УДК 004.421

DOI: 10.20535/2617-0965.eae.227740

Застосування оракульного концептування при програмуванні дизайну електронних мікросхем

Зилевіч М. О., ORCID 0000-0003-1646-0557
Кафедра конструювання електронно-обчислювальної апаратури keoa.kpi.ua
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського», ROR 00syn5v21
Київ, Україна

***Анотація*—У роботі запропоновано розширення принципів адаптивного середовища програмування на основі дослідження інтерсуб'єктивної парадигми шляхом розв'язання програмістських задач з використанням оракульного концептування. Для цього детально розглянуто, що являє собою оракульне концептування, на репрезентативних прикладах показано його використання для вирішення як загального класу задач, так і вузькоспеціалізованих програмістських задач. Розглянуто схему впровадження та реалізації цього методу при розробці цифрового дизайну інтегральних мікросхем. Реалізація даного методу на практиці певним чином уніфікує процес розробки програмного продукту, тим самим зменшує вплив активної ролі суб'єкта і підтримує впровадження інтерсуб'єктивної парадигми.**

Ключові слова — метод; синтез; інтерсуб'єктивна парадигма; оракул; оракульна схема; програмне середовище; програмування; Verilog.

I. Вступ

Сьогодні програмування є одним з технологічних напрямків, що активно розвивається у світі. До того ж із кожним роком ступінь такого розвитку лише збільшується, що проявляється в усе глибшому і всеосяжнішому проникненні різного виду програмних продуктів до різних сфер людської діяльності. Це є невіддільна складовою процесу технологізації сучасного світу.

Говорячи саме про розвиток програмування, вважається, що теорія програмування є вивченою і дослідженою на досить високому рівні. Науково підтверджено доцільність використання різних алгоритмів і підходів у процесі програмування [1–3]. Але, враховуючи практичну сторону програмування, в переважній більшості випадків, процес створення певного програмного продукту, або його частини, є досить індивідуальним. Така індивідуальність пояснюється людським фактором. Тобто можна вважати, що програмування – процес більше суб'єктивізований, ніж строго визначений та уніфікований.

Беззаперечно, в процесі створення програми використовуються загально визначені алгоритми, структури, підходи та методи, проте, це загальні речі, які діють на етапах планування. Коли ж мова йде про конкретний процес створення якоїсь частини кода, яким займається конкретна особа, то саме тут проявляється уся надмірна суб'єктивізація. Адже кожна така особа при написанні коду керується своїми досвідом, який визначається освітою, знаннями, вміннями, навичками, світоглядом та іншими складовими.

Проблеми пов'язані з такою надмірною суб'єктивізацією та варіант їх вирішення описані у роботах [4, 5]. Згідно з останніми , доцільно замінити загальновизнану індивідно-суб'єктну парадигму, що передбачає визначення програми через її творця, на інтерсуб'єктивну, в якій процес створення програми є об'єктом дослідження, а предметом виступає безпосередньо сама програма, яка є втіленням плану цього процесу.

Одним із методів вирішення цієї проблеми, згідно концептомонадної моделі, є застосування оракульного концептування.

Мета роботи полягає у подальшому розвитку засад адаптивного технологічного середовища програмування на основі інтерсуб'єктивної парадигми, шляхом застосування оракульного концептування при розв'язанні програмістських задач [4–6].

Для досягнення описаної вище мети необхідно розробити алгоритм застосування підходу оракульного концептування, визначити його особливості та перспективи подальшого розвитку.

Реалізація цього методу на практиці певним чином уніфікує процес розробки програмного продукту, тим самим зменшує вплив активної ролі суб'єкта і підтримує впровадження інтерсуб'єктивної парадигми.

Copyright (c) 2021 Зилевіч М. О.

DOI: 10.20535/2617-0965.eae.227740

II. Оракульне концептування

Оракульним концептуванням вважається опис концептування через оракульні структури [6]. Відповідно концептуванням вважається процес створення концепту, схемою взаємодії оракулів: обумовлення, концепту, монади, сутності, суті. За допомогою цих оракулів суб'єкт здійснює свою активну роль у концептуванні, актуалізуючи всі або деякі з них, і тим самим конкретизуючи вихідну схему "розділяй і володарюй" [5].

Концептом називають план певної дільності, спрямований на розв'язання конкретної задачі. Така задача може бути будь-якого рівня складності, але згідно з підходом оракульного концептування її можна розбити на скінченну послідовність елементарних підзадач, які називаються оракулами [7]. Тобто, під оракульними структурами слід розуміти сукупність певних елементарних підзадач, що виникли внаслідок концептування конкретної задачі. Схематичне визначення оракульного концептування показано на Рис. 1.

На зображеній вище схемі у квадраті позначено початкове завдання. У колах позначені елементарні підзадачі, які вважаються оракулами. Двонаправлені стрілочки між оракулами відповідають елементарним концептам, а відповідно розбиття певної задачі на елементарні є оракульним концептуванням.

Таким чином, в програмуванні концептування визначається як послідовність дій суб'єкта програмотворення, спрямована на вирішення певного завдання. При цьому кожен етап такого концепту також є певним концептом [8].

Теоретично кожна підзадача аналогічним чином може бути проконцептована нескінченно, проте на практиці все закінчується сукупністю певних елементарних дії, які є характерними для сфери застосування цього підходу.

Також перевагою використання оракульного концептування є можливість використання традиційного математичного апарату для нотації результату та поєднання його з денотативними методами [9].

У процесі програмотворення особа на підсвідомому рівні 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 [5].

Для кращого розуміння принципів і особливостей оракульного концептування розглянемо його на прикладі роботи спрощеного скінченного автомату «кавоварка», схема якого показана на Рис. 2.

Зображена вище схема дає спрощене уявлення про алгоритм роботи кавової машини. Згідно з цим, автомат має стани, кожен з яких відповідає за вирішення певної підзадачі, перехід між станами здійснюється при виконанні певних умов [10]. Вже на цьому етапі можна сказати, що кожен стан автомату – це оракул, а умова переходу – концепт цього оракулу. Таке твердження є вірним, але якщо дивитись з точки зору програмування, а саме програмного коду для машини, то на даному рівні концептування це не має особливого змісту.

Щоб розкрити зміст оракульного концептування, застосуємо його до етапу "Подрібнення кавових зерен". Схема такого застосування показана на Рис. 3.

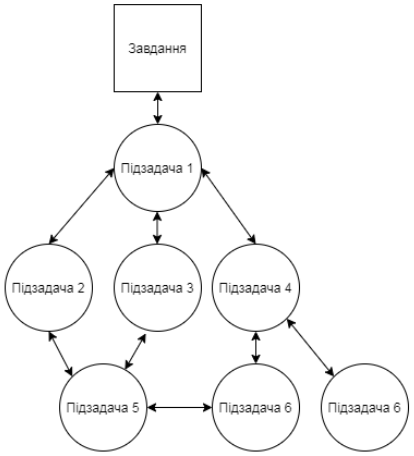

Рис. 1 Схематичний вигляд оракульного концептування

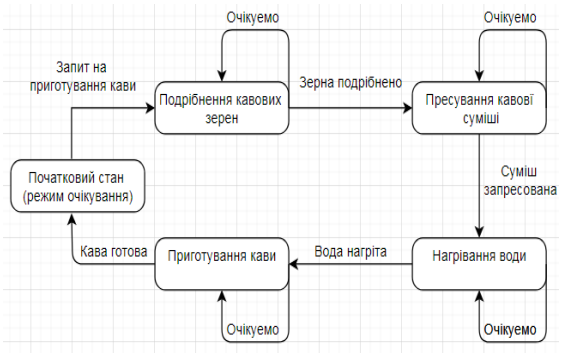

Рис. 2 Спрощена схема скінченного автомату «кавоварка»

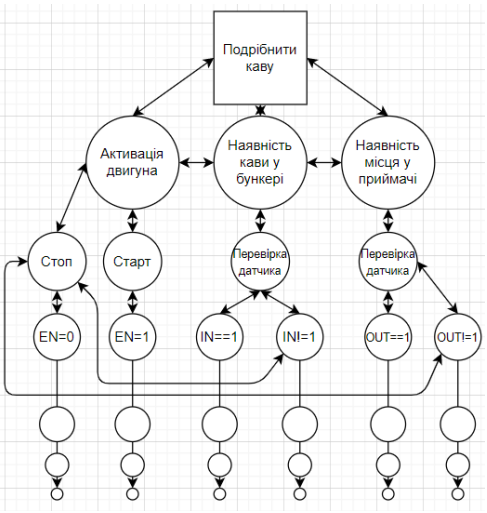

Рис. 3 Схема оракульного концептування

Copyright (c) 2021 Зилевіч М. О.

DOI: 10.20535/2617-0965.eae.227740

Зображена вище схема репрезентує найпростіше застосування оракульного концептування. Відповідно до схеми є початкове завдання “подрібнити каву” і за допомогою концептування початкову задачу розбито на простіші складові – оракули. Чим менше коло, тим простішим (для виконання машиною) є оракул. Таким чином найпростіший оракул являє собою низькорівневий машинний код для процесора. Для програміста рівень композиції відповідає синтаксису мови програмування. Якщо рівень композиції ще зменшити (тобто від найпростіших вище за схемою), то оракули стають логічно зрозумілішими.

Використання такого підходу дає змогу уніфікувати алгоритми та схеми написання програмного коду для будь-якого продукту чи системи. Починаючи від повсякденних гаджетів і закінчуючи системами життєзабезпечення космічних апаратів. Це досягається шляхом усунення надмірного індивідного впливу конкретного програміста на кінцевий продукт.

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 [4]. Така формалізація процесу позитивно впливає на кінцеву надійність продукту, оскільки певною мірою зменшує вплив «людського фактора» на надійність роботи програми.

III. Практичне використання

Практичне використання оракульного концептування показано на прикладі розробки людино-машинного інтерфейсу за кодом програми зображеної на Рис. 4.

Показана вище програма є простим лічильником, який рахує від 0 до 7 з можливістю скидання по інверсному значенню сигналу rstn та можливістю вибору “напрямку” рахування по значенню сигналу up_down. Для мінімізації впливу індивідуальних особливостей конкретного програміста на написання такого коду, код програми максимально уніфікований таким чином, що програмісту необхідно обрати лише модуль, який він хоче реалізувати (в даному випадку – лічильник), та обрати його розрядність. Усе інше здебільшого не потребує втручання. Схематична реалізація такого інтерфейсу показана на Рис. 5.

Відповідно до зображеної схеми програмісту необхідно з існуючого переліку обрати модуль, який йому потрібно реалізувати. Зрозуміло, що перелік має бути стандартизованим відповідно до сучасних норм та не мати доступу до редагування для сторонніх осіб. Допустимі зміни заздалегідь визначаються та можливі до внесення шляхом використання людино-машинного інтерфейсу. Тобто в даному прикладі користувач може змінити лише розрядність лічильника. Усе інше покладається на інтерфейс користувача.

```
module ctr (input              up_down,
                               clk,
                               rstn,
            output reg [2:0]   out);

    always @ (posedge clk)
        if (!rstn)
            out <= 0;
        else begin
            if (up_down)
                out <= out + 1;
            else
                out <= out - 1;
        end
endmodule
```

Рис. 4 Реалізація лічильника на мові Verilog

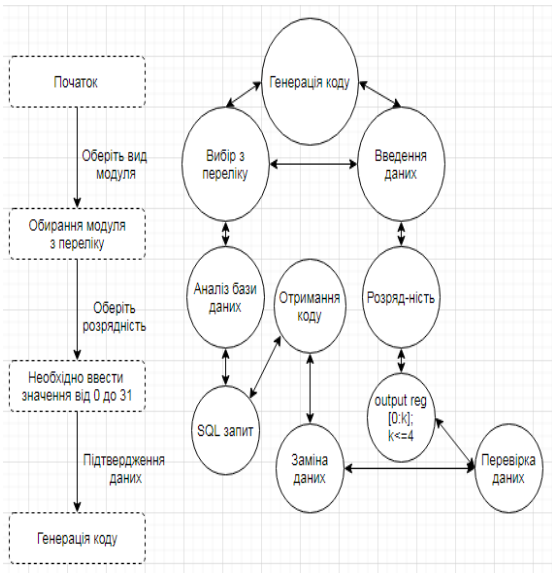

Рис. 5 Схематична реалізація інтерфейсу

Проте залишається питання реалізації такого способу програмування, адже інтерфейс при оракульному концептуванні згенерує код, який можливо буде редагувати як при звичному способі написання програм. І це буде цілком нормальним явищем, оскільки завдання програмування часто дуже специфічні і потребують всестороннього підходу до вирішення, що реалізується саме завдяки описаній на початку роботи надмірній індивідності процесу.

Слід розуміти, що використання оракульного концептування має полегшити рутинні процеси програмування. Усі загально визначені прийоми в написанні, як і власне конкретні модулі, якщо говорити про мову програмування з прикладу будуть уніфікованими і загальнодоступними для усіх, коли надбання кожного окремого суб’єкта будуть доступні для всіх бажаючих.

Висновки

Розглядаючи програмування з точки зору інтер-суб’єктивної парадигми стає зрозумілим, що це процес спрямований на створення програми. Згідно з цією парадигмою одним зі способів досягнення

Copyright (c) 2021 Зилевіч М. О.

DOI: 10.20535/2617-0965.eae.227740

такої спрямованості є оракульне концептування, що описує концептування через оракульні структури. Відповідно, концептуванням вважається процес створення концепту. Концептом називають план певної дільності, спрямований на розв'язання певної конкретної задачі. Тобто, під оракульними структурами слід розуміти сукупність певних елементарних підзадач, що виникли внаслідок концептування конкретної задачі програмування. Для подальшого розвитку засад адаптивного технологічного середовища програмування на основі інтерсуб'єктивної парадигми при розв'язанні програмістських задач розроблено та запропоновано спосіб застосування підходу оракульного концептування. На прикладі програмування дизайну електронних мікросхем показані його особливості та перспективи подальшого розвитку. До особливостей відноситься те, що кожна підзадача може бути розкладена до сукупності найпростіших підзадач. Також використання оракульного концептування дає можливість використання традиційного математичного апарату для нотації результату та поєднання його з денотативними методами. Реалізація такого методу на практиці сприяє уніфікації процесу розробки програмного продукту, тим самим зменшує вплив активної ролі суб'єкта і підтримує впровадження інтерсуб'єктивної парадигми. Подальші дослідження з цієї теми будуть спрямовані на їх розширення та впровадження, вивчення фактографії і розвитку відповідної фактології для оракульного концептування як результативного засобу уніфікації певних видів задач і розвинених на її основі редукційних методів програмування.

Надійшла до редакції 06 березня 2021 р.

ПЕРЕЛІК ПОСИЛАНЬ

[1] R. Harper, *Practical foundations for programming languages, second edition*. Cambridge University Press, 2016, **ISBN:** 9781316576892.

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

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

[4] I. Redko, P. Yahanov, and M. Zylevich, "Concept-Monadic Model of Technological Environment of Programming," in *2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC)*, 2020, pp. 1–5, **DOI:** 10.1109/SAIC51296.2020.9239204.

[5] I. V. Redko and P. O. Yahanov, "CONCEPTUAL MODEL OF TECHNOLOGICAL ENVIRONMENT OF PROGRAMMING," *KPI Sci. News*, no. 1, pp. 18–26, Mar. 2020, **DOI:** 10.20535/kpi-sn.2020.1.197953.

[6] I. V. Redko, D. I. Redko, and T. L. Zakharchenko, *Conceptual basis of programming*. Kyiv, Ukraine: Komprynt, 2016.

[7] B. A. Trakhtenbrot, *Algorithms and computational mashines*. Moscow, Russia: Sovetskoe radio, 1974.

[8] D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[9] G. S. Plesniewicz, D. E. Masherov, Nguen Thi Min Vu, and A. B. Karabekov, "“Binary Knowledge Model”: Specifying, instantiating, and interpreting advanced ontologies," in *2015 9th International Conference on Application of Information and Communication Technologies (AICT)*, 2015, pp. 314–318, **DOI:** 10.1109/ICAICT.2015.7338570.

[10] V. A. Pedroni, *Finite State Machines in Hardware Theory and Design (with VHDL and SystemVerilog)*. 2013, **ISBN:** 9780262019668.

Copyright (c) 2021 Зилевіч М. О.

UDC 004.421

DOI: 10.20535/2617-0965.eae.227740

Oracle Concepting in Programming of the Design of Electronic Chips

M. O. Zylevich, ORCID 0000-0003-1646-0557
Department of design of electronic digital equipment keoa.kpi.ua
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", ROR 00syn5v21
Kyiv, Ukraine

***Abstract*—Today, programming is one of the most actively developing technological areas in the world. Moreover, every year the degree of such development only increases, which is manifested in the deeper and more comprehensive penetration of different types of software products into different spheres of human life. This is an integral part of the process of technologicalization of the modern world. Speaking of the development of programming, it is believed that the theory of programming is studied and researched at a fairly high level. Undoubtedly, the process of creating a program uses generally defined algorithms, structures, approaches, and methods, but these are common things that operate at the planning stages. When it comes to the specific process of creating a piece of code that a particular person is involved in, this is where all the excessive subjectivization manifests itself. After all, each such person when writing code is guided by their experience, which is determined by education, knowledge, skills, worldview, and other components. To solve the problem associated with such excessive subjectivization, it is advisable to replace the generally accepted individual-subject paradigm, which involves defining the program through its creator, with intersubjective, in which the process of creating a program is the object of study and the subject itself a program that is the embodiment of a plan for this process. One of the methods of solving this problem, according to the concept-monad model, is the use of the oracle concept. The purpose of this work is to further develop the principles of the adaptive technological environment of programming based on the intersubjective paradigm, through the use of the oracle concept in solving programming problems. To achieve the above goal, it is necessary to develop an algorithm for applying the oracle concept approach. Identify its features and prospects for further development. The implementation of this method in practice in some way unifies the process of software development, thereby reducing the impact of the active role of the subject and supports the implementation of the intersubjective paradigm. Conceptualization is the process of creating a concept. A concept is a plan of a certain activity, aimed at solving a specific problem. That is, oracular structures should be understood as a set of certain elementary subtasks that have arisen as a result of conceptualizing a specific programming task. To further develop the principles of an adaptive technological programming environment based on the intersubjective paradigm in solving programming problems, a method of applying the oracle concept approach has been developed and proposed. Representative examples show its features and prospects for further development. The peculiarities include the fact that each subtask can be conceptualized as the simplest subtask. The implementation of this method in practice helps to unify the process of software product development, thereby reducing the impact of the active role of the subject and supports the introduction of an intersubjective paradigm. Further research on this topic will focus on the expansion and research, factography, and development of relevant facts for oracle conceptualization as an effective means of unification of certain types of problems and developed on its basis reduction methods of programming.**

Keywords — method; synthesis; intersubjective paradigm; oracle; oracle scheme; software environment; programming; Verilog.

Copyright (c) 2021 Зилевіч М. О.