УДК 681.3.06
DOI https://doi.org/10.32782/2663-5941/2023.5/34

Редько І.В.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Яганов П.О.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Зилевіч М.О.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

КОНЦЕПТОЛОГІЧНІ ЗАСАДИ ТЕХНОЛОГІЧНИХ СИСТЕМ ПРОГРАМУВАННЯ

У статті досліджена і обґрунтована необхідність парадигмальних змін у технологічних основах програмування, переходу від індивідносуб'єктивної парадигми програмування до інтерсуб'єктивної, основу якої складає евідентне основоположення про цілісне розуміння програмування, як діяльності, що обумовлена програмою. Розвиток цієї парадигми, зокрема і технологічного середовища програмування як платформи програмної релятивізації, є покроковим продуктивним збагачення основоположення про цілісне розуміння. Програмування розглянуто у контексті осучаснення його розуміння як діяльність, що обумовлена програмним уподібненням, суб'єктоорієнтованим взаємодоповненням його активної та пасивної форм. Об'єктивізація активно-пасивного взаємодоповнення є основною передумовою реального осучаснення розуміння програмування як рефлексивно-транзитивного замикання породжуваного суб'єктом програмного уподібнення. Визначальною для об'єктивізації активної ролі суб'єкта в контексті побудов у середовищі програмування є концепт – суть – уподібнення згаданого носія, представлена у вигляді тієї чи іншої специфікації. Інструмент логіко-математичних специфікацій семантико-синтаксичних аспектів програмування апробовано на прикладах, продемонстровано загальні особливості застосування технологічного середовища програмування до породження суб'єктоорієнтованих технологічних систем програмування та їх використання. Розглянуті приклади вирішення задач у арифметичній систмі програмування демонструють важливі загальні особливості редукційного концептування оракульних схем. Це, по-перше, гарантована коректність отримуваних рішень, що випливає безпосередньо з їх побудови. По-друге, можливість переходу від рішень окремих задач до вирішення класів подібних задач. І, по-третє, отримані рішення можуть бути реалізовані на різних синтаксичних програмних платформах.

Ключові слова: *концепт, монада, композит, композиція, редукція, оракульні схеми, середовище програмування.*

Постановка проблеми. Відомо, що свідоме розв'язання будь-якої задачі полягає в наявності конкретного плану, виконання якого спрямоване на досягнення поставленої мети. Нині цей план, очевидно чи інтуїтивно, пов'язаний з розумінням розв'язку основної задачі через інтеграцію рішень її підзадач. Методи отримання та формалізація таких рішень залежать від конкретної сфери, складності задачі, наявних ресурсів для виконання, рівня професійної підготовки виконавця, особистих уявлень про ефективність методів і засобів тощо. Це значно впливає на якість отримуваних результатів, ступінь обґрунтованості та достовірність, ефективність формалізації таких рішень та можливості їх подальшого використання, наприклад, як компоненти для розв'язання нових задач.

До тепер ця концепція «розділяй та володарюй» у програмуванні призводила до вражаючих результатів, що очевидно вказували на її переваги, не звертаючи увагу на серйозні труднощі, які супроводжували та супроводжують цей підхід і розглядаються як об'єктивні проблеми розвитку. До певного моменту такий підхід не тільки не завдавав шкоди, а навіть сприяв розвитку програмування, забезпечуючи розширення бази фактів у програмуванні як основи для подальших досліджень. Як приклад, можна згадати відому

енциклопедичну монографію Д. Кнута під назвою «Мистецтво програмування» [1].

Але, як завжди, часом те, що раніше можна було вважати другорядним, стає ключовим аспектом. Програма як продукт повинна ґрунтуватися на технологічних принципах обробки інформації та пов'язаних з ним процесів. З іншого боку, її реалізація залишається великою мірою особистісною, і вона оцінюється лише за кінцевим результатом і не може бути відокремленою від виконавця. Такий підхід надає максимальну свободу для креативності у програмуванні, розглядаючи його перш за все як мистецтво, з усіма властивими цьому підходу перевагами та недоліками. Переваги цього підходу відомі всім, але його головним недоліком є занадто спрощене розуміння програмування, яке не відповідає сучасним вимогам. Це проявляється у фокусуванні на результаті творчого процесу – самій програмі, і у відсутності адекватної уваги до самого процесу. Як наслідок, не відокремленість програми від суб'єкта програмування ускладнює можливість дослідження продуктивних принципів програмування. Оскільки фундаментальні властивості програм формуються під час їх створення, така надмірна спрощеність у розумінні програмування стає серйозним обмеженням для розвитку сучасних технологічних основ програмування. Тому важливим є удосконалення розуміння програмування.

Аналіз останніх досліджень і публікацій. У роботах [2–4] було доведено, що розв'язання цієї проблеми вимагає суттєвих змін – переходу від індивідуально-суб'єктивного підходу до інтерсуб'єктивного підходу у програмуванні, який активно враховує роль суб'єкта у процесі програмування та сприяє його продуктивній технологізації. Цей перехід ґрунтується на концепції суттєсутнього уподібнення (ССУ), що є особливими відносинами, що виникають при розгляді причинно-наслідкового зв'язку між процесом вирішення задачі та його результатом [2, 3]. Точно через цю концепцію виникає програмна релятивізація (ПР), що є суттєсутністною релятивізацією у програмуванні, спрямованою на його технологізацію, і ця тема є основною у даному дослідженні.

У роботах [3, 4] показано, що платформа для вивчення суттєсутніх уподібнень і програмної релятивізації є біабстрактною, тобто вона забезпечує взаємодію двох різних типів абстракцій, які об'єктивно не пов'язані один з одним: закритої інтерсуб'єктивної логіки, що є ядром технологічного середовища програмування (ТСрП), та відкритого різноманіття суб'єктоорієнтованих розширень – технологічних систем програмування (ТСП). Тому саме взаємодія між цими двома аспектами вибрана як об'єкт дослідження. Взаємодія між ними підтримується за допомогою суттєсутніх уподібнень, які спрямовані на врахування активної ролі суб'єкта програмування. Об'єктивізація суб'єкта через актуалізацію інтерсуб'єктивних модальностей, які є характерними для ТСрП, як метод програмної релятивізації, його розвиток і застосування – це ціль даної роботи.

Метою статті є обґрунтування необхідності парадигмальних змін у технологічних основах програмування, переходу від індивідносуб'єктивної парадигми програмування до інтерсуб'єктивної, а також представлення технологічної системи програмування як платформи для сучасного редукційного програмування і її використання в розв'язанні типових задач.

Виклад основного матеріалу дослідження. З вищезазначеного стає зрозумілим, що досягнення нашої мети можливе через збагачення терміна «система програмування» більш глибоким інтелектуальним вмістом, в якому логіка та предметна суть об'єднуються у спільному контексті ТСрП та ТСП – це важлива реалізація цього розуміння.

Основа інтерсуб'єктивної парадигми полягає в твердженні, що програмування повинно бути розглянуте як діяльність, що визначається програмою. Розвиток цієї парадигми, включаючи створення ТСрП як платформи для програмної релятивізації, є етапним процесом поступового збагачення цієї основної ідеї. Ця розбудова відповідає тренду «від цілісного уявлення до продуктивного розуміння» програмування. Тут продуктивність означає спрямованість на технологізацію. Давайте коротко розглянемо основні кроки цього процесу.

З цього основного положення випливає, що для продуктивного збагачення нашого загального розуміння програмування, спочатку необхідно розширити наше розуміння того, що таке програма. Це означає, що збагачення має бути фундаментальним, як, наприклад, у відповідності до принципу достатніх підстав, який був сформульований Лейбніцем [5]. Трактування терміна «програма» як уподібнення суттєвої риси [2, 3] виглядає наибільш відповідним у цьому контексті. Ця ідея допомагає нам розуміти програмування як взаємодоповнення двох об'єктивно незалежних типів абстракцій – сутності, яка визначає «що може бути», та суті, яка визначає «що є». Таким чином, ми отримуємо продуктивне розширення початкового основоположення: програмування –

це діяльність, яка визначається програмним уподібненням (ПУ). У цьому контексті ПУ – це продуктивне збагачення суттєсутнього уподібнення.

Наступним кроком у створенні ТСрП є безумовне збагачення програмного уподібнення. Головною особливістю останнього є його неможливість об'єктивно зведення модальності сутності, яка визначає «що може бути», та реальності суті, яка визначає «що є». Це призводить до біабстрактності ТСрП як носія програмних уподібнень.

Кожне конкретне програмне уподібнення виникає на основі обумовлення сутності – зв'язування її певною умовою. З цього очевидно, що носієм таких умов може бути лише суб'єкт, який обумовлює їх як інструмент об'єктивізації, а уподібнення представляє собою реалізацію взаємодоповнення активного та пасивного обумовлення [3]. Тому об'єктивізація активно-пасивного взаємодоповнення є необхідною передумовою для справжнього оновлення розуміння програмування як рефлексивно-транзитивного зв'язку, що створюється суб'єктом завдяки програмному уподібненню. Ключовою для об'єктивізації активної ролі суб'єкта в контексті побудови ТСрП є концепція – сутність – уподібнення цього носія, яка представлена в формі специфікації. А уподібнення, як ціле (єдине, монадне) наслідок уподібнення, об'єктивізує пасивну частину обумовлення. Таким чином, ми приходимо до наступного етапу розширення програмного (суттєсутнього) уподібнення:

$$\begin{cases} концепт = суть, що обумовлює сутність \\ програма = сутність, що обумовлюється концептом \end{cases}$$

Концепт забезпечує узгодженість у побудові, а програма – їх ефективність. Перехід між ними реалізується через програмне уподібнення. Важливо відмітити, що концепт, програма і обумовлення, що використовуються тут, вводяться в інтенсіональному аспекті – не з точки зору їх конкретних об'ємів, а як абстракції від відповідних суб'єктних умов, які можна розглядати як оракули [2, 4]. Отже, дана концептуальна визначеність представляє собою відкриту систему, яка включає як внутрішні, так і зовнішні аспекти програмування [2, 4]. Враховуючи реальність її створення, ця визначеність є цілісною. Таким чином, вона потребує подальшого розширення для забезпечення ефективності.

Розгляд оракулів, як відкритих систем, приводить до вивчення оракулів більш високого рівня – оракульних схем, які можуть значно підвищити продуктивність побудов. Особливе місце серед них займають композитні схеми, які мають загальне значення та є базовими [2]. Дослідження в галузі композитного програмування (див., наприклад, у [4, 7–13] і посилання на них) переконливо доводять, що таке розширення концептопрограмної парадигми є дуже продуктивним. Тобто, ТСрП, як втілення всебічного розуміння програмування, в остаточному підсумку, може бути побудовано шляхом композитно-композитного розширення базової концептопрограмної визначеності. Таким чином, всі подальші розширення будуть спрямовані на підвищення ефективності і пов'язані з індивідуальними зв'язками ТСрП з відповідними ТСП.

З усього вищезазначеного можна зрозуміти, що будь-яка ТСП виникає через процес програмного уподібнення, який представляє собою послідовне замикання концептопрограмної системи ТСрП відкритого-замкненого композитологічного оракульного середовища. Цей процес полягає в актуалізації відповідних оракулів ТСрП.

Отже, концептопрограмне замикання є основою методу програмної релятивізації, який використовується для специфікації ТСП. Результатом цього процесу є створення суб'єктоорієнтованої системи, яка імплементує активну роль суб'єкта у програмуванні.Цей процес включає кілька етапів:

1. Уточнення базових та похідних генетичних структур як концептів, які властиві як суб'єкту, так і об'єкту програмування.

2. Визначення базових предметних операцій, які не потребують додаткової деталізації з прагматичних причин.

3. Створення композито-композиційних інтерфейсів для взаємодії з ТСП.

Цей процес може бути проілюстрований на прикладах програмування арифметичних перетворень, які демонструють загальні особливості використання ТСрП для створення ТСП та їх використання, зокрема, гарантовану коректність отримуваних рішень, можливість застосування їх для вирішення подібних задач та можливість реалізації отриманих рішень на різних синтаксичних програмних платформах.

Таким чином, носій $\mathfrak{E}$ арифметичної ППА складають n-арні частково-рекурсивні арифметичні функції виду $N^n \to N$ та n-арні частково-рекурсивні арифметичні предикати виду $N^n \to \{T, F\}$, $n \in N$ (далі – функції та предикати) [15]. Сигнатуру ж ППА (позначатимемо тут Ω) складають операції суперпозиції, розгалуження і циклування, що є адекватними уточненнями основних методів конструювання програм [8–12].

Нагадаємо формальні визначення цих операцій, деякі позначення та результати. Зауважимо, що при акцентуванні уваги на генетичних особливостях розглядуваних функцій та предикатів у їх позначенні перевага надаватиметься операторній, а при акцентуванні на результатах застосування композицій – термальній формам запису [15].

Нехай задані m функцій однакової арності, наприклад, $k-f_1,\dots,f_m$, m-арна функція f та -арний предикат h, $m,k\in N$. Розглянемо наступні нові k-арні функції g,d,c, значення яких на аргументі $a_1,...,a_k$ задаються так:

$$g(a_1,...,a_k)\cong f\left(f_1(a_1,...,a_k),\dots,f_m(a_1,...,a_k)\right),$$

$$d(a_1,...,a_k)\cong\begin{cases}f_1(a_1,...,a_k), якщо\, h(a_1,...,a_k)=T\\ f_2(a_1,...,a_k), якщо\, h(a_1,...,a_k)=F\end{cases}$$

$c(a_1,...,a_k)\cong a_1^j$, де a_1^j – перша компонента першого кортежу послідовності кортежів $[a_1^i,...,a_k^i]_{i=1,2,...}$, де $a_s^1=a_s, s=1,...,k$, $a_s^{i+1}=f_s\left(a_1^i,...,a_k^i\right)$, для якого $h\left(a_1^j,...,a_k^j=F\right)$, за умови, що для усіх $r=1,...,j-1$ значення $h\left(a_1^r,...,a_k^r=T\right)$, ($\cong$ - розуміється як умовна рівність).

Будемо казати, що функція g є результатом застосування композиту $m+1$-арної суперпозиції S^{m+1} до кортежу функцій $f,f_1,...,f_m$, функція d – композиту тернарної операції розгалуження $\lozenge$ до кортежу функцій h,f_1,f_2, а c – композиту $k+1$-арного циклування до кортежу $h,f_1,...,f_k$. Тобто, $g\equiv S^{m+1}(f,f_1,...,f_m)$, $d\equiv(h,f_1,f_2)$ та $c\equiv *^{k+1}(h,f_1,...,f_k)$.

Позначимо $[\sigma]_{\&}$ замикання множини функцій та предикатів σ операціями ППА, 0^1 – т. з. 0-функцію, що будь-якому натуральному числу ставить у відповідність 0, тобто: $S^2(0^1,I_1^1)(n)=0\,|_{n\in N}$, s^1 – функцію слідування: $S^2(s^1,I_1^1)(n)=n+1\,|_{n\in N}$, $+^2$ – суму двох натуральних чисел: $S^3(+^2,I_1^2,I_2^2)(n,m)=n+m\,|_{n,m\in N}$, $\times^2$ – добуток двох натуральних чисел: $S^3(\times^2,I_1^2,I_2^2)(n,m)=n\times m\,|_{n,m\in N}$, $<^2$ – предикат «менше»: $S^3(<^2,I_1^2,I_2^2)(n,m)=\begin{cases}T, якщо\, n<m\\ F, якщо\, m\le n\end{cases}|_{n,m\in N}$ та I_n^m – селекторну функцію: $I_n^m(a_1,...,a_m)|_{m,n\in N}=\begin{cases}a_n, якщо\, n\le m\\ інакше\end{cases}$.

Справедливо: $[0^1,s^1,+^2,\times^2,<^2,I_n^m]_\Omega=$ £ [16]. Даний результат разом з вибором операцій з Ω у якості актуалізації базових генетичних структур (композитів) ТСрП забезпечує логіко-предметну основу для продуктивної специфікації арифметичної ТСП. Для цього принциповим є інтерфейс між композитами з Ω та похідними від них композиціями. Обумовлені композитами редукційні структури, як було показано у [4] відіграють ключову роль у побудові таких інтерфейсів. Це забезпечує головну особливість створюваних таким чином ТСП – вони реально, а не лише номінально підтримують причинно-наслідкове взаємодоповнення двох складових вирішення будь-якої програмістської задачі – програмування як породження та застосування композицій та програми як наслідку програмування.

Висновки. Досліджено та обґрунтовано необхідність змін у технологічних засадах програмування, що передбачає перехід від індивідуально-суб'єктної парадигми програмування до інтерсуб'єктивної парадигми, яка дозволить перейти від мистецтва програмування до його технологізації. У цьому контексті створюється відкрито-замкнене оракульне середовище програмування на основі закритої інтерсуб'єктивної логіки та різноманітності предметних суб'єктно-орієнтованих розширень, спрямованих на врахування активної ролі суб'єкта у програмуванні.

Центральною для об'єктивізації активної ролі суб'єкта у технологічному середовищі програмування є концептування, де концепт представляється суб'єктом умовою у вигляді певної специфікації. Це концептування формує основу відкритого-закритого концептопрограмного технологічного середовища, яке інтерпретує програмування як суб'єкто-орієнтовану продуктивну діяльність, обумовлену програмою.

Програмування розглядається як діяльність, обумовлена програмним уподібненням – суб'єктно-орієнтованим взаємодоповненням активної та пасивної форм, що обумовлюється програмою. Отже, об'єктивізація активно-пасивного взаємодоповнення визначається програмою – сутністю, що обумовлюється концептом. У цьому контексті концепт забезпечує цілісність побудов, тоді як програма – їх продуктивність.

Продуктивність програмування досягається за допомогою композитних схем, де абстракції відповідних суб'єктних умов розглядаються як оракули. Концептуально-монадне середовище програмування розглядається як оракульне середовище. Подальші дослідження будуть спрямовані на подальше розширення фактології для вдосконалення методу програмної релятивізації.

Список літератури:

1. D. E. Knuth, The Art of Computer Programming. Addison-Wesley Professional, 2011.
2. І.В. Редько, П.О. Яганов, Концептуальна модель технологічного середовища програмування, Наукові вісті КПІ, № 1, с. 18-26, 2020. DOI: 10.20535/kpi-sn.2020.1.197953.

3. I. Redko, P. Yahanov and M. Zylevich, Concept-Monadic Model of Technological Environment of Programming, 2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC), Kyiv, Ukraine, 2020, pp. 125-130, doi: 10.1109/SAIC51296.2020.9239204.

4. І.В Редько, П.О. Яганов, М.О. Зилевіч, Редукційне концептування оракульних схем, Системні дослідження та інформаційні технології, № 1, с. 21-33, 2021. doi: 10.20535/SRIT.2308-8893.2021.1.02

5. І.В. Редько, Прагматичні основи дескриптивних середовищ, Проблеми програмування, № 3, с. 3-25, 2005.

6. Ґ.В. Лейбніц, Монадологія, Sententiae, № 1, с. 151-177, 2013.

7. В.Н. Редько, Композиції програм і композиційне програмування, Програмування, № 5, с. 3-24, 1978.

8. В.Н. Редько, Дефінітори, метод дефініторного процесування, Кібернетика, № 6, с. 52-56, 1974.

9. І. А. Басараб , Н. С. Нікітченко, В.Н. Редько, Композиційні бази даних. – К. : Либідь, 1992.

10. В.Н. Редько, Основи програмології, Кібернетика і системний аналіз, № 1, с. 35-57, 2000.

11. В.Н. Редько, Основи композиційного програмування, Програмування, № 3, с. 3-13, 1979.

12. В.Н. Редько, Н.В. Гришко, І.В. Редько, Експлікативне програмування у середовищі логіко-математичних срецифікацій, УкрПРОГ98, с. 71-76, 1998.

13. І.В. Редько, Н.В. Гришко, Експлікативне програмування у інтеграційному середовищі, Проблеми програмування, № 2, с. 59-65, 2004.

14. D. I. Redko, I. V. Redko, P. O. Yahanov and T. L. Zakharchenko, Compositional basis in programmer activity, Системні дослідження та інформаційні технології, no. 4, pp. 83-96, 2015.

15. K. B. Gemlau, L. Köhler and R. Ernst, A Platform Programming Paradigm for Heterogeneous Systems Integration, in Proceedings of the IEEE, vol.109, no. 4, pp. 582-603, April 2021, doi: 10.1109/JPROC.2020.3035874.

16. Д. Б. Буй, Теорія програмних алгебр композиційного типу та її застосування: Дис. докт. фіз.-мат. наук: Київ, 2002.

17. J. Printz, D. Krob. System Architecture and Complexity: Contribution of Systems of Systems to Systems Thinking. JOHN WILEY, 2020.

Redko I.V., Yahanov P.O., Zylevich M.O. CONCEPTUAL FOUNDATIONS OF TECHNOLOGICAL PROGRAMMING SYSTEMS

The article investigates and substantiates the need for paradigmatic changes in the technological foundations of programming, the transition from an individual-subjective paradigm of programming to an intersubjective one, the basis of which is the evident premise of a holistic understanding of programming as an activity determined by a program. The development of this paradigm, in particular the technological environment of programming as a platform for software relativization, is a step-by-step productive enrichment of the premise of holistic understanding. Programming is considered in the context of modernizing its understanding as an activity conditioned by program likeness, and subject-oriented complementarity of its active and passive forms. The objectification of active-passive complementarity is the main prerequisite for the real modernization of the understanding of programming as a reflexive-transitive closure of the subject-generated program likeness. Decisive for the objectification of the subject's active role in the context of constructions in the programming environment is the concept – essence – assimilation of the mentioned medium, presented in the form of one or another specification. The tool of logical-mathematical specifications of semantic-syntactic aspects of programming is tested on examples, and the general features of the application of the technological programming environment to the generation of subject-oriented technological programming systems and their use are demonstrated. The considered examples of solving problems in the arithmetic system of programming demonstrate important general features of the reductive conceptualization of oracle schemes. This is, first of all, the guaranteed correctness of the obtained solutions, which follows directly from their construction. Secondly, the possibility of transition from solving individual problems to solving classes of similar problems. And, thirdly, the obtained solutions can be implemented on different syntactic software platforms.

Key words: *concept, monad, composite, composition, reduction, oracle schemes, programming environment.*