УДК 004.[432.4:514]
DOI: 10.18523/2617-3808.2021.4.108-112

Іванюк Н. О., Кучер А. О., Ющенко Ю. О.

РЕАЛІЗАЦІЯ ЗАСОБУ РОЗРОБЛЕННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ ДЛЯ ПРОГРАМ НА МОВІ ПРОЛОГ

Роботу присвячено актуальним проблемам поширення використання логічного програмування при розробленні промислових і комерційних багатоплатформних програмних застосунків, а саме запропонованому засобу зручного розроблення сучасного графічного інтерфейсу до логічних програм. Описано запропоновану концепцію потокового інтерфейсу до Прологу та розроблену бібліотеку Panzer Prolog. Визначено переваги цієї концепції порівняно з наявними засобами приєднання графічного інтерфейсу до програм на мові Пролог. Зазначено шляхи подальшого вдосконалення можливостей реалізованої бібліотеки.

Ключові слова: інтерфейс, Prolog, логічне програмування, пролог, відкритий код, публічна бібліотека, клієнт-серверне застосування, Node.js, JavaScript, JSON, TypeScript, child process, асинхронність, кросплатформність, маніпулювання запитів.

У галузі інформаційних технологій набувають популярності додатки з ознаками розумності. Програмні застосунки зі штучним інтелектом цікавлять користувачів і замовників інформаційних технологій, а розробники програмного забезпечення приділяють все більше уваги засобам розроблення систем штучного інтелекту. Фахівці та вчені прогнозують, що вже найближчим часом системи штучного інтелекту застосовуватимуть повсюдно. Декларативні мови програмування, зокрема логічне програмування, займають свою, окрему нішу серед цих засобів.

Існує помилкова думка, що в мові програмування PLANNER (1965 р.) вперше були наявні можливості декларативних засобів логічного програмування. Насправді Адресна мова програмування (1955 р.) високого рівня [1] має вичерпні можливості декларативного програмування [4], зокрема в ній є унікальна «мінус штрих-операція», яка має виключно декларативний характер. Як показано в [4], можливості «мінус штрих-операції» Адресної мови покривають можливості конструкції JOIN мови SQL, а сама мова має вичерпні декларативні можливості.

Однак концепція декларативного програмування набула широкого визнання після винаходу у 1974 р. логічного програмування, якому вчені пророкували велику перспективу [5]. Загальновідомими є переваги логічного програмування порівняно з імперативними засобами програмування при реалізації багатьох класів задач. Насамперед до таких задач належать логічні ігри, логічні задачі, задачі, пов'язані з необхідністю здійснювати перебір, комбінаторні задачі, задачі символьного оброблення даних тощо. Логічне програмування також успішно застосовують для розроблення складних експертних систем.

Однак поширенню логічного програмування, починаючи від часів його виникнення, перешкоджає низка факторів. Одним із таких є брак програмістів, які вільно володіють мовою Пролог. Цей фактор частково зумовлений суттєвою відмінністю концепції декларативного програмування від широковживаної концепції імперативного програмування. Ще одним із чинників, який перешкоджає поширенню використання мови Пролог, є те, що інтерпретаторам мови Пролог бракує можливостей приєднання сучасного графічного інтерфейсу до логічних програм.

Логічне програмування неухильно набуває популярності у розробників інформаційних технологій попри наявність стримувальних факторів поширення його використання.

Також створюють багатоконцептуальні мови програмування, які об'єднують функційне програмування та логічне програмування. Яскравим прикладом такої інтеграції є цікава, як із теоретичного погляду, так і з практичного, мова Curry [6]. Нещодавно корпорація Google запропонувала новий засіб Logica [7] для спрощення складання SQL-запитів, оскільки ці запити можуть займати багато рядків і є неочевидними для сприйняття. Бібліотека Logica розміщена у відкритому доступі. Є змога запити з Logica компілювати в запити на мові SQL.

Наявні засоби приєднання графічного інтерфейсу не можуть повною мірою задовольнити потреби програмістів і, окрім своїх корисності та популярності, мають певні недоліки. Покращення зручності для програмістів і спрощення процесу створення сучасного графічного інтерфейсу користувачів для логічних програм сприятиме подальшому поширенню використання логічного програмування та є актуальним для розвитку засобів розроблення інтелектуальних інформаційних технологій.

У статті досліджено можливості та недоліки наявних способів розроблення графічного інтерфейсу для логічного програмування. Запропоновано концепцію потокового інтерфейсу до Прологу, яка узагальнює наявні способи розроблення графічного інтерфейсу. Згідно з цією концепцією реалізовано універсальну бібліотеку Panzer Prolog (Node-swipl-io) [8] потокового інтерфейсу для SWI-Prolog. Універсальність бібліотеки полягає в її легкій інтеграції, безпечності типів і можливості масштабування.

Бібліотека Panzer Prolog є вже вельми популярною серед програмістів, її використовують при програмуванні застосунків на мові Пролог. За місяць після розміщення бібліотеку скачано майже 200 разів.

Реалізація універсального методу в бібліотеці Panzer Prolog посилює конкурентну спроможність логічного програмування серед інших засобів програмування, оскільки вона надає клієнту можливість застосовування популярних, широковживаних засобів і каркасів розроблення графічного інтерфейсу, таких як: JavaScript, React, Vue, Angular тощо. Каркаси є популярними засобами створення користувацького інтерфейсу, працюють в усіх браузерах та доступні у багатьох середовищах, постійно підтримуються і розвиваються. Цим зумовлено доцільність використання мови JavaScript бібліотеки середовища Node.js задля пов'язування процесу інтерпретації логічних програм із діями кінцевих користувачів при використанні додатків із графічним інтерфейсом. Для вдосконалення можливостей роботи з типами даних розпочато застосування мови TypeScript до розробленої бібліотеки, що дасть змогу під час програмування та налагодження програм автоматично формувати підказки програмістам.

1. Огляд засобів розроблення графічного інтерфейсу для програм на Пролозі

Можливості та інтерфейс, який пропонує фірма PDC у відомій мові Visual Prolog, не відповідають вимогам розроблення сучасного графічного інтерфейсу. Visual Prolog не підтримує кросплатформних додатків, має застарілий дизайн інтерфейсу, у ньому немає можливості розроблення вебзастосунку.

До одного з поширених засобів приєднання графічного інтерфейсу до програм на мові Пролог належить досить відома бібліотека Pengines [9]. В її основу покладено створення HTTP-серверу, який можна налаштувати до потреб користувача. На сайті є можливість онлайнового програмування та інтерпретації програм на Пролозі.

Зручним і популярним засобом приєднання графічного інтерфейсу до мови пролог є Tau-Prolog, який являє собою інтерпретатор мови Prolog на JavaScript. В основі його роботи лежить розбір запитів мови Пролог за допомогою граматик. Tau-Prolog можна застосовувати як напряму у браузері, так і на серверних застосунках. Інтерпретатор Tau-Prolog з інструкцією та описом його синтаксису розміщено у відкритому доступі [10; 11]. Однак Tau-Prolog через інтерпретацію коду, написаного на мові Prolog та його інтерпретації на JavaScript, втрачає переваги концепції логічного програмування. Зокрема, на офіційному сайті розробника зазначено, що бібліотека Tau-Prolog ще не до кінця розвинута для того, щоб витримувати запити з глибокою рекурсією [11].

Наявні бібліотеки, хоч і створюють засоби зв'язку між мовою Пролог та JavaScript, все ще не відповідають сучасним потребам. Наразі розвиток більшості таких бібліотек триває повільно. Бібліотека node-swipl та її послідовниця node-swipl-stdio мають іншу архітектуру та досконаліші засоби розроблення інтерфейсу. Особливість принципів роботи цих засобів полягає у створенні та використанні потоків для комунікації дій кінцевого користувача з Прологом. У node-swipl немає підтримки Юнікоду, нових версій мови Пролог і визначення типів даних. Більш досконала бібліотека node-swipl-stdio хоча і вирішила частину проблем з node-swipl, але також не має паралельного виконання, а недоліки її архітектури все ще можуть призводити до аварійного завершення роботи застосунку. Жодна концепція наявних бібліотек не відповідає повністю сучасним вимогам, а брак належної підтримки бібліотек викликає недовіру до них. Це наштовхнуло авторів на необхідність розроблення концепції асинхронних паралельних потоків обміну різними типами даних між кінцевими користувачами та інтерпретатором Прологу. Її безпечність полягає в тому, що

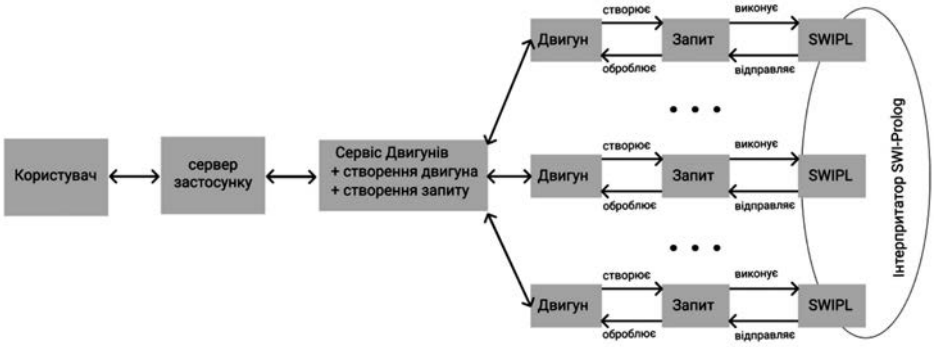

Рис. 1. Принцип роботи за концепцією потокового інтерфейсу до Прологу

всі типи даних та їх сукупності приводяться до стандартного популярного формату, надійність якого автоматично перевіряється наявною в бібліотеці системою тестування.

Розроблена бібліотека задовольняє нагальні потреби щодо приєднання сучасного користувацького інтерфейсу до програм на Пролозі. Передання чітко структурованої, зручної для використання інформації здійснюється потоками та не потребує подальшого перетворення.

2. Опис концепції потокового інтерфейсу до Прологу

Запропонована концепція потокового інтерфейсу до Прологу повною мірою використовує можливості, які надаються асинхронними паралельними потоками з підтримкою їх розподіленого виконання. Безпечність обміну інформації зумовлено тим, що потоки виконуються сервером та втручатися ззовні у їхню роботу неможливо.

За кожний окремий потік обміну повідомленнями між інтерпретатором Прологу та програмою на JavaScript відповідає власний двигун, який має стани і чергу запитів, що дає змогу уникнути недоліків масштабування наявних бібліотек. Двигунами керує сервіс, який оптимально розподіляє навантаження між потоками.

Для програмістів передбачено прості засоби створення та контролю коректності запитів кінцевих користувачів до інтерпретатора Прологу. Сервіс двигунів відповідає за створення процесів, відстежує виклики, обробляє запити та передбачає перевірку синтаксичної коректності запитів. При створенні нового запиту сервіс двигунів перенаправляє його на вільний двигун, який перебуває у стадії очікування. Двигун створює у себе внутрішній компонент запиту для взаємодії через потік з інтерпретатором Прологу. Інтерпретатор формує відповідь, яка після оброблення двигуном повертається кінцевому користувачу як результат його запиту. Концептуальну схему взаємодії компонентів наведено на рис. 1.

Порівняно з Tau-Prolog (рис. 2) концептуальна різниця полягає у тому, що немає етапу інтерпретації коду за допомогою граматик, що пришвидшує виконання коду та вилучає відповідні розрахунки на комп'ютері користувача.

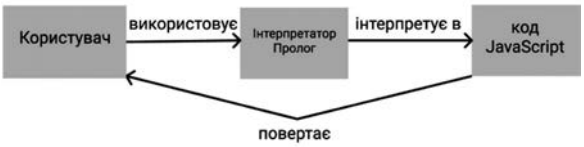

Рис. 2. Принцип роботи Tau-Prolog

Із рисунків 1 і 2 видно переваги концепції потокового інтерфейсу до Прологу над бібліотекою Tau-Prolog, оскільки остання передбачає повернення користувачу результату лише після завершення інтерпретації та унеможливлює до завершення інтерпретації попереднього запиту формування користувачем нових запитів.

Концепція потокового інтерфейсу до Прологу передбачає можливість строгої типізації для забезпечення своєчасного виявлення помилок і доповнення бібліотеки іншими компонентами.

3. Функції бібліотеки Panzer Prolog

Основні функції запропонованої концепції реалізовано в бібліотеці Panzer Prolog, яка є зручним та якісним сервісом програміста для інтеграції графічного інтерфейсу до програм на Пролозі. Бібліотека Panzer Prolog із відкритим кодом, описом її призначення та керівництвом програміста доступна за посиланням [8].

Для використання бібліотеки програмісту потрібно володіти мовою розробки у середовищі Node.js та вміти програмувати на Пролозі.

Для роботи з бібліотекою потрібно мати одне з наявних середовищ розроблення програмного забезпечення, підключити інсталяцію інтерпретатора мови SWI-Prolog та платформу Node.js. Бібліотеку Panzer Prolog можна встановити за допомогою пакетного менеджера NPM.

Триває вдосконалення бібліотеки шляхом використання мови TypeScript. Це дасть змогу посилити безпеку даних і запобігти помилок при перетворенні типів даних. TypeScript забезпечить безпомилкове виконання підготовлених запитів на Пролозі.

Для кожного потоку створюється запит, що обробляється окремим інтерпретатором Прологу. У разі виникнення помилки під час виконання процедури двигун зупиняється та потребує повторного створення. Також двигун має функції, які є кориснимиими для контролю створення запитів до Прологу.

Висновки

У результаті розроблення концепції потокового інтерфейсу до Прологу та реалізації бібліотеки Panzer Prolog створено зручний засіб взаємодії графічного вебінтерфейсу, написаного на JavaScript, із процесом інтерпретації запитів у SWI-Prolog. Використання бібліотеки зменшує розмір початкових текстів та надає можливість використовувати клієнт-серверну технологію без занурення у її реалізацію.

Кінцеві користувачі отримають можливість, працюючи з вебдодатками, отримувати результати роботи інтерпретатора Прологу у сучасному графічному інтерфейсі.

Отримані результати сприятимуть подальшому поширенню використання логічного програмування, а проведені дослідження та їх реалізація у спеціалізованій бібліотеці Panzer Prolog мають велике теоретичне та практичне значення для розвитку інформаційних технологій.

Список літератури

1. Ющенко Е. Л. Адресное программирование / Е. Л. Ющенко. – Киев : Гос. издательство технической литературы, УРСР, 1963. – 287 с. : іл.
2. Ющенко Ю. О. Вступ до логічного програмування : навч. посібник / Ю. О. Ющенко. – Київ : Вид-во Європ. ун-ту, 2006. – 116 с. : іл.
3. Ющенко Ю. О. Засоби керування виконанням логічних програм / Ю. О. Ющенко // Збірник наукових праць МСУ. – 2005. – С. 106–171.
4. Ющенко Ю. О. Окремі аспекти декларативності «мінус штрих-операції» [Електронний ресурс] / Ю. Ющенко // Наукові записки НаУКМА. – 2020. – Т. 3 : Комп'ютерні науки. – С. 19–26. – Режим доступу: https://doi.org/10.18523/2617-3808.2020.3.17-26.
5. Ющенко Ю. О. Перспективи логічного програмування / Ю. О. Ющенко // Збірник праць міжнародної науково-технічної конференції «Авіа-2003» (Київ, квітень 2003 р.). – Київ : Вид-во НАУ, 2003. – С. 14.187–14.190.
6. Curry Programming Language [Elecronic resource]. – Mode of access: https://curry.pages.ps.informatik.uni-kiel.de/curry-lang.org/.
7. Logica [Elecronic resource]. – Mode of access: Logica: organizing your data queries, making them universally reusable and fun | Google Open Source Blog (googleblog.com) https://opensource.googleblog.com/2021/04/logica-organizing-your-data-queries.html.
8. Panzer Prolog (Node-swipl-io) [Elecronic resource]. – Mode of access: https://www.npmjs.com/package/node-swipl-io.
9. Pengines [Elecronic resource]. – Mode of access: https://pengines.swi-prolog.org/docs/getting_started.html.
10. Tau Prolog Grammar specification [Elecronic resource]. – Mode of access: http://tau-prolog.org/files/doc/grammar-specification.pdf.
11. Tau Prolog, an Amazing Prolog Interpreter Fully In JavaScript [Elecronic resource]. – Mode of access: https://phpmagazine.net/2018/11/tau-prolog-a-prolog-interpreter-fully-in-javascript.html.

References

Curry Programming Language. Retrieved from https://curry.pages.ps.informatik.uni-kiel.de/curry-lang.org.

Logica: organizing your data queries, making them universally reusable and fun | Google Open Source Blog (googleblog.com). Retrieved from https://opensource.googleblog.com/2021/04/logica-organizing-your-data-queries.html.

Panzer Prolog (Node-swipl-io). Retrieved from https://www.npmjs.com/package/node-swipl-io.

Pengines. Retrieved from https://pengines.swi-prolog.org/docs/getting_started.html.

Tau Prolog Grammar specification. Retrieved from http://tau-prolog.org/files/doc/grammar-specification.pdf/.

Tau Prolog, an Amazing Prolog Interpreter Fully In JavaScript. Retrieved from https://phpmagazine.net/2018/11/tau-prolog-a-prolog-interpreter-fully-in-javascript.html.

Yushchenko, E. L. (1963). *Adresnoe prohrammyrovanye*. Kyev: Hos. yzdatelstvo tekhnycheskoi lyteratury [in Russian].

Yushchenko, Yu. O. (2003). Perspektyvy lohichnoho prohramuvannia. *Zbirnyk prats mizhnarodnoi naukovo-tekhnichnoi konferentsii "Avia-2003" (Kyiv, kviten 2003 r.)* (pp. 187–190). Kyiv: Vyd-vo NAU [in Ukrainian].

Yuschenko, Yu. O. (2005). Zasoby keruvannia vykonanniam lohichnykh prohram. *Zbirnyk naukovykh prats MSU*, 106–171 [in Ukrainian].

Yuschenko, Yu. O. (2006). *Vstup do lohichnoho prohramuvannia*. Kyiv: Vyd-vo Yevrop. un-tu [in Ukrainian].

Yushchenko, Yu. O. (2020). Okremi aspekty deklaratyvnosti "minus shtrykh-operatsii". *Naukovi zapysky NaUKMA, 3 : Kompiuterni nauky*, 19–26. Retrieved from https://doi.org/10.18523/2617-3808.2020.3.17-26 [in Ukrainian].

N. Ivaniuk, A. Kucher, Y. Yuschenko

IMPLEMENTATION OF A GRAPHIC INTERFACE DEVELOPMENT TOOL FOR PROLOG

The work examines the current problems of the spread of use of logical programming in the development of commercial multi-platform software applications, tools for convenient development of a modern graphical interface to the logical programs. Libraries with similar concepts of use have been analyzed and described. The purpose of the proposed concept, which is implemented as an open source library, is described, and the advantages of the proposed tools over similar existing tools are indicated. The main feature and advantage of the proposed concept is the implementation of Prolog business logic and interface by means of JavaScript usage of child processes. The proposed concept of interface to Prolog takes full advantage of the possibilities provided by async await. A framework library has been created for the use of Logic Programming in graphical interface development without losses in the application performance. The paper describes the proposed concept and the developed framework (library). The ways to further improve the possibilities for expanding the purpose of the implemented library were identified. The directions of further simplification for programmers of integration of the graphic interface to logical programs have been defined. A significant advantage of the proposed tool is the easy-to-use functions to wrap and control the correctness of requests to the Prolog. The main goal of the library is to create an environment for the Prolog developers where they can create any type of software, which is meant to be user friendly, fast, and cross platform using modern and flexible. This concept also tries to solve disadvantages and architectural problems that were found in other libraries. The safety of library functionality has been analyzed. The concept of potential horizontal application scalability is described. Conclusions and future of libraries were introduced, in which the usage of TypeScript for type-safety and avoidance of run-time errors is mentioned. Overall, the library extends the use of Prolog beyond logical programming and takes a leap forward in its progress.

Keywords: interface, Prolog, logical programming, open source, public library, client-server application, Node.js, JavaScript, JSON, TypeScript, child process, frontend frameworks, asynchronous, cross-platform, query manipulation.

Матеріал надійшов 22.05.2021