DOI: 978-966-301-265-0/1.2023.55

Захаренков Д.Ю.
https://orcid.org/0000-0003-3951-022X
e-mail: dmit.zakharen@gmail.com

Яровий Р.О.
https://orcid.org/0000-0001-8978-8137
e-mail: roman.yaroviy@e-u.edu.ua

Улічев О.С.
https://orcid.org/0000-0003-3736-9613
e-mail: o.ulichev@e-u.edu.ua

КЛАСИФІКАЦІЯ ПРОГРАМ, ОРІЄНТОВАНА НА ТЕХНОЛОГІЮ ПРОГРАМУВАННЯ

Анотація. Розглядається класифікація програм щодо їх внутрішньої організації, що визначає інтерфейс із зовнішнім оточенням, форму специфікації програми та інші особливості. Класифікація спрямована на розробку адекватної технології для кожного класу програм. Визначено три класи програм: клас програм-функцій, клас реактивних систем та клас мовних процесорів.

Ключові слова: програма, технологія програмування, визначення вимог, специфікація програми, реактивна система, формальна семантика.

Zakharenkov Dmytro, Yarovyi Roman, Ulichev Oleksandr

CLASSIFICATION OF PROGRAMS ORIENTED ON PROGRAMMING TECHNOLOGY

Abstract. There is a classification of programs based on their internal organization, which means the interface from external details, the form of program specifications and other features. The classification is aimed at developing adequate technology for the skin class of the program. There are three classes of programs: the class of program functions, the class of reactive systems and the class of processors.

Key words: program, programming technology, designation, program specification, reactive system, formal semantics.

Виклад основного матеріалу. Історія розвитку програмної інженерії демонструє калейдоскопічну зміну великої кількості моделей, методів та інструментальних засобів. Технології програмування, що розробляються, в основному орієнтовані на цілком певні обмежені класи програм. Загальновідомі такі класи програм, як, наприклад, реактивні системи, системи реального часу, імовірнісні автомати.

Навіть універсальна технологія програмування не є однаково успішною для всіх програм. Тому корисно визначити клас програм, які відповідають кожній технології. Актуальним завданням є побудова повної системи класів програм, де кожен клас визначається своєю особливою внутрішньою організацією програм, що визначає технологію програмування.

У світовій літературі не виявлено спроб збудувати повну систему класів програм. У роботах зустрічаються лише загальні поверхневі визначення класів, зазвичай, без усталених загальноприйнятих математичних моделей. Відомі інші класифікації: класифікація програм за їх призначенням, класифікація мов програмування та класифікація парадигм програмування. Усі вони мають мало спільного з необхідною класифікацією програм, що враховує внутрішню організацію програм.

Спроба створення загального теоретичного базису для різних класів і парадигм зроблено Хоаром та іншими вченими в рамках серії робіт із уніфікованих теорій програмування (Unifying theories of programming, UTP) [1]. До кожного класу програм визначається мінімальне ядро мови програмування; далі розробляються формальна семантика ядра та методи формальної верифікації. Ці роботи обмежені створенням загальної математичної теорії програмування і поки що не отримали продовження у створенні нових інструментів та технологій програмування.

У цьому контексті слід також відзначити ініціативу SEMAT (Software Engineering Method and Theory) [2], оголошену в 2009 р., яка отримала підтримку великої кількості вчених та наукових організацій. Мета ініціативи — створення єдиної теорії в програмній інженерії, яка стала б основою її подальшого розвитку. Відзначався розрив між науковими дослідженнями та практикою програмування. За відсутності загального теоретичного базису сучасний розвиток програмної інженерії виявився залежним від рекламного продавлювання своїх продуктів великими софтверними корпораціями. Були сформульовані основні положення та цілі ініціативи, які послужили основою для її обговорення на кількох наукових конференціях, у тому числі із

застосуванням мозкового штурму. Через чотири роки роботи в рамках ініціативи було припинено.

Предикатне програмування [3-6] виникло як заперечення імперативного програмування. Клас предикатних програм визначений як клас програм-функцій, що відповідає моделям "програма є функція" або "програма це предикат", аналогічно UTP [1]. Клас реактивних систем реалізований в іншій парадигмі - в автоматному програмуванні [7]. Тут використовується автоматна модель програми як гіперграфа, що є продовженням апарату гіперфункцій [3, 6] в предикатному програмуванні.

Цікавий досвід поширення функціонального програмування на клас реактивних систем, зокрема, з використанням мов Haskell та Erlang, на базі акторської моделі, що допускає взаємодію між паралельними підпрограмами реактивної системи лише через прийом та надсилання повідомлень. Зазначимо, що акторська модель покриває лише частину класу реактивних систем — не допускаються змінні, що розділяються, які обов'язкові, наприклад, у серії протоколів взаємного виключення.

Роботи з класифікації програм розпочалися зі спроби знайти точну межу між класом реактивних систем та класом програм-функцій. Виявилося, що ці два класи не покривають усієї множини програм, зокрема, програм трансляторів з мов програмування.

Справжня робота з класифікації програм — це одна із спроб збудувати єдиний фундамент у програмній інженерії. Визначено три класи програм: клас програм-функцій, клас реактивних систем та клас мовних процесорів.

Клас програм-функцій

Програма належить цьому класу, якщо вона не взаємодіє із зовнішнім оточенням; точніше, якщо можна перебудувати програму таким чином, щоб усі оператори введення даних перебували на початку програми, а весь висновок було зібрано наприкінці програми. Якщо така перебудова програми є принципово неможливою, її слід намагатися визначати у вигляді реактивної системи. Програма повинна завжди нормально завершуватися з отриманням результату, оскільки нескінченно працююча і не взаємодія програма марна. Програма визначає функцію, що обчислює набір вхідних даних (аргументів) деякий набір результатів. Клас програм функцій, що містить, щонайменше, програми для завдань дискретної та обчислювальної математики [8].

Таким чином, оточення програми-функції має просту структуру і складається із засобів введення аргументів та виведення результатів програми.

Програму-функцію U з набором аргументів x та набором результатів y будемо записувати як U(x: y). Специфікацією програми-функції є два предикати: передумова P(x) та постумова Q(x, y). Передумова обмежує допустимий набір аргументів, а постумова визначає зв'язок між значеннями

аргументів та результатів. Специфікацію програми записуватимемо у вигляді [P(x), Q(x, y)].

Програма має відповідати специфікаціям. Ця вимога формулюється у вигляді умови часткової коректності: якщо перед виконанням програми U(x: y) істинно передумова P(x), то у разі завершення програми має бути істинна постумова Q(x, y) у момент її завершення. Обов'язковою є також умова завершення програми: якщо перед виконанням програми U(x: y) істинно передумова P(x), то програма має завершуватися. Поєднання умов часткової коректності та завершення програми визначає умову тотальної коректності.

Адекватна формалізація умов коректності програми можлива лише за умови використання формальної операційної семантики мови програмування.

Адекватна формалізація умов коректності програми можлива лише за умови використання формальної операційної семантики мови програмування. Операційну семантику програми U(x: y) визначимо у вигляді предикату: R(U)(x, y) ≅ для значення набору x виконання програми U завжди завершується і існує виконання програми, у якому результатом обчислення є значення набору y.

Дане визначення виключає ситуацію, коли для деякого значення набору x існують два різних виконання програми, одне з яких завершується, а інше не завершується. У цьому випадку R(U)(x, y) буде хибним для будь-яких y.

Клас програм-процесів

Програма-процес є реактивною системою, що реагує на певний набір подій (повідомлень) у зовнішньому оточенні програми. Взаємодія програми з оточенням реалізується через прийом/надсилання повідомлень і змінні, що розділяються, доступні в програмі та оточенні.

Роздільна змінна є глобальною стосовно програми-процесу. Вона доступна для читання та/або запису всередині програми, а також може бути модифікована в оточенні програми.

Програма-процес є або автоматною програмою, або вона визначається у вигляді композиції декількох автоматних програм, що виконуються паралельно і взаємодіють між собою через повідомлення та змінні, що розділяються.

Автоматна програма складається з одного або кількох сегментів. Сегмент має один вхід, позначений міткою - станом, що управляє. Сегмент має один чи кілька виходів. Вихід реалізується оператором переходу на початок іншого сегмента (або того самого), або як вихід з автоматної програми. Автоматна програма визначає кінцевий автомат у вигляді гіперграфа з набором керуючих станів як вершин і набором сегментів як орієнтовані гіпердуги. Існує один початковий керуючий стан, з якого починається виконання автоматної програми. Автоматна програма може мати один або

кілька вихідних станів керування, якими завершується виконання автоматної програми. Зазначимо, що гіперграфова модель є найбільш загальною серед інших моделей автоматних програм, що використовуються.

Стан автоматної програми визначається значеннями набору змінних, що модифікуються, локальних по відношенню до автоматної програми і глобальних по відношенню до кожного її сегменту.

Клас мовних процесорів

Мовні процесори — інтерпретатори програм, транслятори, оптимізатори, трансформатори, змішані обчислювачі, конвертори та інші операції з програмами.

Мова програмування - формальна мова, що визначає правила запису програми у вигляді тексту в кінцевому алфавіті символів. Лексичні правила визначають правила кодування символів алфавіту (лексем) за допомогою алфавіту комп'ютера, а також програми загалом у конкретній файловій системі.

Опис мови програмування визначає: типи даних, структуру пам'яті програми, що виконується, види мовних конструкцій програми, правила виконання конструкцій кожного виду і програми в цілому.

Для будь-якого виду мовних конструкцій визначено синтаксис та семантику. Синтаксис визначає правила представлення конструкції як послідовності символів алфавіту. Семантика визначає правила виконання довільної конструкції цього виду. Частиною семантики є статична семантика: правила ідентифікації змінних та інших об'єктів програми, сукупність обмежень на типи (та інші атрибути) конструкцій та система умовчань. Друга частина семантики є власне семантикою виконання програми.

Клас мовних процесорів значно складніше класу програм-функцій. Специфікація мовного процесора базується на формальній семантиці, що оперує безліччю виконання довільної програми.

Висновки. У роботі розглядається класифікація програм з їхньої внутрішньої організації, визначальною інтерфейс із зовнішнім оточенням, форму специфікації програми, базисні мовні конструкції та інші особливості. Класифікація спрямовано на розробку адекватної технології кожному за класу програм. На основі загальної системи понять, що використовуються в програмній інженерії, визначено три класи програм: клас програм функцій, клас реактивних систем та клас мовних процесорів. Надалі належить ідентифікувати та визначити клас операційних систем, що є метарівневим по відношенню до перших двох класів. Наведена система класів свідомо неповна. Для продовження класифікації програм необхідно проаналізувати різні види додатків, зокрема, пов'язані з базами даних та фреймворками.

Одним із завдань класифікації програм є подальша деталізація підкласів усередині класу реактивних систем. Завдання наступного рівня — побудова моделей підкласів з орієнтацією технології програмування.

Література

1. Hoare C. A. R., Jifeng He. Uni fying Theories of Programming. Prentice Hall Series in Computer Science. Prentice Hall Europe. 1998.
2. Software Engineering Method and Theory. URL: http:// www.semat.org/
3. Hewitt C., Bisshop P., Steiger R. A Universal Modular Actor Formalism for Artificial Intelligence // 3rd International Joint Conference on Artificial Intelligence, Stanford, CA, 1973. P. 235—245.
4. IEEE Recommended Practice for Software Requirements Specifications. Revision: 29/Dec/11.
5. Mich L., Franch M., Novi Inverardi P. Market research for requirements analysis using linguistic tools // Requirements Engineering 2004. Vol. 9, N. 1. P. 40—56.
6. Zhang W., Beaubouef T., Ye H. Statechar t: A Visual Language for Software Requirement Specification // International Journal of Machine Learning and Computing. 2012. Vol. 2. No. 1. P. 52—61.
7. Abrial J.-R. Modelling in Event-B: System and Software Engineering, Cambridge Univ. Press, 2010.
8. Alur R., Dill D. L. A theory of timed automata // Theor. Comput. Sci. 1994. Vol. 126. N. 2. P. 183—235.