УДК 681.03

К.М. Лавріщева

КОМПОНЕНТНЕ ПРОГРАМУВАННЯ. ТЕОРІЯ І РЕАЛІЗАЦІЯ

Компонентне програмування (КП) є різновидом збирального програмування, де роль елементів зборки відігріває програмний компонент чи компонент повторного використання (КПВ, reuse) й інтерфейс. Для програмування КПВ розроблено теорію моделювання предметної області (ПрО) за об'єктами, подання їх функцій компонентами й інтерфейсами з формальними анотаціями для їх збереження в бібліотеках, необхідних різним програмним системам (ПС). Математичний апарат КП – це моделі, методи, алгебра об'єднання і змінювання КПВ, алгебраїчні системи перебудови типів даних КПВ та моделі варіабельності і взаємодії СПС. Теоретичні аспекти ТП автоматизовані на веб-сайті ІТК (http://sestudy.edu-ua.net).

Вступ

Протягом багатьох років при виконанні фундаментальних проектів (1998–2011) ІПС НАНУ с.н.с. В.М. Грищенко відділу «Програмна інженерія» поступово розробив оригінальну теорію об'єктного і компонентного моделювання ПрО та концепцію її реалізації. Ці результати відображені у ряді статей [1–9] та підготовленої їм докторської дисертації (2007), захист якої не відбувся із-за тяжкої хвороби. В науковому проекті ІІІ–1–07 без нього іншими спеціалістами відділу (Г.І. Коваль, К.М. Лавріщева, О.О. Слабоспицька) продовжено розвиток і вдосконалення теорії КП в напрямку інтероперабельності й варіабельності ПС [10–14], а також створення комплексної технології вироблення ПС, в якої об'єднані на одній концептуальній основі теоретичний апарат об'єктно-орієнтованого аналізу та компонентного програмування ПС із КПВ, реалізованого спектром окремих спрощених ліній: розробка окремих КПВ, збирання КПВ і забезпечення взаємодії різномовних компонентів на основі апарата алгебраїчних систем з еквівалентним перетворенням типів даних до потрібних форматів середовища виконання, варіабельності (варіантності) структури ПС шляхом конфігурування КПВ і їх змінювання тощо [13]. Ці лінії реалізовані в інструментально-технологічному комплексі (ІТК) ІПС НАНУ під керівництвом автора силами трьох студентів КНУ імені Тараса Шевченка і МФТІ (2010–2011) [14].

В даній роботі подані базові положення удосконаленої вітчизняної теорії і практики КП. Разом вони спрощують та підвищують продуктивність розробки з КПВ ПС, сімейств систем, інформаційних систем, веб-застосувань тощо, а також зменшують їх вартість. Теорія і реалізація ставлять КП на рівень промислових технологій програмування для сучасних фабрик програм [15].

Математичне моделювання об'єктної моделі ПрО

Побудова моделі ПрО починається з її декомпозиції для виявлення об'єктів, визначення їх функцій і властивостей засобами логіко-математичного моделювання цієї області [2, 8, 9]. Будується множина об'єктів ПрО та підмножина предикатів певної сигнатури, що відображають загальні та особисті властивості об'єктів.

Об'єкт ПрО – іменована частина дійсної реальності з певним рівнем абстракції відносно неї, є понятійною структурою за трикутником Фреге (денотат, знак, концепт) [16]. Кожний об'єкт (О) як сутність ПрО з множини об'єктів це $O_i = O_i(Na_i, Den_i, Con_i)$, де Na_i, Den_i, Con_i відповідно – знак (ім'я), денотат та концепт об'єкта [1, 2].

Аксіома 1. Предметна область, що моделюється з об'єктів, сама є об'єктом.

Аксіома 2. Предметна область, що моделюється, може бути окремим об'єктом у складі іншої предметної області.

При моделюванні об'єкт ПрО принаймні отримує хоча б одну властивість або характеристику, семантику й унікальну ідентифікацію в множині об'єктів ПрО та підмножини предикатів властивостей об'єктів. Вони (*0-арні, унарні, бінарні)* породжуються за умовами:

– призначення та кількості предикатів є достатніми для концептуального моделювання ПрО із множини об'єктів ПрО;

– кожен предикат, його тип та сигнатура мають відображати суб'єктивне сприйняття сутності об'єкта аналітиком і виконавцем ПрО.

Властивість об'єкта визначається на множині об'єктів ПрО унарним предикатом, що приймає значення істини за його зовнішніми і внутрішніми властивостями моделі ПрО. Множина предикатів моделі ПрО може бути довільним і вводяться за характеристиками.

Характеристика – це сукупність властивостей (унарних предикатів), що є підмножиною множини, виділених у системі предикатів за умов приймання значень істини одночасно не більш, ніж один предикат з сукупності характеристик зовнішнього і внутрішнього типу із області значень властивості за їх істинною.

Відношення завдається бінарним предикатом на множині об'єктів ПрО, приймає значення істини на поданій парі об'єктів. До основних типів взаємовідношень належать:

1) множина – множина;

2) елемент множини – елемент множини;

3) елемент множини – множина;

4) множина – елемент множини.

Цим типам відношень відповідають відомі операції: *узагальнення, спеціалізації, агрегації, асоціаці, деталізації, класифікації та екземплярізації.* Тип 3), 4) має *роде – видове* відношення (*IS–A*) та відношення *частина – ціле* (*PART–OF*). Ці відношення використовуються для проведення структурної впорядкованості пар об'єктів.

Проектування моделі ПрО. Виконується за чотирма рівнями: *узагальнюючий* для визначення базових понять ПрО без урахування їх сутності та властивостей; *структурний* для впорядкування об'єктів у структурі моделі ПрО з відношеннями між ними; *характеристичний* для формування концептів об'єктів на базі їх властивостей та характеристик; *поведінковий* для визначення поведінки об'єктів у залежності від подій.

У відповідності *з узагальнюючим* рівнем об'єкт розглядається як математичне поняття класу згідно з положеннями аксіоматичної теорії множин Геделя – Бернайса.

Над множинами виконуються базові операції: об'єднання множин $A \cup B$, перетину множин $A \cap B$, різниці множин A / B, доповнення множини A новим елементом, булеан множини $A \subseteq B$, потужність $|A|$, приналежність $A \in B$ тощо.

Довільна множини A сигнатури булевої алгебри множини B(A) над A може включати:

– сукупність констант ∅, як порожня підмножина;

– сорти σ, ρ, λ для подання базової множини A, булеану 2^A і множини істинністих значень Bool ={1, 0, ∗};

– предикатні символи $\subset \rho \times \rho \rightarrow$ Bool та $\in \sigma \times \rho \rightarrow$ Bool інтерпретуються як предикати включення й приналежності за операціями об'єднання, перетину, різниці й доповнення множин;

– операції комутативності $A \cup B = B \cup A$, $A \cap B = B \cap A$, асоціативності об'єднання й перетину множин $A \cup (B \cup C)=(A \cup B) \cup C$, $A \cap (B \cap C)=(A \cap B) \cap C$.

Виходячи з цієї теорії, якщо $O = (O_0, O_1, \ldots, O_n)$ – множина об'єктів ПрО і O_0 відповідає самій ПрО, то для елементів *O* виконується відношення:

$$\forall i[(i>0) \,\&\, (O_i \in O_0)]. \qquad (1)$$

У відповідності зі *структурним* рівнем проектування кожен з об'єктів подається як множина або елемент певної множини. У цьому випадку вираз (1) трансформується у такий вигляд:

$$\forall i \; \exists j[(i>0)\&(j\geq 0)\& \; (i \neq j)\& \; (O_i \in O_j)].$$

Тут кожен об'єкт (крім O_0) є множиною або елементом певної множини і до них застосовуються операції теоретико-множинної алгебри. Цей вираз визначає відношення "частина-ціле", екземпляризація, агрегація.

У відповідності *з характеристичним* рівнем для кожного з об'єктів формується відповідний концепт. Якщо $O' = (O_1, O_2, ..., O_n)$ – сукупність об'єктів ПрО, а $P' = (P_1, P_2, \ldots, P_r)$ – множина унарних предикатів, що пов'язані з властивостями об'єктів ПрО, то концепт об'єкта O_i є множиною тверджень, побудованих на основі предикатів з P', що приймають значення істини для відповідного об'єкта. Тобто концепт $Con_i = \{P_{ik}\}$ за умови $P_k(O_i) = true$, де P_{ik} є твердженням для об'єкта O_i відповідно предиката P_k. Згідно зі структурою концептів об'єктів встановлюються відношення типу "рід–вид".

За структурним рівнем визначаються такі поняття як: клас, екземпляр класу, абстрактний клас та ін. Визначення термінів властивостей об'єктів та їх відношень (агрегація, деталізація, класифікація тощо) виконується на характеристичному рівні, коли визначаються терміни: атрибут стану, стан, простір станів, метод та ін.

Згідно з *поведінковим аспектом* визначаються послідовність станів об'єктів та їхні процеси з відображенням переходів станів. Взаємозв'язки між об'єктами формуються на основі бінарних предикатів, які пов'язані з властивостями об'єктів ПрО і деталізуються до рівня взаємозв'язків між станами об'єктів.

Головним завданням усіх рівнів є опис зовнішніх та внутрішніх характеристик за раніш наведеними типами взаємовідношень.

Визначення понять на рівнях виконується за *теоретико-множинною концепцією*. В ній при першому рівні проектування виявляються об'єкти ПрО, які структурно впорядковуються за допомогою бінарного відношення належності.

Поняття класу об'єктів замінюється поняттям множини. Якщо об'єкт є елементом іншого об'єкта, то він визначається множиною. При цьому не кожен об'єкт є елементом будь якого іншого об'єкта (класу). Наприклад, об'єкт, що відповідає усій ПрО для певної об'єктної моделі (ОМ), не є елементом будь якого іншого об'єкта у цієї ОМ. Визначення об'єкта формулюється за умовою: кожен об'єкт обов'язково є множиною або елементом деякої множини.

Впорядкування об'єкта виконується з урахуванням відношення належності, впорядкування елементів множини через числову множини (наприклад, множина природних чисел). Об'єкт вирізняється з множини всіх елементів завдяки побудованого відображення.

Над об'єктами множин за даною концепцією виконуються звичайні операції: об'єднання, перехрещення, різниці, додатку, симетричної різниці, декартового добутку. Конкретизаціями поняття об'єкта у даній концепції є *клас* – це об'єкт, який подає собою множину; *екземпляр класу* – об'єкт, що є елементом певної множини, яка сама є класом; *об'єднаний клас* – множина, що є прямою сумою декількох інших множин; *клас-перехрещення* – це множина, яка є загальною частиною інших множин; *агрегований клас* – це множина, яка є підмножиною певного декартового добутку декількох інших множин.

При першому рівні проектування моделі ПрО формується множина базових функцій, яка пов'язана з декомпозиційними або композиційними змінами денотатів та концептів об'єктів. Вона складається з ряду функцій, які охоплюють трансформації денотатів та концептів у процесі об'єктного аналізу і до складу яких входять зміни, що пов'язані зі збільшенням або зменшенням кількості об'єктів (деталізація, екземпляризація, агрегація та ін.) та розширення або звужування концептів об'єктів. Проте ці зміни підпорядковуються певним правилами та умовами і забезпечують коректність виконання функцій.

Створення певної моделі ПрО має ітеративний характер і починається з визначення самої ПрО як початкового

об’єкта. На кожній ітерації у відповідності до потреб моделювання застосовуються функції аналізу, які наближають структуру та властивості об’єктів ПрО до побудови кінцевій моделі ПрО із урахуванням механізмів кожного рівня абстрагування відповідного математичного апарата.

Побудовано алгебру об’єктного аналізу $\sum=(O',\Psi)$, де $O'=(O_1, O_2,...,O_n)$ – множина об’єктів, а Ψ = {*decds, decdn, comds*, *comdn, conexp, connar*} – множина операцій над елементами O'. Кожна з операцій має певний пріоритет та арність, а також пов’язана з відповідними допустимими змінами денотатів та концептів.

Теорема 1. Множина операцій Ψ алгебри $\sum$ є повною системою операцій щодо функцій об’єктного проектування.

Графове подання моделі ПрО. Результатом структурної упорядкованості об'єктів моделі *ОМ* є об'єктний граф $G=\{O,R\}$, визначений на множині об'єктів O та зв'язків R (relations) за вимогами:

– множина вершин O подає взаємооднозначність відображення об'єктів ПрО;

– для кожної вершини повинен існувати хоча б один зв'язок, що належить множині відношень-зв'язків R;

– існує хоча б одна вершина, що має статус множина-об'єкт і відображає ПрО в цілому.

Зв'язки можуть бути*:* один до одного, один до багатьох, багато до багатьох.

Побудований граф $G=\{O,R\}$ може доповнюватися інтерфейсними об'єктами, потім структурно впорядковується (нагору) з контролем повноти і надмірності елементів графа й усунення дублюючих елементів.

Граф G і множина об'єктів ПрО, що відрізняються один від одного за статусом (елемент, множина або множина елементів) і взаємним порядком утворять *ОМ*. Тобто об'єкти *ОМ* подаються загальними й індивідуальними властивостями та зовнішніми і внутрішніми характеристиками.

В об’єктної моделі ПрО розрізнюються функціональні та інтерфейсні об’єкти. Перевірка властивостей об'єктів проводиться за допомогою операцій (екземпляризації, класифікації, спеціалізації, агрегації та ін.) шляхом попарного порівняння властивостей внутрішніх характеристик об'єкта з множиною властивостей зовнішніх характеристик. Властивості рахуються перевіреними, якщо виконується умова, що кожній внутрішній властивості множини об’єкта відповідає еквівалентна йому зовнішня властивість об'єкта-елемента. Якщо ця умова не виконується, то такий елемент віддалиться зі списку елементів множини та з графа відповідно.

На змістовному рівні множина O являє собою набір методів реалізації віддалених об'єктів ПрО, для кожного з них існує інтерфейсний елемент множини In (типу stub, skeleton). Метод реалізації об'єкта відповідає вершині графа, перехід до якої потребує перетворення даних, що ініціюються повідомленнями і зв'язками.

Теорія компонентного програмування

Основні поняття. Сутність КП полягає у створенні ПС, сімейств систем з базових елементів – програмних компонентів та КПВ. Головна мета КП – створення компонентних програм і ПС із КПВ шляхом застосування концепції їх збирання відповідно їх інтерфейсів з властивостями і характеристиками, накопиченими в репозиторіях (інтерфейсів і реалізацій) компонентного середовища [9, 17, 18].

Під *програмним компонентом* чи просто компонентом розуміється незалежний від МП, самостійно програмний об'єкт, який забезпечує виконання певної сукупності прикладних задач (сервісів), доступ до яких можливий тільки за інтерфейсами, які відображають функціональні можливості компонента, параметри і порядок звертання до них чи інших.

Компонент як елемент (відображення) типового рішення для ПС і подальшої композиції з іншими потрібен мати типову архітектуру, структуру, характеристики і атрибути подані в його інтерфейсній частині для обміну даними і взаємодії компонентів у різних середовищах. Тобто поданий таким чином компонент, стає неподільним та інкапсульованим об’єктом,

який задовольняє функціональним вимогам, а також вимогам щодо архітектури системи і середовища взаємодії компонентній програми.

Компонентна програма чи ПС із компонентів – це сукупність компонентів, що реалізують функціональні та нефункціональні вимоги до неї, будується за правилами компонентних конфігурацій збирального типу для взаємодії у рамках компонентних моделей.

Формально визначена модель ПрО є джерелом трансформаційного переходу від об'єктів до компонентів з використанням формального математичного апарата моделювання: моделей компонента й інтерфейсу, компонентної програми, компонентного середовища, зовнішній і внутрішній компонентної алгебри побудови ПС та алгебраїчні системи перетворення типів даних компонентів на сигнатури операцій для завдання їх взаємодії [9]. Цей апарат слугує формальної побудови компонентних ПС з готових КПВ із різних бібліотек зі значним скороченням витрат і поліпшенням якості майбутнього продукту.

Модель компонента є наслідком узагальнених типових рішень щодо сутності відповідного об'єкта, метод якого перетворюється до компонента за його архітектурою, структурою, властивостями, характеристиками і має вигляд:

$$C = \left(CNa,\ CIn,\ CFa,\ C\,\mathrm{Im},\ CSe\right), \quad (2)$$

де CNa – унікальне ім'я компонента, $CIn = \left\{CIn^i\right\}$ – множина інтерфейсів, пов'язаних з компонентом, CFa – інтерфейс забезпечення функцій керування екземплярами компонента, $C\,\mathrm{Im} = \left\{C\,\mathrm{Im}^{\,j}\right\}$ – множина реалізацій компонента, $CSe = \left\{CSe^r\right\}$ – множина загальносистемних сервісів.

Множина $CIn = \left\{\, CIn\,In \cup\ CIn\,Out \,\right\}$ складається з вхідних $CInIn$ та вихідних $CInOut$ інтерфейсів. Тобто компонент має вхідні інтерфейси при власної реалізації, а вихідні інтерфейси реалізації іншого компонента. Кожен з них має відповідну модель

$$CIn^i = \left(In\,Na^i,\ In\,Fu^i,\ In\,Sp^i\ \right), \quad (3)$$

де $In\,Na^i$ – ім'я інтерфейсу, $In\,Fu^i$ – функціональність (сукупність методів), $In\,Sp^i$ – специфікація інтерфейсу: опис типів, констант, інших елементів сигнатур методів, характеристик тощо.

Інтерфейс CFa визначає методи і звернення до екземплярів компонентів (пошук, вибір, знищення тощо).

Кожна реалізація $C\,\mathrm{Im}^{\,j} \in C\,\mathrm{Im}$ задається моделлю:

$$C\,\mathrm{Im}^{j} = \left(\mathrm{Im}\,Na^{j},\ \mathrm{Im}\,Fu^{j},\ \mathrm{Im}\,Sp^{j}\ \right), \quad \textbf{(4)}$$

де $\mathrm{Im}\,Na^{j}$ – ідентифікатор або ім'я реалізації, $\mathrm{Im}\,Fu^{j}$ – реалізована функціональність, $\mathrm{Im}\,Sp^{j}$ – специфікація опису (умови виконання, залежності від певних платформ тощо).

Необхідною вимогою існування компонента є умова його цілісності:

$$\forall\, CIn^i \in CIn\ \exists$$
$$\exists C\,\mathrm{Im}^{j} \in C\,\mathrm{Im}\ \left[\,\mathrm{Pr}\left(CIn^i\right) \subseteq C\,\mathrm{Im}^{j}\,\right], \quad (5)$$

де $\mathrm{Pr}\left(\mathrm{Pr}\,ovide\right)CIn^i$ означає функціональність з реалізації методів інтерфейсу CIn.

Аксіома 3. Для об'єднання двох різнорідних компонентів C_1 та C_2 існує умова, якщо $CIn^i{}_1 \in CIn_1$, то повинен існувати $CIn^k{}_2 \in CIn_2$ такий, що

$$S\left(CIn^i{}_1\right) = S\left(CIn^k{}_2\right)\ \&\ \mathrm{Pr}\left(CIn^i{}_1\right) \subseteq C\,\mathrm{Im}_2^{j},$$

де $S(sign)\ldots)$ означає сигнатуру відповідного інтерфейсу.

Модель компонентної ПС. Ця модель ПС (*PS*) ідентична моделі програмного компонента з той різніцією, що її елементами можуть бути самі програми, як самостійні компоненти.

$PS = \left\{ PSLm\left\{ Lm^1, ..., Lm^n \right\}, R\left\{ Ri, ..., Rm \right\}, \right.$

$\left. PSLn\left\{ Ln^1, ..., Ln^k \right\} \right.$,

де $PSLm\left\{ Lm^1, ..., Lm^n \right\}$ – множина реалізації компонента, КПВ, ПС;

$R\left\{ Ri, ..., Rm \right\}$ – множина предикатів;

$PSLn\left\{ Ln^1, ..., Ln^k \right\}$ – множина інтерфейсів компонентів і програм.

Операції множини *R* можуть відповідати об'єднанню чи конфігурації КПВ у деякі складні структури програм і ПС з множини компонентів, інтерфейсів.

Компоненти чи КПВ моделі компонента чи ПС можуть змінюватися, замінюватися новими функціонально подібними, еквівалентними чи тотожними КПВ з метою отримання нових варіантів продукту ПП за аксіомами.

Аксіома 4. Два компоненти C_1 та C_2 є тотожними (рівними), якщо тотожними є їх відповідні складові. Як наслідок, заміна C_1 на C_2 не впливає на компонентну програму, до якої належить C_1.

Аксіома 5. Два компоненти C_1 та C_2 є еквівалентними, якщо тотожними є їх множини інтерфейсів та реалізацій. Заміна C_1 на C_2 не змінює функціональності компонентної програми за умови встановлення відповідності між іменами у самій програмі.

Аксіома 6. Два компоненти C_1 та C_2 є подібними, якщо тотожними є їх множини інтерфейсів. Заміна C_1 на C_2 зберігає взаємозв'язки компонентів, але функціональність компонентної програми може змінитись.

Головна мета КП це зв'язування компонентів, а також змінювання їх тотожними чи еквівалентними в структурі ПС. Зв'язування має свою арність і може складатися з множини даних, які входять до класу інтерфейсів означеного класу КПВ. Ця операція виконується за моделлю компонента (2) і інтерфейсу (3). Модель компонента й інтерфейсу завдають операції і предикати, які визначають умову передачі даних іншому компоненту.

Між компонентами можуть існувати відношення: *успадкування, екземпляризаці, контракту, об'єднання (зв'язування).* Усі відношення визначаються на інтерфейсах і предикатах, мають такий зміст.

Відношення екземпляризації. Вираз $CIns_k^{ij} = \left(IIns_k^{ij},\ InFu^i, \operatorname{Im} Fu^j \right)$ описує певний екземпляр компонента *C*, де: $IIns_k^{ij}$ – унікальний ідентифікатор екземпляра, $InFu^i$ – функціональність інтерфейсу $CIn^i \in CIn$, $\operatorname{Im} Fu^j$ – програмний елемент, що забезпечує виконання реалізації $C\operatorname{Im}^i \in C\operatorname{Im}$.

Відношення контракту. Між компонентами C_1 і C_2 існує відношення контракту $Cont_{12}^{im}$ = (CIn_1^i, CIn_2^m, $IMap_{12}^{im}$), де: $CIn_1^i \in CIn_1$ – вихідний інтерфейс першого компонента, $CIn_2^m \in CIn_2$ – вхідний інтерфейс другого компонента, $IMap_{12}^{im}$ – відображення відповідності між методами, які входять до складу обох інтерфейсів з урахуванням сигнатур та типів даних, що передаються; якщо компонент C_2 має реалізацію для інтерфейсу CIn_2^m, яка забезпечує виконання функціональності $InFu_1^i$ інтерфейсу CIn_1^i.

Відношення зв'язування. Якщо між компонентами C_1 і C_2 існує відношення контракту $Cont_{12}^{im}$, то між їх екземплярами $CIns_{1k}^{ij}$ = ($IIns_{1k}^{ij}$, $InFu^i$, $ImFu^j$) та $CIns_{2p}^{mq}$ = ($IIns_{2p}^{mq}$, $InFu^m$, $ImFu^q$) існує відношення зв'язування відносно контракту $Cont_{12}^{im}$, яке описується у вигляді $Bind(IIns_{1k}^{ij}, IIns_{2p}^{mq}, Cont_{12}^{im})$.

Відношення взаємодії. Якщо компоненти ПС розташовані у розподіленому середовищі і вони входять до множини компонентів репозиторію ПС, то використовується модель взаємодії, визначеної на множині інтерфейсів готових КПВ та віддалених викликів [10, 11].

Об'єднання компонентів у ПС. Модель ПС еквівалентна об'єктному графу, елементи якого мають встановлені при моделюванні властивості й характеристики, які використовуються при завданні функції інтерфейсу типу інтерфейсу посереднику (stub, skeleton).

Для визначення семантики об'єднання об'єктів за графом G, використовуються мова опису інтерфейсу об'єктів в IDL (параметрів *In*, *Out*) і операції приналежності:

$O_k \in O, In(O_k)$ – множина вхідних (*In*) інтерфейсних об'єктів;

$O_k \in O, Out(O_k)$ – множина вихідних (*Out*) інтерфейсних об'єктів.

Результатом *об'єднання* двох об'єктів буде компонентний об'єкт, у якого множина вхідних інтерфейсів збігається з множиною вхідних інтерфейсів об'єкта-приймальника, а множина вихідних інтерфейсів з множиною вихідних інтерфейсів об'єкта-передавача:

$$O_k = \left(Out(O_k),\ In(O_k) \right)$$

$$O_l = \left(Out(O_l),\ In(O_l) \right),$$

$$O_k \cdot O_l = \left(Out(O_k),\ In(O_l) \right).$$

Аксіома 7. Композиція об'єктів $O_k \cdot O_l$ є коректною, якщо об'єкт-передавач повністю забезпечує сервіс, необхідний об'єкту-приймальнику, тобто

$$\forall I_m \in In(O_k) \Rightarrow \exists I_n \in Out(O_l) \wedge I_m = I_n .$$

Компонентні об'єкти можуть мати декілька інтерфейсів, які можуть успадковувати інтерфейси інших об'єктів ($O_k \leftarrow O_l$), тоді останні надають сервіс всієї множини вихідних інтерфейсів :

$$O_k \leftarrow O_l \Rightarrow Out(O_k) \subseteq Out(O_l).$$

У випадку, коли об'єкт успадковує інший об'єкт, у якого множина вихідних інтерфейсів містить всі його інтерфейси, а множина вхідних інтерфейсів містить тільки інтерфейси, необхідні для надання сервісу, то маємо

$$O_k \leftarrow O_l = \left(\begin{array}{l} Out(O_k) \cup Out(O_l), \\ \left\{ I_m : \left(I_m \in In(O_k) \cup In(O_l) \right) \wedge \exists I_n \right. \\ \left. \in Out(O_k \leftarrow O_l) : exec(I_n, I_m) \right\} \end{array} \right).$$

Успадкований об'єкт делегує усі інтерфейси і має такі властивості:

транзитивності

$$\forall O_{1,2,3} \in O : O_1 \leftarrow O_2, O_2 \leftarrow O_3 \Rightarrow O_1 \leftarrow O_3,$$

симетричності

$$\forall O_k \in O \Rightarrow O_k \leftarrow O_k \ .$$

Ці властивості об'єктів ідентичні компонентам моделі ПС. Для їх розміщення у деякому середовищі необхідно виконати такі дії:

– визначити критерії й умови розташування компонентів у вузлах середовища з урахуванням зв'язків між ними відповідно об'єктного графа;

– визначити засоби передачі повідомлень від одного компонента до іншого;

– об'єднати програмні об'єкти і їхні інтерфейси у каркас чи структуру ПС .

Таким чином, виходячи з запропонованої моделі ПС, моделі взаємодії [11, 18] та семантики її виконання, компоненти прикладної моделі ПС, можуть бути розподілені по вузлах мережі і взаємодіяти між собою через механізми інтерфейси і повідомлення.

Модель компонентного середовища має наступний вираз:

$$CE = (CNa, InRep, ImRep, CSe, CSeIm), \quad (6)$$

де $CNa = \left\{ CNa^m \right\}$ – множина імен компонентів, які входять до складу середовища, $InRep = \left\{ InRep^i \right\}$ – репозиторій інтерфейсів компонентів середовища, $ImRep = \left\{ ImRep^j \right\}$ – репозиторій реалізацій, – інтерфейс системних сервісів, $CSeIm = \left\{ CSeIm^r \right\}$ – множина реалізацій для системних сервісів.

Кожен елемент з *InRep* це двійка $\left(CIn^i, CNa^m \right)$, де CIn^i – інтерфейс компонента, CNa^m – ім'я компонента, для якого

існує інтерфейс. Аналогічно кожен елемент з *ImRep* описується $\left(CIm^j, CNa^m\right)$, де CIm^j – реалізація, яка описується виразом (4), а CNa^m – ім'я компонента, до якого належить ця реалізація.

Компонентне середовище на рівні моделі розглядається як множина серверів застосувань, де розгортаються компоненти-контейнери, екземпляри яких забезпечують реалізацію функціональності компонента. Взаємозв'язок контейнера з сервером забезпечується через стандартизовані інтерфейси (*CFa*). Зв'язок між КПВ, які розгорнуті у різних серверах забезпечується реалізаціями інтерфейсу *CSe*.

Означення 1. Каркасом компонентного середовища називається середовище, для якого *CNa, InRep, ImRep* – суть порожні множини, тобто:

$$FW = \left(\varnothing, \varnothing, \varnothing, CSe, CSeIm\right). \qquad (7)$$

Нехай $FW_1 = \left(\varnothing, \varnothing, \varnothing, CSe_1, CSeIm_1\right)$ і $FW_2 = \left(\varnothing, \varnothing, \varnothing, CSe_2, CSeIm_2\right)$ – два каркаси.

Означення 2. Каркас FW_1 сумісний з каркасом FW_2, якщо існує відображення $SMap: CSe_1 \to CSe_2$ таке, що $SMap\left(CSe_1\right) \subseteq CSe_2$.

Зовнішня компонентна алгебра визначає множину операцій над множинами компонентів та компонентних середовищ і описується наступним виразом:

$$\Psi = \left\{CSet, CESet, \Omega\right\}, \qquad (8)$$

де *CSet* – множина компонентів, кожен з яких описується виразом (3), *CESet* – множина компонентних середовищ, кожне з яких описується виразом (7), Ω – множина операцій.

До складу множини входять операції:

– інсталяції (розгортання компонента) $CE_2 = C \oplus CE_1$;

– об'єднання компонентних середовищ $CE_3 = CE_1 \cup CE_2$;

– видалення компонента з компонентного середовища $CE_2 = CE_1 \setminus C$.

Мають місце наступні теореми.

Теорема 2. Кожне компонентне середовище *CE* є результатом виконання послідовності операцій розгортання компонентів, які входять до його складу, в певному компонентному каркасі:

$$CE = C_1 \oplus C_2 \oplus \ldots \oplus C_n \oplus FW.$$

Теорема 3. Побудова компонентного середовища не залежить від порядку інсталяції компонентів, які входять до складу цього середовища, тобто:

$$C_1 \oplus \left(C_2 \oplus CE\right) = C_2 \left(C_1 \oplus CE\right).$$

Теорема 4. Операція об'єднання компонентних середовищ асоціативна:

$$\left(CE_1 \cup CE_2\right) \cup CE_3 = CE_1 \cup \left(CE_2 \cup CE_3\right).$$

Теорема 5. Операція об'єднання компонентних середовищ комутативна:

$$CE_1 \cup CE_2 = CE_2 \cup CE_1.$$

Теорема 6. Для будь-якого компонентного середовища

$$CE \cup FW = FW \cup CE = CE.$$

Теорема 7. Для довільних компонентних середовищ CE_1 і CE_2 та компонента *C* завжди виконується

$$C \oplus \left(CE_1 \cup CE_2\right) = \left(C \oplus CE_1\right) \cup CE_2 = \left(C \oplus CE_2\right) \cup CE_1.$$

Теорема 8. Для будь-якого компонента *C* та компонентного середовища *CE* завжди виконується $\left(C \oplus CE\right) \setminus C = CE$.

Модель ПС для певного компонентного середовища описується виразом *CP*= (*CE, Cont, CO*), де: *CE* – компонентне середовище, *Cont* – множина контрактів для компонентів, що входять до складу *CE*, *O* – підмножина компонентів з *CE*, що включають реалізації і звертаються до інших компонентів за допомогою своїх вихідних інтерфейсів. Ці компоненти є початковими для компонентної програми *CP*, наприклад, до їх складу входять компоненти, що безпосередньо взаємодіють з кінцевим користувачем і звертаються до основної функціональності програми з метою виконання певних запитів.

Умова цілісності компонентної програми полягає в існуванні для кожного компонента C_1 з *CE*, що має вихідний інтерфейс $CInO_1^i$, компонента C_2 з відповідним вхідним інтерфейсом $CInI_2^m$ і контракт $Cont_{12}^{im} = (CInO_1^i, CInI_2^m, IMap_{12}^{im})$ входить до складу множини *Cont*.

Задачі КП. Для реалізацій функціональності компонентів застосовуються різні мови програмування з відповідними типами даних в них. При збиранні (об'єднанні, інтеграції) різномовних компонентів вирішується задача перетворення типів даних через інтерфейс шляхом формальних механізмів релевантної конвертації типів даних з урахуванням моделей інтегрованого середовища.

Існує два підходи при вирішенні проблеми інтеграції КПВ: 1) застосування *моделі взаємодії* пари різномовних компонентів в МП у середовищі розподілених систем шляхом їх композиції через модуль посередник (stub, skeleton) з використанням *міжмовного* і *міжмодульного* інтерфейсу [1]; 2) опис інтерфейсу КПВ в мові IDL за параметрами in, out, inout для завдання даних при обміні даними між собою.

В теоретичному аспекті інтеграція різномовних компонентів базується на формальних відображеннях (поданих як суперпозиції базових відображень).

Модель збирання різнородних КПВ. Нехай задана множина компонентів $CSet = \{C_i\}$, які написані на різних МП. При взаємодії компоненти обмінюються даними, кожне з яких визначається трійкою: ім'ям змінної, її типом даних та значенням. Дані, якими обмінюються кожна пара компонентів C_i і C_j, можуть бути еквівалентними, якщо вони мають однакову семантичну структуру і тип, або нееквівалентними і в цьому випадку необхідне їхнє перетворення за допомогою функцій, які подані відображеннями:
FN_{ij}: N_i –> N_j , FT_{ij}: T_i –> T_j , FV_{ij}: :V_i –> V_j, де FN_{ij} встановлюють відповідність між іменами змінних (наприклад, у множинах формальних та фактичних параметрів), FT_{ij} описують еквівалентні відображення для типів даних, FV_{ij} реалізують необхідні перетворення значень даних.

Задача заміни змінних FN_{ij} виконується шляхом впорядкування імен змінних (наприклад, в описі конфігурації для компонентів, що інтегруються). Відображення між типами даних FT_{ij} базується на перетвореннях типів даних, кожен з яких подається як абстрактна алгебра або алгебраїчна система $T = (X, \Omega)$, де X – множина значень, що можуть приймати зміні цього типу, а Ω – множина операцій над цими змінними. Відображення FV_{ij} застосовуються у випадках нееквівалентності типів T_i та T_j (наприклад, перетворення цілого значення у дійсне) [17].

У випадках багаторазових викликів методів компонентів можуть виникати питання прямого та оберненого перетворення даних між формальними та фактичними параметрами і тому є суттєва умова ізоморфізму відображення між відповідними типами даних. Для існування ізоморфізму використано механізм алгебраїчних систем для перебудови даних. *Під перетворенням* типу $T_i = (X_i, \Omega_i)$ у тип $T_j = (X_j, \Omega_j)$ розуміється таке перетворення, при якому семантичний зміст операцій з Ω_i еквівалентний змісту операцій з Ω_j. У загальному випадку перетворення типу T_i у тип T_j може бути однобічним, тобто еквівалентного перетворення не існує. Ці системи для МП, вони поширені на загальні типи даних, що описані у стандарті ISO/IEC 11404 і містить механізми генерації GDT до FDT МП.

Задача забезпечення взаємозв'язку пари компонентів (систем, ПС) виникає при їх збиранні у складні структури, наприклад сімейство систем. Для цього побудовано сукупність відображень для різних видів викликів методів (Call, RPC, RMI), за якими встановлюються однозначні відповідність між множиною фактичних параметрів $V = \{v_1, v_2, ..., v_k\}$ об'єкта, що викликає, і множиною формальних пара-

метрів $F = \{f_1, f_2, ..., f_k\}$ для компонентних об'єктів, що викликається.

Задачі побудови типів даних засобами *алгебраїчних систем* для основних типів даних, що застосовуються у різних МП і ізоморфні відображення між цими алгебраїчними системами детальніше розглянуто у монографії [17].

Але відносно гетерогенного середовища задача взаємодії вирішується таким шляхом:

1) перетворення типів даних, які застосовуються при передачі даних між взаємодіючими компонентами і системами;

2) усунення відмінностей у поданні даних за різними особливостями комп'ютерних платформ і архітектур, а також за принципами кодування (декодування) даних при передачі їх через мережу іншим системам;

3) завдання інтерфейсу взаємодіючих компонентів і систем мовою IDL, API з описом в них даних для передачі через операції віддаленого виклику тощо.

Інтерфейсна модель для КПВ і систем має вираз

$$ISyst = (IFu, IG), \qquad (9)$$

де $IFu = \{IFu^i\}$ – множина інтерфейсів, IG – інтерфейсний граф, який еквівалентний графу G. Для класів визначаються умови, коли вони допускають подання як елементів множини інтерфейсів у інтерфейсному графі.

Теорема 9. Для кожної моделі системи *PS*, зовнішня взаємодія з класами public-методи та керованими змінними, існує єдине інтерфейсне подання *ISyst* з еквівалентною функціональністю.

Ця теорема визначає умови існування еквівалентного відображення між об'єктним та інтерфейсним поданнями програми. Як наслідок, при проектуванні компонентних програм і систем можуть застосовуватись моделі та методи ОКМ методу [8] і формальні моделі КП, які побудовані на єдиній теоретичній основі [7–9].

На даний час проблема перетворення даних базується на наведеному стандарті ISO/IEC 11404 GDT, що механізми їх генерації від GDT до FDT та упорядкування та перетворення їх за механізмами *примітивних функцій* спеціальних бібліотек системи VS.Net (CLR – Common language runtime, CTS – Common type system і CLS – Common language specification) тощо.

Змінювання компонентних ПС. Система будується з КПВ шляхом їх збирання та небхідного змінювання інтерфейсів та реалізацій компонентів або їх додавання.

Компонентна зовнішня алгебра $\Psi = \{CSet, CESet, \Omega\}$ включає множини компонентів, компонентних середовищ і операції над ними, а також внутрішню алгебру з операціями реорганізації вихідного коду, а саме: *рефакторинг* для змінювання внутрішній структури КПВ чи інтерфейсу зі збереженням його поведінки; *реінженерії*, як апарат еволюції КПВ чи ПС шляхом внесення деяких змін в КПВ або перепрограмування логіки, уточнення функції окремих КПВ для їх змінювання в ПС; *реверсної інженерії* шляхом встановлення первісної структури в МП і її реконструювання: $\sum = \{\Psi \cap \varphi\} = \{\text{CSet}, \text{CESet}, \Omega 1\} \cap \cap \{\text{CSet}, \text{CESet}, \Omega 2\}\, \Omega 2\}$, де Ψ – зовнішня алгебра, φ – внутрішня алгебра.

Зовнішня алгебра:

$\Psi = \{CSet, CESet, \Omega 1\}$, де *CSet* – множина компонентів C, CESet – середовище *E* з множини компонентів *C* і інтерфейсів *In*. $\Omega 1 = \{CE1, CE2, CE3, CE4\}$ – операції алгебри: *CE*1 – операції оброблення компонентів; *CE*2 – операції інсталяції; *CE*3 = = *CE*1 ∩ *CE*2 – операції зборки (об'єднання); *CE*4 = *CE*1 \ *C* – операції видалення компонента C з середовища;

*C*2 – *CE*2 = *C*2 ⊕ (*CE*1 \ *C*1) – операції заміщення.

Алгебра еволюції КП. Повторне використання КПВ є базисом еволюції компонентів. Їй приділяється увага за кордоном, вона досліджується у програмах (Японії, Великобританії, США й ін.), у світових корпораціях (IBM, Microsoft, Hewlett–packard й ін.) та постійно проблема reuses розглядаються на міжнародних конференціях протя-

гом десятиріч. В КПВ вкладаються великі кошти для отримання прибутку при розробленні з них складних ПС.

Тому важливе значення мають методи відбору КПВ із різних сховищ, їх еволюційна трансформація чи перебудова під потреби користувача операціями їх реінжинірингу і рефакторінгу.

Методи трансформації поділяються на два види: методи, що змінюють функціональність та поведінку компонентів і методи, які пов'язані з нефункціональними змінами. До перших належать зміни в інтерфейсах (зміна сигнатур інтерфейсів, додавання інтерфейсів) та реалізації (зміна алгоритмів та логіки, заміщення та додавання реалізацій) та ін. До другого виду належать зміни, що пов'язані з нефункціональними характеристиками (збільшення надійності, ефективності, мобільності), мовами та платформами виконання.

При формальному розгляді методів трансформації введено базові поняття методів рефакторінгу, реінжинірингу, реверсної інженерії та розглянута семантика цих понять для КПВ. Виходячи з введених понять і формальної моделі компонента (7), розроблені функції трансформації КПВ і систем: функцій рефакторінгу з виконанням умови незмінності інтерфейсів і компонентної моделі, функцій реінжинірингу з незмінністю компонентної моделі, функцій реверсної інженерії (встановлення первіного алгоритму) [8, 19].

Внутрішня алгебра еволюції:

$$\varphi = \{CSet,\ CESet, \Omega 2\},$$

де $\Omega 2 = \{Oref,\ ORei,\ ORev\}$ – сукупність операцій еволюції компонентів. енерії.

На основі семантики, умов та вимог виконання функцій рефакторінгу побудовано внутрішню алгебра рефакторінгу компонентів $\sum rf = (CSet,\ Ref)$, де $CSet = \{C_n\}$ – множина компонентів, модель кожного з яких наведена виразом (7), а $Ref = \{AdOIm,\ AdNIm,\ RelIm,\ AdIn\}$ – сукупність операцій рефакторінгу, де *AdOIm* – додавання нової реалізації для існуючого інтерфейсу.

$NewC = AdOIm(OldC,\ NewCIm^s,\ NewCInO^s)$ має семантику, де $NewCIn = OldCIn \cup NewCInO^s$ – операція додавання вихідних інтерфейсів для нової реалізації, $NewCIm = OldCIm \cup \{NewCIms\}$ – додавання нової реалізації.

Аксіома. Для $\exists OldCIn^t \in OldCInI$ існує операція додавання вхідних інтерфейсів і відповідної функціональності. Ця операція є асоціативною та комутативною і є умовою цілісності компонента.

Операція *AdNIm* є додавання нової реалізації, вхідний інтерфейс, якої не входить до складу множини інтерфейсів компонента:

$$NewC = AdNIm\left(OldC, NewCIm^s, NewCInO^s\right).$$

В ній $NewCIn = OldCIn \cup NewCInO^s$ – додає нові вихідні інтерфейси для нової реалізації, $NewCIm = OldCIm \cup \left\{NewCIm^s\right\}$ – додає нову реалізацію. Ці операції є асоціативними та комутативними. Цілісність компонента зберігається.

Операція *ReplIm* є заміщенням існуючої реалізації новою без зміни вхідного інтерфейсу: $NewC = ReplIm\left(OldC,\right.$ $\left.NewCIm^s, NewCInO^s, OldCIm^r, OldCInO^r\right).$

В ній $NewCIn = OldCIn \cup \left\{NewCInO^s\right\} \setminus$ $\setminus \left\{OldCInO^r\right\}$ – додавання вихідних інтерфейсів для нової реалізації і видалення вихідних інтерфейсів старої реалізації, $NewIm = OldIml \cup \left\{NewCIm^s\right\} \setminus \left\{OldCIm^r\right\}$ – додавання нової реалізації і видалення застарілої. Умовою виконання операції є незмінність функціональності усіх інтерфейсів, що пов'язані зі старою реалізацією, після заміщення її новою:

$$\forall OldCIn^t \in OldCInI$$
$$\left[\left(Pr\left(OldCIn^t\right) \subseteq OldCIm^r\right) =>\right.$$
$$\left.\left(Pr\left(OldCIn^t\right) \subseteq NewCIm^s\right)\right] \vee \exists OldCIm^j$$
$$\in \left(OldCIm \setminus \left\{OldCIm^r\right\}\right)\left[Pr\left(OldCIn^t\right)\right.$$
$$\left.\subseteq OldCIm^j\right].$$

Лема 1. Операція заміщення існуючої реалізації новою з вищевизначеними умовами та семантикою зберігає цілісність компонента.

Операція *AdIn* є додаванням нового вхідного інтерфейсу для існуючої реалізації: $NewC = AddIn\left(OldC,\ NewCInI^q\right)$. В ній $NewCIn = OldCIn \cup \left\{NewCInI^q\right\}$ – додавання нового вхідного інтерфейсу, $\exists\ OldCIm^s \in OldCImPr\left(NewCInI^q\right) \subseteq \subseteq OldCIm^s\Big]$ з відповідною функціональністю.

Лема 2. Операція додаванням нового вхідного інтерфейсу для існуючої реалізації з вищевизначеними умовами та семантикою зберігає цілісність компонента.

Наведені операції є базовими для більш складних ПС. Наприклад, додавання реалізації разом з вхідним інтерфейсом визначаються наступною суперпозицією:

$$NewC = AddIn\Big(AddNIm\Big(OldC,\ NewCIm^s,\ NewCInO^s\Big),\ NewCInI^q\Big).$$

Сформульовано та доведено наступну теорему.

Теорема 10. Алгебра рефакторінгу компонентів $\sum rf = \left(CSet,\ Refac\right)$ є повна та незаперечна.

Зв'язок алгебри КП з рефакторингом. Згідно з теоремою 10 результатом операцій рефакторінгу або суперпозицій операцій є певний компонент. Тому *CSet* складається з КПВ репозиторію і різних модифікацій компонентів як результатів виконання операцій рефакторінгу. Наприклад, для певної компонентної моделі конфігурація, яка складається з двох компонентів і для другого з них додаються реалізація і вхідний інтерфейс, описується наступним виразом:

$$CE = C_1 \oplus AddIn\Big(AddNIm\Big(C_2, NewCIm^s,\ NewCInO^s\Big),\ NewCInI^q\Big) \oplus FW.$$

Внутрішня компонентна алгебра реінжинірингу компонентів $\sum^{re} = \left(CSet,\ Re\right)$ може бути побудована лише за умови порушення цілісності подання компонентів. Крім операцій рефакторінгу $\left(Ref\right)$ до множини входять операції *Re*, які видаляють з компонента існуючий інтерфейс або змінюють його сигнатуру. Це порушує умову цілісності, бо інші компоненти, які звертаються до нього, не зможуть мати доступу до необхідної функціональності.

У зв'язку з цим розглянуто модель реінжинірингу компонентів $M_{Re} = \left(CSet,\ Re\right)$. У цьому випадку семантика операцій з *Re* може полягати у трансформації не тільки цільового компонента, а й інших. Наприклад, при зміні сигнатури вхідного інтерфейсу необхідна одночасна зміна інших компонентів, в яких існують звернення до методів цього інтерфейсу.

Аналогічним чином сформовано відповідну модель реверсної інженерії $M_{Res} = \left(CSet,\ Rev\right)$, при цьому $Re \subset Rev$. Особливість множини *Rev* полягає у тому, що вона не визначена повністю (кількість операцій є необмеженою), а лише за класифікацією операцій. Наприклад, зміни якісних характеристик КПВ можуть виконуватись довільним способом і відповідні операції складають сукупність операцій для зміни певного показника якості (за класифікаційною ознакою в множини *Rev*).

Таким чином, алгебра Σrf та моделі M_{Re} і M_{Res} складають формальний апарат внутрішній алгебри щодо моделей і методів еволюції компонентів.

Розвиток теорії КП апаратом взаємодії і варіабельності

Починаючи з 2007 р. у відділі активно розвивалася теорія взаємодії і варіантності ПС з КПВ у межах фундаментального проекту з генерувального програмування [10] та комплексна технологія реалізації ПС і сімейств ПС із КПВ в інтегрованому середовищі сучасних систем Eclipse, VS.Net, Corba.

Варіабельність ПС і сімейств. Вона бере свій початок з задач створення різних варіантів програмного продукту (наприклад, дослідження Лемону 210 версій ОС 360, теорія рефакторингу М. Фаулера,

Product line SEI тощо). Нами визначено клас моделей і процес керування варіабельними структурами ПС і СПС з використанням КПВ і точок варіантності у їх складі [12, 13, 20]. Модель керованої варіантами структури і конфігурування членів СПС із КПВ накопичених в стандартному виді у репозиторії ІТК будується на основі теорії КП щодо операції алгебри змінення, поповнення новими формальними моделями, еквівалентними за функціональністю КПВ, які створюють підклас рефакторингу варіабельних членів сімейства і СПС.

Моделі варіабельності СПС є пара підмоделей:

$$VM = \{SV; AV\}, \qquad (10)$$

де SV – підмодель варіабельності структури СПС;

AV – підмодель варіабельності продуктів процесу розроблення СПС.

Моделі VM у процесі розроблення СПС забезпечує рівень змінності продуктів СПС, адекватно заданим умовам їх використання, а також скорочення термінів розроблення продуктів СПС.

Підмодель варіабельності структури СПС має вигляд:

$$SV = \{G1, \{GR\} Con,\ Dep\},$$

де $Gt = \langle F,\ LF \rangle$ – граф, вершини якого $(f \in Ft)$ є унікальні ідентифікатори артефактів різного типу (потреб, вимог, КПВ, ПС тощо), поєднаних зв'язками обов'язкового та варіантного підпорядкування (LFt);

Con і *Dep* – предикати на декартовому добутку множин артефактів усіх типів предикатів, які подають обмеження й залежності між функціями та характерними елементів СПС.

Ці моделі описані в [13]. Їх розвиток з точки зору теорії КП нині виконується аспірантом Колесником А.Л. у дисертаційній роботі.

Розвиток теорія взаємодії КПВ до систем. Модель взаємодії призначена для обміну інформацією між різними компонентами і системами при їх об'єднанні й обчисленні. Під *взаємодією* розуміються зв'язки двох і більше об'єктів КП і відношення між ними. В основі лежить процес обміну повідомленнями (викликами, запитами, протоколами) між програмами, системами і середовищами для сумісного рішення певної задачі. В повідомленнях задається інтерфейс, який специфікується загально прийнятою мовою IDL (Interface Definition Language) або іншими (APL, SIDL тощо). На загальному рівні інтерфейс є механізмом забезпечення взаємодії (interconnection) різнорідних програм для сучасних середовищ, що забезпечують розроблення програм і систем [11, 18].

Нові механізми забезпечення взаємодії програмних об'єктів розробляються нами для сімейств систем з використанням моделі GDM і моделі характеристик окремих елементів, що входять до складу СПС. Вони відповідають деяким функціям або поняттям ПрО, що специфікуються різними МП і реалізуються в одному з діючих середовищ.

Під *моделлю взаємодії* розуміється взаємозв'язок між КПВ і ПС через параметри, що передаються між ними. Модель включає систему понять і відношень, які подаються математичними засобами – абстрактна алгебра, теорія множин тощо.

Модель взаємодії є розвитком розглянутої в теорії КП операцій взаємодії між КПВ стосовно систем і середовищ.

Модель M_{inter} має загальний вигляд:

$$M_{inter} = \{M_{pro},\ M_{sys},\ M_{env}\}, \qquad (11)$$

де M_{pro} = {*C, Int, Pr*} – модель програми, *C* – компонентний об'єкт, *Int*– інтерфейс, *Pr* – протокол для передачі даних;

M_{sys} = {*PS, Int, Pr* } – модель програмної системи *PS*, *Int* – інтерфейс, *Pr* – протокол для передачі даних;

M_{env} = {*Envir, Int, Pr*} – модель середовища, в якому *Int, Pr* містить сукупність зовнішніх інтерфейсів та викликів, що передають дані між програмами через мережу.

Базовими параметрами моделі взаємодії M_{inter} є програма, інтерфейс і повідомлення. По відношенню до моделі відкритих систем OSI, M_{inter} є моделлю верхнього рівня.

Реалізація технології компонентного програмування в ІТК

Наведена теорія КП реалізована практично як комплексна технологія, в якої об'єднанні на однієї концептуальній основі теоретичний апарат моделювання ПрО за об'єктами та теорією компонентного програмування ПС із різномовних КПВ на основі компонентної алгебри і апарата алгебраїчних систем з еквівалентним перетворенням типів даних КПВ до потрібних форматів середовища виконання, а також алгебри рефакторінгу, реінженерії та реверсної інженерії з операціями модифікації компонентів через операції конфігуратора.

Комплексну технологію подано спектром окремих ліній в ІТК [13, 14, 17, 21] за веб-сайтом ІТК (http://sestudy.edu-ua.net), що включає інструментально-технологічні засоби, інструменти проектування і специфікації КПВ, ПС, членів сімейства систем, стандартні системи (Eclipse, Protege, репозиторій, CORBA, MS.Net тощо), МП, онтології тощо.

Спектр е-ліній технологій виробництва програм і систем:

– побудова КПВ і їх розміщення у репозиторії КПВ зі стандартизованим описом типу WSDL для їх повторного використання;

– зборка різномовних програм і КПВ у складні ПС;

– взаємодія різнорідних програм за моделлю взаємодії програм і систем для різних операційних середовищ, переносних у інше середовище для функціонування з даними, що подаються із програм іншого середовища;

– варіабельність ПС і сімейств з використанням операцій еволюції ПС і СПС;

– онтологія опису і подання доменів в DSL (ЖЦ SE і обчислювальна геометрія);

– генерація готових програм і КПВ у варіабельну структуру шляхом їх конфігурування; оцінювання показників якості, вартості та витрат ПС;

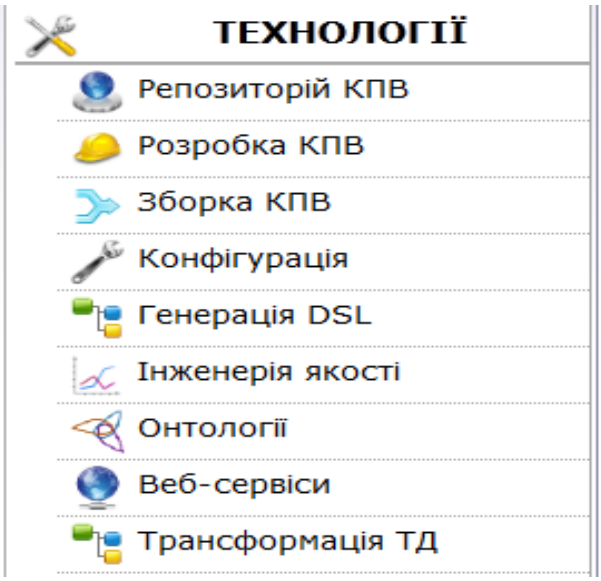
– оцінювання показників якості, вартості та витрат ПС;

– е–програмування мовами C#, Java, Basic, C++ у середовищі VS.Net, Corba та Eclipse;

– е–навчальна технологія проектування ПС за е-підручником "Програмна інженерія";

– е–настанова для моделювання доменів засобами Protégé тощо.

Запропонований теоретичний апарат реалізовано в ІТК, як розділ головного меню веб-сайта «ТЕХНОЛОГІЇ». В ньому приведено реалізацію конкретизованих операцій з класу операцій зборки, змінювання та конфігурування КПВ (див. меню головної сторінки ІТК).



Кожна позиція цього меню має наступне призначення:

– обслуговування репозиторію КПВ та компонентів;

– розроблення нових КПВ;

– збирання КПВ в складні ПС з IDL-інтерфейсом;

– конфігурування КПВ в вихідний код;

–генерація КПВ з використанням варіантних точок;

– оцінювання якості КПВ та ПС;

– генерація DSL опису домену ЖЦ;

– модель онтології обчислювальної геометрії;

– web-сервіси;

– трансформації загальних типів даних GDT до фундаментальних FDT.

Операції зовнішній алгебрі подані в ІТК наступними операціями:

link PS (*A, B, C*) – зборка компонентів *A, B, C*;

config SPS (A^{l1}, B^{l1}, C^{l1} ($In^{idl}A$, $In^{idl}B$, $In^{idl}C$) – конфігуруванням *A, B, C* в *Li* мові і інтерфейсів в *IDL;*

redoing x, y $\Rightarrow$ *BD* – передача даних *x, y* BD за прийнятим форматом;

redo TD (*x,y*) – передача даних з перетворенням їх значень;

interconect PS (*A, B, C, InA, InB, InC*) – об'єднання (взаємодія) *A, B, C* за їх інтерфейсами;

redevelop PS (*InA, InB*) – перебудова типів даних *A, B*.

Конкретизація операцій еволюції, що додані до зовнішньої компонентної алгебри, такі в ІТК:

makeaway $PS(A)$ – віддалити з *PS* компонент *A*;

add $PS(A,C)$ – додати *A, C* до *PS*;

insert $F \Rightarrow PS$ – вставити *F* в *PS*;

reNa $A \Rightarrow B$ – змінити ім'я компоненту А на В;

redact A (*PS*) – редагувати програму *A* в *PS*.

Ці операції поповнені новими операціями і процесами ІТК, пов'язаними з варіабельністю: 1) управління варіабельністю СПС за адаптованою моделлю Product Line та стандартом ISO/IEC 26550 з механізмами варіантності й оптимізації створюваної структури ПС з КПВ, артефактів СПС; 2) конфігурування (збирання) КПВ в адекватну структуру вихідного коду ПС, здатну на повторне конструктивне оновлення СПС тотожними, подібними, чи еквівалентними КПВ; 3) ре факторингу структури СПС за новими вимогами, чи заміни непотрібних КПВ; 4) оцінювання якості КПВ і СПС.

До складу ІТК включено реалізацію моделі взаємодії через плагіни Eclipse за таким практичними моделями:

– *Visual Studio.Net, Eclipse* для реалізації взаємодії окремих програм у мові C#, включаючи специфікацію інтерфейсу та перенесення його і КПВ в репозиторій системи Eclipse з встановленням зв'язків з даним середовищем через механізм stub, skeleton або через конфігураційний файл з операціями оброблення даних;

– *Corba*, *Java*, *MS.Net* забезпечують розроблення програм на МП цих середовищ та встановлення зв'язків між ними для розміщення програм в репозиторій та надання доступу іншим;

– *IBM VSphere*, *МП, Eclipse*, де розробляються нові програми з використанням можливих МП, що допускаються у цьому середовищі або у VSphere.

Розроблений веб-сайт ІТК орієнтований на реалізацію аспектів програмної інженерії, до складу якої входить КП. До нього на даний час звернулися біля 4000 користувачів – викладачів та студентів. Нами були доповіді на конференції ICTERI–2012 [20, 21], пов'язані з методами і засобами технології програмування.

Висновки

Сформульовано теоретичний апарат моделювання об'єктної моделі ПрО, механізми переходу від визначених в них базових характеристик і методів об'єктів до компонентів і їх інтерфейсів за новою теорією КП, що складається з базових понять (компонент, компонентна модель, модель ПС, модель середовища тощо) та математичних методів збирання КПВ за зовнішньої та внутрішньої компонентною алгеброю, методів еволюції компонентів та методології компонентного проектування деяких ліній розроблення нових ПС і інформаційних систем.

В комплексну технологію увійшли спрощені лінії виробництва ПС із КПВ, модель взаємодії різномовних КПВ на основі апарата алгебраїчних систем з еквівалентним перетворенням основних типів даних до потрібних форматів середовища виконання та модель варіабельності ПС і СПС. Крім того, включені оператори еволюції КПВ із алгебри рефакторінгу, ре інженерії та реверсної інженерії з операціями їх модифікації. Теоретичні і прикладні наробки є новими, не мають прототипу. Веб-сайт ІТК (http://sestudy.edu-ua.net) використовується при навчанні студентів технологіям КП.

1. *Лаврищева Е.М., Грищенко В.Н.* Сборочное программирование. – Киев: Наук. думка, 1991. – 213 с.
2. *Грищенко В.М.* Подход к формализации объектно-ориентированной методологии // Проблемы программирования. – 1997. – № 1. –С. 33–39.
3. *Грищенко В.М.* Систематизований підхід до визначення програмних компонентів // Проблеми програмування. – 2001. – № 3 – 4. – С. 23–30.
4. *Грищенко В.М.* Особливості компонентно-орієнтованої розробки програмного забезпечення // Проблеми програмування. – 2001. – № 3 – 4. – С. 23–30.
5. *Грищенко В.Н., Лаврищева Е.М.* Компонентно-ориентированное программирование. Состояние, направления и перспективы развития // Проблеми програмування. Спец. вип. – 2002. – № 1–2. – С. 80–90.
6. *Грищенко В.Н., Лаврищева Е.М.* Методы и средства компонентного программирования // Кибернетика и системный анализ. – 2003. – № 1. – С. 39–55.
7. *Грищенко В.Н.* Формальные модели компонентного программирования // Проблеми програмування. – 2003. – № 2. – С. 42–57.
8. *Грищенко В.Н.* Алгебраїчна модель рефакторінгу компонентів // Проблеми програмування. – 2003. – № 4. – С. 43–53.
9. *Грищенко В.М.* Метод об'єктно-компонентного проектування програмних систем // Проблеми програмування. – 2007. – № 2. – С. 113–125.
10. *Лавріщева К.М.* Генерувальне програмування ПС і сімейств // Проблеми програмування. – 2009. – № 1. – С. 3 – 16.
11. *Лавріщева К.М.* Взаємодія програм, систем й операційних середовищ // Проблеми програмування. – 2011. – № 3. – С. 13–24.
12. *Лавріщева К.М., Слабоспицька О.О., Коваль Г.І., Колесник А.О.* Теоретичні аспекти керування варіабельністю в сімействах програмних систем. – Вісник КГУ, серія фіз.-мат. наук. – 2011. – № 1. – С. 151–158.
13. *Колесник А.Л.* Підходи до конфігурування компонентів повторного використання // Проблеми програмування. – 2011. – № 4. – С. 57 – 66.
14. *Лавріщева К.М.* Інструментально-технологічний комплекс для розробки и навчання прийомам виробництва програмних систем. – К.: Вісник НАН України, 2012. – № 3. – С. 17–26.
15. *Андон П.І., Лавріщева К.М.* Розвиток фабрик програм в інформаційному світі // Вісник НАН України. – 2010. – № 10. – С. 15–41.
16. *Фреге Г.* Логика и логическая семантика. – М: Аспект.
17. *Лаврищева Е.М., Грищенко В.Н.* Сборочное программирование. Основы индустрии программных продуктов. – Второе изд. – К.: Наук. думка, 2009. – 371 с.
18. *Лаврищева Е.М.* Парадигма интеграции в программной инженерии // Материалы второй междунар. науч.-практ. конф. Укр-Прог. – 2000, 23 – 26 мая 2000. – С. 351–360.
19. *Фаулер М.* Рефакторинг: улучшение соответствующего кода. – СПб.: Символ–Плюс, 2003. – 432 с.
20. *Kolesnyk A., Clabospitskaya O.* Tested Approach for Variability Management Enhancing in Software Product Line – Conference ICTERI–12.
21. *Lavrischeva K., Ostrovski A., and Radetskyi I.* Approach to E-Learning Fundamental Aspects of Software Engineering. – Conference ICTERI–12 http://senldogo0039.springer–sbm.com/ocs/home/ICTERI2012

Одержано 16.08.2012

Про автора:

Лавріщева Катерина Михайлівна,
доктор фізико-математичних наук,
професор, завідуюча відділом.

Місце роботи автора:

Інститут програмних систем
НАН України,
03187, Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3470.