Застосування автоматних моделей контролю дає можливість підвищити ефективність тестування компонент програмного забезпечення контролю польотів (ПЗ КП) шляхом автоматизації пошуку моментів контролю за ПІ, здійснити генерацію і внесення подій контролю в копію польоту, автоматизувати пошук етапів і моментів контролю в системах створення тестових наборів даних при сертифікаційних іспитах ПЗ КП [13], проектувати логічні й електричні схеми розпізнавальних автоматів з мінімальним набором елементів, які реалізують будь-які множини алгоритмів контролю польотів ПС, використовувати розпізнавальні автомати на борту ПС для визначення відхилень у процесі польоту і для контролю кабінних табло і приладів та автоматні моделі при створенні ПЗ КП [4].

Список літератури

1. *Малежик А.И.* Основы компьютерных технологий оперативного контроля полетов воздушных судов по полетной информации. – К: КМУГА, 1996. – 124 с.

2. *Малежик А.И., Харченко А.Г.* Математическое обеспечение автоматизированных систем контроля полетов. – К.: КИИГА, 1986. – 60 с.

3. *Яцков Н.А.* Основы построения автоматизированных систем контроля полетов воздушных судов. – К.: КИИГА, 1989. – 344 с.

4. *Райчев И.Э., Харченко А.Г.* Применение конечных автоматов для реализации алгоритмов контроля полетов воздушных судов // Вісн. НАУ. – 2001. – №3. – С.136–140.

5. *Льюис Ф., Розенкранц Д., Стирнз Р.* Теоретические основы проектирования компиляторов. – М.: Мир, 1979. – 654 с.

6. *Вельбицкий И.В., Ходаковский В.Н., Шолмов Л.И.* Технологический комплекс производства программ на машинах ЕС ЭВМ, БЭСМ-6. – М: Статистика, 1980. – 263 с.

7. *Вельбицкий И.В.* Технология программирования. – К: Техника, 1984. – 279 с.

8. *Глушков В.М.* Синтез цифровых автоматов. – М: Физматгиз, 1962. – 476 с.

9. *Кобринский Н.Е., Трахтенброт Б.А.* Введение в теорию конечных автоматов. – М: Физматгиз, 1962. – 404 с.

10. *Горбатов В.А.* Основы дискретной математики. – М.: Высш. шк., 1986. – 311 с.

11. *Глушков В.М.* Теория автоматов и формальные преобразования микропрограмм // Кибернетика. – 1965. – №5. – С. 1–10.

12. *Антонов А.Н.* Язык описания цифровых устройств Altera HDL. – М.: ИП Радиософт, 2002. – 224 с.

13. *Райчев И.Э., Харченко А.Г., Яцков Н.А.* Исследование методов тестирования программных модулей обработки полетной информации // Вісн. КМУЦА. – 2000. – №1–2. – С.127–133.

Стаття надійшла до редакції 29.05.02.

УДК 683. 01

Ю.М. Крамар, асп.

ЗАСОБИ ДЛЯ АВТОМАТИЗОВАНОГО СИНТЕЗУ СТИЛІВ ПРОГРАМУВАННЯ

Розглянуто засоби розв'язування задачі синтезу стилів програмування. Запропоновано схему систематизації правил і загальну форму їхнього подання.

Вступ. Застосування стилю програмування важливе як при розробці програм [1; 2], так і під час навчання програмуванню [3].

Під стилем програмування звичайно розуміють набір правил для написання текстів програм, а сутність застосування стилю полягає у дотриманні набору правил стилю при створенні текстів програм і у перевірці цих текстів на відповідність вимогам стилю.

Засоби подання правил стилів програмування. Стиль програмування – це концепція написання програм, що пронизує всі процеси створення програмного забезпечення та подається за допомогою набору правил. Стиль відбиває не тільки технічний, але й культурний досвід, який існує у період життєвого циклу програмного забезпечення [4]. Беручи до уваги

об'єктивні труднощі, пов'язані, наприклад, з вивченням стилів специфікування вимог або проектування програмного забезпечення, а також враховуючи особливості реалізації процесів життєвого циклу, прояв стилю найпростіше знайти в тексті програми. Стилі, що застосовуються в процесі кодування, є наборами правил для програміста та подаються в програмних середовищах і літературних джерелах [1; 2; 5–9].

Програмні середовища можуть бути трьох типів: редактори вихідних текстів (наприклад, редактор текстів у середовищі програмування Turbo Pascal, редактор утиліти QuickDesk, редактор EMACS), середовища об'єктно-орієнтованого програмування (наприклад, C Builder, Delphi) та спеціальне програмне забезпечення (Lint [2], утиліта cb).

Редактори забезпечують подання стилю, підтримуючи, наприклад, тільки правила відступів і визначений тип коментарів (утиліта QuickDesk). У середовищах програмування при автоматичній генерації коду дотримуються правил побудови ідентифікаторів, що рекомендуються Microsoft, правил розташування відступів, що прийняті в структурному програмуванні [7], деяких правил стилю для об'єктно-орієнтованого програмування (наприклад, «При описі класу використовуйте кожний специфікатор доступу один раз», «Кожний елемент класу описується окремим рядком»). Спеціальне програмне забезпечення (Lint) забезпечує стислий опис наданого їм набору правил стилю та контроль їхнього застосування в текстах програм.

Перераховані засоби автоматизують застосування стилю, проте мають істотні недоліки, наприклад, обмежену кількість правил, що підтримуються (редактори), «нав'язування» правил користувачеві (середовище програмування) і відсутність можливості вибору правил користувачем (Lint).

Літературні джерела [1; 2; 5–9] поділяються на дві групи. У першій групі інформація про стиль є допоміжною і вводиться до основного матеріалу з мови програмування [8; 9], а в другій групі матеріал, присвячений стилям програмування, є основним [1; 2; 5–7].

Характерним прикладом літератури першої групи є підручник з мови програмування C++ [8]. У ньому навчання мові програмування сполучається з вивченням правил стилів програмування, тому правила рознесено по главам підручника. Такий підхід має перевагу – читачеві відразу прищеплюється деякий стиль програмування, проте є і недолік – програмісту, що володіє мовою C++ і бажає вивчити тільки стиль, необхідно прочитати увесь підручник, самостійно відбираючи і систематизуючи інформацію, що стосується стилю. Усі правила в підручнику розподілені по чотирьом групам: гарний стиль програмування, зауваження з мобільності, поради з підвищення ефективності та техніка програмування.

Література другої групи є кориснішою, тому що вона вузькоспрямована на вивчення стилю програмування. У ній матеріал про мову програмування є допоміжним, тому читач повинен мати певний рівень мовної підготовки. Зміст більшості книг будується таким чином, що розділи присвячуються розгляду окремих правил стилю програмування, а не опису конструкцій мови [2].

Керуючись матеріалом книги Б. Кернигана, Ф. Плоджера «Елементи стилю програмування» [5], програміст може написати добре структурований, зрозумілий текст. Опис стилю програмування тут подано вербальними правилами, з поясненнями і прикладами застосування. Кожне правило викладено в такий спосіб: наводиться текст програми, що не відповідає стилю; текст аналізується і подається переписана програма, в якій усунені недоліки; потім формулюється правило стилю. Наприклад, значна частина правил має вигляд таких рекомендацій: «Форматуйте текст програми, щоб у ній було простіше розібратися», «Пишіть зрозуміло – не приносьте ясність у жертву ефективності».

Недоліками таких правил є неконкретність, двозначність і нечіткість їхнього формулювання. Так, у першій рекомендації, виходячи з її формулювання, не можна конкретно визначити, яким чином необхідно форматувати текст, а словосполучення «простіше розібратися», вочевидь, не може використовуватися в чітких визначеннях. Подальшого

роз'яснення також потребує вказівка «писати зрозуміло» у другій рекомендації. Ці недоліки формулювань компенсуються численними прикладами і докладними роз'ясненнями кожної рекомендації.

Пізніші описи правил С. Мейерса [2] і А. Голуба [1], що також відносяться до літератури другої групи, точніше та коректніше описують стилі в об'єктно-орієнтованому програмуванні, але і вони не позбавлені недоліків. Наприклад, до правила №8 «При написанні операторів new і delete дотримуйтеся ряду простих правил» необхідні уточнення, про які ж прості правила йде мова. У правилі №9 «Намагайтеся не приховувати «нормальну» форму new» з контексту опису неясний зміст визначення слова «нормальний».

Більш суворий опис правил, що також відноситься до літератури другої групи, подано Е. Нуквистом [6]. Автор пропонує свою систематизацію правил. Матеріал книги розбито на глави, що відповідають конструкціям мови С++, до яких застосовуються дані правила. У кожній главі в довільному порядку наводяться короткі загальні відомості про конструкцію мови, якій присвячено главу, правила і приклади їхнього застосування. Усі вказівки з написання текстів програм розбиваються на дві групи: правила, виконання яких є обов'язковим для програміста, і правила рекомендаційного характеру. Для правила наводяться винятки. Усі правила і рекомендації подано в єдиній синтаксичній формі.

Недоліком набору правил і рекомендацій Е. Нуквиста є те, що вони орієнтовані тільки на мову С++, а наведені відомості про мовні конструкції часто мають необґрунтовано великий об'єм. Це заважає швидкому вивченню матеріалу, що стосується безпосередньо стилю.

Отже, набори правил різних авторів відрізняються один від одного формулюванням, систематизацією, кількістю правил, що до них входять, складом, а також орієнтацією на різні методології програмування. Їхнє викладання орієнтовано на вивчення, а не на довідкове використання. Правила орієнтовані на визначену мову програмування, а особливості синтаксису цієї мови відбиваються на засобі їхнього подання.

Саме тому такі описи стилю програмування можуть бути корисними при навчанні програмуванню як додаткова література, але не підходять для автоматизованого застосування стилю програмування в процесі розробки програмного забезпечення.

Схема систематизації правил стилів програмування. Для побудови схеми систематизації необхідно вибрати ознаки систематизації [10]. Беручи до уваги результати аналізу правил, будемо їх систематизувати, використовуючи п'ять ознак: характер застосування правила, авторство правила, об'єкт застосування правила у програмному та об'єктному аспекті та рівень інкапсуляції об'єктів [3], до яких застосовується правило.

Характер застосування правила. Багато авторів описів правил, з огляду на характер їхнього застосування, виділяють правила, дотримання яких є обов'язковим (наприклад, «Ідентифікатор повинен мати осмислену позначку, що відбиває призначення даного програмного об'єкта» або «Усі позначки повинні складатися англійською мовою»), і правила, що тільки рекомендуються до використання (наприклад, «Не використовуйте короткі ідентифікатори, що не відображують повністю сутність об'єкта, але уникайте дуже довгих ідентифікаторів» або «Використовуйте символи підкреслення в ідентифікаторах»). Будемо використовувати характер застосування правила як однієї з ознак, що систематизує, і розрізняти два типи правил: вказівки та рекомендації.

Авторство правила. Аналіз правил показує, що деякі правила запропоновані організаціями-розробниками програмного забезпечення (наприклад, «Використовуйте верхній та нижній регістри при побудові ідентифікаторів» Microsoft), інші – окремими програмістами (Угорська нотація Чарльза Симонії: «Використовуйте предикат для вказівки типу значень змінної»), треті пов'язані з визначеними програмними інструментами (наприклад, «Command1, Command2, label1, …» у Delphi), а велика частина з них не має автора (наприклад, «Використовуйте оператори new і delete замість mallock і free»). Тому пропонується систематизувати правила з визначення авторства в такий спосіб: організацій, приватні, інструментів і загальні («народні»).

Об'єкт застосування правила. Залежно від об'єкта застосування правил систематизацію можна виконувати в двох аспектах. Перший засновано на застосуванні правил до програми в цілому, а другий – окремо до програмних об'єктів. У першому аспекті програма традиційно розглядається такою, що складається з тексту і документації. Стилі специфікування вимог і проектування знаходять відображення у техніці програмування, тому враховуються в даному аспекті. У другому аспекті програма розглядається як спорудження, що складається з програмних конструкцій [3]. Так як у літературних джерелах правила стилю програмування часто розглядаються стосовно до програмних об'єктів (наприклад, до змінної, виразу, підпрограми) або до окремих аспектів програми (наприклад, до коментування, позначок), визначимо типи об'єктів застосування правил як двох ознак систематизації.

Рівень інкапсуляції об'єктів, до яких застосовується правило. Відомо, що опис програмної конструкції часто залежить від того, у яку програмну конструкцію вищого рівня вона інкапсульована [3]. Наприклад, для підпрограми як програмної конструкції застосовуються одні правила стилю, якщо вона є процедурою, функцією або макросом, та інші, якщо ця ж підпрограма є методом програмної конструкції – клас. Так, при написанні підпрограми рекомендується застосовувати правило «Не використовуйте в підпрограмі глобальних змінних», але при описі методу це правило не використовують, тому що поняття глобальних змінних відсутнє, а керуються, наприклад, правилом «Виносьте опис тіла методу за межі опису класу». Усі програмні конструкції можна розглядати на підпрограмному, модульному і класному рівнях інкапсуляції [3]. З огляду на подібну особливість застосування тих чи інших правил будемо їх систематизувати за ознакою рівня інкапсуляції об'єктів, до яких застосовується правило.

Отримана схема систематизації відображена на рис. 1.

Подання правил стилів програмування. Для того, щоб автоматизувати розв'язання задач, пов'язаних зі стилем програмування, недостатньо тільки систематизувати правила стилів програмування. Через те, що автоматизація ґрунтується на створенні бази даних правил, розглянемо правила у вигляді ненормалізованого відношення.

Звичайно правила зображають реченнями природної мови [1; 2; 5–7]. Кожне правило будемо описувати кортежем відношення зі схемою:

$S = (D, F)$,

де D – документальна частина подання правила:

$D = (D1, D2, D3, D4)$,

D1 – атрибут, опис правила; D2 – атрибут, приклад застосування правила; D3 – атрибут, пояснення або коментар до правила; D4 – атрибут, виняток; F – фактографічна частина (систематизовані ознаки):

$F = (F1, F2, F3, F4)$,

F1 – атрибут, характер правила; F2 – атрибут, авторство правила; F3 – атрибут, об'єкт застосування правила в програмному аспекті; F4 – атрибут, об'єкт застосування правила в об'єктному аспекті; F5 – атрибут, рівень інкапсуляції об'єктів, до яких застосовується правило.

Так як набір правил, що використовується при розробці програмного забезпечення, є відповідною концепцією написання програм, (наприклад, ефективність, мобільність, зрозумілість), а правило може входити до різних наборів, то для кожного правила вкажемо концепції, згідно з якими воно застосовується.

Для автоматизації розв'язку задачі перевірки текстів на відповідність вимогам стилю кожному правилу поставимо у відповідність норму як значення програмної метрики властивості об'єкта, на який спрямована дія даного правила [11]. Тоді кожне правило – це пара вигляду $(R(S), N)$, де R – правило зі схемою; S, N – норма. Зображені таким способом у базі правила будемо називати нормованими. Приклад подання правил у вигляді ненормалізованого відношення наведено у таблиці.

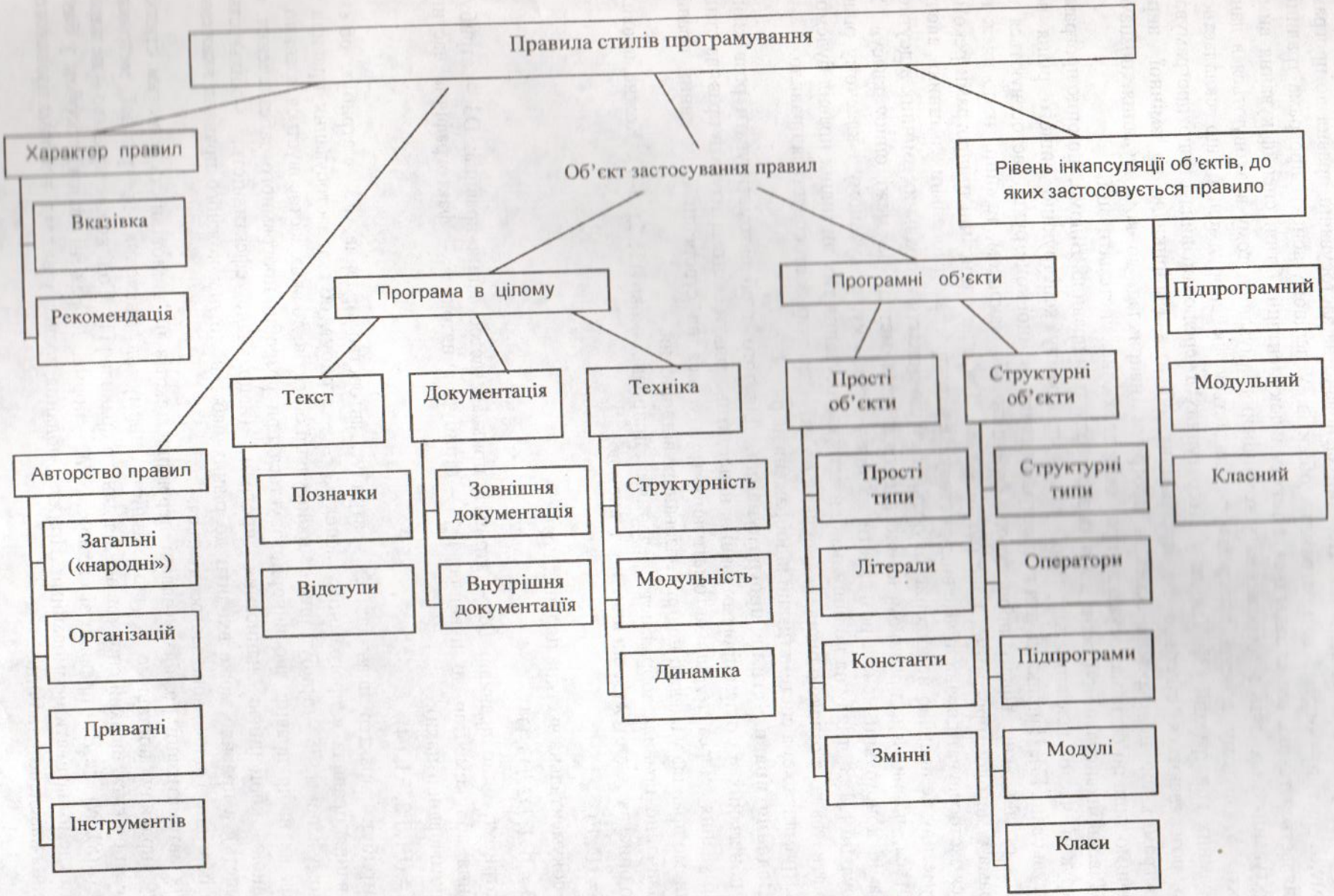

Рис. 1. Схема систематизації правил стилів програмування

Подання нормованого правила

Документальна частина			Фактографічна частина					Норма	Концепція
Опис правила	Приклад правила	Пояснення правила	а	х	п/а	о/а	р/і		
Робочі змінні, що мають малу область дії, описують короткими позначками	int ch; int i, j	Коротка позначка змінної інформує про те, що змінна – тимчасова, введена для збереження проміжних значень або виконання допоміжних функцій	з	р	п	зм	пп	3 символу	рз

Примітка. а – авторство; х – характер рекомендацій; п/а – програмний аспект; о/а – об'єктний аспект; р/і – рівень інкапсуляції; з – загальне; р – рекомендація; п – позначка; зм – змінна; пп – підпрограмний; рз – розуміння програми.

Задача синтезу стилю програмування. Сутність задачі синтезу стилю програмування полягає у формуванні на множині всіх систематизованих правил R^* деякої підмножини правил R_x, що відповідають концепції K_x синтезованого стилю S_x.

При включенні у підмножину R_x правила $r_i \in R^*$ необхідно перевірити виконання двох умов. По-перше, чи відповідає правило r_i концепції стилю, по-друге, чи можливе застосування правила r_i сумісно з іншими правилами підмножини R_x.

У програмуванні за весь період його існування сформувалася низка концепцій написання програм, в яких було відбито різні культури програмування:

$$K^* = \{K_l \mid l \in L\}.$$

Для перевірки виконання першої умови використовується наперед визначена для кожного правила r_i множина концепцій, за якими застосовується це правило:

$$r_i \Rightarrow \{K_{im} \mid m \in M\},\ M \subseteq L.$$

При включенні правила r_i у підмножину правил стилю R_x необхідно слідкувати, щоб одна з концепцій K_{im} відповідала загальній концепції стилю K_x:

$$\exists K_{im}(K_{im} = K_x).$$

Для перевірки виконання другої умови на множині правил R^* повинно існувати таке бінарне відношення G, що пара правил $<r_m, r_n>$ належить цьому відношенню, якщо застосування правила r_m не виключає можливості застосування правила r_n. Сутність відношення G складається в тому, що якщо пара правил $<r_m, r_n>$ належить цьому відношенню, то правила r_m та r_n можуть входити у підмножину правил R_x, яка складає деякий стиль S_x. У протилежному випадку в підмножину правил стилю можливе входження тільки одного з цих правил.

Вибираючи та включаючи правило r_i у підмножину R_x, необхідно спочатку визначити декартів добуток D_x на підмножині правил $(R_x \cup \{r_i\})$, а потім – кожну пару правил множини D_x перевірити на належність відношенню G. Якщо для кожної пари D_x виконується умова належності, то нова підмножина R_x може складати стиль S_x.

Відношення G є рефлексивним, симетричним та нетранзитивним, тому що

$$\forall r(rGr);$$

$$\forall r_m \forall r_n (r_m G r_n \rightarrow r_n G r_m);$$

$$\forall r_m \forall r_n \forall r_r \neg (r_m G r_n \wedge r_n G r_k \rightarrow r_m G r_k).$$

Ознака рефлексивності відношення дозволяє опустити перевірку пар правил виду $<r_m, r_m>$ на належність відношенню G. Ознака симетричності дозволяє перевірити тільки одну з двох пар

$<r_m, r_n>$ та $<r_n, r_m>$ на належність відношенню G. Відсутність ознаки транзитивності вимагає при перевірці пар $<r_m, r_n>$ і $<r_n, r_p>$ перевіряти також пару правил $<r_m, r_p>$ на належність відношенню G.

Отже, стиль S_x подається множиною нормованих правил R_x:

$$S_x = \bigcup_{q \in Q} R_q ,$$

де $\forall r_i \exists k_{im} (K_{im} = K_x)$;

$\forall r_i \forall r_j (r_i G r_j)$, $i \in Q, j \in Q$.

Розв'язання задачі синтезу стилю S можна зобразити таким чином. На вхід деякого пристрою синтезу стилю надходить множина нормованих правил R^* та інформація про концепцію K_x. На виході пристрою формується стиль S_k, що відповідає концепції K_x (рис. 2).

Архітектура засобів автоматизованого синтезу стилю програмування. Автоматизація розв'язання задачі синтезу стилю дозволяє автоматизувати включення правил до набору правил стилю, що синтезується, і отримати засоби, що автоматизують використання стилю при розробці програмного забезпечення.

Засоби вибору правил та підтримки використання стилю мають однакову архітектуру. Відмінності відбиваються в деталях реалізації та наборі функцій, що ними виконуються. Тому роздивимося архітектуру тільки засобів вибору правил.

Для вибору правил стилю програмування пропонується система, в основу якої покладено розроблену схему класифікації правил та форму їхнього зображення. Архітектура системи складається з таких компонентів: інтерфейсу, обчислювального пристрою і бази даних, що містить правила (рис. 3).

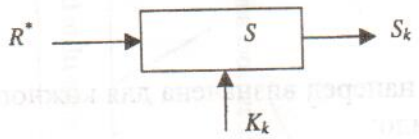

Рис. 2. Схема пристрою автоматизованого синтезу стилю

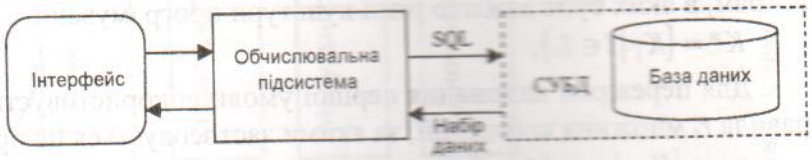

Рис. 3. Архітектура засобу автоматизованого синтезу стилю програмування

Інтерфейс призначений для взаємодії користувача з обчислювальним пристроєм як при заповненні бази даних правилами, так і в процесі синтезу стилю.

Обчислювальна система – це комплекс програм, призначений для розв'язання таких задач:

– формування інструкцій для системи управління базою даних (СУБД), що направлені на заповнення бази даних правилами, які вводяться користувачем;

– побудова обмеженого дерева систематизації правил стилів програмування, гілками котрого є ознаки загальної систематизації, що обрані користувачем, для вибору правил синтезованого стилю і відображення його користувачеві через інтерфейс системи;

– перетворення вказівок користувача про вибір правил в обмеженому дереві систематизації в процесі синтезу стилю програмування у відповідні інструкції для СУБД;

– відображення користувачеві через інтерфейс отриманих від СУБД наборів даних і результатів операцій з базою даних.

Інструкції СУБД являють собою SQL-запити вибірки, вставки і зміни даних.

База даних містить фактографічний та документальний описи правил, а також додаткову інформацію про правила, що використовуються в процесі синтезу стилю (наприклад, рівень вкладеності правила, сумісність правил). Схема бази даних складається з одинадцяти відношень, які отримані шляхом нормалізації схеми відношення, що зображує правила стилю і наведена на рис. 4.

Перші п'ять відношень (Character, Writer, Program_aspect, Object_aspect, Encapsulate) містять описи розділів класифікації за п'ятьма ознаками. Відношення Rule містить описи правил,

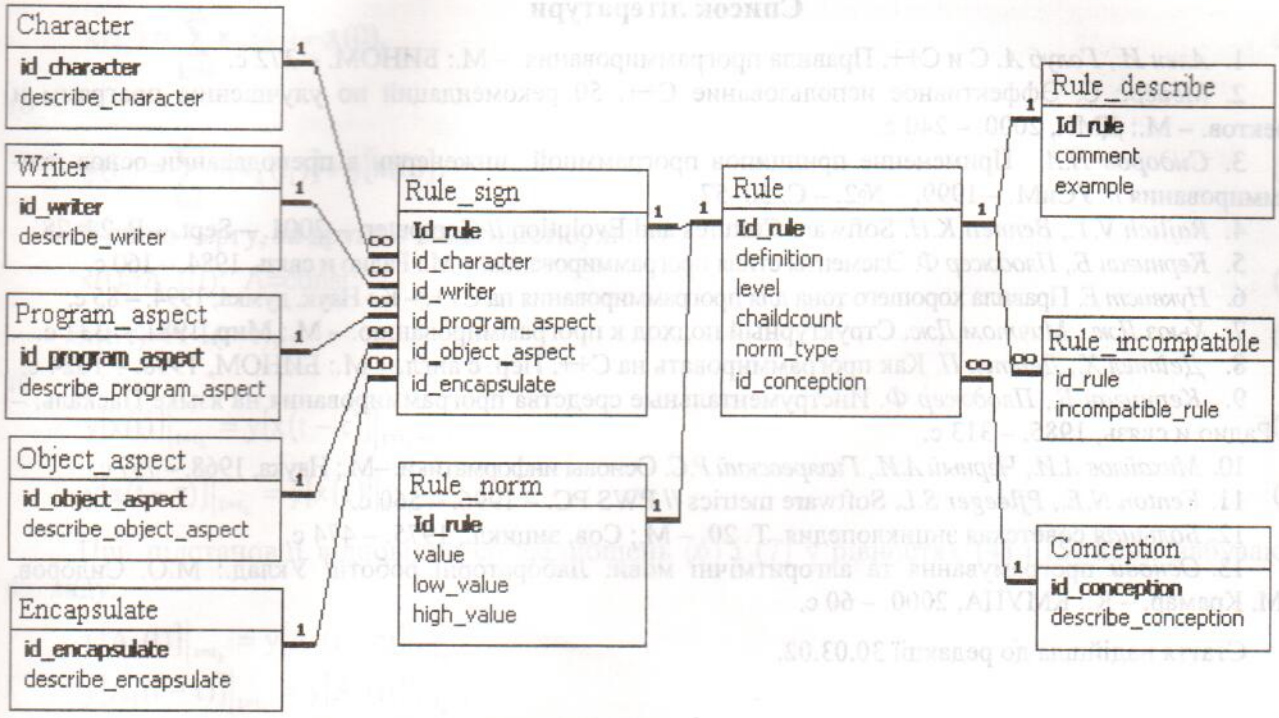

Рис. 4. Схема бази даних правил стилів програмування

ранг правила n [12], кількість вхідних у нього правил рангу (n-1) і тип норми, що підставляється до правила (значення або діапазон значень). Відношення Rule_sign містить значення індексів розділів класифікації, визначених для кожного правила, Rule_norm – норми правил, визначені за умовчанням, Rule_describe – пояснення і приклади застосування правил, Rule_incompatible – інформацію про сумісність правил, а Conception – опис концепцій стилів.

Робота системи в режимі синтезу складається з таких етапів:

– побудова обмеженого дерева систематизації для включення правил синтезованого стилю;

– включення правил як вирішення задачі синтезу стилю;

– формування окремої бази даних, в якій знаходиться інформація про правила тільки визначеного стилю.

Отже, результатом роботи системи є стиль програмування, наданий у вигляді підсистеми, що має власний інтерфейс і базу даних, сформовану в процесі синтезу. Отримана підсистема підвищує ефективність використання розробленого стилю при створенні програмного забезпечення.

Висновок. Розроблена на основі запропонованих систематизації та засобів і реалізована за допомогою СУБД INTERBASE 6.0. система синтезу стилю була використана при написанні навчальних програм у процесі розробки стилів програмування, застосованих студентами, що вивчають дисципліну «Алгоритмічні мови й основи програмування» на кафедрі інженерії програмного забезпечення Національного авіаційного університету.

При розробці стилів основною концепцією було розуміння первинного тексту, а також враховувалися такі чинники, як відносна простота алгоритму, невеличкий обсяг навчальних програм, використання обмеженого набору мовних засобів, зазначеного в завданні, наприклад, використання алгоритмів виконання побітових операцій, застосування операторів вибору, написання програми без використання підпрограм [13].

Досвід використання засобів показав необхідність автоматизації розв'язання не тільки задачі синтезу стилю, але й задачі контролю.

Список літератури

1. *Ален И., Голуб А.* С и С++. Правила программирования. – М.: БИНОМ. – 272 с.
2. *Мейерс С.* Эффективное использование С++. 50 рекомендаций по улучшению программ и проектов. – М.: ДМК, 2000. – 240 с.
3. *Сидоров Н.А.* Применение принципов программной инженерии в преподавании основ программирования // УСиМ. – 1999. – №2. – С. 50–57.
4. *Rajlich V.T., Bennett K.H.* Software Cultures and Evolution // Computer. – 2001. – Sept. – P. 24–28.
5. *Керниган Б., Плоджер Ф.* Элементы стиля программирования. – М.: Радио и связь, 1984. – 160 с.
6. *Нуквист Е.* Правила хорошего тона для программирования на С++. – К.: Наук. думка, 1994. – 85 с.
7. *Хьюз Дж., Мичтом Дж.* Структурный подход к программированию. – М.: Мир, 1981. – 331 с.
8. *Дейтел Х., Дейтел П.* Как программировать на С++. Пер. с англ. – М.: БИНОМ, 1998. – 1024 с.
9. *Керниган Б., Плоджер Ф.* Инструментальные средства программирования на языке Паскаль. – М.: Радио и связь, 1985. – 313 с.
10. *Михайлов А.И., Черный А.И., Гиляревский Р.С.* Основы информатики. –М.: Наука, 1968. – 757 с.
11. *Fenton N.E., Pfleeger S.L.* Software metrics // PWS PC. – 1996. – 560 с.
12. *Большая* советская энциклопедия. Т. 20. – М.: Сов. энцикл., 1975. – 474 с.
13. *Основи* програмування та алгоритмічні мови: Лабораторні роботи/ Уклад.: М.О. Сидоров, Ю.М. Крамар. – К.: КМУЦА, 2000. – 60 с.

Стаття надійшла до редакції 30.03.02.

УДК 629.735:519.835

Л.М. Іванова, асист.

ОЦІНКА ТОЧНОСТІ ТА ВИТРАТ ЧАСУ НА ЦИФРОВЕ МОДЕЛЮВАННЯ ХАРАКТЕРИСТИК ДИНАМІЧНИХ СИСТЕМ З ВИКОРИСТАННЯМ ВЛАСТИВОСТЕЙ ІНТЕГРАЛА ЗГОРТКИ

Розглянуто підхід до побудови цифрової моделі динамічної системи з використанням властивостей інтеграла згортки, підхід до оцінки точності функціонування побудованої моделі. Для цифрової моделі, побудованої з використанням пропонованого підходу, обчислено максимальну відносну точність моделювання δ та витрати часу на моделювання однієї точки реакції $t_м$. У результаті проведених експериментів побудовано поверхню $\pi(\delta, t_м, T) = 0$, де T – крок моделювання. Визначено, як по поверхні π можна вибрати оптимальні значення δ та $t_м$.

Для побудови цифрових моделей (ЦМ) динамічних систем (ДС) при невідомих апріорі законах зміни в часі вхідних сигналів пропонується метод з використанням властивостей інтеграла згортки.

Лінійні ДС вигляду:

$$\sum_{i=0}^{n} a_i y^{(i)}(t) = \sum_{j=0}^{m} b_j x^{(j)}(t), \qquad (1)$$

при заданих початкових умовах (ПУ):

$$y^{(n-1)}(t)\big|_{t=0} = y^{(n-1)}(0), \ldots, y'(t)\big|_{t=0} = y'(0),\ y(t)\big|_{t=0} = y(0),$$

де n, m – вищі порядки відповідно вихідного параметра і вхідного сигналу ($n \geq m$); a_i, b_j – коефіцієнти диференційного рівняння (ДУ), $y^{(i)}(t)$ – i-та похідна вихідного параметра; $x^{(j)}(t)$ – j-та похідна вхідного сигналу мають важливі для подальшого розгляду властивості. Для таких ДС справедливий принцип суперпозиції (накладання) реакцій систем від окремих вхідних впливів. Властивості цих ДС не змінюються в часі, оскільки визначені коефіцієнтами (1), що не залежать від t.

Відповідно до першої властивості для будь-якого значення t_0, якщо