УДК 004.43::004.94 https://doi.org/10.15407/pp2022.02.013

А.Ю. Дифучин

ТРАНСЛЯТОР МОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ ПЕТРІ-ОБ'ЄКТНИХ МОДЕЛЕЙ

Високорівневі засоби програмування спрямовані на підвищення швидкості розробки складних програм за рахунок автоматизації рутинних дій програміста, зменшення кількості помилок при кодуванні та зменшення коду в цілому. Візуальне програмування передбачає кодування на основі візуального представлення завдання на виконання обчислень замість текстового. Запропонований транслятор мови візуального програмування Петрі-об'єктних моделей створений відповідно до визначеної формальної граматики мови та формалізму Петрі-об'єктної моделі. Він виконує перетворення візуального представлення моделі в обчислення алгоритму імітації. Перевагами розробленої мови є невеликий алфавіт символів, реалізація можливостей для тиражування об'єктів та зв'язків між ними, універсальність застосування для розробки моделей дискретно-подійних систем.

Ключові слова: транслятор, формальна граматика, стохастична мережа Петрі, алгоритм імітації, Петрі-об'єктна модель

Вступ

Візуальне програмування є напрямком розробки засобів програмного забезпечення, спрямованим на застосування візуальних об'єктів для опису завдання на виконання обчислень. Для того, щоб візуальне представлення стало візуальним програмуванням необхідно гарантувати однозначне перетворення візуального представлення в обчислення. Таке перетворення забезпечує транслятор мови.

Мережі Петрі є ключовим формалізмом для моделювання дискретно-подійних систем, який покриває широкий клас систем від автоматних до стохастичних [1]. У контексті інженерії програмного забезпечення формалізм мереж Петрі є важливим ще й тому, що є загальноприйнятим для розробки паралельних та розподілених обчислень відповідно до стандарту ISO/IEC 15909-1:2004 [2].

Найбільш відомим засобом моделювання мережами Петрі на сьогоднішній день є CPNTools [3]. CPNTools вирішує проблему великої кількості елементів, необхідних для моделювання складних систем, через розширення типів маркерів, які символізують стан системи, та через використання функціональної мови програмування CPN ML для опису стану системи [4]. Комбінування графічного представлення мережі Петрі з елементами функціональної мови програмування зробило представлення моделі складним для сприйняття, оскільки основний зміст функціональності моделі часто прихований за складними функціональними виразами.

Технологія Петрі-об'єктного моделювання розвивається з 2011 року, коли була опублікована праця [5], в якій були розроблені теоретичні засади цього методу моделювання. Перевагою методу є можливість тиражування елементів моделі з однаковою поведінкою та конструювання моделі з фрагментів. У процесі констурювання забезпечуються умови, коли побудована модель має опис стохастичною мережею Петрі, отриманою об'єднанням мереж Петрі фрагментів моделі. Цей доведений теоретично факт є важливим для забезпечення обчислюваності Петрі-об'єктної моделі. Тобто, на відміну від інших існуючих способів конструювання моделі з фрагментів мереж Петрі, зберігається представлення мережею Петрі всієї моделі, не виникають сторонні елементи у представленні моделі, а також необхідність переходити до багаторівневого представлення моделі.

Технологія застосовувалась для моделювання систем, що містять сотні об'єктів. Вона показала достатньо високу швидкодію як у процесі розробки моделі, так і в процесі експериментування з нею. Проте використання виключно редактора мережі Петрі у програмному забезпечен-

ні виявило, що необхідний потужний візуальний інструментарій для конструювання Петрі-об'єктної моделі, оскільки зв'язування Петрі-об'єктів текстовою мовою програмування потребувало значних зусиль, концентрації уваги і часу на налагодження.

Формальна граматика мови Петрі-об'єктного моделювання розроблена у вигляді правил виведення в праці [7] і є контекстно-вільною (або граматикою типу 2 за ієрархією Хомського), однозначною і приведеною.

У даному науковому дослідженні представлена розробка транслятора мови візуального програмування Петрі-об'єктних моделей, наведений опис семантики граматичних виразів мови та визначений спосіб їх перетворення в обчислення.

1. Конструювання Петрі-об'єктної моделі

Петрі-об'єктні моделі застосовують для конструювання моделей дискретно-подійних систем. Під час побудови складної системи, її розбивають на структурні частини, що взаємодіють між собою, розробляють фрагменти моделі і поступово з'єднують їх.

Кожній елементарній події, що відбувається в системі, ставиться у відповідність перехід мережі Петрі. Умови для виникнення події символізують маркери в позиціях. Петрі-об'єктна модель складається з конструктивних елементів Петрі-об'єктів. Кожний Петрі-об'єкт є об'єктом суперкласу, що відтворює функціонування Петрі-об'єкта відповідно до заданої в об'єкті мережі Петрі. Оскільки алгоритм імітації стохастичної мережі Петрі є універсальним, то побудована модель одразу може запускатись на імітацію та виконання експериментальних досліджень. А зусилля, витрачені на побудову моделі, будуть компенсовані зменшенням витрат на написання та налагодження алгоритму імітації.

Детально поняття Петрі-об'єктної моделі викладено у публікації [8]. Петрі-об'єктом є об'єкт-нащадок суперкласу, що містить мережу Петрі та методи для відтворення функціональності об'єкта відповідно до заданої мережі. Зв'язки між Петрі-об'єктами задаються парами ототожнюваних позицій. Усі пари ототожнюваних позицій двох Петрі-об'єктів утворюють конектор, що з'єднує їх.

У візуальному представленні моделі семантичне значення мають триплети елементів Петрі-об'єкт – конектор – Петрі-об'єкт, які повністю визначають Петрі-об'єктну модель:

$$m = \{(o_i, c, o_j) |\, o_i, o_j \in O, i \neq j, \quad c = \{(o_i.p_a, o_j.p_b)\}\},$$

де o_i – j-ий Петрі-об'єкт, c – конектор відповідних Петрі-об'єктів.

Обчислюваність Петрі-об'єктної моделі випливає з того факту, що її функціонування визначається мережею Петрі, отриманою об'єднанням мереж Петрі, з яких вона складена:

$$m.net = \bigcup_j o_j.net,$$

де m – модель, —◇ – символ операції агрегування, $m.net$ – мережа Петрі моделі, $o_j.net$ – мережа Петрі-об'єкта o_j.

2. Формальна граматика мови Петрі-об'єктного моделювання

Для текстових мов програмування теорія формальних мов розроблена достатньо детально [9, 10]. У застосуванні цієї теорії до візуальних мов основним проблемним питанням є спосіб визначення відношення попередній-наступний елемент у граматичному виразі. Іншим проблемним питанням граматики візуальної мови програмування є визначення слова як лексичної одиниці виведення граматичного виразу.

У випадку графового представлення дискретно-подійної моделі відношення попередній-наступний логічно визначається відповідно до розповсюдження подій. Основною змістовою конструкцією, що формується з візуального представлення моделі, є триплети взаємопов'язаних елементів: позиція-дуга-перехід, перехід-дуга-позиція, об'єкт-конектор-об'єкт, об'єкт-конектор-група об'єктів. Тому лексичною одиницею (словом) обрано триплети елементів.

У публікації [7] визначена формальна граматика мови. Алфавіт термінальних символів граматики визначений такими символами (вказані короткі позначення символів):

p – позиція мережі Петрі;

t – перехід мережі Петрі;

a – дуга мережі Петрі;

o – Петрі-об'єкт;

g – група Петрі-об'єктів;

f – група колекцій Петрі-об'єктів;

s – відкрита для з'єднання позиція Петрі-об'єкта;

l – ототожнювач позицій Петрі-об'єктів.

Алфавіт нетермінальних символів:

E – триплет термінальних символів *pat*, що задає одну вхідну дугу переходу;

Q - триплет термінальних символів *tap*, що задає одну вихідну дугу переходу;

N – комбінація кількох нетермінальних символів *E* та *Q*, що може бути інтерпретована в мережу Петрі;

W – комбінація символів, що визначає пару з'єднаних Петрі-об'єктів;

Y – комбінація символів, що визначає з'єднання Петрі-об'єкта з групою Петрі-об'єктів;

X – комбінація символів, що визначає з'єднання Петрі-об'єкта з групою колекцій;

C – конектор Петрі-об'єктів, що складається з кількох ототожнювачів позицій.

Для кожного символа граматики визначене відповідне візуальне його представлення [7].

Правила виведення граматики мови:

(1) $I \longrightarrow NI \mid N$,
(2) $N \longrightarrow EQ \mid QE$,
(3) $N \longrightarrow EN \mid QN$,
(4) $E \longrightarrow pat$,
(5) $Q \longrightarrow tap$,
(6) $N \longrightarrow p \mid pN$,
(7) $I \longrightarrow WI|YI|XI$,
(8) $Y \longrightarrow oCg \mid g$,
(9) $X \longrightarrow oCf \mid f$,
(10) $W \longrightarrow oCo \mid o$,
(11) $I \longrightarrow oI|gI$,
(12) $C \longrightarrow slsC|sls$.

На рисунку 1 наведений приклад візуального представлення моделі, в результаті розбору якого як граматичного виразу отримуємо:

$$I \longrightarrow YI \longrightarrow oCgI \longrightarrow oslsgI \longrightarrow oslsgNI \longrightarrow$$
$$\longrightarrow oslsgNI \longrightarrow oslsgEQI \longrightarrow oslsgEQQI$$
$$\longrightarrow oslsgpatQQI \longrightarrow oslsgpattapQI \longrightarrow$$
$$\longrightarrow ooslsgpattaptapNI \longrightarrow \cdots$$
$$\longrightarrow ooslsgpattaptapEEQQEEQQ \longrightarrow \cdots$$
$$oslsgpattaptaptaptappatpattaptappatpat$$

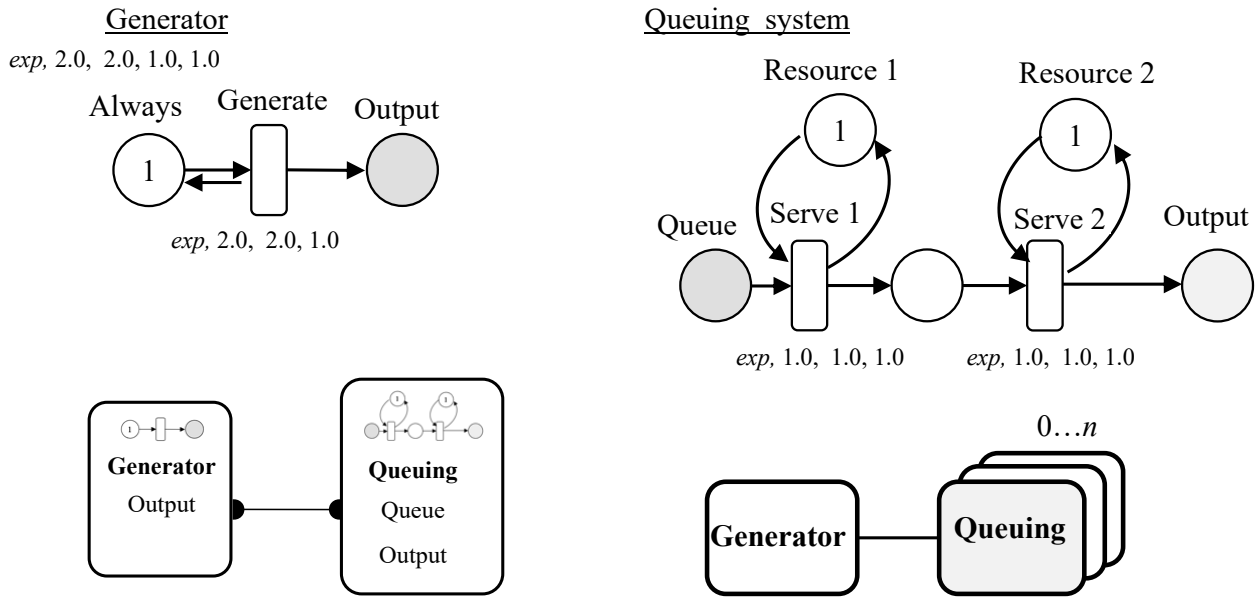


Рис. 1. Візуальне представлення Петрі-об'єктної моделі масового обслуговування: мережі Петрі-об'єктів, ототожнення позицій Петрі-об'єктів, триплет об'єкт-конектор-група.

Тобто модель складається з триплета Петрі-об'єкт – конектор – група Петрі-об'єктів. Мережа Петрі об'єкта Generator складається з одного триплета позиція – дуга – перехід та одного триплета перехід– дуга – позиція. Мережа Петрі об'єкта Queuing з групи складається з 4 триплетів позиція – дуга – перехід та 4 триплетів перехід– дуга – позиція.

3. Транслятор мови

Мова візуального програмування Петрі-об'єктних моделей реалізована у вигляді веб-застосування. Сучасні веб технології надають гнучкості у реалізації візуального редактора Петрі-об'єктних моделей, а віддалене виконання процесу імітації позбавляє залежності від обмеженого локального ресурсу користувача. Візуальний редактор мови програмування Петрі-об'єктних моделей реалізований у клієнтському застосуванні і дозволяє будувати візуальне представлення імітаційної моделі. Перетворення візуального представлення моделі у текстову інтерпретацію та запуск обчислень алгоритму імітації виконується серверним застосуванням. Для цієї мети розроблено транслятор мови.

Транслятор мови програмування включає в себе три послідовні фази: лексичний аналіз, синтаксичний аналіз та семантичний аналіз. Лексичний аналіз візуального представлення моделі - це виявлення у візуальному представленні лексем, тобто наборів візуальних символів. Елементами лексики є візуально представлені символи алфавіту мови: позиція, перехід, дуга, Петрі-об'єкт, група Петрі-об'єктів, відкрита для з'єднання позиція, ототожнювач позицій, конектор. Результат класифікації лексем є вхідною інформацією для синтаксичного аналізатора. Задачею синтаксичного аналізу є побудова синтаксичного дерева, яке являє собою розташування всіх синтаксичних елементів візуального представлення моделі відповідно до правил виведення граматики. На цьому етапі елементи лексики візуальної мови інтерпретуються як трійки термінальних елементів, що визначають зв'язки між елементами мережі Петрі, а також зв'язки між Петрі-об'єктами.

Семантичний аналіз та генерація коду є заключною фазою трансляції, що інтерпретує смисловий зміст та правила виконання конструкцій виділених синтаксичним аналізатором. Оскільки візуальний редактор являє собою клієнтську частину веб застосування, а виконання процесу імітації відбувається на сервері, транслятор виконує перетворення візуального представлення моделі у текстовий формат JSON. Формат передачі даних між клієнтським застосуванням і серверним застосуванням JSON є загальноприйнятим і являє собою текст, що містить структуру даних «ключ – значення». Процес трансляції починається вже під час побудови візуального представлення моделі. В момент створення користувачем зображення елемента мережі Петрі або Петрі-об'єкта, створюється його текстова інтерпретація. В момент з'єднання візуальних елементів між собою будується синтаксичне дерево, що містить трійки елементів. У таблиці 1 наведена структура вихідного JSON мовою TypeScript. Вихідний JSON є результатом роботи транслятора на клієнтській частині та містить інтерпретацію символів алфавіту мови. Під час конструювання моделі формується кінцева текстова інтерпретація моделі у форматі JSON, що відповідає синтаксичному дереву і відправляється на сервер.

Наступний етап трансляції відбувається у серверному застосуванні. Отримавши текстову інтерпретацію моделі серверне застосування здійснює обхід синтаксичного дерева та створення відповідних об'єктів, що передаються в конструктор класу `PetriObjModel` бібліотеки Петрі-об'єктного моделювання [8]. Фінальний етап включає в себе запуск процесу імітації викликом методу `go(t)` класу `PetriObjModel`, аргумент `t` якого вказує на час виконання імітації.

У конекторі важливим є порядок, в якому вказуються з'єднувані об'єкти, оскільки програмно з'єднання реалізується суміщенням адрес пам'яті відповідних позицій об'єктів. Якщо у конструюванні мо-

Таблиця 1

Текстова інтерпретація символів алфавіту мови візуального програмування Петрі-об'єктних моделей мовою TypeScript

Графічне представлення символу	Скорочене позначення символу	Інтерпретація мовою TypeScript

Name
k
	p	`type p = {`
		`  id: number`
		`  name: string`
		`  mark: number`
		`}`

Name
d, r, v, f
	t	`type t = {`
		`  id: number`
		`  name: string`
		`  distribution: string`
		`  delay: number`
		`  deviation: number`
		`  probability: number`
		`  priority: number`
		`}`

q	*a*	`type a = {`
		`  id: number`
		`  name: string`
		`  quantity: number`
		`  placeId: number`
		`  transitionId: number`
		`  info: boolean`
		`}`

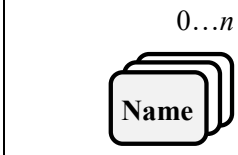
1 — 2	*E*	`type E = [p, a, t]`
1	*Q*	`type Q = [t, a, p]`

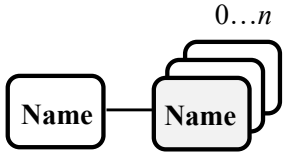
Name
k
	N	`type N = {`
		`  id: number`
		`  name: string`
		`  listE: Array<E>`
		`  listQ: Array<Q>`
		`  sharedPlaces: Array<number>`
		`}`

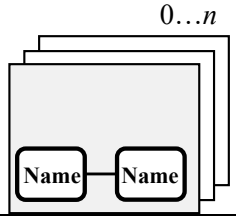
Name
placeName
placeName
placeName
	o	`type o = {`
		`  id: number`
		`  name: string`
		`  petriNetId: number`
		`}`

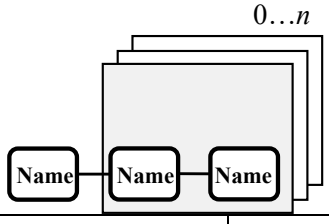
placeName	*s*	`type s = {`
		`  petriObjectId: number`
		`  placeId: number`
		`}`

pName — pName	*sls*	`type sls = [s, s]`
	C	`type C = Array<sls>`
Name — **Name**	*W*	`type W = [o, C, o]`

	*g*	`type g = {`
		`  id: number,`
		`  name: string`
		`  petriObj: o`
		`  quantity: number`
		`}`

	*Y*	`type Y = [o, C, g]`

	*f*	`type f = {`
		`  id: number`
		`  name: string`
		`  listW: Array<W>`
		`  quantity: number`
		`}`

	*X*	`type X = [o, C, f]`

-	*I*	`type model = {`
		`  id: number`
		`  name: string`
		`  time: number`
		`  listObj: Array<o>`
		`  listW: Array<W>`
		`  listY: Array<Y>`
		`  listN: Array<N>`
		`}`

делі використано з'єднання `Link(obj_a, P_a, obj_b, P_b)`, то позиція `P_a` об'єкту `obj_a` буде ототожнена позицією `P_b` об'єкта `obj_b` . Якщо наступним кроком виконано з'єднання `Link(obj_a, P_a, obj_c, P_c)`, то позиція `P_a` об'єкта `obj_a` буде ототожнена позицією `P_c` об'єкта `obj_c`. Тепер адреса позиції `P_a` збігається з позицією `P_c` і попередньо виконане з'єднання з `P_b` втрачено. Щоб організувати з'єднання усіх трьох позицій в одну, потрібно виконати ототожнення позиції `P_b` з `P_a` і в наступному кроці позиції `P_c` з `P_a`. В такому разі всі три позиції вказуватимуть на позицію `P_a` (рис. 1). Саме тому в процесі приєднання до групи в конекторі першим вказується об'єкт групи, а другим – об'єкт, з яким з'єднуються усі об'єкти групи. У конструкторі групи конектор тиражується для всіх об'єктів групи і всі об'єкти будуть отримувати інформацію від одного об'єкта. Отримуємо зв'язок типу one-to-many.

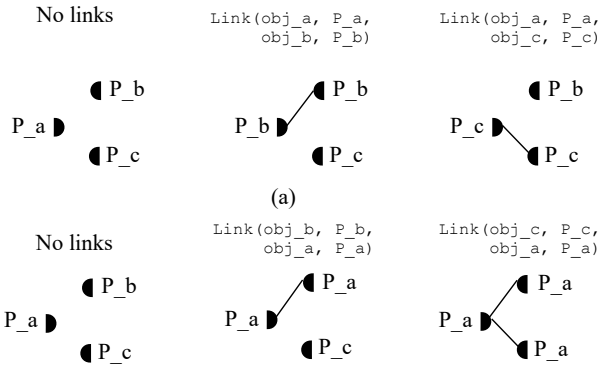


Рис. 2. Встановлення зв'язків між позиці-ями Петрі-об'єктів:
а) послідовність, яка призводить до знищення попереднього зв'язку,
б) послідовність, яка призводить до зв'язку one-to-many

Перетворення Петрі-об'єктної моделі в модель обчислень відбувається з використанням алгоритму імітації на останньому етапі. Оскільки функціонування моделі визначається мережею Петрі, отриманою об'єднанням мереж Петрі усіх об'єктів моделі, то алгоритм імітації виконує обчислення за тими ж правилами, що і алгоритм імітації стохастичної мережі Петрі з часовими затримками з багатоканальними переходами. Проте є суттєвий виграш у складності обчислень через те, що пошук найближчої події здійснюється не по всій мережі, а по моментам найближчої події Петрі-об'єктів. По-друге, через розповсюдження подій в межах одного Петрі-об'єкта (доки не буде досягнута позиція-ототожнювач) перевірка умов виконання події та здійснення події відбувається переглядом тільки елементів одного Петрі-об'єкта замість перегляду всієї мережі Петрі [6].

Імітація моделі здійснюється виконанням таких дій [11]:

- виконати вхід маркерів в переходи для всіх об'єктів;
- доки не вичерпаний час моделювання:
  - визначити момент найближчої події;
  - просунути час у момент найближчої події;
  - змінити стан моделі у відповідності до події.

Зміна стану моделі складається з таких дій:

- виконати вихід маркерів з усіх каналів переходу, для яких час виходу з каналу співпадає з поточним моментом часу;
- визначити переходи, для яких виконана умова входу маркерів, і, за необхідності, розв'язати конфлікт переходів;
- виконати вихід маркерів з обраного на попередньому кроці переходу;
- виконати вхід маркерів в переходи для усіх об'єктів.

Отже, перетворення відбувається через послідовні кроки: 1) транслятор виконує аналіз візуальних символів, які додаються, і визначає лексеми, які утворюються, 2) правильно утворені граматичні вирази (тобто такі, що відповідають визначеним правилам граматики) зберігаються, 3) інтерпретація набору виявлених лексем та формування набору даних моделі у форматі JSON, 4) перетворення моделі з формату JSON у об'єкт класу `PetriObjModel`, 5) запуск обчислень викликом методу `go(t)` класу `PetriObjModel`, аргумент `t` якого вказує на час виконання імітації.

4. Приклад розробки та запуску моделі на обчислення

Через обмежений обсяг матеріалу, який можна розмістити у статті, перетворення візуального представлення моделі у її обчислення наведемо на простому прикладі моделі масового обслуговування (рис. 1). Зауважимо, що завдяки використанню групи об'єктів побудована модель в залежно від параметру *n* відтворює функціонування системи з різною кількістю обробників замовлень, які надходять на обслуговування з генератора - 1. Під час використання звичайних блочних редакторів, які широко використовують у симуляторах, довелось би з'єднувати вручну у візуальному редакторі усі *n* об'єкти. Очевидно, що при великих n це потребує значних зусиль. Окрім того, перегляд усіх *n* зв'язків з метою їх перевірки чи корегування також потребуватиме значних зусиль.

Результат перетворення транслятором візуального представлення моделі у JSON наведений на рисунку 3. Результат перетворення JSON представлення моделі у java код та запуск моделі на імітацію представлений на рисунку 4.

При запуску імітації можемо пересвідчитись у правильності отриманих результатів і переконатись, що час за умови збільшення складності моделі збільшується поліноміально. Оскільки модель допускає теоретичний розрахунок, то за результатом порівняння отриманих результатів імітації з теоретичними робимо висновок про коректність виконаних перетворень. Помилка при часі моделювання 1000000 не перевищувала 5%.

```
Model = {
  "id": 1,
  "name": "SMO",
  "listNet": [{
    "id": 1,
    "name": "Generator",
    "listE": [[{
      "name": "Always",
      "mark": 1,
    }, {
      "quantity": 1,
      "placeId": 1,
      "transitionId": 1
    }, {
      "name": "Generate",
      "distribution": "exp",
      "delay": 2,
      "probability": 1
    }]],
    "listQ": [[{
      "name": "Generate",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "quantity": 1,
      "placeId": 2,
      "transitionId": 1
    }, {
      "name": "Always",
      "mark": 1,
    }], [{
      "name": "Generate",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "quantity": 1,
      "placeId": 2,
      "transitionId": 1
    }, {
      "name": "Output",
      "mark": 0,
    }]]
  }, {
    "name": "Queuing",
    "listE": [[{
      "id": 1,
      "name": "Queue",
      "mark": 0,
    }, {
      "id": 1,
      "quantity": 1,
      "placeId": 1,
      "transitionId": 1
    }, {
      "id": 1,
      "name": "Serve 1",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }], [{
      "id": 2,
      "mark": 0,
    }, {
      "id": 2,
      "quantity": 1,
      "placeId": 2,
      "transitionId": 2
    }, {
      "id": 2,
      "name": "Serve 2",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }], [{
      "id": 3,
      "name": "Resource 1",
      "mark": 1,
    }, {
      "id": 3,
      "quantity": 1,
      "placeId": 3,
      "transitionId": 1
    }, {
      "id": 1,
      "name": "Serve 1",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }], [{
      "id": 4,
      "name": "Resource 2",
      "mark": 1,
    }, {
      "id": 4,
      "quantity": 1,
      "placeId": 4,
      "transitionId": 2
    }, {
      "id": 2,
      "name": "Serve 2",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }]],
    "listQ": [[{
      "id": 1,
      "name": "Serve 1",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "id": 5,
      "quantity": 1,
      "placeId": 2,
      "transitionId": 1
    }, {
      "id": 2,
      "name": "Unnamed",
      "mark": 0,
    }], [{
      "id": 1,
      "name": "Serve 1",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "id": 6,
      "quantity": 1,
      "placeId": 3,
      "transitionId": 1
    }, {
      "id": 3,
      "name": "Resource 1",
      "mark": 0,
    }], [{
      "id": 2,
      "name": "Serve 2",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "id": 6,
      "quantity": 1,
      "placeId": 4,
      "transitionId": 2
    }, {
      "id": 4,
      "name": "Resource 2",
      "mark": 0,
    }], [{
      "id": 2,
      "name": "Serve 2",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "id": 7,
      "quantity": 1,
      "placeId": 5,
      "transitionId": 2
    }, {
      "id": 5,
      "name": "Output",
      "mark": 0,
    }]]
  }],
  "listObj": [{
    "id": 1,
    "name": "Generator",
    "petriNetId": 1,
  }, {
    "id": 1,
    "name": "Queue",
    "petriNetId": 2,
  }],
  "listW": [],
  "listY": [{
    "id": 1,
    "name": "Generator",
    "petriNetId": 1,
  }, [{
    "petriObjId": 1,
    "placeId": 2
  }, {
    "petriObjId": 2,
    "placeId": 1
  }], {
    "id": 1,
    "name": "Queues",
    "petriObj": {
      "id": 2,
      "name": "Queue",
      "petriNetId": 2,
    },
    "quantity": 100
  }]
}
```


Рис. 3. Результат перетворення транслятором візуального представлення моделі у тексто-вий формат JSON.

```
ArrayList<P> listP = new ArrayList<>();
ArrayList<T> listT = new ArrayList<>();
ArrayList<E> listE = new ArrayList<>();
ArrayList<Q> listQ = new ArrayList<>();
listP.add(new P("Always",1));
listP.add(new P("Queue",0));
listT.add(new T("1", "Generate", "exp", 2.0, 1.0, 1.0, 1));
listE.add(new E(listP.get(0),listT.get(0),new Arc(1)));
listQ.add(new Q(listT.get(0),listP.get(1),new Arc(1)));
listQ.add(new Q(listT.get(0),listP.get(0),new Arc(1)));
PetriSim generator = new PetriSim(new Net("Generator",listP,listE,listQ));

listP = new ArrayList<>();
listT = new ArrayList<>();
listE = new ArrayList<>();
listQ = new ArrayList<>();
listP.add(new P("Queue", 0));
listP.add(new P("Resource 1", 1));
listP.add(new P("", 0));
listT.add(new T("1","Serve 1", "exp", 1.0, 1.0, 1.0, 1));
listE.add(new E(listP.get(0), listT.get(0), new Arc(1)));
listE.add(new E(listP.get(1), listT.get(0), new Arc(1)));
listQ.add(new Q(listT.get(0), listP.get(1), new Arc(1)));
listQ.add(new Q(listT.get(0), listP.get(2), new Arc(1)));
listP.add(new P("", 1));
listP.add(new P("Resource 2", 0));
listT.add(new T("2","Serve 2","exp", 1.0, 1.0, 1.0, 1));
listE.add(new E(listP.get(2), listT.get(1), new Arc(1)));
listE.add(new E(listP.get(3), listT.get(1), new Arc(1)));
listQ.add(new Q(listT.get(1), listP.get(3), new Arc(1)));
listQ.add(new Q(listT.get(1), listP.get(4), new Arc(1)));
PetriSim smo = new PetriSim(new Net("Queueing", listP, listE, listQ));

Group group = new Group(smo, numObjs);
Connector connector = new Connector(new Link(smo, 0, generator, 1));
ArrayList<PetriSim> listO = new ArrayList<>();
ArrayList<W> listW = new ArrayList<>();
ArrayList<Y> listY = new ArrayList<>();
listY.add(new Y(group, generator,  connector));
PetriObjModel model = new PetriObjModel("SMO",listO, listW, listY);
model.go(timeModeling);
model.printResultsForAllObj();
```

Рис. 4. Результат перетворення транслятором текстового формату JSON у java код.

Висновки

Розроблений транслятор мови візуального програмування Петрі-об'єктних моделей, який виконує перетворення візуального представлення Петрі-об'єктної моделі у текстову мову програмування та запускає на обчислення. Лексичний аналіз виконується під час створення візуального представлення моделі у клієнтському застосуванні. Семантичний аналіз і виконання обчислень моделі виконується серверним застосуванням.

Поняття Петрі-об'єктної моделі для тиражування зв'язків розширено поняттями конектор Петрі-об'єктів, група Петрі-об'єктів, колекція Петрі-об'єктів.

Наведено приклад, в якому представлено усі етапи перетворення віуального представлення моделі транслятором мови програмування. Порівняння результатів імітації з теоретичними доводить коректність виконаних перетворень.

References

1. Petri nets : fundamental models, verification, and applications. Michel Diaz (ed.) Willey, 2009. 581 p.
2. ISO/IEC 15909-1:2004 Systems and software engineering — High-level Petri nets — Part 1: Concepts, definitions and graphical notation. [Online] – Available from: https://www.iso.org/standard/38225.html, last accessed 2022/07/2.
3. CPNTools [Online] – Available from: https://cpntools.org/, last accessed 2022/07/2.
4. Jensen, K., Kristensen, L.M. (2009). CPN ML Programming. In: Coloured Petri Nets. Springer, Berlin, Heidelberg. https://doi.org/10.1007/b95112_3.
5. Стеценко И.В. (2011) Теоретические основы Петри-объектного моделирования систем. Математичні машини і системи, 136-148.
6. Stetsenko I. V., Dorosh V. I., Dyfuchyn A. Petri-object simulation: Software package and complexity. 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015, pp. 381-385. (Scopus) https://doi.org/ 10.1109/ IDAACS.2015.7340762.
7. Дифучин А.Ю., Стеценко І.В., Жаріков Е.В. Граматика мови візуального програмування Петрі-об'єктних моделей // Проблеми програмування. – Київ, 2021. - №4. – С.82-94. https://doi.org/10.15407pp2021.04.082
8. Stetsenko, I.V., Dyfuchyn, A.: Petri-object Simulation: Technique and Software. Information, Computing and Intelligent Systems 1, 51-59 (2020). https://doi.org/10.20535/2708-4930.1.2020.216057
9. Kimball J. P. The Formal Theory of Grammar Prentice-Hall, 1973 - 127 p.
10. Becerra-Bonache L., Bel-Enguix G., Jiménez-López M. D., Martín-Vide C. (2018). Mathematical Foundations: Formal Grammars and Languages. In: The Oxford Handbook of Computational Linguistics, Second Edition (2nd edn) Ruslan Mitkov (ed.) [Online] – Available from: https://academic.oup.com/edited-volume/42643/chapter-abstract/358148992?redirectedFrom=fulltext, last accessed 2022/07/2.
11. Stetsenko I.V. (2017) Parallel Algorithm for Petri Object Simulation. Cybernetics and Systems Analysis. 53(4), 605–614.

Отримано 16.07.2022

***Про автора**:*

Дифучин Антон Юрійович,
аспірант 4 року навчання кафедри
інформатики та програмної інженерії
НТУУ "КПІ імені Ігоря Сікорського".
Кількість наукових публікацій
в українських виданнях – 3.
Кількість наукових публікацій
в іноземних виданнях – 3. Індекс Хірша
– 1. https://orcid.org/0000-0002-1722-8840

Місце роботи автора:

Національний технічний університет
України «Київський політехнічний
інститут імені Ігоря Сікорського»,
03056, м. Київ,
проспект Перемоги 37.
Тел.: (044) 236-9651
e-mail: difuchin@gmail.com