3. Цой, Ю. А. Процессы и оборудование доильно-молочных отделений животноводческих ферм [Текст] / Ю. А. Цой. – М. : ГНУ ВИЭСХ, 2010. – 424 с.
4. Залманзон, Л. А. Микропроцессоры и управление потоками жидкости и газа [Текст] / Л. А. Залманзон. – М.: Наука, 1984. – 320 с.
5. Аксененко, М. Д. Микроэлектронные фотоприемные устройства [Текст] / М. Д. Аксененко, М. Л. Бараночников, О. В. Смолин. – М.: Энергоатомиздат, 1984. – 208 с.
6. Кузьмичев, В. Е. Законы и формулы физики [Текст] / В. Е. Кузьмичев. – К.: Наукова думка, 1989. – 864 с.
7. Ишанин, Г. Г. Источники и приемники излучения [Текст] : учебное пособие для студентов оптических специальностей вузов / Г. Г. Ишанин, Э. Д. Панков, А. Л. Андреев, Г. В.Польщиков. – СПб. : Политехника, 1991. – 240 с.
8. Интегральные микросхемы : Микросхемы для аналого-цифрового преобразования и средств мультимедиа [Текст] / ДОДЭКА,1996. – 384 с.
9. Де Монмоллен, Н. Системы «человек-машина» [Текст] / Н. Де Монмоллен. – М.: Мир, 1973. – 256 с.
10. Новицкий, П. В. Оценка погрешностей результатов измерений [Текст] : произв. изд. / П. В. Новицкий, И. А. Зограф. – Энергоатомиздат, 1991. – 304 с.

Пропонується метод архітектурної побудови SOA (System Oriented Architectures) з застосуванням парадигми керування потоками даних. Запропоновано метод графічного зображення потоку даних та їх обробки в складній SOA архітектурі, який дозволяє компіляцію основних складових частин в програмний код та полегшує процес створення архітектури інформаційних систем та їх тестування

Ключові слова: SOA (System Oriented Architecture), потоки даних, архітектура інформаційних систем

Предлагается метод архитектурного построения SOA (System Oriented Architectures) с применением парадигмы управления потоками данных. Предложен метод графического изображения потока данных и их обработки в сложной SOA архитектуре, который позволяет компиляцию основных составных частей в программный код и облегчает процесс создания архитектуры информационных систем и их тестирования

Ключевые слова: SOA (System Oriented Architecture), потоки данных, архитектура информационных систем

УДК 004.7

ПАРАДИГМА КЕРУВАННЯ ПОТОКАМИ ДАНИХ ТА ЇХ ГРАФІЧНЕ ПРЕДСТАВЛЕННЯ В SYSTEM ORIENTED ARCHITECTURES

К. В. Харченко
Кандидат технічних наук, доцент
Кафедра системного проектування
Інститут проблем системного аналізу
Національний технічний університет України
«Київський політехнічний інститут»
пр. Перемоги, 37, м. Київ, Україна, 03057

1. Вступ

Дві основні парадигми обчислень конкурують у світі комп'ютерних систем – системи обчислень з керуванням потоком інструкцій (instructions control flow) та системи обчислень з керуванням потоком даних (data flow) [1]. Кожна з них має свої переваги і широко застосовувалась протягом останніх десятиліть. Проте з розвитком різноманітних підходів до інформаційних систем та широким використанням SOA актуальним стає використання парадигми обчислень з керуванням потоками даних. Такий підхід органічно відображає взаємодію компонентів SOA та полегшує процес розробки архітектури складних інформаційних систем, дозволяє спеціалістам в предметній галузі брати більш широку участь у розробці програмних систем. На базі парадигми керування потоків даних розроблялись і використовуються потужні комплекси програмування [2].

Використання графічного представлення архітектури інформаційних систем і програм з об'єктно-орієнтованою моделлю протягом майже двох десятиріч застосовується в UML. Такий підхід має свої переваги що до побудови архітектури об'єктно-орієнтованих систем, але часом ускладнює процес розробки і утворює громіздку внутрішню побудову часом навіть для простих взаємозв'язків між невеликою кількістю сутностей. Також стандарт UML не дозволяє створити

механізми для повного автоматичного транслювання графічного представлення архітектури в програмний код, який є повністю функціональним.

Ще задовго до появи об'єктно-орієнтованого програмування було запропоновано графічний метод опису алгоритмів на основі парадигми керування потоками даних [3–5]. Переваги такого методу опису структури інформаційних систем дали поштовх подальшому розвитку методології графічного представлення архітектури програмних продуктів, і врешті до виникнення стандарту UML.

Запропонована у [6, 7] система графічного зображення коду програми у вигляді Р-схеми (стандарт ISO/IEC 8631) наочно, ясно та компактно описує алгоритми та їх реалізацію на різних мовах програмування за допомогою парадигми керування потоком команд. Але Р-схема не дозволяє відобразити потік даних між компонентами SOA системи, тому що вона орієнтована на керування потоком інструкцій.

В компонентах SOA дані передаються від компоненти до компоненти у вигляді REST або WSDL запитів, тому досить складно відобразити роботу інформаційної системи керуючись парадигмою потоку інструкцій.

Візуальне представлення програмного коду успішно використовувалось в графічних мовах програмування «Дракон», де використовувалася парадигма керування потоком даних. Існують реалізації систем розробки програмного забезпечення мови програмування «Дракон» в програмний код [8].

Системи програмування з використанням парадигми керування потоками даних набирають актуальність для сервіс-орієнтованої архітектури, а особливу увагу заслуговують методи графічного представлення для наочного відображення взаємодії компонентів складних інформаційних комплексів.

2. Аналіз літератури та постановка проблеми

У [9, 10] описано підхід до графічного представлення архітектури сервіс-орієнтованих систем за допомогою шаблонів проектування (SOA Design Patterns). Викладена методика дозволяє описувати SOA інформаційні системи, але цей стандарт є дещо громіздким та ускладнює компіляцію графічного представлення системи у програмний код.

Графічне представлення архітектури сервіс-орієнтованих систем дозволяє описувати структуру та взаємодію складових частин інформаційних систем, аналогічно до UML, з більшою деталізацією аспектів сервісних функцій та методів організації SOA [11]. Недоліком такого підходу є застосування UML стандарту, що приводить до великої кількості діаграм різних видів для повноти опису функціонування системи.

У [12] описаний фреймворк NoFlo для роботи на основі парадигми керування потоками даних, який може широко застосовуватися для рішення задач у інформаційних системах. Але даний фреймворк орієнтований на використання мови JavaScript, що призводить до певних обмежень використання.

Постановка проблеми полягає у створенні простого та інтуїтивно зрозумілого представлення систем, що керуються потоками даних.

Метою дослідження є опис простого та інтуїтивно зрозумілого графічного представлення сервіс-орієнтованої архітектури інформаційних систем з можливістю компіляції в робочий програмний код на об'єктно-орієнтованих мовах програмування.

Задачі дослідження полягають у створенні графічного представлення SOA з такими основними принципами:

– наочність схематичного представлення компонентів системи, мотивація користувача створювати ієрархічні складові загальної системи, використання методів розробки архітектури зверху-вниз та знизу-вверх;

– незалежність реалізації від мови програмування і можливість використання різних мов програмування в різних складових частинах SOA, забезпечення сумісності на рівні запитів REST або WSDL;

– сумісність з об'єктно орієнтованою моделлю та об'єктно орієнтованими мовами програмування, як з парадигмами контролю потоків команд, так і з парадигмами потоків даних;

– можливість агрегації потоків даних, де одне з'єднання між компонентами системи відповідає певній множині даних, що передаються;

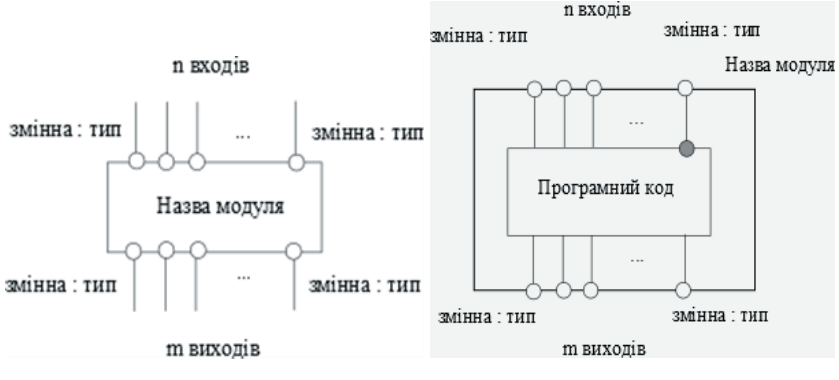
– швидке виявлення несумісності входів та виходів системи на рівні графічного редактору (аналог з'єднання «пазлів»);

– можливість унікальної ідентифікації компонентів SOA для повторного використання;

– можливість застосування коду програм в хмарних системах типу IAAS та PAAS.

3. Представлення основних складових діаграм з керуванням потоками даних в SOA

Пропонується проста та загальна система зображення компонентів системи керування потоками даних, в яких вхідна інформація зображується зверху прямокутника, а вихідна інформація зображується внизу прямокутника (рис. 1, *а*, *б*).



а *б*

Рис. 1. Представлення діаграми керування потоками даних в SOA:
а — загальна діаграма використання модуля керування потоками даних;
б — розгорнута схема модуля керування потоками даних

Така нотація дозволяє легко читати схему потоків даних (рис. 2).

Використання логічних змінних як вхідних параметрів компоненти зображено на рис. 3, *а, б.*

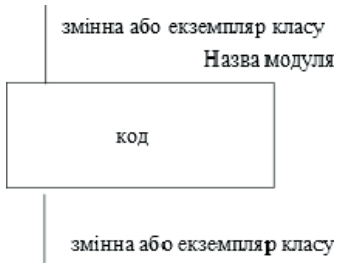
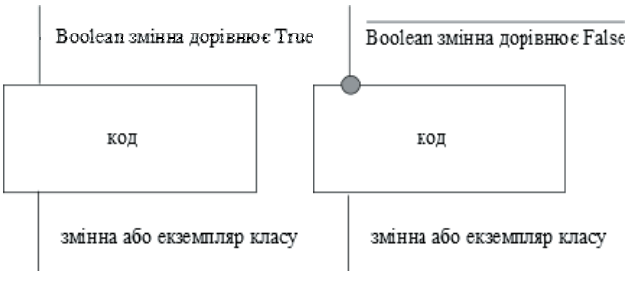


Рис. 2. Загальний вигляд компоненти діаграми керування потоками даних в SOA



а *б*

Рис. 3. Використання логічних змінних як вхідних параметрів компоненти: *а* — код виконується тільки тоді, коли на вході компоненти змінна дорівнює True; *б* — код виконується тільки тоді, коли на вході компоненти змінна дорівнює False

Код компоненти обробки даних виконується лише при наявності всіх вхідних даних. Вихідні дані компоненти генеруються одночасно.

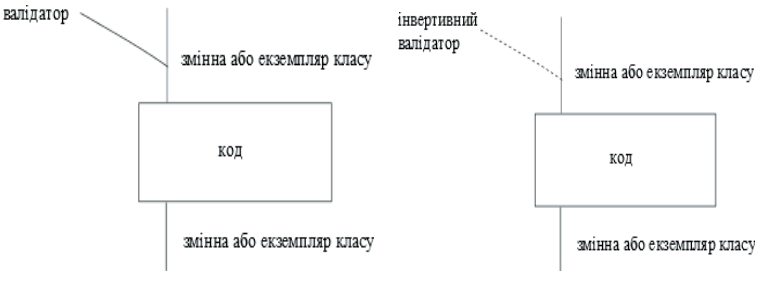
Процес обробки даних компонентою може почати обробляти наступні дані тільки після того, як виконано обробку теперішніх даних (без конвеєра).

4. Валідація вхідних даних

Для полегшення керування потоками даних та зменшенню кількості компонентів пропонується можливість задавати валідатори даних, тобто код програми, який буде перевіряти вхідні дані на відповідність певним умовам (рис. 4, *а*).

Якщо умова перевірки даних не виконується, відповідно код компоненти за даних обставин виконуватися не буде (рис. 4, *б*).

Механізм валідаторів та інвертних валідаторів дозволяє гнучко керувати потоками даних, спрощувати загальний вигляд діаграми, однозначно інтерпретувати логіку потоків даних, просто зображати діаграми потоків даних як за допомогою графічних програм, так і на ескізах.



а *б*

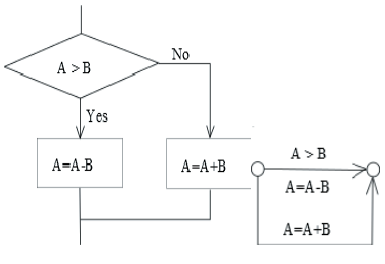
Рис. 4. Валідація вхідних даних в компонентах; *а* — позитивний валідатор, якщо умова виконується, дані потрапляють на вхід компонента; *б* — інвертний валідатор, дані потрапляють на вхід компонента, якщо умова валідатора не виконується

5. Область видимості даних

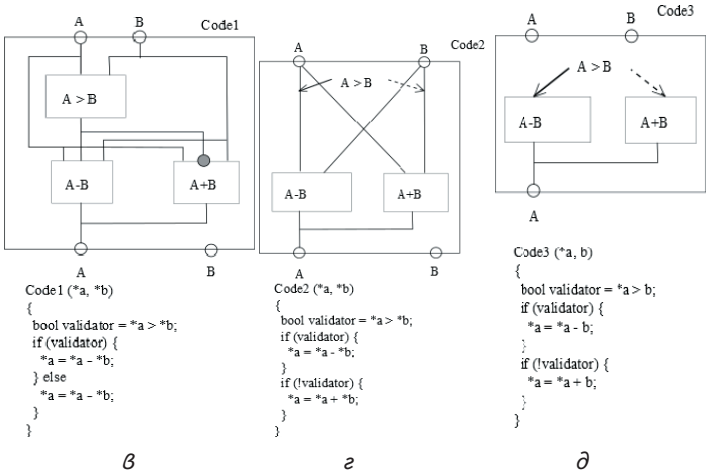
У блок-схемах (рис. 5, *а*) та R-схемах (рис. 5, *б*) раніше було запропоновано візуальних підхід для опису алгоритмів за допомогою потоку команд.

З метою полегшення процесу читання діаграми потоку даних компонентів SOA пропонується використовувати область видимості на рівні опису компонента.

Всі вхідні параметри батьківського класу мають бути доступними на вході компонентів нащадків, тільки на 1 рівень нижче.



а *б*



в *г* *д*

Рис. 5. Приклад використання області видимості даних: *а* — блок-схема потоку команд для простого алгоритму; *б* — відповідна R-схема; *в* — приклад компоненти діаграми керування потоками та відповідний код програми; *г* — код програми з позитивним та інверсним валідатором; *д* — спрощене представлення компонентів з урахуванням області видимості вхідних змінних

Такий метод дає можливість уникнути зображення зайвих з'єднань між вхідними даними батьківської компоненти, що значно спрощує структуру діаграми та підвищує її наочність (наприклад рис. 5, *в*, *г*). Суттєве спрощення представлення компонентів з урахуванням області видимості вхідних змінних можна спостерігати при використанні валідаторів та області видимості даних (рис. 5, *д*).

Таким чином, за допомогою визначення області видимості можливо значно зменшити кількість з'єднань та створювати прості, демонстративні і легко зрозумілі системи опису SOA в інформаційних системах.

6. Компіляція програмного коду в SOA з керуванням потоками даних

Процес компіляції програмного коду з діаграми керування потоками даних можливо чітко формалізувати та реалізувати в достатньо компактній формі. Архітектура системи компіляції коду зображена на рис. 6.

Для пояснення процесу компіляції коду з діаграм потоків даних розглянемо наступний приклад – алгоритм вирішення квадратного рівняння у вигляді компонента SOA (рис. 7).

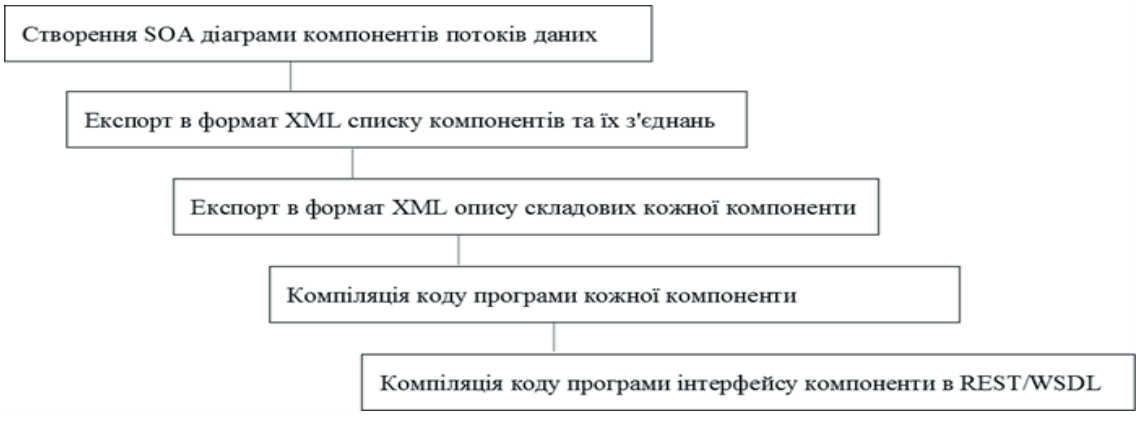


Рис. 6. Етапи компіляції діаграми керування потоками даних в програмний код з графічного опису компоненти SOA

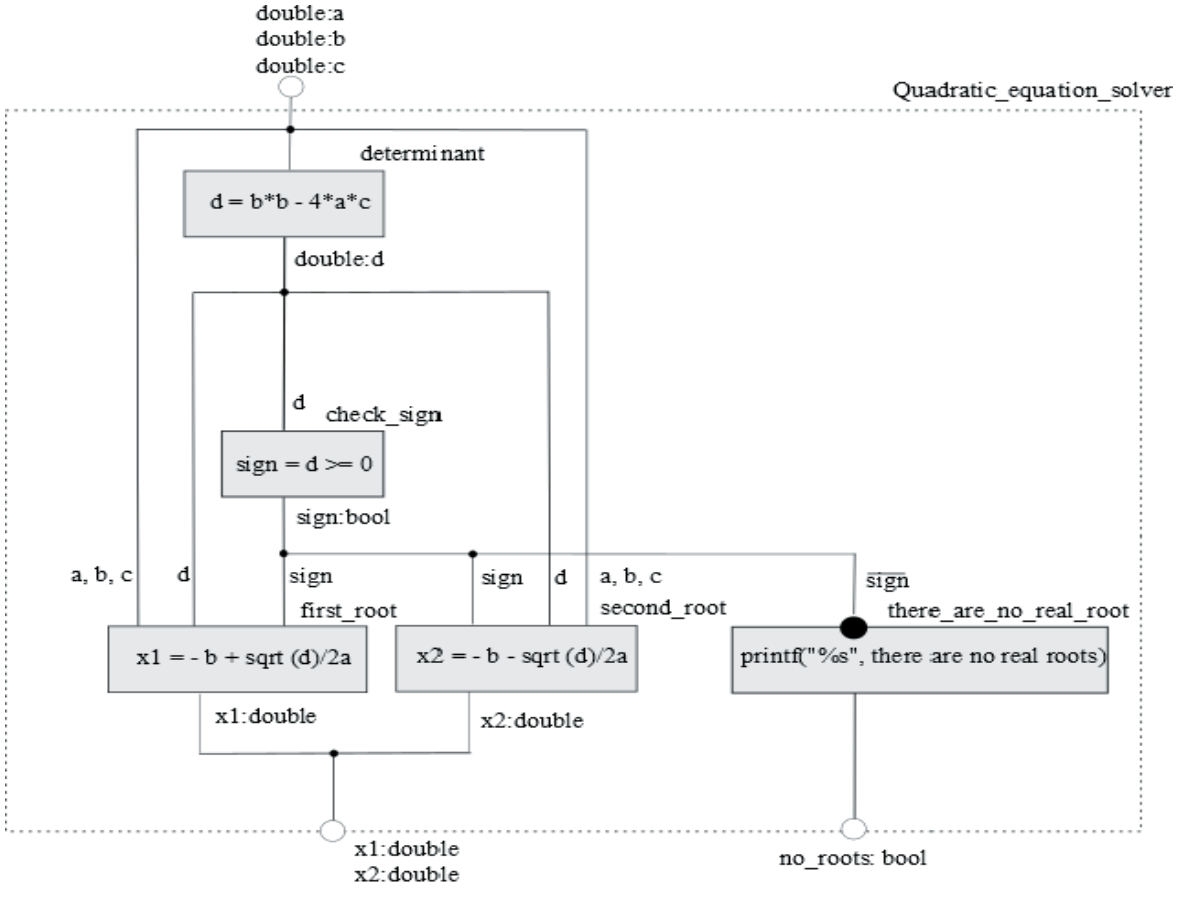


Рис. 7. Діаграма реалізації компоненти SOA для рішення квадратного рівняння

Як готовий компонент в інших діаграмах потоків даних можливо використовувати зображення компонента вирішення квадратного рівняння у вигляді прямокутника з відповідними входами та виходами (рис. 8).

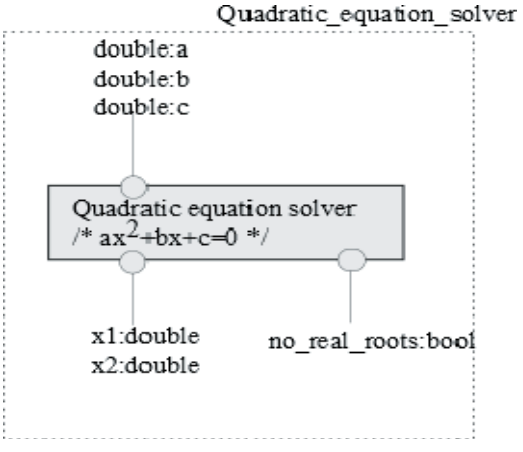


Рис. 8. Відповідне зображення діаграми використання компоненти SOA для рішення квадратного рівняння

Лістинг 1. Представлення реалізації компоненти для вирішення квадратного рівняння у вигляді XML.

```
<xml>
 <name>Quadratic equation solver</name>
 <input>
  <param>double:a</param>
  <param>double:b</param>
  <param>double:c</param>
</input>
<output>
 <param>double:x1</param>
 <param>double:x2</param>
 <param>bool:no_real_roots</param>
</output>
<nodelist>
 <node>double:d</node>
<node>bool:sign</node>
</nodelist>
 <componentlist>
 <component>
  <name>determinant</name>
  <intput>a, b, c<input>
  <output>d</output>
</component>
<component>
  <name>check_sign</name>
  <input>d</input>
  <output>sign</output>
 </component>
<component>
  <name>first_root</name>
  <intput>a, b, c<input>
  <input>d</input>
  <input>sign</input>
  <output>x1</output>
  </component>
<component>
  <name>second_root</name>
  <intput>a, b, c<input>
  <input>sign</input>
  <input>d</input>
  <output>x2</output>
 </component>
 <component>
  <name>there_are_no_roots</name>
  <intput>sign<input>
  <output>x1</output>
 </component>
 </componentlist>
</xml>
```

Лістинг 2. Приклад опису реалізації компоненту determinant у вигляді XML.

```
<xml>
 <name>determinant</name>
  <input>
   <param>double:a</param>
   <param>double:b</param>
   <param>double:c</param>
 </input>
 <output>
  <param>double:d</param>
 </output>
 <code>
  d = b* b - 4 * a * c;
 </code>
</xml>
```

Лістинг 3. Приклад реалізації компоненти рішення квадратного рівняння у вигляді коду на мові програмування C.

```
Quadratic_equation_solver (double a, double b, double c, double &x1, double &x2, boolean &no_roots)
{
 double d;
 determinant (a, b, c, double &d);
 bool sign;
 check_sign (d, &sign);
 first_root (a, b, c, *d, sign, x1);
 second_root (a, b, c, *d, sign, x2);
 there_are_no_real_roots (!sign, no_results);
}

determinant (double a, double b, double c, double &d)
{
 *d = b * b - 4*a*c;
}

check_sign (double d, boolean &sign)
{
 sign = d >= 0 ? True : False;
}

first_root (double a, double b, double c, double d, boolean sign, double *x1)
{
 if (sign)
  *x1 = (-b + sqrt (d)) / (2 * a);
 else
  *x1 = null;
}

second_root (double a, double b, double c, double d, boolean sign, double *x2)
{
```

```
 if (sign)
   *x2 = (-b - sqrt (d)) / (2 * a);
 else
   *x2 = null;
}

there_are_no_real_roots (boolean sign, boolean &no_
roots)
{
 printf ("%s", "there are no real roots");
 *no_roots = ! sign;
}
```

Лістинг 4. Приклад опису REST запиту для компоненти рішення квадратного рівняння.

```
http://localhost/Quadratic_equation_solver

HTTP POST request with JSON:
Content-Type: application/json
Accept: application/json
[ {
   "a": 4.0,
   "b": 5.0,
   "c": 1.0
} ]

Server Reply with JSON:
Content-Type: application/json
[ {
   "x1": -0.25,
   "x2": -1.0,
   "no+real_roots": false
} ]
```

7. Представлення циклів обробки даних компонентів та паралельне виконання коду

В парадигмі програмування керування потоками даних в цілому застосування циклів не заохочується [13, 14].

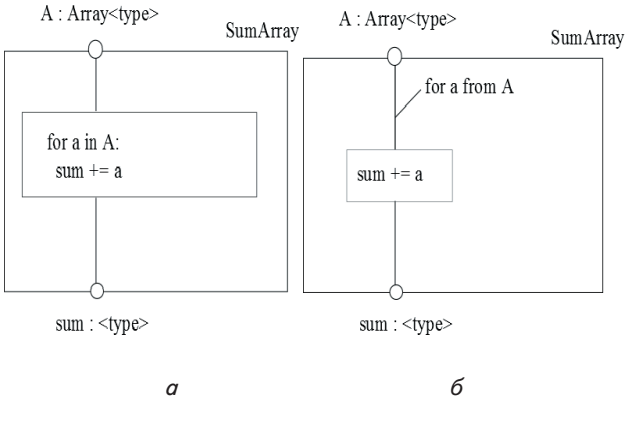


Рис 9. Представлення циклів: *а* — метод представлення циклу обробки даних в коді компонента *б* — метод представлення циклу за допомогою валідатора

В постановці задачі використання сервіс-орієнтованої архітектури в запиті виконання певної компоненти SOA має використовуватися принцип REST про відсутність збереження даних у клієнтській частині. Таким чином, використання циклів буде призводити до збільшення кількості запитів до компоненти. Але в окремих випадках існує необхідність представлення коду компоненти у вигляді циклів, тому для цього пропонується використання нотації циклів на відповідній мові програмування (рис. 9, *а*), або використання валідатора (рис. 9, *б*) з позначенням циклу, що буде доданий перед кодом компоненти. Відповідний код на мові Python представлений на Лістингу 5.

Лістинг 5. Представлення циклів, відповідний код програми на мові Python

```
def SumArray(A):
  sum = ""
   for a in A:
  sum += a
  return sum
```

Аналогічно можна зображувати потоки даних, в процесі обробки яких виникає необхідність використовувати рекурсію. Наприклад, у алгоритмі обчислення факторіалу можна викликати рекурсивну компоненту (рис. 10).

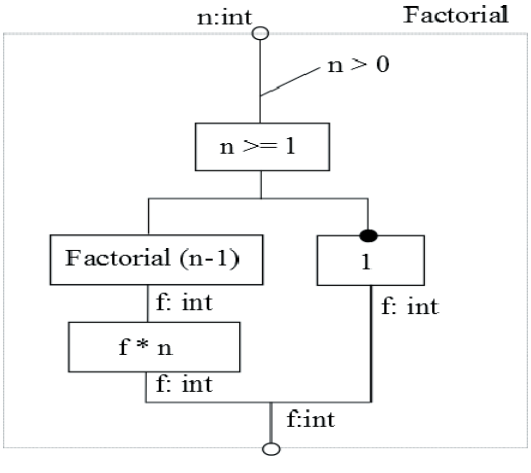


Рис. 10. Використання рекурсії для обробки потоку даних

За своєю природою системи з керуванням потоками даних легко та інтуїтивно дозволяють виділити компоненти, які можна виконувати паралельно [15]. Це дозволяє генерувати код програми, в яких виконання паралельних модулів відбувається у потоках в рамках одного компоненту, або системи передачі повідомлень між компонентами SOA (рис. 11).

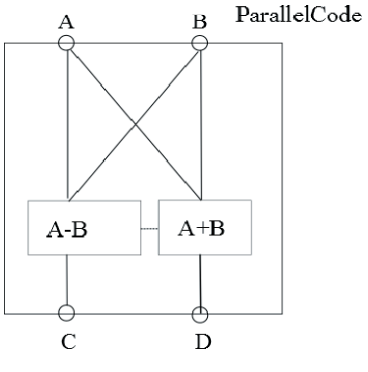


Рис. 11. Ілюстрація паралельного виконання коду компонентів в системі керування потоками даних

Для зображення паралельного виконання коду пропонується з'єднання паралельних модулів на діаграмі за допомогою лінії з ліва або з права від компоненту.

8. Тестування систем з керуванням потоками даних

Пропонується використовувати механізм сповіщення про нештатні ситуації в роботі компонентів за допомогою системи Assertion, аналогічно як це відбувається в сучасних мовах програмування Java, C# та ін. (рис. 12).

Поєднання системи модульних (unit) тестів та системи керування потоками даних можливо на рівні Integrated Development Environment (IDE), коли за допомогою інтерфейсу користувача розробник може натиснути на зображенні компоненти і перейти до іншого рівня ієрархії, на якому буде зображено певну кількість модульних тестів, також у вигляді діаграми керування потоками (рис. 13).

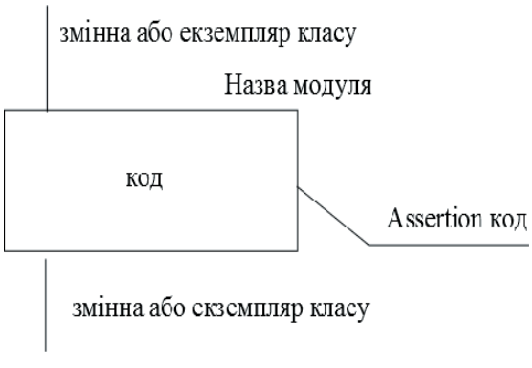


Рис. 12. Використання механізму сповіщення про виникнення нештатної ситуації в роботі компоненти за допомогою Accertion

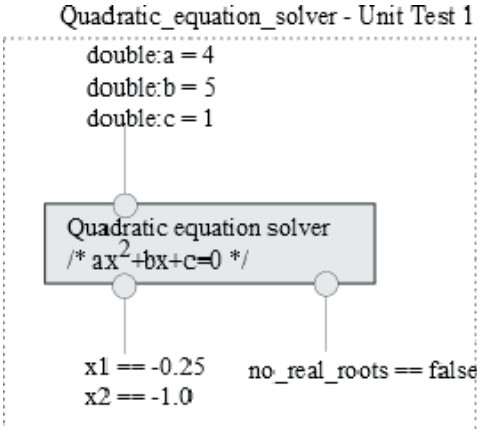


Рис. 13. Система модульного тестування для компонентів SOA

Практичне застосування системи модульних тестів з використанням графічного представлення потоків даних, разом з інтеграцією в IDE та ієрархічним представленням системи тестів дозволить ефективно та наочно проводити пошук помилок та несправностей в SOA інформаційних системах.

9. Висновки

Переваги використання запропонованого методу зображення діаграм потоків даних для розробки інформаційних систем, в тому числі з використанням SOA, полягають у:

– поєднанні процесу створення архітектури інформаційної системи та процесу її реалізації у вигляді програмного коду окремих компонентів;

– ієрархічній побудові архітектури інформаційних систем, можливості вкладеного опису діаграми з керуванням потоками даних;

– можливості використання процесу розробки, що керується тестами (Test Driven Development);

– простому графічному зображенні потоків даних, яке дозволяє відображати інтуїтивно зрозумілу логіку роботи системи, можливість простого відображення у векторних графічних редакторах загального призначення (Visio, Dia, Inkscape і т.д.);

– швидкості зображення діаграм потоків даних на ескізах за допомогою олівця без спеціальних навиків креслення;

– можливості використанні високопотужних систем і парадигм обчислювань, таких як map-reduce, за допомогою паралельного виконання компонентів.

Розроблено метод зображення діаграм керування потоками даних SOA, досліджено його застосування для відображення рекурсії, циклів, складних алгоритмів керування потоками даних. Розглянуто метод компіляції діаграм керування потоками даних в об'єктно-орієнтовані мови програмування, розроблені формати XML для опису діаграм керування потоками даних. Запропоновано метод відображення на діаграмі валідації даних та метод відображення модульних тестів у системах з керуванням потоками даних.

Запропонований метод зображення діаграм потоків даних можливо використовувати в основних галузях: інформаційні системи корпоративного рівня з використанням сервіс-орієнтованої архітектури, системи цифрової обробки сигналів, системи обробки зображень, системи керування, аналіз великих даних, робототехніка.

Література

1. Morrison, J. P. Flow Based Programming [Electronic resource] / J. P. Morrison. – Available at: http://www.jpaulmorrison.com/fbp/#DrawFBP.
2. Графическая среда разработки LabVIEW [Электронный ресурс] / Режим доступа: http://russia.ni.com/labview.
3. Dataflow Diagram [Electronic resource] / Available at: http://en.wikipedia.org/wiki/Data_flow_diagram.
4. Yordon, E. Dataflow Diagrams [Electronic resource] / E. Yordon. – Available at: http://www.yourdon.com/strucanalysis/wiki/index.php?title=Chapter_9.
5. Stevens, W. Structured Design [Text] / W. Stevens, G. Myers, L. Constantine // IBM Systems Journal. – 1974. – № 13 (2). – P. 115–139.
6. Вельбицкий, И. В. Графический стиль программирования для персональной ЭВМ [Текст] / И. В. Вельбицкий, А. Л. Ковалёва // Журнал Микропроцессорные средства и системы. – 1985. – № 4. – С. 46–51.

7. Вельбицкий, И. В. Графическое программирование и доказательство правильности программ [Электронный ресурс] / И. В. Вельбицкий. – Режим доступа: http://glushkov.org/wp-content/uploads/131120-csitd180d183d181-9c2.pdf.
8. Дружелюбный русский алгоритмический язык, который обеспечивает наглядность (ДРАКОН) [Электронный ресурс] / Режим доступа: http://ru.wikipedia.org/wiki/D0%94%D0%A0%D0%90%D0%9A%D0%9E%D0%9D.
9. SOA Design Patterns [Electronic resource] / Available at: http://soapatterns.org/design_patterns/overview.
10. Bellomo, S. Suggestions for Documenting SOA-Based Systems. Technical Report [Text] / S. Belomo. – Hanscome, MA: Carnegie Mellon University, 2010. – 42 p.
11. Reference Architecture Foundation for Service Oriented Architecture [Electronic resource] / Available at: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.html.
12. NoFloJS [Electronic resource] / Avaialble at: http://noflojs.org/documentation/.
13. Parks, T. M. A Comparison of Synchronous and Cyclo-Static Dataflow [Text] : proc. of Asilomar conf./ T. M. Parks, J. L. Pino, E. A. Lee // Signals, Systems and Computers. – 1995. – Vol. 1. – P. 204–210.
14. Blume, P. A. The LabVIEW Style Book [Text] / P.A. Blume. – Prentice Hall, 2007. – 400 p.
15. Falgout, Jim Dataflow Programming: A Scalable Data-Centric Approach to Parallelism [Electronic resource] / Available at: http://soa.sys-con.com/node/1678918.

В статті розглянуто похибки вимірювання температури біологічного палива, що здійснюється на основі використання термоелектричного перетворювача у термоанемометричному витратомірі (ТАВ). Представлено основні вимоги до витратомірів біологічного палива, що впливають на підвищення точності вимірювання та швидкодію ТАВ. Розглянуто складові частини похибки з урахуванням робочих умов використання витратоміра. Виконано експериментальні дослідження похибок витратоміра біологічного палива

Ключові слова: біологічне паливо, термоанемометр, тахометр, електроперетворювачі, термоперетворювач, дифманометр, трубопровід, лічильник, датчик

В статье рассмотрены погрешности измерения температуры биологического топлива осуществляется на основе использования термоэлектрического преобразователя в термоанемометрические расходомере (ТАВ). Представлены основные требования к расходомеров биологического топлива, влияющие на повышение точности измерения и быстродействие ТАВ. Рассмотрены составные части погрешности с учетом рабочих условий использования расходомера. Выполнены экспериментальные исследования погрешностей расходомера биологического топлива

Ключевые слова: биологическое топливо, термоанемометр, дифманометр, электропреобразователи, термопреобразователь, манометр, трубопровод, счетчик, датчик

УДК 531.383

ПОХИБКИ ВИМІРЮВАННЯ ТЕМПЕРАТУРИ БІОЛОГІЧНОГО ПАЛИВА У ТЕРМОАНЕМОМЕТРИЧНОМУ ВИТРАТОМІРІ

Ю. О. Шавурський
Кандидат технічних наук, доцент
Кафедра автоматизованого управління технологічними процесами та комп'ютерних технологій
Житомирський державний технологічний університет
вул. Черняховського 103, м. Житомир, Україна, 10005
E-mail: shavursky@gmail.com

1. Вступ

Активне заміщення біопаливом традиційних нафтопродуктів спостерігається в багатьох країнах світу. В ЄС у 2008 р. працювало більше 40 потужних заводів, які виробили майже 2 млн т біодизеля, зокрема 1000 т. у Німеччині, 350 т. – у Франції, 320 т. – в Італії, 160 т. – у Данії, 60 тис т. – у Чехії. У 2010 р. заплановано виробити близько 6 млн т. біодизельного палива. При цьому потреба ЄС у ріпаковому насінні становить 12 млн т, із них левову частину за сприятливих умов може поставити Дванадцять мільйонів тонн ріпаку – потреба Європи, левову частину з яких Україна разом з Росією за сприятливих умов цілком здатні забезпечити.