Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Лабораторна робота №9

«Взаємодія компонентів системи»

З дисципліни «**Технології розроблення програмного забезпечення**»

Виконав:

Студент групи ІА-31

Дук М. Д.

Перевірив:

Мягкий М. Ю.

Київ 2025

Мета: Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

Технічне завдання:
Архіватор (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, тощо) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Теоретичні відомості

Клієнт-серверна архітектура (Client-Server)

Це найпоширеніша модель розподілених систем, де виділяються два типи компонентів:

- **Клієнти**: ініціюють з'єднання та представляють інтерфейс користувачеві.
- **Сервери**: очікують на запити, обробляють їх та зберігають дані .

Розрізняють два основні підвиди:

- **Тонкий клієнт**: основна бізнес-логіка виконується на сервері, клієнт відповідає лише за відображення (наприклад, класичні веб-сайти) .

- **Товстий клієнт**: значна частина логіки виконується на стороні клієнта, сервер використовується переважно для зберігання даних (наприклад, десктопні ігри, мобільні додатки) . Взаємодія часто організовується через трирівневу структуру: клієнт, сервер застосунків (middleware) та сервер баз даних.

Сервіс-орієнтована архітектура (SOA)

SOA (Service-Oriented Architecture) — це модульний підхід до розробки ПЗ, що базується на використанні розподілених, слабко пов'язаних сервісів. Ключові особливості:

- **Слабка зв'язаність** (Loose Coupling): **сервіси незалежні один від одного.**
- **Стандартизація:** взаємодія відбувається через чітко визначені інтерфейси та протоколи (часто HTTP, SOAP, REST).
- **Повторне використання**: сервіси надають бізнес-функції, які можуть бути використані різними додатками. Часто для обміну повідомленнями в SOA використовується корпоративна сервісна шина (Enterprise Service Bus — ESB).

Peer-to-Peer (P2P) архітектура

Peer-to-Peer (P2P) архітектура — це модель мережевої взаємодії, в якій кожен вузол (комп'ютер або пристрій) є одночасно клієнтом і сервером. У цій архітектурі всі вузли мають рівні права та можливості для обміну даними, ресурсами або виконання завдань. Основні принципи:

- **Децентралізація:** відсутність центрального сервера, що зменшує залежність від одного вузла.
- **Рівноправність вузлів:** кожен вузол може виконувати одночасно функції клієнта (отримувати ресурси) і сервера (надавати ресурси).

Хід Роботи

Проєкт застозунку Архіватора – це десктопна програма, що працює локально. Для реалізації вимог лабораторної роботи, було додано мережеву взаємодію.

Проаналізувавши структуру додатку, мною, відповідно до завдання, було обрано архітектуру Peer-to-Peer. Вона дозволить користувачам обмінюватися архівами напряму без проміжного сервера, що логічно для десктопного архіватора.

Для цього, було розроблено клас P2PFileTransferService, який інкапсулює логіку роботи з сокетами (TcpListener для прийому, TcpClient для відправки). Код написано з використанням асинхронності (async/await) для забезпечення неблокуючої роботи UI.

Новий сервіс інтегровано з існуючим фасадом ArchiveManager для автоматичної перевірки цілісності (Checksum) отриманих файлів.

Сутність реалізації

Реалізація Peer-to-Peer в проєкті виконана шляхом додавання нового сервісу в шар ArchiverCore. Цей сервіс дозволяє екземпляру програми діяти як сервер (очікувати на вхідні файли) та як клієнт (надсилати файли іншому піру).

Нові компоненти

Додано клас P2PFileTransferService у простір імен ArchiverCore.Services.

Фрагменти програмного коду

1. **Сервіс P2P передачі (P2PFileTransferService.cs)** Цей клас реалізує логіку "рівноправного вузла":

```
using System.Net;
using System.Net.Sockets;

namespace ArchiverCore.Services;

/// <summary>
/// Сервіс для Peer-to-Peer обміну архівами.
/// Реалізує роль "Сервера" (прийом файлів) та "Клієнта" (відправка файлів) в одному
```

```
класі.
/// </summary>
public class P2PFileTransferService
{
    private TcpListener? _listener;
    private readonly int _port;
    private CancellationTokenSource? _cts;

    // Подія, що сповіщає про отримання файлу
    public event Action<string>? FileReceived;

    public P2PFileTransferService(int port = 5000)
    {
        _port = port;
    }

    /// <summary>
    /// Запускає режим прослуховування (роль Сервера).
    /// </summary>
    public void StartListening(string saveDirectory)
    {
        _cts = new CancellationTokenSource();
        Task.Run(async () => await ListenLoopAsync(saveDirectory, _cts.Token));
    }

    private async Task ListenLoopAsync(string saveDirectory, CancellationToken token)
    {
        _listener = new TcpListener(IPAddress.Any, _port);
        _listener.Start();

        try
        {
            while (!token.IsCancellationRequested)
            {
                // Очікуємо підключення іншого піра
                using var client = await _listener.AcceptTcpClientAsync(token);
                using var networkStream = client.GetStream();

                // Простий протокол: спочатку читаємо довжину імені файлу, ім'я, потім
дані
                // (Спрощена реалізація для прикладу)
                var fileName = $"received_archive_{DateTime.Now.Ticks}.zip";
                var savePath = Path.Combine(saveDirectory, fileName);

                using var fileStream = File.Create(savePath);
                await networkStream.CopyToAsync(fileStream, token);

                // Сповіщаємо підписників (наприклад, UI або ArchiveManager)
                FileReceived?.Invoke(savePath);
            }
        }
        catch (OperationCanceledException) { }
        finally
        {
            _listener.Stop();
        }
    }

    /// <summary>
    /// Відправляє файл іншому піру (роль Клієнта).
    /// </summary>
    public async Task SendFileAsync(string targetIp, int targetPort, string filePath)
    {
        using var client = new TcpClient();
        await client.ConnectAsync(IPAddress.Parse(targetIp), targetPort);

        using var networkStream = client.GetStream();
        using var fileStream = File.OpenRead(filePath);
```

```
            await fileStream.CopyToAsync(networkStream);
        }

        public void Stop()
        {
            _cts?.Cancel();
            _listener?.Stop();
        }
    }
```

2. **Інтеграція в Facade (ArchiveManager)** Ми розширюємо фасад, щоб він міг обробити отриманий через P2P файл (наприклад, перевірити його контрольну суму, використовуючи вже існуючі механізми):

```
    /// <summary>
    /// Обробляє файл, отриманий через P2P мережу: перевіряє цілісність та реєструє в БД.
    /// </summary>
    public async Task ProcessReceivedP2PFileAsync(string filePath)
    {
        // 1. Використовуємо існуючу фабрику для визначення типу
        if (!ArchiveFactory.IsSupported(filePath))
        {
            throw new NotSupportedException("Received file format is not supported.");
        }

        // 2. Використовуємо існуючий Visitor для розрахунку хешу
        var checksum = await CalculateChecksumAsync(filePath);

        // 3. Логуємо подію отримання через P2P
        await LogOperationAsync(filePath, "P2P_RECEIVE", "Success", $"Received from peer.
    Checksum: {checksum}");

        // 4. Оновлюємо метадані в базі
        var adapter = ArchiveFactory.CreateFor(filePath);
        await UpdateArchiveMetadataAsync(filePath, checksum, adapter,
    CancellationToken.None);
    }
```

Принцип роботи:

1. Користувач А запускає програму. P2PFileTransferService стартує і слухає порт (Сервер).
2. Користувач Б обирає файл архіву та вводить IP користувача А.
3. P2PFileTransferService користувача Б підключається (як Клієнт) і передає байти.
4. На стороні А спрацьовує подія FileReceived.
5. MainForm перехоплює подію і викликає ArchiveManager.ProcessReceivedP2PFileAsync.
6. ArchiveManager запускає ChecksumVisitor (існуючий код) для валідації отриманого файлу.

Переваги:

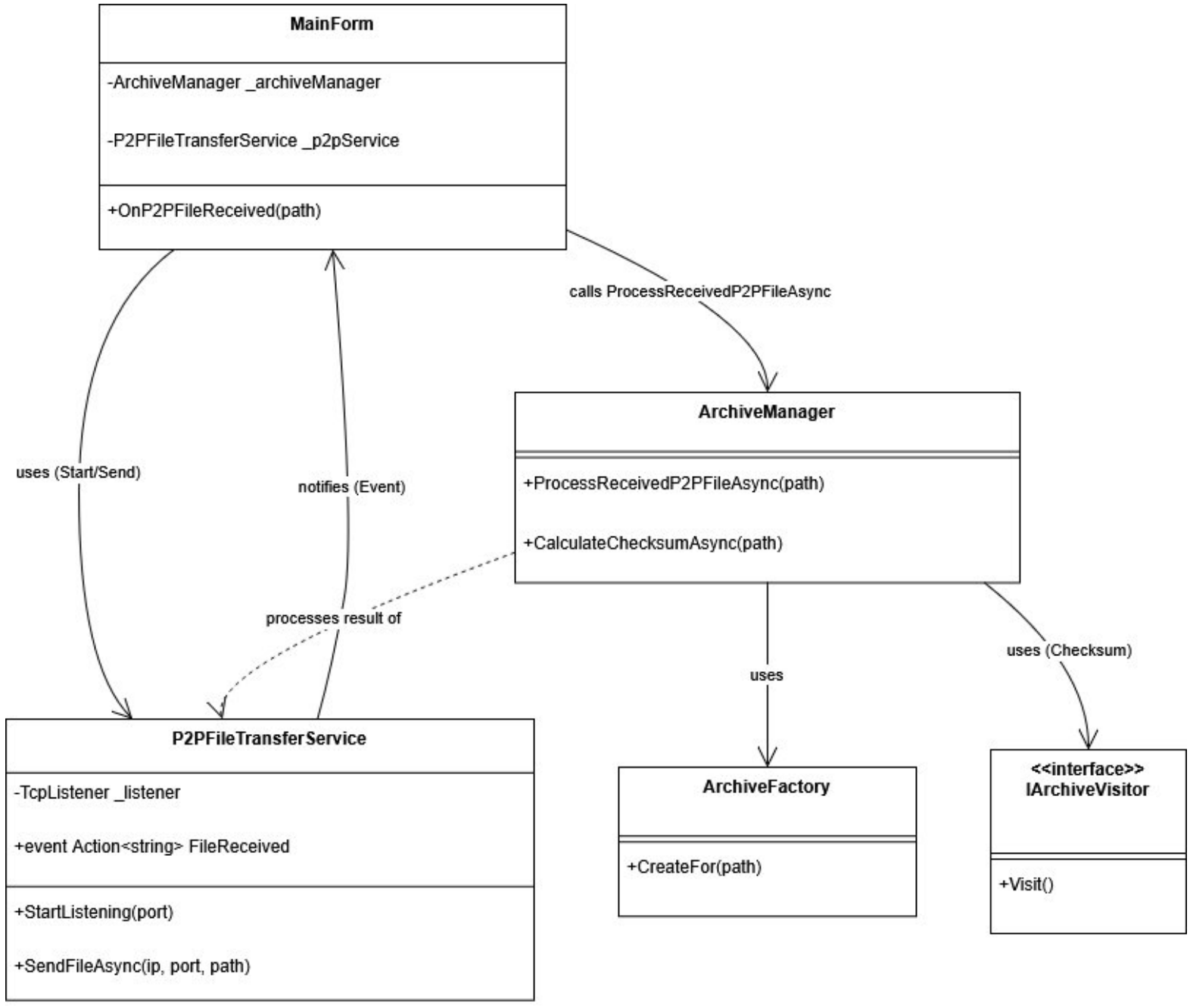
1. **Автономність:** Користувачі можуть передавати архіви в локальній мережі або через інтернет без налаштування FTP-сервера чи використання хмарних сховищ.
2. **Масштабованість:** Навантаження розподіляється між учасниками мережі, а не лягає на центральний вузол.
3. **Повторне використання коду:** Логіка перевірки та обробки отриманих файлів повністю використовує існуючі класи (ArchiveFactory, ChecksumVisitor, ArchiveManager), що підтверджує якість архітектури.

Недоліки:

1. **Безпека:** Відкриття порту для прослуховування (StartListening) створює потенційну вразливість. Необхідна реалізація автентифікації та шифрування.
2. **NAT/Firewall:** У реальних умовах з'єднання може блокуватися брандмауерами або NAT, що вимагає додаткових технік (наприклад, UPnP або STUN), які не реалізовані в рамках лабораторної.
3. **Синхронізація:** Відсутній центральний реєстр пірів, тому користувач повинен знати IP-адресу іншого користувача.

Діаграма Класів реалізованого шаблону

На діаграмі відображено доданий клас P2PFileTransferService та його зв'язок з існуючим фасадом ArchiveManager та формою.



Опис діаграми:

- P2PFileTransferService є незалежним компонентом, який керує мережевим вводом/виводом.
- MainForm підписується на події P2P сервісу.
- При отриманні файлу потік управління передається в ArchiveManager.
- ArchiveManager використовує існуючі ArchiveFactory та IArchiveVisitor для інтеграції нового файлу в систему (розрахунок хешу, запис в БД), демонструючи, що P2P — це лише ще одне джерело даних для системи.

Висновок: У ході виконання лабораторної роботи було досліджено принципи побудови розподілених систем. В існуючий C# проект архіватора було успішно інтегровано концепцію **Peer-to-Peer** взаємодії. Реалізований клас P2PFileTransferService дозволяє кожному екземпляру програми виступати і сервером (приймати файли), і клієнтом (надсилати файли). Інтеграція була проведена без порушення існуючої архітектури: мережевий шар відповідає лише за транспорт, а бізнес-логіка (валідація, збереження) залишається відповідальністю ArchiveManager та відповідних патернів (Visitor, Strategy). Це демонструє гнучкість модульної архітектури.

Питання до лабораторної роботи

1. Що таке клієнт-серверна архітектура?
   Це модель, де виділяється два види додатків: клієнти (представляють додаток користувачеві, ініціюють запити) і сервери (зберігають і обробляють дані, відповідають на запити) .

2. Розкажіть про сервіс-орієнтовану архітектуру (SOA).
   Це модульний підхід, заснований на використанні розподілених, слабо пов'язаних сервісів зі стандартизованими інтерфейсами. Сервіси надають бізнес-функції та можуть бути повторно використані різними додатками.

3. Якими принципами керується SOA?
   Слабка зв'язаність (loose coupling), стандартизація інтерфейсів, повторне використання

сервісів, незалежність від платформи реалізації.

4. Як між собою взаємодіють сервіси в SOA?
   Сервіси взаємодіють за рахунок обміну повідомленнями (зазвичай через HTTP/SOAP/REST) без створення спеціальних жорстких інтеграцій. Часто використовується шина даних (ESB).

5. Як розробники взнають про існуючі сервіси і як робити до них запити?
   Згідно SOA, сервіси реєструються у спеціальних каталогах (реєстрах сервісів), де розробники можуть знайти їх опис (WSDL/Swagger) та кінцеві точки доступу.

6. У чому полягають переваги та недоліки клієнт-серверної моделі?

   - Переваги: Простота адміністрування та захисту даних (все на сервері), централізоване оновлення бізнес-логіки.
   - Недоліки: Сервер є вузьким місцем (single point of failure), високе навантаження на сервер.

7. У чому полягають переваги та недоліки однорангової моделі взаємодії (P2P)?

   - Переваги: Відсутність центрального сервера (децентралізація), масштабованість, розподіл ресурсів, стійкість до збоїв окремих вузлів .
   - Недоліки: Складність контролю даних, проблеми з безпекою, складність пошуку ресурсів та синхронізації.

8. Що таке мікро-сервісна архітектура?
   Це стиль розробки, де додаток складається з набору малих служб (мікросервісів), кожен з яких виконується у своєму процесі, орієнтований на певну бізнес-можливість і може розгортатися незалежно .

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?
   HTTP/HTTPS (REST), WebSockets, AMQP (черги повідомлень), gRPC.

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли ми в проєкті між шаром веб-контролерів та шаром доступу до даних реалізуємо шар бізнес-логіки у вигляді сервісів?

    Ні, це скоріше просто багатошарова архітектура (Layered Architecture) або використання патерну "Service Layer" всередині моноліту. SOA передбачає, що сервіси є розподіленими і незалежними компонентами, які взаємодіють через мережу, а не просто класами в одному проекті.