



Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №9
Технології розроблення програмного забезпечення
«Взаємодія компонентів системи. Peer-to-Peer»
«10. VCS *all-in-one*»

Виконав
студент групи IA-32:
Варивода К. С.

Перевірив:
Мягкий Михайло Юрійович

Зміст

Зміст.....	2
Варіант.....	2
Теоретичні відомості.....	3
Хід роботи.....	4
Висновок.....	5
Відповіді на питання	6

Варіант

10. VCS all-in-one (iterator, adapter, factory method, facade, visitor, p2p)

Клієнт для всіх систем контролю версій повинен підтримувати основні команди і дії (commit, update, push, pull, fetch, list, log, patch, branch, merge, tag) для 3-х основних систем управління версіями (svn, git, mercurial), а також мати можливість вести реєстр репозиторіїв (і їх типів) і відображати дерева фіксації графічно.

Теоретичні відомості

Peer-to-Peer (P2P) архітектура – це модель мережової 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. На відміну від клієнт-серверної моделі, де є чітке розділення на клієнти й сервери, P2P-мережа дозволяє учасникам взаємодіяти безпосередньо, без необхідності в централізованому сервері.

Основними принципами P2P-архітектури є:

- Децентралізація – відсутність центрального сервера, що зменшує залежність від одного вузла, підвищуючи стійкість мережі до збоїв і атак.
- Рівноправність вузлів – кожен вузол може виконувати одночасно функції клієнта (отримувати ресурси) і сервера (надавати ресурси).
- Розподіл ресурсів – вузли надають доступ до своїх власних ресурсів, таких як обчислювальна потужність, дисковий простір або файли.
- Основними сферами де peer-to-peer архітектура знайшла широке застосування є файлообмінники (BitTorrent), криптовалюти та інші блокчайн-технології, інтернет телефонія та відеоконференції (Skype, Zoom), розподілені обчислення (SETI@home, BOINC).
- До основних проблемних зон можна віднести безпеку, синхронізацію даних та пошук ресурсів. Через централізацію складно контролювати дані, які передаються. Ефективність пошуку даних знижується зі збільшенням кількості вузлів у мережі і для підвищення ефективності пошуку потрібно застосовувати спеціальні алгоритми.

Релевантність до проекту:

Архітектура Peer-to-Peer є фундаментальною для моого проєкту, оскільки інтегровані системи Git та Mercurial за своєю природою є розподіленими, де кожен репозиторій виступає рівноправним вузлом (піром). У моїй реалізації це відображене через механізм прямої синхронізації даних між локальними каталогами без необхідності використання проміжного центрального сервера. Це дозволяє кожному екземпляру репозиторію функціонувати автономно, зберігаючи повну історію змін, що значно підвищує надійність та відмовостійкість системи. Такий підхід забезпечує гнучкість робочого процесу, дозволяючи користувачеві синхронізувати будь-які два сумісні репозиторії між собою напряму, імітуючи децентралізовану взаємодію розробників. Отже, використання P2P демонструє відмову від жорсткої клієнт-серверної ієрархії на користь рівноправних горизонтальних зв'язків, що є ключовою перевагою сучасних DVCS.

Хід роботи

```
8 import org.eclipse.jgit.transport.RefSpec;
9 import org.eclipse.jgit.treewalk.TreeWalk;
10 import vcs.repository.classes.*;
11 import vcs.repository.factories.DaoFactory;
12
13 import java.io.File;
14 import java.io.FileInputStream;
15 import java.io.IOException;
16 import java.util.ArrayList;
17 import java.util.List;
18 import java.util.Set;
19
20 public class GitAdapter implements VcsAdapter { 5 usages ± mavrlq
21     DaoFactory daoFactory; 13 usages
22
23     > public GitAdapter(DaoFactory daoFactory) { this.daoFactory = daoFactory; }
24
25     @ private Git openRepo(Repository repo) throws IOException { 13 usages ± mavrlq
26         return Git.open(new File(repo.getUrl())); // repo.getUrl() повертає шлях до папки (напр. "C:/my_repos/repo1")
27     }
28
29     @Override 2 usages ± mavrlq
30     public void createRepository(Repository repo) {
31         File repoDir = new File(repo.getUrl());
32
33         File directory = new File(repo.getUrl());
34         if (directory.exists()) {
35             if (directory.isDirectory() && new File(directory, child: ".git").exists()) {
36                 System.out.println("У папці '" + repo.getUrl() + "' вже існує ініціалізований Git-репозиторій");
37                 return;
38             }
39         } else {
40             repoDir.mkdir();
41             System.out.println("Папки за шляхом '" + repo.getUrl() + "' не існує, створюємо");
42             createRepository(repo);
43         }
44     }
45
46     try {
47 }
```

Рис 1 – частина класу GitAdapter

На рисунку 1 видно, що для роботи я використовую локальні дані без використання окремого серверу. Пірами у мене виступають різні папки

```

51
52
53
54 @Override
55     public void createRepository(Repository repo) {
56         File repoDir = new File(repo.getUrl());
57
58         if (!repoDir.exists()) {
59             repoDir.mkdirs();
60         }
61
62         if (new File(repoDir, ".hg").exists()) {
63             System.out.println("Mercurial репозиторій вже існує в " + repo.getUrl());
64             return;
65         }
66
67         runCommand(repoDir, ...command: "hg", "init");
68
69         try {
70             new File(repoDir, ".hgignore").createNewFile();
71         } catch (IOException e) {
72             e.printStackTrace();
73         }
74
75         System.out.println("Created Mercurial repo at " + repoDir);
76         addToDb(repo);
77     }
78
79 @Override
80     public Commit commit(Repository repo, User user, String message) {
81         File repoDir = new File(repo.getUrl());
82
83         List<String> branchOutput = runCommand(repoDir, ...command: "hg", "branch");
84         System.out.println(branchOutput);
85         String currentBranch = branchOutput.isEmpty() ? "unknown" : branchOutput.getFirst().trim();
86         System.out.println("Committing to Mercurial branch: " + currentBranch);
87
88         runCommand(repoDir, ...command: "hg", "addremove");
89
90         runCommand(repoDir, ...command: "hg", "commit", "-m", message, "-u", user.getUsername());
91
92         addToDb(repo);
93
94         return getLastCommit(repo);
95     }

```

Рис 2 – частина класу MercurialAdapter

Аналогічно до рисунку 1 на рисунку 2 ми використовуємо внутрішню файлову систему для роботи репозиторію.

Висновок

Отже, після виконання даної лабораторної роботи було детально вивчено архітектурний стиль Peer-to-Peer, його принципи децентралізації, сильні сторони у забезпеченні автономності та відмовостійкості вузлів, а також специфіку вирішення конфліктів при синхронізації даних. Для закріплення отриманих знань було імплементовано механізми P2P-взаємодії у розроблюваному проєкті «VCS All-In-One». Було проведено аналіз функціональних потреб системи та визначено необхідність забезпечення прямого обміну версіями коду між локальними репозиторіями без використання проміжного центрального сервера. Реалізація методів push та pull в адаптерах дозволила налаштувати прямий потік даних між файловими системами, фактично перетворивши кожен репозиторій на рівноправний вузол, здатний як ініціювати передачу змін, так і приймати їх від інших вузлів.

Відповіді на питання

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура — це модель мережової взаємодії, в якій завдання або навантаження розподіляються між постачальниками послуг, які називаються серверами, і замовниками послуг, які називаються клієнтами.

- Клієнт: Ініціює з'єднання, надсилає запит і чекає на відповідь (наприклад, веб-браузер).
- Сервер: Очікує на з'єднання, обробляє запити, виконує обчислення або звертається до бази даних і надсилає відповідь (наприклад, веб-сервер Apache або Nginx).

2. Розкажіть про сервіс-орієнтовану архітектуру (SOA).

Service-Oriented Architecture (SOA) — це архітектурний стиль, де програмне забезпечення будується з окремих, незалежних блоків, які називаються сервісами. Ці сервіси надають певну бізнес-функціональність і можуть спілкуватися між собою через мережу за допомогою стандартних протоколів. Головна ідея SOA — дозволити різним системам (написаним на різних мовах, розгорнутим на різних платформах) взаємодіяти як єдине ціле, часто використовуючи Enterprise Service Bus (ESB) для координації.

3. Якими принципами керується SOA?

SOA базується на таких ключових принципах:

- Стандартизований контракт сервісу: Сервіси мають чіткий опис інтерфейсу (WSDL, OpenAPI), який визначає, як з ними взаємодіяти.
- Слабка зв'язаність (Loose Coupling): Сервіси мінімально залежать один від одного. Зміна в одному сервісі не повинна ламати інші.
- Абстракція сервісу: Логіка всередині сервісу прихованана від клієнта; клієнт знає лише інтерфейс.
- Можливість повторного використання (Reusability): Сервіси проєктуються так, щоб їх можна було використовувати в різних бізнес-процесах.
- Відсутність стану (Statelessness): Сервіси в ідеалі не повинні зберігати інформацію про стан сесії між запитами.
- Автономність: Сервіс самостійно контролює своє середовище та ресурси.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють через мережу, використовуючи механізм передачі повідомлень. Існує два основні способи:

1. Пряма взаємодія (Point-to-Point): Сервіс А викликає Сервіс Б напряму (через REST або SOAP).
2. Через посередника (ESB - Enterprise Service Bus): Це "розумна шина", яка маршрутизує повідомлення, трансформує формати даних та забезпечує зв'язок. Сервіс А надсилає повідомлення в шину, а шина доставляє його Сервісу Б.

5. Як розробники знають про існуючі сервіси і як робити до них запити?

Для цього використовується механізм Service Discovery (Виявлення сервісів) та Реєстр сервісів (Service Registry).

- Реєстр: Це каталог, де зберігається інформація про всі доступні сервіси, їхні адреси та контракти (описи методів).
- Контракт: Розробники читають документацію (контракт) сервісу (наприклад, файл WSDL для SOAP або Swagger/OpenAPI для REST), щоб зрозуміти структуру запиту та відповіді.

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

Переваги:

- Централізація: Легке керування даними, безпекою та оновленнями на стороні сервера.
- Масштабованість сервера: Можна збільшувати потужність сервера для обслуговування більшої кількості клієнтів.
- Доступність: Ресурси доступні різним клієнтам одночасно.

Недоліки:

- Єдина точка відмови: Якщо сервер падає, робота зупиняється для всіх клієнтів.
- Перевантаження: Велика кількість запитів може створити "пляшкове горлечко" (bottleneck).
- Вартість: Потужні сервери та їх обслуговування коштують дорого.

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

Переваги:

- Відсутність єдиної точки відмови: Якщо один вузол випадає, мережа продовжує працювати.
- Масштабованість: Чим більше користувачів (вузлів), тим потужніша мережа.
- Економічність: Використовуються ресурси користувачів, а не дорогі виділені сервери.

Недоліки:

- Складність керування: Немає центрального адміністратора; важко оновлювати ПЗ або керувати доступом.
- Безпека: Важче гарантувати безпеку даних та захист від вірусів.
- Нестабільність: Вузли можуть підключатися і відключатися в будь-який момент.

8. Що таке мікросервісна архітектура?

Мікросервісна архітектура — це підхід (різновид SOA), при якому програма розробляється як набір невеликих сервісів, кожен з яких:

- Працює у власному процесі.
- Відповідає за одну конкретну бізнес-функцію (Single Responsibility).
- Має власну базу даних (бажано).
- Може бути розгорнутий незалежно від інших.
- Спілкується з іншими через легковагі механізми (зазвичай HTTP).

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?

Найпоширеніші протоколи:

1. HTTP/HTTPS (REST): Найпопулярніший синхронний протокол, використовує JSON.
2. gRPC: Високопродуктивний протокол від Google (на базі HTTP/2), використовує Protobuf (бінарний формат).
3. AMQP / Kafka Protocol: Для асинхронного обміну повідомленнями через брокери (RabbitMQ, Apache Kafka).

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

Ні, це не є SOA в архітектурному сенсі.

Це називається шаруватою архітектурою (Layered Architecture) або використанням патерну "Service Layer".

- У цьому випадку ваші "сервіси" — це просто класи або інтерфейси всередині однієї програми (моноліту). Виклики методів відбуваються в пам'яті (in-memory).
- Справжня SOA (або мікросервіси) передбачає, що сервіси є розподіленими компонентами, які працюють окремо і спілкуються через мережу.

