транзакцій, які здійснюють низку операцій зчитування й записування у межах однієї атомарної операції.

Програмне забезпечення проміжного рівня дозволяє поєднувати масштабованість і відкритість мережних операційних систем із прозорістю і простотою використання розподілених систем. Таке поєднання дозволяє створювати розподілену інформаційну систему із загальною структурою, показаною на рис. 2.24.

2.5. Поняття розподіленого середовища

Розподілене середовище – віртуальний обчислювальний простір, який може обмежуватися однією розподіленою системою або містити кілька розподілених систем, які взаємодіють між собою. Такий віртуальний обчислювальний простір надається користувачеві у вигляді систематизованого сховища інформаційних та програмних ресурсів, має певну структуру, зрозумілу систему адресації ресурсів та певні моделі обчислювальних процесів або бізнес-процесів цього користувача, які є проблемно-орієнтованими, відповідають певним видам робіт. Користувач звертається до бізнес-процесів, які, у свою чергу, отримують необхідні ресурси для своєї роботи.

Учасниками взаємодії в розподіленому середовищі є окремі сутності, якими можуть бути користувачі, прикладні програми та інші обчислювальні ресурси. Як основу опису взаємодії двох сутностей розглянемо загальну модель взаємодії *клієнт – сервер*, у якій одна зі сторін (клієнт) ініціює обмін даними, надсилаючи запит другій стороні (серверу). Сервер обробляє запит й у разі потреби надсилає відповідь клієнтові (рис. 2.25).

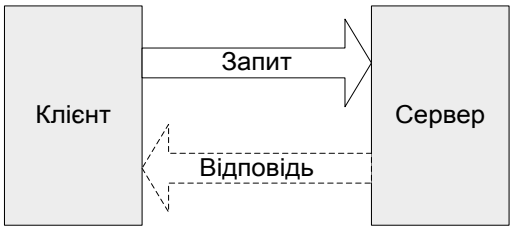

Рис. 2.25. Модель взаємодії *клієнт – сервер*

У базовій моделі *клієнт – сервер* усі процеси в розподілених системах поділяють на дві групи: процеси, які реалізують певну службу, наприклад

службу файлової системи або бази даних, називають **серверами** (servers); процеси, які запитують служби в серверів за рахунок надсилання запиту й очікування відповіді від сервера, називають **клієнтами** (clients). Взаємодія клієнтів і сервера відома також як режим роботи або протокол запит – відповідь (request-reply behavior), зображена на рис. 2.26.

Якщо базова мережа так само надійна, як локальні мережі, то взаємодія між клієнтом і сервером може бути реалізована за допомогою простого протоколу, який не потребує встановлення з'єднання. У цьому разі клієнт, запитуючи службу, перетворює свій запит у форму повідомлення, зазначаючи в ньому службу, якою він бажає скористатися, і необхідні для цього вихідні дані. Потім повідомлення надсилається серверу, який постійно очікує вхідного повідомлення, а отримавши його, обробляє, упаковує результат обробки у відповідне повідомлення й відправляє його клієнтові.

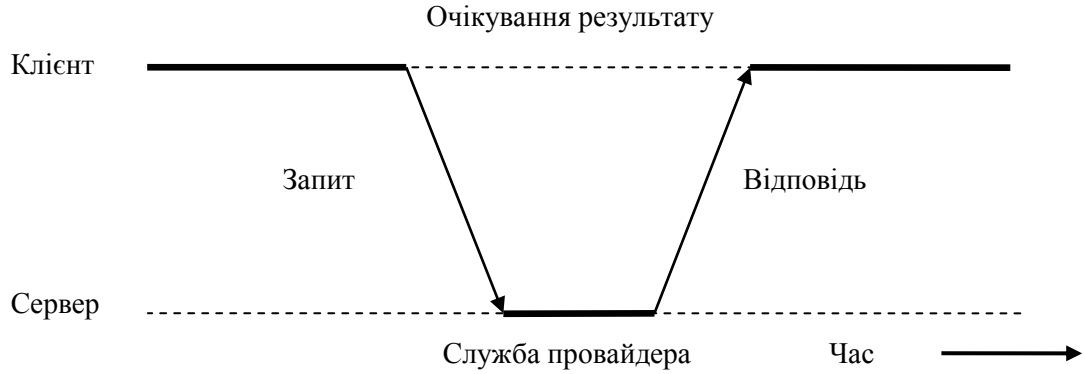

Рис. 2.26. Узагальнена взаємодія між клієнтом і сервером

Використання з'єднання, яке не потребує протоколу, дає істотний виграш в ефективності. До того часу, коли повідомлення не почнуть зникати або ушкоджуватися, можна цілком успішно застосовувати протокол запит – відповідь. На жаль, створити протокол, стійкий до випадкових перебоїв зв'язку, – нетривіальне завдання. Єдине, що можна зробити, – це надати клієнтові можливість повторно надіслати запит, на який не було отримано відповіді. Проблема полягає в тому, що клієнт не може визначити: чи дійсно первинне повідомлення із запитом було загублено чи помилка відбулася під час пере-

дачі відповіді. Якщо втрачено відповідь, то повторне надсилання запиту може призвести до повторного виконання операції.

Взаємодія у межах моделі клієнт – сервер може бути як синхронною, коли клієнт очікує завершення обробки свого запиту сервером, так і асинхронною, коли клієнт надсилає серверу запит і продовжує виконання своєї роботи, не очікуючи відповіді від сервера. Модель клієнта й сервера можна використовувати як основу опису різних способів взаємодії. У цьому контексті важливою є взаємодія складових частин програмного забезпечення, які утворюють розподілену систему.

2.5.1. Розподіл прикладних програм за рівнями

Модель клієнт – сервер потребує розуміння питання, як розподілити програмне забезпечення між клієнтом і сервером. Наприклад, сервер розподіленої бази даних може постійно виконувати роль клієнта, що передає запити на різні файлові сервери, які відповідають за реалізацію таблиць цієї бази даних. У цьому разі сервер баз даних не робить нічого, крім обробки запитів.

Прикладні програми, які створені за технологією клієнт – сервер та призначені для організації доступу користувачів до баз даних, переважно поділяють на три рівні (рис. 2.27): рівень інтерфейсу користувача; рівень обробки; рівень даних.

Рис. 2.27. Логічні рівні прикадної програми

Рівень інтерфейсу користувача містить усе необхідне для безпосереднього спілкування з користувачем, зокрема функції керування дисплеєм, рівень обробки зазвичай містить прикладні програми, а рівень даних – дані, з якими виконується робота.

Рівень інтерфейсу користувача. Рівень інтерфейсу користувача, який зазвичай реалізується на клієнтах, містить програми, за допомогою яких користувач може взаємодіяти з прикладною програмою. Складність програм, що входять до інтерфейсу користувача, досить різна.

Найпростіший варіант програми інтерфейсу користувача, який не містить нічого, крім символьного (не графічного) дисплея, зазвичай використовується у процесі роботи з мейнфреймами. У тому разі, коли мейнфрейм контролює всі процеси взаємодії, включаючи роботу з клавіатурою й монітором, навряд чи можна говорити про використання моделі клієнт – сервер. Однак здебільшого термінали користувачів виконують деяку локальну обробку, здійснюючи, наприклад, еходрук рядків, що вводяться, або надаючи інтерфейс у вигляді форм, у якому можна відредагувати введені дані до їх пересилання на головний комп’ютер. Нині навіть у середовищі мейнфреймів наявні більш досконалі інтерфейси користувачів.

Сучасні інтерфейси користувачів більш функціональні, оскільки вони підтримують спільну роботу прикладних програм через єдине графічне вікно й під час дій користувача забезпечують обмін даними через це вікно. Наприклад, для видалення файлу часто достатньо перенести значок, який відповідає цьому файлу, на значок сміттєвого кошика. Аналогічним способом багато текстових процесорів дозволяють користувачеві переміщувати текст документа в інше місце, користуючись тільки мишею.

Рівень обробки. Багато прикладних програм у моделі клієнт – сервер побудовані з трьох різних частин: частини, яка взаємодіє з користувачем; частини, яка відповідає за роботу з базою даних або файловою системою; і середньої частини, що реалізує основну функціональність прикладної програми та перебуває на рівні обробки. На відміну від інтерфейсів користувача або баз даних, на рівні обробки важко визначити загальні закономірності.

Рівень даних. Рівень даних у моделі клієнт - сервер містить програми, які надають дані прикладним програмам, що їх оброблюють. Специфічною властивістю цього рівня є вимога живучості, тобто коли прикладна програма не працює, дані мають зберігатися в певному місці, оскільки є необхідним їх подальше використання. У найпростішому варіанті рівень даних реалізується файловою системою, але частіше повномасштабною базою даних. У моделі клієнт - сервер рівень даних зазвичай розташований на стороні сервера.

Крім простого зберігання інформації, рівень даних відповідає за підтримку цілісності даних для різних прикладних програм, тобто для бази даних це означає, що метадані (опис таблиць, обмеження і специфічні метадані прикладного програмного забезпечення) зберігаються саме на цьому рівні. Наприклад, якщо у прикладній програмі, яка входить до банківської системи, необхідно сформувати повідомлення щодо боргу клієнта за кредитною карткою, то це може бути зроблено за допомогою тригера бази даних, який у потрібний момент активізує процедуру, яка дозволяє виконати таку дію.

Найчастіше рівень даних формується у вигляді реляційної бази даних. У такому разі ключовою вимогою є незалежність даних, які організуються незалежно від прикладної програми так, щоб зміни в організації даних не впливали на прикладну програму, а прикладна програма не впливала на організацію даних. Використання реляційних баз даних у моделі клієнт - сервер допомагає відокремити рівень обробки від рівня даних, розглядаючи обробку й дані незалежно один від одного.

Однак найбільш поширеними є такі прикладні програми, для яких реляційні бази даних не є найкращим вибором. Характерною рисою таких прикладних програм є робота зі складними типами даних, які простіше моделювати в поняттях об’єктів, а не відношень. До таких типів даних належать дані, які описують об’єкти різної складності: від простих наборів прямокутників і кіл до проекту літака в системах автоматизованого проектування. Мультимедійним системам також значно простіше працювати з відео- та аудіопотоками, використовуючи специфічні для них операції, ніж з моделями цих потоків у вигляді реляційних таблиць.

У тих випадках, коли операції з даними значно простіше виразити в поняттях роботи з об’єктами, має сенс реалізувати рівень даних засобами об’єктно-орієнтованих баз даних, які не тільки підтримують організацію складних даних у формі об’єктів, але і зберігають реалізації операцій над цими об’єктами. Таким чином, частина функціональності, що перебувала на рівні обробки, мігрує на рівень даних.

2.5.2. Варіанти архітектури клієнт - сервер

На практиці різних користувачів розподіленої системи цікавить доступ як до одних і тих самих даних, так і до різних наборів даних. Найбільш простим рознесенням функцій такої системи між декількома комп’ютерами буде поділ логічних рівнів прикладної програми між однією серверною частиною прикладної програми, яка відповідає за доступ до даних, і клієнтськими частинами, що перебувають на декількох комп’ютерах та реалізують інтерфейс користувача. Програмне забезпечення, яке реалізує логіку прикладної програми, може бути віднесене до сервера, клієнтів або розділене між ними (рис. 2.28).

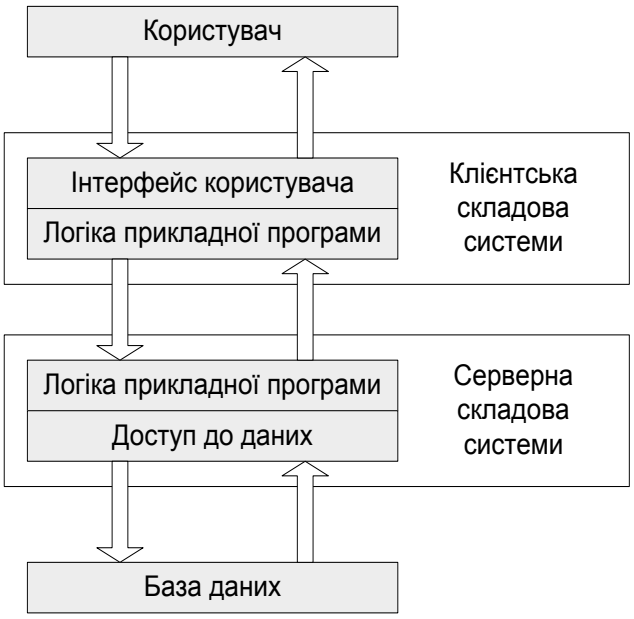

Рис. 2.28. Дволанкова архітектура

Архітектуру прикладного програмного забезпечення, побудовану за таким принципом, називають клієнт серверною або дволанковою. На практиці подібні системи не завжди відносять до класу розподілених, але формально їх можна вважати найпростішою реалізацією розподілених систем.

Наступним етапом розвитку архітектури клієнт - сервер є триланкова архітектура, у якій інтерфейс користувача, логіка прикладної програми й доступ до даних відокремлено в самостійні складові системи, які можуть працювати на незалежних комп'ютерах (рис. 2.29).

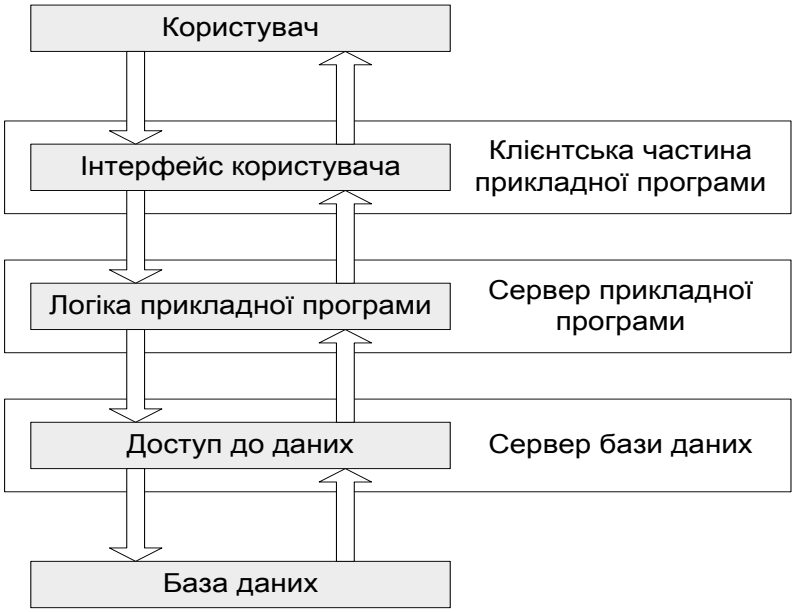

Рис. 2.29. Триланкова архітектура

Запит користувача в таких системах послідовно обробляється клієнтською частиною системи, сервером логіки прикладної програми й сервером баз даних. Однак зазвичай під розподіленою системою розуміють системи з більш складною архітектурою, ніж триланкова.

Розподіленим прикладним програмним забезпеченням, що автоматизує діяльність підприємства або організації, називають системи, логіка прикладних програм яких розподілена між декількома компонентами системи, кожен з яких можна виконувати на окремому комп'ютері.

Приклад. Реалізація логіки прикладної програми системи роздрібної торгівлі має виконувати запити до логіки прикладної програми третіх фірм, зокрема постачальників товарів, систем електронних платежів або банків, що надають споживчі кредити (рис. 2.30).

Таким чином, на практиці під розподіленою системою часто розуміють розширення багатоланкової архітектури, коли запити користувача не проходять послідовно від інтерфейсу користувача до єдиного сервера баз даних.

Приклад. Мережі прямого обміну даними між клієнтами (peer-to-peer networks). Якщо попередній приклад мав «деревоподібну» архітектуру програмного забезпечення, то мережі прямого обміну організовані складніше (рис. 2.31). Подібні системи нині є одними з найбільших серед поширених розподілених систем, що поєднують мільйони комп'ютерів.

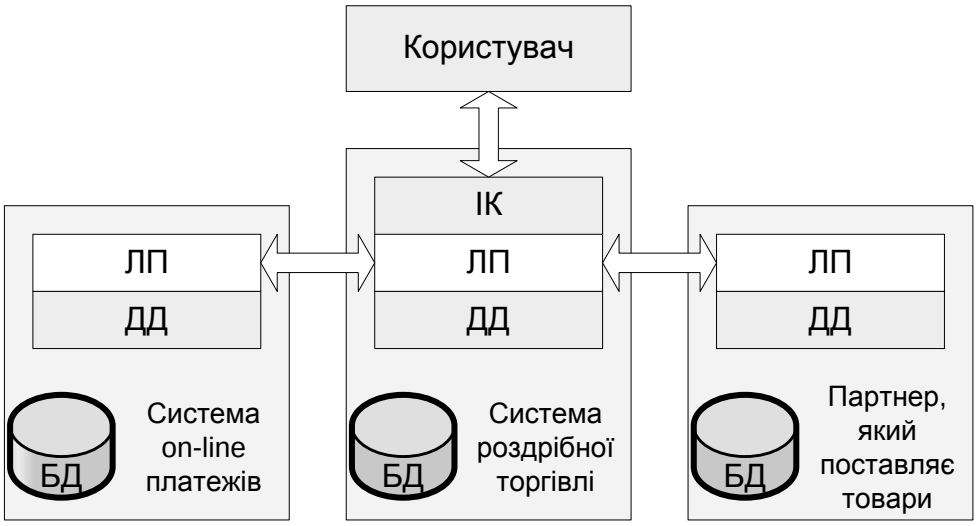

Рис. 2.30. Розподілена система роздрібних продажів

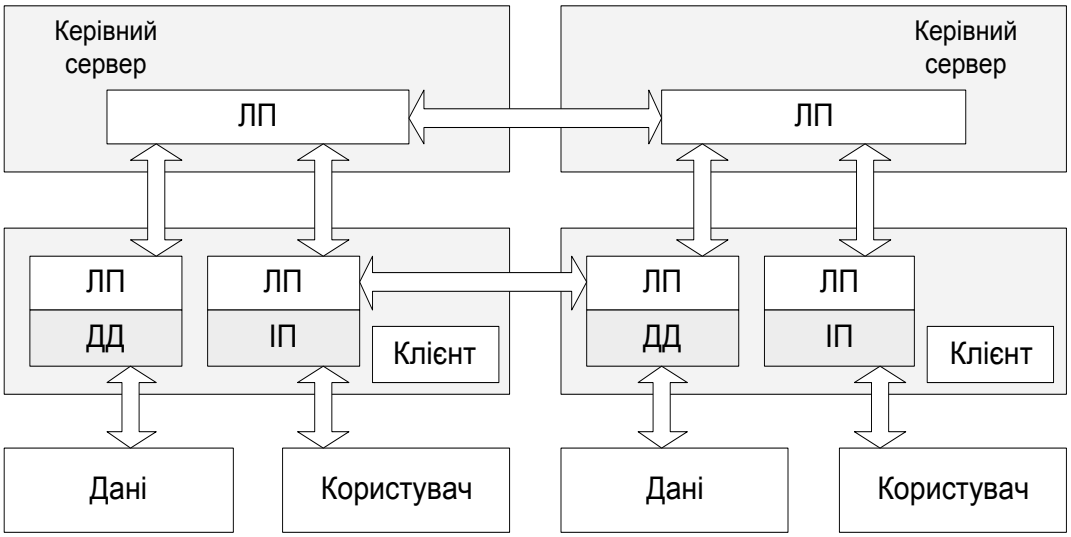

Рис. 2.31. Система прямого обміну даними між клієнтами

Багатоланкові архітектури. Один з підходів до організації взаємодії клієнтів і серверів – це розподіл програм, що перебувають на рівні приклад-

ного програмного забезпечення, який полягає у тому, щоб помістити на клієнтську сторону лише термінальну частину інтерфейсу користувача, як показано на рис. 2.32*а*, дозволивши прикладній програмі віддалено контролювати подання даних. Альтернативою цьому підходу буде передача клієнтові всієї роботи з інтерфейсом користувача (рис. 2.32, *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.

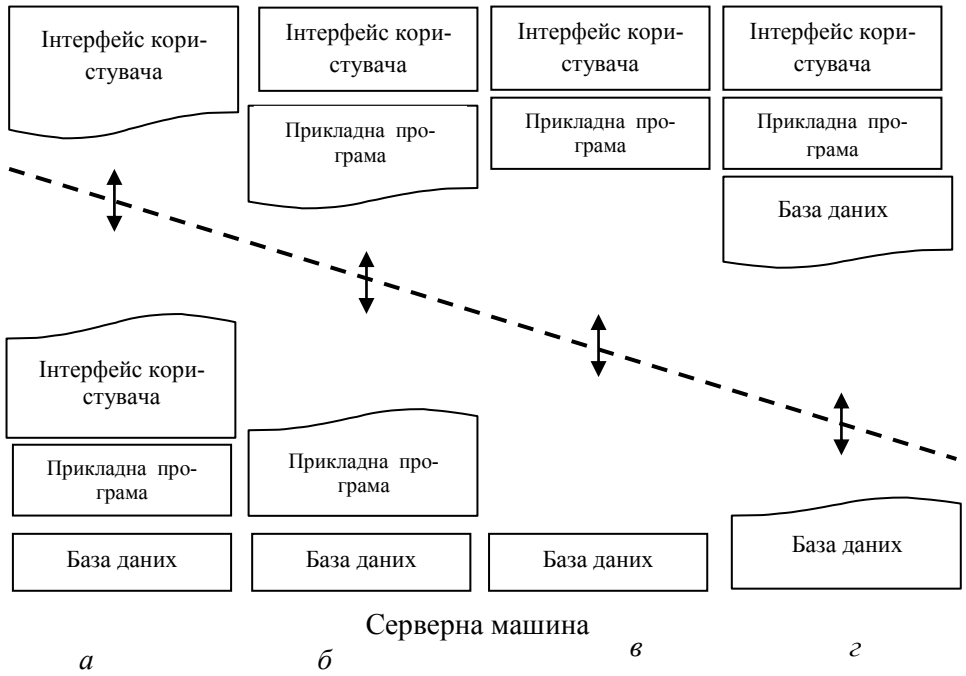

Рис. 2.32. Альтернативні форми організації архітектури клієнт – сервер

У багатьох системах клієнт-сервер поширені типи організації, зображені на рис. 2.32*в* і рис. 2.32*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. 2.32*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 враховувати те, що у сервера іноді виникає потреба працювати як клієнт. У такій ситуації (рис. 2.33) 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.

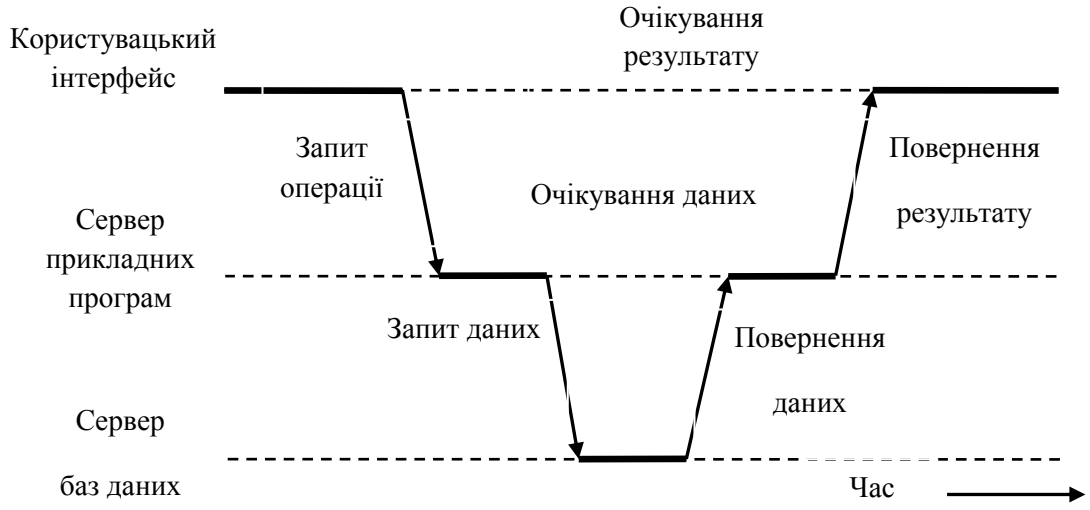

Рис. 2.33. Приклад сервера, що діє як клієнт

Сучасні варіанти архітектури. Багатоланкові архітектури клієнт – сервер є продовженням розподілу прикладних програм на рівні інтерфейсу користувача, компонентів обробки й даних. Різні ланки взаємодіють відповідно до логічної організації прикладних програм. У прикладному програмному забезпеченні розподілена обробка еквівалентна організації багатоланкової архітектури прикладної програми клієнт - сервер. Такий тип розподілу називають **вертикальним** (vertical distribution). Його особливістю є те, що він досягається розміщенням логічно різних компонентів на різних машинах. Це поняття пов’язане з концепцією **вертикальної фрагментації** (vertical fragmentation), яку використовують у розподілених реляційних базах даних, де під цим термі-

ном розуміють розбиттям на стовпці таблиць для їх зберігання на різних машинах.

Вертикальний розподіл – це лише один з можливих способів організації клієнт – серверного прикладного програмного забезпечення. У сучасних архітектурах розподіл на клієнти й сервери відбувається способом, відомим як **горизонтальний розподіл** (horizontal distribution), за якого клієнт або сервер може містити фізично розподілені частини логічно однорідного модуля, причому робота з кожною із частин може виконуватися незалежно для вирівнювання завантаження.

Як найбільш поширений приклад горизонтального розподілу розглянемо web-сервер, реплікований на кілька машин локальної мережі, як показано на рис. 2.34. На кожному із серверів утримується однаковий набір web-сторінок, і щоразу, коли одна з web-сторінок обновляється, її копії негайно розсилаються на всі сервери. Сервер, якому буде переданий вхідний запит, вибирається за правилом «каруселі» (round-robin). Такий варіант горизонтального розподілу досить успішно використовується для вирівнювання навантаження на сервери популярних web-сайтів.

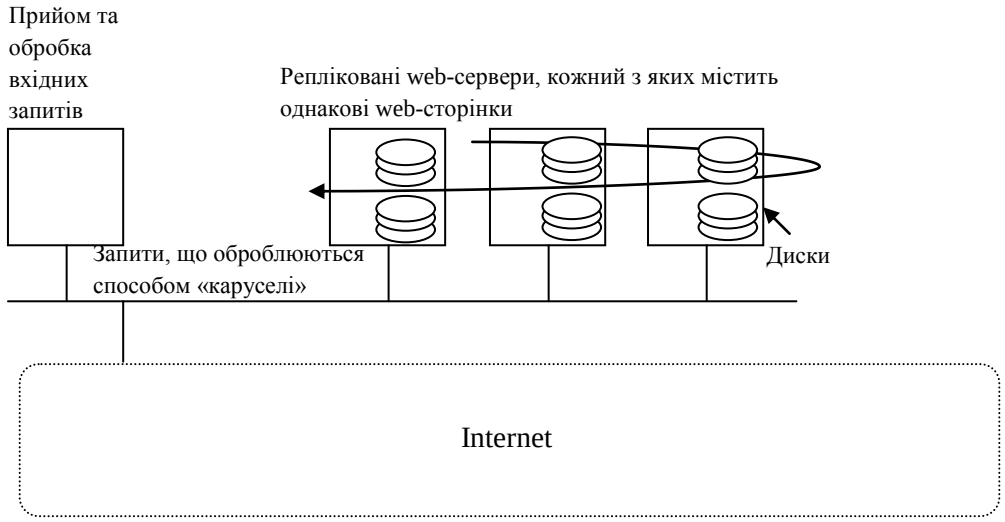

Рис. 2.34. Приклад горизонтального розподілу web-служби

У такий самий спосіб, хоч і менш очевидно, можуть бути розподілені й клієнти. Для нескладного прикладного програмного забезпечення, призначеного для колективної роботи, можна не мати сервера взагалі – це одноранговий розподіл (peer-to-peer distribution), що відбувається, наприклад, якщо

користувач хоче зв’язатися з іншим користувачем і обидва повинні запустити одну прикладну програму, щоб розпочати сеанс. Третій клієнт може також спілкуватися з одним з них або обома, для чого йому потрібно запустити ту саму прикладну програму.

2.5.3. Програмні компоненти розподілених систем

У розподілених системах функції одного рівня прикладної програми можуть бути рознесені між декількома комп’ютерами. Натомість програмне забезпечення, встановлене на одному комп’ютері, може відповідати за виконання функцій, що належать до різних рівнів, тому підхід до визначення розподіленої системи, за яким розподілена система – це сукупність комп’ютерів, є умовним. Щоб мати можливість описувати і реалізовувати розподілені системи, було введено поняття «програмна компонента».

Програмна компонента – це одиниця програмного забезпечення, що виконується на одному комп’ютері в межах одного процесу і надає деякий набір сервісів, які використовуються через її зовнішній інтерфейс іншими компонентами, що виконуються на цьому ж комп’ютері та на віддалених комп’ютерах (рис. 2.35). Певні компоненти інтерфейсу користувача надають свій сервіс кінцевому користувачеві.

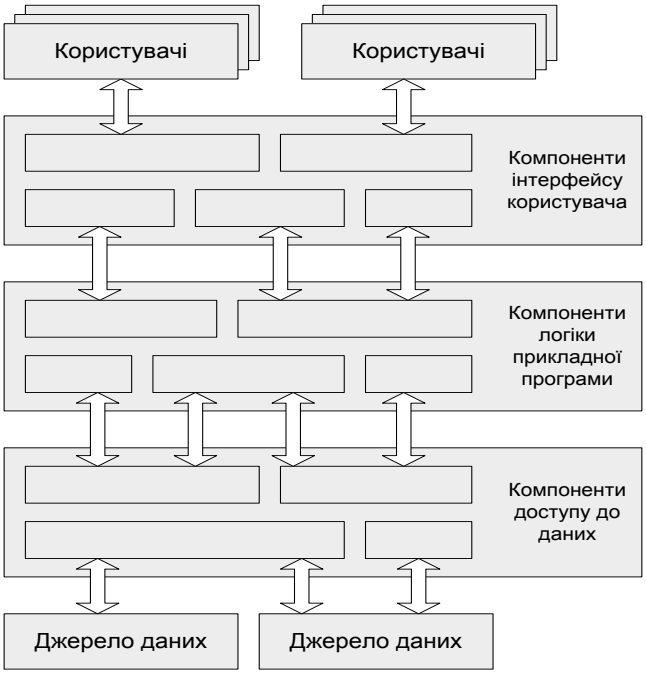

Рис. 2.35. Компоненти розподіленої системи

Виходячи з визначення програмної компоненти, можна дати більш точне визначення розподіленої системи, відповідно до якого **розподілена система** – це набір програмних компонент, які взаємодіють між собою і виконуються на одному або декількох зв'язаних комп'ютерах, становлять з погляду користувача системи єдине ціле. У такому разі прозорість є атрибутом розподіленої системи. Якщо система функціонує коректно, то від кінцевого користувача має бути приховано, де і як виконуються його запити.

Програмна компонента є мінімальною складовою розподіленої системи, що дозволяє у процесі модернізації системи одні компоненти оновлювати незалежно від інших.

У добре спроектованій системі функції кожної компоненти стосуються тільки до одного рівня прикладної програми, однак розподіл на три рівні є недостатнім для класифікації компонент. Наприклад, одна частина компонент інтерфейсу користувача може взаємодіяти з користувачем, а друга – надавати свої сервіси іншим компонентам, але з користувачем не взаємодіяти.

Такі класифікації наявні, однак вони не є загальноприйнятими й часто значною мірою залежать від специфіки конкретних прикладних програм автоматизації діяльності підприємств, організацій, компаній.

Опис програмної компоненти. 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 повідомлень 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 і 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 нижнього рівня, на якому ґрунтується певне проміжне середовище. Роль такого протоколу можуть виконувати протоколи HTTP, TCP, NetBIOS або інший протокол нижнього 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 повідомлення, 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. 2.36).

Рис. 2.36. 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, маршализованого 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.

2.5.4. Основи мережної взаємодії

З точки зору одного з комп'ютерів розподіленої системи, всі інші машини, які належать до неї, є віддаленими обчислювальними системами. Теоретичною основою мережної взаємодії віддалених систем є загальновідома еталонна модель взаємодії відкритих систем (Open Systems Interconnection Reference Model, OSI), яку розроблено Міжнародною організацією стандартизації (International Standards Organization, ISO) та названо OSI/ISO (Open Systems Interface, OSI), що поділяє процес взаємодії двох сторін на сім рівнів: фізичний, канальний, мережний, транспортний, сеансовий, подання, прикладний (рис. 2.37).

У мережах найпоширенішого стека протоколів TCP/IP протокол TCP є протоколом транспортного рівня, а протокол IP – протоколом мережного рівня. Нині забезпечує інтерфейс до транспортного рівня мережна компонента операційної системи, надаючи інтерфейс для верхніх рівнів, який використовує **сокети**, які у свою чергу, забезпечують примітиви низького рівня для безпосереднього обміну потоком байт між двома процесами. Стандартного рівня

представлення або сеансового рівня у стеці протоколів TCP/IP немає, іноді до них відносять захищені протоколи SSL/TLS.

Використання протоколу TCP/IP за допомогою сокетів надає стандартний, міжплатформний, але низькорівневий сервіс для обміну даними між компонентами. Щоб задовольнити вимоги до розподілених систем функції сеансового рівня й рівня представлення має виконувати проміжне програмне забезпечення (рис. 2.37), яке гарантуватиме відкритість, масштабованість і стійкість розподілених систем. Щоб досягнути цієї мети, проміжне середовище має надавати сервіси для взаємодії компонент розподіленої системи, зокрема такі:

– забезпечення єдиного й незалежного від операційної системи механізму використання одними програмними компонентами сервісів інших компонент;

– забезпечення безпеки розподіленої системи, тобто аутентифікація й авторизація всіх користувачів сервісів компоненти й захист переданої між компонентами інформації від перекручування й читання третіми сторонами;

– забезпечення цілісності даних або керування транзакціями, розподіленими між віддаленими компонентами системи;

– балансування навантаження на сервери із програмними компонентами;

– виявлення віддалених компонентів.

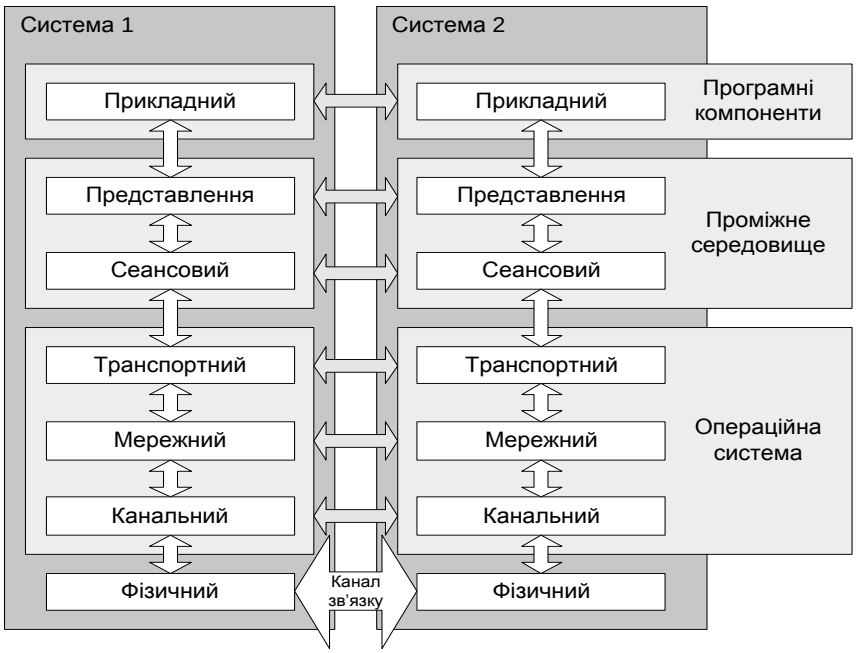

Рис. 2.37. Модель взаємодії обчислювальних систем

У межах однієї розподіленої системи може використовуватися кілька видів проміжних середовищ (рис. 2.38). Якщо підхід до проектування системи правильний, то її кожна розподілена компонента надає свої сервіси, які належать до єдиного проміжного середовища, а також використовує сервіси інших компонент, які також належать до певного проміжного середовища, однак ці середовища можуть бути різними.

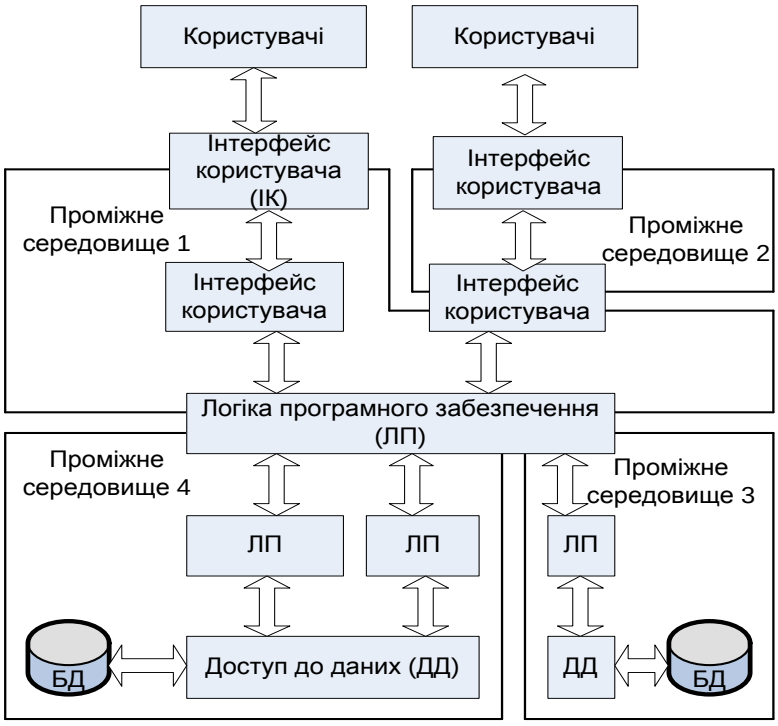

Рис. 2.38. Гетерогенна розподілена система

Розподілену систему, компоненти якої використовують кілька проміжних середовищ, можна називати гетерогенною, на відміну від гомогеної, яка ґрунтується на єдиному проміжному середовищі. Оскільки проміжне середовище може бути реалізовано на різних апаратних платформах і операційних системах, то обидва класи розподілених систем можуть мати комп’ютери, керовані як однією, так і різними операційними системами.

Нині немає проміжного середовища універсального застосування, хоча є певний рух у цьому напрямку. Основною причиною відсутності такого середовища є частково суперечливі вимоги до розподілених систем, а також різний характер мережних з’єднань між компонентами системи: наприклад, взаємодія

компонент усередині одного підприємства, ймовірно, може бути побудована інакше, ніж взаємодія компонент двох різних підприємств, що не повністю довіряють один одному.

Програмні компоненти у межах одного комп'ютера також взаємодіють за допомогою проміжного середовища, але у разі використання декількох проміжних середовищ така взаємодія може бути незручною і неефективною. В ідеальному випадку розподілену компоненту слід реалізувати так, щоб перехід від одного проміжного середовища до іншого відбувався за рахунок змінювання конфігурації програмної компоненти, а не вихідного коду. На практиці таку вимогу реалізувати непросто, однак необхідно мінімізувати можливі корегування програмного коду в разі зміни проміжного середовища.

2.6. Взаємодія компонент розподіленої системи

2.6.1. Концепції взаємодії компонент розподіленої системи

Нині розрізняють дві концепції взаємодії програмних компонент: обмін повідомленнями між компонентами й виклик процедур або методів об'єкта віддаленої компоненти за аналогією до локального виклику процедури. Оскільки будь-яка взаємодія між віддаленими компонентами ґрунтується на сокетах TCP/IP, базовим з погляду проміжного середовища є низькорівневий обмін повідомленнями на основі мережних сокетів, сервіс яких ніяк не визначає формату переданого повідомлення. На основі протоколів TCP або HTTP потім можна побудувати прикладні протоколи обміну повідомленнями більш високого рівня абстрагування, щоб реалізувати складніший обмін повідомленнями або віддалений виклик процедур.

Віддалений виклик є моделлю, що походить від мов програмування високого рівня, а не від реалізації інтерфейсу транспортного рівня мережних протоколів, тому протоколи віддаленого виклику мають обов'язково ґрунтуватися на будь-якій системі передачі повідомлень, включаючи як безпосереднє використання сокетів TCP/IP, так інші проміжні середовища для обміну повідомленнями. Під час реалізації високорівневих служб обміну повідомленнями, у свою чергу, можуть використовуватися віддалений виклик процедур, в основі яко-