НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Розподілений сервіс обміну повідомленнями та потокового зв'язку»

студент (-ка) IV курсу, групи КП-52
Комар Григорій Миколайович _____

Керівник:
Доцент кафедри ПЗКС к.т.н., доцент,
Вунтесмері Ю.В. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
Доцент кафедри математичних
методів системного аналізу ННК» ІПСА» , к.т.н,
Дідковська М.В. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 6.050103 «Програмна інженерія» (Програмне забезпечення комп'ютерних та інформаційно-пошукових систем)

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ

на дипломний проект студенту

Комар Григорію Миколайовичу

1. Тема проекту «Розподілений сервіс обміну повідомленнями та потокового звязку», керівник проекту Вунтесмері Юрій Володимирович, к.т.н., доцент, затверджені наказом по університету від «22» травня 2019 р. №1331-С.

2. Термін подання студентом проекту «16» червня 2019 р.

3. Вихідні дані до проекту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз організації потоків данних у паралельних моделях;
   - розроблення методу побудови оптимальних за єфективністю обміну програм;
   - розроблення програмного забеспечення серверу потокового звязку;
   - розроблення та опис програмного забеспечення кліента потокового звязку по обміну повідомленнями.

5. Перелік обов'язкового графічного матеріалу:
   - структурна організація апаратних ресурсів паралельного моделюючого середовища (плакат)

- передача віддаленого методу посилання на об’єкт (плакат);
- спосіб побудови сервера (плакат);
- інтерфейс компоненти розподіленої системи (плакат)

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	14.11.2018	
2.	Розроблення та узгодження технічного завдання	28.11.2018	
3.	Розроблення моделей потоків даних	15.12.2018	
4.	Підготовка матеріалів першого розділу дипломного проекту	30.12.2018	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2019	
6.	Підготовка матеріалів другого розділу дипломного проекту	20.02.2019	
7.	Програмна реалізація потокового звязку	10.03.2019	
8.	Тестування програмного забеспечення	17.03.2019	
9.	Підготовка матеріалів третього розділу дипломного проекту	30.03.2019	
10.	Підготовка матеріалів четвертого розділу дипломного проекту	11.04.2019	
11.	Підготовка графічної частини дипломного проекту	21.04.2019	
12.	Оформлення документації дипломного проекту	26.05.2019	

Студент Г.М. Комар

Керівник проекту Ю.В.Вунтесмері

АНОТАЦІЯ

Дана робота присвячена розробленню розподіленого сервісу обміну повідомленнями та потокового звязку.

У роботі виконано порівняльний аналіз існуючих рішень для розподіленого сервісу обміну повідомленнями та потокового звязку, обґрунтовано вибір технологій та допоміжних бібліотек серверної та клієнтської частин для реалізації даного додатку. Розроблений сервіс надає працівникам підприємства можливість обмінюватися повідомленнями за допомогою розподіленого сервісу.

У даному дипломному проекті розроблено: архітектуру серверної та клієнтської частини сервісу, алгоритм автоматичного та ручного збору інформації, алгоритм обробки та підбору правил аналізу інформації, алгоритм аналізу інформації, а також графічні елементи та дизайн додатку.

ABSTRACT

This work is devoted to the development of a distributed messaging service and streaming communications.

In this work a comparative analysis of existing solutions for distributed messaging and streaming services is made, the choice of technologies and auxiliary libraries of server and client parts for the implementation of this application is substantiated. The developed service provides employees with the opportunity to exchange messages through a distributed service.

This graduation project has developed: the architecture of the server and client part of the service, the algorithm of automatic and manual information gathering, algorithm for processing and selection of rules for analyzing information, algorithm for analyzing information, as well as graphic elements and application design.

ДП.045440-01-90 Розподілений сервіс обміну повідомленнями та потокового зв’язку
Відомість проекту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045440-02-91	Розподілений сервіс обміну повідомленнями та потокового зв’язку	5	
	Технічне завдання		
ДП.045440-03-81	Розподілений сервіс обміну повідомленнями та потокового зв’язку	72	
	Пояснювальна записка		
ДП.045440-04-51	Розподілений сервіс обміну повідомленнями та потокового зв’язку	4	
	Програма та методика		
	тестування		
ДП.045440-05-34	Розподілений сервіс обміну повідомленнями та потокового зв’язку	9	
	Керівництво користувача		
ДП.045440-06-99	Розподілений сервіс обміну повідомленнями та потокового зв’язку	1	
	Структура модулів		
	серверної частини		
ДП.045440-07-99	Розподілений сервіс обміну повідомленнями та потокового зв’язку	1	
	Структура модулів		
	клієнтської частини		
ДП.045440-08-98	Розподілений сервіс обміну повідомленнями та потокового зв’язку	1	
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

Розподілений сервіс обміну повідомленнями та потокового зв'язку

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Ю.В.Вунтесмері

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____Г.М.Комар

2018

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: **Розподілений сервіс обміну повідомленнями та потокового зв'язку**

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання в якості інформаційне забезпечення для побудови розподіленого сервісу обміну повідомленнями з високою надійністю.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Розподілений сервіс обміну повідомленнями та потокового зв'язку повинен забезпечувати такі основні функції:

1) забезпечення системи обміну повідомленнями , що характеризується високою автономністю клієнтів;
2) можливість відкритого шифрування повідомлень без участі сервера;
3) полегшеним масштабуванням серверів;
4) забезпечення взаємодії клієнтської та серверної частин

Розробку сервера виконати на платформі Java з використанням технології Scala, а клієнтської частини на платформі Rust.

Додаткові вимоги:

1) наявність динамічного меню;
2) наявність анімованих кнопок;
3) дизайн форм з використанням в якості базових білого та синьо-зеленого кольорів.

5.ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;

2) програма та методика тестування;

3) керівництво користувача;

4) креслення:

- «Авторизація користувача. Схема алгоритму»;
- «Синхронізація даних. Схема роботи програмної системи».

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи……..14.11.2018

Розроблення та узгодження технічного завдання……....28.11.2018

Розроблення структури сервера та клієнта..…….....15.12.2018

Розроблення дизайну сторінок та графічних елементів……..03.02.2019

Програмна реалізація web-ресурсу……....17.03.2019

Тестування web-ресурсу…….....03.04.2019

Підготовка матеріалів текстової частини проекту……..….28.04.2019

Підготовка матеріалів графічної частини проекту……..…12.05.2019

Оформлення технічної документації проекту……..…25.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

Розподілений сервіс обміну повідомленнями та потокового зв'язку
Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____Ю.В.Вунтесмері

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Г.М.Комар

2019

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ, ОДИНИЦЬ І ТЕРМІНІВ

RSS – Rich Site Summary (Короткий Зміст Сайту)
СМОП – Система Миттєвого Обміну Повідомленнями
ЧБ – Чат Бот
AIM – AOL Instant Messenger (Месенджер Розроблений Компанією AOL)
API – Application Programming Interface
ППІ – Прикладний Програмний Інтерфейс
B2B – Business-To-Business (Бізнес Для Бізнесу)
B2C – Business-To-Consumer (Бізнес Для Користувача)
ШНМ – Штучна Нейронна Мережа
ReLU – Rectified Linear Unit (Тип Функції Активації)
SGD – Stochastic Gradient Descent (Метод Стохастичного Градієнта)
NER – Named-Entity Recognition (Розпізнавання Названого Суб'єкта)
NLP – Natural-Language Processing (Обробка Природних Мов)
CLR – Common Language Runtime (Загальне Середовище Виконання)
IDE – Integrated Development Environment (Інтегроване Середовище Розробки)
CI – Continuous Integration (Неперервна Інтеграція)
CD – Continuous Delivery (Неперервна Доставка)
QE – Quality Engineer (Інженер з Якості)
QA – Quality Assurance (Забезпечення Якості)
DevOps – Development and Operations (Розробка та Операції)
BA – Business Analytic (Бізнес Аналітик)
PO – Product Owner (Власник Продукту)
GPL – General Public License (Загальна Публічна Ліцензія)

ВСТУП

Актуальність теми. Моделювання складних динамічних систем (СДС) із зосередженими (ДСЗП) і розподіленими (ДСРП) параметрами є одним з основних методів дослідження процесів, перевірки вірності проектних рішень, синтезу оптимальних систем автоматизації в багатьох предметних областях. Модельний супровід СДС-проектів є важливим фактором конкурентоспроможності об'єктів техніки й сучасних технологій. Формальний опис СДС, що є основою для їхнього моделювання, включає засоби подання топології й системи диференціальних рівнянь великої розмірності. У зв'язку із цим при побудові моделей СДС необхідно залучати високопродуктивні обчислювальні ресурси для рішення систем рівнянь і забезпечувати комп'ютерну підтримку завдань подолання складності об'єктів, що моделюються. Вимогам СДС, як об'єктів моделювання, у найбільшій мірі відповідають засоби й методи, які використовують паралельні обчислювальні системи (ПОС). В теперішній час в розпорядження користувачів надані MIMD-ПОС із досить великою кількістю процесорів, із гнучкими засобами зв'язків між вузлами, з розвиненим програмним забезпеченням реалізації паралельних алгоритмів. Досвід розробки й реалізації паралельних моделей складних динамічних систем показує, що ефективне застосування ПОС вимагає рішення завдань системної організації роботи засобів моделювання. Роботи в цьому напрямку ведуться інститутами кібернетики й проблем моделювання в енергетиці НАН України, закордонними університетами й фірмами. Новою формою системної організації засобів моделювання є паралельне моделююче середовище (ПМС). Проблеми його комплексної розробки й реалізації відбиті в науковому співробітництві зі Штутгартським університетом (Німеччина), у розділі української державної програми, присвяченому створенню сучасних методів і засобів моделювання складних систем. Аналіз показує, що ефективність функціонування ПОС і дружність до користувачів і розроблювачів паралельних моделей СДС істотно залежать від алгоритмічної й

системно-програмної організації процесів моделювання паралельного моделюючого середовища, зокрема від організації обміну інформацією між компонентами ПМС.

Об’єкт досліджень – паралельне моделююче середовище як засіб моделювання складних динамічних систем.

Предмет досліджень – процеси обміну інформацією між компонентами ПМС як фактор ефективного рішення задач моделювання.

Мета дипломної роботи полягає в теоретичному обґрунтуванні й системній організації оптимальних процесів обміну інформацією в паралельному моделюючому середовищі, що дозволить підвищити ефективність всіх етапів розробки й використання паралельних моделей складних динамічних систем, розширити сферу застосування паралельних обчислювальних ресурсів.

Для досягнення цієї мети вирішуються наступні задачі досліджень і розробок:

1. Визначення й класифікація потоків інформації в ПМС, розробка схеми функціонування середовища, орієнтованої на оптимізацію обмінів даними між її компонентами

2. Дослідження впливу топологій об'єктів моделювання на організацію потоків даних у паралельних моделях, уніфікація топологічних описів складних динамічних систем

3. Розробка метода побудови оптимальних за ефективністю обміну програм, що реалізують паралельні моделі динамічних систем і алгоритмів відображення моделей на доступні обчислювальні ресурси ПМС

4. Розробка тестів для визначення характеристик функцій бібліотеки MPI, що реалізують обмін інформацією, методик тестування й визначення критеріїв оптимальності програм ПМС

5. Побудова дослідного зразка ПМС, імплементація та експериментальні дослідження розроблених тестів і методів.

Методи дослідження носять теоретичний і експериментальний характер. При проведенні досліджень і розробок використовувалися методи математичного моделювання, прикладної й обчислювальної математики, теорії диференціальних рівнянь, апарат лінійної алгебри, зокрема теорії матриць, теорія графів. Отримані результати перевірялися шляхом проведення обчислювальних експериментів.

1. АНАЛІЗ РОБІТ З ОРГАНІЗАЦІЇ ПРОЦЕСІВ МОДЕЛЮВАННЯ В ПАРАЛЕЛЬНИХ МОДЕЛЮЮЧИХ СЕРЕДОВИЩАХ

1.1. Аналіз поняття розподілених систем

Поняття «розподілена система» в літературних джерелах визначають по-різному. Історично у процесі розвитку апаратної та програмної складових розподілених систем формувалися різні розуміння їх як систем, що відображалося на їх визначеннях. Розуміння того, як має визначатися розподілена система, постійно уточнюється.

1. Розподіленою системою називають низку з'єднаних центральних процесорів (CPU), що працюють разом.

2. Розподіленою системою називають низку машин з нерозділеною пам'яттю. Надалі розподілені системи потрібно розглядати в найбільш загальній формі. До них належать усі різновиди клієнт-серверних систем, а також мультипроцесорні системи, тобто такі, які складаються з вузлів та можуть бути однопроцесорними або мультипроцесорними.

3. Розподіленою називають систему з просторово розподіленими компонентами, які не використовують ніякої спільної пам'яті й не підлягають децентралізованому адмініструванню. Для реалізації спільних цілей можлива кооперація компонентів. Якщо цими компонентами пропонуються послуги або використовуються запропоновані послуги, то виникає клієнт-серверна система, а у разі додаткового центрального службового посередництва – «торговельна» система (Trading-система).

Часто, визначаючи розподілену систему, на перше місце ставлять поділ її функцій між декількома комп'ютерами. За такого підходу розподіленою є будь-яка обчислювальна система, де обробку даних розділено між двома й більше комп'ютерами.

Розподілена система – це набір незалежних комп'ютерів, які користувач сприймає як єдину об'єднану систему.

У цьому визначенні є два однаково важливі моменти: стосовно апаратури – всі машини автономні; стосовно програмного забезпечення – користувачам надається у користування єдина система.

Основними завданнями розподіленої системи є організація ефективного доступу користувачів до інформаційних і програмних ресурсів та ефективна взаємодія як користувачів з ресурсами, так і різних видів ресурсів між собою.

Паралельне моделююче середовище (ПМС) - це нова системна організація паралельних засобів моделювання, що являє собою сукупність апаратних, програмних і інформаційних засобів, що забезпечують підтримку всіх етапів розробки, налагодження й дослідження паралельних моделей ДС реальної складності, формування й візуалізації результатів моделювання і призначена для рішення широкого класу задач моделювання ДС із зосередженими й розподіленими параметрами.

Основою ефективного функціонування ПМС є оптимальна організація обміну інформацією всіх видів між всіма її компонентами й усередині них.

У рамках ПМС повинні бути вирішені наступні завдання організації обміну даними: складання ефективних паралельних програм, що реалізують моделі СДС; оптимальне відображення паралельних програм на цільові паралельні архітектури; вивід формул, що характеризують вплив процесів обміну інформацією на продуктивність паралельних ресурсів ПМС.

Із усього різноманіття програмних засобів створення паралельних моделей СДС у ПМС тільки для бібліотеки MPI (Message Passing Interface) існує реалізація, що підтримує одночасну роботу з декількома паралельними системами в термінах бібліотеки.

Характерні риси властиві розробці моделей за допомогою бібліотеки MPI:

- недружність до користувача - розроблювач моделей СДС самостійно проходить всі етапи побудови моделей, залишаючись на рівні програмування мовою С, С++;

- рівноправність процесорів обчислювальної системи з погляду організації обмінів інформацією й завантаження процесорів (використовуються однакові процесори й мережею зв'язку між ними є повний граф);
- відсутність можливості оптимізації обмінів інформацією з боку користувача вбудованими інструментами або методами.

Як показав аналіз робіт зі створення сучасних і перспективних засобів моделювання СДС, паралельні моделюючі середовища є новими й складними об'єктами досліджень. У проведених дотепер роботах з комплексної розробки паралельного моделюючого середовища як нової форми алгоритмічної й системно-програмної організації засобів моделювання відсутня постановка й рішення задач організації оптимального обміну інформацією в контексті побудови й реалізації паралельних моделей СДС. З наведеного вище аналізу й були визначені актуальні задачі, вирішенню яких присвячена дисертаційна робота.

1.2. Структурна організація обмінів інформацією в паралельному моделюючому середовищі

Розроблювачі й користувачі паралельних моделей ДСЗП і ДСРП одержують доступ до апаратних ресурсів ПМС, структура якого наведена на рис.1.1.

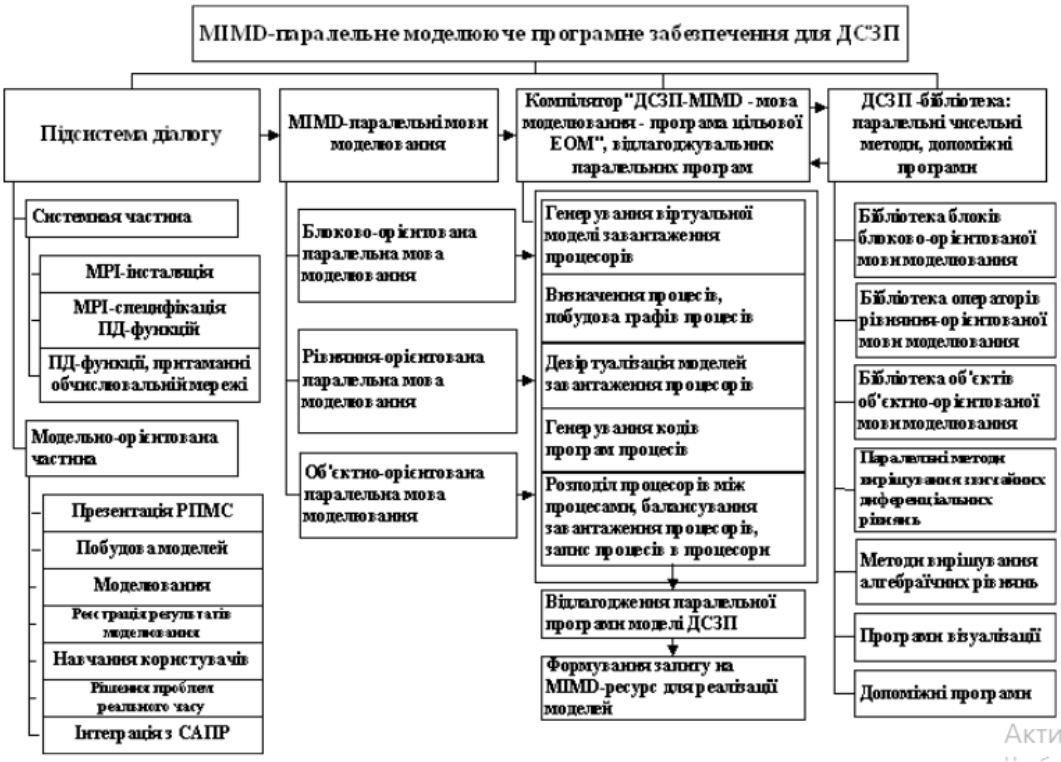


Рис. 1.1. Структурна організація апаратних ресурсів ПМС

З погляду розроблювача паралельних моделей СДС і користувача середовище виглядає так, як показано на рис. 1.2.

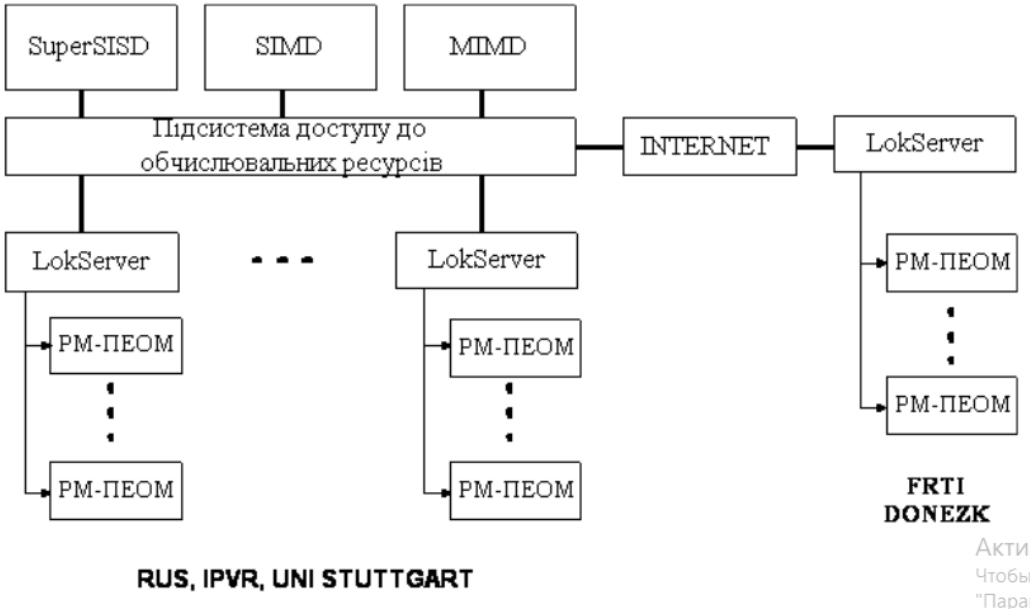


Рис. 1.2. Структура ПМС із погляду розроблювача паралельних моделей СДС і користувача

Тут:

- РМ - робоче місце окремого користувача;
- РМ 1,..., РМ m - інші користувачі;
- ОР(MIMD 1),..., ОР(MIMD n) - обчислювальні ресурси (ОР) MIMD-типу, кожний з яких надає користувачеві свою реалізацію комунікаційної бібліотеки MPI;
- X1, Xn - кількість процесорів у відповідному MIMD-ресурсі;
- КС - комунікаційне середовище.

Будемо характеризувати запропоновану структуру ПМС наступними параметрами:

m - кількість РМ (користувачів і розроблювачів ПМС);

n - кількість доступних MIMD - ресурсів;

X_i , $i \in \{1..n\}$ - загальна кількість процесорів в і - м обчислювальному ресурсі (ВР);

Y_i , $i \in \{1..n\}$ - кількість доступних процесорів на даний момент часу в і-м MIMD-ресурсі;

T_{CS}^i – час затримки доступу до і-го обчислювального ресурсу з боку РМ;

$T_i = \{T\frac{1}{i}, ..., T\frac{k}{i}\}$ – множина характеристик швидкості обмінів комунікаційної бібліотеки (MPI) і-го обчислювального ресурсу, де k – кількість видів обмінів;

$CPUi^p$ – пікова продуктивність одного процесора і-го обчислювального ресурсу (при гетерогенності процесорів у ресурсі береться середня пікова продуктивність);

V_{DATA} – обсяг переданих даних (програма й вхідні дані);

CPU_i^{USER} – кількість процесорів, необхідна програмі користувача від і-го ОР.

Всі види обміну інформацією, що мають місце в ПМС, розділимо на наступні групи:

F1 – РМ $\Leftrightarrow$ ОР (копіювання програм моделювання й даних);

F2 – РМ $\Leftrightarrow$ {$ОР_1$, …, $ОР_n$} розподіл задачи моделювання між декількома ресурсами ПМС одночасно (копіювання моделей і даних);

F3 – обмін в середині ресурсу (паралельної системи MIMD-типу), тобто, безпосередньо, бібліотеки MPI;

F4 – ОР $\Leftrightarrow$ ОР обмін між ресурсами (розподіл задачи моделювання між декількома ресурсами ПМС одночасно, бібліотека РАСХ MPI);

F5 – РМ $\Leftrightarrow$ {$ОР_1$, …, $ОР_n$} збір результатів моделювання;

F6 – одержання інформації про умови моделювання;

F7 – обмін між рівнями програмного забезпечення.

Кожна із груп обмінів впливає на час розробки моделей СДС.

У структурі моделюючого програмного забезпечення ПМС організація обміну інформацією займає проміжне положення між системою візуального проектування моделей (СВПМ) і готовою моделюючою MPI-програмою.

Виходячи із цього визначимо, які кроки необхідно виконати для оптимізації обмінів:

$Р_1$ – тестування характеристик $Т_i$ = { $Т\frac{1}{i}$,.., $Т\frac{k}{i}$ } бібліотек обміну кожного з обчислювальних ресурсів середовища;

$Р_2$ - одержання топологічного опису моделей від СВПМ, його формалізація;

$Р_3$ – оптимізація відображення топології моделі на топологію ресурсів середовища, визначення оптимальної кількості процесорів, необхідних для моделювання.

$Р_4$ – генерування масштабованої MPI-програми;

$Р_5$ – організація методу масштабування з обліком доступних у сучасний момент ресурсів середовища;

$Р_1$ – $Р_5$ – функції організації обмінів.

Користувач ПМС створює паралельні моделі СДС, які являють собою набір взаємозалежних паралельно працюючих процесів, що реалізують які-небудь формули, розрахунки та інш. Звичайна кількість таких процесів досить

велика й перевищує загальну кількість доступних процесорів, проте, у цілому, для такої моделі СДС може бути написана MPI-програма, кожний процес якої відповідає процесу моделі. Така програма могла б бути запущена, якби була доступно необмежена кількість процесорів в одній MIMD-EOM. Назвемо такого виду програму віртуальною MPI-програмою, MIMD-EOM із необмеженою кількістю процесорів – віртуальнюї MIMD-EOM, а такого виду MPI-процеси – віртуальними процесами. Необхідно згрупувати й розподілити віртуальні MPI-процеси на доступній кількості процесорів у доступному ОР ПМС і одержати MPI-программу.

Називатимемо такий процес групування й розподілу віртуальних процесів девіртуалізацією. Отриману програму можна запускати на кількості процесорів ОР від 1 до X_i, при цьому бажано знайти ту кількість процесорів N_{CPU}^{OPT}, на якому буде досягатися мінімум часу моделювання ($1 \le N_{CPU}^{OPT} \le X_i$), тобто оптимізувати MPI-програму та, відповідно, обмін інформацією.

Розроблено єдиний підхід до подання складних динамічних систем.

Елементи технологічних схем (ТС), що моделюються, представлено у вигляді зв'язаних один з одним певним чином блоків з деякою кількістю входів α і виходів β. ТС складається з кінцевої кількості такого виду блоків. Кількість типів блоків також обмежена (звичайно порядку $50 \div 100$). Позначимо кількість типів блоків через E, а кількість блоків у ТС через H. Виходи деяких блоків можуть бути пов'язані з їхніми входами. Кожний із блоків ТС може бути розбитий на деяку кількість елементарних блоків, які позначимо через $E_{\sum}$, а їхню загальну кількість – через $H_{\sum}$.

Для єдиного представлення топологій ТС і МДО необхідно показати можливість однозначного перетворення графа ТС-типу в граф МДО-типу й навпаки.

Перетворення МДО-графа в ТС-граф: нехай G(U, V) – граф МДО, де U – множина вершин графа, V – множина ребер графа. Граф G є замкнутим і орієнтованим. Утворимо новий граф G_1 з множиною вершин U_1, кількість

елементів якого дорівнює числу ребер графа G, кожній вершині G_1 відповідає одне ребро G. Кожній вершині графа G_1 відповідають дві вершини графа G – початкова й кінцева. Аналізуючи матрицю інцидентності графа G, можна з'єднати вершини графа G_1 у такий спосіб: якщо в G ребро і входить у початкову вершину якого-небудь ребра j, то з'єднуємо відповідному цьому ребру і вершину ii в G_1 з вершиною jj, що відповідає ребру j у напрямку від ii до jj ; якщо в G ребро і виходить із кінцевої вершини будь-якого ребра j, то з'єднуємо відповідну цьому ребру і вершину ii в G_1 з вершиною jj, що відповідає ребру j у напрямку від ii до jj. Очевидно, що отриманий граф також буде замкнутим.

Таким чином, показана можливість перетворення МДО-графа в ТС - граф.

Перетворення ТС-графа в МДО-граф: нехай G(U, V) – граф ТС, де U – множина вершин графа, V – множина ребер графа. Граф G є орієнтованим. Запропоновано етапи перетворення:

- якщо G незамкнутий, введемо фіктивну вершину O_G, з якої з'єднаємо всі вільні входи й виходи – одержимо граф G_1;
- у графі G_1 розфарбуємо всі ребра в один колір (наприклад, синій);
- в зв’язку з тим, що граф ТС уже перетворений відповідно до рис. 6, в ньому відсутні петлі, і в кожній вершині є тільки вхідні й вихідні ребра; замінимо всі вершини графа G_1 на орієнтовані ребра – кожній вершині відповідає одне ребро, у початкову вершину якого входять відповідні вхідні ребра вершини, а з кінцевої виходять вихідні ребра заміненої вершини;
- розфарбуємо нові ребра в інший колір (наприклад, червоний);
- сполучимо початкові вершини кожного синього ребра з кінцевими вершинами того ж ребра - сині ребра перетворяться в петлі;
- Видалимо петлі з графа - одержимо новий граф МДО - типу.

Таким чином, показана можливість перетворення Тс-графа в МДО-граф.

Отже, три типи топологій СДС - ТС, ССА й МДО можуть бути після деяких перетворень зведені до представлення у вигляді графа одного із двох типів ТС- і МДО-графа. До всіх трьох типів топологій СДС можуть бути застосовані єдині методи аналізу й перетворень на основі розгляду їхніх графів.

Елементи, що моделюються, являють собою віртуальні MPI-процеси, тобто у віртуальній моделі й реалізується функція P_2.

2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Оптимізація обмінів інформацією в паралельному моделюючому середовищі

У базовій моделі клієнт–сервер усі процеси в розподілених системах поділяють на дві групи. Процеси, які прямо реалізують деяку службу, наприклад службу файлової системи або бази даних, є серверами (servers), а процеси, які вимагають служби у серверів через надсилання запиту і подальшого очікування відповіді від сервера, – клієнтами (clients). Взаємодію між клієнтом та сервером називають режимом запит-відповідь (request-reply behavior).

Модель клієнт-сервер була приводом для багатьох дебатів і суперечок, у яких одне з основних питань полягало в тому, як розподілити частини програмного забезпечення між клієнтом і сервером. Зрозуміло, що зазвичай чіткого розуміння немає. Наприклад, сервер розподіленої бази даних може постійно бути клієнтом, який передає запити на різноманітні файлові сервери, відповідальні за реалізацію таблиць цієї бази даних. У такому разі сервер баз даних сам не робить нічого, крім обробки запитів.

Проте, розглядаючи велику кількість прикладних програм типу клієнт-сервер, передбачених для організації доступу користувачів до баз даних, часто рекомендують поділяти їх на три рівні:

– рівень інтерфейсу користувача;

– рівень обробки;

– рівень даних.

Рівень користувацького інтерфейсу містить усе необхідне для безпосереднього спілкування з користувачем, наприклад для керування дисплеєм. До рівня обробки належить прикладне програмне забезпечення, а до рівня даних – дані, з якими доводиться працювати.

Задача оптимізації моделюючих MPI-програм формулюється в такий спосіб. На основі отриманої віртуальної моделі необхідно розробити MPI-програму й визначити кількість процесорів обчислювального ресурсу ПМС, на якому час виконання даної програми буде найменшим із всіх можливих. Тому як критерій оптимальності виберемо час роботи програми, що реалізує модель

$$T = F(N_i), \qquad (2)$$

де

N_i – кількість використовуваних процесорів.

Як цільова функція виступає функціонал F, мінімум якого необхідно визначити з обмеженням по кількості доступних процесорів обчислювального ресурсу середовища.

Очевидно, що кількість процесорів (N_{CPU}^{OPT}), при якому досягається min(T), буде оптимальним рішенням. При цьому $N_i \in \overline{1, N_{CPU}}$ - область припустимих значень.

Загальна кількість способів розподілу A_P віртуальних процесів на кількості процесорів від 1 до N_{CPU} дорівнює

$$\sum_{i=1}^{N_{CPU}} i^{A_P - 1} \qquad (3)$$

Пропонується наступний спосіб пошуку min(T).

Будемо вважати, що кожному віртуальному процесу виділений для роботи процесор з характеристиками, що відповідають характеристикам процесора обчислювального ресурсу ПМС, тобто по суті віртуальна модель працює на MIMD-системі з необмеженою кількістю процесорів, характеристики яких відповідають характеристикам ресурсу середовища.

Тоді, знаючи параметри обчислювального ресурсу, результати тестів комунікаційної бібліотеки ресурсу й кількість CPU- і MPI- періодів роботи програми, можна розрахувати час роботи кожної групи віртуальних процесів на такій "віртуальній ЕОМ":

$$T_i = T_i^{CPU} + T_i^{MPI},\ i = 1..k. \qquad (4)$$

Тут:

$T_i^{CPU} = CPU^P * OP_i^{CPU}$, OP^{CPU} – кількість CPU-операцій віртуального процесу;

$T_i^{MPI} = \sum_j^f MPI_j^{OP} * OP^{MPI}$, f – кількість типів використаних MPI-операцій;

MPI_j^{OP} – час виконання MPI-операції j-го типу, визначається системою тестів комунікаційної бібліотеки;

OP^{MPI} – кількість MPI -операцій j-го типу.

Необхідно зробити сортування отриманих значень по зростанню (убуванню) і визначити T_{min} і T_{max} .

Введемо ваги груп:

$$Vi = \frac{T_i}{T_{max}}\ ;\ i = 1..k. \qquad (5)$$

Розподілимо всі доступні процесори обчислювального ресурсу ПМС між групами рівномірно, тобто кожній групі буде відповідати

$N_i = quot(V_i * N_{CPU})$ (6)

процесорів (quot[F] - ціла частина числа F, а rem[F] - дробова частина).

Оптимальна кількість процесорів для кожної групи буде досягатися в межах

$N_i^{OPT} \in (1..N_i)$.

Зауважимо, що оптимальний час роботи моделі буде приблизно дорівнювати часу роботи, що досягається на кількості процесорів

$$N_{OPT} = \sum_{i}^{k} N_i^{OPT} . \quad (7)$$

Введено наступні допущення:

- при суміщенні віртуальних MPI-процесів і використанні SPMD-моделі програм загальна кількість періодів CPU- і MPI- операцій залишається незмінною;
- у суміщеному MPI-процесі навантаження на процесор (кількість CPU-операцій) збільшується прямо пропорційно кількості процесів, що суміщуються;
- у суміщеному MPI-процесі кількість MPI-операцій залишається незмінною, збільшується обсяг даних для обмінів;
- усередині групи процеси рівноправні й розподіляються рівномірно між доступними процесорами.

Необхідно розрахувати $min(T_i)$ для кожної групи

$min(T_i) = min[T_i(1),\ldots,T_i(N_J)]$, (8)

Розглядаючи час роботи групи віртуальних процесів як функцію від кількості використовуваних процесорів

$T_i = F(j, N_j)$, $j = 1..N_j$, (9)

цільову функцію T_i визначимо формулою

$$\mathrm{Ti} = \mathrm{quot}(\frac{\mathrm{n_i}}{\mathrm{Nj}}) * \mathrm{T_i^{CPU}} + \alpha_{1j}(*T_i^{MPI(ptp)}) + \alpha_{2j}(*T_i^{MPI(COLL)}) \quad (10)$$

де:

n_i – кількість віртуальних процесів в і-й групі;

N_j – поточне значення кількості використовуваних процесорів ($1 \le N_j \le N_i$);

T_i^{CPU} – час роботи всіх CPU-періодів одного віртуального процесу;

$T_i^{MPI(ptp)}$ - час роботи всіх MPI-періодів одиночних обмінів одного віртуального процесу;

$T_i^{MPI(coll)}$ - час роботи всіх MPI-періодів колективних обмінів і синхронізації одного віртуального процесу;

$\alpha_{1j}; \alpha_{2j}$ - коефіцієнти зміни часу виконання MPI-операцій залежно від кількості використовуваних процесорів; визначаються дослідним шляхом на підставі результатів тестування бібліотеки обмінів (розділ 4).

Показнику $\min(T_i)$ відповідає оптимальна для групи кількість процесорів Ni_{OPT}.

Оптимальне значення загальної кількості використовуваних процесорів буде дорівнювати:

$$N_{OPT} = N1_{OPT} + ... + NK_{OPT} \quad (11)$$

Необхідно рівномірно розподілити віртуальні процеси і-ї групи між доступними цій групі процесорами, кількість яких N_i^{AV} визначається за формулою

$$N_i^{AV} = \text{quot}[N_{iOPT} * \frac{N^{AV}}{N_{OPT}}], \quad (12)$$

де N^{AV} – загальна кількість доступних процесорів.

Такий розподіл означає масштабоване разпаралелювання.

Знайдемо показники $\text{quot}[\frac{n_i}{N_i^{AV}}]$ й $rem[\frac{n_i}{N_i^{AV}}]$. Тоді

$$n_i = \text{quot}[\frac{n_i}{N_i^{AV}}] * N_i^{AV} + rem[\frac{n_i}{N_i^{AV}}] \quad (13)$$

Таким чином, на N_i^{AV} процесорах, що виділені для n_i віртуальних процесів і-й групі, процеси розподілені по

$(\text{quot}[\frac{n_i}{N_i^{AV}}])$ на $(N_i^{AV} - r)$ процесорах і $(\text{quot}[\frac{n_i}{N_i^{AV}}]+1)$ – на r процесорах

При виборі обчислювального ресурсу (ОР) будемо виходити з міркувань мінімізації часу одержання результатів моделювання t_{REZ}. Той ОР, у якого цей час приблизно є мінімальним, і буде обраний для паралельного моделювання (задача користувача). Розглянемо два підходи до оцінки складових цього часу:

1) при використанні оптимальної кількості процесорів

$$t_{REZ}^{OPTIM} = 2 * T_{KS}^i + (V_{DATA\ IN} + V_{DATA\ OUT}) * V_i + t_{model}(N_{OPTIM}^{CPU}) + t_{wait}, \quad (14)$$

де $t_{wait}=0$, при $N_{OPTIM}^{CPU} \le C_a^i$ та $t_{wait} \ne 0$, при $N_{OPTIM}^{CPU} > З_a^i$

2) при використанні максимально доступного числа процесорів

$$t_{REZ}^{AV} = 2 * T_{KS}^i + (V_{DATA\ IN} + V_{DATA\ OUT}) * V_i + t_{model}(N_{CPU}^{AV}). \quad (15)$$

Тут

$2*T_{KS}^{i}$ – час затримки передачі вихідних даних на ресурс і одержання результатів моделювання;

$(V_{DATA\ IN} + V_{DATA\ OUT})*V_i$ – сумарний час передачі вихідних даних на ресурс і одержання результатів моделювання;

$t_{model}(N_{OPTIM}^{CPU})$ – апріорна оцінка часу моделювання на оптимальній кількості процесорів;

t_{wait} – час затримки постановки завдання на моделювання, що залежить від обчислювального ресурсу. Можливе визначення величини цієї складової за допомогою статистичної обробки результатів роботи ОР.

$t_{model}(N_{CPU}^{AV})$ – апріорна оцінка часу моделювання на доступній кількості процесорів.

Визначивши t_{REZ}^{OPTIM} і t_{REZ}^{AV} для кожного з ресурсів, знайдемо

$$t_{model} = t_{REZ} = \min(t_{REZ}^{OPTIM_i}, t_{REZ}^{AV_i}), \qquad (16)$$

де $i = 1..B_a$.

Таким чином, за отриманими формулами можна вибирати обчислювальні ресурси ПМС для моделювання СДС різних предметних областей.

2.2. Обмін повідомленнями

Розрізнюють два методи передачі повідомлень від однієї віддаленої системи до другої: безпосередній обмін повідомленнями й використання черг повідомлень. У разі безпосереднього обміну передача відбувається прямо, і можлива лише за умови, що сторона, яка приймає, готова одержати повідомлення в цей самий момент, інакше використовується посередник – менеджер черг повідомлень, тобто компонента надсилає повідомлення в одну із черг менеджера, після чого вона може продовжити свою роботу. Надалі сторона, яка отримує повідомлення, вилучить його із черги менеджера й розпочне обробку.

Найпростішим видом реалізації безпосереднього обміну повідомленнями є використання транспортного рівня мережі через інтерфейс сокетів, минаючи будь-яке проміжне програмне забезпечення. Однак такий спосіб взаємодії зазвичай не застосовують у розподілених системах, оскільки в цьому разі реалізують усі функції проміжного середовища розробники прикладних програм. За такого підходу складно отримати розширювану й надійну розподілену систему, тому для розробки прикладних розподілених систем здебільшого використовують системи черг повідомлень.

Наявні кілька розробок у сфері проміжного програмного забезпечення, що реалізують високорівневі сервіси для обміну повідомленнями між програмними компонентами, зокрема Microsoft Message Queuing, IBM MQSeries і Sun Java System Message Queue. Ці системи дають можливість прикладним програмам використовувати такі базові примітиви обробки черг:

– додати повідомлення в чергу;

– взяти перше повідомлення із черги, процес блокується до появи в черзі хоча б одного повідомлення;

– перевірити чергу на наявність повідомлень;

– установити обробник, який викликається з появою повідомлень у черзі.

Менеджер черги повідомлень у таких системах може перебувати на комп’ютері, розміщеному окремо від комп’ютерів з компонентами, що беруть участь в обміні. У цьому разі повідомлення спочатку поміщується у вихідну чергу на комп’ютері з компонентою, що надсилає повідомлення, а потім пересилається менеджерові необхідної компоненти. Для створення великих систем обміну повідомленнями може використовуватися маршрутизація повідомлень, за якої повідомлення не передаються прямо менеджерові, що підтримує чергу, а проходять через низку проміжних менеджерів черг повідомлень (рис. 1.3).

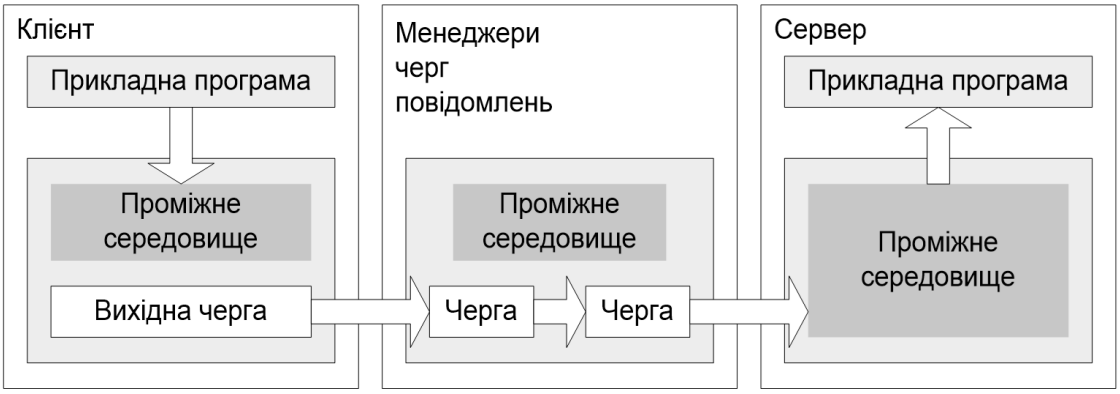


Рис. 2.1. Системи черг повідомлень

Використання черг повідомлень орієнтовано на асинхронний обмін даними. Основні переваги таких систем:

– час функціонування сервера може не залежати від часу роботи клієнтів;

– незалежність проміжного середовища від засобу розробки компонент і мови програмування;

– зчитувати й обробляти заявки із черги можуть кілька незалежних компонент, що дає можливість досить просто створювати стійкі й масштабовані системи.

Недоліки систем черг повідомлень такі:

– необхідність явного використання черг розподіленою прикладною програмою;

– складність реалізації синхронного обміну;

– певні витрати на використання менеджерів черг;

– складність отримання відповіді, оскільки передача відповіді може вимагати окремої черги на кожний компонент, що надсилає заявки.

Щоб віддалений виклик відбувався прозоро з погляду прикладної програми, яка виконує виклик, проміжне середовище має надати процедуру-заглушку (stub), що буде викликатися клієнтською прикладною програмою. Після виклику процедури-заглушки проміжне середовище надає переданим їй аргументам виду, придатного для передачі за транспортним протоколом, і спрямовує їх на віддалений комп’ютер з викликуваною функцією. На віддаленому комп’ютері параметри вилучаються проміжним середовищем з повідомлення транспортного рівня й передаються викликуваній функції (рис. 1.4). Аналогічно на клієнтську машину надсилається результат виконання функції, викликаної з сервера.

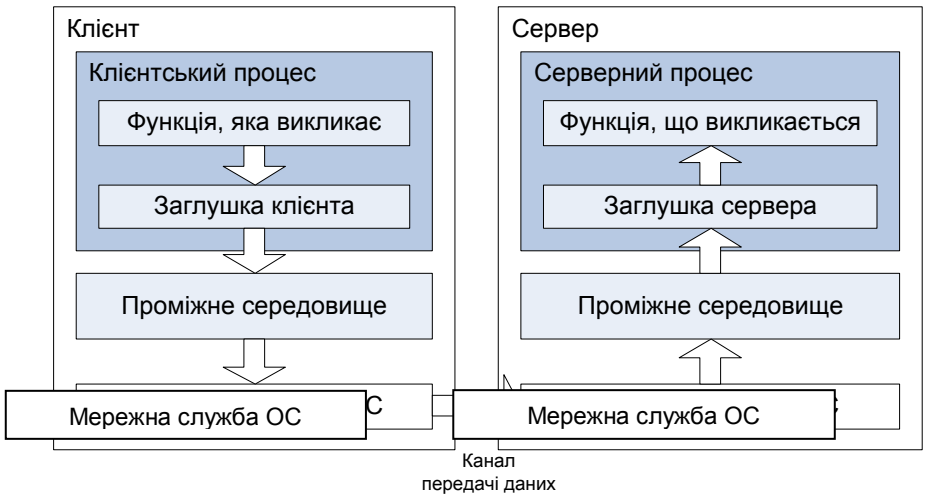


Рис. 2.2. Віддалений виклик процедур

Розрізняють три можливі варіанти віддаленого виклику процедур:

Синхронний виклик – клієнт очікує завершення процедури сервером і, у разі потреби, отримує від нього результат виконання віддаленої функції.

Однонапрямлений асинхронний виклик – клієнт продовжує виконувати своє завдання, не отримуючи відповіді від сервера, відповідь або відсутня, або її реалізація якось інакше передбачена під час розробки (наприклад, через функцію клієнта, яку віддалено викликає сервер).

Асинхронний виклик – клієнт продовжує виконувати завдання, після завершення сервером процедури він отримує повідомлення й результат її виконання, наприклад через callback-функцію, яка викликається проміжним середовищем під час одержання результату від сервера.

Процес перетворення параметрів для їх передачі між процесами (або доменами прикладної програми у разі використання .NET) під час віддаленого виклику називають маршалізацією (marshalling). Перетворення екземпляра якого-небудь типу даних у придатний для передачі за межі викликаючого процесу набір байтів називають серіалізацією.

Десеріалізація – процедура, обернена серіалізації, – полягає у створенні копії серіалізованого об’єкта на основі отриманого набору байтів. Такий підхід до передачі об’єкта між процесами за допомогою створення його копій називають маршалізацією за значенням (marshal by value), на відміну від маршалізації за посиланням.

Маршалізація за посиланням під час передачі параметрів за посиланням використовує серіалізацію не самих вказівників, а об’єктів, на які вказують вказівники.

Процес серіалізації повинен бути визначений для всіх типів даних, переданих у процесі віддаленого виклику, зокрема для параметрів функції, яка викликається, й результату, який повертає функція. У разі передачі параметрів за посиланням серіалізації підлягають об’єкти, на які посилаються, до самих вказівників серіалізація не може бути застосована, оскільки це ускладнює

використання механізму віддаленого виклику в мовах, які підтримують вказівники на об’єкти невідомого типу.

Використання віддалених об’єктів

У зв’язку з переходом розробників прикладних програм від структурної парадигми до об’єктної з’явилася необхідність у використанні віддалених об’єктів (remote method invocation, RMI). Віддалений об’єкт становить деякі дані, сукупність яких визначає його стан, який можна змінювати за рахунок виклику його методів. Зазвичай можливий прямий доступ до даних віддаленого об’єкта, за якого виконується неявний віддалений виклик, необхідний для передачі значення поля даних об’єкта між процесами. Методи й поля об’єкта, які можна використовувати через віддалені виклики, доступні через деякий зовнішній інтерфейс класу об’єкта. Зовнішній інтерфейс компоненти розподіленої системи в таких системах зазвичай збігається із зовнішнім інтерфейсом одного з класів, на якому побудована компонента.

У момент, коли клієнт починає використовувати віддалений об’єкт, з боку клієнта створюється клієнтська заглушка, яку називають посередником (proxy), який реалізує той самий інтерфейс, що й віддалений об’єкт. Процес, який виконує виклик, використовує методи посередника, який маршалізує їх параметри для передачі мережею, і передає їх мережею серверу. Проміжне середовище, яке десеріалізує параметри й передає їх заглушці з боку сервера, називають каркасом (skeleton) або, як і у разі віддаленого виклику процедур, заглушкою (рис. 1.5). Каркас зв’язується з деяким екземпляром віддаленого об’єкта, як заново створеним, так і наявним екземпляром об’єкта, залежно від застосовуваної моделі використання віддалених об’єктів.

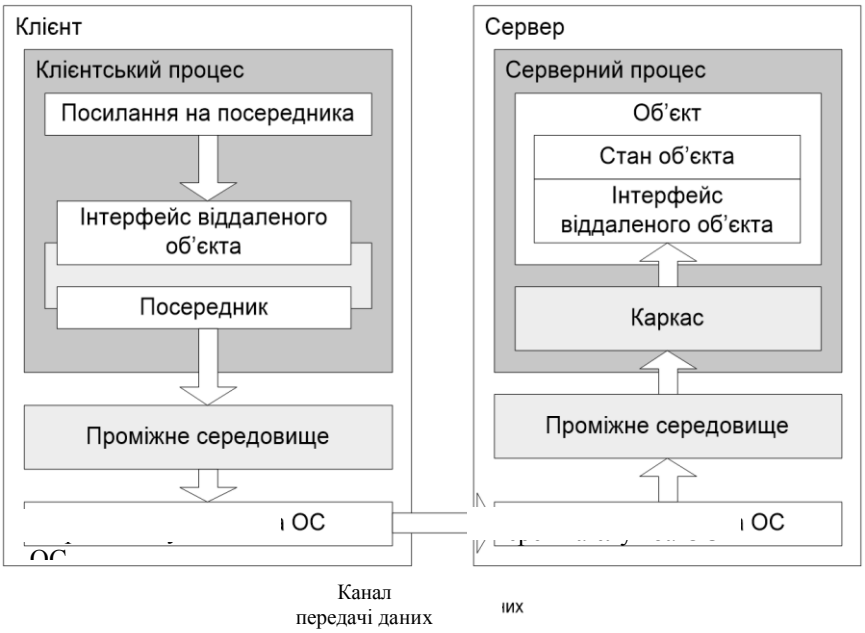


Рис. 2.3. Використання віддалених об’єктів

Весь описаний процес називають маршалізацією віддаленого об’єкта за посиланням (marshal by reference). На відміну від маршалізації за значенням, екземпляр об’єкта перебуває у процесі сервера й не залишає його, а для доступу до об’єкта клієнти використовують посередників. У разі маршалізації за значенням саме значення об’єкта серіалізується в набір байтів для його передачі між процесами, після чого необхідне створення його копії в іншому процесі.

Аналогічно до віддаленого виклику процедур, виклик методу віддаленого об’єкта може бути як синхронним, так і асинхронним. Процесу використання віддалених об’єктів, що не зустрічалися у віддаленому виклику процедур, характерні такі особливості. По-перше, якщо на момент формування концепції віддаленого виклику процедур виключення (exceptions) ще не підтримувалися й не використовувалися найбільш поширеними мовами програмування, то надалі вони стали методом інформування сторони, яка виконує виклик, про проблеми, що виникли у сторони, яка викликається. Таким чином, у системах, що використовують віддалені об’єкти, серіалізації підлягають як параметри методу і його результат, так і виключення, які з’являються у процесі виконання

віддаленого методу. По-друге, як параметр або результат методів можуть теж передаватися посилання на віддалений об’єкт (рис. 2.4), якщо віддалений метод – клієнт, який виконує виклик, –також є сервером RMI.

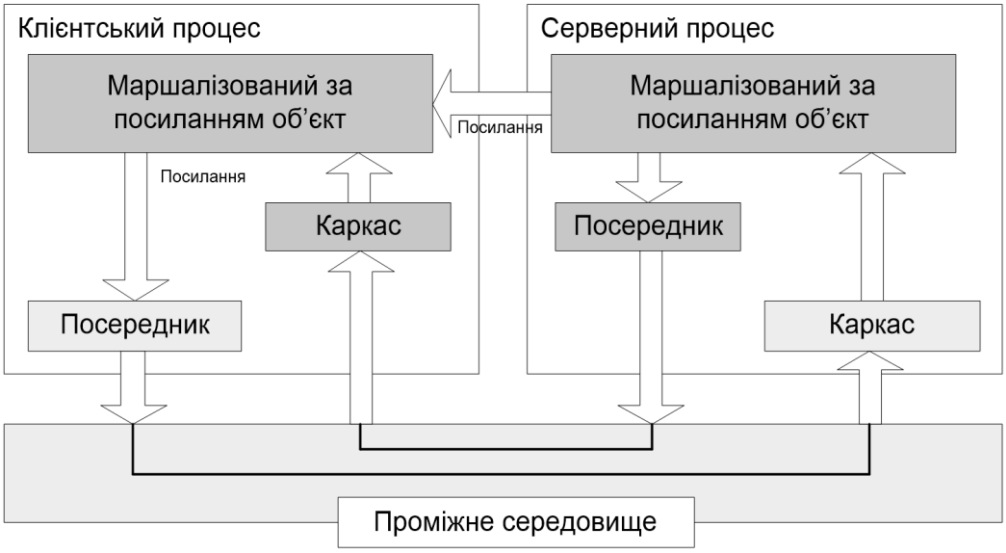


Рис. 2.4. Передача віддаленому методу посилання на об’єкт, що маршалізується за посиланням

У разі використання віддалених об’єктів важливими є питання про час їх функціонування: у який момент часу утворюється екземпляр віддаленого об’єкта; протягом якого проміжку часу він існує.

Щоб описати життєвий цикл у системах із віддаленими об’єктами, використовують такі поняття прикладного програмування:

– активацію об’єкта – процес переведення створеного об’єкта у стан обслуговування віддаленого виклику, тобто зв’язування його з каркасом і посередником;

– деактивацію об’єкта – процес переведення об’єкта у стан невикористання.

Розрізняють три моделі використання віддалених об’єктів: модель єдиного виклику (singlecall), модель єдиного екземпляра (singleton), а також модель активації об’єктів за запитом клієнта (client activation). Перші дві моделі

також іноді називають моделями серверної активації (server activation), хоча активація завжди відбувається на сервері після будь-якого запиту від клієнта.

Модель єдиного виклику. В разі використання цієї моделі об’єкт активується на час єдиного віддаленого виклику. В найпростішому випадку для кожного виклику віддаленого методу об’єкта клієнтом на сервері створюється й активується новий екземпляр об’єкта, що деактивується й потім знищується відразу після завершення віддаленого виклику методу об’єкта. Таким чином, віддалені виклики різних клієнтів ізольовано один від одного. За рахунок видалення об’єктів після виклику досягається раціональна витрата ресурсів пам’яті, але можуть витрачатися значні ресурси процесора на постійне створення об’єктів. Програма-посередник на клієнті й заглушка на сервері існують до знищення посередника об’єкта (рис. 1.7).

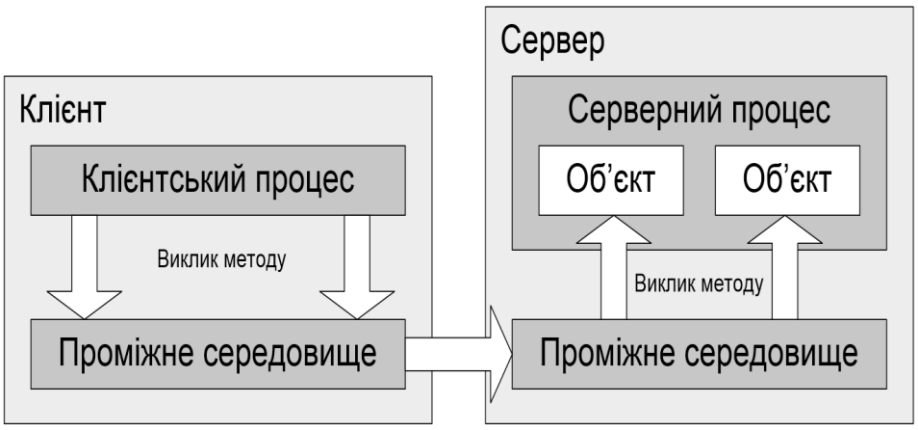


Рис. 2.5. Режим єдиного виклику віддаленого методу

Такий метод застосування віддалених об’єктів можна розглядати як деякий варіант віддаленого виклику процедур, оскільки об’єкт не зберігає свого стану між викликами. Проте сервер використовує свої ресурси для підтримки каркасу й каналу між посередником і заглушкою.

Недоліком методу одного виклику є часте створення й видалення екземплярів об’єктів, тому в проміжному середовищі може міститися сервіс, що дозволяє підтримувати деяку кількість уже створених, але ще не активованих об’єктів, які використовуються для обробки віддалених викликів. Такий набір

об’єктів, що очікують своєї активації, називають пулом об’єктів (object pooling). Після завершення віддаленого виклику об’єкти деактивуються й можуть бути поміщені в пул і використані повторно надалі або видаляються, якщо розмір пулу досягнув деякого максимального значення. Така технологія дозволяє отримати баланс між швидкістю обробки запиту й обсягом використовуваних ресурсів сервера. Як бачимо з опису, в системі з пулом об’єктів активація не завжди потрібна безпосередньо після створення об’єкта, а видалення не завжди виконується відразу після деактивації.

Відмітною рисою методу одного виклику є мінімальні витрати на організацію системи балансування навантаження та її найбільша ефективність, оскільки кожний сервер, який обслуговує запити, може обробити виклик будь-якого віддаленого методу.

Модель єдиного екземпляра. У разі використання моделі єдиного екземпляра віддалений об’єкт існує не більш ніж в одному екземплярі. Створений об’єкт існує, поки є хоч один клієнт, який його використовує (рис. 2.6).

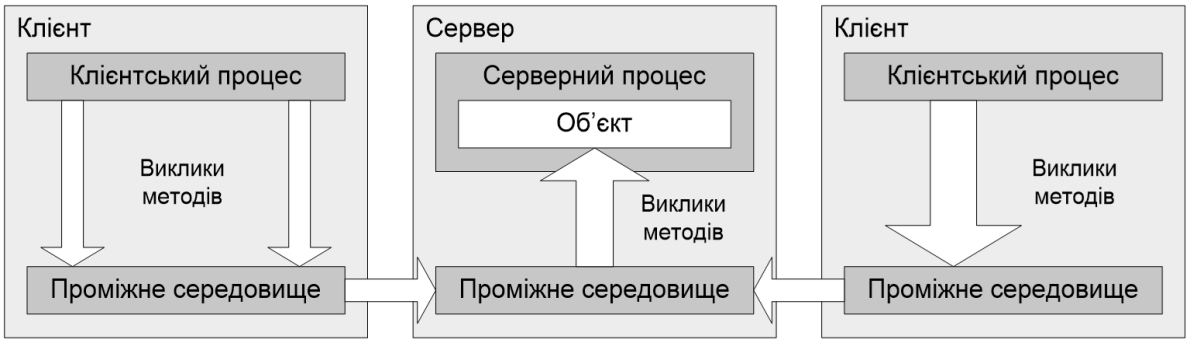


Рис. 2.6. Використання віддалених об’єктів у режимі єдиного екземпляра

У разі використання моделі єдиного об’єкта виклики різних клієнтів працюють одним екземпляром віддаленого об’єкта. Оскільки виклики клієнтів не ізольовані один від одного, то використовуваний об’єкт не повинен мати будь-якого внутрішнього стану. Модель єдиного об’єкта дозволяє отримати найбільш високу продуктивність, оскільки об’єкти не створюються й не активуються сервером під час кожного виклику методу об’єкта.

Активація за запитом клієнта. Під час кожного створення клієнтом посилання на віддалений об’єкт (точніше, на посередника) на сервері виникає новий об’єкт, що існує, поки клієнт не видалить посилання на посередника. За такого методу виклики різних клієнтів ізольовано один від одного і кожний об’єкт зберігає свій стан між викликами, що призводить до найменш раціонального використання ресурсів пам’яті сервера (рис.2.7).

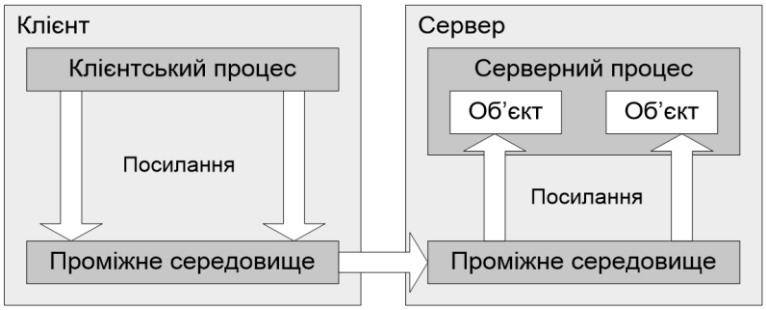


Рис. 2.7. Об’єкти, що активуються клієнтом

3.Проектування системи обміну повідомленнями в розподілених мережах

3.1. Вибір фреймворку та бібліотек для розробки додатку

3.1.1.WSDL: опис інтерфейсу програмної компоненти

Для опису інтерфейсу програмної компоненти, включаючи специфікацію коректних повідомлень, було розроблено мову WSDL. Опис мовою WSDL містить сім складових (рис. 3.1).

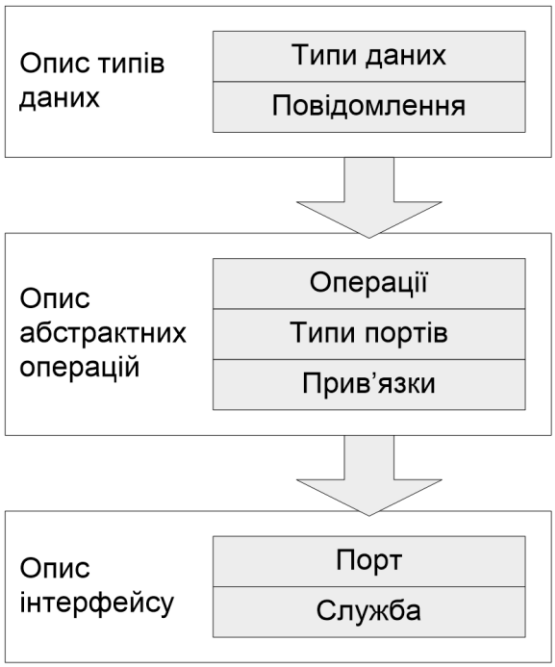


Рис. 3.1. Складові WSDL-документа

3.1.2/Серіалізація об'єктів .NET Framework

На відміну від прикладних програм на некерованому коді, прикладна програма .NET Framework не обов'язково виконується у вигляді окремих процесів, а може перебувати в межах одного процесу операційної системи у власних областях – доменах прикладної програми, які можна розглядати як деякі логічні процеси віртуальної машини Common Language Runtime (CLR). Використання керованого коду за таких умов дозволяє гарантувати ізоляцію прикладної програми у межах своїх областей. У процесі передачі повідомлення між доменами прикладної програми деякого об'єкта для його класу слід визначати такі параметри: процедуру серіалізації, що дозволяє зберегти стан

об’єкта в деякому зовнішньому сховищі (наприклад, у файлі або в повідомленні транспортного протоколу) за допомогою потоків введення-виведення; процедуру десеріалізації, що створює копію об’єкта за збереженим станом (рис. 3.2).

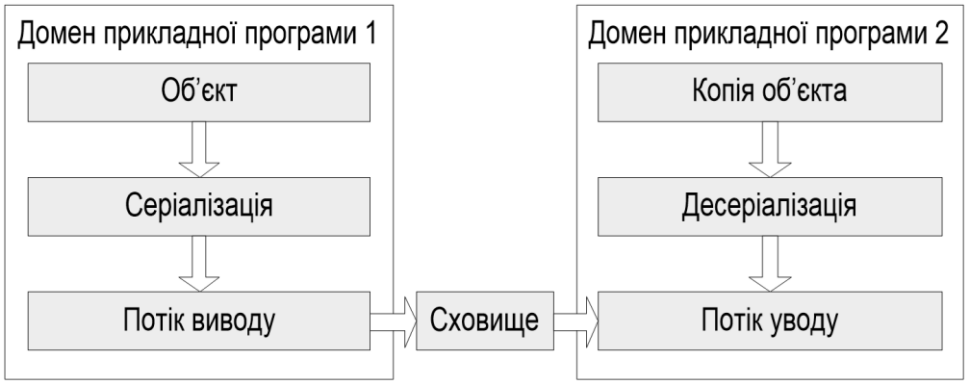


Рис. 3.2. Серіалізація й десеріалізація об’єкта

Слід зазначити, що загалом такі об’єкти можуть бути об’єктами різних класів і навіть створеними в різних системах розробки прикладних програм. Завдання серіалізації об’єкта, який містить лише поля з елементарних типів значень (спадкоємців класу System.ValueType) і рядків, не становить принципових труднощів. Для такого об’єкта у процесі серіалізації в потік записують самі значення всіх полів об’єкта, однак загалом об’єкт містить посилання на інші об’єкти, які, у свою чергу, можуть посилатися один на одного, утворюючи граф об’єктів (object graph). Самі посилання не можуть зберігатися в потоці введення-виведення, тому основне питання серіалізації – у який спосіб заміняти посилання.

Граф об’єктів – орієнтований граф G = < V, E >, у якому вершини – це об’єкти (множина V), а ребра направлені від об’єктів, що містять посилання, до об’єктів, на які посилаються (рис. 2.51). Усі об’єкти, які розглядаються у процесах серіалізації або десеріалізації, описують у термінах об’єктно-орієнтованого підходу. В такому разі об’єкти, властивості яких наслідують інші об’єкти, є батьками, а об’єкти, які наслідують властивості батьків, – дітьми.

Наявність посилання на об’єкт B у об’єкті A є приналежністю пари <A, B> множині E. У графі всі поля об’єктів, які належать до простих типів значень і

рядків, можна вилучити з розгляду через тривіальність їх серіалізації. Хоча формально рядки є посилальними типами, обмеженням реалізації рядків у CLI є те, що немає можливості змінити вже створений рядок, але є можливість тривіально обробляти рядки під час серіалізації, зберігаючи в потоці їх вміст.

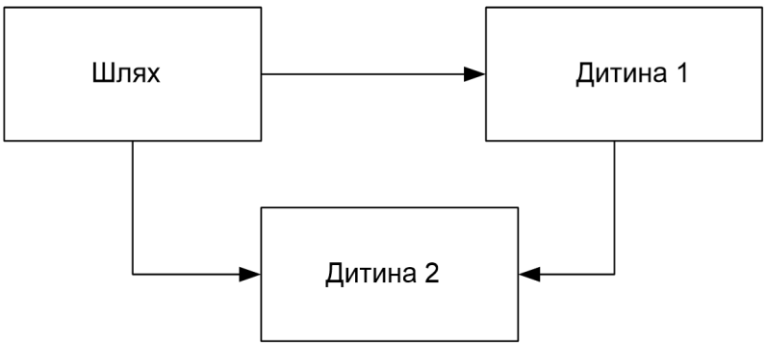


Рис. 3.3. Граф об’єктів

Існує окремий випадок, коли серіалізація виконується тривіально, у такому разі граф повинен мати вигляд орієнтованого дерева. У кожну вершину дерева, відмінну від кореня, направлене єдине ребро, існує єдина вершина, у яку не направлене жодне ребро, – корінь. За таких умов серіалізація може виконуватися від кореня методом у глибину, серіалізація полів-посилань функціонує аналогічно до серіалізації полів-значень. Якщо виявлено посилання на об’єкт, то замість нього у сховище розміщують значення полів об’єкта, на який посилаються, і деяку інформацію прикладної програми щодо типу об’єкта. Наприклад, якщо наявні ребра <A, B_1>, ... <A, B_n>, то функцію серіалізації S для об’єкта A можна подати як S(A) = <V(A), <S(B_1), ... S(B_n)>>, де функція V – значення полів-значень і рядків цього об’єкта.

Проблема серіалізації графу об’єктів полягає в тому, що посилання на той самий об’єкт може бути значенням поля різних об’єктів (наявні ребра <A_1, B> і <A_2, B>). Можлива також наявність циклів у вигляді A → ... → A.

У цьому разі у процесі серіалізації об’єктам мають відповідати деякі ідентифікатори, й у сховищі окремо зберігається список об’єктів, позначений ідентифікаторами, а під час серіалізації замість посилань записують ідентифікатори об’єктів, на які посилаються. Якщо A_1, ... A_n – усі вершини

графу, то після серіалізації утвориться множина {<id_1, $S(A_1)$>, ... <id_{An}, $S(A_n)$>}, де $S(A) = < V(A), <id_1, ... id_{Bn}> >$, а $B_1, ... B_n$ – об'єкти, посилання на які безпосередньо міститься в об'єкті А. У програмі роль ідентифікатора об'єкта виконує його адреса, але замість неї зручніше обрати деякі ідентифікатори у процедурі серіалізації для більш легкого читання людиною отриманого образу. Під час серіалізації потрібно включити список адрес уже записаних об'єктів як для ведення списку ідентифікаторів, так і для виявлення можливих циклів у разі обходу графу методом у глибину.

Слід зазначити, що результат серіалізації дерева легко подати у вигляді XML, коли вміст кожного об'єкта є одним елементом з деяким набором атрибутів і вкладених елементів, тому, розглядаючи проблему серіалізації, можна сформулювати рекомендацію, щоб класи, передані між віддаленими компонентами, були коренем дерева об'єктів. Зокрема це дерево може бути навіть виродженим, тобто клас не містить полів-посилань взагалі.

3.1.3.Технологія Java Server Paqes.

Технологія JSP від компанії Sun Microsystems стала надбудовою над технологією Java Servlets та забезпечує більш швидку і просту розробку web-прикладних програм за рахунок застосування підходу, який використовує шаблони програмування.

Для розуміння архітектури і переваг JSP необхідно знати технологію Java Servlets, оскільки вони тісно пов'язані. Сторінки JSP являють собою шаблони сторінок HTML, аналогічні шаблонам PHP і ASP. Основною відмінністю від інших подібних технологій є те, що код, який міститься всередині спеціальних тегів, не інтерпретується під час звертання до сторінки, а попередньо компілюється в Java Servlet. Статичні ділянки шаблона перетворюються на виклики до функцій для їх розміщення в потік виведення. Код компілюється так, неначе він міститься всередині сервлета. Компіляція JSP

сторінок у сервлети є трудомісткою, але виконується одноразово або під час першого звертання до сторінки, або у процесі запуску сервлет-контейнера.

Технологія JSP вдало поєднує підхід з використанням шаблонів до побудови сайтів і всі переваги Java-платформи, завдяки чому набула значного поширення як для створення професійних комерційних розробок, так і для відкритих безкоштовних проектів. Важливим кроком до розширення підходу з використанням шаблонів стали бібліотеки тегів (tag libraries), які створили можливість інтегрувати стандартні, сторонні або власні програмні компоненти у сторінки. Простота створення та використання зумовила популярність бібліотек тегів.

Завдяки роботі на основі Java технологія JSP не прив'язана до конкретної апаратної або програмної платформи, тому JSP є зручним рішенням для використання в гетерогенних середовищах.

Продуктивність технології обмежена такими об'єктивними особливостями архітектури: по-перше, сторінки мають бути відкомпільованими в сервлети, що потребує багато часу; по-друге, сервлети виконуються в середовищі Java, тобто в режимі інтерпретації.

Однак ці обмеження компенсуються додатковими можливостями, оскільки сучасні контейнери підтримують кластеризацію серверів, то навантаження перекладається на апаратне забезпечення. Це є економічно виправданим і простим рішенням. Завдання компіляції в сервлети є одноразовим та виконується або під час першого звертання, або у процесі запуску сервлет-контейнера, тому не позначається на загальній продуктивності системи у разі розгляду за достатній період.

Основними перевагами JSP є простота програмування, характерна для підходу з використанням шаблонів, наявність великої кількості сторонніх бібліотек, легкість їх застосування, потужні й різноманітні середовища розробки. Завдяки цим факторам JSP є найбільш перспективною базовою технологією розробки у процесі створення Web-сайтів. Однак у разі створення

складних Web-систем обмеження, які накладаються підходом з використанням шаблонів, стають серйозною перешкодою для розвитку цієї технології.

4. ОСОБЛИВОСТІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1. Опис розробленої системи

Один з підходів до організації взаємодії клієнтів і серверів – це розподіл програм, що перебувають на рівні прикладного програмного забезпечення, який полягає у тому, щоб помістити на клієнтську сторону лише термінальну частину інтерфейсу користувача, як показано на рис. 3.1, а, дозволивши прикладній програмі віддалено контролювати подання даних. Альтернативою цьому підходу буде передача клієнтові всієї роботи з інтерфейсом користувача (рис. 3.1, б). В обох випадках відокремлюється від прикладної програми графічний зовнішній інтерфейс, який є пов'язаним з іншою частиною прикладної програми, що перебуває на сервері, за допомогою специфічного для цієї прикладної програми протоколу. В такій моделі зовнішній інтерфейс виконує лише те, що необхідно для надання інтерфейсу прикладній програмі.

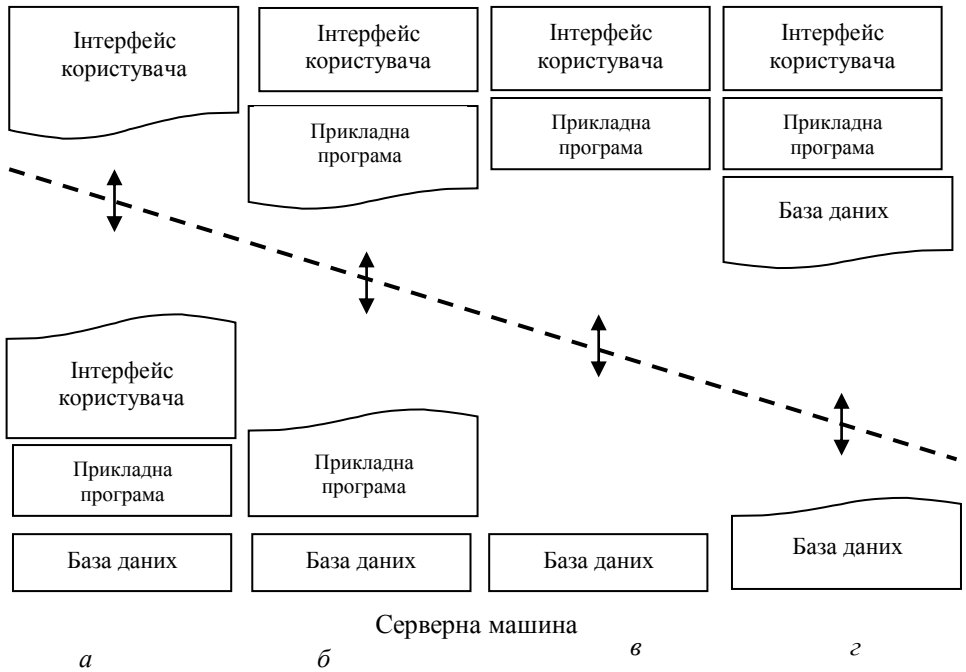


Рис. 4.1. Архітектура клієнт–сервер

У багатьох системах клієнт–сервер поширені типи організації, зображені на рис. 3.1, в і рис. 3.1, г, які застосовують у разі, коли клієнтська машина

(персональний комп’ютер або робоча станція) з’єднана мережею з розподіленою файловою системою або базою даних. Більша частина прикладних програм працює на клієнтській машині, а всі операції з файлами або базою даних передаються на сервер. Рис. 3.1, г зображає ситуацію, коли частина даних утримується на локальному диску клієнта. Наприклад, у процесі роботи в Internet клієнт може поступово створити на локальному диску величезний кеш найвідвідуваніших Web-сторінок.

Розглядаючи розподілення програмного забезпечення на клієнтське і серверне, слід ураховувати те, що у сервера іноді виникає потреба працювати як клієнт. У такій ситуації (рис. 4.2) маємо фізично триланкову архітектуру (physically three-tiered architecture), у котрій програми, що становлять частину рівня обробки, виносяться на окремий сервер, але прикладні програми можуть частково перебувати й на машинах клієнтів і серверів. Типовим прикладом триланкової архітектури є обробка транзакцій, за якою окремий процес, монітор транзакцій, координує роботу всіх транзакцій.

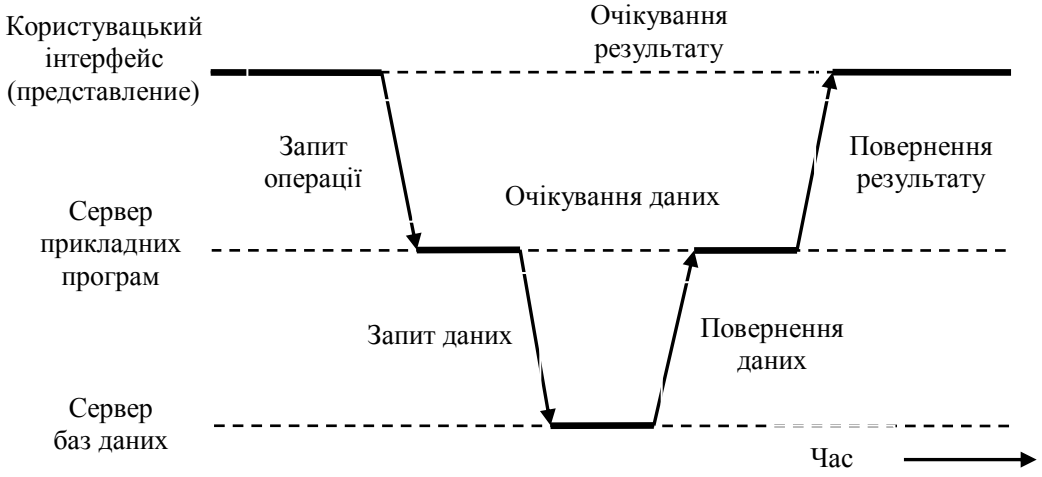


Рис. 4.2. Сервер, що діє як клієнт

Хоча багатопотокові клієнти мають істотні переваги, основні переваги багатопоточності в розподілених системах забезпечуються на стороні сервера. Практика показує, що багатопотоковість не лише істотно спрощує код сервера, але й робить набагато простішою розробку тих програмних серверів, у яких для

досягнення високої продуктивності потрібне паралельне виконання декількох програмних комплексів, до яких належать і *мультипроцесорні системи*. Навіть нині, коли мультипроцесорні комп'ютери активно випускають у вигляді робочих станцій загального призначення, використання для паралельної обробки багатопотоковості не втратило своєї актуальності.

Щоб відчути переваги потоків виконання для написання кодів серверів, розглянемо організацію *файлового сервера*, який періодично блокується, очікуючи доступу до диска. Файловий сервер зазвичай чекає вхідного запиту на операції з файлами, після чого обробляє отриманий запит і повертає відповідь.

Одне з можливих архітектурних рішень показано на рис. 3.3, на якому один з потоків виконання (*диспетчер*) прочитує запити на файлові операції, які надсилають клієнти із зазначенням кінцевої точки цього сервера. Після перевірки запиту сервер обирає (тобто блокує) *робочий потік виконання*, який перебуває у стані очікування, і передає запит йому.

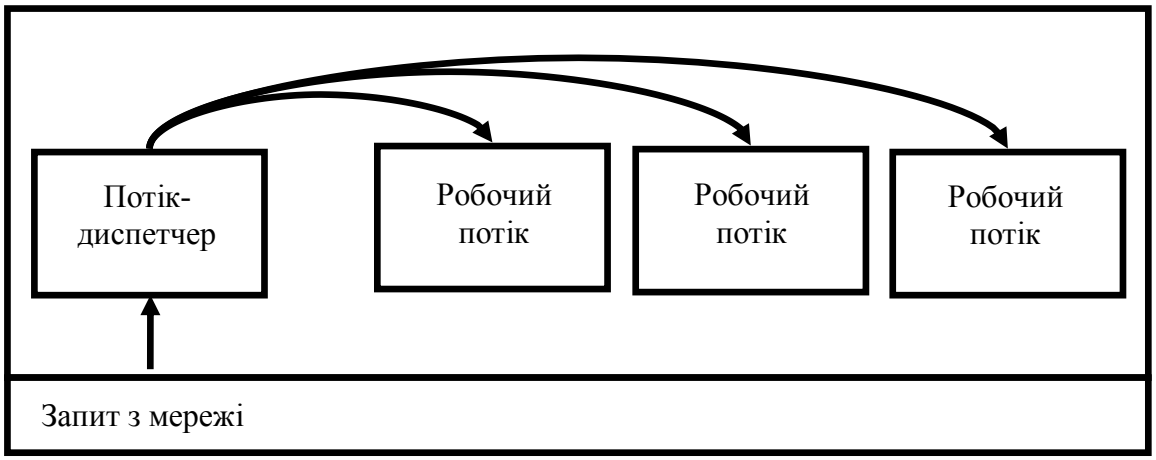


Рис. 4.3. Багатопотоковий сервер, організований
за схемою «диспетчер-робочий потік»

Робочий потік здійснює блокуюче читання з локальної файлової системи, що призупиняє потік виконання на час читання даних з диска. На час призупинення потоку виконання керування може бути передане іншому потоку виконання, наприклад потоку-диспетчеру, або диспетчер може обрати інший готовий до запуску робочий потік.

Структура з диспетчером – не єдиний спосіб організації багатопотокового сервера. У моделі «команда» всі потоки виконання еквівалентні, кожен отримує й обробляє власні запити (рис. 4.4). Іноді завдання надходять,

а потрібний потік зайнятий, що зазвичай відбувається, коли кожен потік спеціалізується на виконанні особливого виду робіт. У цьому разі створюється *черга незавершених робіт*, за якої потоки виконання мають спочатку переглядати чергу робіт, а потім виконувати завдання, яке щойно надійшло.

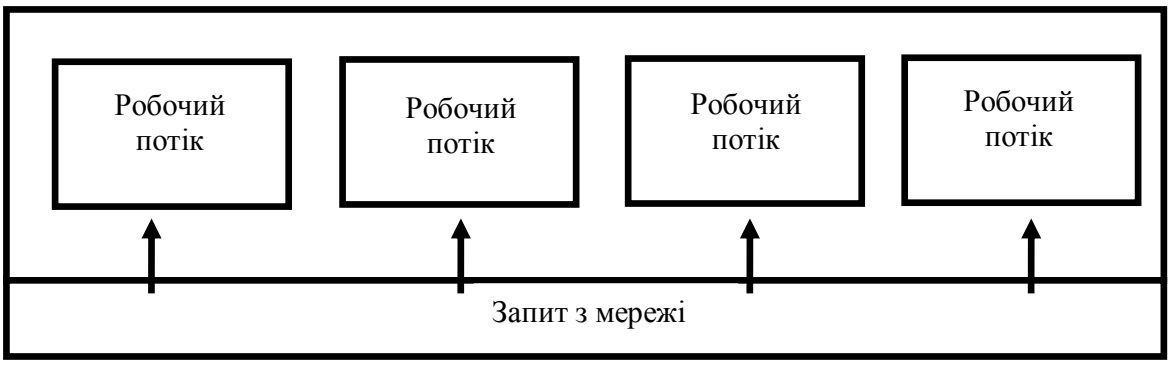


Рис. 4.4. Багатопотоковий сервер, який працює за схемою «команда»

Потоки виконання можуть також мати організацію розкладу обчислень у вигляді *конвеєра* (рис. 4.5), за якої перший потік породжує деякі дані й передає їх для обробки наступному потоку виконання і т. д. Незважаючи на те, що така організація і не підходить для файлового сервера, для інших завдань, зокрема задач «виробник-споживач», це рішення є ефективним.

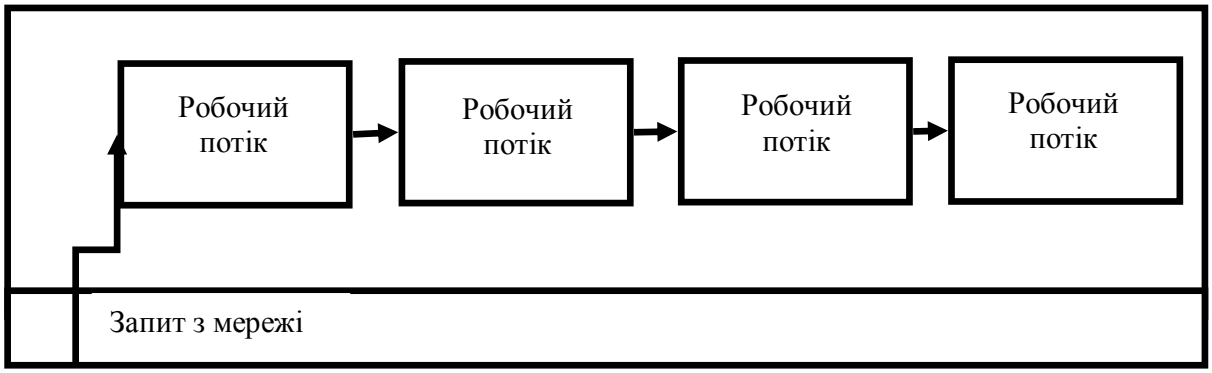


Рис. 4.5. Багатопотоковий сервер, який працює за схемою «конвеєр»

Розглянемо, як файловий сервер міг би виконувати запис, якщо немає потоків виконання. Одна з можливостей – виконувати обчислення так, ніби файловий сервер працює з єдиним потоком виконання. Основний потік файлового сервера отримує запит, перевіряє його і передає на виконання раніше, ніж отримає наступний. Поки сервер чекає закінчення дискових операцій, він не обробляє інших запитів від клієнтів. Окрім того, якщо файловий сервер працює на віддаленій машині, то процесор під час дискових

операцій не зайнятий, у результаті чого обробляється значно менша кількість запитів за секунду. Таким чином, потоки виконання забезпечують суттєве підвищення продуктивності, незважаючи на те, що вони запускаються на виконання по черзі.

Наявні такі можливі організації обчислювального процесу на сервері: *багатопотоковий; однопотоковий сервер;* потоків виконання немає зовсім, але за умови, коли спадання продуктивності через використання однопотокової архітектури робить її застосування можливим. За іншої організації обчислювального процесу сервер використовують як великий кінцевий автомат (рис. 4.6).

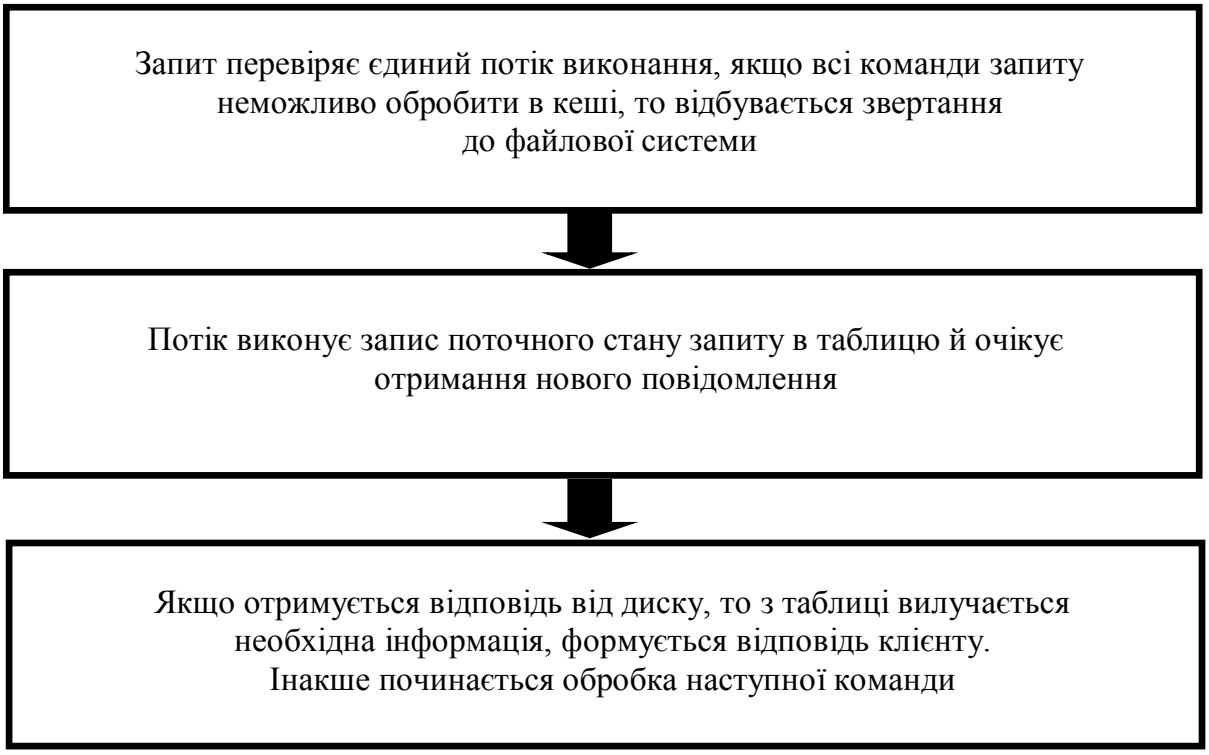


Рис. 4.6. Алгоритм роботи кінцевого автомата

Після надходження запиту його перевіряє єдиний потік виконання, якщо всі команди запиту неможливо обробити в кеші, то відбувається звертання до дискової системи. Проте замість блокування потік виконання записує стан поточного запиту в таблицю і переходить до стану очікування й отримання нового повідомлення. Нове повідомлення може бути як запитом на здійснення нової операції, так і відповіддю на попередній запит від дискової підсистеми. Якщо це новий запит, то починається нова робота, якщо це відповідь від диска, то з таблиці вилучається відповідна інформація, відповідь формується і

передається клієнтові. Згідно з цією схемою сервер повинен мати можливість здійснювати неблокуючі виклики *send* і *receive*.

У такій архітектурі моделі «послідовних процесів» немає. Стан обчислень для кожного повідомлення, яке приймається і відправляється, має повністю зберігатися й отримуватися з таблиці, у результаті чого потоки виконання та їх стеки складно моделювати. Процес, який проходить у кінцевому автоматі, полягає у фіксації подій і, залежно від їх типу, реакції на ці події.

Потоки виконання уможливлюють одночасне збереження ідеї послідовних процесів, які здійснюють блокуючі системні виклики (як і під час викликів RPC для роботи з диском), і паралельної роботи. Блокуючі системні виклики спрощують програмування, а паралельність обробки потоків підвищує продуктивність. Однопотокові сервери зберігають простоту блокуючих системних викликів, але втрачають у продуктивності. Підхід, використовуваний у кінцевих автоматах, дозволяє завдяки паралелізму досягти високої продуктивності, але через наявність неблокуючих викликів важко програмується (табл. 4.7).

Таблиця 4.7. Способи побудови сервера

Модель	**Характеристики**
Потоки виконання	Паралельність обробки потоків, блокуючі системні виклики
Однопотоковий процес	Немає паралелізму, блокуючі системні виклики
Кінцевий автомат	Паралелізм, неблокуючі системні виклики

4.2 Структурна організація web-сервісу

Ключовим сервісом проміжного середовища розподілених систем є сервіс забезпечення обміну даними між компонентами розподіленої системи.

Щоб повністю формально описати взаємодію двох компонент розподіленої системи, необхідні три мови:

– мова повідомлень, яка описує результат серіалізації об’єктів;

– мова опису специфікацій повідомлень, що визначає коректність пові-
домлень для сервісів компоненти;

– мова опису інтерфейсу компоненти, яка призначена для подання набору її сервісів.

Мови опису інтерфейсу і специфікацій повідомлень часто на практиці об’єднуються та подаються однією мовою програмування.

Програмна компонента, що звертається до сервісів, для роботи з ними має повністю узгоджені власні інтерфейси з інтерфейсами сервісів. Незважаючи на суттєві відмінності між моделлю передачі повідомлень і моделлю віддаленого виклику, для них інтерфейс компоненти розподіленої системи можна описати як сукупність адрес і форматів повідомлень її сервісів. Роль сервісу, що надається програмною компонентою, відіграє одне з таких понять:

– методи об’єкта, який активується сервером;

– об’єкт, що активується клієнтом разом зі своїми полями, властивостями й методами;

– черга з повідомленнями – запитами, які зчитуються програмною компонентою.

Адреса сервісу залежить від проміжного середовища і складається з мережної адреси компоненти й певного публічного імені сервісу. Мережна адреса програмної компоненти залежить від імені її комп’ютера для систем віддаленого виклику або адреси менеджера черги для систем обміну повідомленнями. Така адреса є адресою протоколу низького рівня, на якому ґрунтується певне проміжне середовище. Роль такого протоколу можуть виконувати протоколи HTTP, TCP, NetBIOS або інший протокол низького рівня проміжного середовища. Складовою адреси сервісу також є його ідентифікатор, роль якого може відігравати певний ідентифікатор класу, який активується для середовищ віддаленого виклику, або ж ім’я черги повідомлень, з якої сервіс зчитує повідомлення запиту. Хоча ім’я викликаного методу часто

зазначено в самому повідомленні, його варто розглядати як складову частину адреси сервісу, оскільки формати повідомлень неоднакові для різних методів того самого класу.

Якщо компонента системи передачі повідомлень надсилає повідомлення – відповіді клієнтові, то можна вважати, що сервіс такої компоненти має дві адреси – одну для черги запитів і другу для черги відповідей (ім’я черги відповідей може бути задано й у повідомленні запиту).

Крім інформації про повну адресу сервісу, програмі – клієнтові компоненти необхідно знати формат повідомлень, які одержуються і повертаються сервісу. Повідомлення, які одержуються, – повідомлення з параметрами віддаленого виклику і повідомлення – запити в чергах повідомлень, а повідомленнями, які повертаються, є повідомлення з результатом виконання методу і повідомлення-відповіді. До параметрів методу віддаленого виклику варто віднести й деякий ідентифікатор активованого об’єкта сервера у разі активації об’єктів за запитом клієнта. Можна стверджувати, що кожному сервісу компоненти мають відповідати єдина специфікація формату одержаних ним повідомлень і єдина специфікація прийнятих від нього повідомлень (інколи ця специфікація інформує про те, що немає відповіді від компоненти).

Важливою відмінністю систем обміну повідомленнями від систем віддаленого виклику є те, що немає обмежень на формат повідомлення. Таким чином, формально в них є можливість використовувати для опису формат повідомлення, наприклад, контекстно вільних формальних граматик. Однак було б природно вважати, що формат повідомлення має бути еквівалентним опису полів деякого класу CLI (Call Level Interface), об’єкт якого перетвориться в результаті серіалізації в передане повідомлення.

Якщо кожне повідомлення в системах черг повідомлень і параметри методу віддаленого виклику становитимуть єдиний серіалізований об’єкт деякого складного типу даних, то відмінності між системами з активованими

сервером об’єктами й системами передачі повідомлень стають мінімальними. Крім того, єдиний параметр віддаленого виклику є вирішенням проблеми недоступності у повідомленні властивостей активованих сервером об’єктів, тому рекомендують створювати віддалені методи з єдиним параметром складного типу.

Таким чином, кожний сервіс програмної компоненти характеризують за трьома складовими: повною адресою сервісу; єдиною специфікацією одержаних сервісом повідомлень (запитів); єдиною специфікацією прийнятих від сервісу повідомлень (відповідей).

Сукупність специфікацій усіх сервісів програмної компоненти утворює її інтерфейс (рис. 4.8).

Рис. 4.8. Інтерфейс компоненти розподіленої системи

Оскільки повідомлення є результатом серіалізації певного класу, то однією зі специфікацій повідомлення можна вважати сукупність серіалізованих полів і властивостей об’єкта, маршалізованого за значенням. Для систем віддаленого виклику специфікацією інтерфейсу може бути, наприклад, опис класу .NET. Таким чином, метадані зі списків з описом інтерфейсу або класу віддаленого об’єкта і класами параметрів його методів повністю визначають інтерфейс програмної компоненти. Однак такий підхід часто незручний, оскільки, хоч і зумовлює відкритість системи, але й ставить у залежність опис інтерфейсу програмної компоненти від засобу розробки, використовуваного

для її створення, та вимагає надання клієнтові списків із класами компоненти. У зв’язку з цим існує потреба в загальноприйнятих і незалежних від засобів розробки програмних компонент мовах опису їх інтерфейсу.

Миттєві повідомлення або повніше система обміну миттєвими повідомленнями, скорочено ІМ) — телекомунікаційна служба для обміну текстовими повідомленнями між комп'ютерами або іншими пристроями користувачів через комп'ютерні мережі (як правило через інтернет). Зазвичай і від початку, це були невеликі текстові повідомлення. Але з розвитком у систему були додані й інші функції, такі як передавання файлів, зображень, звукових сигналів та повідомлень, відео, а також здійснення спільних дій, таких як малювання або ігри.

Для користування цим видом комунікації необхідна клієнтська програма. Клієнтську програму системи миттєвих повідомлень часто називають інтернет пейджером або месенджером.

Відмінність миттєвих повідомлень від, наприклад, електронної пошти тут в тому, що обмін повідомленнями відбувається в реальному часі. При відправлені повідомлення по електронній пошті, повідомлення зберігається у поштовій скриньці на сервері. Для того, щоб отримати повідомлення, отримувач повинен сам перевірити свою поштову скриньку і забрати їх. У інтернет пейджерах зв'язок між користувачами утримується постійно і відправлене повідомлення одразу передається користувачу.

Обмін повідомленнями може бути або між двома, або між декількома співрозмовниками (конференція, чат).

Система миттєвих повідомлень працює за деяким протоколом. Протоколи бувають серверні або безсерверні. Найпоширенішими є серверні протоколи, коли месенджери не працюють самостійно, а підключаються до центрального комп'ютера мережі обміну повідомленнями, який називають сервером. Тому месенджери й називають клієнтами (клієнтськими програмами).

У безсерверних протоколах (FChat, NASSI, UChat) повідомлення передаються безпосередньо від одного співрозмовника до іншого.

Кожна система миттєвого обміну повідомленнями повинна мати такі складові:

- Система ідентифікації (адресації) клієнтів.
- Система обліку стану клієнтів (хто є підключений, а хто ні)
- Система доставки повідомлень (зазвичай передає повідомлення через комп'ютерні мережу, але може наприклад безпосередньо до іншого користувача на тому самому комп'ютері)

Система ідентифікації (адресації) клієнтів

У безсерверних протоколів виникають проблеми ідентифікації співрозмовника. Для точної ідентифікації клієнта вони можуть використовувати лише фізичну мережеву адресу (IP). Це створює складність, тому що на одному комп'ютері можуть знаходитися декілька користувачів. Для вирішення цієї проблеми і були створені серверні протоколи. У серверних протоколах виділяється сервер, який веде облік користувачів. На сервері потрібно зареєструватися використовуючи ідентифікатор та пароль (необов'язково). Потім ідентифікація проходить через сервер. Ідентифікатори користувачів це зазвичай номер (ICQ) або спеціальне ім'я користувача (логін). У таких протоколах, як, наприклад, Extensible XMPP (Jabber) логін так само як і у електронних адресах містить домен і має вигляд ім'я_користувача@домен

Система обліку стану користувачів

Більшість ІМ-клієнтів дозволяє користувачам бачити, чи підключені до мережі їх співрозмовники в цей момент. Стан користувачів у месенджерах називається статусом. Існує три основних статуси, що відображають присутність/відсутність користувача у мережі:

- В мережі / Онлайн (англ. Online) - користувач під'єднаний до мережі і готовий до спілкування.
- Не в мережі / Офлайн (англ. Offline) - користувач поза мережею.

- Невидимий (англ. Invisible) - користувач знаходиться в мережі, але цей спеціальний статус не дозволяє бачити його всім іншим (або лише деяким) користувачам. Натомість вони бачать статус користувача як Не в мережі.

Більшість протоколів дозволяє використовувати безліч статусів і вони можуть містити заданий користувачем текст (наприклад «Зайнятий, пишу дипломну» або «Вийшов в магазин») і додаткове зображення-іконку. Зазвичай використовуються такі статуси:

Відійшов (англ. Away). Традиційно вмикається коли користувач деякий час не користується комп'ютером. Може бути встановлений самим користувачем коли він відходить від комп'ютера або вмикатися автоматично самою програмою-клієнтом завдяки реєстрації часу бездіяльності користувача.

Недоступний (англ. N/A - Non-Available). Традиційно вмикається коли користувач довгий час не користується комп'ютером. Може вмикатися автоматично самою програмою-клієнтом завдяки реєстрації часу бездіяльності користувача.

- Зайнятий (англ. Busy).
- Не турбувати (англ. DND - Do Not Disturb).
- Готовий побалакати (англ. Free for Chat).
- Вдома (англ. At home).
- На роботі (англ. At work).
- Їм (англ. Eating).
- Злий (англ. Evil)

Список контактів

Користувач може створювати власний список контактів. Контакти можуть бути згруповані у групи з назвою. Більшість протоколів дозволяють зберігати список контактів на сервері, що створює певні зручності:

Користувач може отримати його, знаходячись на іншому комп'ютері.

Користувач може задавати власні правила для контактів. Наприклад список заблокованих (заборонених) контактів від яких він не бажає отримувати повідомлення («чорний список»). Або список контактів для яких його статус завжди видимий.

Користувач може зберігати власні примітки для контактів

Система журналювання повідомлень

Повідомлення можуть бути збережені у так званому журналі (історії) повідомлень. Цей журнал може бути переглянутий. Деякі протоколи (GTalk/Extensible Messaging and Presence Protocol|XMPP) дозволяють, так само як і контакти, зберігати журнал на сервері.

Інформація про користувачів

Майже всі поширені протоколи (окрім IRC) дозволяють користувачам задавати власну інформацію про себе у спеціальних анкетах. Деякі протоколи (Extensible Messaging and Presence Protocol|XMPP) використовують для цього стандарт так званої візитної картки VCard. Зазвичай анкети містять такі поля як:

- Нік (нікнейм, псевдонім)
- Справжнє ім’я
- Фото користувача або зображення (аватар)
- Адреса
- Дата народження та вік
- Інтереси
- Про себе

Самі ІМ-клієнти можуть використовувати одну або кілька служб. Останнім часом програми миттєвого обміну повідомленнями стають найпопулярнішим засобом спілкування.

Розроблений програмний продукт повинен забезпечувати:

1. З'єднання клієнта з сервером
2. Реєстрацію користувача

3. Прийом повідомлень
4. Передачу повідомлень

Структура чату повинна містити наступні елементи:

- Головну веб-сторінку - index.html,
- Конфігурацію бази даних - ajax.php,
- Менеджер бази данних - DB.class.php,
- Базовий клас конструктора - ChatBase.class.php,
- Рядок чату - ChatLine.class.php,
- Клас користувачів - ChatUser.class.php,
- Клас методів конфігурації бази даних - Chat.class.php,
- Контейнер класу - chat.css,
- Клас обробки подій - script.js.

Для створення клієнта чату потрібно, натиснути Файл- Створити проект- Visual C#- Додаток Windows Forms.

Під час розробці я використовував такі елементи:

- Button(кнопка Вхід, Надіслати повідомлення) - Label (Опис полів)

- RichTextBox(Вікно відправки та вхідних повідомлень) - MenuStrip(Меню навігації)

Для того щоб наш клієнт міг спілкуватися з сервером потрібно завантажити с файлу налаштування, IP сервера та порт сервера.

```
try {
var sr = new StreamReader(@"Client_info/data_info.txt"); string buffer = sr.ReadToEnd();
sr.Close();

string[] conect_info = buffer.Split(':'); ip = IPAddress.Parse(conect_info[0]); port = int.Parse(conect_info[1]);

label4.ForeColor = Color.Green;
```

label4.Text = "Налаштуваня: \n IP сервера:" + conect_info[0] + "\n Порт сервера" + conect_info[1];

}

Якщо в нас немає даного файлу ми запропонуємо користувачу самостійно вести налаштування. Для цього нам потрібна нова форма Проект-Добавить форму Windows (див. рис. 4.8).

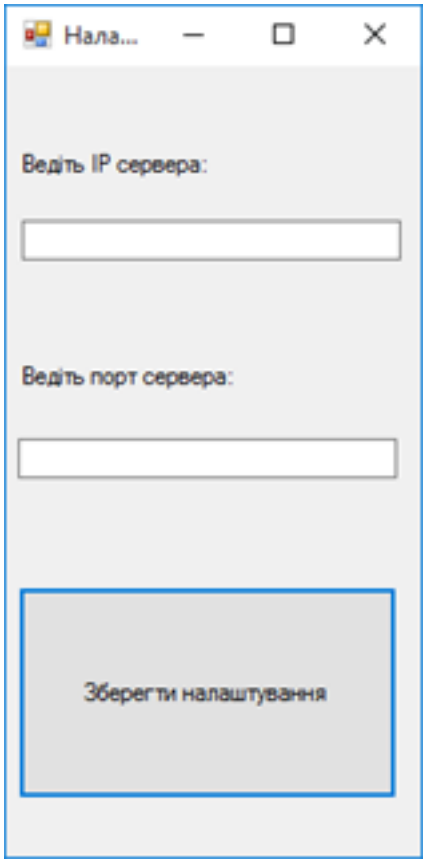


Рис. 4.8.1. - Форма для налаштування з’єднання

private void button1_Click(object sender, EventArgs e)

{

if (textBox1.Text != "" && textBox1.Text != "" && textBox1.Text != "" && textBox1.Text != "")

{ try

```
{

DirectoryInfo data = new DirectoryInfo("Client_info"); data.Create();
var sw = new StreamWriter(@"Client_info/data_info.txt");
sw.WriteLine(textBox1.Text + ":" + textBox2.Text);
sw.Close();
this.Hide(); Application.Restart(); }
catch (Exception ex) {
MessageBox.Show("EROR:" + ex.Message); }
} }
```

Підключення Форми Налаштування до Головної форми.

```
catch (Exception ex) {
label4.ForeColor = Color.Red; label4.Text = "Налаштування не знайдені";
Form2 form = new Form2();
form.Show();
}
private void налаштуванняToolStripMenuItem_Click(object sender,
EventArgs e)
{

Form2 form = new Form2(); form.Show();
}
```

Для того щоб підключитись до серверу нам потрібно вести логін та натиснути на кнопку Вхід.

```
private void button2_Click(object sender, EventArgs e)
```

```
{

if(textBox1.Text!="" && textBox1.Text !="") {
button1.Enabled = true; richTextBox2.Enabled = true;
Client = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
if (ip!=null) {
Client.Connect(ip, port);

th = new Thread(delegate() { RecvMessage(); }); th.Start();
} }
```

Для відправки повідомлення на сервер використовуємо функцію SendMessage.

```
void SendMessage(string message)

{
if (message != "" && message != "") {
byte[] buffer = new byte[1024];

buffer = Encoding.UTF8.GetBytes(message); Client.Send(buffer);
}
}
```

Для отримання повідомлень від сервера використовуємо функцію RecvMessage.

```
void RecvMessage() //сообщения от сервера {
byte[]buffer=new byte[1024];
```

```
for (int i = 0; i < buffer.Length; i++) {
buffer[i] = 0; }
for (; ; ) {
try {
Client.Receive(buffer);

string message =Encoding.UTF8.GetString(buffer); int count =
message.IndexOf(";;;5");
if (count == -1) {
continue; }
string Clear_Message = "";

for (int i = 0; i < count; i++) {
Clear_Message += message[i]; }
for (int i = 0; i < buffer.Length; i++) {
buffer[i] = 0;
}

this.Invoke((MethodInvoker)delegate() {
richTextBox1.AppendText(Clear_Message);
});
}
catch (Exception ex) { } }
}
```

На рис.3.9 зображено форму для здійснення спілкування в чаті зі сторони клієнта. Вона має декілька полів. Поле ведення логіну має формат TexBox та призначено для ведення свого логіну для входу до чату.

Поле відображення тексту спілкування має формат RichTexBox та забезпечує відображення повідомлень всіх користувачів чату які пройшли реєстрацію.

Поле для введення тексту повідомлення має формат RichTexBox та призначено для створення тексту повідомлення. Для відправки повідомлення використовується кнопка Відправити.

Також на формі є поля Меню для налаштування чату та виходу с нього. Також в меню є інформативна складова про автора, який створив програмне забезпечення

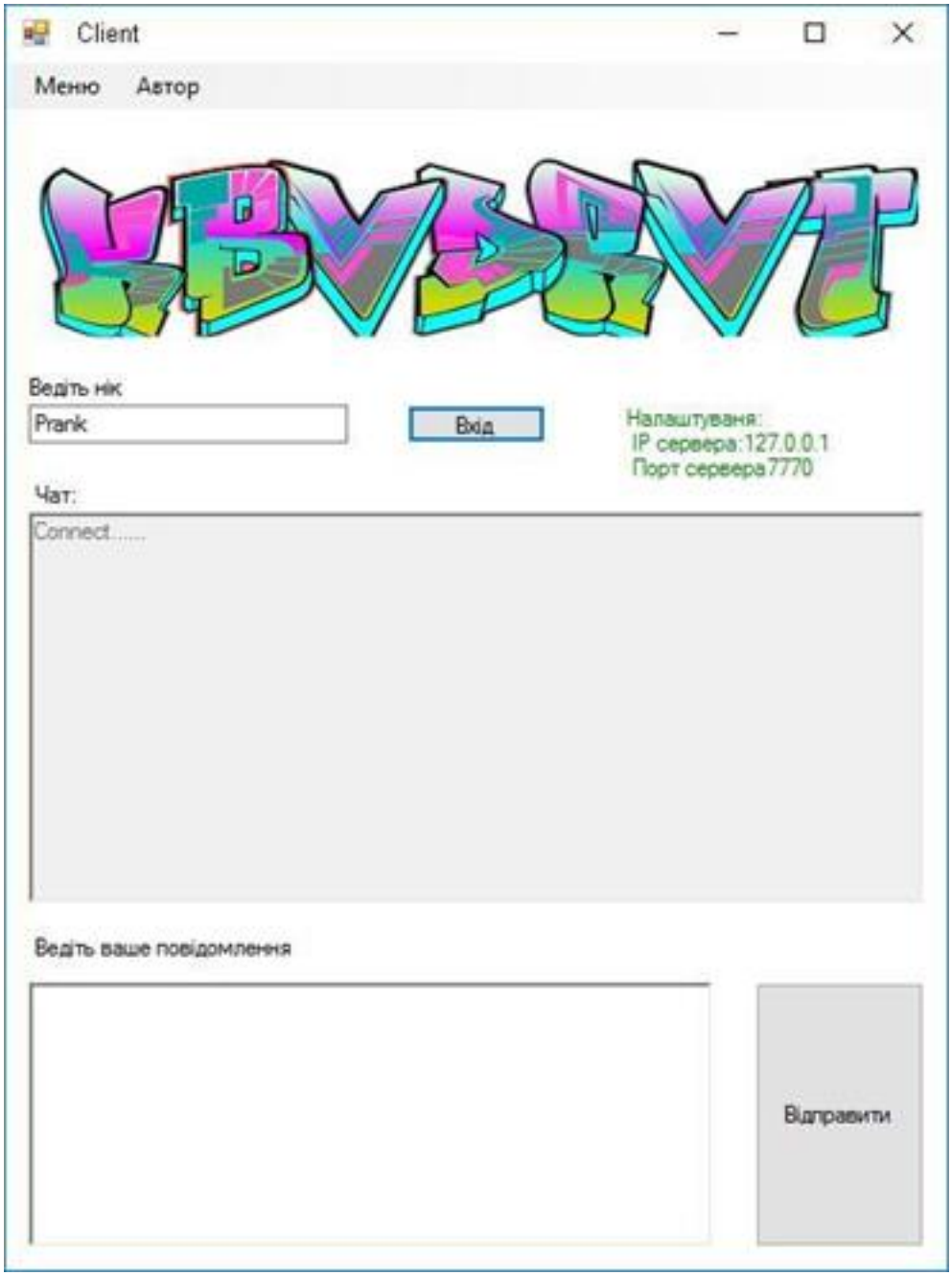


Рис.4.9 - Форма клієнт чату

Active Server Page - активні серверні сторінки - мова програмування, що використовується для створення динамічних веб-сторінок, які обробляються на стороні сервера.

Динамічні сторінки - це такі сторінки, вміст яких змінюється в залежності від дій користувача. Навпаки, статичні сторінки виглядають завжди однаково, незалежно від того, хто і коли її переглядає. Спочатку всі сторінки були

статичними, проте розвиток Інтернету викликало потребу в наданні мінливої інформації. Найпростіші приклади - курси валют, прогнози погоди, які оновлюються новини. Складні - це інтернет-магазини, on-line видання. Так, система Яндекс визначає географічну адресу Вашого IP і пропонує пошук, карту і інші сервіси, актуальні для Вашого регіону.

Із зростанням потреби в динамічно змінних web-сторінках стали з'являтися і технології їх створення, одним з яких і стала технологія ASP.

ASP - технологія від Microsoft, що дозволяє легко розробляти програми для World Wide Web. ASP працює на платформі операційних систем лінії Windows NT і на веб-сервері IIS. ASP не є мовою програмування - це лише технологія попередньої обробки, що дозволяє підключати програмні модулі під час процесу формування Web-сторінки. Відносна популярність ASP заснована на простоті використовуваних мов сценаріїв (VBScript або JScript) і можливості використання зовнішніх COM-компонент.

Технологія ASP передбачає широке використання серверних сценаріїв і об'єктів COM для створення динамічних web-серверів. Засобами технології ASP можна легко створювати інтерактивні web-сторінки, виконувати обробку даних введених користувачем через форми, звертатися до баз даних.

Найбільш цікавими і корисними якостями, якими нас приваблює технологія ASP, можна вважати:

• зручний спосіб об'єднання серверних сценаріїв с HTML;

• скриптовий підхід (інтерпретована мова) - тобто файл з вихідним кодом ASP одночасно є його виконуваним файлом, що спрощує процеси розробки та підтримки;

• концепція "Session" - змінні для кожного користувача з'єднання, як вдале рішення вічної проблеми stateless-протоколу HTTP;

• можливість організації розподіленої архітектури на основі інфраструктури COM, DCOM, COM +. Додаткові можливості, що надаються MTS - такі, наприклад, як контекст об'єктів, пул і т.д .;

• зручний набір об'єктів-утиліт: Server, Application, Request, Response, Session, ObjectContext.

Користувач не може будь-яким чином отримати вміст сторінки ASP, так як web-сервер відправляє йому не саму сторінку, а результат її інтерпретації, таким чином, логіка роботи сторінки прихована від користувачів. Для перегляду потрібно www-браузер, такий як Netscape Navigator, або Microsoft Internet Explorer.

Використання ASP не вимагає специфічних браузерів. Все ASP-скрипти запускаються і виконуються на веб-сервері, причому браузер отримує тільки підсумкові HTML-файли. Microsoft Internet Information Server, начинаючи з версії 3.0, підтримує Active Server Pages.

Функціонування ASP має наступну послідовність. Клієнт запитує ASP-сторінку на веб-сервері. Сервер приймає запит і починає його обробляти. З розширення файлу (".asp") визначає, що даний файл містить ASP-скрипт і починає аналізувати його вміст, послідовно інтерпретуючи і виконуючи вставки ASP-коду. ASP-код, в свою чергу, може містити звернення до різних джерел даних, здійснювати обробку отриманих даних і додавати вміст генерується сторінки. В результаті формується "звичайна" HTML-сторінка (вже не містить ASP-коду), яка і відправляється назад клієнту.

Зовні ASP функціонує також як CGI. При передачі даних від клієнтської форми сервера, останній кодує вхідні дані, а сценарій CGI декодує їх, а потім функціонально обробляє і повертає вихідні дані браузеру.

Аналогічним чином передаються параметри (формат рядка запиту) і здійснюється висновок результатів. Однак продуктивність ASP виявляється набагато вище, тому що при кожному запиті не відбувається окремої завантаження ASP-інтерпретатора. Використання компонент ActiveX також значно підвищують продуктивність веб-сервера.

Крім підвищення продуктивності ASP вирішують також проблему оформлення динамічних web-сторінок. Раніше при використанні CGI

доводилося або вбудовувати текст оформлення web-сторінки в програмний код CGI-сценарію, що ускладнювало подальшу зміну дизайну, або змушувало розробників створювати свої власні системи шаблонів. Використання ASP дозволяє розробнику одночасно працювати над програмним кодом і над оформленням Web-сторінки.

Використовувані засоби для програмування

Web - нормальне середовище програмування, якщо правильно зрозуміти, що є що. У VBScript є всі нормальні конструкції структурного програмування (if, while, case, etc). Є змінні (описувати не обов'язково, тип явно не задається). Підтримуються об'єкти. Робота з ними звичайна - Object.Property, Object.Method. Є ряд вбудованих об'єктів (Request, Response, Session, Server, Connection, Recordset). Можна довстановлювати інші компоненти (завантажувати, купувати, програмувати), наприклад для роботи з електронною поштою.

Переваги ASP.NET

- Компільований код виконується швидше, більшість помилок відловлюється ще на стадії розробки.
- Значно Поліпшена обробка помилок часу виконання, з використанням блоків try…catch
- Користувальницькі Елементи управління (controls) дозволяють виділяти часто використовувані шаблони, такі як меню сайту
- Використання Метафор, вже застосовуються в Windows-додатках, наприклад, таких як елементи управління і події
- Розширений набір елементів управління і бібліотек класів дозволяє швидше розробляти додатки
- ASP.NET Спирається на багатомовні можливості .NET, що дозволяє писати код сторінок на VB.NET, Delphi.NET, Visual C #, J # і т. Д.
- Можливість кешування всієї сторінки або її частини для збільшення продуктивності

- Можливість кешування даних, що використовуються на сторінці
- Можливість поділу візуальної частини та бізнес логіки по різних файлах («code behind»)
- Розширена Модель обробки запитів
- Розширена Подієва модель
- Розширена Модель серверних елементів управління
- Наявність Master-сторінок для завдання шаблонів оформлення сторінок

Починати створення web-сайту на платформі ASP.NET, як і будь-якого іншого, найкраще з розробки проекту, який повинен включати в себе детальний опис функціональності сайту, його архітектуру і приблизний дизайн.

При розробці дизайну, найкраще використовувати Майстер сторінки (Master Pages), які, по суті, є деякими шаблонами сторінок сайту. Майстер сторінки - одна з найсучасніших технологій web-програмування, використовуючи їх, Ви зможете легко підтримувати єдиний дизайн сайту. У разі необхідності змінити дизайн буде досить відредагувати Майстер сторінки для розділів сайту, всі інші сторінки, яких може бути кілька сотень або тисяч, змінювати не доведеться.

Після створення Майстер сторінок для розділів сайту, слід перейти до створення звичайних сторінок. Категорично не рекомендується зберігати контент сайту всередині звичайних ASP сторінок, краще використовувати для цього XML - файли або базу даних.

Приступаючи до реалізації функціональності сайту, слід ретельно ознайомитися з класами стандартної бібліотеки, особливо тими, які можуть бути корисні в кожному конкретному випадку. Дотримуючись даного раді, Ви можете істотно скоротити час, що витрачається на програмування функціональності web-сайту.

Використання додаткових коштів

У разі якщо користувачеві необхідно додати до свого web-проекту деякі стандартні функції, такі як, наприклад, форум або гостьова, то цілком ймовірно має сенс використовувати готове рішення, а не програмувати все самому.

Останнім часом в мережі стало з'являтися велика кількість ASP скриптів і різних бібліотек класів для платформи Asp.Net 2.0, багато з яких поширюються безкоштовно, або коштують значно дешевше вашого часу, необхідного для реалізації подібної функціональності самостійно.

Для прискорення процесу розробки web-проекту, можна взяти за його основу, яку або з вільно розповсюджуваних або комерційних CMS під платформу Asp.Net 2.0.

У мережі доступні багато бібліотек класів, призначені для вирішення різних завдань:

· SharpZipLib.dll - для обробки заархівіруваних за допомогою алгоритму стиснення zip файлів і потоків

· HtmlAgilityPack.dll - бібліотека, призначена для ефектіфной роботи з Html форматуванням. Дозволяє легко здійснювати пошук різних тегів, читати і змінювати їх значення і атрибути. Прекрасно працює з погано форматувати html даними, які мають помилки в розмітці.

· EdtFTPnet.dll - набір класів для роботи з ftp, за допомогою яких можна переглядати вміст ftp серверів, завантажувати і видаляти файли, створювати і перейменовувати каталоги, а також виконувати багато інших дій, передбачені протоколом ftp.

У разі необхідності, також можна розробити власні бібліотеки класів або настроюються серверні елементи управління, які можна багаторазово використовувати на різних сторінках сайту, і навіть в інших ваших проектах.

Основні характеристики ASP.NET

ASP.NET є єдиною моделлю для розробки веб-додатків із застосуванням мінімуму коду, яка містить служби, необхідні для побудови веб-додатків для підприємств. ASP.NET є частиною платформи .NET Framework, а тому

забезпечує доступ до класів цієї платформи. Додатки можуть бути написані на будь-якій мові середовища CLR, включаючи Microsoft Visual Basic, C #, JScript .NET і J #. Ці мови дозволяють розробляти програми ASP.NET, які можуть використовувати всі переваги середовища CLR, типовий безпеки, успадкування і т. Д.

У ASP.NET входить:

- Платформа для розробки сторінки і елементів управління
- Компілятор ASP.NET
- Інфраструктура захисту даних
- Можливості по управлінню станом
- Конфігурація додатків
- Спостереження і настройка продуктивності
- Підтримка налагодження
- Платформа веб-служб XML
- Розширюване середовище розміщення і управління життєвим циклом додатка

Функціональні можливості ASP.NET

Розширюване середовище конструктора

Платформа для розробки сторінки і елементів управління

Структура сторінок і елементів управління ASP.NET - структура програмування, яка виконується на веб-сервері для динамічного створення та відображення веб-сторінок ASP.NET. Веб-сторінки ASP.NET можна переглядати в будь-яких веб-браузерах або клієнтських пристрої, ASP.NET відображає розмітку (таку як HTML) в запитуючій оглядачі. Як правило, можна використовувати одну і ту ж сторінку для різних оглядачів, так як ASP.NET відображає відповідну розмітку для запитувача оглядача. Однак можна розробляти веб-сторінки ASP.NET для певних оглядачів, наприклад для Microsoft Internet Explorer 6, і використовувати широкі можливості конкретного

оглядача. ASP.NET підтримує елементи управління для мобільних пристроїв, наприклад таких пристроїв веб-доступу, як стільникові телефони, портативні комп'ютери і PDA.

Веб-сторінки ASP.NET є повністю об'єктно-орієнтованими. На сторінках ASP.NET з елементами HTML можна працювати, використовуючи властивості, методи і події. Структура сторінок ASP.NET надає єдину модель відгуку на клієнтські події в коді, що виконується на сервері, тому реалізація поділу клієнта і сервера, яка використовується в веб-додатках, не потрібна. Вона також автоматично обробляє стану сторінки і її елементів управління під час циклу обробки сторінки. Додаткові відомості див. У розділі Загальні відомості про веб-сторінках ASP.NET.

Структура сторінок і елементів управління ASP.NET також інкапсулює загальні функціональні можливості призначеного для користувача інтерфейсу в зручні повторно використовувані елементи управління. Елементи управління, написані одного разу, можна використовувати в багатьох сторінках. Вони вбудовуються в веб-сторінку ASP.NET, на якій вони розміщуються під час відтворення.

Структура сторінок і елементів управління ASP.NET також надає можливості управління відображенням і поведінкою веб-вузла за допомогою тем і обкладинок. Можна визначити теми і обкладинки і потім застосувати їх на рівні сторінки або елементу управління. Додаткові відомості див. У розділі Загальні відомості про теми та обкладинках ASP.NET.

Крім тим можна визначити головні сторінки, що дозволяють створити макет сторінки, який можна буде використовувати для всіх сторінок в додатку. Одна головна сторінка визначає макет і стандартну поведінку, які можна використовувати для всіх сторінок (або групи сторінок) в додатку. Потім можна створити окремі сторінки вмісту, що включають вміст, пов'язане зі сторінкою, яке Ви бажаєте бачити. Коли користувачі запитують сторінку вмісту, вихідна сторінка являє собою поєднання структури головної сторінки і вмісту зі

сторінки вмісту. Додаткові відомості див. У розділі Загальні відомості про головних сторінках ASP.NET.

Весь код ASP.NET компілюється, що дозволяє використовувати строгий контроль типів, оптимізації продуктивності, раннє зв'язування і інші переваги. Після компіляції коду середу CLR компілює код ASP.NET в машинний код, тим самим забезпечуючи підвищення продуктивності.

У ASP.NET включений компілятор, що виконує компіляцію всіх компонентів програми, включно зі сторінками і елементи управління, в збірку, що ефір розміщення ASP.NET може використовувати в подальшому для обслуговування запитів користувача. Додаткові відомості див. У розділі Загальні відомості про компіляції в ASP.NET.

Інфраструктура захисту даних

Крім можливостей захисту даних, .NET, ASP.NET надає додаткову інфраструктуру для перевірки автентичності та авторизації доступу користувачів, а також інших завдань безпеки. Можна виконувати перевірку автентичності за допомогою перевірки автентичності Windows, що надається службами IIS, або за допомогою власної бази даних користувача, використовуючи перевірку справжності форм ASP.NET і членство ASP.NET. Також можна управляти перевіркою достовірності веб-додатки за допомогою груп Windows або власної бази даних ролей, використовуючи ролі ASP.NET. Ці схеми легко додати, видалити або замінити в залежності від вимог до додатка.

ASP.NET завжди запускається з певним ідентифікатором Windows, тому можна захищати програму за допомогою можливостей Windows, таких як списки управлінням доступу NTFS, дозволу баз даних і т. Д. За докладнішою інформацією про ідентифікатор ASP.NET см. В розділах Налаштування посвідчення процесу ASP. NET і Уособлення ASP.NET.

Можливості по управлінню станом

ASP.NET забезпечує вбудовану функціональність для управління станом, що дозволяє зберігати дані між запитами сторінок, наприклад, відомості про

клієнтів або вміст кошика покупок. Можна зберігати і управляти відомостями, пов'язаними з додатком, сеансом, сторінкою і користувачем, а також відомостями, визначеними розробником. Ці відомості можуть не залежати від елементів управління на сторінці.

ASP.NET надає розподілені кошти управління станом, що дозволяють управляти відомостями про стан в декількох примірниках одного додатка на одному або декількох комп'ютерах. Додаткові відомості див. У розділі Загальні відомості про управління станом ASP.NET.

конфігурація ASP.NET

Програми ASP.NET використовують систему конфігурації, що дозволяє визначати параметри конфігурації для веб-сервера, веб-вузла і окремих додатків. Параметри конфігурації можна застосовувати в момент першого розгортання додатків ASP.NET, а також в будь-який момент додавати або переглядати параметри конфігурації з мінімальним впливом на працюючі веб-додатки і сервери. Параметри конфігурації ASP.NET зберігаються в файлах XML. Так як ці файли XML є текстовими ASCII-файлами, які можна читати і змінювати, вносити зміни в конфігурацію веб-додатки нескладно. Можна розширити схему конфігурації відповідно до своїх уподобань. Додаткові відомості див. У розділі Загальні відомості про конфігурації ASP.NET.

ASP.NET містить засоби забезпечення безпеки, що дозволяють вести спостереження і налаштовувати продуктивність програми ASP.NET. Спостереження за станом системи ASP.NET дозволяє повідомляти користувача про ключові події, що надають відомості про стан програми та умов виникнення помилок. Ці події відображають поєднання характеристик діагностики і моніторингу і забезпечують більшу ступінь гнучкості в питаннях, що стосуються протоколювання. Додаткові відомості див. У розділі Загальні відомості про моніторинг працездатності системи ASP.NET.

ASP.NET підтримує дві групи лічильників продуктивності, доступних в додатку:

Група лічильників системної продуктивності ASP.NET;

ДГрупа лічильників продуктивності додатка ASP.NET.

У ASP.NET інфраструктура налагодження під час виконання використовується для того, щоб забезпечити підтримку налагодження програм на різних мовах і на різних комп'ютерах. Можна налагоджувати керовані і некеровані об'єкти, а також всі мови, які підтримуються середовищем CLR, і мови сценаріїв. Додаткові відомості див. У розділі Налагодження ASP.NET.

ASP.NET підтримує веб-служби XML. Веб-служба XML - це компонент, який містить функціональні можливості для ведення бізнесу, що дозволяє веб-додаткам обмінюватися відомостями, використовуючи стандарти повідомлень HTTP і XML для передачі даних через брандмауери. Веб-служби XML не прив'язані до будь-якої технології компонентів або певним угодам виклику об'єктів. В результаті веб-служби XML можуть використовуватися програмами, написаними на будь-якій мові, що використовують будь-яку компонентну модель і працюють в будь-якій операційній системі. Додаткові відомості див. У розділі XML Web Services Using ASP.NET.

```
Структура таблицы `webchat_lines`
CREATE TABLE `webchat_lines` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `author` varchar(16) NOT NULL,
  `gravatar` varchar(32) NOT NULL,
  `text` varchar(255) NOT NULL,
  `ts` timestamp NOT NULL default CURRENT_TIMESTAMP,
  PRIMARY KEY  (`id`),
  KEY `ts` (`ts`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;
-- Структура таблицы `webchat_users`
CREATE TABLE `webchat_users` (
  `id` int(10) unsigned NOT NULL auto_increment,
```

```
  `name` varchar(16) NOT NULL,
  `gravatar` varchar(32) NOT NULL,
  `last_activity` timestamp NOT NULL default CURRENT_TIMESTAMP,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `name` (`name`),
  KEY `last_activity` (`last_activity`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;
<?php
class DB {
        private static $instance;
        private $MySQLi;
        private function __construct(array $dbOptions){
        $this->MySQLi = @ new mysqli(       $dbOptions['db_host'],
        $dbOptions['db_user'],

$dbOptions['db_pass'],
        $dbOptions['db_name'] );
                if (mysqli_connect_errno()) {
                        throw new Exception('Ошибка базы данных.');
                }
                $this->MySQLi->set_charset("utf8");
        }
        public static function init(array $dbOptions){
                if(self::$instance instanceof self){
                        return false;
                }
                self::$instance = new self($dbOptions);
        }
        public static function getMySQLiObject(){
```

```
                return self::$instance->MySQLi;
        }
        public static function query($q){
                return self::$instance->MySQLi->query($q);
        }
        public static function esc($str){
                return self::$instance->MySQLi-
>real_escape_string(htmlspecialchars($str));
        }
}
?>
```

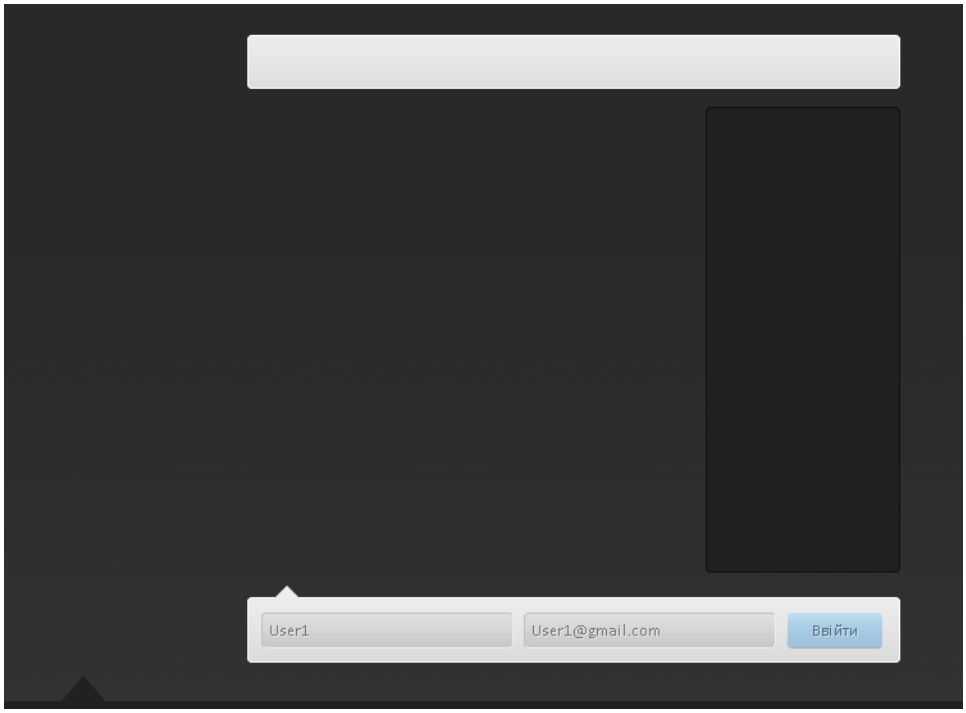


Рис. 4.10 - Занесення користувача в базу даних

Приклад роботи додатку

Код сторінки веб-чату:

<! DOCTYPE html>

```
<Html>
<Head>
<Meta http-equiv = "Content-Type" content = "text / html; charset = utf-8" />
<Title> веб чат </ title>
<Link rel = "stylesheet" type = "text / css" href = "js / jScrollPane / jScrollPane.css" />
<Link rel = "stylesheet" type = "text / css" href = "css / page.css" />
<Link rel = "stylesheet" type = "text / css" href = "css / chat.css" />
</ Head>
<Body>
<Div id = "chatContainer">    <Div id = "chatTopBar" class = "rounded"> </ div>
  <Div id = "chatLineHolder"> </ div>
      <Div id = "chatUsers" class = "rounded"> </ div>
  <Div id = "chatBottomBar" class = "rounded">
  <Div class = "tip"> </ div>       <Form id = "loginForm" method = "post" action = "">
      <Input id = "name" name = "name" class = "rounded" maxlength = "16" />
      <Input id = "email" name = "email" class = "rounded" />
      <Input type = "submit" class = "blueButton" value = "Увійти" />
    </ Form>       <Form id = "submitForm" method = "post" action = "">
      <Input id = "chatText" name = "chatText" class = "rounded" maxlength = "255" />
      <Input type = "submit" class = "blueButton" value = "Відправити" />
    </ Form>    </ Div>
</ Div>
<Div id = "footer">
<Div class = "tri"> </ div>
<H1> Робимо AJAX веб чат з використанням PHP і jQuery </ h1>
</ Div>
```

<Script src = "http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"> </ script>

<Script src = "js / jScrollPane / jquery.mousewheel.js"> </ script>

<Script src = "js / jScrollPane / jScrollPane.min.js"> </ script>

<Script src = "js / script.js"> </ script>

</ Body>

</ Html>

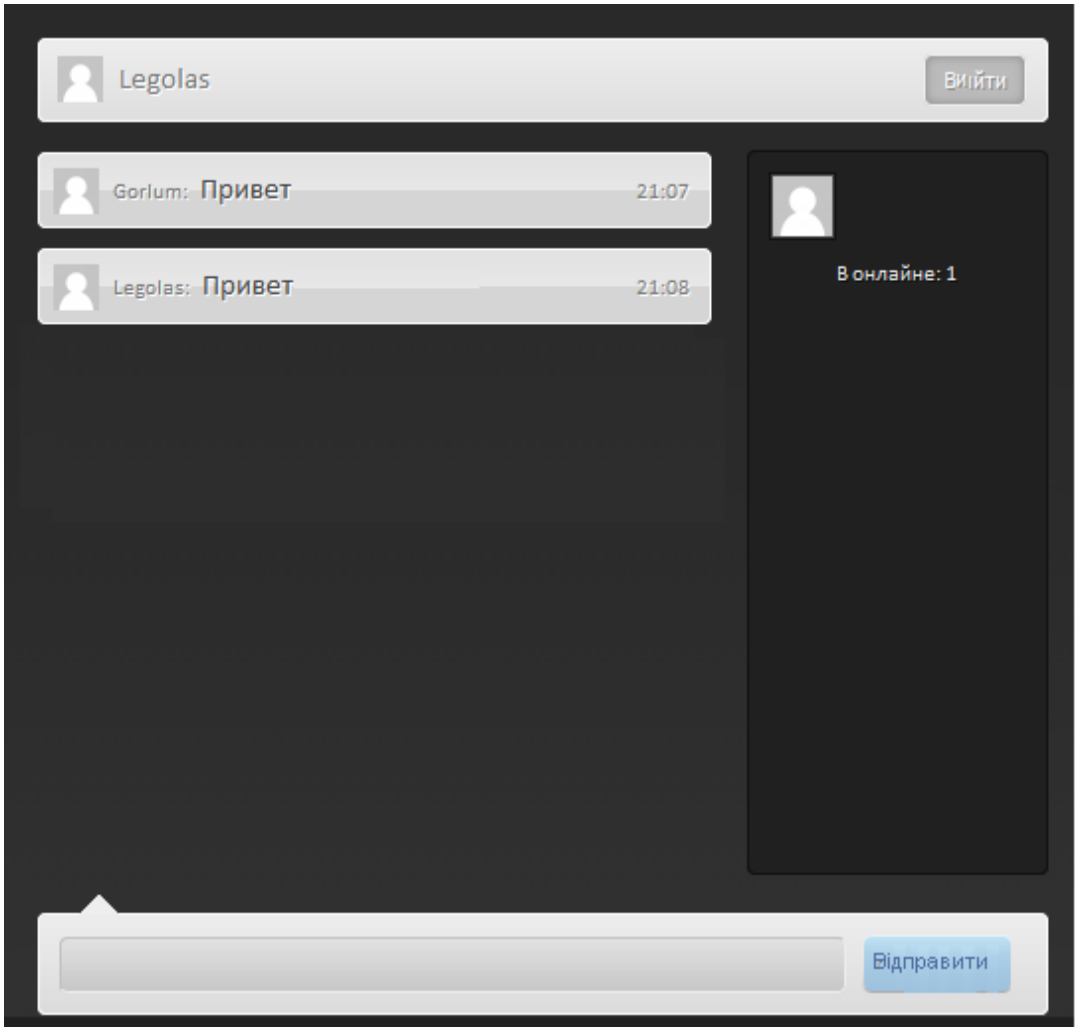


Рис. 4.11 - Переписка в чаті в режимі реального часу

ВИСНОВКИ

- У дипломній роботі розглянуті існуючі протоколи та реалізації розподілених систем обміну повідомленнями.
-
- Запропонована архітектура системи обміну повідомленнями, що характеризується високою автономністю клієнтів, можливістю відкритого шифрування повідомлень без участі сервера та полегшеним масштабуванням серверів.
-
- Розроблений протокол взаємодії клієнтської та серверної частин.
-
- Розроблена архітектура серверного та клієнтського ПЗ.
-
- Реалізований макет сервісу.

Результати роботи можуть бути застосовані для побудови розподіленого сервісу обміну повідомленнями з високою надійністю.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1.Дипломне проектування за напрямами підготовки ”Прикладна математика”, „Комп’ютерна інженерія”, „Програмна інженерія” [Текст] : навч.-метод. посіб. / Є.С. Сулема : за заг. ред. І.А. Дички — К. : НТУУ «КПІ», 2011. — 224 с. – Бібліогр. : с. 77 — 400 пр.

2.Положення про організацію дипломного проектування та державну атестацію студентів НТУУ “КПІ” [Текст] / уклад. В.Ю. Угольніков ; за заг. ред. Ю.І. Якименка. — К. : ВПК "Політехніка", 2006. — 84 с.

3.Положення про організацію навчального процесу в НТУУ “КПІ” [Текст] / уклад. Г.Б. Варламов, В.П. Головенкін, В.І. Тимофєєв, В.І. Шеховцов ; за заг. ред. Ю.І. Якименка. — К. : ІВЦ «Видавництво "Політехніка"», 2004. — 72 с.

4. Дуванов, А.А. Web-конструирование HTML [Текст] / А.А. Дуванов. — СПб. : БХВ-Петербрург, 2003. — 250 с.

5. Парс, Р. Основы ASP.NET AJAX [Текст] / Р. Парс, Л. Морони, Дж. Гриб. — М. : Символ-Плюс, 2004. — 314 с.

6. Смирнова, И.Е. Начала Web-дизайна [Текст] / И.Е. Смирнова. — СПб. : БХВ-Петербург, 2003. — 125 с.

7. Шварц, Р. Изучаем Perl [Текст] / Р. Шварц, Т. Кристиансен. — К. : BHV, 2000. — 320 с.

8. Ратшиллер, Т. PHP4: разработка Web-приложений [Текст] / Т. Ратшиллер, Т. Геркен. — СПб. : Питер, 2001. — 384 с.

9. Томсон, Л. Разработка Web-приложений на PHP и MySQL [Текст] / Л. Томсон, Л. Веллинг. — К. : ДиаСофт, 2001. — 672 с.

10. Web-програмування [Електронний ресурс]. — Режим доступу : http://webstudio2u.net/ua/programming/96-cms.html

11. Спейнауэр, С. Справочник Web-мастера [Текст] / С. Спейнауэр, В. Куэрсиа. — К. : BHV, 1997. — 368 с.

12. Яргер, Р. MySQL и mSQL. Базы данных для небольших предприятий и Интернета [Текст] / Р. Яргер, Дж. Риз, Т. Кинг. — СПб. : Символ-Плюс, 2000. — 560 с.

13. Хилайер, С. Программирование Active Server Pages [Текст] / С. Хилайер, Д. Мизик. — М. : Русская редакція

14. Биков В.Ю. Моделі організаційних систем відкритої освіти : монографія / В.Ю. Биков. – К. : Атіка, 2009. – 682 с.

15. Носенко Ю.Г. Еволюція хмарних обчислень як актуального засобу навчання / Носенко Ю.Г. // Інформатика та інформаційні технології в навчальних закладах. – № 5. – 2015. – С. 16-21.

16. Носенко Ю.Г. Хмарні технології у просторі відкритої освіти / Юлія Носенко // Моделювання й інтеграція сервісів хмаро орієнтованого навчального середовища : монографія; за заг. ред. С.Г. Литвинової. – К. : ЦП «Компринт», 2015. – С. 24-34.

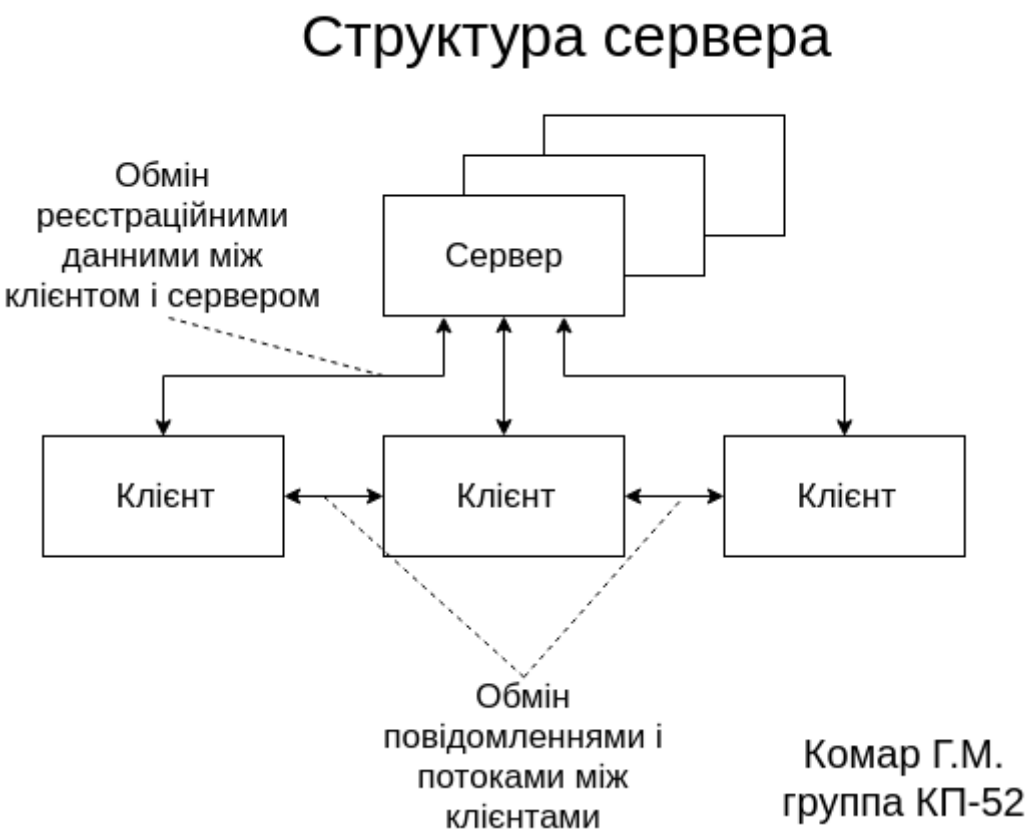
17. Шишкіна М.П. Актуальні напрями розвитку хмаро орієнтованого навчально-наукового середовища педагогічних систем: з досвіду роботи Інституту інформаційних технологій і засобів навчання НАПН України / Шишкіна М.П., Носенко Ю.Г. // Науковий часопис НПУ імені М.П. Драгоманова. Серія 2. Комп'ютерно-орієнтовані системи навчання: Зб. наук. праць / Редрада. – К. : НПУ імені М.П. Драгоманова, 2015. – 16 (23). – С. 153-158.

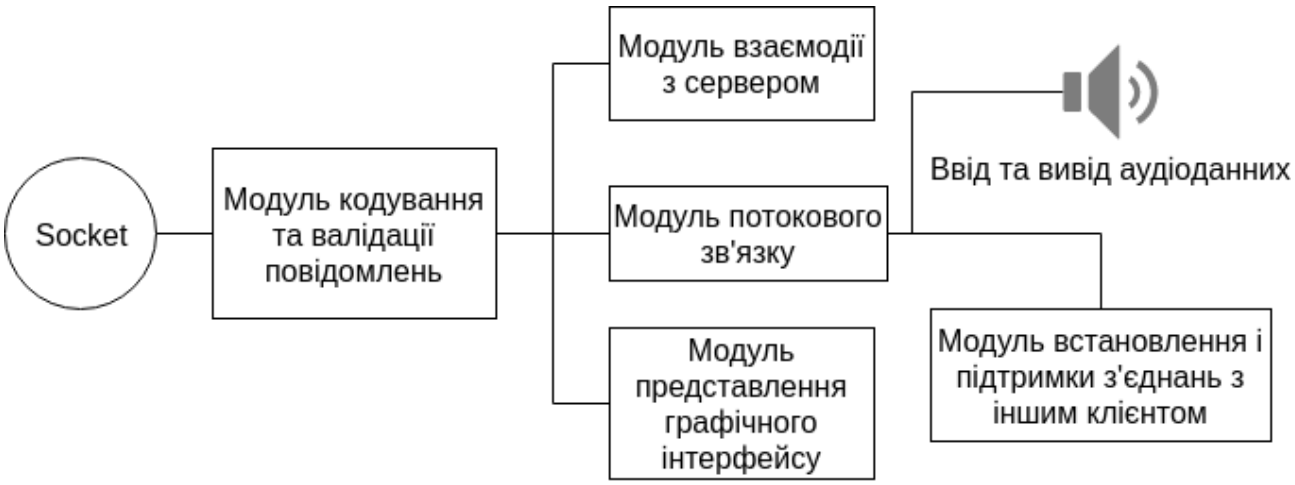
18. ISO/IEC 17788:2014(E). Information technology – Cloud computing – Overview and vocabulary : International Standard. – Switzerland : ISO/IEC, 2014. – 14 p.

19. The NIST Definition of Cloud Computing : Recommendations of the National Institute of Standards and Technology [Electronic resource]. – Access mode: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

Д О Д А Т К И

ДОДАТОК 1.Копії графічних зображень





Д.П.045440-08-99
Розподілений сервіс обміну повідомленнями та потокового зв’язку.
Структура модулів клієнтської частини

Схема алгоритму встановлення потокового зв'язку

Комар Г.М.
група КП-52

Д.П.045440-08-99
Розподілений сервіс обміну повідомленнями та потокового зв'язку.
Структура модулів серверної частини

Додаток 2. Лістинги програм

ДОДАТОК А . КОД СЕРВЕРУ МЕРЕЖЕВОГО ЧАТУ

Код main.cpp

```
#pragma comment(lib,"Ws2_32.lib")
#include <WinSock2.h>
#include <iostream>
#include <WS2tcpip.h>

SOCKET Connect;
SOCKET*Connections;
SOCKET Listen;

int ClientCount=0;

void SendMessageToClient(int ID) //отправка сообщений всем пользователям
{
 char* buffer =new char[1024];
 for(;;Sleep(75))
 {
 memset(buffer,0,sizeof(buffer));
 if(recv(Connections[ID],buffer,1024,NULL))
 {
 printf(buffer);
 printf("\n");
 for(int i=0; i<=ClientCount;i++)
 {
 send(Connections[i],buffer,strlen(buffer),NULL);
 }
```

```
 }
 }
 delete buffer;
}

int main()
{

 setlocale(LC_ALL,"russian");
 WSAData data;
 WORD version= MAKEWORD(2,2);
 int res = WSAStartup(version,&data);
 if(res!=0)
 {
 return 0;

 }

 struct addrinfo hints;
 struct addrinfo * result;

 Connections=(SOCKET*)calloc(64,sizeof(SOCKET));

 ZeroMemory(&hints,sizeof(hints));

 hints.ai_family=AF_INET;
 hints.ai_flags=AI_PASSIVE;
 hints.ai_socktype=SOCK_STREAM;
 hints.ai_protocol=IPPROTO_TCP;

 getaddrinfo(NULL,"7770",&hints,&result);
```

```
Listen=socket(result->ai_family,result->ai_socktype,result->ai_protocol);
bind(Listen,result->ai_addr,result->ai_addrlen);
listen(Listen,SOMAXCONN);

freeaddrinfo(result);
printf("Start server");
char m_connect[]="Connect.....;;;5";
for(;;Sleep(75))
{

if(Connect=accept(Listen,NULL,NULL))//потключение клиентов
{

printf("Client connect...\n");
Connections[ClientCount]=Connect; //сохраняем сокет клиента
send(Connections[ClientCount],m_connect,strlen(m_connect),NULL);
ClientCount++;

CreateThread(NULL,NULL,(LPTHREAD_START_ROUTINE)SendMessageToClie
nt,(LPVOID)(ClientCount-1),NULL,NULL);


}

}

return 1;

}
```

Додаток Б. КОД КЛІЄНТА МЕРЕЖЕВОГО ЧАТУ

Код Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Threading;

namespace MyClient_V22
{
 public partial class Form1 : Form
 {
 static private Socket Client;
 private IPAddress ip = null;
 private int port = 0;
 private Thread th;

 public Form1()
 {
 InitializeComponent();
 richTextBox1.Enabled = false;
 richTextBox2.Enabled = false;
 button1.Enabled = false;
```

```
try
{
var sr = new StreamReader(@"Client_info/data_info.txt");//считуем настройки
string buffer = sr.ReadToEnd();
sr.Close();
string[] conect_info = buffer.Split(':');
ip = IPAddress.Parse(conect_info[0]);
port = int.Parse(conect_info[1]);

label4.ForeColor = Color.Green;
label4.Text = "Налаштуваня: \n IP сервера:" + conect_info[0] + "\n Порт сервера" + conect_info[1];

}

catch (Exception ex)
{
label4.ForeColor = Color.Red;
label4.Text = "Налаштування не знайдені";
Form2 form = new Form2();
form.Show();

}



}

private void label4_Click(object sender, EventArgs e)
{
```

```
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {

        }

        private void menuStrip1_ItemClicked(object sender,
ToolStripItemClickedEventArgs e)
        {

        }

        private void налаштуванняToolStripMenuItem_Click(object sender,
EventArgs e)
        {
        Form2 form = new Form2();
        form.Show();
        }

        void SendMessage(string message) //отправка сообщений на сервер
        {

        if (message != "" && message != "")
        {
        byte[] buffer = new byte[1024];
        buffer = Encoding.UTF8.GetBytes(message);
        Client.Send(buffer);

        }
        }
```

```
void RecvMessage() //сообщения от сервера
{
byte[]buffer=new byte[1024];
for (int i = 0; i < buffer.Length; i++)
{
buffer[i] = 0;
}
for (; ; )
{
try
{
Client.Receive(buffer);
string message =Encoding.UTF8.GetString(buffer);
int count = message.IndexOf(";;;5");
if (count == -1)
{
continue;
}
string Clear_Message = "";
for (int i = 0; i < count; i++)
{
Clear_Message += message[i];
}
for (int i = 0; i < buffer.Length; i++)
{
buffer[i] = 0;
}
this.Invoke((MethodInvoker)delegate()
{
richTextBox1.AppendText(Clear_Message);
```

```
        });

        }
        catch (Exception ex) { }
        }
        }
        private void button2_Click(object sender, EventArgs e)
        {//потключение к серверу
        if(textBox1.Text!="" && textBox1.Text !="")
        {
        button1.Enabled = true;
        richTextBox2.Enabled = true;
        Client = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
        if (ip!=null)
        {
        Client.Connect(ip, port);
        th = new Thread(delegate() { RecvMessage(); });
        th.Start();

        }



        }
         }

        private void button1_Click(object sender, EventArgs e)
        {
        SendMessage("\n" + textBox1.Text + ":" + richTextBox2.Text + ";;;5");
        richTextBox2.Clear();
```

```
        }
        private void авторToolStripMenuItem_Click(object sender, EventArgs e)
        {
        MessageBox.Show("Беспалов Алексей Александрович.\n Студент групы ДИ-112");
        }
        private void вихідToolStripMenuItem_Click(object sender, EventArgs e)
        {
        if(th!=null) th.Abort();
        if (Client != null) { Client.Close(); }
        Application.Exit();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
        }
        private void richTextBox1_TextChanged(object sender, EventArgs e)
        {
        }

        private void менюToolStripMenuItem_Click(object sender, EventArgs e)
        {
        }
        }
       }
```

Додаток В. КОД ФОРМИ НАЛАШТУВАННЯ МЕРЕЖЕВОГО ЧАТУ

Код Form2.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
namespace MyClient_V22
{
 public partial class Form2 : Form
 {
 public Form2()
 {
 InitializeComponent();
 }

 private void button1_Click(object sender, EventArgs e)
 {
 if (textBox1.Text != "" && textBox1.Text != "" && textBox1.Text != "" && textBox1.Text != "")
 {
 try
 {
 DirectoryInfo data = new DirectoryInfo("Client_info");
 data.Create();
```

```
var sw = new StreamWriter(@"Client_info/data_info.txt");
sw.WriteLine(textBox1.Text + ":" + textBox2.Text);
sw.Close();
this.Hide();
Application.Restart();
}
catch (Exception ex)
{
MessageBox.Show("EROR:" + ex.Message);
}

}
}

private void Form2_Load(object sender, EventArgs e)
{

}
}
}
```

ДОДАТОК 3 Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Розподілений сервіс обміну повідомленнями та потокового зв'язку

Виконав: Комар Григорій Миколайович
Керівник к.т.н., доцент кафедри ПЗКС Вунтесмері Ю.В.

Постановка задачі

Об'єкт дослідження: Розподілений сервіс обміну повідомленнями і потокового зв'язку

Предмет дослідження: доставка повідомлень і передача потоків данних в розподіленому середовищі

Мета: розробити розподілений сервіс доставки повідомлень і передачі потоків даних з незалежними серверами, стійкістю до блокувань і обміном данними безпосередньо між клієнтами

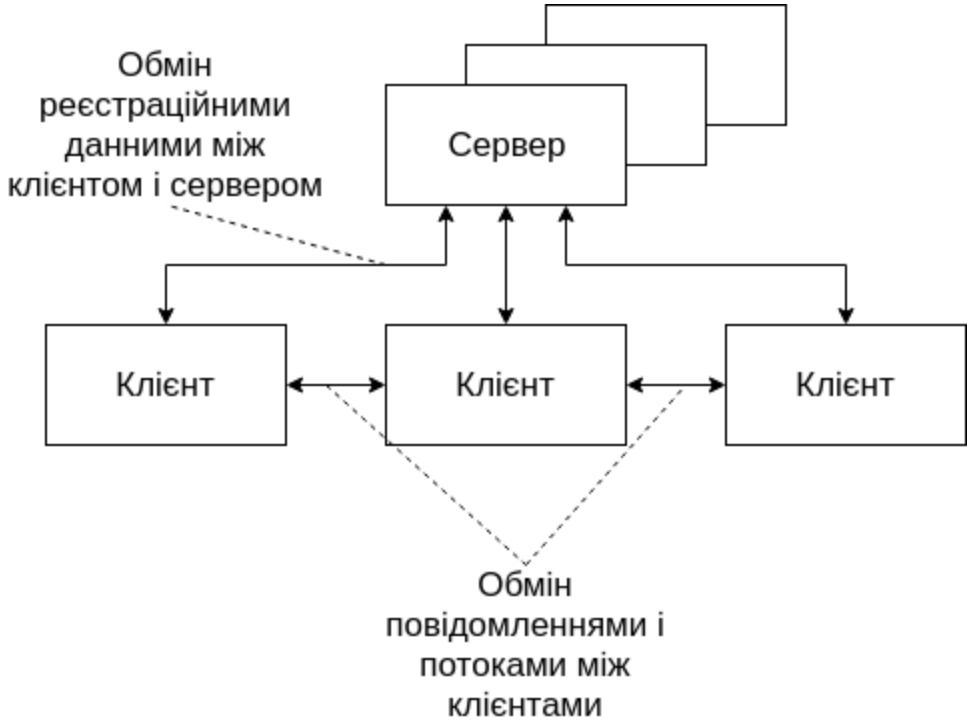
Прототипи та аналоги

- XMPP
- Skype
- Telegram
- SIP

Відмінності аналогів у порівнянні з розроблюваним сервісом:

- Неможлива або ускладнена міграція клієнта на новий сервер
- Вразливість до блокування
- Передача даних клієнтів через сторонні сервери

Структура сервера



Протокол реєстрації клієнта

Доставка і відправка повідомлень клієнта

Клієнт під час періодичних запитів до сервера отримує системні повідомлення необхідні для налаштування зв`язку і текстові повідомлення від користувачів.

Кожне повідомлення містить унікальний ідентифікатор згенерований сервером(GUID)

Відправка повідомлень іншому клієнту відбувається за допомогою запита на сервер. Повідомлення містять цифровий підпис для верифікації на стороні іншого клієнта

Обробка викликів

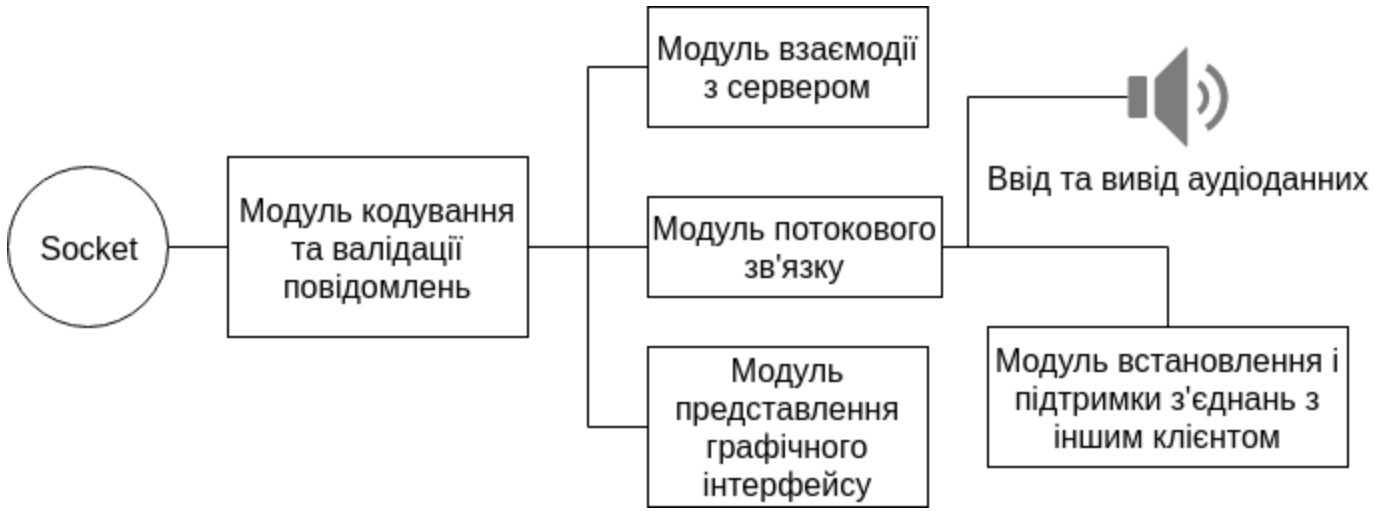
Оновлення інформації клієнтів

Періодично клієнт і сервер обмінюються PING/PONG повідомленнями для перевірки стану з'єднання.

Клієнт має можливість запросити у сервера інформацію про іншого клієнта

Архітектура сервера

Архітектура клієнта



Використані технології

- Сервер реалізовано на мові високого рівня Scala з використанням ООП та функціональних парадигм. Виконання сервера забезпечується віртуальною машиною Java(JVM)
- Клієнт реалізовано на мові високого рівня Rust. Rust компілюється в машинний код і не потребує віртуальної машини для виконання клієнта, завдяки чому забезпечується ни

Перспективи

- Вдосконалення алгоритмів стиснення аудіо-потоків
- Наскрізне шифруфання повідомлень
- Автоматична міграція серверів

Основні результати роботи

- Досліджені існуючі реалізації розподілених систем обміну повідомленнями
- Запропонована структура високонадійного розподіленого сервісу обміну повідомленнями та потокового зв'язку
- Розроблені протоколи обміну між клієнтською та серверною частинами сервісу
- Розроблена архітектура серверного та клієнтського ПЗ
- Реалізований макет сервісу

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

Розподілений сервіс обміну повідомленнями та потокового зв'язку
Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Ю.В.Вунтесмері

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Г.М.Комар

2018

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Розподілений сервіс обміну повідомленнями та потокового зв’язку,який являє собою клієнт –серверний додаток створений на платформі Java та Rust.

2.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність протоколів взаємодії клієнтської та серверної частин;

2) архітектура клієнтського та серверного програмного забезпечення;

3) можливість відкритого шифрування повідомлень без участі сервера та полегшеним масштабуванням серверів;

4) забезпечення належного рівня безпеки даних;

5) зручність роботи ;

6) відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1) функціональне тестування, зокрема на рівні Critical path test (базове тестування);

2) тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);

3) тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію SpecFlow.

Працездатність розподіленого сервісу обміну повідомленнями та потокового зв’язку перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) статичного тестування коду;
4) тестування web-ресурсу в різних web-браузерах;
5) тестування при максимальному навантаженні;
6) тестування стабільності роботи при різних умовах;
7) тестування зручності використання;
8) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

Розподілений сервіс обміну повідомленнями та потокового зв'язку
Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____Ю.В.Вунтесмері

Нормоконтроль:

_____ М.В. Онай

Виконавець:

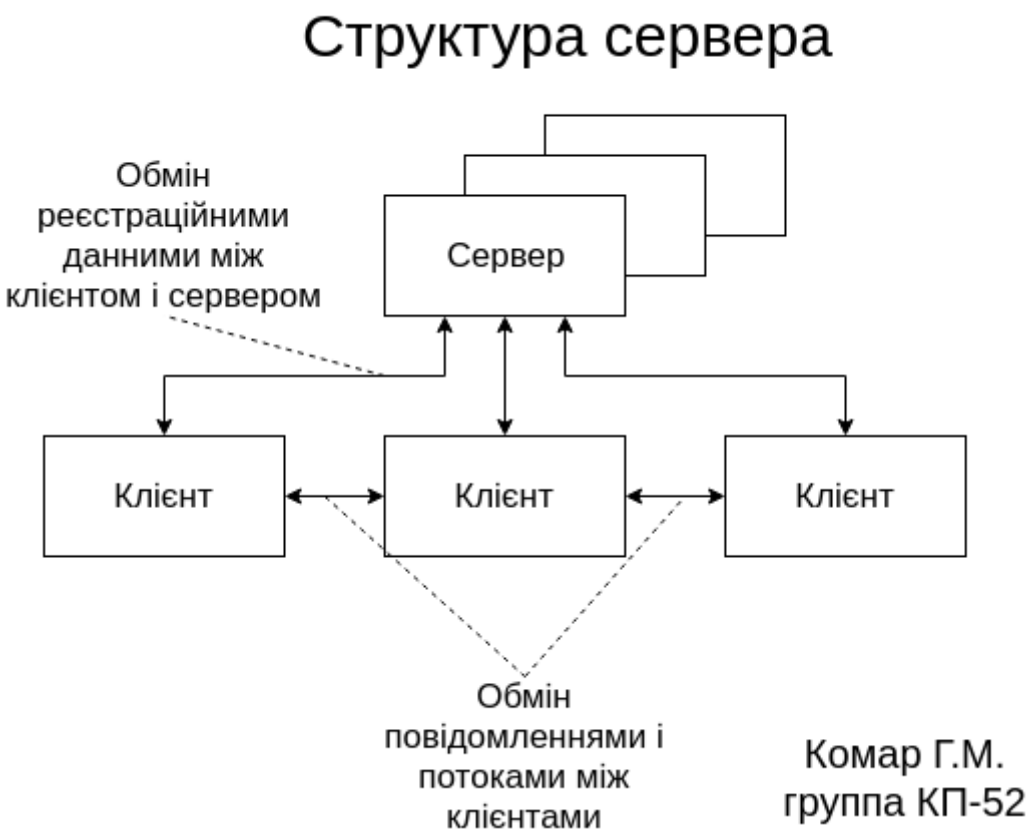
_____ Г.М.Комар

2018

ЗМІСТ

1. Схема структури розподіленого сервісу

Принципова схема структури розподіленого сервісу представлена нижче:



2. Опис вмісту форм клієнта

Для того щоб наш клієнт міг спілкуватися з сервером потрібно завантажити с файлу налаштування, IP сервера та порт сервера.

```
try {
var sr = new StreamReader(@"Client_info/data_info.txt"); string buffer = sr.ReadToEnd();
sr.Close();

string[] conect_info = buffer.Split(':'); ip = IPAddress.Parse(conect_info[0]); port = int.Parse(conect_info[1]);

label4.ForeColor = Color.Green;

label4.Text = "Налаштуваня: \n IP сервера:" + conect_info[0] + "\n Порт сервера" + conect_info[1];
}
```

Якщо в нас немає даного файлу ми запропонуємо користувачу самостійно вести налаштування. Для цього нам потрібна нова форма Проект-Добавить форму Windows (див. рис. 4.8).

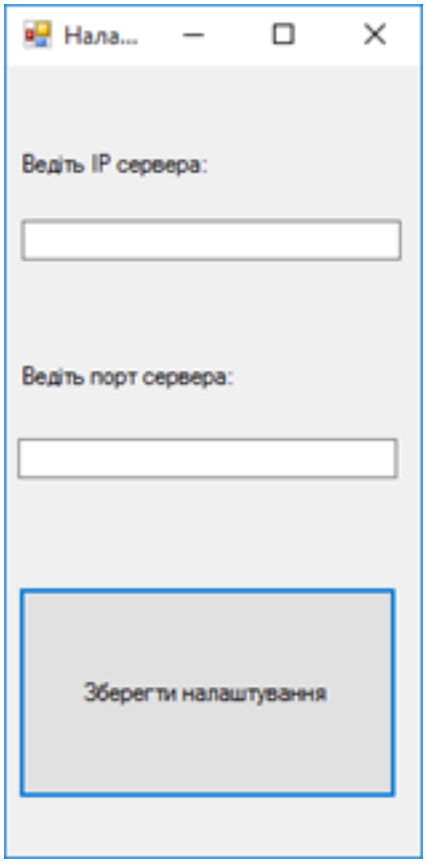


Рис. 4.8.1. - Форма для налаштування з’єднання

```
private void button1_Click(object sender, EventArgs e)
{

if (textBox1.Text != "" && textBox1.Text != "" && textBox1.Text != "" && textBox1.Text != "")
{ try
{

DirectoryInfo data = new DirectoryInfo("Client_info"); data.Create();
var sw = new StreamWriter(@"Client_info/data_info.txt");
sw.WriteLine(textBox1.Text + ":" + textBox2.Text);
sw.Close();
this.Hide(); Application.Restart(); }
catch (Exception ex) {
MessageBox.Show("EROR:" + ex.Message); }
```

```
} }
```

Підключення Форми Налаштування до Головної форми.

```
catch (Exception ex) {
label4.ForeColor = Color.Red; label4.Text = "Налаштування не знайдені";
Form2 form = new Form2();
form.Show();
}
private void налаштуванняToolStripMenuItem_Click(object sender,
EventArgs e)
{

Form2 form = new Form2(); form.Show();
}
```

Для того щоб підключитись до серверу нам потрібно вести логін та натиснути на кнопку Вхід.

```
private void button2_Click(object sender, EventArgs e)
{

if(textBox1.Text!="" && textBox1.Text !="") {
button1.Enabled = true; richTextBox2.Enabled = true;
Client = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
if (ip!=null) {
Client.Connect(ip, port);

th = new Thread(delegate() { RecvMessage(); }); th.Start();
} }
```

Для відправки повідомлення на сервер використовуємо функцію SendMessage.

```
void SendMessage(string message)

{
if (message != "" && message != "") {
byte[] buffer = new byte[1024];

buffer = Encoding.UTF8.GetBytes(message); Client.Send(buffer);
}
}
```

Для отримання повідомлень від сервера використовуємо функцію RecvMessage.

```
void RecvMessage() //сообщения от сервера {
byte[]buffer=new byte[1024];

for (int i = 0; i < buffer.Length; i++) {
buffer[i] = 0; }
for (; ; ) {
try {
Client.Receive(buffer);

string message =Encoding.UTF8.GetString(buffer); int count =
message.IndexOf(";;;5");
if (count == -1) {
continue; }
string Clear_Message = "";

for (int i = 0; i < count; i++) {
```

```
Clear_Message += message[i]; }
for (int i = 0; i < buffer.Length; i++) {
buffer[i] = 0;
}

this.Invoke((MethodInvoker)delegate() {
richTextBox1.AppendText(Clear_Message);
});
}
catch (Exception ex) { } }
}
```

На рис.3.9 зображено форму для здійснення спілкування в чаті зі сторони клієнта. Вона має декілька полів. Поле ведення логіну має формат TexBox та призначено для ведення свого логіну для входу до чату.

Поле відображення тексту спілкування має формат RichTexBox та забезпечує відображення повідомлень всіх користувачів чату які пройшли реєстрацію.

Поле для введення тексту повідомлення має формат RichTexBox та призначено для створення тексту повідомлення. Для відправки повідомлення використовується кнопка Відправити.

Також на формі є поля Меню для налаштування чату та виходу с нього. Також в меню є інформативна складова про автора, який створив програмне забезпечення

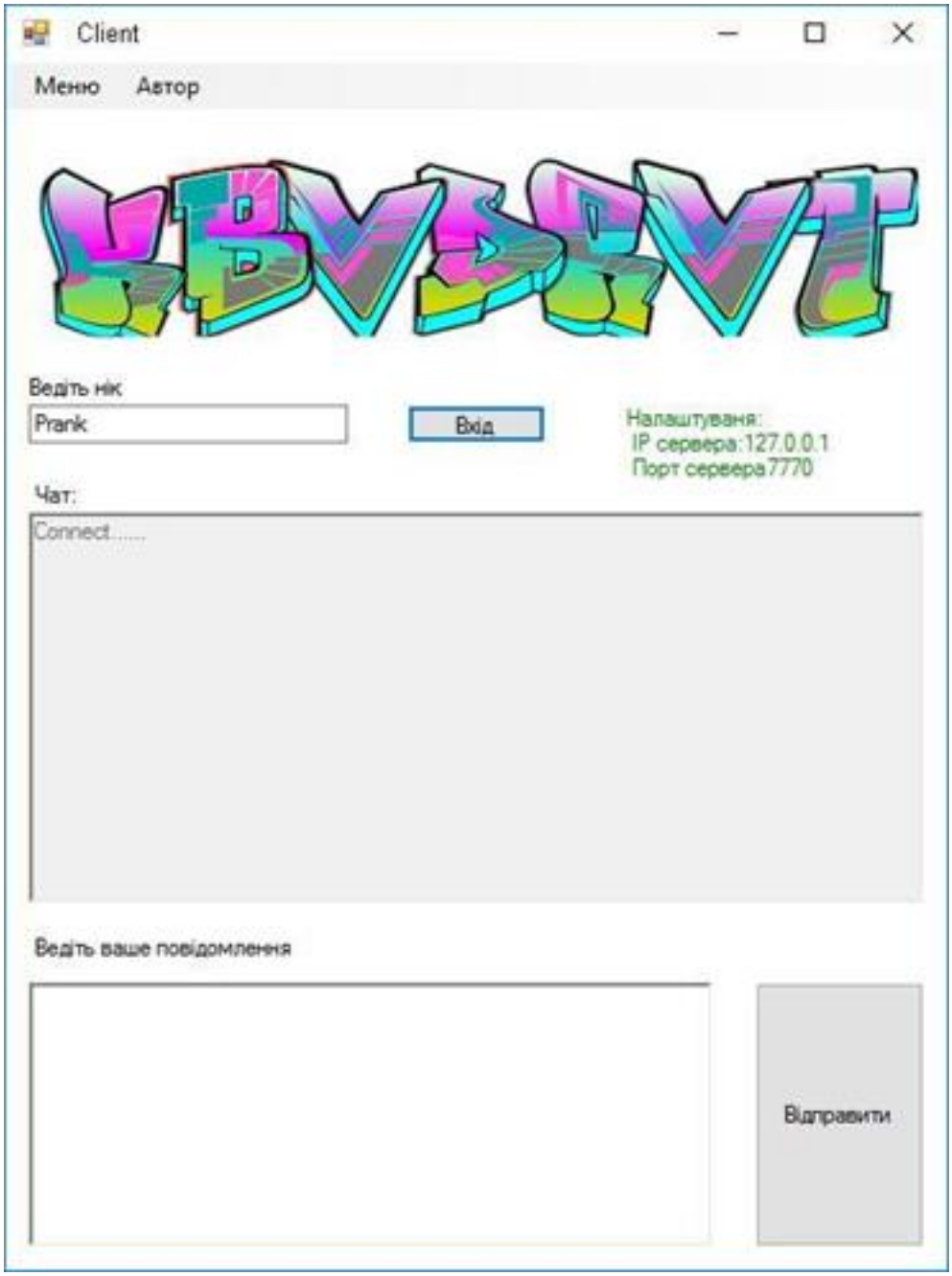


Рис.4.9 - Форма клієнт чату