Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

ДИФУЧИН АНТОН ЮРІЙОВИЧ

УДК 004.43::004.94

ДИСЕРТАЦІЯ
МЕТОДИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ ПЕТРІ-ОБ'ЄКТНИХ МОДЕЛЕЙ

121 Інженерія програмного забезпечення
12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело_____А.Ю. Дифучин

Науковий керівник Жаріков Едуард В’ячеславович, д.т.н., доцент

Київ – 2022

АНОТАЦІЯ

Дифучин А.Ю. Методи візуального програмування Петрі-об’єктних моделей. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 121 – Інженерія програмного забезпечення з галузі знань 12 – Інформаційні технології. – Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2022.

Дисертаційна робота присвячена розробці методів та засобів візуального програмування моделей дискретно-подійних систем, формалізованих у вигляді Петрі-об’єктної моделі. Візуальне програмування є способом розробки програм, що використовує візуальні об’єкти для опису завдання на виконання обчислень. Для того, щоб візуальне представлення стало візуальним програмуванням необхідно гарантувати однозначне перетворення візуального представлення у обчислення. Таку однозначність перетворення гарантує розробка формальної мови програмування. На сьогоднішній день положення теорії формальних мов сформульовані в припущенні, що мова є текстовою, і адаптації цих положень до мови програмування, в якій символами є елементи візуального представлення, є невирішеним завданням.

Особливе місце візуальні засоби представлення займають у програмних засобах з імітаційного моделювання, оскільки слугують не тільки для конструювання моделі, але й для її відлагодження та представлення кінцевого результату імітації. Існуючі програмні засоби з імітаційного моделювання широко застосовують графічні редактори для конструювання моделі з блоків або з більш низькорівневих елементів, наприклад, елементів мережі Петрі. Процес розробки моделі у графічному редакторі є трудомістким і довготривалим, а у випадку великої кількості елементів часто є неможливим, оскільки потребує великого обсягу рутинної роботи з елементами та налаштуванням їх параметрів.

Імітаційні моделі є одним з потужних інструментів обробки даних з метою пошуку оптимальних умов або короткострокового прогнозування. У поєднанні з

даними, що постачаються у реальному часі, вони можуть стати потужним компонентом сучасних систем прийняття рішень та управління. Тому розробка нових методів та програмних засобів, що спрямовані на підвищення ефективності процесу програмної реалізації моделі за рахунок автоматизації кодування, є актуальним науковим завданням.

Метою наукового дослідження є підвищення ефективності програмних засобів представлення складних моделей систем для цілей імітаційного моделювання за рахунок зменшення складності конструювання моделі та збільшення швидкодії алгоритмів імітації, зручності сприйняття моделі, зменшення кількості помилок при створенні зв'язків між елементами моделі та зменшення часу на модифікацію моделі.

У **першому розділі** наведено огляд існуючих програмних засобів моделювання систем стохастичними мережа Петрі та обґрунтовано розробку мови візуального програмування. Виявлено, що графічні редактори не вирішують низку проблем: неможливість коригування параметрів моделі без коригування її візуального представлення, неможливість тиражування однотипних елементів у великій кількості (оскільки усі вони будуть займати візуальний простір), неможливість тиражування зв'яків при повторному використанні фрагментів мережі Петрі, неможливість налаштування параметрів елементів без коригування кожного окремого параметра у спеціально відведеному для цього вікні.

У **другому розділі** наведені відомості з теорії стохастичних мереж Петрі та основні теоретичні положення Петрі-об'єктного моделювання. Формальний опис Петрі-об'єктної моделі розвинутий за рахунок введення понять конектора Петрі-об'єктів, групи Петрі-об'єктів, колекції Петрі-об'єктів. Введені поняття відкривають можливість тиражування зв'язків при конструюванні Петрі-об'єктної моделі. Проте відсутність візуального представлення моделі робить процес розробки моделі складним через рутинні та схильні до помилок операції кодування зв'язків між елементами.

У **третьому розділі** представлена формальна граматика мови Петрі-об’єктного моделювання, яка розроблена. Алфавіт мови складається з графічних елементів візуального представлення, а дозволені мовою набори графічних елементів утворюють лексеми мови. Синтаксис мови визначений правилами виведення граматики, в основі яких правила утворення триплетів елементів. Для розробленої контекстно-вільної граматики встановлено, що вона є однозначною і приведеною. Наведений приклад розробки моделі інформаційної системи розробленою мовою.

У **четвертому розділі** представлена розробка транслятора мови візуального програмування Петрі-об’єктних моделей. На клієнтській частині веб застосування реалізована частина транслятора, що відповідає за лексичний та синтаксичний аналіз мовного виразу. На серверній частині веб застосування реалізована частина транслятора, що виконує семантичний аналіз мовного виразу. Для передачі даних між клієнтом та сервером використовується JSON формат. У розділі представлена інтерпретація символів алфавіту мови візуального програмування Петрі-об’єктних моделей мовою TypeScript. Запуск на обчислення відбувається після перетворення отриманих у форматі JSON даних в об’єкт PetriObjModel бібліотеки Петрі-об’єктного моделювання PetriObjLIb. Обчислення Петрі-об’єктної моделі – це відтворення подій в часі алгоритмом імітації.

П’ятий розділ містить результати експериментального дослідження процесу розробки моделей, точності моделювання та швидкодії обчислення моделі.

Результати, отримані у дисертаційному дослідженні, містять **наукову новизну**. **Вперше** розроблено візуальну мову програмування Петрі-об’єктних моделей, яка дозволяє спростити процес побудови моделей, підвищити наочність сприйняття моделей та час виконання імітаційного моделювання шляхом організації дворівневого способу побудови моделей. На відміну від існуючих рішень в області імітаційного моделювання, візуальна мова програмування Петрі-об’єктних моделей надає універсальний та гнучкий інструмент для

побудови моделей дискретно-подійних систем, а процес виконання імітаційного моделювання не залежить від обчислювальних ресурсів користувача.

Удосконалено Петрі-об'єктну модель за рахунок введення поняття групи Петрі-об'єктів, колекції Петрі-об'єктів та групи колекцій Петрі-об'єктів, що, на відміну від існуючих засобів представлення імітаційної моделі, надають можливість тиражувати Петрі-об'єкти з заданими наборами параметрів, тиражувати колекції взаємопов'язаних Петрі-об'єктів та тиражувати зв'язки між Петрі-об'єктом та групою Петрі-об'єктів, між Петрі-об'єктом та групою колекцій Петрі-об'єктів. За рахунок тиражування однотипних фрагментів моделі та тиражування зв'язків створюються умови для швидкого конструювання моделей з великою кількістю елементів та значно скорочується обсяг їх візуального представлення.

Вперше запропоновано клієнт-серверну архітектуру серед програмного забезпечення з імітаційного моделювання на основі Петрі-об'єктного підходу, використання якої дозволяє задіяти ресурси віддаленого серверу для проведення імітаційного моделювання для забезпечення стабільного часу виконання, зменшення витрат на інфраструктуру обчислювальних ресурсів серед користувачів та організації спільного доступу до розробки моделей.

Результати дисертаційної роботи опубліковано у 7 наукових публікаціях, серед яких 1 стаття у періодичному науковому виданні, проіндексованому у Web of Science Core Collection та Scopus базах даних, 2 статті у фаховому науковому журналі категорії «Б», 1 стаття у фаховому науковому журналі категорії «В», 1 стаття у науковому журналі, проіндексованому GoogleScholar, 2 публікації у матеріалах міжнародних наукових конференцій, проіндексованих у Scopus.

Ключові слова: візуальне програмування, мова програмування, формальна граматика, імітаційне моделювання, стохастична мережа Петрі, Петрі-об'єкт.

Список публікацій здобувача

Наукові праці, в яких опубліковано основні наукові результати дисертації:

Стаття у періодичному науковому виданні, проіндексованому у базах даних Web of Science Core Collection та Scopus:

1. Stetsenko I.V., Dyfuchyn A. (2021) Petri-object Simulation Two Level Visual Programming Language. *Advances in Intelligent Systems and Computing*, 1265, 266-276. Springer, Cham. ISSN 2194-5357. https://doi.org/10.1007/978-3-030-58124-4_26 (Scopus)

Статті у наукових виданнях, включених на дату опублікування до переліку наукових фахових видань України:

2. Дифучин А.Ю. (2022) Транслятор мови візуального програмування Петрі-об'єктних моделей. *Проблеми програмування*, 2, 13-21. ISSN 1727-4907. https://doi.org/10.15407/pp2022.02.013 (Фахове видання, "Б")

3. Дифучин А.Ю., Стеценко І.В., Жаріков Е.В. (2021) Граматика мови візуального програмування Петрі-об'єктних моделей. *Проблеми програмування*, 4, 82-94. ISSN 1727-4907. https://doi.org/10.15407pp2021.04.082 (Фахове видання, "Б")

4. Дифучин А. Ю., Томашевський В. М. (2017) Веб-сервіс моделювання дискретно-подійних систем. *Вісник Національного технічного університету України «КПІ»*, 66, 32-36 (Фахове видання, "В")

Стаття у науковому журналі, проіндексованому GoogleScholar, Directory of Open Access Journals (DOAJ):

5. Stetsenko, I.V., Dyfuchyn, A.: Petri-object Simulation: Technique and Software. Information, Computing and Intelligent Systems 1, 51-59 (2020). ISSN 2708-4930 https://doi.org/10.20535/2708-4930.1.2020.216057

Публікації у матеріалах міжнародних наукових конференцій, проіндексованих у базах даних Web of Science Core Collection та Scopus:

6. Stetsenko I.V., Dyfuchyn A., Leshchenko K., Davies J. Web application for visual modeling of discrete event systems. 2017 Internet Technologies and Applications

(ITA), 2017, pp. 86-91. Wrexham, North Wales, UK - United Kingdom, 12-15 Sept. ISBN-9781509048168 https://doi.org/10.1109/ITECHA.2017.8101916 (Scopus)

7. 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. ISBN: 978-1-4673-8359-2. https://doi.org/10.1109/IDAACS.2015.7340762. (Scopus)

ABSTRACT

Dyfuchyn A. Methods of visual programming of Petri-object models. – Qualifying scientific work on the rights of the manuscript.

Thesis for the degree of Doctor of Philosophy in specialty 121 - Software engineering in the field of knowledge 12 - Information technologies. - National Technical University of Ukraine "Ihor Sikorsky Kyiv Polytechnic Institute", Kyiv, 2022.

Ph.D. thesis is devoted to the development of methods and tools for visual programming of models of discrete-event systems, formalized in the form of a Petri-object model. Visual programming is a way of developing programs that uses visual objects to describe a computational task. In order for the visual representation to become visual programming, it is necessary to guarantee an unambiguous transformation of the visual representation into computations. The development of a formal programming language guarantees such unambiguity of the transformation. Nowadays, the conditions of the theory of formal languages are formulated under the assumption that the language is textual. The adaptation of these conditions to the programming language, in which symbols are elements of visual representation, is an unsolved task.

Visual representation tools occupy a special place in software tools for simulation, as they serve not only to construct the model but also to debug it and present the final result of the simulation. Existing software tools for simulation widely use graphic editors to construct a model from blocks or from lower-level elements, for example, Petri net elements. The process of developing a model in a graphic editor is time-consuming, and in the case of a large number of elements, it is often impossible, as it requires a large amount of routine work with the elements and setting their parameters.

Simulation models are one of the powerful data processing tools for finding optimal conditions or short-term prediction. Combined with real-time data, they can become a powerful component of modern decision-making and management systems. Therefore, the development of new methods and software tools aimed at increasing the

efficiency of the process of software implementation of the model due to the automation of coding is an urgent scientific task.

The purpose of scientific research is to increase the effectiveness of software tools for presenting complex models of systems for the purposes of simulation by reducing the complexity of model construction and increasing the speed of simulation algorithms, ease of perception of the model, reducing the number of errors when creating connections between model elements and reducing the time for model modification.

The first section provides an overview of existing software tools for systems modeling by stochastic Petri net and substantiates the development of a visual programming language. It was found that graphic editors do not solve a number of problems: the impossibility of adjusting the parameters of the model without adjusting its visual representation, the impossibility of replicating the same type of elements in a large number (since they will all occupy the visual space), the impossibility of replicating connections when reusing fragments of the Petri net, the impossibility setting the parameters of the elements without adjusting each individual parameter in a specially dedicated window.

The second section provides information on the theory of stochastic Petri nets and the main theoretical principles of Petri-object modeling. The formal description of the Petri-object model is developed by introducing the concepts of a connector of Petri-objects, a group of Petri-objects, and a collection of Petri-objects. The introduced concepts open up the possibility of replicating connections when constructing a Petri object model. However, the lack of a visual representation of the model makes the process of model development difficult due to the routine and error-prone operations of coding the relationships between elements.

The third section presents the formal grammar of the Petri-object modeling language, which has been developed. The alphabet of a language consists of graphic elements of visual representation, and language-allowed sets of graphic elements form lexemes of a language. The syntax of the language is determined by the rules of grammar derivation, which are based on the rules of formation of elements triplets. For

the developed context-free grammar, it was established that it is unambiguous and reduced. An example of developing an information system model in a developed language is given.

The fourth section presents the development of a translator for the visual programming language of Petri-object models. The client side of the web application implements the part of the translator responsible for the lexical and syntactic analysis of the language expression. On the server side of the web application, a part of the translator, which performs semantic analysis of the language expression, is implemented. The JSON format is used for data transfer between the client and the server. The section presents the interpretation of the symbols of the alphabet of the visual programming language of Petri-object models in the TypeScript language. The computation is started after the data received in JSON format is converted into the PetriObjModel object of the Petri-object modeling library PetriObjLIb. Computing a Petri object model is the reproduction of events in time by a simulation algorithm.

The fifth section contains the results of an experimental study of the model development process, modeling accuracy, and model computation speed.

The results obtained in the research contain **scientific novelty**. **For the first time**, a visual programming language for Petri-object models was developed, which allows to simplify the process of building models, to increase the clarity of perception of models and the performance time of simulation by organizing a two-level method of constructing models. Unlike existing solutions in the field of simulation, the visual programming language of Petri-object models provides a universal and flexible tool for constructing models of discrete-event systems, and the process of performing simulation does not depend on the user's computing resources.

The Petri-object model has been **improved** due to the introduction of the concept of a group of Petri-objects, a collection of Petri-objects, and a group of collections of Petri-objects, which, unlike the existing tools of presenting a simulation model, provide the opportunity to replicate Petri-objects with given parameter sets, to replicate collections of interconnected Petri-objects and to replicate relationships between a Petri-object and a group of Petri-objects, between a Petri-object and a group of

collections of Petri-objects. Due to the replication of the same type of model fragments and the replication of connections, conditions for fast models construction with a large number of elements are created and the volume of their visual representation is significantly reduced.

For the first time, a client-server architecture is proposed among simulation software based on the Petri-object approach, the use of which allows you to use the resources of a remote server to conduct simulation to ensure stable execution time, reduce the cost of the infrastructure of computing resources among users and to organize shared access to the development of models.

The results of the dissertation were published in 7 scientific publications, including 1 paper in a periodical scientific publication indexed in the Web of Science Core Collection and Scopus databases, 2 papers in a professional scientific journal of category "B", 1 paper in a professional scientific journal of category "B ", 1 paper in a scientific journal indexed by GoogleScholar, 2 publications in the materials of international scientific conferences indexed in Scopus.

Keywords: visual programming, programming language, formal grammar, simulation, stochastic Petri net, Petri-object.

ЗМІСТ

ВСТУП

Актуальність теми. Візуальне програмування є способом розробки засобів програмного забезпечення, що спрямований на застосування візуальних об’єктів для опису завдання на виконання обчислень. Проте для того, щоб візуальне представлення стало візуальним програмуванням необхідно гарантувати однозначне перетворення візуального представлення у обчислення. Таку однозначність перетворення гарантує розробка формальної мови програмування. На сьогоднішній день положення теорії формальних мов сформульовані в припущенні, що мова є текстовою, і адаптації цих положень до мови програмування, в якій символами є елементи візуального представлення, є невирішеним завданням.

Особливе місце візуальні засоби представлення займають у програмних засобах з імітаційного моделювання, оскільки слугують не тільки для конструювання моделі, але й для її відлагодження та представлення кінцевого результату імітації. Існуючі програмні засоби з імітаційного моделювання широко застосовують графічні редактори для конструювання моделі з блоків або з більш низькорівневих елементів, наприклад, елементів мережі Петрі. Процес розробки моделі у графічному редакторі є трудомістким і довготривалим, а у випадку великої кількості елементів часто є неможливим, оскільки потребує великого обсягу рутинної роботи з елементами та налаштуванням їх параметрів. Побудувати модель з використанням універсальної мови програмування у випадку великої кількості однотипних елементів є більш реальною задачею на сьогоднішній день, ніж у середовищі моделювання. Проте такий процес буде теж довготривалим через складність відлагодження зв’язків між елементами моделі за відсутності наочного їх представлення.

Імітаційні моделі є одним з потужних інструментів обробки даних з метою пошуку оптимальних умов або короткострокового прогнозування. Імітація означає експериментування з моделлю системи замість реальної системи з метою поліпшення характеристик функціонування системи (підвищення

продуктивності або зменшення ризиків) або передбачення, або оцінки впливу. У поєднанні з даними, що постачаються у реальному часі, вони можуть стати потужним компонентом сучасних систем прийняття рішень та управління. Тому розробка нових методів та програмних засобів, що спрямовані на підвищення ефективності процесу програмної реалізації моделі за рахунок автоматизації кодування, є актуальним науковим завданням.

Зв’язок роботи з науковими програмами, планами, темами.

Наукове дослідження за темою дисертації проводилось у Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського» у відповідності до Переліку пріоритетних тематичних напрямів наукових досліджень і науково-технічних розробок на період до 2022 року, затвердженого постановою Кабінету Міністрів України №942 від 7.09.2011 (напрям ”Технології та засоби розробки програмних продуктів і систем”), та у відповідності до тематики наукових розробок кафедри. Результати наукового дослідження є результатом виконання таких науково-дослідних робіт: «Створення засобів імітаційного моделювання дискретно-подійних систем» (державний реєстраційний номер 0117U000923), «Методи візуального програмування Петрі-об'єктних моделей» (державний реєстраційний номер 0117U000918).

Мета і завдання дослідження. Метою наукового дослідження є підвищення ефективності програмних засобів представлення складних моделей систем для цілей імітаційного моделювання за рахунок зменшення складності конструювання моделі та збільшення швидкодії алгоритмів імітації, зручності сприйняття моделі, зменшення кількості помилок при створенні зв’язків між елементами моделі та зменшення часу на модифікацію моделі. Для досягнення мети поставлені та вирішені такі завдання:

- аналіз існуючих програмних засобів моделювання систем на основі формального опису стохастичною мережею Петрі;
- розробка елементів візуального представлення Петрі-об’єктів та зв’язків між ними;

- розробка формальної граматики візуального представлення Петрі-об’єктної моделі;
- розробка транслятору мови візуального представлення Петрі-об’єктної моделі;
- реалізація розробленої мови програмними засобами;
- дослідження коректності та складності розробки Петрі-об'єктної моделі розробленою мовою на моделях прикладного призначення.

Об’єкт дослідження – процеси розроблення засобів візуального програмування моделей складних систем**.**

Предмет дослідження – методи та засоби візуального програмування імітаційних моделей систем на основі формального опису мережами Петрі.

Методи дослідження: теорія формальних мов, теорія алгоритмів, теорія мереж Петрі, теорія ймовірностей та математичної статистики, імітаційне моделювання.

Наукова новизна отриманих результатів:

– **Вперше** розроблено візуальну мову програмування Петрі-об’єктних моделей, яка дозволяє спростити процес побудови моделей, підвищити наочність сприйняття моделей та час виконання імітаційного моделювання шляхом організації дворівневого способу побудови моделей. На відміну від існуючих рішень в області імітаційного моделювання, візуальна мова програмування Петрі-об’єктних моделей надає універсальний та гнучкий інструмент для побудови моделей дискретно-подійних систем, а процес виконання імітаційного моделювання не залежить від обчислювальних ресурсів користувача.

– **Удосконалено** Петрі-об’єктну модель за рахунок введення поняття групи Петрі-об’єктів, колекції Петрі-об’єктів та групи колекцій Петрі-об’єктів, що, на відміну від існуючих засобів представлення імітаційної моделі, надають можливість тиражувати Петрі-об’єкти з заданими наборами параметрів, тиражувати колекції взаємопов’язаних Петрі-об’єктів та тиражувати зв’язки між Петрі-об’єктом та групою Петрі-об’єктів, між

Петрі-об’єктом та групою колекцій Петрі-об’єктів. За рахунок тиражування однотипних фрагментів моделі та тиражування зв’язків створюються умови для швидкого конструювання моделей з великою кількістю елементів та значно скорочується обсяг їх візуального представлення.

– **Вперше** запропоновано клієнт-серверну архітектуру серед програмного забезпечення з імітаційного моделювання на основі Петрі-об’єктного підходу, використання якої дозволяє задіяти ресурси віддаленого серверу для проведення імітаційного моделювання для забезпечення стабільного часу виконання, зменшення витрат на інфраструктуру обчислювальних ресурсів серед користувачів та організації спільного доступу до розробки моделей.

Практичне значення отриманих результатів.

Розроблено програмну реалізацію (програмні засоби) мови візуального програмування Петрі-об’єктної моделі, що є логічним завершення виконання науково-дослідної роботи «Методи візуального програмування Петрі-об'єктних моделей» (державний реєстраційний номер 0117U000918). Запропоноване програмне забезпечення використовувалась при розробці моделей телекомунікаційної системи, інформаційної розподіленої системи, масового обслуговування, які були представлені на міжнародних конференціях та отримали позитивні відгуки.

Особистий внесок здобувача.

Усі результати наукового дослідження, представлені до захисту, отримані автором особисто. У написаних у співавторстві публікаціях автору належать такі результати: [1] – розробка дворівневого візуального представлення Петрі-об’єктної моделі; [3] – розробка формальної граматики мови візуального програмування Петрі-об’єктних моделей; [4] – розробка та дослідження ефективності веб-сервісу моделювання систем на основі стохастичних мереж Петрі; [5] – огляд існуючого програмного забезпечення з моделювання мережами Петрі, удосконалений математичний опис Петрі-об’єктної моделі, а також

приклад розробки Петрі-об’єктної моделі інформаційної системи та її дослідження; [6] – порівняльний аналіз симуляторів мереж Петрі, розробка клієнт-серверного застосування моделювання стохастичною мережею Петрі; [7] - дослідження складності алгоритму імітації Петрі-об'єктної моделі.

Апробація результатів дисертації.

Основні результати наукового дослідження доповідались та отримали позитивні відгуки на таких наукових конференціях: Mathematical Modeling and Simulation of Systems (MODS 2020), 2017 Internet Technologies and Applications (ITA), 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS).

Публікації.

Результати наукового дослідження опубліковано у 7 наукових публікаціях, серед яких 1 стаття у періодичному науковому виданні, проіндексованому у Web of Science Core Collection та Scopus базах даних, 2 статті у фаховому науковому журналі категорії «Б», 1 стаття у фаховому науковому журналі категорії «В», 1 стаття у науковому журналі, проіндексованому GoogleScholar, 2 публікації у матеріалах міжнародних наукових конференцій, проіндексованих у Scopus, 1 публікація у матеріалах міжнародної науково-технічної конференції.

Структура і обсяг роботи. Дисертація складається з вступу, 5 розділів, висновків, списку використаних джерел з 57 найменування та 2 додатків. Загальний обсяг роботи складає 100 сторінок, з них 81 сторінка основного тексту, 29 рисунків, 7 таблиць.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПРОГРАМНИХ ЗАСОБІВ МОДЕЛЮВАННЯ СТОХАСТИЧНИМИ МЕРЕЖАМИ ПЕТРІ

1.1 Роль і місце програмних засобів моделювання стохастичними мережами Петрі в розвитку інформаційних технологій

Переважна більшість розроблених інформаційних систем залишаються ‘data rich, information poor’, тобто насиченими даними, але бідними на інформацію. IBM Center for The Business of Government серед тенденцій розвитку інформаційних систем вказує надання даних майже в реальному часі і спрощення інтерпретації даних [1]. Щоб бути корисними, дані повинні бути перетворені в інформацію. Таке перетворення виконують моделі, серед яких особливе місце займають імітаційні моделі дискретно-подійних систем. На сьогоднішній день цей метод моделювання може бути застосований до широкого кола складних систем (з метою пошуку оптимальних умов або короткострокового прогнозування), і у поєднанні з даними в реальному часі компоненти моделювання можуть створювати потужні системи прийняття рішень і управління. Однак виникають спеціальні вимоги до імітаційної моделі, а саме, код моделі повинен бути реалізований як частина програмного забезпечення, а модель повинна мати можливість інтегруватися зі сховищем даних та інтерфейсом кінцевого користувача. Потребує також удосконалення процес розробки моделей складних систем, оскільки часто цей процес є настільки довготривалим і трудомістким, що переваги від використання моделі можуть не покрити витрати на її створення. Тому розробка нових методів та програмних засобів моделювання є важливою для розвитку інформаційних технологій.

Особливе місце в імітаційному моделюванні займають візуальні засоби представлення моделей та конструювання моделі з використанням графічного редактора та засобів анімації. Ті з них, що не побудовані на універсальному формалізмі опису моделі, приречені на невпинне збільшення кількості різнотипних елементів, необхідних для покриття широкого класу систем, що моделюються, і відповідно збільшення складності застосування, оскільки кожен

елемент потребує окремих знань про його параметри та логіку функціонування (наприклад, Simio [2]). А ті, що використовують обмежений набір елементів для представлення моделі, страждають від необхідності використовувати велику кількість елементів для представлення складних систем і намагаються вирішувати цю проблему введенням більш складного опису для параметрів елементів (наприклад, CPNTools використовує функціональну мову програмування ML для опису параметрів елементів розфарбованої мережі Петрі [3]).

Мережі Петрі є ключовим формалізмом для моделювання дискретно-подійних систем, який покриває широкий клас систем від автоматних до стохастичних [4]. В контексті інженерії програмного забезпечення формалізм мереж Петрі є важливим ще й тому, що є загальноприйнятим для розробки паралельних та розподілених обчислень у відповідності до стандарту системної та програмної інженерії ISO/IEC 15909-1:2004. Стандарт підкреслює такі переваги використання мереж Петрі в системах програмування та проектуванні програмного забезпечення: математичний формалізм мережі Петрі забезпечує однозначні специфікації та описи додатків, графічне представлення дозволяє візуалізувати потік ресурсів та потік керування для кращого розуміння поведінки системи, імітація дозволяє перевірити ідеї у найдешевший спосіб [5]. Застосування мереж Петрі для розробки алгоритмів і програмного забезпечення з часом може втілитись у парадигму програмування на мережах Петрі [6]. Отже, мережі Петрі як засіб побудови моделей є надзвичайно важливим для розвитку програмної інженерії.

Недолік, з яким доводиться стикатись при розробці моделей мережами Петрі - це ускладнення сприйняття та побудови моделі при зростанні складності системи, що моделюється (складність може бути оцінена кількістю елементарних подій). Через нагромадження зв’язків та елементів перевага візуального сприйняття моделі зникає, модифікація параметрів моделі потребує значної кількості рутинних дій. Тому розвиток методів, що спрощують і

прискорюють розробку моделей на основі мереж Петрі є актуальним науковим завданням.

1.2 Огляд сучасних програмних засобів моделювання стохастичними мережами Петрі

Стохастичні мережі Петрі використовують для розробки імітаційних моделей. На відміну від класичних мереж Петрі переходи мережі Петрі містять часові затримки, задані випадковими числами. В залежності від реалізації програмного засобу часові затримки можуть бути обмежені виключно розподіленими за експоненціальним законом, або кількома законами розподілу серед яких, як правило, експоненціальний, рівномірний та нормальний (Гауса), оскільки для цих законів розподілу відомі генератори випадкових чисел.

Якщо для класичних мереж Петрі є велике різноманіття програмних засобів, то для стохастичних мереж Петрі існує не так багато широковідомих засобів.

1.1.1 Oris Tool

Розроблений на java, підтримує ОС Linux/macOS, Windows. Тьюторіал програмного забезпечення міститься за посиланням [7]. Має графічний редактор мережі Петрі (рис. 1.1). Для дуг не передбачений параметр кратності, що значно звужує коло систем, які можуть бути модельовані такою мережею. Параметри елементів задаються у вікні, що відкривається подвійним кліком на елементі. Параметри пріоритету чи ймовірності запуску для переходу не передбачені і у тьюторіалі програмного продукту не вказано за яким правилом вирішується конфлікт між переходами. Немає ознак того, що виконується підрахунок статистичних характеристик за результатами імітації. Проте виконується аналіз досяжності стану з визначенням ймовірності перебування у кожному досяжному стані. Протокол подій містить інформацію про стан мережі у кожний момент часу. Час при цьому вказаний з детермінованим кроком. Збереження моделей відбувається у вигляді XML опису [8].

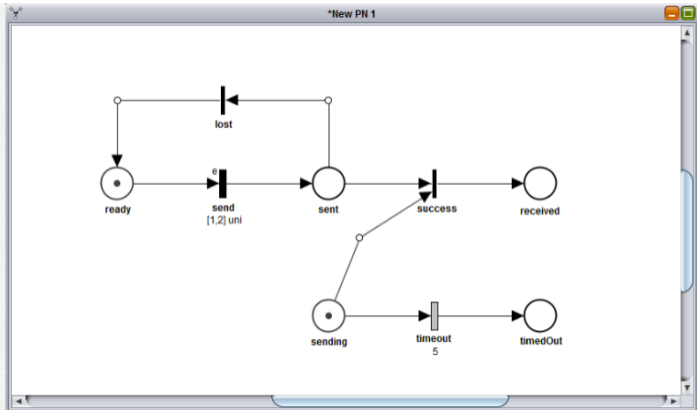


Рисунок 1.1 – Візуальне представлення мережі Петрі в редакторі Oris Tool [6]

1.1.2 TimeNET

Розроблений на Java під ОС Windows та UNIX. Містить графічний редактор мережі Петрі, в якому окрім звичайних дуг використовуються також інгібіторні дуги, для яких використовується спеціальне графічне позначення з кружечком на кінці замість стрілки (рис. 1.2). Параметр кратності дуги може бути вказаний для звичайної дуги додатково у вікні параметрів відповідної дуги. Переходи можуть бути задані як миттєві, так і з часовою затримкою і для них використовують різне графічне представлення (чорний та білий прямокутник відповідно). Часова затримка може бути задана детермінованим значенням або випаковим числом за заданим законом розподілу, серед яких трикутний, рівномірний, експоненціальний, або випадковим числом за емпіричним законом розподілу [9]. Для вирішення конфліктів між переходами використовуються пріоритети та ймовірності запуску, але ці параметри передбачені тільки для миттєвих переходів. Це означає, що у випадку моделювання розгалуження подій доведеться спочатку робити розгалуження серед миттєвих переходів і потім тільки описувати події з часовими затримками. Очевидно, це призводить до надлишкових елементів в описі моделі.

Для переходу з часовою затримкою може бути заданий параметр «тип переходу», що вказує на можливість запуску переходу або по одному, або без обмеження. Такий підхід об'єднує в собі два можливі підходи при використанні переходів мереж Петрі: одноканальні та багатоканальні. Контролювати обидва

підходи одночасно не є правильним з точки зору математичного формалізму і, по-друге, незручно для користувача, оскільки потрібно тримати увагу на визначенні типу кожного переходу.

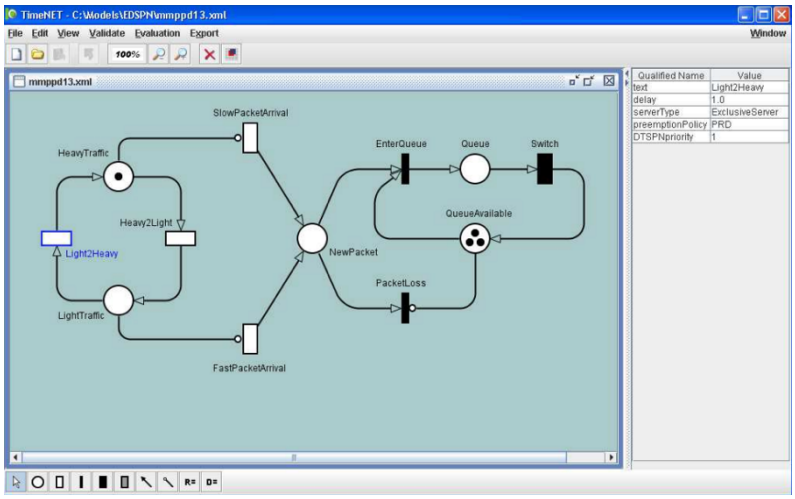


Рисунок 1.2 – Візуальне представлення мережі Петрі в редакторі Time NET 4.0 [8]

Маркери можуть бути різних типів з обмеженням, що в одній позиції можуть знаходитись тільки маркери одного типу (тобто підтримуються розфарбовані мережі Петрі). Відповідно, в переходах мають бути задані специфічні умови для запуску у вигляді булевих функцій, що визначають необхідну для запуску кількість маркерів кожного типу у вхідних позиціях переходу, замість простого правилу запуску класичної мережі Петрі, що читається (сприймається) з її візуального зображення.

Програмний засіб підтримує анімацію змінювання стану мережі Петрі, імітацію з обчисленням статистичних значень величин кількості маркерів у позиції та пропускної здатності переходу, а також аналіз досяжних станів мережі Петрі.

1.1.3 CPNTools

Найбільш відомим засобом моделювання мережами Петрі на сьогоднішній день є CPNTools [10]. Цей програмний засіб використовує графічну мову представлення розфарбованої мережі Петрі з підтримкою ієрархічної структури моделей за принципом “мережа в мережі”: переходи вищого рівня мережі

можуть містити в собі мережу Петрі (а позиції – не можуть). Копії фрагменту мережі Петрі нижнього рівня можуть використовуватись у переходах вищого рівня. Обмеження на кількість рівнів в моделі немає. При вході в перехід верхнього рівня активуються події мережі Петрі нижнього рівня. Умовою входу у перехід вищого рівня є, як і для звичайного переходу, наявність маркерів у вхідних позиціях переходу. Для переходів можуть бути задані часові затримки з заданим законом розподілу та пріоритети запуску. Маркери визначаються з приналежністю до певного типу і, відповідно, умова запуску переходу формулюється з указуванням не тільки кількості маркерів, але і їх типу.

CPN-Tools 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 [3]. 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. Застосування розфарбованих мереж Петрі для розробки моделі протоколу обміну даними представлено у роботі [11].

Зауважимо, що графічно мова CPNTools значно відходить від традиційного представлення елементів мережі Петрі: замість кількості маркерів у позиції вказується ім’я (кількість вказується у полі поруч), у зображенні переходу з часовою затримкою з’являється ім’я, яке вказує або на ім’я переходу або на ім’я мережі нижчого рівня (підмережі), а на дугах вказуються функціональні вирази (рис. 1.3). Переходи можуть містити часові затримки, гард-блоки та блоки з фрагментами коду (рис. 1.4). Такі елементи опису сприяють гнучкості опису моделі, сприяють розширенню класу задач, які можуть бути представлені цим описом, але, водночас, збільшують складність представлення і, по суті, знищують основну перевагу мереж Петрі – простоту та наочність представлення процесу функціонування.

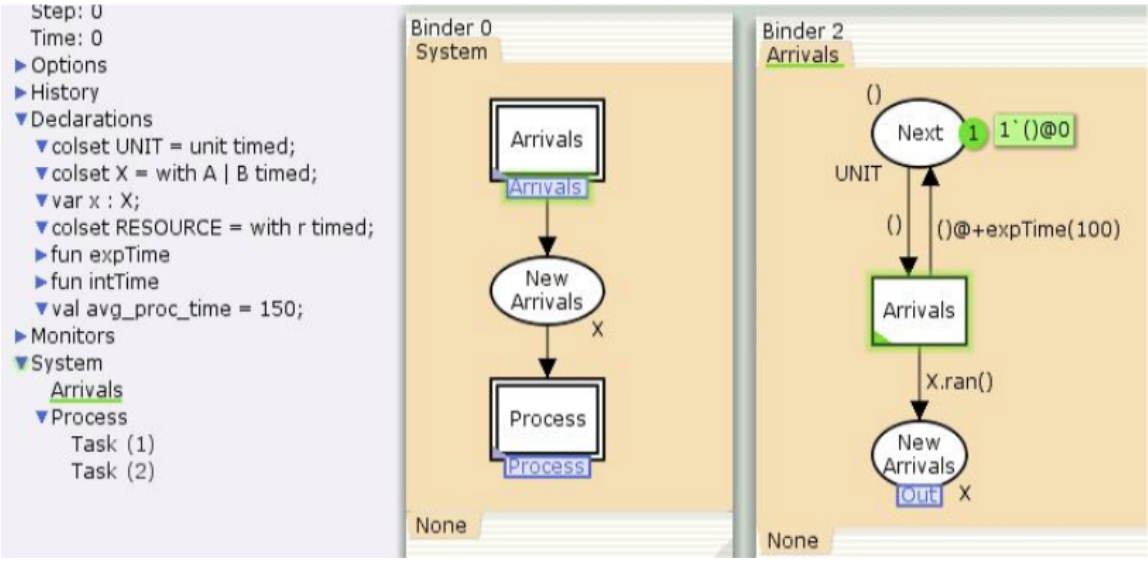
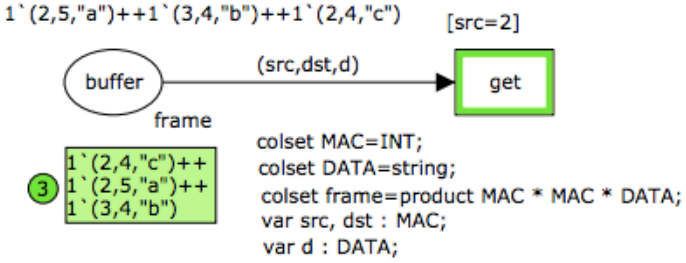
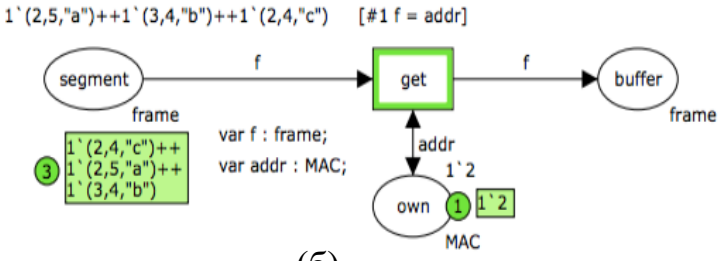


Рисунок 1.3 – Візуальне представлення ієрархічної мережі Петрі в редакторі CPNTools [12].



(а)



(б)

Рисунок 1.4 – Приклади застосування переходу мережі Петрі в редакторі CPNTools: а) з часовою затримкою, б) з гард-блоком [13]

1.1.4 Renew 4.0

Розроблений мовою java симулятор мереж Петрі Renew 4.0 втілює концепцію nets-within-nets, тобто мережа в мережі, в якій позиції можуть містити посилання на фрагмент мережі Петрі. Версія 4.0 програми випущена у 2022 році [14]. В оновленій версії не відмовились від Java plug-in, що визнана небезпечною і не рекомендується для використання. Автори програмного продукту визнають цей баг і обіцяють в подальшому виправити. Цікаво також, що може запускатись віддалено з використанням технології RMI (розробники Java визнали цю технологію застарілою і видалили її зі стандартних бібліотек у Java SE 17.0 [15]).

Часові затримки, у тому числі випадкові, використовують у Renew-реалізації мережі Петрі на дугах замість переходів (рис. 1.3). Маркери можуть бути об’єктами різних класів (в термінах об’єктно-орієнтованого програмування) і при цьому не має обмеження, що в одній позиції знаходяться маркери одного типу. Окрім звичайних дуг, використовують резервні та тестові дуги. Резервна дуга є комбінацією вхідної та вихідної дуги між переходом та позицією, а тестова дуга надає можливість перевіряти наявність маркеру кільком переходам одночасно.

Фрагменти мереж Петрі можуть використовуватись повторно, якщо вони збережені у репозиторії. Концепція мережа в мережі надає можливість створювати моделі поступово спускаючись від верхнього рівня до нижнього. Ієрархічність побудови моделі є суттєвим обмеженням, оскільки елементи нижнього рівня не можуть взаємодіяти між собою. При такій конструкції будь-який елемент нижнього рівня може взаємодіяти тільки з елементом верхнього рівня, який його створив. Окрім того концепція “мережа в мережі” не вирішує проблеми створення великої кількості типових фрагментів мережі Петрі, оскільки кожен фрагмент має бути створений окремо і прикріплений до елементу верхнього рівня моделі.

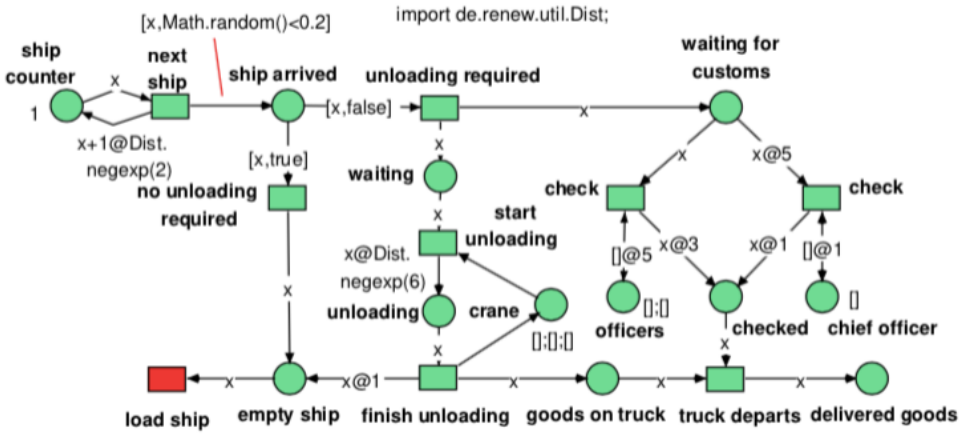


Рисунок 1.3 – Візуальне представлення мережі Петрі в редакторі Renew 4.0 [16]

1.1.5 P*AOSE

Програмний продукт Petri Net-based Agent-Oriented Software Engineering approach (P*AOSE) розроблений на основі Renew і продовжує реалізацію концепції nets-within-nets тобто мережі в мережі для розробки моделей паралельних та розподілених систем на етапі проектування програмного забезпечення [17]. Окрім редактора мереж Петрі цей програмний продукт містить UML-діаграми опису програмного забезпечення для опису моделей на верхньому рівні побудови. Отже, при переході до моделювання складних програмних систем обмежитись представленням виключно мережею Петрі авторам Renew не вдається. Побудована у такий спосіб модель втрачає єдність формального опису у вигляді мережі Петрі і взагалі втрачає формальність свого опису, оскільки для поєднання мережі Петрі та UML-діаграм формалізму, який їх об’єднує на сьогоднішній день не розроблено.

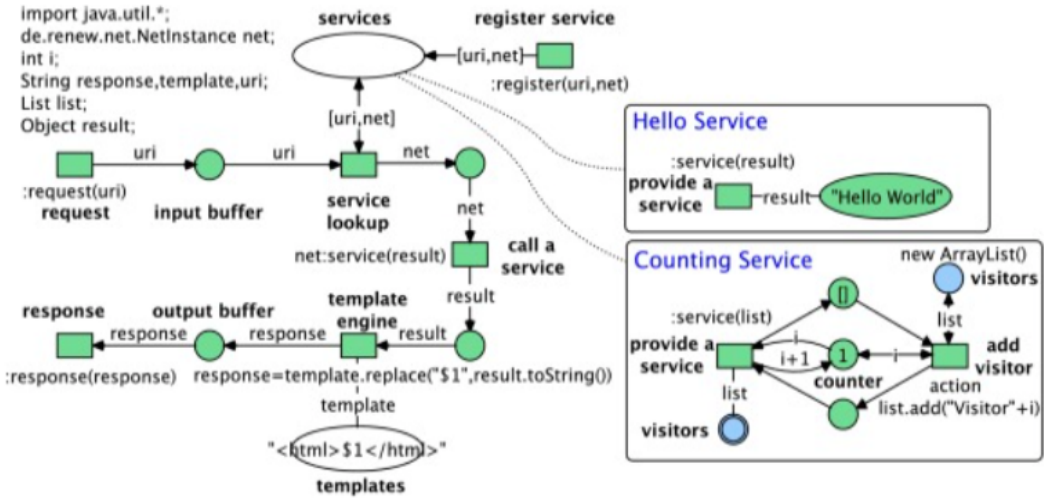


Рисунок 1.4 – Модель веб-сервера, представлена [17]

1.1.6 PetriObjLib

Петрі-об’єктні моделі вирішують проблему тиражування фрагментів мереж Петрі з заданими параметрами та конструювання моделі системи з великої кількості елементів. Якщо система складається зі 100 однотипних вузлів, опис кожного з яких потребує 10 переходів, 20 позицій та 40 дуг, то в термінах звичайної мережі Петрі довелось би створювати 1000 переходів, 2000 позицій, 4000 дуг та щонайменше 100 дуг для зв’язування фрагментів між собою. В термінах Петрі-об’єктної моделі один раз розроблена мережа Петрі, що описує функціонування одного вузла системи, використовується для створення 100 вузлів (із різними значеннями параметрів), отже, розробка моделі значно спрощується і прискорюється. Скорочується також обсяг графічної інформації, яку потрібно зберігати для відтворення моделі. Широко відомий засіб для моделювання мережами Петрі CPNTools надає можливість використовувати фрагменти мережі Петрі повторно, проте налагоджувати і вбудовувати фрагмент в модель доведеться для кожного фрагменту окремо. Звісно, такий спосіб спрощує створення нового фрагменту мережі Петрі на основі вже існуючого, але не спрощує створення множинних наборів типових фрагментів. По суті, надається можливість копіювання фрагментів і створення на їх основі нових, але не тиражування за заданим шаблоном з заданими параметрами. Зберігатись в

CPNTools будуть усі скопійовані фрагменти як наново створені, отже, на відміну від Петрі-об’єктного підходу, скорочення обсягів пам’яті на збереження фрагментів моделі не відбувається. Порівняння програмного забезпечення з Петрі-об’єктного моделювання з відомими симуляторами мереж Петрі такими, як Coopn (Concurrent Object-Oriented Petri net) builder, JSARP (Simulator and Analyzer Petri net in Java), PNTalk, Renew, CPN (Coloured Petri nets) Tools, Petri.NET Simulator, WoPeD(Workflow Petri Net Designer), PIPE2 (Platform-Independent Petri net Editor), наведено у робті [18].

Петрі-об’єктні моделі надають можливість створювати моделі засобами мереж Петрі там, де раніше вони призводили до надмірного ускладнення: моделювання розповсюдження комп’ютерних вірусів [19], передача повідомлень у телекомунікаційних системах [20], моделювання алгоритмів паралельних обчислень [21]. Застосування Петрі-об’єктного підходу для моделювання систем різного призначення показало ефективність цієї технології конструювання моделей, що поєднує в собі переваги об’єктно-орієнтованого підходу та стохастичних мереж Петрі. Проте кодування зв’язків між елементами мережі Петрі є процес, схильний до помилок. Відшукання помилок у структурі моделі безпосередньо в коді потребує концентрації уваги. Якщо модель містить опис двох десятків подій, з цим ще можна впоратись. При більшій кількості подій процес розробки моделі настільки трудомісткий, що результати моделювання можуть не виправдати зусиль, витрачених на створення моделі.

Рішення цієї проблеми можна шукати у двох напрямках: 1) автоматизація, інтелектуалізація чи інші способи запобігання та виявлення помилок безпосередньо в коді, 2) візуалізація зв’язків та моделі в цілому для поліпшення сприйняття моделі, що будується. Перевірити правильність зв’язків чи стану моделі в початковий момент часу з використанням інтелектуальних підходів не представляється можливим, оскільки можливі (допустимі) послідовності подій, які відповідають задуму моделі, є надзвичайно великою множиною навіть для відносно простих моделей.

Підходи, які використовують виключно візуальне представлення моделі, примушують програміста відкривати та модифікувати модель навіть при зміні її параметрів. А якщо потрібна зміна водночас в кількох однотипних елементах, то доведеться повторювати рутинну дію. Такі дії швидше виконати безпосередньо в коді (при цьому потрібно забезпечити, щоб при відкритті моделі такі зміни потрапили у візуальне її представлення).

Ідеальним вбачається рішення, при якому програміст може застосовувати візуальне представлення і, водночас, використовувати кодоване представлення моделі. Проте перетворення візуального представлення моделі у кодоване відбувається, як правило, з втратою даних про розміщення графічних елементів. Тому перетворення з кодованого представлення моделі у візуальне її представлення потребує інтелектуальної обробки інформації про взаємозв'язки елементів. Ускладнює таке перетворення той факт, що не існує однозначної відповідності між кодованим представленням та візуальним представленням моделі, оскільки одній моделі може відповідати кілька варіантів візуального представлення з різним розміщенням графічних об'єктів.

Конфлікт між графічним представленням і кодованим представленням моделі зникає, якщо припустити, що візуальне представлення є водночас кодованим представленням моделі. Візуальні мови програмування використовують графічні об'єкти як алфавіт мови, отже структури, складені з таких об'єктів є, фактично, кодом для представлення моделі.

1.3 Висновки до розділу 1

Виконано огляд існуючих програмних засобів моделювання систем засобами стохастичних мереж Петрі, які використовують для представлення моделей графічні редактори або графічні мови. Виявлено, що графічні редактори не вирішують низку проблем: неможливість коригування параметрів моделі без коригування її візуального представлення, неможливість тиражування однотипних елементів у великій кількості (оскільки усі вони будуть займати візуальний простір), неможливість тиражування зв'яків при повторному

використанні фрагментів мережі Петрі, неможливість налаштування параметрів елементів без коригування кожного окремого параметра у спеціально відведеному для цього вікні. Шляхом до вирішення вказаних проблем визначено побудови мови візуального програмування.

2 ФОРМАЛІЗМ ПЕТРІ-ОБ'ЄКТНОГО МОДЕЛЮВАННЯ

2.1 Стохастична мереже Петрі з багатоканальними та конфліктними переходами

Мережа Петрі (Petri net), винайдена Карлом Адамом Петрі [22] для проектування цифрових автоматів, була розвинута і удосконалена багатьма дослідниками для цілей моделювання дискретно-подійних систем та дослідження властивостей дискретно-подійних систем. Найбільш значними публікаціями в теорії мереж Петрі є роботи Т. Мурати [23] та Дж. Пітерсона [24], де викладено основні положення теорії мереж Петрі та математичні методи дослідження їх властивостей. Класична теорія мереж Петрі розроблена для мереж, в яких перехід з одного стану в інший спрацьовує миттєво. Такі мережі корисні для дослідження автоматних систем і відтворюють послідовність змін станів цифрового автомату.

Мережі Петрі з часовими затримками (timed Petri nets) доповнюють опис переходу параметром, що характеризує час виконання певних дій, пов'язаних з подією. Математичні рівняння мережі Петрі з детермінованими часовими затримками отримані в роботі [25]. Запуск переходу здійснюється шляхом двох змін стану з інтервалом часу між ними замість миттєвої однієї зміни стану. Одна зміна стану відповідає змінам, які відбуваються на початку виникнення події, а друга – змінам, які відбуваються при завершення події.

Стохастична мережа Петрі має часові затримки у переходах, визначені як випадкові числа з заданим законом розподілу. Закон розподілу може бути заданий відомим теоретичним законом розподілу або емпіричним законом розподілу. Способи побудови генераторів випадкових чисел у відповідності до заданого закону розподілу описані у монографії [26]. Математична теорія стохастичної мережі Петрі викладена у монографії [27].

Універсальною стохастичною мережею Петрі (generalized stochastic Petri net) називають мережу Петрі, в якій часова затримка може бути миттєвою, детермінованою або стохастичною. Є варіанти універсальної мережі Петрі, в

яких миттєві затримки розглядаються як окрема множина переходів з правилом запуску як у класичній мережі. Є також варіанти, в яких миттєвий перехід реалізується через перехід з нульовою часовою затримкою. Універсальні стохастичні мережі Петрі значно розширили коло задач, які можна вирішувати мережами Петрі, від автоматних систем до дискретно-подійних систем.

У класичній мережі Петрі вхід в перехід здійснюється однократно. Багатоканальні переходи означають, що в мережі Петрі здійснюється багатократний вхід в переходи до досягнення стану, коли жоден з її переходів не запускається. Використання багатоканальних переходів надає можливість значно зменшити кількість елементів, необхідних для представлення моделі у випадку, коли одна й та сама подія може відбуватись одночасно для кількох. Багатоканальні переходи запропоновані у [25].

У роботі [28] виведені логіко-алгебраїчні рівняння станів стохастичної мережі Петрі з багатоканальними та конфліктними переходами і доведено, що вони є універсальними, оскільки можуть бути приведені до рівнянь стану детермінованої мережі Петрі та до рівнянь стану класичної мережі Петрі. Оскільки саме така мережа Петрі використовується для цілей Петрі-об'єктного моделювання, наведемо коротко її опис. Будемо далі всюди під 'мережею Петрі' розуміти 'стохастичну мережу Петрі з багатоканальними та конфліктними переходами'.

За визначенням, стохастична мережа Петрі з багатоканальними та конфліктними переходами є мультимножиною [28]:

$$N = \{\mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{I}, \mathbf{W}, \mathbf{R}\} \tag{2.1}$$

де $\mathbf{P}$ – множина позицій, $\boldsymbol{T}$ – множина переходів,

$\mathbf{T} \cap \mathbf{P} = \emptyset$, $\mathbf{A} \subseteq (\mathbf{P} \times \mathbf{T} \cup \mathbf{T} \times \mathbf{P})$ – множина дуг,

$\mathbf{I} \subseteq (\mathbf{P} \times \mathbf{T})$ – множина інформаційних дуг,

$\mathbf{W}$: $\mathbf{A} \longrightarrow \mathbb{N}$ – множина натуральних чисел, що задають кратності дуг,

$\mathbf{R}$: $\mathbf{T} \longrightarrow (\mathbb{N} \times \mathbb{V} \times \Re_{.})$, $\mathbb{V} = \{v_{<} \in [0;1] | \, T \in \mathbf{T}\}$ – множина параметрів переходів, що задають пріоритет (натуральне число), ймовірність запуску

(дійсне число в межах від 0 до 1) та часову затримку (дійсне невід'ємне число) для кожного переходу.

Стан мережі Петрі $\mathbf{S}(t)$ у будь-який момент часу *t* визначається станом її позицій $\mathbf{M}(t) = \{M_{\boldsymbol{P}}(t)|P \in \mathbf{P}\}$ та станом її переходів $\mathbf{E}(t) = \{E_<(t)|T \in \mathbf{T}\}$. Стан позиції, як і вкласичній мережі Петрі, визначається кількістю маркерів в позиції $M_{\boldsymbol{P}}(t) \in \mathrm{N} \cup \{0\}, P \in \mathbf{P}$. Стан переходу визначається множиною дійсних невід'ємних значень, що визначають моменти виходу з його каналів $E_<(t) = \mathrm{N}[E_<(t)]_0|[E_<(t)]_0 \in \Re_:, j \in \mathbb{N}\mathrm{Q}$. У початковий момент часу $E_<(t) = \{\infty\}$ (у програмній реалізації нескінченність реалізується максимально можливим дійсним значенням). В інші моменти часу може бути будь-скільки значень у множині. Значення ∞ має такий зміст, що в переході не оікується вихід маркерів у майбутньому.

Предикати $I(T, t_\mathrm{T})$ та $O(T, t_\mathrm{T})$ визначають умови входу та виходу маркерів з переходів мережі Петрі:

$$I(T, t_\mathrm{T}) = \begin{matrix} 1, & \forall P\colon (P,T) \in \boldsymbol{A} \quad M_{\boldsymbol{P}}(t_{\mathrm{TYZ}}) \geq W_{],<} \\ 0, & \exists P\colon (P,T) \in \mathbf{A} \quad M_{\boldsymbol{P}}(t_{\mathrm{TYZ}}) < W_{],<} \end{matrix}, \tag{2.2}$$

$$O(T, t_\mathrm{T}) = \begin{matrix} 1, & \min_{\mathrm{C}}[E_<(t_{\mathrm{TYZ}})]_0 = t_\mathrm{T} \\ 0, & \min_{\mathrm{C}}[E_<(t_{\mathrm{TYZ}})]_0 \neq t_\mathrm{T} \end{matrix}. \tag{2.3}$$

Рівняння станів математично описують правила змінювання стану мережі Петрі і мають такий вигляд:

$$\begin{cases} t_\mathrm{T} = \min_{<} \mathrm{h}\min_{\mathrm{C}}[E_<(t_{\mathrm{TYZ}})]_0\mathrm{i}, \\ \mathbf{S}(t_\mathrm{T}) = (D^\mathrm{Y})^\mathrm{k} D^{\cdot}\mathrm{l}\mathbf{S}(t_{\mathrm{TYZ}})\mathrm{m}, \ m\colon \bigvee_{<} I(T, t_\mathrm{T}) = 0, \\ n = 1{,}2 \ldots \end{cases} \tag{2.4}$$

де $D^{:}$– перетворення стану мережі Петрі, що відповідає виходу маркерів з її переходів,

D^Y– перетворення стану мережі Петрі, що відповідає входу маркерів з її переходів,

$I(T, t_\mathrm{T})$ – предикат, який приймає значення 1, якщо умова входу маркерів для переходу T виконана в момент t_T ,

m – кількість входів в переходи мережі Петрі до досягнення стану, в якому для жодного з її переходів не виконана умова входу маркерів.

У порівнянні з [28] представлення рівнянь станів скорочено за рахунок відкидання рівняння з початковим перетвореннням мережі Петрі. Оскільки у початковому стані для всіх переходів мережі Петрі не виконана умова виходу маркерів, то перетворенння $D^{\cdot}$ не буде виконуватись для жодного з переходів. Тому немає потреби окремо вказувати рівняння для першого перетворення мережі від початкового стану.

Перше рівняння системи (2.4) визначає просування часу до моменту найближчої події. Друге рівняння визначає композицію перетворень, що здійснюються при кожному просуванні часу. Зауважимо, що в кожний момент часу відбувається спочатку вихід маркерів з переходів, а потім здійснюється вхід маркерів в переходи. Логіко-алгебраїчні рівняння перетворень описані у роботі [28]. Умови для здійснення перетворень описуються предикатами, а перетворення – набором арифметичних та логічних дій.

У перетворенні $D^{\cdot}$приймають участь усі переходи, для яких $O(T, t_{\mathrm{T}}) = 1$. Вихід маркерів означає, що значення $[E_<(t_{\mathrm{TYZ}})]_{\mathrm{s}} = t_{\mathrm{T}}$ видаляються з множини $E_<$: $E_<(t_{\mathrm{T}}) = E_<(t_{\mathrm{TYZ}}) \backslash \{[E_<(t_{\mathrm{TYZ}})]_{\mathrm{s}}\}$. Якщо після видалення множина $E_<$ стала порожньою, то в неї додається значення ∞. При кожному виході маркерів з каналу переходу відбувається додавання у вихідні позиції переходу:

$$O(T, t_{\mathrm{T}}) = 1 \;\Rightarrow\; \forall P: (T, P) \in \boldsymbol{A} \quad M_{\boldsymbol{P}}(t_{\mathrm{T}}) = M_{\boldsymbol{P}}(t_{\mathrm{TYZ}}) + W_{<,]} \qquad (2.5)$$

У перетворенні D^{Y} приймають участь переходи, для яких $I(T, t_{\mathrm{T}}) = 1$. Вхід маркерів означає, що у множину $E_<$ додається значення моменту виходу маркерів з переходу: $E_<(t_{\mathrm{T}}) = E_<(t_{\mathrm{TYZ}}) \cup \{t_{\mathrm{TYZ}} + R_<\}$, а з вихідних позицій переходу маркери вилучаються:

$$I(T, t_{\mathrm{T}}) = 1 \;\Rightarrow\; \forall P: (P, T) \in \boldsymbol{A} \quad M_{\boldsymbol{P}}(t_{\mathrm{T}}) = M_{\boldsymbol{P}}(t_{\mathrm{TYZ}}) - W_{],<} \qquad (2.6)$$

Оскільки кількість маркерів, які можна відняти, обмежена, то у випадку, коли більше ніж для одного переходу виконан умови запуску, потрібно розв’язувати конфлікт. Саме тому вхід маркерів здійснюється багатократно.

Кожного разу з множини переходів, для яких $I(T, t_{\mathrm{T}}) = 1$ обчирається тільки один перехід і для нього виконується вхід маркерів.

Розв'язання конфліктів здійснюється таким чином, що з усіх переходів, для яких $I(T, t_{\mathrm{T}}) = 1$, обираються ті, що з найвищим пріоритетом. Якщо таких виявляється декілька, то конфлікт між ними вирішується за заданими ймовірностями.

Логіко-алгебраїчні вирази для перетворень використовуються у цій дисертаційній роботі в такому вигляді, як вони описані у публікації [28].

Таким чином, математично визначені такі правила функціонування мережі Петрі:

- просування часу здійснюється до найближчої події, який визначається найменшим з усіх моментів виходу маркерів з переходів мережі Петрі;
- при кожному просуванні часу здійснюється вихід маркерів для усіх переходів, що мають у своїх множинах моментів виходу з переходу такі, що співпадають з поточним моментом часу, і здійснюється стільки разів, скільки знайдено таких значень;
- після виходу маркерів з переходу відбувається багатократний вхід маркерів в переходи мережі Петрі, для яких виконана умова входу маркерів, до досягнення стану, коли для жодного з переходів не виконана умова входу маркерів.

Стохастична мережа Петрі є універсальним формалізмом для опису дисретно-подійних систем. Однак при значному збільшенні кількості подій застосування її стає проблемним. Численні дуги створюють заплутану та незрозумілу мережу, яку важко редагувати та змінювати. Ось чому було зроблено багато спроб удосконалити цей формалізм, щоб подолати цей недолік: об'єктно-орієнтована мережа Петрі (місце, перехід і мережа можуть бути об'єктом) [29], мережі в мережах (місце може містити мережу) [30] , ієрархічна мережа Петрі (перехід може містити мережу) [31], розфарбована мережа Петрі (використовуються типи для маркерів) [32]. З наведених спроб тільки мережі в мережах намагались зберегти формалізм стохастичної мережі Петрі і

математичний опис моделі, яка будується. Інші спроби удосконалювали мережу Петрі за рахунок інтеграції з такими технологіями як об’єктно-орієнтоване програмування, функціональне програмування. Проте при цьому зникала найбільша перевага застосування мережі Петрі – математичний формалізм опису моделі.

Петрі-об’єктне моделювання, що розглянуто у наступному розділі, зберігає формалізм опису моделі стохастичною мережею Петрі і, водночас, за рахунок використання об’єктно-орієнтованого програмування розширює можливості застосування стохастичної мережі Петрі для моделювання систем з великою кількістю елементів.

2.1 Метод Петрі-об’єктного моделювання

Метод Петрі-об’єктного моделювання розвивається з 2011 року, коли була опублікована робота [33], в якій були розроблені теоретичні засади цього методу моделювання. Перевагою методу є можливість тиражування елементів моделі з однаковою поведінкою та конструювання моделі з фрагментів. При конструюванні забезпечується, що 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 з’являються сторонні елементи у представленні моделі і не виникає необхідності переходити до багаторівневого представлення моделі.

Технологія Петрі-об’єктного моделювання застосовувалась для моделювання систем, що містять сотні об’єктів, і показала достатньо високу швидкодію як в процесі розробки моделі, так і в процесі експериментування з нею [34]. Проте використання виключно редактора мережі Петрі у програмному забезпеченні 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 відлагодження. Параметризація опису об’єктів та автоматизація перетворення мережі Петрі у код, що були запропоновані у роботі [35], частково вирішили питання зменшення зусиль з розробки мереж Петрі для створення Петрі-об’єктів. Проте через складність розробки зв’язків між Петрі-об’єктами, побудова моделі потребувала значної рутинної роботи програміста, що досконало володіє відповідним програмним забезпеченням.

При побудові складної системи методом Петрі-об’єктного моделювання її розбивають на структурні частини, що взаємодіють між собою, розробляють фрагменти моделі і поступово з’єднують їх. При розробці фрагменту моделі визначають його множину подій та описують правила зміни стану стохастичною мережею Петрі. Для цього кожній елементарній події, що відбувається в системі, ставиться у відповідність перехід мережі Петрі. Передумови події перевіряють по кількості маркерів у вхідних позиціях переходу (тобто таких, що з’єднані дугою в напрямку від позиції до переходу). Як тільки передумови виконані, подія запускається на виконання і цей факт відтворюється відніманням маркерів з вхідних позицій. Післяумови події, які є результатом виконання події, відтворюються додаванням маркерів у вихідні позиції переходу (тобто такі, що з’єднані дугою в напрямку від переходу до позиції). Віднімання та додавання маркерів відбувається у кількості, що дорівнює кратності дуги.

Розроблені фрагменти моделі оформлюють у вигляді Петрі-об’єктів. Петрі-об’єкт за визначенням, введеним у [33], є об’єктом суперкласу, який містить методи для відтворення динаміки об’єкта у відповідності до мережі Петрі, що зберігається у спеціально призначеному для цього полі. При створенні об’єкта конструктор суперкласу використовує мережу Петрі та, можливо, набір параметрів, тому об’єкт можна представити виразом [36]:

$$o = (net, list), \tag{2.7}$$

де o - Петрі-об’єкт,

net - мережа Петрі,

$list$ - список параметрів.

Петрі-об’єктна модель створюється з множини Петрі-об’єктів, динаміка яких взаємопов’язана через спільні позиції. Поєднання Петрі-об’єктів через спільні позиції гарантує, що динаміка моделі визначається мережею Петрі, яка є об’єднанням мереж Петрі-об’єктів [33]. Програмно спільні позиції реалізуються через присвоєння адрес пам’яті відповідних об’єктів-позицій, що можна представити наступним виразом:

$$o_{\sim}.net.p_{\bullet} = o_{,}.net.p_{\mathrm{f}}, \tag{2.8}$$

де $o_{\sim}$, $o_{,}$ – Петрі-об’єкти, які з’єднуються,

$o.net$ – мережа Петрі-об’єкта o ,

$o.net.p$ – позиція мережі Петрі-об’єкта o.

Усі пари спільних позицій двох Петрі-об’єктів, що утворюють з’єднання між ними, будемо називати конектором [36]:

$$connector(o_{\sim}, o_{,}) = \{(o_{\sim}.net.p_{\bullet}, o_{,}.net.p_{\mathrm{f}})\}. \tag{2.9}$$

Модель утворюється з Петрі-об’єктів операцією агрегування (в термінах об’єктно-орієнтованої технології), тому модель можна представити наступним виразом [37]:

$$m \longrightarrow \mathrm{N}o_{\mathrm{Z},\ldots,}o_{\mathrm{T}}\mathrm{Q}, \tag{2.10}$$
$$m.net = \bigcup_{\mathrm{O}} o_{\mathrm{C}}.net,$$

де m – модель,

$\longrightarrow$ – символ операції агрегування,

o_{O} – Петрі-об’єкти, які агрегуються,

$m.net$ – мережа Петрі моделі,

$o.net$ – мережа Петрі-об’єкта o.

Перетворення стану мережі Петрі всієї моделі, як доведено у [33], розпадається на перетворення стану мереж Петрі-об’єктів:

$$D^{\mathrm{Y}}(\mathbf{S}(t_{\mathrm{T}})) = \mathrm{N}D^{\mathrm{Y}}\mathrm{l}\mathbf{S_j}(t_{\mathrm{T}})\mathrm{m}, j = 1, \ldots q\mathrm{Q}, \tag{2.11}$$
$$D^{\cdot}(\mathbf{S}(t_{\mathrm{T}})) = \mathrm{N}D^{\cdot}\mathrm{l}\mathbf{S_j}(t_{\mathrm{T}})\mathrm{m}, j = 1, .. q\mathrm{Q},$$

де q – кількість Петрі-об’єктів, з яких складаєтсья модель.

Рівняння станів Петрі-об'єктної моделі можуть бути записані у вигляді (2.4), де $\mathbf{S}(t_{\mathrm{T}})$ – стан мережі $m.net$ (див. (2.10)). Проте з урахуванням (2.11) рівняння можуть бути переписані у наступному вигляді:

$$\begin{cases} t_{\mathrm{T}} = \min\limits_{<} \mathrm{hmin}_{0}[E_{<}(t_{\mathrm{TYZ}})]_{\mathrm{O}}\mathrm{i}, \\ \mathbf{S_j}(t_{\mathrm{T}}) = (D^{\mathrm{Y}})^{\mathrm{k}\%}\mathcal{D}\cdot\mathrm{l}\mathbf{S}(t_{\mathrm{TYZ}})\mathrm{m},\ m_{\mathrm{C}}\colon \bigvee\limits_{<} I(T, t_{\mathrm{T}}) = 0, \quad j = 1,..q \\ n = 1{,}2\ \ldots \end{cases} \tag{2.12}$$

Виконання перетворення в окремих фрагментах мережі Петрі замість виконання перетворення всієї мережі збільшує швидкодію імітації, оскільки, по-перше, при відтворенні події відбуваються зміни тільки у відповідному Петрі-об'єкті, по-друге, пошук найближчої події відбувається переглядом стану усіх об'єктів моделі замість перегляду стану усіх переходів моделі. Цей факт доведено теоретично та експериментально у публікації [34].

Оскільки при з'єднанні кількох Петрі-об'єктів утворюється знову мережа Петрі, а не структура більш високого рівня, то можуть бути введені операції над Петрі-об'єктами. Операція тиражування *multiply* означає, що мережа Петрі використовується для створення групи Петрі-об'єктів. При цьому списки чисельних параметрів елементів мережі Петрі можуть відрізнятись (окрім кількості маркерів у позиціях, які використовуються для з'єднання з іншими Петрі-об'єктами) [36]:

$$g = multiply(net, lists, k) \Leftrightarrow$$
$$\Leftrightarrow \forall i\colon o_{\check{\mathrm{Z}}} = (net, list_{\check{\mathrm{Z}}}), i = 1..k,$$

де g – група Петрі-об'єктів,

net - мережа Петрі, що використовується для створення групи об'єктів,

$lists = \{list_{\check{\mathrm{Z}}}\}$ - множина списків параметрів, що застосовуються для створення мережі Петрі конкретного об'єкта,

k – кількість створюваних об'єктів.

З'єднання Петрі-об'єкта o з групою g Петрі-об'єктів за правилом, сформульованим для пари Петрі-об'єктів, фактично тиражує зв'язок між Петрі-об'єктом o та кожним Петрі-об'єктом з групи g:

$$g.net.p_{\bullet} = o.net.p_{\mathrm{f}} \Leftrightarrow$$

$$\Leftrightarrow \forall o_{\check{Z}} \in g:\ o_{\check{Z}}.net.p_{\bullet} = o.net.p_{ƒ}.$$

Об’єкти, взаємопов’язані між собою, можуть утворювати колекції $(objects, connectors)$, для яких можна здійснювати тиражування за наступними правилами:

$$f = multiply(objects, connectors, k) \Leftrightarrow$$

$$\forall o \in objects, \forall c \in connectors, \forall u, v \in objects | (u.net.p_{\bullet}, v.net.p_{ƒ}) \in c:$$

$$g' = multiply(o.net, lists, k),$$

$$`\forall j: c_0 = \text{“}(u_0.net.p_{\bullet}, v_0\, net.p_{ƒ})\text{”}\, u_0 \in g_{\sim‰} v_0 \in g_{,‰} \bullet \bullet, j = 1..k,$$

де f – група колекцій Петрі-об’єктів, $objects$ - Петрі-об’єкти, що використовується для створення колекції, $connectors$ – конектори між Петрі-об’єктами, k – кількість операцій тиражування колекції.

Інші об’єкти можуть створювати з’єднання з групою колекцій :

$$f.g_{\sim}.net.p_{\bullet} = o.net.p_{ƒ} \Leftrightarrow$$

$$\Leftrightarrow \forall u \in g_{\sim}: f.u.net.p_{\bullet} = o.net.p_{ƒ},$$

де з’єднання встановлюється між Петрі-об’єктом o та групою Петрі-об’єктів $g_{\sim}$ з колекції.

У [38] розглянуто застосування дворівневої структури графічного представлення Петрі-об’єктної моделі на прикладі моделі телекомунікаційної системи і визначені переваги використання мови візуального програмування. У наступному розділі статті для запропонованої мови даний опис у вигляді формальної граматики та зроблено висновки про її властивості.

2.3 Алгоритм імітації Петрі-об’єктної моделі

Існуючі імітатори стохастичних мереж Петрі реалізують прості і чітко визначені правила функціонування мережі Петрі по-різному. При використанні програмного забезпечення для побудови моделі необхідно враховувати правила, за якими відбувається відтворення функціонування мережі Петрі, інакше результат моделювання може не співпадати з очікуванням. Наприклад, коли маркери можуть бути використані при запуску 2 і більше переходів, в алгоритмі

може бути передбачена можливість запускати кілька переходів одночасно (якщо це дозволяє кількість маркерів в позиціях) або запуск здійснюється по одному з вирішенням конфлікту. Реалізація багатоканального переходу взагалі передбачена тільки в деяких з існуючих програмних засобів і тільки тестуванням можна встановити чи реалізований багатоканальний перехід у мережі Петрі.

Алгоритми імітації відрізняються способом просування часу (з детермінованим кроком чи до найближчої події), способом запуску переходу (з затримкою маркерів в позиціях до моменту завершення часової затримки, з окремою реалізацією подій входу маркерів та виходу маркерів, між якими відбувається часова затримка), способом вирішення конфліктів (з неявним пріоритетом переходів, з урахуванням заданого пріоритету переходів, за заданими ймовірностями, а також з урахуванням як пріоритетів, так і ймовірностей для вирішення конфлікту).

Алгоритм імітації Петрі-об’єктної моделі розроблений у відповідності до рівнянь (2.12) і складається з таких кроків (позначено $Tmod$ – час моделювання, протягом якого виконується імітація, t – поточний момент часу):

1 Встановити поточний момент часу $t = 0$.

2 Доки $t < Tmod$:

визначити момент найближчої події як найменше значення min з усіх моментів виходу з переходів мережі Петрі;

2.1 Якщо $min < Tmod$, то просунути час у момент найближчої події: $t = min$;

2.2 Змінити стан моделі у відповідності до події:

2.2.1 Для кожного Петрі-об’єкта виконати перетворення стану мережі Петрі $D^{:}$, тобто виконати вихід маркерів з усіх переходів, для яких виконана умова виходу маркерів (див. (2.5)).

2.2.2 Доки є переходи з виконаною умовою запуску:

2.2.2.1 З множини об’єктів, в яких є переходи з виконаною умовою запуску, вибрати один (рівномірно серед усіх об’єктів з найвищим пріоритетом).

2.2.2.2 Для обраного об’єкта виконати перетворення стану мережі Петрі D^{Y}(див. (2.6)), тобто виконати вхід маркерів в один перехід, який обраний серед множини переходів, для яких виконана умова входу маркерів, рівноймовірно або за заданою ймовірністю запуску переходу серед всіх переходів з найвищим пріоритетом перехід.

3 Вивести стан моделі та статистичні результати моделювання.

У бібліотеці java-класів Петрі-об’єктного моделювання PetriObjLib алгоритм імітації реалізує метод `go()` класу `PetriObjModel`. Перетворення $D^{:}$ та D^{Y} виконуються методами `input()` та `output()` класу `PetriSim`, який є суперкласом Петрі-об’єктів. Розроблена у [38] бібліотека класів уточнена процедурою вибору об’єкта для входу маркерів у перехід та повторенням процедури вибору об’єкта після кожного входу маркерів у перехід. Щоб скоротити час на вибір об’єкта корисно встановлювати більший пріоритет для об’єктів (та переходів), які майже завжди знаходяться в конфлікті, щоб зробити меншою кількість об’єктів, серед яких здійснюється вибір.

Оскільки алгоритм імітації стохастичної мережі Петрі є універсальним, то зібрана модель одразу може запускатись на імітацію та виконання експериментальних досліджень, а зусилля, витрачені на побудову моделі, будуть компенсовані зменшенням витрат на написання та відлагодження алгоритму імітації.

2.4 Висновки до розділу 2

У розділі наведені відомості з теорії стохастичних мереж Петрі та основні теоретичні положення Петрі-об’єктного моделювання. Формальний опис Петрі-об’єктної моделі розвинутий за рахунок введення понять конектора Петрі-об’єктів, групи Петрі-об’єктів, колеції Петрі-об’єктів. Введені поняття відкривають можливість тиражування зв’язків при конструюванні Петрі-об’єктної моделі.

Петрі-об’єктний підхід до розробки імітаційних моделей дискретно-подійних систем комбінує переваги використання стохастичних мереж Петрі та об’єктно-орієнтованої технології для опису моделей. Наявність суперкласу, до якого належать Петрі-об’єкти, гарантує, що усі об’єкти будуть відтворювати функціонування за однаковими правилами стохастичної мережі Петрі з багатоканальними та конфліктними переходами. При створенні Петрі-об’єкта один раз розроблена мережа Петрі може використовуватись для створення великої кількості однотипних об’єктів, тобто тиражуватись. При цьому, список параметрів мережі Петрі може бути унікальним при створеннні кожного тиражованого об’єкта.

Петрі-об’єктні моделі мають такі переваги при розробці моделей: універсальність формального опису динаміки дискретно-подійної системи; тиражування елементів моделі, динаміка яких визначена мережею Петрі; більша швидкодія у порівнянні зі звичайною стохастичною мережею Петрі. Проте відсутність візуального представлення моделі робить процес розробки моделі складним через рутинні та схильні до помилок операції кодування зв’язків між елементами.

3 ГРАМАТИКА МОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ ПЕТРІ-ОБ'ЄКТНИХ МОДЕЛЕЙ

3.1 Формальні мови та засоби імітаційного моделювання дискретно-подійних систем

З появою формальних граматик з'явились машино-незалежні мови програмування. Першою мовою програмування, побудованою на основі формальної граматики, стала мова ALGOL [39]. Мова імітаційного моделювання Simula, побудована на основі мови ALGOL, була задумана розробниками спочатку виключно як мова для цілей імітаційного моделювання дискретно-подійних систем, а стала першою об'єктно-орієнтованою мовою [40]. Її версія Simula 67 була написана вже як універсальна об'єктно-орієнтована мова програмування. На мові Simula було написано провідне на свій час програмне забезпечення Arena. Використання блочних діаграм для побудови моделей і надзвичайно зручна розбудова ієрархічних моделей – основні переваги цього програмного забезпечення. Найбільш популярне в наші дні програмне забезпечення з імітаційного моделювання Simio, по суті, є продовженням концепції Arena з більш розвинутими засобами анімації, у тому числі 3D, та можливостями завдання розкладів [41].

Вказані програмні засоби та багато інших (AnyLogic, VisualSim, Simul8) ґрунтуються на описі дискретно-подійних систем в термінах блочних діаграм, що описують процес обробки об'єктів. Складні моделі зі специфічною взаємодією елементів важко вкласти у логіку спрацьовування достатньо обмеженого набору блоків, тому кількість блоків у бібліотеках постійно зростає, що негативно впливає на швидкість вивчення програмного забезпечення. Окрім того, логіка блочних діаграм завжди спирається на опис процесу обробки об'єктів, при цьому управління ресурсами для обробки обмежено тільки їх кількістю, місткістю та розкладом. Деякі з існуючих програмних продуктів (Simio, Simul8) визнали обмеженість блочної побудови моделей і надають можливість розробникам створювати власні фрагменти коду і вбудовувати їх в

моделі. Наприклад, Simio пропонує додавати написані на .NET фрагменти коду для передачі даних, опису специфічних алгоритмів вибору об’єктів з черг, опису серії експериментів [42]. Simula8 надає можливість додаткового опису функціональності блоків мовою Visual Logic[43].

Іншим підходом до побудови симуляції є розробка формалізації у вигляді мережі Петрі і запуск універсального алгоритму імітації для побудованої мережі. Програмне забезпечення CPNTools [10] використовує формалізацію опису моделі у вигляді ієрархічних розфарбованих мереж Петрі та функціональну мову CPN ML для опису даних, змінних та функцій. Мова CPN ML є розширенням мови функціонального програмування Standard ML. Використання функціональної мови програмування надає можливість CPNTools досягти необхідної гнучкості в побудові моделей, проте значно ускладнює сприйняття моделі з графічного представлення, оскільки, окрім графічних об’єктів та їх параметрів, опис моделі містить вирази-функції. Якщо CPN ґрунтується на математичному формалізмі розфарбованих мереж Петрі, то при додаванні до опису моделі фрагментів коду функціональною мовою математичний формалізм руйнується [44].

Отже, використання програмних засобів імітаційного моделювання свідчить про переваги використання графічного представлення моделі, такі як наочність представлення, швидкість розробки, зменшення кількості помилок при відтворенні структури. Водночас, забезпечити необхідну гнучкість розробки моделей не вдається без надання розробнику доступу до програмного коду моделі та/або надання дозволу вбудовувати фрагменти програмного коду у код моделі.

3.2 Поняття мови візуального програмування

Терміни «візуальне моделювання» та «візуальне програмування» суттєво відрізняються. Візуальне моделювання - це застосування графічних об’єктів для створення моделей. У загальному випадку візуальні моделі не призначені для обчислень. Наприклад, моделі UML використовують для опису програмного

забезпечення, що проектується. Візуальне програмування – це застосування графічних об’єктів для розробки програмного коду, який реалізує певний алгоритм обчислень. Відповідно, існує суттєва різниця між мовою візуального моделювання та мовою візуального програмування. Мова візуального моделювання – це набір графічних об’єктів для представлення систем, об’єктів або понять, що проектуються або існують. Мова візуального програмування – це набір взаємопов’язаних графічних об’єктів (разом з чисельними параметрами), який однозначно може бути трансформований у програмний код.

На відміну від природніх мов, мови програмування є формальними, тобто породжуються формальними граматиками. Опис будь-якої формальної граматики складається з алфавіту термінальних символів, алфавіту нетермінальних символів, початкового символу граматики та правил виведення, що використовують для розпізнавання наборів символів. Алфавіти термінальних та нетермінальних символів разом утворюють алфавіт мови. Граматика мови програмування породжує множини слів (виведені з термінальних символів за правилами виведення), що можуть бути інтерпретовані (перетворені) в модель обчислень. Приклади розробки формальних граматик, що орієнтовані на використання упорядкованих наборів символів, викладені у [45]. Математична теорія формальних граматик викладена у [46]. Формальне представлення граматики мови важливе для гарантування однозначності інтерпретації мовного виразу та однозначного перетворення його у алгоритм (перелік інструкцій для обчислень), що запускається на виконання.

Окрім алфавіту, неодмінними ознаками формальної мови є синтаксис мови та її семантика. Синтаксис визначає дозволені конструкції мови, семантика - зміст цих конструкцій. Синтаксис визначає набір правил, за якими складаються мовні конструкції. Розбір мовного виразу, складеного за синтаксичними правилами, виконується компілятором мови у відповідності до правил виведення (продукцій), що задані граматикою мови. Семантика – це зв’язок між синтаксисом мови та моделлю обчислення. Формальна семантика задається математичною моделлю, що описує обчислення у відповідності до виразу,

заданого мовою. Синтаксичні помилки виявляються автоматизовано під час компіляції програмного коду, а семантичні – під час виконання програми.

Алфавіт мови візуального програмування – це набір графічних елементів, що виконують роль символів. У текстовій мові програмування слово (лексема) – це послідовність символів, утворена за правилами, що визначені синтаксисом мови. Розташування символів у послідовності зліва направо означає фактично їх з'єднання, і це з'єднання інтерпретується граматикою (при розборі виразу). У візуальній мові, що використовує графічне представлення для опису мовного виразу, на розташування елементів у графічному просторі не накладається ніяких обмежень, тобто послідовності символів у тому розумінні, як вони присутні у традиційних мовах програмування, відсутні. Замість цього у графічному представленні встановлюються зв'язки між елементами. Наприклад, якщо кінець дуги співпадає з місцем розташування вершини графу, то елемент дуга поєднаний з елементом вершина. Якщо такі зв'язки можуть утворювати фрагменти, які однозначно інтерпретуються як лексеми мови, то графічне представлення може використовуватись для візуального представлення виразів програмного коду.

У випадку текстової мови програмування текст перетворюється в алгоритм виконання обчислень за правилами, що визначені семантикою мови. У мові візуального програмування зображення, складене з взаємопов'язаних графічних елементів, трансформується у змістові конструкції, що запускаються на обчислення. Опис графічного елементу може супроводжуватись набором чисельних параметрів, що впливають на обчислення. Отже, виразом мови візуального програмування є набір взаємопов'язаних графічних об'єктів, що може бути запущений на обчислення.

Граматика текстової мови породжує послідовності символів термінального алфавіту, виведених з початкового символу. Граматика мови візуального програмування має породжувати графічні зображення, утворені з графічних елементів, що складають алфавіт термінальних символів мови. Далі, на прикладі мови візуального програмування Петрі-об'єктних моделей, буде

показано, що мова візуального програмування може бути визначена формальною граматикою, так само, як і текстова мова програмування, за умови, що визначені зв'язки між графічними елементами.

3.3 Граматика мови візуального програмування Петрі-об'єктних моделей

Для текстових мов програмування теорія формальних мов розроблена достатньо детально [47, 48]. При 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 у відповідності до розповсюдження подій. Основною змістовою конструкцією, що черпається 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.

Мова візуального програмування визначає набори графічних елементів (символів), що можуть бути перетворені у змістові конструкції Петрі-об'єктної моделі. Графічні елементи, передбачені для конструювання Петрі-об'єктної моделі, є символами мови. Набори елементів, які можуть бути перетворені в імітаційні моделі і запущені на виконання – це лексеми (слова) мови. Мережі Петрі-об'єктів – це лексеми в граматиці. Вирази мови – це фрагменти Петрі-об'єктної моделі.

Введемо алфавіт термінальних символів для опису мережі Петрі: p - позиція, t - перехід, a – дуга. Введемо алфавіт нетермінальних символів: E - комбінація термінальних символів, що задає одну вхідну дугу переходу, Q - комбінація термінальних символів, що задає одну вихідну дугу переходу, N -

комбінація кількох нетермінальних символів E та Q, що може бути інтерпретована в мережу Петрі. Символом I позначимо початковий символ граматики. Вважатимемо, що дуга, яка поєднує елементи позиція та перехід, визначає відношення «попередній – наступний» у послідовності символів, що задає лексему мови програмування. Правила виведення граматики для введених символів представлені у таблиці 3.1. Правила визначають, що усі набори символів інтерпретуються як трійки нетермінальних елементів pat та tap. У найпримітивнішому випадку мережа Петрі складається з однієї позиції (без переходів). Мережа Петрі може містити позиції, які не з'єднані з переходами (такі позиції зберігають інформацію про стан об'єкта, а зміни в такому стані можуть відбутись тільки в результаті подій, що відбуваються в інших Петрі-об'єктах).

Таблиця 3.1 – Правила виведення 1-6 граматики (синтаксис мови) [36]

Правило	Зміст правила
1. $I \longrightarrow NI \mid N$	Для визначення Петрі-об'єктів має бути визначена одна або кілька мереж Петрі.
2. $N \longrightarrow EQ \mid QE$	Якщо вказана вхідна дуга, то обов'язково має бути вихідна. Якщо вказана вихідна дуга, то обов'язково має бути вхідна.
3. $N \longrightarrow EN \mid QN$	Може бути задано будь-скільки вхідних та вихідних дуг у довільному порядку.
4. $E \longrightarrow pat$	Трійка термінальних символів, що визначають вхідну дугу переходу
5. $Q \longrightarrow tap$	Трійка термінальних символів, що визначають вихідну дугу переходу
6. $N \longrightarrow p \mid pN$	Одна або декілька позицій в мережі Петрі

Для опису Петрі-об'єктної моделі доповнимо алфавіт термінальних символів граматики такими символами: o - Петрі-об'єкт, g - група Петрі-об'єктів, f – група колекцій Петрі-об'єктів, s – позиція Петрі-об'єкта, що

використовується для з'єднання з іншими Петрі-об'єктами, l – ототожнення позицій Петрі-об'єктів. Вважатимемо, що конектор, який поєднує Петрі-об'єкт з іншим Петрі-об'єктом або Петрі-об'єкт з групою Петрі-об'єктів, визначає відношення «попередній-наступний» у послідовності символів, що задає лексему мови програмування. Доповнимо алфавіт нетермінальних символів граматики такими символами: 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 ототожнень позицій Петрі-об'єкта з позиціями іншого Петрі-об'єкта. Правила виведення 7-12 граматики (для введених символів) представлені у таблиці 3.2.

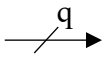
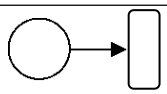
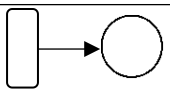
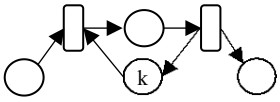
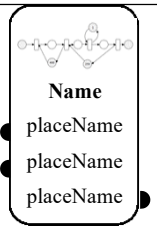
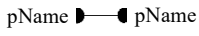
Таблиця 3.2 – Правила виведення 7-12 граматики (синтаксис мови) [36].

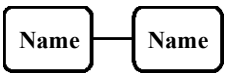
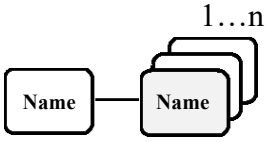
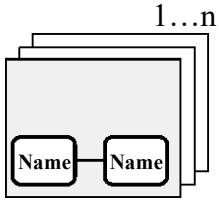
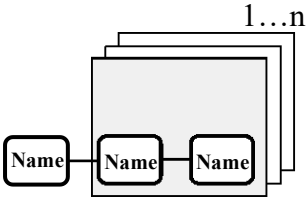
Правило	Зміст правила
7. $I \longrightarrow WI \mid YI \mid XI$	Модель складається з трійок елементів, що визначають зв'язки між двома Петрі-об'єктами, між Петрі-об'єктом і групою Петрі- об'єктів або групою колекцій Петрі-об'єктів, та мереж Петрі для створення Петрі-об'єктів (див. правило 1)
8. $Y \longrightarrow oCg \mid g$	Трійка символів, що визначають зв'язок Петрі-об'єкта з групою Петрі-об'єктів. Ототожнення позицій, що вказані в конекторі, тиражуються для всіх Петрі-об'єктів групи. Допускається конструювання моделі з групи Петрі-об'єктів, що не мають зв'язків з іншими Петрі-об'єктами.

Правило	Зміст правила
9. $X \longrightarrow oCf \mid f$	Трійка символів, що визначають зв'язок Петрі-об'єкта з групою Петрі-об'єктів, що входить до складу колекції. Ототожнення позицій, що вказані в конекторі, тиражуються для всіх Петрі-об'єктів групи. Допускається конструювання моделі з групи колекцій Петрі-об'єктів, що не мають зв'язків з іншими Петрі-об'єктами.
10. $W \longrightarrow oCo \mid o$	Трійка символів, що визначають зв'язок Петрі-об'єкта з іншим Петрі-об'єктом. Допускається конструювання моделі з одного Петрі-об'єкта, що не має зв'язків з іншими Петрі-об'єктами.
11. $I \longrightarrow oI \mid gI$	Один Петрі-об'єкт, або група Петрі-об'єктів в моделі, або кілька Петрі-об'єктів без зв'язків, або кілька груп Петрі-об'єктів без зв'язків.
12. $\mathrm{C} \longrightarrow slsC \mid sls$	Трійка символів, що визначають ототожнення позиції одного Петрі-об'єкта з позицією іншого Петрі-об'єкта. Конектор складається з одного або кількох ототожнювань позицій двох Петрі-об'єктів.

Графічні зображення символів мови представлені у таблиці 3.3. Параметри елементів, які представлені символами мови, мають у графічному представленні спеціально відведені поля. Параметр Name призначений для введення назви відповідного елементу.

Таблиця 3.3 - Графічне зображення символів алфавіту мови візуального програмування Петрі-об’єктних моделей [36].

Графічне представлення символу	Скорочене позначення символу	Інтерпретація
	*p*	Позиція мережі Петрі з кількістю маркерів *k.*
 *d*	*t*	Перехід мережі Петрі з часовою затримкою *d*, зі значенням пріоритету *r* (ширина прямокутника розраховується за формулою $w \cdot (1 + 0{,}5^{f})$, де w – значення ширини, що відповідає пріоритету 1).
	*a*	Дуга з кількістю зв’язків *q*. Значення *q*=1 є таким, що приймається за замовчування.
	*E*	Вхідна дуга переходу (з позиції)
	*Q*	Вихідна дуга переходу (у позицію)
	*N*	Фрагмент мережі Петрі, що складається з трьох вхідних дуг та трьох вихідних дуг: EEQEQQ.
	*o*	Петрі-об’єкт з вказівниками на позиції, що можуть використовуватись для з’єднання з іншими Петрі-об’єктами.
	s	Вказівник на позицію Петрі-об’єкта, що використовується для з’єднання
	*sls*	Ототожнення позиції Петрі-об’єкта з позицією іншого Петрі-об’єкта
	C	Конектор

Графічне представлення символу	Скорочене позначення символу	Інтерпретація
	*W*	Спрощене представлення Петрі-об'єкта та з'єднання двох Петрі-об'єктів
	*g*	Група *n* Петрі-об'єктів, створених за однаковим шаблоном Петрі-об'єкта з назвою Name
	*Y*	З'єднання Петрі-об'єкта з групою *n* об'єктів
	*f*	Група *n* колекцій Петрі-об'єктів, створених за однаковим шаблоном. По суті, фрагмент моделі тиражується у кількості *n*.
	*X*	Група *n* колекцій Петрі-об'єктів, створених за однаковим шаблоном. По суті, це фрагмент моделі, що тиражується у кількості *n*.

Правила виведення 1-12 визначають граматику мови візуального програмування Петрі-об'єктних моделей. Наведена граматика є контекстно-вільною або граматикою типу 2 за ієрархією Хомського, оскільки інтерпретація будь-якого нетермінального символу не залежить від символів, які його оточують. Приклад виведення за визначеною граматикою (породження виразу з застосуванням послідовності правил 7, 10, 12, 7, 8, 12, 1, 3, 5, 2, 4, 5, 1, 3, 5, 3, 4, 2, 4, 5):

$$I \longrightarrow WI \longrightarrow oCoI \longrightarrow oslsoI \longrightarrow oslsoYI \longrightarrow$$
$$\longrightarrow oslsooCgI \longrightarrow oslsooslsgI \longrightarrow$$
$$\longrightarrow oslsooslsgNI \longrightarrow oslsooslsgQNI \longrightarrow$$
$$\longrightarrow oslsooslsgtapNI \longrightarrow$$

$$\longrightarrow slsooslsgtapEQI \longrightarrow$$
$$\longrightarrow ooslsooslsgtappatQI \longrightarrow$$
$$\longrightarrow oslsooslsgtappattapI \longrightarrow$$
$$\longrightarrow oslsooslsgtappattapI \longrightarrow$$
$$\longrightarrow oslsooslsgtappattapN \longrightarrow$$
$$\longrightarrow oslsooslsgtappattaptapQN \longrightarrow$$
$$\longrightarrow oslsooslsgtappattaptapN \longrightarrow$$
$$\longrightarrow oslsooslsgtappattaptapEN \longrightarrow$$
$$\longrightarrow oslsooslsgtappattaptappatN \longrightarrow$$
$$\longrightarrow oslsooslsgtappattaptappatEQ \longrightarrow$$
$$\longrightarrow oslsooslsgtappattaptappatpatQ \longrightarrow$$
$$\longrightarrow oslsooslsgtappattaptappatpattap.$$

Виведення може бути також представлено у вигляді дерева виведення (рис. 3.1). Наведений ланцюжок відповідає Петрі-об'єктній моделі, що складається з двох Петрі-об'єктів одного типу та групи Петрі-об'єктів іншого типу. Мережа Петрі об'єктів першого типу складається з однієї вхідної дуги та двох вихідних дуг з відповідно визначеними для них позиціями та переходами, а мережа Петрі другого типу складається з двох вхідних дуг та двох вихідних дуг.

При створенні елементів у графічному середовищі елементи одного типу зберігаються в масиві у порядку їх створення розробником моделі. Враховуючи однозначність послідовності елементів у масивах, очевидно, що існує тільки одне дерево виведення для будь-якого візуального представлення моделі. Оскільки існує тільки одне дерево виведення для візуального представлення моделі, то введена граматика є однозначною.

Алгоритми пошуку непродуктивних символів та недосяжних символів наведені у монографії [45]. Складаємо список нетермінальних символів, які можуть бути виведені з термінальних символів: E, Q, N, C, Y, I. Оскільки усі нетермінальні символи алфавіту потрапили у цей список, то непродуктивних (зайвих) символів немає. Складаємо список нетермінальних символів, які можуть бути виведені з початкового символу: I, W, Y, N, C, E, Q. Оскільки у

список потрапили усі нетермінальні символи граматики, то недосяжних символів у визначеній граматиці немає. Ланцюгових правил серед правил 1-11 немає. Отже, граматика, що визначена, є приведеною.

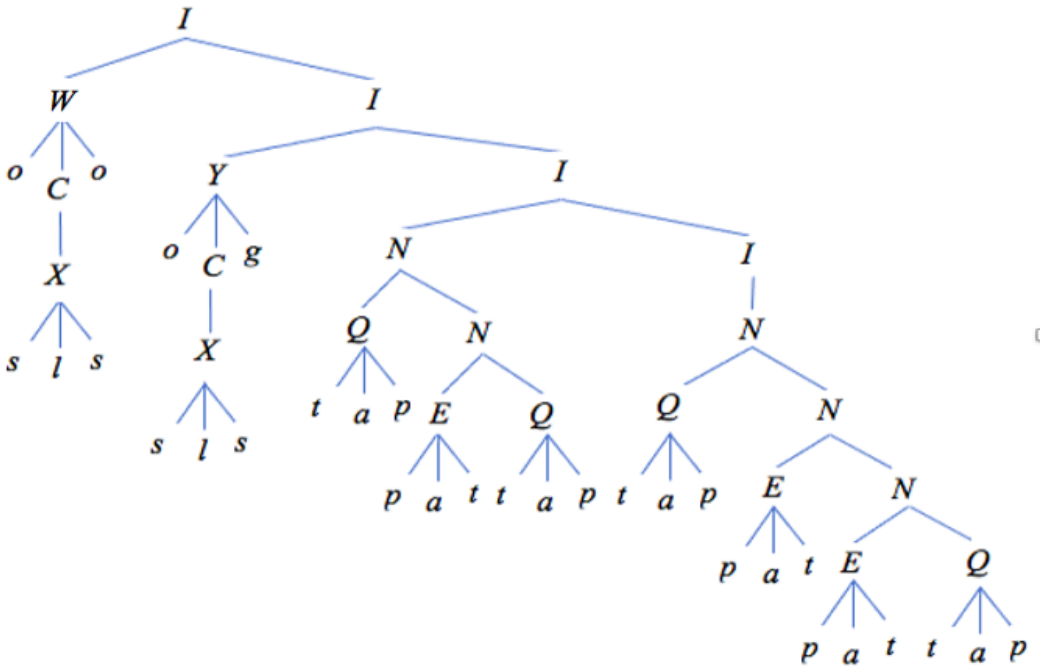


Рисунок 3.1 – Дерево виведення для моделі, представленої мовою візуального програмування.

Форма Бекуса-Наура – це інший спосіб введення граматики, який використовують, коли символів надто багато і правил виведення відповідно надто багато. Очевидно, що правила граматики 1-11 можна описати у вигляді розширеної нотації Бекуса-Наура. У статті [38] наведений приклад опису граматики мови візуального програмування у вигляді нотації Бекуса-Наура.

Петрі-об’єктна-модель, описана візуальною мовою програмування, трансформується в алгоритмічну мову і запускається на виконання алгоритмом імітації. Правила перетворення візуальної моделі у модель обчислень визначають семантику мови і реалізуються транслятором мови, який представлений у розділі 4 цієї дисертації.

3.4 Приклад представлення моделі мовою візуального програмування

Наведемо приклад розробки моделі з використанням розробленої мови візуального програмування. Побудуємо імітаційну модель, що відтворює обробку запитів клієнт-серверним застосунком. Запити надходять від користувачів двох типів: «авторів» та «гостей». Кожний тип користувача має свою логіку роботи з клієнт-мерверним застосунком. Користувачі-автори, на відміну від користувачів-гостей, можуть додавати та змінювати інформаційний контент.

Загальною проблемою таких систем може бути довготривала обробка запиту через очікування доступу до бази даних та обробку даних серверним застосунком. Імітація системи на моделі може допомогти розробникам визначитись з параметрами обчислювальних ресурсів, які забезпечують прийнятне значення очікування відповіді на запит, та з вимогами до швидкодії окремих процесів обробки запитів.

Фрагмент моделі, що містить обробку запитів, які надходять від користувачів-гостей, представлений на рисунку 3.2. Мережа Server імітує обробку запитів на сервері. Оскільки запити надходять на пошук інформації або на перегляд, то використовуються два Петрі-об’єкти, створені з мережею Server, зі спільними позиціями-ресурсами Limit queue, Threads, DB access.

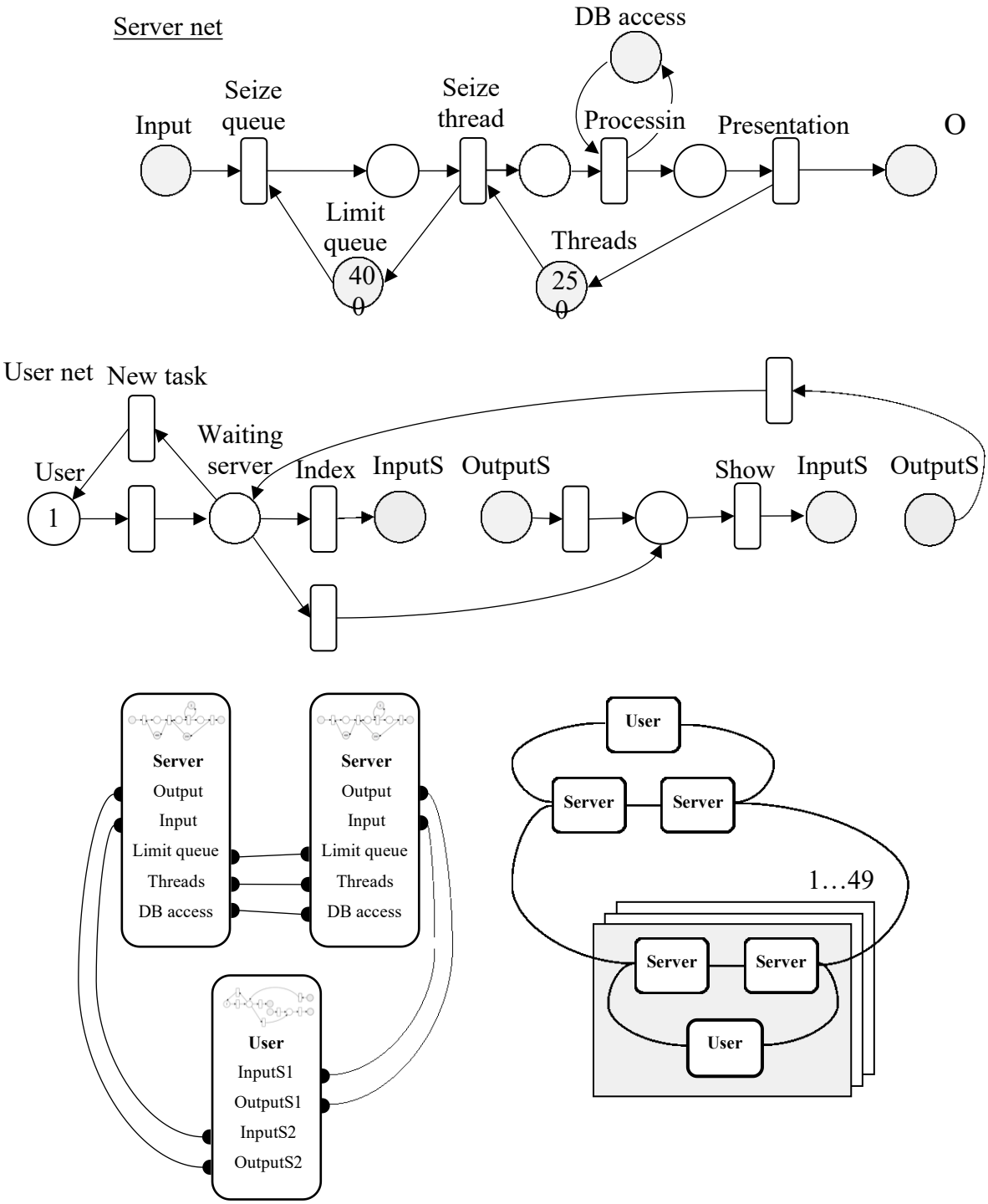


Рисунок 3.2 - Фрагмент Петрі-об’єктної моделі інформаційної системи, розроблений мовою візуального програмування

Використання двох Петрі-об’єктів (замість одного) дозволяє відслідковувати обробку різних видів запитів окремо, а також використовувати в кожному фрагменті часові затримки, характерні для обробки відповідного запиту. Позиції мереж Петрі, що можуть використовуватись для з’єднання з

іншими Петрі-об’єктами, утворюють список позицій-сокетів Петрі-об’єкта. Такі позиції відмічені сірим кольором на зображенні мережі Петрі (див. рис. 3.2) та представлені у списку доступних для з’єднання позицій у зображенні Петрі-об’єкта (див. рис. 3.2). Три з’єднаних Петрі-об’єкти User, Server, Server утворюють фрагмент моделі, що імітує обробку запитів одного користувача. Щоб створити 50 користувачів, побудований фрагмент тиражуємо та з’єднуємо утворену групу колекцій з Петрі-об’єктами Server конектором Server-Server.

Модель, побудована з графічних елементів, перетворюється в обчислення алгоритмом імітаційного моделювання стохастичної мережі Петрі. Результати імітації Петрі-об’єктної моделі інформаційної системи наведені у публікації [37]. Дослідження моделі надає можливість дослідити вплив таких як, кількість потоків та співвідношення часу рендерінгу зображення та часу обробки запиту, на загальний час обробки запиту в інформаційній системі.

3.5 Висновки до розділу 3

Розроблена формальна граматика мови Петрі-об’єктного моделювання. Алфавіт мови складається з графічних елементів візуального представлення, а дозволені мовою набори графічних елементів утворюють лексеми мови. Синтаксис мови визначений правилами виведення граматики, в основі яких правила утворення триплетів елементів. Для розробленої контекстно-вільної граматики встановлено, що вона є однозначною і приведеною. Наведений приклад розробки моделі інформаційної системи розробленою мовою демонструє переваги застосування тиражування зв’язків як в описі моделі, так і при її конструюванні.

4 ТРАНСЛЯТОР МОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ ПЕТРІ-ОБ'ЄКТНИХ МОДЕЛЕЙ

4.1 Семантика мови візуального програмування Петрі-об'єктних моделей

Мова візуального програмування Петрі-об'єктних моделей реалізована у вигляді веб-застосування. Сучасні веб технології надають гнучкість 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 розроблений 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 та 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 4.1 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.

Таблиця 4.1 – Текстова інтерпретація символів алфавіту мови візуального програмування Петрі-об'єктних моделей мовою TypeScript [49]

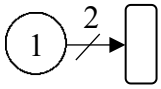
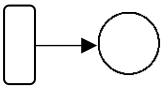
Графічне представлення символу	Скорочене позначення символу	Інтерпретація мовою TypeScript

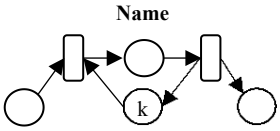
Name
k
	p	`type p = {`
		`  id: number`
		`  name: string`
		`  mark: number`
		`}`

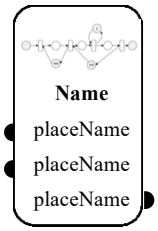
Name
d, r, v, f
	t	`type t = {`
		`  id: number`
		`  name: string`
		`  distribution: string`
		`  delay: number`
		`  deviation: number`
		`  probability: number`
		`  priority: number`
		`}`

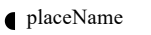
Графічне представлення символу	Скорочене позначення символу	Інтерпретація мовою TypeScript

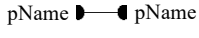
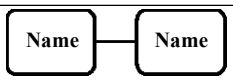
	*a*	`type a = {`
		`  id: number`
		`  name: string`
		`  quantity: number`
		`  placeId: number`
		`  transitionId: number`
		`  info: boolean`
		`}`

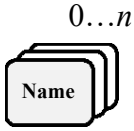
	*E*	`type E = [p, a, t]`
	*Q*	`type Q = [t, a, p]`

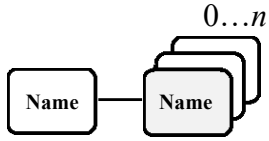
	*N*	`type N = {`
		`  id: number`
		`  name: string`
		`  listE: Array<E>`
		`  listQ: Array<Q>`
		`  sharedPlaces: Array<number>`
		`}`

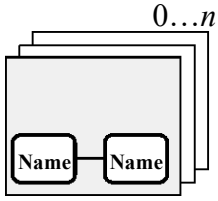
	*o*	`type o = {`
		`  id: number`
		`  name: string`
		`  petriNetId: number`
		`}`

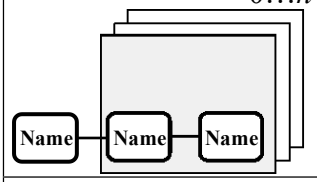
	*s*	`type s = {`
		`  petriObjectId: number`
		`  placeId: number`
		`}`

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

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

Графічне представлення символу	Скорочене позначення символу	Інтерпретація мовою TypeScript
	*Y*	`type Y = [g, C, o]`

	*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>`
		`}`

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 при конструюванні моделі 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'єкту 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. У такому випадку усі три позиції вказуватимуть на позицію P_a (рис. 4.1). Саме тому при приєднанні 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 усіх об’єктів групи і всі об’єкти будуть отримувати інформацію від одного об’єкта. Отримуємо зв’язок типу one-to-many.

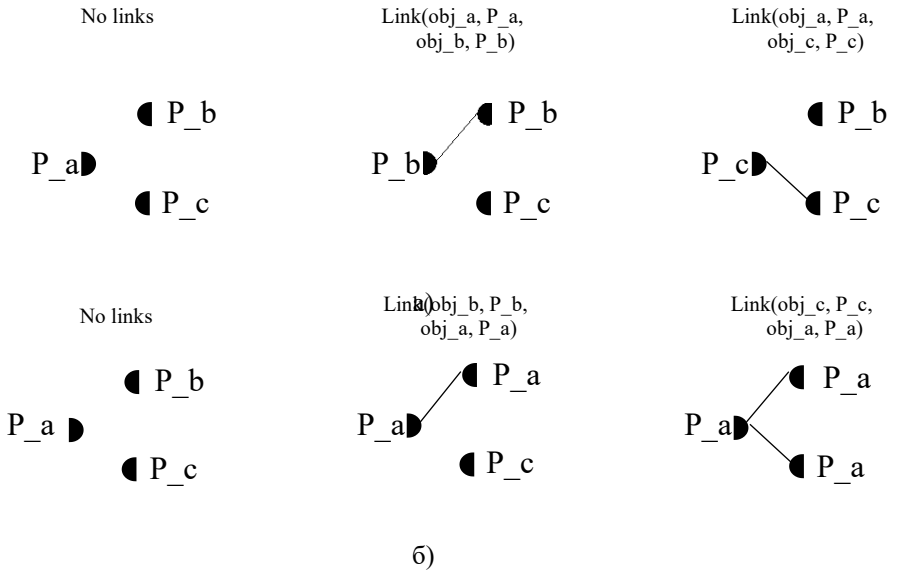


Рисунок 4.1 – Встановлення зв’язків між позиціями Петрі-об’єктів:
а) послідовність, яка призводить до знищення попереднього зв’язку,
б) послідовність, яка призводить до зв’язку one-to-many. [49]

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 [50]. Фінальний етап включає в себе запуск процесу імітації викликом методу `go(t)` класу `PetriObjModel`, аргумент `t` якого вказує на час виконання імітації.

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’єкту (доки не буде досягнута позиція-ототожнювач) перевірка умов виконання події та здійснення події відбувається переглядом тільки елементів одного Петрі-об’єкту замість перегляду всієї мережі Петрі [34].

Імітація моделі здійснюється виконанням таких дій [51]:

- виконати вхід маркерів 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.

Таким чином, перетворення відбувається у такі 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.

4.2 Архітектура траслятора

Архітектура будь-якого транслятора мови програмування складається з послідовності програмних компонент, що виконують сканування послідовності символів, парсинг мовного виразу, семантичний аналіз та генератора коду [52].

Сканування послідовності символів має на меті виявлення лексичних структур мови. Так само, як програма написана на текстовій мові програмування складається з послідовності текстових символів, програма, написана на візуальній мові програмування, складається з послідовності візуальних символів. Послідовність візуальних символів також потрібно розпізнати, виявити синтаксичні конструкції, перевірити на наявність помилок та перетворити в обчислення. Виявлення синтаксичних конструкцій та побудова синтаксичного дерева відбувається на етапі парсингу мовного виразу. Перевірка на наявність помилок – задача семантичного аналізу. Перетворення в обчислення включає в себе фінальне перетворення в програмний код нижчого рівня та запуск на виконання обчислень.

Оскільки візуальна мова програмування реалізована як веб застосування, роботу транслятора розподілена між клієнтською частиною та серверною серверній. Загальна структура транслятора представлена на рисунку 4.1.

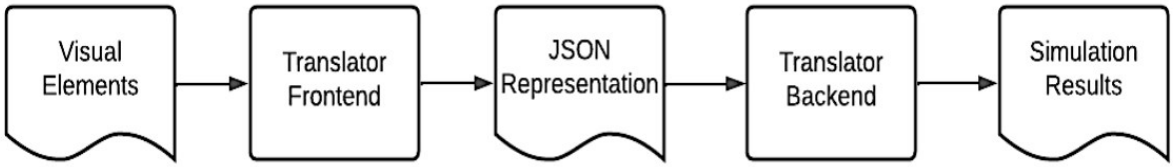


Рисунок 4.1 – Сема роботи транслятора

ake

На вхід транслятора подаються візуальні елементи створені користувачем у відповідності до алфавіту візуальної мови.

В процесі створення візуальних елементів здійснюється перевірка на відповідність правилам виведення граматики. Сканування послідовності символів відбувається в момент створення символу «дуга», що з’єднує позицію з переходом або перехід з позицією. На наступному кроці сканована

послідовність підлягає перевірці на коректність синтаксису. У випадку якщо користувач намагається утворити некоректну синтаксичну конструкцію, наприклад, провести дугу з позиції до іншої позиції, синтаксичний аналізатор поверне помилку, а дана синтаксична конструкція утворена не буде.

Після синтаксичної перевірки створені візуальні елементи розпізнаються як трійки елементів. Трійки елементів утворюють синтаксичне дерево моделі, що має бути перетворено у формат JSON для подальшої обробки серверною частиною транслятора. Перетворення синтаксичного дерева у формат JSON відбувається у відповідності до таблиці 4.1. Структура клієнтської частини транслятора наведена на рисунку 4.2.

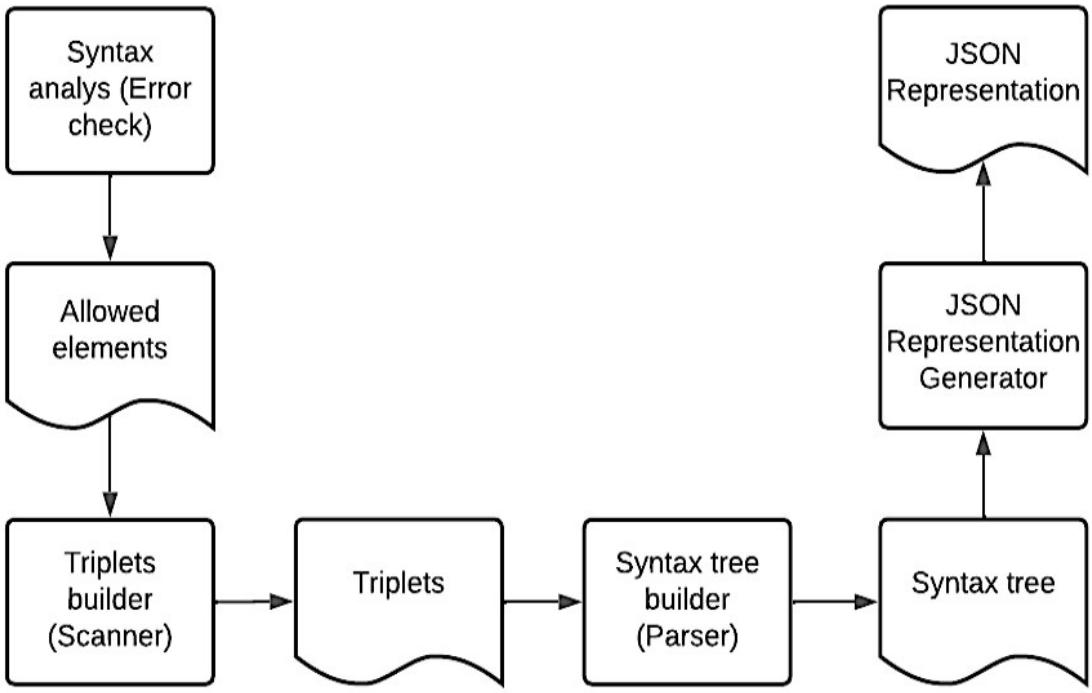


Рисунок 4.2 – Клієнтська частина транслятора

Серверна частина транслятора призначена для перетворення синтаксичного дерева в обчислення. Схема роботи серверної частини транслятора представлена на рисунку 4.3. Текстова інтерпретація моделі у форматі JSON потрапляє на сервер, де відбувається парсинг та створення об’єкта класу PetriObjModel. Java-бібліотека Jackson має вбудований клас ObjectMapper, що відповідає за парсинг JSON та створення відповідних екземплярів класів.

Важливо щоб структура JSON була аналогічною структурі класів бібліотеки PetriObjLib, оскільки ObjectMapper робить обхід полів JSON та шукає відповідні методи *get/set* в класі для того, щоб заповнити об'єкт даними. Таким чином текстова інтерпретація моделі перетворюється в Java-код, що виконує побудову імітаційної моделі, запуск процесу імітації та отримання результатів. Перетворення JSON в Java-код відбувається у відповідності до таблиці 4.2.

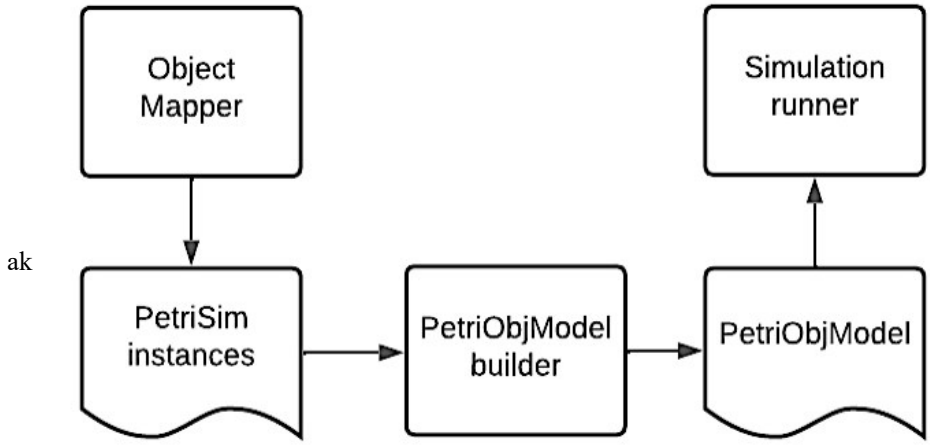


Рисунок 4.3 – Серверна частина транслятора

Таблиця 4.2 – Генерація Java-коду відповідно до текстової інтерпретації моделі.

Інтерпретація мовою TypeScript	Інтерпретація мовою Java

`type p = {`
`  id: number`
`  name: string`
`  mark: number`
`}`
	`listP.add(`
	`  new Place(`
	`  id, name, mark`
	`  )`
	`);`

`type t = {`
`  id: number`
`  name: string`
`  distribution: string`
`  delay: number`
`  deviation: number`
`  probability: number`
`  priority: number`
`}`
	`listT.add(`
	`  new Transition(`
	`  id,name, distribution,`
	`  delay, deviation,`
	`  probability, priority`
	`  )`
	`);`

Інтерпретація мовою TypeScript	Інтерпретація мовою Java

type a = {
id: number
quantity: number
info: boolean
}
	new Arc(id, quantity)

type E = [p, a, t]	listE.add(new E(p, a, t));
type Q = [t, a, p]	listQ.add(new Q(t, a, p));

type N = {
id: number
name: string
listE: Array<E>
listQ: Array<Q>
sharedP: Array<number>
}
	listN.add(
	new Net(
	id, name,listE, listQ, sharedP
	)
	);

type o = {
id: number
name: string
netId: number
parameters: object
}
	listO.add(
	new PetriSim(
	id,name,
	new Net(listN.get(netId).clone(),
	parameters
	)
	)
	);

type s = {
objId: number
pId: number
}
	PetriSim o = listO.get(objId);
	o.getSharedP().add(
	o.getListP(pId)
	);

type sls = [s, s]	listL.add(new Link(s_a, s_b));
type C = Array<sls>	new Connector (listL);
type W = [o, C, o]	listW.add(new W(o_a, connector, o_b));

type g = {
id: number,
name: string
obj: o
quantity: number
}
	new Group(id, name, obj, quantity);

type Y = [g, C, o]	listY.add(new Y(g, connector, o));

type f = {
id: number
name: string
listW: Array<W>
quantity: number
}
	new F(id, name, listW, quantity);

Інтерпретація мовою TypeScript	Інтерпретація мовою Java
`type X = [o, C, f]`	`new X(o, connector, f);`

`type model = {`
`  id: number`
`  name: string`
`  time: number`
`  listObj: Array<o>`
`  listW: Array<W>`
`  listY: Array<Y>`
`}`
	`new Model(id,name,time,`
	`  listObj,listW,listY`
	`);`

Запропонована архітектура транслятора мови візуального програмування Петрі-об’єктних моделей. Роботу транслятора мови розподілено між клієнтським застосуванням і серверним. У клієнтському застосуванні транслятор починає виконувати свою роботу вже на етапі створення візуальних елементів, таким чином, в реальному часі будуючи текстову інтерпретацію моделі. Всі перетворення відбуваються відповідно до граматики мови візуального програмування. У серверному застосуванні транслятор перетворює текстову інтерпретацію моделі в Java-код та виконує обчислення. Результати обчислень відправляються в клієнтське застосування. Транслятор мови візуального програмування Петрі-об’єктів реалізує всі необхідні етапи трансляції програмного коду в програмний код мови нижчого рівня такі, як сканування послідовності символів, синтаксичний і семантичний аналіз та генерація коду.

4.3 Тестування транслятора

На рисунку 4.4 наведений приклад візуального представлення моделі, в результаті розбору якого як граматичного виразу отримуємо:

I⟶YI⟶oCgI⟶oslsgI⟶oslsgNI⟶oslsgNI⟶oslsgEQI⟶oslsgEQQI⟶
⟶oslsgpatQQI⟶oslsgpattapQI⟶ooslsgpattaptapNI⟶⋯
⟶ooslsgpattaptapEEQQEEQQ⟶⋯⟶oslsgpattaptaptaptappatpattaptappatpat

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.

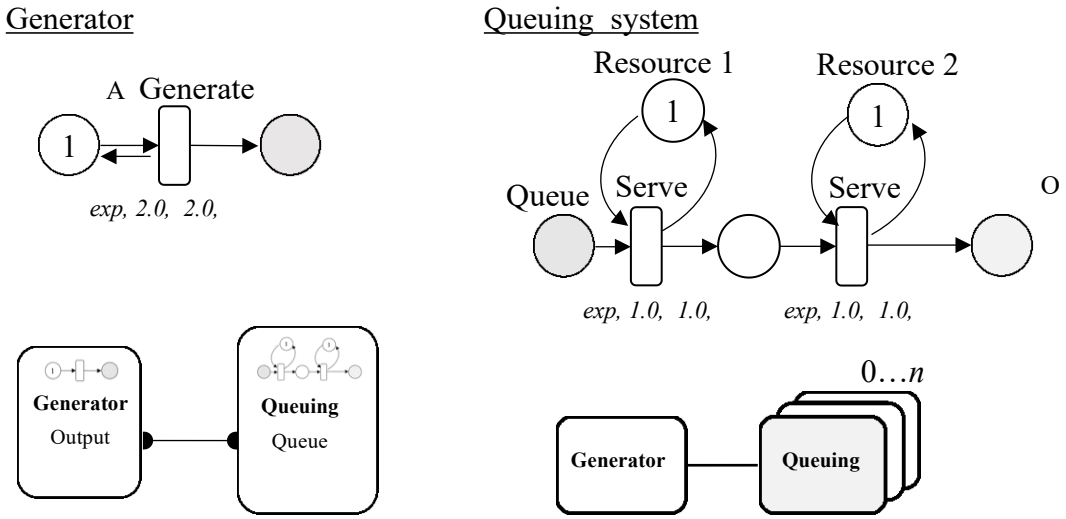
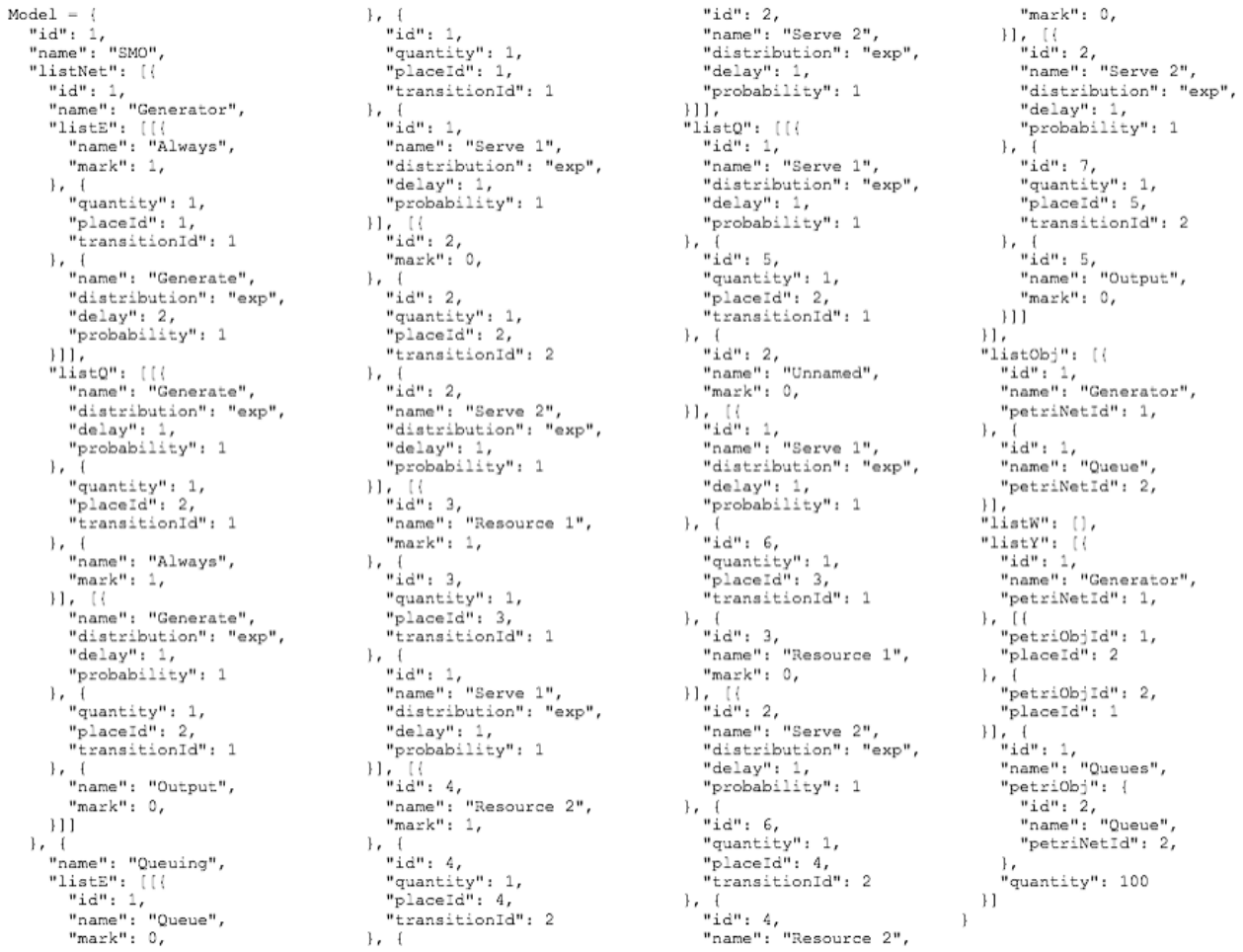


Рисунок 4.4 – Візуальне представлення Петрі-об’єктної моделі масового обслуговування: мережі Петрі-об’єктів, ототожнення позицій Петрі-об’єктів, триплет об’єкт-конектор-група.

Через обмеженість матеріалу, який можна розмістити в межах тексту дисертаційної роботи, перетворення візуального представлення моделі у її обчислення наведемо на простому прикладі моделі масового обслуговування (див. рис. 4.4). Зауважимо, що завдяки використанню групи об’єктів побудована модель в залежності від параметру *n* відтворює функціонування системи з різною кількістю обробників замовлень, які надходять на обслуговування з генератора -1. При використанні звичайних блочних редакторів, які широко використовують у симуляторах, довелось би з’єднувати вручну у візуальному редакторі усі *n* об’єктів. Очевидно, що при великих n це потребує значних зусиль. Окрім того, перегляд усіх *n* зв’язків з метою їх перевірки чи коригування також потребуватиме значних зусиль.

Результат перетворення транслятором візуального представлення моделі у JSON наведений на рисунку 4.5. Результат перетворення JSON представлення моделі у java код та запуск моделі на імітацію представлений на рисунку 4.6.



```
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
  }]
}
```


Рисунок 4.5 – Результат перетворення транслятором візуального представлення моделі у текстовий 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();
```

Рисунок 4.6 – Результат перетворення транслятором текстового формату JSON у java код.

При запуску імітації можемо пересвідчитись у правильності отриманих результатів і переконатись, що час при збільшенні 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%.

4.6 Висновки до розділу 4

Представлена розробка транслятора мови візуального програмування Петрі-об’єктних моделей. На клієнтській частині веб застосування реалізована частина транслятора, що відповідає за лексичний та синтаксичний аналіз мовного виразу. На серверній частині веб застосування реалізована частина

транслятора, що виконує семантичний аналіз мовного виразу. Для передачі даних між клієнтом та сервером використовується JSON формат. У розділі представлена інтерпретація символів алфавіту мови візуального програмування Петрі-об’єктних моделей мовою TypeScript. Запуск на обчислення відбувається після перетворення отриманих у форматі JSON даних в об’єкт PetriObjModel бібліотеки Петрі-об’єктного моделювання PetriObjLIb. Обчислення Петрі-об’єктної моделі – це відтворення подій в часі алгоритмом імітації.

5 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОЗРОБКИ ПЕТРІ-ОБ'ЄКТНИХ МОДЕЛЕЙ МОВОЮ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ

5.1 Дослідження коректності результатів обчислень алгоритму імітації

Тест 1. Розглянемо імітацію тривалістю 10 одиниць модельного часу моделі, що складається з одного Петрі-об'єкта `Queueing` (рис. 5.1), що відтворює обробку об'єктів, які очікують обслуговування в позиції `Input`, по одному. Позиція `Resource` контролює наявність ресурсу для обробки об'єктів. Очевидно, що історія змінювання маркірування у позиції `Input` має бути такою: 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 у моменти часу 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 відповідно. Тому розрахунок середнього значення маркірування позицій `Input` має вигляд [53]: $\frac{¤\cdot(ZY'):“\cdot(«YZ):\cdots:'\cdot(Z'Y¤)}{Z'} = 4{,}5$. Середнє завантаження буферу переходу `Serve` дорівнює 1, оскільки усі 10 одиниць часу стан переходу зберігав одне значення у буфері значень виходу маркерів. Результат запуску імітації моделі у розробленому програмному забезпеченні представлений на рисунку 5.2.

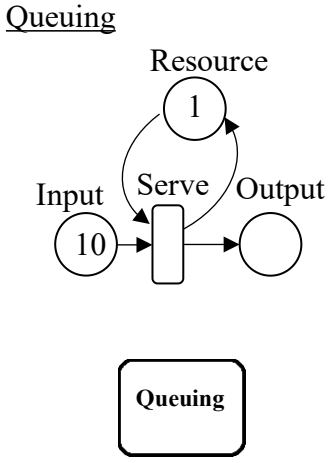


Рисунок 5.1 – Модель, що складається з одного Петрі-об'єкта, який відтворює одну подію з детермінованою затримкою.

```
Mean value of markers in places and mean value of buffers in transitions for Queueing object
Input  4.5
Resource  0.0
Output   4.5
Serve_1  1.0
```

Рисунок 5.2 – Результат запуску імітації моделі, що складається з одного Петрі-об’єкта.

Тест 2. Побудуємо модель з двох Петрі-об’єктів `Queueing`, з’єднаних спільною позицією `Input` (рис. 5.3). Оскільки з однієї позиції віднімають маркери два переходи, то історія зміни стану позиції складається із значень 8, 6, 4, 2, 0 у моменти часу 0, 1, 2, 3, 4. Отже, середнє значення маркірування: $\frac{\text{“·(ZY'):‹·(«YZ):›·(fiY«):«·(›Yfi):'·(Z'Y›)}}{\text{Z'}} = 3{,}5$. Оскільки перехід `Serve` в інтервалі часу (0; 5) знаходився у стані 1 значення у буфері значень виходу маркерів, а в інтервалі (5; 10) знаходився у стані 0 значень, то середнє значення завантаження буферу $\frac{\text{Z·(flY'):'·(Z'Yfl)}}{\text{Z'}} = 0{,}5$. Порівнюємо результати обчислень з результатами імітації (рис. 5.4).

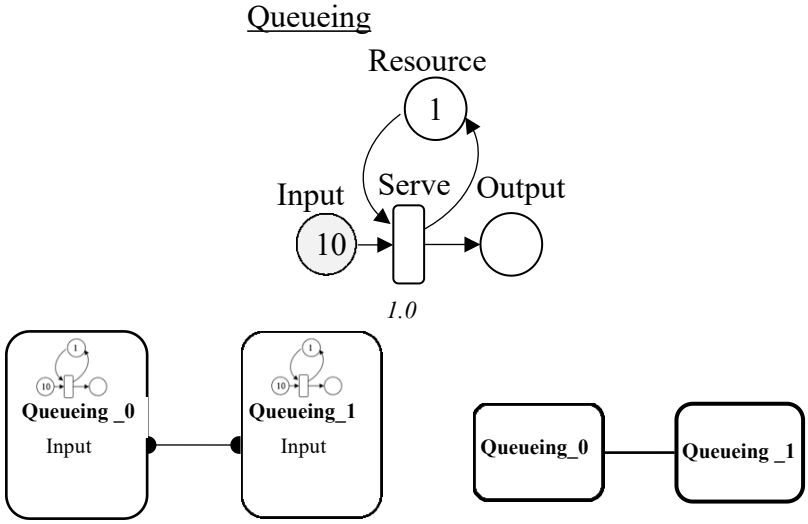


Рисунок 5.3 – Модель, що складається з одного триплета Петрі-об’єкт – конектор – Петрі-об’єкт.

```
Mean value of markers in places and mean value of buffers in transitions for Queueing_0 object
Input  2.0
Resource_1  0.5
Output   3.5
Serve_1  0.5

Mean value of markers in places and mean value of buffers in transitions for Queueing_1 object
Input  2.0
Resource_1  0.5
Out   3.5
Serve_1  0.5
```

Рисунок 5.4 – Результат запуску імітації моделі, що складається з одного триплета Петрі-об’єкт – конектор – Петрі-об’єкт.

Тест 3. Розглянемо модель, що складається з чотирьох Петрі-об’єктів `Queueing` , з’єднаних спільною позицією `Input` (рис. 5.5). Створимо групу з трьох Петрі-об’єктів `Queueing` і з’єднаємо її конектором з іншим об’єктом `Queueing`. Сірий колір першого елемента групи підказує, що з’єднання об’єкта буде тиражуватись на усі елементи групи.

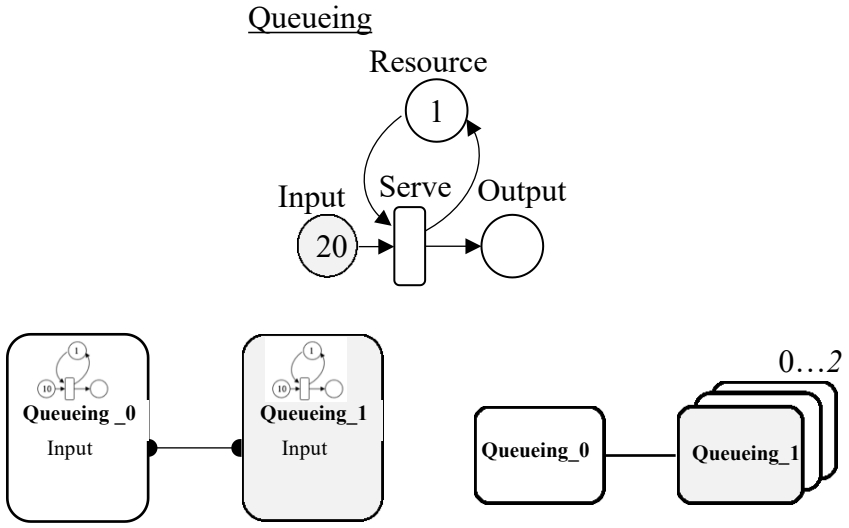


Рисунок 5.5 – Модель, що складається з одного триплета Петрі-об’єкт – конектор – група Петрі-об’єктів.

Збільшимо кількість маркерів у позиції `Input` до 20, оскільки водночас 4 об’єкти будуть віднімати маркери з цієї позиції (імітуючи обслуговування замовлення). Оскільки в один момент часу 4 маркери віднімаються, то

розрахунок середнього значення у позиції приймає вигляд: $\frac{Z‹\cdot(ZY'):Z«\cdot(«YZ):“\cdot(fiY«):›\cdot(›Yfi):'\cdot(Z'Y›)}{Z'} = 4{,}0.$ Оскільки кожний перехід опрацював 5 маркерів, то витратив він на це 5 одиниць часу, отже середнє завантаження переходу $\frac{fl}{Z'} = 0{,}5$. Порівняння аналітично розрахованих значень з тими, що отримані за результатом запуску моделі у розробленому програмному забезпеченні (рис.5.6), доводить коректність обчислень.

```
Mean value of markers in places and mean value of buffers in transitions for Queueing_0 object
Input  4.0
Resource_1  0.5
Output   3.5
Serve_1  0.5

Mean value of markers in places and mean value of buffers in transitions for Group_Queueing_1.0 object
Input  4.0
Resource_1  0.5
Out   3.5
Serve_1  0.5

Mean value of markers in places and mean value of buffers in transitions for Group_Queueing_1.1 object
Input  4.0
Resource_1  0.5
Out   3.5
Serve_1  0.5

Mean value of markers in places and mean value of buffers in transitions for Group_Queueing_1.2 object
Input  4.0
Resource_1  0.5
Out   3.5
Serve_1  0.5
```

Рисунок 5.6 – Результат запуску імітації моделі, що складається з одного триплета Петрі-об’єкт – конектор – група Петрі-об’єктів.

Тест 4. Результати імітаційного моделювання перевіримо на мережі масового обслуговування, для якої відомі результати теоретичного розрахунку. Аналітичний розрахунок мережі масового обслуговування можливий за умови експоненціальних розподілів для усіх часових затримок, відсутності обмежень на кількість місць у черзі та відсутність блокувань маршрутів. Послідовність розрахунку та формули для обчислень наведені у [53]. Наприклад, порівняємо результати імітації з результатами аналітичного розрахунку мережі, що складається з 4 систем масового обслуговування (СМО) таким чином, що замовлення надходять у першу СМО, після обробки у першій СМО з ймовірностями 0,15, 0,13, 0,3, 0,42 надходять у другу, третю, четверту СМО або

на вихід. Після обробки у другій, третій чи четвертій СМО повертаються до першої СМО. Перші три СМО одноканальні, а четверта – двохканальна. Середній інтервал надходження 2 одиниці часу, середній час обслуговування 0,6, 0,3, 0,4 та 0,1 одиниць часу відповідно. У результаті розрахунку вихідних значень мережі отримаємо значення, представлені у таблиці 5.1

Таблиця 5.1 – Результати аналітичного розрахунку мережі масового обслуговування

Показник	СМО1	СМО2	СМО3	СМО4
Середня довжина черги	1,786	0,003	0,004	0,00001
Середнє завантаження пристроїв	0,714	0,054	0,062	0,036

Петрі-об’єктну модель такої мережі масового обслуговування складемо з п’яти об’єктів: одного `Generator`, одного `QueueFork` та трьох `Queuing`, які з’єднаємо у відповідності до вказаних маршрутів (рис. 5.7). Об’єкт `QueueFork` містить фрагмент мережі Петрі, який реалізує обслуговування у першій системі масового обслуговування та вибір маршруту з заданими ймовірностями. Розгалуження маршруту реалізують 4 переходи з пріоритетом 2 та з відповідними значеннями ймовірностей. Більш високий пріоритет надає можливість відокремити групу конфліктних переходів від інших переходів, для яких, можливо, у даний момент часу виконана умова запуску. Результат запуску імітації моделі на інтервалі часу 10° представлений на рисунку 5.8.

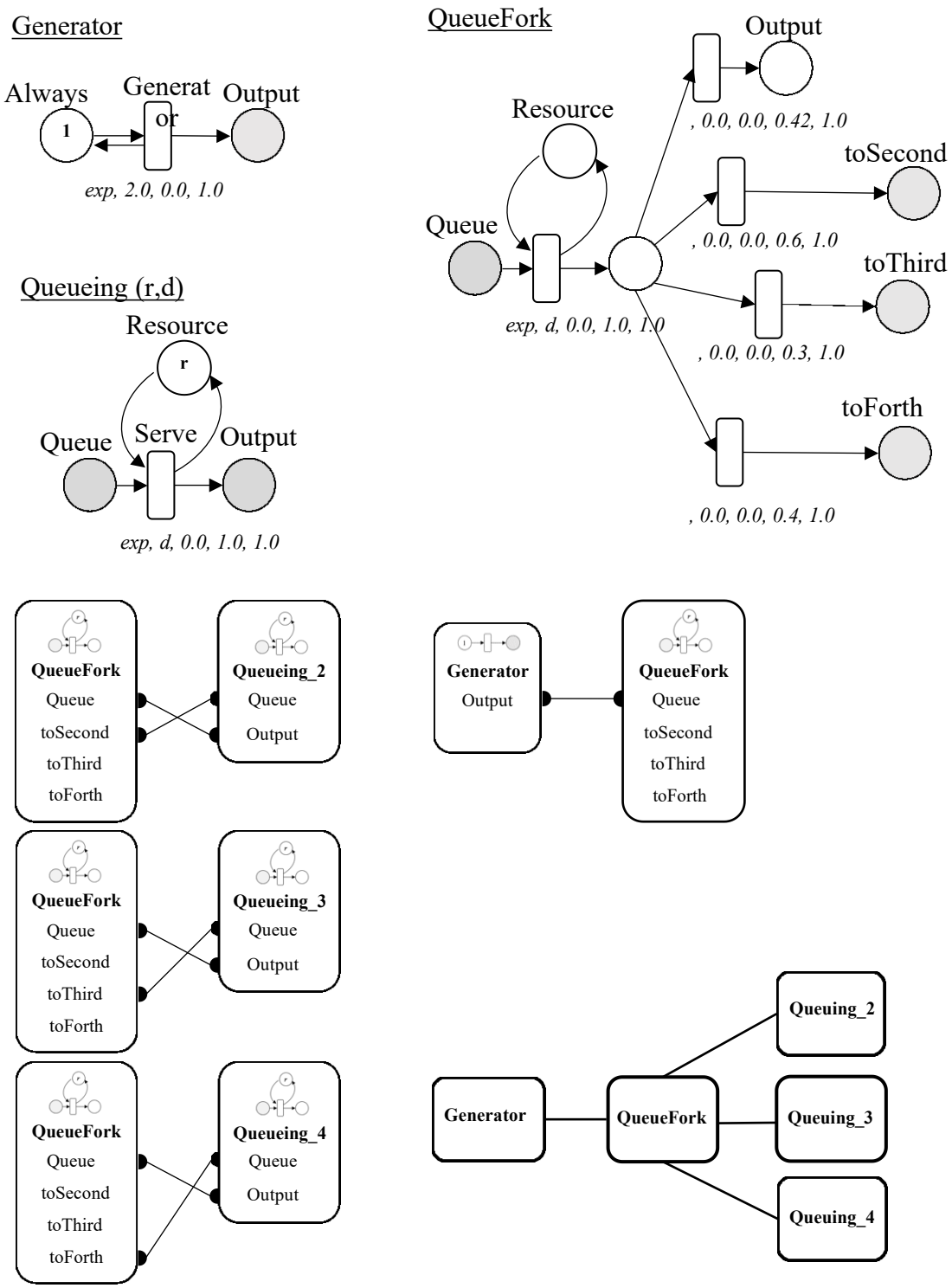


Рисунок 5.7 – Модель масового обслуговування,
що складається з 4 триплетів Петрі-об’єкт – конектор – Петрі-об’єкт.

```
Mean value of markers in places and mean value of buffers in transitions for Generator object
Always  0.0
Queue  1.7922835807814128
Generate  1.0

Mean value of markers in places and mean value of buffers in transitions for QueueForkFirst object
Queue  1.7922835807814128
Resource  0.2857725259584792
Fork  0.0
Queue  0.0030125716239068462
Queue  0.004090064816848448
Queue  1.098706746477244E-5
Output  2498436.705595427
Serve  0.7142274740414309
fork_12  0.0
fork_13  0.0
fork_14  0.0
fork_10  0.0

Mean value of markers in places and mean value of buffers in transitions for QueueingSecond object
Queue  0.0030125716239068462
Resource  0.946491721323168
Queue  1.7922835807814128
Serve  0.05350827867682572

Mean value of markers in places and mean value of buffers in transitions for QueueingThird object
Queue  0.004090064816848448
Resource  0.9379860260639625
Queue  1.7922835807814128
Serve  0.06201397393607098

Mean value of markers in places and mean value of buffers in transitions for QueueingForth object
Queue  1.098706746477244E-5
Resource  1.9642145072907493
Queue  1.7922835807814128
Serve  0.035785492709238725
```

Рисунок 5.8 – Результат запуску імітації моделі масового обслуговування, що складається з 4 триплетів Петрі-об’єкт – конектор – група Петрі-об’єктів.

Похибка для усіх значень, окрім одного (середнє значення черги у четвертій СМО), не перевищує 3% (табл. 5.2). Для середнього значення черги у четвертій СМО похибка в межах 20%, проте значення цієї черги близьке до нуля ($\sim 10^{\text{Yfl}}$), тому абсолютне значення похибки надзвичайно мале ($\sim 10^{\text{Y‹}}$). Отже, доведено, що результат обчислень алгоритму імітації збігається з результатом теоретичних розрахунків.

Таблиця 5.2 – Порівняння результатів аналітичного розрахунку та результатів імітації мережі масового обслуговування (зауважимо, що значення округлені до 0,001, а похибка розрахована за значеннями з 16 цифрами після коми)

Елемент мережі масового обслуговування	Теоретично розраховане значення	Отримане за результатом імітації значення	Похибка, %
Середня довжина черги			
СМО1	1,786	1,792	0,351
СМО2	0,003	0,003	0,419
СМО3	0,004	0,004	2,252
СМО4	0,00001	0,00001	9,871
Середнє завантаження пристроїв			
СМО1	0,714	0,7142	0,032
СМО2	0,054	0,054	0,911
СМО3	0,062	0,062	0,023
СМО4	0,036	0,036	0,596

Тест 5. Кілька систем масового обслуговування, послідовно з’єднаних між собою, за умови експоненціального закону розподілу, також допускають аналітичний розрахунок. Для одноканальних СМО з інтервалом надходження в середньому 2 одиниці часу та середнім часом обслуговування 1 одиниця часу середня довжина черги розраховується як $\frac{z}{«} = 0{,}5$. Побудуємо модель візуальною мовою програмування, використовуючи об’єкти Generator та об’єкти Quiueing (див. рис. 4.1) з послідовним зв’язком між ними (рис. 5.9).

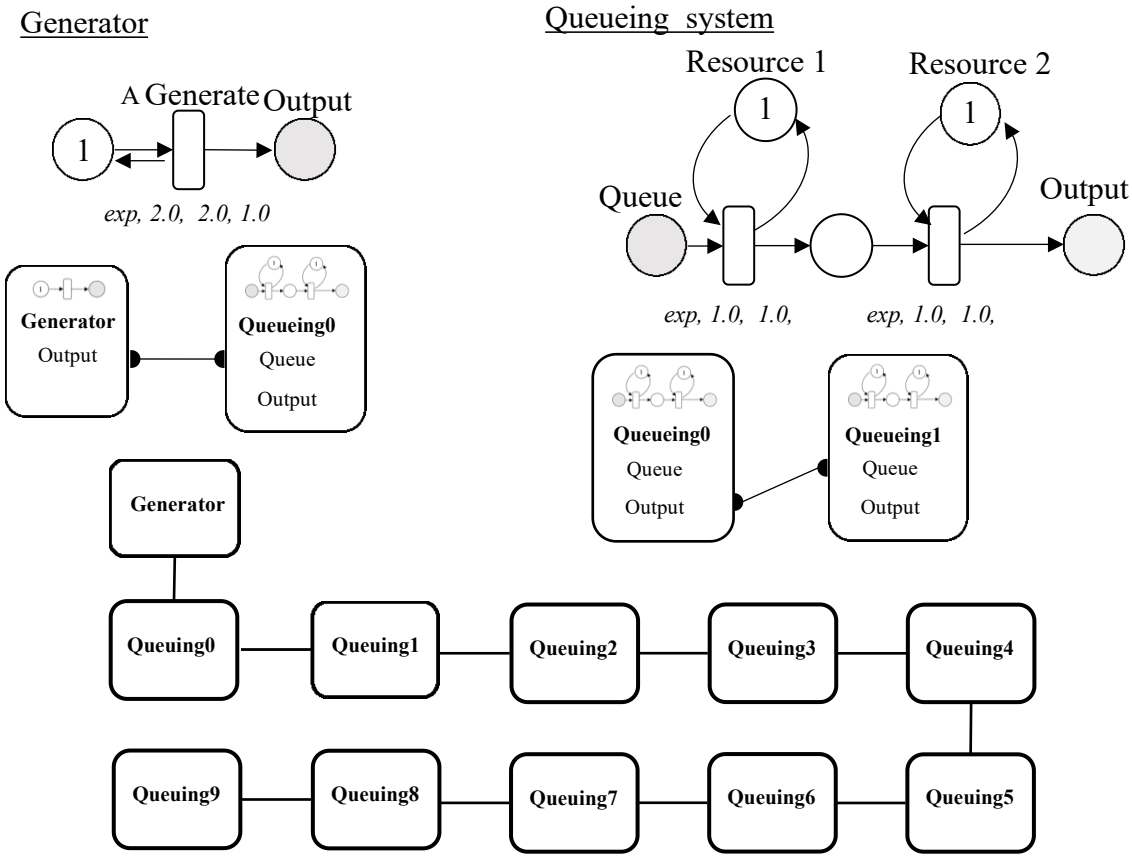


Рисунок 5.9 – Петрі-об’єктна модель мережі 20 послідовно з’єднаних систем масового обслуговування.

При запуску імітації на інтервалі 10^6 одиниць модельного часу пересвідчуємось, що усі черги мають середнє значення близьке до 0,5 (рис. 5.10). Відхилення від теоретично розрахованого значення в усіх чергах не перевищує 0,01 (2%).

```
Mean value of markers in places and mean value of buffers in transitions for QueueingObj_0 object
Queue_0  0.5019567520480057
Resource_1  0.5004460081784322
Queue_2  0.5042391344951225
Resource_3  0.49917570451758236
Queue_4  0.5035545939585718
Service_0  0.49955399182149984
Service_1  0.5008242954823952

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_1 object
Queue_4  0.5035545939585718
Resource_1  0.4990287535877032
Queue_2  0.504561894650231
Resource_3  0.4992194668275591
Queue_4  0.5055123708656566
Service_0  0.5009712464122595
Service_1  0.5007805331724516

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_2 object
Queue_4  0.5055123708656566
Resource_1  0.49918722847435076
Queue_2  0.49847173780088005
Resource_3  0.4994780218066335
Queue_4  0.5044331068018733
Service_0  0.5008127715256048
Service_1  0.5005219781933589

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_3 object
Queue_4  0.5044331068018733
Resource_1  0.498830149546271
Queue_2  0.5003161371446495
Resource_3  0.499369931216009
Queue_4  0.4988249697998211
Service_0  0.5011698504535563
Service_1  0.5006300687839925

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_4 object
Queue_4  0.4988249697998211
Resource_1  0.5002905595168233
Queue_2  0.5088668977387476
Resource_3  0.4976721108980357
Queue_4  0.4983713917233437
Service_0  0.4997094404831854
Service_1  0.5023278891019993
```

Рисунок 5.10 – Фрагмент результату запуску імітації Петрі-об’єктної моделі мережі 20 послідовно з’єднаних систем масового обслуговування зі статистичними даними для перших 10 систем масового обслуговування.

5.2 Дослідження складності процесу розробки моделі мовою візуального програмування у порівнянні з існуючими програмними засобами

Процес розробки моделі можна розбити на такі етапи:

- розробка мереж Петрі для створення об’єктів;
- розробка Петрі-об’єктів;
- розробка груп об’єктів (тиражування об’єктів);
- розробка зв’язків між Петрі-об’єктами;
- розробка зв’язків між об’єктами та групами об’єктів;
- розробка колекції об’єктів;

– розробка зв’язків об’єктів з колекцією.

Конструювання моделі можливе, якщо задана хоч одна мережа Петрі і хоч один Петрі-об’єкт. Отже, перші два кроки виконуються навіть для найпростішої моделі, що складається з одного Петрі-об’єкта. Інші етапи виконуються при побудові більш складних моделей.

Одним з класичних демонстраційних прикладів з мереж Петрі є модель «Філософів, що обідають». Ця модель є важливою для розуміння виникнення дедлоку в паралельних обчисленнях. Порівняємо розробку візуального представлення цієї моделі звичайною стохастичною мережею Петрі, розфарбованою мережею Петрі та візуальною мовою програмування Петрі-об’єктних моделей.

Реалізація моделі стохастичною мережею Петрі має вигляд, представлений на рисунку 5.11 [54].

ake

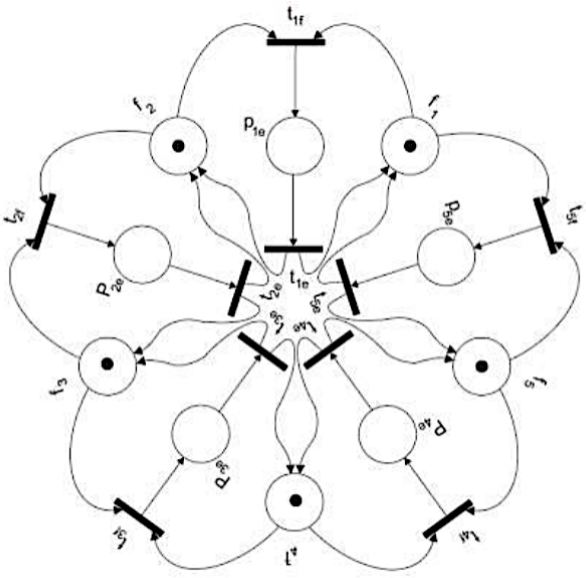


Рисунок 5.11 – Модель «філософів», представлена стохастичною мережею Петрі [54].

Якщо доведеться збільшити кількість філософів, розробнику моделі доведеться додавати нові елементи і під’єднувати. У кращому випадку графічний редактор мережі містить інструмент для копіювання фрагменту, тоді така модифікація буде трохи простішою. Проте операцію копіювання потрібно буде зробити для кожного «філософа», що додається, і після копіювання

встановити зв’язки нового фрагменту з елементами, які вже є в моделі. Тобто таке конструювання буде потребувати наполегливої рутинної роботи розробника. А уявити модель у такому представленні для 20 філософів взагалі важко, оскільки, щоб побачити усі елементи моделі, доведеться зменшити масштаб її представлення (такий інструмент не в усіх редакторах є) і при зменшенні зображення моделі зв’язки між елементами стануть візуально майже непомітними.

Якщо розробник забажає змінити параметри моделі, то йому доведеться відкривати модель в редакторі, знаходити відповідний параметр, змінювати, зберігати модель. Уявіть, що потрібно відредагувати час події «обід», тоді доведеться заходити у кожний фрагмент і змінювати відповідний параметр (тобто стільки разів, скільки філософів). Тобто редагування моделі теж є проблематичним.

Широко відомий CPNTools також розглянув у своєму тьюторіалі мережу Петрі для задачі про філософів (рис. 5.12) [55]. Замість 5 фрагментів, що відповідають 5 філософам, розфарбована мережа Петрі містить тільки один, який ілюструє правила функціонування одного «філософа». Кожному окрему філософу поставлений у відповідність маркер з типами, визначені набором PH. Цей тип разом з іншими змінними заданий у фрагменті опису мовою функціонального програмування CPN ML. Палички, якими обідають філософи, також потребують опису окремими типами маркерів. Ці типи визначені у наборі CS. Правило, за яким виймаються палички з позиції Unused Chopsticks, описано функцією Chopsticks(p), де змінна p вказує на філософа, який обраний для запуску переходу Take Chopsticks. За виразом функції Chopsticks(p), заданим у фрагменті опису мовою CPN ML, можна зрозуміти, що для кожного *i*-ого філософа витягуються палички з індексами *i* та (*i*+1). При цьому *n*-ий філософ буде потребувати для обіду палички з індексами *n* та 1. Рожевий та сині кольори виставлені розробником моделі для більш чіткого розуміння, в якій частині моделі «пересуваються» маркери-філософи і маркери-палички. Ці кольори не

виставляються автоматизовано при побудові моделі, але можуть налаштовуватись розробником моделі.

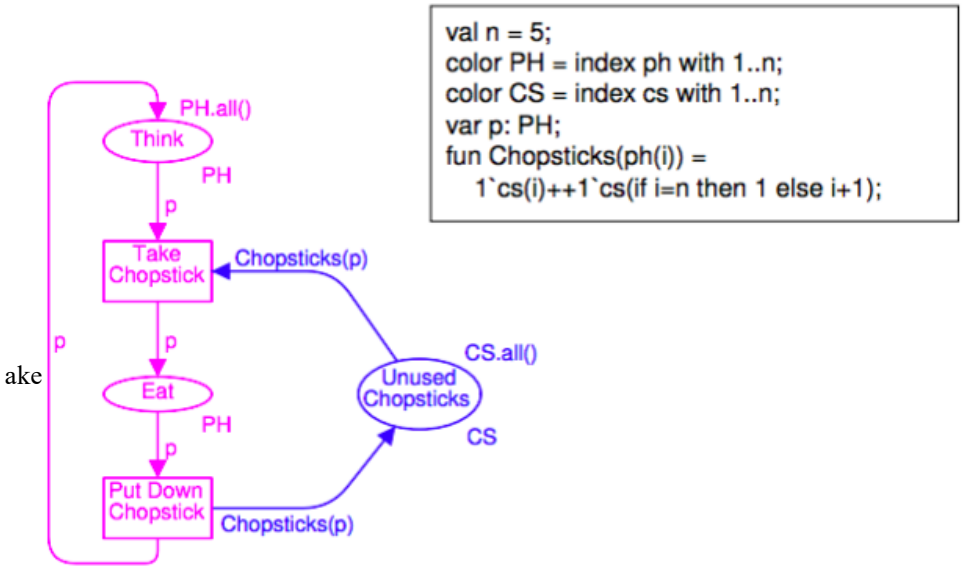


Рисунок 5.12 – Модель «філософів», представлена розфарбованою мережею Петрі [55].

Отже, представлення задачі про філософів розфарбованою мережею Петрі надає можливість розробнику легко змінювати кількість філософів та паличок на відміну від звичайної стохастичної мережі Петрі. При цьому не доведеться змінювати власне мережу Петрі, а тільки визначення типів маркерів. Проте опис типів маркерів та опис функцій, використовуваних для визначення правил витягування маркерів, міститься не в мережі Петрі, а в окремому фрагменті опису змінних та функцій моделі мовою ML функціонального програмування. Правила функціонування приховані за описами мовою ML у такій мережі Петрі. Уявити, яким чином різні типи маркерів взаємодіють в моделі, за представленням виключно мережею Петрі неможливо (доки не прочитаєте опис змінних та функцій моделі). Через це наочність представлення правил функціонування моделі, яка є у класичній мережі Петрі, втрачається. Таким чином, отримана перевага у кількості використовуваних елементів призвела до втрати найбільшої переваги мережі Петрі.

Розглянемо представлення задачі про філософів у вигляді Петрі-об’єктної моделі, побудованої мовою візуального програмування.

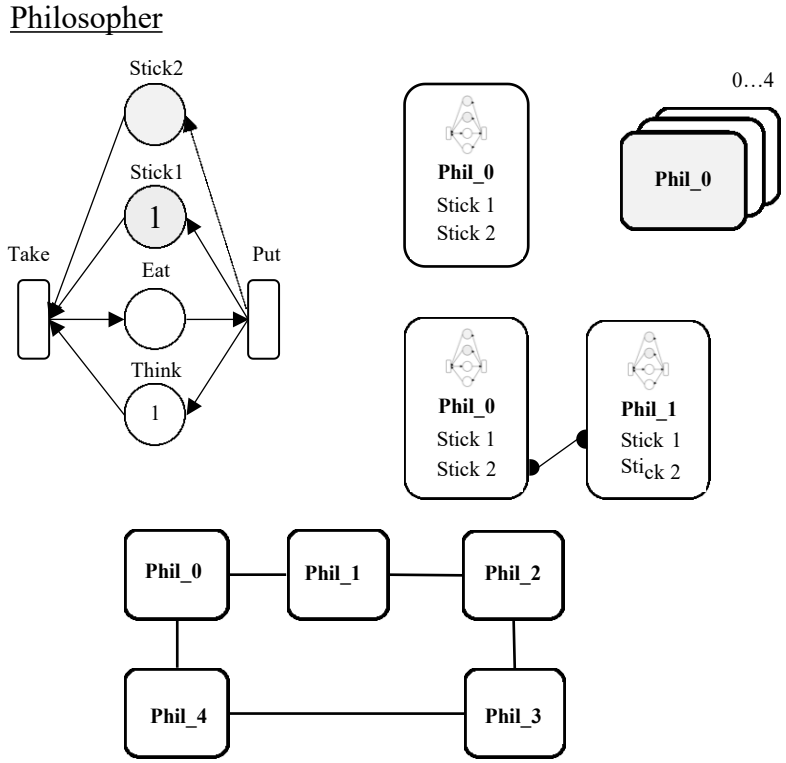


Рисунок 5.13 – Петрі-об’єктна модель «філософів»,
розроблена мовою візуального програмування

Візуальне представлення Петрі-об’єктної моделі містить мережу Петрі ‘Philosopher’, що описує події, які відбуваються у «житті» одного «філософа». Петрі-об’єкт ‘Phil_0’ побудований з заданою мережею Петрі ‘Philosopher’. Модель конструюється з Петрі-об’єктів, створених тиражуванням з Петрі-об’єкта ‘Phil_0’ у заданій кількості. Зв’язки між Петрі-об’єктами задаються для пар об’єктів ‘Phil_0’ та ‘Phil_1’, ‘Phil_1’ та ‘Phil_2’, … ‘Phil_4’ та ‘Phil_0’. Зв’язки між об’єктами групи можуть дублюватись за допомогою інструментарію Apply connector для обраного конектора і вказування індексів об’єктів, для яких має бути застосований конектор. Представлення схематичного зображення побудованих зв’язків надає можливість перевірити утворені зв’язки.

Отже, Петрі-об’єктна модель так само, як і розфарбована мережа Петрі, використовує фрагмент для опису поведінки одного філософа як основний для

побудови всієї моделі. Проте, на відміну від розфарбованої мережі Петрі, зміни стану відтворюються у мережі Петрі кожного філософа (а не в одному фрагменті, як в розфарбованій мережі Петрі). Кількість філософів можна легко налаштовувати через параметр групи, а додавати зв'язки між ними через інструмент Apply connector.

Якщо порівняти побудову моделі мовою візуального програмування Петрі-об'єктних моделей з розробкою текстовою об'єктно-орієнтованою мовою програмування, то, очевидно, що кількість рутинних дій, які виконує розробник моделі для створення одного Петрі-об'єкта, буде набагато більшою при програмуванні текстовою мовою. Дійсно, набір відповідного рядка програми – це від 10 до 50 символів (див. табл. 4.2), а клік на об'єкті, перетягування його у поле редактора, визначення параметрів об'єкта – це не більше 10 дій. Окрім того, кодування зв'язків між об'єктами є схильним до помилок, оскільки рядки коду різняться тільки індексами об'єктів і без наочного представлення їх неможливо написати. Пошук помилки у зв'язках в коді, написаному текстовою мовою, як правило, вимагає від розробника відновлення наочного представлення, тому процес є довготривалим.

У порівнянні зі звичайною стохастичною мережею Петрі представлення Петрі-об'єктної моделі візуальною мовою надає можливість розробнику концентрувати увагу на окремих фрагментах моделі, коли це потрібно, і, навпаки, абстрагуватись від деталей опису функціонування об'єкта, сприймаючи тільки його контактні позиції. Водночас, опис моделі в цілому визначається стохастичною мережею Петрі. Тому, коли розробник створює Петрі-об'єкт, це не означає, що він переходить на інший, більш високий, рівень опису. Це тільки абстрагування від деталей опису. Представлення Петрі-об'єктної моделі не є ієрархічним, тому можна безперешкодно добиратись до будь-якого елемента мережі Петрі і створювати нові зв'язки.

5.3 Дослідження швидкості обчислень алгоритму імітації

Результати експериментального дослідження швидкодії алгоритму імітації Петрі-об’єктної моделі представлені у роботі [34]. Для експерименту використовувалась модель масового обслуговування, в якій послідовно з’єднані генератор та $|\mathbf{T}|$ систем масового обслуговування. При діленні ланцюжка на q фрагментів з послідовно з’єднаних $k = |\mathbf{T}|/\mathrm{q}$ систем масового обслуговування можна легко налаштовувати складність одного об’єкта. На рисунку 5.14 представлено порівняння швидкодії Петрі-об’єктної моделі та стохастичної мережі Петрі для випадку $k = 50$ систем масового обслуговування у кожному об’єкті . Для того, щоб гарантувати однаковість застосовуваного алгоритму імітації експеримент зі звичайною стохастичною мережею Петрі проводився для Петрі-об’єктної моделі, що побудована з одного Петрі-об’єкта. З результату дослідження випливає, що при зростанні складності моделі виграш в часі виконання імітації збільшується. При цьому час виконання Петрі-об’єктної моделі, що складається з 1000 подій, у 5 разів менший за час виконання стохастичної мережі Петрі.

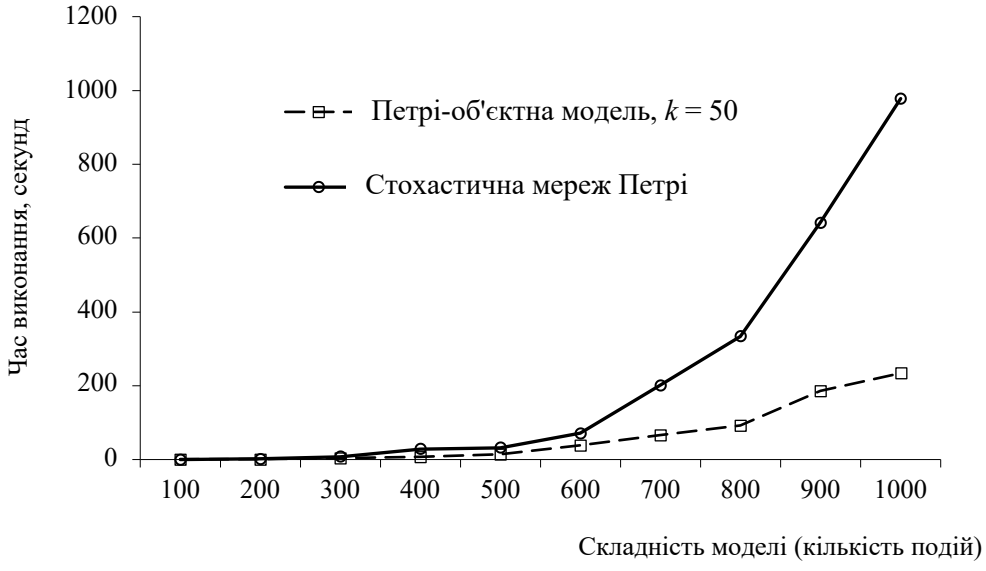


Рисунок 5.14 – Порівняння часу виконання алгоритму імітації Петрі-об’єктної моделі (складність одного Петрі-об’єкта 50) та стохастичної мережі Петрі при зростанні складності моделі.

На рисунку 5.15 представлено дослідження впливу складності одного Петрі-об’єкта на час виконання алгоритму імітації. При збільшенні складності одного Петрі-об’єкта і відповідності зменшенні кількості Петрі-об’єктів, з яких побудована модель, від 5 до 20 спостерігається значне зменшення часу виконання. Проте при подальшому ускладненні об’єкта від 20 до 50 зменшення часу виконання алгоритму майже не зменшується. З огляду на те, що фрагмент мережі Петрі, в якому 20 переходів, є більш зрозумілим ніж такий, в якому 50 переходів (особливо при візуальному представленні), слід зробити висновок, що 20 переходів є рекомендованої складністю одного Петрі-об’єкта.

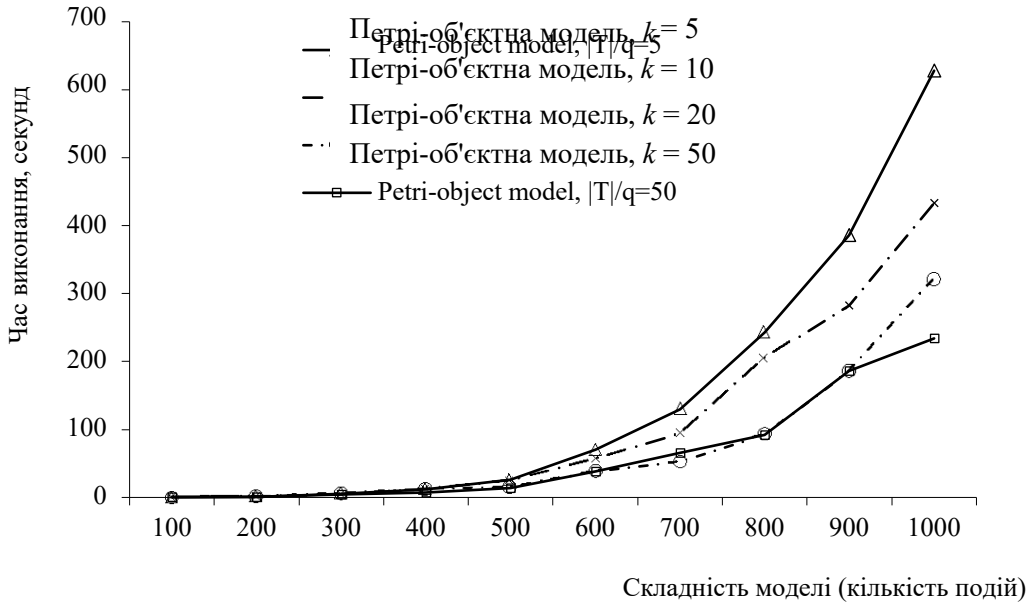


Рисунок 5.15 – Дослідження впливу складності одного Петрі-об’єкта на час виконання алгоритму імітації Петрі-об’єктної при зростанні складності моделі.

Таким чином, доведено ефективність алгоритму імітації Петрі-об’єктної моделі за показником швидкодії та визначено рекомендований рівень складності одного Петрі-об’єкта 20 подій. У багатьох випадках складність елементів, з яких будується модель, дійсно, знаходиться в межах 20. Наприклад, у роботі [56] для реалізації парадигми обчислень на мережах Петрі розглядається мережа Петрі, що складається з нескінченної кількості фрагментів, складність яких характеризується від 6 до 18 подій.

Швидкодія алгоритму імітації є важливою характеристокою для сучасних імітаційних моделей. Хоча Петрі-об’єктна технологія зменшує час виконання алгоритму імітації, абсолютний час виконання залежить від потужності обчислювальних ресурсів користувача. Клієнт-серверна реалізація алгоритму імітації, що запропонована в роботі [55], дозволяє використовувати потужні віддалені ресурси що можуть зменшити час отримання результатів імітації для користувача.

5.4 Висновки до розділу 5

Виконано експериментальне дослідження точності результатів обчислень моделі, побудованою мовою візуального програмування Петрі-об’єктних моделей. Порівняння з результатами аналітичного розрахунку (для випадку, коли можливі обидва способи моделювання – імітаційне та аналітичне) свідчить, що похибка для більшості результатів не перевищує 3% і тільки для одного результату перевищує цю межу, але слід зазначити, що абсолютне значення похибки при цьому менше за 10^{-6}.

Експериментально доведено, що алгоритм імітації Петрі-об’єктної моделі є швидшим за алгоритм імітації стохастичної мережі Петрі. При цьому значний вплив на час виконання алгоритму здійснює складність одного Петрі-об’єкта і відповідно загальна кількість Петрі-об’єктів, з яких побудована модель. Побудова моделі з великої кількості нескладних об’єктів призводить до збільшення часу виконання алгоритму і, водночас, використання надмірно великих Петрі-об’єктів у невеликій кількості також не сприяє простоті їх візуального сприйняття. За результатами дослідження рекомендовано при побудові моделей використовувати Петрі-об’єкти складністю близько 20 подій.

ВИСНОВКИ

У дисертації вирішено важливе для розвитку інженерії програмного забезпечення наукове завдання підвищення ефективності програмних засобів розробки складних моделей систем на основі формалізму стохастичних мереж Петрі. Петрі-об’єктні моделі вирішують проблему тиражування фрагментів мереж Петрі з заданими параметрами та конструювання моделі системи з великої кількості елементів.

У результаті дисертаційного дослідження отримано такі основні наукові результати:

1. Розроблена мова візуального програмування Петрі-об’єктних моделей, що надає можливість зменшити кількість помилок при конструюванні моделі за рахунок автоматизації кодування зв’язків між елементами моделі та графічного представлення моделі. Окрім тиражування Петрі-об’єктів, мова реалізує тиражування зв’язків між Петрі-об’єктами. Формалізація граматики мови візуального програмування представлена у вигляді правил виведення та зроблені висновки про її властивості.

2. Розроблена формальна граматика мови візуального програмування Петрі-об’єктних моделей. Синтаксичні правила складання мовних виразів визначені правилами виведення (продукціями), з яких слідує, що визначена граматик є контекстно-вільною і є приведеною. Петрі-об’єктні моделі, породжені граматикою, можуть бути перетворені в обчислення, а саме запущені на виконання алгоритмом імітації. Семантика мови програмування визначається правилами перетворення мовних виразів в обчислення.

3. 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.

4. Поняття Петрі-об’єктної моделі розширено поняттями конектор Петрі-об’єктів, група Петрі-об’єктів, колекція Петрі-об’єктів та група колекції Петрі-об’єктів, що надає можливість здійснювати тиражування об’єктів та тиражування зв’язків між об’єктами. Реалізація введених понять у мові візуального програмування надає можливість швидко створювати та групувати елементи моделі, динаміка яких визначається однаковим набором подій. За рахунок такого тиражування досягається також компактність візуального представлення для складних моделей. Представлено усі етапи перетворення візуального представлення моделі транслятором мови програмування. Порівняння результатів імітації з теоретичними доводить коректність виконаних перетворень.

5. На тестових моделях, розроблених мовою візуального програмування Петрі-об’єктних моделей, доведено коректність побудови моделей та результатів моделювання. За порівнянням з аналітичними моделями визначено похибку у результатах імітації, що не перевищує 3%.

6. Визначено переваги використання мови візуального програмування Петрі-об’єктних моделей у порівнянні з існуючими програмними засобами з моделювання стохастичними мережами Петрі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kamensky, J. (2018) ‘Data rich, but information poor’, IBM Center for The Business of Government. Available at: http://www.businessofgovernment.org/blog/data-rich-information-poor (Accessed 28 August 2022).

2. Simio. Other libraries and resources. Available at: https://textbook.simio.com/SASMAA/ch-misc-modeling-topics.html#extras-library (Accessed 28 August 2022)

3. Jensen, K. *et al.* (2009). CPN ML Programming. In: Coloured Petri Nets. Springer, Berlin, Heidelberg. https://doi.org/10.1007/b95112_3 (Accessed 28 August 2022)

4. Diaz, M. (ed.) (2009) *Petri nets : fundamental models, verification, and applications,* Willey.

5. ISO/IEC 15909-1:2004 Systems and software engineering — High-level Petri nets — Part 1: Concepts, definitions and graphical notation. Available at: https://www.iso.org/standard/38225.html (Accessed 28 August 2022).

6. Zaitsev, D.A.(2014) Paradigm of Computations on the Petri Nets, *Automation and Remote Control*, 75(8), 1369–1383.

7. Oris Tool. Tutoril. Available at: https://www.oris-tool.org/tutorial (Accessed 28 August 2022)

8. Paolieri, M. *et al.* (2021) The ORIS Tool: Quantitative Evaluation of Non-Markovian Systems. IEEE Trans. Software Eng. 47(6): 1211-1225

9. Zimmermann, A. *et al.* TimeNET 4.0 A Software Tool for the Performability Evaluation with Stochastic and Colored Petri Nets. Available at: https://d-nb.info/1212508939/34 (Accessed 28 August 2022)

10. CPNTools. Available at: http://cpntools.org/ (Accessed 28 August 2022).

11. Pröll, B. et al. (2022) Modelling proof-of-work agreement protocol by coloured Petri nets. *International Journal of Parallel, Emergent and Distributed Systems*, DOI: 10.1080/17445760.2022.2113398

12. CPNTools. Time attributes Available at: http://cpntools.org/2018/01/09/time-attributes-in-tokens/ (Accessed 28 August 2022)

13. Zaitsev, D.A. *et al.* (2006) Simulating of Telecommunication Systems with CPN Tools. Students' book on the course «Mathematical Modeling of Information Systems» for teaching of masters in communications. Available at: http://daze.ho.ua/cpnmp2.pdf (Accessed 28 August 2022)

14. Renew. Available at: http://www.renew.de/#:~:text=Renew%20is%20a%20Java%2Dbased,means%20to%20describe%20concurrent%20systems (Accessed 28 August 2022)

15. Oracle JDK Migration Guide. Removed tools and components. [https://docs.oracle.com/en/java/javase/17/migrate/removed-tools-and-components.html#GUID-F182E075-858A-4468-9434-8FC1704E7BB7] (Accessed 28 August 2022)

16. Kummer, O. *et al.* (2022) Renew – User Guide. University of Hamburg Department for Informatics. Theoretical Foundations Group. Release 4.0.

17. Cabac, L. *et al.* (2018) Software development with Petri nets and agents: Approach, frameworks and tool set. *Science of Computer Programming* 157, 56-70.

18. Stetsenko I.V. *et al.* (2017) 'Web application for visual modeling of discrete event systems', *2017 Internet Technologies and Applications (ITA)*, 2017, pp. 86-91.

19. Stetsenko I.V. *et al.* (2020) Computer Virus Propagation Petri-Object Simulation. *Advances in Intelligent Systems and Computing*, 1019, 103-112. Springer, Cham. https://doi.org/10.1007/978-3-030-25741-5_11

20. Shmeleva, T.R. *et al.* (2021) 'Modeling Unconditional Forwarding Decision Within Switching Lattice', *Lecture Notes in Networks and Systems*, 212, 171-186.

21. Stetsenko, I.V. *et al.* (2021) Parallel algorithm development and testing using Petri-object simulation. *International Journal of Parallel, Emergent and Distributed Systems.* Taylor & Francis. 1-16. https://doi.org/10.1080/17445760.2021.1955113 (Accessed 28 August 2022).

22. Petri, C.A. (1973) ‘Concepts of Net Theory’ *Mathematical Foundations of Computer Science*, pp. 137-146.

23. Murata, T. (1989) ‘Petri nets: Properties, Analysis and Applications’, *Proceedings of the IEEE*, 77(44), 541-580.

24. Peterson, J.L. (1981) *Petri Net Theory and the Modeling of Systems* Prentice-Hall, Englewood Cliffs, New Jersy. ISBN:0-13-661983-5.

25. Zaitsev, D.A. *et el.* (1997) 'State equations and equivalent transformations of timed Petri Nets', *Cybernetics and System Analysis*, 33 (5), 659-672.

26. Law, A.M. *et al.* (1991) *Simulation modeling and analysis*, 2nd ed., McGraw-Hill.

27. Haas P.J. (2002) *Stochastic Petri nets : modelling, stability, simulation,* Springer-Verlag, New York.

28. Stetsenko I.V. (2012) State equations of stochastic timed petri nets with informational relations, *Cybernetics and Systems Analysis*, 48 (5), 784–797.

29. Miyamoto, T.(2005) ‘A Survey of Object-Oriented Petri Nets and Analysis Methods’ *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E88–A (11), 2964-2971.

30. Valk, R. (2004) ‘Object Petri Nets’ in Desel J. *et al.* (eds) Lectures on Concurrency and Petri Nets. ACPN 2003. *Lecture Notes in Computer Science*, 3098, Springer, Berlin, Heidelberg.

31. Fehling, R. (1993) ‘A concept of hierarchical Petri nets with building blocks’ in Rozenberg G. (eds) Advances in Petri Nets *1993*. ICATPN 1991. *Lecture Notes in Computer Science*, 674, Springer, Berlin, Heidelberg.

32. Jensen, K. *et al.* (2009) Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer-Verlag Berlin Heidelberg.

33. Стеценко И.В. (2011) Теоретические основы Петри-объектного моделирования систем. *Математичні машини і системи*, 4, 136-148.

34. Stetsenko, I. V. *et al.* (2015) ‘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. https://doi.org/ 10.1109/IDAACS.2015.7340762.

35. Стеценко І.В., Лещенко К.С. (2016) Інтелектуальний компонент візуального програмування стохастичних мереж Петрі. *Технічні науки та технології,* 4 (6), 139-147.

36. Дифучин, А.Ю. *et al.* (2021) Граматика мови візуального програмування Петрі-об'єктних моделей. *Проблеми програмування*, 4, 82-94. https://doi.org/10.15407pp2021.04.082 (Accessed 28 August 2022)

37. Stetsenko I.V. *et al.* (2020) Petri-object Simulation: Technique and Software. *Information, Computing and Intelligent Systems* 1, 51-59. https://doi.org/10.20535/2708-4930.1.2020.216057

38. Stetsenko I.V. *et al.* (2021) Petri-object Simulation Two Level Visual Programming Language. *Advances in Intelligent Systems and Computing*, 1265, 266-276. Springer, Cham. ISSN 2194-5357. https://doi.org/10.1007/978-3-030-58124-4_26

39. Johnson, M. *et al.* Formal Grammars. (2012) Available at: https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/080%20Formal%20Grammars.pdf (Accessed 28 August 2022)

40. Dahl, O.-J. *et al.* (1970). Simula information. Common base language. (Report). Norwegian Computing Center. [Online] – Available from: https://bibsys-k.userservices.exlibrisgroup.com/view/delivery/47BIBSYS_UBO/12216823070002204 (Accessed 28 August 2022).

41. Prochaska, K. *et al.* Introduction to Simio. (2016). T. M. K. Roeder *et al.* (eds.) *Proceedings of the 2016 Winter Simulation Conference*.

42. Simio. The future of Simulation, Growing with you. Available at: https://www.simio.com/about-simio/why-simio/simio-is-the-future-of-simulation-software-growing-with-you.php (Accessed 28 August 2022)

43. Simul8. Visual Logic Tutorial. Available at: https://www.simul8.com/support/help/doku.php?id=features:visual_logic:tutorial (Accessed 28 August 2022)

44. Jensen, K. *et al.* (2015) ‘Colored Petri nets: a graphical language for formal modeling and validation of concurrent systems’, *Communications of the ACM*, 58(6), 61-70. DOI: 10.1145/2663340

45. Формальні мови, граматики та автомати: Навчальний посібник / Гавриленко С.Ю. – Харків: НТУ « ХПІ », 2021. – 133 с.

46. Формальні мови та автомати / Підручник для студ спец. 124 Системний аналіз / І.Я.Спекторський, В.М.Статкевич; КПІ ім. Ігоря Сікорського. – Київ: КПІ ім. Ігоря Сікорського, 2019. – 167 с.

47. Kimball J. P. *The Formal Theory of Grammar*. Prentice-Hall, 1973. 127p.

48. Becerra-Bonache, L. *et al.* (2018) ‘Mathematical Foundations: Formal Grammars and Languages’ in Mitkov, R. (ed.) *The Oxford Handbook of Computational Linguistics, Second Edition* (2nd edn) Available at: https://academic.oup.com/edited-volume/42643/chapter-abstract/358148992?redirectedFrom=fulltext, last accessed 2022/07/2

49. Дифучин А.Ю. (2022) ‘Транслятор мови візуального програмування Петрі-об’єктних моделей’, *Проблеми програмування*, 2, 13-21.

50. GitHub. Available at: https://github.com/StetsenkoInna/PetriObjModelPaint (Accessed 28 August 2022)

51. Stetsenko, I.V. (2017) ‘Parallel Algorithm for Petri Object Simulatio’,. *Cybernetics and Systems Analysis*, 53(4), 605–614.

52. Minnesota Extensible Language Tools Group. Silver. Silver Tutorial. Language Translator Architecture. Available at: https://melt.cs.umn.edu/silver/tutorial/1_language_translator_architecture/ (Accessed 28 August 2022)

53. Стеценко І.В. Моделювання систем: навч. посібник. М-во освіти і науки України, Черк. держ. технол. ун-т. - Черкаси: видавництво „Маклаут”, 2011. – 501 с. ISBN 978-966-2200-13-3.

54. Perháč, J. *et al.* (2017) ‘Modeling Synchronization Problems: From Composed Petri Nets to Provable Linear Sequents’, *Acta Polytechnica Hungarica* 14(8), 165-182.

55. CPNTools. ‘Dining Philisophers’. Available at: http://cpntools.org/wp-content/uploads/2018/01/diningphilosophers.pdf

56. Zaitsev, D.A. *et al.* (2022) ‘From strong to exact Petri net computers’, *International Journal of Parallel, Emergent and Distributed Systems*, 37(2), 167-186. DOI: 10.1080/17445760.2021.1991340

57. Дифучин А.Ю., Томашевський В.М. (2017) ‘Веб-сервіс моделювання дискретно-подійних систем’, *Вісник Національного технічного університету України «КПІ»*, 66, 32-36.

ДОДАТКИ

ДОДАТОК А

Приклад JSON представлення моделі

```
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
}]
       }
```

ДОДАТОК Б

Результати імітаційного моделювання тестових моделей

Модель «Тест 4»

Mean value of markers in places and mean value of buffers in transitions for Generator object

Always 0.0

Queue 1.7922835807814128

Generate 1.0

Mean value of markers in places and mean value of buffers in transitions for QueueForkFirst object

Queue 1.7922835807814128

Resource 0.2857725259584792

Fork 0.0

Queue 0.0030125716239068462

Queue 0.004090064816848448

Queue 1.098706746477244E-5

Output 2498436.705595427

Serve 0.7142274740414309

fork_12 0.0

fork_13 0.0

fork_14 0.0

fork_10 0.0

Mean value of markers in places and mean value of buffers in transitions for QueueingSecond object

Queue 0.0030125716239068462

Resource 0.946491721323168

Queue 1.7922835807814128

Serve 0.05350827867682572

Mean value of markers in places and mean value of buffers in transitions for QueueingThird object

Queue 0.004090064816848448

Resource 0.9379860260639625

Queue 1.7922835807814128

Serve 0.06201397393607098

Mean value of markers in places and mean value of buffers in transitions for QueueingForth object

Queue 1.098706746477244E-5

Resource 1.9642145072907493

Queue 1.7922835807814128

Serve 0.035785492709238725

Mean value of queue

1.7922835807814128

0.0030125716239068462

0.004090064816848448

1.098706746477244E-5

Mean value of channel worked

0.7142274740415209

0.053508278676832055

0.06201397393603747

0.03578549270925069

Estimation precision

Mean value of queue precision:

0.35182423188201445 %

0.41905413022820587 %

2.251620421211205 %

9.870674647724389 %

Mean value of channel worked precision:

0.03185910945670912 %

0.9105950429036006 %

0.022538606512048732 %

0.595853585414748 %

Модель «Тест 5»

State of places and transitions:

Mark in Net Generator 0 0

Buffer in Net Generator 1

Mark in Net QueueingObj_0 0 1 0 1 0

Buffer in Net QueueingObj_0 0 0

Mark in Net QueueingObj_1 0 1 0 0 3

Buffer in Net QueueingObj_1 0 1

Mark in Net QueueingObj_2 3 0 0 1 0

Buffer in Net QueueingObj_2 1 0

Mark in Net QueueingObj_3 0 1 0 1 0

Buffer in Net QueueingObj_3 0 0

Mark in Net QueueingObj_4 0 1 3 0 0

Buffer in Net QueueingObj_4 0 1

Mark in Net QueueingObj_5 0 0 0 1 0

Buffer in Net QueueingObj_5 1 0

Mark in Net QueueingObj_6 0 0 3 0 1

Buffer in Net QueueingObj_6 1 1

Mark in Net QueueingObj_7 1 0 0 1 0

Buffer in Net QueueingObj_7 1 0

Mark in Net QueueingObj_8 0 1 0 0 0

Buffer in Net QueueingObj_8 0 1

Mark in Net QueueingObj_9 0 1 0 1 500978

Buffer in Net QueueingObj_9 0 0

Mean value of markers in places and mean value of buffers in transitions for Generator object

Always 0.0

Queue_0 0.5019567520480057

Generate 1.0

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_0 object

Queue_0 0.5019567520480057

Resource_1 0.5004460081784322

Queue_2 0.5042391344951225

Resource_3 0.49917570451758236

Queue_4 0.5035545939585718

Service_0 0.49955399182149984

Service_1 0.5008242954823952

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_1 object

Queue_4 0.5035545939585718

Resource_1 0.4990287535877032

Queue_2 0.504561894650231

Resource_3 0.4992194668275591

Queue_4 0.5055123708656566

Service_0 0.5009712464122595

Service_1 0.5007805331724516

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_2 object

Queue_4 0.5055123708656566

Resource_1 0.49918722847435076

Queue_2 0.49847173780088005

Resource_3 0.4994780218066335

Queue_4 0.5044331068018733

Service_0 0.5008127715256048

Service_1 0.5005219781933589

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_3 object

Queue_4 0.5044331068018733

Resource_1 0.498830149546271

Queue_2 0.5003161371446495

Resource_3 0.499369931216009

Queue_4 0.4988249697998211

Service_0 0.5011698504535563

Service_1 0.5006300687839925

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_4 object

Queue_4 0.4988249697998211

Resource_1 0.5002905595168233

Queue_2 0.5088668977387476

Resource_3 0.4976721108980357

Queue_4 0.4983713917233437

Service_0 0.4997094404831854

Service_1 0.5023278891019993

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_5 object

Queue_4 0.4983713917233437

Resource_1 0.4994500666090504

Queue_2 0.5053771408371015

Resource_3 0.49950009324096245

Queue_4 0.5046580513538179

Service_0 0.5005499333909667

Service_1 0.500499906759043

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_6 object

Queue_4 0.5046580513538179

Resource_1 0.49977699928032576

Queue_2 0.5036611015084519

Resource_3 0.4993663428530494

Queue_4 0.509293653140287

Service_0 0.5002230007196484

Service_1 0.5006336571471199

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_7 object

Queue_4 0.509293653140287

Resource_1 0.49880536722110186

Queue_2 0.5053806119409078

Resource_3 0.4988182883629276

Queue_4 0.5102903156905264

Service_0 0.5011946327789681

Service_1 0.5011817116371053

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_8 object

Queue_4 0.5102903156905264

Resource_1 0.498835475022724

Queue_2 0.5010631783206198

Resource_3 0.4992747877723262

Queue_4 0.5040089022843596

Service_0 0.5011645249773165

Service_1 0.5007252122277276

Mean value of markers in places and mean value of buffers in transitions for QueueingObj_9 object

Queue_4 0.5040089022843596

Resource_1 0.4986850006612158

Queue_2 0.5018417636426658

Resource_3 0.4992398185540471

Queue_4 250183.720054379

Service_0 0.5013149993387525

Service_1 0.5007601814459729