DOI: https://doi.org/10.25140/2411-5363-2025-3(41)-261-271
УДК 004.42:37.091.3:378.018.43

Андрій Васильович Хижняк[1], Володимир Вікторович Казимир[2]

[1]аспірант, старший викладач кафедри інформаційних та комп'ютерних систем
Національний університет «Чернігівська політехніка» (Чернігів, Україна)
E-mail: alf.andrey@gmail.com. **ORCID:** http://orcid.org/0009-0008-8655-3736

[2]доктор технічних наук, професор, професор кафедри інформаційних та комп'ютерних систем
Національний університет «Чернігівська політехніка» (Чернігів, Україна)
E-mail: vvkazymyr@gmail.com. **ORCID:** https://orcid.org/0000-0001-8163-1119. **ResearcherID:** Q-2925-2016

ДОМЕННО-ОРІЄНТОВАНА МОВА ОПИСУ ПЕРСОНАЛІЗОВАНИХ ПРАКТИЧНИХ ЗАВДАНЬ ДЛЯ ІНЖЕНЕРНИХ СПЕЦІАЛЬНОСТЕЙ

У роботі представлений опис формальної граматики нової доменно-орієнтованої мови Learning Task Definition Language (LTDL), призначеної для опису структури, етапів, залежностей, параметризації та запуску і налаштування навчальних середовищ для виконання практичних завдань в інженерній освіті. Визначені особливості мови, сфера її застосування. Проведене тестування мови: парсинг тести, лексичне тестування, тестування генерації коду, побудова синтаксичного дерева, визначення непродуктивних та недосяжних символів. Показані можливості візуалізації завдання за допомогою графічного варіанту мови.

Ключові слова: *практичні завдання; автоматична генерація; автоматичне оцінювання; персоналізація навчання; формальна граматика; доменно-орієнтована мова.*

Рис.: 3. Табл.: 2. Бібл.: 14.

Актуальність теми дослідження. Останніми роками зростає попит на масштабовані та персоналізовані освітні інструменти, спрямовані на підтримку автоматизації та академічної доброчесності. Особливо це актуально в розрізі практичних завдань, які служать наріжним каменем навчання студентів інженерних спеціальностей. Важливо, щоб завдання були структурованими, машиночитаними та параметризованими; ці аспекти забезпечуються формальним визначенням внутрішньої структури цих завдань [1]. Збалансованість практичних завдань може бути досягнута, наприклад, шляхом використання доменно-орієнтованої мови (DSL), за допомогою якої можна спростити розробку навчальних компонентів як викладачами при створенні методичних вказівок до лабораторних і практичних занять, так і студентами в процесі самостійного опанування навчального контенту.

Постановка проблеми. Спроби використати DSL в освітніх цілях фіксуються впродовж останніх років [2, 3, 4, 5, 6], однак розробка спеціалізованих мов для формалізації опису шаблонів практичних завдань залишається недостатньо розвиненою. Існуючі DSL мають обмежений функціонал, іншу направленість та низку концептуальних і технологічних недоліків щодо опису типів завдань, способів перевірки, рівнів складності, контекстів виконання тощо.

Аналіз останніх джерел і публікацій. У роботі [2] розроблено DSL для автоматизованого створення та налаштування віртуальних класів у рамках LMS. Вона забезпечує високий рівень абстракції та візуальність за рахунок використання графічного режиму в рамках Eclipse Graphical Modeling Framework і дозволяє викладачам створювати класи схематично. Ця мова має низку концептуальних і технологічних обмежень: тісно пов'язана з конкретною платформою, спрямована на структурування середовища навчання, але не моделює сам зміст завдань, не передбачено адаптивних механізмів і не придатна для складних сценаріїв. У роботі [3; 4] представлена нова вбудована домен-специфічна мова EDSL. Ця мова є центральним елементом фреймворку, призначеним для опису параметризованих завдань для курсу з програмування на Haskell-I/O та їх правильних рішень. Автори вказують на наявні обмеження та недоліки: залежність від якості генераторів специфікацій, які повинні бути надані викладачем, ризик створення нерозв'язних завдань, а система генерації тестів стикається з проблемами застрягання при пошуку тестів. Quiz-game DSL представлена в роботі [5] для створення вікторин/освітніх

ігор і дозволяє описати структуру вікторини: запитання, рівні складності, логіку, але сфокусована лише на вікторинах – не підтримує складні сценарії, симуляції, інтеракції зі складною логікою. У роботі [6] представлено генератор питань, який використовує власну легку вбудовану домен-специфічну мову для опису шаблонів питань, ключовою особливістю якої є можливість параметризації аспектів питань, визначення діапазонів їх значень та потенційних взаємодій чи залежностей між ними, що дозволяє генерувати цілі родини схожих питань, але якість залежить від розробника, а не генератора. Сама мова має дуже обмежений функціонал і підходить тільки для запитань з однією відповіддю в рамках тільки однієї LMS.

Виділення недосліджених частин загальної проблеми. Незважаючи на доцільність запропонованих рішень, вони мають низку концептуальних і технологічних обмежень їх застосування для гнучкої генерації персоналізованих практичних завдань. Існуючі рішення мають такі спільні обмеження, як: залежність від конкретної платформи, недостатня підтримка формального опису логіки перевірки, орієнтація переважно на макрорівень (курси, ресурси), а не мікрорівень (структура і зміст завдань), відсутність механізмів адаптації до опису практичних завдань.

З погляду на це, дуже перспективним є створення доменно-специфічної мови, яка забезпечить викладачів формальним і гнучким інструментом для створення персоналізованих практичних завдань, підтримуватиме контрольовану генерацію тексту завдання та розгортання навчального середовища, міститиме правила для автоматичної перевірки прогресу виконання завдання і може бути використана в сучасних інформаційних системах автоматизованої генерації та перевірки параметризованих практичних завдань [7].

Мета статті. Метою статті є опис формальної граматики доменно-орієнтованої мови Learning Task Definition Language та представлення результатів її тестування.

Виклад основного матеріалу.

1. Опис домену персоналізованих практичних завдань.

Практичне завдання (task) направлене на розвиток або перевірку навичок студента з певної теми. Для створення завдання треба розуміти, яка саме мета чи основна ідея закладена у виконання цього завдання, а головне, яка ціль повинна бути досягнута під час виконання завдання. Частіше за все ціллю має бути конкретний стан навчального середовища або об’єкт, існування якого можна в подальшому перевірити.

Можлива ситуація, коли для досягнення загальної мети практичного завдання потрібно виконати більше, ніж один етап (stage). Тоді для кожного етапу треба мати ціль, досягнення якої підтверджує виконання етапу і яку можна перевірити. Етап може мати окрему мету, а може бути кроком для досягнення основної. Кожен етап має свій створений об’єкт, запущений процес або конкретний стан навчального середовища. Етапи можуть бути залежними від інших етапів. Етапи, які повинні бути виконані послідовно і тільки в певному порядку, для зручності варто згрупувати в окрему сутність – трек (track). Наприклад, для виконання задачі за допомогою програми на мові С студенти повинні написати код програми, скомпілювати вихідні коди, запустити програму.

Параметризація дозволяє автоматично генерувати різні варіанти одного і того ж завдання, змінюючи певні його елементи. Для цього треба виділити частини, які потребують персоналізації для кожного етапу: імена змінних, назви файлів або директорій, ім’я користувача, функцій, порядок параметрів тощо. Таким чином, для кожного студента буде згенеровано унікальне завдання, яке за дидактичною складністю не буде відрізнятися від завдань інших студентів. Такі частини завдання іменуються параметризованими змінними (variables).

Важливою частиною завдання є метаінформація, яка може бути використана поза межами самого завдання для розуміння, кому і коли це завдання можна видавати, скільки часу орієнтовно повинно займати його виконання.

Для виконання практичного завдання зазвичай треба розгортати окреме навчальне середовище. Для розгортання навчального середовища потрібно створити Learning Environment deployment instructions, за допомогою яких відповідний модуль системи зможе розгорнути навчальне середовище. Ці інструкції поділяються на інструкції запуску (create instructions) середовища та інструкції налаштування середовища (provision instructions). У цих інструкціях використовуються додаткові параметри, які є персоналізованими змінними для завдання. Також додатково вказується, який саме тип навчального середовища треба запустити: який backend буде використовуватися, протокол доступу до навчального середовища та його параметри.

2. Розробка формальної граматики Learning Tasks Definition Language (LTDL).

Розробка LTDL здійснювалась згідно із принципів, закладених в [8, 9, 10, 11].

Формальною породжувальною граматикою (генеративною граматикою, граматикою з фразовою структурою, граматикою з переписуванням) називають кортеж [12]:

$$G = (N, T, P, S),$$

де N – множина нетермінальних символів (нетерміналів, змінних);

T – множина термінальних символів (терміналів, основних символів);

$N \cap T = \emptyset$; P – множина правил виведення (продукцій, правил підстановки) $\{(\alpha, \beta)\}$;

S – початковий символ (стартовий символ, аксіома). Початковим символом є task.

Введемо алфавіт нетермінальних символів для опису сутностей персоналізованого практичного завдання: TC – вміст завдання, LE – середовище навчання, LEC – його вміст, CI – інструкції створення навчального середовища, PI – інструкції налаштування навчального середовища, V – параметризована змінна, TR – послідовність етапів завдання, TRC – вміст послідовності, ST – етап послідовності, STC – вміст етапу, I – інструкція або дія, яку треба виконати в навчальному середовищі, A – сама дія, D – визначення та опис завдання, C – перевірка та фіксування результатів етапу. Доповнимо алфавіт допоміжними нетерміналами, які задають списки сутностей: VL, TRL, STL, DL, CL, IL. Алфавіт термінальних символів складається з набору зарезервованих слів під кожну сутність ("task", "learning_environment", "create_instructions", "provision_instructions", "variable", "track", "stage", "instruction", "definition", "check"), знаків пунктуації ("=", "{", "}", "[", "]", ":", ",", "$", ".") та загальної буквено-цифрової послідовності (string_literal).

Правила виведення граматики представлені у таблиці 1.

Таблиця 1 – Правила виведення граматики (синтаксис мови)

№	Правило	Зміст правила
1	2	3
1	S → "task" string_literal ":" "{" TC "}"	Створення сутності практичного завдання з іменем.
2	TC → LE | "stages" "=" "["STL"]" | "tracks" "=" "["TRL"]" | "variables" "=" "[" VL "]"	Завдання може складатися з опису навчального середовища (НС) та списків: етапів, послідовностей етапів, змінних.
3	LE → "learning_environment" string_literal ":" "{" LEC "}"	Створення сутності НС з іменем.
4	LEC → VL | CI | PI	НС складається зі списків: змінних, інструкцій створення та налаштування.
5	CI → "create_instructions" "=" "[" IL "]"	Створення списку інструкцій створення НС.
6	PI → "provision_instructions" "=" "[" IL "]"	Створення списку інструкцій налаштування НС.

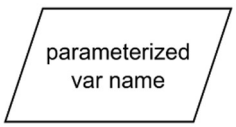
Закінчення табл. 1

1	2	3
7	IL → I | I "," IL	Список інструкцій може складатися з однієї інструкції або списку інструкцій, розділених комою.
8	I → "instruction" string_literal ":" "{" A "}"	Створення інструкції з іменем.
9	A → "action" "= " (string_literal | "$" | ".")	Безпосередня дія, яка є послідовністю звичайних символів або спеціальною послідовністю для інтерполяції значення змінної у вигляді: $scope.name.
10	TRL → TR | TR "," TRL	Список послідовностей може складатися з однієї послідовності або декількох послідовностей, розділених комою.
11	TR → "track" string_literal ":" "{" TRC "}"	Створення послідовності з іменем.
12	TRC → "stages" "=" "[" STL "]" | "variables" "=" "[" VL "]"	Послідовність складається зі списків: змінних та етапів.
13	STL → ST | ST "," STL	Список етапів може складатися з одного етапу або декількох послідовностей, розділених комою.
14	ST → "stage" string_literal ":" "{" STC "}"	Створення етапу з іменем.
15	STC → "variables" "=" "[" VL "]" | "definitions" "=" "[" DL "]" | "checks" "=" "["CL"]" | "preconditions" "=" "[" IL "]"	Етап складається зі списків: змінних, визначень, перевірок, інструкцій.
16	VL → V | V , VL	Список змінних може складатися з однієї змінної або декількох змінних, розділених комою.
17	V → "variable" string_literal ":" "{" "}"	Створення змінної з іменем.
18	DL → D | D "," DL	Список визначень може складатися з одного визначення або декількох визначень, розділених комою.
19	D → "definition" string_literal ":" "{" "}"	Створення визначення та опису з іменем.
20	CL → C | C "," CL	Список перевірок може складатися з однієї перевірки або декількох перевірок, розділених комою.
21	C → "check" string_literal ":" "{" A "}"	Створення перевірки з іменем.

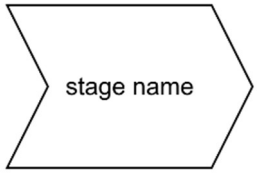
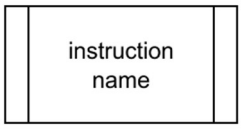
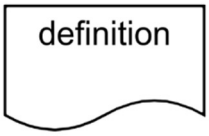
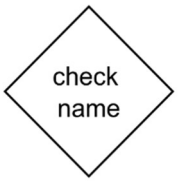
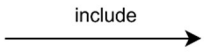
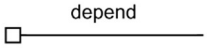
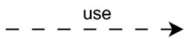
Джерело: розроблено авторами.

Сутності персоналізованого практичного завдання представимо у таблиці 2 у вигляді графічних зображень символів мови. Повний перелік параметрів елементів не виводиться у графічному представленні, окрім параметра *name*, який призначений для відображення назви відповідного елементу.

Таблиця 2 – Графічні зображення символів алфавіту мови LTDL

Графічне представлення символу	Сутність	Опис
1	2	3
-	**task**	Це основна сутність і початкове правило граматики; включає в себе опис завдання, його метадані, змінні, треки і етапи, опис навчального середовища.
	**variable**	Задає параметризацію завдання для застосування у темплейті або інструкції. Змінна може бути рівня всього завдання, треку або етапу.
	**track**	Використовується для групування етапів, які залежать один від одного. Це лінійна послідовність виконання кроків, складається зі змінних та етапів.

Закінчення табл. 2

1	2	3
	**stage**	Описує один із етапів виконання завдання і складається зі змінних, попередніх умов, опису та перевірок. Попередні умови -– це список інструкцій, які треба виконати в екземплярі навчального середовища, перед тим як починається виконання завдання.
	**instruction**	Описує дію, яка повинна бути виконана для запуску індивідуального середовища або в ньому. Використовується в описі навчального середовища (**learning environment**) та в попередніх умовах етапу (**stage**).
	**definition**	Описує темплейт, який видається студенту перед тим, як він почне виконувати завдання і описує, що потрібно виконати в конкретному етапі (**stage**).
	**check**	Задає дії, які треба виконати для відстежування успішності виконання конкретного етапу (**stage**). Виконуються при підключенні до навчального середовища, залежно від протоколу.
	**learning environment**	Опис навчального середовища, в якому буде виконуватися завдання. Має параметри та інструкції по запуску і налаштуванню конкретного екземпляра навчального середовища.
	**include**	Тип зв'язку, який встановлює вкладеність однієї сутності в іншу – наприклад, етап є частиною треку.
	**depend**	Тип зв'язку, який встановлює залежність етапів один від одного.
	**use**	Тип зв'язку, який показує, де буде використана параметризована змінна.

Джерело: розроблено авторами.

Серед особливостей мови можна виділити наступні: відсутність лівої рекурсії, граматика підходить для LL-парсингу, має структуровану ієрархію (task → tracks → stages → preconditions/definitions/checks), підтримує інтерполяцію змінних через режими лексера, має типізовані значення, лексер використовує стек режимів для обробки рядків. Граматика дозволяє описувати складні навчальні сценарії із залежностями, перевірками та параметризацією через змінні. Для опису елементів домену було обрано синтаксис у вигляді патерну:

[модифікатори] ключове_слово ім'я [параметри] { тіло }.

Подібна структура використовується для опису сутностей у багатьох мовах програмування та конфігураційних мовах. Переваги цього патерну в тому, що він легко читається людьми, зручний для парсингу, дає чітку структуру для автоматичної генерації, дозволяє легко розширювати синтаксис модифікаторами та параметрами.

3. Тестування та оцінювання LTDL.

Для тестування створеної граматики та згенерованих лексичного та синтаксичного аналізатора було створено та проведено наступні тести: парсинг тести, лексичне тестування, тестування генерації коду, побудова синтаксичного дерева, визначення непродуктивних та недосяжних символів.

У межах тестування парсингу за допомогою мови Python створені парсинг тести на валідні конструкції та неправильний синтаксис. Для лексичного тестування граматики було проведено тестування токенізації, тести на спеціальні символи, граничні випадки, порожні файли, зайві пробіли та довгі вирази.

Для побудови дерева виводу граматики було виконано вивід завдання, в якому є параметризовані змінні, треки зі вкладеннями етапів та окремі етапи з перевірочною дією. Опис цього завдання представлено в лістингу 1.

```
task "complex_task": {
  variables = [  variable "topvar1": {
       type = "RandomString" }]
  tracks = [ track "t1": {
          stages = [ stage "stage1": {
                    checks = [check "check1": {action = "a1" }]},
                stage "stage2": {
                    checks = [ check "check2": {action = "a2"}]}]}]
```

Лістинг. 1. Опис практичного завдання з кількома етапами

Джерело: власна розробка.

У результаті синтаксичного та лексичного аналізу побудовано дерево виводу, представлене на рис. 1.

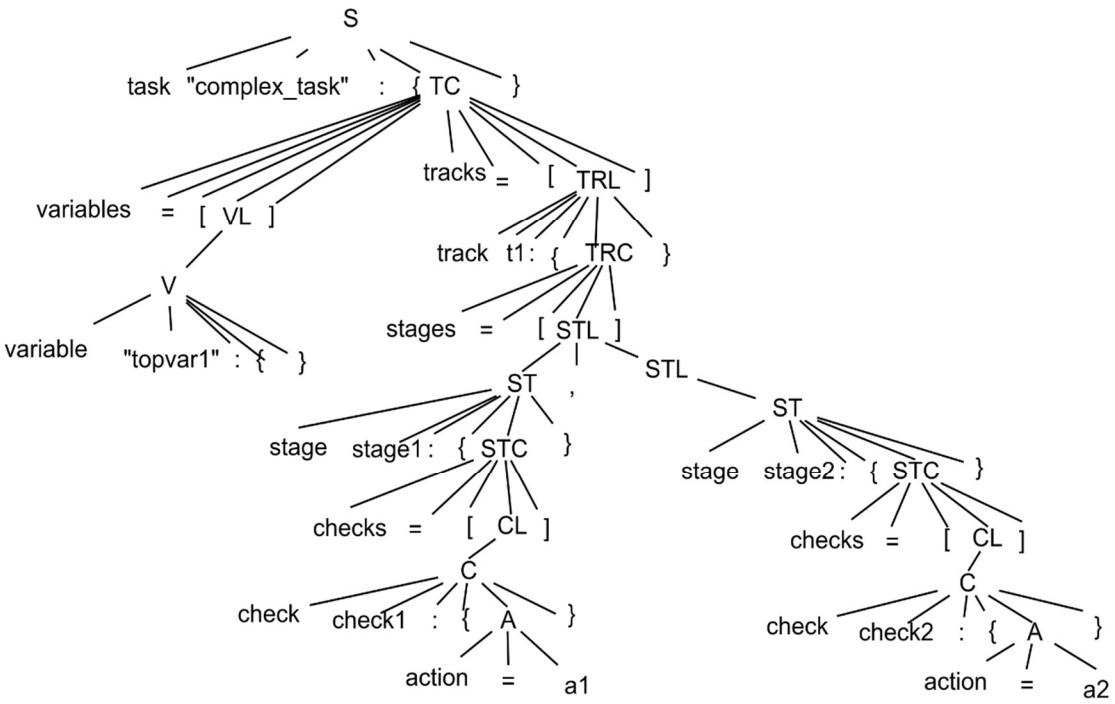


Рис. 1. Дерево виводу опису практичного завдання з кількома етапами

Джерело: власна розробка.

Окремим, але важливим елементом практичного завдання є опис навчального середовища, тому для нього теж було побудовано синтаксичне дерево. Для прикладу було створено опис навчального середовища, яке являє собою віртуальну машину, запущену за допомогою гіпервізора VirtualBox. Для налаштування середовища описані інструкції, які створюють користувача і задають йому пароль. Опис навчального середовища представлено в лістингу 2.

```
task "just_le" : { learning_environment "main": {
        variables = [  variable "login: {type = "RandomString" },
                       variable "pass: {type = "RandomString" }]
        create_instructions = [
                instruction "startvm": {
                        action = "VBoxManage startvm"}]
        provision_instructions = [
                instruction "create_user": {
                        action = "create user ${main_le.login}"},
                instruction "set_password": {
                        action = "echo  ${main_le.login} | passwd ${main_le.pass}" }]}}
```

Лістинг 2. Опис навчального середовища з кількома інструкціями

Джерело: власна розробка.

У результаті синтаксичного та лексичного аналізу побудовано дерево виводу, яке представлене на рис. 2.

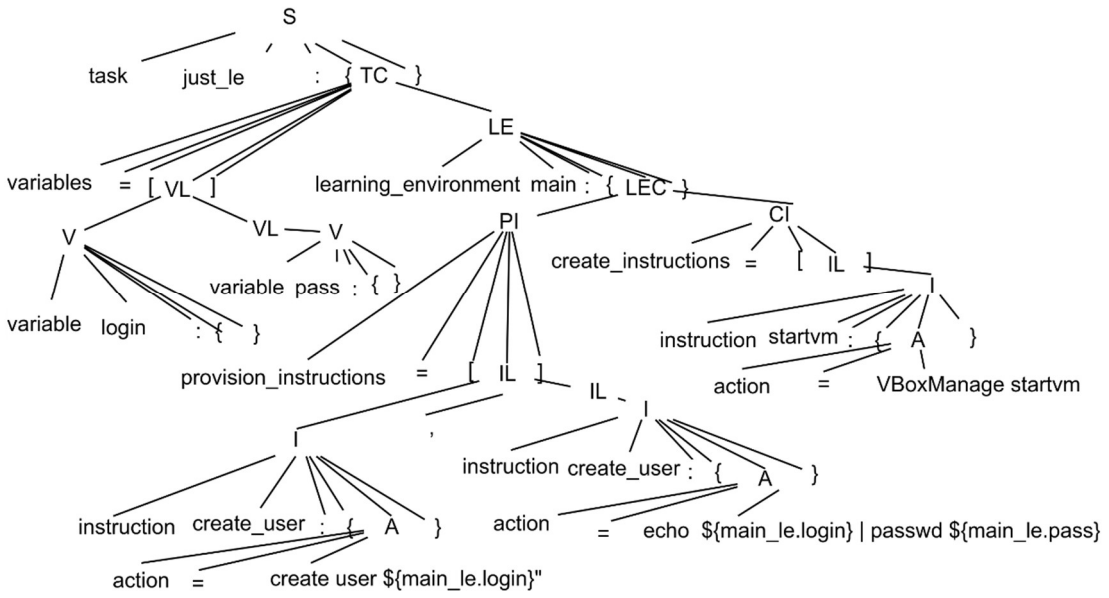


Рис. 2. Дерево виводу опису навчального середовища з кількома інструкціями за допомогою LTDL

Джерело: власна розробка.

Використовуючи алгоритми тестування граматики, які наведені у монографії [13], було проведено визначення непродуктивних та недосяжних символів. Символ A є продуктивним, якщо існує вивід A ⇒* w, де w – рядок термінальних символів. Символ A є непродуктивним, якщо не існує виводу A ⇒* w для жодного рядка w, що складається лише з термінальних символів.

Після повного аналізу граматики виявляється, що множина непродуктивних символів є пустою множиною, і всі нетермінальні символи можуть вивести термінальні рядки. Це досягається завдяки:

1) Використанню оператора * для опціональних повторень;
2) Наявності альтернативних правил виводу;
3) Правильній структурі граматики без циклічних залежностей;
4) Можливості виводу мінімальних конструкцій для кожного правила.

Символ A є досяжним, якщо існує вивід $S \Rightarrow^* \alpha A \beta$, де S – стартовий символ, α і β – довільні рядки символів.

Символ A є недосяжним, якщо не існує виводу $S \Rightarrow^* \alpha A \beta$ для жодних α і β.

Граматика LTDL не містить недосяжних символів. Усі нетермінальні символи можуть бути досягнуті від стартового символа task через певну послідовність виводів.

Ключові особливості, які забезпечують досяжність всіх символів:

– Ієрархічна структура – граматика побудована як дерево з чітким корінням (task);
– Центральне правило – аксіома, що об'єднує всі основні блоки граматики;
– Взаємозв'язки – всі структури пов'язані через правила виводу;
– Відсутність ізольованих правил – немає правил, що не використовуються.

Це означає, що граматика не містить зайвих символів та всі визначені правила мають практичне застосування і можуть бути досягнуті під час парсингу.

4. Приклад представлення персоналізованого практичного завдання мовою LTDL.

Для демонстрації можливостей мови зроблений опис прикладу персоналізованого практичного завдання за допомогою LTDL нотації та графічним способом. У якості прикладу створимо завдання для вивчення команд роботи з файлами та директоріями в ОС на базі ядра Linux. У цьому завданні пропонується знайти файл, який містить наперед заданий текст, і скопіювати його в директорію. Перед тим, як копіювати знайдений файл, треба створити цільову директорію. Для ускладнення завдання і внесення елементів гейміфікації через пошук і усунення першопричини проблеми додаємо в файлову систему паразитний файл із таким самим іменем, як цільова директорія, що унеможливлює її створення. Отже, у завданні буде 4 етапи, розбиті на 2 треки, які можна почати виконувати паралельно, але є взаємозалежність: 1) перш ніж, як скопіювати файл, його треба знайти та створити директорію, 2) перш ніж, як створити директорію, треба видалити паразитний файл.

Також треба задати 3 параметризовані змінні – змінна "pattern" рівня етапу "find_file", яка буде використовуватися тільки в цьому етапі, змінна "file_name" рівня треку, яка буде використовуватися в обох етапах треку, і змінна "dir_name" рівня завдання, яка буде використовуватися в етапах обох треків. На цьому прикладі можна продемонструвати випадки, коли етап не містить опису завдання (етап створення паразитного файлу), але містить перевірочні інструкції. Таким чином студент повинен зрозуміти наявність перепони й усунути її, а результат виконання цієї дії повинен бути зафіксований в фінальному звіті про виконання завдання. Натомість етап створення файлу зі згенерованим текстом, який студент повинен знайти, не містить перевірочних умов, оскільки в результаті виконання цього етапу потрібний файл повинен опинитися в цільовій директорії, що і буде успішним виконанням цього етапу. Тільки так можна зрозуміти, що файл був знайдений. Іноді буває так, що правильні команди виконані, але студент не розібрався з результатом роботи команди.

Для спрощення сприйняття та можливості охоплення усіх аспектів розробленого персоналізованого практичного завдання авторами пропонується можливість графічного представлення елементів завдання. На рис. 3 показано, як наведений вище приклад візуалізується за допомогою графічного варіанту мови.

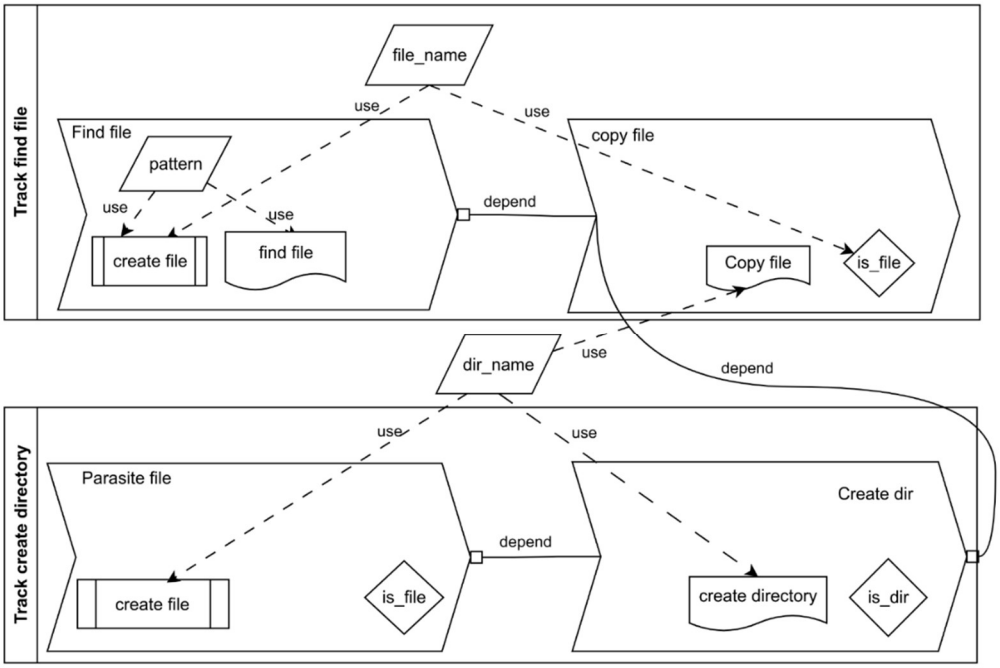


Рис. 3. Приклад представлення персоналізованого практичного завдання графічним варіантом мови LTDL

Джерело: власна розробка.

Висновки. У дослідженні вперше запропоновано формальну доменно-специфічну мову Learning Task Definition Language (LTDL), яка, на відміну від існуючих DSL-рішень, фокусується на мікрорівні завдань, глибокій параметризації, описі структури, залежностей, розгортанні навчальних середовищ та гнучких перевірках. Розроблена мова може поєднуватися з генеративними AI-системами для контрольованого процесу автоматичної генерації персоналізованих практичних завдань для інженерних спеціальностей.

Розроблена формальна граматика мови, проведене тестування створеної граматики та згенерованих лексичного та синтаксичного аналізатора, розроблене графічне представлення мови. Графічне представлення мови дозволяє викладачам охопити всі аспекти персоналізованого практичного завдання та спрощує його огляд та аналіз.

Подальші дослідження будуть спрямовані на:

1) інтеграцію LTDL у навчальні платформи та розробку методів оцінки ефективності застосування мови у реальних освітніх сценаріях;

2) підтримку конкретних рівнів персоналізації практичних завдань [14];

3) дослідження інтеграції LTDL з генеративними моделями для контрольованої автоматизації;

4) створення комбінованого підходу до персоналізації навчання завдяки поєднанню переваг формальної мови (структурованість, перевірюваність) та можливостей AI (масштабованість, варіативність).

Заява про використання генеративного ШІ та технологій на основі ШІ в процесі написання тексту статті

Автори використали ШІ (Chat GPT) для покращення читабельності та виправлення стилістичних і граматичних помилок у цій статті. Після використання цього інструменту автори переглянули та відредагували зміст за потреби і взяли на себе повну відповідальність за зміст публікації.

Список використаних джерел

1. Liang, Percy & Jordan, Michael & Klein, Dan. (2010). Learning Programs: A Hierarchical Bayesian Approach. *Proceedings of the 27th International Conference on Machine Learning (ICML-10),* 639-646.

2. Jiménez, David & Guerrero, Ana-Elena & Prieto-Blazquez, Josep & Conesa, Jordi. (2014). A domain-specific language for Virtual Classrooms. *International Journal of Metadata Semantics and Ontologies*, *9,* 312-323. https://doi.org/10.1504/IJMSO.2014.065444.

3. Westphal, Oliver. (2020). A Framework for Generating Diverse Haskell-IO Exercise Tasks. *Pre-proceedings of the 28th International Workshop on Functional and Logic Programming (WFLP 2020)* https://doi.org/10.48550/arXiv.2008.12751.

4. Westphal, Oliver & Voigtländer, Janis. (2020). Describing Console I/O Behavior for Testing Student Submissions in Haskell. *Electronic Proceedings in Theoretical Computer Science*,*321,* 19-36. https://doi.org/10.4204/EPTCS.321.2.

5. González García, Cristian & Núñez Valdez, Edward & Moreno Ger, Pablo & Gonzalez Crespo, Ruben & Pelayo García-Bustelo, B. & Cueva Lovelle, Juan. (2019). Agile development of quiz-based multiplatform educational games using a Domain-Specific Language. *Universal Access in the Information Society.* IP. 1-20. https://doi.org/10.1007/s10209-019-00681-y.

6. Willert, N., Thiemann, J. Template-Based Generator for Single-Choice Questions. (2024). *Tech Know Learn,* 29, 355-370. https://doi.org/10.1007/s10758-023-09659-5.

7. Хижняк, А.В., & Пріла, О.А. (2025). Розробка системи автоматизованої генерації та перевірки параметризованих практичних завдань. *Технічні науки та технології*, *(2 (40),* 221-233. https://doi.org/10.25140/2411-5363-2025-2(40)-221-233.

8. Wasowski, A., & Berger, T. (2023). Domain-Specific Languages: Effective Modeling, Automation, and Reuse. *Springer*.

9. Ramos-Díaz, J. G., Navarro, I., Silva, J., & Arroyo, G. (2012). Defining DSL design principles for enhancing the requirements elicitation process. *Acta* Universitaria*, 22,* 126-133. https://doi.org/10.15174/au.2012.352.

10. Strembeck, M., & Zdun, U. (2009). An Approach for the Systematic Development of Domain-Specific Languages. *Software: Practice and Experience* (*SP&E), 39(15)*, 1253-1292. http://dx.doi.org/10.1002/spe.936.

11. Mernik M., Heering J., Sloane A.M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*. http://dx.doi.org/10.1145/1118890.1118892.

12. Стативка, Ю. І. (2023). *Формальні мови. Основні концепти і представлення*. КПІ ім. Ігоря Сікорського. https://ela.kpi.ua/items/5f179fe8-05de-4b23-9563-13fd4d24e37e.

13. Гавриленко С.Ю. (2021). Формальні мови, граматики та автомати. НТУ ХПІ. https://repository.kpi.kharkov.ua/server/api/core/bitstreams/5847efdd-6ff5-4f7f-9de8-6f6555ad4cc0/content.

14. Хижняк, А., & Казимир, В. (2025). Узагальнена класифікація рівнів персоналізації практичних завдань в іт- освіті. *Наука і техніка сьогодні, (7(48)).* https://doi.org/10.52058/2786-6025-2025-7(48)-1932-1949.

References

1. Liang, Percy & Jordan, Michael & Klein, Dan. (2010). Learning Programs: A Hierarchical Bayesian Approach. *Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp.* 639-646.

2. Jiménez, David & Guerrero, Ana-Elena & Prieto-Blazquez, Josep & Conesa, Jordi. (2014). A domain-specific language for Virtual Classrooms. *International Journal of Metadata Semantics and Ontologies*. 9. pp. 312-323. DOI: https://doi.org/10.1504/IJMSO.2014.065444.

3. Westphal, Oliver. (2020). A Framework for Generating Diverse Haskell-IO Exercise Tasks. *Pre-proceedings of the 28th International Workshop on Functional and Logic Programming (WFLP 2020)* DOI: https://doi.org/10.48550/arXiv.2008.12751.

4. Westphal, Oliver & Voigtländer, Janis. (2020). Describing Console I/O Behavior for Testing Student Submissions in Haskell. *Electronic Proceedings in Theoretical Computer Science*. 321. 19-36. DOI: https://doi.org/10.4204/EPTCS.321.2.

5. González García, Cristian & Núñez Valdez, Edward & Moreno Ger, Pablo & Gonzalez Crespo, Ruben & Pelayo García-Bustelo, B. & Cueva Lovelle, Juan. (2019). Agile development of quiz-based multiplatform educational games using a Domain-Specific Language. *Universal Access in the Information Society.* IP. 1-20. DOI: https://doi.org/10.1007/s10209-019-00681-y.

6. Willert, N., Thiemann, J. Template-Based Generator for Single-Choice Questions. (2024). *Tech Know Learn* 29, 355-370. DOI: https://doi.org/10.1007/s10758-023-09659-5.

7. Khyzhniak A.V., Prila O.A. Rozrobka systemy avtomatyzovanoi heneratsii ta perevirky parametryzovanykh praktychnykh zavdan. [Designing a system for automated generation and automated assessment of parameterized practical assignments.] *Tekhnichni nauky ta tekhnolohii - Technical sciences and technologies, Vol.2 (40)*, 2025. pp.221-233. DOI: https://doi.org/10.25140/2411-5363-2025-2(40)-221-233.

8. Wasowski, A., & Berger, T. (2023). Domain-Specific Languages: Effective Modeling, Automation, and Reuse. *Springer*.

9. Ramos-Díaz, J. G., Navarro, I., Silva, J., & Arroyo, G. (2012). Defining DSL design principles for enhancing the requirements elicitation process. *Acta Universitaria*, 22, pp. 126-133. DOI: https://doi.org/10.15174/au.2012.352.

10. Strembeck, M., & Zdun, U. (2009). An Approach for the Systematic Development of Domain-Specific Languages. *Software: Practice and Experience* (SP&E), 39(15), pp. 1253-1292. DOI: http://dx.doi.org/10.1002/spe.936.

11. Mernik M., Heering J., Sloane A.M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*. DOI: http://dx.doi.org/10.1145/1118890.1118892.

12. Statyvka, Y. I. Formalni movy. Osnovni kontsepty i predstavlennia. [Formal language. Basic concepts and representations.] KPI im. Ihoria Sikorskoho – *Kyiv Polytechnic Institute named after Ihor Sikorskyi* – Available from: https://ela.kpi.ua/items/5f179fe8-05de-4b23-9563-13fd4d24e37e.

13. Havrylenko S.Y. (2021). Formalni movy, hramatyky ta avtomaty. [Formal Languages, Grammars, and Automata.] Textbook. Kharkiv: NTU KhPI. – 133 p. –Available from: https://repository.kpi.kharkov.ua/server/api/core/bitstreams/5847efdd-6ff5-4f7f-9de8-6f6555ad4cc0/content.

14. Khyzhniak A.V., Kazymyr V.V. Uzahalnena klasyfikatsiia rivniv personalizatsii praktychnykh zavdan v it- osviti. [A generalized classification of personalization levels in practical assignments for IT-education.] *Nauka i tehnica syogodni – Science and technology today*, 7(48), 2025. pp.1932-1949. DOI: https://doi.org/10.52058/2786-6025-2025-7(48)-1932-1949. – Available from http://perspectives.pp.ua/index.php/nts/article/view/26982/26951.

Отримано 08.09.2025

UDC 004.42:37.091.3:378.018.43

Andrii Khyzhniak[1], Volodymyr Kazymyr[2]

[1]PhD Student, Senior Lecturer at Information and Computer Systems Department
Chernihiv Polytechnic National University (Chernihiv, Ukraine)
E-mail: alf.andrey@gmail.com. **ORCID:** http://orcid.org/0009-0008-8655-3736

[2]Doctor of Sciences, Professor, Professor of the Department of Information and Computer Systems
Chernihiv Polytechnic National University (Chernihiv, Ukraine)
E-mail: vvkazymyr@gmail.com. **ORCID:** https://orcid.org/0000-0001-8163-1119. **ResearcherID:** Q-2925-2016

A DOMAIN-SPECIFIC LANGUAGE FOR DESCRIBING PERSONALIZED PRACTICAL TASKS IN ENGINEERING EDUCATION

In contemporary education, there is a growing demand for scalable and personalized tools that can automate practical task creation, while preserving academic integrity.

An analysis of recent studies has shown that existing domain-specific languages (DSLs) used in an educational context have a number of limitations, are mostly focused on the macro level (courses, virtual classrooms, quizzes), and do not provide a formalized and flexible description of practical tasks at the micro level.

This article presents Learning Task Definition Language (LTDL) for the first time – a formal DSL designed to describe the structure of practical tasks and the stages of their implementation. The proposed language has a specific focus on the micro level of tasks, deep parameterization, description of learning environments, automatic generation, and flexible verification. LTDL supports both textual and graphical representations of tasks, which increases its usability for teachers and the possibility of integration with automatic deployment and verification systems.

Testing results demonstrate the correctness of LTDL grammar, the absence of unproductive and unreachable symbols, and the language's ability to model complex scenarios. The novelty of the research lies in the creation of a flexible formal tool that can combine task personalization with controlled generation of text instructions, automated deployment of learning environments, and rules for checking their progress. LTDL can be integrated into modern information systems for automated generation and evaluation of parameterized practical tasks.

Keywords: *practical assignments; automated generation; automated assessment; personalized learning; formal grammar; domain-specific language.*

Fig.: 3. Table: 2. References: 14.

Хижняк А. В., Казимир В. В. Доменно-орієнтована мова опису персоналізованих практичних завдань для інженерних спеціальностей. *Технічні науки та технології,* 2025. № 3(41). С. 261-271. DOI: https://doi.org/10.25140/2411-5363-2025-3(41)-261-271.