МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Східноєвропейський національний університет імені Лесі Українки
Кафедра прикладної математики та інформатики

В. В. Булатецький, Л. В. Булатецька

ТЕХНОЛОГІЇ ПРОМІЖНОГО КОДУ В КОРПОРАТИВНИХ ІНФОРМАЦІЙНИХ СИСТЕМАХ

Текст лекцій
нормативної навчальної дисципліни
“Платформи корпоративних
інформаційних систем”

Луцьк-2018

УДК 004.655
Б 90

Рекомендовано до друку науково-методичною радою
Східноєвропейського національного університету імені Лесі Українки
(протокол № 9 від 20 червня 2018 р.)

Рецензенти:

Федонюк А. А. – кандидат фізико-математичних наук, доцент, завідувач кафедри вищої математики та інформатики Східноєвропейського національного університету імені Лесі Українки;

Пех П. А. – кандидат технічних наук, доцент, завідувач кафедри комп'ютерної інженерії Луцького національного технічного університету

Б 90 Технології проміжного коду в корпоративних інформаційних системах : *Текст лекцій нормативної навчальної дисципліни "Платформи корпоративних інформаційних систем" / Булатецький Віталій Вікторович, Булатецька Леся Віталіївна*. – Луцьк : Східноєвропейський національний університет імені Лесі Українки, 2018. – 48 с.

Текст лекцій призначений для студентів напряму підготовки 122 Комп'ютерні науки та є логічним доповненням до лекційного курсу нормативної навчальної дисципліни "Платформи корпоративних інформаційних систем". Призначений для оволодіння теоретичними та практичними навиками з проектування, розробки, використання сучасних платформ Java і .NET. В розробці подано тексти лекцій. Рекомендовано викладачам та студентам, які викладають або вивчають сучасні методи програмування у вищих навчальних закладах.

УДК 004.655

ВСТУП

Предметом вивчення навчальної дисципліни є базові визначення та поняття корпоративних систем та основні платформи їх розробки, а саме основні їх можливості, особливості програмування, та їх багаторівневі архітектури.

Метою викладання навчальної дисципліни "Платформи корпоративних інформаційних систем" – сформувати у слухачів знання, вміння та навички, необхідні для ефективного використання засобів розробки сучасних корпоративних інформаційних систем у своїй майбутній професійній діяльності. Формування у слухачів знань, вмінь та навичок з проектування, розробки, використання сучасних платформ Java і .NET.

Згідно з вимогами освітньо-професійної програми студенти повинні знати методи побудови та проектування корпоративних систем; технології розробки корпоративних інформаційних систем; сучасні напрями досліджень в області корпоративних інформаційних систем та вміти визначати апаратну платформу та програмне середовище, що відповідають обраній архітектурі; проектувати інформаційні веб-ресурси з інтеграцією зовнішніх даних і програмних продуктів за допомогою технологій Java та .NET. Пропонований текст лекцій підготовлено відповідно до нормативної програми підготовки бакалавра галузі знань 12 Інформаційні технології, спеціальності 122 Комп'ютерні науки, освітньої програми Комп'ютерні науки та інформаційні технології. Згідно програми навчальної дисципліни в даній розробці висвітлено загальні характеристики корпоративних інформаційних систем (КІС) та платформ їх розробки, розглянуто багаторівневі архітектури корпоративних інформаційних систем та загальні поняття Java і .NET. Весь матеріал подано у вигляді лекцій. Вкінці кожної лекції подано список літературних джерел.

Лекція 1

ПОНЯТТЯ КІС, ХАРАКТЕРИСТИКИ, ОЗНАКИ, ВИМОГИ. ОГЛЯД ІСНУЮЧИХ КІС

Корпоративна інформаційна система (КІС) – управлінська ідеологія, що поєднує бізнес-стратегію підприємства (з вбудованою для її реалізації структурою) і передові інформаційні технології.

Прийнято вважати, що головну увагу при цьому приділяють відпрацьованій структурі керування, що складає функціональну частину підприємства, а автоматизація виконує другорядну, інструментальну роль.

Корпоративна ІС є інформаційно-керуючою системою, що включає бізнес-архітектуру підприємства, його персонал, ІТ-архітектуру та є діючою частиною кіберкорпорації.

«Корпоративність» у терміні КІС означає відповідність системи вимогам великої фірми, що має складну структуру, велику кількість взаємодіючих компонентів з ієрархічністю підпорядкування цілей їх діяльності загальній меті усієї системи. Інформаційні системи окремих підрозділів фірми (фінансових, економічних, маркетингових та ін.) не можуть претендувати на корпоративні. Тільки повнофункціональна система може дійсно бути охарактеризована як КІС.

Корпоративні інформаційні системи є розвитком систем для робочих груп, орієнтованих на масштаб підприємства, які можуть підтримувати територіально рознесені вузли та мережі та має ієрархічну структуру з декількох рівнів.

КІС, окрім забезпечення доступу до інформаційного фонду робочої групи, забезпечує доступ з будь-якого підрозділу до централізованої або розподіленої бази даних підприємства (організації). Для таких систем характерна архітектура «клієнт-сервер» зі спеціалізацією серверів.

Характерні ознаки КІС:

- тривалий життєвий цикл; (більший ніж час використання апаратних засобів)
- міграція успадкованих систем;
- різноманітність використовуваного апаратного забезпечення, життєвий цикл якого менший, ніж життєвий цикл створюваної системи;
- різноманітність використовуваного програмного забезпечення;
- масштабованість та складність розв’язуваних задач;
- перетин множини різних предметних сфер;
- орієнтація на аналітичну обробку даних;
- територіальна розподіленість;
- використання корпоративних SQL-серверів БД (Oracle і Informix-OnLine, Informix-DSA, Sybase, CA-Ingress та ін.) та відповідних інструментальних засобів (окрім власних засобів розробки також

використовуються незалежні багатоплатформні інструментальні засоби, доповнені інтерфейсами, драйверами та шлюзами для зв'язку з різними СУБД);
- використання Web-технологій;
- підвищені вимоги до надійності функціонування та збереження даних.

Ефективне управління сучасною організацією сьогодні є досить нетривіальним завданням, з огляду на різноманіття використовуваних ресурсів і високу швидкість зміни операційного оточення. Основними функціями управління є, як відомо, **планування, організація, активізація, координація, контроль й аналіз**, які здійснюються в багатомірному просторі різних областей діяльності підприємства. Сформовані управлінські рішення служать відправним моментом для конкретних виконавців. У зв'язку з тим, що автоматизація виконання посадових обов'язків й окремих доручень фактично стала останнім часом стандартом де-факто, особливу гостроту набуває проблема автоматизації безпосередніх управлінських функцій. Таким чином, найбільш істотною рисою комплексної інформаційної системи має стати розширення контуру автоматизації для одержання замкнутої, саморегулюючої системи, здатної гнучко й оперативно перебудовувати принципи свого функціонування.

До складу КІС повинні ввійти засоби документаційного забезпечення управління, інформаційної підтримки предметних сфер, комунікаційне програмне забезпечення, засоби організації колективної роботи співробітників й інші допоміжні (технологічні) продукти. Очевидно, що характерною особливістю КІС є *інтеграція великої кількості програмних продуктів*. Подібна широко-профільна система має максимально рівноправно та толерантно задовольнити всі підрозділи організації, по можливості зберегти існуючі бізнес-процеси, а також методи й структуру управління. Зрозуміло, що без залучення автоматизації практично не можна контролювати постійно мінливі бізнес-процеси.

I. **Масштабованість** – одна із важливих характеристик, що враховує масштаби діяльності корпорації. Вона повинна функціонувати на масштабній програмно-апаратній платформі (сервери, операційні системи, системи комунікації, СКБД), що потребує значних зусиль спеціалістів з проектування й впровадження.

Оскільки варіантів конфігурації базового устаткування і програмного забезпечення може бути багато, то КІС має бути багатоплатформною.

II. **Багатоплатформність** проявляється в тому, що прикладна програма може працювати на кількох платформах. При цьому забезпечуються однакові інтерфейс і логіка роботи на всіх платформах (тобто, подібність схем екрана, елементів меню і діалогової інформації, що надається користувачеві різними платформами; інтегрованість з користувацьким операційним середовищем; однакова поведінка на різних платформах; узгоджена підтримка незалежно від платформи тощо). Реалізувати прикладну програму одночасно в кількох середовищах нелегко. Тому з'явилися інтегровані програмні середовища

розробки (frameworks), які значно полегшують перенесення прикладних програм між різними середовищами. До них належать Windows Open Services Architecture (WOSA); Win 32, загальне відкрите програмне середовище UNIX COSE (Common Open Software Environment), AppWare Foundation та інші.

III. **Неоднорідність** обчислювального середовища проявляється можливістю одночасної роботи в мережах, до яких входять комп'ютери, що працюють під управлінням різних операційних систем або побудовані на різних обчислювальних платформах. При цьому забезпечується взаємодія всіх використовуваних робочих обчислювальних платформ і операційних систем.

IV. **Розподілені обчислення –** це один із видів роботи в клієнт-серверній архітектурі, коли вхідні дані чи запити з клієнтських машин розподіляються поміж кількома серверами, що збільшує пропускну здатність для користувача і дає можливість багатозадачної роботи. Це сприяє максимальному використанню обчислювальних ресурсів, зниженню витрат і підвищенню ефективності системи.

V. **Можливість роботи в архітектурі** Internet/Intranet – стало невід'ємною складовою частиною вимог до корпоративних ІС. КІС надає користувачеві можливість вирішення таких глобальних задач:

- зробити прозорим для керівництва корпорацією використання вкладених бізнес-капіталів;
- надати повну інформацію для економічної доцільності стратегічного планування;
- професійно управляти витратами, наочно і своєчасно показувати, як мінімізувати витрати;
- реалізувати оперативне управління підприємством згідно вибраних ключових показників (собівартість продукції, структура витрат, рівень прибутковості тощо);
- забезпечити гарантовану прибутковість підприємства за рахунок оптимізації і прискорення ряду процесів (строків виконання нових замовлень, перерозподілу ресурсів тощо).

Повноцінна КІС повинна забезпечити інформаційну прозорість організації, формувати єдиний інформаційний простір, який об'єднує інформаційні потоки, що йдуть від виробництва до нього, з даними фінансово-господарських служб і видавати необхідні повідомлення для всіх рівнів управління підприємства.

Отже, **корпоративна інформаційна система** – це цілісний програмно-апаратний комплекс, що дозволяє задовольнити як поточні, так і стратегічні потреби підприємства в опрацюванні даних.

Можна виділити чотири чинники цілісності цього комплексу:

- концептуальна узгодженість бізнес-процесів, для автоматизації яких створюється ІС, що зберігається на всьому протязі її життєвого циклу;

- технологічна цілісність, яка проявляється в застосуванні погодженого набору промислових інформаційних технологій для управління інформаційними ресурсами підприємства;
- відповідність функціональності робочих місць співробітників їхнім посадовим обов'язкам;
- єдиний регламент обслуговування й експлуатації всіх компонентів ІС, розроблений при її створенні.

В даний час виділяють такі види КІС: управління ресурсами підприємств (ERP); управління відносинами з клієнтами (CRM); управління ланцюжками постачань (SCM) і ряд інших, що з'явилися останнім часом (наприклад, системи електронної комерції і системи управління майном підприємств EAM (Enterprise assetmanagement).

LanDocs

Система LanDocs призначена для автоматизації процесів управління документами, документообігом і діловодством на підприємствах і в організаціях різного профілю і масштабу. Система LanDocs реалізована як адаптивна CASE-модель електронного офісного документообігу і діловодства. Налагодження системи на конкретні умови експлуатації здійснюється модифікацією параметрів CASE-моделей без зміни програмного коду.

Постачається в двох варіантах: як закінчена система (програмне забезпечення, документація, навчання користувачів) чи як відкритий до розвитку варіант (базовий набір CASE-моделей, спеціалізована бібліотека діалогових елементів, CASE-технологія адаптації і підтримки, навчання користувачів, розроблювачів і фахівців групи підтримки).

Lotus Notes

Продукт Lotus Notes фірми Lotus Development Corporation являє собою засіб проектування систем підтримки групової роботи і може розглядатися як стандарт у цій області. Середовище Lotus Notes і додатки, створені на його основі, задовольняють основним вимогам до єдиної системи управління документообігом великих організацій і мають наступні функціональні можливості:

- **Облікова обробка усіх видів документів.** У базах даних Notes можна зберігати інформацію практично будь-якого типу й організовувати внутрішніми засобами облік і контроль за проходженням документів. Система Lotus Notes забезпечує побудову різних форм для реєстрації документів, що включають у себе як формальні реквізити, так і сам зміст документів. При необхідності власне зміст документа може зберігатися в окремій базі даних, у тому числі як образ сканованого паперового документа.

- **Унікальність номера.** У системі є можливість присвоєння документам унікальних номерів, у тому числі й у розподіленій обчислювальній мережі.
- **Узгодженість з іншими підсистемами.** Система Notes, поряд із засобами її розширення, забезпечує можливість інтеграції з іншими інформаційними підсистемами, зокрема поштовими, а також із системами, що працюють з реляційними базами даних. Підтримується доступ на рівні ODBC.
- **Розподілений документообіг.** Notes являє собою систему, що масштабується в широких межах, від окремо узятого ПК до систем з десятками серверів, до кожного з яких можуть бути підключені до декількох сотень користувачів. На кожному сервері можуть розміщатися кілька баз даних. Необхідна інформація розподіляється між цими базами, а інформація, необхідна для всіх користувачів, періодично синхронізується.
- **Обмеження доступу.** Захист даних реалізується на всіх рівнях: сервер, база даних, форма, документ, секція документа й окреме поле. При встановленні сеансу зв'язку обов'язково здійснюється аутентифікація користувача. Підтримується електронний підпис і шифрування даних.

В останні роки в Україні досить стрімко на великих підприємствах почались впроваджуватися корпоративні інформаційні системи КІС, що базуються на клієнт-серверній архітектурі. Цими системами почались витіснятися традиційні АСУП і цей процес набирає оберти.

Корпоративна інформаційна система R/3

Автоматизована система R/3 розроблена німецькою компанією — акціонерним товариством SAP AG, яка є безперечним світовим лідером по обсягу продаж прикладного програмного забезпечення архітектури клієнт-сервер (в 1996 році обсяг продаж її склав більше 1,5 млрд $, що в три рази перевищує продажі цього продукту відомою компанією Oracle, що йде на другому місці). Продукт компанії R/3 впроваджений більш ніж 15000 підприємствах світу. Зокрема, клієнтами SAP є такі відомі фірми як BMW, Mersedes-Benz AG, Adidas, General Electric, Philips, IBM, Telecom AG та багато інших. В Україні корпоративна система R/3 застосовується на Жидачівському ЦПК, на Чорнобильській АЕС, в Міненерго України, на комбінаті «Азовсталь» та ін.

Система R/3 реалізована на базі сучасної архітектури клієнт-сервер, що дає можливість організувати ефективну розподілену обробку інформації і працювати в UNIX- та WindowsNT-середовищах на обладнаннях провідних фірм-виробників обчислювальної техніки. Як системи управління базою даних можуть використовуватися сервери Oracle, Informix, Microsoft та ін.

Відмітні особливості програми корпоративної системи R/3 такі:

- **інтеграція** всіх виробничих сфер (послідовності функцій дозволяють з'єднати виробництво, збут і сферу обліку);

- **модульний принцип** побудови, який допускає ізольоване використання окремих компонентів програми, а також їх комбінації, що диктуються виробничо-економічною необхідністю;
- **наскрізний облік:** втому числі в сфері логістики визначаються значення, які негайно запам'ятовуються на носіях інформації в системі обліку (бухоблік, калькуляція витрат);
- **незалежність від конкретної галузі:** програмне забезпечення R/3 застосовується на виробничих підприємствах, в торгівлі, в банках і страхових агентствах, а також в сфері державної служби і в різних союзах; дружний призначений для користувача інтерфейс полегшує індивідуальну адаптацію;
- **інтернаціональність:** програма враховує різні вимоги найбільших промислових країн, вона не залежить від мови користувача і валюти країни;
- **структурування** шляхом чіткого розділення базового програмного забезпечення з технічними функціями (управління діалогом і базами даних) і прикладних виробничо-економічних програм;
- **відкритість:** концепція програмного продукту R/3 орієнтується на використання у відкритих системних середовищах; прикладні програми можна інсталювати на обчислювальних машинах найбільш відомих фірм-виробників.

Інші:
- Система управління бізнесом і фінансами Scala 5
- Система управління ресурсами підприємства Oracle Application
- Інформаційна система управління ресурсами підприємств Baan–IV
- Комплексна система управління діяльністю підприємства ГАЛАКТИКА

Список використаних джерела

1. Літнарович Р. М. Платформи корпоративних інформаційних систем : курс лекцій / Р. М. Літнарович. МЕГУ : Рівне, 2012. — 130 с.

2. Автоматизированные информационные технологии в экономике : Учебник / под ред. проф. Г. А. Титоренко. — М. : Компьютер, 1998. — 400 с.

3. Береза А. М. Основи створення інформаційних систем : Навчальний посібник / Береза А. М. — К.: КНЕУ, 1998.— 140 с.

4. Галузинський Г. П. Сучасні технологічні засоби обробки інформації : Навчальний посібник / Г. П. Галузинський, І. В. Гордієнко. — К. : КНЕУ, 1998. —224 с.

5. Єрьоміна Н. В. Проектування баз даних : Навчальний посібник / Н. В. Єрьоміна. — К. : КНЕУ, 1998. — 208с.

6. Єрьоміна Н. В. Банківські інформаційні системи : Навчальний посібник / Н. В. Єрьоміна — К. : КНЕУ, 2000. — 230с.

7. Информационные системы в экономике : Учебник / под ред. проф. В. В. Дика. — М. : Финансы и статистика, 1996. — 272 с.

8. Компьютеризация информационных процессов на промышленных предприятиях / [В. Ф. Сытник, Х. Срока, Н. В. Еремина и др.] — К. : Техника; Катовице : Экономическая академия им. Кароля Адамецкого, 1991. — 215 с.

9. Основи інформаційних систем : Навчальний посібник / За ред. проф. В.Ф.Ситника. — К. : КНЕУ, 1997. — 252 с.

Лекція 2

КЛІЄНТ-СЕРВЕРНІ АРХІТЕКТУРИ

Системи Клієнт-Сервер історично сформувались із перших централізованих ОС, які розвивались у 70-х роках. Вони одержали найбільше поширення у інформаційній сфері. В цій технології відступають від одного з головних принципів, а саме – ***відсутність єдиного центрального пристрою***.

Можна виділити **дві ідеї:**

1) загальні для всіх користувачів дані знаходяться на одному або декількох серверах;

2) наявність багатьох користувачів, які одночасно і паралельно обробляють свої дані, інакше кажучи, системи клієнт-сервер розподілені по відношенню до користувачів.

Під ***сервером*** розуміють будь-які системи, процес або комп'ютер, який володіє певним обчислювальним ресурсом: пам'ять, час, потужність процесора.

Під ***клієнтом*** розуміють систему, процес або комп'ютер, які:

1) вносять запит серверу, на який-небудь ресурс;

2) користуються цим ресурсом;

3) обслуговуються сервером.

У своєму розвитку Клієнт-Серверні моделі пройшли декілька етапів, у ході яких формувались різні моделі.

Виходячи із принципів функціонування розрізняють

Чотири моделі технології Клієнт-Сервер:

1. Модель файлового сервера (File Server – FS).
2. Модель віддаленого доступу (Remote Data Access – RDA).
3. Модель сервера бази даних (Data Base Server – DBS).
4. Модель сервера додатків (Application Server – AS).

Модель файлового сервера характеризується способом взаємодії робочих станцій у локальній мережі. Один із комп'ютерів визначається файловим сервером. Це є місце збереження даних.

У даній моделі (рис.1.) всі основні компоненти розміщуються на клієнтському пристрої. При зверненні до даних ядро СКБД звертається із запитами до FS сервера. З допомогою функцій ОС в пам'ять клієнтської установки на час роботи сеансу копіюється файл, сервер виконує пасивну роль.

Перевагами даної моделі є:

- простота;
- відсутність високих вимог до потужності сервера (головне – об'єм дискового простору);
- програмні компоненти СУБД не розподілені, ніяка частина СУБД на сервері не розміщена.

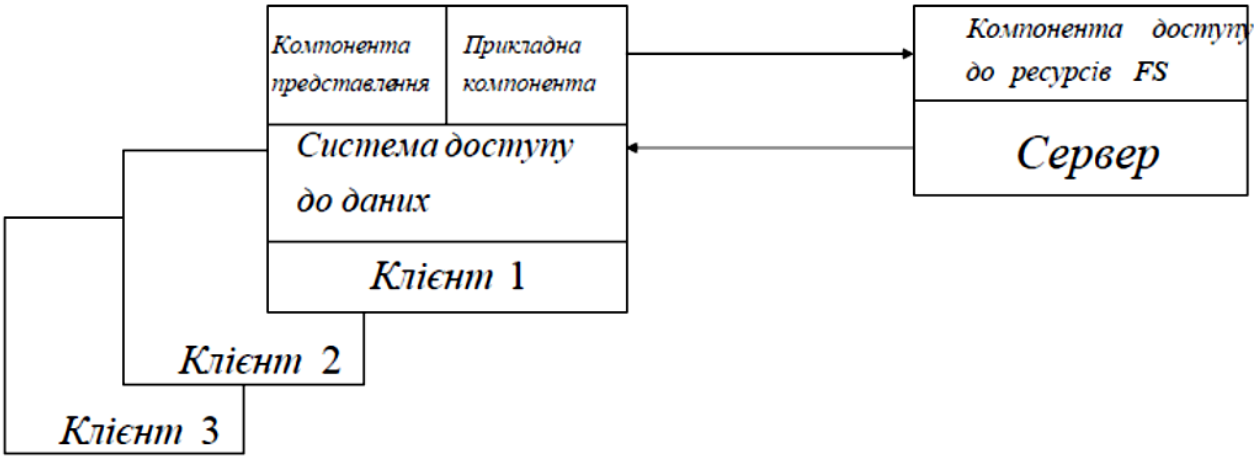

Рис. 1. Модель файлового сервера

Недоліки :

- високий мережевий трафік, що досягає пікових значень в момент масового підключення PC;
- відсутність спеціальних механізмів безпеки файлів БД зі сторони СКБД;
- розподілення даних між користувачами та паралельна робота із одним файлом відбувається лише засобами FS.

Модель віддаленого доступу до даних (RDA-модель). В моделі віддаленого доступу до даних (рис. 2.) компонент доступу до даних в СКБД повністю відділений від двох інших компонент – представлення і прикладної компоненти. Цей компонент доступу до даних розміщений на сервері.

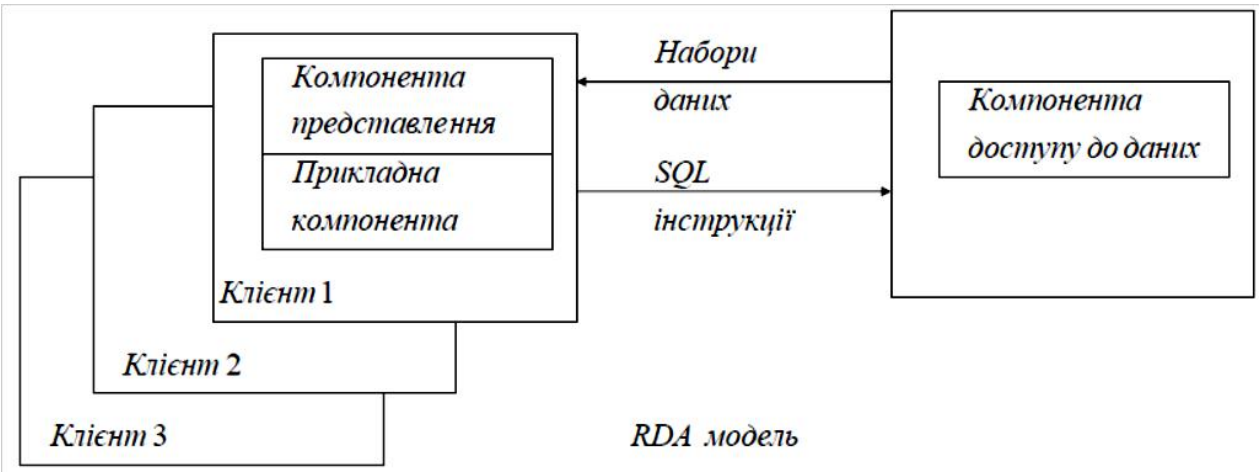

Рис. 2. Модель віддаленого доступу до даних

Функції SQL-сервера обмежуються низькорівневими операціями по організації, розміщенню, зберіганню, маніпулюванню даними в пам'яті сервера.

Компонента доступу до даних реалізована у вигляді самостійної програмної частини СКБД, яка називається *SQL- сервером* і інсталюється на обчислювальному пристрої - сервері системи. У файлах БД, що розміщуються на сервері системи, знаходиться системний каталог БД, у якому розміщуються відомості про зареєстрованих клієнтів і їх можливості. На клієнтських установках реалізуються окремі програмні частини СКБД, які реалізують *інтерфейсні і прикладні* функції. Користувач, який входить в клієнтську частину системи, реєструється через неї на сервері і починає обробку даних.

Прикладна компонента системи – бібліотеки запитів і процедури обробки даних повністю розміщені і виконуються на клієнтському пристрої. При реалізації своїх функцій прикладна компонента формує необхідні SQL-інструкції, що направляються SQL-серверу. SQL-сервер являє собою спеціальну програмну компоненту, що орієнтована на інтерпретацію SQL-інструкцій і високошвидкісне виконання низькорівневих операцій з даними.

SQL-сервер також приймає і керує SQL-інструкціями від різних клієнтів, виконує їх, перевіряє і направляє клієнтам результати обробки SQL-інструкцій, якими є таблиці даних. Після цього відбувається обробка даних на клієнтській установці. Це спілкування клієнта з сервером відбувається через SQL- інструкції, а з сервера на клієнтські пристрої передаються лише результати обробки інформації, тобто набори даних, які значно менші по об'єму, ніж вся БД.

В результаті цього різко зменшується завантаження мережі, а ***сервер виконує активну центральну функцію***. Крім того ядро СКБД забезпечує також функції по збереженню цілісності і безпеки даних при сумісній роботі кількох користувачів. Іншою перевагою є ***інтероперабельність*** системи, тобто незалежність від типу СКБД на клієнтських установках у розподіленій системі. Це означає, що спеціальна компонента ядра СКБД на сервері (драйвер ODBC) сприймає і обробляє запити та направляє результати їх обробки клієнтській установці із СКБД іншого типу. Це підвищує гнучкість розподіленої системи на базі інтеграції вже існуючих на фірмі локальних баз даних під керуванням довільного типу СКБД.

Недоліками даної моделі є високі вимоги до клієнтських ЕОМ, оскільки обробка даних відбувається на них; суттєвий трафік мережі, оскільки з сервера БД клієнтам направляються таблиці даних, які можуть мати великий об'єм.

Модель сервера бази даних (модель DBS). Розвитком RDA моделі стала модель DBS. Її основою є механізм збережених процедур. На відміну від RDA моделі, визначені для інформаційних систем події, правила і процедури, які описані запитами мови SQL, зберігаються разом з даними на

сервері системи і на сервері виконуються, тобто прикладна компонента повністю розміщена і виконується на сервері (рис .3).

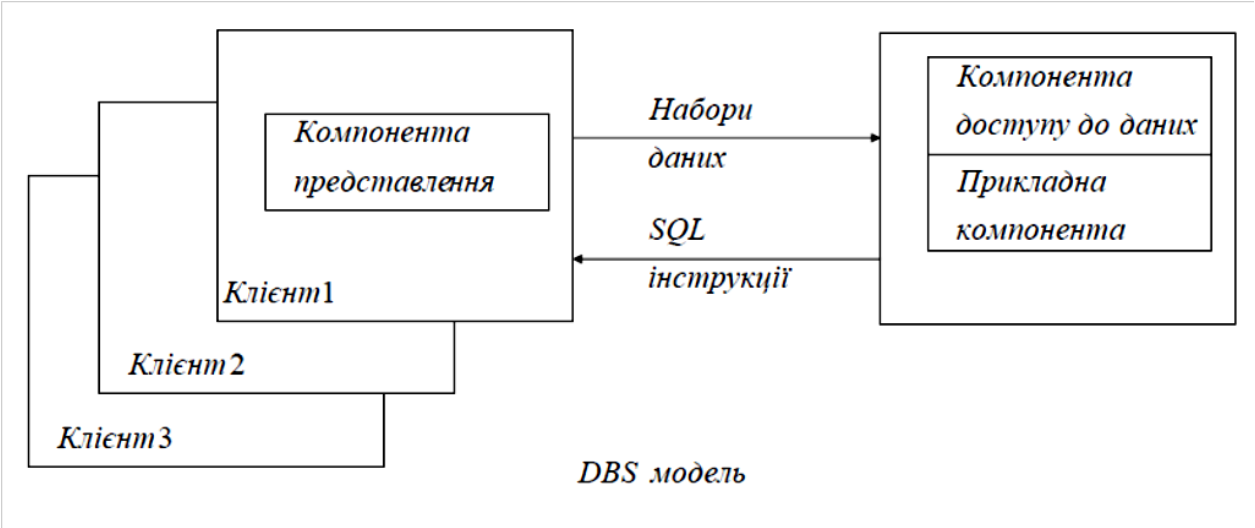

Рис. 3. Модель сервера бази даних

На клієнтських установках у DBS моделі розміщується лише компонента представлення (рис. 3). Це суттєво зменшує вимоги до клієнтських пристроїв. Користувач через інтерфейс системи на клієнтському пристрої направляє на сервер БД лише виклики запитів по обробці даних.

Всі операції по доступу і обробці даних виконуються на сервері і клієнту направляються лише результати обробки даних, а не набори даних як в RDA моделі. Цим забезпечується суттєве зниження трафіку мережі. На сервері виконуються процедури прикладних задач одночасно для всіх користувачів системи. В результаті цього різко зростають вимоги до обчислювальних пристроїв серверів – об'єм жорсткого диску, оперативної пам'яті.

Перевагою є більш активна роль сервера мережі, оскільки на ньому відбувається розміщення і обробка даних, подій, правил, процедур. А також більш надійно здійснюється узгодження зміни стану даних, координується колективна робота користувача із спільними даними.

Модель сервера додатків (AS модель)

Для того, щоб розподілити обчислювальні ресурси сервера щодо швидкодії і використанню пам'яті у різних обчислювальних пристроях використовують модель сервера додатків.

Суть цієї моделі полягає у перенесенні прикладної компоненти на додатковий сервер. Як і в DBS моделі тут на клієнтському пристрої розміщується лише інтерфейсна частина системи, тобто компонента представлення.

Трансакція – послідовна сукупність операцій над даними, а саме SQL-інструкцій, які мають конкретне смислове значення. У цьому відношенні

сервер додатків від імені клієнтів системи керує формуванням трансакцій, які виконує SQL-сервер, тому програмну компоненту СУБД, яка інсталюється на сервері додатків називають монітором обробки трансакцій (TRM – Transaction Processing Monitors).

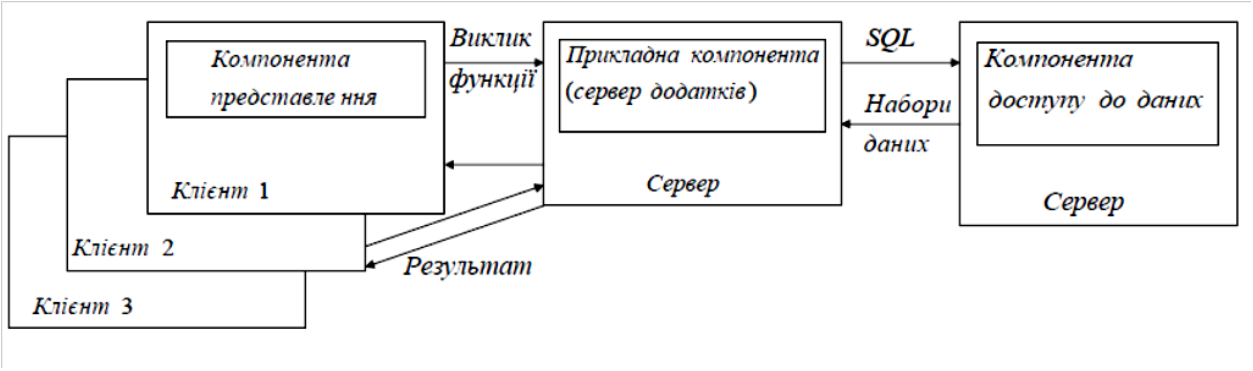

Рис. 4. Модель сервера додатків

Виклики функцій обробки даних направляються на сервер додатків і на ньому всі функції колективно виконуються для всіх користувачів системи. Так само як і в RDA моделі, сервер додатків звертається до SQL-сервера для виконання низькорівневих операцій по доступу і зміні даних. При цьому сервер додатків направляє SQL-серверу виклики SQL-процедур і одержує від нього набори даних.

Сьогоднішній світ обчислень орієнтований в основному на мережеві додатки. В основі цих додатків лежить модифікована **архітектура клієнт-сервер** – так звана трирівнева архітектура. Відмітна її риса – наявність на стороні сервера додатка, який, власне, і реалізує бізнес-логіку в середовищі сервера додатків. Додаток взаємодіє з сервером баз даних з одного боку і з віддаленої клієнтською частиною, яка зазвичай виконується в середовищі веб-браузера або в середовищі програми з графічним інтерфейсом.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного (рис. 5). В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- *модель тонкого клієнта*, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

- *модель товстого клієнта*, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

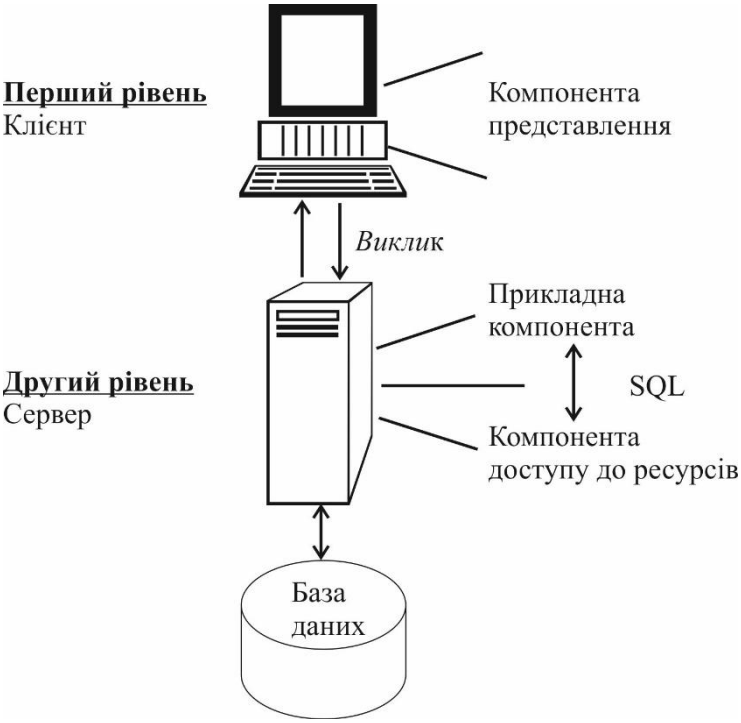

Рис. 5. Дворівнева клієнт-серверна архітектура

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення *прикладного* рівня від управління даними. Відокремлюється окремий *програмний* рівень, на якому зосереджується прикладна логіка застосунку (рис. 6).

Програми проміжного рівня можуть функціювати під управлінням спеціальних *серверів застосунків*, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється *сервером даних*.

Багаторівнева клієнт-серверна архітектура. Проникнення інформаційних технологій в сферу бізнесу як невід'ємної умови успішного управління призводить до того, що системи корпоративних масштабу потребують поєднання різних клієнт-серверних моделей в залежності від завдань, що вирішуються на різних конкретних напрямках діяльності підприємства. Можна прийти до висновку, що найбільш оптимальним вибором, з точки зору керованості та надійності системи, є поєднання різних моделей взаємодії клієнтської і серверної частини. По суті, ми приходимо навіть не до трьох-рівневої, а багаторівневої (N-ярус) моделі, яка поєднує

різних за "товщиною" клієнтів, сервери баз даних і безліч спеціалізованих серверів додатків, взаємодіючих на базі відкритих об’єктних стандартів.

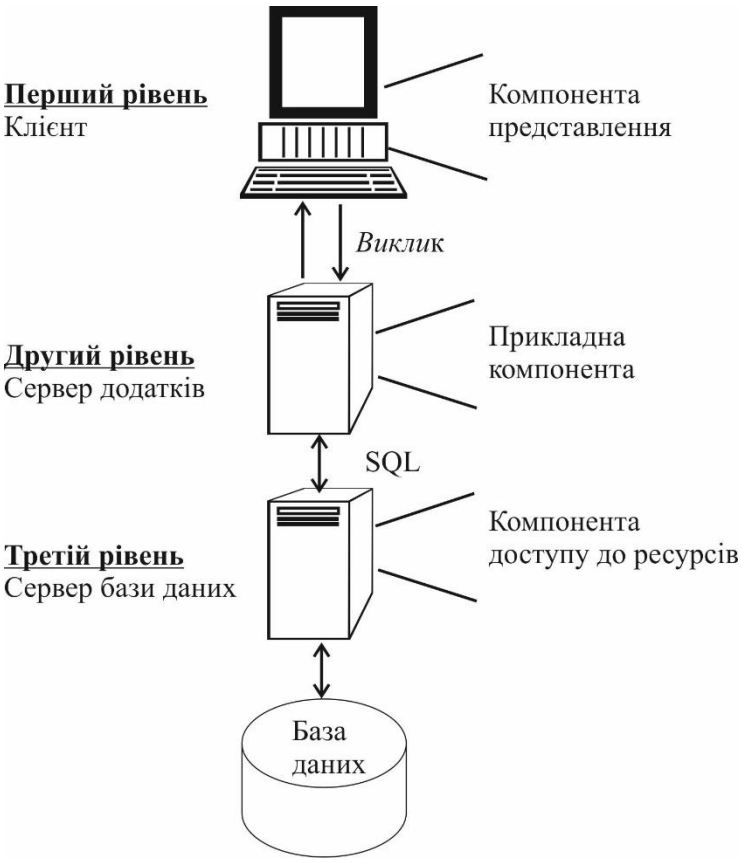

Рис. 6. Трирівнева клієнт-серверна архітектура

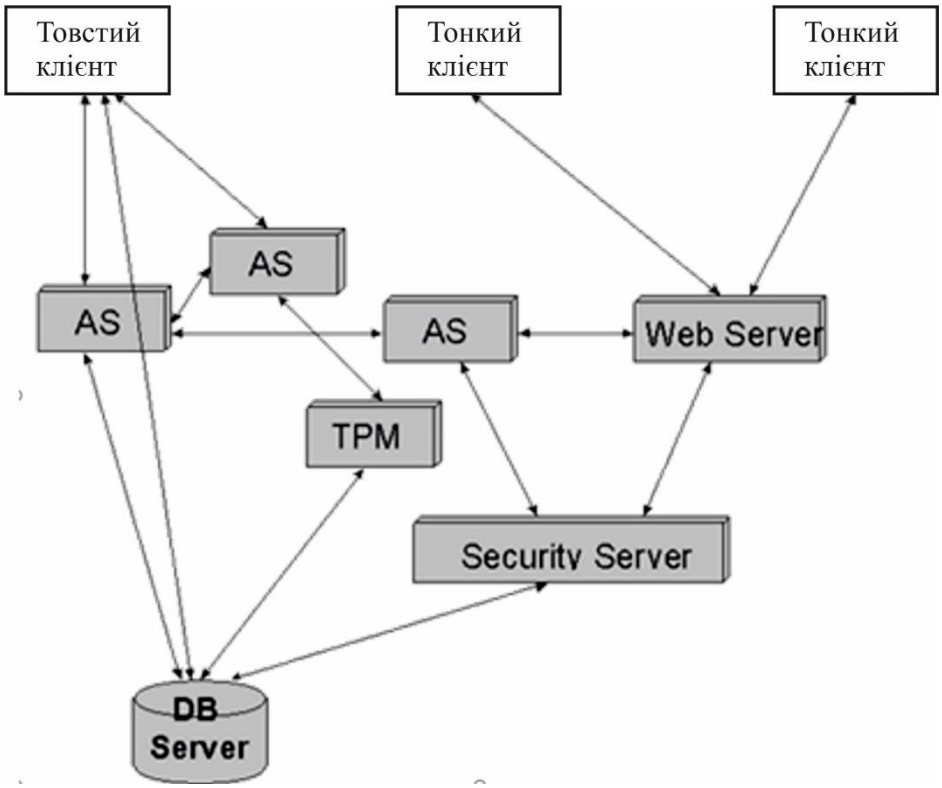

Рис. 7. Багаторівнева клієнт-серверна архітектура

Істотним полегшенням в реалізації багаторівневих гетерогенних систем є напрям роботи ряду виробників програмного забезпечення, спрямований на створення перехідного ПЗ. На відміну від продуктів проміжного рівня, що забезпечує верхній транспортний рівень (універсальні інтерфейси доступу до даних ODBC, JDBC, BDE; проміжне програмне забезпечення Message Oriented Middleware – MOM; об'єктних запитів Брокер – ОРБ; ...), перехідне ПЗ відповідає за трансляцію викликів в рамках одного стандарту обміну в виклики іншого – мости ODBC / JDBC і BDE / ODBC, COM / CORBA, Java / ActiveX і т.п.

Багаторівнева модель клієнт-серверної системи може бути представлена наступним чином:

Trusted Platform Module (TPM) – назва специфікації, яка описує криптопроцесор, в якому зберігаються криптографічні ключі для захисту інформації, а також узагальнене найменування реалізацій зазначеної специфікації. **Security Server –** сервер, що відповідає за прозоре, потокове шифрування даних для їх зберігання.

На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними — СКБД MySQL. Таким чином, зв'язок PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів.

Проте такі зв'яки не здатні реалізувати усіх вимог, що ставляться до копоративних систем, тому були розроблені такі платформи як JAVA (Java Platform, Enterprise Edition) та Microsoft .NET.

Список використаних джерела

1. Конспект лекций по дисциплине "Информационные технологии" [Електронний ресурс] — Режим доступу : https://studfiles.net/preview/2868013/
2. Орлик С. Многоуровневые модели в архитектуре клиент-сервер / С. Орлик. [Електронний ресурс] — Режим доступу : http://citforum.ck.ua/database/kbd97/22.shtml
3. Клієнт-серверна архітектура та ролі серверів. [Електронний ресурс] — Режим доступу : https://medium.com/@IvanZmerzlyi

Лекція 3

ЗАГАЛЬНІ ПОНЯТТЯ .NET.

Навіть невелика за тривалістю історія існування комп'ютерної техніки дозволяє стверджувати, що програмування відноситься до числа найбільш складних областей людської діяльності. Трудомісткість створення програмних систем управління критичними виробництвами, корпоративних систем великого розміру, систем рішення великих науково-технічних проблем, систем підтримки процесів прийняття рішень в різних областях додатків порівнянна (а часом і перевищує) складність розробки масштабних реальних об'єктів. Для подолання труднощів при розробці такого роду програмних систем теорія і практика в кожен конкретний історичний момент часу формує цілком певний перелік проблем, вирішення яких сприятиме виходу з чергової "кризи програмування".

Виділимо цей критичний список проблем:

- **Різноманітність приватних рішень для вирішення задач розробки великомасштабного програмного забезпечення**. Як результат, явно відчувалася потреба в розробці деякого загального підходу, в якому б критично враховувалися всі наявні рішення, і в рамках якого з єдиних позицій можна було б вирішувати багато проблем інформаційної індустрії.
- **Складність інтеграції існуючих рішень в рамках єдиних програмних систем**. Відмінність апаратно-програмних платформ, пропонованих корпоративних рішень, варіантність (версійність) програмних компонент виводить проблему інтеграції розроблюваного ПО в число найбільш гострих завдань програмування.
- **Трудомісткість розробки розподілених програмних систем**. При розробці розподілених систем проблеми забезпечення надійності, безпеки і масштабованості вимагали створення більш загальних засобів вирішення, визначення визнаних підходів і стандартів,
- **Широке поширення Інтернет технологій.** Світ Інтернету вимагав осмислення накопичених рішень і чекав промислового переходу на технології сервіс-орієнтованого програмного забезпечення та ін.

Наведений перелік не дає, звичайно, вичерпного уявлення про всі проблеми програмування кінця XX століття, але і цього списку досить для розуміння того, що світ програмування потребував деякого нового загального і промислового підходу до розробки великомасштабного програмного забезпечення.

JAVA. В кінці минулого століття компанія Sun Microsystems розробила прогресивну програмну технологію, за якою програма компілюється не в машинний, а в деякий проміжний, так званий байт-код. Байт-код виконується

віртуальною машиною. Технологія отримала назву Java. Однойменну назву отримала мова програмування високого рівня для даної технології. Таким чином, щоб використовувати програми розроблені на Java на новій апаратній платформі, варто лише розробити для неї віртуальну java-машину (JVM).

Особливістю подібного виконання JAVA-програм є:

- **Переносимість.** Програми, написані на мові Java, після одноразової трансляції в байт-код можуть бути виконані на будь-якій платформі, для якої реалізована віртуальна Java-машина. Найбільш ефективно можливості реального комп'ютера можуть використовувати тільки програми, написана з використанням «рідного» машинного коду.
- **Безпека**. Функціонування програми повністю визначається (і обмежується) віртуальною Java-машиною. Відсутні покажчики та інші механізми для безпосередньої роботи з фізичною пам'яттю і іншим апаратним забезпеченням комп'ютера. Додаткові обмеження знижують можливість написання ефективних Java-програм.
- **Надійність**. В мові Java відсутні механізми, які потенційно призводять до помилок: арифметика покажчиків, неявні перетворення типів з втратою точності і т.п. Присутній суворий контроль типів, обов'язковий контроль виняткових ситуацій. Багато логічнічних помилок виявляються на етапі компіляції. Наявність додаткових перевірок знижує ефективність виконання Java-програм.
- **«Збирач сміття»**. Звільнення пам'яті при роботі програми здійснюється автоматично за допомогою «збирача сміття», тому програмувати з використанням динамічно розподіленою пам'яттю простіше і надійніше. При інтенсивній роботі з динамічно розподіленою пам'яттю можливі помилки через те, що «збирач сміття» не встигає звільнити області пам'яті.
- **Стандартні бібліотеки.** Багато завдань, що зустрічаються при розробці програмного забезпечення, вже вирішені в рамках стандартних бібліотек. Використання об'єктно-орієнтованого підходу дозволяє легко використовувати готові об'єкти в своїх програмах. Для запуску програми необхідна установка JRE, що містить повний набір бібліотек, навіть якщо всі вони не використовуються в додатку. Відсутність бібліотеки необхідної версії може перешкодити запуску додатка.
- **Самодокументований код.** Є в наяності механізм автоматичного генерування документації на основі коментарів, розміщених в тексті програм.
- **Різноманіття типів застосунків**. Мовою Java можливо реалізувати абсолютно різні за способом функціонування і сфер використання програми.

PHP. PHP – це мова серверних сценаріїв, який можна вмонтувати в HTML-сторінку. Коли клієнт запитує цю сторінку у Web-сервера, запит проходить обробку перед відправкою даних клієнту. Зазвичай це виконується

за допомогою PHP-модуля, який входить до складу Web-сервера. PHP аналізує файл, код виконується і результат вставляється в сторінку яку відправляє. Концепція PHP чимось схожа на ідею JSP (JavaServer Pages) і Microsoft ASP (Active Server Pages). Типовий Web-додаток може вимагати підключення до бази даних, якщо звичайно весь вміст (контент) зберігається в БД, потребує виконання складних операцій на сервері до того, як сторінка буде повернена клієнтові. PHP забезпечує безліч розширень, які полегшують рішення подібних задач.

Переваги мови:

- **Легкість вивчення**. Дану мову можна швидко вивчити і домогтися високої продуктивності програмування. PHP призначений для Web-розробників і HTML-кодувальників, дозволяє їм без проблем додавати до своїх Web-сайтів сучасні можливості, такі як динамічна генерація сторінок.
- **Відкриті джерела**. PHP розповсюджується за ліцензією Apache, яка передбачає комерційне і некомерційне використання і розробку. Це означає, що програмою можна вільно користуватися без відрахування ліцензійних зборів. Крім того, існує всесвітня мережа талановитих розробників, які постійно покращують і розвивають PHP. Завдяки доступності вихідного програмного коду можна налагодити програму або налаштувати її під свої потреби.
- **Підтримка баз даних**. PHP забезпечує ефективну підтримку бах даних. Може працювати з ODBC, відкритими базами даних (MySQL і PostgreSQL), а також з комерційними (Microsoft SQL Server, Oracle і Sybase).
- **Розширення**. Існує безліч вільно доступних розширень і вихідного коду для будь-яких прикладних задач: від маніпуляції з XML до доступу до каталогів. Програмісти можуть використовувати цю масу готового коду для швидкого компілювання найсучасніших додатків.

Недоліки мови:

- **Мова не в повному обсязі об'єктно-орієнтована.** Мова не має таких властивостей повністю об'єктно-орієнтованої мови, як індивідуальні змінні, множинне спадкування і.т.д.
- **Не відповідність корпоративним вимога.** Мова досить популярна в світі програм з відкритим кодом і технічно перевершує багато комерційних аналогів. Однак їй не вистачає деяких важливих, з точки зору корпоративного середовища-особливостей. Це означає, що якщо необхідно використати PHP в корпорації, то це або взагалі не вдасться зробити, або буде потрібно значно більше додаткових програмних засобів, ніж при використанні Java або C ++.

Платформа .NET. У жовтні 1997 року Microsoft програла судовий позов фірми Sun, щодо порушення авторських прав використання технології

та торгової марки «Java». Фактично з того часу Microsoft розпочала створення власної технології схожої на Java. І назвала її **.NET** (читається "дот-нет"). Перша бета версія Microsoft .NET Framework розроблена в 2000 році. Нині Microsoft .NET активно використовується в складі операційних систем Windows XP, Windows Vista та Windows 7. Крім того .NET має всі передумови для активної підтримки та поширення на більшості сучасних операційних систем, як то Linux чи Free BSD.

Подібно до технології Java, ***середовище розробки .NET***, теж створює байт-код призначений для виконання віртуальною машиною, яка носить назву .NET Runtime. Так само, як і байт-код для віртуальної java-машини програма для середовища .NET Runtime може виконуватися будь-якою операційною системою, в якій ***NET Runtime*** встановлена. Хоча кількість операційних систем, де може виконуватись .NET Runtime, постійно зростає, .NET Runtime дещо відстає від віртуальної машини Java в плані втілення її на інших платформах.

Вхідна мова цієї машини в .NET називається MSIL (Microsoft Intermediate Language – проміжна мова Microsoft). Іноді використовується інша назва проміжної мови – CIL (Common Intermediate Language – загальноприйнята проміжна мова), або просто IL (проміжна мова) (рис.8).

Найбільш повну підтримку проміжної машинної мови CIL, поки що, реалізовано лише в операційних системах фірми Microsoft починаючи з Windows XP. Цікавим в пізнавальному плані є міжнародний проект «Mono». Це проект повноцінного використання системи .NET на базі вільного програмного забезпечення.

Рис. 8. Етапи процесу перетворення коду мови програмування в машинний код в .NET

Платформа .NET – сукупність множини служб і компонентів, які дозволяють розробленим на мові C# (а також будь-якій іншій .NET-підтримуваній мові високого рівня) додатку працювати в середовищі Windows або іншої ОС. Двома базовими складовими платформи .NET є загальномовне середовище виконання (**CLR**-) і бібліотека спільних класів **(.NET Framework**).

CLR є деякою програмною оболонкою, яка управляє виконанням вашого коду. Це управління проявляється в пам’яті, в потоках додатків, у віддаленій взаємодії, а також в строгому контролі відповідності виконуваного коду безлічі вимог. Тобто, **CLR** – це набір деяких служб, які і виконують ваш код.

Отже, не кожен код зможе бути виконаний під управлінням **CLR**. Природно, існує безліч мов програмування і кожна з них так, чи інакше пов’язана з деякою платформою. З огляду на це кажуть, що той код, який розроблявся для виконання **CLR** (чи то платформою .NET) називається керованим. А той, який не був розрахований на виконання під .NET, називається некерованим. Але ці поняття стосуються тільки самої .NET.

Common Type System (**CTS**) – стандарт, що задає правила визначення типів (рис. 9.). Мова не може містити особливості які не підтримуються **CTS**.

Common Language Specification (**CLS**) – мінімальний набір правил, яким повинна задовольняти кожна мова (рис. 10.). Модуль на C# може викликати тільки ті методи модуля на VB.NET, які написані відповідно до **CLS.**

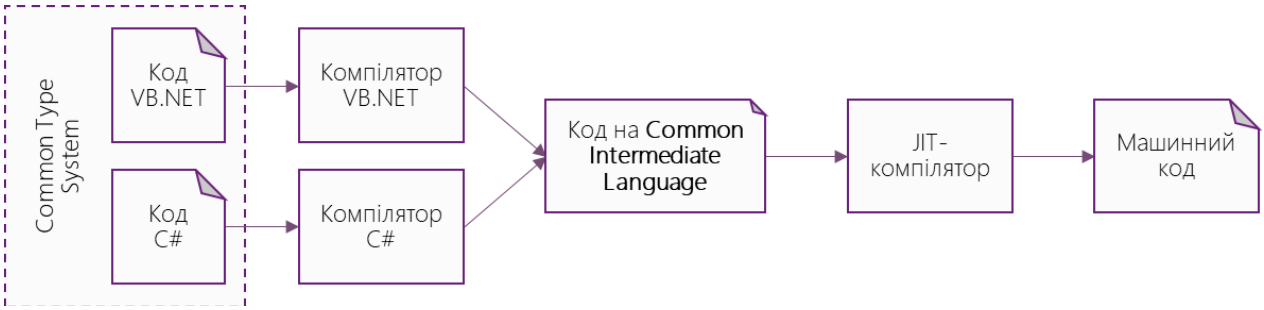

Рис. 9. Common Type System

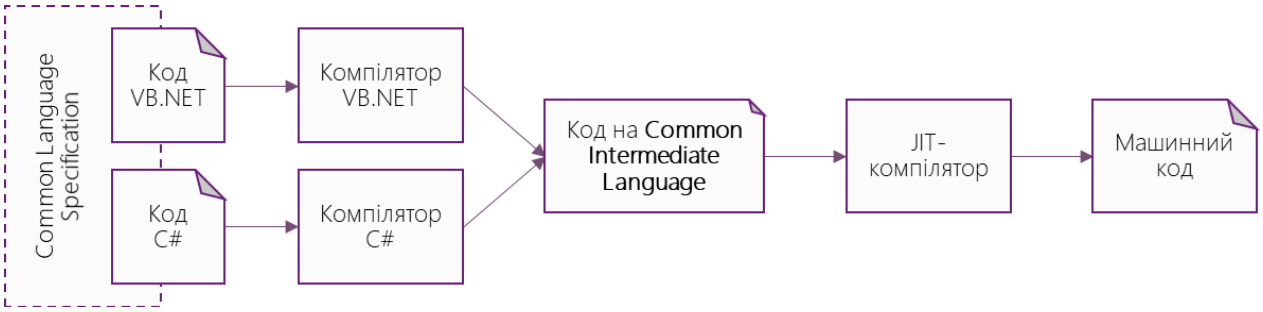

Рис. 10. Common Language Specification

.NET Framework – бібліотека класів являє собою величезний набір всіляких класів, які в процесі їх використання вами, як розробником, в кінцевому підсумку надають необхідну функціональність для вашої розробки. Відповідно, всі класи строго структуровані і розбиті по групах, адже кожен з них інкапсулює в собі деяку унікальну функціональність. В окремі групи потрапляють ті класи, функціональність яких схожа, або відноситься до однієї теми, або по ще яким-небудь ознаками.

Отже, у **CLR** є взаємодія з операційною системою. **CLR** також має доступ до бібліотеки базових класів .NET Framework і користувацьких класів. Зауважте, що призначені для користувача класи можуть базуватися на базових класах.

Рис. 11. Зв'язки між управляючим кодом, бібліотеками,
CLR та операційною системою

Переваги в використанні.

- Для розробки програми під .NET можливо користуватися однією з багатьох .NET-підтримуваних мов. Код, написаний на різних мовах, буде працювати однаково. Це ще дає ту перевагу, що код, написаний на одній мові, може використовуватися кодом, написаним на іншій.
- Наявність спільної бібліотеки базових класів, використовувати яку можуть всі .NET-сумісні мови.
- Спільний механізм CLR для будь-якої .NET-сумісної мови;
- Керування і мінімізація всіляких конфліктів версій при розгортанні додатків на цільовій машині. Крім того немає необхідності в реєстрації компонентів в системному реєстрі.
- Процес розробки будь-якого типу додатків подібний і багато в чому ідентичний, як WindowsForms-додатків, так і веб-додатків.
- Повна підтримка і контроль відповідності структури виконуваного коду стандартам об'єктно-орієнтованого програмування.

Механізм роботи CLR. Отже, ми маємо вихідний код, написаний на мові C#. Проект готовий повністю і необхідно передати його на запуск. Перш за все, необхідно C# -компілятором скомпілювати цей код в проміжний код, так званий MSIL-код (MSIL – це абревіатура Microsoft Intermediate Language). Взагалі то, ми можемо використовувати будь-який компілятор, але тільки той який відповідає тій .NET-сумісній мові, на якій писався вихідний код програми. Якщо ми використовували мову C#, а тому і компілювали C# -компілятором. На даному етапі ми отримали проміжний код у вигляді .exe, або .dll -файлів. Ці файли називаються збірками. Далі, при безпосередньому запуску на виконання цих файлів, CLR аналізує проміжний код і завантажує

в пам’ять необхідні класи. Далі, за допомогою JIT (Just In Time)-компілятора перетворює проміжний код в конкретний набір інструкцій для даної платформи. І тільки після цього отримані інструкції запускаються на виконання (рис.12).

Чому необхідна проміжна компіляція вихідного коду? Базовою причиною цього є підтримка платформою .NET вихідного коду безлічі різних мов програмування. Таким чином, незалежно від того, якою мовою було написано додаток, при його запуску CLR завжди отримує практично ідентичний MSIL-код (збірки). Це дає перевагу в виборі улюбленої мови для розробки, а також використання декількох мов при створенні одного проекту.

Коли компілятор створює код MSIL, одночасно створюються ***метадані***. ***Метадані*** – опис високорівневої структури коду, містять опис типів в коді, включаючи визначення кожного типу, підписи кожного члена типу, члени, на які є посилання в коді, а також інші відомості, що використовуються середовищем виконання під час виконання.

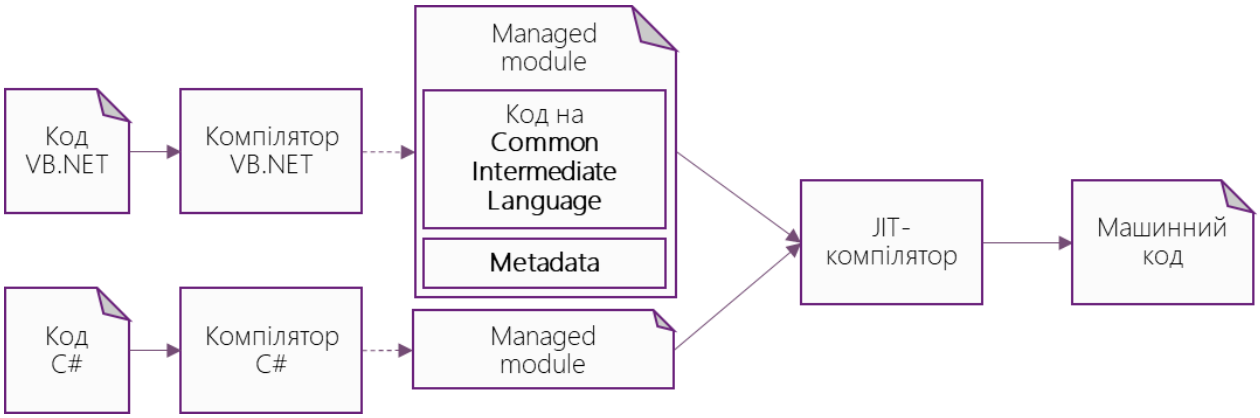

Рис. 12. Механізм роботи CLR

Середовище виконання знаходить і витягує метадані з файлу в міру необхідності при виконанні. JIT-компілятор компілює відразу весь код. Компіляція відбувається по частинам, в разі потреби. Тобто, компілюється та ділянку коду, яка безпосередньо в даний момент повинна піти на виконання. Одного разу скомпілювавши деякий шматок коду, компілятор видаляє його інструкції з пам’яті, а зберігає на випадок повторного виклику. Таким чином, одного разу скомпільований метод більше не компілюється і всі його наступні неодноразові виклики безпосередньо звертаються до вже машинного коду. Але JIT-компіляція відбувається кожного разу при запуску програми, адже вихідні виконувані файли програми зберігаються у вигляді збірок (EXE, DLL файлів). Тобто кожен раз, при запуску програми, CLR на льоту підключає необхідні класи і виконує JIT-компіляцію наданих йому збірок (рис.13).

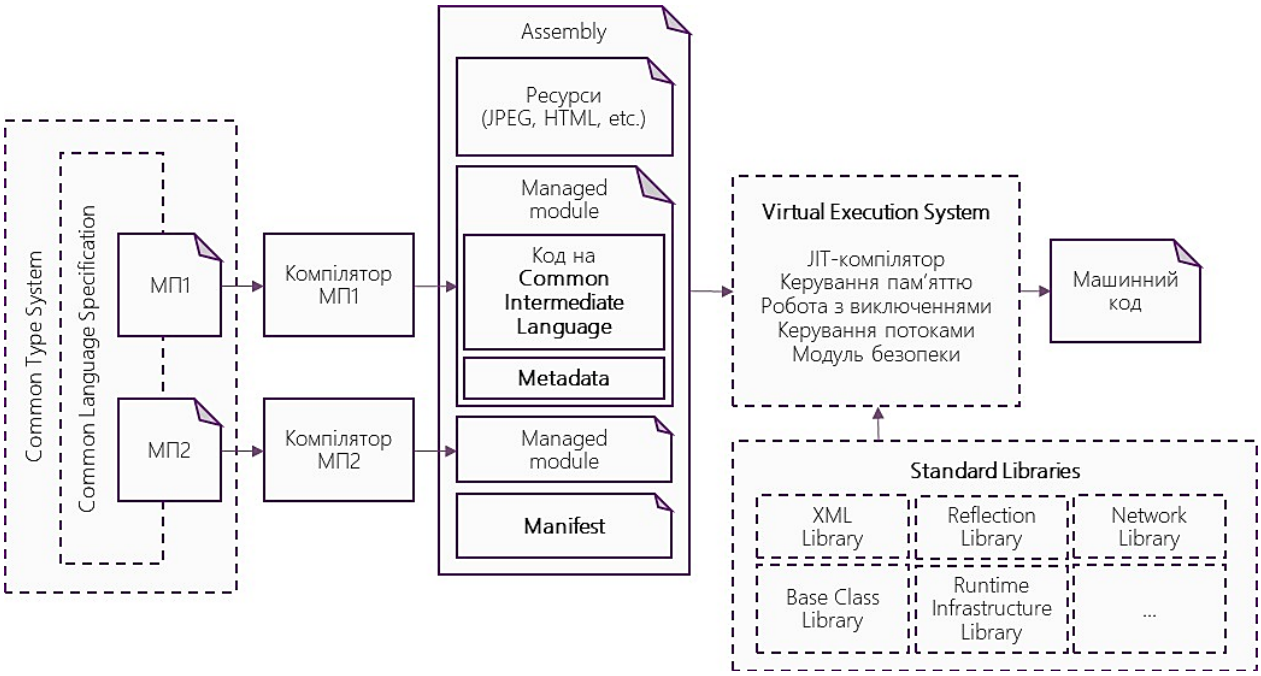

Рис. 13. Загальна схема перетворення мови програмування в машинний код

Список використаних джерела

1. Введение в платформу .NET FrameWork [Електронний ресурс] — Режим доступу : http://skillcoding.com/Default.aspx?id=79
2. Меженин М. Microsoft .NET. Введение в .NET : лекции / Меженин Михаил // кафедра "Системное программирование", ВМИ, ЮУрГУ, 2014

Лекція 4

.NET ЯК ПІДХІД ДО ПРОЕКТУВАННЯ І РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Основні особливості .NET:
- виконання у віртуальній машині;
- незалежність від мови;
- велика бібліотека вбудованих класів;
- переносимість на інші платформи.

.NET – це підхід до проектування і реалізації програмного забезпечення, що включає щонайменше чотири наступних компоненти:
- **ідеологія** проектування і реалізації програмного забезпечення;
- **модель** ефективної підтримки життєвого циклу прикладних систем;
- уніфікована, інтегрована технологічна **платформа** для програмування;
- сучасний, зручний у використанні, безпечний **інструментарій** для створення, розміщення і підтримки програмного забезпечення.

Ідеологія. Корпорацією-розробником сформульовані приблизно такі найважливіші аспекти бачення (vision) ідеології .NET:
- легкість розгортання додатків в глобальному середовищі Інтернет;
- економічна розробка програмного забезпечення;
- «безшовна», гнучка інтеграція програмних продуктів і апаратних ресурсів;
- надання програмного забезпечення як сервісу;
- новий рівень безпеки і зручності використання.

.NET Як обчислювальна модель
- компонентний підхід як розвиток об’єктно-орієнтованої моделі;
- універсальна система типізації: «будь-яка сутність є об’єкт»; уніфікація даних і метаданих;
- строго ієрархічна організація коду, просторів імен і класів;
- універсальний інтерфейс .NET Framework (включаючи підтримку різних підходів до програмування);
- висока варіативність примірників реалізації (зокрема, на основі веб-сервісів).

Розглянемо докладніше, як ідеологія .NET втілюється в практичні питання проектування програмного забезпечення.

Корпорацією Microsoft запропонований новаторський компонентно-орієнтований підхід до проектування, який є розвитком об’єктно-орієнтованого напряму. Відповідно до цього підходу, інтеграція об’єктів (можливо, гетерогенної природи), здійснюється на основі інтерфейсів, які

представляють ці об’єкти (або фрагменти програм) як незалежні компоненти. Такий підхід суттєво полегшує написання і взаємодію програмних «молекул» – компонент в гетерогенному середовищі проектування і реалізації. Стандартизується зберігання і повторне використання компонент програмного проекту в умовах розподіленого мережевого середовища обчислень, де різні комп’ютери і користувачі обмінюються інформацією, наприклад, взаємодіючи в рамках дослідницького або бізнес-проекту. Істотною перевагою є і можливість практичної реалізації принципу «будь-яка сутність є об’єктом гетерогенного програмного середовища». Це стало можливим завдяки вдосконаленій, узагальненій системі типізації **Common Type System**, або **CTS**, яка буде докладніше розглянута в одній з наступних лекцій.

Сувора ієрархічність організації просторів для типів, класів і імен сутностей програми дозволяє стандартизувати і уніфікувати реалізацію.

Новий підхід до інтеграції компонент застосунків в середовищі обчислень Інтернет (або так звані веб-сервіси), дає можливість прискореного створення додатків для глобальної аудиторії користувачів.

Універсальний інтерфейс .NET Framework забезпечує інтегроване проектування і реалізацію компонент додатків, розроблених відповідно до різних підходів до програмування.

.NET як технологічна платформа

- підтримка багатьох (десятків мов програмування);
- використання технології веб-сервісів для забезпечення інтероперабельності і масштабованості в глобальному мережевому середовищі;
- уніфікація доступу до бібліотек API-інтерфейсу незалежно від мови і програмної моделі;
- відповідність сучасним технологічним стандартам.

.NET забезпечує одночасну підтримку проектування і реалізації програмного забезпечення з використанням ***різних мов програмування***. При цьому підтримуються десятки мов програмування, починаючи від найперших (зокрема, COBOL і FORTRAN) і закінчуючи найсучаснішими (наприклад, C# і Visual Basic). Ранні мови програмування до цих пір активно використовуються, зокрема, для забезпечення сумісності з раніше створеними додатками, критичними для бізнесу (скажімо, COBOL дуже широко використовувався для створення прикладних програм, що підтримують фінансову діяльність).

Застосування ***технології веб-сервісів*** – це реальна найбільш прийнятна практична можливість забезпечення масштабованості і інтероперабельності додатків. Під масштабованістю розуміють можливість плавного зростання часу відповіді програмної системи на запит з ростом числа одночасно працюючих користувачів; у випадку веб-сервісів масштабованість реалізується за допомогою розподілу обчислювальних ресурсів між

сервером, на якому виконується прикладна програма (або зберігаються дані) і комп'ютером користувача. Під *інтероперабельністю* розуміється можливість інтегрованої обробки гетерогенних даних, що надходять від різнорідних прикладних програм.

Саме завдяки інтероперабельності можлива уніфікація взаємодії користувачів через додаток з операційною системою на основі спеціалізованого інтерфейсу прикладних програм, або API-інтерфейсу (Application Programming Interface).

Важливо відзначити і ту обставину, що нова технологія .NET не тільки затребувана світовою громадськістю, а й офіційно визнана, що відображено в відповідних стандартах ECMA (European Computer Manufacturers Association).

.NET – універсальний інструментальний засіб

- підтримка багатомовного середовища CLR (Common Language Runtime);
- можливість створювати компоненти проекту в єдиному середовищі на найбільш підходящій мові програмування;
- доступність всіх засобів .NET для кожної з широкого спектра мов програмування;
- сервісні можливості для розробників, (налагодження, аналіз коду, ...) однакові для всіх мов;
- Можливість полегшеної самостійної розробки транслятора для будь-якої мови програмування (Microsoft - VB, C#, ... інші – APL, COBOL, Eiffel, Fortran, Haskell, SML, Perl, Python, Scheme, Smalltalk, ...).

Необхідно відзначити підтримку багатомовного середовища розробки додатків CLR (Common Language Runtime). Ця можливість з'явилася завдяки універсальному міжмовному інтерфейсу Common Language Infrastructure, або CLI, який підтримує розробку програмних компонент на різних мовах програмування.

При цьому безперечною перевагою для програмістів є та обставина, що вони можуть розробляти (або допрацьовувати) програмне забезпечення на найбільш відповідній мові програмування. Тут слід враховувати характер завдання (скажімо, рекурсію або символьну обробка прозоріше і з меншими затратами можна реалізувати на мові функціонального програмування, а формалізація структури предметної області – на об'єктно-орієнтованій мові). Крім того, необхідно брати до уваги досвід роботи програмістів в команді розробників і ту мову програмування, на якому з самого спочатку створювалася система.

Відзначимо ще два істотні обставини.

По-пе*рше*, основні сервісні можливості для розробників, які надаються .NET (налагодження, аналіз коду і т. д.) залежить від конкретної мови програмування, і, отже, програмістам немає необхідності заново вивчати особливості середовища розробки, якщо необхідно перейти з однієї мови на іншу.

По-друге, незважаючи на те, що ще не всі мови програмування підтримуються .NET, існує можливість самостійної розробки транслятора для будь-якої мови програмування, причому його реалізація не викликає труднощів навіть у програмістів, практично не мають професійної підготовки в області розробки компіляторів.

Висновки

- Стратегічне бачення і технологічна платформа Microsoft для наступного десятиліття.
- Якісні переваги над аналогами шляхом сумісності та мовної взаємодії, багаторівневої безпека, інтеграція з веб-сервісами, простота розгортання і використання.
- .NET – розвиток платформи Windows.
- .NET – фундамент для створення корпоративних застосунків нового покоління.
- Платформа .NET надзвичайно перспективна, швидко розвивається і знаходить широке застосування.
- .NET – технологічна платформа. Багатомовна підтримка (десятки мов програмування).
- Використання технології веб-сервісів для забезпечення інтероперабельності і масштабованості в глобальному мережевому середовищі.
- .NET - універсальний інструментальний засіб. Підтримка багатомовного середовища CLR(Common Language Runtime).
- Можливість створювати компоненти проекту в єдиному середовищі на найбільш прийнятній мові програмування
- Стратегічна мета .NET - створення інфраструктури для розвитку та роботи розподілених додатків на основі стандартів Інтернету.

Список використаних джерела

1. Основы технологии программирования .NET [Електронний ресурс] — Режим доступу : https://mmezhenin.wordpress.com/net/
2. Введение в платформу .NET FrameWork [Електронний ресурс] — Режим доступу : http://skillcoding.com/Default.aspx?id=79
3. Меженин М. Microsoft .NET. Введение в .NET : лекции / Меженин Михаил // кафедра "Системное программирование", ВМИ, ЮУрГУ, 2014
4. Richter J. CLR via C# (4th Edition) / Jeffrey Richter . — Microsoft Press, 2012. — 896 p.
5. Albahari J. C# 6.0 in a Nutshell / Joseph Albahari. — O'Reilly Media, 2015. — 1136 p.

Лекція 5
АРХІТЕКТУРНА СХЕМА
.NET Framework і Visual Studio.NET

Істотною перевагою конструктивного рішення є компонентно-орієнтований підхід до проектування і реалізації програмного забезпечення. Суть підходу полягає в принциповій можливості створення незалежних складових програмного забезпечення з уніфікованою інтерфейсною частиною для багаторазового повторного і розподіленого використання. При цьому продуктивність рішення обумовлена багатомовністю інтегрованих програмних проектів (.NET концепція потенційно підтримує довільні мови програмування, в числі найбільш відомих мов - C #, Visual Basic, C ++ та ін).

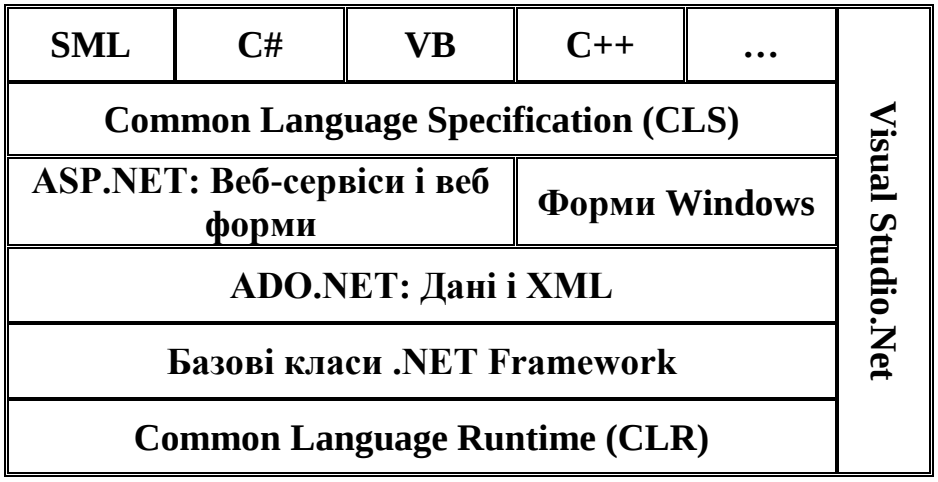

Рис. 14. Архітектурна схема .NET Framework

В ході компіляції програма на .NET-сумісній мові програмування трансформується відповідно до заздалегідь заданої узагальненої специфікацію мови. Система типів CTS повністю описує всі типи даних, підтримувані середовищем виконання, визначає їх взаємозв'язок і зберігає їх відображення в систему типів .NET (рис.14).

Під Common Language Specification (або CLS) розуміється набір правил, що визначають підмножину узагальнених типів даних, у відношенні яких гарантується, що вони безпечні при використанні у всіх мовах .NET. Інтерфейси реалізуються за допомогою форм для Windows і ASP.NET для веб-додатків.

Архітектура .Net Framework. Платформа складається з двох частин. Основою є виконуючого середовища Common Language Runtime (CLR), яка може виконувати як звичайні програми, так і серверні додатки. Друга, не менш важлива частина, це бібліотека класів Framework Class Library (FCL), що містить в собі безліч компонентів для роботи з базами даних, мережею, введенням/виведенням, файлами, призначеним для інтерфейсу користувача і

т.д. Це дозволяє розробнику не займатися низькорівневим програмуванням, а використовувати вже готові класи.

Важливі ***частини бібліотеки класів*** Windows Forms – відповідає за розробку графічного інтерфейсу. Фактично є оболонкою над Win32 API.

ADO.NET – надає доступ даними. В основному використовується для роботи з базами даних.

ASP.NET – технологія розробки веб-сайтів, веб-додатків і веб-сервісів.

Language Integrated Query (LINQ) – реалізація мови запитів, що нагадує по синтаксису SQL в програмах на .Net.

Windows Presentation Foundation (WPF) – система створення графічних інтерфейсів, що використовує мову розмітки XAML. На відміну від Windows Forms використовує графічну технологію DirectX, що забезпечує швидшу роботу за рахунок апаратного прискорення графіки.

Windows Communication Foundation (WCF) – система обміну даними між додатками .Net. Використовується для створення розподілених додатків

Бібліотека класів .NET Framework містить собою колекцію типів, які тісно інтегруються із середовищем CLR. Бібліотека класів .NET Framework являє собою бібліотеку типів класів, інтерфейсів, і значення типів, які забезпечують доступ до функціональних можливостей системи. Бібліотека класів є об’єктно-орієнтованою, надаючи типи, з яких керований код користувача може успадковувати функції. Це не тільки спрощує роботу з типами .NET Framework, але також зменшує час, що витрачається на вивчення нових засобів платформи .NET Framework. Крім того, компоненти незалежних виробників можна легко поєднувати з класами платформи .NET Framework.

Наприклад, в класах колекцій .NET Framework реалізується набір інтерфейсів, які можна використовувати для розробки призначених для користувача класів колекцій. Користувацькі класи колекцій будуть без труднощів об’єднуватися з класами .NET Framework.

Як і очікується від об’єктно-орієнтованої бібліотеки класів, типи .NET Framework дозволяють вирішувати типові завдання програмування, включаючи роботу з рядками, збір даних, підключення до баз даних і доступ до файлів.

В доповнення до звичайних задач бібліотека класів містить типи, що підтримують багато спеціалізованих сценаріїв розробки. Наприклад, можна використовувати платформу .NET Framework для розробки наступних типів додатків і служб:

- консольні додатки;
- додатки з графічним інтерфейсом користувача Windows (Windows Forms);
- додатки Windows Presentation Foundation (WPF);
- додатки ASP.NET;
- служби Windows (Windows Service Applications);

- сервісно орієнтовані додатки за допомогою Windows Communication Foundation (WCF).
- додатки, які підтримують бізнес-процеси Windows Workflow Foundation (WF)

Універсальна схема типізації.

Суттєвою позитивною відмінністю Microsoft .NET від існуючих аналогів на сучасному ринку програмного забезпечення є універсальна система типізації.

В ході компіляції програма на .NET-сумісній мові програмування трансформується відповідно до заздалегідь заданої узагальненої специфікацією мови Common Type System (CTS). Система типів CTS повністю описує всі типи даних, підтримувані середовищем виконання, визначає їх взаємозв'язок і зберігає їх відображення в систему типів .NET. Система типізації Microsoft .NET являє собою частково впорядковану множину, яку на якісному рівні можна вважати як ISA-ієрархія (ISA походить від англійських слів "is a", які означають «є одним з»).

Так, наприклад, висловлювання STUDENT ISA PERSON означає, що тип STUDENT є підтипом типу PERSON (тут цілком доречна аналогія з множинами і цілком точна аналогія з доменами). Таким чином, система типів Microsoft .NET утворює ієрархію із зростанням множини знизу до верху, в якій явно виділяються дві великі групи типів, а саме, типи-посилання і типи-значення. Відмінність в них полягає в особливостях їх виклику в процедурах: по імені або за значенням (call-by-name, CBN) і за посиланням (call-by-reference, CBR).

Зауважимо також, що система типізації Microsoft .NET крім розвиненої ієрархії визначених типів дозволяє користувачеві створювати власні типи (як типи-посилання, так і типи-значення) на основі вже існуючих.

Веб-сервіси в .NET

Призначення веб-сервісів полягає в розподілі можливостей розроблених прикладних систем по каналах глобальної мережі Інтернет.

Зауважимо, що центральним блоком у схемі є .NET Framework, який можна розглядати як бібліотеку базових об'єктів і операцій над ними. Як середовище розробки прикладних систем доцільно використовувати Microsoft Visual Studio .NET, що надає цілий комплекс розвинених засобів створення, редагування та налагодження програмного коду на різних мовах програмування. В випадку нескладних завдань можна обмежитися примітивними редакторами тексту програм, подібних Notepad.

Інтерфейсна частина прикладної програмної системи в Інтернет-архітектурі представлена так званими веб-формами, призначеними для введення і виведення даних в уніфікованому форматі.

Як мова реалізації може використовуватися мова гіпертекстової розмітки HTML (HyperText Markup Language). Взаємодія між клієнтом і

додатком в найпростішому випадку здійснюється з використанням традиційного Інтернет-протоколу передачі даних HTTP (HyperText Transfer Protocol).

Структуровані дані зберігаються у форматі XML (варіант HTML з більш суворим синтаксисом).

Зауважимо, що технологія веб-сервісів, реалізована Microsoft, допускає інтеграцію з компонентами сторонніх виробників.

Спробуємо сформулювати визначення поняття «веб-сервіс» (або, інакше, «веб-служба»).

Під веб-сервісами зазвичай розуміють програмовані компоненти прикладних програмних систем, які доступні для клієнта (користувача) за допомогою стандартних протоколів, що застосовуються для роботи в Інтернет-середовищі.

Як уже згадувалося раніше, саме веб-сервіси є однією з найважливіших складових ідеології .NET і центральною частиною даної архітектури, оскільки призначені для реалізації декларованого Microsoft основоположного принципу ***«Програмне забезпечення як сервіс».***

Сенс використання веб-сервісів полягає в можливості розподілу функціональних можливостей розроблених прикладних систем по глобальній мережі. Для реалізації цього завдання веб-сервіси надбудовуються на традиційних, які пройшли тривалу апробацію в минулому стандартах взаємодії додатків в Інтернет, а також на тих що і далі розвиваються, а саме:

- HTTP – стандартний протокол обміну гіпертекстовими документами в Інтернет з можливістю передачі даних за допомогою веб-форм;
- XML – формат зберігання структурованих даних з можливістю обміну ними по Інтернет-каналах;
- SOAP – стандартний протокол взаємодії компонентів (глобально) розподіленого додатка (Simple Object Access Protocol);
- UDDI – стандарт інтеграції додатків (Universal Description, Discovery and Integration);
- WSDL – універсальна мова опису веб-сервісів (Web Service Description Language);

а також цілий ряд інших менш уживаних протоколів.

Компонентний підхід до програмування

Одним з принципових технологічних переваг проектування і реалізації програмного забезпечення, що декларуються Microsoft, є так званий компонентний підхід до програмування.

У своїй основі цей підхід збігається з традиційним об’єктно-орієнтованим, однак має ряд важливих особливостей. Оскільки основною метою є гетерогенне компонентне програмування. Необхідно з самого початку усвідомити суть основних понять, на яких грунтується компонентний підхід.

Центральної концепцією підходу (і це очевидно вже з назви) є поняття ***компонента***.

Під компонентою в подальшому будемо розуміти незалежний модуль програмного забезпечення, який можливо повторно використовувати, а також тиражувати.

На відміну від «традиційних» об'єктів ООП компоненти мають наступні властивості:

- в цілому компонент володіє більш ***високим рівнем абстракції*** в порівнянні з об'єктом (якщо під останнім розуміють конструкцію рівня мови програмування);
- компоненти можуть містити в своєму складі ***множинні класи***;
- компоненти з точки зору користувача є ***інваріантами*** (незмінними і такими що не містять протирічь) по відношенню до тієї мови програмування, на якій вони реалізовані.

Таким чином, виявляється, що в загальному випадку розробник і користувач компоненти можуть бути територіально розділені і можуть використовувати різні мови програмування в рамках єдиного середовища розробки додатків Microsoft .NET.

Перш за все, охарактеризуємо компонентну модель Microsoft, яка зазвичай іменується в літературі абревіатурою COM (що походить від слів Component Object Модель).

Компонентна об'єктна модель COM є основним стандартом для Microsoft компонентного проектування і реалізації програмного забезпечення. На сьогодні це найрозвиненіша, і, мабуть, найбільш вдала в практичному плані модель, яка практично забезпечує можливість ініціалізації і використання компонентів як всередині одного процесу, так і між процесами або між комп'ютерами незалежно від мови реалізації. COM-модель підтримується в ідеології .NET для цілого ряду мов програмування (C#, SML, Visual Basic, C++ та ін.), є основою для ActiveX, OLE, а також для багатьох інших технологій Microsoft.

На відміну від COM, модель Java Beans, базовий стандарт компанії Sun Microsystems для компонент, є залежною від мови реалізації.

Порівняння КОП і ООП

В об'єктно-орієнтованому підході ключовими є, зокрема, поняття ***класу*** і ***інтерфейсу***. Зауважимо, що в компонентно-орієнтованому підході ці поняття також є системоутворюючими.

При цьому під ***класом*** розуміється базова сутність, яка визначається як сукупність своїх елементів.

Під ***інтерфейсом*** розуміється набір семантично пов'язаних абстрактних елементів.

Для компонентно-орієнтованого підходу поняття інтерфейсу має першорядне значення, оскільки виключно за допомогою цього механізму

клієнт в архітектурі з COM моделлю може безпосередньо здійснювати взаємодію з COM-класом.

Зауважимо, що інтерфейси підвищують безпеку коду, тому що взаємодія з об'єктом відбувається не безпосередньо, а через покажчик (посилання). Поняття ***властивості*** (як атрибута об'єкта) і ***методу*** (як операції над об'єктом), також як і ***механізму подій*** (співвідношення над об'єктами предметної області) властиві обом підходам.

Принципово новим є наявність в COM-моделі ***збірок*** – самодостатніх одиниць інформації для інсталяції та поширення програмних продуктів.

В цілому COM-підхід є більш зручним з практичної точки зору, хоча механізми, реалізовані в ньому, принципово можна порівняти з можливостями ООП.

.NET – найбільш істотні недоліки

Незважаючи на перераховані вище інновації в області теорії, технології та практичної реалізації, з огляду на масштабованість ідеології і новизни досліджуваної проблематики, підхід .NET не позбавлений окремих недоліків, більшість з яких, мабуть, носить тимчасовий характер. Відзначимо, найбільш істотні з них.

По-перше, розробники відзначають порівняно високі вимоги до апаратного забезпечення (зокрема, обсяг оперативної пам'яті повинен бути не менше 256 мегабайт, вільний обсяг жорсткого диску для роботи з Microsoft Visual Studio .NET – не менше 10 гігабайт).

По-друге, некомерційні версії програмних продуктів Microsoft, які найчастіше надають нові суттєві можливості, в недостатній мірі стійкі в роботі; документація по ряду нових функцій програмного забезпечення представлена не в повному обсязі.

По-третє, підтримка ряду теоретично цікавих і практично корисних мов програмування реалізована в обмеженому обсязі (скажімо, компілятор для мови програмування SML для Visual Studio .NET знаходиться в процесі реалізації). Оскільки цілий ряд компіляторів для мов програмування надається сторонніми по відношенню до Microsoft компаніями-розробниками або некомерційними установами, результати їх діяльності піддаються контролю і доопрацюванню з обмеженнями.

По-четверте, комплекс програмно-інструментальних засобів, який реалізує підхід .NET (включаючи і компілятори для мов програмування) ратифікований за міжнародними стандартам не в повному обсязі.

Висновки

Безумовно, .NET є видатним досягненням сучасної індустрії програмування. Досить сказати, що корпорація Microsoft вважає саме .NET своєю стратегічною ідеологією і технологічною платформою на найближче десятиліття.

Безсумнівна якісна перевага над існуючими засобами автоматизованого проектування і швидкої реалізації прикладного програмного забезпечення (зокрема, Inprise Delphi і JBuilder, Oracle Developer, Microsoft Visual Studio та ін.) досягається за рахунок наступних основних факторів:

- інтероперабельність і міжмовна взаємодія;
- багаторівнева, гнучка і надійна політика безпеки;
- інтеграція з технологією веб-сервісів;
- спрощення процедури розгортання і використання створюваного програмного забезпечення.

Незважаючи на деяку незавершеність рішення для широкого комерційного використання в силу концептуальної новизни і грандіозності проекту, підхід .NET, безумовно, значно впливає на комерційну індустрію програмування в цілому і сприяє радикальному вдосконаленню галузі під час ринкової конкуренції.

Список використаних джерела

1. Введение в платформу .NET FrameWork [Електронний ресурс] — Режим доступу : http://skillcoding.com/Default.aspx?id=79
2. Общие сведения о платформе .NET Framework [Електронний ресурс] — Режим доступу : https://msdn.microsoft.com/ru-ru/library/zw4w595w(v=vs.110).aspx
3. Меженин М. Microsoft .NET. Введение в .NET : лекции / Меженин Михаил // кафедра "Системное программирование", ВМИ, ЮУрГУ, 2014
4. Nathan A. .NET and COM: The Complete Interoperability Guide. / Adam Nathan. — Sams, 2002. — 1608 p.
5. Box D. Essential .NET, Vol.1: The Common Language Runtime / Don Box, Chris Sells. — Addison Wesley, 2002. — 432p.
6. Grimes F. Microsoft .NET for Programmers / Fergal Grimes. — Manning Publications, 2002. — 386 p.
7. Richter J. Applied Microsoft .NET Framework Programming / J.Richter . — Microsoft Press, 2002. — 556 p

Лекція 6

ПРОЦЕС КЕРОВАНОГО ВИКОНАННЯ

Процес керованого виконання включає наступні кроки.

- **Вибір компілятора**. Щоб скористатися перевагами середовища CLR, необхідно використовувати один або кілька мовних компіляторів, які звертаються до середовища виконання.
- **Компіляція коду в MSIL.** При компіляції вихідний код перетвориться в MSIL і створюються необхідні метадані.
- **Компіляція інструкцій MSIL в машинний код.** Під час виконання JIT-компілятор перетворює інструкції MSIL в машинний код. Під час цієї компіляції виконується перевірка коду і метаданих MSIL з метою встановити, чи можна для них визначити, чи є код типобезпечним.
- **Виконання коду.** Середовище CLR надає інфраструктуру, яка забезпечує виконання коду, і ряд служб, які можна використовувати при виконанні.

Вибір компілятора

Щоб скористатися перевагами, наданими середовищем CLR, необхідно застосувати один або кілька мовних компіляторів, орієнтованих на середовище виконання, наприклад компілятор Visual Basic, C#, Visual C++, F# або один з багатьох компіляторів від незалежних розробників, наприклад компілятор Eiffel, Perl або COBOL. Оскільки середовище виконання є багатомовним, воно підтримує широкий набір різноманітних типів даних і мовних засобів. Доступні засоби середовища виконання визначаються використовуваним мовним компілятором, і розробники створюють код з використанням цих засобів.

Використовуваний в коді синтаксис визначається компілятором, а не середовищем виконання**.** Якщо компонент повинен бути повністю доступний для компонент, написаних на інших мовах, то експортовані цим компонентом типи повинні надавати виключно мовні функції, включені до складу специфікації CLS (CLS). Атрибут CLSCompliantAttribute дозволяє гарантувати, що код є CLS-сумісним.

Компіляція в MSIL

При компіляції в керований код компілятор перетворює вихідний код в проміжну мову Microsoft (MSIL), що представляє собою незалежний від процесора набір інструкцій, який можна ефективно перетворити в машинний код. *Мова MSIL включає інструкції для: завантаження, збереження, ініціалізації і виклику методів для об'єктів, а також інструкції для арифметичних і логічних операцій, потоків управління, прямого доступу до пам'яті, обробки виключень і інших операцій.*

Перед виконанням код MSIL необхідно перетворити в код для конкретного процесора, зазвичай за допомогою JIT-компілятора. Оскільки середовище CLR надає для кожної підтримуваної комп'ютерної архітектури один або кілька JIT-компіляторів, один набір інструкцій MSIL можна компілювати і виконувати в будь-який підтримуваній архітектурі.

Коли компілятор створює код MSIL, одночасно створюються метадані. *Метадані містять опис типів в коді, включаючи визначення кожного типу, сигнатури кожного члена типу, члени, на які є посилання в коді, а також інші відомості, що використовуються середовищем виконання під час виконання*. MSIL і метадані містяться в переносимому виконуваному (PE) файлі, який ґрунтується на форматах Microsoft PE і COFF, що раніше використовувалися для виконуваного контенту, але при цьому розширює їх можливості.

PE- Portable Executable – формат виконуваних файлів, об'єктного коду та динамічних бібліотек, використовуваний в 32- і 64-бітових версіях операційної системи Microsoft Windows. Формат PE є структурою даних, що містить всю інформацію, необхідну PE-завантажувачу для відображення файлу в пам'ять. Виконуваний код включає в себе посилання для зв'язування динамічно-завантажуваних бібліотек, таблиці експорту і імпорту API функцій, дані для управління ресурсами і дані локальної пам'яті потоку (TLS). В операційних системах сімейства Windows NT формат PE використовується для EXE, DLL, SYS (драйверів пристроїв) та інших типів виконуваних файлів.

Цей формат файлів, що дозволяє розміщувати код MSIL або машинний код, а також метадані, дозволяє операційній системі розпізнавати образи середовища CLR. Наявність у файлі метаданих поряд з MSIL дозволяє коду описувати себе. Це усуває потребу в бібліотеках типів і у мові IDL (Interactive Data Language – мова програмування для аналізу даних, використовується в астрономії та медичній візуалізації). Середовище виконання знаходить і видобуває метадані з файлу в міру необхідності при виконанні.

Компіляція MSIL в машинний код

Перед запуском MSIL його необхідно скомпілювати в машинний код в середовищі CLR для архітектури кінцевого комп'ютера. Платформа .NET Framework надає два способи такого перетворення:

- JIT-компілятор платформи .NET Framework;
- .NET Framework ***ngen.exe*** (генератор образів в машинному коді).

Компіляція з допомогою JIT-компілятора

При JIT-компіляції (рис. 15) мова MSIL перетвориться в машинний код під час виконання додатка за вимогою, коли завантажується і виконується

вміст збірки. Оскільки середовище CLR надає JIT-компілятор для кожної підтримуваної архітектури процесора, розробники можуть створювати набір збірок MSIL, які можуть компілюватися за допомогою JIT-компілятора і виконуватися на різних комп'ютерах з різною архітектурою. Якщо *керований код* викликає специфічний для платформи машинний API або бібліотеку класів, то він буде виконуватися тільки у відповідній операційній системі.

При JIT-компіляції враховується можливість того, що певний код може ніколи не викликатися під час виконання. Щоб не витрачати час і пам'ять на перетворення всього, що міститься в PE-файл, при компіляції MSIL перетвориться в машинний код в міру необхідності під час виконання. Отриманий таким чином машинний код зберігається в пам'яті, що дозволяє використовувати його при подальших викликах в контексті цього процесу. Завантажувач створює і приєднує *заглушки* до кожного методу в типі, коли тип завантажується і ініціалізується. При першому виклику методу заглушка передає управління JIT-компілятору, який перетворює MSIL для цього методу в машинний код і замінює заглушку на створений машинний код. Тому наступні виклики методу, скомпільованого за допомогою JIT-компілятора, ведуть безпосередньо до машинного коду.

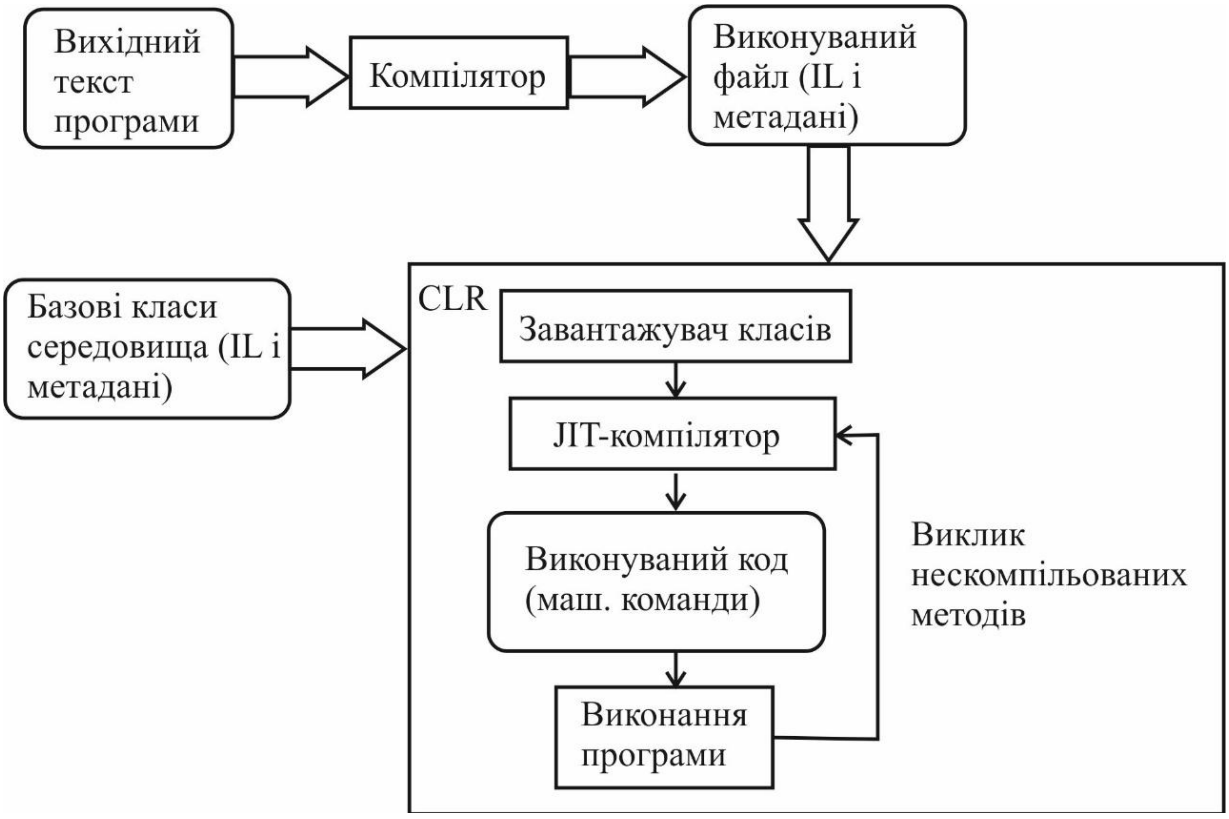

Рис. 15. Схема виконання програми в .NET

Створення коду під час установки за допомогою NGen.exe

Той факт, що JIT-компілятор перетворює MSIL-код збірки в машинний код при виклику окремих методів, визначених у цій збірці, негативно позначається на продуктивності під час виконання. У більшості випадків зниження продуктивності не є важливим. Що важливіше, код, створений JIT-компілятором, буде прив'язаний до процесу, який викликав компіляцію. Його не можна зробити загальним для декількох процесів. Щоб створений код можна було використовувати в декількох викликах додатків або в декількох процесах, які спільно використовують набір збірок, середовище CLR надає режим попередньої компіляції. У такому режимі компіляції для перетворення збірок MSIL в машинний код в стилі JIT-компілятора використовується програма **Ngen.exe** (генератор образів в машинному коді). Однак, робота Ngen.exe відрізняється від JIT-компілятора в трьох аспектах:

- Ngen.exe виконує перетворення з MSIL-коду в машинний код перед виконанням програми, а не під час його виконання;
- при цьому збірка компілюється повністю, а не по одному методу за раз;
- він зберігає створений код в кеші засобу виконання машинного коду у вигляді файлу на диску.

Перевірка коду

В процесі компіляції в машинний код MSIL-код повинен пройти перевірку, якщо тільки адміністратор не встановив політику безпеки, що дозволяє пропустити перевірку коду. MSIL-код і метадані перевіряються на типобезопеку. Це означає, що код повинен звертатися тільки до тих адрес пам'яті, до яких йому дозволено доступ. Типобезпека допомагає ізолювати об'єкти один від одного і сприяє їх захисту від ненавмисного або зловмисного пошкодження. Вона також гарантує надійне застосування умов безпеки для коду.

Середовище виконання використовує істинність наступних тверджень для коду, що піддається типобезпеці:

- посилання на тип строго сумісна з типом, що адресується;
- для об'єкта викликаються тільки правильно визначені операції;
- посвідчення є справжніми.

У процесі перевірки коду MSIL робиться спроба підтвердити, що код може отримувати доступ до розміщення в пам'яті і викликати методи тільки через правильно визначені типи. Наприклад, код не повинен дозволяти доступ до полів об'єкта так, щоб можна було виходити за межі розміщення в пам'яті. Крім того, перевірка визначає, чи правильно був створений код MSIL, оскільки невірний код MSIL може призводити до порушення правил суворої типізації. У процесі перевірки передається правильно визначений типобезпечний код. Однак іноді типобезпечний код може не пройти перевірку через обмеження процесу перевірки, а деякі мови за своєю структурою не дозволяють створювати код, що піддається перевірці на типобезпеку. Якщо відповідно до політики безпеки використання

типобезпечного коду є обов'язковим і код не проходить перевірку, то при виконанні коду можна створити виняток.

Виконання коду

Середовище CLR це інфраструктура, що забезпечує кероване виконання коду, і ряд служб, які можна використовувати при виконанні. Перед виконанням методу його необхідно скомпілювати в код для конкретного процесора. Кожен метод, для якого створено MSIL-код, компілюється за допомогою JIT-компілятора при першому виклику і потім запускається. При наступному виклику методу буде виконуватися існуючий JIT-скомпільований код. Процес JIT-компіляції та подальшого виконання коду повторюється до завершення виконання.

Під час виконання для керованого коду доступні такі служби, як прибирання сміття, забезпечення безпеки, взаємодія з некерованим кодом, підтримка налагодження на декількох мовах, а також підтримка розширеного розгортання і управління версіями.

У Microsoft Windows XP і Windows Vista завантажувач операційної системи виконує пошук керованих модулів шляхом аналізу біта в заголовку COFF (*Common Object File Format (COFF) – формат виконуваних файлів, файлів об'єктного коду та динамічних бібліотек*). Встановлений біт позначає керований модуль. При виявленні керованих модулів завантажується бібліотека Mscoree.dll, а підпрограми CorValidateImage і CorImageUnloading повідомляють завантажувач про завантаження та вивантаження образів керованих модулів. Підпрограма _CorValidateImage виконує наступні дії:

- перевіряє, чи є код допустимим керованим кодом;
- замінює точку входу в образі на точку входу в середовищі виконання;

У 64-розрядних системах Windows CorValidateImage змінює образ, що знаходиться в пам'яті, шляхом перетворення його з формату PE32 в формат PE32+.

Висновки

Загальна схема використання збірки в середовищі CLR подана на рис 16.

1. Під час запуску збірки завантажувач розпізнає .Net додаток і передає управління середовищу CLR, далі завантажувач класів знаходить і завантажує клас, в якому міститься точка початку роботи збірки (зазвичай функція Main), потім CLR передає управління збірці.

2. Завантажувач класів виконується кожен раз при першому зверненні до необхідних для виконання збірки класів. Завантажувач знаходить потрібний клас, розміщує інформацію про типи класу в кеші і завантажує клас для виконання (при цьому в методах класу, що завантажується робиться відмітка, що вони не є ще готовими для виконання, оскільки не пройшли через JIT-компіляцію).

3. Верифікатор є частиною JIT-компілятора і відповідає за перевірку коректності CIL-коду і метаданих. Подібний контроль підвищує надійність роботи програм, з іншого боку, верифікація може і не здійснюватися (наприклад, для довіреного коду).

4. JIT-компілятори викликаються при зверненні до CIL-коду, який раніше при роботі програми ще не компілювався. В результаті компіляції виходить машинний код, оптимізований для обраної платформи, для відкомпільованого методу встановлюється адреса отриманого машинного коду і управління передається виконуваній збірці. Як результат, компіляція методів класів здійснюється тільки в момент першого звернення до них. Подібна схема компіляції в міру необхідності може істотно знизити розміри породжуваного програмного коду і скоротити час підготовки збірки для виконання, разом з цим, в середовищі CLR може бути виконана і повна компіляція збірки перед початком виконання (pre-JITing).

5. Безпосереднє виконання машинного коду також відбувається при активній взаємодії з середовищем CLR. Функції середовища виконання полягає в управлінні вільною пам’яттю, обробці виключень, підтримки безпеки і ін.

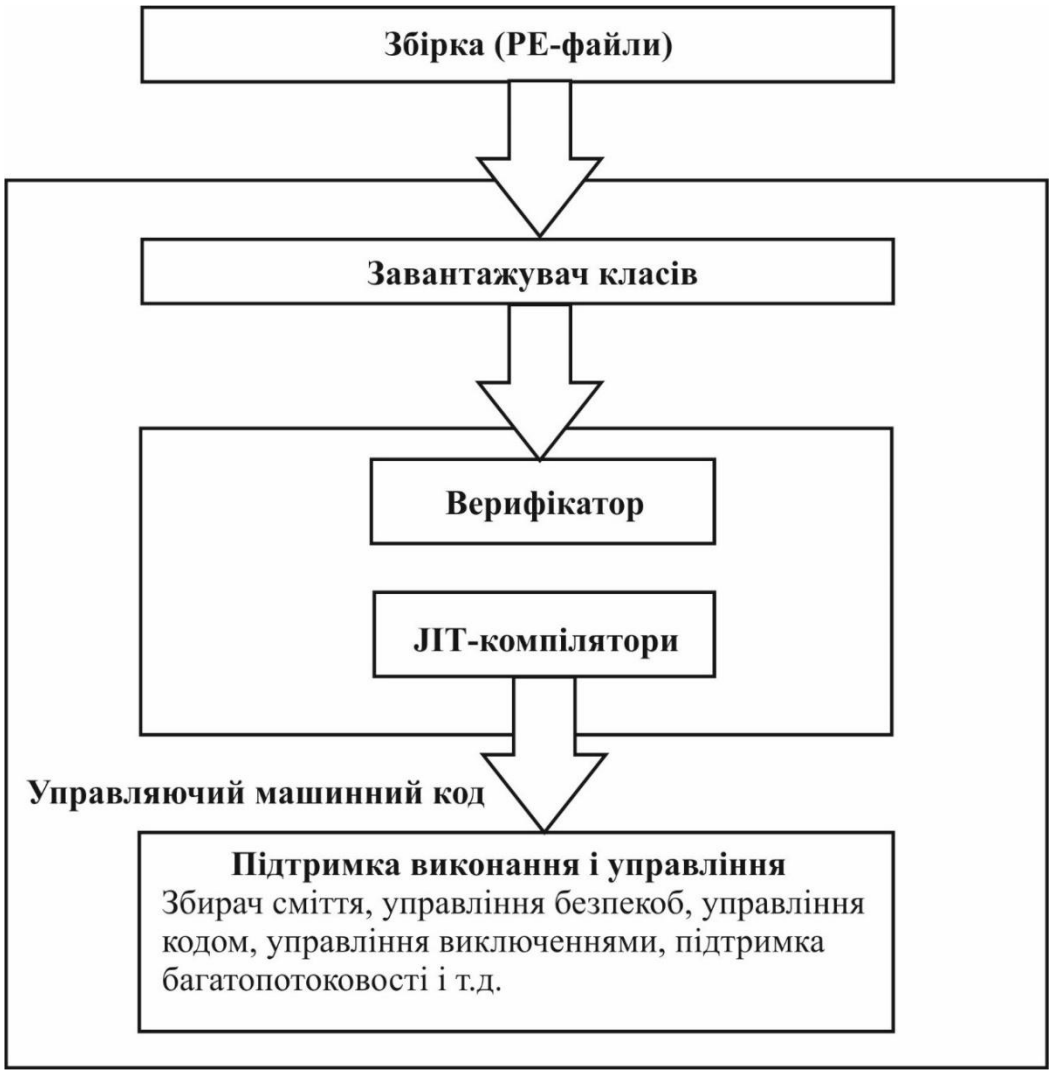

Рис. 16. Середовище CLR: загальна схема використання збірки

Список використаних джерела

1. Введение в платформу .NET FrameWork [Електронний ресурс] — Режим доступу : http://skillcoding.com/Default.aspx?id=79
2. Общие сведения о платформе .NET Framework [Електронний ресурс] — Режим доступу : https://msdn.microsoft.com/ru-ru/library/zw4w595w(v=vs.110).aspx
3. Меженин М. Microsoft .NET. Введение в .NET : лекции / Меженин Михаил // кафедра "Системное программирование", ВМИ, ЮУрГУ, 2014.
4. Nathan A. .NET and COM: The Complete Interoperability Guide. / Adam Nathan. — Sams, 2002. — 1608 p.

5. Box D. Essential .NET, Vol.1: The Common Language Runtime / Don Box, Chris Sells. — Addison Wesley, 2002. — 432p.
6. Grimes F. Microsoft .NET for Programmers / Fergal Grimes. — Manning Publications, 2002. — 386 p.
7. Richter J. Applied Microsoft .NET Framework Programming / J.Richter . — Microsoft Press, 2002. — 556 p

ЗМІСТ

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

УДК УДК 004.655

Навчально-методичне видання

Булатецький Віталій Вікторович
Булатецька Леся Віталіївна

ТЕХНОЛОГІЇ ПРОМІЖНОГО КОДУ В КОРПОРАТИВНИХ ІНФОРМАЦІЙНИХ СИСТЕМАХ

Текст лекцій нормативної навчальної дисципліни “Платформи корпоративних інформаційних систем”
Гарн. Times. Обсяг 2,79 ум. друк. арк.
Наклад 30