СИСТЕМИ ТА АЛГОРИТМИ КОНСТРУКТОРСЬКОГО ПРОЕКТУВАННЯ

УДК 681.3.049

Д. Корпильов, С. Ткаченко
Національний університет "Львівська політехніка",
кафедра систем автоматизованого проектування

ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ CORBA ПРИ ПОБУДОВІ СИСТЕМНОГО СЕРЕДОВИЩА САПР ГІБРИДНИХ ІНТЕГРАЛЬНИХ СХЕМ

Розподілена обробка даних в сучасних комп'ютерних системах нині стає актуальною. Використання індустріальних стандартів для створення розподілених систем, таких, як Object Management Group (OMG) CORBA, дає змогу робити системи "відкритими" в умовах гетерогенного розподіленого оточення. Стандарт CORBA розраховано на створення нових об'єктно-орієнтованих прикладних застосувань, він може успішно використовуватися і для розподілення існуючих об'єктних систем.

Вступ

Гетерогенні комп'ютерні середовища стали сьогодні реальністю у багатьох галузях. У зв'язку з цим підвищуються вимоги до інтеграції застосувань, які автоматизують діяльність і функціонують у розподілених середовищах із широким діапазоном апаратних платформ і мереж. Прикладні компоненти, а саме самостійні блоки програмного коду багаторазового використання, розподілені по мережі, стають усе популярнішими як будівельні блоки для створення розподілених систем.

В результаті системні середовища САПР повинні будуватися вже не з розрахунку на автоматизацію окремих задач, а передбачати можливість автоматизації різних етапів автоматизованого проектування у конструкторських бюро, володіючи властивостями масштабування, інтеграції та адаптації. У разі великих, територіально розподілених систем додатково з'являються такі вимоги, як можливість розподілених обчислень, функціонування в гетерогенних середовищах. Як правило, гетерогенно розподілені системи масштабу великих конструкторських бюро використовують більшість етапів автоматизованого проектування, а, отже, стають одним з чинників, що впливає на ефективність виробничої діяльності.

Реалізуючи проекти створення гетерогенно розподілених систем, на різних етапах розробники все активніше застосовують об'єктні методології та технології. Зростання популярності об'єктних методологій і технологій супроводжується появою, з одного боку, об'єктних методологій аналізу і проектування, які в сукупності складають методологічний базис, а з іншого боку, об'єктних технологій побудови гетерогенно розподілених інформаційних систем, якими є CORBA, DCOM [1, 2].

Розподілені об'єктні технології стають ефективним засобом побудови гетерогенних розподілених систем. Їх правильне використання може істотно знизити ризик, пов'язаний з невдалим створенням проектів автоматизованих систем, спростити етапи впровадження і супроводу систем.

1. Основні підходи до побудови складних систем

Основний принцип управління будь-якою складною системою давно відомий: "devide et impera" – "розділяй і володарюй". Відповідно до цього принципу складна програмна система на верхньому рівні повинна складатися з невеликої кількості порівняно незалежних компонентів з чітко визначеними інтерфейсами. Потім декомпозиції підлягають виділені на першому етапі компоненти , і так далі до заданого рівня деталізації.

Кінцевою метою декомпозиції є розбиття простору змінних

$$\{y_1, y_2, ... y_q, x_1, x_2, ..., x_p, v_1, v_2, ... v_r, f_1, f_2, ..., f_s\},$$

де $\vec{V} = \{v_1, v_2, ..., v_r\}$, $\vec{F} = \{f_1, f_2, ..., f_s\}$ – відповідно збурення, що спостерігаються та не спостерігаються, на q підпросторів меншого розміру, в яких враховується тільки зв'язок даного виходу y_i з відповідними змінними. Якщо будь-який вихід має зв'язок з рештою виходів, то декомпозиція практично неможлива [4]. Отже, система являє собою ієрархію з декількома рівнями абстракцій.

Сьогодні в розробці програмного забезпечення існують два основні підходи, відмінність між якими зумовлена критеріями декомпозиції. Перший підхід називають функціонально-модульним, або структурним, він базується на принципі алгоритмічної декомпозиції, коли виділяються функціональні елементи системи і встановлюється строга послідовність виконання дій. Другий, об'єктно-орієнтований підхід використовує об'єктну декомпозицію, поведінка системи описується у термінах взаємодії об'єктів.

В основу функціонально-модульного підходу покладено принцип алгоритмічної декомпозиції, відповідно до якого відбувається розподіл функцій програмних систем на модулі за функціональним призначенням, коли кожен модуль системи реалізує один з етапів спільного процесу. Традиційний функціонально-модульний підхід до розробки інформаційної системи передбачає строгу послідовність дій (модель водоспаду). На думку Страуструпа [6], головний недолік моделі водоспаду полягає у схильності інформації текти у один бік. Якщо проблема знаходиться "внизу", то виконуються лише обмежені виправлення і вирішують проблему без впливу на попередні стадії проекту. Зміна вимог до системи може призвести до її повного перепроектування, тому помилки, що закладені на попередніх етапах, позначаються на часі і остаточній ціні розробки.

Ще однією проблемою в побудові системних середовищ є різнорідність інформаційних ресурсів, що використовуються у середовищах САПР.

Проблема різнорідності вимагає вирішення у вигляді методик інтеграції ресурсів системних середовищ. Така методика повинна визначати системну архітектуру, що дає змогу забезпечити взаємодію компонентів середовища САПР. Внаслідок організаційних і технічних причин така інтеграційна архітектура повинна базуватися на розподіленій моделі обчислень.

Основні поняття об'єктно-орієнтованого підходу – об'єкт, клас, екземпляр. Об'єкт – це абстракція множини предметів реального світу, що володіють однаковими характеристиками та законами поведінки. Об'єкт являє собою типовий невизначений елемент такої множини.

Екземпляр об’єкта – це конкретний визначений елемент множини. Клас – це множина предметів реального світу, що зв’язані спільною структурою та поведінкою. Елемент класу – це конкретний елемент даної множини. Наступними поняттями об’єктного підходу є інкапсуляція, успадкування і поліморфізм. Об’єктний підхід передбачає, що власні ресурси, якими можуть маніпулювати тільки методи самого об’єкта, сховані від зовнішніх компонентів. Приховування даних і методів як власних ресурсів об’єкта називається інкапсуляцією. Поняття поліморфізму може бути інтерпретоване як здатність об’єкта належати більш ніж до одного типу. Наслідування означає побудову нових класів на основі існуючих з можливістю додавання або перевизначення даних і методів.

1.1. Використання об’єктного підходу

Об’єкти – сутності, що інкапсулюють дані, – це основні елементи, що моделюють реальний світ. На відміну від структурного підходу, де основна увага приділялась функціональній декомпозиції, в об’єктному підході предметна область ділиться на деяку множину порівняно незалежних сутностей – об’єктів. Об’єктна декомпозиція базується на виділенні агентів, які або самі діють, або на яких виконуються дії.

Об’єктно-орієнтована система з самого початку будується з врахуванням її еволюції. Ключові елементи об’єктного підходу – наслідування і поліморфізм – забезпечують можливість визначення нової функціональності класів об’єктів за допомогою створення похідних класів – нащадків базових класів. Нащадки наслідують характеристики батьківських класів без змін їх початкового опису і додають, якщо необхідно, власні структури даних і методи. Визначення похідних класів, при яких задаються тільки відмінності або уточнення, економить час і зусилля при розробці і використанні специфікацій і програмного коду [8].

Також важливою властивістю об’єктного підходу є узгодженість моделей системи від стадії аналізу до програмних модулів. Вимога узгодженості моделей виконується внаслідок можливості використання абстрагування, модульності, поліморфізму на усіх стадіях розробки. Моделі аналізу можуть бути безпосередньо порівняні з моделями реалізації. За об’єктними моделями можна простежити відображення реального змісту предметної області, яка моделюється у об’єкти і класи системи, що проектується.

2. Об’єктні розподілені технології на основі специфікацій консорціуму OMG

Дві концепції – об’єктно-орієнтований спосіб розробки і розподілені обчислення – стали основними предметами розгляду для створеного в квітні 1989 року консорціуму Object Management Group (OMG), членами якого сьогодні є більше ніж 500 провідних комп’ютерних компаній, таких як, Sun, DEC, IBM, HP, Motorola та інші. Завданням консорціуму є розроблення специфікацій і стандартів, що дають змогу будувати розподілені об'єктні системи в різнорідних середовищах. Базовими стали специфікації, що отримали назву Object Management Architecture (OMA) [3].

OMA складається з чотирьох основних компонентів, що являють собою специфікації різних рівнів підтримки застосувань:

- архітектура брокера запитів об'єктів (CORBA – Common Object Request Broker Architecture) визначає механізми взаємодії об'єктів у гетерогенній мережі [2];
- сервіси об’єктів (Object services) є основними системними сервісами, що використовуються розробниками для створення додатків [3];

• універсальні засоби (Common Facilities) є системними сервісами високого рівня, орієнтованими на підтримку призначених для користувача додатків, таких, як електронна пошта, засоби друку;

• прикладні об’єкти (Application Objects) призначені для розв’язання конкретних прикладних задач.

Концептуально специфікація CORBA належить до двох верхніх рівнів семирівневої моделі взаємодії відкритих систем (OSI – open System Interconnection) (рис. 1) [2]. Характерні особливості проведення розробок у технології CORBA такі.

• Мова опису інтерфейсів OMG IDL дає змогу визначити інтерфейс, незалежний від мови реалізації.

• Високий рівень абстракції CORBA в семирівневій моделі OSI позбавляє програміста від роботи з мережевими протоколами на низькому рівні.

• Програмісту не потрібна інформація про місце сервера в мережевій системі і спосіб його активації.

• Розробка клієнтської програми не залежить від операційної системи сервера і апаратної платформи.

Об'єктна модель CORBA визначає взаємодію між клієнтами і серверами. Клієнти – це застосування, які запитують сервіси, що надаються серверами. Об'єкти-сервери містять набір сервісів, що розділяються між багатьма клієнтами. Операція вказує сервіс, що запитується. Інтерфейси об'єктів описують безліч операцій, які можуть бути викликані клієнтами певного об'єкта. Реалізації об'єктів – це додатки, що виконують сервіси, які запитуються клієнтами.

3. Проектування і реалізація підсистем

Перш ніж перейти до питань, пов’язаних з проектуванням і реалізацією підсистем, зупинимося на рішеннях, якими можуть керуватися розробники при проектуванні компонентів, що створюються в рамках технології CORBA. Під компонентою будемо розуміти програмний модуль, що виконується в рамках окремого процесу операційної системи. Крім цього, будемо вважати, що мова реалізації C++.

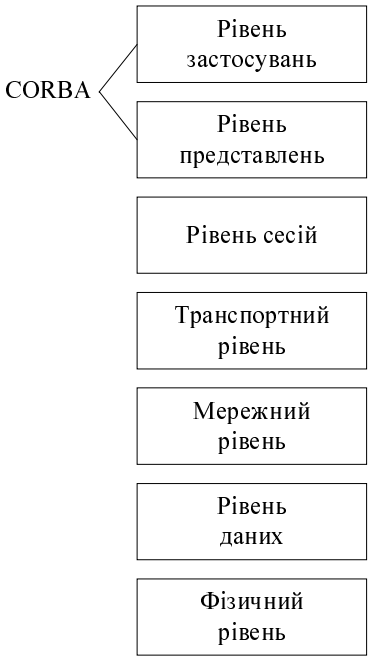

Рис. 1. Модель з семи рівнів мережної взаємодії OSI

Дослідження проектів побудови розподілених систем показало, що основними стадіями етапу проектування, яким приділяється підвищена увага, є:

• проектування інтерфейсів, що підтримуються об’єктами;

• визначення принципів взаємодії об’єктів на рівні середовища реалізації;

• розподіл (декомпозиція) об'єктів по компонентах інформаційної системи.

Створення середовищ САПР з використанням технології CORBA передбачає наявність всередині компонентів об'єктів двох типів: об'єктів, для яких визначений IDL – (Interface Definition Language – мова опису інтерфейсів) – інтерфейс, і об'єктів, для яких IDL-інтерфейс не визна-

чається. Правильне проектування інтерфейсів об'єктів багато в чому визначається специфікою предметної галузі, що розглядається, методологіями аналізу, що використовуються, і проектування, кваліфікацією проектувальників

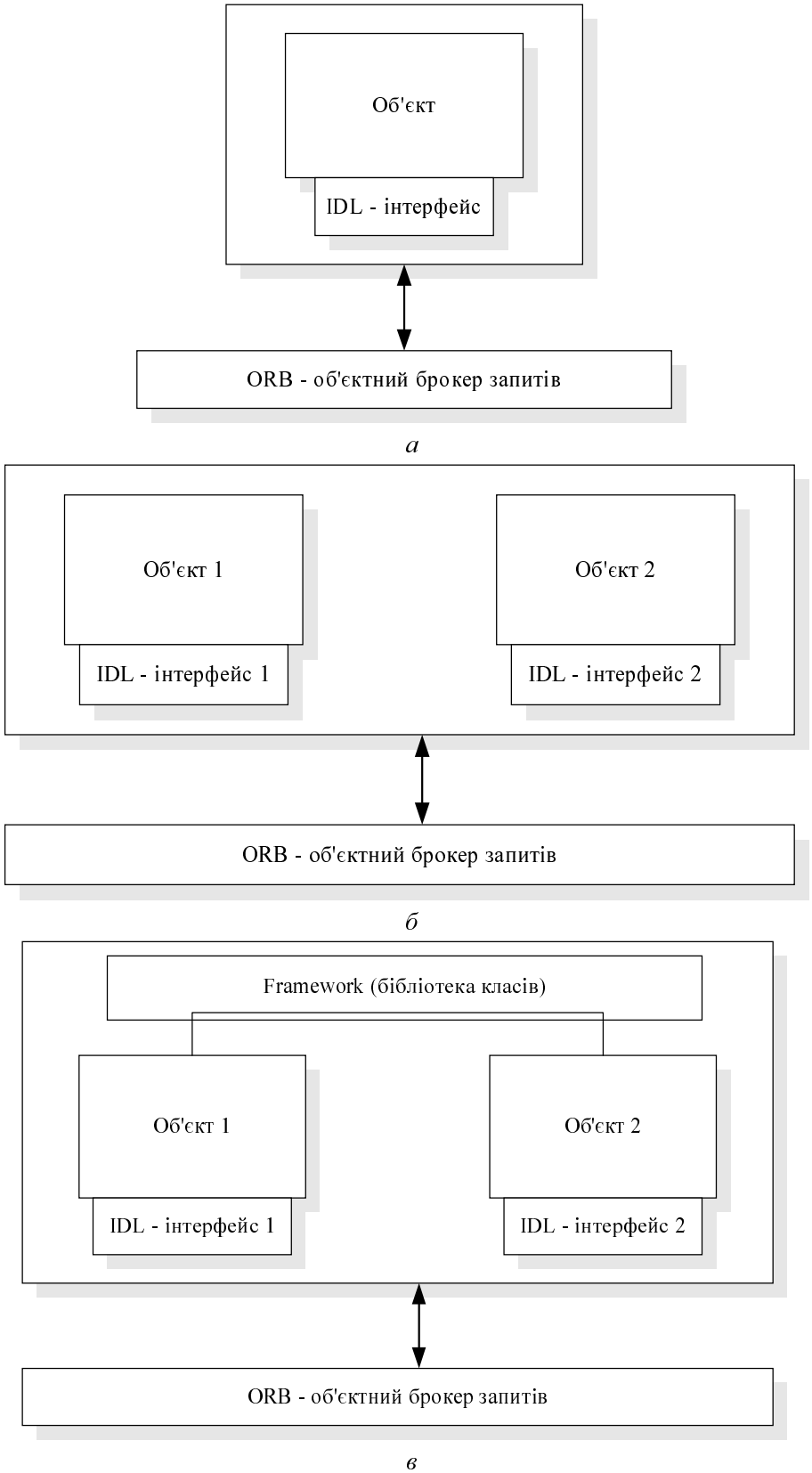

Рис. 2. Варіанти декомпозиції CORBA-об'єктів

і заслуговує окремого розгляду. З усіх об'єктів, що виникають на етапах аналізу, а потім і проектування, здебільшого лише небагато вимагає специфікації для них IDL-інтерфейсів. Це пов'язано з тим, що об'єкти, що володіють IDL-інтерфейсами і входять до складу деякої компоненти, фактично визначають інтерфейс її взаємодії з іншими компонентами. Ці об'єкти приховують деталі реалізації функціональності, що надається компонентою, і дають змогу на вищих рівнях деталізування розглядати їх як вершину айсберга, під якою розташовуються звичайні C++ об'єкти (списки, дерева, асоціативні масиви тощо). Спроба ж постачати кожний C++ об'єкт IDL-інтерфейсом досить часто приводить до збільшення кількості віддалених викликів. Істотно ускладнюється модифікація компонент, оскільки з'являється велика кількість зв'язків між розподіленими об'єктами.

Існує досить багато літератури, що описує принципи реалізації компонентів і об'єктів на мові C++. Тому, не зупиняючись на цьому, перейдемо до обговорення основних принципів взаємодії і варіантів декомпозиції CORBA-об'єктів, тобто об'єктів, для яких визначаються IDL-інтерфейси. Для цього розглянемо архітектурні рішення, зображені на рис. 2.

3.1. Варіанти декомпозиції CORBA-об'єктів по компонентах

Перший з них (рис. 2, а) передбачає, що в компоненті існує лише один CORBA-об'єкт. Хоч дане архітектурне рішення і тривіальне, іноді його використання виправдане (два випадки наведено нижче). По-перше, при інтеграції успадкованих додатків в розподілену інформаційну систему, коли потрібно просто постачати успадкований додаток відкритим інтерфейсом, і тим самим надати іншим компонентам можливість користуватися його послугами. І, по-друге, при реалізації компонентів, в яких необхідна наявність єдиного погляду на сукупність звичайних C++ об'єктів, в сумі надають деяку осмислену послугу. Небезпека використання даного рішення полягає в тому, що надмірна кількість розподілених компонентів в інформаційній системі може не збільшити, а зменшити швидкість обробки інформації, оскільки супроводиться зростанням кількості викликів до віддалених об'єктів, збільшуючи мережевий трафік.

На відміну від першого, друге рішення не обмежує кількість CORBA-об'єктів, що підтримуються однієї компонентою. У побудові розподілених систем саме воно є найпоширенішим і дає змогу найбільшою мірою відчути ефективність використання технології CORBA. Особливе значення надається правильному вибору засобів взаємодії CORBA-об'єктів, розташованих в одній компоненті. Це зумовлено тим, що частина з них, з одного боку, є звичайними C++ об'єктами, а з іншої – підтримує деякий набір IDL-інтерфейсів. Такий стан справ робить можливим забезпечити їх взаємодію або засобами, що надаються ORB (Object Request Broker – брокер об’єктних запитів) (рис. 2, б), або за допомогою C++ бібліотеки, реалізуючи деякий локальний framework взаємодії (рис. 2, в). Дослідження показують, що перший спосіб доцільно застосовувати, якщо існує хоч би невелика імовірність того, що об'єкти під час експлуатації інформаційної системи можуть бути рознесені по різних компонентах. Так, наприклад, нерідко для досягнення вищої продуктивності і/або завадостійкості потрібна присутність в системі декількох однакових компонентів, об'єкти яких взаємодіють один з одним. На початкових стадіях проектування необхідність у розділенні не завжди очевидна, і якщо проектувальник її не передбачив, і реалізовував взаємодію за допомогою локальної framework's взаємодії, то надалі можуть виникнути істотні витрати на перепрограмування компонентів. З іншого боку, не можна не відзначити,

що використання локального framework's збільшує швидкість взаємодії об'єктів, полегшує програмування компонентів. Практика показує, що найбільшої гнучкості і продуктивності можна досягти лише при розумному поєднанні першого і другого способів взаємодії, коли одні CORBA-об'єкти-компоненти взаємодіють за допомогою локального framework's, а інші користуються послугами, що надаються ORB.

Описані вище рішення торкаються проектування компонентів. Однак, як вже згадувалося, у великих системах компоненти на етапі проектування об'єднуються в підсистеми, що є ефективним засобом боротьби зі складністю. При цьому рішення і принципи, що використовуються при проектуванні і реалізації підсистем, відрізняються від рішень і принципів, що використовуються при створенні компонентів. Більш того, при проектуванні підсистем, залежно від задач, що вирішуються ними, також застосовуються різні технологічні рішення. У зв'язку з цим розглянемо, як можуть створюватися підсистеми гетерогенної розподіленої системи, що базується на технології CORBA. Як підсистеми будуть виступати: підсистема доступу до баз даних, підсистема призначеного для користувача інтерфейсу, підсистема, що реалізує логіку, і підсистема управління процесами, що автоматизуються (керуюча структура). Крім цього, будемо передбачати, що система, яка створюється, задовольняє такі вимоги:

- постійна (безупинна) робота компонентів з моменту запуску системи;
- підтримка транзакцій;
- наявність розподілених, можливо неоднорідних баз даних;
- підтримка примусового інформування користувачів про події, що відбуваються в системі;
- незалежність компонент призначеного для користувача інтерфейсу від апаратних і програмних платформ;
- підтримка можливості модифікації компонентів без зупинки системи;
- наявність можливості динамічної реконфігурації системи.

Дослідження проектів показують, що перераховані вимоги характерні для багатьох великих розподілених інформаційних систем, що функціонують в гетерогенних середовищах.

4. Концепція інформаційної архітектури системного середовища ГІС

Для написання розподілених обчислювальних систем недостатньо однієї об’єктно-орієнтованої мови. Часто різні компоненти програмної системи вимагають реалізації на різних мовах і, можливо, на різних апаратних платформах. За допомогою об`єктних моделей множини об’єктів застосувань, зокрема, і на різних апаратних платформах, взаємодіють одна з одною і реалізують процеси, створюючи видимість одного цілого. Для забезпечення взаємодії об’єктів та їх інтеграції у єдину систему архітектура проміжного рівня повинна реалізовувати декілька базових принципів:

- незалежність від фізичного розміщення об’єкта: компоненти програмного забезпечення не повинні знаходитися в одному виконуваному файлі, виконуватися в межах одного процесу або розміщатися на одній апаратній платформі;
- незалежність від платформи: компоненти можуть виконуватися на різних апаратних і операційних платформах, взаємодіючи у межах єдиної системи;
- незалежність від мови програмування: різниця у мовах, які використовуються при створенні компонентів, не перешкоджають їх взаємодії один з одним.

Технологія взаємодії між клієнтським процесом і сервером об’єкта – це процес, який породжує і обслуговує екземпляри об’єкта, використовуючи механізм об’єктного виклику віддаленої процедури (RPC, remote procedure call). На рис. 3 наведено типову структуру RPC – технологію проміжного програмного забезпечення. Механізм RPC реалізує схему переда-

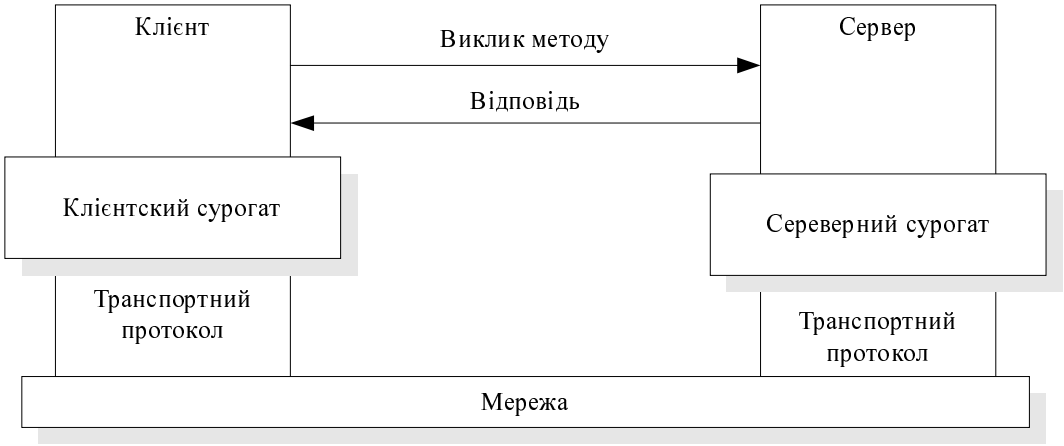

Рис. 3. Механізм виклику віддаленої процедури

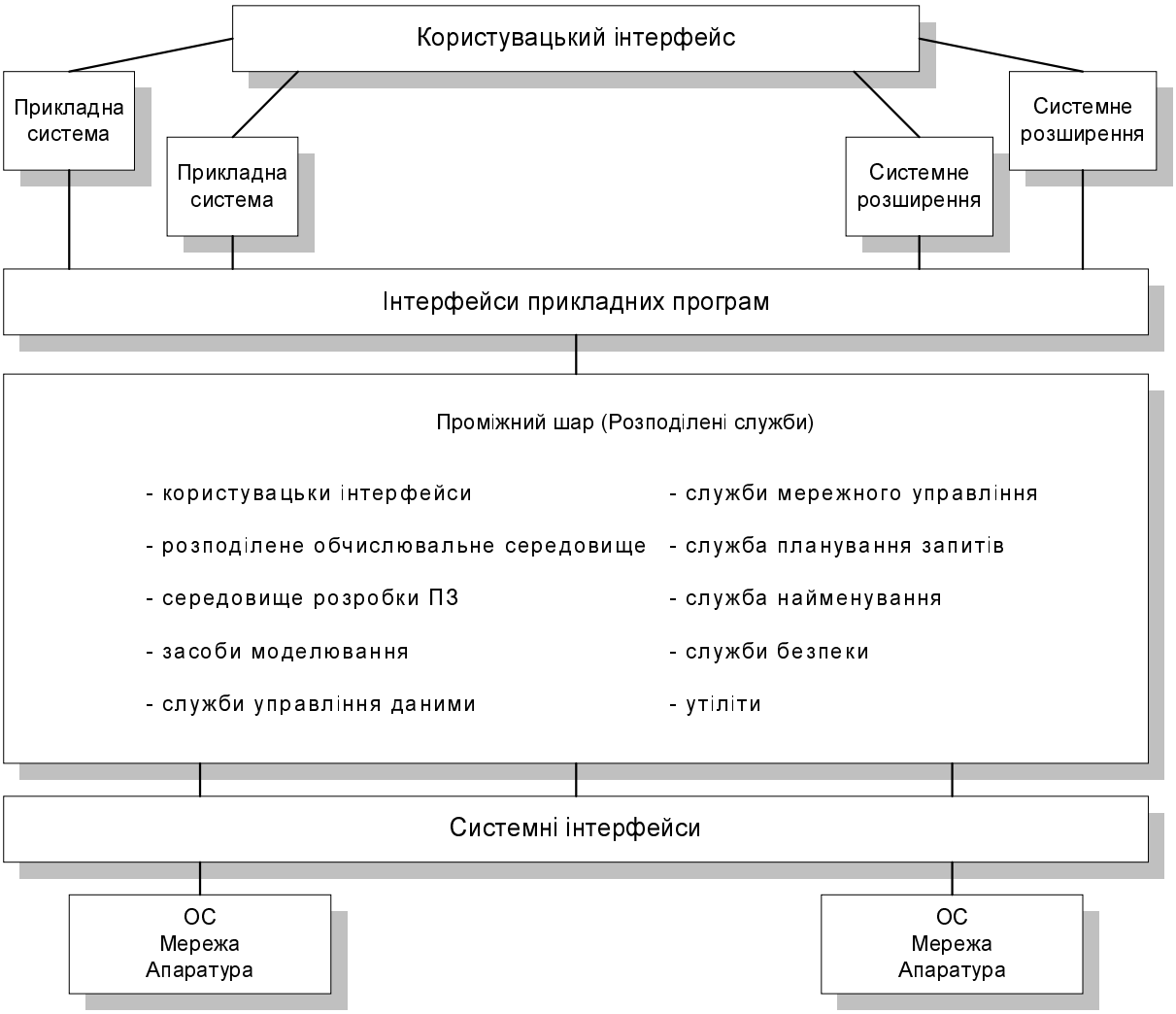

Рис. 4. Концепція проміжного шару

чі повідомлень, відповідно до якої у розподіленому клієнт-серверному застосуванні процедура-клієнт передає спеціальне повідомлення з параметрами виклику по мережі у віддалену серверну процедуру, а результати її виконання повертаються у іншому повідомленні клієнтському процесу.

Основу інформаційної архітектури системи складає концепція проміжного шару (middleware) – базових служб у спеціальному шарі архітектури, яка розміщена між операційною системою і засобами управління комп'ютерними мережами та прикладними системами (рис. 4) [5].

До проміжного шару належать засоби управління і доступу до даних, засоби розробки програмного забезпечення, засоби управління розподіленими обчисленнями, засоби підтримки користувацького інтерфейсу тощо.

Технічно інтероперабельність компонентів вирішена введенням базової об'єктної моделі, уніфікованої мови специфікацій інтерфейсів об'єктів, відокремленням реалізації компонентів від специфікації їх інтерфейсів, введенням спільного механізму підтримки інтероперабельності об'єктів. Тим самим досягається однорідність представлення компонентів та їх взаємодії.

Висновок

Успіх розробки середовища САПР ГІС "TOPOS" з використанням технології CORBA залежить від методологічного і технологічного базисів, які застосовуються розробниками на етапах проектування і реалізації компонентів інформаційної системи. Важливим позитивним чинником стає наявність програмної інфраструктури, яка виступає як підмурівок, що істотно полегшує побудову гетерогенних розподілених інформаційних систем, а також створити систему, що відповідає міжнародним стандартам САПР.

1. Роджерсон Д. Основы COM. М., 1997. 2. Common Object Request Broker Architecture (CORBA) web site: http: //www. corba.org. 3.Object Management Group Web site: http: //www. omg.org. 4. Молчанов А.А. Моделирование и проектирование сложных систем. К., 1988. 5. Броди М.Л. Интероперабельные информационные системы в науке // Сб. материалов сем. М., 6–7 апреля, 1995. 6. Страуструп Б. Язык программирования С++. 2-е изд. М.; СПб., 1999. 7. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. 2-е изд. М.; СПб, 1998. 8. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. М., 2000.