технологий, Институт специальной связи и защиты информации Национального технического университета Украины “Киевский политехнический институт имени Игоря Сикорского”, Киев, Украина.

Dmyto Sharadkin, candidate of technical sciences, associate professor, associate professor at the cybersecurity and application of automated information systems and technologies academic department, Institute of special communication and information protection National technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Kyiv, Ukraine.

УДК 004(94+41)

ВОЛОДИМИР СОКОЛОВ

ЗАСТОСУВАННЯ ФУНКЦІОНАЛЬНОЇ ТА РЕЛЯЦІЙНОЇ МОДЕЛЕЙ В ОБ’ЄКТНО-ОРІЄНТОВАНОМУ ПРОГРАМУВАННІ

В роботі представлено результати досліджень та практичної апробації формальних методів опису об’єктно-орієнтованих програм, придатних для автоматичної генерації тексту програм на мові програмування. В якості формальних моделей обрано функціональну та реляційну моделі. Функціональна модель представляє програму як схему сполуки функціональних атомарних об’єктів, здатних до безпосередньої взаємодії шляхом утворення динамічних сполук та автоматичних обчислень, яка може представлятися у графічній формі. При цьому схема з’єднання об’єктів розглядається як схема розв’язку задачі. Показано, що формалізація саме схеми розв’язку задачі, а не всієї парадигми програмування, робить процес створення програми більш наближеним до практики. Визначені вимоги до атомарних об’єктів як таких, що складають елементну базу об’єктно-орієнтованої програми. Реляційна модель представляє об’єкт як віртуальне відношення, схема якого задається класом, який реалізує функціональну залежність неключових атрибутів від ключових шляхом їх обчислень, що дозволяє застосовувати реляційні операції для опису схеми розв’язку задачі. Реляційна модель дозволяє використовувати мову, подібну до структурованої мови запитів до баз даних, для опису схеми розв’язку задачі та її автоматичного виконання. Показано, що функціональна та реляційна моделі придатні для графічного представлення схеми розв’язку задачі і є достатньовиразними для безпосередньої генерації програм. Фактично, розроблені моделі дозволяють підняти процес створення об’єктно-орієнтованих програм на рівень вище, зосередитись на структурі програми, а не на її складових, і доповнити прогалину в існуючих методах представлення програм. В якості основи для практичної реалізації використана технологія програмування активних динамічних сполук об’єктів.

Ключові слова: об’єктно-орієнтоване програмування, формалізація програм, реляційна модель, функціональна модель, активні динамічні сполуки об’єктів.

Постановка проблеми. Об’єктно-орієнтоване програмування (ООП), незважаючи на його широке використання, має низку проблем, найсуттєвішими з яких є: наявність “крихкого” базового класу в успадкуванні, відсутність формальної теорії опису об’єктно-орієнтованих програм (ООПр) для їх верифікації, перевірки коректності, доведення вірності й оптимізації, а також відсутність засобів високо рівня побудови ООПр з об’єктів та уніфікованих механізмів їх взаємодії. Проблеми успадкування в першу чергу пов’язані з тим, що успадкований базовий клас потрапляє в простір імен похідного класу, що часто призводить до збігу імен, наявність в структурі пам’яті об’єкту похідного класу всього об’єкту базового класу та непередбачуваність впливу змін в базовому класі на всі класи-нащадки.

Відсутність формальної теорії опису ООП разом зі свободою, що воно надає програмісту, не дає можливості досліджувати ООПр формальними методами. Використання універсальної мови моделювання (Unified Modeling Language, UML) для аналізу та проектування ООПр є достатньо ефективним, але UML має низьку ефективність в питаннях програмування, тому що не містить жодної діаграми, яка б цілком визначала структуру програми в термінах об'єктів і була б придатною для автоматичної генерації програм. Діаграма діяльності, яка претендує на цю роль, не є здобутком UML і відома ще до появи ООП як блок-схема алгоритму, а діаграма класів визначає тільки структури класів та їх взаємовідношення, хоча врешті решт програма будується з об'єктів, а не з класів.

Засоби взаємодії об'єктів в ООП не визначені. Інколи взаємодію розглядають як обмін повідомленнями між об'єктами, хоча на практиці це звичайне використання даних та виклики функцій об'єктів. Саме відсутність уніфікованих засобів прямої взаємодії об'єктів не дозволяє розробляти ООПр виключно з об'єктів, не опускаючись на рівень даних та функцій.

Тому вирішення проблем успадкування, формалізації, взаємодіїта генерації програм в ООП дозволить значно підвищити ефективність створення програмного забезпечення.

Аналіз останніх досліджень і публікацій. Роботи, що присвячені або можуть використовуватись для формалізації ООП, можна розділити за наступними напрямами:

1. Формалізація об'єктно-орієнтованої парадигми цілком. Так в роботах [1] - [4] надано формальне представлення класу, об'єкту, інкапсуляції, успадкування та поліморфізму через автомат Мура безвідносно до конкретної ООПр.
2. Формальна специфікація програм [5], [6], яка не є спеціалізованою для ООП, а більше підходить для імперативної парадигми. Існуючі мови формальної специфікації програм (VDM, RAISE, Spec#) маніпулюють типами даних і функціями.
3. Гібридні парадигми, що використовують об'єктно-реляційне відображення в об'єктно-орієнтованих базах даних (БД) та об'єктно-функціональний підхід. Наприклад, в композиційному програмуванні [7] вся програма розглядається як функція, яка є композицією інших функцій, а стосовно ООП розглядає процес створення програми як композицію існуючих об'єктів в структурі нового об'єкта. Ці підходи дещо близькі до описаних в даній роботі на рівні ідей.
4. Засоби графічного (візуального) моделювання такі як UML, засоби проектування схем БД, застосування Р-схем для опису алгоритмів та класів та інші, існування яких доводить необхідність схематичного підходу до програмування, в тому числі й до ООП. Взагалі всі способи представлення програм можна поділити на математичні, графічні та текстові (мова програмування).

У зазначених напрямах робіт не розглядаються конкретні ООПр як предмет формалізації в термінах ООП, а мають більше теоретичне застосування, ніж практичне. Загалом, причини відсутності формалізації ООП, придатної для практичного застосування, полягають в тому, що:

1. Поняття ООП не є чітко визначеним, а відоме визначення Граді Буча штучно пов'язує об'єктну орієнтованість з ієрархічним успадкуванням класів.
2. Не визначені уніфіковані засоби взаємодії об'єктів.
3. Відсутні рекомендації щодо побудови об'єктів(класів), які б упорядкували процес програмування (як в свій час технологія модульного структурного програмування).

Фактично, в об'єктно-орієнтованій парадигмі відсутня технологія програмування.

Таким чином, за результатами аналізу останніх досліджень і публікацій перспективними напрямками досліджень є використання формальних (функціональних та реляційних) моделей та графічних способів опису ООПр, придатних для автоматичної генерації тексту програми на мові програмування, а також розробка рекомендацій щодо структури об'єктів та засобів їх взаємодії.

Метою статті є формальне представлення ООПр як схеми розв'язку задачі на основі функціональної та реляційної моделей, придатне для автоматичної генерації програми на мові програмування. Для досягнення мети потрібно формалізувати клас, об'єкт, взаємодію

об'єктів, та схему розв'язання задачі для практичного застосування на основі технології програмування активних динамічних сполук об'єктів (АДС-технологія) [8].

Основна ідея досліджень. З одного боку, об'єкт можна розглядати як функцію, що реалізує відображення вхідних даних у вихідні результати функціями класу, а з іншого – як відношення між вхідними даними та результатами. Так як в основі ООП лежить об'єктна модель даних, то вочевидь існують еквівалентні методи її представлення в інших моделях, що і буде продемонстровано у викладенні подальшого матеріалу.

Не претендуючи на універсальність підходу, будемо розглядати тільки обчислювальні задачі, в яких потрібно по заданих вхідних даних обчислити результат. Відповідно до шаблону проектування MVC (Model – View – Controller) зосередимось на моделі, а питання інтерфейсу та контролера розглянемо пізніше.

Функціональна модель ООПр. Будемо розглядати клас як функцію або як функціональну залежність (ФЗ), що відображає значення вхідних даних у значення вихідних результатів, а об'єкт – як екземпляр функції, що для обчислювальних задач цілком прийнятний підхід.

Приймемо, що елементною базою ООПр є атомарні об'єкти або просто *атоми*, з яких будується програма. Об'єкт є атомом (і відповідно його клас є *атомарним класом*), якщо він відповідає наступним вимогам;

- вхідні дані та вихідні результати є атомарними (простими типами даних);
- має нуль або більше входів і один вихід;
- реалізує одну функцію;
- вихід функціонально повно залежність від входів;
- атом є АДС-об'єктом (таким, що здатний утворювати активні динамічні сполуки з іншими атомами).

Атомарність вхідних даних та вихідних результатів за рахунок звуження множини типів даних забезпечує можливість взаємодії об'єктів будь-яких класів, які містять входи-виходи цих типів. Об'єкт, що не має входів, може бути константою або генератором. Один вихід забезпечує не надлишковість обчислень, а повна залежність виходу від входів гарантує не надлишковість вхідних даних. Фактично атом є ідеальним об'єктом, що не підлягає удосконаленню.

Утворення сполуки з атомів полягає в тому, щоби входи одних атомів зв'язувати з виходами інших. Будь-яка обчислювальна задача може бути представлена сукупністю сполук атомів.

Так як програма призначена для розв'язку певної задачі, а точніше програма реалізує певну схему розв'язку задачі, то доцільно спочатку формалізувати саме схему розв'язку задачі.

Представлення схеми розв'язку задачі. Схема *S* розв'язку обчислювальної задачі *Z* вважається заданою, якщо визначено:

1. Множину атомарних класів *Cls*;
2. Множину атомарних об'єктів *Obj*, кожний з яких є екземпляром класу з *Cls*;
3. Схему сполуки атомів *Con*, яка реалізує перетворення вхідних значень у вихідні в потрібному порядку;
4. Множина вхідних даних *Inp*, що є входами атомів;
5. Множина вихідних результатів *Out*, що є шуканими виходами атомів;

$$S = \langle Cls, Obj, Con, Inp, Out \rangle, \tag{1}$$

де *Cls* представляє множину атомарних класів, кожний з яких має назву *ClName* і реалізує відображення входів *x* у виходи *y:*

$$Cls = \{ClName : x \to y\};$$

Obj задає множину об'єктів класів

$$Obj = \{ClName :: ObjName\};$$

Con задає схему з'єднання атомів в сполуку

$$Con = \{ObjName.x = ObjName.y\};$$

Inp – це підмножина імен вільних входів атомів

$$Inp = \{ObjName.x\};$$

Out – це підмножина імен виходів атомів

$$Out = \{ObjName.y\}.$$

Фактично, вся задача реалізує відображення $Z : Inp \xrightarrow{Con} Out$.

Приклад. Потрібно представити схему розв'язання задачі за формулою $Y = (x_1 + x_2)^2 + x_2$.

Елементарні операції замість функцій в цій задачі обрано виключно для демонстрації. Функціональний вираз, що відповідає даній задачі, виглядає так:

$$Y = Plus(Mult(Plus(X1, X2), Plus(X1, X2)), X2). \quad (2)$$

З огляду на (1) та (2) отримаємо таку схему розв'язання задачі:

1. *Cls* = {*Plus*:*A*,*B*→*C*, *Mult*:*A*,*B*→*C*, *Data*:*a*→*A*}.
2. *Obj* = {*Data::X2, Plus::P1, Plus::P2, Mult::M1*}.
3. *Con* = {*P1.B=X2.A, M1.A=P1.C, M1.B=P1.C, P2.A=M1.C, P2.B=X2.A*}.
4. *Inp = {P1.A , X2.a}.*
5. *Out={P2.C}.*

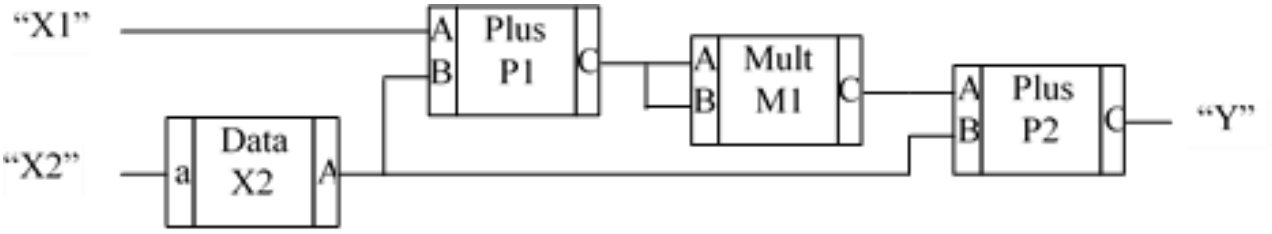

Рисунок 1 – Графічне представлення схеми розв'язання задачі

Об'єкт даних *Data::X2* включено до схеми з метою одноразового введення даних, так як в формулі задачі він використовується двічі. Синтезована сполука може бути перетворена в ООПр

```
#include<Connectors> // реалізує АДС-технологію
#include<Data>       // включення тексту класу Data
#include<Plus>       // включення тексту класу Plus
#include <Mult>      // включення тексту класу Mult
#include<iostream>   // стандартні потоки введення/виведення
int main()
{
 floatfloat_val;     // робоча змінна
  // створення об'єктів
 Data<float> *X2 = new Data<float>;
 Plus<float> *P1  = new Plus<float>;
 Plus<float> *P2  = new Plus<float>;
Mult<float> *M1 = new Mult<float>;
// утворення сполуки
Connection(&P1->B, X2->A.getadr());
Connection(&M1->A, P1->C.getadr());
Connection(&P2->A, M1->C.getadr());
Connection(&P2->B, X2->A.getadr());
```

```
 // інтерфейс користувача – введення даних
cout<<"Enter X1="; cin>>float_val; P1->A.set_value(float_val);
cout<<"Enter X2="; cin>>float_val; X2->a.set_value(float_val);
if(P2->C.get_value()==NULL)  cout<<"Error";
else
cout<<"Y="<<*P2->C.get_value();      // виведення результату
delete X2;delete P1;delete P2;delete M1;// знищення об'єктів
return 0;
}
```

Обчислення сполуки в АДС-технології після задання значень входів здійснюється шляхом звертання до всіх виходів в довільному порядку (в наведеному прикладі є тільки один вихід–*P*2.*C*) і відбувається:

– вихідний конектор *P*2.*C* викликає функцію ядра об'єкта *P*2, який в свою чергу спочатку звертається до входу *A*, який з'єднаний з виходом *M*1.*C*;

– об'єкт *M*1 робить аналогічно, але спочатку звертається до входу *A*, що викликає обчислення об'єкту *P*1, коли він отримує власне значення входу *A* та вихід *X*2;

– при обчисленні входу *M*1.*B* об'єкт *P*1 не буде повторно обчислений завдяки перевірці того, що значення входів не змінились і він має стан обчисленого;

– *P*2.*A* отримує значення, потім отримує значення входу *B* як вихід *X*2.*A* і поверне результат *P*2.*C* як результат розв'язання задачі.

Питання практичної реалізації атомарних об'єктів. Кожний атомарний клас містить функцію ядра, стан та вхідні і вихідні конектори, які в свою чергу є шаблонами класів.

Вхідний конектор містить покажчик на вихідний конектор відповідного типу та власне значення, а також посилання на стан об'єкту. Вхідні дані об'єкт може отримувати або через покажчик, або шляхом присвоєння власного значення. В будь-якому випадку зміна власного значення або покажчика переводить об'єкт у невизначений (не обчислений) стан. При використанні покажчика власне значення використовується для кешування попереднього значення входу і запобігає зайві обчислення при незмінних вхідних даних.

Вихідний конектор містить покажчик на стан і на функцію ядра. Може повертати свою адресу для утворення сполуки. При звертанні до значення викликає функцію ядра та при обчисленому стані об'єкта повертає адресу результату або *NULL*.

Вхідні та вихідні конектори не залежать від класу, а залежать тільки від типу конектора, до якого підключаються. Це надає можливість сполучати об'єкти будь-яких класів.

Функція ядра має однакову назву у всіх класах та відповідає за обчислення об'єкту, для чого звертається до значень входів, перевіряючи чи змінились входи і при задовільній області допустимих значень (ОДЗ) обчислює результат або переводить об'єкт в не обчислений стан.

За рахунок такої реалізації сполуки завжди автоматично обчислюються в порядку з'єднання об'єктів схеми з використанням відкладених обчислень та попередньо обчислених значень. Обчислення схеми не зупиняється, якщо вхідні дані хоча б одного об'єкта не відповідають ОДЗ, а повертає "пусте" значення *NULL* на відповідних виходах.

Приклад реалізації зовнішнього відносно класів шаблону функції з'єднання:

```
template<classT> // T визначає атомарний тип конекторів, що з'єднуються
int Connection(TInConnector<T> *in, TOutConnector<T> *out)
{ if (in->connect(out)!=1) return 0;   return(1); }
```

Приклад реалізації атомарного класу *Plus*:

```
template<class T>
class Plus
{protected:
 int state; // стан об'єкту
 public:
TInConnector<T>A; // вхід
TInConnector<T>B; // вхід
```

TOutConnector<T>C; // вихід
*static int act(Plus *obj);*// функція ядра
Plus();// конструктор
};

Реалізацію схеми сполуки задачі зручно виконувати у вигляді *графічного редактора* схем сполук. Причому, структуру класу можна безпосередньо отримувати з тексту класу шляхом синтаксичного аналізу його визначення.

Правила побудови сполук полягає в тому, що входи одних атомів з'єднуються з виходами інших з однаковим типом даних без утворення зворотних зв'язків (ациклічний граф), причому декілька входів можуть з'єднуватись з одним виходом. З'єднання одного входу з декількома виходами недопустимо.

Інтерфейс та контролер. Так як сполука реалізує модель обчислень, зменшуючи до мінімуму обсяг допоміжного коду, то виникає питання інтерфейсу (користувача) та контролера (активатора).

Для консольних програм інтерфейс генерується тривіально. Для візуального інтерфейсу (наприклад, *Windows*), рішенням є надання елементам інтерфейсу (*Edit, CheckBox, Label* і т.п.) властивостей АДС шляхом створення з них нових похідних класів і додання для вхідних даних – вихідних конекторів, а для вихідних – вхідних. Тоді утворивши сполуки вхідних інтерфейсів з входами моделі, а вихідних – з виходами, отримаємо інтерфейс без надлишкового коду.

Роль контролера може виконувати будь-яка функція обробки потрібної події (наприклад, подія *OnClick* натискання кнопки *Button*), завданням якої є перебрати всі вихідні інтерфейси, що в свою чергу призведе до обчислення моделі і відображення результатів у відповідних елементах інтерфейсу.

Успадкування. Успадкування не є необхідністю. Альтернативою є композиція за посиланням або інтеграція. Успадкування атомів робить непрозорою його структуру і ускладнює синтаксичний аналіз. Тому доцільно при необхідності використовувати успадкування в атомарних класах тільки до рівня атома, а від успадкування атомів відмовитись. Це надасть програмі більшої надійності та зрозумілості, зменшить взаємний вплив класів.

З іншого боку, описаний підхід без успадкування може вийти за рамки визначення ООП, але шляхом створення абстрактного базового класу для всіх атомів, що міститиме стан об'єкту та чисту віртуальну функцію ядра, ця формальність може бути дотримана.

Реляційна модель ООПр. Так як функція є частковим випадком відношення, то можна розглядати ООПр з точки зору відношень:

– клас задає схему відношення *ClName*(*x*,*y*), де *ClName* – ім'я класу, *x* – імена ключових атрибутів, *y* – множина імен залежних атрибутів, та реалізує ФЗ $x \rightarrow y$;

– об'єкт класу *ClName*::*ObjName* – це екземпляр віртуального відношення з іменем *ObjName* та схемою *ClName*, що може містити не більше одного обчисленого кортежу.

Виходячи з цього, розглянемо реляційні операції для об'єктів:

– селекція об'єкту *ObjName* по значеннях ключових атрибутів *x* – це обчислення об'єкту з формуванням повного кортежу зі схемою (*x*,*y*) і, можливо, з *NULL*-значеннями атрибутів;

– проекція об'єкту *ObjName* на підмножину *y'* атрибутів *y* – це вибір виходів об'єкту (часткове обчислення) зі схемою результуючого кортежу (*x*, *y'*).

– з'єднання $ClName_i$::$ObjName_j(x_i,y_i)$ з $ClName_k$::$ObjName_m(x_k,y_k)$ за умови, що $x_k = y_i$ – задає ситуативне з'єднання по зовнішньому ключу.

– перейменування атрибутів має відповідний сенс.

Використовуючи реляційні операції, схему розв'язку задачі (2) можна виразити наступним чином:

USING / класи, що використовуються */*
Data, Plus, Mult
SELECT / виходи */*

P2.C as Y
FROM / об'єкти */*
Data<float>::X2, Plus< float >::P1, Plus< float >::P2, Mult< float >::M1
JOIN / схема сполуки */*
P1.B = X2.A, M1.A = P1.C, M1.B = P1.C, P2.A = M1.C, P2.B = X2.A
WHERE / входи */*
((P1.A as X1)=?) and ((X2.a as X2) =?)

Як видно, зв'язки між відношеннями (об'єктами) задаються тільки в операції з'єднання. Так як в роботі розглядаються тільки атомарні об'єкти, то фактично операція *SELECT* неявно задає виходи, а *WHERE* – входи, що фактично задає схему задачі в реляційній моделі (РМ). В операції з'єднання операндами можуть бути як об'єкти, так і таблиці БД.

Атомарні об'єкти з точки зору РМ знаходяться в третій нормальній формі, а наявність тільки одного виходу гарантує не надлишковість обчислень.

Теоретичним апаратом, що поєднує функціональний та реляційний підходи є аксіоми функціональних залежностей (ФЗ) Армстронга (та інші, що виведені на їх основі), які можна застосувати для побудови та аналізу сполук. Особливістю застосування аксіом ФЗ є те, що на множині класів не визначена семантика атрибутів (однойменні атрибути різних класів мають різну семантику на відміну від схем відношень), тому певні аксіоми можна застосовувати тільки для конкретної схеми сполуки:

1. Самовизначення $A \to A$ – представляє об'єкт даних (вихід дорівнює входу);
2. Транзитивність $(A \xrightarrow{f1} B) \wedge (B \xrightarrow{f2} C) \Rightarrow (A \xrightarrow{f1,f2} C)$ – реалізує послідовну композицію функцій *f*1, *f*2 за умови, що *f2.B = f1.B*;
3. Композиція $(A \to B) \wedge (C \to D) \Rightarrow (AC \to BD)$ – визначає спосіб агрегації атомів;
4. Декомпозиція $(A \to BC) \Rightarrow (A \to B) \wedge (A \to C)$ – забезпечує розклад ФЗ на атоми;
5. Об'єднання $(A \to B) \wedge (A \to C) \Rightarrow (A \to BC)$ – задає спосіб інтеграції атомів в сполуки.

Замикання множини атрибутів A^+ відносно сполуки зі схемою *S*–це множина вихідних результатів всіх об'єктів сполуки (включаючи проміжні), яка може бути обчислена по входах *A* для заданої схеми.

Замикання множини ФЗ F^+ відносно схеми *S* з входами *A* та виходами *B* – це множина композицій функцій (задач) для заданої схеми, яка використовується для обчислення результатів A^+. Задача *Z* з даними A_z та шуканими результатами B_z розв'язується даною сполукою *S*, якщо ФЗ *Z* є однією з композицій в F^+.

Верифікація коректності схеми *S* розв'язку задачі *Z* полягає в наступному:

1. Коректність, повнота та не надмірність вхідних даних: $A = A_z$.
2. Коректність та не надмірність вихідних результатів: $B = B_z$.
3. Досяжність результатів: $B \subseteq = A^+$ або $Z \in F^+$.
4. Ненадлишковість схеми: множина вільних виходів об'єктів $\subseteq B$.

Часткові обчислення. Так як перевірка ОДЗ покладається на функцію ядра класу (об'єкта), то можливо, що значення вхідних даних певного об'єкта не належать його ОДЗ або відсутні. Таку ситуацію можна розглядати як часткове обчислення сполуки. Так як реалізація атомів передбачає отримання покажчика на результат, який може бути *NULL*, то така ситуація є цілком прийнятною. У реляційній моделі наявність "пустого" значення означає те ж саме.

Висновки. Формалізована схема задачі на основі застосування функціонального та реляційного підходів, що дозволяє розглядати ООПр як композицію атомарних об'єктів та представляти її у вигляді формули, графічної схеми або не процедурної мови в термінах реляційних операцій з подальшою автоматичною генерацію тексту програми. Застосовано аксіоми ФЗ для аналізу, верифікації та формального доведення вірності програми. Сформульовано визначення атомарного об'єкта як елемента побудови ефективних ООПр на базі АДС-технології.

Перспективними напрямами досліджень розглянутого підходу є формалізація операцій інтеграції та дезінтеграція сполук, оптимізації їх структури; аналіз топології схем сполук, побудова схем зі зворотнім зв'язком та елементами управління; формалізація ізомерних об'єктів, формули сполуки та динамічний синтез сполук.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Д.Б. Буй, С.В. Компан, “Формализация объектов, классов, методов в объектно-ориентированных базах данных”, на *Международной конференции Проблемы теоретической кибернетики*, Нижний Новгород, 2011, с. 81-85.

[2] Д.Б. Буй, С.В. Компан, “Диаграммы классов ООП: формализация и анализ”, *Труды института прикладной математики и механики,* т. 27, Донецк, с. 51-65, 2013.

[3] А.Г. Пискунов “Формализация парадигмы объектно-ориентированного программирования”. [Электронный ресурс]. Доступно: http://www.realcoding.net/dn/docs/machine.pdf. Дата обращения: Янв. 19, 2016.

[4] D. Buy, S. Kompan, “The Concepts of Object, Class, Inheritance, LifeCycle: Formalization”, in *Proc. of the First International Workshop Critical infrastructures safety and security* (*CrlSS-Dessert’11*). Kirovograd, 2011, pp. 236–244.

[5] Е.М. Лаврищева, “Генерирующее и сборочное программирование. Аспекты разработки семейств программных систем”, *Кибернетика и системный анализ,* т. 49, №1, с. 129-144, 2013.

[6] Л.Л. Омельчук, *Формальні методи специфікації програм*. Київ, Україна: УкрІНТЕІ, 2010.

[7] В.Н. Редько, “Экзистенциальные основания композиционной парадигмы”, *Кибернетика и системный анализ*. № 2, с. 3–12, 2008.

[8] В.В. Соколов, “Технологія програмування активних динамічних сполук об’єктів” на *V науково-технічній конференції Пріоритетні напрямки розвитку телекомунікаційних систем та мереж спеціального призначення,* Київ, 2010, с. 232.

Стаття надійшла до редакції 24 березня 2017 року.

REFERENCE

[1] D.B. Buy, S.V. Kompan, “Formalization of objects, classes, methods in the object-oriented databases”, in *International conference of Problem of theoretical cybernetics*, Nizhniy Novgorod, 2011, pp. 81-85.

[2] D.B. Buy, S.V. Kompan, “Diagrams of classes of OOP : formalization and analysis”, *Proc. institute of the applied mathematics and mechanics*, vol. 27, Donetsk, pp. 51-65, 2013.

[3] A.G. Piskunov, Formalization of paradigm of the object-oriented programming. [Online]. Available: http://www.realcoding.net/dn/docs/machine.pdf. Accessed on: Jan. 19, 2016.

[4] D. Buy, S. Kompan, “The Concepts of Object, Class, Inheritance, LifeCycle: Formalization”, in *Proc. of the First International Workshop Critical infrastructures safety and security* (*CrlSS-Dessert’11*). Kirovograd, 2011, pp. 236–244.

[5] E.M. Lavrishcheva, “Generating and assembly programming. Aspects of developing families of software systems”, Cybernetics and Systems Analysis, vol. 49, No. 1, pp. 129-144, 2013.

[6] L.L. Omelychuk, *Formal Methods of Program Specification*. Kyiv, Ukraine: UkrINTEI, 2010.

[7] V.N. Redko, “Existential Foundations of the Compositional Paradigm”, *Cybernetics and Systems Analysis*. № 2, pp. 3-12, 2008.

[8] V. Sokolov, “Programming technology of active dynamic connections of objects” in *V scientific conference Priority directions of development of telecommunication systems and networks for special purposes,* Kyiv, 2010, p. 232.

ВЛАДИМИР СОКОЛОВ

ПРИМЕНЕНИЕ ФУНКЦИОНАЛЬНОЙ И РЕЛЯЦИОННОЙ МОДЕЛЕЙ В ОБЪЕКТНО-ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ

В работе представлены результаты исследований и практической апробации формальных методов описания объектно-ориентированных программ, пригодных для автоматической генерации текста программ на языке программирования. В качестве формальных моделей

использовано функциональную и реляционную модели. Функциональная модель представляет программу как схему соединения функциональных атомарных объектов, способных к непосредственному взаимодействию путем образования динамических соединений и автоматических вычислений, которая может представляться в графической форме. При этом схема соединения объектов рассматривается как схема решении задачи. Показано, что формализация именно схемы решения задачи, а не всей парадигмы программирования, делает процесс создания программы больше приближенным к практике. Определены требования к атомарным объектам как таковым, что составляют элементную базу объектно-ориентированной программы. Реляционная модель представляет объект как виртуальное отношение, схема которого задается классом, который реализует функциональную зависимость неключевых атрибутов от ключевых путем их вычислений, что позволяет применять реляционные операции для описания схемы решении задачи. Реляционная модель позволяет использовать язык, подобный структурированному языку запросов к базам данных, для описания схемы решения задачи и ее автоматического выполнения. Показано, что функциональная и реляционная модели пригодны для графического представления схемы решения задачи и являются достаточно выразительными для непосредственной генерации программ. Фактически, разработанные модели позволяют поднять процесс создания объектно-ориентированных программ на уровень выше, сосредоточиться на структуре программы, а не на ее составляющих, и дополнить пробел в существующих методах представления программ. В качестве основы для практической реализации использована технология программирования активных динамических соединений объектов.

Ключевые слова: объектно-ориентированное программирование, формализация программ, реляционная модель, функциональная модель, активные динамические соединения объектов.

VOLODYMYR SOKOLOV

APPLICATION OF FUNCTIONAL AND RELATIONAL MODELS IN OBJECT-ORIENTED PROGRAMMING

The paper presents the results of research and practical approbation of formal methods for describing object-oriented programs suitable for automatic generation of programs text to the programming language. As formal models, the functional and relational models are used. The functional model represents the program as a scheme of the connection of functional atomic objects that are capable of direct interaction by forming dynamic connections and automatic computations, which can be represented graphically. In this case, the connection scheme of objects is considered as a scheme of task solving. It is shown that the formalization of the task solution scheme, rather than the entire programming paradigm, makes the process of program creating more suitable for the practice. The requirements for atomic objects are determined as such, which form the elemental basis of the object-oriented program. The relational model represents the object as a virtual relationship, the scheme which is specified by the class that implements the functional dependence of non-key attributes from the key ones by their calculations, which makes it possible to apply relational operations to describe the scheme of task solving. The relational model allows using a language which is similar to the structured query language of databases, to describe the scheme of solving the task and to perform it automatically. It is shown that the functional and relational models are suitable for graphical representation of the task solution scheme and are sufficiently expressive for direct generation of programs. In fact, the developed models allow us to increase the process of creating object-oriented programs to a higher level, focusing on the structure of the program, rather than on its components, and to eliminate the lack of methods for presenting programs. The programming technology of active dynamic connections of objects has been used as a basis for practical implementation.

Keywords: object-oriented programming, programs formalization, relational model, functional model, active dynamic connections of objects**.**

Володимир Володимирович Соколов, кандидат технічних наук, доцент, доцент кафедри кібербезпеки та застосування автоматизованих інформаційних систем та технологій, Інститут спеціального зв'язку та захисту інформації Національного технічного університету "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна.

E-mail: vsokolov@i.ua.

Владимир Владимирович Соколов, кандидат технических наук, доцент, доцент кафедры кибербезопасности и применения автоматизированных информационных систем и технологий, Институт специальной связи и защиты информации Национального технического университета "Киевский политехнический институт имени Игоря Сикорского", Киев, Украина.

Volodymyr Sokolov, candidate of technical sciences, associate professor, associate professor at the cybersecurity and application of automated information systems and technologies academic department, Institute of special communication and information protection National technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine..

UDC 510.254:517.977.5

HRYHORII KRAVTSOV

SEMANTIC INTEROPERABILITY AS A BASIS OF MEANINGFUL ANALYTICS

The main objective of the article is to show how ontologies, which are basic items of the semantic interoperability, can be used as a basis of meaningful analytics. It opens the way to solve a lot of problems like expert selection. The author discovers the pointed problem on the boarder between information technologies and recruitment problems like boolean search. The article contains a real example of boolean search query regarding information technologies such J2EE, JDBC, JAXB, JPA, Servlets, JAX-WS and others. The weekness of understanding of meaning some terms by a finder leads to incorrect results of the search. Only ontologies can handle synonyms and other kinds of relations between two terms – it is very important for avoiding of a confusion. Unfortunately, where is a misunderstanding of meaning following terms – classification, taxonomy, ontology. The existing articles do not solve the problem of unified understanding – it is an aspect of author's investigation. The author shows how the analytical hierarchy process in the couple with ontologies can be used for empowering of existing approaches for solving the problem of expert selection. All conclusions are based on the detailed analysis of existing open publications. The pointed combination opens new horizons for predictive meaningful analytics.

Keywords: semantic web, semantic interoperability, ontology, taxonomy, classification, intelligent search, predictive analytics, prescriptive analytics, meaningful analytics.

Problem statement. Search engines, used in the real life for finding of experts, are not very intelligent and therefore it is often difficult to find what you are looking for. I would like to draw specialist's attention to the problem of using "boolean" search. It has been more and more obvious that keywords matching including synonyms is not sufficient for recruitment. As an Mathematician and computer scientist I have many times noticed that the boolean search leads to various issues. Furthermore, the boolean search has disadvantage – you cannot define priorities for terms. It means that they do not provide any functionality for meaningful analysis.

Analysis of recent researches and publications [1] – [17]. I will investigate the "smart" query pointed out by Jacco Valkenburg on "Global Recruiting Roundtable" [5] and will explain why a search engine returns a great number of false-positive CVs.