УДК 004.42

DOI: 10.20535/2617-0965.2020.3.3.198584

# Концептомонадна модель технологічного середовища програмування

Кудлай С. В., ORCID 0000-0003-3972-405X
Зилевіч М. О., ORCID 0000-0003-1646-0557
Яганов П. О.. к.т.н., ORCID 0000-0001-7358-9846
Редько І. В., д.фіз.-мат.н., ORCID 0000-0002-3121-1412
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
Київ, Україна

***Анотація*—У даній роботі запропоновано розширення принципів адаптивного середовища програмування на основі розглянутих інтерсуб'єктивної та міжсуб'єктної парадигм програмування. Для цього розуміння програмування адекватно збагачується введенням концептуальних логічних структур, які засновані на взаємодії оракулів, композиційному збагаченні та схематизації концептувань. Наведено приклади використання редукції для демонстрації зв'язку композиції та декомпозиції в концептуванні. Це дозволяє відійти від загальної концепції програмування як предметно-визначеної діяльності та закласти основи її технологізації.**

***Ключові слова — концепт; монада; композит; оракул; оракульна схема; програмне середовище; редукція; інтерсуб'єктивна парадигма; редукція.***

## I. Вступ

На сьогоднішній день важко уявити сучасний технологічний напрямок, який би досяг високого рівня розвитку без програмування та інтелектуалізації систем розробки. Не дивно, що програма, якою щось має керуватися чи на основі чого буде працювати, вважається метою і засобом досягнення успіху. При цьому цікавим є те, що безпосередньо теорія програмування є гарно вивченою і дослідженою, науково визначено раціональні алгоритми програмування. Проте практика показує, що кожна програма є предметом творчості програміста. Тобто є такою, яку неможливо тотожно повторити, оскільки манера створення програми є занадто суб'єктивізованою через особливості професійних вмінь програмістів. До таких вмінь належать: досвід та навички, світогляд, авторський стиль у написанні програм і тому подібне. Тож коректно вважати програмування мистецтвом [1].

Але, з іншого боку, враховуючи технологізацію сучасного світу, через засилля різного роду програм,без яких ми вже не уявляємо свого повсякденного життя, наприклад операційної системи смартфона чи алгоритмів роботи кавоварки; виникає логічне запитання, яке полягає у безпеці, а точніше у впливі накопичуваних помилок різного роду у програмних продуктах і утому,як вони себе проявлятимуть в тій чи іншій ситуації. Адже враховуючи темпи сучасної технологізації неможливо уникнути накопичення принципових проблем, які за своєю кількістю можуть вважатися проявом кризових явищ у програмуванні. Внаслідок стрімкого накопичення проблемних явищ виникає ризик техногенної аварії, спричиненої розтиражованими програмними продуктами, які не мають можливості для користувача перевіряти та контролювати їх безпечність та нешкідливість на етапах розробки.

Для уникнення протиріччя між процесом створення програми і самою програмою, яка часто спровокована надмірною суб'єктивізацією програмістської діяльності, що ускладнює розуміння причинно-наслідкових зв'язків всередині програми,доцільно внести зміни в парадигму програмування. Загальновизнана індивідно-суб'єктна парадигма, що визначає програму виключно через процес програмування, має бути замінена на інтерсуб'єктивну [2], в якій програмний продукт вважається інтерсуб'єктивним предметом, на який власне і спрямована дія створення програми. Тобто об'єктом дослідження є процес створення програми, а предметом виступає безпосередньо сама програма, яка є втіленням плану цього процесу.

Метою цієї роботи є розвиток засад адаптивного технологічного середовища програмування на основі інтерсуб'єктивної парадигми.

Для досягнення вказаної вище мети необхідно предметно розширити розуміння програмування за допомогою введення певних концептуально-логічних структур. Спираючись на такі структури можна закласти основи технологізації процесу програмування, коли надбання кожного окремого суб'єкта будуть доступні для всіх бажаючих.

## II. Інтерсуб'єктивна парадигма програмування

Відповідно до цієї парадигми найважливішим у процесі програмування є створення плану програми. Така діяльність визначається активною роллю суб'єкта, який безпосередньо реалізовує цей план.

DOI: 10.20535/2617-0965.2020.3.3.198584

Він називається концептом, а діяльність по його реалізації – концептуванням. У такому випадку результатом програмотворчого процесу є суть, що визначає сутність. Поняття «суть і сутність» розкрито у працях [3]–[5]. Сутність, що визначається концептом, називається монадою. Тобто можна сказати, що концепт – це суть, що визначає сутність. Де план – це концепт, а створення плану – це концептування.

Таке визначення вводить важливі види сутностей – концепти, монади, обумовлення. Їхня необхідність визначається тим, що вони є носіями якісних суб'єктивних обумовлень, які за Б. А. Трахтенбротом [6], [7], називаються оракулами.

Наведені вище визначення складають оракульну відкрито-замкнену систему, що описує загальну схему програмотворення. Така схема заснована визначальній активній ролі суб'єкта, розглядаючи процес програмотворення як такий, що визначається безпосередньо суб'єктом [3]. Виходячи з цього інтерсуб'єктивна парадигма визначає процес програмування як суть, що визначається самою програмою.

## III. Концептування як схема взаємодії оракулів

Описання концептування через оракульні структури дозволяє використовувати можливості традиційного математичного апарату для нотації результату та поєднання його з денотативними методами. В процесі програмотворення суб'єкт так чи інакше об'єктивізує концепт схемою взаємодії оракулів. Оракульні структури збагачують концептування, формують точку зору на програму, як на структурну діяльність, а на концепт, як на відповідну структуру.

Оракульна природа концептування описується наступним прикладом. При позначенні операцій і предикатів використовують як операторну, так і термальну форми запису. Таким чином $S^2(\times, S^2(+, I_1^3, I_2^3), I_3^3)\big|_{N^3 \to N}$ і $(x+y)\times z_{|x,y,z \in N}$ позначають тотожні арифметичні операції [8]. Перший запис відображає генезис цієї операції з простіших, обумовлений інтенсіоналом – оператором суперпозиції. Другий, описує цю ж арифметичну операцію екстенсіонально, як функцію через зазначення об'єму властивих їй арифметичних дій. Форми цих записів взаємозамінні, але в першому занотована абсолютно природна домінанта генезису по відношенню до його результату, а в другому описано деяку сутність актуально заданої множини дій, про генезис якої можна говорити лише опосередковано і не детерміновано, як про «функцію об'єму дій».

Аналогом таких структур можна вважати генетичні структури [9]. Програма сама по собі характеризується генезисом, що визначає основну парадигму програмування. Особливе місце серед таких структур займають композиційні структури.

## IV. Композиційне збагачення концептування

Роль суб'єкта під час створення програми проявляється тим, що саме він визначає сутність програми. Оракульність тут підкреслюється у поступовості визначення сутності програми і її результату. В процесі створення програми її сутність є такою, що визначається сама собою. Це можна розглядати як наслідок концептування, до того ж його структура може вважатися композицією.

Особливістю цих структур є те, що їх неможливо заздалегідь визначити однозначно та вичерпно. Вони є результатом суб'єктивного вкладу творця і носіями його суб'єктивних розумінь сутностей. Такі структури ще називають базовими засобами генезису [3], [5], на основі яких виникають іншікомпозиції, як наслідок поетапності таких базових структур.

Таким чином можна визначити концептування, як послідовність дій суб'єкта програмотворення, концептом якої є композиція. При цьому кожен етап такого концепту також є певним концептом[10].

На основі усього вище сказаного можна виділити два основних види композиту. До першого відносяться композити, що визначають розуміння композиції, як поетапності композитів. До другого відносять базисні композити, що визначають активну роль суб'єкта в концептуванні.

## V. Схематизація концептувань

Для більш формального представлення вводяться наступні поняття. Сутність ENT, суть GST, концепт CON і монада MON. При використанні запису $Ent_1 \succ Ent_2$ мається на увазі, що сутність $Ent_1$ обумовлює сутність $Ent_2$. Для оберненої дії використовується оператор $\prec$. Тобто запис $Ent_1 \prec Ent_2$ позначає, що сутність $Ent_1$ обумовлюється сутністю $Ent_1$. Ввівши дані поняття можна представити концептування наступною системою відкрито-замкнутих систем оракульних обумовлень:

$$\begin{cases} Comt \overset{def}{=} Gst \ {}^{B}\!\succ Ent \\ CtMon \overset{def}{=} Ent \prec Comt \end{cases},$$

де $Comt$ – оракул «композит», ${}^{B}\!\succ$ – оракул «базисно обумовлює».

Останнє означає, що для обумовлення суб'єкту не потрібно додаткової деталізації, тому цю операцію можна виконати за один крок. $CtMon$ є збагаченням оракула $Mon$ завдяки обумовленню концепта композитом. Маючи дану систему, можна вивести схему концептування:

$$\begin{cases} Comp \overset{def}{=} Gst \ {}^{G}\!\succ Ent \\ CMon \overset{def}{=} Ent \prec Comt \end{cases},$$

де $Comp$ – оракул «композиція», ${}^{G}\!\succ$ – оракул «генезисно обумовлює», $CMon$ – збагачення оракула $Mon$ завдяки обумовленню концепта

Наведені схеми є концептами для найрізніших композитних і композиційних концептувань. Останні отримуються внаслідок актуалізації оракулів, що входять до схем. Оракули $CtMon$, $CMon$, ${}^{B}\!\succ$, ${}^{G}\!\succ$ і їх

DOI: 10.20535/2617-0965.2020.3.3.198584

носії залежать від обумовлення оракулів $Ent$, $Comt$ та $Comp$.

Причому, залежності виду $Ent \xrightarrow{Comt} CtMon$ або $Ent \xrightarrow{Comp} CMon$, як було зазначено вище, є функціональними. А значить і композитні, і композиційні монади можуть бути експліковані як композиції сутностей:

$$CtMon \Rightarrow Comt(Ent_{i_1},....,Ent_{i_k}) \text{ і}$$

$$CMon \Rightarrow Comp(Ent_{j_1},....,Ent_{j_p}),$$

де $Ent_{i_1},....,Ent_{i_k}, Ent_{j_1},....,Ent_{j_k}$ – деякі сутності, а $\Rightarrow$ означає «експлікативно зводиться». Розглядаючи ці експлікативні зведення в контексті концептування,можнаспроектувати їх на відповідні функціональні залежності, що дає можливість отримати наступні оракульні схеми:

$$Ent \xrightarrow{Comt} Comt(Ent_{i_1},....,Ent_{i_k}) \text{ і}$$

$$Ent \xrightarrow{Comp} Comp(Ent_{j_1},....,Ent_{j_p}).$$

Незважаючи на те, що ці схеми дуже схожі по зображенню, смисли їх принципово різні. Почнемо з того, що перша використовує базову композицію, але при цьому на сутності $Ent_{i_1},....,Ent_{i_k}$ ніяких додаткових обмежень не накладається. У другій же, навпаки, композиція не обов'язково базова, швидше вона похідна від композитів, а сутності $Ent_{j_1},....,Ent_{j_k}$ є елементарними, тобто достатньо деталізованими з точки зору суб'єкта. Перша є концептом реалізації активної ролі суб'єкта в будь-якому кроці концептування, друга ж являє собою рефлексивно-транзитивне замикання породжуваної концептуванням функціональної залежності, а значить дає бачення наслідку концептування як композиції елементарних, з точки зору суб'єкта, сутностей.

Наведені схеми важливі ще й тому, що в них явно відображений факт взаємодоповнення двох полюсів (початків) концептування – синтезу та аналізу, композиції і декомпозиції. Синтез представлений в них оракулами $Comt$ і $Comp$, а аналіз (декомпозиція) – $Ent_{i_1},....,Ent_{i_k}, Ent_{j_1},....,Ent_{j_k}$. Зв'язок композиції і декомпозиції в концептуванні добре представлений першою схемою. Змістовно кажучи, будь-яка розумна декомпозиція завжди індукована відповідною їй композицією. Але ж саме перша схема оперує базовими для суб'єкта композиціями (композитами). Будемо говорити, що кортеж $< Ent_{i_1},....,Ent_{i_k} >$ є $Comt$ - редукцією сутності $Ent$ якщо існує функціональна залежність $Ent \xrightarrow{Comt} Comt(Ent_{i_1},....,Ent_{i_k})$. Змістовний сенс редукції полягає в тому, що вона природним чином імплементує парадигму «розділяй і володарюй» в розумінні активної ролі суб'єкта в концептуванні.

## VI. Приклад з використанням редукції

Оскільки в прикладі використовуються іменні множини та іменні функції, потрібно ввести їх визначення. Іменна множина складається зі списку змінних. А змінна (M,m) в свою чергу складається з імені змінної (її адреси) M та зі значенням змінної (записане за її адресою) m. Отже іменна множина має вигляд $\{(x_1,d_1),(x_2,d_2),\ ...(x_n,d_n)\}$. Іменна функція має аналогічний принцип дії зі звичайною функцією. Вона ставить аргументу відповідне вихідне значення, але для іменних функцій не для всіх комбінацій аргументів визначено вихідне значення. Іменна функція записується як $F:\{(M,m),(N,n)\} \to \{M,x),(N,y)\}$. Визначення іменної функції та іменної множини необхідно для наступного прикладу.

Існує ЕОМ, яка вміє робити наступні дії:

- запис цілого додатного значення до змінної,
- додавання до змінної одиниці (+1),
- віднімання від змінної одиниці (-1),
- порівняння двох чисел(=, ≠, ≤, <, ≥, >).
- ЕОМ підтримує:
- послідовне виконання операцій, $f_1 \times f_2(d) = f_2(f_1(d))$.
- гілкування $\diamondsuit(p,\ f_1,\ f_2)(d) = \begin{cases} f_1(d), & p(d) = True \\ f_2(d), & p(d) = False \end{cases}$.
- циклування $*(p,f)(d) = f^k(d)$.

Також для даної ЕОМ була реалізована функція віднімання двох додатних значень. Необхідно реалізувати операцію остачі від ділення (mod) для двох цілих додатних чисел. Для вирішення даної задачі буде використано редукційний метод.

Редукцією функції f називається функція g, для якої виконується умова:

$$f = g \times f \qquad (1)$$

З цього запису випливає, що: $f = while\ p\ do\ g$, де p – предикат, який знаходиться при знаходженні редукції функції f.

Отже процес вирішення задачі зводиться до отримання функцій g та p.

Для розв'язання рівняння необхідно виявити характеристику функції. Першим кроком потрібно визначитись при якій комбінації аргументів відповідь буде очевидною і на виході буде отримано одне з вхідних значень:

$$Mod(m,\ n) = m,\ n > 0,\ m <= n \qquad (2)$$

На наступному етапі розглядається випадок, відмінний від (2). При цьому нова функція є редукцією функції mod.

DOI: 10.20535/2617-0965.2020.3.3.198584

$$MOD:\ \{(M,\ m),\ (N,\ n)\} \rightarrow \{(M,\ mod(m,\ n)),\ (N,\ n)\}$$
$$Mod(m,\ n) = mod(m - n,\ n),\ m > n$$

У цьому виразі вже наявні функція g та предикат p

$$G:\ \{(M,m),\ (N,n)\} \rightarrow \{(M,m-n),\ (N,n)\}$$
$$P:\begin{cases} m > n = True \\ m \le n = False \end{cases}$$

```
def mod(m, n) :
    while (m > n) :
        m = sub(m, n)
    return m
```
(3)

Вираз (3) є реалізацією алгоритму знаходження остачі від ділення для даної ЕОМ. Ця функція написана в мнемоніці мови програмування Python.

У наступному прикладі для тієї ж ЕОМ потрібно реалізувати функцію найбільшого спільного дільника (gcd). Розв'язується задача за допомогою редукції.

Першим кроком визначається випадок, під час якого функція нічого не робить.

$$gcd(n,\ n) = n \tag{4}$$

Наступним кроком визначається функція при відмінному випадку, яка відповідає умові(1)

$$gcd(m,n)=\begin{cases} \gcd(m-n,n), m > n \\ \gcd(m,n-m), m < n \end{cases} \tag{5}$$

Функція по різному визначена для двох випадків, що призводить до використання гілкування. Отже з (4) отримується предикат p, а (5) вже є функцією G

```
def gcd(m, n):
    while m != n :
        if m > n :
            m = m - n
        elif m < n :
            n = n - m
    return m
```
(6)

Вираз (6) є готовою реалізацією на мові програмування Python.

## Висновки

Інтерсуб'єктивна парадигма передбачає, що програмування є процесом, націленим на створення програми. Тому реально підтримати програмування означає залучити у якості об'єкта до розгляду як процес програмотворення, так і його наслідок у їх причинно-наслідковомувзаємодоповненні. Предметом такого розгляду є структури програм, що підтримують відповідні причинно-насідкові зв'язки, а також способи, методи та засоби їх специфікації. Особливість таких структур полягає у тому, що вони не можуть бути заздалегідь актуалізовані однозначно раз і назавжди, а можуть бути лише породжені суб'єктом програмотворення у вигляді суттєсутніснних відношень. У роботі обґрунтовано зведення таких породжень до концептування.В якості концептів розглянуто композити – спеціальні класи базових композицій. Саме ж концептування на предметному рівні зводиться до вирішення рівнянь простої редукції. Як наслідок, отримувані рішення не потребують верифікації, адже їх коректність випливає безпосередньо з побудови. Отримані у роботі результати продемонстровані на простих і репрезентативних прикладах концептувань.

## Перелік посилань


[1] D. E. Knut, The art of computer programming. Fundamental Algorithms [Iskusstvo programmirovaniya. Osnovnyye algoritmy]. Moscow: Vyliams, 2006.

[2] E. G. Husserl, Logical Studies. Cartesian Reflections [Logicheskiye issledovaniya. Kartezianskiye razmyshleniya]. Minsk, 2000.

[3] J. W. Backus, Algebra of functional programs: functional level thinking, linear equations and generalized definitions [Algebra funktsional'nykh programm: myshleniye funktsional'nogo urovnya, lineynyye uravneniya i obobshchennyye opredeleniya, Matematicheskaya logika v . Moscow: Mir, 1991.

[4] G. Buch, Object-oriented analysis and design. Moscow: Binom, 1998.

[5] J. McCarthy, "Recursive Functions of Symbolic Expressions Their Computation by Machine, Part I," CACM 1960, 1960. [Online]. Available: https://aiplaybook.a16z.com/reference-material/mccarthy-1960.pdf.

[6] N. Wirth, "A Personal Computer for the Software Engineer. Proceedings," in ICSE 81.

[7] V. N. Redko and N. V. Grishko, "Conceptological foundations essential platform," in TAAPSD'2012, 2013.

[8] I. V. Redko, "The theory of descriptive environments and its applications," National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," 2008.

[9] I. V. Redko, D. I. Redko, and T. L. Zakharchenko, "Conceptual basis of programming," Komprynt, 2016.

[10] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," CACM 1972, 1972. [Online]. Available: https://www.win.tue.nl/~wstomv/edu/2ip30/references/criteria_for_modularization.pdf

UDC 004.421

DOI: 10.20535/2617-0965.2020.3.3.198584

# Concept-Monad Model of Technological Environment of Programming

S. V. Kudlai, ORCID 0000-0003-3972-405X
M. O. Zylevich, ORCID 0000-0003-1646-0557
P. O. Yahanov. PhD, ORCID 0000-0001-7358-9846
I. V. Redko, Dr.Sc.(Phys.-Math.), ORCID 0000-0002-3121-1412
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine

***Abstract*—This paper proposes an extension of the principles of an adaptive programming environment based on the considered intersubject and interpersonal programming paradigms and the use of oracles, concepts, copmozies, and open-loop systems. The purpose of this work is to develop the foundations of an adaptive technological programming environment based on an intersubjective paradigm. In order to achieve the above goal, it is necessary to substantially expand the understanding of programming by introducing certain conceptual and logical structures. Based on such structures, you can lay the foundations for the technological process of programming, when the property of each individual subject of programming will be available to everyone. An oracle open-loop system describing the general scheme of programming is considered. In such a scheme, the active role of the subject is decisive, since the process of creating the program is regarded as determined by the subject. This suggests that the interest-based paradigm defines the programming process as the essence that is determined by the program itself. Conceptualization in the context of the oracle interaction scheme is considered. Oracle structures enrich the conception, form a point of view on the program as a structural activity, and on the concept as an appropriate structure. Conceptualization is defined as the sequence of actions of the subject who creates the program, the concept of which is a composition. Moreover, each stage of such a concept is also a specific concept. There are two types of composites. The first is composites that determine the understanding of a composition as the stages of composites, and the second includes basic composites that determine the active role of the subject in conception. Examples of using reduction to demonstrate the relationship of composition and decomposition in conceptualization. Schemes of concepts for various composite and composite concepts are given. These schemes are also important because they clearly reflect the fact of complementarity between the two poles of conception - synthesis and analysis, composition and decomposition. The meaningful meaning of reduction considered in the paper is that it naturally implements the "divide and conquer" paradigm in understanding the subject's active role in conceptualization. This allows you to depart from the general concept of programming as a subject-specific activity and to lay the foundations for its technologicalization. To this end, among the vast variety of compositions, composites stand out as the basic gene structures. Compared to them, any other composition is formed as a derivative of the consistent use of composites. An effective object programming environment is developed based on the concept of reduction. Adaptation of this environment is made for the standard system of software algebras. This demonstrates the solution of a number of representative numerical analysis problems.**

***Keywords — concept; monad; composite; oracle; oracle diagram; softwawe enviroment; reductio; intersubjective paradigm; reduction.***