ІНСТРУМЕНТАЛЬНІ ЗАСОБИ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ

УДК 681.3.049

Д.В. Корпильов, С.П. Ткаченко, В.М. Теслюк,
Національний університет "Львівська політехніка",
кафедра САПР

БАГАТОРІВНЕВА ОБ'ЄКТНО-ОРІЄНТОВАНА АРХІТЕКТУРА СИСТЕМНОГО СЕРЕДОВИЩА САПР ГІБРИДНИХ ІНТЕГРАЛЬНИХ СХЕМ

Для об'єктно-орієнтованих інформаційних систем рекомендується використання багаторівневої архітектури, яка виконує розподіл обов'язків, виконуваних об'єктами класичної трирівневої архітектури. Розподілена обробка даних в сучасних комп'ютерних системах на даний час стає актуальною. Використання індустріальних стандартів для створення розподілених систем, таких як Object Management Group (OMG) CORBA, дозволяє робити системи "відкритими" в умовах гетерогенного розподіленого оточення. Стандарт CORBA розрахований на створення нових об'єктно-орієнтованих прикладних застосувань, він може успішно використовуватися і для розподілення існуючих об'єктних систем.

Allotted data treatment becomes actual in contemporary computer systems for a given while. Use of industrial standards for creation of allotted systems, such as Object Management Group (OMG) CORBA, allows to make the systems "open" in conditions geteregenion of distributed environment. A CORBA Standard is counted on creation of new object-orientated applied applications, it can be successfully used a for allotting of objective systems.

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: самостійні блоки програмного коду багаторазового використання, розподілені по мережі, стають усе більш популярними як будівні блоки для створення розподілених систем.

Отже, системні середовища САПР повинні будуватися вже не з розрахунку на автоматизацію окремих задач, а передбачати можливість автоматизації різних етапів автоматизованого проектування, що відбуваються у конструкторських бюро, маючи при цьому властивості масштабування, інтеграції та адаптації. Великі, територіально

розподілені системи додатково висувають такі вимоги, як можливість розподілених обчислень, функціонування в гетерогенний 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.

При реалізації проектів створення гетерогенно розподілених систем на різних етапах розробники все більш активно застосовують об'єктні методології і технології. При цьому 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 (Common Object Request Broker Architecture), 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.

Багаторівнева об'єктно-орієнтована архітектура

Типова архітектура інформаційних систем містить інтерфейс користувача і збереження даних на постійному носії і називається трирівневою архітектурою (three-tier architecture) (рис 1). При такій архітектурі застосування ділиться по вертикалі на три рівні.

1. Рівень представлення (Presentation) — вікна, звіти тощо.
2. Рівень логіки застосування (Application Logic) — задачі і правила керування процесом.
3. Рівень даних (Stogare) — механізм постійного збереження даних.

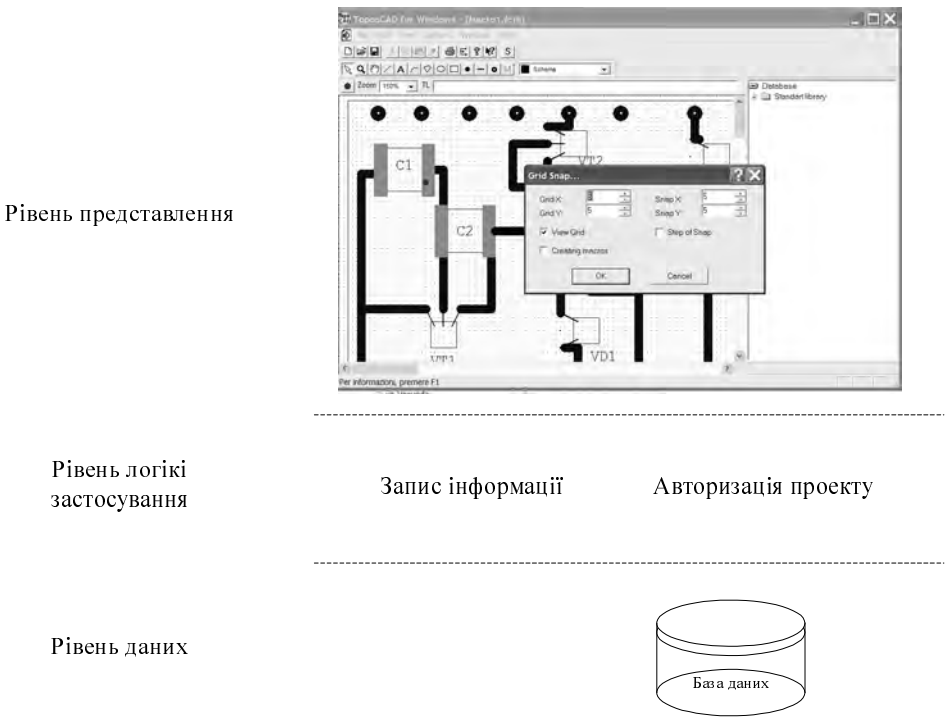


Рис. 1. Схема трирівневої архітектури

Для об’єктно-орієнтованих інформаційних систем рекомендується використовувати багаторівневу архітектуру, яка передбачає розподіл обов’язків, виконуваних об’єктами класичної трирівневої архітектури.

В об’єктно-орієнтованих системах виконується декомпозиція рівня логіки застосування на декілька рівнів. Така архітектура представляється в термінах програмних класів (рис. 2).

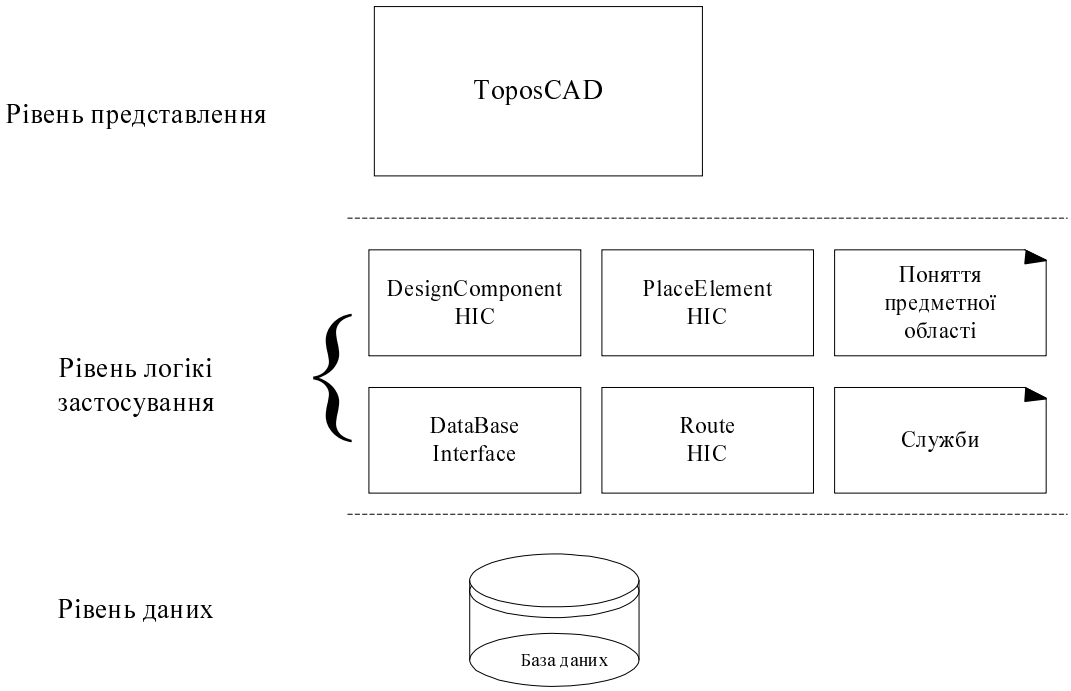


Рис. 2. Декомпозиція рівня логіки застосування на декілька менших рівнів

Після виконання декомпозиції рівня логіки застосування трирівнева архітектура системи перетворюється в багаторівневу архітектуру (multi-tiered architecture) без виділених границь рівня логіки.

Застосування з трирівневою логічною архітектурою можуть фізично розміщатися (розгортатися) в різних конфігураціях, зокрема:

— компоненти рівня представлення і логікі застосування розміщаються на комп’ютері клієнта, а база даних — на сервері;

— компоненти рівня представлення розміщаються на комп’ютері клієнта, рівень логікі застосування — на сервері застосувань, а дані — на окремому сервері баз даних.

З використанням мов і технологій програмування підтримка розподілених обчислень, таких як Java, CORBA, застосування стають все більш розподілені.

На користь багаторівневої архітектури можна навести такі аргументи:

— логіка застосування представляється у вигляді ізольованих компонентів, які можна використовувати в інших системах;

— різні рівні застосування можна розподілити по різних комп’ютерах та/або декількома процесами, що дозволить підвищити продуктивність і поліпшити координацію і сумісне використання інформації в системах клієнт/сервер;

— розробку окремих рівнів можна доручити спеціалізованим групам розробників, наприклад групі розробників інтерфейсу; при цьому підтримується високий рівень спеціалізації професіоналів і можливості паралельної розробки всіх рівнів застосування.

3. Архітектура CORBA.

Основними елементами архітектури, що складається з трьох ланок, є клієнти, сервери застосувань і серевери баз даних. Клієнти отримують доступ до серверів застосування, використовуючи експортовані останніми IDL-інтерфейси (Interface Definition Language – мова опису інтерфейсів) .

Сервери застосувань грають подвійну роль: з одного боку, серверів CORBA (Common Object Request Broker Architecture), з іншого – клієнтів баз даних, що встановлюють з'єднання з серверами баз даних для отримання відомостей. Три ланцюги такої архітектури представляються так (рис. 3).

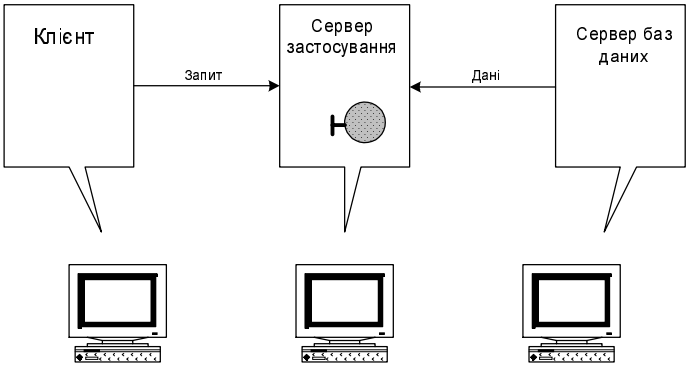


Рис. 3. Архітектура CORBA з трьох ланцюгів

Трирівнева схема утворює тільки основу для реальних обчислень на основі компонентів, які називаються ***N*** – ланцюговими обчисленнями. Інтерфейс користувача – це тільки одна частина ***N*** – ланцюгової схеми (рис. 4). Середній ланцюг – це набір взаємодіючих компонентів, кожен з яких може отримати доступ до служб інших компонентів за допомогою добре визначених інтерфейсів CORBA. Кожен компонент нічого не знає про деталі внутрішньої реалізації інших компонентів, механізм доступу до бази даних і схеми довготривалого збереження даних. Інкапсуляція такої внутрішньої складності компонентів і експортування тільки добре описаних на IDL інтерфейсів є основною метою інтеграції технології CORBA і баз даних.

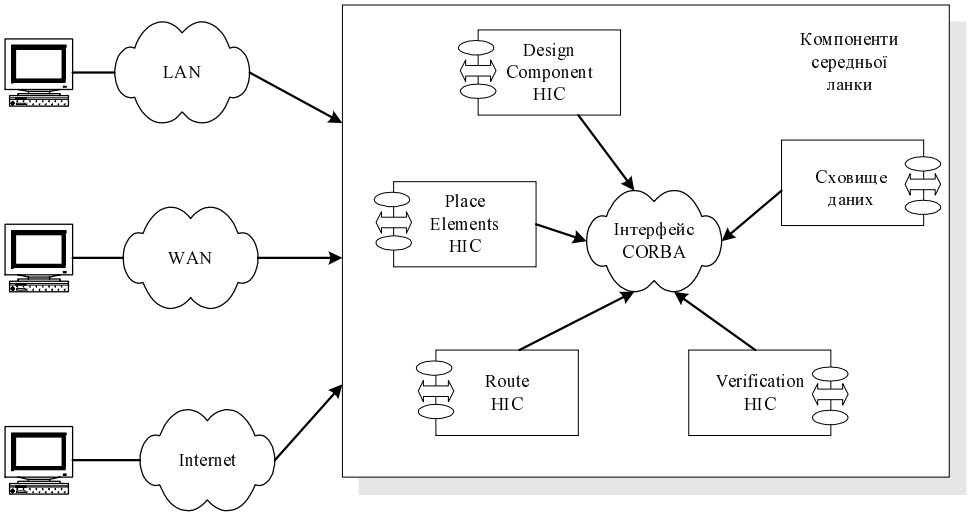


Рис. 4. Архітектура з **N**-*ланцюгів*

Складна програмна система на верхньому рівні повинна складатися з невеликого числа відносно незалежних компонентів із чітко визначеними інтерфейсами. Потім декомпозиції підлягають виділені на першому етапі компоненти та решта – 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 під просторів меншого розміру, в яких враховується тільки зв'язок даного виходу y_i з відповідними змінними. Якщо будь-який вихід має зв'язок з рештою виходами, то декомпозиція практично неможлива [4]. Тобто система становить ієрархію з декількома рівнями абстракцій.

Сьогодні в розробці програмного забезпечення існує два основні підходи, відмінність між якими зумовлена критеріями декомпозиції. Перший підхід називається функціонально-модульним, або структурним, він базується на принципі алгоритмічної декомпозиції, коли виділяються функціональні елементи системи і встановлюється строгий порядок виконання дій. Другий — об'єктно-орієнтований підхід з використанням об'єктної декомпозиції, коли поведінка системи описується у термінах взаємодії об'єктів.

В основу функціонально-модульного підходу покладений 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 строгий порядок 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 проблему вирішують без впливу на попередні стадії проекту. Зміна вимог до системи приводить до її повного перепроектування, тому помилки, що закладені на попередніх етапах, позначаються на часі і кінцевій ціні розробки.

Ще однією проблемою в побудові системних середовищ є різнорідність інформаційних ресурсов, що використовується середовищах САПР.

Проблема різнорідності вимагає рішення у вигляді методик інтеграції ресурсів системних средовищ. Така методика повинна визначати системну архітектуру, що дозволяє забезпечити взаємодію компонентів середовища САПР. В силу організаційних і технічних причин така інтеграційна архітектура повина базуватися на розподіленій моделі обчислень.

Основні поняття об'єктно-орієнтованого підходу — об'єкт, клас, екземпляр. Об'єкт — це абстракція множини предметів реального світу, що мають однакові характеристики та закони поведінки. Об'єкт — це 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.

4. Відображення архітектури з використанням пакетів UML

Для відображення груп елементів або підсистем в мові UML використовується механізм пакетів (packages). Пакет — це група елементів моделі будь-якого виду, наприклад класів, прецедентів, діаграм кооперації або інших пакетів вкладених пакетів. Всю систему можна розглядати як єдиний пакет верхнього рівня — пакет System. У пакет вкладений простір імен, тому в різних пакетах можуть зустрічатися елементи з однаковими іменами.

Графічно пакет зображається у вигляді папки (рис. 5).

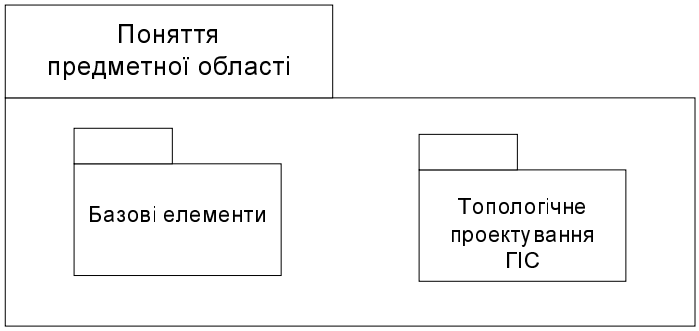


Рис. 5. Пакети в мові UML

Для ілюстрації архітектури системи засобами мови UML можна використосувати пакети. На рис. 6 архітектура, яка представлена на рис. 2., наведена за допомогою звичайних пакетів.

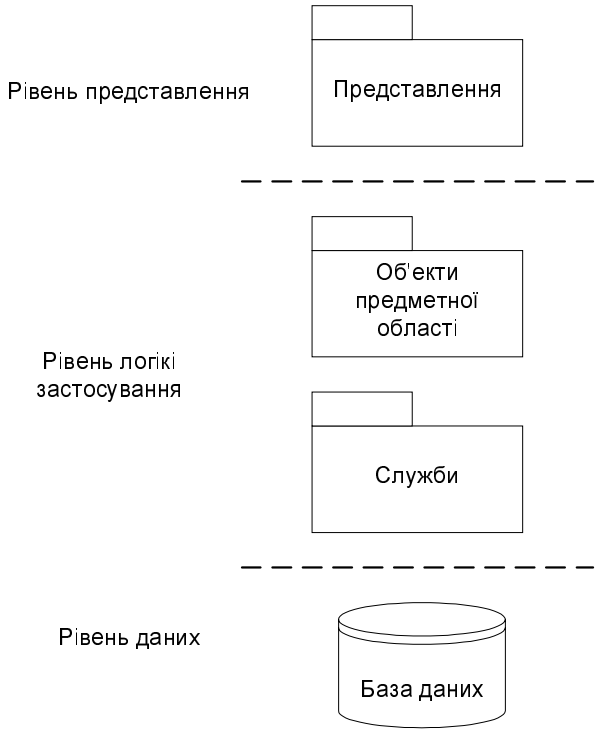


Рис. 6. Архітектура системи в термінах пакетів UML

Перш ніж перейти до обговорення питань, пов’язаних з проектуванням і реалізацією підсистем, зупинимося на рішеннях, якими можуть керуватися розробники при

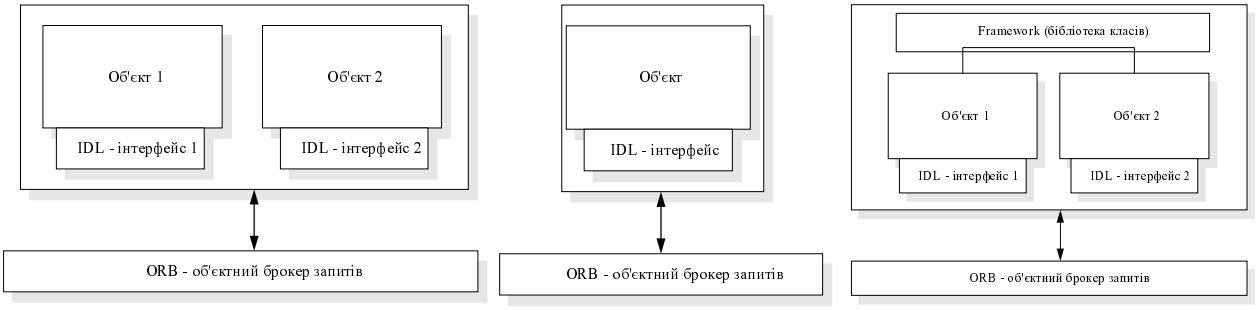
проектуванні компонентів, що створюються у межах технології CORBA. Під компонентою будемо розуміти програмний модуль, що виконується у межах окремого процесу операційної системи. Крім цього, будемо вважати, що мова реалізації C++.

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 за компонентами 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 визначається. Правильне проектування інтерфейсів об’єктів багато в чому визначається специфікою предметної області, що розглядається, методологіями аналізу і проектування, що використовуються, кваліфікацією проектувальників і заслуговує окремого розгляду. З всієї безлічі об'єктів, виникаючих на етапах аналізу, а потім і проектування, в більшості випадків лише деякі вимагають специфікації для них IDL – інтерфейсів. Це пов’язано з тим, що об’єкти, що володіють IDL – інтерфейсами і що входять до складу деякої компоненти, фактично визначають інтерфейс її взаємодії з іншими компонентами. При цьому ці об’єкти приховують за собою деталі реалізації функціональності, що надається компонентою, і дозволяють на більш високих рівнях деталізації розглядати їх як вершину айсберга, під якою розташовуються звичайні C++ об’єкти (списки, дерева, асоціативні масиви і інш.). Спроба ж подати кожному C++ об’єкту IDL – інтерфейс досить часто приводить до збільшення числа віддалених викликів. Істотно ускладнюється процес модифікації компонент, оскільки з’являється велика кількість зв’язків між розподіленими об’єктами.

У даний момент 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 наступні архітектурні рішення, які представлені на рис. 1.



(а) (б) (в)

Рис. 1. Варіанти декомпозиції CORBA-об'єктів

4.1. Варіанти декомпозиції CORBA — об'єктів за компонентами

Перше з них (рис. 1, а) передбачає, що в компоненті існує лише один CORBA – об’єкт. Хоч дане архітектурне рішення є тривіальним, існують випадки його виправданого

використання, два з яких наведені нижче. По-перше, при інтеграції успадкованих додатків в розподілену інформаційну систему, коли потрібно просто забезбечити 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, які 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 числа викликів до віддалених об’єктів, збільшуючи мережний трафік.

На відміну від першого, друге рішення не обмежує число CORBA-об’єктів, що підтримуються однією компонентою. При побудові розподілених систем саме воно є найбільш поширеним і дозволяє найбільшою мірою відчути ефективність використання технології CORBA. При цьому особливе значення надається правильному вибору засобів взаємодії CORBA-об'єктів, розташованих в одній компоненті. Це зумовлене тим, що частина з них, з одного боку, є звичайними C++ об’єктами, а з іншого – підтримує деякий набір IDL-інтерфейсів. Такий стан справ дозволяє забезпечити їх взаємодію або засобами, що надаються ORB (Object Request Broker — брокер об’єктних запитів) (рис. 1, б), або з допомогою C++ бібліотеки, реалізуючий деякий локальний framework взаємодії (рис. 1, в). Дослідження показують, що перший спосіб доцільно застосовувати у випадку, якщо існує хоч би невелика імовірність того, що об’єкти в процесі експлуатації інформаційної системи можуть бути рознесені за різними компонентами. Так, наприклад, нерідко для досягнення більш високої продуктивності і/або завадостійкості потрібна присутність в системі декількох однакових компонентів, об’єкти яких взаємодіють один з одним. На початкових стадіях проектування необхідність в розділенні не завжди очевидна, і якщо проектувальник її не передбачив і реалізовував взаємодію за допомогою локальної framework'a взаємодії, то надалі можуть виникнути істотні витрати на перепрограмування компонентів. З іншого боку, не можна не зазначити, що використання локального framework'a збільшує швидкість взаємодії об'єктів, полегшує процес програмування компонентів. Практика показує, що найбільшої гнучкості і продуктивності можна добитися лише при розумному поєднанні першого і другого способів взаємодії, коли одні CORBA-об’єкти компоненти взаємодіють з допомогою локального framework'a, а інші користуються послугами, що надаються ORB.

Описані вище рішення, зачіпають процес 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. Більше того, при проектуванні підсистем залежно від задач, які вони вирішують, також застосовуються різні технологічні рішення. У зв'язку з цим розглянемо, як можуть створюватися підсистеми гетерогенної розподіленої системи, що базується на технології CORBA. При цьому 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 динамічного реконфігурації системи.

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

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. При цьому важливим позитивним чинником стає наявність програмної інфраструктури, яка виступає як підмурівок, що істотно полегшує процес побудови гетерогенний розподілених інформаційних систем, а також дає змогу створити систему, що відповідає міжнародним стандартам САПР.

1. Цимбал А. Технология CORBA для професионалов -СПб: Питер, 2001. –621 с.: ил. 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.: Вища школа, 1988. 5. Броди М. Л. Интероперабельные информационные системы в науке / Матер. семинара. – М., 6-7 апреля, 1995. 6. Страуструп Б. Язык программирования С++, 2-е изд. / Пер. с англ. — М.: СПб., 1999. 7. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++, 2-е изд. / Пер. с англ. — М., СПб:, 1998. 8. Бучч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя / Пер. с англ. — М.: ДМК, 2000.