

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ  
СІКОРСЬКОГО»  
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курсова робота з освітнього компоненту  
«Технології паралельних обчислень. Курсова робота»

**Тема: Імітаційна модель системи процесорів на основі  
формального опису мережею Петрі**

**Керівник:**

ст.викл. Дифучина Олександра Юріївна

«Допущено до захисту»

---

«\_\_\_» \_\_\_\_\_ 2024 р.

Захищено з оцінкою

---

Члени комісії:

---

---

**Виконавець:**

Панченко Сергій Віталійович  
студент групи ІП-11  
залікова книжка № ІП-1123

«5» грудня 2024 р.

Інна СТЕЦЕНКО

Олександра ДИФУЧИНА

**Київ – 2024**

Національний технічний університет України  
“Київський політехнічний інститут імені Ігоря Сікорського”

Кафедра інформатики та програмної інженерії

Дисципліна «Моделювання систем»

Спеціальність 121 Інженерія програмного забезпечення

Курс 4 Група ІП-11 Семестр 1

## ЗАВДАННЯ

на курсову роботу студента

Панченка Сергія Віталійовича  
(прізвище, ім'я, по батькові)

---

1. Тема роботи «Імітаційна модель системи процесорів на основі формального опису мережею Петрі»

---

2. Термін здачі студентом закінченої роботи "5" грудня 2024р.

3. Зміст розрахунково-пояснювальної записки

1. Опис 2. Псевдокод 3. Реалізація 4. Реалізація 5. Реалізація 6. Проведення експериментів над моделями. Висновки.

4. Дата видачі завдання "29" жовтня 2024 року

---

## КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання курсової роботи	Термін виконання етапів роботи	Примітка
1	Отримання індивідуального завдання на курсову роботу	29.10.2024	
2	Розробка концептуальної моделі системи	29.11.2024	
3	Розробка формалізованої моделі системи	03.11.2024	
4	Алгоритмізація моделі системи та її програмна реалізація	08.11.2024	
5	Експериментальне дослідження моделі системи	13.11.2024	
6	Інтерпритація результатів моделювання, формулювання висновків та пропозицій	15.11.2024	
7	Оформлення пояснювальної записки	17.11.2024	
8	Захист КР	5.12.2024	

Студент \_\_\_\_\_ Панченко С. В.  
(підпис)

Керівник \_\_\_\_\_ Дифучина О.Ю.  
(підпис)

## АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка курсової роботи складається з 5 розділів, містить 2 рисунки, 2 таблиці, 1 додаток, 2 джерела.

Мета: визначення статистичних характеристик роботи багатопроцесорної обчислювальної системи, зокрема середнього значення, середнього квадратичного відхилення та закону розподілу для таких вихідних значень, як: часу виконання завдання в системі; завантаження чотирьох дисків, каналу передачі даних і процесорів; використання пам'яті; кількості завдань, які очікують виділення ресурсу, та часу їхнього очікування.

У розділі розробки концептуальної моделі було окреслено концептуальну модель та визначено необхідні властивості імітаційних алгоритмів.

У розділі аналіз розробки формалізованої моделі було описано моделі у рамках формалізму мереж Петрі.

У розділі алгоритмізації моделі та її реалізації було описано усі деталі щодо алгоритму та його реалізації.

У розділі експериментів на моделі було проведено верифікацію моделі та факторний експеримент.

У розділі інтерпретації результатів моделювання та експериментів було описано отримані результати в рамках проведеного моделювання та експериментування.

**КЛЮЧОВІ СЛОВА:** МОДЕЛЮВАННЯ СИСТЕМ, МЕРЕЖІ ПЕТРІ, БАГАТОПРОЦЕСОРНА ОБЧИСЛЮВАЛЬНА СИСТЕМА.

## ЗМІСТ

Постановка завдання.....	7
Вступ.....	9
1 Розробка концептуальної моделі.....	11
1.1 Опис задачі.....	11
1.2 Структурна схема моделі.....	11
1.3 Опис процесу.....	11
1.4 Вхідні змінні.....	12
1.5 Вихідні змінні.....	12
1.6 BPNM-діаграма.....	13
1.7 Висновки до розділу.....	15
2 Розробка формалізованої моделі.....	16
2.1 Вхідні параметри.....	17
2.2 Вихідні параметри.....	19
2.3 Діаграма.....	20
2.4 Висновки до розділу.....	23
3 Алгоритмізація моделі та її реалізація.....	24
3.1 Опис алгоритму імітації мережі Петрі.....	24
3.2 Модифікації алгоритму.....	27
3.2.1 Додавання розподілу Пуассона для генерації подій.....	27
3.2.2 Додавання списку кількості фішок.....	27
3.2.3 Вдосконалення роботи з NetLibrary.....	28
3.2.4 Покращення графічного інтерфейсу.....	28
3.2.5 Клас CourseWorkPetriSim.....	29
3.2.5.1 Опис атрибутів.....	29
3.2.5.2 Опис методів.....	30
3.2.6 Клас CourseWorkNet. Протокол подій.....	31
3.2.6.1 Опис атрибутів.....	31
3.2.6.2 Опис TaskObject.....	33

3.2.6.3 Опис методів.....	33
3.3 Верифікація та обчислення вихідних характеристик.....	35
4 Експерименти на моделі.....	40
5 Інтерпретація результатів моделювання та експериментів.....	41
Висновки.....	42
Список використаних джерел.....	43

## ПОСТАНОВКА ЗАВДАННЯ

Багатопроцесорна обчислювальна система складається з двох процесорів із загальною оперативною пам'яттю обсягом 131 сторінка, чотирьох накопичувачів на дисках, кожний із яких доступний обом процесорам, і одного каналу передачі даних. Завдання надходять у систему із середньою інтенсивністю, рівною 12 завданням у хвилину відповідно до розподілу Пуассона. Загальний час, необхідний процесору на обробку завдання, розподілено нормально з математичним сподіванням 10 секунд та середнім квадратичним відхиленням 3 секунди. Час обробки процесором включає переривання, необхідні для здійснення обміну по каналу вводу-виводу. Інтервали між перериваннями розподілені за негативний експоненціальний розподілом з математичним сподіванням, що дорівнює оберненій величині середньої інтенсивності операцій вводу-виводу завдання. Середня інтенсивність операцій введення-виведення розподілена рівномірно на інтервалі від 2 секунд до 10 секунд. Операції введення-виведення призначаються конкретному диску. Завданню, що надходить у систему, призначається пріоритет, що є величиною, оберненою до потреби в пам'яті. Потреба завдання в пам'яті розподілена рівномірно в інтервалі від 20 до 60 сторінок. Як тільки пам'ять виділена для завдання, один з вільних процесорів починає його обробку. При видачі запиту на здійснення введення-виведення завдання може продовжувати використання процесора доти, доки в черзі залишиться тільки один запит. Таким чином, якщо зроблений запит на здійснення введення-виведення і один запит вже очікує в черзі, то процесор звільняється, а запит на введення-виведення розміщується в черзі. Після виконання поточного запиту введення-виведення процесор може відновити обробку завдання в тому випадку, якщо вона вільна. Після переривання процесора автоматично виконується запит введення-виведення з призначеним завданню диском. Таким чином, здійснюється прямий доступ до диска з процесора. Передбачається, що час позиціонування диска розподілено рівномірно на інтервалі від 0,0 до 0,075 секунд. Одночасно може здійснюватися

тільки одна операція позиціонування диска. Після позиціонування здійснюється обмін даними по каналу передачі даних. Час обміну дорівнює  $0,001 \times (2,5 + h)$ , де  $h$  - рівномірно розподілена на інтервалі від 0 до 25 величина. Після здійснення обміну запит введення-виведення вважається виконаним. Визначити загальний час виконання завдання в системі, а також статистичні оцінки завантаження усіх чотирьох дисків, каналу передачі даних та обох процесорів. Крім того, необхідно одержати оцінку середнього використання пам'яті, статистику щодо кількості завдань, які очікують виділення ресурсу, та щодо часу очікування.



## ВСТУП

У сучасному світі багатопроцесорні обчислювальні системи відіграють ключову роль у вирішенні складних обчислювальних задач. Ефективність їх роботи залежить від багатьох факторів, включаючи управління пам'яттю, розподіл процесорного часу та організацію введення-виведення. Тому дослідження характеристик таких систем є актуальним завданням для оптимізації їх роботи та підвищення продуктивності.

Метою даної курсової роботи є визначення статистичних характеристик роботи багатопроцесорної обчислювальної системи, зокрема середнього значення, середнього квадратичного відхилення та закону розподілу для таких вихідних значень, як-от: часу виконання завдання в системі; завантаження чотирьох дисків, каналу передачі даних і процесорів; використання пам'яті; кількості завдань, які очікують виділення ресурсу, та часу їхнього очікування.

Для досягнення поставленої мети використовується апарат мереж Петрі, який є потужним математичним інструментом для моделювання та аналізу паралельних та розподілених систем. Мережі Петрі дозволяють ефективно описувати та досліджувати асинхронні, паралельні процеси в системі, враховуючи стохастичну природу процесів, що відбуваються в ній. У роботі застосовуються різні типи ймовірнісних розподілів для моделювання надходження завдань (розподіл Пуассона), часу обробки завдань (нормальний розподіл), інтервалів між перериваннями (експоненційний розподіл) та інших характеристик системи.

Для реалізації моделі використовується спеціалізоване програмне забезпечення PetriObjModelPaint, розроблене для моделювання та аналізу мереж Петрі. Цей інструмент надає можливості для створення, візуалізації та дослідження мережевих моделей, а також дозволяє отримувати статистичні оцінки завантаження всіх компонентів системи, включаючи процесори, диски, канал передачі даних, аналізувати використання пам'яті та характеристики черг завдань.

Результати дослідження дозволять оцінити ефективність роботи системи та виявити потенційні "вузькі місця" в її функціонуванні, що може бути використано для подальшої оптимізації роботи багатопроцесорних обчислювальних систем.

## **1 РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ**

У цьому розділі представлено концептуальну модель багатопроцесорної обчислювальної системи, яка включає опис її складових, основних процесів, правил взаємодії елементів та параметрів, необхідних для подальшого моделювання і аналізу.

### **1.1 Опис задачі**

Мета моделювання – визначити показники роботи багатопроцесорної обчислювальної системи, зокрема:

- загальний час виконання завдань;
- завантаження дисків, каналу передачі даних і процесорів;
- середнє використання пам'яті;
- кількість завдань, що очікують на виділення ресурсу;
- час очікування виділення пам'яті.

### **1.2 Структурна схема моделі**

Об'єктом моделювання є багатопроцесорна система, яка включає:

- два процесори, що використовують спільну оперативну пам'ять на 131 сторінку;
- чотири диски, доступні обом процесорам;
- один канал передачі даних, який використовується для обміну між процесорами та дисками.

### **1.3 Опис процесу**

Процес складається з таких елементів:

- завдання надходять у систему з інтенсивністю 12 завдань/хв за розподілом Пуассона;
- кожному завданню призначається пріоритет, обернений до його потреби в пам'яті;

- потреба завдань у пам'яті розподілена рівномірно в межах 20–60 сторінок;
- якщо пам'ять доступна, одразу призначається один із вільних процесорів;
- час виконання завдання розподілений нормально із середнім 10 секунд та відхиленням 3 секунди;
- у процесі виконання завдань виникають переривання для обміну даними; інтервали між перериваннями розподілені експоненційно з параметром, оберненим до інтенсивності операцій введення-виведення (2–10 секунд);
- операції введення-виведення включають;
- позиціонування диска ( $0-0,075$  с);
- передачу даних ( $0,001 \times (2,5+h)$ ), де  $h$  розподілено рівномірно в межах 0–25);
- завдання звільняє процесор, якщо черга операцій введення-виведення складається з максимум одного завдання; в іншому випадку, воно продовжує займати процесор.

#### 1.4 Вхідні змінні

Модель має такі вхідні змінні, як-от:

- інтенсивність надходження завдань (12 завдань/хв);
- потреба завдань у пам'яті (20–60 сторінок);
- час виконання завдання процесором (нормальний розподіл: середнє 10 с, відхилення 3 с);
- інтенсивність операцій введення-виведення (2–10 с);
- час позиціонування диска ( $0-0,075$  с);
- час передачі даних ( $0,001 \times (2,5+h)$ ),  $h$  у межах 0–25).

#### 1.5 Вихідні змінні

Модель має такі вихідні змінні, як-от:

- загальний час виконання завдань;
- завантаження процесорів, дисків, каналу передачі даних;
- середнє використання пам'яті;
- кількість завдань, що очікують виділення пам'яті чи ресурсів;
- час очікування виділення пам'яті.

## 1.6 BPMN-діаграма

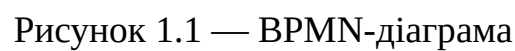
BPMN-діаграма (Business Process Model and Notation[1]) – це графічне представлення процесів, яке дозволяє детально описати їхню послідовність, взаємодію елементів системи та потік даних між ними. Вона використовується для візуалізації концептуальної моделі, допомагаючи зрозуміти логіку роботи системи, виявити можливі вузькі місця та полегшити подальшу формалізацію і моделювання.

На рисунку 1.1 представлено BPMN-діаграму моделі, яка відображає ключові етапи обробки завдань і взаємодію різних компонентів системи через обмін повідомленнями. Діаграма ілюструє потік даних і послідовність дій від генерації завдання до його завершення.

Розглянемо блок генерації завдань. Через визначений проміжок часу, вказаний у коментарі, відбувається подія створення нового завдання. Після цього завданню призначається обсяг пам'яті (кількість сторінок), необхідний для його обробки, і воно передається в оперативну пам'ять.

Оперативна пам'ять отримує повідомлення про нове завдання, додає його в чергу й перевіряє наявність вільних сторінок для виконання хоча б одного завдання з черги. Щойно вільна пам'ять стає доступною, вона виділяється, і завдання надсилається до процесорів.

Процесори приймають повідомлення про нове завдання і додають його в чергу. Як тільки один із процесорів стає вільним, він починає обробку завдання. По завершенню обробки перевіряються три умови: наявність вільного диска, невиконаний запит введення-виведення та отримання відповідних повідомлень. Коли ці умови виконано, процесор звільняється, а система надсилає кілька



Розглянемо блок генератора запитів. Через заданий час відбувається подія створення нового запиту, який додається в чергу. Далі постійно перевіряється наявність хоча б одного невиконаного запиту в черзі. Якщо такий запит є, повідомлення надсилається процесорам.

Коли диск отримує повідомлення про готовність захопити завдання, він розпочинає позиціонування своєї головки зчитування, після чого виконується передача даних через канал введення-виведення. Завдання вважається завершеним після успішного виконання цих операцій.

### **1.7 Висновки до розділу**

У висновку можна зазначити, що розроблена концептуальна модель багатопроцесорної обчислювальної системи забезпечує базу для формалізації процесів, побудови алгоритму імітації та проведення експериментального дослідження. Визначені вхідні змінні, параметри та вихідні характеристики дозволяють ефективно аналізувати роботу системи та оптимізувати її продуктивність.

## 2 РОЗРОБКА ФОРМАЛІЗОВАНОЇ МОДЕЛІ


У цьому розділі представлено формалізовану модель багатопроцесорної обчислювальної системи, розроблену з використанням мережі Петрі. Модель описує елементи системи, такі як пам'ять, процесори, диски та канал передачі даних, а також події, що визначають їх взаємодію. Вказано числові параметри для кожного компонента, правила спрацьовування переходів, а також описано генерацію випадкових величин для вхідних змінних. У розділі також подано формули для обчислення вихідних характеристик системи, які дозволяють аналізувати її продуктивність, завантаженість та ефективність роботи. Створена модель є основою для подальшого алгоритму імітації та експериментального дослідження.

Мережа Петрі – це математичний апарат для моделювання дискретних систем, який складається з позицій, переходів та дуг. Позиції представляють стани системи, переходи – події або дії, що змінюють ці стани, а дуги визначають зв'язки між позиціями і переходами. Стан мережі визначається кількістю маркерів у позиціях, які називаються токенами [2][с. 67].


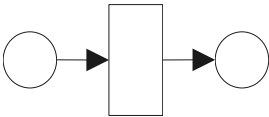



В основі роботи мережі лежить правило спрацьовування переходу: якщо у всіх вхідних позиціях переходу кількість маркерів не менша за кратність відповідних дуг, перехід активується. Під час спрацьовування маркери переміщуються з вхідних позицій до вихідних відповідно до кратності дуг. Мережа Петрі дозволяє описувати процеси з конфліктними переходами, багатоканальні операції та стохастичні події, що забезпечує її високу гнучкість для моделювання складних систем.

У таблиці 2.1 наведено елементи формалізації мережі Петрі [2][с. 69].

Таблиця 2.1 Мережа Петрі

Назва	Позначення	Опис
Перехід		Позначає подію



Назва	Позначення	Опис
Позиція		Позначає умову
Дуга		Позначає зв'язки між подіями та умовами
Маркер(один)		Позначає виконання (або не виконання) умови
Багато фішок		Позначає багатократне виконання умови
Багато дуг		Позначає велику кількість зв'язків

## 2.1 Вхідні параметри

Розглянемо параметри переходів моделі у таблиці 2.2. Нехай маємо відрізок  $[M; N]$ , що позначає діапазон можливої кількості сторінок, що завдання може займати. Нехай  $K \in [M; N]$ , де  $K$  — кількість сторінок, що займає довільне завдання.

Таблиця 2.2 Параметри переходів

Назва переходу	Часова затримка	Значення пріоритету	Значення ймовірності запуску
generate_task	Poisson(5.0)	0	1

Назва переходу	Часова затримка	Значення пріоритету	Значення ймовірності запуску
generate_task_K_pages	0	0	$\frac{1}{M-N}$
fail_allocate_task_K_pages	0	-1	1
try_allocate_task_K_pages	0	0	1
wait_allocate_task_20_pages	0	0	1
process_task_K_pages	Norm(10, 3)	$N-K$	1
generate_io_request	Unif(2, 10)	0	1
create_io_task_20_pages	0	$N-K$	1
take_up_disk_task_K_pages	0	$N-K$	1
generate_interrupt_K_pages	Exp(6)	0	1
drop_interruprt	0	-1	1
place_disk_K_pages	Unif(0, 0.075)	$N-K$	1
io_channel_transfer_task_K_pages	Unif(0.0025, 0.0275)	$N-K$	1

Розглянемо параметри позицій у таблиці 2.3.

Таблиця 2.3 Параметри позицій

Назва позиції	Початкове значення
generator_task	1
generated_task	0
task_K_pages	0

Назва позиції	Початкове значення
fail_allocate_token_task_K_pages	0
allocated_task_K_pages	0
total_wait_allocate_task	0
pages	131
generator_io_request	1
processors	2
generated_request	0
io_task_K_pages	0
generator_interrupt	1
generated_interrupt	0
busy_disk_task_K_pages	0
free_disks	4
disk_placed_task_K_pages	0
is_disk_placement_available	1
finished_tasks_task_K_pages	0
finished_tasks	0

## 2.2 Вихідні параметри

Для початку визначимо загальний час виконання завдання в системі. Для цього зафіксуємо моменти часу, коли відбувається вихід з переходів `generate_task_K_pages` та `finish_task_K_pages`. Тоді ми, маючи різницю цих двох моментів часу, можемо визначити час, що провело певне завдання у системі. Далі можна знайти суму усіх відрізків часу, розділити її на кількість виконаних завдань та отримати середній витрачений завданням час в системі.

$$\Delta T_{\text{average time in system}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{K \text{ pages}}} [T_{K \text{ pages, finish, } i} - T_{K \text{ pages, start, } i}] \right]}{\sum_{K=M}^N I_{K \text{ pages}}},$$

де  $N$  - максимальна кількість сторінок,  $M$  - мінімальна кількість сторінок,  $I$  - кількість завершених з кількістю сторінок  $K$ .

Час очікування на виділення пам'яті розраховується аналогічно:

$$\Delta T_{\text{average wait allocation}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{K \text{ pages}}} [T_{K \text{ pages, wait alloc, } i} - T_{K \text{ pages, fail alloc, } i}] \right]}{\sum_{K=M}^N I_{K \text{ pages}}}.$$

Завантаження є результат ділення часу, що елемент працював, до всього часу моделювання.

$$L_{\text{procs}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{\text{procs task, } K, \Delta T}} \Delta T_{\text{procs task, } K, i} \right]}{\sum_{K=M}^N I_{\text{process task, } K, \Delta T}}, L_{\text{disks}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{\text{place disk, } K, \Delta T}} \Delta T_{\text{place disk, } K, i} \right]}{\sum_{K=M}^N I_{\text{place disk, } K, \Delta T}},$$

$$L_{\text{io channel}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{\text{io channel transfer, } K, \Delta T}} \Delta T_{\text{io channel transfer, } K, i} \right]}{\sum_{K=M}^N I_{\text{io channel transfer, } K, \Delta T}},$$

де  $I_{\text{io channel transfer}}$ ,  $I_{\text{place disk}}$ ,  $I_{\text{procs task}}$  - це кількість виходів з відповідних переходів.

## 2.3 Діаграма

На рисунку 2.1 можна розглянути діаграму мережі Петрі. Для зручності вона побудована лише з двома типами завдань. Це робить її менш нагромадженою, оскільки частини схеми для кожного завдання є однаковими.

Основними компонентами мережі є генератор завдань, оперативна пам'ять, процесори, генератор запитів введення-виведення, диски, канал передачі даних та блок завершення роботи.



відповідають їхньому обсягу пам'яті, наприклад, `task_20_pages` для завдань із потребою у 20 сторінок.

Оперативна пам'ять представлена позицією `pages`, яка відображає загальну кількість доступних сторінок (131 сторінка). Переходи `try_allocate` і `wait_allocate` перевіряють наявність вільної пам'яті та виділяють її для завдань відповідного обсягу. У разі успішного виділення токени передаються до позиції `allocated`, а у разі невдачі – у `fail_allocate_token`.

Процесори моделюються позицією `processors`, яка визначає кількість доступних процесорів (2 процесори). Переходи `process` відповідають за обробку завдань із використанням нормального розподілу часу.

Генератор запитів введення-виведення моделюється позицією `generated_io_request`, яка активує перехід `create_io`. Цей перехід генерує запити введення-виведення для завдань, які були оброблені процесорами.

Диски представлені позицією `free_disks`, яка відображає кількість доступних дисків (4 диски). Перехід `place_disk` імітує позиціонування головки зчитування з використанням рівномірного розподілу часу. Після виконання цієї операції токени передаються через канал введення-виведення.

Канал введення-виведення моделюється позицією `is_disk_placement_available`, яка визначає готовність каналу до передачі даних. Перехід `io_channel_transfer` відповідає за передачу даних із використанням рівномірного розподілу часу. Завершення роботи моделюється позицією `finished_tasks`, яка накопичує виконані завдання. Це дозволяє аналізувати продуктивність системи та обчислювати необхідні показники.

Логіка роботи мережі ґрунтується на потоках токенів між позиціями через переходи. Завдання генеруються з певним пріоритетом і обсягом пам'яті, який визначається рівномірним розподілом (від 20 до 60 сторінок). Якщо завдання не отримало пам'ять, воно повертається до черги. Виділені завдання передаються до процесорів, які здійснюють їх обробку із врахуванням стохастичних затримок. Після обробки завдання надсилають запити на диски. Якщо диск доступний, він виконує операцію зчитування та передачу даних через канал

введення-виведення. Завдання вважається завершеним після успішної передачі даних і повернення ресурсів системі, включаючи пам'ять, процесори та диски.

## **2.4 Висновки до розділу**

У цьому розділі було створено формалізовану модель багатопроцесорної обчислювальної системи, яка дозволяє математично описати її компоненти та взаємодії. Модель побудована з використанням мережі Петрі, що забезпечує можливість детального моделювання процесів генерації, обробки завдань, запитів введення-виведення та управління ресурсами. Було визначено всі необхідні вхідні параметри, їх числові характеристики, а також формули для обчислення вихідних характеристик системи. Створена модель є основою для розробки алгоритму імітації та проведення експериментального дослідження ефективності роботи системи.

### 3 АЛГОРИТМІЗАЦІЯ МОДЕЛІ ТА ЇЇ РЕАЛІЗАЦІЯ

Цей розділ присвячений алгоритмізації та програмній реалізації імітаційної моделі багатопроцесорної обчислювальної системи, створеної за допомогою мережі Петрі. Метою є опис роботи стандартного алгоритму моделювання, наданого бібліотекою PetriObjModelPaint [3], а також внесених модифікацій, які забезпечують відповідність алгоритму вимогам поставленої задачі. Розділ містить опис основних етапів алгоритму, реалізацію додаткового функціоналу, оптимізацію коду під конкретну задачу, а також протокол подій, що фіксуються під час моделювання. Наприкінці представлено результати верифікації моделі, що підтверджують коректність її реалізації.

#### 3.1 Опис алгоритму імітації мережі Петрі

Алгоритм імітації мережі Петрі базується на послідовному просуванні моделі в часі та обробці подій, які відбуваються у системі. Основна мета алгоритму — забезпечити виконання правил роботи мережі Петрі, таких як спрацювання переходів за наявності необхідних умов та обчислення статистичних характеристик позицій і переходів.

Алгоритм визначає мінімальний час спрацювання серед усіх активних переходів (eventMin) і просуває глобальний час моделі до цього моменту. Це забезпечує обробку подій у правильному хронологічному порядку.

На кожному етапі перевіряються активні переходи. Якщо перехід може спрацювати (умови виконуються), він захоплює необхідні маркери з пов'язаних позицій. Конфлікти між переходами вирішуються за пріоритетами або ймовірностями.

Після спрацювання перехід розподіляє маркери по вихідних позиціях. Якщо перехід має накопичувальний буфер, процес виходу маркерів може повторюватися, поки буфер не стане порожнім.

Алгоритм збирає статистичні дані для кожної позиції і переходу: середню кількість маркерів у позиціях і середній час обробки переходів.



Якщо кілька переходів можуть спрацювати одночасно, вибір здійснюється на основі пріоритетів або ймовірностей. Це забезпечує гнучкість і відповідність алгоритму до стохастичної природи системи.

Розглянемо псевдокод алгоритму нижче. Функція `go` виконує ітерації, поки поточний час не досягне часу моделювання. На кожній ітерації обчислюються статистичні показники, відбувається перехід до часу наступної події, а також виконується обробка поточного переходу.

Функція `input` шукає активні переходи, які можуть спрацювати. Виконує вирішення конфліктів між переходами, які мають однаковий пріоритет.

Функція `output` Обробляє спрацьовування переходів і розподіл маркерів у відповідних позиціях. Виконує додаткові перевірки для переходів із накопичувальними буферами, щоб повністю обробити всі події, які можуть відбутися в поточний момент часу.

---

```

1 def go():
2     # Initialize simulation time and current time
3     set_simulation_time(time_modelling)
4     set_current_time(0)
5     # Initialize transitions and positions
6     input() # Activate initial transitions
7     # Main simulation loop
8     while current_time < simulation_time:
9         # Update statistics for the current time period
10        do_statistics()
11        # Move to the next event's time
12        set_current_time(get_time_min())
13        # Log event details (if logging is enabled)
14        if logging_enabled:
15            log("Time:", current_time, "; Event:", event_min.get_name())
16
17        # Process the event and its effects
18        output()
19        # Reactivate transitions after event execution
```

---

---

```

20     input()
21
22 def input():
23     # Find all active transitions
24     active_transitions = find_active_transitions()
25     # If no transitions are active and all buffers are empty, stop further processing
26     if active_transitions is empty and all_buffers_are_empty():
27         set_time_min(infinity)
28     else:
29         # While there are active transitions
30         while active_transitions is not empty:
31             # Resolve conflicts between active transitions
32             selected_transition = resolve_conflicts(active_transitions)
33             # Activate the selected transition
34             selected_transition.act_in(listP, current_time)
35             # Update the list of active transitions
36             active_transitions = find_active_transitions()
37             # Update the minimum time for the next event
38             find_event_min()
39
40 def output():
41     # Ensure that the simulation time has not been exceeded
42     if current_time <= simulation_time:
43         # Execute the event with the minimum time
44         event_min.act_out(listP, current_time)
45
46     # Handle remaining events in the transition's buffer
47     while event_min.has_buffer():
48         event_min.update_event_time()
49         if event_min.get_min_time() == current_time:
50             event_min.act_out(listP, current_time)
51         else:
52             break
53     # Check other transitions for events occurring at the current time

```

---

---

```

54     for transition in listT:
55         if transition.has_buffer() and transition.get_min_time() == current_time:
56             transition.act_out(listP, current_time)
57             # Process additional events in the buffer
58         while transition.has_buffer():
59             transition.update_event_time()
60             if transition.get_min_time() == current_time:
61                 transition.act_out(listP, current_time)
62             else:
63                 break

```

---

### 3.2 Модифікації алгоритму

У процесі реалізації моделі були внесені зміни до базового алгоритму, щоб адаптувати його до специфіки поставленої задачі. Основні модифікації стосувалися покращення роботи мережі Петрі та збору додаткової інформації для аналізу. Зміни можна поділити на два напрямки: функціональні вдосконалення алгоритму та зручність роботи з мережами.

#### 3.2.1 Додавання розподілу Пуассона для генерації подій

Для точного моделювання вхідних потоків завдань у систему було реалізовано підтримку розподілу Пуассона. Це дозволило описати інтенсивність надходження завдань у вигляді стохастичного процесу, що відповідає заданій середній інтенсивності. Реалізація виконана шляхом додавання відповідної функції до генератора подій у мережі.

---

```

1  public static double poisson(final double timeMean) {
2      PoissonDistribution poisson = new PoissonDistribution(timeMean);
3      return poisson.sample();
4  }

```

---

#### 3.2.2 Додавання списку кількості фішок

Для того щоб дізнатися розподіл кількості завдань, що очікують виділення пам'яті, треба реєструвати число фішок в цій позиції. Для цього достатньо

модифікувати метод `changeMean`. На кожну зміну середнього числа записувати поточне число фішок.

---

```

1 private ArrayList<Integer> marks = new ArrayList<>();
2 public void changeMean(double a) {
3     marks.add(mark);
4     mean = mean + (mark - mean) * a;
5 }
6 public ArrayList<Integer> getMarks() {
7     return marks;
8 }

```

---

### 3.2.3 Вдосконалення роботи з NetLibrary

Для зручності роботи з мережами Петрі було додано клас `NetLibraryManager`, який дозволяє динамічно додавати методи в бібліотеку `NetLibrary`. Завдяки цьому можна легко інтегрувати нові мережі в графічний інтерфейс, використовувати рефлексію для виклику методів і забезпечувати їх коректність під час компіляції. Також додано перевірку методів генерації мережі за допомогою анотацій `NetLibraryMethod` та `NetLibraryMethodProcessor`.

---

```

1 @NetLibraryMethod
2 public static PetriNet createNet() throws ExceptionInvalidTimeDelay {
3     final CourseWorkNet net = new CourseWorkNet();
4     return net.net;
5 }

```

---

### 3.2.4 Покращення графічного інтерфейсу

Був виправлений метод `generateGraphNetBySimpleNet`, що забезпечує коректне графічне відображення мережі Петрі. Помилка полягала у тому, що оскільки не було перевірки на включення переходів та позицій в контейнери `inTrans` та `inPlaces`, то дублікати додавалися у графічне відображення мережі.

---

```

1 if (
2     !inTrans.contains(tran) // Нова перевірка
3     && tran.getNumber() == outArc.getNumT()

```

---

---

```

4 ) {
5   inTrans.add(tran);
6 }
7 // ...
8 if (
9   !inPlaces.contains(place) // Нова перевірка
10  && place.getNumber() == inArc.getNumP()
11 ) {
12   inPlaces.add(place);
13 }

```

---

### 3.2.5 Клас CourseWorkPetriSim

Клас CourseWorkPetriSim є модифікованою версією базового симулятора мережі Петрі. Він реалізує алгоритм імітації, включаючи обробку подій, просування часу, збір статистики та вирішення конфліктів між переходами.

#### 3.2.5.1 Опис атрибутів

Одним з ключових атрибутів є timeState - об'єкт класу StateTime, який зберігає інформацію про поточний час моделювання (currentTime) та час завершення моделювання (simulationTime). Він є центральним для управління часом у моделі, дозволяючи просувати глобальний час до наступної події.

Атрибут timeMin містить мінімальний час спрацювання серед усіх переходів у мережі. Цей атрибут оновлюється під час виконання моделі, щоб визначити найближчу подію для обробки.

У класі також присутній масив listP, що містить усі позиції мережі Петрі (PetriP). Кожен елемент описує стан позиції, зокрема кількість маркерів у ній. Цей масив використовується для перевірки умов активації переходів і переміщення маркерів.

Важливим компонентом є масив listT, що містить усі переходи мережі Петрі (PetriT). У ньому зберігається інформація про затримки, умови спрацювання, ймовірності, пріоритети, буфери та час наступного спрацювання кожного переходу.

Останнім ключовим атрибутом є `eventMin` - об'єкт типу `PetriT`, що представляє перехід із мінімальним часом спрацювання. Цей атрибут визначає, який перехід буде оброблено наступним.

### 3.2.5.2 Опис методів

Метод `go` є головним у класі. Він виконує симуляцію моделі, ініціалізуючи час моделювання та поточний час. У циклі він забезпечує просування часу до наступної події, виконує обробку цієї події (`output`) та активує нові переходи (`input`). Логування подій виконується за потреби.

Метод `input` шукає переходи, які можуть бути активовані, і виконує їх активацію. Він також вирішує конфлікти між переходами, якщо вони мають однакові пріоритети, вибираючи один із них або за допомогою ймовірностей, або випадковим чином. Після активації переходу оновлюється список активних переходів, а також мінімальний час спрацювання наступної події.

Метод `output` відповідає за обробку подій. Він виконує спрацювання переходу, що має найменший час, та переміщує маркери з буфера переходу у відповідні позиції. У разі, якщо перехід має додаткові події в буфері, вони обробляються в поточний момент часу.

Метод `doStatistics` відповідає за збір статистичних даних під час імітації. Він оновлює середнє значення кількості маркерів у кожній позиції та середнє значення часу затримки в кожному переході.

Метод `findActiveT` знаходить усі переходи, які можуть спрацювати на поточному етапі імітації, перевіряючи умови для їх активації. Якщо кілька переходів активні одночасно, вони сортуються за пріоритетом.

Метод `doConflict` вирішує конфлікти між активними переходами. Якщо кілька переходів мають однаковий пріоритет, вибір здійснюється або випадково, або на основі їхніх ймовірностей спрацювання.

Методи `setTimeCurr`, `getCurrentTime`, `setSimulationTime` та `getSimulationTime` використовуються для управління часом моделювання, встановлюючи або повертаючи поточний час та загальний час моделювання.

Метод printMark виводить у консоль кількість маркерів у кожній позиції мережі, що дозволяє візуально оцінити стан моделі на кожному етапі симуляції.

Нарешті, метод isEmpty перевіряє, чи всі буфери переходів порожні, що дозволяє визначити, чи є ще події для обробки в моделі.

### **3.2.6 Клас CourseWorkNet. Протокол подій**

Клас CourseWorkNet розроблений для аналізу та моделювання роботи багатопроцесорної обчислювальної системи за допомогою мережі Петрі. Він є обгорткою для створення та налаштування компонентів мережі, забезпечуючи зручний інтерфейс для опису та управління моделлю.

#### **3.2.6.1 Опис атрибутів**

Позиція generated\_task зберігає кількість згенерованих завдань у системі. Ця позиція використовується для активізації переходів, що створюють нові завдання. Її основне призначення полягає у фіксації вхідного потоку завдань до системи.

Позиція generated\_io\_request відповідає за генерацію запитів введення-виведення для завдань після їх обробки. Її призначення полягає в забезпеченні інтеграції завдань із підсистемою дисків.

Позиція generated\_interrupt використовується для моделювання переривань, які запускають запити до дисків. Її призначення - імітувати реальні події введення-виведення у багатозадачній системі.

Позиція processors відображає кількість вільних процесорів у системі, яка початково становить два. Вона використовується для визначення доступності процесорів для обробки завдань.

Позиція pages відображає кількість доступних сторінок пам'яті в системі, початково маючи 131 сторінку. Її призначення полягає у визначенні можливості виділення пам'яті для завдань.

Позиція free\_disks відображає кількість доступних дисків, яких початково чотири. Вона забезпечує розподіл завдань між доступними дисками.

Позиція `total_wait_allocate_task` підраховує загальний час очікування завдань на виділення пам'яті. Вона використовується для аналізу затримок у системі.

Позиція `finished_tasks` зберігає кількість завершених завдань. Її призначення полягає у відображенні загальної продуктивності системи.

Позиція `is_disk_placement_available` вказує на доступність каналу введення-виведення для передачі даних. Вона контролює одночасний доступ до каналу.

Перехід `generate` генерує нові завдання у системі із заданими параметрами, такими як обсяг пам'яті. Його призначення полягає у забезпеченні створення потоку завдань.

Перехід `try_allocate` перевіряє наявність вільної пам'яті для завдань. У разі успіху пам'ять виділяється. Його призначення - запускати обробку завдань, які отримали доступ до пам'яті.

Перехід `fail_allocate` відображає спробу виділення пам'яті, яка завершилась невдачею. Його призначення полягає у поверненні завдання у чергу для подальших спроб.

Перехід `wait_allocate` переводить завдання з черги у стан обробки після виділення пам'яті. Цей перехід забезпечує повторне надання пам'яті для завдань.

Перехід `process` імітує обробку завдань процесорами із використанням нормального розподілу часу. Це основний перехід для виконання завдань.

Перехід `create_io` створює запити введення-виведення для завдань, які були оброблені. Його призначення полягає в інтеграції із підсистемою дисків.

Перехід `take_up_disks` захоплює доступний диск для обробки запиту. Він контролює завантаження дисків.

Перехід `place_disk` імітує позиціонування головки диска перед операцією. Його призначення - затримувати виконання запиту для моделювання реального часу доступу.



Перехід `io_channel_transfer` імітує передачу даних через канал введення-виведення. Цей перехід завершує обробку запитів введення-виведення.

### 3.2.6.2 Опис TaskObject

Об'єкт `TaskObject` представляє завдання з конкретним обсягом пам'яті (від 20 до 60 сторінок) та визначеним життєвим циклом у системі. Цей об'єкт містить низку важливих полів, що керують його функціонуванням у системі.

Поле `generate` є генератором завдання цього типу. Поле `task` являє собою позицію, що зберігає завдання. `Try_allocate` є переходом для перевірки доступності пам'яті, тоді як поле `allocated` - це позиція, яка фіксує завдання, що отримали пам'ять.

`Fail_allocate` є переходом, що фіксує невдалі спроби виділення пам'яті. Позиція `fail_allocate_token` призначена для невдалих завдань, які очікують повторного виділення пам'яті. `Wait_allocate` представляє собою перехід для повторного виділення пам'яті.

`Process` є переходом, що імітує обробку завдання, а `create_io` - це перехід, який створює запити введення-виведення. `Take_up_disks` виступає переходом для захоплення дисків завданням.

Поле `busy_disk` є позицією, яка показує, що диск зайнятий завданням. `Place_disk` представляє перехід, що імітує позиціонування диска, а `disk_placed` - це позиція, яка фіксує успішне позиціонування диска.

`Io_channel_transfer` є переходом, що забезпечує передачу даних, а `finish` - це позиція, яка зберігає завершені завдання.

Об'єкти `TaskObject` використовуються для відстеження стану кожного типу завдань окремо, включаючи їх генерацію, обробку, введення-виведення та завершення. Завдяки цьому можна проводити детальний аналіз роботи системи та оптимізувати її параметри.

### 3.2.6.3 Опис методів

Метод `create_task_generator` створює генератор завдань, який додає нові завдання до системи відповідно до розподілу Пуассона. Він приймає параметри

d\_P (список позицій), d\_T (список переходів), d\_In (список вхідних дуг) та d\_Out (список вихідних дуг). В результаті повертає позицію generated\_task, яка накопичує створені завдання.

Метод create\_processors створює позицію processors, яка відображає кількість доступних процесорів. Він приймає єдиний параметр d\_P (список позицій) та повертає позицію processors.

Метод create\_pages створює позицію pages, що відображає кількість доступних сторінок пам'яті. Приймає параметр d\_P (список позицій) та повертає позицію pages.

Метод create\_free\_disks створює позицію free\_disks, яка відповідає кількості доступних дисків. Приймає параметр d\_P (список позицій) та повертає позицію free\_disks.

Метод create\_io\_request\_generator створює генератор запитів введення-виведення з рівномірним розподілом часу. Він приймає параметри d\_P (список позицій), d\_T (список переходів), d\_In (список вхідних дуг) та d\_Out (список вихідних дуг). В результаті повертає позицію generated\_io\_request.

Метод create\_interrupt\_generator створює генератор переривань із експоненційним розподілом часу. Він приймає параметри d\_P (список позицій), d\_T (список переходів), d\_In (список вхідних дуг) та d\_Out (список вихідних дуг). В результаті повертає позицію generated\_interrupt.

Конструктор CourseWorkNet є основним конструктором класу, який створює та ініціалізує всі компоненти мережі Петрі. Він створює генератори завдань, запитів введення-виведення та переривань, позиції для обробки завдань, пам'яті, процесорів, дисків, а також переходи для роботи з ресурсами. Додаткова логіка включає генерацію завдань з різними параметрами (обсяг пам'яті від 20 до 60 сторінок), розрахунок ймовірностей для кожного типу завдання та додавання дуг між позиціями та переходами.

Метод generate\_task\_objects створює об'єкти завдань TaskObject для кожного обсягу пам'яті між pages\_start та pages\_end із заданою ймовірністю. Він приймає параметри pages\_start (мінімальний обсяг пам'яті завдання), pages\_end

(максимальний обсяг пам'яті завдання) та probability (ймовірність створення завдання кожного типу). В результаті додає об'єкти TaskObject до списку taskObjects.

### 3.3 Верифікація та обчислення вихідних характеристик

Код у методі main реалізує симуляцію роботи моделі мережі Петрі та обчислює ключові характеристики продуктивності системи. Нижче наведено пояснення кожного блоку обчислень та результати верифікації у таблиці 3.1.

Таблиця 3.1: Результати верифікації

Прогін	Кількість сторінок	Кількість процесорів	Кількість дисків	Початок сторінок	Кінець сторінок	Час в системі	Час очікування	Завантаження дисків	Завантаження каналу передачі	Завантаження процесорів	Зайняті сторінки	Кількість завдань в очікуванні
1	131	2	4	20	60	165.73	154.70	0.00	0.00	1.15	115.55	449.37
2	300	5	4	10	30	187.20	170.42	0.01	0.00	1.71	288.50	181.56
3	400	7	4	20	40	173.70	138.57	0.01	0.00	1.71	383.30	177.11
4	500	10	10	20	40	196.82	154.61	0.01	0.00	1.72	481.15	182.68
5	600	10	4	20	40	210.72	173.74	0.01	0.00	1.72	578.12	180.16
6	600	10	4	20	45	193.00	153.83	0.01	0.00	1.72	578.35	179.65
7	700	12	10	20	30	249.09	197.06	0.01	0.00	1.73	671.85	159.14
8	400	10	10	10	40	190.84	157.79	0.01	0.00	1.71	385.04	183.17
9	900	30	30	10	40	223.02	156.66	0.01	0.00	1.75	862.56	165.10
10	1000	100	100	20	60	199.28	139.64	0.01	0.00	1.74	964.89	172.37
11	2000	200	200	20	60	226.16	158.66	0.01	0.00	1.76	1,907.28	156.66
12	3000	300	300	20	60	228.52	134.04	0.01	0.00	1.77	2,795.18	138.02

На початку коду визначаються основні параметри експерименту. Створюється матриця paramMat, яка містить різні конфігурації системи - кожен рядок представляє окремий набір параметрів, включаючи кількість сторінок,

процесорів, дисків та інші значення. Також встановлюється кількість ітерацій (20) та час моделювання (10000).

---

```

1 final int iterations = 20;
2 final int[][] paramMat = {
3     { 131, 2, 4, 20, 60 }, // ...
4 };
5 final double timeModelling = 10000;
6 final String outputFileName = "experiment_results.csv";

```

---

Далі код створює файл CSV для запису результатів експерименту. У перший рядок записується заголовок з назвами колонок українською мовою, які включають різні метрики системи: номер прогону, параметри конфігурації та результати вимірювань.

---

```

1 try (FileWriter writer = new FileWriter(outputFileName)) {
2     writer.write("Прогін,Кількість сторінок,Кількість процесорів,Кількість дисків,Початок
    сторінок,Кінець сторінок,Час в системі,Час очікування,Завантаження
    дисків,Завантаження каналу передачі,Завантаження процесорів,Зайняті
    сторінки,Кількість завдань в очікуванні\n");

```

---

Основна частина коду представляє собою цикл по всім рядкам матриці параметрів. Для кожної конфігурації створюються атомні посилання для накопичення загального часу роботи дисків, каналу введення-виведення та процесорів. Також створюються списки для збору статистики про час перебування завдань у системі та час очікування.

---

```

1 for(int runIndex = 0; runIndex < paramMat.length; runIndex++) {
2     final int[] paramRow = paramMat[runIndex];
3     final int pagesNum = paramRow[0];
4
5     final AtomicReference<Double> totalPlaceDiskWorkTime = new
    AtomicReference<>(((double) 0);
6     final AtomicReference<Double> totalIoChannelWorkTime = new
    AtomicReference<>(((double) 0);
7     final AtomicReference<Double> totalProcessorsWorkTime = new
    AtomicReference<>(((double) 0);

```

---

---

```

8  final ArrayList<Double> timeInSystemList = new ArrayList<>();
9  final ArrayList<Double> waitAllocateTimeList = new ArrayList<>();
10 final ArrayList<ArrayList<Integer>> pagesMarks = new ArrayList<>();
11 final ArrayList<ArrayList<Integer>> totalWaitAllocateTaskMarks = new ArrayList<>();

```

---

Для кожної конфігурації виконується внутрішній цикл з визначеною кількістю ітерацій. В кожній ітерації створюється нова мережа Петрі (CourseWorkNet) з поточними параметрами та запускається її симуляція. Під час симуляції збираються дані про час перебування завдань у системі, час очікування, завантаження різних компонентів системи та використання пам'яті.

---

```

1  IntStream.rangeClosed(0, iterations).forEach(iteration -> {
2    final CourseWorkNet courseWorkNet;
3    try {
4      courseWorkNet = new CourseWorkNet(pagesNum, paramRow[1], paramRow[2],
paramRow[3], paramRow[4]);
5    } catch (ExceptionInvalidTimeDelay e) {
6      throw new RuntimeException(e);
7    }
8    final CourseWorkPetriSim sim = new CourseWorkPetriSim(courseWorkNet.net);
9    sim.go(timeModelling, false);
10   for (final CourseWorkNet.TaskObject taskObject : courseWorkNet.taskObjects) {
11     for (int i = 0; i < taskObject.io_channel_transfer.getOutMoments().size(); i++) {
12       final double timeInSystem = taskObject.io_channel_transfer.getOutMoments().get(i)
13         - taskObject.generate.getOutMoments().get(i);
14       timeInSystemList.add(timeInSystem);
15     }
16     for (int i = 0; i < taskObject.wait_allocate.getOutMoments().size(); i++) {
17       final double waitTime = taskObject.wait_allocate.getOutMoments().get(i)
18         - taskObject.fail_allocate.getOutMoments().get(i);
19       waitAllocateTimeList.add(waitTime);
20     }
21     totalPlaceDiskWorkTime.updateAndGet(v -> (double) (v +
taskObject.place_disk.getTotalTimeServ()));

```

---



---

```
7 writer.write(formattedRow);
```

---

Після завершення всіх обчислень виводиться повідомлення про успішне збереження результатів у файл.

---

```
1 }
```

```
2 }
```

```
3 System.out.println("Results have been saved to " + outputFileName);
```

---

## **4      ЕКСПЕРИМЕНТИ НА МОДЕЛИ**



## **5      ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ ТА ЕКСПЕРИМЕНТІВ**

## **ВИСНОВКИ**

### **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

- 1., Business Process Model and Notation,
2. І. В. Стеценко, Моделювання систем, 2011
- 3: , PetriObjModelPaint,