УДК 681.513

ПРОЕКТУВАННЯ ПЕРЕБІРНИХ АЛГОРИТМІВ МГУА ЯК ОСНОВНИХ КОМПОНЕНТІВ ПІДСИСТЕМИ МОДЕЛЮВАННЯ

О.А. Самойленко
Міжнародний науково-навчальний центр інформаційних технологій та систем НАН України, Київ, пр-т Академіка Глушкова, 40
soa0pga@gmail.com

В статті розглядаються основні компоненти алгоритмів МГУА перебірного типу, вимоги до проектування та реалізації таких алгоритмів. Розглядаються існуючі підходи ефективного проектування програмних систем та застосування цих підходів у проектуванні алгоритмів МГУА. На основі розглянутих підходів розроблено і наведено діаграми класів, що реалізують основні компоненти перебірних алгоритмів МГУА.
Ключові слова: підсистема моделювання, МГУА, OOA, OOD, GRASP, GoF, шаблони проектування.

The paper considers the main components of GMDH combinatorial algorithms, the requirements for design and implementation of these algorithms. The main approaches of effective programming design and application of these techniques to design GMDH algorithms are considered. Based on these approaches the diagrams of classes implemented the main GMDH combinatorial algorithms components are developed.
Key words: modeling system, GMDH, OOA, OOD, GRASP, GoF, design patterns.

В статье рассматриваются основные компоненты алгоритмов МГУА переборного типа, требования к проектированию и реализации этих алгоритмов. Рассматриваются существующие подходы эффективного проектирования программных систем и использование этих подходов в проектировании алгоритмов МГУА. Основываясь на рассмотренных подходах, разработаны и представлены диаграммы классов, что реализуют основные компоненты переборных алгоритмов МГУА.
Ключевые слова: подсистема моделирования, МГУА, OOA, OOD, GRASP, GoF, шаблоны проектирования.

Вступ

В роботі [1] розглянуто кілька компонентів, з яких може складатися система інформаційної підтримки рішень. Досить важливим її компонентом є підсистема моделювання. Перебірні алгоритми МГУА складають невід'ємну частину цієї підсистеми. Ефективність розробки, супроводу та функціонування всієї системи суттєво залежить від того, на скільки ефективно спроектовані і реалізовані алгоритми підсистеми моделювання.

1. Постановка задачі

Під перебірними алгоритмами МГУА [2, 3] будемо розуміти такі алгоритми, що будують можливі структури моделей обраного класу, оцінюють коефіцієнти

для кожної структури і оцінюючи за певним критерієм кожну з побудованих таких чином моделей, відбирають найкращі з них.

Алгоритми, що перебирають всі можливі структури моделей певного класу будемо називати алгоритмами повного перебору, а алгоритми, які для визначення кращих моделей виконують частковий перебір алгоритмами неповного перебору.

Підсистема моделювання, реалізована за допомогою перебірних алгоритмів МГУА, насамперед призначена для побудови моделей на основі отриманої вибірки, і вибору з цих моделей найкращої за певним критерієм CR. Отже, на вході підсистеми маємо вибірку даних $W = (Xy)$, а на виході – модель вигляду $\hat{y} = X_0\hat{\theta}$. Тут X_0 – підматриця матриці X, що визначає набір аргументів від яких залежить значення $\hat{y}$ в побудованій моделі. Підматрицю X_0 зазвичай називають структурою моделі, а набір аргументів – структурним вектором. Кількість аргументів, що входять до складу моделі визначає складність s моделі. $\hat{\theta}$ – оцінені коефіцієнти моделі [2].

Коефіцієнти $\hat{\theta}$ оцінюються за допомогою методу найменших квадратів, відповідно до якого мінімізується похибка $Q(\theta)$ моделі. Похибка моделі визначається за формулою:

$$Q(\theta) = \|y - X\theta\|^2 = y^T y - 2\theta^T X^T y + \theta^T X^T X\theta \qquad (1)$$

Задача оцінки коефіцієнтів моделі полягає у визначенні таких коефіцієнтів, при яких похибка моделі є найменшою, тобто $\hat{\theta} = \underset{\theta \epsilon R^m}{argmin} Q(\theta)$. Для мінімізації $Q(\theta)$ візьмемо похідну по θ^T і прирівняємо її до нуля:

$$\frac{\partial Q(\theta)}{\partial \theta^T} = 0 - 2X^T y + 2X^T X\theta - 2X^T y + 2X^T X = X^T X\theta - X^T y = 0 \qquad (2)$$

В результаті отримаємо систему нормальних рівнянь:

$$X^T X\theta = X^T y \qquad (3)$$

Отже, розв’язання задачі оцінки коефіцієнтів моделі зводиться до розв’язання системи нормальних рівнянь (3) [2].

Побудовані моделі оцінюються за критерієм точності, для розрахунку якого використовується формула (1), в якій в якості θ виступають оцінені коефіцієнти за формулою (3).

Критерії селекції моделей CR поділяються на:

- зовнішні критерії;
- критерій RSS.

Критерій RSS обчислюється за формулою:

$$RSS = \left\| y - X_{ws}\hat{\theta}_{ws} \right\|^2 \quad (4)$$

За цією формулою модель складності s з коефіцієнтами, визначеними на вибірці W, оцінюється також на вибірці W за квадратом помилки моделі зі структурою X_{ws}, побудованою на вибірці W з s аргументами, відповідно до формули (1).

Зовнішні критерії зумовлюють розподіл вибірки W на дві частини (навчальну і перевірочну):

$$W = (Xy) = \begin{bmatrix} X_A & y_A \\ X_B & y_B \end{bmatrix} \quad (5)$$

Існують дві групи зовнішніх критеріїв:

- критерії групи точності (Accuracy);
- критерії групи узгодженості (Coordination).

Кожен з критеріїв може бути виражений через формулу:

$$CR_{Q|R} = \left\| y_Q - X_Q\hat{\theta}_R \right\|^2, \text{ де } Q, R = A, B, W \quad (6)$$

За цією формулою модель з коефіцієнтами оціненими на вибірці R оцінюється на вибірці Q.

Критерії точності є мірою точності моделей і прив'язані до помилки моделей.

Критерії узгодженості враховують не помилку моделей, а порівнюють дві моделі, побудовані на різних частинах вибірки.

До критеріїв точності відносяться наступні:

- *Критерій регулярності*

  Обчислюється за формулою (6) у загальному випадку, але найбільш типовою є наступна формула:

$$AR_{B|A} = \|y_B - X_B\theta_A\|^2 = y_B^T y_B - 2y_B^T X_B\hat{\theta}_A + \hat{\theta}_A^T X_B^T X_B\hat{\theta}_A \quad (7)$$

- *Симетричний критерій регулярності:*

$$AD = AR_{B|A} + AR_{A|B} \quad (8)$$

- *Критерій стабільності:*

$$AS = AR_{W|A} + AR_{W|B} \quad (9)$$

До групи критеріїв узгодженості належать такі критерії:

- *Критерій незміщеності рішень:*

$$CB = \left\| X_W\hat{\theta}_B - X_W\hat{\theta}_A \right\|^2 = (\hat{\theta}_B - \hat{\theta}_A)^T X_W^T X_W (\hat{\theta}_B - \hat{\theta}_A) \quad (10)$$

- *Критерій варіативності:*

$$CV = \left(X_W\hat{\theta}_W - X_W\hat{\theta}_A\right)^T \left(X_W\hat{\theta}_B - X_W\hat{\theta}_W\right) = (\hat{\theta}_W - \hat{\theta}_A)^T X_W^T X_W (\hat{\theta}_B - \hat{\theta}_W) \quad (11)$$

- *Критерій незміщеності помилок:*

$$BS = CS = \left| AR_{W|A} - AR_{W|B} \right| \quad (12)$$

Існує кілька видів перебірних алгоритмів, за допомогою яких можна реалізувати підсистему моделювання. Всі вони поділяються на дві групи:

- Алгоритми повного перебору
  - COMBI, COMBIS [3]
- Алгоритми послідовної селекції аргументів
  - MULTI (Коппа, Костенко), FSS, BSS, CSS [4]

Всі ці алгоритми включають наступні компоненти:

- Вибір класу моделей;
- Генератор структур;

- Оцінка коефіцієнтів моделі заданої структури;
- Оцінка моделі за обраним критерієм.

Алгоритми послідовної селекції аргументів мають ще один компонент:

- Оцінка інформативності аргументів.

Під інформативністю аргументу будемо розуміти ступінь його впливу на вихідний показник y.

В даній роботі перед нами стоїть задача ефективного проектування алгоритмів МГУА, які разом з переліченими компонентами, що входять до їх складу, представляють невід'ємну частину підсистеми моделювання.

Основною метою проектування є створення хорошої архітектури системи.

Архітектура – це опис організації і структури системи. При розробці програмних систем розглядаються декілька різних рівнів архітектури, починаючи з фізичної або апаратної архітектури і закінчуючи логічною архітектурою каркасів програмного комплексу [5].

Хорошій архітектурі притаманні наступні властивості:

- Вона є багаторівневою системою абстракцій. На кожному рівні абстракції співпрацюють одна з одною, мають чіткий інтерфейс із зовнішнім світом і ґрунтуються на так само добре продуманих засобах нижнього рівня.
- На кожному рівні інтерфейс абстракції строго відмежований від реалізації. Реалізацію можна змінювати, не зачіпаючи при цьому інтерфейс. Змінюючись внутрішньо, абстракції продовжують відповідати очікуванням зовнішніх клієнтів.
- Архітектура проста, тобто не містить нічого зайвого: загальна поведінка досягається загальними абстракціями і механізмами.

Абстракція – виділення важливих або загальних властивостей подібних понять, а також виділення в результаті важливих характеристик сутності.

2. Проектування систем. Сучасні підходи.

Під проектуванням будемо розуміти процес розробки специфікацій для реалізації системи на основі результатів аналізу і логічний опис принципів роботи системи.

В сфері сучасних інформаційних технологій для ефективного проектування і розробки програмного забезпечення як правило використовується об'єктно-орієнтований підхід [6].

Об'єктно-орієнтована розробка програмного забезпечення пов'язана з застосуванням об'єктно-орієнтованих методологій (технологій) [7]. Зазвичай ці

об’єктно-орієнтовані методології підтримуються інструментальними програмними засобами, але і без таких засобів вони корисні, так як дозволяють добре зрозуміти різні аспекти і властивості розроблюваної програмної системи, що надалі суттєво полегшує її реалізацію, тестування, супровід, розробку нових версій і більш суттєву модифікацію.

Кожна з методологій об’єктно-орієнтованого підходу базується на наступних трьох компонентах життєвого циклу:

- проведення об’єктно-орієнтованого аналізу предметної області (Object-Oriented Analysis - OOA);
- об’єктно-орієнтоване проектування (Object-Oriented Design - OOD);
- розробка програмного забезпечення з використанням об’єктно-орієнтованої мови програмування (Object-Oriented Programming OOP).

Так як для проектування системи використовуються перші два компоненти, розглянемо їх більш детальніше.

ООА – це метод ототожнення важливих сутностей реального світу для розуміння і пояснення того, як вони взаємодіють між собою. Вважається також, що ООА – це моделювання проблеми з метою формування словника предметної області, визначення об’єктів і класів. [8]

ООА – це методологія, при якій вимоги до системи сприймаються з точки зору класів і об'єктів, виявлених в предметній області. [9]

Відомі кілька підходів проведення ООА.

В книзі [] виділено три етапи ООА:

- Побудова інформаційної моделі, абстрагування реальних сутностей в термінах об’єктів і атрибутів.
- Побудова моделі станів для формалізації життєвих циклів об’єктів і відображення цієї моделі діаграмами і таблицями переходів, взаємодія між об’єктами здійснюється шляхом передачі повідомлень про події, що з ними відбуваються.
- Розробка моделі процесів, в якій дії в моделях станів розділяються на фундаментальні і багаторазово використовувані процеси.

В книзі Граді Буча [9] відмічаються альтернативні підходи до ООА:

- Метод неформального опису, в якому виділяються іменники і дієслова в описі предметної області. Іменники розглядаються як кандидати для створення класів, а дієслова – як кандидати в операції над класами.
- Структурний аналіз, при якому на основі моделі системи, представленої діаграмами потоків даних, виділяються зовнішні події і об’єкти, база даних, потік управління, перетворення потоку управління. Далі, на основі аналізу потоку даних і потоку управління, виділяються класи і методи класів.

В [5] під ООА розуміють дослідження предметної області або системи в термінах понять предметної області, таких як концептуальні класи, асоціації і зміна станів.

Об'єктно-орієнтоване проектування (OOD) - це методологія проектування, що поєднує в собі процес об'єктної декомпозиції і прийоми представлення логічної і фізичної, а також статичної і динамічної моделей проектованої системи.

OOD – визначення логіки програмного рішення в термінах програмних об’єктів, а саме – їх класів, атрибутів, методів і їх взаємодії.

Виділимо основні концепції OOD:

- Визначення об’єктів. Створення діаграм класів.
- Встановлення атрибутів.
- Використання шаблонів проектування.
- Визначення і використання каркасу додатків (application framework - інтегроване середовище, що містить бібліотеки класів і визначає структуру застосування, що розробляється).

Каркас (Framework) – набір взаємодіючих абстрактних і конкретних класів, який можна використовувати в якості шаблону для рішення групи взаємопов’язаних проблем. Каркас зазвичай доповнюється дочірніми класами з конкретною поведінкою.

Вдалий підбір і застосування відповідного каркасу чи каркасів дозволяє значно ефективніше розв’язувати задачі проектування і побудувати хорошу архітектуру проекту. Але якщо для вирішення певної задачі не можна підібрати того чи іншого каркасу, тоді на допомогу приходять шаблони проектування, які є більш універсальними в порівнянні з каркасами і також допомагають у створенні хорошої архітектури.

Шаблон (Pattern) – іменований опис проблеми, її рішення, області застосування цього рішення і способів його застосування в нових ситуаціях.

У розробці програмного забезпечення, шаблон проектування (design pattern) - повторювана архітектурна конструкція, що є вирішенням проблеми проектування у рамках деякого часто виникаючого контексту. Зазвичай шаблон не є закінченим зразком, який може бути прямо перетворений в код; це лише приклад рішення задачі, який можна використати в різних ситуаціях. Об'єктно-орієнтовані шаблони показують взаємовідносини і взаємодію між класами або об'єктами, без визначення того, які кінцеві класи або об'єкти додатка будуть використовуватись.

Розглянемо дві найбільш відомі групи шаблонів:

- GRASP
- GoF

GRASP - (General Responsibility Assignment Software Patterns) - набір шаблонів (принципів), що дозволяють вирішувати проблеми розподілу обов'язків між різними класами.

За своєю суттю, цей набір шаблонів більш абстрактний, ніж загально відомий каталог шаблонів GoF [11].

Під шаблонами проектування GoF (Gang of Four) розуміється опис взаємодії об'єктів і класів, адаптованих для рішення загальної задачі проектування в конкретному контексті.

Шаблон проектування іменує, абстрагує і ідентифікує ключові аспекти структури загального рішення, які і дозволяють застосовувати його для створення повторно використовуваного дизайну.

В книзі [11] розглянуто 23 шаблони. За призначенням шаблони можна розділити на три групи:

- Породжуючі шаблони (пов'язані з процесом створення об'єктів)
  - Factory method (фабричний метод)
  - Abstract factory (абстрактна фабрика)
  - Singleton (одинак)
  - Prototype (прототип)
  - Builder (будівельник)
- Структурні шаблони (мають відношення до композиції об'єктів і класів);
  - Adapter (адаптер)
  - Decorator (декоратор)
  - Proxy (замісник)
  - Composite (компонувальник)
  - Bridge (міст)
  - Flyweight (пристосованець)
  - Facade (фасад)
- Шаблони поведінки (характеризують як класи і об'єкти взаємодіють між собою).
  - Interpreter (інтерпретатор)
  - Template method (шаблонний метод)
  - Iterator (ітератор)
  - Command (команда)
  - Observer (спостерігач)
  - Visitor (відвідувач)

- Mediator (посередник)
- State (стан)
- Strategy (стратегія)
- Memento (хранитель)
- Chain of responsibility (ланцюг обов'язків)

Застосування цих шаблонів на практиці полегшує задачу ефективного проектування програмних систем. Системи, спроектовані за допомогою таких шаблонів, мають властивості, притаманні системам з хорошою архітектурою.

3. Проектування підсистеми моделювання, реалізованої за допомогою перебірних алгоритмів МГУА

Як було зазначено раніше, перебірні алгоритми МГУА складаються з 4 – 5 основних компонентів. Кожен із цих компонентів повинен представлятися певною обстракцією, яка має забезпечити інкапсуляцію їх реалізації і незалежність від інших компонентів системи. Для досягнення цієї задачі, а також основної мети проектування, застосуємо деякі шаблони GRASP і GoF.

Перед тим як перейти безпосередньо до проектування, проведемо деякий аналіз.

Як бачимо з формул (1) і (3), для оцінки коефіцієнтів моделі і критерію $Q(\theta)$ побудованої моделі використовуються матриці: $X^T X_{(m\times m)}$ $X^T y_{(m\times 1)}$ і $y^T y_{(1\times 1)}$, де m – кількість аргументів. Значить, для побудови моделі заданої структури X_s достатньо мати матриці: $X_s^T X_s$, $X_s^T y$ і $y^T y$. Тут s – кількість аргументів що входять до заданої структури.

Розглянемо матрицю $X^T X$. Матрицю X можна представити у вигляді:

$$X = (x_1, x_2, \ldots, x_m), \tag{13}$$

де x_i – вектор i-ого аргументу з n спостереженнями. Тоді

$$X^T X = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & \cdots & x_1^T x_m \\ x_2^T x_1 & x_2^T x_2 & & x_2^T x_m \\ \vdots & & \ddots & \vdots \\ x_m^T x_1 & x_m^T x_2 & \cdots & x_m^T x_m \end{bmatrix} \tag{14}$$

Квадратна нормальна матриця $X_s^T X_s$ є симетричною і невиродженою. Звідси випливає, що для того, щоб з матриці $X^T X$ отримати матрицю $X_s^T X_s$ треба вибрати з матриці $X^T X$ тільки ті елементи, що знаходяться на перетині тих рядочків і стовпчиків, індекси яких співпадають з індексами аргументів, що повинні ввійти до матриці $X_s^T X_s$.

Таким чином, для побудови моделей різних структур достатньо один раз побудувати матриці $X^TX_{(m\times m)}$ $X^Ty_{(m\times 1)}$ і $y^Ty_{(1\times 1)}$, і потім використовувати їх для побудови матриць $X_s^TX_s$, X_s^Ty кожної структури, шляхом виділення потрібних елементів [3].

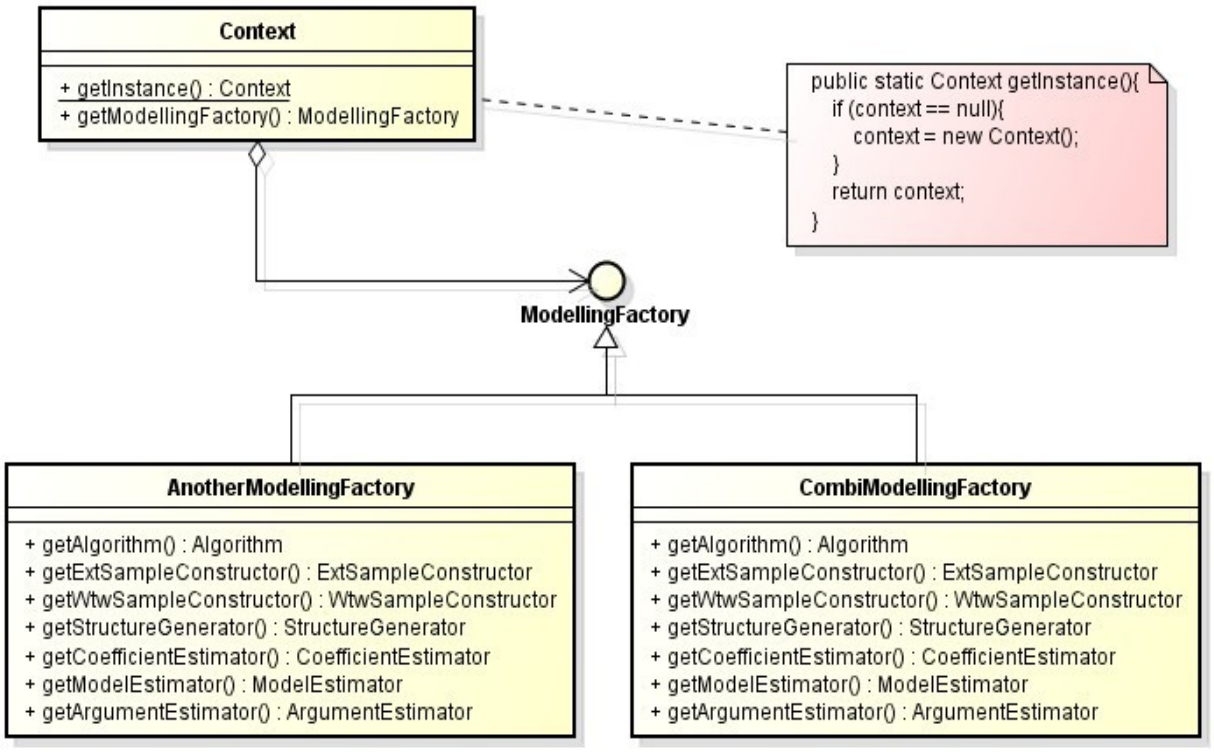

Рис. 1 Застосування фабрики і шаблону одинак для реалізації доступу до основних компонентів підсистеми моделювання.

Під генератором структур будемо розуміти генератор структурних векторів, а побудову матриць $X_s^TX_s$, X_s^Ty для кожного структурного вектору будемо виконувати в окремому компоненті. Також додамо окремий компонент для побудови матриць X^TX, X^Ty та y^Ty.

Вибір класу моделей реалізуємо за допомогою фабричного методу. Побудову моделі певного класу (лінійної, тригонометричної, степеневої) в перебірних алгоритмах МГУА можна реалізувати шляхом розширення матриці X додатковими аргументами, що є функціями обраного класу, аргументами яких є аргументи матриці X. Моделі, побудовані на вибірці з такою розширною матрицею, матимуть

клас, що застосовувався при побудові додаткових аргументів матриці і які ввійшли до моделі.

Виходячи з вище наведених міркувань, довіримо створення перелічених компонентів алгоритмів абстрактній фабриці ModellingFactory (Рис. 1). До фабрики також включимо створення самого алгоритму, так як деякі алгоритми МГУА можуть делегувати частину своєї роботи іншим алгоритмам, більш загальним. Наприклад, в основі алгоритму COMBI лежить алгоритм COMBIS. COMBIS також використовується в алгоритмах CSS, FSS та BSS. Оскільки самі компоненти можуть використовуватись не тільки в алгоритмах, а й поза їх межами в підсистемі моделювання, розмістимо створення конкретної фабрики в класі Context. Цей клас реалізуємо за допомогою шаблону Singleton. Це дозволить мати загальний доступ до фабрики як з алгоритму так і з підсистеми. Крім фабрики компонентів моделювання Singleton Context може надавати загальний доступ до конфігураційних властивостей підсистеми, що полегшить їх використання в різних компонентах.

Об’єкт класу Context як Singleton створюється за допомогою методу

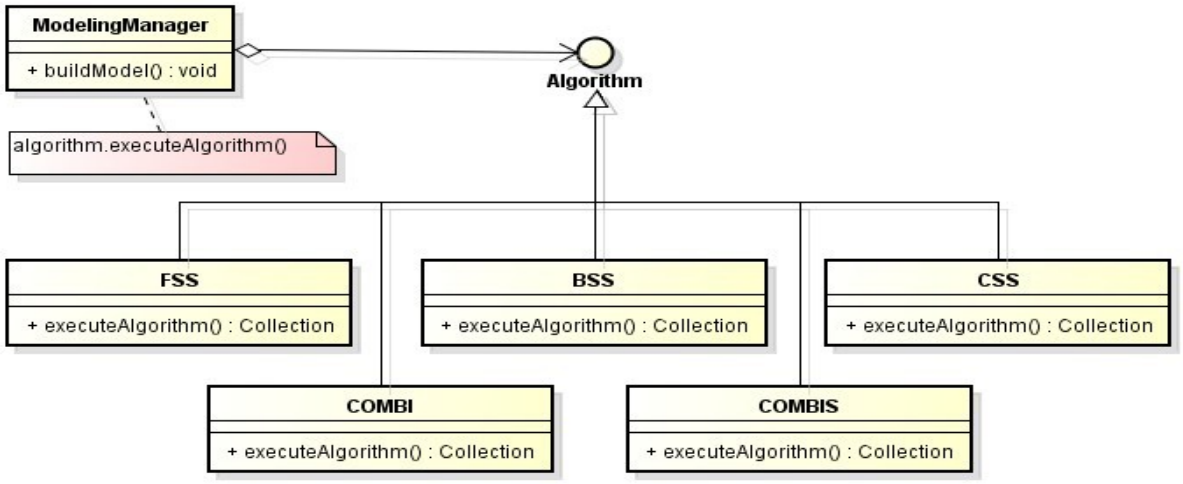

Рис. 2 Представлення реалізації алгоритмів МГУА за допомогою шаблону стратегія.

Context.getInstance(), показаного на (Рис.1). Цей метод реалізований за допомогою прийому відкладеної ініціалізації, що забезпечує створення об’єкту тільки в разі першої його потреби і не допускає повторних його створень при наступних викликах, коли вже такий об’єкт існує.

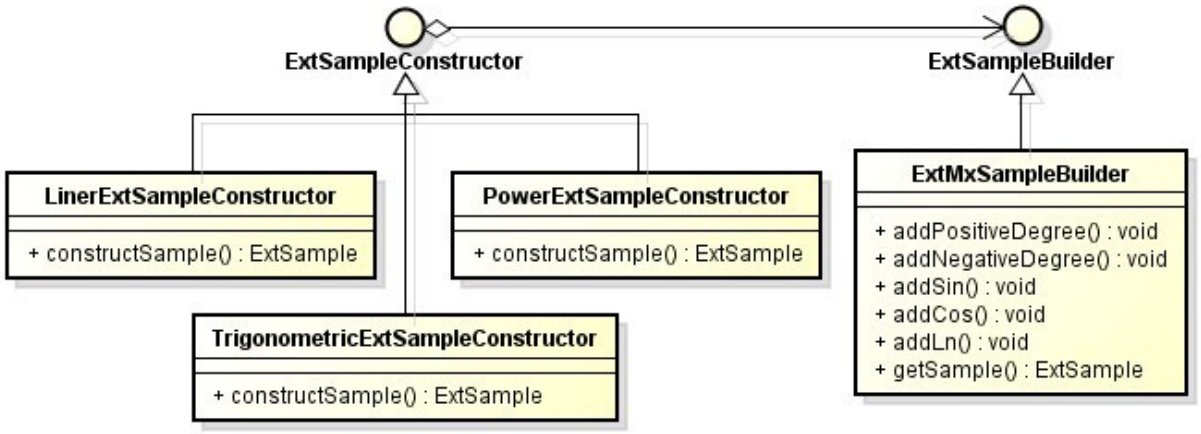

Рис. 3 Застосування шаблону Builder для проектування компоненту створення розширеної вибірки.

Класи алгоритмів можна представити у вигляді стратегії (Рис.2). Всі алгоритми мають загальний інтерфейс Algorithm. Клас ModelingManager працює з алгоритмом тільки через цей інтерфейс. Таким чином код в класі ModelingManager не залежить від конкретного алгоритму. Такий підхід дає змогу відокремити реалізацію кожного класу алгоритму від його використання і полегшує додання нових алгоритмів.

На вході кожного алгоритму маємо вибірку $W = (Xy)$, а на виході – список найкращих моделей, побудованих на цій вибірці. Вид конкретного алгоритму а також деякі інші параметри, необхідні для виконання алгоритму, можуть бути вказаними користувачем і передані за допомогою запиту чи повідомлення. Ці дані можуть бути опрацьовані фабрикою чи класом-одинаком для створення конкретного алгоритму і його компонентів. Знання про конкретні класи інкапсульовані в місці створення об’єктів, а саме в фабриці або в фабричному методі. Компоненти, що використовують ці об’єкти, співпрацюють з ними тільки через інтерфейси.

Для побудови розширеної вибірки у відповідності з обраним класом моделей доцільно скористатися шаблоном проектування Builder. Для цього створимо клас ExtSampleBuilder, в якому реалізуємо логіку генерації і додання до вибірки аргументів певного типу (Рис.3).

Класи-розпорядники (LinerExtSampleConstructor, PowerExtSampleConstructor, TrigonometricExtSampleConstructor) використовують один і той же клас-

будівельник для побудови різних видів вибірок, комбінуючи методи, представлені інтерфейсом SampleBuilder. Таким чином, спільна логіка по створенню окремих частин вибірки винесена в окремий клас-будівельник. Це дозволяє створювати нові типи вибірок за допомогою нових розпорядників без дублювання коду.

Спільний інтерфейс ExtSampleConstructor класів-розпорядників дає змогу інкапсулювати реалізацію конкретних класів і забезпечити слабку залежність між компонентами підсистеми у відповідності з шаблоном GRASP Low Coupling. В результаті, забезпечується можливість додання нової реалізації інтерфейсу ExtSampleConstructor, тобто нового класу моделей, без впливу на інші компоненти системи.

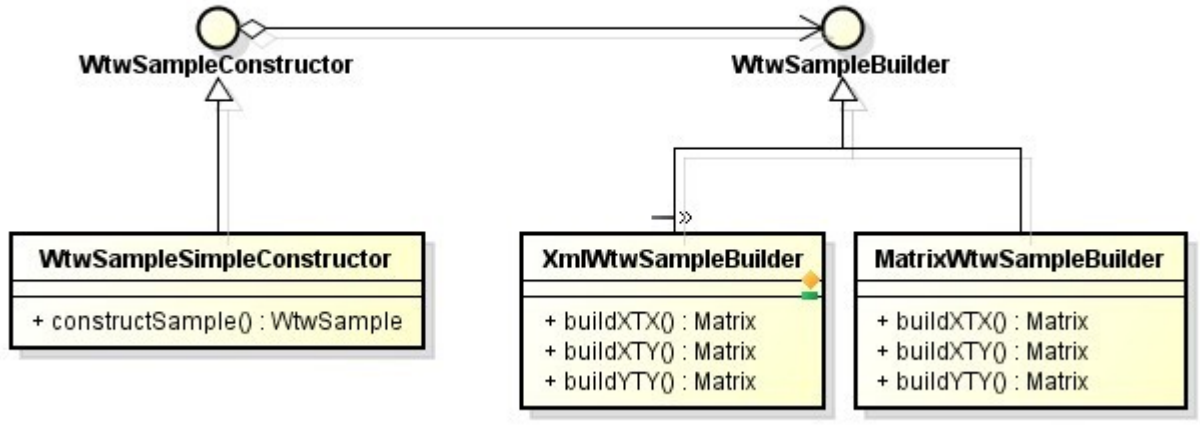

Рис. 4 Застосування шаблону Builder для проектування компоненту створення матриць X^TX, X^Ty та y^Ty

Компонент, що відповідає за конструювання квадратних нормальних матриць також краще реалізувати за допомогою шаблону Builder (Рис.4). Класи, що реалізують інтерфейс WtwSampleBuilder відповідають за побудову окремих типів квадратних матриць. Реалізація інтерфейсу WtwSampleConstructor використовує будівельник для конструювання вибірки, представленої у вигляді матриць X^TX, X^Ty та y^Ty. Розпорядник не залежить від конкретної реалізації будівельника.

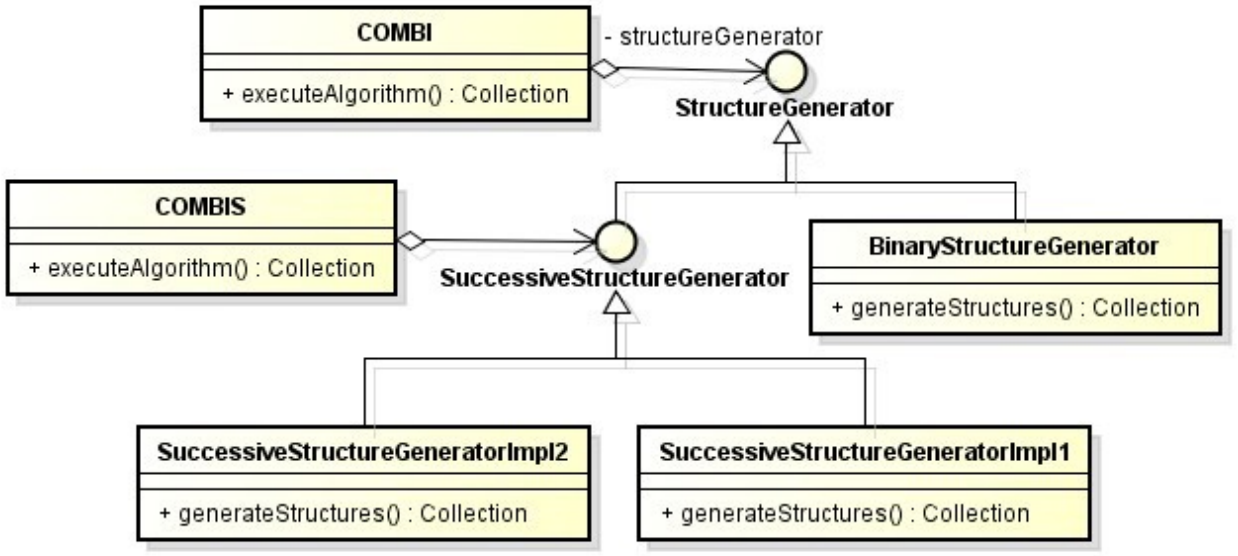

Рис. 5 Застосування шаблону Strategy для проектування генератора структур.

Перебірні алгоритми МГУА використовують зовнішні критерії оцінки моделей. Критерії такого типу потребують розбиття вибірки на кілька підвибірок (навчальну, перевірочну та екзаменаційну). Саме клас-розпорядник відповідає за конструювання цих підвибірок, він виконує розбиття і представляє будівельнику частини вибірки для побудови квадратних нормальних матриць.

Генератор структур може мати кілька реалізацій. Найбільш відомі з них це двійковий і послідовний генератори структур. Двійковий генератор відомий своєю простотою і швидкодією. Послідовний – можливістю генерувати структури заданої складності. Обидва типи генераторів мають загальний інтерфейс StructureGenerator і реалізують метод generateStructures(). В алгоритмі COMBI можна використовувати як послідовні генератори так і бінарні не залежно від реалізації. Алгоритм COMBIS і алгоритми послідовної селекції аргументів допускають використання тільки послідовних генераторів. При чому, реалізацій таких генераторів може бути кілька. У такому випадку, блок генератора структур можна реалізувати, двічі застосувавши шаблон Strategy, як показано на (Рис. 5).

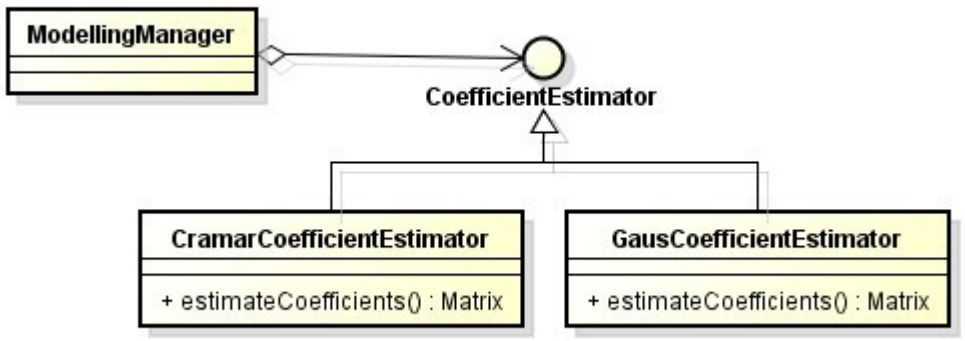

Рис. 6 Застосування шаблону Strategy для проектування компоненту оцінки коефіцієнтів

Як було зазначено раніше, оцінка коефіцієнтів зводиться до розв'язання системи лінійних рівнянь. Реалізувати розв'язання цієї системи можна кількома

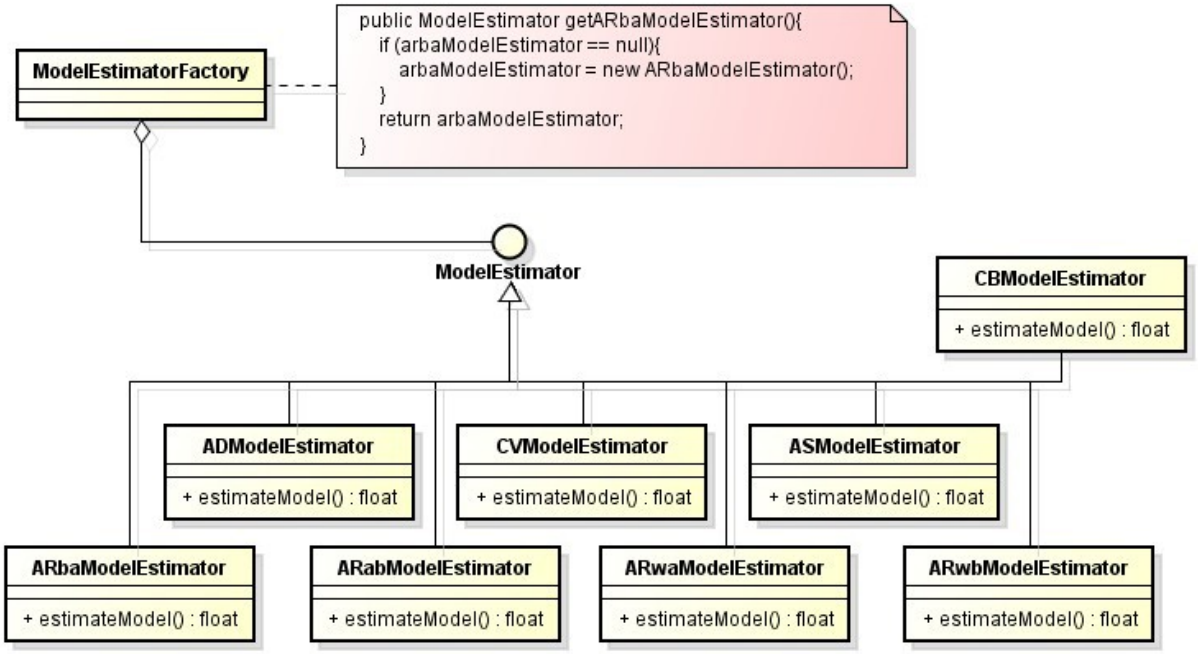

Рис. 7 Застосування шаблону Strategy для проектування компоненту оцінки моделей

способами. Отже, система повинна забезпечити різні варіанти поведінки. Така задача легко вирішується за допомогою шаблону Strategy (Рис. 6). В результаті, підсистема моделювання чи алгоритм не залежить від того, який спосіб розв'язання системи лінійних рівнянь застосовується.

Для проектування такої різноманітності критеріїв (6-12) скористаємося шаблоном Strategy (Рис.7). Критерії визначають алгоритм, за яким буде оцінюватись модель. Кожна реалізація алгоритму оцінки моделі представлена єдиним інтерфейсом ModelEstimator.

Створення конкретних класів відбувається в фабриці ModelEstimatorFactory за допомогою методів відкладеної ініціалізації, що забезпечує створення потрібного об'єкту тільки в разі необхідності і тільки один раз. В результаті ми маємо централізований доступ до всіх видів алгоритмів оцінки моделей і позбавляємося від потреби багаторазового створення об'єктів, що підвищує ефективність виконання програмного коду.

З формул (6-12) очевидно, що кожен критерій оцінки моделей залежить від

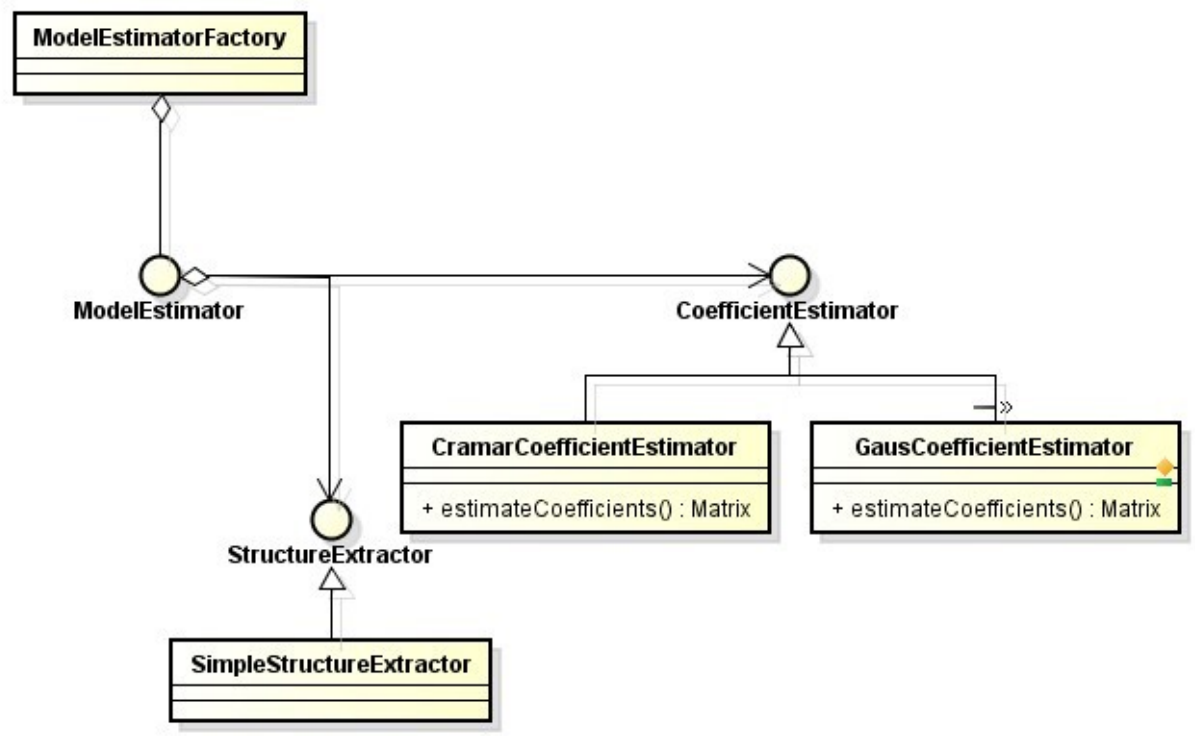

Рис. 8 Взаємодія компоненту оцінки моделей з іншими компонентами підсистеми моделювання

оцінки коефіцієнтів. Саме під час оцінки моделей за певним критерієм відомо на якій саме вибірці і які саме квадратні матриці повинні використовуватись, на якій саме вибірці повинна будуватися структура моделі. Також тут визначається яка

частина вибірки буде використана для оцінки коефіцієнтів, а яка для оцінки самої моделі. Отже, звідси випливає, що класи оцінки моделей повинні мати відношення агрегації з інтерфейсами класів, що реалізують функції оцінки коефіцієнтів і побудови структур типу $X_s^T X_s$, $X_s^T y$ за згенерованим структурним вектором на вибірках представлених у вигляді: $X^T X$, $X^T y$. Ці класи мають інтерфейси CoefficientEstimator і StructureExtractor відповідно (Рис. 8). Доступ до конкретних класів через інтерфейси забезпечує виконання принципу інкапсуляції. Завдяки чому класи-клієнти не залежать від реалізації. Таким чином, досить легко змінювати окремі компоненти без впливу на інші.

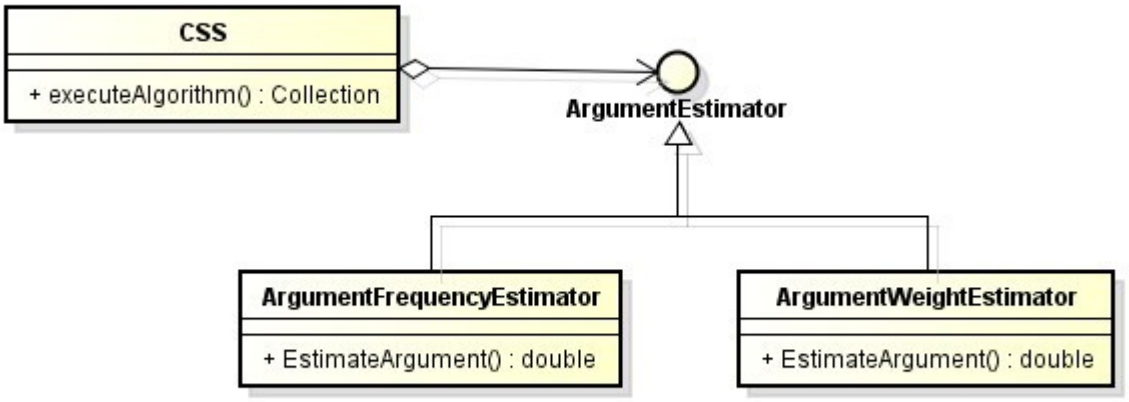

Рис. 9 Застосування шаблону Strategy для проектування компоненту оцінки аргументів

З огляду на те, що аргументи можуть оцінюватись за різними критеріями і різними способами, буде доцільним реалізувати класи оцінки аргументів за допомогою шаблону Strategy, як проілюстровано на (Рис. 9)
Розглянуті вище можливі способи проектування перебірних алгоритмів МГУА із застосуванням шаблонів GoF дозволяють побудувати досить хорошу архітектуру підсистеми моделювання.

Підсистема має кілька рівнів абстракцій:

- алгоритми МГУА;
- компоненти, з яких складаються алгоритми;
- додаткові класи нижнього рівня, що обслуговують окремі компоненти.

Кожен із цих рівнів співпрацює один з одним за допомогою інтерфейсів. На кожному рівні інтерфейс строго відмежований від реалізації. Реалізацію можна змінювати без впливу на інтерфейс. Змінюючись внутрішньо, компоненти продовжують відповідати очікуванням зовнішніх клієнтів. Архітектура досить проста. Загальна поведінка досягається загальними абстракціями і механізмами.

4 Висновки

В роботі розглянуто основні положення OOA і OOD. Розглянуто основні шаблони проектування GRASP і GoF. Для проектування підсистеми моделювання і її основних компонентів застосовано такі шаблони як: Abstract Factory, Builder, Singleton, Factory Method і Strategy. Завдяки використанню цих шаблонів спроектовано підсистему, яка відповідає основним вимогам, що висуваються до програмних систем з хорошою архітектурою.

Література

1. Самойленко О.А., Степашко В.С. Конструювання комплексної системи інформаційної підтримки управлінських рішень // Збірник праць. – Київ: МННЦ ІТС. 2009. — С. 211 - 219.
2. Ивахненко А.Г., Степашко В.С. Помехоустойчивость моделирования. – Киев: Наук. думка, 1985. – 216 с.
3. Степашко В.С. Комбинаторный алгоритм МГУА с оптимальной схемой перебора моделей // Автоматика. – 1981. – №3. – С. 31 – 36.
4. Samoilenko O., and Stepashko V. A method of Successive Elimination of Spurious Arguments for Effective Solution of the Search-Based Modelling Tasks. – Proceedings of the II International Conference on Inductive Modelling ICIM-2008, 15-19 September 2008, Kyiv, Ukraine. – Kyiv: IRTC ITS NANU, 2008. – P. 36-39
5. Ларман К. Применение UML2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ и итеративную разработку. – Киев: Вильямс. 2007. — 727 с.
6. Орлов С.А. Технологии разработки программного обеспечения: Уч. пособ. – Киев: Питер. 2003. — 473 с.
7. Гайсарян С.С. Объектно-ориентированные технологии проектирования прикладных программных систем. М.: ЦИТ. 1998.
8. Патрикеев Ю.Н. Объектно-ориентированный анализ программирование. – М.: Московский государственный университет экономики, статистики и информации. 2002.
9. Гради Буч, Объектно-ориентированное проектирование. – Киев: Диалектика и М.: И.В.К., 1992.
10. Салли Шлеер, Стефан Меллор. Объектно-ориентированный анализ: моделирование мира в состояниях. – Киев: Диалектика, 1993.
11. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – Киев: Питер. 2008. — 361 с.