Valery BUDAK, Oleg PAVLENKO
Mykolaiv

FREE VIBRATION ANALYSIS OF LAMINATED SHALLOW SHELLS USING MINDLIN THEORY BY SPLINE COLLOCATION METHOD

The present study deals with the free vibration of symmetrically laminated doubly curved shallow shells with variable thickness and rectangular platform. The equations of motion of the shell are derived using first order shear deformation theory. Spline function approximation technique, which includes B-splines of the third order, is used to reduce two-dimensional system of coupled differential equations in terms of displacement and rotational functions to one-dimensional. A generalized eigenvalue problem is obtained by applying a point collocation method with suitable boundary conditions. The vector-matrix form of the governing equations with different boundary conditions, from which values of a frequency parameter and the corresponding mode shapes of vibration are obtained, is presented.

Key words: free vibrations, laminated shallow shell, spline collocation method, Mindlin theory.

Валерий БУДАК, Олег ПАВЛЕНКО
г. Николаев

АНАЛИЗ СВОБОДНЫХ КОЛЕБАНИЙ СЛОИСТЫХ ПОЛОГИХ ОБОЛОЧЕК В ПОСТАНОВКЕ МИНДЛИНА МЕТОДОМ СПЛАЙН-КОЛЛОКАЦИИ

В статье рассматриваются свободные колебания симметричных слоистых пологих оболочек двоякой кривизны переменной толщины с прямоугольным планом. Уравнения движения оболочки выведены в рамках уточненной теории первого порядка. Полученная двухмерная система дифференциальных уравнений относительно перемещений и углов поворота, сведена к одномерной методом сплайн-аппроксимации с использованием В-сплайнов третього порядка. Применив метод коллокации, получена задача на собственные значения с соответствующими краевыми условиями. Разрешающая система с разными граничными условиями, из которой можно получить значения собственных частот и соответствующих им форм колебаний, представлена в векторно-матричной форме.

Ключевые слова: свободные колебания, слоистые пологие оболочки, метод сплайн-коллокации, теория Миндлина.

Стаття надійшла до редколегії 19.02.2016

УДК 004.415.28

Олександра БУЛГАКОВА*, *Павло КИСЛИЧЕНКО
м. Миколаїв
sashabulgakova@list.ru, pavelkislichenko@gmail.com

АРХІТЕКТУРА КРУПНИХ МАСШТАБОВАНИХ ДОДАТКІВ В КОНТЕКСТІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

У статті розглянута архітектура крупно-маштабованих додатків в контексті об'єктно-орієнтованого програмування. Наведена методологія декомпозиції у сфері Інтернет-комерції, на спрощеному прикладі проектування модуля персонального кошика.

Для забезпечення високорівневого інтерфейсу для клієнтських класів, та приховування від них низькорівневого інтерфейсу класів підсистеми, запропоновано використовувати найбільш популярні абстрактні архітектурні рішення, які також називаються шаблонами проектування або патернами.

Ключові слова: програмні системи, архітектура, об'єктно-орієнтоване програмування, проектування програмного забезпечення, система, додаток.

Одним з основних питань, яке найчастіше виникає після завершення розробки та запуску програмної системи, як готового продукту, є можливість маштабування. Тобто можливість інтегрування у систему нових модулів, які забезпечать додатковий функціонал, або модифікацію вже існуючих. Неможливість цього зробити без значної зміни інших частин системи – є результатом нестійкої архітектури усього додатку, тобто великої кількості залежностей між класами і формулює основну проблему, яка розглянута у статті.

З приходом об'єктно-орієнтованного програмування (надалі ООП) у світ інформаційних технологій робота на стадії проектування значно полегшилась, так як ООП – мови надають спеціальні мовні конструкції для абстрагування процесу, що моделюється та формулювання високорівневого інтерфейсу для клієнтських класів, такі як абстрактний клас або інтерфейс.

На протязі останніх десятиліть розробки програмних систем за допомогою ООП – мов, співтовариство програмістів виявило ряд схожих задач, які повторюються при моделюванні різних процесів та розробило ряд архітектурних рішень, які на даний момент є загальносхваленими. Однак правильної, як такої, архітектури не існує, так як кожна програмна система моделює конкретний процес у конкретній сфері діяльності, отже майже кожна система – унікальна. Більш правильним буде сказати, що архітектура є найбільш відповідна або належна до додатку, що розробляється.

Найчастіше знайдені рішення являються лише компонентами загальної архітектури програмної системи, так як підходять для моделювання конкретних локальних частин додатку. На даний момент у світі існують 23 найбільш популярних абстрактних архітектурних рішень, які також називаються шаблонами проектування або патернами.

Патерн – це лише абстрактний приклад правильного використання невеликої кількості комбінацій найпростіших технік ООП [1, 19]. Патерни проектування – це прості абстраговані приклади, що показують правильні способи організації взаємодій між класами або об'єктами [1, 19].

Розроблені шаблони проектування поділяють на три категорії по меті [1, 23]:

- пораждаючі (описують процес створення нових екземплярів класів у системі);
- структурні (описують правильну організацію екземплярів класів);
- поведінкові (описують взаємодію між екземплярами класів);
- та на дві категорії по застосовності [1, 28]:
- до класів;
- до об'єктів.
- Можна сформулювати основні причини, які аргументують важливість правильного проектування додатку [5, 1]:
- складність системи, як правило, зростає значно швидше її розмірів і якщо не потурбуватися про це завчасно, то достатньо швидко наступає момент, коли команда розробників перестає контролювати систему;
- правильно побудована архітектура дозволяє розпаралелити процес розробки системи. З цього випливає економія часу, сил та фінансових витрат;
- добре спроектований додаток має відносно легку читаємість коду, що дозволяє впроваджувати до розробки нових членів команди, без затрат часу на вивчення екзотичної організації системи;
- програмну систему з добре продуманою архітектурою легше розширювати, змінювати, тестувати, підлагоджувати та розуміти.

Виходячи з вище описаних причин, постановкою задачі є розробка архітектури програмного забезпечення з рядом, цілком усвідомлених, критеріїв, а саме:

*Ефективність системи***.** Сюди можна віднести такі характеристики, як безпека, надійність, маштабованість, можливість впоратися з навантаженням.

Гнучкість системи. Чим швидше та зручніше можна внести зміни у вже існуючий функціонал, чим менше проблем та помилок це викликає – тим гнучкіша і конкурентоспроможна система.

Можливість розширення. Можливість додавати до системи нові сутності та функції, не порушуючи її основної структури. Ця вимога являється настільки важливою, що вона навіть сформульована у вигляді окремого принципу – *Принципу відкритості/закритості* (Open-Closed – Principle – другий, з п'яти принципів SOLID): *Програмні сутності (класи, модулі, функції, тощо) повинні бути відкритими для розширювання, але закритими для модифікації*.

Можливість повторного використання. Систему бажано проектувати таким чином, щоб її підсистеми або модулі можна було б повторно використовувати у інших системах.

Гарно структурований, зрозумілий код. У процесі розробки, як правило, беруть участь декілька розробників і є ймовірність, що вони залишать проект та прийдуть нові. Супроводжувати програмну систему також, як правило, доводиться новим людям, котрі не брали участі у розробці. Тому гарна

архітектура повинна надавати можливість відносно легко та швидко зрозуміти систему новим розробникам.

Не дивлячись на різноманітність критеріїв, все ж таки головною при проектуванні крупних масштабованих додатків є зниження складності. Основним шляхом для досягнення цієї цілі є *декомпозиція* програмної системи на окремі частини.

Однією з не багатьох архітектур, котрі можна умовно назвати універсальними є *модульна архітектура*. Декомпозиція на окремі модулі у даному рішенні є основною. Складна програмна система повинна складатися з більш простіших підсистем, а ті, в свою чергу, також поділяються на частини меншої степені складності і так до тих пір, поки не будуть виявлені найпростіші, безпосередньо зрозумілі, частини.

Поряд зі зниженням складності така архітектура забезпечує гнучкість системи, надає гарні можливості для масштабування, а також дозволяє підвищувати стійкість.

У процесі декомпозиції програмної системи на модулі відбувається послаблення їх залежності один від одного, що дозволяє достатнього зрозуміло організувати модульну взаємодію між собою та зовнішнім світом. Існує ряд правил, котрих слід дотримуватись використовуючи методологію декомпозиції. Поділяти систему слід ієрархічно, рис. 1. Кожна підсистема (функціональні модулі, сервіси, шари програми) повинні виконувати свою певну, конкретну функцію і в ідеальному випадку можуть бути працездатними автономно або у іншому оточенні.

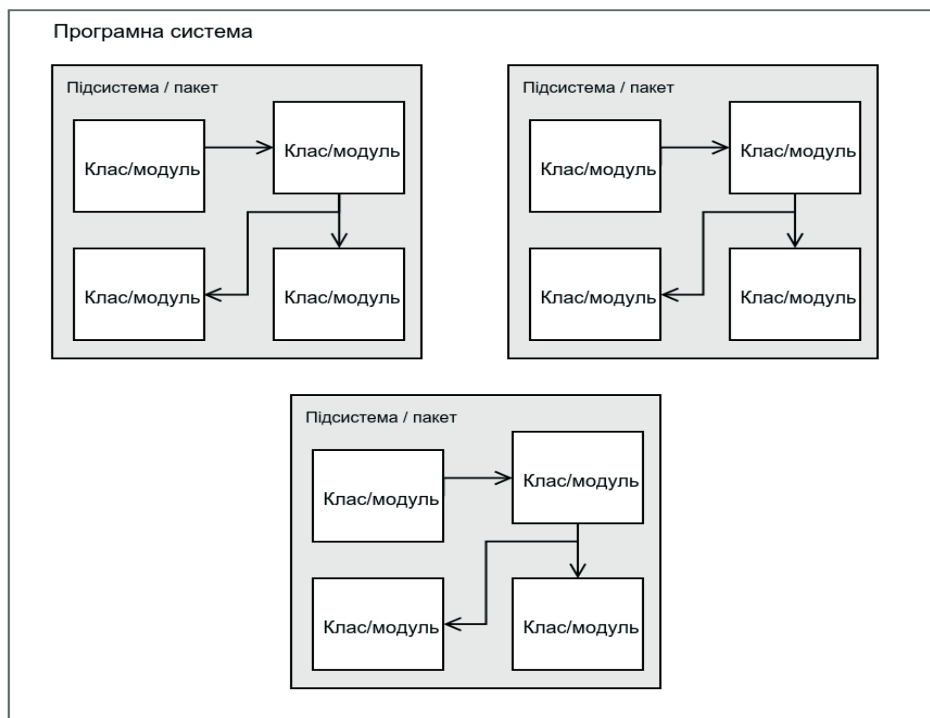

Рис. 1. Ієрархічна декомпозиція програмної системи

Результатом такого поділу є програма – конструктор, що складається з набору організованих модулів/частин, котрі взаємодіють між собою по гарно обмиркованим правилам.

Розглянемо методологію декомпозиції детальніше у сфері Інтернет-комерції, на спрощеному прикладі проектування модуля персонального кошика. Спочатку слід виявити основні функції, котрі виконує персональний кошик користувача:

– перелік продуктів/атрибутів продуктів (ціна, назва, ідентифікатор, тощо);
– розрахунок знижки для конкретного продукту;
– розрахунок вартості доставки у конкретний регіон;
– розрахунок загальної вартості, враховуючи знижки і доставку;

Виходячи з першого із принципів *SOLID* (*на кожен клас покладений один єдиний обов'язок/функція*), не важко розподілити даний модуль на окремі програмні одиниці/частини:

– Product – клас, котрий буде описувати низькорівневий (відносно сутності кошик) інтерфейс для роботи з продуктом (отримання/встановлення атрибутів);
– ShippingCalculator – клас, що відповідає за розрахунок вартості доставки, з урахуванням конкретної країни, міста та податку;
– Discount Calculator – клас, який описує інтерфейс розрахунку знижки;
– ShoppingCart – безпосередньо клас самого кошика, описуючий високорівневий інтерфейс (відносно клієнтських класів) взаємодії з ним.

У даному випадку для забезпечення високорівневого інтерфейсу для клієнтських класів, та приховування від них низькорівневого інтерфейсу класів підсистеми можна спроектувати дещо схоже на патерн «Фасад». Основна ідея цього патерну закладена у управлінні та перенаправленні запитів. Під клієнтськими класами розуміються класи – посередники між підсистемами/модулями/пакетами, котрі реалізують їх взаємодію.

UML – діаграма класів зображена на рис. 2.

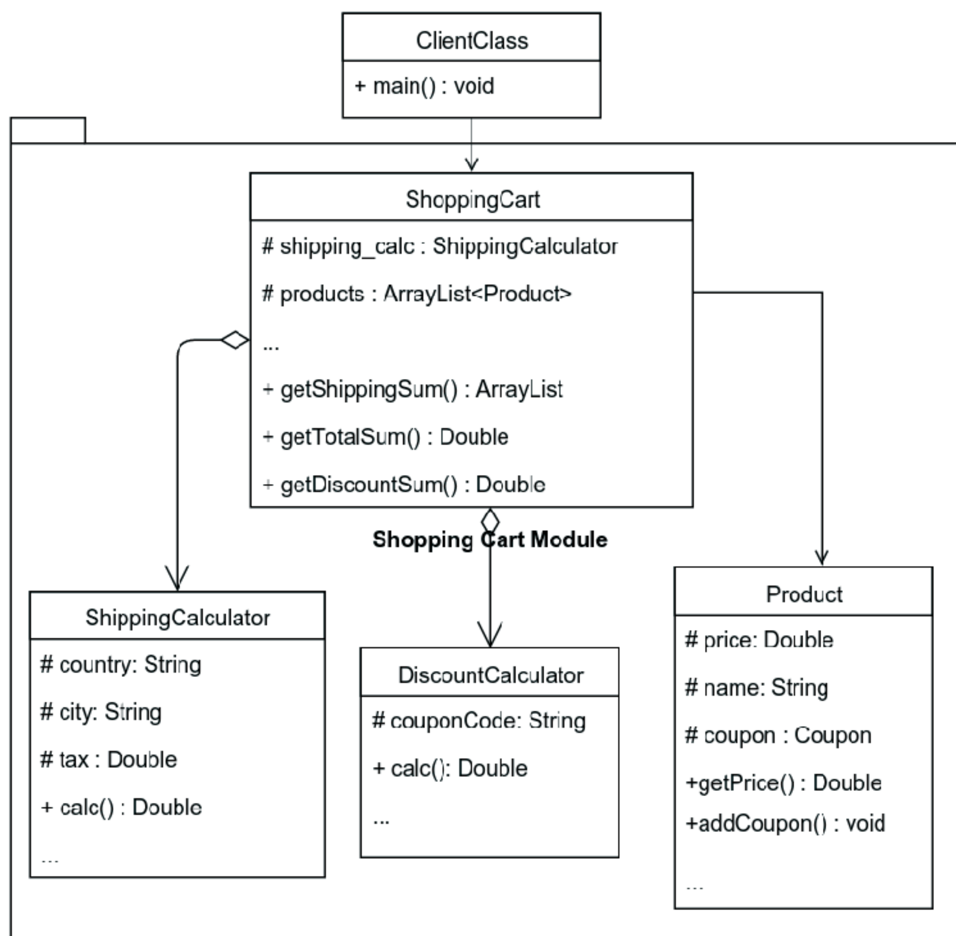

Рис. 2. UML – діаграма класів модуля персонального кошика

Слід зазначити, що дана діаграма достатньо стисла, та для простоти сприйняття не описує усіх методів, котрі можуть бути реалізовані у реальному проекті. Однак, на ній чітко видно, що кожен клас підсистеми займається своєю справою, і майже нічого один про одного не знає. А керує процесом клас ShoppingCart, з котрим і взаємодіють клієнтські класи із інших підсистем.

Кожен клас модуля можна використовувати повторно, так як кожен з них може працювати автономно або у іншому оточені. Це забезпечено пониженням кількості зв'язків між класами підсистеми та організовану взаємодію між ними.

Отже результати проектування відповідають усім, вище поставленим, критеріям достатньо стійкої архітектури додатку.

Масштабування. Отримана підсистема має можливість маштабування, за рахунок додавання нових класів, без значної зміни інших.

Повторне використання. Модуль може бути використаний у інших програмних системах або у іншому оточенні. Це справедливо також і для класів модуля.

Супроводжуваність. Дана спроектована підсистема достатньо проста та інтуїтивно зрозуміла.

Найголовнішим досягненням є *ефективність* спроектованої підсистеми, вона чітко виконує покладену на неї функцію.

Архітектура програмного забезпечення являється вже достатньо не молодим напрямом але стрімко розвивається. З ростом технічних можливостей, ростуть і можливості розробки більш складних, масштабних алгоритмів та систем, які породжують за собою достатньо цікаві і такі важливі архітектурні рішення.

Список використаних джерел

1. Шевчук А. Design Patterns via C#. Приемы объектно-ориентированного проектирования / А. Шевчук, Д. Охрименко, А. Касьянов. – 2015. – 200 с.
2. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – Addison Wesley Longman Inc., 1995. – 196 с.
3. Мартин Фаулер. Patterns of Enterprise Application Architecture, 2007.
4. Цвалина К. Инфраструктура программных проектов. Соглашения, идиомы и шаблоны для многократно используемых библиотек.NET / К. Цвалина, Б. Абрамс. – Издательство Вильямс, 2011.
5. Создание архитектуры программы [Электронный ресурс]. – Режим доступа : https://habrahabr.ru/post/276593/

Oleksandra BULGAKOVA, Pavlo KYSLYCHENKO
Mykolaiv

THE LARGE SCALABLE APPLICATIONS ARCHITECTURE IN OBJECT-ORIENTED PROGRAMMING CONTEXT

The paper present the large-scalable applications architecture in object-oriented programming context. Decomposition methodology in the e-commerce field, in example of the personal basket design module was presented.

To ensure a high-level interface for client classes, and concealing from them the low-level interface subsystem classes, proposed to use the most popular abstract architectural solutions, which are also called templates or design patterns.

Key words: software systems, architecture, object-oriented programming, software design, system application.

Александра БУЛГАКОВА, Павел КИСЛИЧЕНКО
г. Николаев

АРХИТЕКТУРА КРУПНЫХ МАСШТАБИРУЕМЫХ ПРИЛОЖЕНИЙ В КОНТЕКСТЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

В статье рассмотрена архитектура крупно-масштабируемых приложений в контексте объектно-ориентированного программирования. Приведена методология декомпозиции в сфере Интернет-коммерции, на упрощенном примере проектирования модуля персональной корзины.

Для обеспечения высокоуровневого интерфейса для клиентских классов, и сокрытие от них низкоуровневого интерфейса классов подсистемы, предложено использовать наиболее популярные абстрактные архитектурные решения, которые также называются шаблонами проектирования или паттернами.

Ключевые слова: программные системы, архитектура, объектно-ориентированное программирование, проектирование программного обеспечения, система, приложение.

Стаття надійшла до редколегії 26.02.2016