Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

Методичні вказівки до курсу

“Технологія клієнт-сервер”

для студентів факультету прикладної математики та інформатики

ч. IV. MIDAS реалізація багатошарової архітектури клієнт-сервер

Львів
Видавничий центр ЛНУ імені Івана Франка
2004

Рекомендовано до друку науково-методичною радою
факультету прикладної математики та інформатики
Протокол № 8 від 22.09.04

Уклали: Володимир Дмитрович Вовк,
Богдан Михайлович Голуб,
Іван Іванович Дияк

Рецензент Г. Шинкаренко
Редактор І. Лоїк
Коректор Р. Спринь

Видавничий центр Львівського національного університету імені Івана Франка. 79000 Львів, вул. Дорошенка, 41.

ЗМІСТ

ВСТУП

Архітектура *клієнт-сервер* є результатом застосування об'єктного підходу до вирішення задач побудови складних інформаційних систем. Згідно з ним інформаційна система є набором самостійно функціонуючих компонент, взаємодія яких визначається ієрархічними відносинами, що формулюються у термінах *сервер даних, сервер застосувань* та *клієнт*. У випадку складної системи зі значною кількістю серверів та клієнтів задача організації взаємодії між ними є багатокритеріальною і для однозначного розв'язування потребує застосування спеціальних концепцій та програмних засобів, котрі входять у склад основних на сьогодні технологій розподілених обчислень DCOM, CORBA та RMI. Однак для потреб практики часто достатньо розроблення значно простіших (а, отже, дешевших) програмних систем, наприклад, з одним-двома виділеними серверами застосувань. Найпростіші засади такої архітектури для баз даних обговорено у ч. I. "Основні ідеї та положення" нашої серії методичних вказівок. Метою цієї роботи є вивчення практичної реалізації тришарових моделей технології *клієнт-сервер*. Вибір програмних засобів для цього є достатньо широким. Проте завдяки простоті використання, ефективності роботи та доступності, широкої популярності у сфері розроблення баз даних здобула система візуального програмування Delphi (C++Builder у варіанті C++) від фірми Borland. Вона володіє власною технологією створення сервера застосувань під назвою MIDAS, яку використано авторами для розроблення низки успішно функціонуючих програмних продуктів. Саме вона і стане головним об'єктом вивчення.

1. ТРИШАРОВА АРХІТЕКТУРА MIDAS

Нагадаємо класичну схему DBS-моделі двошарової архітектури клієнт-сервер, наведеної у ч. I. “Основні ідеї та положення” цієї серії методичних вказівок.

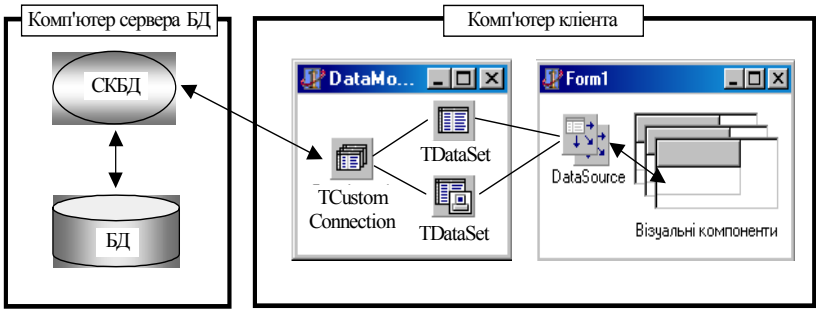

Рис.1. Взаємодія Delphi/CBuilder – компонент у двошаровій моделі

Проте, на відміну від оригіналу, у цій схемі ми конкретизували головні складові DBS-моделі відповідними компонентами Delphi (C++ Builder). Тут TCustomConnection позначає одну з реалізованих в Delphi/C++ Builder технологій доступу до даних: BDE (TDataBase), dbExpress (TSQLConnection), InterBase Express (TIBDataBase) та ADO (TADOConnection). Вибір потрібної технології визначається постановкою вихідної задачі та параметрами сервера БД.

Центральною проблемою двошарової моделі є розміщення громіздких обчислень з підтримання бізнес-правил. Якщо вони виконуються сервером БД, то це суттєво сповільнює обслуговування запитів від великої кількості клієнтів. Та й володіє він досить бідним набором засобів для реалізації складної бізнес-логіки (збережені процедури, тригери, перегляди і т.п.). Через це розробники змушені істотно ускладнювати програмний код на стороні клієнта, що спричинює зростання вимог до потужності його комп’ютера. Окрім цього, за великої кількості клієнтів загострюється проблема супроводу та внесення змін до вже працюючих програм.

Як зрозуміло з самої назви моделі, цю проблему тришарова архітектура клієнт-сервер вирішує утворенням додаткового програмного шару (сервера застосувань) для реалізації бізнес-правил, котрий дає змогу виконувати основну обчислювальну роботу на окремому комп’ютері (чи на декількох для великих систем). Сервер застосувань перебирає на себе обчислювальне навантаження як від сервера БД, так і від клієнта, даючи змогу останньому працювати навіть на слабких обчислювальних установках (так званий *худий клієнт*).

Зважаючи на важливість задачі, принципи побудови та функціонування сервера застосувань стали об’єктом досліджень провідних розробників програмного забезпечення. Сьогодні утримують лідируючі позиції дві головні технології організації розподілених обчислень – DCOM та CORBA. Отже, будь-яке середовище розроблення програмного забезпечення, котре підтримує створення інформаційних систем з багатошаровою архітектурою, надає програмісту інструментарій реалізації вищеперелічених технологій[1]. Проте навіть для найбільш кваліфікованих розробників програмного забезпечення пряме його застосування є доволі складним і трудомістким. Тому, слідуючи своїм давнім традиціям RAD (швидкого програмування), фірма Borland розробила власну технологію автоматизації процесів створення багатошарових інформаційних систем, відому під абревіатурою MIDAS[2] (Multi-Tier Distributed Application Services), що перекладається як “набір сервісів для багаторівневих розподілених застосувань”. Технологію MIDAS реалізовано як набір VCL компонент для Delphi та C++ Builder. З їхньою допомогою процедуру створення сервера застосувань зведено до традиційних візуальних маніпуляцій з компонентами із визначенням невеликої кількості їхніх властивостей. При цьому розробнику надається вибір компонент, що підтримують як DCOM, так і COBRA технології.

В основу технології MIDAS покладено концепцію модуля даних (TDataModule), котра дає змогу розділити функціональну логіку програми та логіку інтерфейсу користувача (такий модуль зображено на рис 1). У двошаровій моделі він не є обов’язковим і застосовується лише з міркувань зручності розташування невізуальних компонент. Однак у багатошаровій архітектурі MIDAS його значення сильно зростає. Він реалізується окремим проектом, тобто оформляється у вигляді *окремого виконуваного файла* (інколи dll-бібліотеки), який встановлюється на обчислювальний сервер, і відіграє ключову роль інтелектуального посередника між сервером БД та клієнтами. Відтепер серверу БД відведено роль

[1] Власною RMI-технологією та інструментарієм розподіленої обробки даних володіє також система програмування JAVA. Однак, зважаючи на її замкнутість і самодостатність, у нашій публікації вона не обговорюється.

[2] Починаючи з 6-ої версії Delphi, компоненти та об’єкти, що забезпечують розробку багатошарових застосувань, об’єднано спільною назвою DataSnap.

виконавця лише найелементар-ніших операцій з пошуку та вибірки даних, а клієнтські застосування переважно лише відображають на моніторі користувача інформацію, котра надається сервером застосувань.

Пояснимо сказане конкретизацією тришарової схеми AS-моделі клієнт-сервер, наведеної у ч. I. "Основні ідеї та положення" цієї серії методичних вказівок.

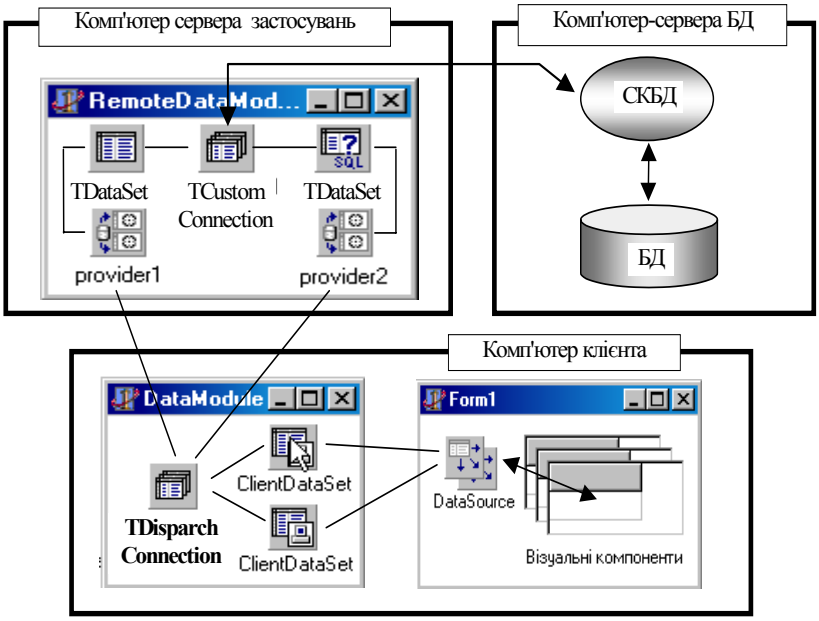

Рис. 2. Взаємодія компонент за технологією MIDAS

Відзначимо головні внесені позитивні зміни щодо наведеної на рис. 1 схеми двошарової моделі :

- модуль з компонентами доступу до БД став окремим віддаленим (RemoteDataModule) застосуванням і разом з відповідними засобами зв'язку з БД розташовується на окремому комп'ютері сервера застосувань.
- на комп'ютері клієнта більше немає потреби утримувати "масивну" клієнтську частину вибраної СКБД (наприклад, BDE).
- замість компонент зв'язку з БД та наборів даних, похідних від TDataSet, у клієнтському модулі даних залишились лише "заглушки" TClientDataSet, котрі через посередника TDispatchConnection (один з протоколів DCOM/Socket/WEB) з боку клієнта та TProvider з боку сервера отримують та належним чином представляють інформацію з БД.

Зауваження. *Встановлення окремих елементів багатошарової моделі на окремих комп'ютерах є зовсім не обов'язковим. Усі застосування можна розмістити і на одному комп'ютері, що особливо зручно на етапі розробки інформаційної системи.*

2. СЕРВЕР ЗАСТОСУВАНЬ

Усі запити клієнтів до БД попередньо потрапляють до сервера застосувань, який, за необхідності, перетворює їх у зручну для сервера БД форму і переправляє до СКБД. Результати виконання запиту на зворотному шляху також проходять подібну "цензуру". Отож з допомогою внутрішніх процедур сервера застосувань розробник системи має змогу потрібним чином впливати на процес проходження запитів від клієнтів і отриманих вибірок даних з сервера. Наприклад, у багатьох випадках виявляється, що необхідна клієнтові інформація у сервера застосувань вже є і немає змісту зайвий раз завантажувати ресурси сервера БД та мережі.

Звісно, що використання сервера застосувань не обмежується функціями інтелектуального посередника між клієнтами та БД. Він також є зручним місцем зберігання (і виконання) багатьох спільних для клієнтських застосувань процедур з підтримки бізнес-правил і забезпечення інших сервісів.

Загальні зауваження

Технологія MIDAS реалізації сервера застосувань здатна підтримувати ідеології DCOM та CORBA[3] (додатково є змога використовувати MTS, яка фактично є удосконаленням DCOM, та SOAP). Ці ідеології

[3] Починаючи з 7-ої версії Delphi підтримку технології CORBA призупинено, зате доповнено нові засоби підтримки технології SOAP.

проповідують різні підходи до вирішення багатьох проблем взаємодії серверів і клієнтів, таких як, наприклад, пошук потрібних алгоритмів і засобів їхнього виконання у загальній системі, рівномірного розподілення обчислювального навантаження між серверами застосувань, питання секретності тощо. У нашій архітектурно простій системі з одним-двома серверами застосувань такі питання не є надто актуальними, тому надалі використо-товуватимемо доступніший за наявністю потрібного програмного інструментарію (вбудовано у Windows) підхід DCOM.

За такого підходу MIDAS реалізує сервер застосувань у вигляді сервера Автоматизації. Тому, як і будь-який COM об'єкт, він автоматично розпочинає роботу на вимогу зв'язку від першого ж клієнта і самостійно закривається разом із завершенням зв'язку з останнім[4].

Складовими частинами сервера застосувань є:

- віддалений модуль даних[5];
- інтерфейсна форма(и) для зв'язку з адміністратором сервера;
- компоненти-провайдери.

Віддалений модуль даних є основною частиною сервера застосувань[6].

По-перше, подібно до звичайного модуля даних, він є контейнером для розміщення невізуальних компонент доступу до даних (похідні від TDataSet) та трансакцій, а також компонент з'єднання з сервером БД (похідні від TCustomConnection). Окрім того, для передачі даних клієнтам віддалений модуль даних містить компоненти TDataSetProvider, кожна з яких має бути пов'язана з відповідним набором даних. TDataSetProvider виконує підготовку (пакетування, шифрування та ін.) даних для їхньої передачі клієнтові через мережу, або ж розпаковує отриману від клієнта інформацію для внесення змін у потрібний набір даних.

По-друге, за рахунок надання клієнтам інтерфейсу IAppServer[7] (або його наслідника), віддалений модуль даних реалізує основні функції сервера застосувань.

Демонстраційний приклад

Розглянемо послідовно етапи процесу створення найпростішого сервера застосувань як окремої програми, котра виконує функції віддаленого модуля даних.

Відкриваємо новий проект (у головному меню Delphi/ C++ Builder вказуємо File->New->Application) і збережемо його під довільною назвою, наприклад, PAppServer. Форма, що з'явилась на екрані монітора, виконуватиме функції зв'язку між адміністратором і сервером застосувань (друга складова частина, див. п. 3.1). Решта компонент останнього будуть невізуальними. Далі створюємо віддалений модуль даних: File->New->Other. В отриманому вікні всеможливих пропонованих до створення об'єктів вибираємо закладку Multitier (її відсутність засвідчує, що при інсталяції Delphi/C++ Builder можливість побудови багатошарових застосувань не передбачено). Розміщені на цій сторінці компоненти відповідають основним технологіям DCOM та CORBA реалізації серверів застосувань. За оговореним у попередньому параграфі принципом зупинимося на виборі компонента Remote Data Module. Вибір супроводжується вимогою уточнення трьох його головних параметрів:

- CoClass Name – ім'я класу сервера, котре автоматично буде доповнено першою літерою "T" та занотовано у реєстрі Windows і, відповідно, надалі використовуватиметься усіма клієнтами для встановлення зв'язку із сервером застосувань. Для першого разу вкажіть, наприклад, MyFirstAppServer.
- Instancing.
  - Internal – модуль створюється як частина активної бібліотеки DLL (тобто забезпечується функціонування лише внутрішнього сервера автоматизації) і працює він у межах процесу сервера застосувань.
  - Single/Multiple – для кожного з'єднання з клієнтом утворюється власна копія віддаленого модуля даних. Усі модулі або ділять між собою єдиний процес на серверній машині (multiple), або ж запускаються у власних процесах (single).

    Для початку роботи найкраще підійде вибір Multiple.

- Threading Model.
  - Single – усі запити клієнта виконуються у єдиному потоці процесу, а, отже, послідовно.

[4] Нагадаємо, що реалізація такої можливості відбувається завдяки прописуванню властивостей COM сервера у системному реєстрі Windows у момент першого ж його запуску на комп'ютері.

[5] Один сервер застосувань може містити і декілька віддалених модулів даних, котрі, наприклад, виконують різні функції, або звертаються до різних серверів БД.

[6] Окрім сервера автоматизації Remote Data Module, системи Delphi/C++ Builder підтримують ще декілька різновидностей віддалених модулів даних: Transactioiial, SOAP, WebSnap, CORBA (закладки Multitier, WebSnap и WebServices палітри компонент) .

[7] Реалізацію методів інтерфейсу забезпечує динамічна бібліотека MIDAS.dll, котру небхідно зареєструвати на комп'ютері серевера застосувань.

- Apartment, Free, Both Neutral – різновиди багатопотоко-вого виконання запитів, завдяки яким можна значно збільшити ефективність роботи сервера. Проте для забезпечення безконфліктної паралельної роботи декількох потоків від програміста вимагається чимало зусиль і досвіду.

Найбезпечнішою (з огляду на безконфліктність корпоратив-ної роботи) є конфігурація Single.

Тепер новостворений модуль потрібно наповнити компонентами доступу до інформації в базі даних та зв'язку з клієнтськими застосуваннями. Історично склалось так, що на перших порах з цією метою використовували компоненти TDataBase, котрі застосовують BDE засоби доступу до БД. Проте сьогодні з цією метою використовують універсальні TSQLConnection та TADOConnection, або спеціалізовану TIBDataBase. Вибір компонент доступу до БД залежить від багатьох чинників, зокрема від обраної СКБД. Досвід авторів дає підставу сьогодні рекомендувати технологію ADO для СКБД MS SQL, Oracle, Sybase не через меншу ефективність інших технологій (зокрема, Borland), а з огляду на сучасні тенденції ринку програмного забезпечення і пов'язані з ними перспективи подальшої підтримки та розвитку цих технологій.

Для визначеності зупинимось на TSQLConnection. Для її роботи мінімально необхідно зазначити такі властивості:

- ConnectionName – назва конфігурації з'єднання, котра визначається користувачем (вкажіть, наприклад, My_IB_Con).
- DriverName – драйвер для зв'язку з потрібною СКБД *(оберемо, наприклад, Interbase).*
- Params – набір параметрів зв'язку, котрий залежить від типу обраної СКБД, і у якому обов'язковими є лише пари:
  DataBase = < шлях до файла бази даних >.
  User_Name = < ім'я зареєстрованого в СКБД користувача бази даних > (SYSDBA по замовчуванню для Interbase).
  Password = < його пароль > (masterkey для SYSDBA).

Важливо знати, що всі ці властивості можна динамічно визначати під час роботи процедур сервера застосувань безпосередньо перед встановленням з'єднання з БД (Connected = true). Це означає, що одна і та ж компонента здатна почергово підтримувати зв'язок з різними базами даних, і навіть з тими, котрі керуються різними СКБД.

Згадана властивість DataBase, окрім вказування місцезна-ходження файла БД, додатково визначає протокол мережі, за яким спілкуватимуться сервери застосувань і бази даних. Наприклад, для застосування внутрішнього (для WINDOWS) протоколу мережі – NETBEUI шлях до БД необхідно позначати так:

\\ < ім'я комп'ютера > \ < повний шлях до файла >, наприклад,
\\ComputerDBServer\c:\...\DBDir\Simple.gdb.

Зауважимо, що тут під поняттям < повний шлях до файла > розуміємо не відданий під деяким іменем ресурс мережі, а фізичне розташування файла БД, починаючи від кореневого каталогу.

Для зв'язку ж за TCP/IP протоколом згадана вище стрічка має виглядати так:
ComputerDBServer : c:\...\DBDir\Simple.gdb.

Зв'язок за протоколом TCP/IP є дещо універсальнішим, бо дає змогу замість доменного імені комп'ютера вказувати його TCP/IP адресу, що часто допомагає встановити доступ до нього навіть у погано сконфігурованих багатодоменних мережних системах. Утім розміщувати працюючі бази даних "далеко" (в сенсі простоти доступу) від сервера застосувань із зрозумілих причин не рекомендовано. Коректність налаштування властивостей компоненти TSQLConnection проконтролюйте успішністю з'єднання з БД (Connected = true). Причиною відсутності зв'язку за правильного визначення місце-знаходження файла БД може бути непрацюючий у даний момент сервер БД, або проблеми з'єднання у мережі між комп'ютерами, які легко виявляються з допомогою системної утиліти PING.

Компонента TSQLConnection встановлює лише канал зв'язку з сервером БД. Для формування ж запитів на вибірку даних у віддаленому модулі даних необхідно мати компоненти набори даних, похідні від класу TDataSet. Їх можна знайти на закладці dbExpress палітри компонент Delphi/C++ Builder. Для прикладу перенесемо в наш модуль даних компоненту TSQLQuery. Після подачі запиту серверу БД (через TSQLConnection) вона отримує результуючі дані вибірки з БД. Для перевірки її роботи вкажіть у властивості SQL довільний коректний запит на вибірку даних, після чого властивість Active спробуйте перевести у значення True. Успішність останньої операції засвідчує працездатність усіх ланок у ланцюжку доступу до інформації в БД. Тепер, за потреби, будь-яка процедура сервера застосувань може потрібним чином використати інформацію в TSQLQuery. Залишилось тільки зробити її доступною для клієнтських програм, котрі функціонують десь у мережі. Для зв'язку з клієнтськими застосуваннями кожна TSQLQuery (чи інша TDataSet) повинна бути з'єднаною з власним провайдером (TDataSetProvider на закладці Data Access) (див. рис. 2). Отже, для обслуговування кожного окремого запиту до БД потрібно мати відповідну пару TDataSet + TDataSetProvider. Проте, на щастя, велика різноманітність запитів зовсім не означає необхідності утримувати у віддаленому модулі даних таке ж велике число згаданих пар завдяки можливості динамічного налаштування властивості SQL компоненти TSQLQuery. Особливістю такого налаштування є

те, що його можуть виконувати не тільки процедури самого сервера застосувань, а й набори даних з програм клієнтів, передаючи текст потрібного запиту через TDataSetProvider. Зауважимо, що за замовчуванням (з огляду на безпеку доступу до даних) клієнтські застосування не мають права змінювати згадану властивість. Надання такого права відбувається переведенням у TRUE параметра poAllowCommandText властивості Options відповідного провайдера.

За наявності успішного зв'язку з інформацією в БД скомпілюйте проект сервера застосувань і бодай один раз запустіть отриманий *.exe файл на виконання на тому комп'ютері, де передбачається його подальша робота, для реєстрації отриманого COM сервера в операційній системі Windows.

3. РОЗРОБКА КЛІЄНТСЬКОГО ЗАБЕЗПЕЧЕННЯ

Розробка клієнтського застосування у багатошаровій моделі, на перший погляд, не особливо й відрізняється від двошарової. Окрім того, чисто формально, навіть вже готовий програмний продукт досить легко перевести на тришарову ідеологію. Інша справа, що користі від такого переводу буде небагато. Реальний же перехід вимагає серйозного перепланування структури програмного забезпечення з перенесенням центру ваги реалізації бізнес-правил на сервер застосувань. Втім, справа ця хоч і кропітка, але зовсім не безнадійна, якщо від початку клієнт розроблявся з використанням об'єктного підходу.

Спершу розглянемо формальні засади організації взаємодії між клієнтом і сервером застосувань.

Зв'язок клієнта із сервером застосувань

Як уже згадувалось вище, сервер застосувань перебирає на себе значну частку обчислювального навантаження клієнта, залишивши йому такі функції :

- з'єднання з сервером застосувань, приймання та передача даних;
- робота з локальними копіями даних;
- відображення інформації засобами інтерфейсу користувача;
- найпростіші операції редагування.

Перша із перелічених функцій підтримується набором компонент із закладки DataSnap. Ці компоненти інкапсулюють стандартні транспортні протоколи обміну даними між сервером і клієнтами DCOM (TDCOMConnection), HTTP (TWebConnection) та сокети (TSocketConnection) і забезпечують доступ до функцій сервера застосувань за рахунок використання інтерфейсу IAppServer[8]. Коротко охарактеризуємо кожну з них.

TDCOMConnection використовує розроблений фірмою Microsoft DCOM протокол обміну даними, котрий використовуть у Windows, починаючи з 98-ої версії. Інтегрованість DCOM в операційну систему забезпечує максимальну ефективність зв'язку сервера з клієнтом. Додатковою перевагою є вбудована підтримка моделі безпеки Windows NT.

Проте зазначена технологія добре працює лише в межах правильно сконфігурованих локальних доменних Windows–мереж. А для використання служби безпеки сервер застосувань має знаходитись на комп'ютері контролера домену. Відсутність DCOM з'єднання компенсують застосуванням компоненти TSocketConnection, котра для зв'язку сервера з клієнтом використовує протокол TCP / IP на основі виклику низькорівневих API-функцій Windows - сокетів[9]. Такий зв'язок забезпечує максимальну швидкість обміну даними та дає змогу використовувати мережу Internet (у тім числі через модемне з'єднання).

Особливістю технології сокетів є відсутність захисту даних у мережі від несанкціонованого доступу. Потенційно будь-яке клієнтське програмне забезпечення, знаючи ім'я класу сервера застосувань та ім'я провайдера у віддаленому модулі даних, може підключитись і працювати з ним без будь-якого контролю за правами доступу. Тому вирішення проблеми безпеки даних у мережі цілком покладається на аутентифікацію користувачів програмними засобами сервера застосувань. З іншого боку, саме відсутність згаданого контролю мінімізує вимоги до програмного оточення інформаційної системи та значно спрощує процес налагодження зв'язку між сервером і клієнтами при її інсталяції (розгортанні). З цієї ж причини технологію сокетів нижче використано у демонстраційному прикладі.

Крім NETBEUI та TCP/IP протоколу, MIDAS технологія дає змогу встановлювати та підтримувати зв'язок між віддаленим сервером застосувань та клієнтами за допомогою протоколу HTTP, для чого використовується компонента TWebConnection[10]. У цьому випадку вважають, що сам сервер реалізовано засобами ActiveX Page і експонується він наявним у мережі Web-сервером з допомогою спеціалізованої

[8] З'єднання з сервером застосувань забезпечує динамічна бібліотека MIDAS.dll, котра має бути зареєстрованою на комп'ютері клієнта

[9] Для використання технології сокетів на комп'ютері сервера застосувань попередньо має бути стартована утиліта SCKRTSRVR.exe (її можна знайти в підкаталозі BIN директорії DELPHI).

[10] На машині сервера в системному каталозі має бути бібліотека HTTPSRVR.dll (входить в комплект DELPHI в директорії BIN).

динамічної бібліотеки HTTPsrvr.dll. Тобто замість прямого звертання до віддаленого модуля даних клієнт безпосередньо зв'язується з HTTPsrvr.dll (за аналогією з SCKTsrvr.exe в технології з сокетами), котра надалі приймає від нього всі запити і самостійно працює з сервером застосувань (з використанням тепер вже COM технології).

Головною особливістю такого підходу є можливість реалізації справді "тонкого" міжплатформенного клієнта (наприклад, у вигляді WEB сторінки), оскільки технології DCOM та сокети передбачають наявність Windows платформи. Важливою перевагою Web-з'єднання (особливо перед сокетами) є можливість використання засобів системи безпеки SSL, котра забезпечується бібліотекою клієнта wininet.dll. TWebConnection дає змогу створювати клієнтів, які зв'язуються з сервером застосувань через захисні засоби *firewall*, використовуючи найпоширеніший порт зв'язку. Недоліком такої технології є текстовий формат даних протоколу HTTP, котрий за необхідності інтенсивного обміну інформацією між сервером застосувань і клієнтом є порівняно повільним.

Кожна з описаних вище компонент забезпечує лише канал зв'язку клієнта з сервером. Для організації ж запитів по інформацію до БД та отримання і зберігання результуючої вибірки даних (друга з перелічених вище функцій) клієнтську програму необхідно доповнити компонентою TClientDataSet із закладки Data Access.

Роль цієї компоненти для розробки багатошарових застосувань важко переоцінити. Вона з успіхом замінює раніше використовувані TQuery та їй подібні нащадки від TDataSet вже тим, що не використовує громіздку BDE. Проте головною перевагою TClientDataSet є те, що результат запиту вона закачує у власний буфер і подальша робота йде вже з локальною копією обраного фрагмента бази даних. Зрозуміло, що цю властивість необхідно використовувати обережно, не допускаючи надмірних розмірів вибірок (і, відповідно, вимог до оперативної пам'яті) засобами SQL команди. Втім, за потреби оперування великими обсягами результуючих даних, програмісту надається цілковитий контроль над кількістю записів, які буде запаковано провайдером у пакет даних. Цю кількість необхідно вказати у властивості PacketRecords клієнтського набору даних (за замовчуванням воно встановлене в -1, що дає вказівку провайдеру запаковувати в пакет даних всі доступні записи). Крім того, властивості FetchOnDemand потрібно надати значення True. З цими установками TClientDataSet вибиратиме записи тільки тоді, коли вони знадобляться, наприклад, при переході за останній прийнятий клієнтським набором даних запис.

Додатковою цікавою особливістю компоненти TClientDataSet є можливість організації відкладеної обробки даних (технологія портфеля *briefcase*) з допомогою методів SaveToFile та Load FromFile. Суть її полягає у тому, що клієнт може розірвати зв'язок із сервером застосувань і потрібний час працювати в автономному режимі. Усі зміни до БД компонента здатна зберігати у власних файлах на комп'ютері клієнта і реалізувати їх у момент наступного сеансу підключення до сервера застосувань.

Окрім нових компонент для зв'язку із сервером застосувань у клієнтському застосуванні для відображення даних використовують стандартні компоненти TDBGrid та ін., а також звичні схеми зв'язування візуальних компонент з набором даних через TDataSource.

Демонстраційний приклад

Створимо нове застосування File->New->Application і модуль даних у ньому File->New->Data Module. Для зв'язку із сервером застосувань у модуль даних залучаємо, наприклад, TSocketConnection, розміщену на закладці DataSnap палітри компонент. У її властивості Host вказуємо доменне ім'я або IP-адресу комп'ютера, на якому працюватиме створений нами у п. 3.2 сервер застосувань (за замовчуванням приймається, що і він, і клієнт розташовані на одному комп'ютері). Тепер у властивості ServerName необхідно віднайти ім'я цього сервера, зазначене у полі CoClassName при його створенні (вище запропоновано MyFirstAppServer). Переведенням властивості Connected у True перевірте наявність зв'язку із сервером застосувань. Про це свідчитиме його автоматичний старт, якщо тільки він не був запущеним раніше. За відсутності зв'язку перевірте наявність працюючої утиліти SCKTSRVR.exe. Якщо сервер застосувань розміщений на окремому комп'ютері, перевірте наявність доступу до нього у мережі за допомогою утиліти PING.

Після визначення каналу зв'язку у модуль даних додайте компоненту TClientDataSet (закладка Data Access). Для налаштування цієї компоненти досить спрямувати її властивість RemoteServer на TSocketConnection та уточнити у властивості ProviderName ім'я компоненти провайдера на віддаленому модулі даних сервера застосувань. Щоб перевірити доступ до інформації в БД, вкажіть довільний коректний запит у властивості CommandText та переведіть Active у True.

Для представлення даних з буфера компоненти TClientDataSet на формі клієнтського застосування організуйте класичну зв'язку TClientDataSet -> TDataSource -> TDBGrid (або інші компоненти із закладки Data Controls палітри компоненти). За правильного налаштування всіх ланок ланцюжка доступу до БД в таблиці TDBGrid буде зображено вибірку, яка є результатом запиту, визначеного властивістю CommandText компоненти TClientDataSet. Ще раз наголосимо, що під час роботи програми, динамічно змінюючи згадану властивість, можна отримати різноманітну інформацію з БД за допомогою навіть однієї компоненти.

4. РОЗШИРЕННЯ ІНТЕРФЕЙСУ СЕРВЕРА ЗАСТОСУВАНЬ

У попередніх параграфах описано класичну схему проходження інформації у тришаровій архітектурі клієнт-сервер :

- формування запиту клієнтом і передача його серверу застосувань;
- аналіз і корекція запиту сервером застосувань і передача його серверу БД;
- виконання запиту сервером БД, формування результуючої вибірки даних і повернення її серверу застосувань;
- аналіз і корегування отриманої вибірки сервером застосувань і пересилання остаточного набору даних клієнтові.

Підтримку перелічених операцій з боку сервера застосувань (як COM сервера) реалізовано у методах його стандартного інтерфейсу IAppServer. Саме до цих методів звертаються компоненти T**xxx**Connection та TClientDataSet для посилання запитів та отриму-вання результуючих наборів даних. Однак розробник сервера застосувань має змогу доповнити його (розширити) будь-яким власним набором процедур обробки даних як внутрішніх, так і отриманих від клієнта, чи з БД. Зокрема, це можуть бути алгоритми бізнес-правил, не пов'язані з інформацією в БД (правила заокруглення сум, нарахування податків тощо). У цьому ж місці зручно розташувати процедури синхронізації виконання окремих технологічних операцій декількома клієнтами і т.п.

Залишається лише реалізувати доступ до цих процедур з боку клієнтської програми, зареєструвавши їх як розширення стандартного інтерфейсу IAppServer. За правилами створення ActiveX об'єктів, оболонка Delphi/C++ Builder у момент компіляції проекту сервера застосувань вносить описи всіх його інтерфейсів у спеціальну бібліотеку типів, котра має ту ж назву, що і проект, і розширення *.tlb. Важливим для розробника є наявність зручного редактора роботи з бібліотеками типів, включеного в оболонку Delphi/C++ Builder. Його роботу розглянемо на простому прикладі.

Задамося ціллю обчислити середнє арифметичне двох цілих чисел не в клієнтській програмі, а саме на сервері застосувань. Відкриємо ще раз створений у п. 3.2 проект PAppServer. Серед модулів проекту знаходимо PAppServer.tlb і засобами програмної оболонки відкриваємо його форму, реалізовану у вигляді редактора бібліотеки типів. За його допомогою до інтерфейсу IMyFirstAppServer додаємо "New Method" і назвемо його, наприклад, myAverage. На закладці Params додаємо два вхідних (type=int, modifier=in) параметри P1 і P2 та вихідний параметр результату AVG (type=float, modifier=out). Для автоматичного формування інтерфейсної частини майбутньої процедури знайдіть і натисніть кнопку Refresh Implementation. В результаті новий метод буде не тільки додано до інтерфейсу нашого сервера, а й у віддаленому модулі даних MyFirstAppServer з'явиться заготовка нової процедури з іменем myAverage і визначеними нами параметрами P1, P2 та AVG. Залишилось лише заповнити цю процедуру кодом, що присвоює змінній AVG півсуму вхідних параметрів P1 і P2.

Можливий і зворотний процес вводу у список інтерфейсів існуючої процедури з описом у секції public. Для цього з допомогою опції Add to Interface контекстного меню редактора коду викликають діалогову форму, в якій потрібно навести оголошення процедури. Наприкінці оголошення доцільно задати тип виклику (stdcall).

Аналогічно інтерфейс сервера застосувань можна розширити довільним необхідним числом процедур. Якщо ж, окрім того, взяти до уваги, що їхніми параметрами можуть бути змінні типу OLEVariant, з допомогою яких є змога передавати динамічні масиви різноманітних типів даних, то можливості спілкування сервера з клієнтами справді є достатньо широкими.

Залишилось лише визначити спосіб виклику нової процедури сервера в клієнтській програмі. Зробимо це легко з допомогою властивості AppServer компоненти SocketConnection у модулі даних клієнта:

```
SocketConnection.AppServer.myAverage(5,7,myVarAVG);     //Delphi.
SocketConnection->AppServer->myAverage(5,7,myVarAVG) //CBuilder.
```

Зауважимо, що на етапі компіляції такого виклику процедури інтерфейс сервера застосувань компілятору невідомий, тому останній змушений приймати таке звертання "на віру", тобто без перевірки як наявності самої процедури, так і кількості та типів її параметрів. Цей спосіб доступу до інтерфейсу сервера називають пізнім (динамічним) зв'язуванням. Залучення бібліотеки типів сервера застосувань у проект клієнта дає змогу виконати згадані дії ще на етапі компіляції і оптимізувати набір операцій з виклику процедури.

5. РОЗГОРТАННЯ СИСТЕМИ

За умови розробки і тестування сервера застосувань і клієнтського програмного забезпечення на одному комп'ютері проблем з їхньою взаємодією здебільшого не виникає. Підставою для цього є наявність повного комплекту програмних засобів підтримки технології MIDAS, яка гарантується інсталяцією середовищ Delphi/C++ Builder на цьому комп'ютері. До того ж відсутні будь-які проблеми, пов'язані з корпоративною роботою елементів інформа-ційної системи у мережі. Перенесення останньої в умови

реальної експлуатації нерідко пов'язане з низкою труднощів, котрі передусім виникають з двох причин: відсутності необхідних зовнішніх dll-бібліотек і проблем з правами доступу клієнтів до ресурсів сервера застосувань. Отож цей процес розглянемо покроково.

1. Першим етапом розгортання готової інформаційної системи у замовника є, звісно, інсталяція обраного сервера БД та копіювання або створення нової БД. Наголосимо тут лише на необхідності узгодження списків (і паролів) користувачів інформацією з боку серверів застосувань та БД.
2. Наступним кроком на визначений комп'ютер встановлюємо програмне забезпечення сервера застосувань і конфігуруємо його оточення:
   - У випадку встановлення сервера застосувань на відмінний від сервера БД комп'ютер інсталюйте на ньому клієнтську частину обраної СКБД.
   - Для підтримки роботи компонент TClientDataSet забезпечте наявність бібліотеки MIDAS.dll. Краще розмістити її у системній директорії Windows, щоб уникнути проблем з конфліктністю інформації у системному реєстрі за потреби перенесення сервера застосувань в інший каталог.
   - Якщо для зв'язку з сервером БД у віддаленому модулі даних використано компоненту dbExpress, то в його оператор USES необхідно добавити модуль Crtl з драйверами доступу до вибраної СКБД. У протилежному випадку системну директорію Windows необхідно доповнити однією із зовнішніх бібліотек залежно від використовуваного SQL сервера: для MySQL – dbExpMy.dll, для InterBase – dbExpInt.dll, для Oracle – dbExpOra.dll, для DB2 - dbExpDb2.dll.
   - Перший раз запустіть сервер застосувань засобами Windows для реєстрації його як COM-сервера у реєстрі операційної системи.
   - Наступні налаштування мають за мету уникнути конфліктів клієнта з підсистемою безпеки Windows щодо прав запуску COM-сервера застосувань на виконання. За умов вибору наступного транспортного протоколу між сервером застосу-вань та клієнтами:
     - DCOM [11]:

       за допомогою системної утиліти DCOMCNFG.exe сконфігуруйте права користувачів на роботу з сервером застосувань[12], для чого:

       - на закладці Applications у списку COM-серверів знайдіть власний зареєстрований сервер застосувань і натисніть кнопку Propeties;
       - на закладці Security визначте список користувачів, котрі мають доступ до вашого сервера застосувань, та вкажіть права його запуску на виконання (для першого разу, не переймаючись питаннями безпеки, найпростіше вибра-ти Everyone – "будь-хто");
       - на закладці Identify вкажіть, від імені (облікового за-пису - account) якого користувача стартуватиме сервер. Якщо цей account збігатиметься з обліковим записом поточного користувача комп'ютера, то візуальну форму сервера застосувань буде відображено на екрані монітора (і заховано в протилежному випадку). Тому для тестування роботи системи вигідно вибрати опцію Interactive user. Однак зауважимо, що при своєму старті MIDAS-сервер намагатиметься модифікувати реєстр Windows. Отож якщо поточний користувач комп'ютера, на якому функціонує сервер застосувань, прав на внесення таких змін не має, то виберіть опцію This user і вкажіть account користувача з відповідними правами (наприклад, одного з адміністраторів домену).

     - Сокети:
       - Забезпечіть наявність і запуск Borland сокет-сервера SCKTSRVR.exe перед встановленням зв'язку із сервером застосувань. Це можна організувати, наприклад, з допомогою каталогу автозапуску Windows. Як альтернативний варіант, стартуйте SCKSTRVR.exe один раз з параметром /Install, після чого він надалі працюватиме як системний сервіс.
     - HTTP:
       - На комп'ютері сервера застосувань має бути інстальовано (і запущено на момент зв'язку з клієнтом) Internet Information Server версії не нижче 4.0 або Netscape Enterprise версії не нижче 3.6.
       - Встановіть бібліотеку HTTPsrvr.dll на WEB-сервері комп'ютера, до якої звертатиметься компонента TWEBConnection клієнта по зв'язок із сервером застосувань. Фактично вона є ISAPI DLL і має бути розміщена в логічному каталозі Scripts Web-сервера. Крім того, не забудьте саме цей каталог використати для визначення властивості URL компоненти TWEBConnection у програмному забезпеченні клієнта.

[11] За такого способу доступу до сервера необхідно, щоб усі клієнти і сам сервер застосувань функціонували в межах одного домена.

[12] Для внесення змін у конфігурацію DCOM користувач системи повинен мати права адміністратора.

- У системній директорії клієнтської машини для підтримки SSL-технології підсистеми безпеки обов'язкова наявність бібліотеки wininet.dll (що автоматично гарантується інсталяцією Internet Explorer 3 або вищої версії).

3. Для інсталяції клієнтського програмного забезпечення крім копіювання виконуваних файлів забезпечіть наявність бібліотеки MIDAS.dll у системній директорії Windows (для нижчих від Delphi 5 версій, DBClient.dll).

6. ЗАКЛЮЧНІ ЗАУВАЖЕННЯ

Технологія MIDAS поєднала у собі складну в реалізації ідеологію багатошарової розподіленої обробки даних і RAD-технологію швидкої розробки програмних систем. Результатом такого симбіозу є компактний набір зручних засобів створення масштабних інформаційних систем, надійність та ефективність яких перевірена практикою.

Втім, застосування будь-якого інструментарію повинно відповідати умовам вихідної задачі. Отож завершимо роботу такими зауваженнями:

1. Використання багатошарової архітектури не повинно бути самоціллю чи наслідуванням модних тенденцій, а мусить мати достатнє обґрунтування її необхідності : значні обсяги обчислень з підтримки бізнес-правил, велика кількість підключень клієнтських застосувань, складна структура системи з розподіленою обробкою даних тощо. Для невеликих простих систем двошарові моделі у багатьох випадках залишаються ефективнішими і зручнішими у використанні.
2. Багато проектів лише на перших порах не вимагають тришарової архітектури. Проте з часом, унаслідок збільшення обсягів даних, зростають вимоги користувачів до функціональності і різноплановості клієнтських програм (Windows-застосування, HTML-сторінки тощо). Тому навіть початково просту інформаційну систему вже на етапі проектування варто розділити на частини представлення даних, бізнес-логіки та збереження даних. При цьому віддалений модуль TRemoteDataModule на перших порах можна зімітувати з допомогою TDataModule, компоненту зв'язку – використовуючи TLocalConnection і т.п. Додаткові зусилля на супровід такого підходу надалі окупляться мінімальними затратами при переході на тришарову архітектуру.
3. Застосування RAD-технологій загалом зменшує затрати праці для виготовлення кінцевого програмного продукту, проте не зменшує рівня кваліфікаційних вимог до розуміння внутрішніх механізмів ідеології розподіленої обробки даних. Видима простота використання засобів MIDAS має достатньо "підводних каменів", здатних звести нанівець їхню ефектив-ність.
4. Матеріал цієї праці не претендує на вичерпність опису як самої технології MIDAS, так і способів її застосування. Запропоновано лише стартові вказівки для швидкого початку її використання та висвітлено окремі "вузькі" місця, обхід котрих у початківця може забрати невиправдано багато часу.

СПИСОК ЛІТЕРАТУРИ

1. *Васкевич Д.* Стратегии клиент/сервер. Руководство по выживанию для специалистов по реорганизации бизнеса – К.: Диалектика, 1996. – 384 с.
2. *Елманова Н., Трепалин С.* Delphi 4: технология COM, OLE, ActiveX, Автоматизация MIDAS, Microsoft Transaction Server – М.: ДИАЛОГ-МИФИ, 1999, – 320 с.
3. *Фараонов В., Шумаков П.* Delphi 5. Руководство разработчика баз данних. – М.: ”Нолидж”, 2000. – 640с.: ил.
4. *Хармон Э.* Разработка COM-приложений в среде Delphi: Пер. с англ.: Уч. пос. – М.:Издательський дом “Вильямс”, 2000. –464 с.: ил.

Internet ресурси

1. http://venera.work.kemsu.ru/Delphi/06/Index.shtml
2. http://www.interface.ru/magazine/tcs/Archive/298/midas2.htm
3. http://olegmotov.h1.ru/articles/inprisemidas/MiltiTierDevelopmentMIDAS.htm
4. http://www.compress.ru/Temp/3292/index.htm
5. http://podgoretsky.com/ftp/Docs
6. http://infocity.kiev.ua/prog
7. http://www.cps.ru/vendors/borland/midas.shtml
8. http://docs.h1.ru/deldocs/d5_midas/index.html
9. http://www.3ka.mipt.ru/vlib/citforum/programming/distr/distr02_4.shtml
10. http://www.interface.ru/fset.asp?Url=/borland/bas451.htm
11. http://www.softforum.ru/html/index.asp?id=items&group=cps.borland.delphi&topic=item121101