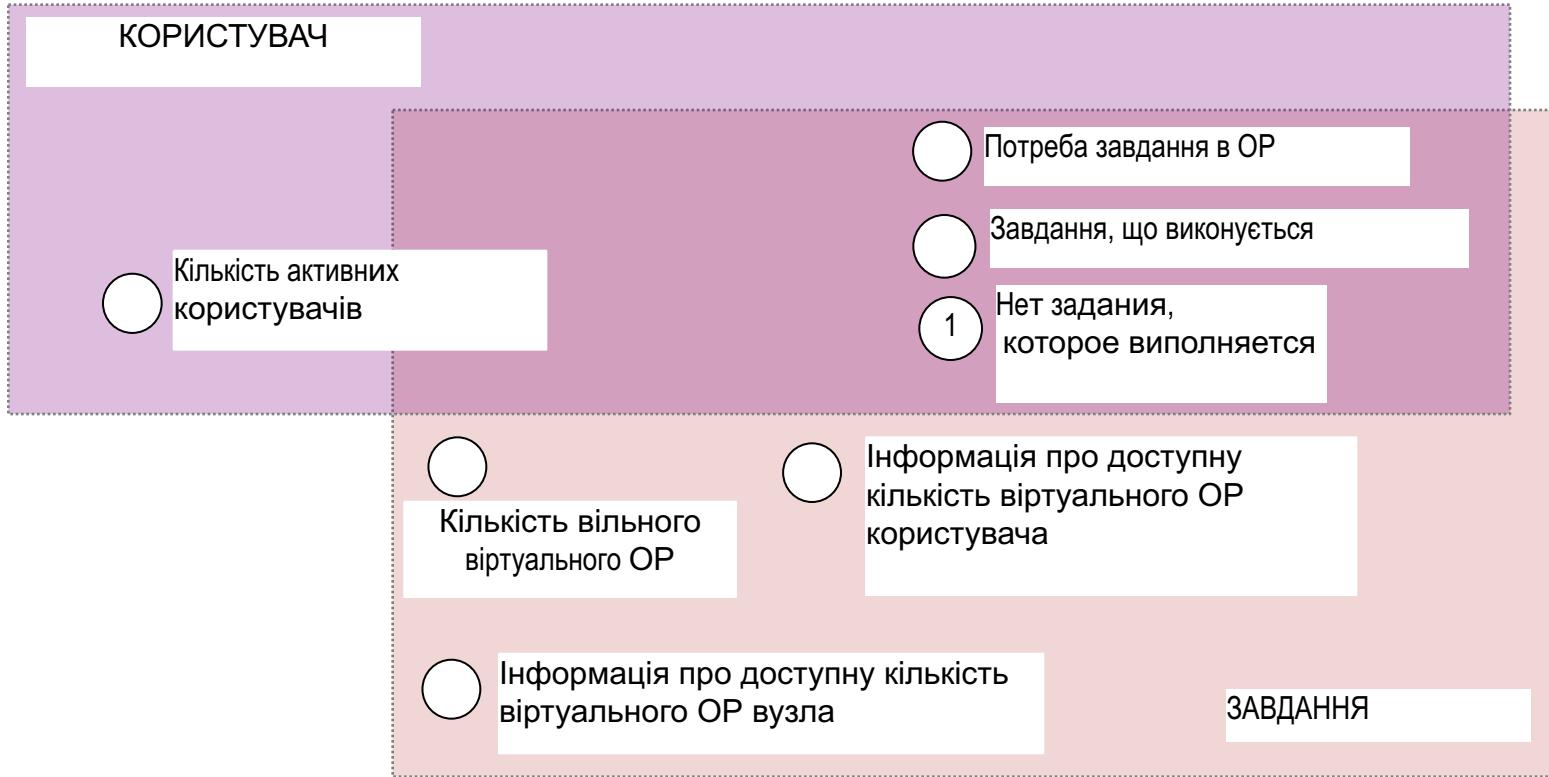
Частина доступного ресурсу:

$$x_i = \frac{p_i}{\sum_{i \in A} p_i} \quad \sum_i x_i = 1$$

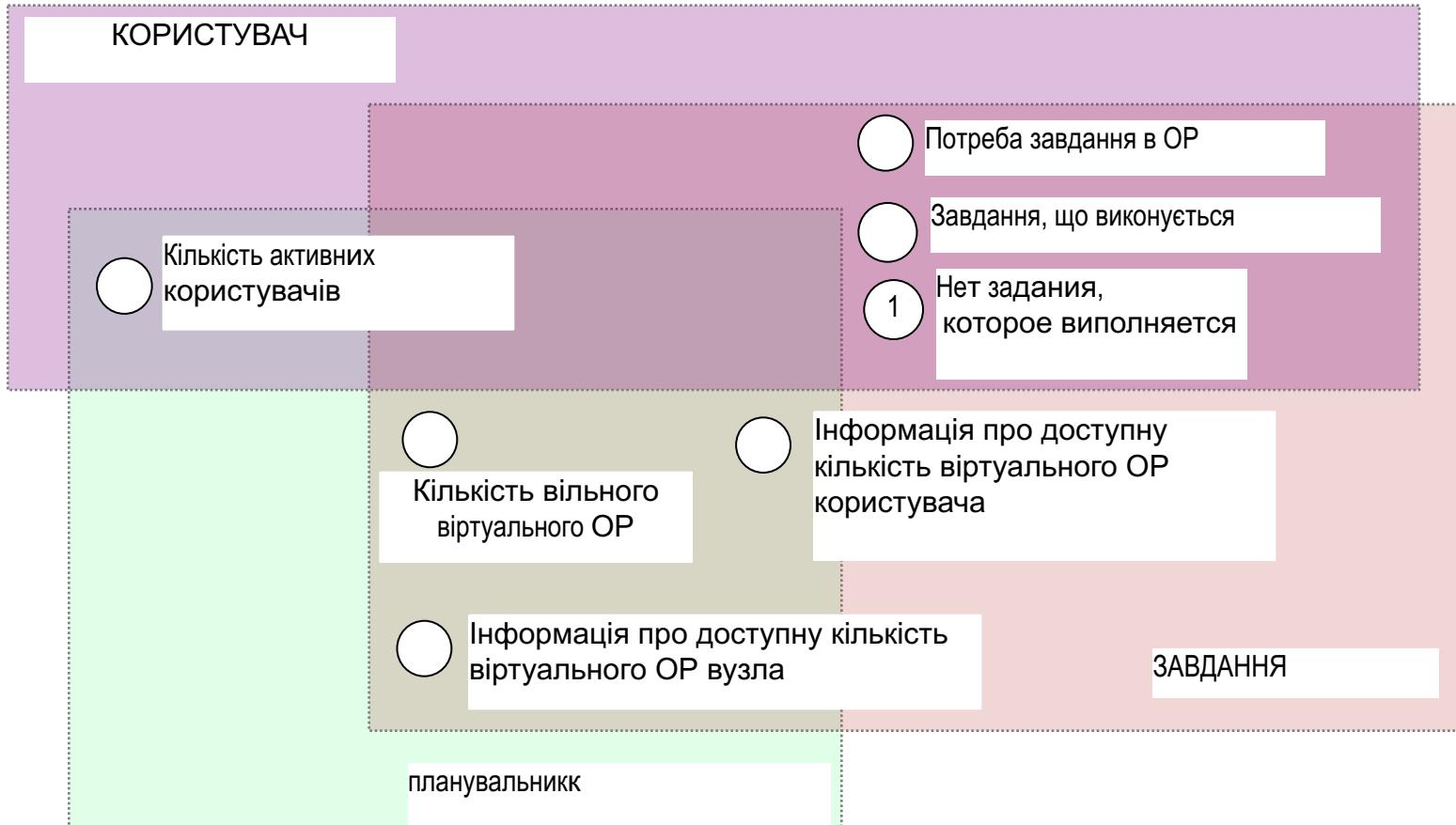
Петрі-об'єктна модель системи управління розподіленими обчислювальними ресурсами



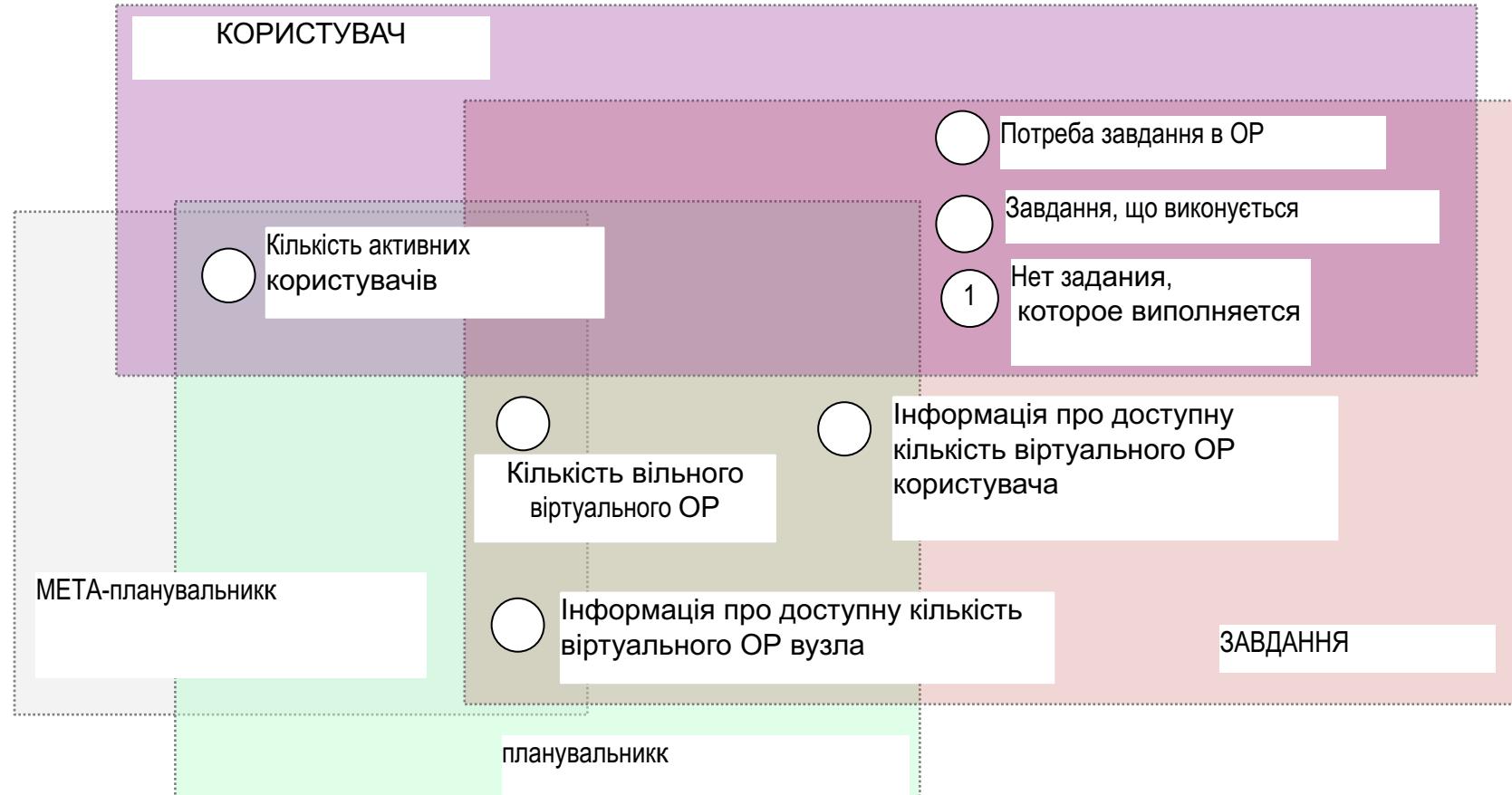
Петрі-об'єктна модель системи управління розподіленими обчислювальными ресурсами



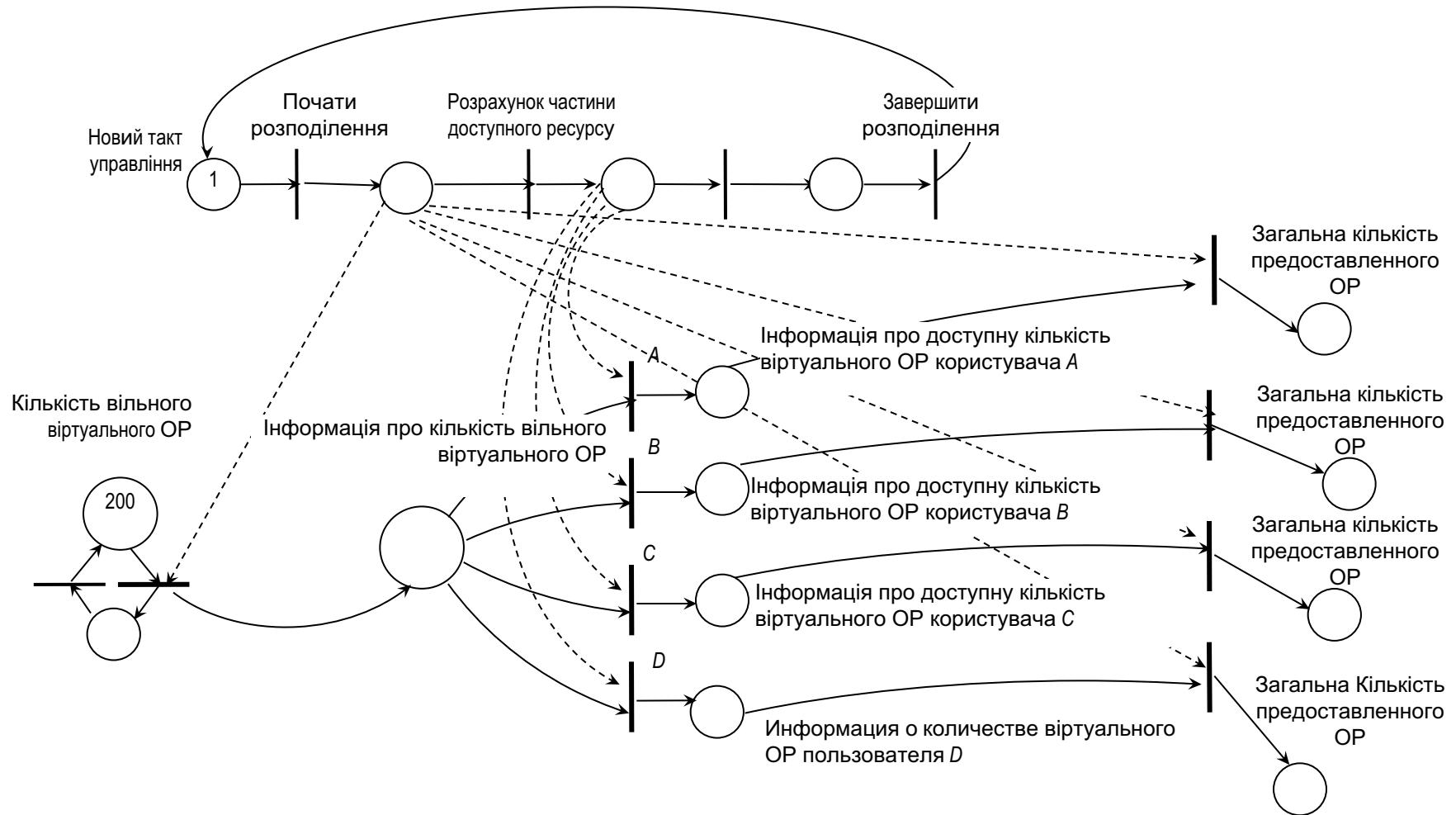
Петрі-об'єктна модель системи управління розподіленими обчислювальними ресурсами



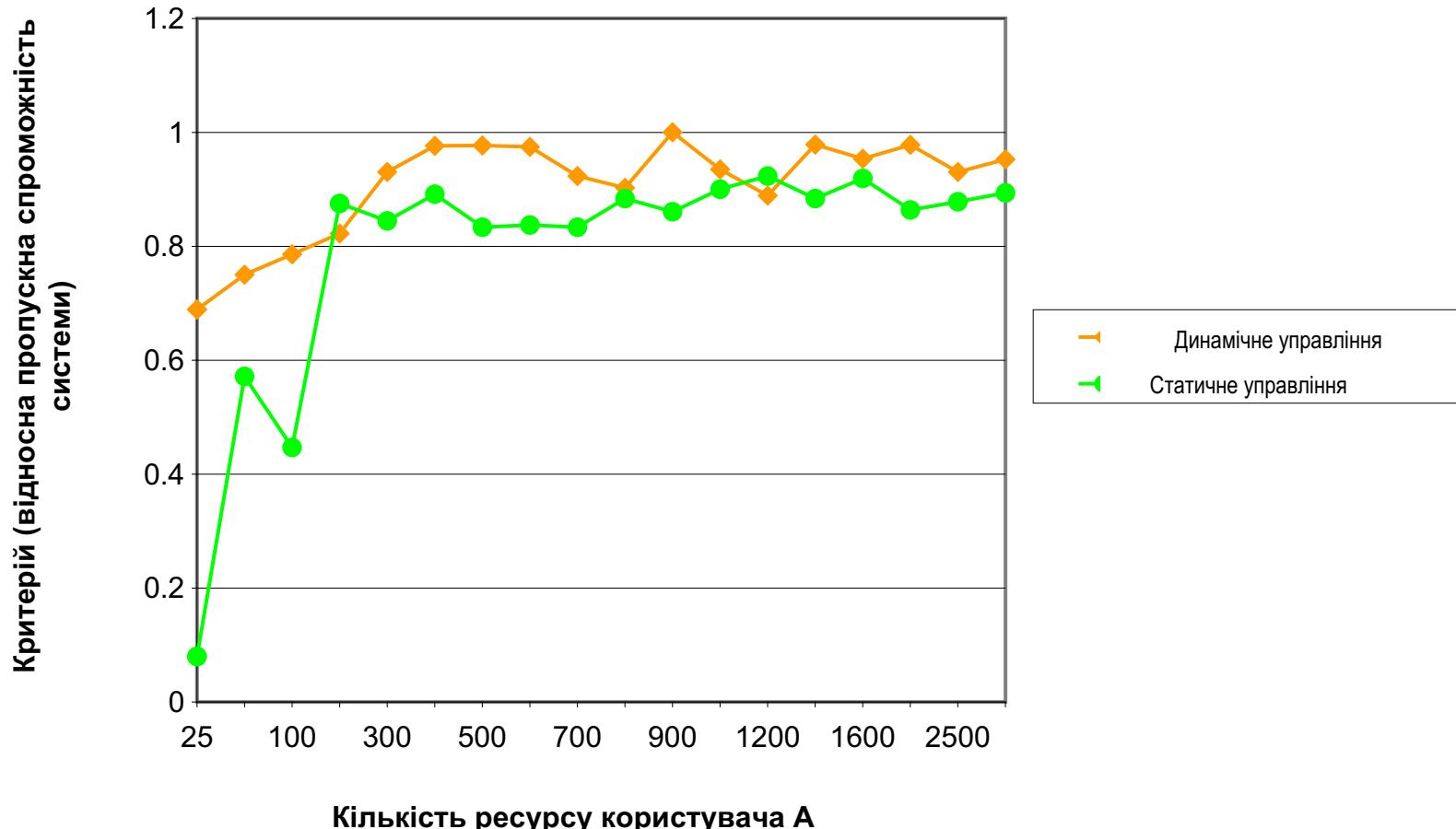
Петрі-об'єктна модель системи управління розподіленими обчислювальними ресурсами



Мережа Петрі-об'єкта «планувальник»



Результати дослідження впливу типу управління на ефективність функціонування системи



Петрі-об'єктна модель системи управління транспортним рухом. Постановка задачі

Входні змінні моделі:

- структура ділянки дорожнього руху,
- Інтенсивності надходження авто у вхідні точки ділянки дорожнього руху
- середня швидкість руху та довжина шляху між сусідніми перехрестями
- засоби регулювання дорожнім рухом на перехрестях
- параметри світлофорних об'єктів управління
- вимоги безпеки дорожнього руху

Вихідні змінні моделі:

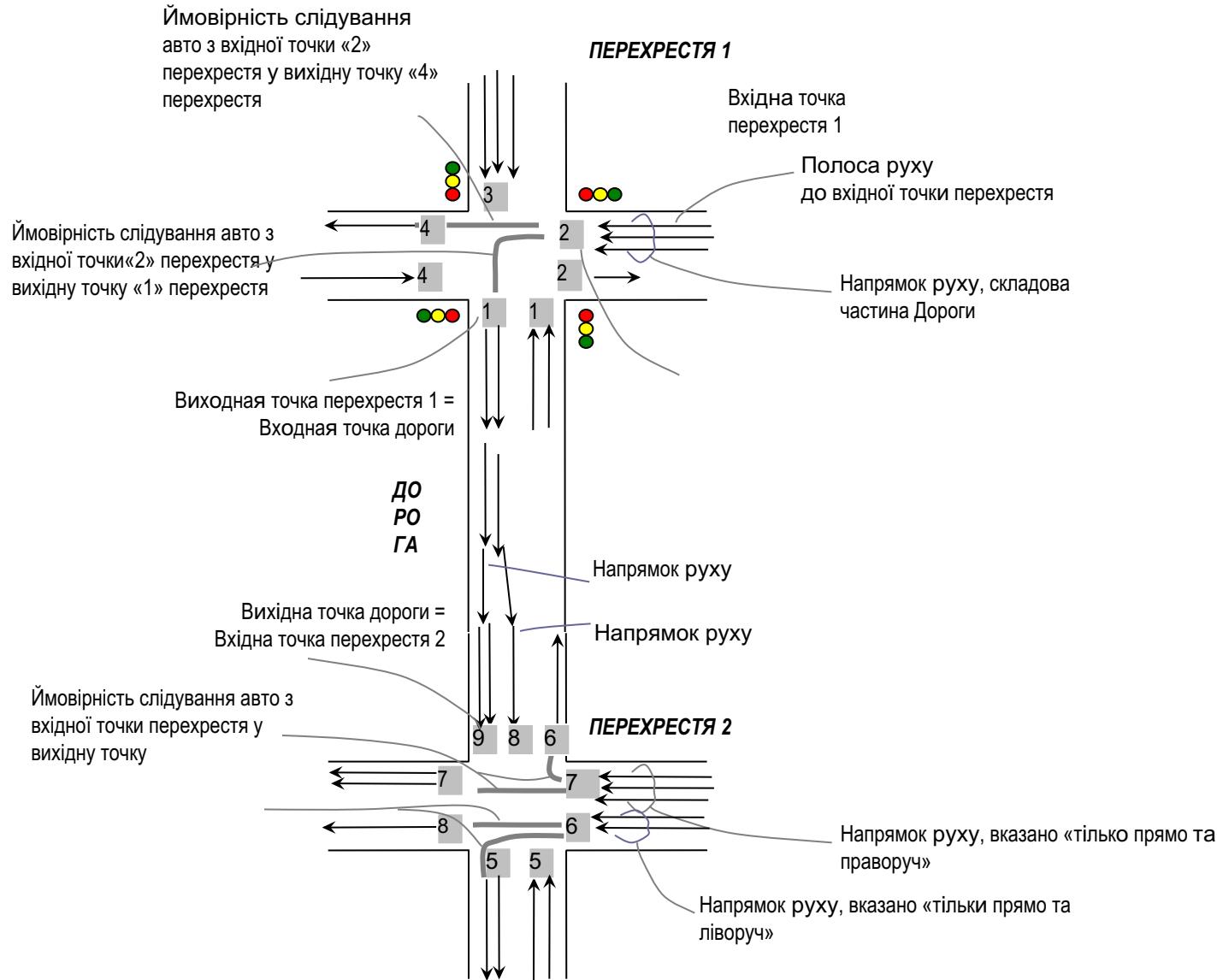
- середня кількість авто, що очікують переїзду, на кожному перехресті в кожному напрямку,
 - найбільше зі значень середньої кількості авто, що очікують переїзду, на кожному перехресті в кожному напрямку
 - оптимальні значення параметрів управління

Критерій оптимальності

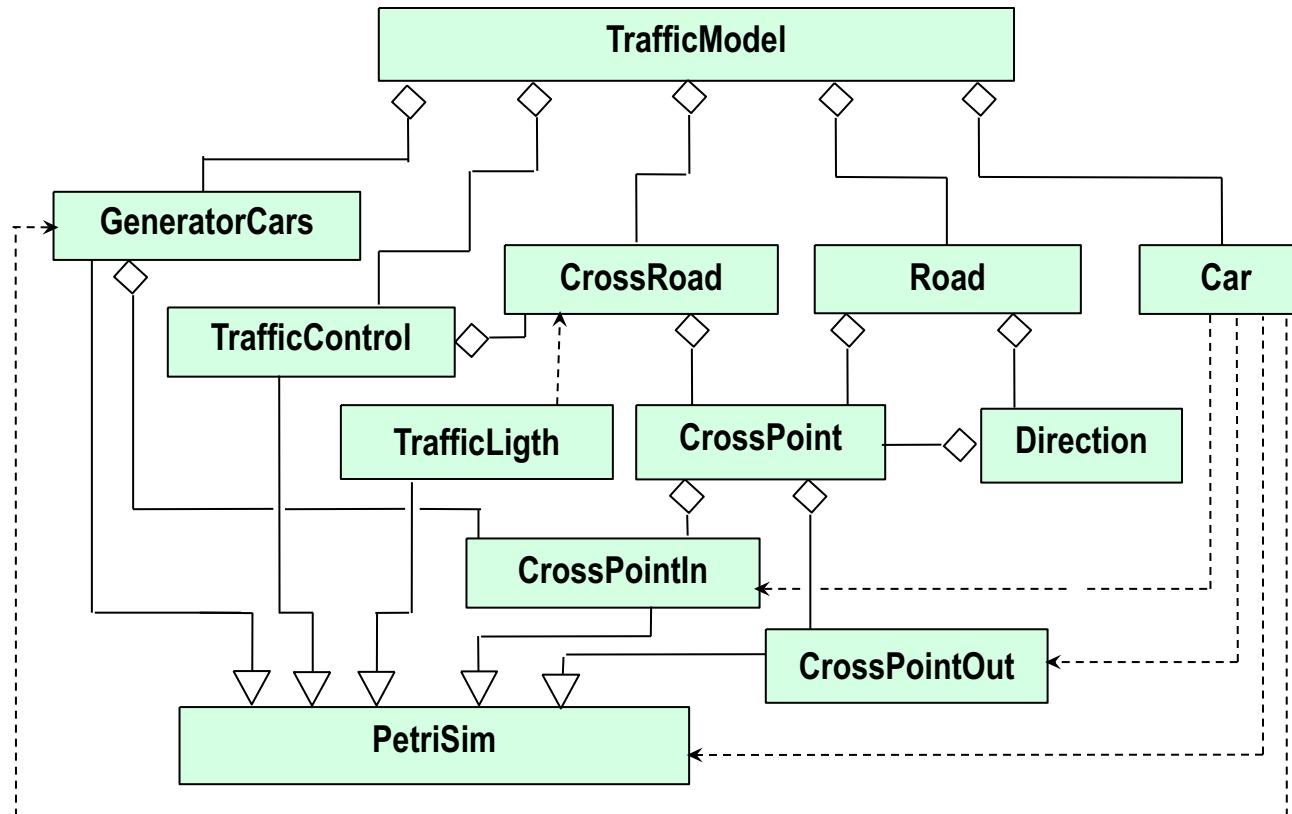
$$z = \max(L_1, L_2, L_3, \dots, L_k) \rightarrow \min$$

Среднее Кількість авто, ожидающих
переезда в i-ой точке

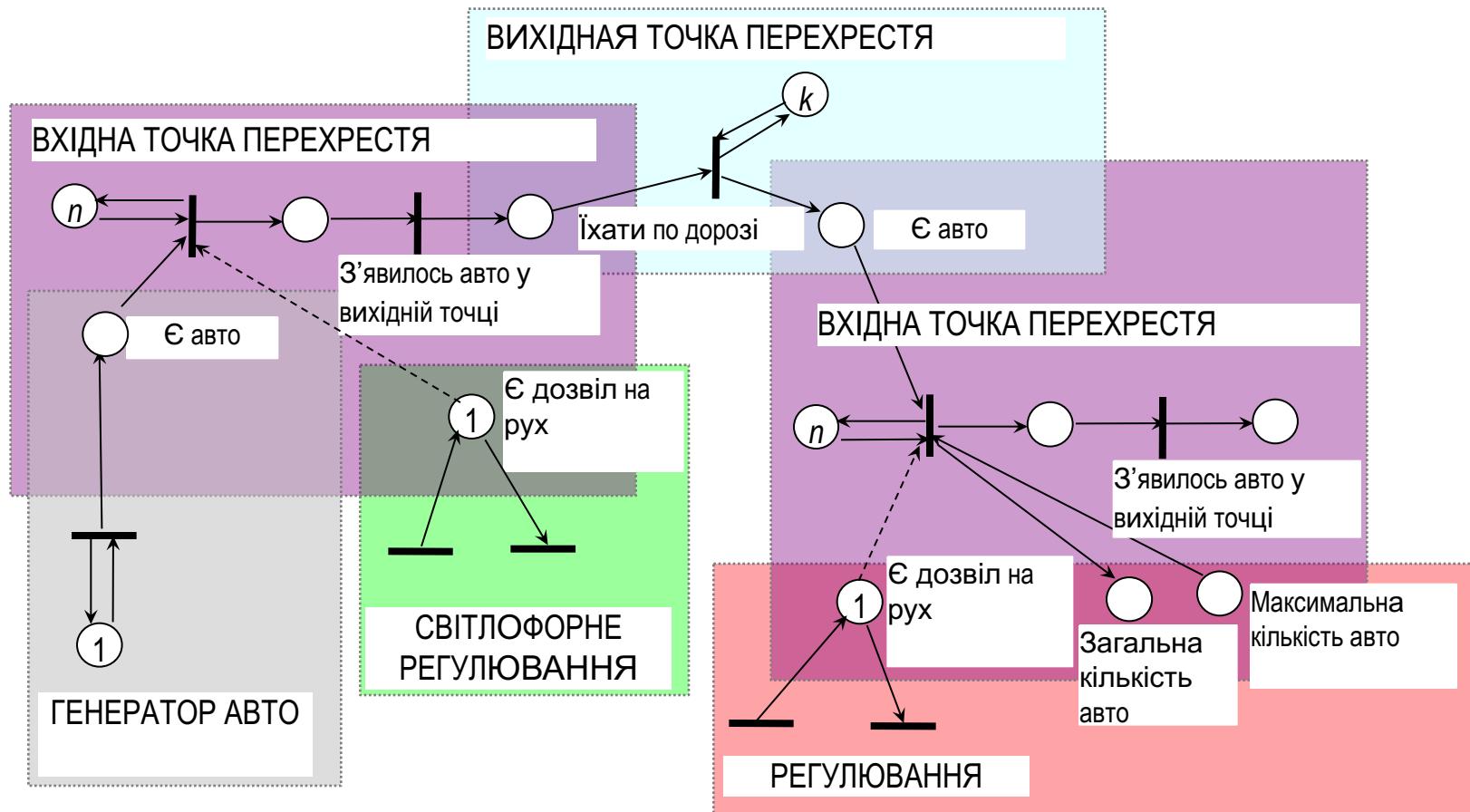
Схема ділянки дорожного руху



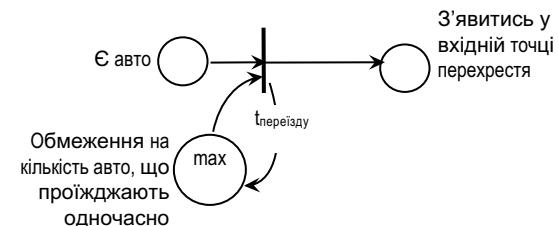
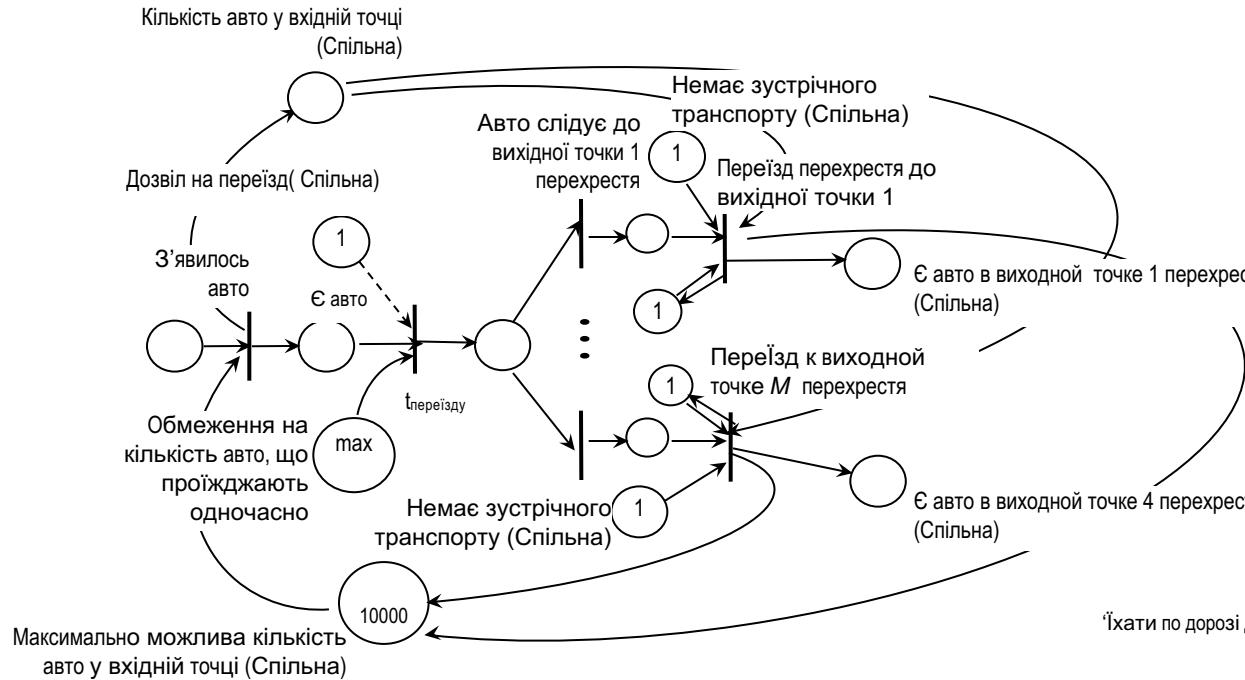
Діаграма класів Петрі-об'єктної моделі системи управління дорожнім рухом



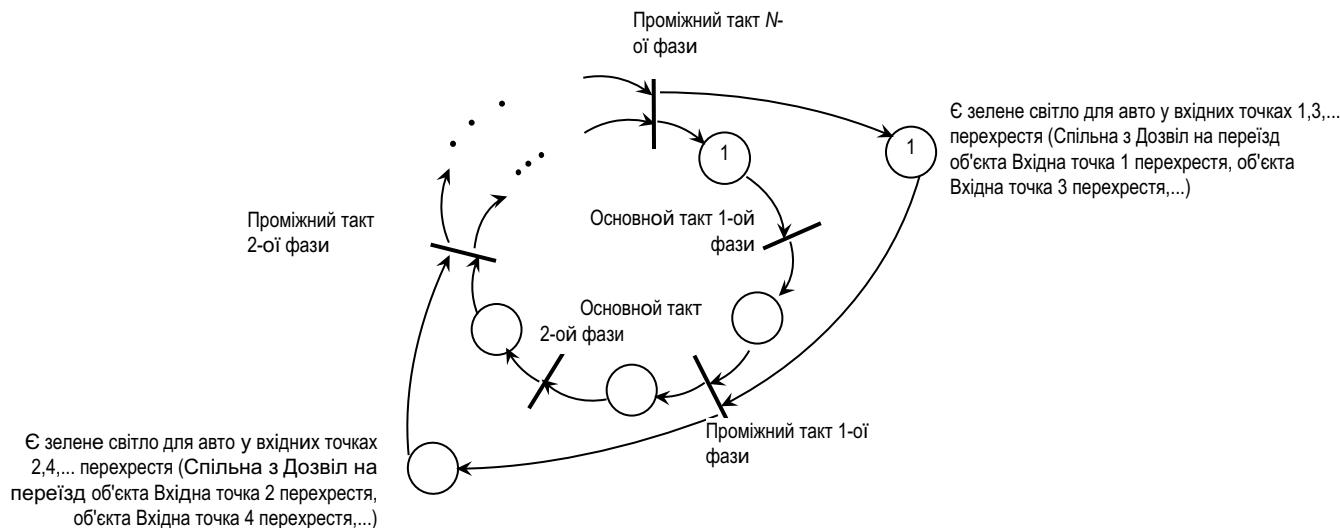
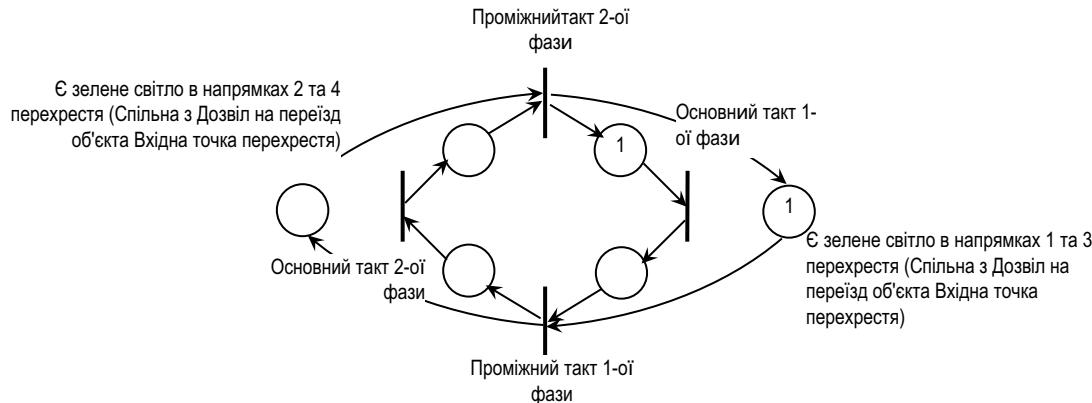
Діаграма зв'язків Петрі-об'єктів моделі системи управління транспортним рухом



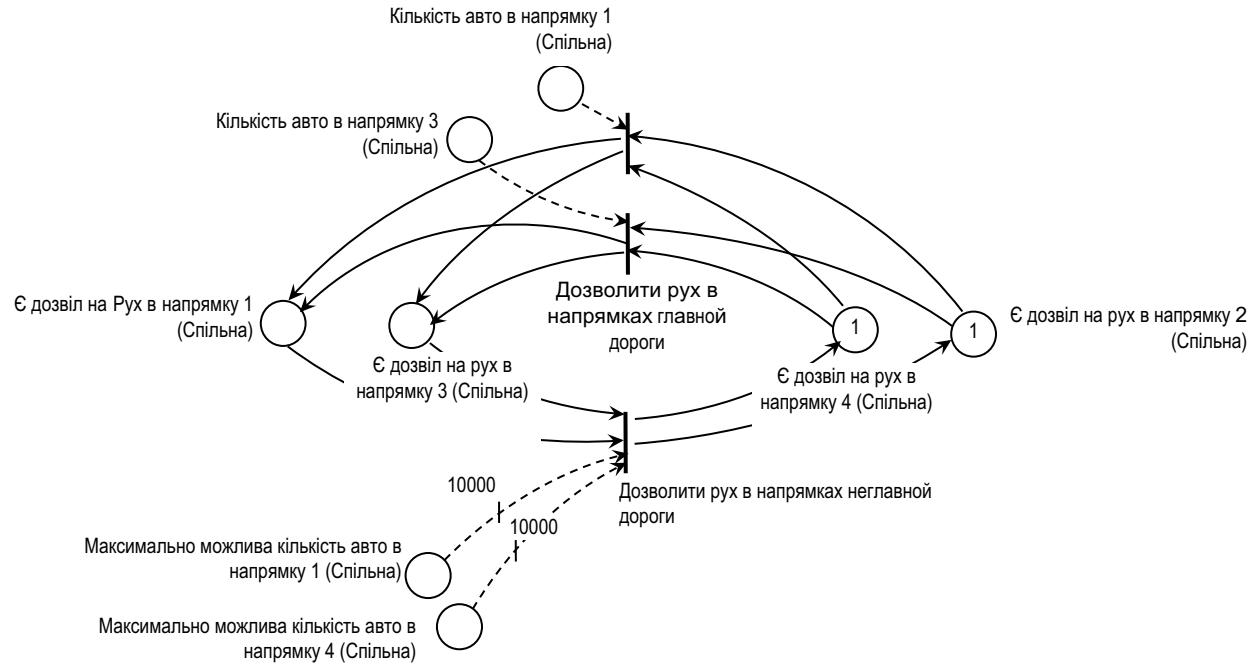
Мережа Петрі-об'єктів Вхідна та Вихідна точки перехрестя



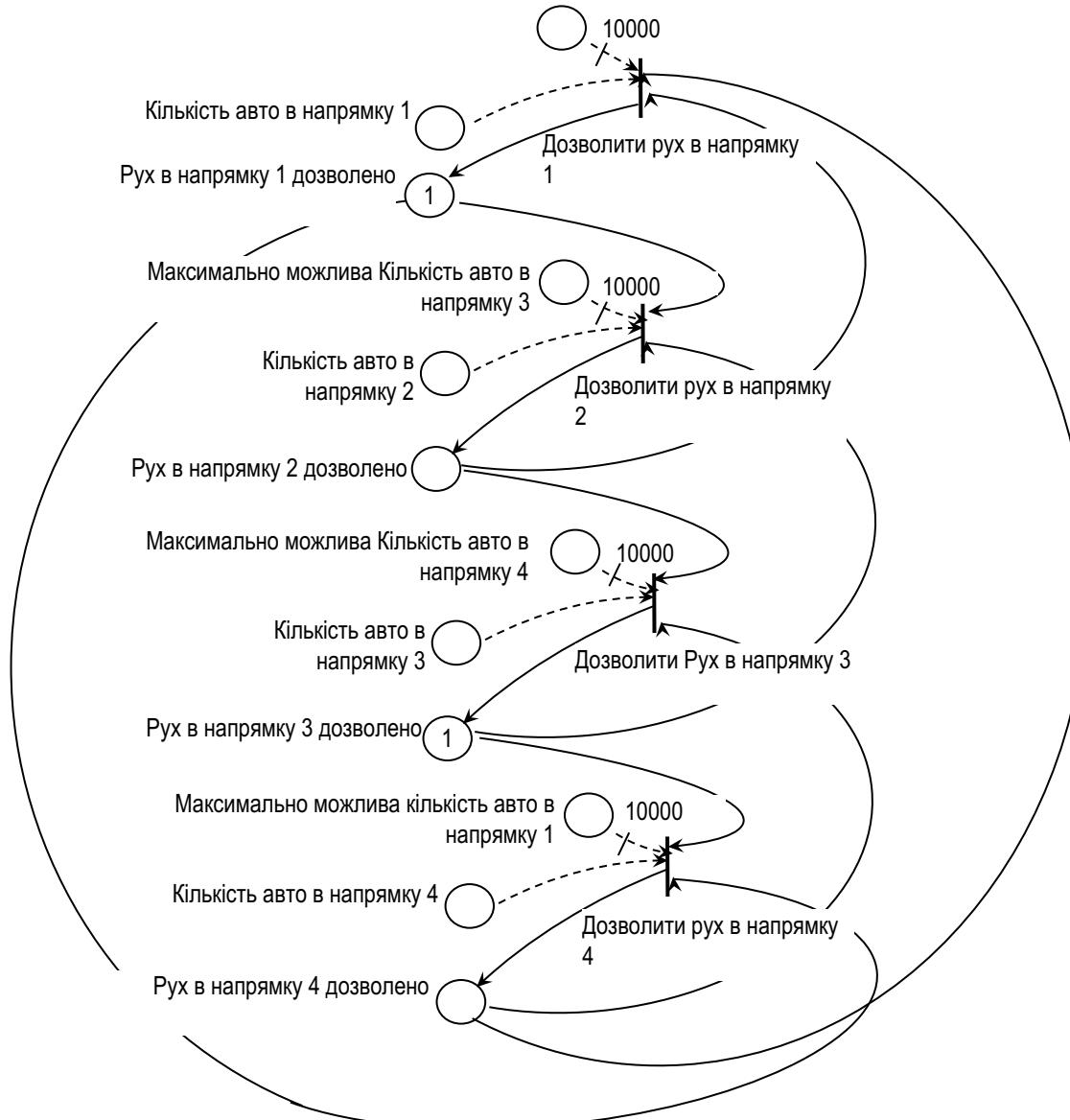
Мережа Петрі-об'єкта світлофорне регулювання



Сеть Петри-об'єкта Регулювання знаками дорожнього руху



Мережа Петрі-об'єкта Регулювання (правило «Пропустити авто справа»)



МОДЕЛЮВАННЯ
РОЗПОДІЛЕНИХ ТА ПАРАЛЕЛЬНИХ
ОБЧИСЛЕНИЙ СТОХАСТИЧНИМИ
МЕРЕЖАМИ ПЕТРІ

Термінологія

Розподілені обчислення
Distributed Computing

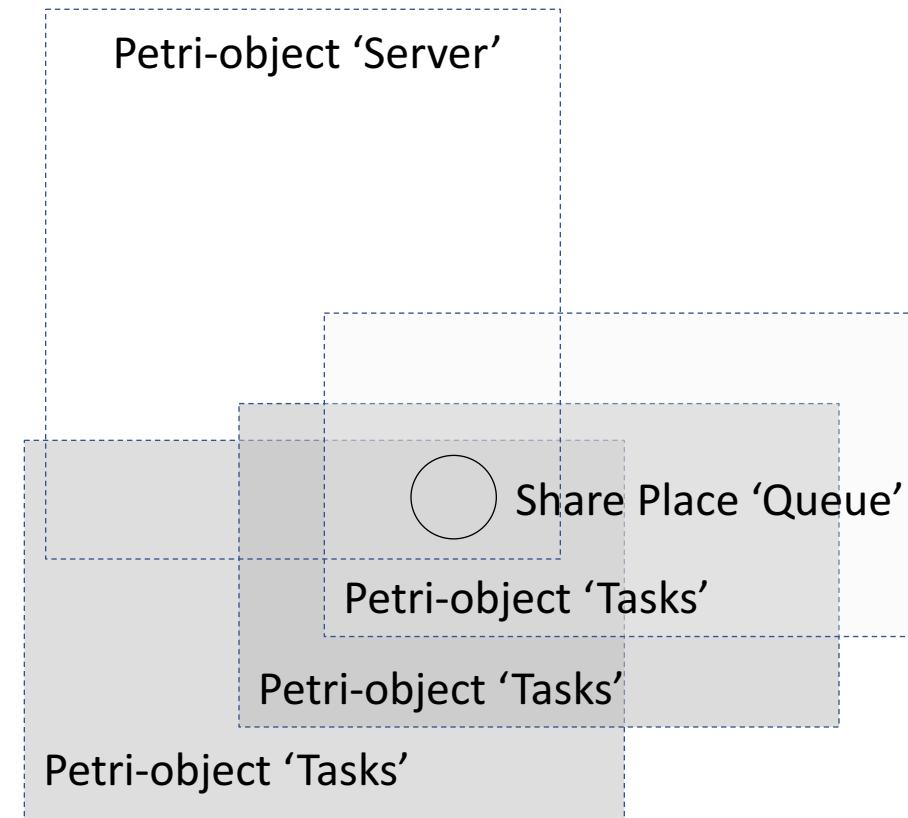
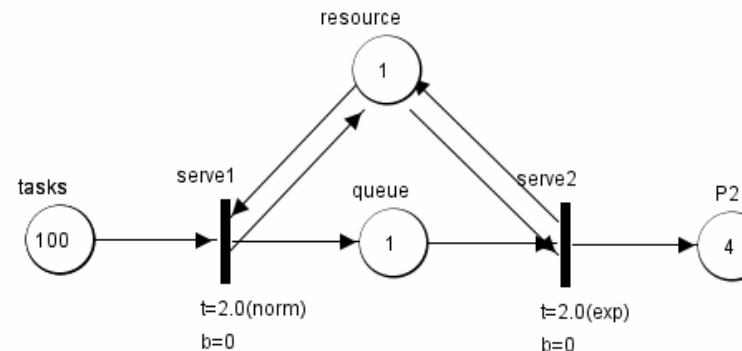
Багатопроцесорні обчислення
Multiprocessor Computing

Паралельні обчислення
Parallel Computing

Багатоядерні обчислення
Multicore Computing

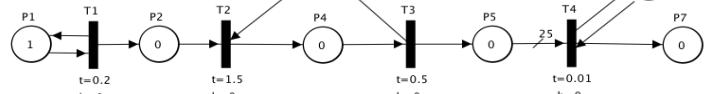
Багатопотокові обчислення
Concurrent Computing

Стохастичні мережі Петрі та Петрі-об'єктний підхід

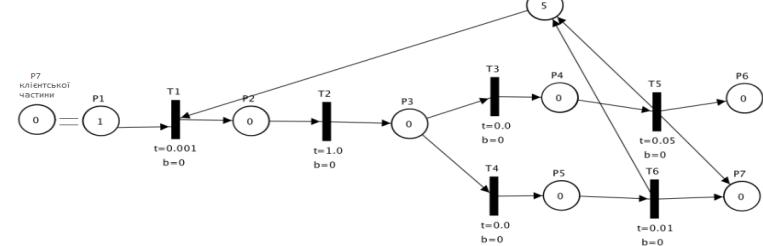


Моделювання клієнт-серверного додатку (розділені обчислення)

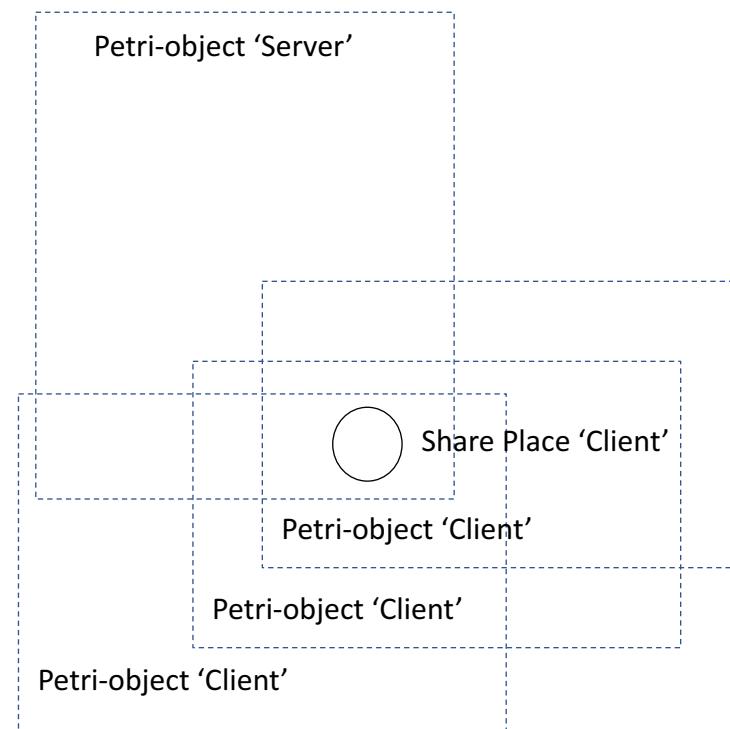
Petri-object 'Client'



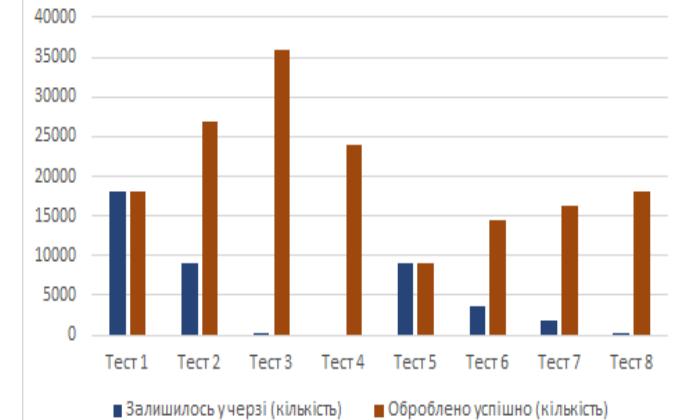
Petri-object 'Server'



Petri-object 'Server'



Тестування параметрів системи



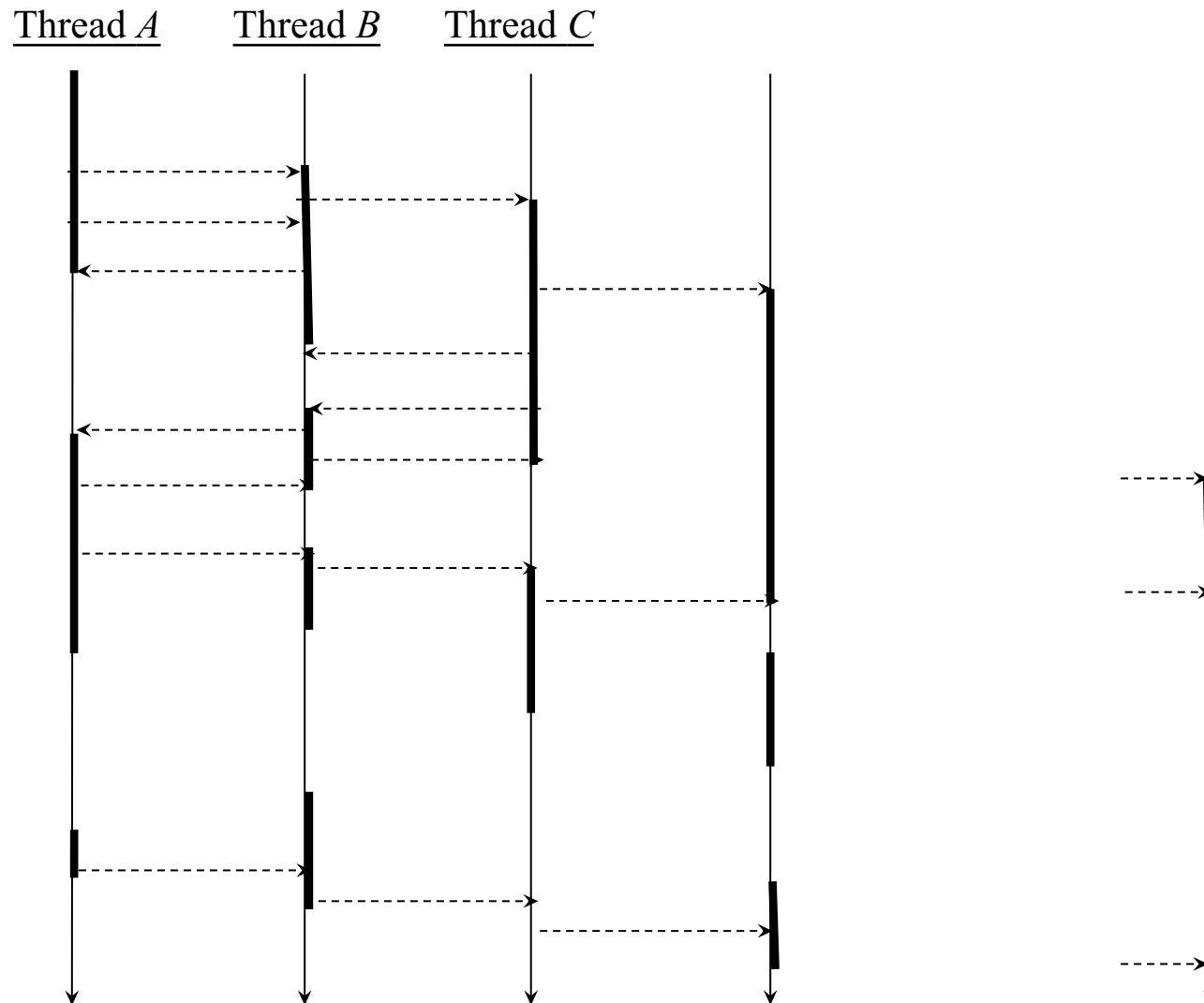
[Шишкін, В. І. Програмно-апаратний комплекс розумного відеореєстратора : магістерська дис. : 126 Інформаційні системи та технології / Шишкін Владислав Ігорович. – Київ, 2018. – 85 с.]

Багатопоточне програмування

Проблеми:

- Deadlock
- Starvation
- Livelock
- Memory consistency error

Існуючі дебагери
орієнтовані на
відлагодження
послідовних, не
паралельних програм



Проблеми розробки паралельних програм

Проблеми паралельних програм:

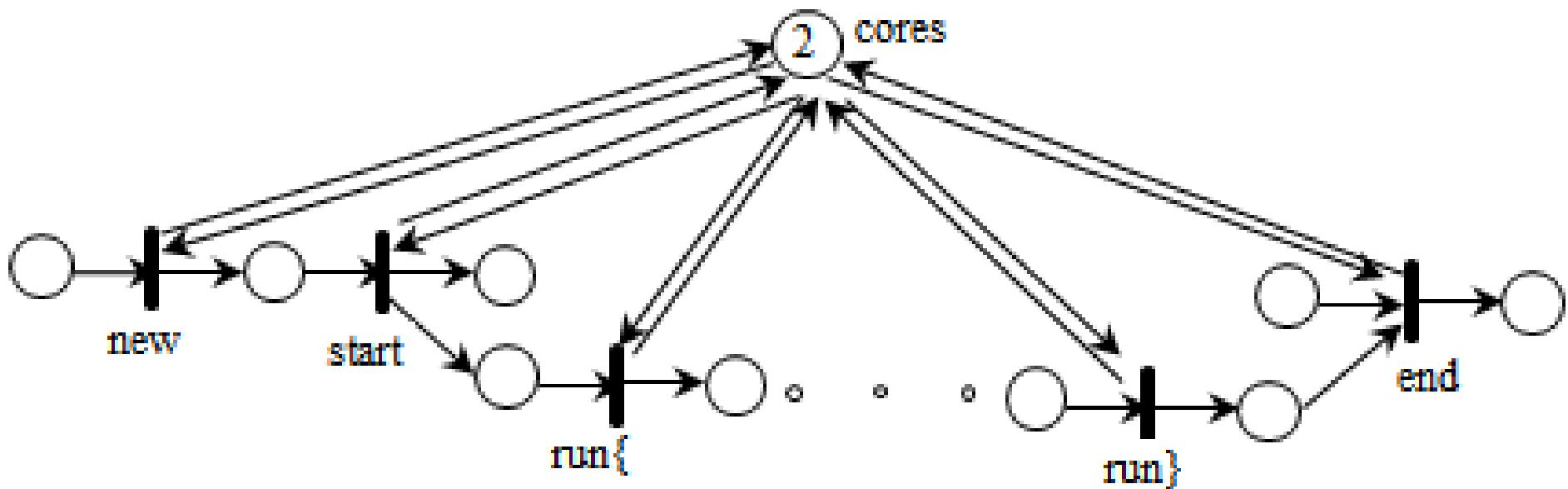
- можливість виникнення взаємного блокування роботи потоків (deadlock), неможливість завершення роботи потоків (livelock), неможливість захоплення ресурсу потоком (starvation),
- помилка при спільному використанні ресурсу (memory consistency error, «гонка» потоків),
- сповільнення роботи потоків через синхронізацію дій.

Проблеми, що ускладнюють процес розробки та тестування паралельних програм:

- недетермінований порядок інструкцій, виконуваних потоками,
- залежність результату запуску паралельної програми від обчислювальних ресурсів, на яких вона запускається.

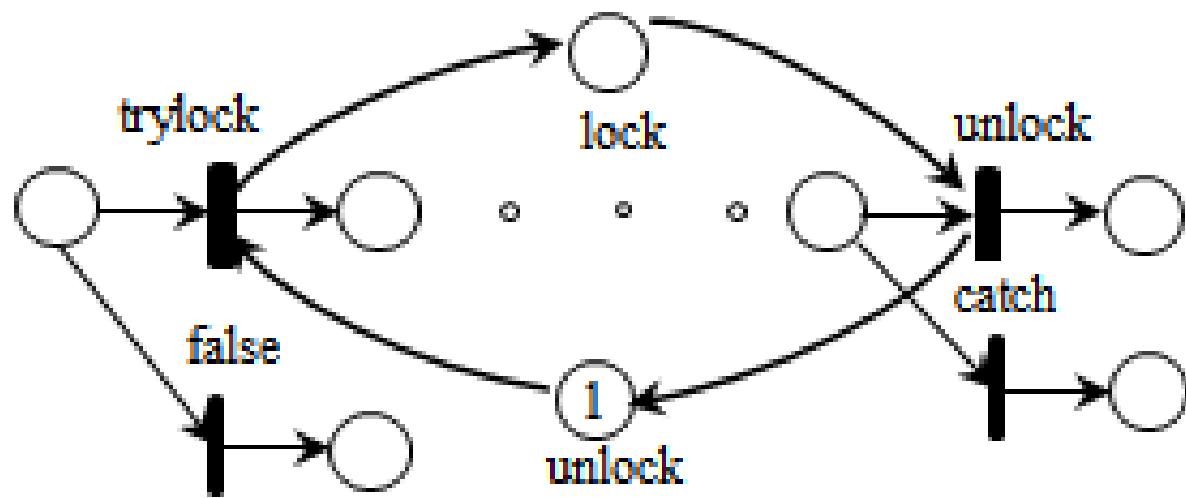
Реалізація створення потоку, початку та кінця його роботи

```
public static void main(String[] args) {  
    Thread thread = new Thread(new Runnable());  
    thread.start();  
}
```



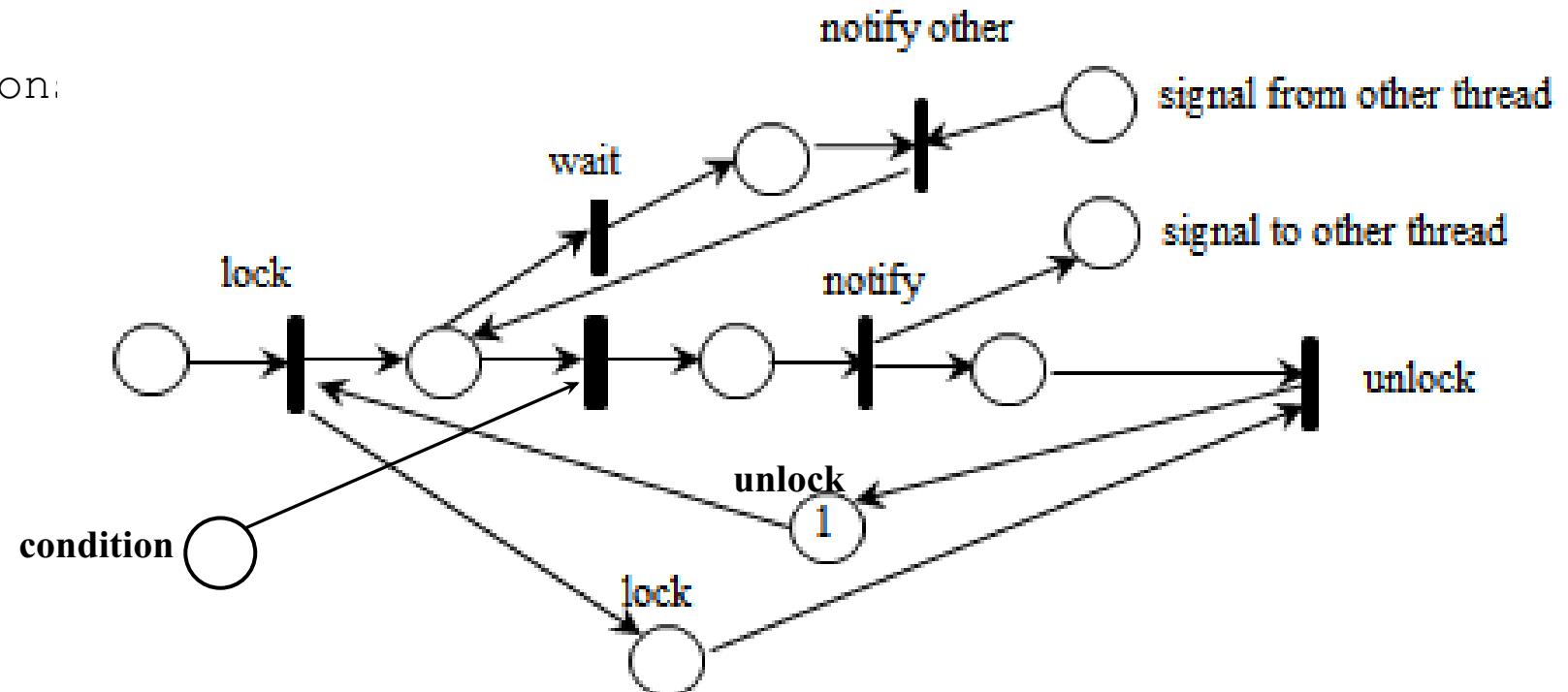
БЛОКУВАННЯ ПОТОКУ

```
//https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Lock.html#tryLock()
Lock lock = ...;
if (lock.tryLock()) {
    try {
        // manipulate protected state
    } finally {
        lock.unlock();
    }
} else {
// perform alternative
}
```



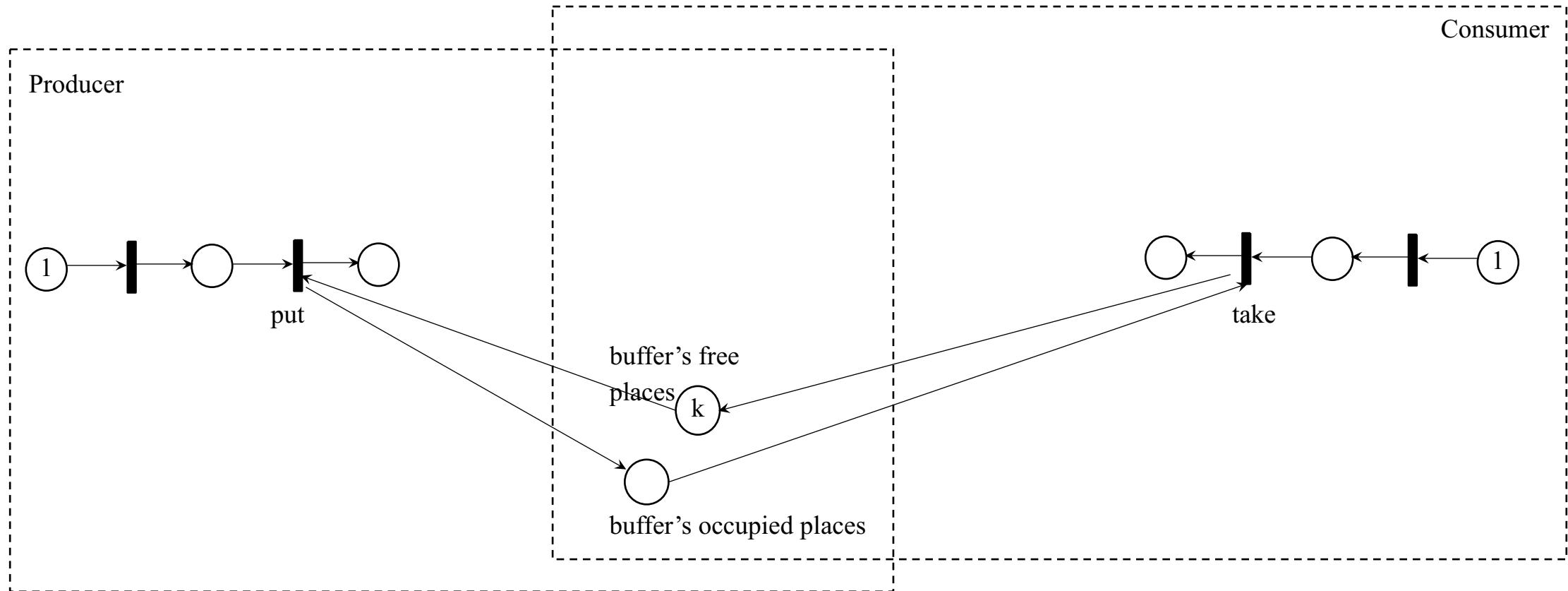
Синхронізація дій потоку

```
public synchronized void method() throws InterruptedException {  
    while (!condition()) { // guarded block  
        wait();  
    }  
    ...// some action:  
    notifyAll();  
}
```



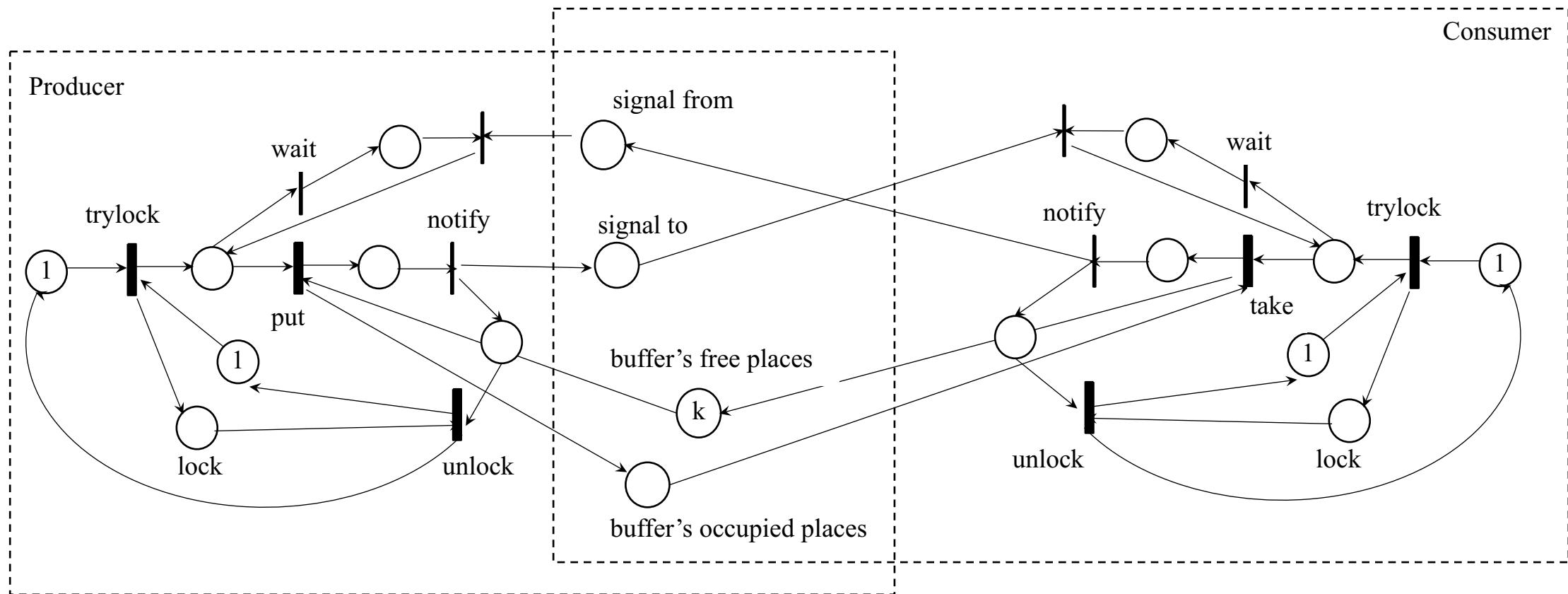
Модель Producer – Consumer.

Приклад Guarded block [Oracle: The Java Tutorials]



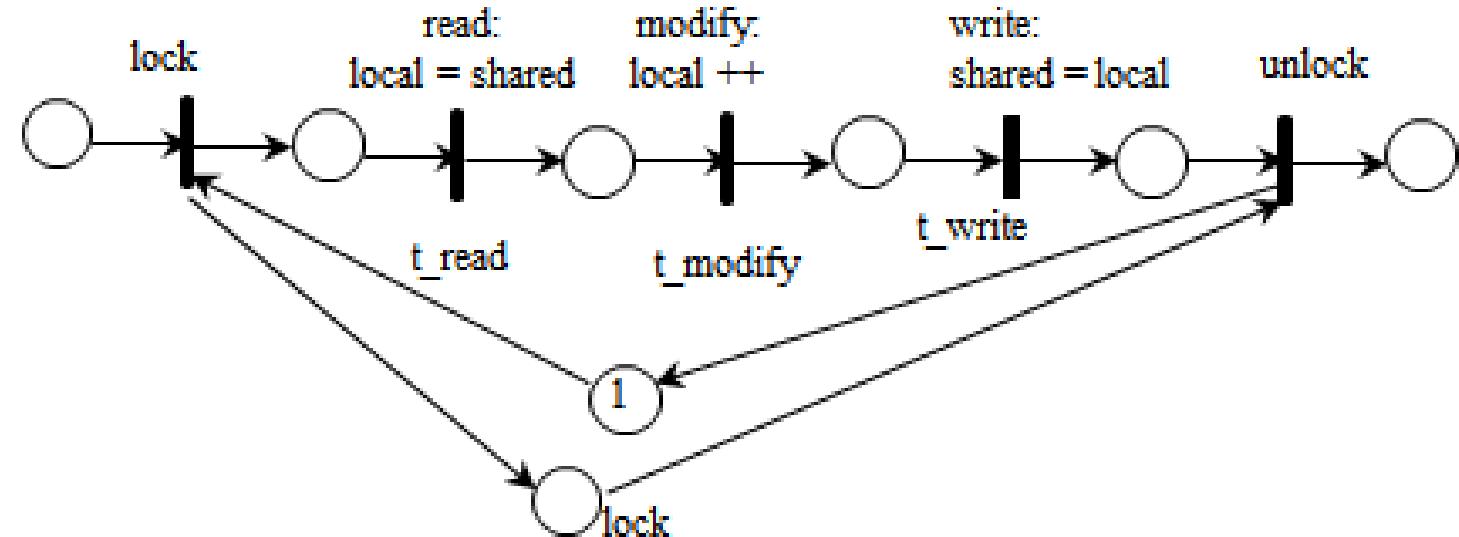
Модель Producer – Consumer.

Приклад Guarded block [Oracle: The Java Tutorials]



Доступ до спільних даних

```
public synchronized void incMethod(){  
    local++;  
}
```



Deadlock: приклад об'єктів Friend [Oracle: The Java Tutorials]

```
public class Deadlock {
    static class Friend {
        private final String name;
        public Friend(String name) { this.name = name; }
        public String getName() { return this.name; }
        public synchronized void bow(Friend other) {
            System.out.format("%s: %s" + " has bowed to me!%n", this.name, bower.getName());
            other.bowBack(this);
        }
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: %s" + " has bowed back to me!%n", this.name, bower.getName());
        }
    }
    public static void main(String[] args) {
        final Friend a = new Friend("A");
        final Friend b = new Friend("B");
        new Thread(new Runnable) {
            public void run() { a.bow(b); }
        }.start();
        new Thread(new Runnable) {
            public void run() { b.bow(a); }
        }.start();
    }
}
```

Deadlock: приклад об'єктів Friend з The Java Tutorials

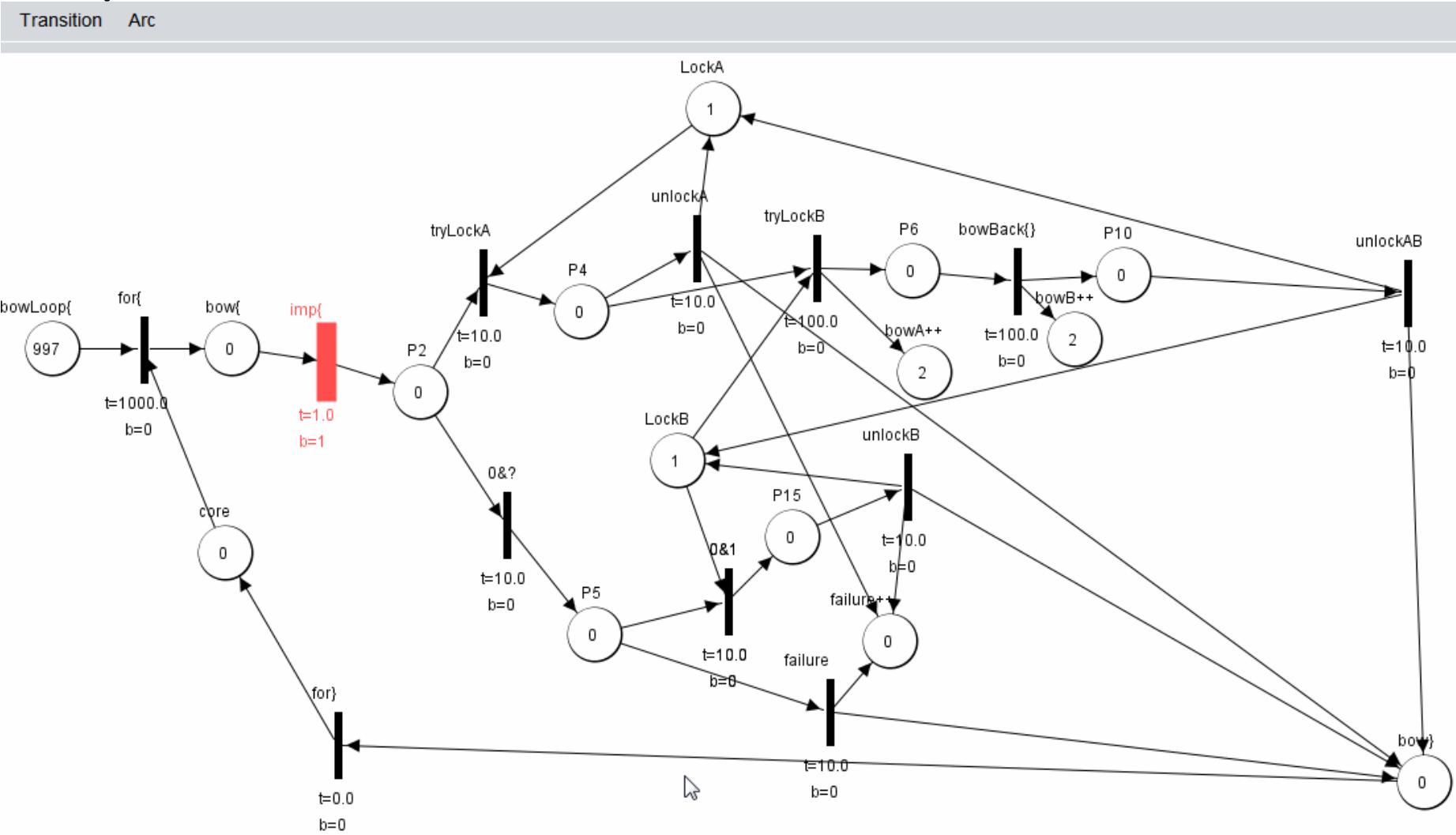
```
public class Deadlock {
    static class Friend {
        private final String name;
        public Friend(String name) { this.name = name; }
        public String getName() { return this.name; }
        public synchronized void bow(Friend other) {
            System.out.format("%s: %s" + " has bowed to me!%n", this.name, bower.getName());
            other.bowBack(this); // захоплення локера об'єкта other
        }
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: %s" + " has bowed back to me!%n", this.name, bower.getName());
        }
    }
    public static void main(String[] args) {
        final Friend a = new Friend("A");
        final Friend b = new Friend("B");
        new Thread(new Runnable()) {
            public void run() { a.bow(b); } } // захоплення локера a, потім b
            ).start();
        new Thread(new Runnable()) {
            public void run() { b.bow(a); } } // захоплення локера b, потім a
            ).start();
    }
}
```


Safelock: приклад потоків Friend з The Java Tutorials

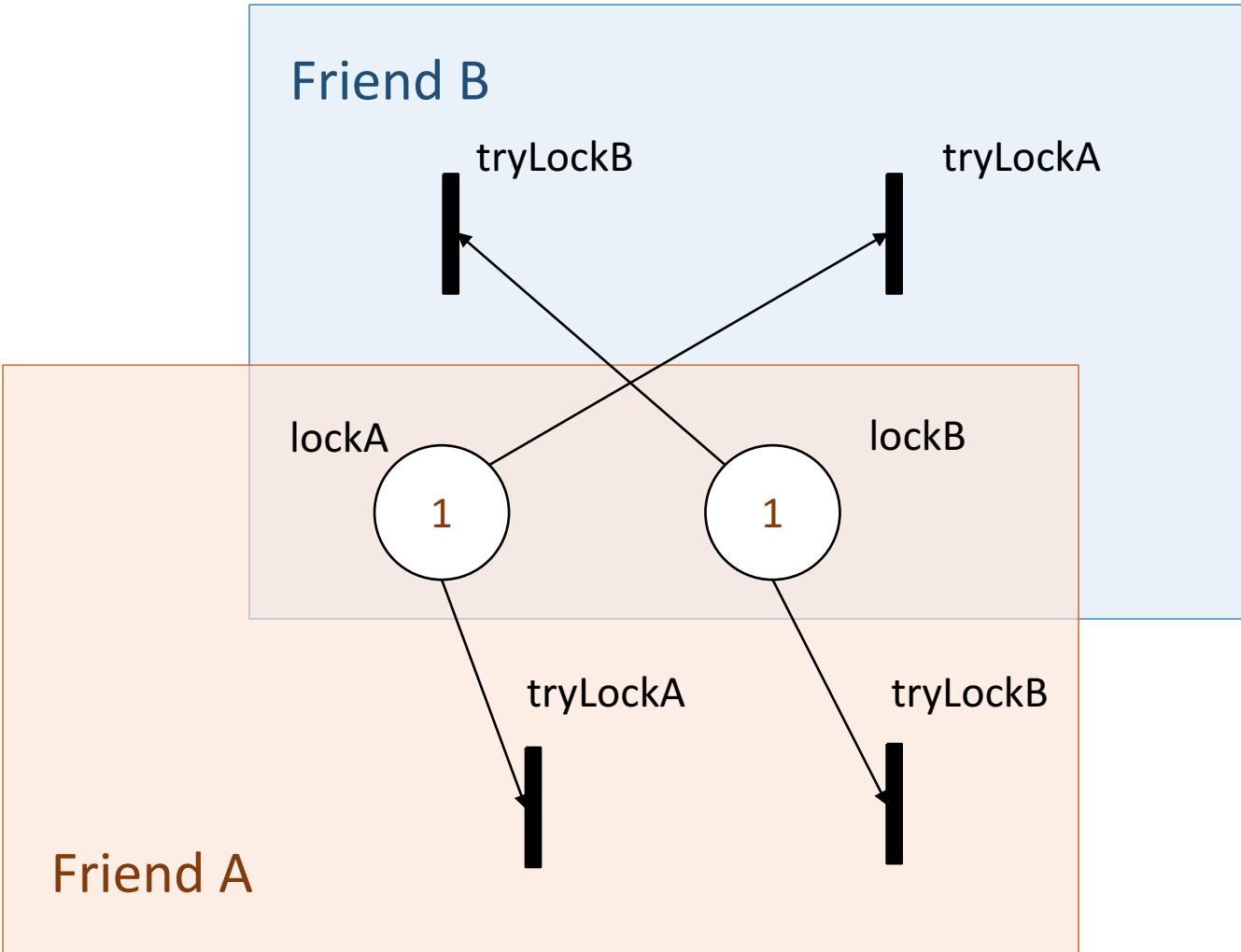
```
static class BowLoop implements Runnable {
    private Friend bower; private Friend bowee;
    public BowLoop(Friend one, Friend other) {
        this.bower = one;
        this.bowee = other;
    }
    public void run() {
        Random random = new Random();
        for (;;) {
            try {
                Thread.sleep(random.nextInt(10));
            } catch (InterruptedException e) {}
            bowee.bow(bower);
        }
    }
}

public static void main(String[] args) {
    final Friend a = new Friend("A");
    final Friend b = new Friend("B");
    new Thread(new BowLoop(a, b)).start();
    new Thread(new BowLoop(b, a)).start();
}
```

Моделювання конфлікту потоків: мережа Петрі об'єкту Friend



Зв'язки між Петрі-об'єктами



Програмний код створення Петрі-об'єктів

```
//Petri-objects creation
class Friend extends PetriSim {
    public Friend(String name, int loop) throws ExceptionInvalidNetStructure {
        super(NetLibrary.CreateNetFriendUsingCores(name, loop, 2)); // 2 cores
    }
    public void addFriend(Friend other){
        this.getNet().getListP()[7] = other.getNet().getListP()[2]; //lockOther = lock
        this.getNet().getListP()[15] = other.getNet().getListP()[15]; // coresOther = cores
    }
}
Friend friendA = new Friend("A", 1000);
Friend friendB = new Friend("B", 1000);
Friend friendC = new Friend("C", 1000);
Friend friendD = new Friend("D", 1000);

friendA.addFriend(friendB);
friendA.addFriend(friendC);
friendA.addFriend(friendD);

friendB.addFriend(friendA);
friendB.addFriend(friendC);
friendB.addFriend(friendD);

friendC.addFriend(friendA);
friendC.addFriend(friendB);
friendC.addFriend(friendD);

friendD.addFriend(friendA);
friendD.addFriend(friendB);
friendD.addFriend(friendC);
```

Програмний код створення Петрі-об'єктної моделі

```
public static void main(String[] args) throws ExceptionInvalidNetStructure {
    ArrayList<PetriSim> list = new ArrayList<>();

    //Petri-objects creation

    list.add(friendA);
    list.add(friendB);
    list.add(friendC);
    list.add(friendD);

    PetriObjModel model = new PetriObjModel(list);

    model.setIsProtokol(false);
    model.go(100000000);
}
```

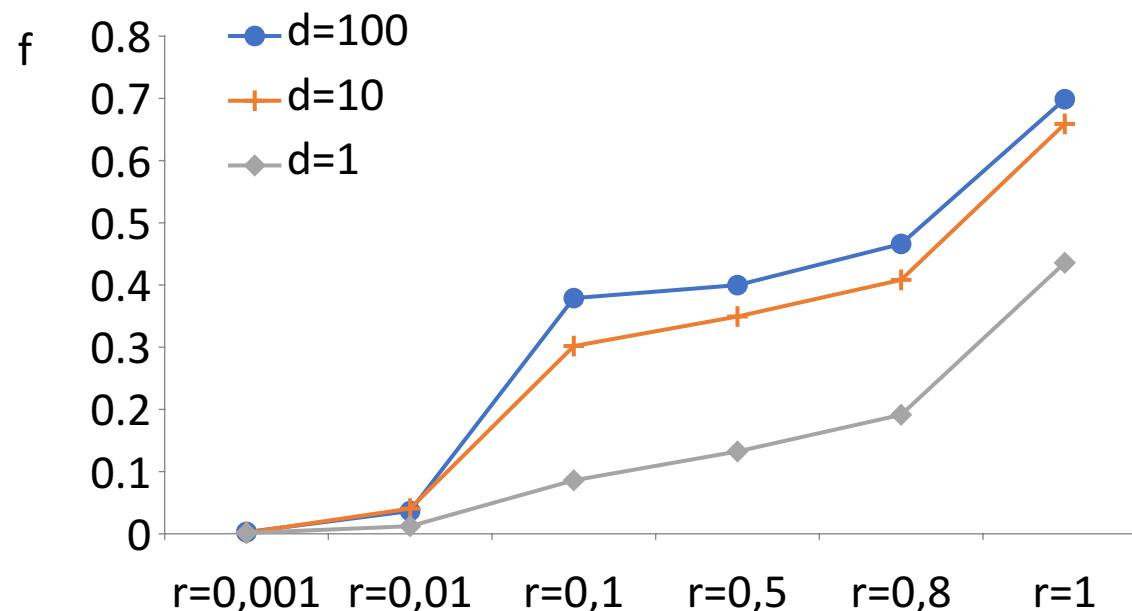
Експериментальне дослідження залежності появи конфлікту переходів від часових затримок

4 • 3 = 12 потоків

d - часова затримка переходу “for{}“ (відповідає команді sleep() у програмі)

d • r – часова затримка інших переходів, де r – співвідношення часових затримок переходів (відповідає виконанню простих інструкцій)

f – відносна частота появи невдалої спроби (конфлікту потоків)



[Stetsenko I.V., Dychyna O. Simulation of Multithreaded Algorithms Using Petri-Object Models. In: Hu Z., Petoukhov S., Dychka I., He M. (eds) Advances in Computer Science for Engineering and Education. ICCSEEA 2018. Advances in Intelligent Systems and Computing, vol 754, Springer, Cham, pp.391-401.]

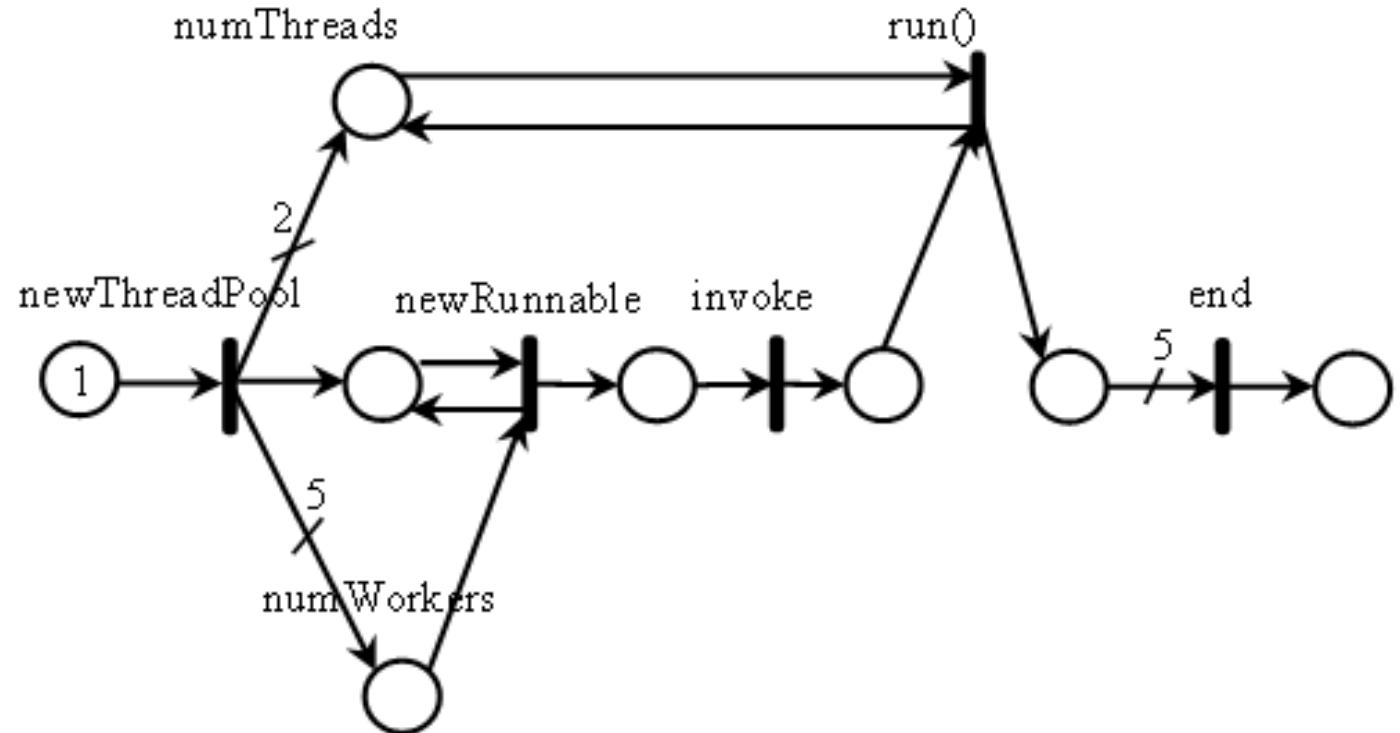
Оцінка точності результатів моделювання

Кількість потоків	Багатопоточна програма	Імітаційна модель (d=100, r=0.01)	Похибка
2	0.981450	0.978650	0.29%
12	0.959042	0.963417	0.46%
10 (90 workers)	0.987777	0.979080	0.88%
20 (380 workers)	0.988047	0.980910	0.72%
50 (2450 workers)	0.995212	0.981585	1.37%

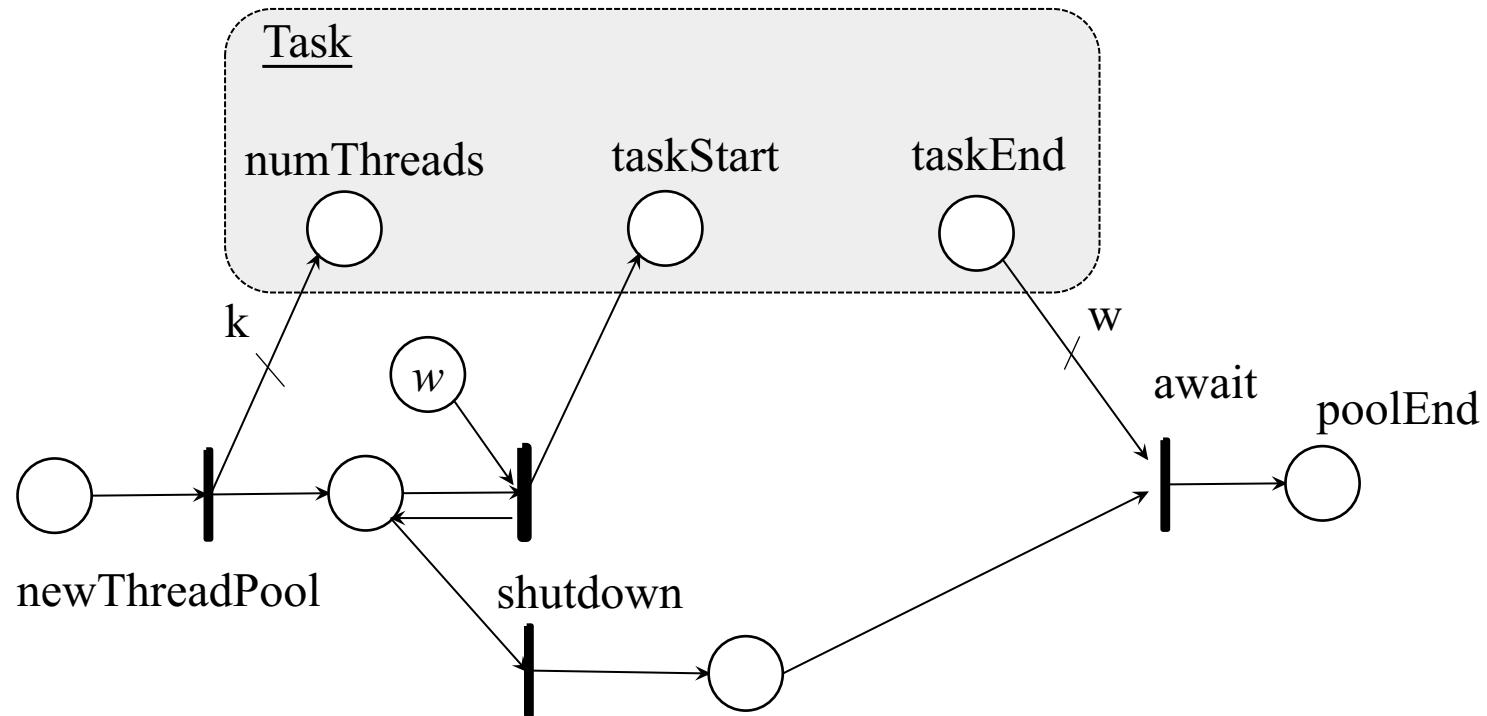
Пул потоків

```
ExecutorService executor =  
    Executors.newFixedThreadPool(2);  
for (int i = 0; i < 5; i++) {  
    Runnable task =  
        new TaskThread();  
    executor.execute(task);  
}
```

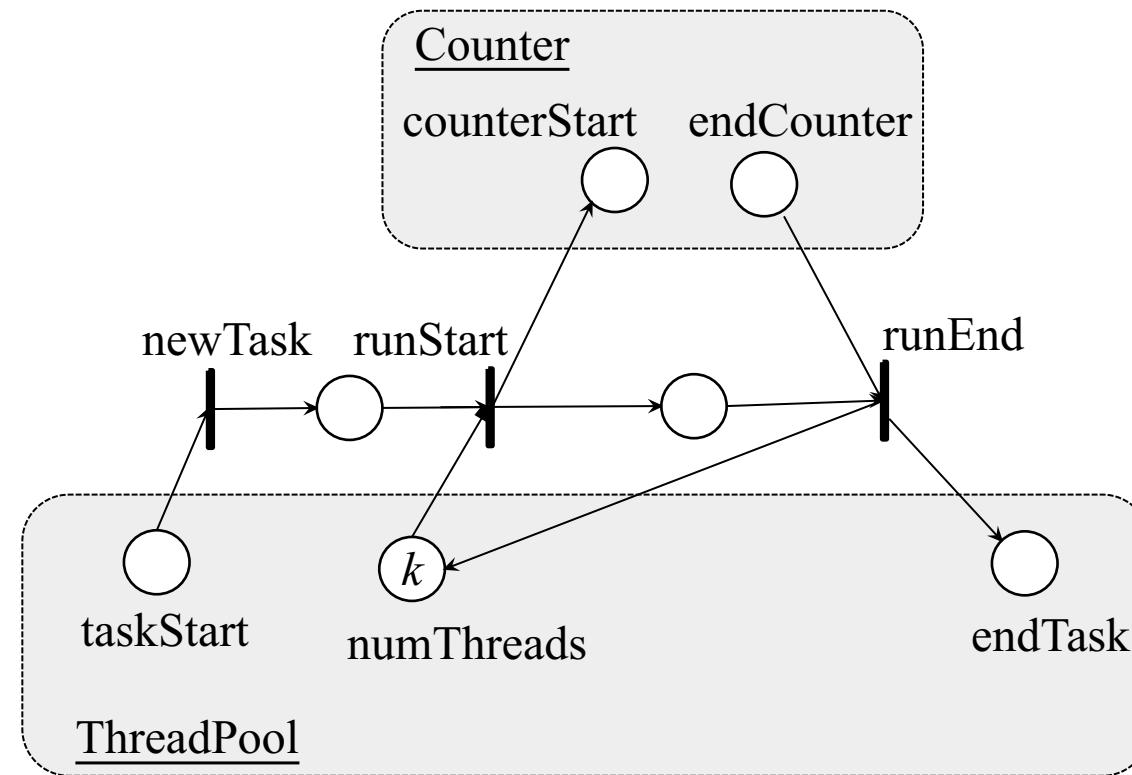
TaskThread – клас,
що імплементує інтерфейс Runnable.



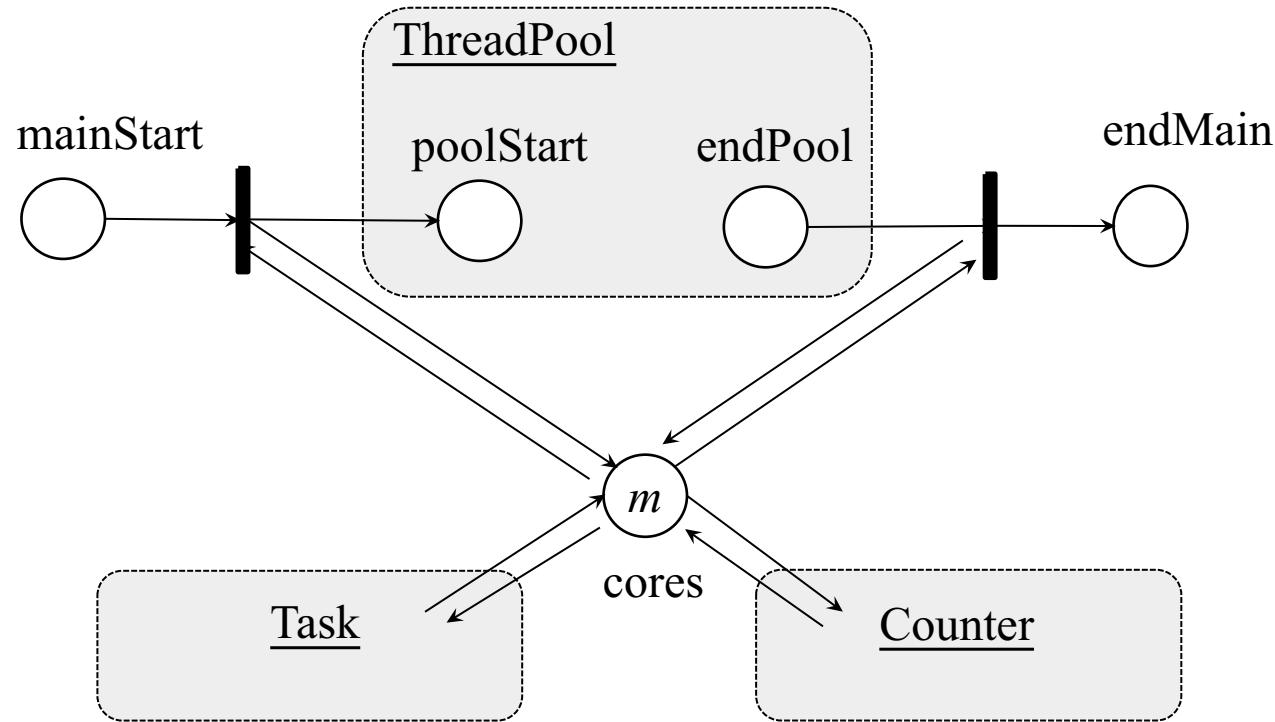
Петрі-об'єкт *ThreadPool*



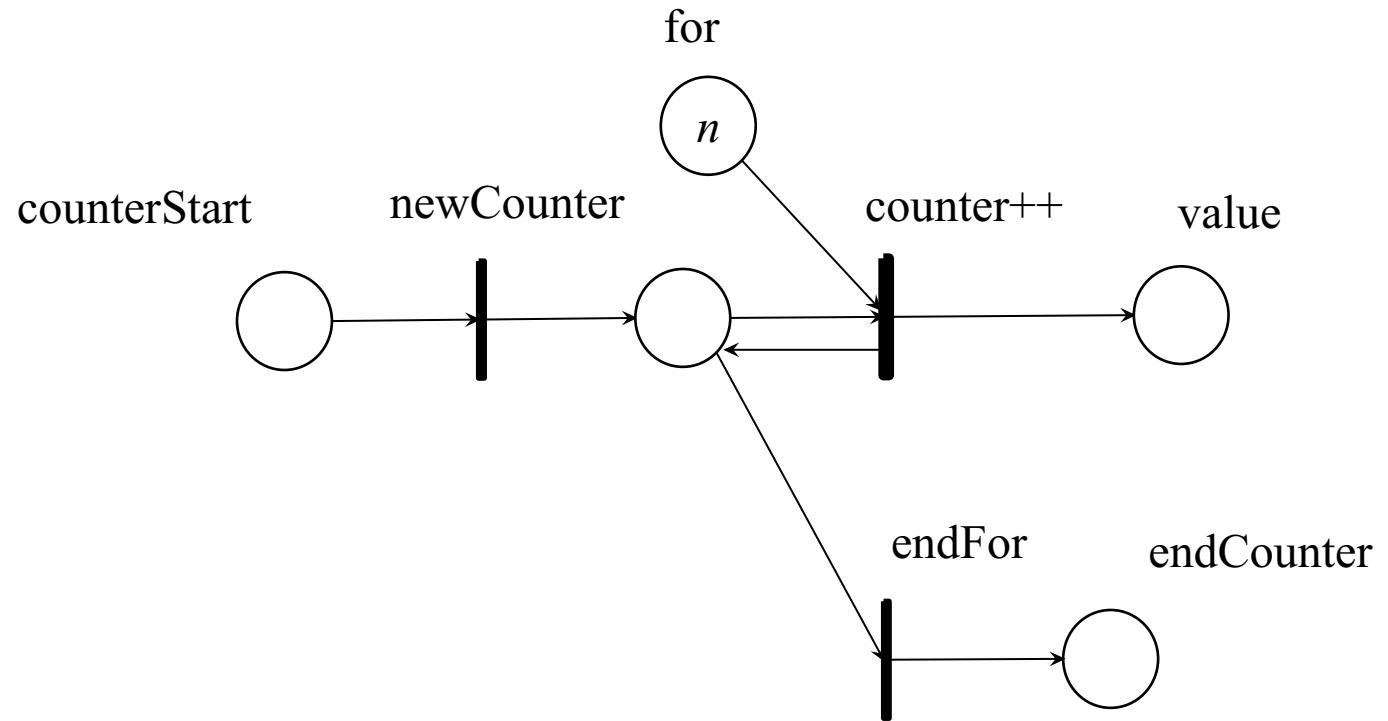
Петрі-об'єкт Task



Моделювання пулу потоків. Петрі-об'єкт Main

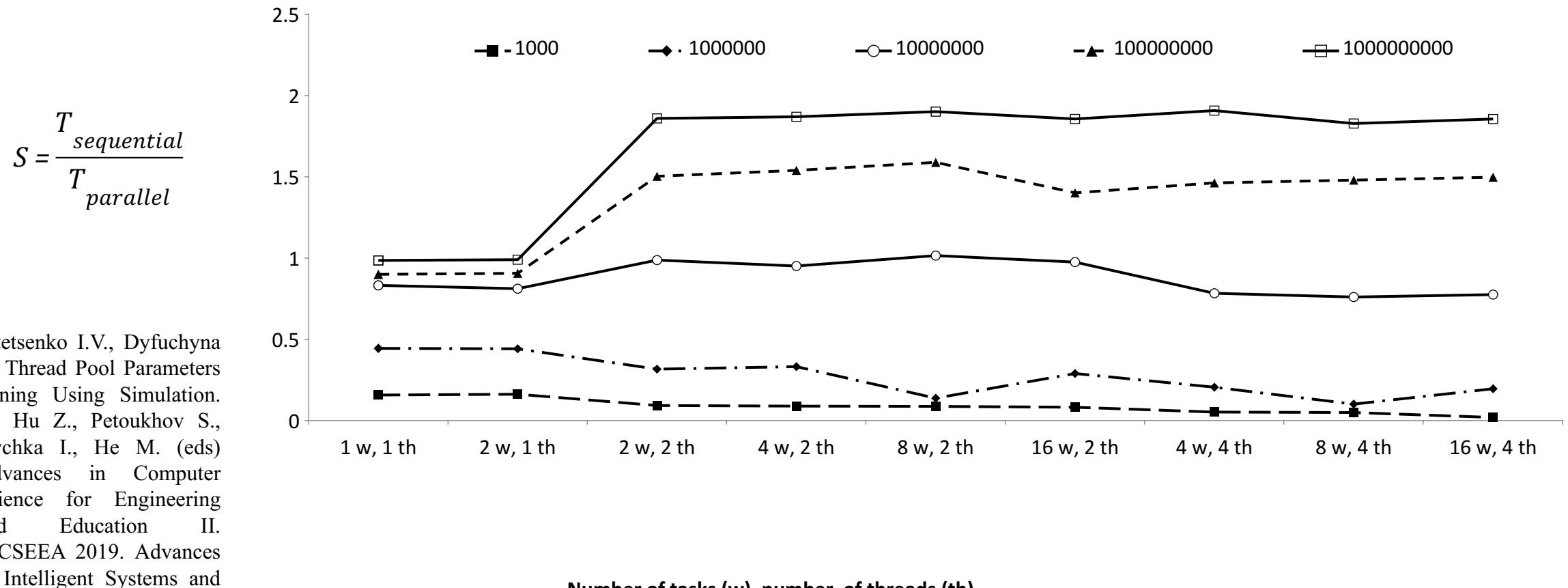


Петрі-об'єкт Counter



Експериментальне дослідження Java Thread Pool

Залежність прискорення від кількості потоків та кількості задач для різної складності обчислень, що запускаються на обчислення в пул потоків (1-ядерний комп'ютер)



[Stetsenko I.V., Dychyna O. Thread Pool Parameters Tuning Using Simulation. In: Hu Z., Petoukhov S., Dychka I., He M. (eds) Advances in Computer Science for Engineering and Education II. ICCSEEA 2019. Advances in Intelligent Systems and Computing, vol 938, Springer, Cham., pp.78-89.]

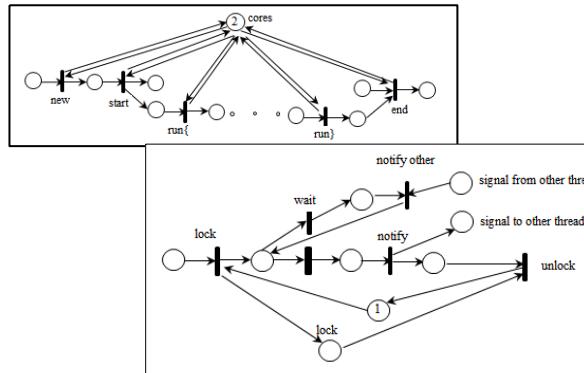
Точність результатів моделювання

Number of tasks (w) and threads (th)	1 w, 1 th	2 w, 1 th	2 w, 2 th	4 w, 2 th	8 w, 2 th	16 w, 2 th	4 w, 4 th	8 w, 4 th	16 w, 4 th
Program	0.985	0.990	1.859	1.869	1.901	1.856	1.907	1.828	1.855
Model	0.998	0.998	1.989	1.984	1.976	1.977	1.985	1.977	1.960
Accuracy	1%	1%	7%	6%	4%	7%	4%	8%	6%

Проектування паралельних програм

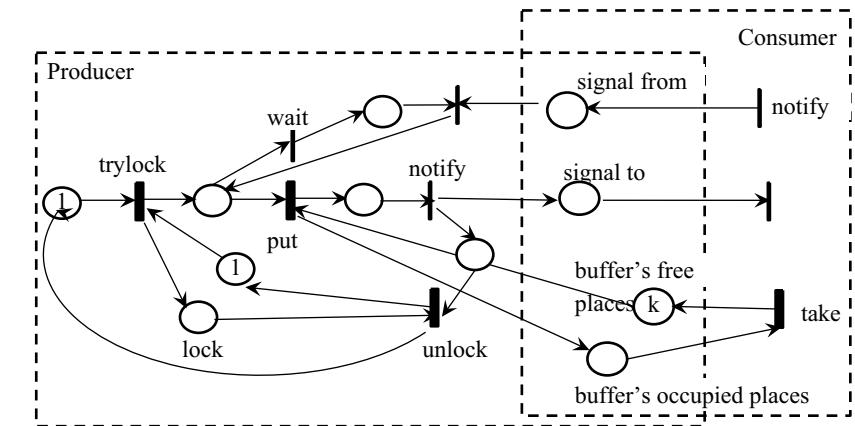
Програмний код

Мережі Петрі-об'єктів



```
private final Lock lock = new ReentrantLock();
try{
    lock = lock.tryLock();
    ...// synchronized actions
} finally {
    lock.unlock();
}
```

Петрі-об'єктна модель



- Виявлення помилок в управлінні потоками
- Оцінювання продуктивності програми при різній кількості обчислювального ресурсу

Методи дослідження імітаційних моделей систем

- Методи дослідження впливу факторів
 - Факторний експеримент
- Методи оптимізації
 - Метод градієнтного підйому
 - Еволюційні методи

Факторний експеримент

Відгук моделі – вихідна характеристика моделі, що досліджується в експерименті. *Фактор* – вхідна змінна моделі, вплив якої на відгук моделі досліджується в експерименті.

Дослідник задає:

- кількість варійованих факторів k (2, 3, 4 або 5);
- кількість рівнів для кожного фактору q (2 у випадку припущення лінійної залежності відгуку моделі від факторів);
- область змінювання факторів $\Delta x_j, j = 1..k$;
- необхідну точність ε та довірчу ймовірність β вимірювання відгуку моделі у.

На етапі **тактичного планування** експерименту визначають такі умови експерименту, що забезпечать вимір відгуку моделі з заданою точністю та довірчою ймовірністю. У результаті цього етапу планування повинні бути визначені тривалість одного прогону (в одиницях модельного часу) та кількість прогонів r .

На етапі **стратегічного планування** експериментів визначають таку серію експериментів, що забезпечить отримання бажаної інформації про відгук моделі при мінімальних витратах. У результаті цього етапу планування потрібно визначити тип аналізу, який буде використаний, та побудувати матрицю планування відповідно до обраного типу аналізу та заданої кількості факторів.

Тактичне планування факторних експериментів

1. Визначення кількості прогонів

за нерівністю Чебишева

$$P = \frac{\sigma^2}{\varepsilon^2(1 - \beta)}$$

де σ^2 – дисперсія відгуку моделі, β – довірча ймовірність, ε – точність вимірювання відгуку моделі

за центральною граничною теоремою

$$P = \frac{\sigma^2 \cdot t_\varphi}{\varepsilon^2}$$

де σ^2 – дисперсія відгуку моделі, ε – точність вимірювання. t_φ - аргумент функції Лапласа такий, що ,

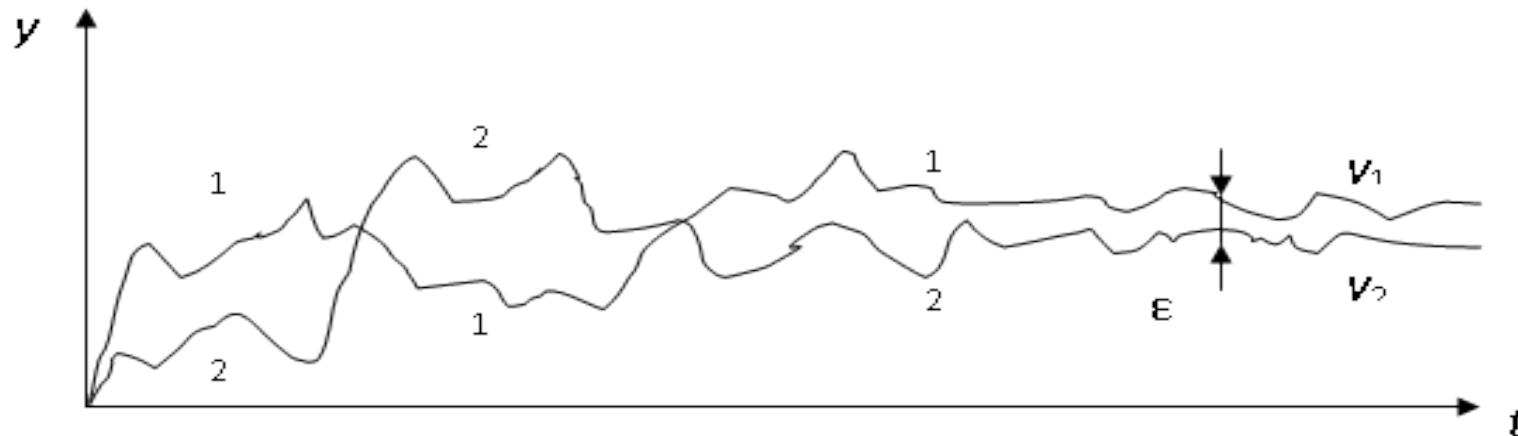
Тактичне планування факторних експериментів

Порівняння оцінки кількості прогонів за нерівністю Чебишева та за центральною граничною теоремою

ε	$\frac{\sigma^2}{\varepsilon^2(1-0.95)}$	$\frac{\sigma^2 \cdot 1.96^2}{\varepsilon^2}$
σ	20	4
$\sigma/2$	80	15
$\sigma/4$	320	61
$\sigma/6$	720	138
$\sigma/8$	1280	246
$\sigma/10$	2000	384

Тактичне планування факторних експериментів

2. Визначення тривалості одного прогону



Для того, щоб зменшити вплив переходного періоду на результати моделювання, виконують такі дії:

- вибирають такі початкові умови моделі, що зменшують її переходний період;
- вилучають значення переходного періоду при обчисленні середнього значення відгуку моделі;
- використовують такі тривалі прогони, щоб кількість даних, що потрапили в переходний період, була незначною у порівнянні з кількістю даних сталого стану.

Стратегічне планування факторних експериментів

При дослідженні впливу якісних факторів метою експерименту є якісна оцінка впливу факторів, тобто відповідь на запитання «впливає чи не впливає значення фактора на відгук моделі».

При дослідженні впливу кількісних факторів метою експерименту є кількісна оцінка впливу факторів.

Для якісної оцінки впливу факторів використовують **дисперсійний** аналіз, а для кількісної оцінки впливу факторів - **регресійний** аналіз.

Регресійний аналіз впливу факторів

$$x_i = \frac{X_i - X_{i0}}{\Delta_i} \quad X_{i0} = \frac{X_{i\max} + X_{i\min}}{2} \quad \Delta_i = \frac{X_{i\max} - X_{i\min}}{2}$$

2^3	x_0	x_1	x_2	x_3	x_1x_2	x_1x_3	x_2x_3	$x_1x_2x_3$	y
1	+	+	+	+	+	+	+	+	y_1
2	+	-	+	+	-	-	+	-	y_2
3	+	+	-	+	-	+	-	-	y_3
4	+	-	-	+	+	-	-	+	y_4
5	+	+	+	-	+	-	-	-	y_5
6	+	-	+	-	-	+	-	+	y_6
7	+	+	-	-	-	-	+	+	y_7
8	+	-	-	-	+	+	+	-	y_8

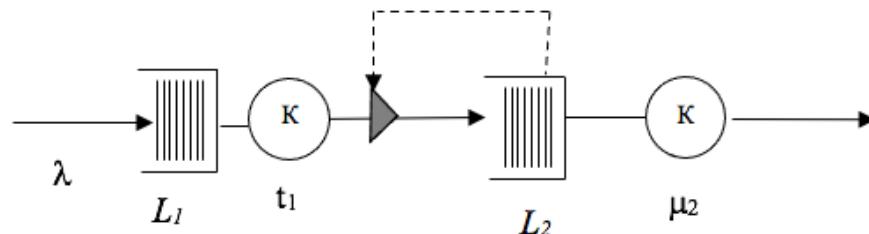
$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_1x_2 + b_5x_1x_3 + b_6x_2x_3 + b_7x_1x_2x_3$

Формула обчислення коефіцієнтів b :

$$b_k = \frac{\sum_{i=1}^N y_i x_{ik}}{N}, \quad k = 0, \dots, N-1$$

Приклад

Метою проведення експериментів з мережею МО є оцінка середньої довжини черги L у другій СМО. В ході експериментів змінювали інтенсивність надходження λ та інтенсивність обробки μ_2 .



Результати експериментів, які проводились, наведені в таблиці:

Знайдіть рівняння регресії та зробіть висновки про вплив вхідних змінних моделі на вихідну змінну.

λ	μ_2	L
20	15	4
15	15	5
20	5	18
15	5	17

Як потрібно змінювати інтенсивність надходження вимог та інтенсивність обслуговування їх в каналі другої СМО, щоб зменшити довжину черги другої СМО?

Розв'язання

Результати експериментів відповідають повному факторному плану для двох факторів.
Нехай $X_1=\lambda$, $X_2=\mu_2$. Формули нормалізації факторів приймають вигляд:

$$x_1 = \frac{X_1 - 17,5}{2,5} \quad x_2 = \frac{X_2 - 10}{5}$$

Складаємо матрицю планування ПФЕ:

2^2	x_0	x_1	x_2	x_1x_2	y
	+	+	+	+	4
	+	-	+	-	5
	+	+	-	-	18
	+	-	-	+	17

Складаємо рівняння регресії:

$$b_0 = \frac{\sum y}{4} = \frac{4 + 5 + 18 + 17}{4} = 11$$

$$b_1 = \frac{\sum yx_1}{4} = \frac{4 - 5 + 18 - 17}{4} = 0$$

$$b_2 = \frac{\sum yx_2}{4} = \frac{4 + 5 - 18 - 17}{4} = -\frac{13}{2}$$

$$b_3 = \frac{\sum yx_1x_2}{4} = \frac{4 - 5 - 18 + 17}{4} = -\frac{1}{2}$$

$$y = 11 - \frac{13}{2}x_2 - \frac{1}{2}x_1x_2$$

З рівняння регресії слідує, що інтенсивність вхідного потоку вимог (фактор x_1) мало впливає на довжину черги в СМО2 (відгуку y), а інтенсивність обслуговування (фактор x_2) – сильно впливає. Знак «мінус» свідчить про те, що при збільшенні значення фактору x_2 значення y буде зменшуватися. Для зменшення довжини черги потрібно збільшувати інтенсивність обслуговування в другій СМО.

Статистична обробка результатів факторних експериментів

- 1) Оцінка відтворюваності експерименту
- 2) Оцінка значимості коефіцієнтів регресії
- 3) Оцінка адекватності рівняння регресії
(після відкидання незначимих коефіцієнтів)

Приклад

В результаті проведення повного факторного експерименту типу 2^3

при кількості прогонів чотири отримані наступні середні значення відгуку моделі y_i , та значення дисперсій D_i :

2^3	x_1	x_2	x_3	y_{j1}	y_{j2}	y_{j3}	y_{j4}
	-	-	-	174,6	181,0	167,7	207,7
	+	-	-	102,0	159,9	102,6	111,7
	-	+	-	211,8	191,8	198,8	152,0
	+	+	-	115,4	103,3	137,2	137,2
	-	-	+	264,0	286,0	313,7	283,0
	+	-	+	275,2	254,9	201,3	232,9
	-	+	+	287,5	308,6	279,7	288,7
	+	+	+	189,4	251,3	178,2	202,3

Проведіть статистичну обробку результатів факторного експерименту та зробіть висновки про вплив факторів на відгук моделі.

Розв'язання

y_{j1}	y_{j2}	y_{j3}	y_{j4}	$y_j = \frac{1}{4} \sum_{i=1}^4 y_{ji}$	$D_j = \frac{1}{4-1} \sum_{i=1}^4 (y_{ji} - y_j)^2$
174,6	181,0	167,7	207,7	182,8	306,2
102,0	159,9	102,6	111,7	119,0	759,9
211,8	191,8	198,8	152,0	188,6	663,5
115,4	103,3	137,2	137,2	123,3	282,8
264,0	286,0	313,7	283,0	286,7	420,1
275,2	254,9	201,3	232,9	241,1	1003,2
287,5	308,6	279,7	288,7	291,1	150,9
189,4	251,3	178,2	202,3	205,3	1037,8

Кількість прогонів складає чотири, а кількість експериментів вісім, тому у формулах (6.20-6.22) $p=4$, а $N=8$.

1) Знаходимо значення критерію Кочрена за формулами (6.20), (6.21):

$$D_{\max} = 1037,8$$

$$D_{\Sigma} = 4624,4$$

$$G = \frac{1037,8}{4624,4} \approx 0,224$$

При $\alpha=0,05$, $p-1=4-1=3$, маємо $G_{kp}=0,4377$. Оскільки $0,224 < 0,4377$, то факторний експеримент є відтворюваним і величина $D = \frac{4624,4}{8} = 578,05$ є оцінкою дисперсії генеральної сукупності.

2) Знаходимо коефіцієнти рівняння регресії за формулами (6.19):

$$b_0 = \frac{\sum y_i}{8} = 204,73 \quad b_3 = \frac{\sum x_{i3}y_i}{8} = 51,31$$

$$b_1 = \frac{\sum x_{i1}y_i}{8} = -32,56 \quad b_4 = \frac{\sum x_{i1}x_{i2}y_i}{8} = 11,10$$

$$b_2 = \frac{\sum x_{i2}y_i}{8} = -2,66 \quad b_5 = \frac{\sum x_{i1}x_{i3}y_i}{8} = -0,30$$

$$b_6 = \frac{\sum x_{i2}x_{i3}y_i}{8} = -5,18$$

$$b_7 = \frac{\sum x_{i1}x_{i2}x_{i3}y_i}{8} = -4,83$$

Таблиця значень критерію Кочрена

Кількість
прогонів-1

Кількість
експери-
ментів

		Рівень значимості $\alpha = 0,05$							
		1	3	6	10	16	36	144	∞
f $k $	2	0.9985	0.9392	0.8534	0.7880	0.7341	0.6602	0.5813	0.5000
	3	0.9669	0.7977	0.6771	0.6025	0.5466	0.4748	0.4031	0.3333
	4	0.9065	0.6841	0.5598	0.4884	0.4366	0.3720	0.3093	0.2500
	5	0.8412	0.5981	0.4783	0.4118	0.3645	0.3066	0.2513	0.2000
	6	0.7808	0.5321	0.4184	0.3568	0.3135	0.2612	0.2129	0.1667
	7	0.7271	0.4800	0.3726	0.3154	0.2756	0.2278	0.1833	0.1429
	8	0.6798	0.4377	0.3362	0.2829	0.2462	0.2022	0.1616	0.1250
	9	0.6385	0.4027	0.3067	0.2568	0.2226	0.1820	0.1446	0.1111
	10	0.6020	0.3733	0.2823	0.2353	0.2032	0.1655	0.1308	0.1000
	15	0.4709	0.2758	0.2034	0.1671	0.1429	0.1144	0.0889	0.0667
	20	0.3894	0.2205	0.1602	0.1303	0.1108	0.0879	0.0675	0.0500
	30	0.2929	0.1593	0.1137	0.0921	0.0771	0.0604	0.0457	0.0333
	40	0.2370	0.1259	0.0887	0.0713	0.0595	0.0462	0.0347	0.0250
	60	0.1737	0.0895	0.0623	0.0497	0.0411	0.0316	0.0234	0.0167
	120	0.0998	0.0495	0.0337	0.0260	0.0218	0.0165	0.0120	0.0083
	∞	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Складаємо відповідне до матриці планування рівняння регресії:

$$y^{pez} = 204,73 - 32,56x_1 - 2,66x_2 + 51,31x_3 + 11,10x_1x_2 - 0,30x_1x_3 - 5,18x_2x_3 - 4,83x_1x_2x_3$$

При $\alpha=0,05$, числі ступенів вільності $N(p-1)=8\cdot3=24$, маємо $t_{kp}=2,06$. Оцінюємо значимість коефіцієнтів b_j за формулою (6.22):

$$t_0 = 204,73 \sqrt{\frac{8 \cdot 4}{578,05}} = 48,17 > 2,06 \Rightarrow b_0 \text{ є значимим,}$$

$$t_1 = 7,66 > 2,06 \Rightarrow b_1 \text{ є значимим,}$$

$$t_2 = 0,63 < 2,06 \Rightarrow b_2 \text{ є незначимим,} \quad t_3 = 12,07 > 2,06 \Rightarrow b_3 \text{ є значимим,}$$

$$t_4 = 2,61 > 2,06 \Rightarrow b_4 \text{ є значимим,} \quad t_5 = 0,07 < 2,06 \Rightarrow b_5 \text{ є незначимим,}$$

$$t_6 = 1,21 < 2,06 \Rightarrow b_6 \text{ є незначимим,} \quad t_7 = 1,14 < 2,06 \Rightarrow b_7 \text{ є незначимим.}$$

Викреслюємо незначущі доданки з виразу (6.31) і одержуємо наступне рівняння регресії:

$$y^{pez} = 204,73 - 32,56x_1 + 51,31x_3 + 11,10x_1x_2. \quad (6.32)$$

Таблиця значень критерію Стьюдента

Кількість
експериментів*
(Кількість прогонів-1)

Число ступенів свободи f	Рівень значимості α		
	0,10	0,05	0,01
1	6,31	12,70	63,70
2	2,92	4,30	9,92
3	2,35	3,18	5,84
4	2,13	2,78	4,60
5	2,01	2,57	4,03
6	1,94	2,45	3,71
7	1,89	2,36	3,50
8	1,86	2,31	3,36
9	1,83	2,26	3,25
10	1,81	2,23	3,17
11	1,80	2,20	3,11
12	1,78	2,18	3,05
13	1,77	2,16	3,01
14	1,76	2,14	2,98
15	1,75	2,13	2,95
16	1,75	2,12	2,92
17	1,74	2,11	2,90
18	1,73	2,10	2,88
19	1,73	2,09	2,86
20	1,73	2,09	2,85
21	1,72	2,08	2,83
22	1,72	2,07	2,82
23	1,71	2,07	2,81
24	1,71	2,06	2,80
25	1,71	2,06	2,79
26	1,71	2,06	2,78
27	1,71	2,05	2,77
28	1,70	2,05	2,76
29	1,70	2,05	2,76
30	1,70	2,04	2,75
40	1,68	2,02	2,70
60	1,67	2,00	2,66
120	1,66	1,98	2,62
∞	1,64	1,96	2,58

x_1	x_2	x_3	y	y^{reg}
-	-	-	182,8	197,1
+	-	-	119,0	109,8
-	+	-	188,6	174,9
+	+	-	123,3	132,0
-	-	+	286,7	299,7
+	-	+	241,1	212,4
-	+	+	291,1	277,5
+	+	+	205,3	234,6

3) Знаходимо значення адекватності дисперсії адекватності за формулою (6.23):

$$D_{ad} = \frac{1}{8-4} \sum_{i=1}^8 (y_i - y_i^{reg})^2 = \frac{2593,8}{4} = 648,45$$

Обчислюємо значення критерію Фішера за формулою (6.24):

$$F = \frac{D_{ad}}{D} = \frac{648,45}{578,05} = 1,12$$

Критичне значення критерію Фішера при рівні значимості 0,05 і кількості ступенів вільності $8 \cdot (4-1)=24$ та $4-1=3$ складає

$$F_{kp}=qF(0.95,24,3)=8,64.$$

Оскільки $F < F_{kp}$, рівняння регресії (6.32) визнається адекватним результатам факторного експерименту.

Дисперсійний аналіз впливу факторів

Метою дисперсійного аналізу являється визначення впливу фактора на рівні «впливає» або «не впливає». Основне питання, на яке дає відповідь дисперсійний аналіз впливу фактора, формулюється так: різниця у значеннях відгуку моделі, отриманих при різних значеннях фактора обумовлена випадковістю, чи пояснюється виключно дією фактора?

Ідея факторного експерименту $y_{ij} = \bar{y} + \delta_j + \varepsilon_{ij}$

Дисперсійний аналіз впливу факторів

$$d_{\text{общ}} = p \cdot \sum_{j=1}^2 \left(\bar{y}_j - \bar{y} \right)^2, \quad d_{\text{част}} = \frac{\sum_{i=1}^p \left(y_{i1} - \bar{y}_1 \right)^2 + \sum_{i=1}^p \left(y_{i2} - \bar{y}_2 \right)^2}{2(p-1)}$$

$$F = \frac{d_{\text{общ}}}{d_{\text{част}}}$$

Критичне значення критерію Фішера F_{kp} знаходять на основі рівня значимості $\alpha = 0,05$, ступенів вільності числівника $k_1 = q - 1 = 1$ та ступенів вільності знаменника $k_2 = q(p - 1) = 2(p - 1)$. Якщо $F > F_{kp}$, то середні \bar{y}_1 та \bar{y}_2 розрізняються значимо і вплив фактора є значимим. В протилежному випадку вплив фактора визнається не значимим, а різниця у середніх значеннях пояснюється випадковими причинами.

Приклад

Результати експерименту наведені у таблиці:

Визначить, чи впливає даний фактор на відгук моделі?

x=5	33,8	11,7	28,3	10,1	10,7	14,1	6,0	9,5
x=7	4,4	11,0	1,9	6,4	8,1	23,1	23,7	44,4

Розв'язання. Кількість прогонів складає 8. Розрахуємо середні значення за формулами (6.26):

$$\bar{y}_1 = \frac{1}{8} \sum_{i=1}^8 y_{i1} = 15,525, \quad \bar{y}_2 = \frac{1}{8} \sum_{i=1}^8 y_{i2} = 15,375, \quad \bar{y} = \frac{1}{2} \sum_{j=1}^2 \bar{y}_j = 15,45.$$

Для оцінки впливу фактору виконаємо дисперсійний аналіз. За формулами (6.27) знайдемо величини:

$$S_{\text{факт}} = 8 \cdot \sum_{j=1}^2 (\bar{y}_j - \bar{y})^2 = 0,09, \quad S_{\text{залиш}} = \sum_{i=1}^p (y_{i1} - \bar{y}_1)^2 + \sum_{i=1}^p (y_{i2} - \bar{y}_2)^2 = 2120.$$

Потім за формулами (6.29) визначимо факторну та залишкову дисперсії:

$$d_{\text{факт}} = S_{\text{факт}} = 0,09, \quad d_{\text{залиш}} = \frac{S_{\text{залиш}}}{2(8-1)} = 151,404.$$

Приклад

Значення критерію Фішера визначається за формулою (6.30):

$$F = \frac{d_{\text{факт}}}{d_{\text{залиш}}} = 0,00059$$

Критичне значення критерію Фішера знайдемо зі статистичних таблиць при значеннях параметрів $\alpha = 0,05$, $k_1 = 1$ та $k_2 = 2(p-1) = 14$:

$$F_{kp} = 4,6.$$

Оскільки $F < F_{kp}$, то вплив фактора є незначущим. Різниця в значеннях відгуку моделі пов'язана з випадковим її характером і не пов'язана зі зміною значення фактора.

Паралельні обчислення в алгоритмах імітації

Прискорення експериментальних досліджень

- кілька прогонів однакової моделі виконують одночасно для отримання більш точного результату (див. тактичне планування експериментів):
 - прогони виконуються в окремих потоках, основний метод очікує завершення роботи всіх потоків, потім виконує обчислення середнього значення
- кілька моделей з різними параметрами виконують одночасно (див. стратегічне планування експериментів, оптимізація еволюційними методами):
 - основний метод здійснює обчислення параметрів, при яких виконується моделювання, та обробку результатів імітації
 - імітація моделей для різних наборів параметрів здійснюється в окремих потоках

Графічний інтерфейс

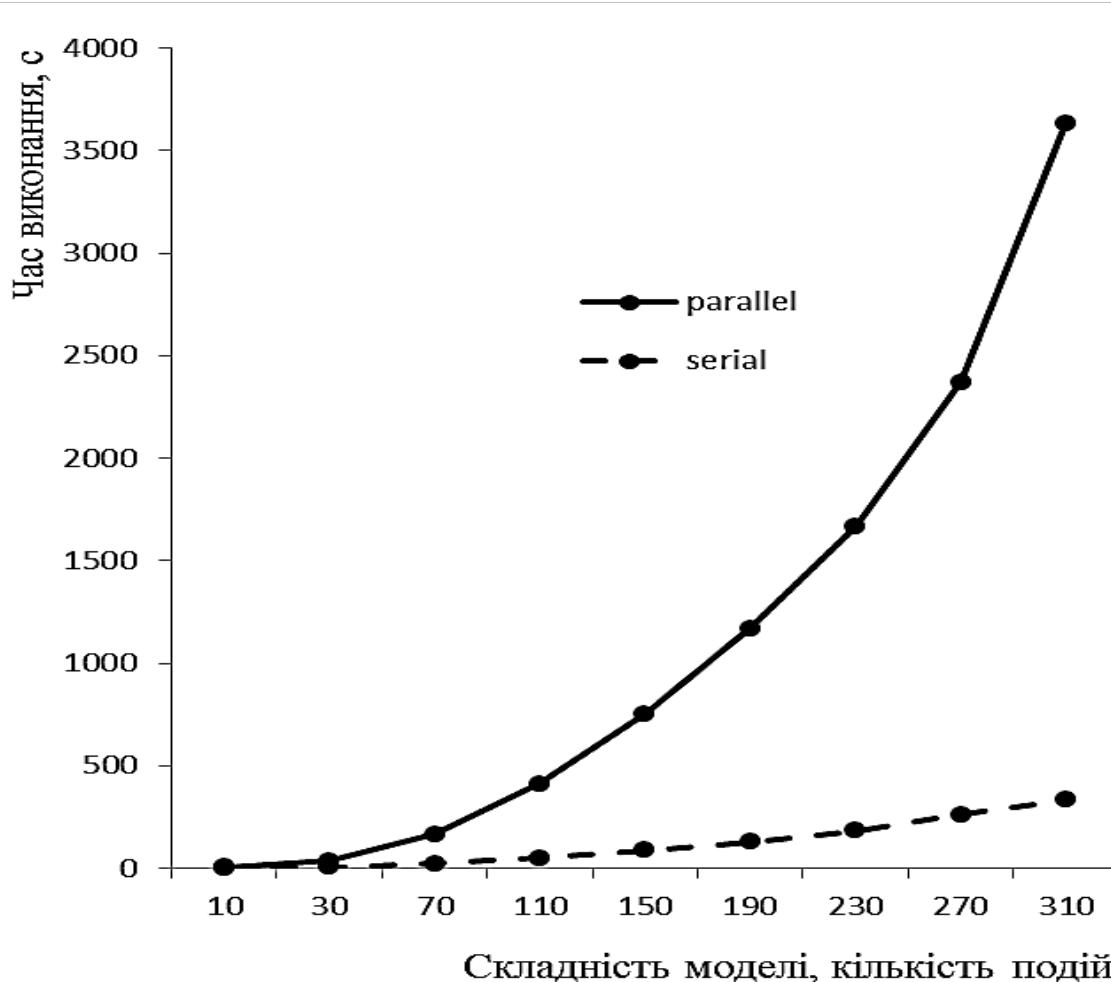
- Динамічне відтворення результатів моделювання одночасно з обчисленням моделі:
 - в одному потоці відбувається імітація, інший з затримкою виводить поточні значення вихідних характеристик
- Анімація моделювання одночасно з обчисленням моделі:
 - перемальовування елементів моделі виконується з затримкою

Прискорення виконання алгоритму імітації

Варіанти розпаралелювання:

- Відшукати незалежні частини моделі, що можна виконувати одночасно
 - Проте виявлення таких частин є самостійною складною задачею
- Виконувати розпаралелювання в межах операцій, що часто повторюються. Наприклад, для мережі Петрі можна виконувати одночасно перевірку запуску переходів та вхід маркерів в переходи
 - Накладні витрати на створення потоків (або задач) перевищують вигранш в часі, отриманий за рахунок одночасного виконання задач
- Виконувати імітацію окремих фрагментів моделі одночасно
 - Через використання спільної змінної часу алгоритм тільки сповільнюється

Ефективність паралельного алгоритму імітації у випадку глобальної змінної часу



Прискорення виконання алгоритму імітації

Варіант розпаралелювання:

- Виконувати імітацію окремих фрагментів моделі одночасно не використовуючи спільну змінну часу

Варіант розпаралелювання Петрі-об'єктної моделі:

- Фрагменти моделі – це Петрі-об'єкти, з яких вона складається
- Кожний Петрі-об'єкт відтворює своє функціонування з урахуванням подій в Петрі-об'єктах, з якими він пов'язаний.
- Інформація про події надходить від об'єктів і розглядається як зовнішня

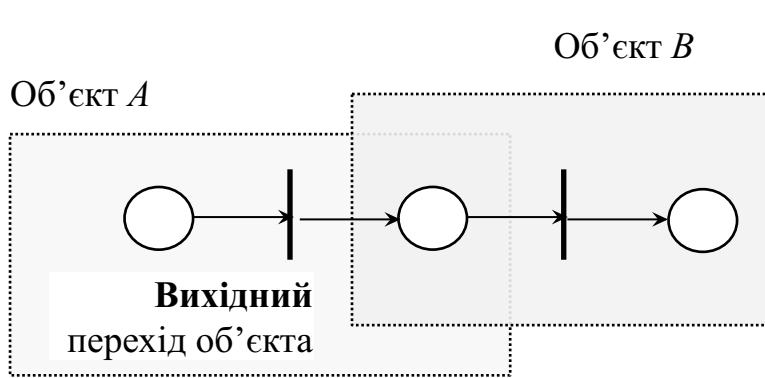
Паралельний алгоритм імітації Петрі-об'єктної моделі

Означення:

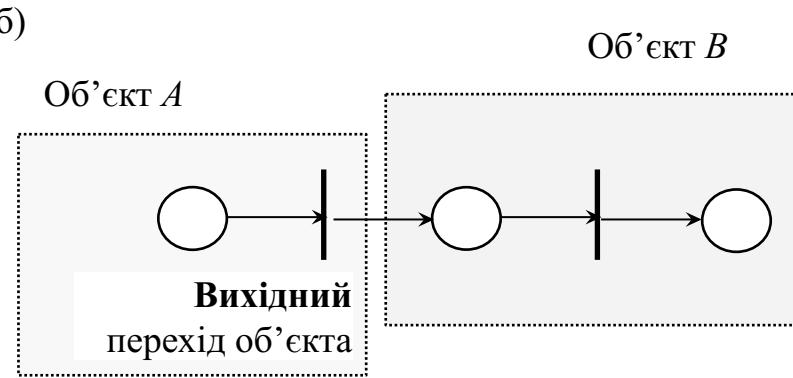
- Переход мережі Петрі-об'єкта є вихідним, якщо при його запуску здійснюється вихід у спільну позицію з іншим об'єктом або здійснюється вихід у позицію іншого об'єкта (рис.1 а,б).
- Позиція об'єкта є вхідною, якщо вона є спільною з позицією іншого об'єкта або у неї здійснюється вихід з переходу іншого об'єкта (рис.1 в,г).
- Якщо об'єкт A має вихідний переход, з якого здійснюється вихід маркерів у позицію об'єкта B, то об'єкт B є next-об'єктом для об'єкта A
- Якщо об'єкт B має позицію, в яку здійснюється вихід маркерів з вихідного переходу об'єкта A, то об'єкт A є previous-об'єктом для об'єкта B.
- Безпечним інтервалом імітації об'єкта є інтервал часу між двома послідовними виходами маркерів з вихідного переходу його previous-об'єкта.

Взаємодія Петрі-об'єктів

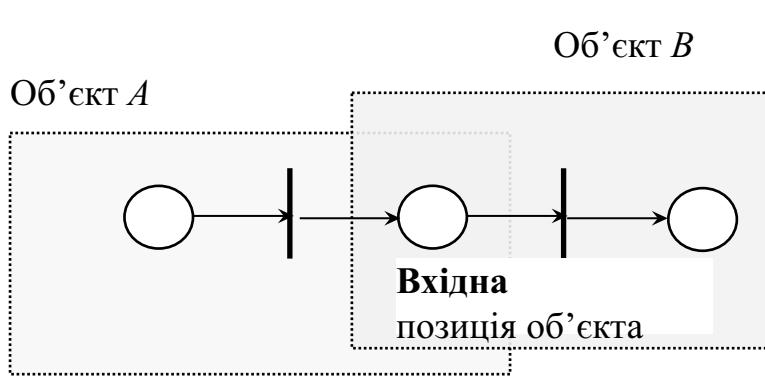
а)



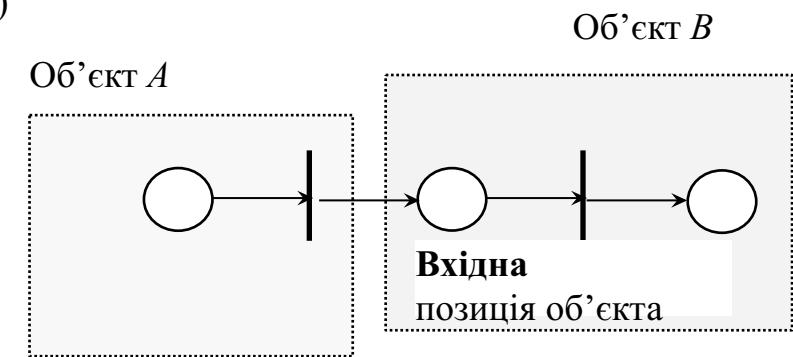
б)



в)



г)



а) вихід маркерів з переходу об'єкта у спільну позицію його next-об'єкта, б) вихід маркерів з переходу об'єкта у позицію його next-об'єкта, в) вхід маркерів у спільну позицію об'єкта з його previous-об'єкта, г) вхід маркерів у позицію об'єкта з переходу його previous-об'єкта.

Правила паралельного функціонування Петрі-об'єктів

1. Кожний Петрі-об'єкт здійснює імітацію в локальному часі в окремому потоці.
2. Кожний об'єкт, який має next-об'єкт, передає йому інформацію про моменти виходу маркерів з об'єкта і призупиняє своє функціонування при значному накопиченні такої інформації.
3. Кожен об'єкт, який має previous-об'єкт, здійснює імітацію в межах до наступної події входу в нього маркерів з previous-об'єкта, поступово просуваючи свій локальний час до повного вичерпання накопиченої інформації про вхідні події, або очікує надходження інформації про вхідні події зі свого previous-об'єкта.
4. Об'єкт, який має previous-об'єкт, при досягненні моменту часу входу маркерів в об'єкт, відновлює вихід маркерів у спільну позицію з переходу previous-об'єкта та продовжує імітацію.
5. Об'єкт, який має next-об'єкт, при досягненні моменту виходу маркерів з об'єкта, призупиняє вихід маркерів у позицію next-об'єкта. Цей вихід відновить next-об'єкт у відповідний момент часу свого функціонування.
6. Об'єкт, який вичерпав час моделювання, передає повідомлення про це next-об'єкту, якщо такий є, і завершує свою роботу. Разом з завершенням роботи об'єкта завершує роботу і потік, ним генерований.
7. Об'єкт, який отримав від previous-об'єкта сигнал про завершення його роботи і, водночас, вичерпав усі накопичені події, припиняє свою роботу.

Буфер зовнішніх подій

- зберігає моменти часу надходження маркерів з інших об'єктів, що не опрацьовані на поточний момент часу
- в межах від одного моменту зовнішньої події до наступної гарантовано не виникає зовнішніх подій, отже, об'єкт може виконувати імітацію своїх внутрішніх подій
- порожній стан буфера означає, що наступний момент зовнішньої події невідомий (на поточний момент виконання алгоритму імітації)
- стан буфера, в якому останній його елемент є нескінченність (максимально допустиме число), означає, що previous-об'єкт завершив свою роботу і надходження зовнішніх подій не очікується.

Локальний час об'єкта просувається за такими правилами

Локальний час об'єкта просувається за такими правилами:

- якщо час найближчої події менший за межу безпечного інтервалу, то просунути локальний час у момент найближчої події, виконати (внутрішні) події, для яких час збігається з поточним моментом, та продовжити імітацію об'єкта;
- інакше
 - якщо межа безпечного інтервалу менша за межу інтервалу моделювання
 - просунути локальний час на межу безпечного інтервалу та виконати зовнішню подію (відновлення маркерів у вхідних позиціях),
 - інакше просунути локальний час на межу безпечного інтервалу та завершити імітацію об'єкта

Межа безпечноого інтервалу

Межа безпечноого інтервалу визначається за наступними правилами:

- якщо немає previous-об'єкта, то межа встановлюється в час моделювання;
- інакше
 - якщо буфер зовнішніх подій не порожній і перша подія менша за час моделювання, то межа встановлюється в момент першої зовнішньої події;
 - інакше - в час моделювання.

Об'єкт

Об'єкт знаходитьться в очікуванні (призупиняє своє функціонування), якщо і тільки якщо:

- він знаходиться у stop-стані: кількість маркерів недостатня для запуску переходів і момент виходу маркерів з переходу більший за границю безпекного інтервалу і, якщо є previous-об'єкт, перша подія в буфері менша за границю безпекного інтервалу;
- є previous-об'єкт і буфер зовнішніх подій порожній;
- є next-об'єкт і його буфер зовнішніх подій перевищує заданий ліміт.

Об'єкт розблоковується (припиняє очікування) якщо і тільки якщо:

- його previous-об'єкт здійснив вихід у позицію, спільну з даним об'єктом;
- його next-об'єкт подав сигнал про відновлення роботи об'єкта у разі, якщо він вичерпав буфер подій до заданого ліміту.

Об'єкт завершує імітацію, якщо досягнутий кінець інтервалу моделювання.

Синхронізація виходу маркерів у спільну позицію

Синхронізація виходу маркерів у спільну позицію відбувається таким чином:

- у момент здійснення виходу маркерів з вихідного переходу Петрі-об'єкта вихід маркерів у позицію next-об'єкта не здійснюється, замість цього поточний часу запам'ятовується в буфері зовнішніх подій next-об'єкта;
- коли локальний час об'єкта досягає первого значення буфера подій, здійснюється вхід маркерів у вхідну позицію об'єкта, а перше значення з буфера зовнішніх подій видаляється.

Умова завершення роботи потоків

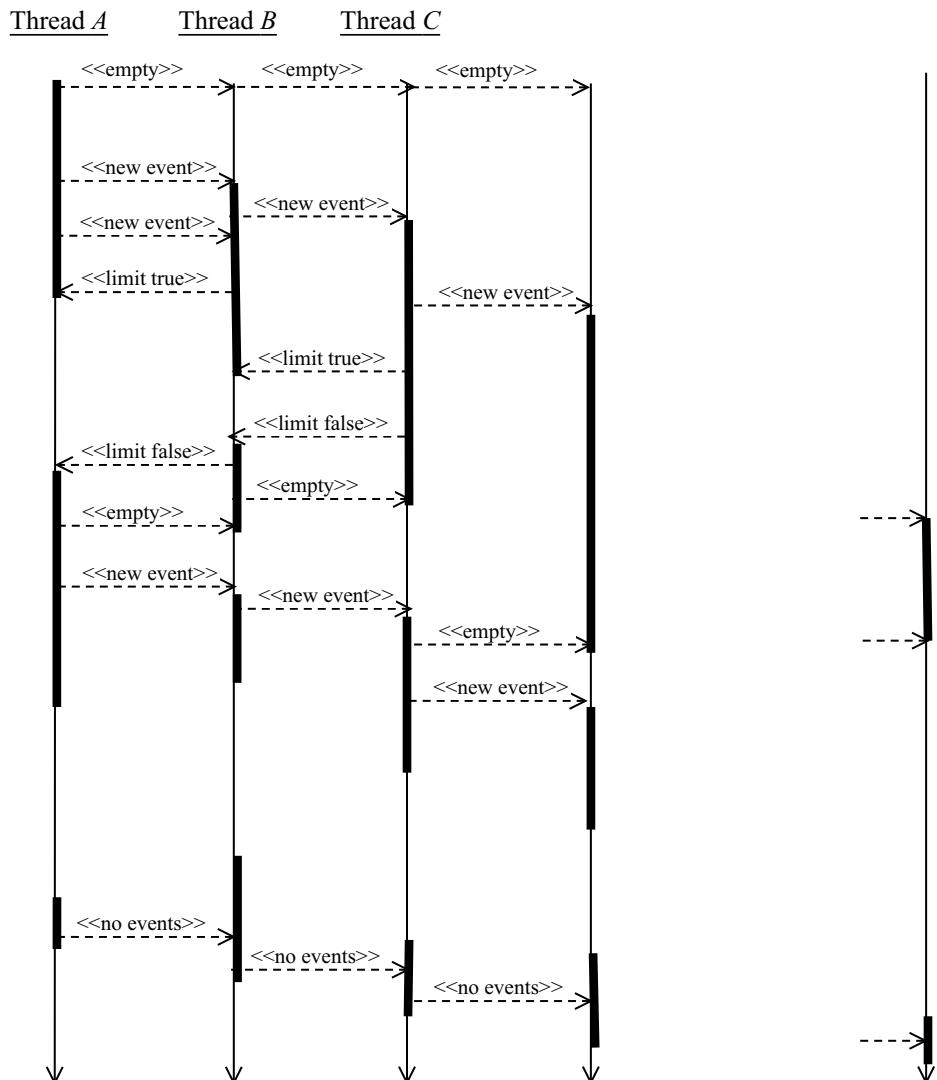
Якщо в буфері об'єкта перший елемент нескінченність, це означає, що його previous-об'єкт завершив імітацію на інтервалі імітації і зовнішніх подій більше не очікується.

Об'єкт продовжує імітацію до завершення інтервалу моделювання без очікування зовнішніх подій.

Алгоритм, описаний вище, виходить з таких припущень: модель має хоча б один об'єкт, для якого не заданий previous-об'єкт, і хоча б один об'єкт, для якого не заданий next-об'єкт. В реальних умовах це відповідає відкритій системі.

- Якщо об'єкт має декілька previous-об'єктів, то для кожного з них створюється свій буфер подій. Момент найближчої події можна визначити тільки, коли надійшла інформація про наступні події від усіх previous-об'єктів. Тому робота такого об'єкта призупиняється, якщо хоч один буфер подій порожній, і відновлюється, якщо усі буфери подій не порожні. Щоб запобігти взаємному блокуванню об'єктів, потрібно відслідковувати стан, в якому об'єкт і його previous-об'єкти одночасно знаходяться в стані очікування (коли хоч один з їх буферів подій є порожнім). В цьому випадку потрібно визначити найближчу подію для об'єкта і його previous-об'єктів, просунути локальний час в момент найближчої події і продовжити моделювання.
- Якщо об'єкт має декілька next-об'єктів, то робота такого об'єкта призупиняється, якщо досягає ліміту буфер подій хоч в одному з його next-об'єктів, і відновлюється, якщо в усіх next-об'єктах кількість подій в буфері подій менша за назначений ліміт.

Взаємодія потоків паралельного алгоритму імітації Петрі-об'єктної моделі



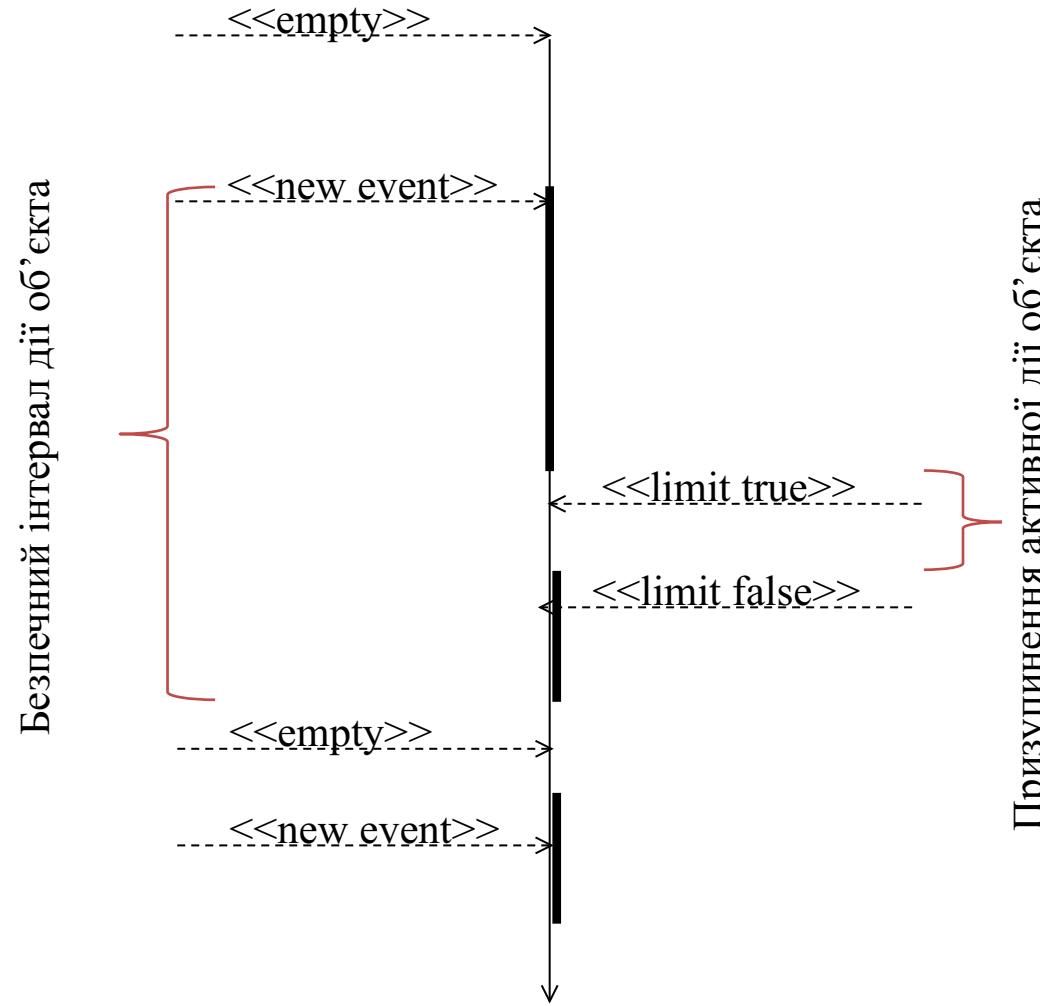
Потоки A , B , C ілюструють дію потоків, що запускають імітацію Петрі-об'єктів A , B , C таких, що:

- $A \in \text{previous-об'єктом для } B$,
- $B \in \text{next-об'єктом для } A \text{ і previous-об'єктом для } C$,
- $C \in \text{next-об'єктом для } B$.

Пара повідомень <<new event>> та <<empty>> інформує об'єкт про надходження нової події від previous-об'єкта та, навпаки, що таких подій немає. Повідомлення <<no event>> сповіщає next-об'єкт про завершення імітації об'єктом, а значить про припинення надходження подій від previous-об'єкта.

Пара повідомень <<limit true>>, <<limit false>> інформує об'єкт про перевищення ліміту буфера зовнішніх подій його next-об'єкту або, навпаки, про те, що такого перевищення немає.

Управління потоком паралельного алгоритму імітації Петрі-об'єктної моделі



Створення Петрі-об'єктів та зв'язків між ними

```
public static PetriObjModel getModelsMOgroupForTestParallel(int numGroups, int numInGroup) throws ExceptionInvalidNetStructure {
    ArrayList<PetriSim> list = new ArrayList<>();
    int numSMO = numGroups - 1;

    list.add(new PetriSim(libnet.NetLibrary.CreateNetGenerator(2.0)));
    for (int i = 0; i < numSMO; i++) {
        list.add(new PetriSim(libnet.NetLibrary.CreateNetSMOgroup(numInGroup, 1, 1.0, "group_"+i))); // group1,group2,group3...
    }
    list.get(0).getNet().getListP()[1] = list.get(1).getNet().getListP()[0]; //gen = > group1
    list.get(0).addOutT(list.get(0).getNet().getListT()[0]);
    list.get(1).addInT(list.get(1).getNet().getListT()[0]);
    list.get(0).setNextObj(list.get(1));
    list.get(1).setPreviousObj(list.get(0));
    if (numSMO > 1) {
        for (int i = 2; i <= numSMO; i++) {
            int last = list.get(i-1).getNet().getListP().length-1;

            list.get(i).getNet().getListP()[0] = list.get(i-1).getNet().getListP()[last]; //group1 = > group2, group2 = > group3, ...

            int lastT = list.get(i-1).getNet().getListT().length-1;
            list.get(i-1).addOutT(list.get(i-1).getNet().getListT()[lastT]);
            list.get(i).addInT(list.get(i).getNet().getListT()[0]);
            list.get(i-1).setNextObj(list.get(i));
            list.get(i).setPreviousObj(list.get(i-1));
        }
    }
    //корегування списку розицій для статистики
    for (int i = 1; i <= numSMO; i++) {
        ArrayList<PetriP> positionForStatistics = new ArrayList<>();
        PetriP[] listP = new PetriP[list.get(i).getNet().getListP().length];
        listP = list.get(i).getNet().getListP();
        for(int j=0;j<listP.length-1;j++){ //окрім останньої, наприклад
            positionForStatistics.add(listP[j]);
        }
        list.get(i).setListPositionsForStatistica(positionForStatistics);
    }
}

PetriObjModel model = new PetriObjModel(list);
return model;
}
```

Запуск потоків на виконання

```
PetriObjModel model = getModelsMOgroupForTestParallel(numObj,10);
model.setTimeMod(time);
PetriSim.setLimitArrayExtInputs(3);
model.setIsProtokol(true);

model.getListObj().forEach((PetriSim e) -> {
    Thread petriObj = new Thread(e);
    threads.add(petriObj);
    petriObj.start();
});

threads.forEach((thread) -> {
    try {
        thread.join();
    } catch (InterruptedException ex) {
        Logger.getLogger(TestParallel.class.getName()).log(Level.SEVERE, null, ex);
    }
});

printResultsForAllObj(model);
```

Метод run () Петрі-об'єкту

```
@Override
public void run() {
    while (getTimeLocal() < getTimeMod()) {
        double limitTime = getTimeMod(); //для об'єкта без зовнішніх подій
        if (previousObj != null) {
            this.getLock().lock();
            try {
                while (getTimeExternalInput().isEmpty()) {
                    this.getCond().await();
                }

                limitTime = getFirstTimeExternalInput(); //встановлення інтервалу імітації до моменту події входу маркерів в об'єкт ззовні
                if (limitTime > getTimeMod()) {
                    limitTime = getTimeMod();
                }
            } catch (InterruptedException ex) {
                Logger.getLogger(PetriSim.class.getName()).log(Level.SEVERE, null, ex);
            } finally {
                this.getLock().unlock();
            }
        } else { // previousObj == null
            limitTime = getTimeMod();
        }

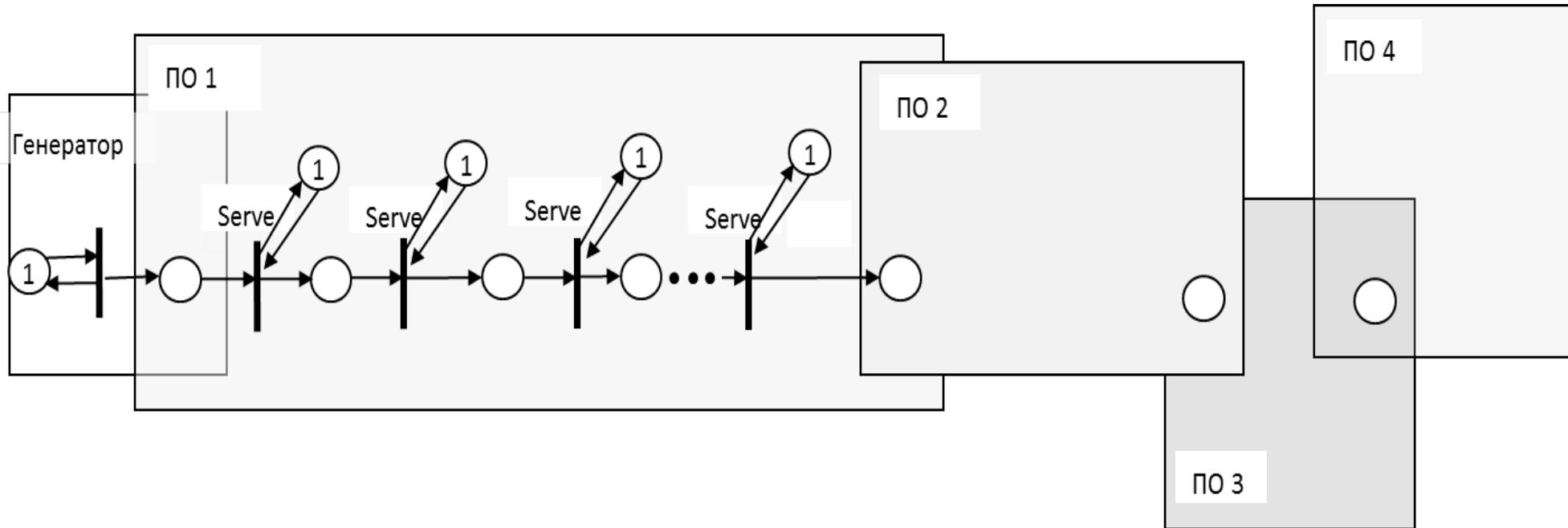
        if (getTimeLocal() < limitTime) {
            goUntil(limitTime);
        } else { // повідомлення про неправильне визначення limitTime
            return;
        }
    }
}
```

- Повний код програми

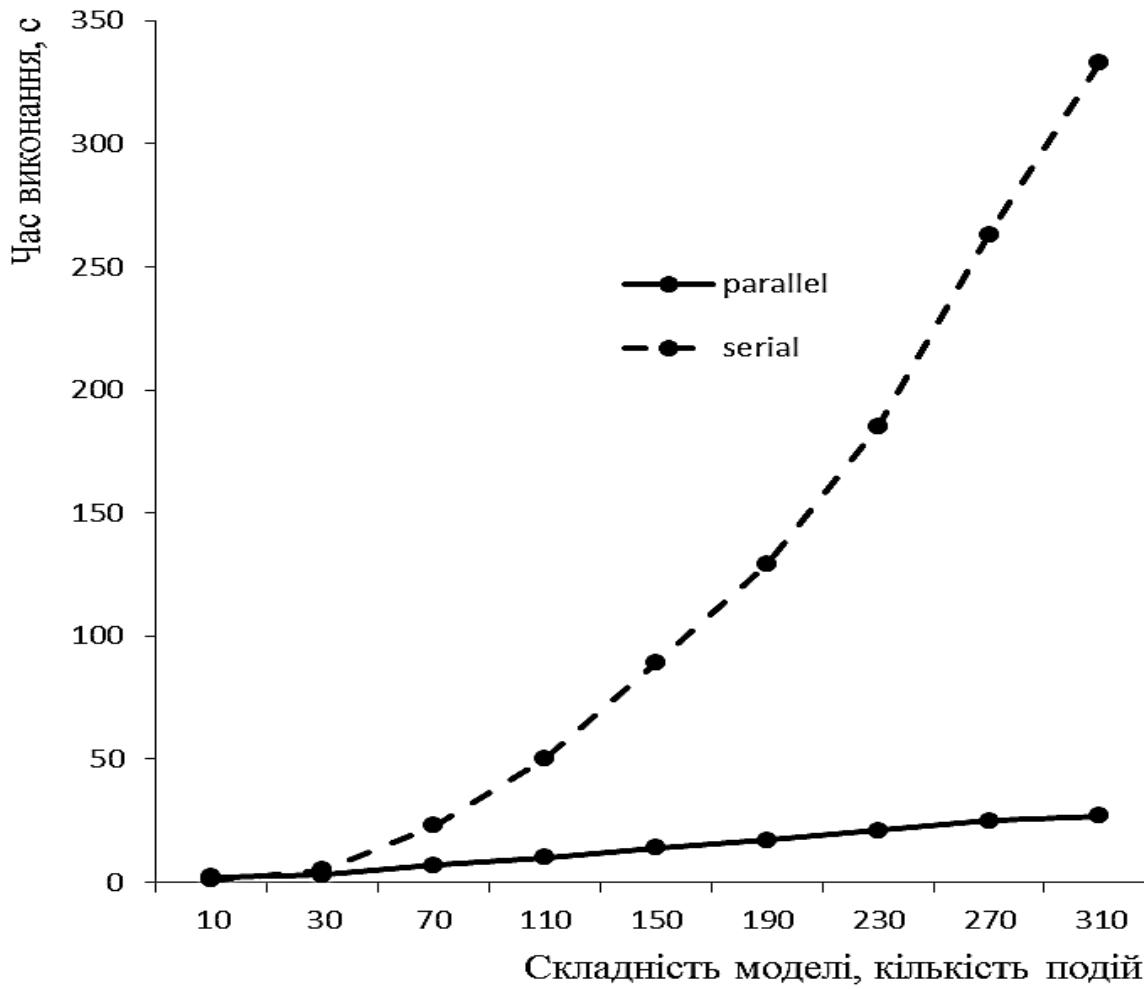
<https://github.com/StetsenkoInna/ParallelDESS>

Тестування алгоритму

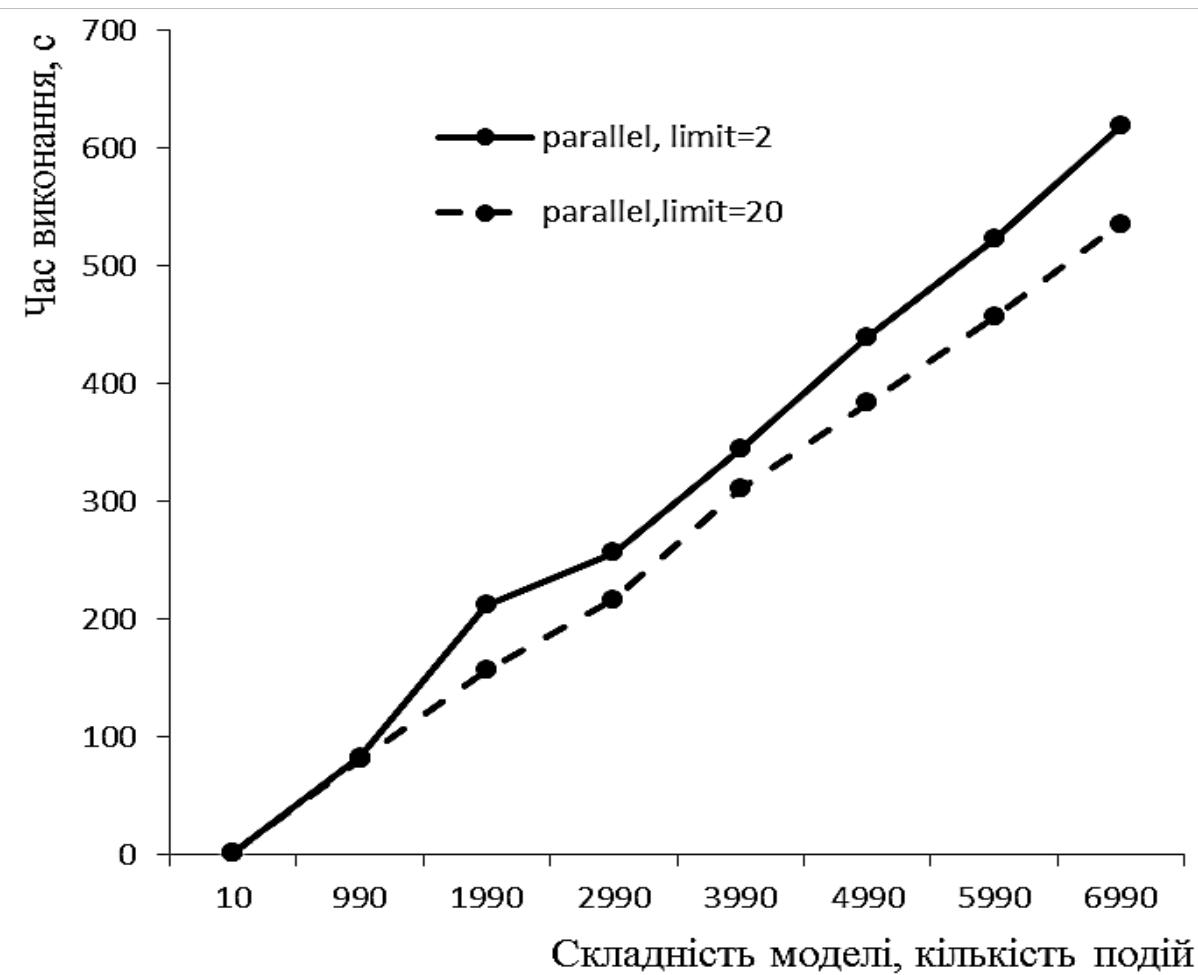
Тестова модель:



Ефективність паралельного алгоритму імітації у випадку локальної змінної часу

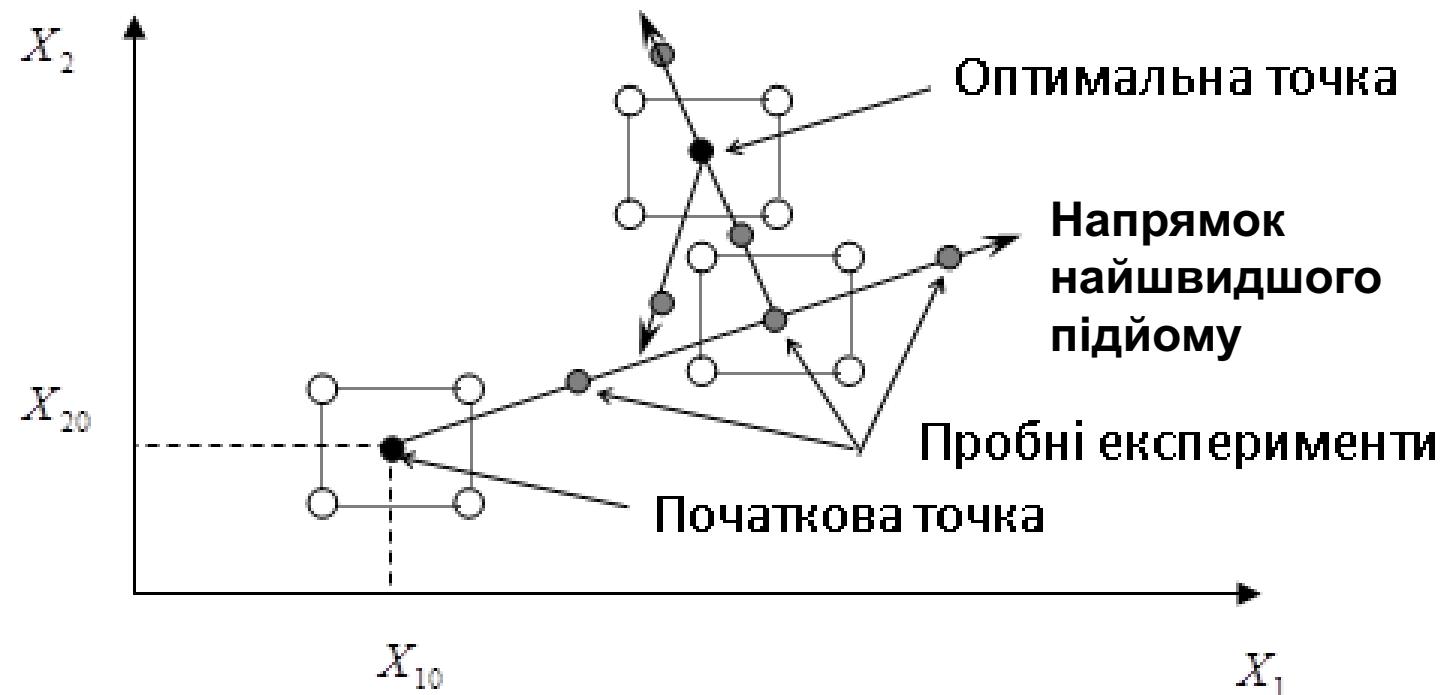


Дослідження впливу розміру буфера зовнішніх подій на час виконання алгоритму



Методи оптимізації імітаційних моделей

Методи оптимізації імітаційних моделей: Метод найшвидшого підйому



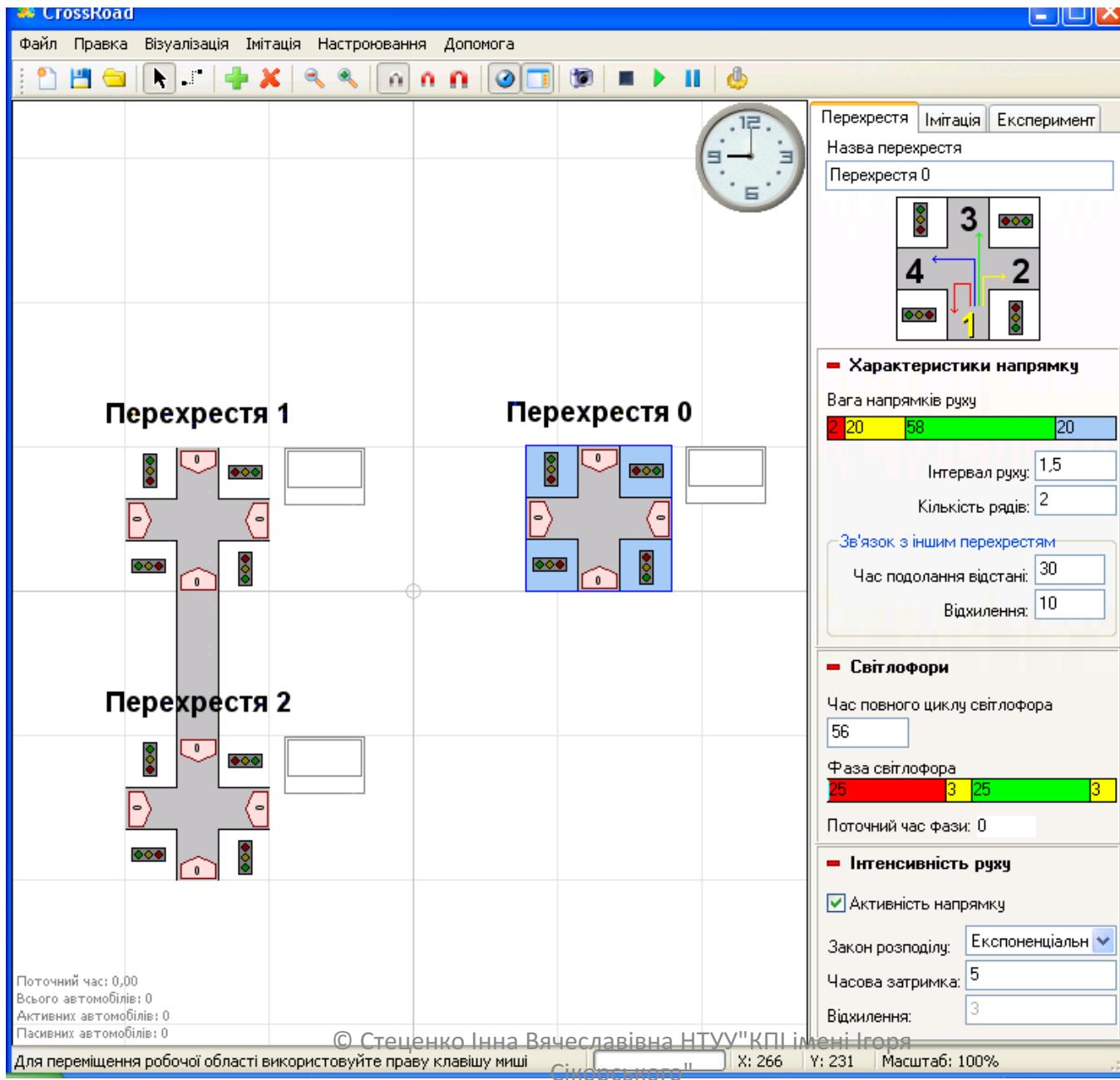
Еволюційні методи пошуку оптимального значення

- Елементом **популяції** є набір параметрів, пошук яких здійснюється.
- Початкова популяція (генерування 0) формується з випадкових значень, розкиданих в області допустимих значень параметрів.
- Кожний елемент популяції запускається у «життя», тобто в імітаційну модель. Результатом такої **життєдіяльності елемента популяції** є відгук моделі. Набори параметрів, які виявилися «неспроможними», тобто дістали в процесі імітації великі значення відгуку моделі, «гинуть» або знищуються. Таким чином за значенням відгуку моделі здійснюється відбір елементів популяції.
- Елементи популяції, що пройшли відбір, допускаються до схрещування. **Схрещування** здійснюється для випадково обраних пар елементів популяції склеюванням частин наборів параметрів. Нехай для схрещування обрані елементи популяції A_j та A_k . В результаті роботи **оператора кросовера** випадковим чином обираються компоненти, параметри яких в елементі-нащадку будуть прийняті такими, як в елементі A_j , інші компоненти елемента-нащадка приймають значення параметрів такі, як в елементі A_k :
- **Мутація** здійснюється додаванням випадкового відхилення до результату, який отриманий в результаті схрещування, наприклад, додаванням з рівною ймовірністю -1, 0 або 1.
- Кожна наступна популяція (генерування j) формується з елементів, що пройшли відбір на попередньому генеруванні (генерування $j-1$), та з елементів, що створені в результаті схрещування та мутації.
- У **правилі зупинки еволюційного пошуку** користувач задає точність визначення оптимального значення (перехід до наступної популяції не суттєво поліпшує оптимальне значення) та максимальну кількість генерувань.

Алгоритм еволюційного методу пошуку оптимального значення

- Задати $2N$ елементів початкової популяції A_0 та точність ε визначення оптимального значення.
- Доки не досягнута задана точність $(\Delta_j < \varepsilon)$ визначення оптимального значення:
 - Для кожного елемента популяції A_j розрахувати показник його життєдіяльності та визначити найліпше досягнуте на даному етапі еволюції значення
 - Упорядкувати елементи популяції за спаданням значення показника життєдіяльності та відкинути половину найгірших з них.
 - Оператором кросовера створити N нових елементів популяції, використовуючи схрещування тільки для елементів, що залишились після відбору.
 - Виконати мутацію нових елементів
 - Створити нову популяцію з N елементів, які виявились найкращими в результаті відбору, та N новостворених елементів.
 - Розрахувати Δ , тобто різницю між найліпшими значеннями показника життєдіяльності попередньої та нової популяції.
- Визначити найліпший елемент в останній популяції та значення його показника життєдіяльності

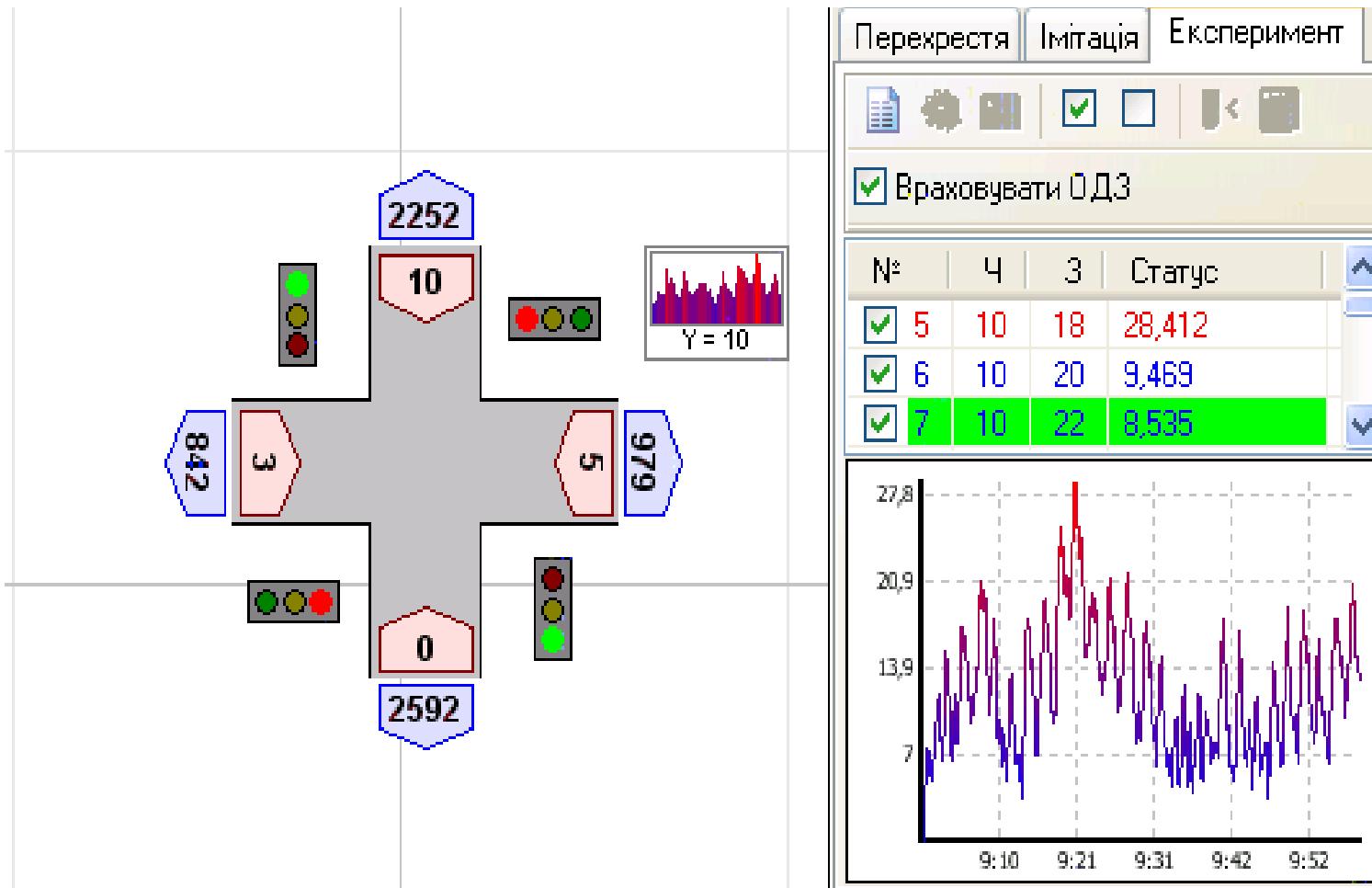
Приклад: оптимізація параметрів управління транспортним рухом



Результати імітаційного моделювання ділянки транспортного руху



Результати визначення оптимальних параметрів управління окремого перехрестя



Застосування еволюційної стратегії для визначення оптимальних параметрів управління світлофорними об'єктами

Елемент популяції

$$A = (r_1, g_1, r_2, g_2, \dots, r_n, g_n)$$

Початкова популяція: A1, A2, ...A20.

Відбір та знищення: A1, A2, ...A20 \Rightarrow імітаційна модель транспортного руху \Rightarrow y1, y2,...y20 \Rightarrow
 \Rightarrow сортування за значенням відгуку моделі \Rightarrow знищення неспроможних елементів \Rightarrow
 \Rightarrow A1, A2, ...A10

Кросовер:

$$\begin{aligned} A^{(j)} &= \left(\begin{matrix} r_1^{(j)} & g_1^{(j)} & r_2^{(j)} & g_2^{(j)} & \dots & r_{n-1}^{(j)} & g_{n-1}^{(j)} & r_n^{(j)} & g_n^{(j)} \end{matrix} \right) \\ A^{(k)} &= \left(\begin{matrix} r_1^{(k)} & g_1^{(k)} & r_2^{(k)} & g_2^{(k)} & \dots & r_{n-1}^{(k)} & g_{n-1}^{(k)} & r_n^{(k)} & g_n^{(k)} \end{matrix} \right) \\ A^{(m)} &= \left(\begin{matrix} r_1^{(j)} & g_1^{(j)} & r_2^{(k)} & g_2^{(k)} & \dots & r_{n-1}^{(k)} & g_{n-1}^{(k)} & r_n^{(j)} & g_n^{(j)} \end{matrix} \right) \end{aligned} \Rightarrow$$

Мутація

$$A = (r_1 + \xi_1, g_1 + \xi_2, r_2 + \xi_3, g_2 + \xi_4, \dots, r_n + \xi_{2n-1}, g_n + \xi_{2n})$$

Наступна популяція: A1, A2, ...A10 , A11, A12..., A20 – елементи попередньої популяції та знов створені елементи

Результати визначення оптимальних параметрів управління системи перехресть

genetic Imitation

Популяція

Початкова популяція	Янська-Гоголя	Смілянська-Благовісна	Смілянська-Лійна	Леніна-Гоголя	Благовісна-Леніна	Лійна-Леніна	Y
Популяція 1	4; 3: 8	4: 11; 3: 13	4: 17; 3: 20	4: 22; 3: 25	4: 29; 3: 32	4: 34; 3: 37	297,05
Популяція 2	4; 3: 34	4: 38; 3: 40	4: 8; 3: 11	4: 14; 3: 17	4: 20; 3: 22	4: 26; 3: 29	96,11
Популяція 3	4; 3: 26	4: 29; 3: 32	4: 35; 3: 38	4: 6; 3: 8	4: 12; 3: 15	4: 17; 3: 20	378,35
Популяція 4	4; 3: 16	4: 19; 3: 23	4: 25; 3: 28	4: 32; 3: 34	4: 37; 3: 5	4: 8; 3: 11	377,78
Популяція 5	4; 3: 8	4: 11; 3: 14	4: 16; 3: 20	4: 23; 3: 25	4: 29; 3: 32	4: 35; 3: 37	304,04
	4; 3: 34	4: 36; 3: 40	4: 8; 3: 10	4: 13; 3: 17	4: 19; 3: 22	4: 26; 3: 28	162,32
	4; 3: 25	4: 28; 3: 31	4: 34; 3: 37	4: 40; 3: 8	4: 10; 3: 14	4: 17; 3: 20	446,15
	4; 3: 16	4: 19; 3: 23	4: 25; 3: 28	4: 31; 3: 34	4: 37; 3: 5	4: 7; 3: 11	356,23
	4; 3: 8	4: 11; 3: 14	4: 17; 3: 20	4: 23; 3: 26	4: 28; 3: 32	4: 35; 3: 38	92,91
	4; 3: 34	4: 36; 3: 40	4: 8; 3: 10	4: 13; 3: 17	4: 20; 3: 22	4: 25; 3: 29	89,27
	4; 3: 23	4: 27; 3: 30	4: 32; 3: 35	4: 39; 3: 6	4: 9; 3: 13	4: 15; 3: 18	479,82
	4; 3: 15	4: 18; 3: 22	4: 24; 3: 27	4: 30; 3: 32	4: 36; 3: 39	4: 6; 3: 10	341,29
	4; 3: 6	4: 10; 3: 13	4: 15; 3: 19	4: 22; 3: 24	4: 27; 3: 31	4: 34; 3: 36	416,44
	4; 3: 33	4: 36; 3: 38	4: 7; 3: 10	4: 12; 3: 15	4: 19; 3: 21	4: 24; 3: 28	74,06
	4; 3: 23	4: 27; 3: 30	4: 33; 3: 35	4: 39; 3: 7	4: 9; 3: 12	4: 16; 3: 18	467,21
	4; 3: 14	4: 17; 3: 20	4: 23; 3: 26	4: 29; 3: 31	4: 35; 3: 38	4: 5; 3: 8	41,62
	4; 3: 5	4: 9; 3: 12	4: 14; 3: 17	4: 21; 3: 23	4: 26; 3: 29	4: 33; 3: 35	396,23
	4; 3: 32	4: 35; 3: 38	4: 6; 3: 9	4: 12; 3: 15	4: 19; 3: 21	4: 24; 3: 27	78,62
	4; 3: 24	4: 27; 3: 30	4: 33; 3: 35	4: 39; 3: 7	4: 9; 3: 12	4: 16; 3: 18	498,71
	4; 3: 15	4: 17; 3: 20	4: 24; 3: 26	4: 29; 3: 32	4: 35; 3: 38	4: 6; 3: 9	44,13

Сума: 5438,35213045844

Налаштування

Наступне покоління Наступні 2 покол. Кросинговер Мутація

Популяція

Початкова популяція	Янська-Гоголя	Смілянська-Благовісна	Смілянська-Ільїна	Леніна-Гоголя	Благовісна-Леніна	Ільїна-Леніна	Y
Популяція 1							
; 3: 14	4: 17; 3: 20	4: 23; 3: 26	4: 30; 3: 30	4: 36; 3: 37	4: 6; 3: 7	151,03	
; 3: 33	4: 34; 3: 39	4: 6; 3: 9	4: 12; 3: 15	4: 19; 3: 22	4: 23; 3: 28	200,91	
; 3: 20	4: 24; 3: 26	4: 13; 3: 14	4: 18; 3: 21	4: 24; 3: 27	4: 30; 3: 33	74	
; 3: 34	4: 36; 3: 41	4: 7; 3: 10	4: 13; 3: 17	4: 19; 3: 22	4: 26; 3: 29	158,53	
; 3: 8	4: 10; 3: 14	4: 15; 3: 21	4: 21; 3: 26	4: 29; 3: 32	4: 34; 3: 37	284,41	
; 3: 14	4: 17; 3: 20	4: 23; 3: 25	4: 30; 3: 30	4: 36; 3: 39	4: 5; 3: 8	56,41	
; 3: 33	4: 35; 3: 38	4: 6; 3: 9	4: 12; 3: 16	4: 20; 3: 20	4: 23; 3: 28	178,43	
; 3: 20	4: 24; 3: 26	4: 12; 3: 15	4: 18; 3: 21	4: 24; 3: 27	4: 31; 3: 32	42,1	
; 3: 34	4: 36; 3: 41	4: 7; 3: 11	4: 12; 3: 17	4: 19; 3: 22	4: 26; 3: 28	139,09	
; 3: 8	4: 11; 3: 13	4: 17; 3: 19	4: 23; 3: 24	4: 30; 3: 31	4: 34; 3: 37	309,46	
; 3: 14	4: 17; 3: 20	4: 23; 3: 26	4: 29; 3: 32	4: 34; 3: 39	4: 4; 3: 9	139,4	
; 3: 14	4: 18; 3: 19	4: 24; 3: 26	4: 29; 3: 32	4: 35; 3: 38	4: 7; 3: 8	60,98	
; 3: 34	4: 35; 3: 39	4: 8; 3: 9	4: 11; 3: 15	4: 19; 3: 21	4: 24; 3: 28	88,46	
; 3: 32	4: 35; 3: 38	4: 5; 3: 10	4: 13; 3: 14	4: 20; 3: 21	4: 24; 3: 27	119,35	
; 3: 34	4: 36; 3: 40	4: 8; 3: 10	4: 13; 3: 18	4: 19; 3: 23	4: 24; 3: 30	50,35	
; 3: 9	4: 10; 3: 15	4: 17; 3: 20	4: 23; 3: 26	4: 28; 3: 33	4: 34; 3: 37	143,96	
; 3: 33	4: 39; 3: 39	4: 9; 3: 10	4: 15; 3: 17	4: 20; 3: 22	4: 26; 3: 29	144,19	
; 3: 34	4: 36; 3: 40	4: 7; 3: 11	4: 12; 3: 18	4: 18; 3: 22	4: 26; 3: 28	89,89	
; 3: 8	4: 11; 3: 13	4: 17; 3: 20	4: 23; 3: 24	4: 30; 3: 31	4: 35; 3: 38	279,45	
; 3: 9	4: 10; 3: 14	4: 16; 3: 20	4: 23; 3: 25	4: 29; 3: 33	4: 34; 3: 38	268,87	

Сума: 2979,26098684142

Налаштування

Наступне покоління

Наступні 2 покол.

 Кросинговер Мутація

genetic imitation



Популяція

Початкова популяція	Іванська-Гоголя	Смілянська-Благовісна	Смілянська-Ільїна	Леніна-Гоголя	Благовісна-Леніна	Ільїна-Леніна	Y
Популяція 1	2; 3: 22	4: 27; 3: 28	4: 16; 3: 18	4: 20; 3: 24	4: 25; 3: 30	4: 15; 3: 17	6,31
Популяція 2	1; 3: 18	4: 19; 3: 24	4: 16; 3: 21	4: 22; 3: 27	4: 29; 3: 32	4: 19; 3: 19	7,41
Популяція 3	2; 3: 34	4: 36; 3: 40	4: 7; 3: 10	4: 14; 3: 16	4: 20; 3: 21	4: 25; 3: 29	93,3
Популяція 4	3; 3: 24	4: 26; 3: 28	4: 15; 3: 17	4: 19; 3: 23	4: 25; 3: 30	4: 14; 3: 18	5,76
Популяція 5	3; 3: 25	4: 31; 3: 31	4: 11; 3: 12	4: 14; 3: 19	4: 21; 3: 24	4: 28; 3: 29	125,85
	3; 3: 23	4: 25; 3: 30	4: 14; 3: 19	4: 19; 3: 25	4: 25; 3: 30	4: 15; 3: 18	7,44
	1; 3: 18	4: 20; 3: 23	4: 17; 3: 19	4: 24; 3: 25	4: 30; 3: 31	4: 19; 3: 19	6,4
	3; 3: 35	4: 36; 3: 40	4: 7; 3: 11	4: 13; 3: 17	4: 18; 3: 23	4: 23; 3: 30	69,81
	3; 3: 22	4: 27; 3: 27	4: 15; 3: 17	4: 19; 3: 23	4: 26; 3: 29	4: 16; 3: 16	6,16
	1; 3: 26	4: 29; 3: 33	4: 9; 3: 13	4: 13; 3: 19	4: 21; 3: 24	4: 27; 3: 30	64,89
	3; 3: 23	4: 27; 3: 29	4: 15; 3: 17	4: 21; 3: 23	4: 27; 3: 29	4: 16; 3: 17	7,48
	3; 3: 23	4: 26; 3: 30	4: 15; 3: 18	4: 20; 3: 26	4: 24; 3: 32	4: 13; 3: 20	6,43
	3; 3: 16	4: 16; 3: 19	4: 24; 3: 26	4: 29; 3: 32	4: 35; 3: 37	4: 9; 3: 6	231,69
	3; 3: 20	4: 22; 3: 28	4: 10; 3: 15	4: 18; 3: 21	4: 24; 3: 27	4: 31; 3: 32	89,82
	3; 3: 34	4: 36; 3: 40	4: 6; 3: 12	4: 15; 3: 16	4: 21; 3: 21	4: 24; 3: 30	105,32
	3; 3: 33	4: 36; 3: 40	4: 7; 3: 11	4: 13; 3: 18	4: 18; 3: 24	4: 24; 3: 30	74,45
	3; 3: 35	4: 34; 3: 40	4: 7; 3: 10	4: 10; 3: 16	4: 18; 3: 20	4: 25; 3: 27	150,82
	3; 3: 12	4: 19; 3: 17	4: 25; 3: 26	4: 28; 3: 31	4: 35; 3: 38	4: 7; 3: 7	135,09
	3; 3: 33	4: 35; 3: 39	4: 5; 3: 11	4: 11; 3: 18	4: 17; 3: 21	4: 24; 3: 28	109,44
	3; 3: 19	4: 25; 3: 25	4: 14; 3: 15	4: 17; 3: 20	4: 23; 3: 28	4: 29; 3: 34	83,41

Сума: 1387,27481453834

Налаштування

Наступне покоління

Наступні 2 покол.

 Кросинговер Мутація



Популяція

Початкова популяція	Іянська-Гоголя	Смілянська-Благовісна	Смілянська-Ільїна	Леніна-Гоголя	Благовісна-Леніна	Ільїна-Леніна	Y
Популяція 1); 3: 23	4: 24; 3: 28	4: 13; 3: 18	4: 20; 3: 22	4: 25; 3: 28	4: 14; 3: 17	6,1
Популяція 2); 3: 21	4: 27; 3: 27	4: 15; 3: 16	4: 20; 3: 23	4: 26; 3: 29	4: 13; 3: 17	6,75
Популяція 3); 3: 22	4: 24; 3: 28	4: 14; 3: 18	4: 19; 3: 25	4: 23; 3: 32	4: 15; 3: 18	6,46
Популяція 4); 3: 19	4: 23; 3: 25	4: 15; 3: 18	4: 21; 3: 25	4: 26; 3: 31	4: 15; 3: 19	5,78
Популяція 5); 3: 22	4: 26; 3: 29	4: 14; 3: 19	4: 19; 3: 23	4: 27; 3: 28	4: 16; 3: 16	7,18
Популяція 6); 3: 23	4: 25; 3: 27	4: 14; 3: 16	4: 20; 3: 22	4: 26; 3: 27	4: 14; 3: 17	5,88
); 3: 22	4: 27; 3: 27	4: 15; 3: 17	4: 19; 3: 24	4: 24; 3: 31	4: 13; 3: 16	6,3
); 3: 20	4: 25; 3: 27	4: 14; 3: 18	4: 19; 3: 25	4: 24; 3: 31	4: 15; 3: 20	6,17
); 3: 20	4: 21; 3: 27	4: 13; 3: 19	4: 20; 3: 25	4: 26; 3: 31	4: 16; 3: 18	7,29
); 3: 22	4: 27; 3: 28	4: 15; 3: 17	4: 21; 3: 22	4: 28; 3: 27	4: 16; 3: 16	8,51
); 3: 23	4: 26; 3: 27	4: 15; 3: 17	4: 19; 3: 24	4: 25; 3: 29	4: 13; 3: 19	6,98
); 3: 23	4: 26; 3: 27	4: 15; 3: 18	4: 19; 3: 23	4: 25; 3: 29	4: 16; 3: 16	6,13
); 3: 23	4: 26; 3: 28	4: 14; 3: 18	4: 17; 3: 24	4: 25; 3: 30	4: 15; 3: 17	5,55
); 3: 21	4: 28; 3: 27	4: 17; 3: 17	4: 21; 3: 23	4: 26; 3: 29	4: 14; 3: 18	6,45
); 3: 24	4: 26; 3: 29	4: 14; 3: 18	4: 18; 3: 25	4: 23; 3: 31	4: 13; 3: 19	6,77
); 3: 20	4: 23; 3: 26	4: 15; 3: 19	4: 21; 3: 26	4: 25; 3: 33	4: 16; 3: 19	6,69
); 3: 21	4: 22; 3: 26	4: 14; 3: 20	4: 20; 3: 25	4: 26; 3: 31	4: 17; 3: 18	7,03
); 3: 20	4: 22; 3: 28	4: 15; 3: 17	4: 23; 3: 24	4: 28; 3: 31	4: 16; 3: 19	6,19
); 3: 22	4: 27; 3: 28	4: 15; 3: 19	4: 20; 3: 25	4: 25; 3: 30	4: 14; 3: 18	7,61
); 3: 22	4: 26; 3: 28	4: 15; 3: 17	4: 20; 3: 22	4: 28; 3: 28	4: 18; 3: 15	13,07

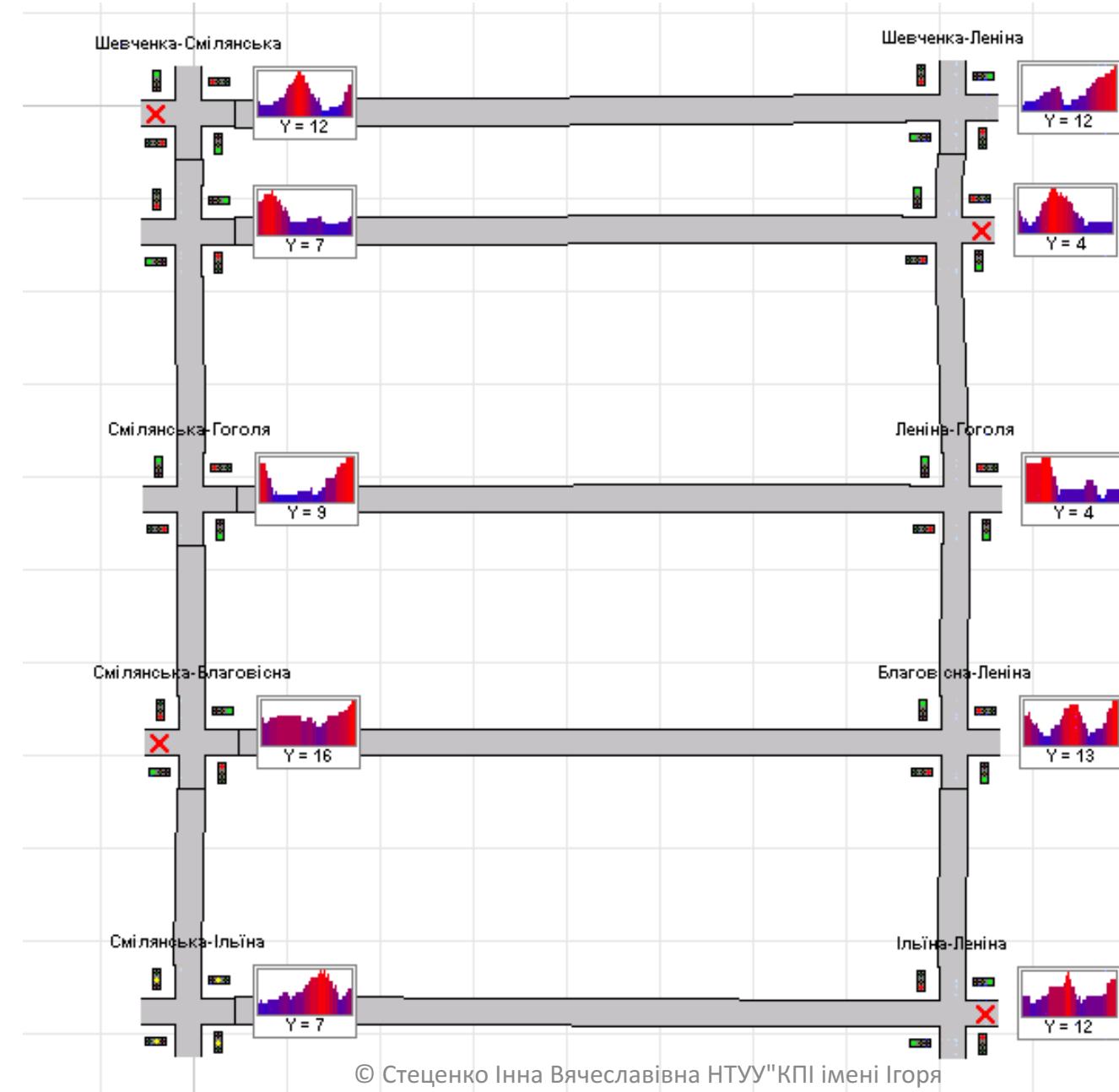
Сума: 138,896065808533

Налаштування

Наступне покоління

Наступні 2 покол.

 Кросинговер Мутація



Висновки

- ✓ Факторний аналіз слід використовувати у випадку невеликої кількості факторів та дослідження лінійної залежності від факторів. Дисперсійний аналіз використовують для констатації факту залежності. Регресійний аналіз - для кількісної оцінки впливу.
- ✓ Методи оптимізації слід використовувати для систем, в яких наявність оптимального значення обґрунтована.
- ✓ Для систем з великою кількістю факторів слід використовувати методи еволюційного пошуку.

Програмне забезпечення Arena імітаційного моделювання систем

- Arena Simulation Software – розробка компанії Systems Modeling Arena. На сьогодні цим софтом займається компанія Rockwell Automation.
<https://www.rockwellautomation.com/en-us/products/software/arena-simulation/buying-options/download.html>
- Основа технологій Arena - мова моделювання SIMAN і система Cinema Animation. SIMAN (перша реалізація 1982р.)
- Переваги Arena: 1) модель будується з конструктивних блоків, набір яких достатньо широкий, 2) ієархічна модель будується дуже просто і наочно, 3)анімаційні засоби представлені широко (від примітивних під час запуску моделі до графічних з анімацією персонажів і елементів моделі), 4) широкий спектр звітів з моделювання, 5) набір інструментів для статистичного збору інформації та її графічного представлення.
- Аналогічний підхід використовує програмне забезпечення Simio (з більш сучасною анімацією та більшою швидкодією)

Відеоресурси Arena

- Тьюторіал для початківців
<https://www.youtube.com/watch?v=dbW8WFen1s>
- Відео з демонстрацією анімаційних можливостей Arena
<https://www.youtube.com/watch?v=6EFPB0FUgCM>
- Відео з демонстрацією анімаційних можливостей Simio <https://www.simio.com/applications/aerospace-and-defense-simulation-and-scheduling-software/index.php>

Інтерфейс системи Arena

Вікно додатку розділене на три області:

- вікно робочого поля;
- вікно властивостей модулів;
- вікно проекту.

Вікно проекту включає декілька панелей:

- Basic Process (панель основних процесів) - містить модулі, які використовуються для моделювання;
- Reports (панель звітів) - панель повідомлень: містить повідомлення, які відображають результати імітаційного моделювання;
- Navigate (панель навігації) - панель управління дозволяє відображати всі види моделі, включаючи управління через ієрархічні підмоделі.

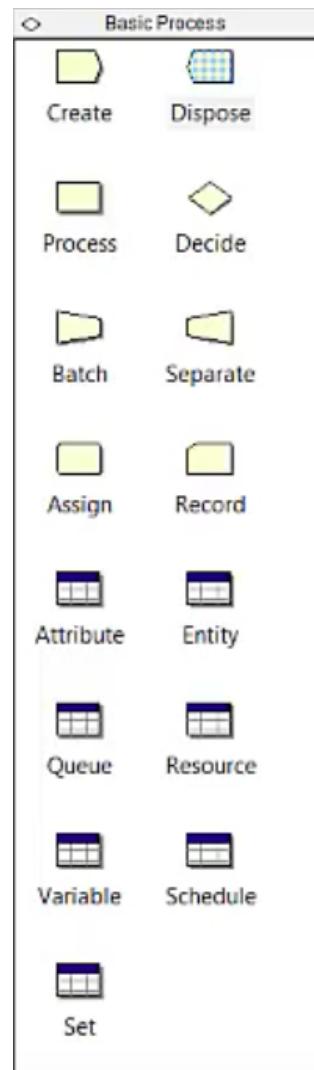
Вікно робочого поля представляє зображення моделі, включаючи блок-схему процесу, анімацію і інші елементи.

Вікно властивостей модуля використовуються для настройки параметрів моделі таких як: час, витрати і інші параметри.

Конструктивні елементи Arena

- Розділені на групи Basic Process, Advanced Process та інші
- Кожна група містить набір модулів для побудови моделі. Модулі перетягаються у вікно робочого поля та з'єднуються у відповідності до логіки процесу, який моделюється
- Кожен блок містить набір параметрів, які має налаштовувати користувач у вікні властивостей модулю

Basic Process



Модуль *Create*

Модуль *Create* є відпрівною точкою для вимоги в імітаційній моделі. Вимоги - це об'єкти, які обробляються в системі. Створення вимоги модулем відбувається за розкладом, або ж ґрунтуючись на значенні часу між прибуттям вимог в модель. Залишаючи модуль, вимога починає оброблятися в системі. Тип створюваної вимоги визначається в цьому модулі. Параметри модуля та їх призначення:

- *Name* - унікальне ім'я модуля, яке буде відображене в блок схемі.
- *Entity Type* - назва типу вимоги, який створюватиметься модулем.
- *Type* - спосіб формування потоку прибуття. Type може мати значення Random (використовується експоненціальний розподіл з середнім значенням, визначенім користувачем), Schedule (визначається модулем Schedule), Constant (використовуватиметься, визначене користувачем, постійне значення) або Expression (потік прибуття формуватиметься по певному виразу).
- *Value* - визначає середнє значення експоненціального розподілу (Random) або постійне значення часу між прибуттям вимоги (якщо Type = Constant).
- *Schedule Name* - ім'я розкладу, який визначає характер прибуття вимоги в систему.
- *Expression* - задає тип розподілу або вираз, що визначає час між прибуттями вимоги в модель.
- *Units* - Одиниці вимірювання часу між прибуттям (день, година, хвилини, секунда).
- *Entities per arrival* - Кількість вимог, яка входить в систему за одне прибуття.
- *Max arrivals* - Максимальне число вимог, яке може створити цей модуль.
- *First Creation* - Час, через який прибуде перша вимога в модель від початку симуляції.

Модуль *Process*

Модуль *Process* є основним модулем процесу обробки в імітаційній моделі. окрім стандартного модуля *Process*, можна використовувати підмодель, додаючи її особливу, визначену користувачем, ієрархічну логічну схему. У модулі можна також задавати додаткові вартісні і часові характеристики процесу обробки об'єкта. Параметри модуля та їх значення:

- *Name* - унікальне ім'я модуля, яке відображається в блок схемі.
- *Type* - визначає логічну схему модуля. *Standard* означає, що логічна схема знаходитьться усередині модуля і залежить від параметра *Action*. *Submodel* показує, що логічна схема знаходитиметься нижче в ієрархічній моделі. Підмодель може містити будь-яку кількість логічних модулів.
- *Action* - тип обробки що відбувається усередині модуля. *Delay* просто показує про те, що процес займає деякий час і не відображає використання ресурсів. *Seize Delay* показує на те, що в цьому модулі були розміщені ресурси і відбуватиметься затримка, ресурси захоплюватимуться (тобто будуть зайняті обробкою вимоги), і їх звільнення буде відбувається пізніше. *Seize Delay Release* показує на те, що ресурс(-и) були захоплені, а потім через час звільнилися. *Delay Release* означає, що ресурси до цього були захоплені вимогою, а в такому модулі вимога затримається і звільнить ресурс. Всі ці параметри доступні тільки

Модуль *Process*

- *Priority* - значення пріоритету модулів що використовують один і той же ресурс де завгодно в моделі. Ця властивість не доступна, якщо Action = Delay або Delay Release, або коли Type = Submodel.
- *Resources* - визначає ресурси або групи ресурсів, які оброблятимуть вимогу в цьому модулі (див. Модуль Process - Ресурси).
- *Delay Type* - тип розподілу або процедура, що визначає параметри затримки.
- *Units* - Одиниці вимірювання часу затримки (день, година, хвилинилини, секунда).
- *Allocation* - Визначає вартісні характеристики обробки. Value Added - означає враховувати вартісні характеристики, а Non-Value Added не враховувати.
- *Minimum* - мінімальне значення для рівномірного і трикутного розподілу.
- *Maximum* - максимальне значення для рівномірного і трикутного розподілу.
- *Value* - середнє значення для нормального і трикутного розподілу або значення для постійної часової затримки.
- *Std Dev* - стандартне відхилення для нормального розподілу.

Модуль *Decide*

Модуль Decide дозволяє враховувати прийняття рішень в моделі. Модуль включає опції прийняття рішень заснованих на умові By Condition (наприклад, якщо тип вимоги Car) або заснованих на вірогідності By Chance (наприклад, 75% - true, а 25% - false). Умови можуть бути засновані на значенні атрибуту Attribute, значенні змінної Variable, типі об'єкта Entity Type або засновані на виразі Expression. Якщо поставлена умова не виконується, то об'єкт залишатиме модуль через гілку False.

Параметри модуля та їх значення:

- Name - унікальне ім'я модуля, яке відображається в блок схемі.
- Type - тип ухвалення рішення. By Chance - вибір напряму ґрунтуються на ймовірності. By Condition - перевірка на виконання умови
- Percent True - значення, що визначає відсоток вимог, який підуть по на-пряму True
- If - Тип умови, яка перевірятиметься на виконання.
- Named - ім'я змінної, атрибуту або типу вимоги, який перевірятимуться при вході вимоги в модуль.
- Is - математичний знак умови, наприклад більше, менше, рівно і т.д.
- Value - значення, з яким порівнюватиметься атрибут або змінна вимоги, що прийшла. Якщо тип умови - Expression, то у виразі повинен стояти знак умови, наприклад Color<> Red
- Даний модуль дозволяє виконувати перевірку не тільки однієї умови, але і декілька. Це досягається за допомогою властивості Type N-way by Chance/by Condition. Залежно від умови об'єкт йде по потрібній вітці.

Модуль *Batch*

Модуль **Batch** відповідає за механізм угрупування в імітаційній моделі.

Наприклад, зібрати необхідну кількість даних, перш ніж починати їх обробку, зібрати раніше розділені копії одної форми, з'єднати пацієнта і його лікарняну карту прийому до лікаря. Угрупування може бути постійним або тимчасовим. Тимчасово згруповани комплекти пізніше можуть бути роз'єднані за допомогою модуля Separate. Комплекти можуть складатися з будь-якої кількості входних об'єктів, визначеної користувачем, або ж об'єкт може об'єднуватися в комплект залежно від його атрибуту. Часові і вартісні характеристики об'єктів, які входять у групу, будуть рівні сумі характеристик вимог, які увійшли до групи.

- вимога прибуває в модуль, стає в чергу і залишається там до тих пір, поки в модулі не буде набрано задану кількість вимог. Коли набереться потрібне число об'єктів створюється об'єкт, який представляє комплект. Параметри модуля та їх значення:
- *Name* - унікальне ім'я модуля, яке відображається в блок схемі.
- *Type* - Спосіб угрупування вимоги, може бути Temporary (тимчасова), Permanent (постійна).
- *Batch Size* - Число вимог, які утворюють один комплект.
- *Rule* - Визначає, за якою ознакою групуватимуться. Якщо Rule = Any Entity, це означає що перші 3 (якщо Batch Size = 3) вимоги будуть згруповано. Якщо Rule = By Attribute, то об'єднуватиметься задана кількість вимог з певним атрибутом. Наприклад, якщо Attribute Name = Color, то всі вимоги, які мають одинаковий атрибут Color, буде згрупована.
- *Attribute Name* - Ім'я атрибуту, по значенню якого групуватимуться вимоги.

Модуль *Separate*

Модуль *Separate* може використовуватися як для створення копій вхідної вимоги, так і для розділення раніше згрупованих вимог. Наприклад, для роз'єднання раніше згрупованих комплектів документів, для паралельної обробки рахунків і документів по одному замовленню. Правило для розділення вартісних і часових характеристик копій вимоги і розділеної вимоги визначається користувачем. Коли тимчасово згруповані вимоги прибувають в модуль, вони розкладаються на складену вимогу. Вимога покидає модуль в тій же послідовності, в якій вони додавалися в комплект. Якщо модуль створює копії вимоги, то користувач може задати кількість дублікатів вимоги. У дубльованої вимоги значення атрибуту, а також анімаційна картинка такі ж, як і оригіналу. Оригінальна вимога також покидає модуль. Параметри модуля та їх значення:

- *Name* - унікальне ім'я модуля, яке відображається в блок схемі.
- *# of Duplic* - Кількість створюваних копій вхідної вимоги.
- *Type* - Спосіб розділення вхідної в модуль вимоги. *Duplicate Original* - просто робить дублікати вхідної вимоги. *Split Existing Batch* вимагає щоб вхідна вимога була заздалегідь тимчасово згруповано.
- *Cost to Duplicates* - Розділення вартісних і часових характеристик вхідної вимоги між тими, що виходять. Це значення визначається користувачем у відсотках, тобто скільки відсотків від вартісних і часових характеристик вхідної вимоги піде копіям (характеристики між копіями діляться порівну).
- *Allocation Rule* - Метод розділення вартості і часу, якщо вибраний *Type=Split Existing Batch*. *Retain Original Entity Values* - зберігає оригінальні значення вимоги. *Take All Representative Values* - вся вимога приймає однакове значення. *Take Specific Representative Values* - вимога приймає специфічне значення.

Модуль *Assign*

Модуль *Assign* призначений для завдання нового значення змінній, атрибуту вимоги типу вимоги, анімаційній картинці вимоги або іншої змінної в системі. Наприклад, для встановлення пріоритету для клієнтів, для привласнення номера наказу, що вийшов. У одному модулі можна зробити тільки одне призначення. Параметри модуля та їх значення:

- *Name* - унікальне ім'я модуля, яке відображається в блок схемі.
- *Type* - тип призначення, яке здійснюватиметься. *Other* може включати вбудовані в Арену змінні, такі як місткість ресурсу або кінцевий час симуляції.
- *Variable Name* - ім'я змінної, яка змінюватиметься в цьому модулі.
- *Attribute Name* - ім'я атрибуту, який змінюватиметься в цьому модулі.
- *Entity Type* - новий тип вимоги, який привласнюється вимозі в цьому модулі.
- *Entity Picture* - нова анімаційна картинка для вимоги, яка пройшла цей модуль.
- *Other* - ім'я змінної в системі, яка змінюється.
- *New Value* - привласнюване нове значення для атрибуту, змінної.
-

Модуль *Record*

Модуль *Record* призначений для збору статистики в імітаційній моделі. Наприклад, для підрахування, яка кількість замовень була виконана із запізненням, підрахування кількості роботи, що здійснюється за одну годину. Модуль може збирати різні типи статистики, включаючи час між виходами вимоги з модуля, статистику вимоги (час циклу, вартість), статистику за період часу (період часу від заданої точки до теперішнього моменту). Також доступний кількісний тип статистики. Параметри модуля та їх значення:

- *Name* - унікальне ім'я модуля, яке відображається в блок схемі.
- *Type* - визначає тип статистики, яка збиратиметься. *Count* - збільшуватиме або зменшуватиме статистику на задане значення. *Entity Statistics* збиратиме загальну статистику про вимогу, наприклад, час циклу, вартісні характеристики і т.д. *Time Interval* рахуватиме різницю між значенням атрибуту і поточним часом моделювання. *Time Between* відстежуватиме час між входженням вимоги в модуль. *Expression* просто фіксуватиме значення визначене виразом.
- *Attribute Name* - ім'я атрибуту, значення якого використовуватиметься для інтервальної статистики.
- *Value* - значення, яке додаватиметься до статистики, коли в модуль прибуватиме вимога.

Модуль *Dispose*

Модуль *Dispose* є вихідною точкою з імітаційної моделі. Наприклад, клієнти покидають відділ, закінчення бізнес-процесу. Статистика про вимогу може збиратися до того моменту поки вона не вийде з системи. Параметри модуля та їх значення:

- *Name* - унікальне ім'я модуля, яке відображається в блок схемі.
- *Record Entity Statistics* - визначає, чи вестиметься статистика про вихід вимоги з системи.

Виконання моделювання

Перед тим, як розпочати процес імітації потрібно зайти в меню Run, підменю Setup і ввести в установки Replication Parameters час моделювання (Replication Length) та одиниці, в яких він вимірюється (Time Units), а також одиниці часу, в яких будуть представлені результати моделювання (Basic Time Units). Бажано також ввести в установки Project Parameters ім'я проекту (Project Title) обрати статистичні дані, що збираються (Entities, Queues, Resources, Processes, а також Costing).

Запуск імітації здійснюється вибором команди Go меню Run. Для перевірки моделі зручно скористатись командою Step, що запускає покрокову імітацію модельованого процесу. Управління процесом імітації здійснюють команди Pause - часова зупинка процесу імітації, Fast Forward - швидка імітація, End - повернення до корегування моделі або нового запуску.

Виведення результатів моделювання

- В результаті успішної імітації моделі пакетом Arena створюються звіти про результати моделювання. Переглянути їх можна скориставшись панеллю звітів *Reports*. На панелі звітів представлено декілька видів звітів. Звіт «Короткий огляд категорій» і звіти по чотирьох категоріях, такі як Вимоги, Процеси, Черги і Ресурси.
- За допомогою панелі звітів можна переглянути звіт *Category Overview*, який відображає підсумкову інформацію про вимоги, процеси, черги і ресурси. Також показує інформацію про заданих користувачем змінних і інформацію, зібрану модулем *Record*.
- Звіт про вимоги *Entities* поділений на декілька частин. У частині звіту *Cycle Time* показаний середній, максимальний і мінімальний час існування вимоги. Час існування вимоги вважається з моменту її прибуття в систему і до того моменту, коли вимога потрапляє в модуль *Dispose*. Нижче представляється гістограма середнього часу циклу для кожного типу вимоги.

Приклад

Експериментальна роботизована гнучка виробнича система має два верстати із числовим пультом керування, три роботи, пункт прибуття і склад оброблених деталей. Деталі прибувають на пункт прибуття кожні 40 секунд згідно з експоненціальним законом розподілу, захоплюються одним з вільних роботів і переміщаються ним до першого верстата, після чого робот звільняється. Після завершення обробки на першому верстаті деталь захоплюється одним з роботів і переміщується на другий верстат, а після обробки на другому верстаті – одним з роботів переміщується на склад оброблених деталей. Кожний з верстатів може одночасно обробляти до трьох деталей.

Час переміщення робота між пунктом прибуття і першим верстатом, першим і другим верстатом, другим верстатом і пунктом зберігання оброблених деталей складає відповідно 6, 7, і 5 секунд незалежно від того, холостий це хід, чи ні. Роботу потрібний час 8 ± 1 секунд на захоплення або вивільнення деталей. Час обробки на першому верстаті розподілений за нормальним законом із середнім значення 60 секунд і має стандартне відхилення 10 секунд. Середній час обробки на другому верстаті дорівнює 100 секунд і має експоненціальний закон розподілу.

Визначить найкращий (з точки зору підвищення пропускної здатності гнучкої виробничої системи) спосіб закріплення роботів до операцій. Можливі варіанти закріплення:

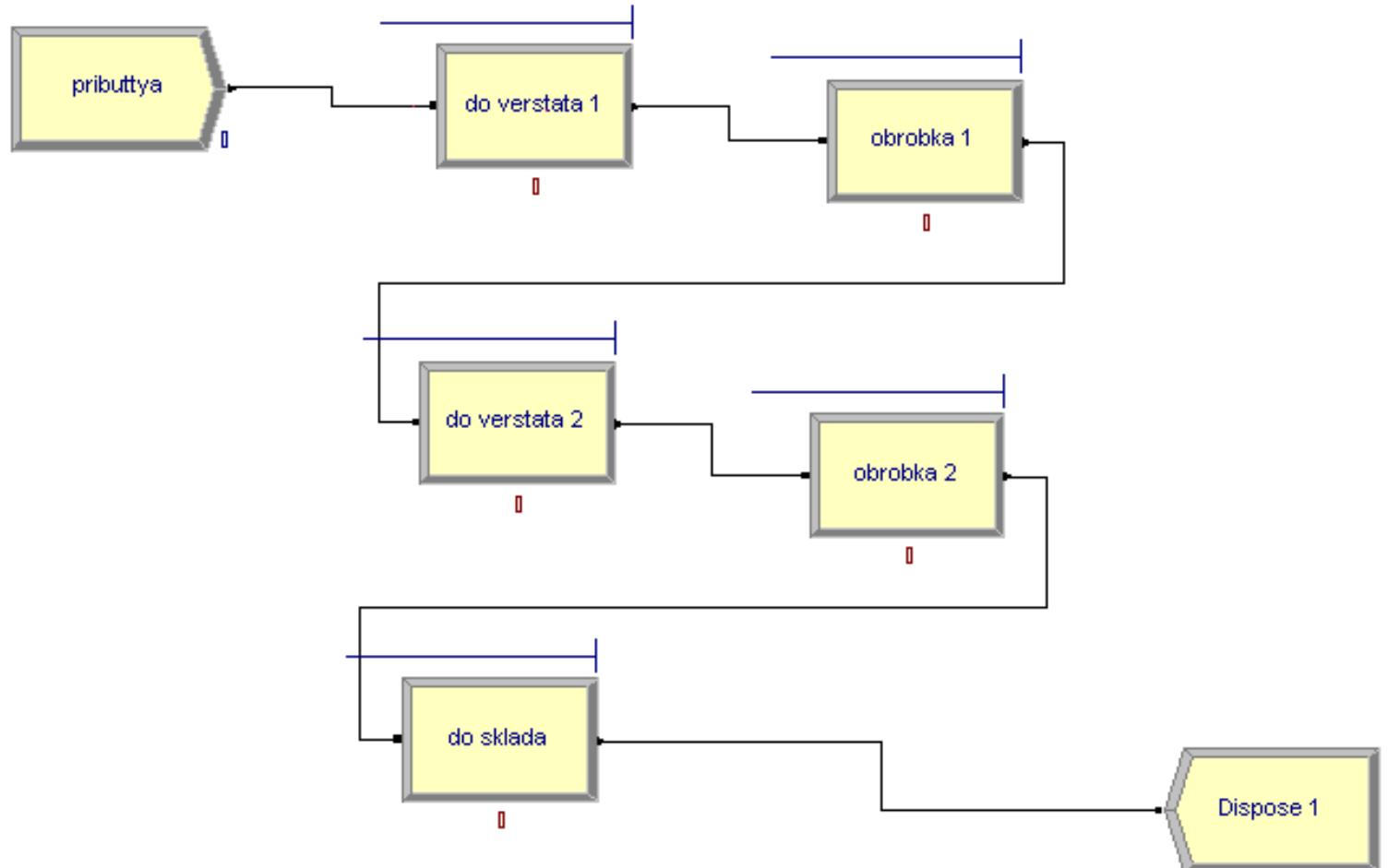
1. по одному роботу на кожний з трьох шляхів переміщення деталей (пункт прибуття – перший верстат, перший верстат – другий верстат, другий верстат, склад);
2. кожний робот може використовуватися на кожному шляху переміщення деталей (при цьому повинен займатися найближчий з роботів).

Знайдіть коефіцієнти використання роботів і верстатів, максимальну місткість місця зберігання деталей на ділянці прибуття.

Пояснення до моделі

- Обробку деталі першим верстатом моделює блок Process з параметрами Name=obrobka 1, Type=Standard, Action=Seize Delay Release, Resources=verstat 1, Delay Type=Normal, Units=Seconds , Value=60, Std Dev=10. Щоб верстат обслуговував одночасно три деталі, потрібно зайти в блок Resources і встановити параметр Capacity=3 для ресурсу з ім'ям verstat 1.
- Захоплення деталі роботом, переміщення її до другого верстата та вивільнення робота моделює блок Process з параметрами Name=do verstata 1, Type=Standard, Action=Seize Delay Release, Resources= Robot 2, Delay Type=Uniform, Units=Seconds , Minimum=22, Maximum=24.
- Змінемо закрілення роботів до операцій так, що кожний з вільних роботів може використовуватися на кожному шляху переміщення деталей. Для цього в моделі, яка складена, параметр Resources для процесів do verstata 1, do verstata 1, do sklada, потрібно задати як множина ресурсів, тобто тип Set, з ім'ям Set Robot. Потім за допомогою блоку Set задати елементи множини ресурсів Robot 1, Robot 2, Robot 3.

Модель роботизованої системи



Результати моделювання

1-ий варіант

- кількість оброблених деталей – 716 деталей;
- середнє завантаження першого верстата – 0,51;
- середня кількість деталей на обробці на першому верстаті – 1,52;
- середнє завантаження другого верстата – 0,81;
- середня кількість деталей на обробці на другому верстаті – 2,43;
- середнє завантаження первого робота – 0,56;
- середнє завантаження другого робота – 0,55;
- середнє завантаження третього робота – 0,55;
- середній час обробки однієї деталі – 314 с;
- середній час очікування однієї деталі – 90с, з них в очікуванні вільного робота – 6с, в очікуванні вільного першого верстата – 4с, в очікуванні вільного другого верстата – 80с;

Результати моделювання

2-ий варіант

- кількість оброблених деталей – 745 деталей;
- середнє завантаження першого верстата – 0,52;
- середня кількість деталей на обробці на першому верстаті – 1,56;
- середнє завантаження другого верстата – 0,80;
- середня кількість деталей на обробці на першому верстаті – 2,40;
- середнє завантаження первого робота – 0,57;
- середнє завантаження другого робота – 0,60;
- середнє завантаження третього робота – 0,54;
- середній час обробки однієї деталі – 356 с;
- середній час очікування однієї деталі – 137с, з них в очікуванні вільного робота – 35с, в очікуванні вільного першого верстата – 1с, в очікуванні вільного другого верстата – 101с.

Програмне забезпечення GPSS (General Purpose Simulation System)

Мова імітаційного моделювання GPSS

- GPSS (General Purpose Simulation System) – система моделювання складних об'єктів загального призначення, що розроблена Джефрі Гордоном приблизно у 1960 році. Спочатку розроблювалась і підтримувалась компанією IBM.
- GPSS розроблений низькорівневою мовою програмування Assembly (Assembler).
- Мова GPSS є мовою загального призначення для моделювання мереж масового обслуговування, що набула широкого розповсюдження завдяки книзі [Schriber, T. J. *Simulation using GPSS*. New York, 1974, Wiley], в якій розглядається велика кількість прикладів моделювання систем різного призначення мовою GPSS.
- Найбільш відомі версії GPSS V, GPSS World (Windows), JGPSS, aGPSS (Mac)
- Переваги GPSS: 1) процесно-орієнтоване представлення моделі, 2) гнучкість формування звітів, 2) реалізація для різних платформ.
- Недоліки: обмежений набір операторів, лінійна структура програми.

Основні правила мови GPSS

- Вимоги, що надходять на обслуговування у мережу масового обслуговування, в мові GPSS називаються *транзактами*.
- Процес обслуговування транзакту описується послідовністю *операторів* мови GPSS, які визначають усе, що відбувається з транзактом з моменту його надходження на обслуговування до моменту завершення обслуговування.
- Оператори мови GPSS характеризують процеси обробки вимог – виникнення транзактів, затримки їх в пристроях, очікування транзактів в черзі, вихід зі СМО.

Структура операторів GPSS

- У записі оператора виділяють три частини:
мітка, назва, поле змінних
AAA ADVANCE 20, 5
- Мітка – використовується в операторах if і loop для вказування на оператор, в який слід зайти. Отже, мітка не завжди потрібна і часто відсутня в описі оператора.
- Назва – завжди з переліку дозволених операторів
- У полі змінних можуть використовуватись як числа, так і змінні.

Оператори GENERATE TERMINATE

- **GENERATE 15,6,120,50,1** - генерація транзактів, інтервали часу між надходженнями транзактів розподілені рівномірно в діапазоні (15-6, 15+6), перший транзакт з'явиться із затримкою в 120 одиниць модельного часу, всього буде створено 50 транзактів, пріоритет транзактів рівний одиниці.
- **GENERATE 6, FN\$FFF,120,50,1** - те ж, але інтервал часу між появами транзактів є ціла частина добутку числа 6 та значення функції FFF.
- **TERMINATE 1** - видалення транзакта з системи, при цьому значення підсумкового лічильника зменшується на одиницю, а моделювання закінчується, якщо значення лічильника стане рівним або менше нуля.

Оператор FUNCTION

- опис функції FTIM, її аргументом являється випадкова величина (на це указує значення RN1), рівномірно розподілена в діапазоні (0,1), функція є неперервною числовою (вказівник C), заданою чотирма точками:

(0;0), (0.1; 0.8), (0.5, 1.6), (1.0; 1.9)

FTIM FUNCTION RN1,C4

0,0/0.1,0.8/0.5,1.6/1.0,1.9

FTIM FUNCTION *2,D4

0,12/1,9/2,8/3,6

- те ж, але аргументом є значення другого параметру транзакта, для якого розраховується значення дискретної величини (D) числової функції FTIM, заданої чотирма вузловими точками. Це поточне значення округляється до найближчого значення аргументу у вузловій точці.

Оператори QUEUE and DEPART

- **QUEUE QAA** - оператор реєстрації черги, довжина черги QAA збільшується на одиницю.
- **QUEUE QAA,2** – те ж , але довжина черги QAA збільшується на 2 одиниці.
- **DEPART QAA** - оператор реєстрації черги, довжина черги QAA зменшується на одиницю.
- **DEPART QAA,4** - те ж, але довжина черги QAA зменшується на 4 одиниці.

Оператори SEIZE, ADVANCE, RELEASE

- **SEIZE DEV** - зайняття пристрою DEV транзактом; якщо пристрій зайнятий, то транзакт затримується в черзі до цього пристрою.
- **ADVANCE A,B** - затримка транзакта на час, визначений вмістом полів А та В, зміст величин, які записані в цих під полях, такий же, як і в блоці GENERATE.
- **RELEASE DEV** - звільнення пристрою DEV транзактом.

Оператори STORAGE, ENTER, LEAVE

- **MEM STORAGE 40** - опис блоку пристрой WORK ємністю 40 одиниць.
- **ENTER MEM,12** - зайняття транзактом 12 одиниць ємності в накопичувачі MEM.
- **LEAVE MEM,*2** - звільнення k одиниць пам'яті в накопичувачі MEM, де k - значення 2-го параметра транзакта.

Оператори TRANSFER and TEST

- **TRANSFER ,MIT** - безумовна передача управління оператору з міткою (номером) MIT.
- **TRANSFER BOTH,LAB1,ONE** - перехід до оператора з міткою LAB1, якщо він неможливий, то до оператора з міткою ONE , якщо і він неможливий, то транзакт затримується до наступного моменту модельного часу, в який повторюються указані спроби переходу.
- **TRANSFER .4,AAA,END** - транзакт з ймовірністю 0,4 переходить до оператору з міткою END та з ймовірністю 0,6 до оператору з міткою AAA.
- **TRANSFER PICK,FIN7,FIN21** - перехід із рівною ймовірністю до операторів з номерами FIN7, FIN 7+1, FIN 7+2, . . . , FIN 21.
- **TRANSFER FN,AAA,2** - перехід до оператору, мітка якого рівна сумі значень функції AAA і числа 2.
- **TRANSFER P,4,41** - перехід до оператору, мітка якого рівна сумі значень параметра з номером 4 транзакта і числа 41.
- **TRANSFER SBR,PRC,7** - перехід до оператора PRC із записом в параметр з номером 7 транзакта мітки даного оператора.
- **TEST E V7,K256,END** - перехід за умовою (умовна передача управління): в позиціях 13-18 записується знак відношення, в перших двох підполях поля змінних записуються величини, що порівнюються. Якщо умова виконується, то перехід не виконується, інакше - перехід здійснюється до оператору з міткою LAB. Символи відношень: G - більше, L - менше, E - дорівнює, NE – не дорівнює, LE - менше або дорівнює, GE - більше або дорівнює. В даному прикладі перехід не виконується, якщо V7 = 256, інакше перехід виконується до оператору з номером END.

Оператор LOOP

- **LOOP 6,MIT** - організація циклу – кожний раз при вході в оператор LOOP перевіряється значення шостого параметру: якщо значення не дорівнює нулю, то транзакт переходить до блока з міткою MIT, а значення параметра зменшується на одиницю. Якщо значення шостого параметру при вході в блок LOOP дорівнює нулю, то транзакт слідує в наступний за блоком LOOP перехід

Оператори VARIABLE, ASSIGN, SAVEVALUE

- **5 VARIABLE X2-,25** - розрахунковий оператор, в даному випадку з величини за номером 2, що зберігається, віднімається число 25 і результат присвоюється змінній за номером 5.
- **ASSIGN 2,APP** - змінювання параметрів транзактів, в даному випадку другий параметр транзакта отримає значення APP.
- **SAVEVALUE 5+,*3** - величина за номером 5, що зберігається, збільшується на значення третього параметра транзакта.

Приклад: обслуговування клієнтів службою таксі

Служба замовлення таксі має 5 каналів для одночасного прийняття замовень по телефону. Час між спробами виклику таксі розподілений за законом Ерланга другого порядку із середнім 180 секунд. Абонент затрачає 30 секунд для набирання номера і, якщо застає всі канали служби замовлення зайнятими або після з'єднання з'ясовує, що черга на обслуговування перевищує 10 замовень (в такому випадку замовлення не приймаються), то через 60 секунд він повторює набирання номера. Після п'яти спроб абонент припиняє набирання. Служба замовлення таксі має у своєму розпорядженні 30 машин таксі для обслуговування замовень. Час, витрачений на проїзд до клієнта, залежить від відстані до нього. Імовірності можливих відстаней розподіляються таким чином: 2 км – з імовірністю 0,1, 8 км - з імовірністю 0,2, 9 км - з імовірністю 0,25, 11 км - з імовірністю 0,17, 12 км - з імовірністю 0,23, 20 км - з імовірністю 0,05. Вартість проїзду до клієнта клієнтом не сплачується. Швидкість руху машин рівномірно розподілена в інтервалі 45 ± 5 км/год. Час обслуговування клієнта рівномірно розподілений в інтервалі 50 ± 20 хвилин. Вартість попереднього замовлення складає 2 гривні, вартість проїзду 1 км складає 2 гривні.

Метою моделювання є визначення такої кількості операторів-телефоністів та водіїв таксі, при якій максимізується прибуток служби замовлення.

Приклад. Код мовою GPSS

GPSS World - [TAXI1]

File Edit Search View Command Window Help

TAXI STORAGE 30 ;кількість машин таксі
TEL STORAGE 5 ;кількість телефоністів
DIS FUNCTION RN1,C24 ;функція експоненціального закону розподілу
0.,0./.100,.104/.200,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.750,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
DOBSL FUNCTION RN2,C2 ;функція час обслуговування клієнта в секундах
0.,1800/.9999,4200
DSPEED FUNCTION RN3,C2
0.,11.111/.9999,13.888 ;функція швидкості руху машини в метрах на секунду
VIDST FUNCTION RN4,D6 ;функція, що задає відстань до клієнта в метрах
0.1,5000/0.3,8000/0.55,9000/0.72,11000/0.95,12000/1.00,20000
CHAS VARIABLE FN\$VIDST/FN\$DSPEED ;час руху машини до клієнта
COST VARIABLE FN\$DOBSL#FN\$DSPEED#0.2 ;вартість обслуговування в копійках за метр
GENERATE 90,FN\$DIS ;з'явився клієнт, що бажає таксі
ADVANCE 90,FN\$DIS ;затримка ерлангового розподілу
ASSIGN 1,5 ;не більше п'яти спроб набирання номера
DOZVON ADVANCE 30 ;набирання номера
TRANSFER BOTH,VIDP,VIDM ;якщо телефоніст вільний то відповідь інакше відмова
VIDP ENTER TEL ;зайняти телефоніста
ADVANCE 30 ;тривалість розмови в секундах
LEAVE TEL ;звільнити телефоніста
TEST L Q\$Klient,10,VIDM ;якщо черга на машини більше 10, то відмова
SAVEVALUE DOHOD+,200 ;підрахувати доход за попереднє замовлення
TRANSFER ,OBSL ;перейти до обслуговування
VIDM ADVANCE 60 ;затримка після невдалого дзвінка
LOOP 1,DOZVON ;цикл, що здійснює дозвон
SAVEVALUE NEOBSL+,1 ;підрахувати необслугованих клієнтів
TERMINATE ;вихід із системи клієнта, що не дозвонився
OBSL QUEUE Klient ;очікувати вільне таксі
ENTER TAXI ;таксі виконує замовлення
DEPART Klient ;зменшити кількість очікуючих клієнтів на одиницю
ADVANCE V\$CHAS ;таксі прямує до клієнта
ADVANCE FN\$DOBSL ;таксі обслуговує клієнта
LEAVE TAXI ;звільнити таксі
SAVEVALUE DOHOD+,V\$COST ;підрахувати доход за обслуговування
SAVEVALUE KLOBS+,1 ;підрахувати кількість обслугованих клієнтів
TERMINATE ;вихід із системи клієнта, що обслугувався
GENERATE 864000 ;час моделювання 240 годин
TERMINATE 1

Фрагмент звіту

LABEL	LOC	BLOCK TYPE	ENTRY COUNT	CURRENT COUNT	RETRY
	1	GENERATE	9627	0	0
	2	ADVANCE	9627	4	0
	3	ASSIGN	9623	0	0
DOZVON	4	ADVANCE	28823	0	0
	5	TRANSFER	28823	0	0
VIDP	6	ENTER	28487	0	0
	7	ADVANCE	28487	1	0
	8	LEAVE	28486	0	0
	9	TEST	28486	0	0
	10	SAVEVALUE	6833	0	0
	11	TRANSFER	6833	0	0
VIDM	12	ADVANCE	21989	1	0
	13	LOOP	21988	0	0
	14	SAVEVALUE	2788	0	0
	15	TERMINATE	2788	0	0
OBSL	16	QUEUE	6833	10	0
	17	ENTER	6823	0	0
	18	DEPART	6823	0	0
	19	ADVANCE	6823	2	0
	20	ADVANCE	6821	28	0
	21	LEAVE	6793	0	0
	22	SAVEVALUE	6793	0	0
	23	SAVEVALUE	6793	0	0
	24	TERMINATE	6793	0	0
	25	GENERATE	1	0	0
	26	TERMINATE	1	0	0

```

QUEUE      MAX. CONT. ENTRY ENTRY(O) AVE. CONT. AVE. TIME   AVE. (-O) RETRY
Klient       10     10    6833      30     9.306    1176.647   1181.835     0

```

STORAGE	CAP.	REM.	MIN.	MAX.	ENTRIES	AVL.	AVE.C.	UTIL.	RETRY	DELAY
TAXI	30	0	0	30	6823	1	29.949	0.998	0	10
TEL	5	4	0	5	28487	1	0.989	0.198	0	0

SAVEVALUE	RETRY	VALUE
DOHOD	0	52311875.943
KLOBS	0	6793.000
NEOBSL	0	2788.000

- Для самостійного опрацювання

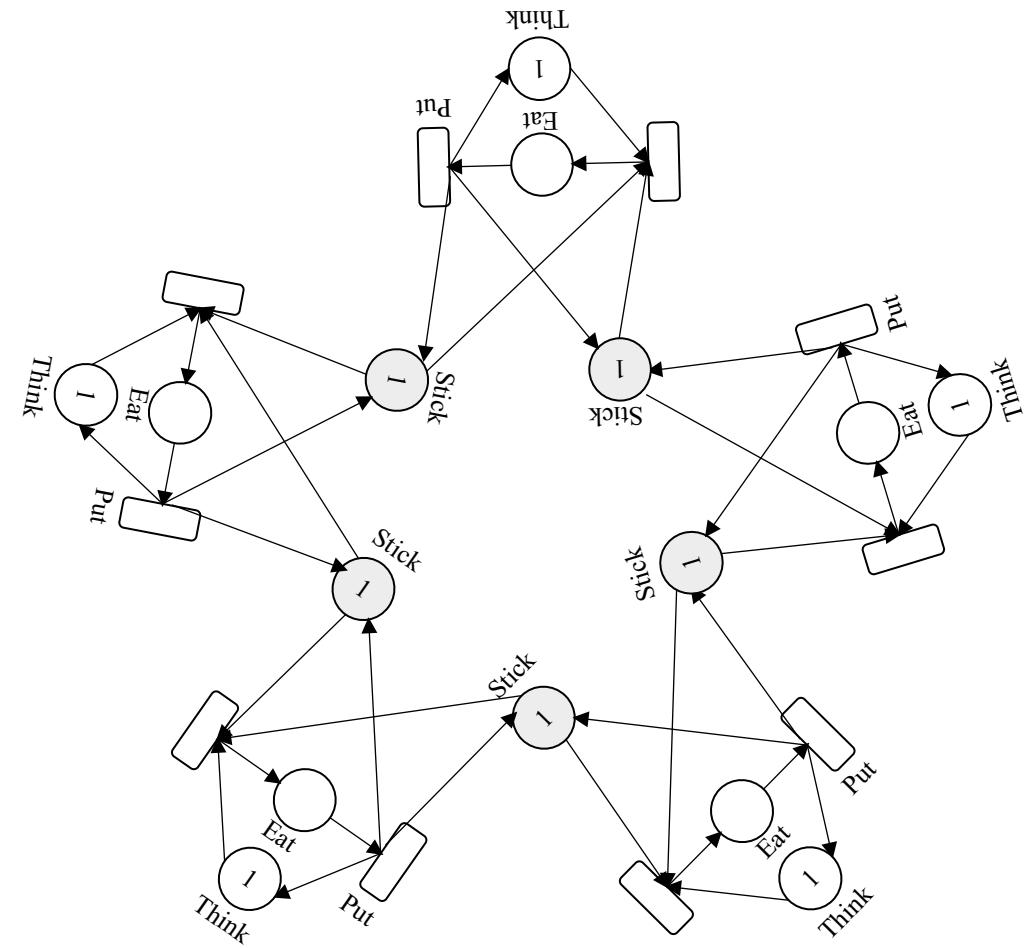
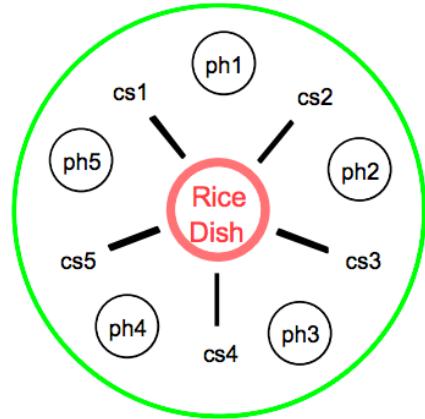
Програмне забезпечення CPNTools (General Purpose Simulation System)

Colored Petri Net

Розфарбовані мережі Петрі

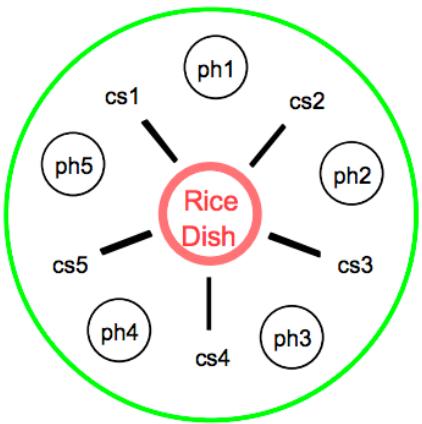
- У розфарбованій мережі Петрі маркерам можна присвоїти тип. Правило запуску переходу відповідно може бути розширене таким чином, що для запуску переходу необхідною є наявність маркерів конкретного типу. При цьому на дузі формулюється вимога щодо наявності маркерів одного чи більше типів. Може бути так, що входять в переход маркери одного типу, а виходять – іншого.
- Запам'ятовування типу маркера надає можливість зменшити кількість переходів, необхідних для моделювання складної системи. Власне, це і є основною метою такого розширення мереж Петрі.

Задача про філософів, представлена класичною мережею Петрі

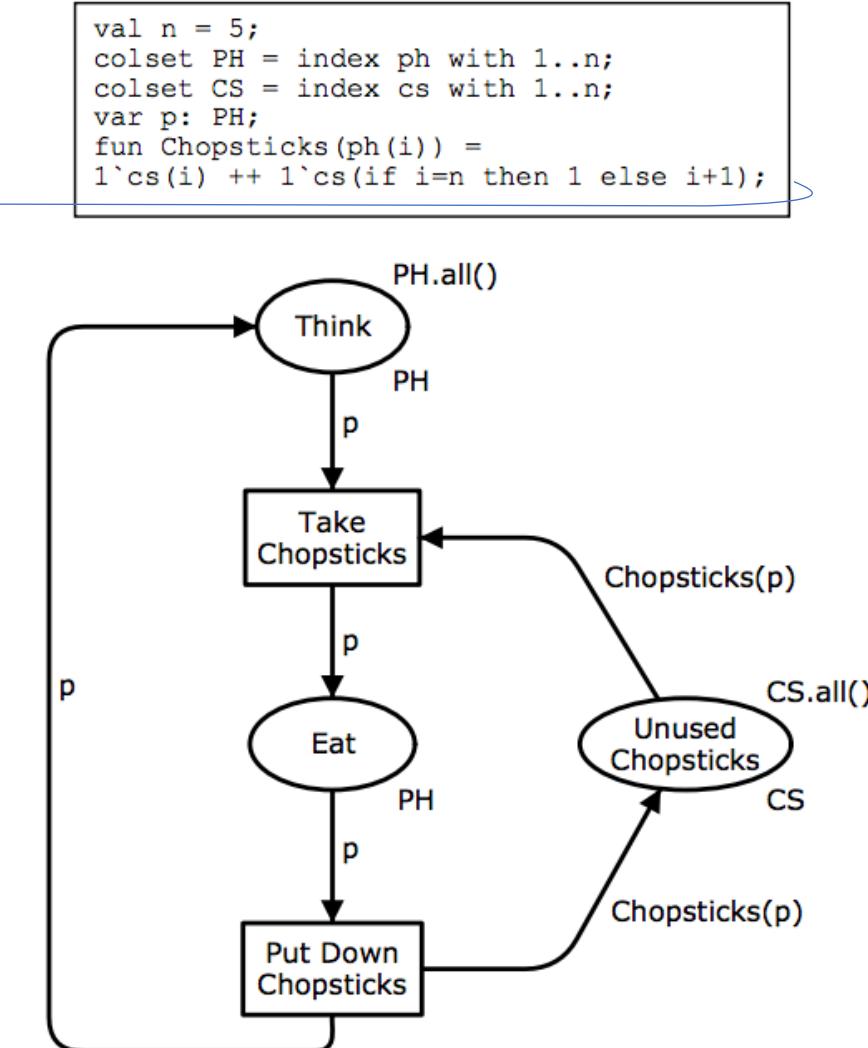


Приклад: задача про філософів

<http://cpn-tools.org/wp-content/uploads/2018/01/manual.pdf>



```
val n = 5;
colset PH = index ph with 1..n;
colset CS = index cs with 1..n;
var p: PH;
fun Chopsticks(ph(i)) =
  1`cs(i) ++ 1`cs(if i=n then 1 else i+1);
```



Приклад: передача повідомлень

Пояснення до розробки моделі:

Kurt Jensen 'An Introduction to the Practical Use of Coloured Petri Nets'

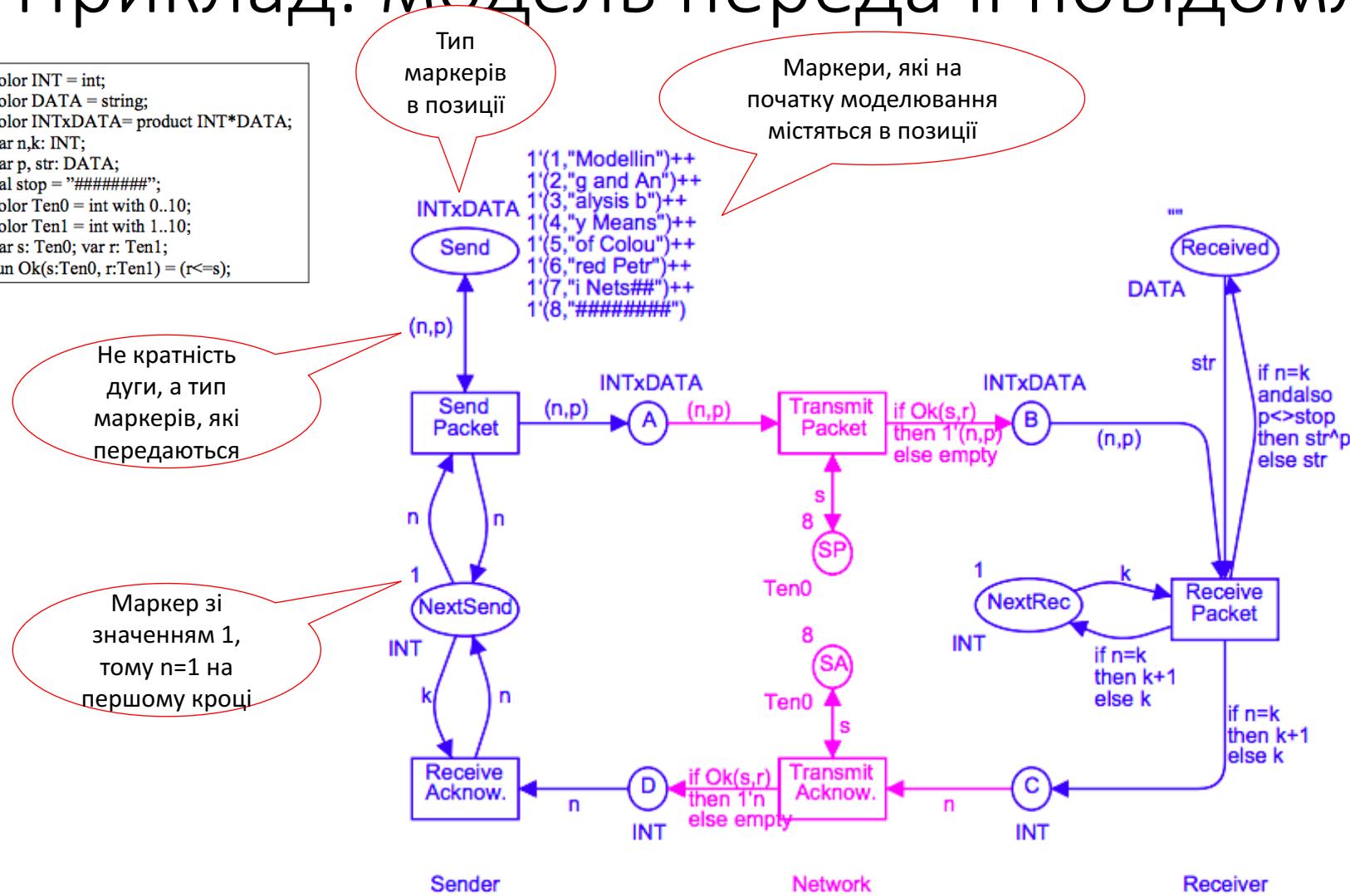
Пояснення до реалізації в CPNTools

<http://cpn-tools.org/wp-content/uploads/2018/01/simpleprotocol.pdf>

Приклад: модель передачі повідомень

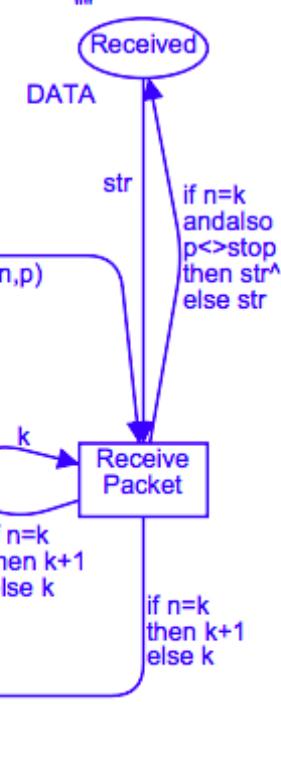
```

color INT = int;
color DATA = string;
color INTxDATA= product INT*DATA;
var n,k: INT;
var p, str: DATA;
val stop = "#####";
color Ten0 = int with 0..10;
color Ten1 = int with 1..10;
var s: Ten0; var r: Ten1;
fun Ok(s:Ten0, r:Ten1) = (r<=s);
  
```



Маркери, які на початку моделювання містяться в позиції

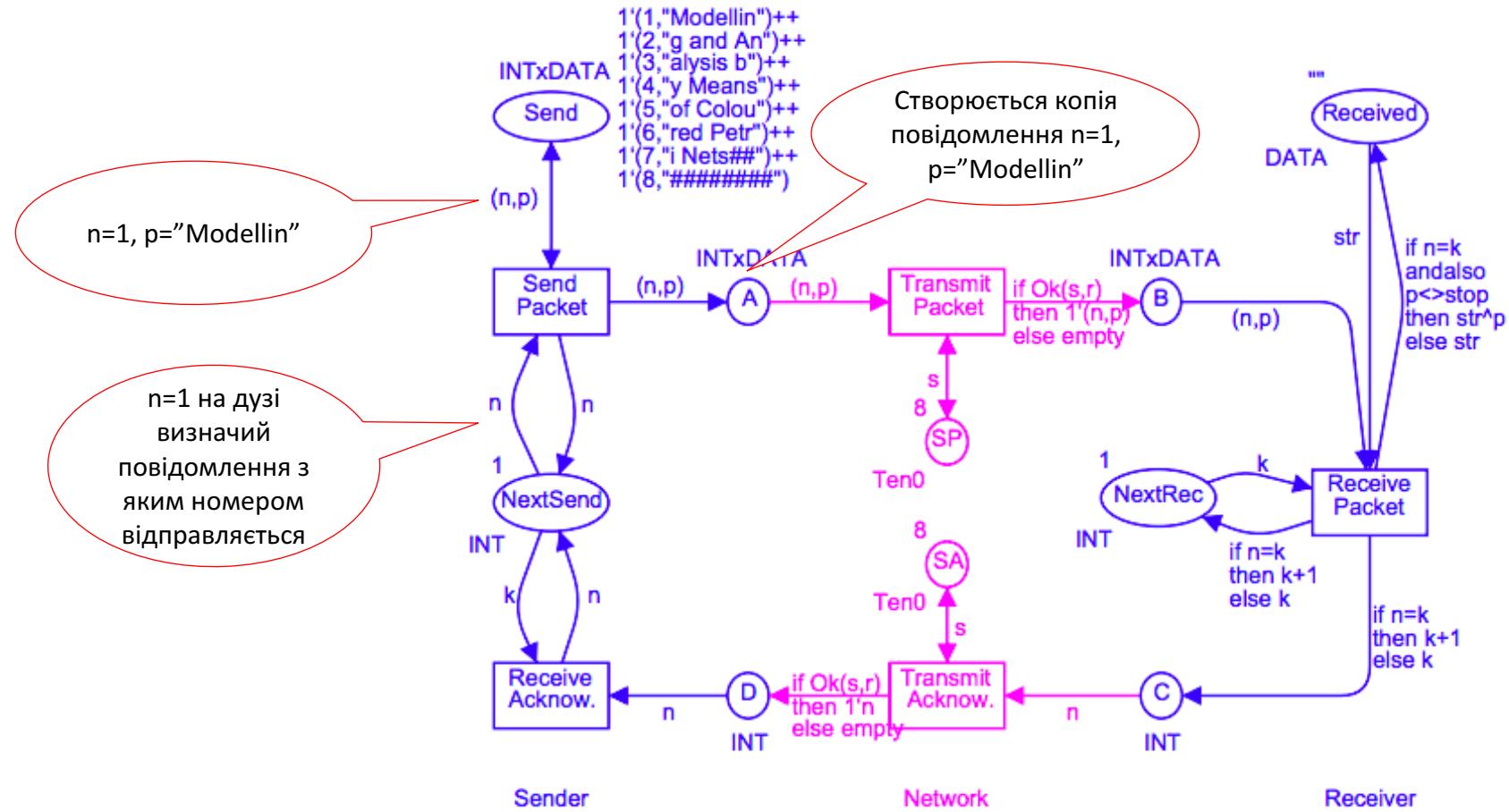
INTxDATA
Send
1'(1,"Modellin")++
1'(2,"g and An")++
1'(3,"alysis b")++
1'(4,"y Means")++
1'(5,"of Colou")++
1'(6,"red Petr")++
1'(7,"i Nets##")++
1'(8,"#####")++



Пояснення до параметрів дуг:

(n,p) \rightarrow $(1, "Modellin")$
 n \rightarrow 1 .

Перший крок: спрацьовує перехід SendPacket

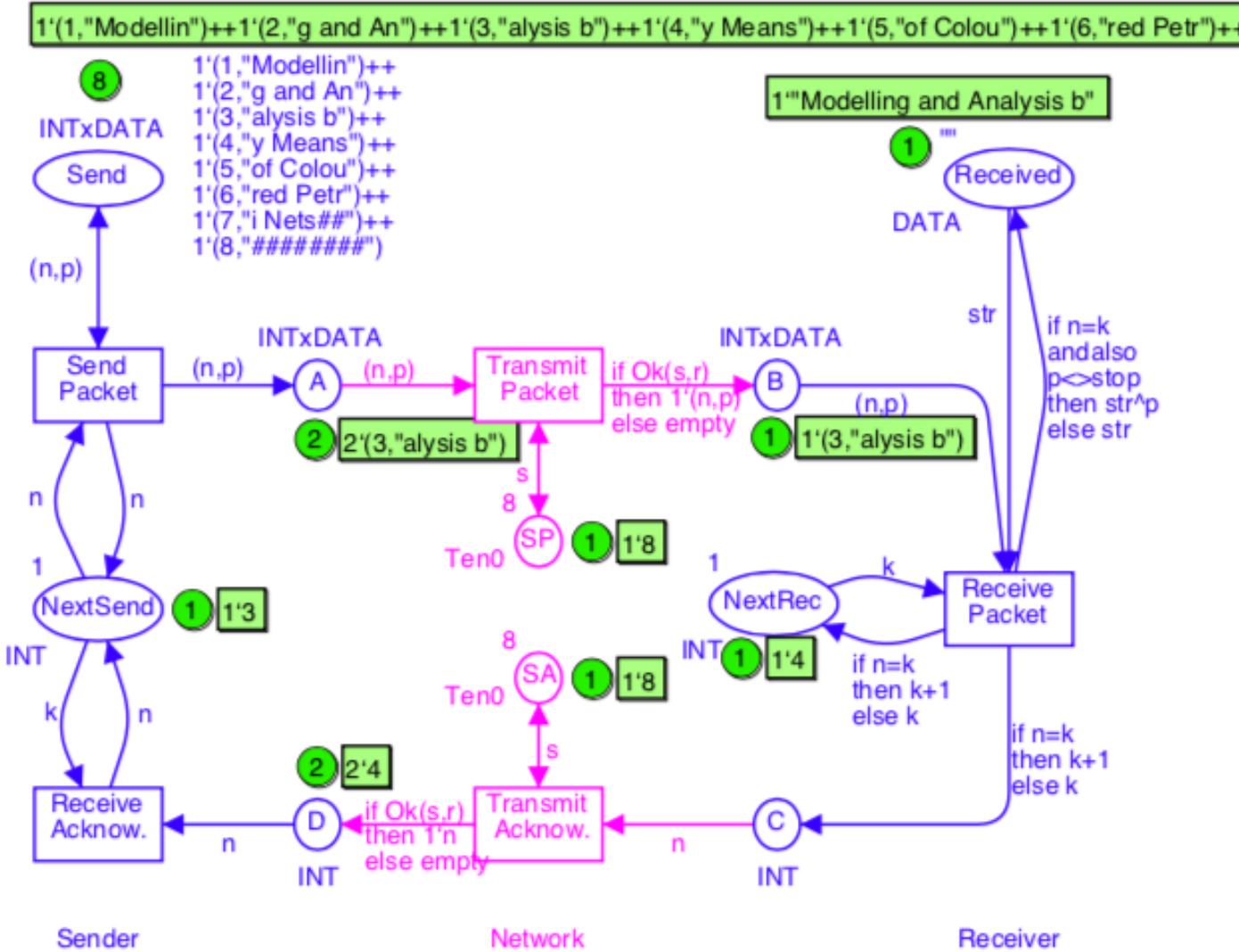


```

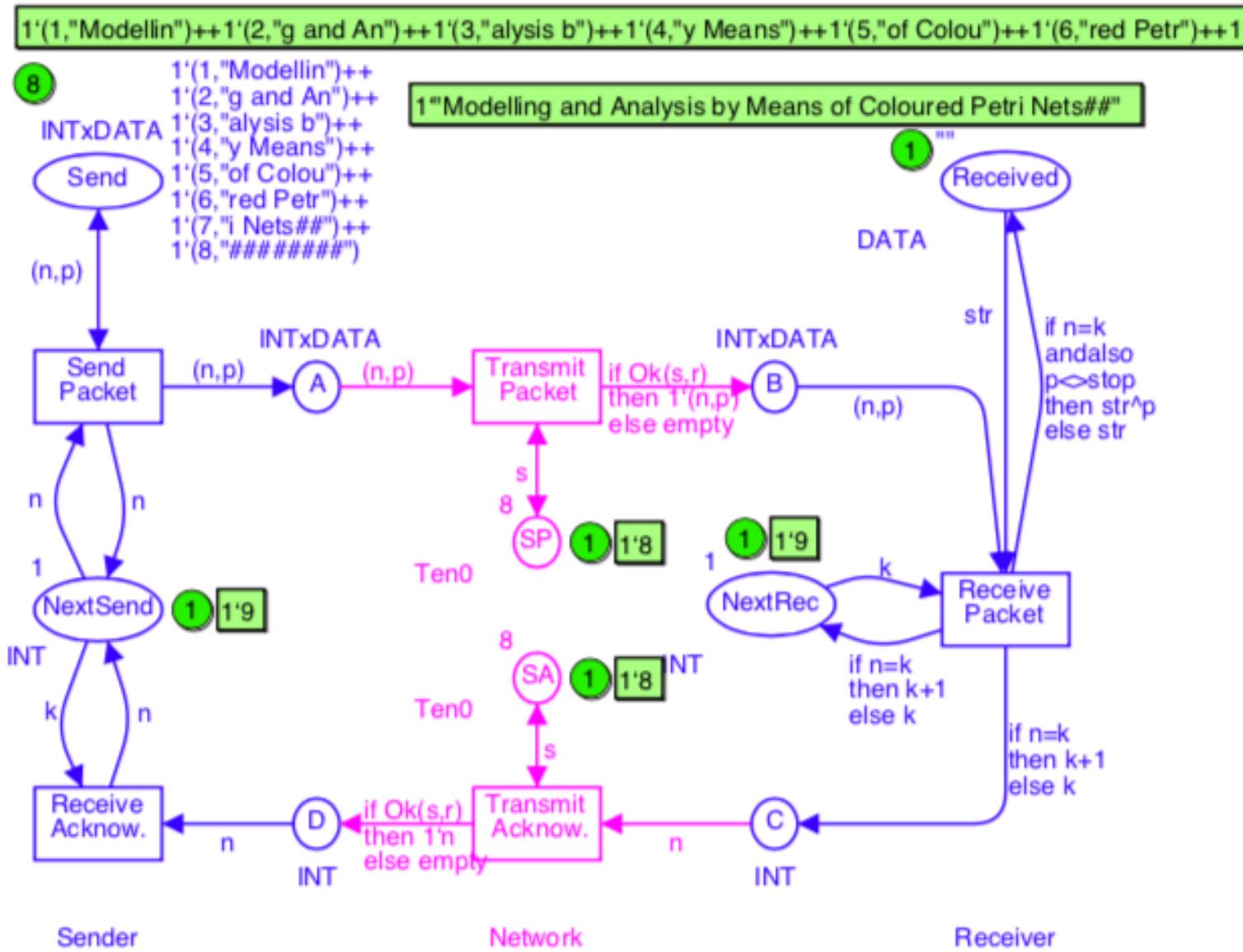
color INT = int;
color DATA = string;
color INTxDATA= product INT*DATA;
var n,k: INT;
var p, str: DATA;
val stop = "#####";
color Ten0 = int with 0..10;
color Ten1 = int with 1..10;
var s: Ten0; var r: Ten1;
fun Ok(s:Ten0, r:Ten1) = (r<=s);

```

Перебіг імітації (один з кроків)



Фінальне маркування



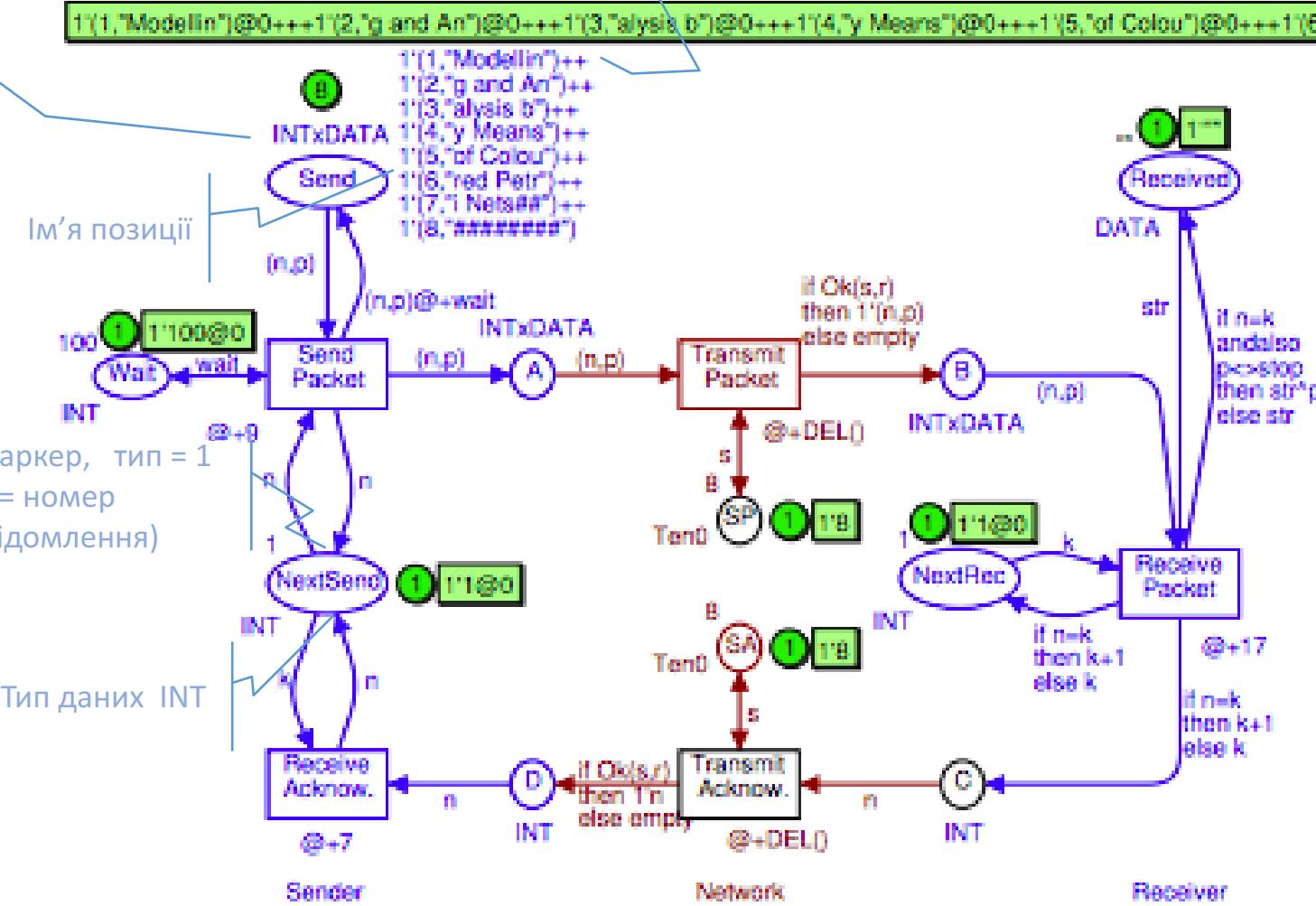
Модель передачі повідомлень з часовими затримками

Пояснення до реалізації в CPNTools

<http://cpntools.org/wp-content/uploads/2018/01/timedprotocol.pdf>

Опис типу маркера, INT and DATA

1' = Один маркер (1 = номер повідомлення в пакеті,
"Modellin" - вміст повідомлення)



```

color INT = int timed;
color DATA = string;
color INTxDATA= product INT*DATA;
var n,k, wait: INT;
var p, str: DATA;
val stop = "#####";

color Ten0 = int with 0..10;
color Ten1 = int with 1..10;
var s: Ten0; var r: Ten1;
fun Ok(s:Ten0, r:Ten1) = (r<=s);

color NetDelay = int with 25..27;
fun DEL() = NetDelay.ran();

```

Пояснення до параметрів дуг:

(n,p)	→	(1,"Modellin")
n	→	1.

CPNTools

- Програмне забезпечення для моделювання розфарбованими мережами Петрі: базові і з часовими затримками, імітація і аналіз властивостей, прості та ієрархічні.
- Документація <http://cpntools.org/2018/01/16/documentation-2/>

Програмне забезпечення GPSS (General Purpose Simulation System)

Основні компоненти ПЗ імітаційного моделювання дискретно-подійних систем

- Комплекс елементів для складання моделей.
- Графічний редактор для складання, редагування та збереження моделі з набору елементів.
- Модуль для збору статистичних даних (стандартний або з можливістю налаштування).
- Модуль для управління збором статистики (час моделювання, час розгону моделі, налаштування набору даних, по яких збирається статистика).
- Модуль для проведення експериментальних досліджень з моделлю (дослідження відгуку в часі, факторний експеримент, експеримент з метою оптимізації).
- Анімація процесу функціонування моделі (2D, 3D).
- Тестові моделі для демонстрації імітаційного моделювання.
- Модуль для аналізу властивостей моделі.

Сучасні тенденції розвитку ПЗ імітаційного моделювання

- Підтримка паралельної реалізації алгоритму імітації (одно чи багатопроцесорної)
- Підтримка віддаленої роботи з моделлю системи (хмарні технології)
- Предметно-орієнтовані середовища моделювання (VisSim, Network Simulator)

Найбільш відомі програмні продукти з імітаційного моделювання

- GPSS
- Arena Simulation
- Simio
- SIMUL8
- CPNTools
- PIPE
- AnyLogic
- https://en.wikipedia.org/wiki/List_of_discrete_event_simulation_software

- Для самостійного опрацювання