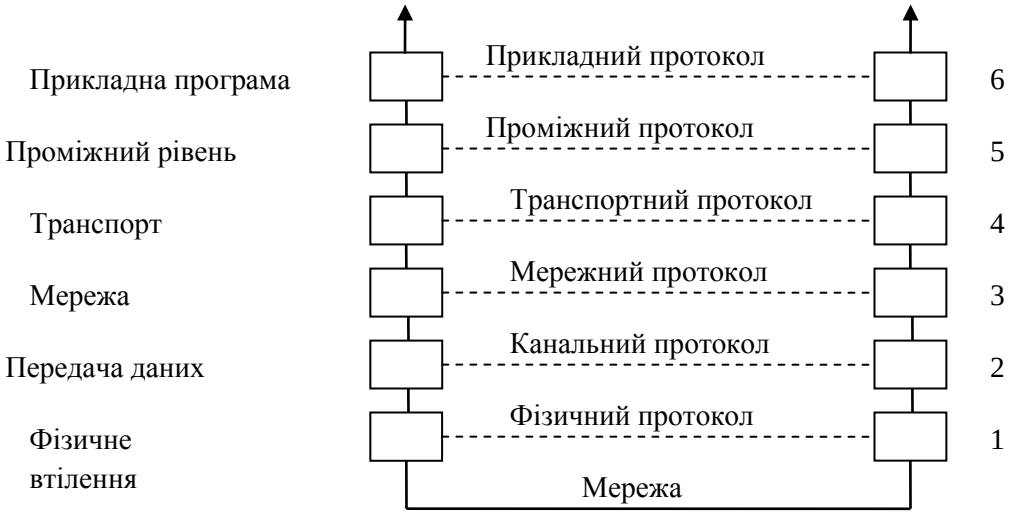

Рис. 3.3. Змінена еталонна модель мережної взаємодії

3.2. Віддалений виклик процедур

Основою більшості розподілених систем є явний обмін повідомленнями між процесами, однак процедури *send* та *receive* не приховують взаємодії, необхідної для забезпечення прозорості доступу. Було запропоновано дозволити програмам викликати процедури, які перебувають на інших машинах. Коли процес, запущений на машині *A*, викликає процедуру з машини *B*, то процес, який викликається на машині *A* припиняється, а виконання викликаної процедури відбувається на машині *B*. Інформація може бути передана від процесу, який викликає, до процедури, яка викликається, через параметри й повернута процесу у вигляді результату виконання процедури. Цей метод називають **віддаленим викликом процедур** (Remote Procedure Call, RPC*).*

Ідея **виклику віддалених процедур** полягає в розширенні добре відомого і зрозумілого механізму підміни процесу керування обчисленнями й даними, які містяться в середині програми, що виконується на одній машині, на процес, який виконується через мережу. Засоби віддаленого виклику процедур призначені для полегшення організації розподілених обчислень і створення розподілених клієнт-серверних інформаційних систем. Найбільша ефективність використання RPC досягається в тих прикладних програмах, у яких наявний інтерактивний зв’язок між віддаленими компонентами з невеликою

тривалістю відповідей і порівняно малою кількістю переданих даних. Такі прикладні програми називають RPC-орієнтованими.

Характерні риси виклику локальних процедур такі: асиметричність, тобто одна зі сторін, які взаємодіють, є ініціатором; синхронність, тобто виконання процедури, яка викликає віддалену процедуру, припиняється з моменту виклику нею запиту й відновлюється лише після повернення результатів запиту з викликаної процедури.

Реалізація віддалених викликів значно складніша від реалізації викликів локальних процедур. Можна назвати проблеми й завдання, які необхідно вирішити у процесі реалізації RPC: оскільки процедура, яка викликає, й процедура, яку викликають, виконуються на різних машинах, то вони мають різні адресні простори, що створює проблеми під час передачі параметрів і результатів, зокрема якщо машини керовані різними операційними системами або мають різну архітектуру (наприклад, використовується прямий або зворотний порядок байтів). Через те, що RPC не може розраховувати на поділювану пам'ять, параметри RPC не мають містити вказівників на чарунки нестекової пам'яті, а значення параметрів – копіюватися з одного комп'ютера на другий. Для копіювання параметрів процедури й результату її виконання через мережу виконується їх серіалізація. Під **серіалізацією** розуміють процес переведення будь-якої структури даних у послідовність бітів. Оберненою до серіалізації є операція **десеріалізації** – відновлення початкового стану структури даних з бітової послідовності. Серіалізацію використовують для передачі об'єктів мережею й для збереження їх у файли.

Приклад. Розглянемо, як використовують серіалізацію для розподіленого програмного забезпечення, різні частини якого мають обмінюватися даними зі складною структурою. У такому разі для типів даних, які передбачається передавати, використовується код, який здійснює серіалізацію й десеріалізацію. Об'єкт заповнюється потрібними даними, потім викликається код серіалізації, за допомогою якого створюється XML-документ. Результат серіалізації передається приймальній стороні, наприклад електронною поштою або HTTP. Програмне забезпечення одержувача створює об'єкт того ж типу й викликає код десеріалізації, у результаті чого одержується об'єкт із тими ж даними, які були в об'єкта програмного забезпечення відправника. За такою схемою працює, зокрема, серіалізація об'єктів з використанням протоколу SOAP у Microsoft .NET.

На відміну від локального, віддалений виклик процедур обов'язково використовує транспортний рівень мережної архітектури (наприклад, TCP), однак цей факт залишається прихованим від розробника програмного забезпечення.

Виконання програми, яка викликає віддалену процедуру, і викликуваної локальної процедури в одній машині реалізується у межах єдиного процесу, але в реалізації RPC беруть участь як мінімум два процеси – по одному в кожній машині. У разі, якщо один з них завершиться аварійно, можливі такі ситуації: у разі аварії процедури, яка викликає віддалені процедури, віддалено викликані процедури втратять процедури верхнього рівня, а у разі аварійного завершення віддалених процедур, процедури верхнього рівня втратять підпорядковані процедури.

Виникає низка проблем, зумовлених неоднорідністю мов програмування й операційних середовищ: структури даних і структури виклику процедур, підтримувані в будь-якій мові програмування, не підтримуються так само в усіх інших мовах, тобто наявна проблема сумісності, яку досі не розв'язано ні за рахунок введення одного загальноприйнятого стандарту, ні за рахунок реалізації декількох конкуруючих стандартів на всіх архітектурах і в усіх мовах.

Приклад. Припустімо, програма має зчитати деякі дані з файлу. Для читання з файлу необхідних даних програміст поміщає в код виклик *read*. У традиційній (однопроцесорній) системі процедура *read* витягається компонувальником з бібліотеки й вставляється в об'єктний код програми. Навіть якщо *read* – це системний виклик, він відпрацьовується аналогічно, через розміщення параметрів у стек. RPC організує свою прозорість у такий же спосіб. Якщо *read* є віддаленою процедурою (тобто буде виконуватися на машині файлового сервера), то у бібліотеці розміщується спеціальна версія *read*, яку називають *клієнтською заглушкою (client stub).* Як і оригінальна функція, вона також викликається відповідно до розглянутої послідовності і викликає локальну операційну систему, але, на відміну від оригінальної функції, клієнтська заглушка не запитує даних в операційної системи, а упаковує параметри в повідомлення й викликом процедури *send* вимагає переслати це повідомлення на сервер, як показано на рис. 3.4. Після виклику процедури *send* клієнтська заглушка викликає процедуру *receive*, блокуючись до отримання відповіді.

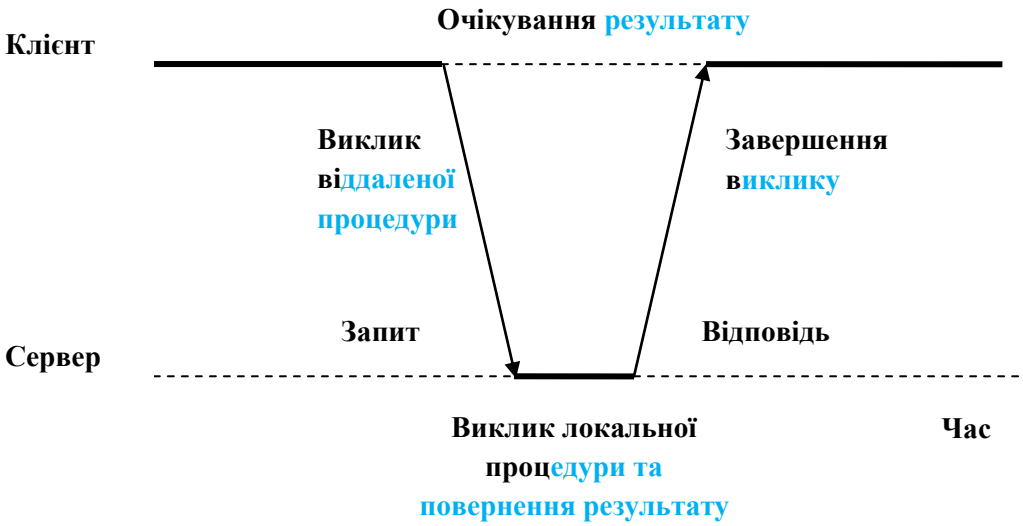

Рис. 3.4. Схема RPC між програмами клієнта й сервера

Коли повідомлення надходить на сервер, операційна система сервера передає його серверній заглушці (*serverstub*), яка є еквівалентною клієнтській, але працює на стороні сервера.

Підведемо підсумки. У разі віддаленого виклику процедур відбуваються такі дії:

1. Процедура клієнта звичайним способом викликає клієнтську заглушку.
2. Клієнтська заглушка створює повідомлення й викликає локальну операційну систему.
3. Операційна система клієнта пересилає повідомлення віддаленій операційній системі.
4. Віддалена операційна система передає повідомлення серверній заглушці.
5. Серверна заглушка вилучає з повідомлення параметри й викликає сервер.
6. Сервер виконує виклик і повертає результати заглушці.
7. Серверна заглушка запаковує результати в повідомлення й викликає свою локальну операційну систему.
8. Операційна система сервера пересилає повідомлення операційній системі клієнта.
9. Операційна система клієнта приймає повідомлення й передає його клієнтській заглушці.
10. Заглушка вилучає результати з повідомлення й передає їх клієнтові.

Ефекти взаємодії в мережі за допомогою цих кроків полягають у тому, що клієнтська заглушка перетворює локальний виклик процедури клієнта в локальний виклик процедури сервера, причому ні клієнт, ні сервер нічого не знають про проміжні дії.

3.3. Звертання до віддалених об'єктів

Розглянемо, як реалізує RPC звертання до віддалених об'єктів, і як подібний підхід підвищує прозорість мережної взаємодії (розподілу програмного забезпечення між вузлами мережі) порівняно з викликами віддалених процедур RPC.

3.3.1. Розподілені об'єкти

Ключова особливість об'єкта полягає в тому, що він інкапсулює стани й методи (*methods*): **стани** – дані, які інкапсулює об'єкт; **методи** – операції над даними, які також інкапсулює об'єкт. Доступ до методів можна одержати через **інтерфейс.** Важливо зрозуміти, що єдино правильним способом доступу або маніпулювання станом об'єкта є використання методів, доступ до яких здійснюється через інтерфейс цього об'єкта, який може реалізовувати декілька інтерфейсів. Для такого опису інтерфейсу може бути кілька об'єктів, які надають його реалізацію.

Розподіл на інтерфейси й об'єкти, які реалізують інтерфейси, є дуже важливим для розподілених систем, оскільки чіткий поділ дозволяє розміщувати інтерфейс на одній машині за умови, що сам об'єкт перебуває на другій.

Структуру розподіленого об'єкта (distributed object) показано на рис. 3.5. Коли клієнт виконує прив'язку до розподіленого об'єкта, в адресний простір клієнта завантажується реалізація інтерфейсу об'єкта, яку називають **замісником** (*proxy*). Замісник клієнта є аналогічним клієнтській заглушці в системах RPC. Єдине, для чого він призначений, – виконувати *маршалінг* параметрів у повідомленнях у разі звертання до методів і *демаршалінг* даних із повідомлень відповіді, які містять результати звертання до методів, передаючи їх клієнтові. Самі об'єкти перебувають на сервері й надають необхідні клієнтсь-

кій машині інтерфейси. Вхідний запит на звертання до методу спочатку потрапляє на серверну заглушку.

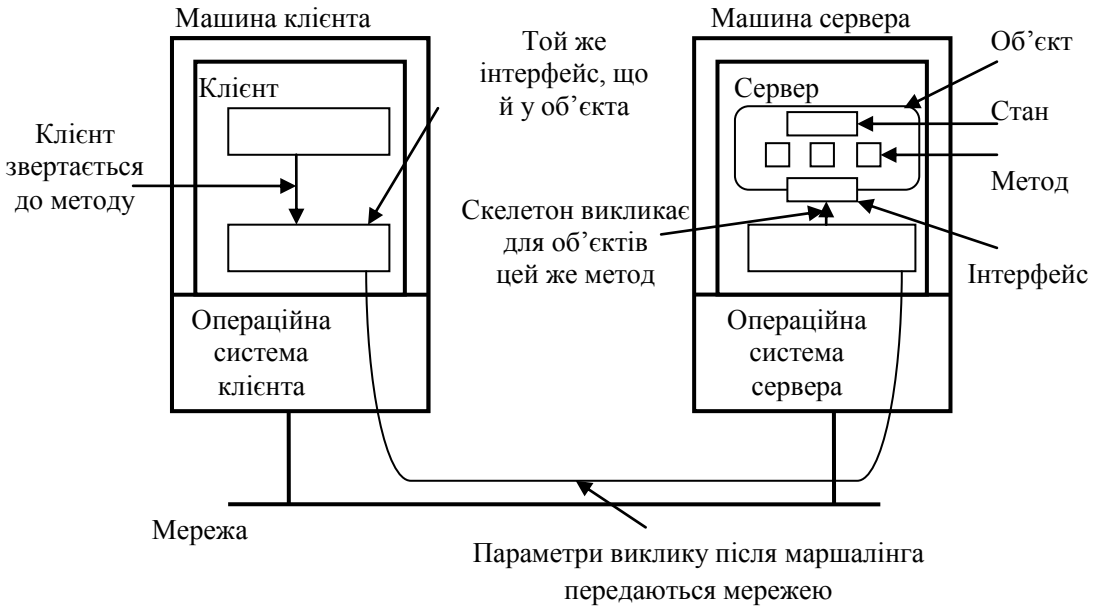

Рис. 3.5. Узагальнена організація віддалених об'єктів з використанням замісника клієнта

Скелетон – серверна заглушка, яка перетворить звертання клієнта у правильне звертання до методу через інтерфейс об'єкта, який перебуває на сервері. **Серверна заглушка** також відповідає за маршалінг параметрів у повідомленнях-відповідях і пересилання їх замісникові клієнта.

3.3.2. Прив'язка клієнта до об'єкта

Відмінність між традиційними системами RPC і системами, що підтримують розподілені об'єкти, полягає в тому, що останні зазвичай надають посилання на об'єкти, унікальні в межах системи. Коли процес зберігає посилання на об'єкт, перед звертанням до кожного з методів об'єкта процес має передусім виконати прив'язку до цього об'єкта. Результатом прив'язки буде замісник, який розташовується в адресному просторі процесу й реалізує інтерфейс із методами, до яких звертається процес. Це означає, що клієнтові надається простий механізм, який дозволяє напряму запитувати методи, використовуючи тільки посилання на об'єкт. У разі неявної прив'язки клієнт

прозоро зв'язується з об'єктом у момент дозволу на виконання посилання й одержання цього об'єкта насправді.

Явна прив'язка *(explicit binding)* полягає у тому, що клієнт має перед звертанням до методу викликати спеціальну функцію для прив'язки до об'єкта.

3.3.3. Статичне й динамічне віддалене звертання до методів

Віддалене звертання до методів (RMI) – звертання клієнта через замісника до методів об'єкта, тобто виклик спеціальної функції (замісника) для звертання до методів об'єкта.

Стандартний спосіб підтримки RMI – описати інтерфейси об'єктів мовою визначення інтерфейсів, як і у RPC. Окрім того, можна використати об'єктну мову, наприклад Java, що забезпечує автоматичне генерування заглушок. Такий підхід до застосування попередньо визначених описів інтерфейсів називають **статичним звертанням** *(static invocation).*

Статичне звертання *(static invocation)* вимагає, щоб інтерфейси об'єкта під час розробки клієнтської прикладної програми були відомі, а також передбачає, що зі зміною інтерфейсу клієнтську прикладну програму перед використанням нових інтерфейсів буде перекомпільовано.

Альтернативою є звертання до методів, яке може здійснюватися більш динамічно, зокрема, іноді зручніше визначити параметри звертання до методу під час виконання. Цей процес відомий під назвою **«динамічного звертання»** *(dynamic invocation)*, тобто звертання, під час виконання якого прикладна програма обирає, який метод віддаленого об'єкта буде викликаний.

Динамічне звертання зазвичай записують таким чином: *invoke (object, method. input_parameters. output_parameters),* де *object* – ідентифікує розподілений об'єкт; *method* – параметр, який точно задає метод, який викликається; *input_parameters* – структура даних, у якій утримуються значення вхідних параметрів методу; *output_parameters* – структура даних, у якій зберігаються значення, які повертаються.

Приклад. Розглянемо додавання цілого числа *int* до об'єкта *fobject* файлу. Для виконання цієї дії об'єкт надає метод *append*. У такому разі статичне звертання матиме

вигляд *fobject.append (int),* а динамічне – *invoke(fobject, id(append). int),* де операція *id(append)* повертає ідентифікатор методу *append*.

Інша область застосування динамічних звертань – *служби пакетної обробки*, для яких запити на звертання можуть оброблятися протягом часу, поки звертання очікує виконання.

3.3.4. Передача параметрів

Розглянемо ситуацію, коли всі об'єкти в системі є розподіленими, тобто доступними з віддалених машин. У цьому разі під час звертань до методів потрібно постійно використовувати посилання на об'єкти, як на параметри. Посилання передаються за значенням і копіюються з однієї машини на другу. У такому разі це посилання копіюється й передається як параметр-значення лише тоді, коли воно вказує на віддалений об'єкт. Саме в цьому разі відбувається передача об'єкта за посиланням. Однак, якщо посилання вказує на локальний об'єкт, тобто об'єкт в адресному просторі клієнта, то об'єкт, на який указує посилання, повністю копіюється й у процесі звертання передається клієнтові. У такому разі об'єкт передається за значенням.

Приклад. Ці два способи реалізації звертань показано на рис. 3.6, на якому зображені клієнтська програма, яка виконується на машині *A*, і програма-сервер, яка виконується на машині *C*. Клієнт володіє посиланням на локальний об'єкт *01,* який використовується як параметр під час виклику серверної програми на машині *C,* а також посиланням на віддалений об'єкт *02*, який міститься на машині *B* і також використовується як параметр. Під час виклику сервера на машину *C* передається копія всього об'єкта *01* і копія посилання на об'єкт *02*.

Додатковий ефект звертання до методів з використанням як параметра посилання на об'єкт полягає в можливості копіювати об'єкти, які приховати неможливо, а тому потрібно явно вказувати на відмінності між локальними й розподіленими об'єктами.

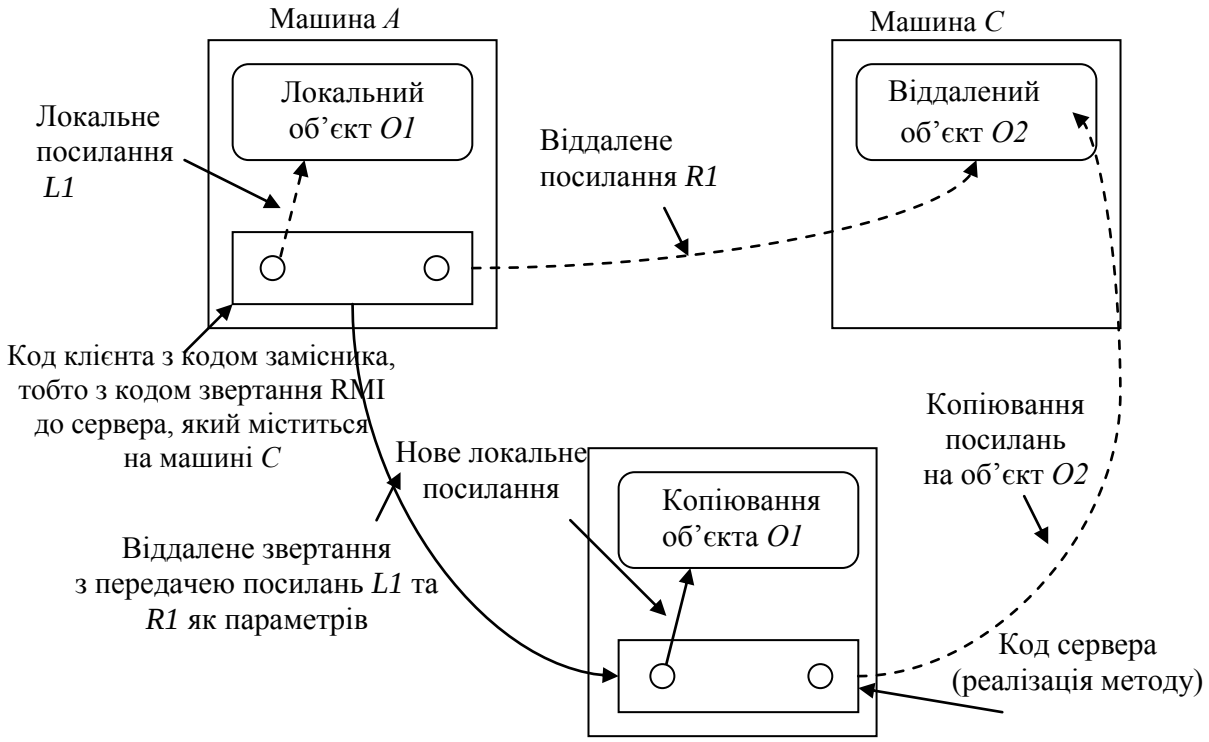

Рис. 3.6. Передача об'єкта за посиланням і за значенням

3.4 Зв'язок на основі потоків даних

Розглянемо, які засоби можуть використовуватися розподіленими системами для роботи з інформацією, критичною до часових характеристик передачі, зокрема з відео- та аудіопотоками.

3.4.1 Підтримка безперервних середовищ

Підтримка обміну критичною до часових характеристик передачі інформацією часто зводиться до підтримки **безперервних середовищ.** Під **середовищем** розуміють те фізичне середовище, що несе інформацію. Найважливіша характеристика середовища – спосіб представлення інформації.

Безперервне середовище представлення (continuous representation media) – це часові співвідношення між різними елементами даних, які лежать в основі коректної інтерпретації змісту даних.

Приклад. Одержання звуку під час відтворення аудіопотоку.