УДК 004.896

Д.Б. Єльчанінов[1], М.С. Косіло[1], Н.В. Бєлова[2]

[1] *Національний технічний університет «Харківський політехнічний інститут», Харків*
[2] *Харківський національний університет радіоелектроніки, Харків*

ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ ПРОЕКТУВАННЯ ПРОГРАМНИХ СИСТЕМ

Розроблено метод автоматизації процесу аналізу технічного завдання на основі технології синтаксичного аналізу тексту. Показано зв'язок між текстовим описом технічного завдання, його синтаксичним деревом та акторами, об'єктами й повідомленнями діаграм послідовностей (UML sequence diagram). Адаптовано основні поняття теорії генетичних алгоритмів (ген; генотип; популяція; цільова функція; оператори відбору, схрещування, мутації та редукції; критерії зупинення) до автоматизації побудови цих діаграм.

Ключові слова: *програмні системи, автоматизація проектування, синтаксичний аналіз, генетичні алгоритми, UML.*

Вступ

Постановка проблеми. Сьогодні актуальною практичною проблемою є позбавлення людини від повсякденної та робочої рутинної діяльності. З наукової точки зору це пов'язано, наприклад, з вирішенням проблем автоматичного аналізу тексту (зокрема, перекладу) або автоматичного синтезу системи з бажаною поведінкою з визначених компонентів (синтез електронних систем, проектування бізнес-процесів).

У галузі об'єктно-орієнтованого аналізу та проектування вищезазначені проблеми пов'язані з процесом аналізу технічного завдання на розробку програмної системи та її проектування з визначених бібліотечних програмних компонентів.

Аналіз останніх досліджень і публікацій. Жодна з існуючих систем проектування не здатна в автоматичному режимі вирішувати зазначені завдання. Наприклад, продукти сімейств IBM Rational, Borland Together, Microsoft Visio, Sparx Systems Enterprise Architect, Gentleware Poseidon, SmartDraw, Dia, Telelogic TAU G2 та StarUML автоматизують процес проектування програмних систем, але аналіз технічного завдання та побудову діаграм UML проектувальники здійснюють вручну [1].

Використовуючи інформаційні лінгвістичні технології, розроблені для аналізу та розуміння текстів на природних мовах [2; 3], діяльність проектувальника щодо аналізу технічного завдання [4, с. 154; 5, с. 221] можна повністю автоматизувати, абсолютно виключивши з неї людину: на вхід системи подається текстовий опис, з якого на виході маємо готову модель класів програмної системи.

Основною метою проектування програмних систем є їх постійне вдосконалення [6; 7]. За аналогічними принципами працюють генетичні алгоритми, які, на відміну від численних стратегій та методів вирішення складних проблем, орієнтуються перш за все на вдосконалення існуючого рішення, але не на пошук найкращого, котрого взагалі може не існувати [8]. Генетичні алгоритми є стратегією, яку потрібно пристосовувати до певної предметної галузі [9].

Формулювання мети статті. Адаптація технології синтаксичного і семантичного аналізу тексту та основних понять теорії генетичних алгоритмів до вирішення задачі автоматизації проектування програмних систем.

Автоматизація процесу аналізу технічного завдання

Розглянемо фрагмент текстового опису потоку подій у технічному завданні (російською мовою): «Человек подходил, вставлял карточку, набирал пин-код, а банкомат деньги не выдавал». Цей фрагмент за допомогою інформаційної лінгвістичної технології автоматично перетворюється у відповідне синтаксичне дерево, яке наведене на рис. 1 [10].

Іменники істота «Человек» та неістота «Банкомат», що знаходяться у предикативних зв'язках з певними дієсловами, можуть бути автоматично перетворені відповідно у актора (істота) та об'єкт (неістота) діаграми послідовності.

Дієслова «Вставлять», «Набирать» та «Выдавать», що належать до певних сурядних зв'язків, та іменники «Карточка», «Пин-код» та «Деньги», що знаходяться з цими дієсловами у відповідних комплетивних зв'язках, можуть бути автоматично перетворені у повідомлення «Вставлять карточка», «Набирать пин-код», «Деньги не выдавать» діаграми послідовності (рис. 2).

За наявності відповідної бібліотеки класів, об'єкт «Банкомат» та повідомлення «Вставлять карточка», «Набирать пин-код» та «Деньги не выдавать» можуть бути автоматично зіставлені з класом «ATM» та його операціями «InsertCard», «EnterPIN» та «NoCash» (рис. 3).

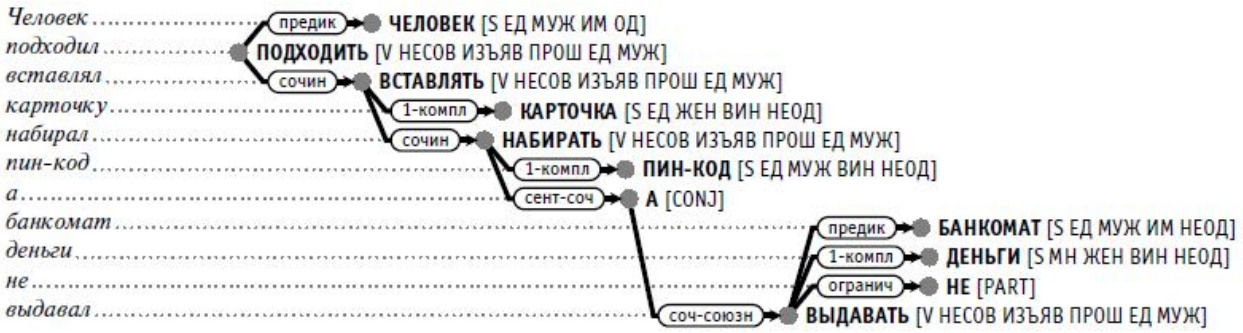

Рис. 1. Синтаксичне дерево

Рис. 2. Актор, об'єкт та повідомлення діаграми послідовності

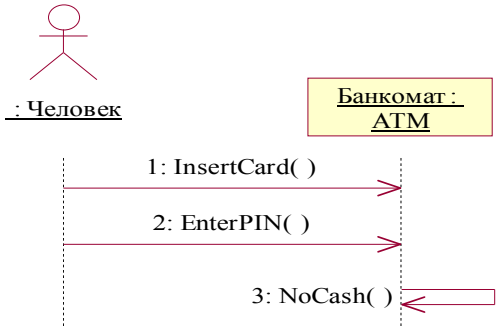

Рис. 3. Клас та операції діаграми послідовності

Автоматизація процесу побудови діаграми послідовності

Наведемо приклад процесу автоматичної побудови діаграми послідовності із застосуванням генетичних алгоритмів. Нехай необхідно створити ПС, яка перетворює число, що має користувач, наступним чином: розкладає його на цілу та дробову частину, перетворює їх відповідним шляхом та повертає результат користувачеві.

Припустимо, що перетворення числа здійснюється двома об'єктами програмної системи: перший об'єкт розкладає число, перетворює його цілу частину та підсумовує результати перетворення цілої та дробової частини; другий об'єкт перетворює дробову частину. Отже, структура діаграми послідовності об'єктів програмної системи має такий вигляд, як на рис. 4.

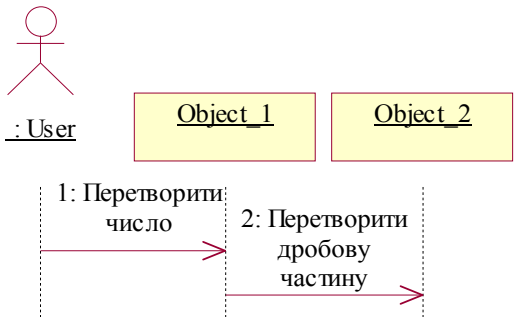

Рис. 4. Структура діаграми послідовності

Опис гена. Може бути декілька варіантів реалізації об'єктів. Наприклад, перший об'єкт може мати одну з чотирьох реалізацій, поданих на рис. 5.

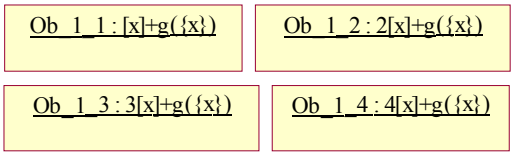

Рис. 5. Варіанти реалізації першого об'єкта

Перша реалізація Ob_1_1 першого об'єкта розкладає число x на цілу[x] та дробову {x} частину, множить цілу частину [x] на одиницю та складає її з результатом перетворення дробової частини g({x}). Іншими словами, перша реалізація Ob_1_1 першого об'єкта виконує операцію f(x)=[x]+g({x}). Отже, ця реалізація об'єкта належить до певного класу, що має відповідну операцію.

Інші реалізації першого об'єкта Ob_1_2, Ob_1_3 та Ob_1_4 відрізняються від першої реалізації Ob_1_1 функцією f(x): для Ob_1_2 вона має вигляд 2[x]+g({x}), для Ob_1_3: 3[x]+g({x}), для Ob_1_4: 4[x]+g({x}).

Другий об'єкт може мати одну з чотирьох реалізацій, наведених на рис. 6.

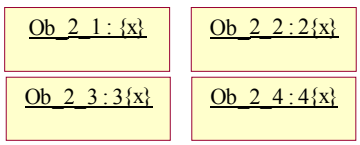

Рис. 6. Варіанти реалізації другого об'єкта

Перша реалізація Ob_2_1 другого об'єкта множить дробову частину {x} на одиницю. Іншими словами, виконує операцію g({x})={x}. Отже, ця реалізація об'єкта належить до певного класу, що має від-

повідну операцію. Інші реалізації другого об'єкта Ob_2_2, Ob_2_3 та Ob_2_4 відрізняються від першої реалізації Ob_2_1 функцією g({x}): для Ob_2_2 вона має вигляд 2{x}, для Ob_2_3: 3{x}, для Ob_2_4: 4{x}.

У термінах генетичних алгоритмів варіант реалізації об'єкта є геном. Отже, варіанти Ob_1_1, Ob_1_2, Ob_1_3 та Ob_1_4 реалізації Object_1, а також варіанти Ob_2_1, Ob_2_2, Ob_2_3 та Ob_2_4 реалізації Object_2 є генами.

Опис генотипу. Може бути декілька варіантів реалізації програмної системи. Наприклад, якщо Object_1 має реалізацію Ob_1_1, а Object_2 має реалізацію Ob_2_1, то програмна система буде реалізована так, як показано на рис. 7.

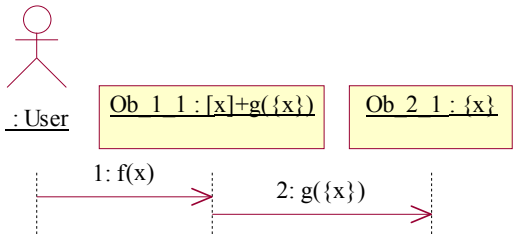

Рис. 7. Варіант реалізації програмної системи

У термінах генетичних алгоритмів варіант реалізації програмної системи є генотипом. Формально цей генотип можна позначити вектором

(Ob_1_1; Ob_2_1),

де перший компонент вказує, яку реалізацію має Object_1, а другий компонент – на реалізацію Object_2.

Опис популяції. Розглянемо інший варіант реалізації програмної системи: Object_1 має реалізацію Ob_1_2, а Object_2 має реалізацію Ob_2_2 (рис. 8).

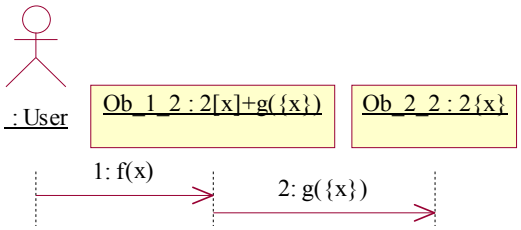

Рис. 8. Інший варіант реалізації

У термінах генетичних алгоритмів цей варіант реалізації програмної системи є генотипом, що формально можна позначити вектором

(Ob_1_2; Ob_2_2).

Разом з першим варіантом цей варіант реалізації програмної системи утворює популяцію, яка складається з двох генотипів:

Genotype_1_1 = (Ob_1_1; Ob_2_1);

Genotype_2_2 = (Ob_1_2; Ob_2_2).

Формально популяція описується множиною генотипів:

Population = {Genotype_1_1; Genotype_2_2}.

Опис цільової функції. Для порівняння варіантів реалізації програмної системи можна використовувати такі критерії:

– кількість об'єктів, які складають варіант реалізації програмної системи;

– кількість зв'язків між об'єктами, які складають варіант реалізації програмної системи;

– відхилення результату перетворення від очікуваного користувачем.

Якщо порівнювати два варіанти реалізації, то серед них кращим є той, який має меншу кількість об'єктів, зв'язків між ними та менше відхилення результату. Наприклад, якщо користувач очікує у результаті перетворення додатного числа щонайменшого значення, тоді Genotype_1_1 є кращим за Genotype_2_2. Тому що вони мають однакову кількість об'єктів та зв'язків між ними, але Genotype_1_1 відповідає функція f(x)=[x]+{x}=x, що фактично не змінює вхідне число x, в той час як Genotype_2_2 f(x)=2[x]+2{x}=2x, що фактично збільшує вхідне число x в два рази.

У випадку, якщо варіанти реалізації не можна порівняти безпосередньо (наприклад, перший має більшу кількість об'єктів, а другий – більшу кількість зв'язків між ними), то треба враховувати, котрий з цих двох критеріїв є важливішим для реалізації даної програмної системи.

Опис оператора відбору. Очевидно, практичну цінність мають кращі варіанти реалізації програмної системи. Їх відбір забезпечується методами селекції, описаними вище. Оскільки у нашому випадку цільова функція є фактично функцією мінімізації, доцільно використовувати метод рангової селекції. Згідно з цим методом оператор відбору порівнює варіанти за допомогою цільової функції і впорядковує їх у порядку зростання її значення, тобто від кращих до гірших. Наприклад, якщо на вхід оператора V відбору подати Genotype_1_1 та Genotype_2_2, то він упорядкує їх наступним чином:

V(Genotype_1_1; Genotype_2_2) =

= Genotype_1_1 > Genotype_2_2,

де знак «>» позначає, який генотип є кращим.

Якщо є три та більше генотипи, то парним порівнянням можна визначити відношення між ними та упорядкувати їх (строго лінійно або кусочно-лінійно).

Опис оператора схрещування. Поліпшити варіант реалізації програмної системи можна за рахунок обміну об'єктами з іншим варіантом реалізації. Формально цей процес забезпечує оператор схрещування, який виконує обмін генами між генотипами. У даному випадку доцільно використовувати одноточкове схрещування. Наприклад, якщо на вхід оператора S схрещування подати Genotype_1_1 та Genotype_2_2, то він змінить їх на Genotype_1_2 та Genotype_2_1 так, як показано на рис. 9.

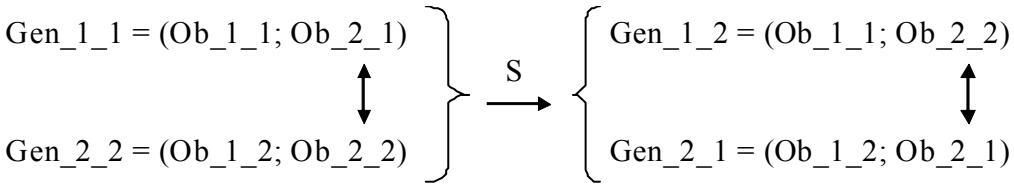

Рис. 9. Дія оператора схрещування

У термінах генетичних алгоритмів Genotype_1_1 та Genotype_2_2 є «Батьками», а Genotype_1_2 та Genotype_2_1 – «Нащадками».

Варіант Genotype_1_2 є гіршим за Genotype_1_1, але кращим за Genotype_2_2 тому, що

$$[x]+\{x\} < [x]+2\{x\} < 2[x]+2\{x\}.$$

Реалізацію програмної системи, що відповідає Genotype_1_2, показано на рис. 10.

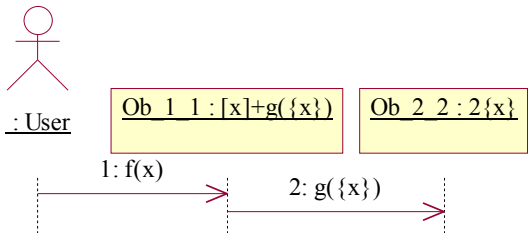

Рис. 10. Genotype_1_2

Варіант Genotype_2_1 є гіршим за Genotype_1_1, але кращим за Genotype_2_2 тому, що

$$[x]+\{x\} < 2[x]+\{x\} < 2[x]+2\{x\}.$$

Реалізацію програмної системи, що відповідає Genotype_2_1, показано на рис. 11.

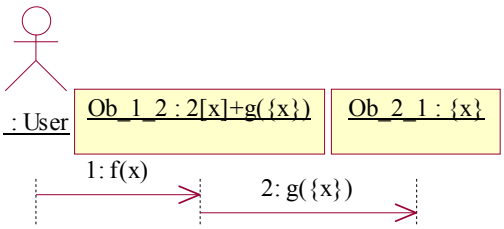

Рис. 11. Genotype_2_1

Варіанти Genotype_1_2 та Genotype_2_1 у загальному вигляді не можна порівняти тому, що:

а) $[x]+2\{x\} > 2[x]+\{x\}$, при $0 < x < 1$;

б) $[x]+2\{x\} < 2[x]+\{x\}$, при $x >= 1$.

Тобто, якщо користувач має додатні числа, що менше 1, тоді для нього кращим є Genotype_2_1, у іншому випадку – Genotype_1_2.

Опис оператора мутації. Поліпшити варіант реалізації програмної системи можна за рахунок зміни варіанта реалізації об'єкта. Формально це забезпечує оператор мутації, який виконує заміну гена у генотипі. Зі всієї популяції випадковим чином вибирається до 5% генотипів. Після цього у кожному генотипі випадково вибирається ген, значення якого міняється на інше, випадково вибране з множини допустимих значень (тобто у варіанті реалізації програмної системи варіант реалізації одного об'єкта замінюється на будь-який інший варіант реалізації цього об'єкта). Наприклад, якщо на вхід оператора M мутації подати Genotype_1_1, то він може змінить його на Genotype_2_1:

$$M((Ob\_1\_1; Ob\_2\_1)) = (Ob\_1\_2; Ob\_2\_1)$$

У результаті дії оператора мутації генотип може як погіршитись, так і покращитись (наприклад, якщо би мутація відбувалась у протилежному напряму: від Genotype_2_1 до Genotype_1_1).

Опис оператора редукції. Найгірші варіанти реалізації програмної системи повинні бути виключені з розгляду. Формально це забезпечує оператор редукції, який видаляє найслабкіші генотипи з популяції, що зросла у результаті поповнення нащадками. Наприклад, якщо на вхід оператора R редукції подати Population, яка була поповнена «нащадками» (Genotype_1_2 та Genotype_2_1) у результаті схрещування «батьків» (Genotype_1_1 та Genotype_2_2), то (якщо користувач має додатні числа, що менше 1) оператор редукції зменшить популяцію таким чином:

$$R(Population \cup \{Genotype\_1\_2; Genotype\_2\_1\}) =$$
$$= R(\{Genotype\_1\_1; Genotype\_2\_2;$$
$$Genotype\_1\_2; Genotype\_2\_1\}) =$$
$$= \{Genotype\_1\_1; Genotype\_2\_1\}.$$

Genotype_2_2 був видалений як найслабкіший. Genotype_1_2 є гіршим за Genotype_2_1 тому, що користувач має додатні числа, що менше 1.

Видаляється стільки найгірших варіантів реалізації, скільки було створено нащадків в результаті схрещування для збереження постійного розміру популяції.

Опис критерію зупинення. Після виконання оператора редукції генетичний алгоритм можна запускати спочатку для пошуку більш кращих варіантів реалізації програмної системи. Теоретично, це може тривати нескінченно.

Одним з критеріїв зупинення роботи генетичного алгоритму може бути поява генотипу, що відповідає варіанту реалізації з певною кількістю об'єктів та зв'язків між ними. Але, якщо такого варіанта не існує, то алгоритм ніколи не зупиниться. Іншим критерієм може бути поява генотипу, що відрізняється від бажаного результату з певною похибкою. Але якщо таких генотипів не існує, то алгоритм не зупиниться.

Критерієм, що гарантує зупинення роботи генетичного алгоритму є заздалегідь задана кількість циклів, при досягненні якої він припиняє свою роботу.

Приклад процесу автоматичної побудови діаграми послідовності. Нехай користувач має додатні числа, не менші за 1. Треба створити програм-

ну систему, яка має структуру, об'єкти, та варіанти їх реалізації, що описані вище, та щонайменше перетворює число користувача. Припустимо, що випадковим чином створено наступну популяцію:

Population = {Genotype_3_3; Genotype_4_4};

Genotype_3_3 = (Ob_1_3; Ob_2_3);

Genotype_4_4 = (Ob_1_4; Ob_2_4).

Діаграми послідовностей програмної системи, що відповідають Genotype_3_3 та Genotype_4_4, показано на рис. 12 та 13. Схрещування Genotype_3_3 та Genotype_4_4 показано на рис. 14. Діаграми послідовностей програмної системи, що відповідають Genotype_3_4 та Genotype_4_3, показано на рис. 15 та 16.

Припустимо, що ймовірність застосування оператора мутації настільки мала, що на першому кроці роботи генетичного алгоритму він не застосовується.

Оператор редукції виконується таким чином:

R(Population ∪ {Genotype_3_4; Genotype_4_3}) =

= R({Genotype_3_3; Genotype_4_4;

Genotype_3_4; Genotype_4_3}) =

= {Genotype_3_3; Genotype_3_4}.

Це виходить з того, що при x >= 1

3[x]+3{x} < 3[x]+4{x} < 4[x]+3{x} < 4[x]+4{x}.

Отже, склад нової популяції покращився у порівнянні з начальною популяцією.

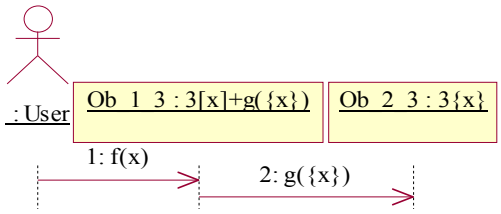

Рис. 12. Genotype_3_3

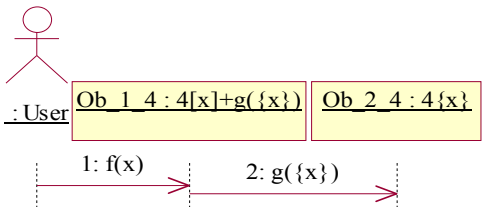

Рис. 13. Genotype_4_4

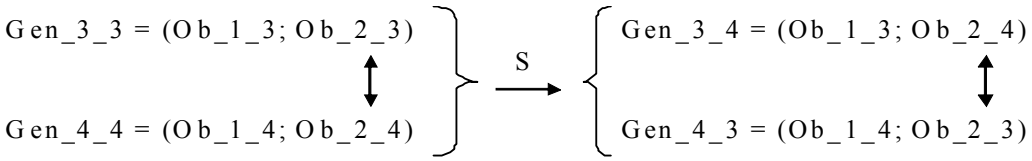

Рис. 14. Схрещування Genotype_3_3 та Genotype_4_4

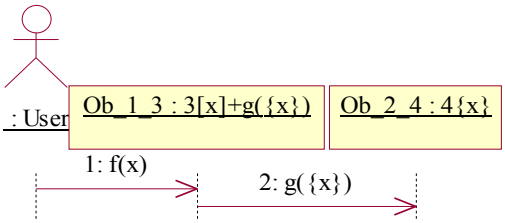

Рис. 15. Genotype_3_4

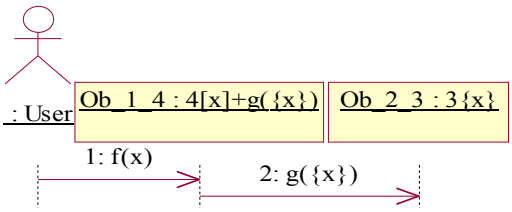

Рис. 16. Genotype_4_3

Висновки

Ефект впровадження методів та технологій синтаксичного аналізу та генетичних алгоритмів в практику проектування програмних систем істотно залежить від якості бібліотек компонентів, на базі яких будується проект програмної системи.

Перспективи подальших розвідок полягають у дослідженні особливості реалізації запропонованих методів у системах хмарних обчислень (Cloud Computing).

Список літератури

1. Бабич А.В. UML: первое знакомство / А.В. Бабич. – М.: Интернет-Ун-т информ. технологий: БИНОМ. Лаб. знаний, 2008. – 175 с.

2. Petrova M.A. The Compreno Semantic Model: The Universality Problem / M.A. Petrova // International Journal of Lexicography. – 2014. – Vol. 27, Issue 2. – P. 105-129.

3. The Compreno Semantic Model as an Integral Framework for a Multilingual Lexical Database / E. Manicheva, M. Petrova, E. Kozlova, T. Popova // Proceedings of the 3rd Workshop on Cognitive Aspects of the Lexicon (CogALex-III), December 2012. – Mumbai: COLING, 2012. – P. 215-230.

4. Новиков Ф.А. Моделирование на UML / Ф.А. Новиков, Д.Ю. Иванов. – СПб.: Наука и техника, 2010. – 635 с.

5. Рамбо Дж. UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо, М. Блаха. – СПб.: Питер, 2007. – 544 с.

6. Гринфилд Д. Фабрики разработки программ / Д. Гринфилд, К. Шорт. – М. : Диалектика, 2007. – 591 с.

7. Лаврищева Е.М. Сборочное программирование. Основы индустрии программных продуктов / Е.М. Лаврищева, В.Н. Грищенко. – К.: Наукова думка, 2009. – 372 с.

8. Genetic Algorithms for Applied CAD Problems /, V.M. Kureichik, S.P. Malioukov, V.V. Kureichik, A.S. Malioukov. – Berlin: Springer, 2009. – 256 p.

9. Скобцов Ю.А. Основы эволюционных вычислений / Ю.А. Скобцов. – Донецк: ДонНТУ, 2008. – 326 с.

10. Синтаксическое дерево [Электронный ресурс] / Национальный корпус русского языка. – Режим доступа к ресурсу: http://ruscorpora.ru/syntax/2003/51_105.pdf.

Надійшла до редколегії 12.05.2014

Рецензент: д-р техн. наук, проф., І.В. Рубан, Харківський університет Повітряних Сил ім. І. Кожедуба, Харків.

ТЕХНОЛОГИИ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ ПРОГРАММНЫХ СИСТЕМ

Д.Б. Ельчанинов, Н.С. Косило, Н.В. Белова

Разработан метод автоматизации процесса анализа технического задания на основе технологии синтаксического анализа текста. Показана связь между текстовым описанием технического задания, его синтаксическим деревом и актерами, объектами и сообщениями диаграмм последовательностей (UML sequence diagram). Адаптированы основные понятия теории генетических алгоритмов (ген; генотип; популяция; целевая функция; операторы отбора, скрещивания, мутации и редукции; критерии останова) к автоматизации построения этих диаграмм.

Ключевые слова: *программные системы, автоматизация проектирования, синтаксический анализ, генетические алгоритмы, UML.*

PROGRAM SYSTEMS DESIGN AUTOMATION TECHNOLOGIES

D.B. Elchaninov, N.S. Kosilo, N.V. Belova

The method of automation of process of the analysis of the specification on the basis of technology of parse of the text is developed. Relationship between the text description of the specification, its syntax tree and actors, objects and messages of UML sequence diagram is shown. The basic concepts of the genetic algorithms theory (a gene; genotype; population; target function; operators of selection, transposition, mutation and reduction; criteria of break) are adapted to automation of creation of these diagrams.

Keywords: *program systems, design automation, parse, genetic algorithms, UML.*