Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

Технології розроблення програмного забезпечення

РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ:CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Варіант №5

Виконав
студент групи ІА-13:
Засінець А. Є.

Київ 2023

Тема: РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Хід роботи:

Тема проєкту:

..5 Аудіо редактор (singleton, adapter, observer, mediator, composite, client-server)

Аудіо редактор повинен володіти наступним функціоналом: представлення аудіо даних будь-якого формату в WAVE-формі, вибір і подальші операції копіювання / вставки / вирізання / деформації по сегменту аудіозапису, можливість роботи з декількома звуковими доріжками, кодування в найбільш поширених форматах (ogg, flac, mp3).

Завдання:

1. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

Для початку визначимо та опишемо кожен із шаблонів, що розглядаються та застосовуються під час виконання даної лабораторної роботи.

Клієнт-серверні додатки

Рис. 1 - Структура клієнт-серверного додатка

Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти.

Тонкий клієнт - клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт - набір форм відображення і канал зв'язку з сервером.

У такому варіанті використання дуже велике навантаження лягає на сервер. Раніше такий варіант був доступний, оскільки обчислень було небагато, термінали були дорогими і як пра-вило йшли з урізаним обчислювальним пристроєм. У нинішній час зі зростанням обчислювальних можливостей клієнтських машин має сенс частину обчислень перекладати на клієнтські комп'ютери.

Товстий клієнт - антипод тонкого клієнта, більшість логіки обробки даних містить на стороні клієнта. Це сильно розвантажує сервер. Сервер в таких випадках зазвичай працює лише як точка доступу до деякого іншого ресурсу (наприклад, бази даних) або сполучна ланка з іншими клієнтськими комп'ютерами.

У моделі клієнт-серверної взаємодії також важливе місце займає модель «підписки / видачі». Комп'ютери клієнтів можуть «підписуватися» серверу на певний набір подій (поява нових користувачів, зміна даних тощо), а сервер в свою чергу сповіщає клієнтські комп'ютери про походження цих подій. Даний спосіб взаємодії реалізується за допомогою шаблону «оглядач»; він несе велике навантаження на сервер (необхідно відстежувати хто на що підписався) і на канали зв'язку (багато повторюваних даних відправляються на різні точки в мережі). З іншого боку, це дуже зручно для клієнтських машин для організації оновлення даних без їх повторного перезапросу.

Peer-to-peer

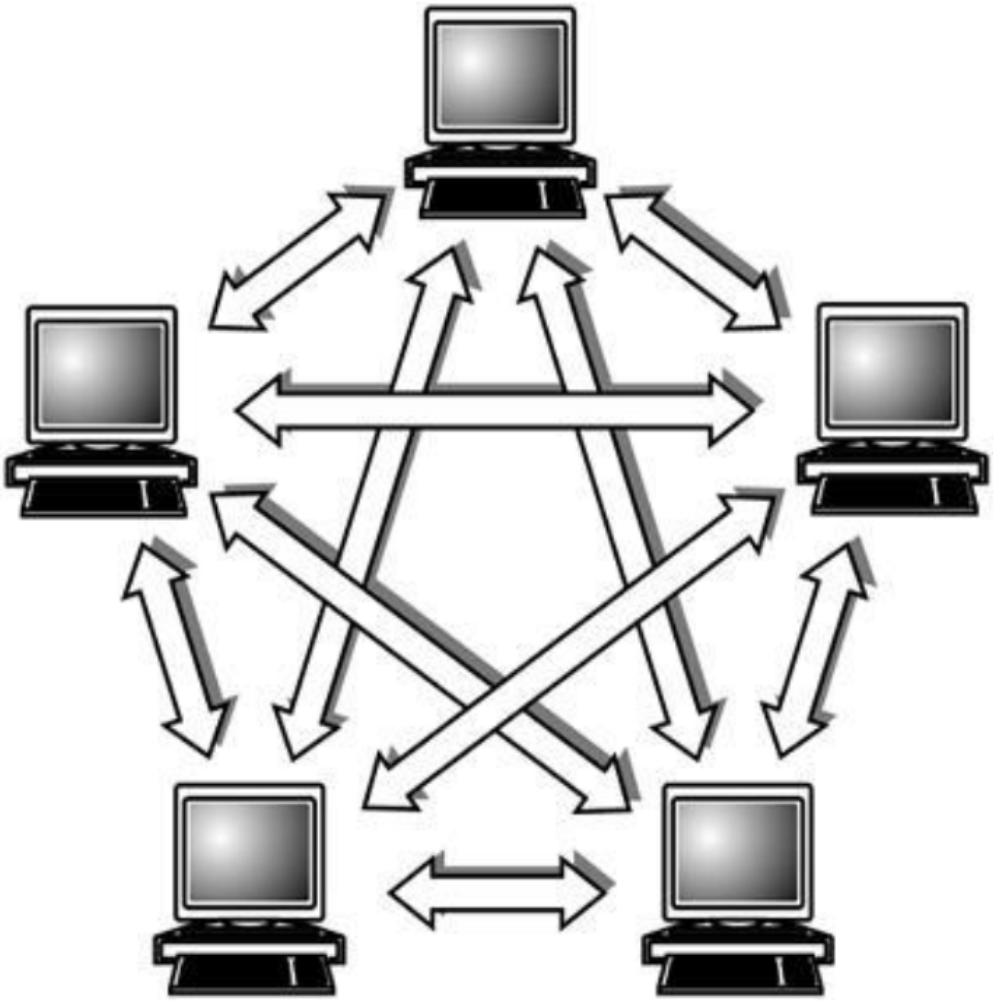

Рис. 2 - Структура peer-to-peer додатку

Модель взаємодії додатків типу peer - to - peer має на увазі рівноправ'я клієнтських програм і відсутність серверної програми. Усі клієнтські програми контактують між собою для виконання спільних цілей. У таких мережах найчастіше виникають наступні проблеми: синхронізація даних, пошук клієнтських застосувань.

Для пошуку клієнтських застосувань може використовуватися одна загальна адреса, що містить набір адрес підключених клієнтів. Це в деякому роді нагадує сервер. Проте сервер є лише центром сходження безлічі клієнтів, а адреси можуть зберігатися в самих клієнтських застосуваннях. Це називається структуровані однорангові мережі.

Сервіс-орієнтована архітектура

Сервіс-орієнтована архітектура (SOA, англ. service-oriented architecture) —- модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов’язаних (англ. Loose coupling) замінних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами.

Програмні комплекси, розроблені відповідно до сервіс-орієнтованою архітектурою, зазвичай реалізуються як набір веб-служб, взаємодіючих за протоколом SOAP, але існують і інші реалізації (наприклад, на базі jini, CORBA, на основі REST).

SaaS (англ. Software as a service - програмне забезпечення як послуга; також англ. Software on demand - програмне забезпечення на вимогу) - бізнес-модель продажу та використання програмного забезпечення, при якій постачальник розробляє веб-додаток і самостійно керує ним, надаючи замовнику доступ до програмного забезпечення через Інтернет. Основна перевага моделі SaaS для споживача послуги полягає у відсутності витрат, пов'язаних з установкою, оновленням і підтримкою працездатності обладнання і працюючого на ньому програмного забезпечення.

З точки зору розробника пропрієтарного програмного забезпечення модель SaaS дозволяє ефективно боротися з неліцензійним програмним забезпеченням, оскільки ПО як таке не потрапляє до кінцевих замовників.

Мікро-сервісна архітектура

Сама назва дає зрозуміти, що архітектура мікрослужб є підходом до створення серверного додатку як набору малих служб. Це означає, що архітектура мікрослужб головним чином орієнтована на серверну частину, не дивлячись на те, що цей підхід так само використовується для зовнішнього інтерфейсу, де кожна служба виконується в своїому процесі і взаємодіє з іншими службами за такими протоколами, як HTTP/HTTPS, WebSockets чи AMQP.

2. Застосування одного з розглянутих шаблонів при реалізації програми.

Як зазначено в завданні, необхідно реалізувати архітектуру client-server.

Отже, створимо сервер та клієнт.

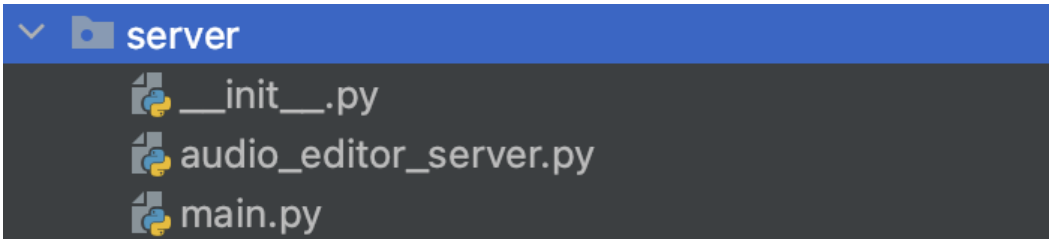

Рис. 3 - Структура сервера

У файлі audio_editor_server.py знаходяться головні методи для роботи з аудіо.Також у цьому файлі реалізовано команди, за допомогою яких клієнт буде користуватися застосунком.

```python
from cur.adapter.audio_adapter import AudioAdapter


class AudioEditorServer:
    def __init__(self, observer):
        self.audio_data = None
        self.audio_tracks = []
        self.adapter = AudioAdapter()
        self.clipboard = None
        self.current_track = None
        self.observer = observer
        self.track_names = {}
        self.commands = {
            "open": self.open_audio,
            "copy": self.copy_audio,
            "paste": self.paste_audio,
            "cut": self.cut_audio,
            "encode": self.encode_audio,
            "add_track": self.add_audio_track,
            "deform": self.deform_audio,
            "mix": self.mix_audio_tracks,
            "split": self.split_audio_track,
            "move": self.move_audio_track,
            "copy-segment": self.copy_segment,
            "paste-segment": self.copy_segment,
            "cut-segment": self.cut_segment,
            "select_segment": self.select_segment,
            "pitch_shift": self.pitch_shift,
            "sat": self.select_audio_track,
            "rat": self.remove_audio_track_by_name,
            "lat": self.list_audio_tracks,
            "sct": self.save_current_track,
            "duration": self.get_audio_track_duration,
        }

    def list_audio_tracks(self):
        try:
            track_list = list(self.track_names.keys())
            track_list_message = "\n".join(track_list)
            self.observer.notify("list_audio_tracks", track_list_message)
            if track_list:
                return track_list_message
            else:
                return "No active audio tracks"
```

```
        except Exception as e:
            return f"Error getting list of audio tracks: {str(e)}"

    def mix_audio_tracks(self, new_track_name, *track_names_or_indices):
        try:
            tracks_to_mix = []
            for name_or_index in track_names_or_indices:
                if name_or_index.isdigit():
                    index = int(name_or_index)
                    if 0 <= index < len(self.audio_tracks):
                        tracks_to_mix.append(self.audio_tracks[index])
                elif name_or_index in self.track_names:
                    index = self.track_names[name_or_index]
                    if 0 <= index < len(self.audio_tracks):
                        tracks_to_mix.append(self.audio_tracks[index])
            if len(tracks_to_mix) >= 2:
                mixed_track = sum(tracks_to_mix)
                self.audio_tracks.append(mixed_track)
                self.track_names[new_track_name] = len(self.audio_tracks) - 1
                self.observer.notify("TracksMixed", new_track_name)
                return ''
            else:
                return "At least two tracks (by name or index) are required for mixing."
        except Exception as e:
            return f"Error mixing tracks: {str(e)}"

    def split_audio_track(self, track_name_or_index, start_time, end_time):
        try:
            if track_name_or_index.isdigit():
                index = int(track_name_or_index)
                if 0 <= index < len(self.audio_tracks):
                    track = self.audio_tracks[index]
                else:
                    return "Invalid audio track index"
            elif track_name_or_index in self.track_names:
                index = self.track_names[track_name_or_index]
                if 0 <= index < len(self.audio_tracks):
                    track = self.audio_tracks[index]
                else:
                    return "Invalid audio track index"
            else:
                return "Track with the specified name or index not found"
            start_time = int(start_time)
            end_time = int(end_time)
```

```
            split_tracks = [track[start_time:end_time], track[:start_time] + track[end_time:]]
            del self.audio_tracks[index]
            self.audio_tracks.extend(split_tracks)
            self.observer.notify("split_audio_track")
            return ''
        except Exception as e:
            return f"Error splitting track: {str(e)}"

    def get_audio_track_duration(self, track_name_or_index):
        try:
            if track_name_or_index.isdigit():
                index = int(track_name_or_index)
                if 0 <= index < len(self.audio_tracks):
                    duration = len(self.audio_tracks[index])
                    self.observer.notify("DurationObtained",duration)
                    return ''
                else:
                    return "Invalid audio track index"
            elif track_name_or_index in self.track_names:
                index = self.track_names[track_name_or_index]
                if 0 <= index < len(self.audio_tracks):
                    duration = len(self.audio_tracks[index])
                    self.observer.notify("DurationObtained", duration)
                    return ''
                else:
                    return "Track with the specified name not found"
            else:
                return "Track with the specified name or index not found"
        except Exception as e:
            return f"Error getting audio track duration: {str(e)}"

    def move_audio_track(self, track_name_or_index, time_shift):
        try:
            if track_name_or_index.isdigit():
                index = int(track_name_or_index)
                if 0 <= index < len(self.audio_tracks):
                    track = self.audio_tracks[index]
                else:
                    return "Invalid audio track index"
            elif track_name_or_index in self.track_names:
                index = self.track_names[track_name_or_index]
                if 0 <= index < len(self.audio_tracks):
                    track = self.audio_tracks[index]
                else:
```

```
                    return "Invalid audio track index"
            else:
                return "Track with the specified name or index not found"

            time_shift = int(time_shift)
            self.audio_tracks[index] = track.fade_in(time_shift)
            self.observer.notify("move_audio_track")
            return ''
        except Exception as e:
            return f"Error moving track: {str(e)}"

    def copy_segment(self):
        if self.audio_data:
            self.clipboard = self.audio_data
            self.observer.notify("SegmentCopied")
            return ''
        else:
            return "No audio selected for copying"

    def paste_segment(self):
        if self.clipboard:
            self.audio_data += self.clipboard
            self.observer.notify("SegmentPasted")
            return ''
        else:
            return "No data in clipboard to paste"

    def cut_segment(self):
        if self.audio_data:
            self.clipboard = self.audio_data
            self.audio_data = None
            self.observer.notify("SegmentCut")
            return ''
        else:
            return "No audio selected for cutting"

    def select_segment(self, start_time, end_time):
        if self.audio_data:
            try:
                start_time = int(start_time)
                end_time = int(end_time)
                self.audio_data = self.audio_data[start_time:end_time]
                self.observer.notify("SegmentSelected")
                return ''
```

```
            except Exception as e:
                return f"Error selecting audio segment: {str(e)}"
        else:
            return "No audio selected for segment selection"

    def add_audio_track(self, file_path, format, track_name):
        try:
            audio_track = self.adapter.load_audio(file_path, format)
            self.audio_tracks.append(audio_track)
            self.track_names[track_name] = len(self.audio_tracks) - 1
            self.observer.notify("AudioTrackAdded")
            return ''
        except Exception as e:
            return f"Error adding audio track: {str(e)}"

    def deform_audio(self, start_time, end_time, effect_type):
        if self.audio_data:
            try:
                start_time = int(start_time)
                end_time = int(end_time)
                if effect_type == "reverse":
                    self.audio_data = self.audio_data[start_time:end_time].reverse()
                elif effect_type == "speedup":
                    self.audio_data = self.audio_data[start_time:end_time].speedup(playback_speed=1.5)
                self.observer.notify("AudioDeformed")
                return ''
            except Exception as e:
                return f"Error deforming audio: {str(e)}"
        else:
            return "No audio selected for deformation"

    def pitch_shift(self, amount):
        if self.current_track:
            try:
                amount = float(amount)
                self.current_track = self.current_track.speedup(playback_speed=amount)
                self.observer.notify("PitchShifted")
                return ''
            except Exception as e:
                return f"Error applying pitch shift effect: {str(e)}"
        else:
            return "No selected audio track to apply the effect"

    def encode_audio(self, output_format):
```

```
        if self.audio_data:
            try:
                if output_format in ["ogg", "flac", "mp3"]:
                    output_path = "output." + output_format
                    self.adapter.save_audio(self.audio_data, output_path, output_format)
                    self.observer.notify("AudioEncoded")
                    return ''
                else:
                    return "Invalid format for encoding"
            except Exception as e:
                return f"Error encoding audio: {str(e)}"
        else:
            return "No data to encode"

    def save_current_track(self, output_path, output_format):
        try:
            if self.current_track:
                self.adapter.save_audio(self.current_track, output_path, output_format)
                self.observer.notify("AudioSaved")
                return ''
            else:
                return "No data to save. Select an audio track and apply pitch_shift effect first."
        except Exception as e:
            return f"Error saving audio: {str(e)}"

    def select_audio_track(self, track_name_or_index):
        try:
            if track_name_or_index.isdigit():
                index = int(track_name_or_index)
                if 0 <= index < len(self.audio_tracks):
                    self.current_track = self.audio_tracks[index]
                    self.observer.notify("AudioTrackSelected")
                    return ''
                else:
                    return "Invalid audio track index"
            elif track_name_or_index in self.track_names:
                index = self.track_names[track_name_or_index]
                if 0 <= index < len(self.audio_tracks):
                    self.current_track = self.audio_tracks[index]
                    self.observer.notify("AudioTrackSelected")
                    return ''
                else:
                    return "Track with the specified name not found"
            else:
```

```
                return "Track with the specified name or index not found"
        except Exception as e:
            return f"Error selecting audio track: {str(e)}"

    def remove_audio_track_by_name(self, track_name):
        try:
            if track_name in self.track_names:
                track_index = self.track_names[track_name]
                del self.track_names[track_name]
                if 0 <= track_index < len(self.audio_tracks):
                    del self.audio_tracks[track_index]
                    self.observer.notify("AudioTrackRemoved")
                    return ''
                else:
                    return "Invalid audio track index"
            else:
                return "Audio track with the specified name not found"
        except Exception as e:
            return f"Error removing audio track: {str(e)}"

    def open_audio(self, file_path, format):
        try:
            self.audio_data = self.adapter.load_audio(file_path, format)
            self.observer.notify("AudioOpened")
            return ''
        except Exception as e:
            return f"Error loading audio: {str(e)}"
    def copy_audio(self):
        if self.audio_data:
            self.clipboard = self.audio_data
            self.observer.notify("AudioCopied")
            return ''
        else:
            return "No audio selected for copying"
    def paste_audio(self):
        if self.clipboard:
            self.audio_data = self.clipboard
            self.observer.notify("AudioPasted")
            return ''
        else:
            return "No data in clipboard to paste"

    def cut_audio(self):
        if self.audio_data:
```

```
            self.audio_data = None
            self.observer.notify("AudioCut")
            return ''
        else:
            return "No audio selected for cutting"

    def handle_command(self, command):
        parts = command.split(" ")
        if parts[0] in self.commands:
            if len(parts) > 1:
                return self.commands[parts[0]](*parts[1:])
            else:
                return self.commands[parts[0]]()
        else:
            return "Invalid command"
```

Рис. 4 - Вміст audio_editor_server.py

У файлі main.py реалізовано запуск та роботу сервера.

```python
import socket
import threading

from cur.mediator.mediator import Mediator
from cur.observer.observer import AudioEditorObserver
from cur.server.audio_editor_server import AudioEditorServer


def handle_client(client_socket, mediator):
    while True:
        command = client_socket.recv(1024).decode()
        if not command:
            break
        response = mediator.handle_command(command)
        client_socket.sendall(response.encode())

    client_socket.close()
    print("Client disconnected.")

def main():
    host = "localhost"
    port = 5513
    observer = AudioEditorObserver()
    server = AudioEditorServer(observer)
    mediator = Mediator(server, observer)

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        server_socket.bind((host, port))
        server_socket.listen()

        print(f"Waiting for connection on {host}:{port}")
        client_sockets = []
        while True:
            client_socket, _ = server_socket.accept()
            client_sockets.append(client_socket)
            observer.subscribe(client_socket)
            print(f'subscriber: {observer.subscribers}')

            client_thread = threading.Thread(target=handle_client, args=(client_socket, mediator))
            client_thread.start()

if __name__ == '__main__':
    main()
```

Рис. 5 - Вміст main.py

Рис. 6 - Структура клієнта

У файлі client.py реалізовано підключення клієнта до сервера та введення ним команд.

```python
import socket

def main():
    host = "localhost"
    port = 5513

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
        client_socket.connect((host, port))

        while True:
            command = input("Enter a command: ")
            client_socket.sendall(command.encode())
            response = client_socket.recv(1024).decode()
            print(response)

if __name__ == '__main__':
    main()
```

Рис. 7 - Вміст client.py

Перелік команд та короткий опис того, що вони роблять:

завантаження аудіофайлу: `open test.mp3 mp3`

копіювання поточного аудіо: `copy`

вставка поточного аудіо: `paste`

видалити з буфера поточне аудіо: `cut`

кодування в інший формат (автозбереження): `encode ogg` (на вибір із трьох основних)

додати аудіодоріжку: `add_track test.mp3 mp3 newtrack`

деформація поточного аудіо за вказаним проміжком (два типи ефекту reverse і speedup): `deform 0 100 reverse`

змішування аудіодоріжок: `mix newtrack2 track1 track2`

поділ аудіодоріжки на дві доріжки за початковим і кінцевим часом: `split newtrack2 0 3500`

перенесення на вказаний час в аудіодоріжці: `move newtrack2 2000`

вибір сегмента (шматок) аудіо: `select_segment 100 200`

копіювання сегмента: `copy-segment`

вставка поточного сегмента: `paste-segment`

видалення поточного сегмента: `cut-segment`

ефект зміни висоти тону до поточного аудіо: `pitch_shift 1.5`

вибір аудіодоріжки: `sat trackname`

видалення аудіодоріжки: `rat trackname`

список аудіодоріжок: `lat`

збереження поточної аудіодоріжки в зазначеному форматі: `sct namefile.wav sct namefile.wav wav`

дізнатися тривалість аудіодоріжки: `duration track1`

Висновки: Під час виконання даної лабораторної роботи ми вивчили інформацію про різні види взаємодії додатків: client-server, peer-to-peer,

service-oriented architecture та реалізували архітектуру клієнт-сервер для нашого додатку.