УДК 681.3.06

В.Г. Левицький, аспір.
Житомирський інженерно-технологічний інститут

ПРОГРАМНА РЕАЛІЗАЦІЯ МОВ ОПИСУ МАТЕМАТИЧНИХ СТРУКТУР

(Представлено к.т.н., доц. М.М. Колодницьким)

Наведено коротку характеристику програмної реалізації підсистеми лінгвістичного забезпечення чисельного аналізу математичних задач у програмній системі "DSR Open Lab 1.0". Програмна інженерія передбачає існування декількох етапів створення програмного забезпечення, одним з яких є етап проектування. Представлені результати саме цієї частини життєвого циклу програмних систем. Викладений матеріал ілюструє застосування концепцій об'єктно-орієнтованої парадигми програмування до побудови підсистеми лінгвістичного забезпечення.

Вступ

На даний час розроблено великий арсенал програмних засобів чисельного аналізу математичних задач. Але практика доводить, що їх функціональні можливості та темпи розвитку не повною мірою задовольняють нові, всезростаючі вимоги до такого класу програмного забезпечення. Зокрема, як неодноразово зазначалось на недавніх міжнародних конференціях, що присвячені розв'язку математичних задач за допомогою обчислювальної техніки, існуючі програми чисельного аналізу (рішення) математичних задач (такі, наприклад, як MATLAB, Maple V, Mathematica, Mathcad) "не досить гнучкі та інтерактивні" та "не мають дружнього користувацького інтерфейсу". Аналіз загальних характеристик згаданого програмного забезпечення дозволяє стверджувати, що однією з їх основних проблем є занадто жорсткий інтерфейс та використання проблемно-орієнтованих мов, що ускладнені в частині лексики, синтаксису та семантики.

Тому в рамках роботи над програмною системою (ПС) "DSR Open Lab 1.0" [1–3] було застосовано новий підхід до побудови лінгвістичного забезпечення чисельного аналізу математичних задач. Особливістю даного підходу є нетрадиційний для математичного програмного забезпечення розподіл функцій між графічним інтерфейсом програмної системи та її лінгвістичним забезпеченням [4]. Як критерій розподілу функціональності використовується типологія математичних структур [5–7]. На основі запропонованого підходу до побудови лінгвістичного забезпечення було розроблено мови опису ряду математичних структур (скінченні множини, числові системи, функціональні відношення, дискретні групи, поліноми, векторний простір, системи лінійних алгебраїчних рівнянь, системи нелінійних алгебраїчних рівнянь, звичайні диференціальні рівняння, задачі оптимізації).

У даній роботі наведено коротку характеристику програмної реалізації підсистеми лінгвістичного забезпечення чисельного аналізу математичних задач в ПС "DSR Open Lab 1.0". Програмна інженерія передбачає існування декількох етапів створення програмного забезпечення, одним з яких є етап проектування [8]. Результати саме цієї частини життєвого циклу програмних систем представлені далі. Викладений матеріал ілюструє застосування концепцій об'єктно-орієнтованої парадигми програмування до побудови підсистеми лінгвістичного забезпечення.

Загальна характеристика програмної реалізації проекту лінгвістичного забезпечення

Концептуально програмна реалізація підсистеми лінгвістичного забезпечення логічно розбивається на декілька важливих блоків: структури та типи даних; компілятор (шаблон лексичного аналізатора та скінченні автомати, шаблон синтаксичного аналізатора, спільна для всіх мов опису частина синтаксичного та семантичного аналізу, блок синтаксично-керованої генерації коду, характерні для конкретної мови опису структури даних); віртуальна машина; блок зв'язку з підсистемами аналізу математичних структур (інформація про час виконання аналізу); система обробки помилок часу трансляції та виконання коду; тестовий проект.

Структури та типи даних зведено в проект AlmaData, який використовується деякими підсистемами аналізу математичних структур ПС "DSR Open Lab 1.0". Інші пункти утворюють проект DSRLanguage. Коротку характеристику цих проектів наведено в табл. 1.

Таблиця 1

Характеристика проектів лінгвістичного забезпечення

Проект	Розмір	Рядки коду мовою C++	Кількість пакок	Кількість файлів		
				.cpp	.h	(.grm, .h, .lrt, .rul)
DSRLanguage	1905Kb	51491	52	89	138	51
AlmaData	356Kb	14066	9	16	28	–
Разом	2261Kb	65557	61	105	166	51

Далі розглянемо коротко програмну реалізацію деяких із вищезгаданих блоків підсистеми лінгвістичного забезпечення.

Програмна реалізація підпроекту скінченних автоматів

Підпроект скінченних автоматів складається з двох частин: базового шаблону скінченного автомату та реалізацій скінченних автоматів, що призначені для розбору таких лексем мов опису математичних структур, які мають порівняно складну структуру (числові константи, рядкові константи, спеціальні оператори, ідентифікатори змінних і функцій), рис. 1, *а–г*.

Ієрархія класів підпроекту скінченних автоматів зображена на рис. 2.

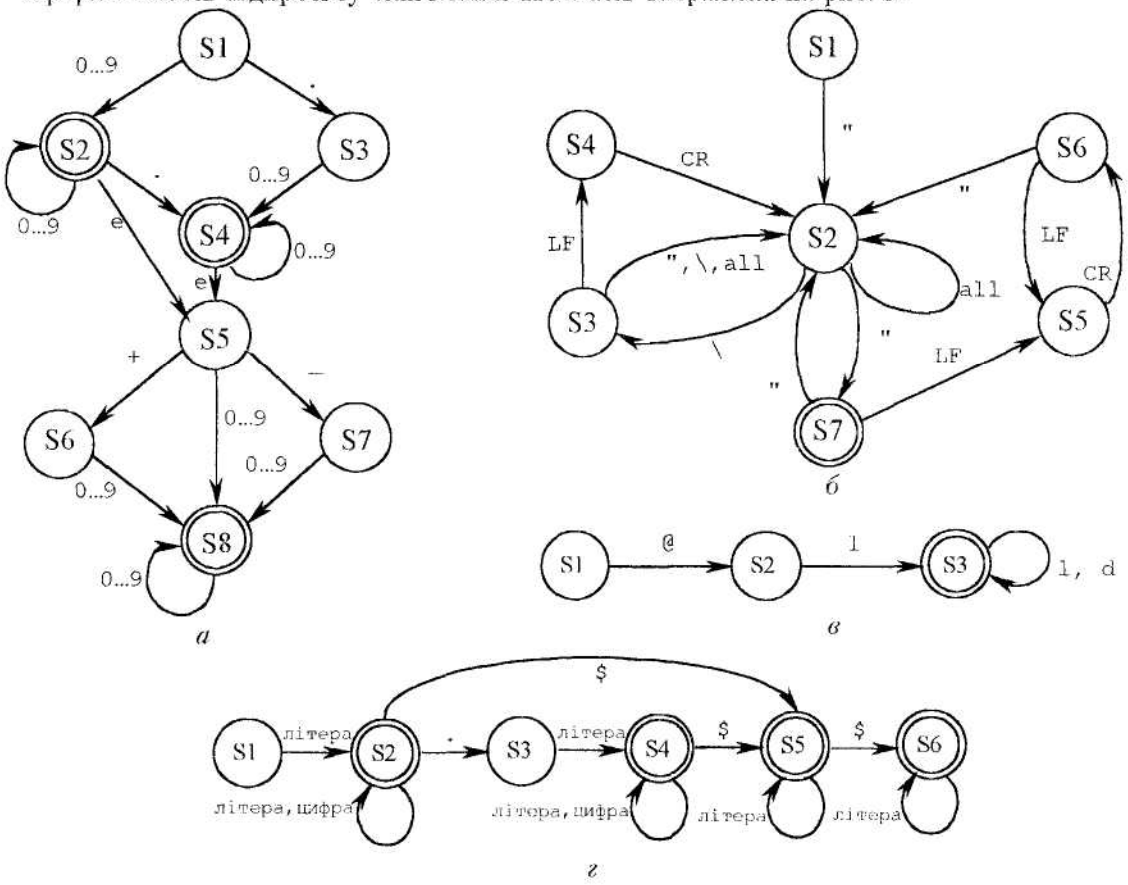

Рис. 1. Скінченні автомати підсистеми лінгвістичного забезпечення

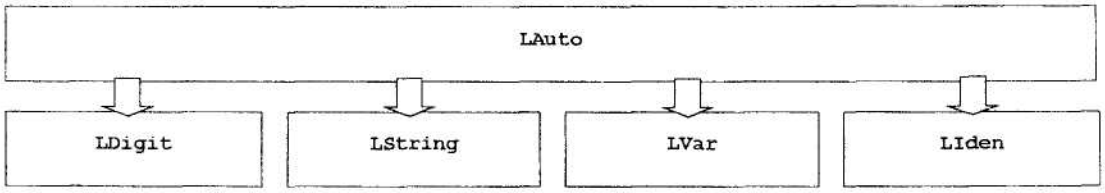

Рис. 2. Ієрархія класів підпроекту скінченних автоматів

Базовий клас LAuto є шаблоном скінченного автомата. Він реалізує загальний алгоритм роботи автомата; представлення станів, алфавіту тощо структурами даних мови програмування та інші спільні для всіх лексем функції. Інші класи ієрархії на рис. 2 програмно реалізують математичні моделі конкретних автоматів для лексем мов опису математичних структур.

Таким чином, для створення нового класу скінченного автомата, як мінімум, потрібно визначити конкретні значення множин станів і кінцевих станів, функцію переходу та алфавіт. При цьому вважається, що на вхід скінченного автомата подається ланцюжок символів із набору ASCII. Перекодування вхідних символів алфавіту визначається в функції What. У необхідності можна визначити дії автомата при переходах між станами в класах шляхом перевизначення функції Deeds (та, можливо, Analisys).

Програмна реалізація шаблону синтаксичного та лексичного аналізатора

Ієрархія класів даного підпроекту зображена на рис. 3. Вона включає в себе клас повідомлення про помилку та попередження (ErrInfo та WarnInfo), клас-контейнер повідомлень про помилки та попередження (TErrorMessages та TWarningMessages), сервісний клас повідомлення про помилку та попередження (Error та Warning), клас фатальної помилки (FatalError), клас опису керуючої таблиці синтаксичного аналізу (LRTDATA), клас комірки керуючої таблиці синтаксичного аналізу (ActionItem), клас керуючої таблиці синтаксичного аналізу (ActionT), клас лексеми (Lexem), клас проміжного представлення вхідного потоку символів (Source), клас шаблону лексичного аналізатора (Lex), клас шаблону контейнера загальних опцій лексичного та синтаксичного аналізу (Common), клас контейнера загальних опцій лексичного та синтаксичного аналізу (DefaultCommon), клас базового шаблону синтаксичного аналізатора (InfoParser), клас шаблону синтаксичного аналізатора з обробкою помилок та попереджень (DiagParser), клас повного шаблону синтаксичного аналізу (BaseParser).

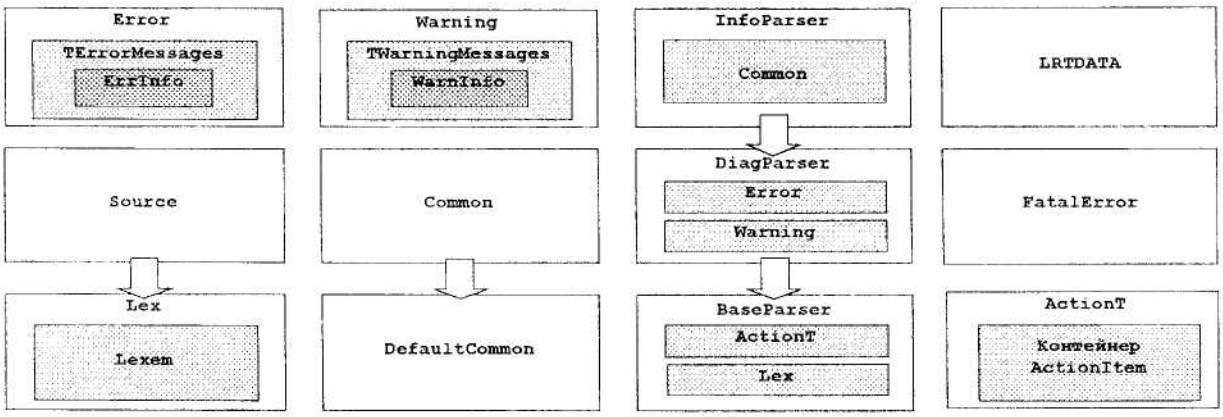

Рис. 3. Ієрархія класів підпроекту шаблону парсеру та сканеру

Згідно із загальними принципами об'єктно-орієнтованої парадигми, що покладена в основу програмної реалізації підсистеми лінгвістичного забезпечення, ефективним способом організації розбору мови опису є породження власних класів лексичного та синтаксичного аналізу на основі розробленого шаблону. Для лексичного аналізатора ця процедура повинна включати крім безпосереднього породження нового класу від класу Lex, як мінімум ще й перевизначення функції NextLexem, що відповідає за аналіз лексики конкретної мови. При побудові лексичного аналізатора подібним способом доцільним є використання підпроекту скінченних автоматів.

Для створення нового класу синтаксичного аналізатора на основі шаблону, як мінімум, потрібно після породження класу-нащадка, перевизначити функцію NewLex таким чином, щоб вона повертала покажчик на реальний клас лексичного аналізу, а не на базовий шаблон сканера Lex, як це відбувається за замовчуванням. Подібна структура нового синтаксичного аналізатора лише частково забезпечить етап аналізу та дозволить вирішувати питання, чи є коректним, з точки зору синтаксису, вхідний текст опису. Для реалізації функцій етапу синтезу (генерації коду) необхідно, крім того, перевизначити функцію GenCode, реалізувавши реальні дії за синтезом даних, що необхідні для семантичного аналізу, генерації коду та оптимізації.

Для повної працездатності отриманий клас синтаксичного аналізатора потребує керуючі таблиці синтаксичного аналізу та програмні тексти, які є специфічними для конкретної мови опису. Передбачається, що такі дані готуються за допомогою компілятора компіляторів.

Програмна реалізація генерації об'єктного коду та віртуальної машини

Зважаючи на те, що задачі чисельного аналізу в загальному випадку можуть мати складну структуру, віртуальна машина підсистеми лінгвістичного забезпечення ПС "DSR Open Lab 1.0" реалізує блочну організацію структур даних, з якими вона працює. При такій побудові адресний простір машини складає певна кількість блоків (сегментів), кожен з яких або асоціюється з компонентою математичної структури, або резервується для внутрішнього використання віртуальною машиною. Обмежень за розміром блоків адресного простору не існує.

Наведемо список ряду блоків даних віртуальної машини:

1. Блок статичних даних. До цього блоку заносяться всі статичні дані (іменовані числа) опису математичної структури. Всі елементи в сегменті розташовані послідовно; в залежності від типу під кожен елемент відводиться 4, 8, 10, 12, 16, 24 або 32 байти.

2. Блок стеку. Цей блок використовується віртуальною машиною для виконання дій, які потребують організації пам'яті у вигляді стека (наприклад, для реалізації механізму виклику функції користувача).

3. Стек обчислень зберігає проміжні результати під час інтерпретації машинних команд віртуальною машиною. Для прискорення обчислень розмір одного елемента (слова) для стека обчислень є постійним і дорівнює найбільшому з розміру типів даних.

4. Блок-помилка. Єдина функція даного сегмента – обробка помилок користувача, що виникають при описі математичної структури. Він містить також внутрішні змінні, які дозволяють компілятору ігнорувати деякі помилки з тим, щоб провести якомога детальніший аналіз опису математичної задачі.

5. Блок функцій користувача. Всі визначені користувачем функції, що описані в математичній структурі типу, відмінного від "Функціональних відношень", містяться в даному блоці.

6. Блоки, які асоційовані з певною математичною структурою. Кожен блок містить ідентифікатор типу математичної структури; її ім'я; таблицю змінних, що визначені в математичній структурі, код, що згенерований компілятором для даної математичної структури. Крім того, в залежності від класу математичної структури, блок може містити деякі специфічні дані.

Ієрархія класів генерації об'єктного коду та віртуальної машини наведена на рис. 4.

Наведена ієрархія класів включає в себе блок генерації коду, блок зв'язку з підсистемами аналізу математичних структур (інформація часу виконання аналізу) та блок інтерпретації коду віртуальної машини.

До першого блоку відносяться: клас-менеджер пам'яті (C_Heap2Free), базові класи підтримки інформації часу виконання аналізу математичної структури (TSoulHandle, BaseSoulItem, BaseProjectSoulItem), клас зберігання текстової інформації (TextSoul), клас підтримки формування блоку віртуальної машини (SegmentsInfoSoul), базовий клас шаблону специфічної для конкретної математичної структури інформації (BaseModelStorage), класи специфічної для конкретної математичної структури інформації (PolynomStorage, NumeryStorage, FunctionStorage, GroupsStorage, DFMSetStorage, ExtrStorage, OdeStorage, VectorStorage, SNAUStorage, SLAUStorage), класи керування генерацією коду (EvalStack та CompilerEvalStack), класи представлення змінних (VariableRec, VariableList), шаблонів функцій (FunctionRecord, FunctionTable), математичних функцій (StdFunctionRecord, StdFunctionTable), функцій користувача (UserFunctionRecord, UserFunctionTable), клас представлення спеціального оператора (SpecialOp), класи представлення адреси коду віртуальної машини (EWAdressStorage, CRelativeAddressStorage).

До блоку підтримки часу виконання аналізу відносяться: клас-контейнер інформації про поточну математичну задачу (CTaskRTI), клас-шаблон даних часу виконання аналізу математичної структури (CDSRMathModelRTI), класи даних для змінних (CDSRBaseRTVar), функцій (CDSRFunctionRTI) та рівнянь (CDSREquationRTI).

До блоку інтерпретації коду віртуальної машини відносяться: клас елементарних функцій (TRTServantReserve), клас слова коду віртуальної машини (RunWord), клас-контейнер регістрів і блоків (RunTimePointers), клас керування та інтерпретації віртуальної машини (BaseRuner).

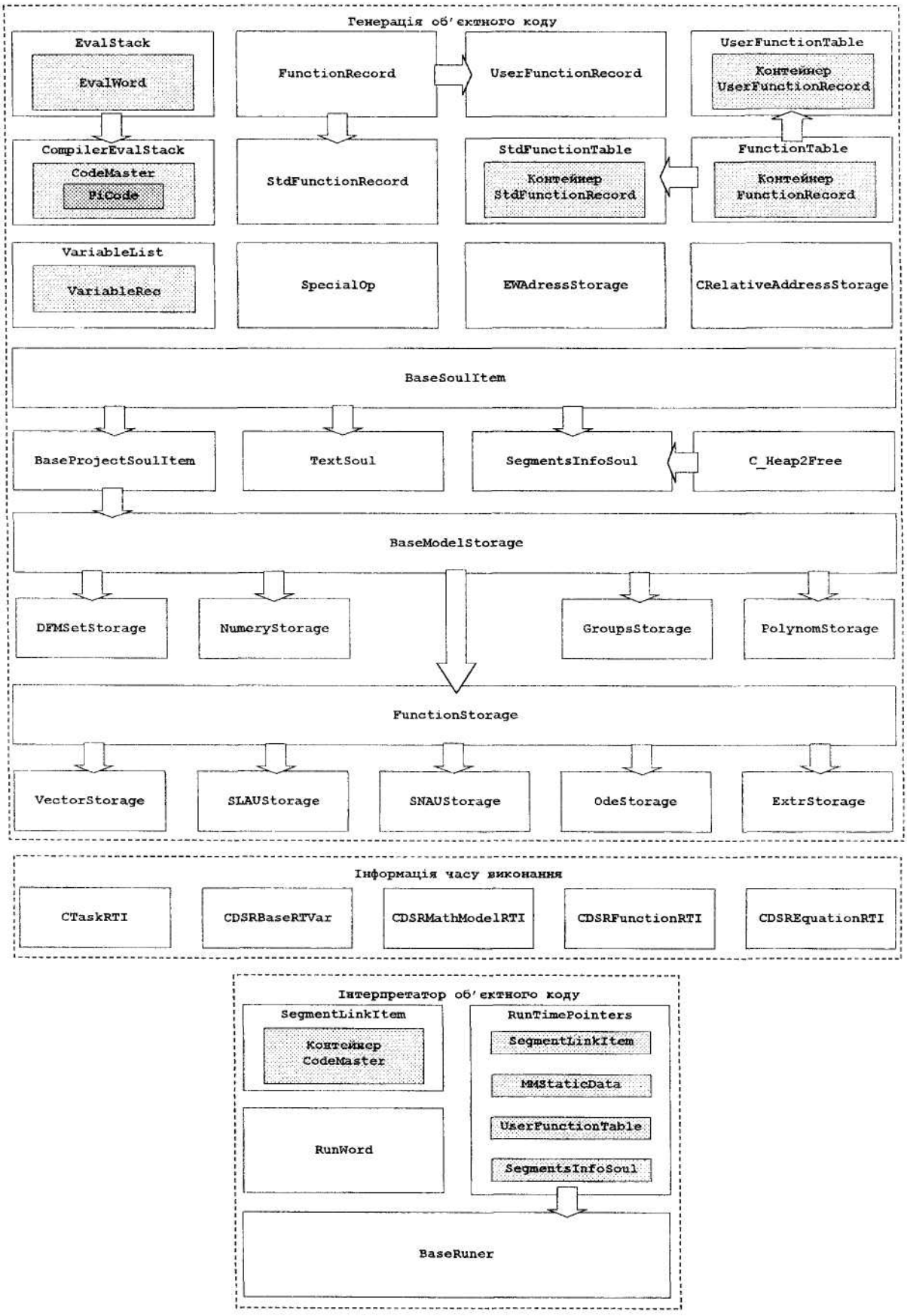

Рис. 4. Ієрархія класів генерації об'єктного коду та віртуальної машини

ЛІТЕРАТУРА:

1. *Колодницький М.М.* Тривимірна компонентна архітектура прикладної програмної системи "DSR Open Lab 1.0" як втілення концепцій реінженерії // Проблемы программирования. – 1998. – Вып. 4. – С. 37–45.
2. *Kolodnytsky M., Kovalchuk A., Kuryata S., Levitsky V.* The Mathematical Software Implementation for Computational Algebra and Number Theory // Computer Mathematics. Proceedings of the 4th Asian Symposium. Chiang Mai, Thailand, December 17–21, 2000 / Lecture Notes Series on Computing. Volume 8. – World Scientific, Singapore. – P. 291–294.
3. *Kolodnytsky M., Kovalchuk A., Kuryata S., Levitsky V.* "A Theory of Approximation of Functions" Courseware // ATCM 2000. Proceedings of the 5th Asian Technology Conference in Mathematics, December 17–21, 2000. Chiang Mai, Thailand / Published by ATCM, Inc., USA. – P. 86–95.
4. *Левицький В.Г.* Розробка мов опису задач математичного моделювання в програмному комплексі "DSR Open Lab 1.0" // Вісник ЖІТІ. – 2001. – № 15. – С. 195–202.
5. *Колодницький М.М.* Типологія математичних моделей технічних систем. Частина 2 // Вісник ЖІТІ. – 1998. – № 7. – С. 208–218.
6. *Колодницький М.М.* Елементи теорії САПР складних систем. – Житомир: ЖІТІ, 1999. – 512 с.
7. *Колодницький М.М.* Основи теорії математичного моделювання систем. – Житомир, 2001. – 700 с. (в друці).
8. *Ian Somerville.* Software engineering. – ADDISON-WESLEY, 2000.

ЛЕВИЦЬКИЙ В'ячеслав Георгійович – аспірант Житомирського інженерно-технологічного інституту.

Наукові інтереси:
- комп'ютерні інформаційні технології;
- моделювання і розв'язок задач за допомогою обчислювальної техніки;
- побудова компіляторів.

Подано 14.02.2001