УДК 004.94:004.42
DOI https://doi.org/10.36994/2788-5518-2025-02-10-18

АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ ВЕРИФІКАЦІЇ ТА ВИКОНАННЯ КООПЕРАТИВНИХ СЦЕНАРІЇВ У РОЗПОДІЛЕНИХ ТРЕНАЖЕРНИХ СИСТЕМАХ

***Шевченко С. С.**, аспірант, Інститут проблем моделювання в енергетиці ім. Г. Є. Пухова НАН України, Київ, Україна. https://orcid.org/0009-0000-3146-3591, st.shevchenko@united.productions*

***Анотація.** У статті розв'язується актуальна науково-прикладна задача програмної реалізації та автоматизованої верифікації сценаріїв для багатокористувацьких 3D-тренажерів в енергетиці. В умовах зростання складності технологічних процесів та вимог до резильєнтності критичної інфраструктури, ефективність навчання персоналу залежить не лише від індивідуальних навичок, але й від здатності до скоординованої командної взаємодії. Існуючі підходи до створення контенту часто покладаються на ненадійну ручну перевірку або використовують великовагові формальні методи, складні для інтеграції у веб-середовища. На основі формальної моделі рольової взаємодії (MPB) запропоновано комплексне алгоритмічне забезпечення, що включає структуру даних на базі JSON, алгоритм статичного аналізу та ядро динамічного виконання. Розроблений метод статичної верифікації, що базується на модифікованому пошуку в глибину (DFS), дозволяє виявляти специфічні логічні колізії («синхронізаційні тупики» (Sync Deadlocks) та недосяжні стани) безпосередньо на етапі проектування. Для забезпечення коректного виконання сценаріїв розроблено алгоритм роботи серверного ядра (Runtime Engine), що реалізує архітектуру «авторитетного сервера», гарантуючи узгодженість станів учасників навіть при стохастичних системних подіях. Ефективність підходів підтверджено експериментально. Навантажувальне тестування на синтетичних графах продемонструвало лінійну залежність часу верифікації від розмірності сценарію, що підтверджує можливість перевірки складних структур у реальному часі. Порівняльний аналіз показав значну перевагу автоматизованого методу над експертним за швидкістю та точністю виявлення помилок. Також розглянуто використання верифікатора як «фільтра безпеки» при генерації сценаріїв за допомогою великих мовних моделей (LLM), що дозволяє нівелювати ризики логічних галюцинацій штучного інтелекту.*

***Ключові слова:** кооперативне навчання, 3D-тренажер, верифікація сценаріїв, алгоритм синхронізації, клієнт-серверна архітектура, перевірка моделей.*

ALGORITHMIC SUPPORT FOR VERIFICATION AND EXECUTION OF COOPERATIVE SCENARIOS IN DISTRIBUTED TRAINING SYSTEMS

***Shevchenko S. S.**, PhD Student, G. E. Pukhov Institute for Modelling in Energy Engineering of the National Academy of Sciences of Ukraine, Kyiv, Ukraine. https://orcid.org/0009-0000-3146-3591, st.shevchenko@united.productions*

***Abstract.** The article addresses the urgent scientific and applied problem of software implementation and automated verification of scenarios for multi-user 3D simulators in the energy sector. Given the increasing complexity of technological processes and requirements for critical infrastructure resilience, personnel training efficiency depends not only on individual skills but also on the ability for coordinated teamwork. Existing approaches to content creation often rely on unreliable manual checking or use heavyweight formal methods that are difficult to integrate into web environments. Based on the formal Role Interaction Model (RIM), the paper proposes comprehensive algorithmic support, including a JSON-based data structure, a static analysis algorithm, and a dynamic execution engine. The developed static verification method, based on a modified Depth-First Search (DFS), allows for the detection of specific logical collisions ("Sync Deadlocks" and unreachable states) directly at the design stage. To ensure correct scenario execution, a server kernel algorithm (Runtime Engine) has been developed, implementing an "authoritative server"architecture that guarantees the consistency of participant states even during stochastic system events. The effectiveness of the proposed approaches is confirmed experimentally. Stress testing on synthetic graphs demonstrated a linear dependence of verification time on scenario dimension, confirming the feasibility of verifying complex structures in real-time. A comparative analysis showed a significant advantage of the automated method over expert analysis in terms of speed and accuracy of error detection. The paper also discusses the potential of using the verifier as a "safety filter" when generating scenarios using Large Language Models (LLM), which helps mitigate the risks of logical hallucinations in artificial intelligence.*

***Keywords:** cooperative learning, 3D simulator, scenario verification, synchronization algorithm, client-server architecture, model checking.*

Вступ

Забезпечення стійкості (резильєнтності) об'єктів критичної інфраструктури в умовах сучасних викликів вимагає від персоналу не лише індивідуальних технічних навичок, але й здатності до ефективної командної взаємодії [1]. Традиційні методи навчання часто не дозволяють відпрацювати сценарії складної координації дій (наприклад, спільний ремонт агрегату або ліквідація аварії) без ризику для обладнання. Вирішенням цієї проблеми є впровадження інтерактивних автоматизованих навчально-тренувальних платформ (АНТП) з підтримкою багатокористувацького режиму у 3D-середовищі [2].

У попередніх дослідженнях автора [3; 13] було запропоновано комплексний підхід до проектування таких систем та розроблено формальну модель ролевої взаємодії (МРВ), яка математично описує стани об'єктів, ролі учасників та точки їх синхронізації. Проте наявність формальної моделі є необхідною, але недостатньою умовою для створення працездатного програмного продукту.

Практична реалізація МРВ у програмному коді стикається з низкою інженерних викликів: необхідністю обробки паралельних подій від різних клієнтів, забезпеченням цілісності даних на сервері та, що найважливіше, гарантуванням логічної коректності сценаріїв. Складні сценарії з розгалуженою логікою та множинними точками синхронізації схильні до помилок проектування, таких як "дедлоки" (deadlocks) – стани взаємного блокування, коли учасники очікують дій один від одного, які ніколи не відбудуться. Ручне тестування таких сценаріїв є трудомістким та ненадійним.

Аналіз останніх досліджень і публікацій

Питання створення ефективних тренажерних комплексів для енергетики розглядаються у багатьох сучасних дослідженнях [1; 4; 18]. Особлива увага приділяється забезпеченню резильєнтності енергосистем [2] та використанню VR-технологій для навчання персоналу в умовах підвищеного ризику [1]. Разом з тим, сучасні дослідження [15] підтверджують, що інтеграція генеративного штучного інтелекту у VR-середовища значно підвищує адаптивність навчання, проте вимагає суворого контролю якості контенту.

Однак, із зростанням складності сценаріїв навчання критично важливим стає питання їхньої логічної коректності. Класичним підходом до моделювання та верифікації паралельних процесів є апарат мереж Петрі [6, 9], який і сьогодні активно розвивається для аналізу програмного коду [7]. Також застосовуються методи на основі темпоральної логіки [8] та дерев поведінки (Behavior Trees). Фундаментальні принципи виявлення взаємних блокувань (deadlocks) були закладені ще у класичних працях [10] і зараз адаптуються для розподілених систем [11].

Проте, існуючі формальні методи часто вимагають високої математичної кваліфікації від методистів, що ускладнює їх інтеграцію у сучасні Web/VR-редактори сценаріїв. У роботах, присвячених мультикористувацькій взаємодії у VR [5], акцент робиться на UX та мережевій синхронізації, але недостатньо уваги приділяється автоматизованій верифікації логіки сценарію «на льоту». Це обумовлює необхідність розробки спеціалізованого легкого алгоритмічного забезпечення.

Постановка задачі

Метою роботи є розробка алгоритмічного забезпечення для автоматизації процесів верифікації та виконання кооперативних сценаріїв у середовищі розподілених тренажерних систем.

Для досягнення мети необхідно вирішити такі завдання:

1. Розробити структуру даних для програмного представлення формальної моделі МРВ, придатну для обробки в сучасних веб-орієнтованих архітектурах.

2. Створити алгоритм статичного аналізу (формальної верифікації) сценарію для виявлення логічних колізій до етапу компіляції або запуску (“Model Checking”).

3. Розробити алгоритм роботи серверного ядра (Engine), що забезпечує динамічну обробку подій та синхронізацію клієнтів згідно з логікою МРВ.

Об'єктом дослідження є процеси управління даними в інтерактивних багатокористувацьких системах. Робота базується на архітектурі системи, описаній у патенті на корисну модель № 149786 [4], зокрема, деталізує логіку роботи блоків «Обробка сценарію» та «Центральна логіка і перевірка результатів».

Математична формалізація та структура даних

Для забезпечення строгості верифікації перейдемо від описової моделі до формальної. Сценарій кооперативного навчання можна представити як орієнтований граф станів.

Визначення 1. Граф сценарію G – це кортеж $G = \langle R, S, T, s_0 \rangle$, де:

- $R = \{r_1, r_2, ..., r_k\}$ – скінченна множина ролей учасників (наприклад, «Майстер», «Оператор»);
- S – скінченна множина станів системи, яка є об'єднанням локальних станів усіх ролей $S = \bigcup_{r \in R} S_r$;
- $s_0 \in S$ – початковий стан;
- T – множина переходів між станами.

Визначення 1.1 (Глобальний стан). Глобальний стан системи S_{global} визначається як вектор локальних станів усіх активних ролей у момент часу τ:

$$S^{\tau}_{global} = \left\langle s^{r_1}_{curr}, s^{r_2}_{curr}, \ldots, s^{r_k}_{curr} \right\rangle$$

Перехід системи з одного глобального стану в інший відбувається асинхронно при зміні стану будь-якої окремої ролі.

Визначення 1.2 (Умова завершення). Сценарій вважається успішно завершеним, якщо вектор глобального стану досягає термінальної конфігурації, де кожна роль r_i знаходиться у своєму цільовому стані $s_{end} \in \Omega_{r_i}$.

Визначення 2. Перехід $t \in T$ визначається як кортеж:

$$t = \langle s_{src}, s_{dst}, r, E_{trig}, C_{guard} \rangle$$

де:

- $s_{src}, s_{dst} \in S_r$ – вихідний та цільовий стани для ролі r;
- E_{trig} – подія-тригер (дія користувача або системний таймер);
- C_{guard} – умова переходу (guard condition).

Саме компонент C_{guard} дозволяє реалізувати механізм синхронізації. У контексті запропонованого підходу виділимо специфічний тип переходу – **синхронізаційний перехід**.

Визначення 3 (Семантика синхронізації). Перехід t для ролі r_i є синхронізаційним, якщо його умова C_{guard} залежить від стану іншої ролі r_j:

$$C_{sync}(r_i) = true \Leftrightarrow CurrentState(r_j) == s_{req}$$

Це реалізує механізм «бар'єру» або «рукостискання» (handshake), коли виконання сценарію для r_i блокується до моменту, поки r_j не досягне необхідного стану s_{req}.

Визначення 4 (Умова Sync Deadlock). Логічна колізія типу «Синхронізаційний тупик» (*Sync Deadlock*) виникає, якщо для активного стану $s_{curr} \in S_{r_i}$ існує перехід із умовою синхронізації з r_j, але у графі ролі r_j відсутній шлях від його поточного стану до необхідного стану s_{req}:

$$\sigma path(s_{curr}^{r_j} \rightarrow s_{req})$$

Наведена математична модель трансформується у структуру даних JSON для обробки програмними засобами. Структура об'єкта *Scenario* включає:

- *roles* – масив ідентифікаторів ролей (наприклад, *['master', 'mechanic']*);
- *states* – словник станів, де ключем є ID стану (s_i), а значенням – об'єкт з описом властивостей та доступних переходів;
- *transitions* – масив правил переходу, що відповідає множині T у формальній моделі.

Особливістю запропонованої структури є явне виділення умов синхронізації у полі *condition* об'єкта переходу. Нижче наведено фрагмент JSON-структури, що описує точку синхронізації, де «Механік» (*mechanic*) очікує виконання дії «Майстром» (*master*):

```
"s1": {
  "description": "Очікування фіксації деталі",
  "transitions": [
   {
    "action_id": "sync_action_01",
    "type": "synchronization",
    "role": "mechanic",
    "condition": {
           "partner_role": "master",
           "partner_state_required": "part_fixed"
    },
    "target_state": "s2"
   }
  ]
}
```

Така структура дозволяє інтерпретатору сценарію (Блок 6 на схемі архітектури [4]) однозначно ідентифікувати поточний стан системи та перевіряти умови переходів. Загальна логіка взаємодії розроблених програмних модулів у процесі проектування та виконання сценарію наведена на рис. 1.

Алгоритм формальної верифікації сценарію (Static Analysis)

Однією з ключових проблем проектування нелінійних сценаріїв є ризик виникнення логічних колізій, таких як тупикові стани (deadlocks), коли учасники взаємно блокують один одного, або недосяжні стани (unreachable states). Для вирішення цієї проблеми розроблено алгоритм статичного аналізу графа станів сценарію, який виконується перед завантаженням сценарію на сервер.

Алгоритм базується на модифікованому методі пошуку в глибину (Depth-First Search, DFS) [12] з емуляцією станів партнерів.

Формальний опис алгоритму верифікації:

1. **Ініціалізація:** Створення стеку для обходу *Stack* та множини відвіданих станів *Visited*. Поміщення початкового стану s_0 у стек.

2. **Обхід:** Поки *Stack* не порожній:
o Вилучити поточний стан s_{curr} зі стеку.
o Якщо $s_{curr} \in$ *Visited*, перейти до наступної ітерації (запобігання зацикленню).
o Додати s_{curr} до множини *Visited*.
o Визначити множину доступних переходів T_{avail} із s_{curr}.
o Якщо $T_{avail} = \emptyset$ і $s_{curr} \neq s_{final}$ (цільовий стан), зафіксувати помилку типу *Deadlock*.
3. **Перевірка синхронізації:** Для кожного переходу $t \in T_{avail}$:
o Якщо t є точкою синхронізації (тип *synchronization*), перевірити досяжність необхідного стану партнера ($s_{partner\_req}$) у паралельній гілці графа. Для умовних розгалужень (OR-splits) алгоритм використовує стратегію екзистенційної досяжності: стан вважається досяжним, якщо існує хоча б один шлях виконання, що веде до нього, що є достатнім для сценаріїв навчання з варіативним проходженням.
o Якщо $s_{partner\_req}$ недосяжний, зафіксувати помилку типу *Sync Deadlock*.
4. **Завершення:** Якщо після обходу всього графа цільовий стан s_{final} відсутній у множині *Visited*, зафіксувати помилку *Goal Unreachable*.

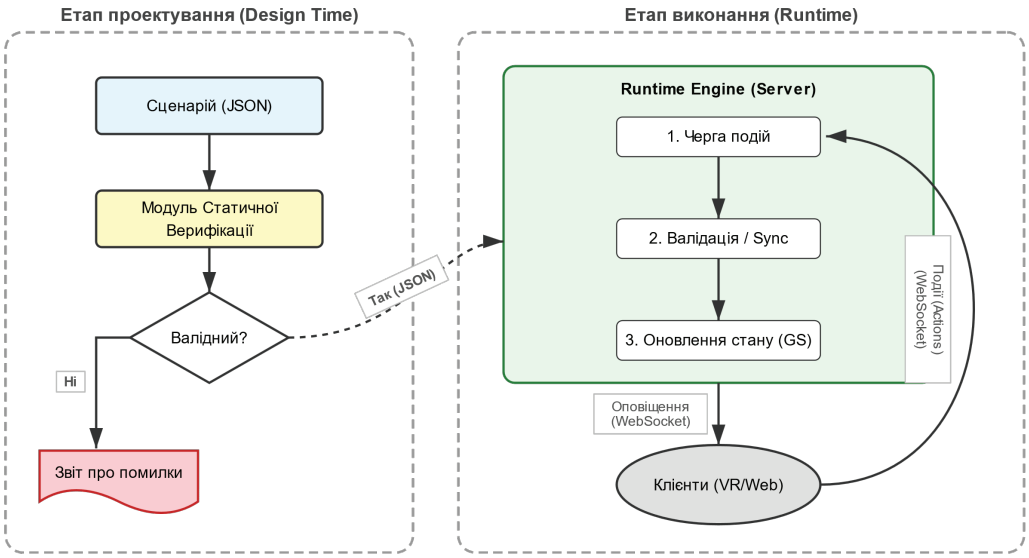

Рис. 1. Структурно-функціональна схема алгоритмічного забезпечення системи

Застосування цього алгоритму дозволяє скоротити час відлагодження сценаріїв, автоматично вказуючи розробнику на проблемні вузли логіки.

Алгоритм динамічної синхронізації та виконання (Runtime Engine)

Безпосереднє виконання сценарію забезпечується серверним ядром ("двигуном"), яке реалізує функції блоку "Центральна логіка" [4]. Ядро працює за принципом розширеної подійно-орієнтованої машини станів (Event-driven State Machine), яка обробляє гетерогенний потік подій.

Алгоритм обробки подій у реальному часі виглядає наступним чином:

1. **Реєстрація події:** У єдину чергу обробки надходять події двох типів:
o Користувацькі (e_{user}): дії учасників (натискання, переміщення об'єктів), отримані через мережевий інтерфейс.
o Системні (e_{sys}): події, генеровані внутрішнім планувальником (спрацювання таймерів, випадкові відмови обладнання, зміна параметрів середовища).
2. **Валідація:**
o Для e_{user}: перевірка, чи дозволена дія у поточному стані s_{curr} для ролі r.
o Для e_{sys}: перевірка умов активації (наприклад, чи не завершився сценарій, перевірка таймерів).
3. **Обробка синхронізації:** Якщо подія вимагає синхронізації, сервер перевіряє стан інших учасників у глобальному об'єкті сесії *GS*. Якщо умова не виконана, клієнту надсилається статус *WAIT*. Якщо умова не виконана, сервер додає ідентифікатор сесії та події у чергу очікування *PendingEvents*, а клієнту надсилається статус *WAIT* (блокування інтерфейсу). При кожній наступній зміні глобального стану (*GS'*) сервер перевіряє чергу *PendingEvents*: якщо умови для відкладених подій стають істинними, вони автоматично виконуються, а клієнтам надсилається *RESUME*.
4. **Оновлення стану:** $GS' = update(GS, e)$. При цьому системні події можуть змінювати стан примусово (наприклад, перехід у стан "Аварія" при вичерпанні часу).

5. **Оповіщення:** Генерація та розсилка пакету оновлень (*State Update*) усім клієнтам (включно з повідомленнями про системні події, наприклад, аварійну зупинку за таймером) для синхронізації візуального відображення на клієнтських пристроях.

Використання асинхронної моделі обробки подій (наприклад, на базі Node.js) дозволяє обслуговувати сотні паралельних сесій з мінімальною затримкою, що є критичним для комфортного сприйняття VR-середовища.

Експериментальні дослідження та результати

Для всебічної оцінки запропонованого алгоритмічного забезпечення було проведено два етапи експериментів: оцінка обчислювальної ефективності (масштабованості) на синтетичних даних та валідація практичної користі на реальному сценарії.

1. Аналіз масштабованості (Scalability Analysis).

Метою цього етапу було визначення залежності часу верифікації від складності сценарію. Для цього було розроблено генератор синтетичних графів сценаріїв, який створює випадкові структури з заданою кількістю станів (*N*) та переходів (*M*), імітуючи розгалужену логіку з циклами та точками синхронізації.

Тестування проводилося на стенді з процесором Intel Core i5 (3.2 GHz) та 16 GB RAM у середовищі Node.js runtime. Результати вимірювань (усереднені за 10 запусками для кожної розмірності) наведено у Таблиці 1.

Таблиця 1

Залежність часу верифікації від розмірності графа сценарію

Кількість станів (N)	Кількість переходів (M)	Час верифікації (мс)	Споживання пам'яті (MB)
50	120	15	4,2
100	280	32	6,5
500	1450	145	18,1
1000	3100	290	34,4
5000	15500	1480	128,0

Як видно з таблиці, алгоритм демонструє лінійну залежність часу виконання $T \approx k \cdot (N + M)$, що узгоджується з теоретичною складністю алгоритму пошуку в глибину ($O(N + M)$). Це означає, що навіть для надзвичайно складних сценаріїв (5000 станів, що відповідає багатогодинному тренінгу на АЕС), час автоматичної перевірки не перевищує 1.5 секунди. Це дозволяє інтегрувати верифікатор безпосередньо у веб-редактор сценаріїв для перевірки «на льоту» (real-time validation).

2. Порівняльний аналіз на реальному сценарії.

Другий етап експерименту мав на меті підтвердити практичну цінність методу у порівнянні з традиційним підходом. В якості об'єкта використано реальний сценарій «Спільне обслуговування насосного агрегату» (45 станів, 3 ролі, 12 точок синхронізації).

Для порівняння застосовано метод експертної оцінки («настільної перевірки» діаграми). Результати наведено у Таблиці 2.

Таблиця 2

Порівняння ефективності методів виявлення логічних помилок

Критерій	Експертний аналіз	Автоматизована верифікація
Час виконання	~90 хвилин	< 15 мс
Повнота покриття	~80 % (вибіркова перевірка гілок)	100% (повний перебір всіх досяжних станів)
Виявлені помилки	2 з 3 (одну пропущено)	3 з 3 (100 %)
Людський фактор	Високий (втома, втрата контексту)	Відсутній

Експеримент показав, що автоматизований алгоритм не лише радикально перевершує ручну перевірку за швидкістю, але й, що важливіше, гарантує виявлення помилок синхронізації у глибоких гілках сценарію, які важко відстежити людині через когнітивне перевантаження при аналізі паралельних процесів.

Обговорення результатів

Аналіз отриманих даних дозволяє позиціонувати запропонований підхід у контексті існуючих рішень. На відміну від класичних формальних методів, таких як мережі Петрі [6; 7] або використання інструментів Model Checking (наприклад, SPIN або UPPAAL) [8], запропонований алгоритм не вимагає трансляції сценарію у проміжні математичні нотації (Promela, PNML). Це забезпечує можливість інтеграції верифікатора безпосередньо у веб-редактори навчальних курсів (LMS), дозволяючи методистам без математичної підготовки перевіряти логіку «на льоту».

1. Порівняння з аналогами.

Як показав експеримент (Таблиця 1), час верифікації для типових сценаріїв вимірюється мілісекундами. Для порівняння, налаштування та запуск зовнішнього верифікатора типу SPIN для аналогічної задачі займає хвилини (експорт моделі, компіляція верифікатора, прогонка). Отже, ми свідомо йдемо на компроміс: ми жертвуємо універсальністю (наш алгоритм шукає лише специфічні помилки синхронізації ролей) заради швидкодії та юзабіліті. Це робить метод «легковагою» альтернативою важким академічним інструментам.

Хоча універсальні інструменти Model Checking (наприклад, SPIN або UPPAAL) забезпечують вищу математичну строгість та здатні виявляти складні колізії (circular waits), вони вимагають конвертації сценарію у специфічну мову (Promela/TLA+) та мають експоненційну складність верифікації ($O(e^N)$). Натомість, запропонований метод працює напряму з JSON-графом за лінійний час $O(N+M)$, що робить його придатним для вбудовування безпосередньо у редактор для перевірки в режимі реального часу (editor-time validation), де миттєвий зворотний зв'язок важливіший за академічну повноту перевірки.

Також варто розглянути запропонований підхід у контексті генеративного штучного інтелекту. Хоча використання великих мовних моделей (LLM) дозволяє автоматизувати створення сценаріїв, імовірнісна природа цих моделей призводить до ризику виникнення «галюцинацій логіки» [16]. Як показано у роботі [17], застосування формальної верифікації (Model Checking) до згенерованого коду дозволяє виявити значну кількість функціональних помилок ще до етапу запуску. Саме тому запропонований нами алгоритм верифікації може слугувати необхідним «фільтром безпеки» (safety filter) у ланцюжку генерації контенту штучним інтелектом, гарантуючи структурну цілісність сценаріїв, створених нейромережами.

2. Обмеження методу та динамічні колізії.

Варто зазначити, що запропонований статичний аналіз (розділ 4) гарантує структурну досяжність станів синхронізації (*pairwise synchronization reachability*). Важливо підкреслити, що поточна реалізація алгоритму фокусується на виявленні локальних помилок синхронізації, коли партнер технічно не може дійти до точки зустрічі. Виявлення складних багатосторонніх циклічних блокувань (*circular waits*, де $r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow r_1$) вимагає побудови глобального графа очікування (*Wait-For Graph*), що підвищує обчислювальну складність і є завданням наступних етапів дослідження. Однак, як показує практика, більшість помилок у навчальних сценаріях мають саме локальний характер.

Щодо виконання сценарію в реальному часі, система не може гарантувати повну відсутність мережевих затримок. Це питання вирішується на рівні архітектури Engine (розділ 5). Використання однопотокової моделі обробки подій (*Event Loop* у Node.js) дозволяє лінеаризувати потік запитів від клієнтів. Архітектура типу "Authoritative Server" виступає єдиним джерелом істини (*Single Source of Truth*), що дозволяє уникнути конфліктів стану (*race conditions*) шляхом послідовної обробки подій: якщо два користувачі одночасно надсилають несумісні команди, сервер обробить першу, а другу відхилить валідатором як неактуальну для нового стану.

При цьому модель runtime-виконання покладається на гарантовану доставку та впорядкування повідомлень (*ordered delivery*), що забезпечується транспортним рівнем (протоколи TCP/WebSocket). Питання обробки розривів з'єднання (*network partitions*) або критичних мережевих затримок, які можуть вплинути на узгодженість глобального стану, виходять за межі цієї роботи і вирішуються стандартними механізмами реконнекту та синхронізації стану (*state reconciliation*) на рівні мережевого клієнта.

3. Практичне значення.

Впровадження таких алгоритмів є критичним для забезпечення резильєнтності тренажерних систем. Усунення логічних помилок на етапі проектування запобігає ситуаціям, коли група з 5 осіб витрачає годину на проходження сценарію, щоб застрягти в кінці через помилку дизайнера. Для енергетичної галузі, де ціна помилки висока, така надійність навчального ПЗ є обов'язковою вимогою.

Висновки

У роботі вирішено задачу інженерної реалізації формальної моделі ролевої взаємодії (МРВ) для систем кооперативного навчання.

1. Розроблено JSON-орієнтовану структуру даних, яка дозволяє уніфікувати опис сценаріїв та інтегрувати їх у веб-середовище.

2. Запропоновано та реалізовано алгоритм статичного аналізу, який, на відміну від існуючих підходів, враховує специфіку точок синхронізації ролей, що дозволяє виявляти помилки типу *Sync Deadlock* на етапі проектування.

3. Алгоритм динамічної синхронізації розширено обробкою системних подій, що дозволяє моделювати не лише дії користувачів, а й стохастичні процеси (аварії, таймери), підвищуючи реалістичність навчання.

Подальші дослідження будуть спрямовані на впровадження адаптивних механізмів, які дозволять ядру автоматично спрощувати або ускладнювати сценарій в залежності від успішності дій команди.

Література

1. Shehadeh A., Alshboul O. Enhancing Engineering and Architectural Design Through Virtual Reality and Machine Learning Integration. *Buildings*. 2025. № 3. С. Т. 15. 328.
2. Саух С., Борисенко А. 2025. Моделювання електроенергетичної системи України та оцінювання її резильєнтності в умовах систематичних терористичних атак. *Технічна електродинаміка*. 2025. № 057. Т. 2.
3. Shevchenko S., Shevchenko S. Increasing the Operational Safety of NPP Pumping Equipment by Using Interactive Automated Remote Educational and Training Systems. *Nuclear and Radiation Safety*. 2024. № 1 (101). С. 19–27.
4. Radianti J. et al. A systematic review of immersive virtual reality applications for higher education: Design elements, lessons learned, and research agenda. *Computers & Education*. 2020. Т. 147. Ст. 103778.
5. Rasch J. et al. Going, Going, Gone: Exploring Intention Communication for Multi-User Locomotion in Virtual Reality. *CHI Conference on Human Factors in Computing Systems*. 2023.
6. Murata T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*. 1989. № 4. Т. 77. С. 541–580.
7. Zhang K., Liu G. TRustPN: Transforming Rust Source Code to Petri Nets for Checking Deadlocks. *10th International Conference on Control, Decision and Information Technologies (CoDIT)*. 2024.
8. Biggar O., Zamani M. A Framework for Formal Verification of Behavior Trees With Linear Temporal Logic. *IEEE Robotics and Automation Letters*. 2020.
9. Jensen K., Kristensen L. M. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer, 2009.
10. Coffman E. G., Elphick M., Shoshani A. System Deadlocks. *ACM Computing Surveys*. 1971. № 2. Т. 3. С. 67–78.
11. Wijethunga W. A. T. G. R., Kumara B. T. G. S. Systematic Review of Graph Neural Network and Consensus Algorithm-Based Approaches for Proactive Deadlock Detection in Distributed Systems. *Preprints*. 2025.
12. Tarjan R. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*. 1972. № 2. Т. 1. С. 146–160.
13. Шевченко С. С. Комплексний підхід та модель ролевої взаємодії при проектуванні кооперативних сценаріїв. *Електронне моделювання*. 2025. № 5. Т. 47. С. 75–86.
14. Інтерактивна автоматизована дистанційна навчально-тренувальна система : пат. 149786 Україна / С. С. Шевченко. Опубл. 01.12.2021, Бюл. № 48.
15. Hemminki-Reijonen U. et al. Design of generative AI-powered pedagogy for virtual reality environments in higher education. *npj Science of Learning*. 2025. № 1. Т. 10. Ст. 31.
16. Gao L. et al. Constructing a Dialogue-Level Hallucination Evaluation Benchmark for Large Language Models. *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. 2024.
17. Gadde D. N. et al. All Artificial, Less Intelligence: GenAI through the Lens of Formal Verification. *arXiv preprint arXiv:2403.16750*. 2024.
18. Самойлов В. Д. та ін. Комп'ютерні технології розробки тренажерних систем для енергетичної галузі. *Електронне моделювання*. 2020. № 3. Т. 42. С. 89–98.

References

1. Shehadeh, A., Alshboul, O. (2025). Enhancing Engineering and Architectural Design Through Virtual Reality and Machine Learning Integration. *Buildings*. 15 (3). 328.
2. Saukh, S., Borysenko, A. (2025). Modeling the electric power system of Ukraine and assessing its resilience under conditions of systematic terrorist attacks. *Tekhnichna Elektrodynamika*. 2025. (2) [in Ukrainian].
3. Shevchenko, S., & Shevchenko, S. (2024). Increasing the Operational Safety of NPP Pumping Equipment by Using Interactive Automated Remote Educational and Training Systems. *Nuclear and Radiation Safety*. 1 (101). 19–27.
4. Radianti, J., et al. (2020). A systematic review of immersive virtual reality applications for higher education: Design elements, lessons learned, and research agenda. *Computers & Education*. 147. 103778.
5. Rasch, J., et al. (2023). Going, Going, Gone: Exploring Intention Communication for Multi-User Locomotion in Virtual Reality. *CHI Conference on Human Factors in Computing Systems.*
6. Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*. 77 (4). 541–580.
7. Zhang, K., Liu, G. (2024). TRustPN: Transforming Rust Source Code to Petri Nets for Checking Deadlocks. *10th International Conference on Control, Decision and Information Technologies (CoDIT).*
8. Biggar, O., Zamani, M. (2020). A Framework for Formal Verification of Behavior Trees With Linear Temporal Logic. *IEEE Robotics and Automation Letters.*
9. Jensen, K., Kristensen, L. M. (2009). Coloured Petri Nets: Modelling and Validation of Concurrent Systems. *Springer.*
10. Coffman, E. G., Elphick, M., Shoshani, A. (1971). System Deadlocks. *ACM Computing Surveys*. 3 (2). 67–78.
11. Wijethunga W. A. T. G. R., Kumara B. T. G. S. (2025). Systematic Review of Graph Neural Network and Consensus Algorithm-Based Approaches for Proactive Deadlock Detection. *Preprints*.
12. Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM journal on computing*. 1 (2). 146–160.
13. Shevchenko, S. (2025). A comprehensive approach and a role-based interaction model for designing cooperative scenarios in interactive training systems. *Electronic Modeling*. 47 (5). 75–86 [in Ukrainian].

14. Shevchenko, S. (2021). *Interactive automated remote educational and training system*. (Patent of Ukraine No. 149786). Ukrainian Institute of Intellectual Property [in Ukrainian].

15. Hemminki-Reijonen, U., Hassan, N. M. A. M., Huotilainen, M., Koivisto, J.-M., & Cowley, B. U. (2025). Design of generative AI-powered pedagogy for virtual reality environments in higher education. *npj Science of Learning*. 10 (1). 31.

16. Gao, L., et al. (2024). Constructing a Dialogue-Level Hallucination Evaluation Benchmark for Large Language Models. *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*.

17. Gadde, D. N., Kumar, A., Nalapat, T., Rezunov, E., & Cappellini, F. (2024). All Artificial, Less Intelligence: GenAI through the Lens of Formal Verification. *arXiv preprint arXiv:2403.16750*.

18. Samoilov, V., et al. (2020). Computer technologies for the development of training systems for the energy industry. *Electronic Modeling*. 42 (3). 89–98 [in Ukrainian].

Дата першого надходження статті до видання: 20.10.2025
Дата прийняття статті до друку після рецензування: 25.11.2025
Дата публікації (оприлюднення) статті: 26.12.2025