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-

го лежить більш низькорівнева передача повідомлень, що використовує, наприклад, безпосередньо мережні сокети. Таким чином, одне проміжне середовище може використовувати для свого функціонування сервіси іншого, аналогічно до того, як один протокол транспортного або мережного рівня може працювати поверх іншого у разі тунелювання протоколів.

2.6.2. 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. 2.39).

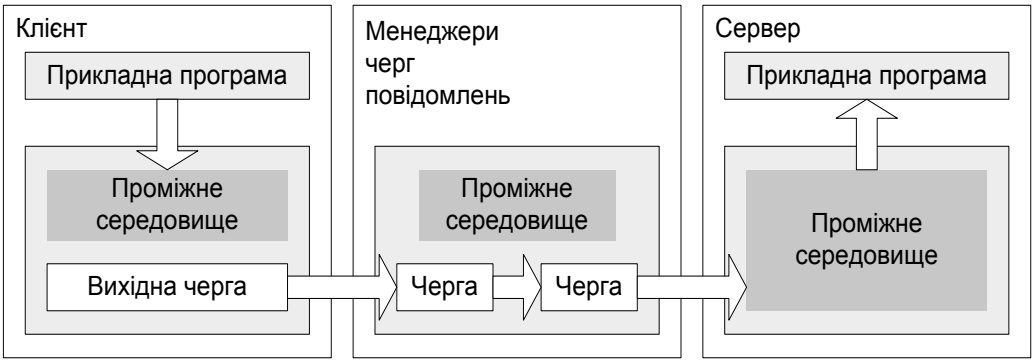

Рис. 2.39. Системи черг повідомлень

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.

2.6.3. Віддалений виклик процедур

Ідея **віддаленого виклику процедур** (Remote Procedure Call, RPC) з'явилася в середині 80-х років і полягала в тому, що за допомогою проміжного програмного забезпечення функцію на віддаленому комп'ютері можна викликати так само, як і функцію на локальному комп'ютері. 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. 2.40). Аналогічно на клієнтську машину надсилається результат виконання функції, викликаної з сервера.

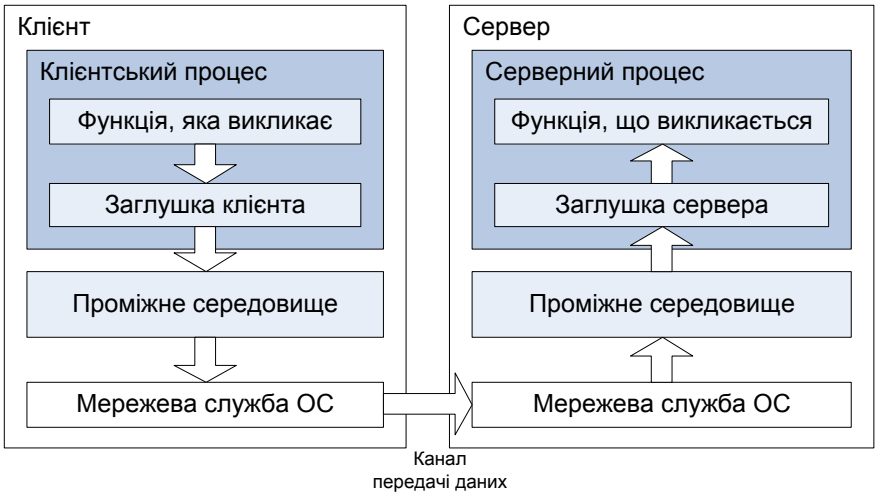

Рис. 2.40. Віддалений виклик процедур

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.

Однонаправлений 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 називаються **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 сериалізації підлягають об’єкти, на які посилаються, до самих вказівників сериалізація не може бути застосована, оскільки це ускладнює використання механізму віддаленого виклику в мовах, які підтримують вказівники на об’єкти невідомого типу.

2.6.4. 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 являє собою 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. 2.41). 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.

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. Уразі 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.

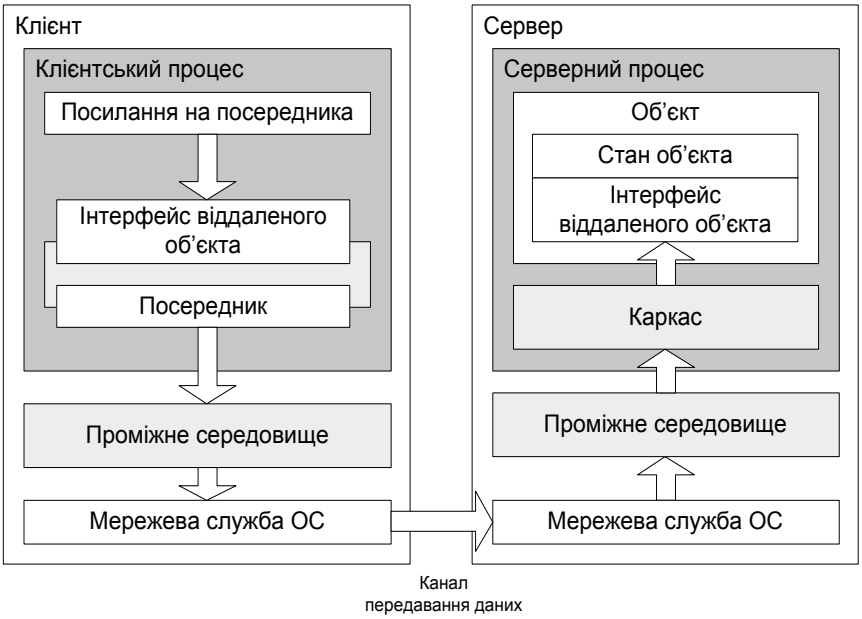

Рис. 2.41. Використання віддалених об'єктів

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, параметр або результат методів можуть теж передаватися посилання на віддалений об'єкт (рис. 2.42), якщо віддалений метод – клієнт, який виконує виклик, також є сервером RMI.

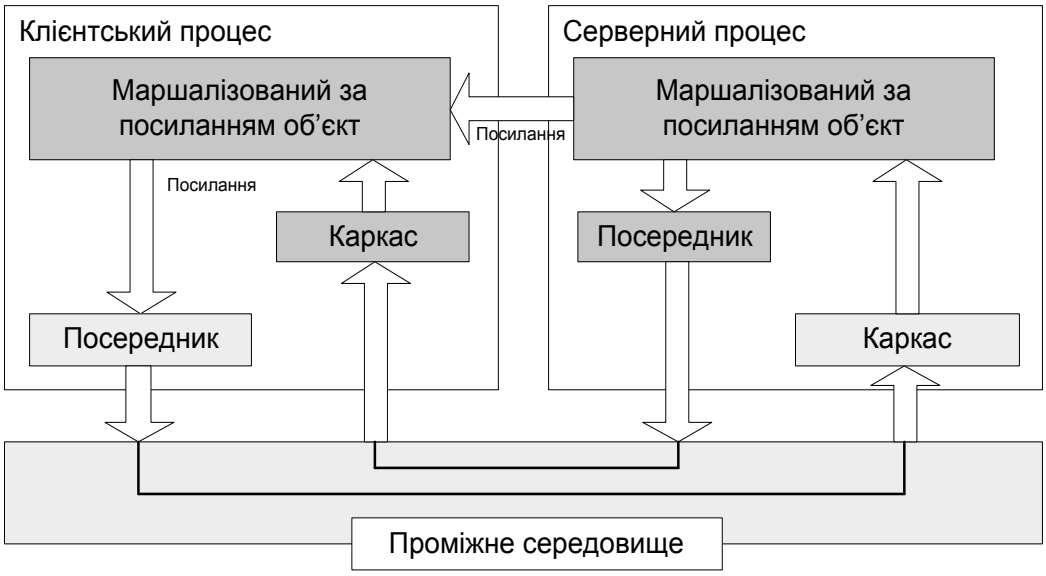

Рис. 2.42. Передача віддаленому методу посилання на об'єкт, що маршалізується за посиланням

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. 2.43).

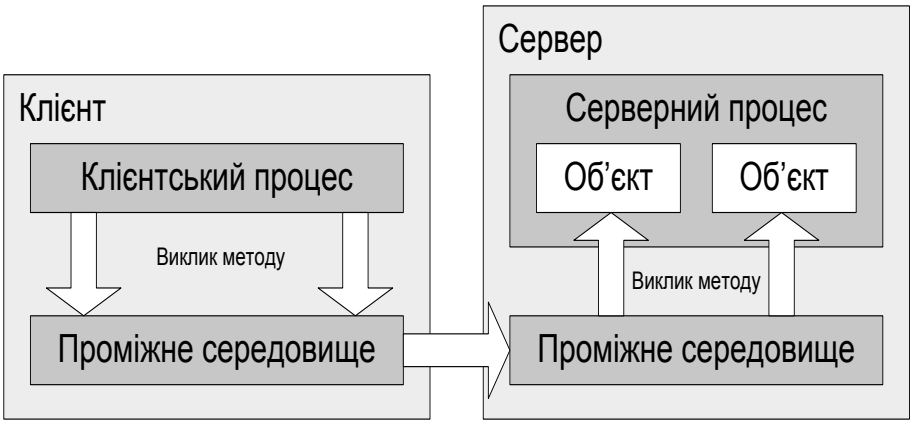

Рис. 2.43. Режим єдиного виклику віддаленого методу

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. 2.44).

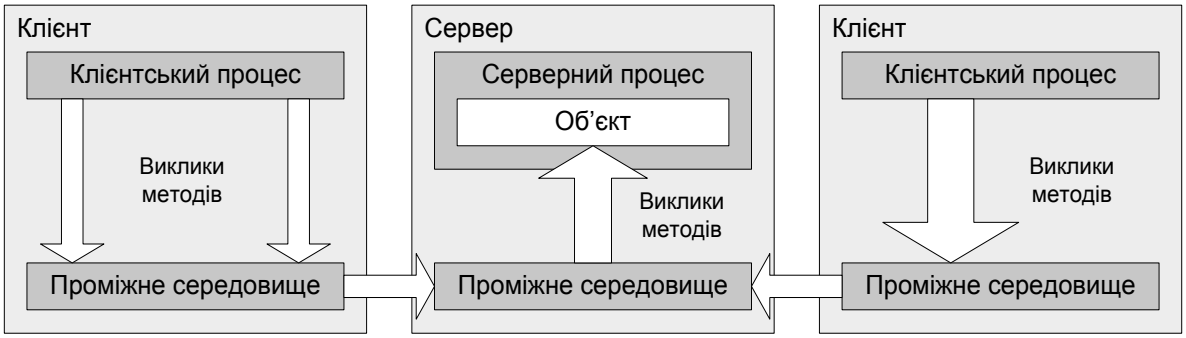

Рис. 2.44. Використання віддалених об’єктів у режимі єдиного екземпляра

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. 2.45).

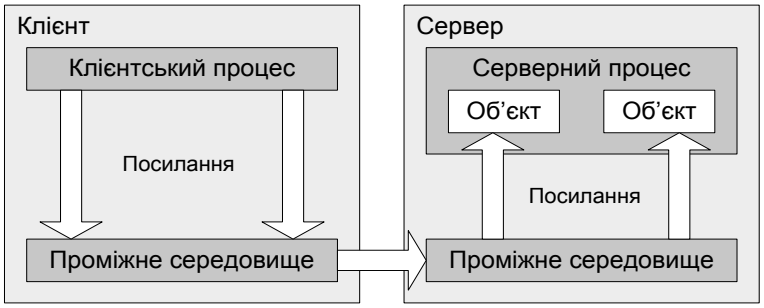

Рис. 2.45. Об'єкти, що активуються клієнтом

Стан компоненти розподіленої системи. Програмні компоненти з погляду користувачів сервісів можна поділити на дві категорії: компоненти без внутрішнього стану**, що зберігається між віддаленими викликами своїх методів (***stateless components***);** компоненти із внутрішнім станом**, що зберігається між віддаленими викликами своїх методів (***statefull components***).**

Під станом у цьому разі розуміють сукупність значень полів об'єктів, які реалізують компоненти, що зберігаються в пам'яті сервера. Якщо компонента у процесі своєї роботи зберігає які-небудь дані в зовнішньому сховищі, наприклад у базі даних або у черзі повідомлень, це зазвичай не розглядається як її внутрішній стан.

Модель єдиного виклику не зберігає стану віддаленого об'єкта між викликами його методів, тому таку модель можна використовувати лише з розподіленими компонентами без внутрішнього стану. Модель одного екземпляра може бути використана для виклику компонентів із внутрішнім станом, але це нераціонально, оскільки її стан буде змінюватися кожним з клієнтів у довільному порядку. Модель активації за запитом клієнта може застосовуватися з будь-якими компонентами, але для компонент без внутрішнього стану такий підхід зазвичай зумовлює непродуктивне використання пам'яті за деякого виграшу у витратах часу процесора порівнянно з моделлю одного виклику.

Компоненти без збереження внутрішнього стану, які використовують разом з моделлю єдиного виклику з пулом об'єктів, мають найбільші можливості масштабування системи за оптимального балансу між витратами пам'яті й навантаженням на процесор.

2.6.5. Розподілені події

У процесі розробки програмного забезпечення часто потрібно отримувати повідомлення про які-небудь події, що виникають асинхронно, тобто в деякі довільні моменти. У розподілених системах також є необхідність використання таких повідомлень, що отримуються від віддаленої системи. Можна визначити два підходи до обробки подій: **тісно пов'язані й слабопов'язані події**.

У разі тісно пов'язаної події виконується пряме повідомлення однієї сторони іншою стороною. Незважаючи на те, що цей метод можна використовувати разом з однонаправленим асинхронним викликом, йому властива низка недоліків, що обмежують його застосування в розподілених системах, зокрема:

– обидві компоненти системи мають виконуватися одночасно;

– для повідомлення декількох компонент про одну подію стороною, що повідомляє, мають використовуватися механізми для ведення списку одержувачів подій;

– утруднена фільтрація або протоколювання подій.

Виходячи з цього, в розподілених системах також застосовуються слабо пов'язні події, коли джерела події (видавці) не взаємодіють прямо з одержувачами подій (передплатниками). Проміжне середовище в цьому разі має надати сервіс, що дозволяє передплатникові замовити будь-яку подію або відмовитися від підписки, а видавцеві – ініціювати подію для розсилання передплатникам (рис. 2.46).

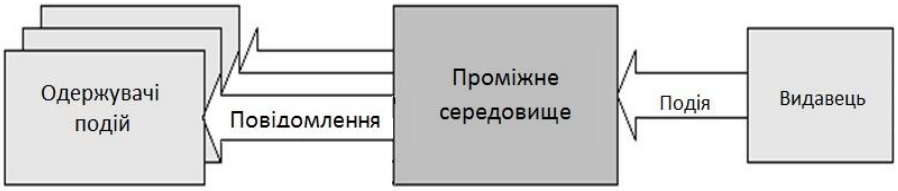

Рис. 2.46. Передплатники й видавці слабопов'язаних подій

У разі використання слабо пов'язних подій передплатники, видавці й менеджер подій можуть розташовуватися на різних комп'ютерах. Сама подія

може бути реалізована як, наприклад, виклик менеджером подій деякого зареєстрованого методу віддаленого об’єкта.

2.6.6. Розподілені транзакції

Транзакція – послідовність операцій з якими-небудь даними, що або успішно виконується повністю, або не виконується взагалі. Якщо неможливо успішно виконати всі дії, то відбувається повернення до початкових значень усіх змінених протягом транзакції даних (відкат транзакції). Транзакція має задовольняти таким вимогам:

– атомарність, тобто транзакція виконується за принципом «все або нічого»;

– погодженість – після успішного завершення або відкату транзакції всі дані перебувають у погодженому стані, їх логічна цілісність не порушена;

– ізоляція – для об’єктів поза транзакцією не видно проміжних станів, яких можуть набувати дані, що змінюються у транзакції; «зовнішні» об’єкти до успішного завершення транзакції повинні мати той самий стан, у якому перебували до її початку;

– сталість – якщо транзакція успішна, то внесені зміни повинні мати постійний характер (тобто збережені в енергонезалежній пам’яті).

Принцип роботи розподіленої транзакції показано на рис. 2.47. Транзакції є основою прикладних програм, які працюють із базами даних, однак у розподіленій системі недостатньо використовувати лише транзакції систем керування базами даних. Наприклад, у розподіленій системі у виконанні транзакції може брати участь кілька розподілених компонент, що працюють із декількома незалежними базами даних (рис. 2.47).

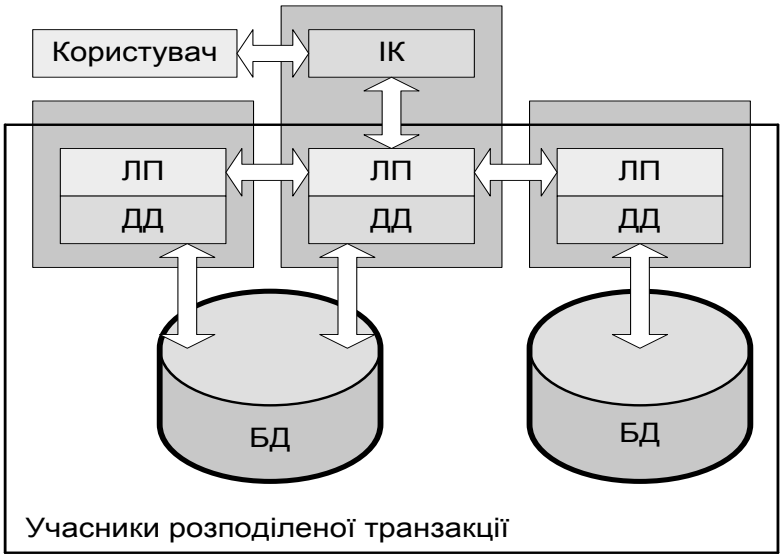

Рис. 2.47. Розподілена транзакція, де ІК – інтерфейс користувача, ЛП- логіка прикладного програмного забезпечення, ДД – доступ до даних, БД - база даних

Розподіленою називають транзакцію, яка охоплює операції декількох компонент розподіленої системи, що взаємодіють між собою, і кожна з яких може працювати з певними СУБД або іншими службами, наприклад використовувати черги повідомлень або навіть працювати з файлами. У разі відкату транзакції всі ці операції має бути скасовано, для чого необхідно виконання таких умов: проміжне середовище має підтримувати керування розподіленими між декількома компонентами транзакціями; компоненти розподіленої системи не можуть працювати з якими-небудь службами або ресурсами, які не беруть участі у транзакції.

Розподілені транзакції є найважливішим елементом підтримування цілісності даних у розподіленій системі, тому для ширшого їх застосування проміжне середовище може містити механізми, які у разі потреби (і певних витрат часу на написання коду) дозволять використовувати в розподілених транзакціях зовнішні служби, що не підтримують транзакції. Такий механізм називають компенсуючим ресурс менеджером (compensating resource manager). Компенсація в цьому разі означає повернення ресурсу до початкового стану під час відкату транзакції.

Одночасно відбувається формування і стандартизація ще одного поняття, пов'язаного з підтримкою цілісності даних у розподілених системах, – господарської діяльності (business activity). Business activity зазвичай є відображенням деякого реального процесу, наприклад купівлі в магазині, процеси якої описують від оформлення замовлення до підтвердження доставки кур'єром. Business activity може охоплювати транзакції, які включають усі процеси у точній відповідності, як вони відбуваються у реальному житті (оформлення замовлення покупця, замовлення товару в постачальника і так далі – до підтвердження покупцем доставки). На відміну від транзакції, час життя якої передбачено коротким, business activity може тривати довго (наприклад, місяць), вона може підтримувати скасування внесених змін (зокрема, оформлення повернення товару постачальнику в разі відмови покупця) за рахунок використання компесуючих завдань.

2.6.7. Безпека в розподілених системах

Для гарантування безпеки розподіленої системи проміжне середовище має забезпечувати функціонування трьох загальновідомих функцій, необхідних для створення безпечних систем:

1. Перевірка того, що користувач сервісів компоненти розподіленої системи дійсно є користувачем, якому дозволено мати доступ до сервісів та даних системи (аутентифікація http://www.intuit.ru/department/se/msfdev/2/footnote.5.1.htm). Така перевірка може бути однобічною, коли тільки сервер переконується в тому, що клієнту дозволено мати доступ, або двобічною, коли клієнт теж переконується в тому, що цей сервер надає саме ті сервіси, які йому потрібні.
2. Обмеження доступу до сервісів компонента залежно від результатів аутентифікації (авторизація). Для вирішення такого завдання проміжне середовище має підтримувати обмеження доступу, в основі якого покладено **ролі** (role based security). Оскільки немає можливості позначити рівні доступу через конкретних користувачів або груп користувачів системи, то слід використовувати деякі абстрактні ролі, які у процесі розгортання компоненти пов'язуються адміністратором системи з обліковими записами користувачів системи.

3. Захист даних, переданих між компонентами системи, від перегляду і змінювання третіми сторонами потребує, щоб передані між компонентами повідомлення візувати електронним підписом і шифрувалися як клієнтом, так і сервером.

Функції безпеки можна забезпечувати транспортним протоколом, який використовується проміжним середовищем, самим середовищем, або ними обома одночасно.

2.7. Опис інтерфейсу програмної компоненти

2.7.1. Мова і схеми XML (Extensible Markup Language)

Мова XML нині набула низки різноманітних застосувань і є основою для великої кількості загальноприйнятих специфікацій, які використовують у розподілених системах (зокрема у мовах XML, XSD, SOAP, WSDL), та основні з яких подано на рис. 2.48.

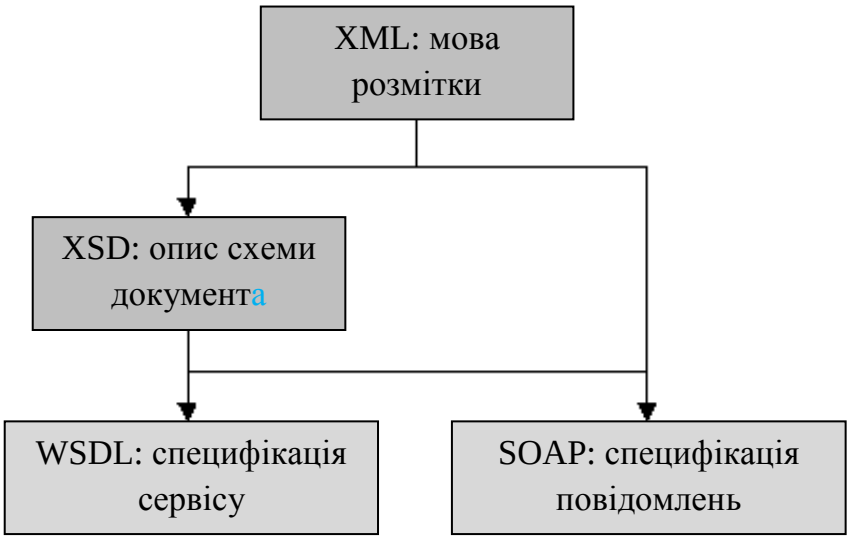

Рис. 2.48. Специфікації, які мають в основі XML-формат

Мова XML – це мова розмітки текстового документа, поданого у вигляді сукупності іменованих, деревоподібних вкладених елементів, кожний з яких може мати деяке текстове значення й набір атрибутів, у яких є ім’я і просте значення (рядок). Мова XML є абстрактною мовою розмітки, яка не визначає ніякого змісту елементів документа. Документи XML добре читає як людина, так і численні програмні аналізатори. У разі природного підходу до