Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

Конспект лекцій з навчальної дисципліни **"Розподілені сервісні системи"** для студентів спеціальності «Телекомунікації та радіотехніка» відділу Інфокомунікацій та інженерії

Чернівці
Чернівецький національний університет
імені Юрія Федьковича
2021

Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

Конспект лекцій з навчальної дисципліни
“Розподілені сервісні системи”
для студентів спеціальності «Телекомунікації та радіотехніка» відділу Інфокомунікацій та інженерії

Ухвалено Методичною радою відділу Інфокомунікацій та інженерії
(протокол № 10 від 29. 05. 2021 р.)

Чернівці
Чернівецький національний університет
імені Юрія Федьковича
2021

Конспект лекцій з навчальної дисципліни «Розподілені сервісні системи» для студентів всіх форм навчання спеціальності «Телекомунікації та радіотехніка» відділу Інфокомунікацій та інженерії. / Уклад.: О.Д. Архелюк – Чернівці.: ЧНУ імені Юрія Федьковича 2021. – 94 с.

Навчальне видання

Конспект лекцій з навчальної дисципліни

РОЗПОДІЛЕНІ СЕРВІСНІ СИСТЕМИ

Укладач: *Архелюк Олександр Дмитрович, канд. фіз.-мат. наук, доц.*

В курсі лекцій розглянуто архітектура інформаційних систем, організація розподілених обчислень з використанням серверів додатків, віддалений виклик процедур, компонентні системи, сервіс-орієнтована архітектура, веб-сервіси, основи хмарних обчислень, технології Грід.

Для студентів технічних факультетів за спеціальністю «Інформаційні мережі зв’язку».

Відповідальний редактор

Максимяк Петро Петрович, докт. фіз.-мат. наук, професор.

Зміст

Вступ	5
Тема 1. Вступ в розподілені обчислювальні системи	6
Тема 2. Архітектура інформаційних систем	14
Тема 3. Організація розподілених обчислень з використанням серверів додатків	24
Тема 4. Віддалений виклик процедур	34
Тема 5. Компонентні системи	48
Тема 6. Сервіс-орієнтована архітектура	61
Тема 7. Веб-сервіси	67
Тема 8. Основи хмарних обчислень	75
Тема 9. Технології Грід	83
Питання для самостійної роботи	93
Список використаної літератури	94

Вступ

Інформаційні системи забезпечують збирання, зберігання, обробку, пошук, видачу інформації, яка потрібна в процесі прийняття рішень задач з будь-якої галузі. Вони допомагають аналізувати проблеми та створювати нові продукти. Інформаційною системою (ІС), або автоматизованою ІС (АІС), будемо називати програмно-апаратну систему, призначену для автоматизації цілеспрямованої діяльності кінцевих користувачів, що забезпечує у відповідність із закладеною в неї логікою обробки можливість отримання, модифікації та зберігання інформації. Сучасне розуміння інформаційної системи передбачає використання як основний технічний засіб переробки інформації комп'ютера. У великих організаціях поруч із персональними комп'ютерами у складі технічної бази інформаційної системи може входити мейнфрейм чи суперевм. Крім того, технічне втілення інформаційної системи саме по собі нічого не буде означати, якщо не враховано роль людини, для якої призначена інформація, що виробляється, і без якої неможливе її отримання та подання. Необхідно розуміти різницю між комп'ютерами та інформаційними системами. Комп'ютери, що оснащені спеціалізованими програмними засобами, є технічною базою та інструментом для інформаційних систем. Інформаційна система немислима також без персоналу, що взаємодіє з комп'ютерами та телекомунікаціями. Інформаційна система є середовищем, складовими елементами якого є комп'ютери, комп'ютерні мережі, програмні продукти, бази даних, люди, різного роду технічні та програмні засоби зв'язку і т.д. Основна мета інформаційної системи - організація зберігання та передачі інформації. Інформаційна система є людинокомп'ютерною системою обробки інформації. В результаті еволюційного розвитку технології зберігання та використання даних утвердилася концепція автоматизованих інформаційних систем (АІС) або банків та баз даних. Широкого поширення набули бази даних, побудовані з урахуванням різних підходів до організації даних. В результаті еволюційного розвитку технології зберігання та використання даних утвердилася концепція автоматизованих інформаційних систем (АІС) або банків та баз даних. Широкого поширення набули бази даних, побудовані з урахуванням різних підходів до організації даних. В результаті еволюційного розвитку технології зберігання та використання даних утвердилася концепція автоматизованих інформаційних систем (АІС) або банків та баз даних. Широкого поширення набули бази даних, побудовані з урахуванням різних підходів до організації даних.

Тема 1. Вступ в розподілені обчислювальні системи

Формального визначення розподіленої системи (РС) в теперішній час не існує. З багатьох різних визначень, можна виділити іронічне визначення першого лауреата премії Дейкстри за досягнення в області розподілених обчислень Леслі Лампертa:

*«**Розподіленою системою** можна назвати таку систему, в якій відмова комп'ютера, про існування якого ви навіть не підозрювали, може зробити ваш особистий комп'ютер не придатним до використання».*

Це визначення він дав в травні 1987 року, в своєму листі колегам з приводу чергового відключення електроенергії в машинному залі.

Трохи більш суворе визначення, якого ми в подальшому і будемо дотримуватися, запропонував Ендрю Таненбаум в своїй фундаментальній праці «Розподілені системи. Принципи та парадигми »:

*«**Розподілена система** – це набір з'єднаних каналами зв'язку незалежних комп'ютерів, які з точки зору користувача деякого програмного забезпечення виглядає єдиним цілим».*

В цьому визначенні фіксуються два істотних моменти, характерних для РС:

- Автономність вузлів;
- Представлення системи користувача, як єдиної структури.

При цьому, основною сполучною ланкою розподілених систем є програмне забезпечення.

Класифікація розподілених систем

Проміжне програмне забезпечення

Розподілена обчислювальна система (РОС) являє собою програмно-апаратний комплекс, орієнтований на вирішення певних задач. З одного боку, кожен обчислювальний вузол є автономним елементом. З другого боку, програмна складова РОС повинна забезпечувати користувачам видимість роботи з єдиною обчислювальною системою. У зв'язку з цим виділяють наступні

Важливі характеристики РОС:

1. Можливість роботи з різними типами пристроїв:
   - З різними постачальниками пристроїв;
   - З різними операційними системами,
   - З різними апаратними платформами.

Обчислювальні середовища, які складаються з багатьох обчислювальних систем на базі різних програмно-апаратних платформ, називаються **гетерогенними.**

2. Можливість простого розширення і масштабування;
3. Перманентна (постійна) доступність ресурсів (навіть якщо деякі елементи РОС деякий час можуть знаходиться поза доступом);
4. Приховування особливостей комунікації від користувачів.

Для забезпечення роботи гетерогенного обладнання РОС у вигляді одного цілого, стек програмного забезпечення (ПЗ) зазвичай розбиваються на 2 шари.

На верхньому шарі озташовуються розподілені додатки, які відповідають за вирішення певних прикладних задач засобами РОС.

Їх функціональні можливості базуються на нижньому шарі – проміжним програмним забезпеченням (ППЗ), яке взаємодіє з системним ПЗ і мережевим рівнем, для забезпечення прозорості роботи додатків РОС.

Для того, щоб РОС могла бути представлена користувачу як єдина система, використовують наступні **типи прозорості РОС:**

- ***прозорий доступ до ресурсів*** - від користувачів повинна бути прихована різниця в представлені даних і в способах доступу до ресурсів РОС;
- ***прозоре місце розташування ресурсів*** - місце фізичного розташування необхідного ресурсу повинно бути несуттєво для користувача;
- ***реплікація*** - приховування від користувача того, що в реальності існує більше однієї копії використовуваних ресурсів;
- ***паралельний доступ*** - можливість спільного (одночасного) використання одного і того ж ресурсу різними користувачами незалежно один від одного. При цьому факт спільного використання ресурсу повинен залишатися прихованим від користувача;
- ***прозорість відмов*** - відмова (відключення) будь-яких ресурсів РОС не повинен впливати на роботу користувача і його додатки.

Термінологія РОС

Ресурс - будь-яка програмна або апаратна сутність, представлена або використовувана в розподіленій мережі (наприклад, комп'ютер, пристрій зберігання, файл, комунікаційний канал, сервіс і т. д.)

Вузол – будь-який апаратний пристрій в розподіленій обчислювальній системі.

Сервер - це постачальник інформації в РОС (наприклад, веб-сервер).

Клієнт - це споживач інформації в РОС (наприклад, веб-браузер).

Пір - це вузол, який поєднує в собі як клієнтську, так і серверну частину (тобто і постачальник, і споживач інформації одночасно).

Сервіс - це мережева сутність, що надає певні функціональні можливості (наприклад, веб-сервер може надавати сервіс передачі файлів по протоколу HTTP). В рамках одного вузла можуть надаватися кілька різних сервісів.

Кожен комп'ютер або пристрій являє собою сутність в розподіленій обчислювальній системі у вигляді вузла. При цьому на кожному вузлі може розташовуватися кілька клієнтів, серверів, сервісів або пірів. Важливо зауважити, що будь-який вузол, сервер, пір або сервіс (але не клієнт!) є ресурсами РОС.

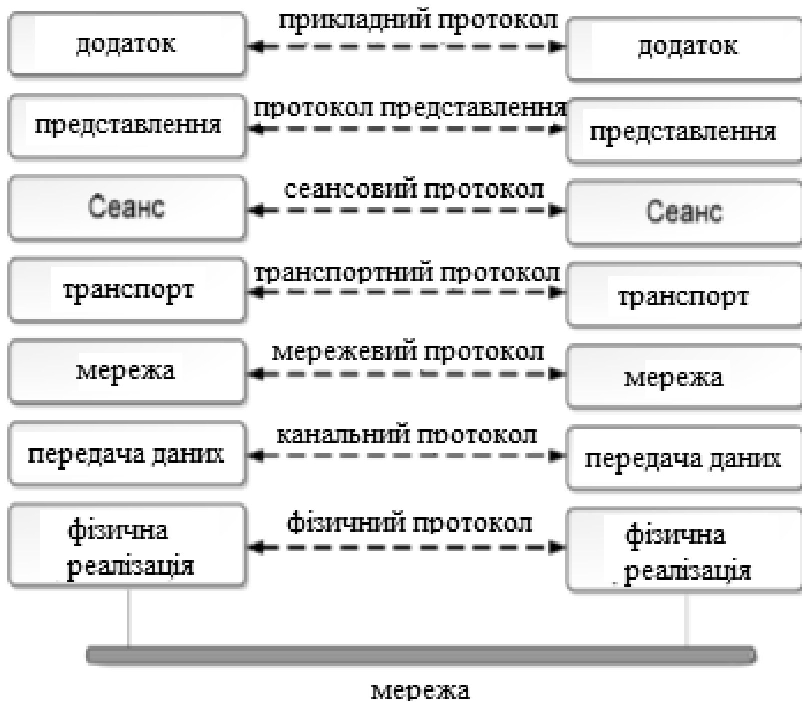
Сервіс отримує запит на надання певних даних (майже як аргументи, що передаються при виклику локальної функції) і повертає відповідь. Таким чином, сервіс можна визначити як певну заміну виклику функції на локальному комп'ютері. Існує безліч технологій, що забезпечують створення і супровід сервісів в РОС: технологія XML веб-сервісів, сервіси REST і ін.

Класифікація РОС

	По методу знаходження ресрсів	По доступності ресурсів	По методу взаємодії ресурсів
Централізовані	запит на пошук відправляється на деякий центральним сервер (приклад: служба DNS (Domain Name System-система доменних імен), яка по імені сайту вертає його IP - адресу)	існує один сервер, який надає певний ресурс або сервіс (приклад: веб-сервіс)	взаємодія між вузлами виконується через центральний сервер, один вузон не може звернутись до іншого напряму
Децентралізовані	запит на пошук відправляється всім вузлам, які відомі відправнику (приклад: система Gnutella)	кожен вузол грає роль клієнта та сервера, який може надавати ресурси і сервіси, аналогічні іншим пристроям даної мережі (приклади: BitTorrent, Gnutella, Napster)	пряма взаємодія між вузлами РРС, оскільки кожен вузол грає роль сервера та клієнта (приклад: однорангові розрахункові системи)

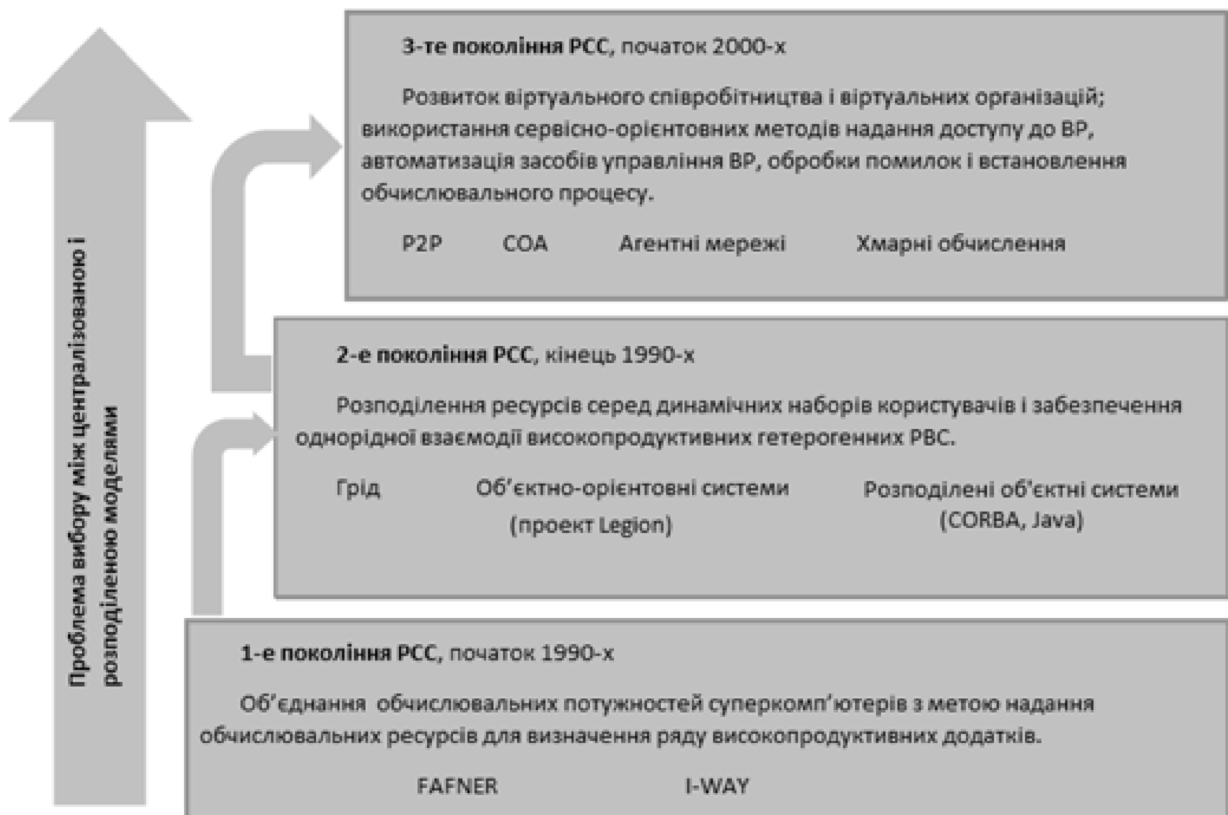
Зв’язок в РОС

Поняття *«розподілена обчислювальна система»* має на увазі, що компоненти такої системи розподілені, тобто віддалені один від одного. Очевидно, що функціонування подібних систем неможливо без ефективного зв'язку між її компонентами, яка реалізується за допомогою об'єднання пристроїв, що належать до РОС, в мережі.



Взаємодія в обчислювальних мережах базується на ***протоколах***, що представляють собою набори правил і угод, що описують процедуру взаємодії між компонентами системи. В області обчислювальних комунікацій вже тривалий час існує загальноприйнята система протоколів - мережева модель OSI (англ. Open Systems Interconnection basic reference model - базова еталонна модель взаємодії відкритих систем), яка являє собою стек протоколів різного рівня.

Історія розвитку розподілених обчислень



Проект FAFNER (Factoring via Network - Enabled Recursion – мережеве розкладання на множники за допомогою рекурсії) був створений в 1995 році, для вирішення задачі розкладання великих чисел на основі потужностей географічно – розподілених обчислювальних тільних систем (ВС). Знаходження простих множників великих чисел дозволяє розшифрувати дані, які зашифровані на основі алгоритму RSA (абревіатура із перших букв фамілій розробників даного методу : Rivest, Shamir і Adleman).

Особливості:

- реалізація NFS – Network File System (Мережева Файлова Система) дозволяла навіть малим робочим станціям (з 4 Мб оперативної пам'яті) виконувати корисну роботу, розраховуючи свій маленький фрагмент задачі;
- анонімна реєстрація учасників;
- ієрархічна структура веб-серверів, які представляли кістяк ВС, що зменшувало можливість виникненню «вузького місця» у ВС.

I-WAY – Information Area Year (Рік Інформації Глобальних Мереж) – експериментальна високопродуктивна мережа, спроектована на початку 1995, з метою об'єднання високошвидкісних мереж, які існували на той час. В рамках проекту, була побудована апаратна інфраструктура, за допомогою якої здійснювався доступ до ресурсів мережі I-WAY. Вона складалась із базових робочих станцій під керівництвом операційної системи UNIX, на яких було встановлено спеціальне ППО (сервер I-POP). Система I-POP брала на себе функції шлюзу до ресурсам мережі I-WAY. Кожен такий сервер підтримував стандартні процедури аутентифікації, резервування ресурсів створення процесів і комунікації. Проект I-WAY використовувався для вирішення наступних задач: супер комп'ютерні обчислення; доступ до віддалених ресурсів; задачі віртуальної реальності.

Проекти високопродуктивних обчислювальних систем, реалізовані на початку 1990-х років, виявили основні проблеми, які необхідно буду вирішувати для розгортання стабільних РОС. Для вирішення усіх вказаних задач була розроблена концепція **грід** – РОС, яка забезпечує *«гнучке, безпечне, координоване розподілення ресурсів серед динамічних наборів користувачів, організацій і ресурсів».*

Основною задачею грід була побудова інфраструктура, яка забезпечує *«обчислення за вимогою».* Для того, щоб називатись «грід», система повинна відповідати наступним **вимогам**:

1. **Гетерогенність.** Обчислювальне середовище грід може складатись з безлічі різних ресурсів, що володіють різними характеристиками та параметрами.
2. **Масштабованість.** Грід може складатись з як завгодно великого числа ресурсів.
3. **Пристосовність.** Середовище грід може складатись із сотні комп'ютерів, і помилки в роботі десятка з них не повинні впливати на отримані результати рішення.

В якості прикладу реалізації грід концепції можна назвати проект Globus, запущений в 1997 році.

Наступний кроком в розвитку РОС стала поява систем, заснованих на **об'єктно – орієнтованому підході**. Однією з перших таких систем став **проект Legion,** запущений в листопаді 1997 році, який надавав програмну оболонку для організації однорідної взаємодії високопродуктивних гетерогенних РОС. Метою проекту було *представлення користувачам єдиної інтегрованої інфраструктури РОС , незалежно від масштабу, географічного положення, мови та операційної системи*. Legion надавав користувачу набір об'єктів, що надають базові сервіси:

- о*б'єкти обчислювачів* – абстракції, які реалізують базові принципи роботи із обчислювальними ресурсами;
- *об'єкти систем збереження даних* – абстракції, які надають базові методи роботи із системами збереження даних;
- *об'єкти зв'язування* – об'єкти, які забезпечують зв'язок між абстрактним ідентифікатором об'єкту та його фізичним адресом;
- *об'єкти контексту* – об'єкти, які реалізують проекцію користувацьких імен об'єктів на абстрактні ідентифікатори об'єктів в системі Legion.

При розгляді другого покоління розподілених обчислювальних систем, не можна залишити поза увагою таких клас, як ***розподілені об'єктні системи.*** Вони надають базові метода для реєстрації, *серіалізації* (процес переводу будь-якої структури даних у послідовність бітів) та *десеріалізації* (відновлення початкового стану структури даних з бітової послідовності) об'єктів забезпечуючи віддалений виклик методів.

Серед представників даного класу слід виділити архітектуру CORBA і модель Java.

CORBA (Common Object Request Broker Architecture – загальна архітектура брокера об'єктних запитів) – протокол взаємодії об'єктно – орієнтованих систем, опублікований в 1997-1998 роках консорціумом OMG (Object Management Group). Специфікація CORBA наказує об'єднання програмного коду в об'єкт, який повинен містити інформацію про функціональність коду і інтерфейси доступу. Готові об'єкти можуть викликатись із інших програм (або об'єктів CORBA), розташованих в мережі.

Програмна платформа Java – ряд програмних продуктів і специфікацій компанії Sun Microsystems (перший офіційний випуск 23 травня 1995), які спільно надають систему для розробки прикладного ПО і вбудовування її в будь яку крос-платформну ПО. На відміну від CORBA, що використовує високо рівневі стандарти взаємодії, в Java боротьба с гетерогенністю здійснюється шляхом використання єдиного віртуального середовища.

Однорангові мережі (peer-to-peer, P2P) почали з'являтися в 1999-2000 рр. В таких мережах комп'ютери обмінюються ресурсами безпосередньо один з одним, без використання центрального сервера. Використання такої технології *спрощує масштабованість* мережі і *підвищує її відмовостійкість,* оскільки збій

будь якого обчислювального вузла не приводить до зупинки функціонування мережі цілком. До недоліків P2P можна віднести:

1. *збільшення вимог до виробництва* кожного комп'ютера, включеного в таку мережу;
2. *низьку степінь захищеності машин*, що беруть участь в P2P мережі, в силу надання відкритого доступу до своїх ресурсів;
3. необхідність подолання можливої *гетерогенності апаратного й програмного забезпечення* її потенціальних учасників;
4. *трудомісткість й ресурсомісткість пошуку доступних ресурсів* без використання централізованої точки управління. В якості прикладів P2P можна назвати такі проекти як Gnutella, BitTorrent, Napster, Skype.

На початку 2000-х років бізнес-співтовариств зайнялось розробкою специфікацій, покликаних вирішувати проблеми ранніх стандартів розподілених об'єктних технологій за допомогою **веб-сервісів** і **сервіс-орієнтовної архітектури (СОА, service-Oriented Architecture - SOA).** Стандарти веб-сервісів були розроблені по ініціативі організацій, які займаються наданням віддаленого доступу до певних обчислювальних ресурсів, і закріплені консорціумом W3C. До основних стандартів розробки і функціонування веб-сервісів можна віднести протоколи:

- SOAP – заснований на XML протокол взаємодії веб-сервісів;
- WSDL (Web Services Description Language – мова описання веб-сервісів) – це методологія описання ресурсів, наданих веб-сервісом;
- UDDI (Universal Description Discovery and Integration – універсальний метод пошуку і інтеграції) – метод описання, пошуку, взаємодії і використання веб-сервісів.

На сьогоднішній день, сервіс-орієнтований підхід є стандартом «де-факто» при розробці РОС.

З метою побудови широкомасштабних обчислювальних мереж, пристосованих для функціонування в динамічно-змінному навколишньому середовищі з середини 1990-х почали з'являтись ***агентні системи. Програмний агент*** – це автономний процес, здатний реагувати на середовище виконання і викликати зміни в середовище виконання, можливо, в кооперації з користувачем або іншими агентами.

Основними принципами роботи агентних мереж є:

- *автономність* – агенти функціонують автономно, без можливості стороннього втручання в їх внутрішній стан;
- *соціальна поведінка* – агенти взаємодіють один з одним за допомогою певної мови (Agent Communication Languages, ACLs);
- *активність* – агенти взаємодіють із навколишнім середовищем, отримуючи певні сигнали і відповідаючи на них;
- *про-активність* – агенти діюсь цілеспрямовано.

Одним із найбільш відомих прикладів є архітектура взаємодії FIRA (Foundation for Intelligent Physical Agents –базис інтелектуальних фізичних агентів), які стандартизує методи взаємодії агентів і агентних систем.

В 2007-2008 рр. почав набирати вагу термін **хмарні обчислення**. Задачею таких систем є забезпечення повсюдного та зручного мережевого доступу за потребою до спільного пулу обчислювальних ресурсів, що конфігуруються, які можуть бути оперативно представлені і звільнені із мінімальними експлуатаційними затратами і/чи зверненнями до провайдера. На сьогоднішній день вже можна говорити про те, що хмарні обчислення міцно увійшли у повсякденне життя кожного користувача Інтернету (хоча більшість про це і не підозрюють). Однак до сих пір немає єдиної думки про те, що таке «Хмарні обчислення» і таким чином вони співвідносяться з парадигмою грід-обчислень.

Обов'язкові характеристики хмарних обчислень:

Самообслуговування за вимогою – споживач самостійно визначає та змінює обчислювальні потреби (серверний час, швидкість доступу і ін.), без взаємодії із представником постачальника послуг;

Універсальний доступ до мережі – послуги доступні споживачам через мережу передачі даних незалежно від використовуваного термінального пристрою;

Об'єднання ресурсів – постачальник послуг об'єднує ресурси для обслуговування великого числа споживачів в єдиний пул для динамічного перерозподілу потужностей між споживачем в умовах постійного змінення попиту на потужність;

Еластичність – послуги можуть бути надані, розширені, звужені в будь який момент часу, як правило, в автоматичному режимі;

Облік споживання – постачальник послуг автоматично обчислює використані ресурси на визначеному рівні абстракції (наприклад, об'єм збережених даних, пропускна здатність), і на основі цих даних оцінює об'єм наданих споживачам послуг.

Тема 2. Архітектура інформаційних систем

Інформаційна система – організаційно впорядкована сукупність документів (масивів документів) і інформаційних технологій, в тому числі з використанням засобів техніки для обчислення і зв'язку, які реалізують інформаційні процеси. Інформаційні системи призначені для зберігання, обробки, пошуку, розповсюдження, передачі і надання інформації.

Архітектура інформаційної системи – концепція, яка визначає модель, структуру, функції які виконуються і взаємозв'язок компонентів інформаційної системи.

Звичайними словами:

Інформаційна система – це сукупність програмного забезпечення, яке вирішує певну прикладну задачу.

Архітектура інформаційної системи – абстрактне поняття, яке визначає з яких складових частин (елементів, компонентів) складається додаток і як ці частини між собою взаємодіють.

Під **складовими частинами (елементами, компонентами)** додатка зазвичай трактуються як програми або програмні модулі, які виконують окремі, відносно ізольовані задачі.

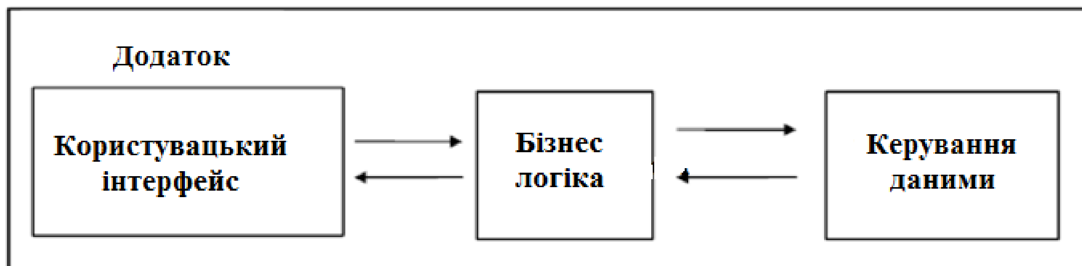


Рис. 4 – Компоненти інформаційної системи

Компоненти інформаційної системи за виконуваними функціями можна поділити на три шари:

- **Шар представлення (користувацький інтерфейс)** – все, що пов’язано з взаємодією з користувачем: натискання кнопок, рух комп'ютерної миші, малюнок зображення, виведення результатів пошуку і т.п.
- **Бізнес логіка –** правила, алгоритми реакції додатка на дію користувача або на внутрішні події, правила обробки даних.
- **Шар доступу до даних –** зберігання, вибір, модифікація і видалення даних, пов’язаних з задачею, яка виконується додатком.

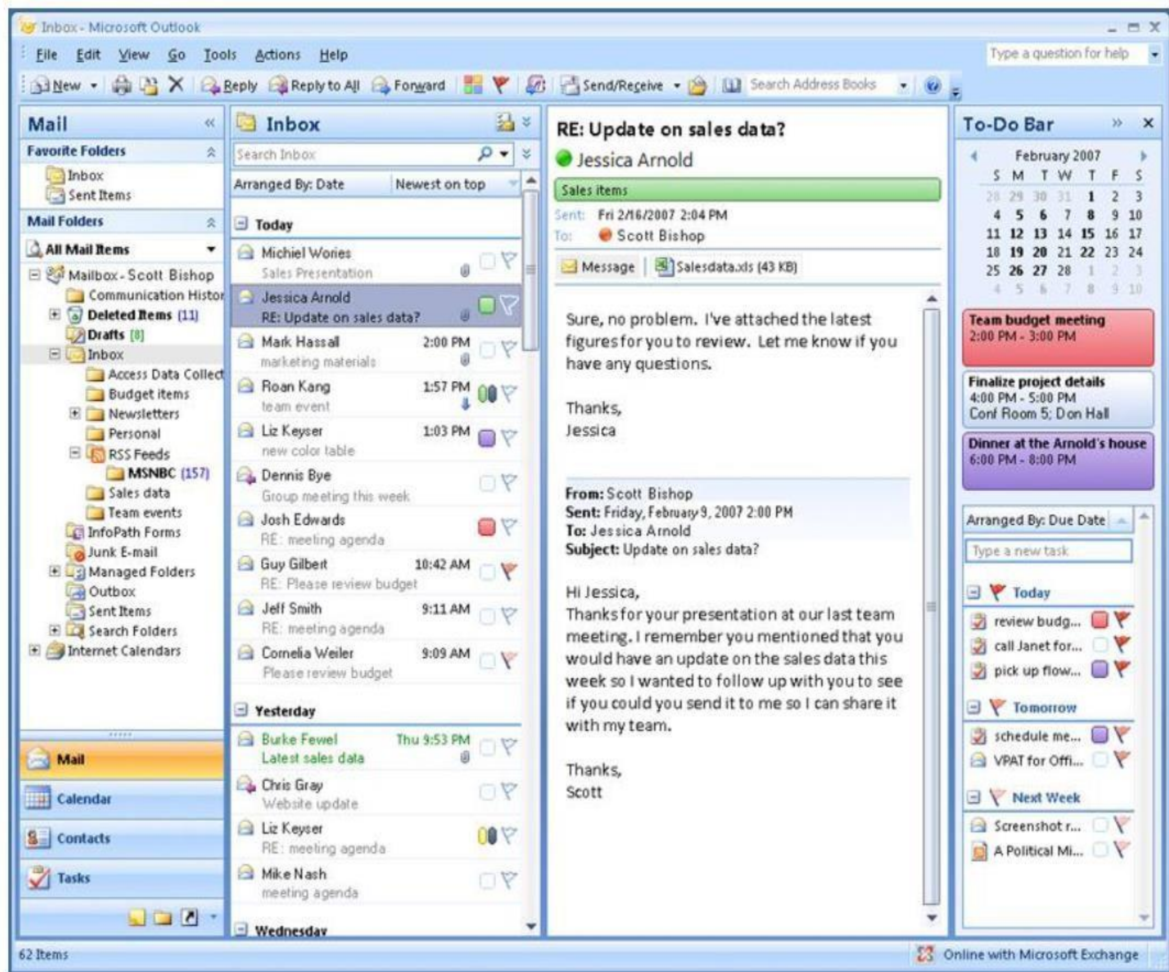


Рис. 5 – Вікно програми Microsoft Outlook

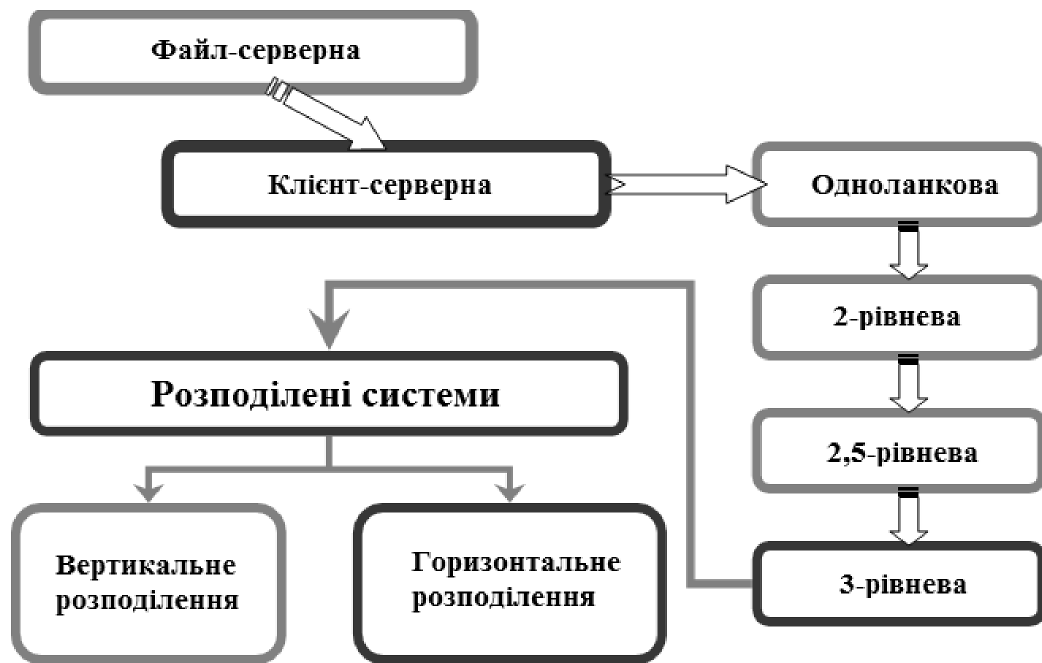
Приклад: ***Microsoft Outlook***

Користувацький інтерфейс – кнопки, меню, списки листів, дерево папок (folders) і т.д.

Бізнес логіка – правила обробки листів (Rules), отримання і відправка листів, сповіщення про надходження зустрічей (Meetings), подій (Events), запис дій користувача і т.д.

Керування даними – зберігання листів, контактів,даних про наради, події і т.д.

Класифікація архітектур ІС



Файл-серверна архітектура

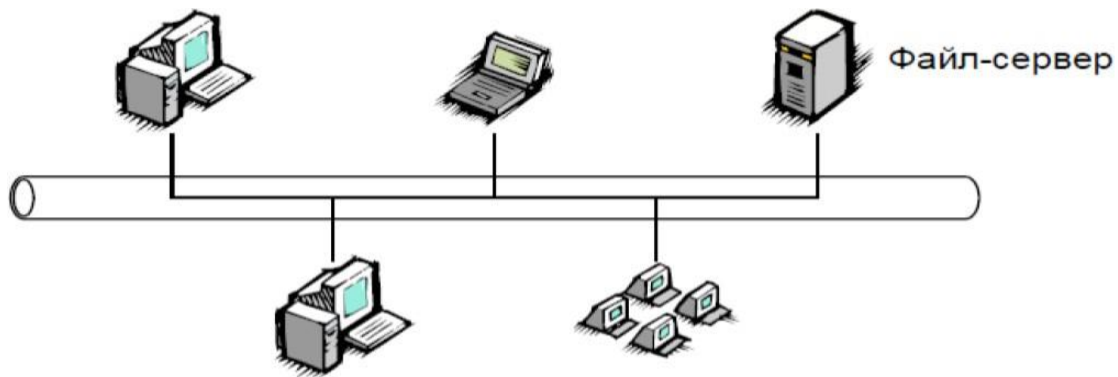


Рис. 6 – Файл-серверна архітектура

Усі загальнодоступні файли зберігаються на виділеному комп'ютері – **файл-сервері.**

Файл-серверні додатки – додатки, які використовують мережевий ресурс для зберігання програми і даних.

Функції сервера: зберігання даних і коду програми.

Функції клієнта: обробка даних. Кількість клієнтів обмежене десятками.

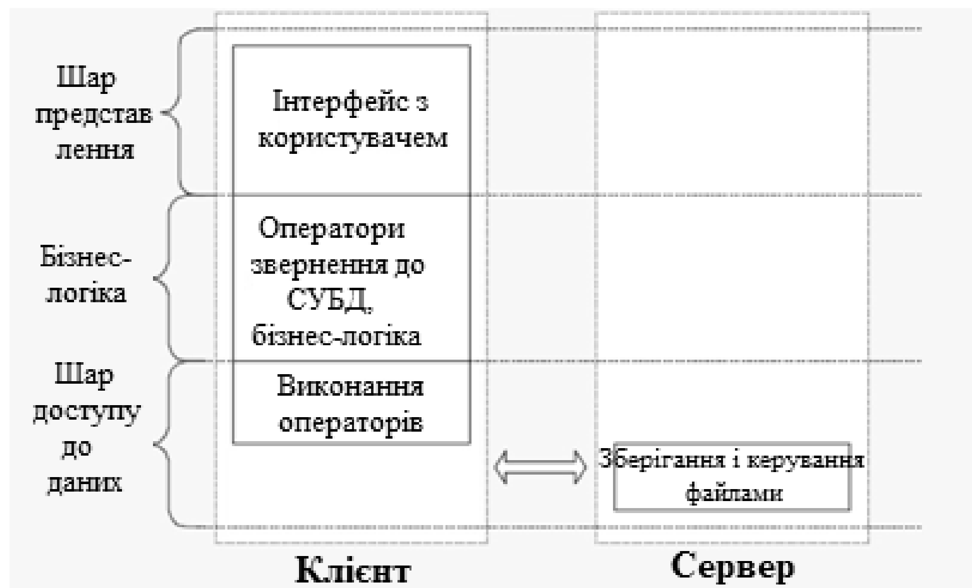


Рис. 8 – Модель файлового сервера

Плюси:

- розрахований на режим роботи з великою кількістю користувачів;
- зручне централізоване керування доступом;
- низька ціна розробки.

Мінуси:

- низька продуктивність;
- низька надійність;
- слабкі можливості розширення.

Недоліки архітектури з файловим сервером виходять, основним чином, з того , що дані зберігаються в одному місці, а обробляються в іншому. Їх потрібно передавати по мережі, що призводить до дуже високим навантаженням на мережу і різкому зниженню продуктивності додатка при збільшенні числа одночасно працюючих клієнтів. Другим важливим недоліком такої архітектури є децентралізоване вирішення проблем цілісності і узгодженості даних і одночасного доступу до даних.

Клієнт-серверна архітектура

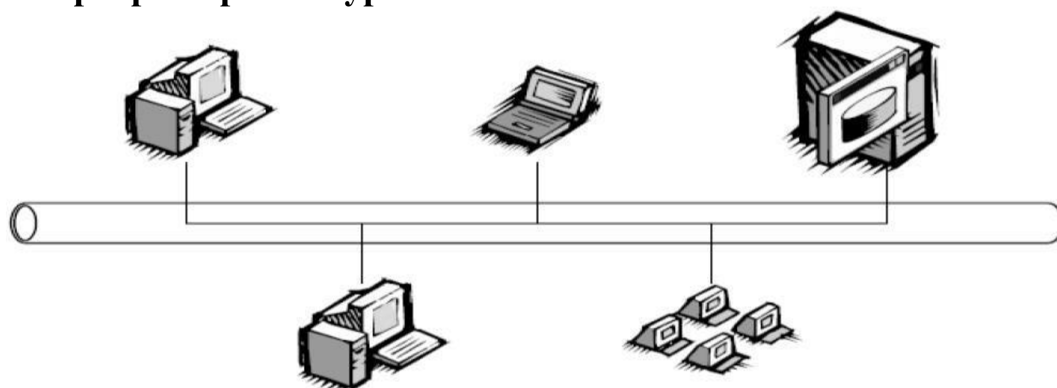


Рис. 9 – Клієнт-серверна архітектура

Ключові відмінності від архітектури ***файл-сервера*** **–** абстрагування від фізичної схеми даних і маніпуляції даних клієнтськими програмами на рівні

логічної схеми. Це дозволило створювати надійні багатокористувацькі ІС з централізованою базою даних (БД), незалежні від апаратної (а часто і програмної) частини сервера БД і підтримувати графічний інтерфейс користувача (ГІП) на клієнтських станціях, зв'язаних локальною мережею.

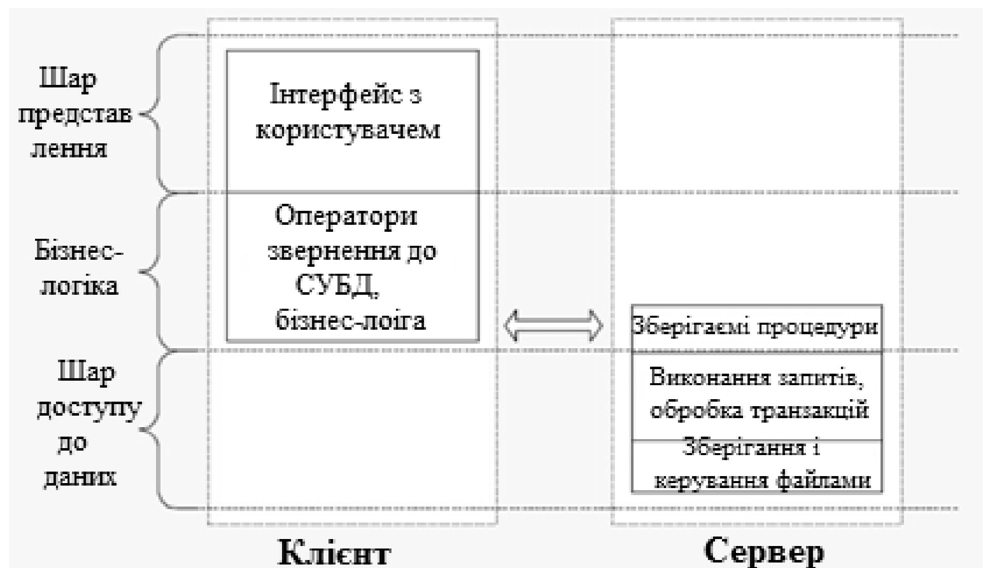


Рис. 10 – Модель СУБД

Основні особливості:

► клієнтська програма працює з даними через запити до серверного ПЗ;

► базові функції програми розділені між клієнтом і сервером.

Плюси:

► повна підтримка багатокористувацької роботи;

► гарантія цілісності даних.

Мінуси:

► Бізнес логіка додатків залишилася в клієнтському ПЗ. => При будь-якій зміні алгоритмів, треба оновлювати призначене для користувача ПЗ на кожному клієнті.

► Високі вимоги до пропускної здатності комунікаційних каналів з сервером.

► Слабкий захист даних від злому, особливо від недобросовісних користувачів системи.

► Висока складність адміністрування і настройки робочих місць користувачів системи.

► Необхідність використовувати потужні ПК на клієнтських місцях.

► Висока складність розробки системи через необхідність виконувати бізнес-логіку і забезпечувати користувальницький інтерфейс в одній програмі.

Неважко помітити, що більшість недоліків *класичної* або 2-х *шарової (2-рівневої)* архітектури клієнт-сервер виникають від використання клієнтської станції в якості виконавця бізнес-логіки ІС. Тому очевидним кроком подальшої еволюції архітектур ІС з'явилася ідея *"тонкого клієнта":* алгоритми обробки даних розбивалися на частини пов'язані з виконанням бізнес-функцій і відображенням інформації в зручному для людини поданні, частина, пов'язана з

первинною перевіркою і відображенням інформації залишалася на клієнтській машині, а вся реальнафункціональність системи переносилася на серверну частину.

Перехідна архітектура (2,5-шаровий клієнт-сервер)

Основні особливості:

► Використання збережених процедур і обчислення даних на стороні сервера;

► використання систем управління базами даних (СУБД) з усіма їх перевагами;

► написання програм для серверної частини, в основному, на спеціалізованих вбудованих мовами СУБД, які не дозволяють написати всю бізнес-логіку додатка, внаслідок чого частина бізнес-логіки все одно реалізується на стороні клієнта;

► фізично ІС складається з двох компонентів.

Плюси:

► реалізація обчислень на серверній стороні і передача по мережі готових результатів обчислень => зниження вимог до швидкості передачі даних між клієнтської і серверної частинами;

► істотне поліпшення захисту інформації <= користувачам даються права на доступ до функцій системи, а не до її даними і т.д.

Мінуси:

► обмежена масштабованість;

► залежність від програмної платформи ;

► обмежене використання мережевих обчислювальних ресурсів;

► написання програм для серверної частини системи на слабо призначених для цього вбудованих в СУБД мовами опису процедур =>

1. низька швидкодія системи;
2. висока трудомісткість створення і модифікації ІС;
3. висока вартість апаратних засобів, необхідних для функціонування ІС.

Трирівнева клієнт-серверна архітектура

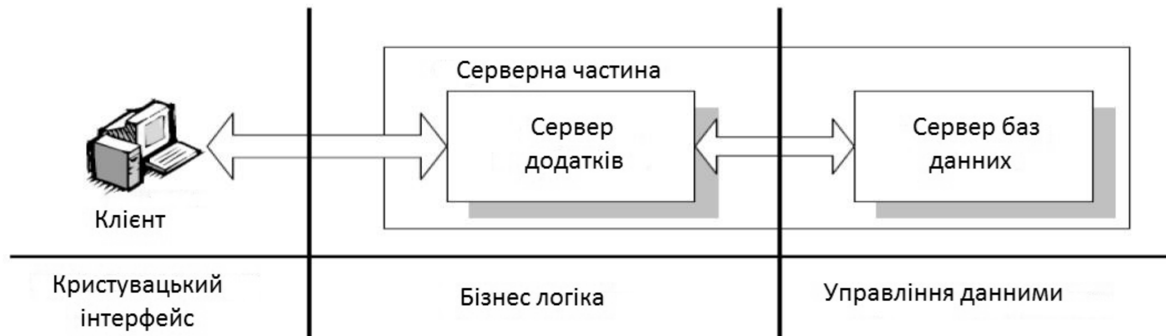


Рис. 11 - Трирівневий клієнт-сервер

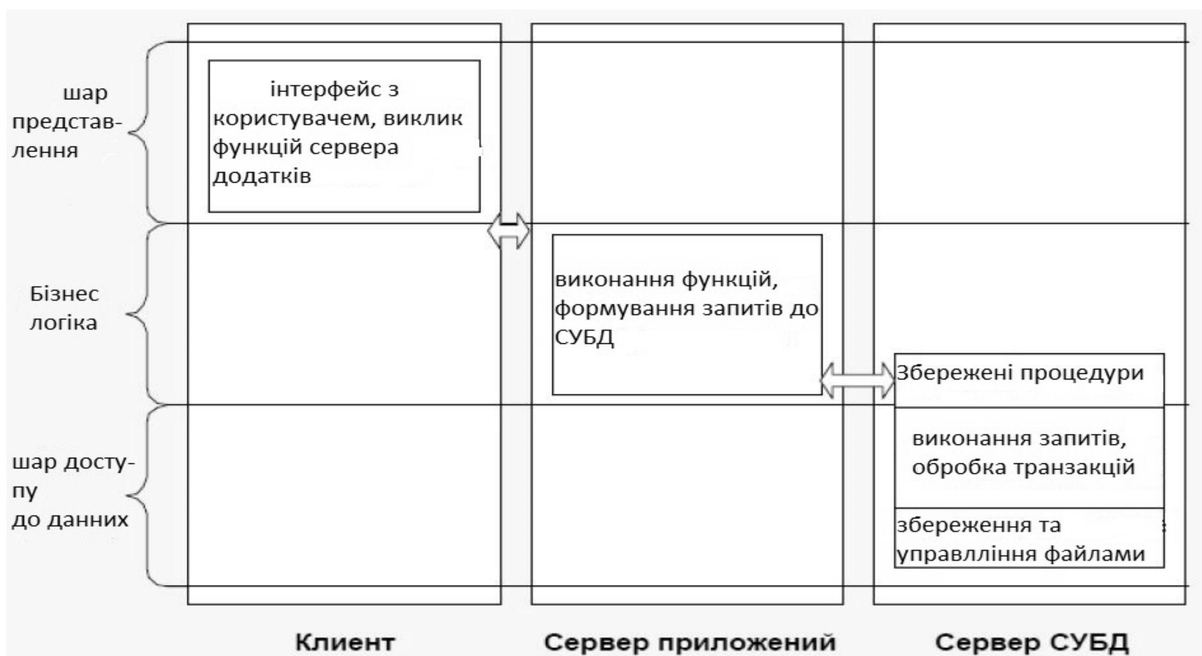


Рис. 12 - Модель сервера додатків

Основна відмінність від *архітектури* 2.5 - фізкабінет-ческий поділ прог-рам, що відповідають за зберігання даних (СКБД) і їх обробку (сервер додатки (СП), applica-tion сервер (AS)). Такий поділ програмних компонент дозволяє оптимізувати навантаження як на мережеве, так і на обчислювальний обладнання комплексу.

Плюси:

► Тонкий клієнт;

► Між клієнтською програмою і сервером додатка передається лише мінімально необхідний потік даних - аргументи викликаються функцій і повертаються від них значення. Це теоретичну межу ефективності використання ліній зв'язку (!!!).

► Сервер додатка ІВ може бути запущений в одному або декількох екземплярах на одному або декількох комп'ютерах, що дозволяє

використовувати обчислювальні потужності організації настільки ефективної безпечно як цього забажає адміністратор ІС.

► Дешевий трафік між сервером додатків і СУБД. Трафік між сервером додатків і СУБД може бути великим, проте це завжди трафік локальної мережі, а їх пропускна здатність досить великаі дешева. В крайньому випадку, завжди можна запустити СП і СУБД на одній машині, що автоматично зведе мережевий трафік до нуля.

► Зниження навантаження на сервер даних в порівнянні з 2.5-шарової схемою, а значить і підвищення швидкості роботи системи в цілому.

► Дешевше нарощувати функціональність і оновлювати ПО.

Мінуси:

► Вище витрати на адміністрування і обслуговування серверної частини.

Особливості:

1. ***Широкі можливості масштабування.***Одна і та ж система може працювати як на одному окремому комп'ютері, виконуючи на ньому програми СУБД, СП і клієнтської частини, так і в мережі, що складається з сотень і тисяч машин. Єдиним фактором, що перешкоджає нескінченної масштабованості, є лише вимога ведення єдиної бази даних.

2. ***Спрощення розширення функціональних можливостей.***

-На відміну від 2,5-шарової схеми немає необхідності міняти всю систему - досить встановити новий СП з необхідною функцією.

-У порівнянні з 2-шарової схемою зменшується число проблем, пов'язаних зі встановленням заново клієнтських частин програми на безлічі комп'ютерів, можливо досить віддалених.

Приклад: пошукова машина в Інтернеті

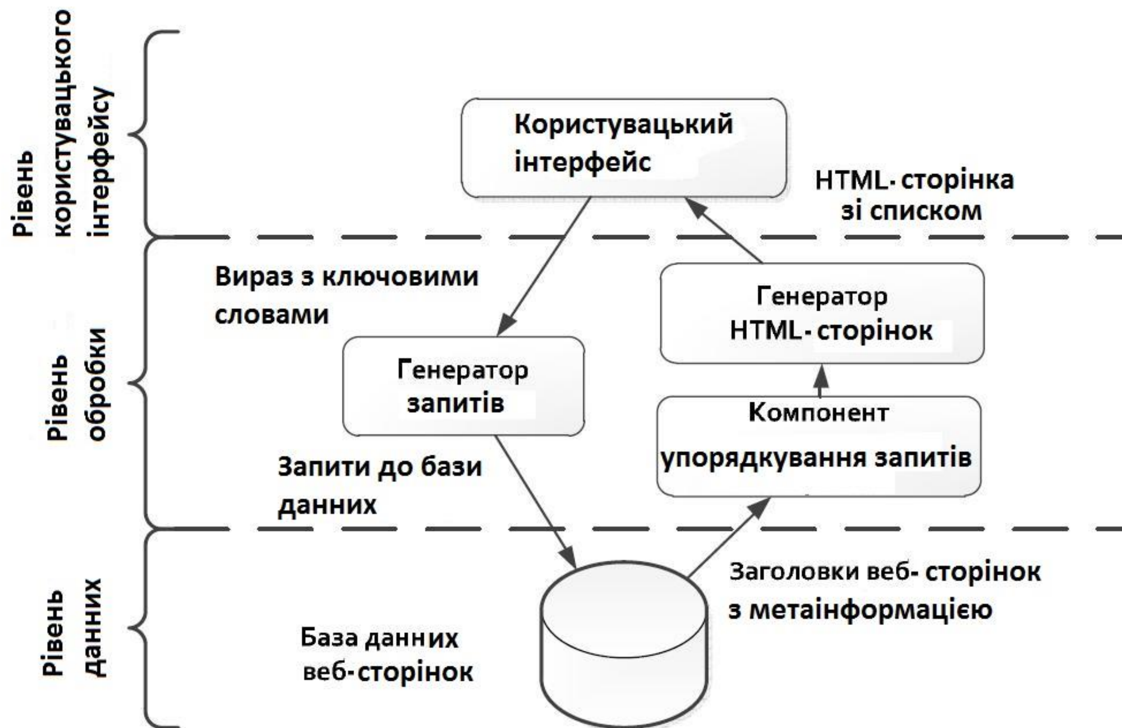


Рис. 13- узагальнена організація пошукової машини для Інтернету

Користувач вводить рядок, що складається з ключових слів, і отримує список заголовків веб-сторінок. Результат формується з гігантської бази просмот-корінних і проиндексируется-ванних веб-сторінок. Ядром пошукової машини є програма, трансформи-рующая введену пользо-вателем рядок в один або кілька запитів до бази даних. Потім вона поміщає результати запиту в список і перетворює цей список в набір HTML-сторінок. Частина, яка відповідає за вибірку інформації, зазвичай знаходиться на рівні обробки.

Багатоланкові архітектури клієнт-сервер

Багатоланкові архітектури клієнт-сервер є прямим продовженням поділу додатків на рівні призначеного для користувача інтерфейсу, компонентів обробки і даних. Різні ланки взаємодіють відповідно до логічної організацією додатки. У безлічі бізнес-додатків розподілена обробка еквівалентна організації многозвенной архітектури додатків клієнт-сервер. Такий тип розподілу називається ***вертикальним (ВР).***Характеристичної особливістю вертикального розподілу є те, що воно досягається *розміщенням логічно різних компонентів на різних машинах*.

В сучасних архітектурах розподіл на клієнти і сервери відбувається способом, відомим як ***горизонтальне розподіл (ГР).***

При такому типі розподілу *клієнт або сервер можуть містити фізично розділені частини логічно однорідного модуля*, причому робота з кожною з частин може відбуватися незалежно. Це робиться для вирівнювання завантаження.

Приклад: веб-сервер, реплицироваться на кілька машин локальної мережі

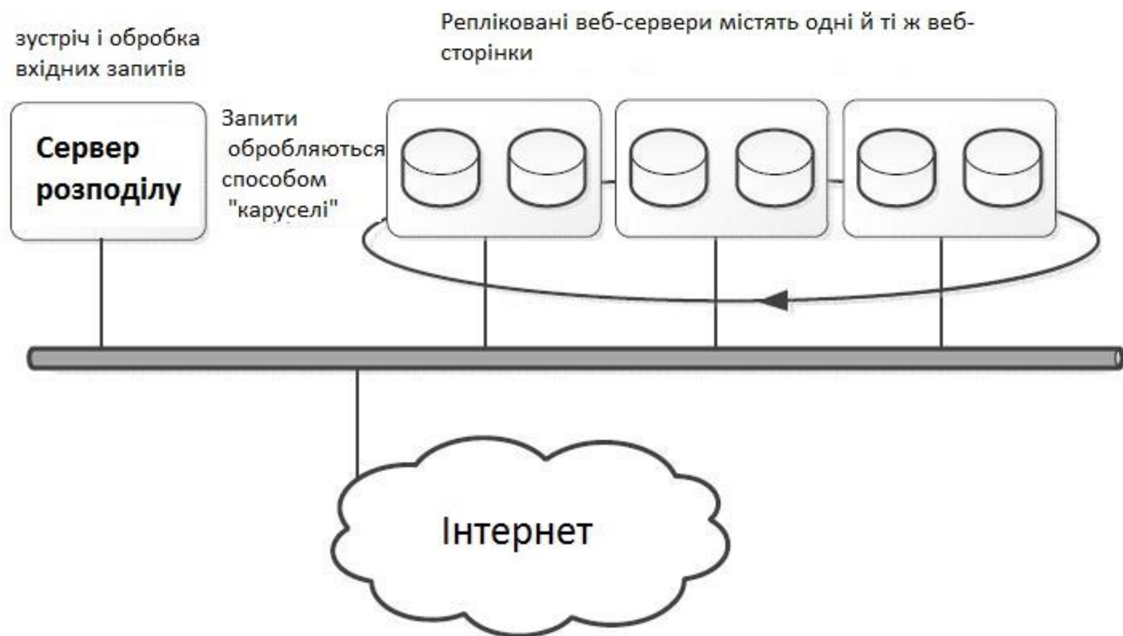


Рис. 14 - Приклад горизонтального розподілу веб-сервера

На кожному з серверів міститься один і той же набір веб-сторінок. Коли одна з веб-сторінок обнов-ляется, її копії відразу рас-силаются на всі сервери. Сервер, для якого буде переданий приходить запит, вибирається за правилом «каруселі». Ця форма ГР успішно використовується для вирівнювання навантаження на сервери популярних веб-сайтів.

Таким же чином, хоча і менш очевидно, можуть бути розподілені і клієнти. Для нескладного додатку, призначеного для колективної роботи, ми можемо не мати сервера взагалі. В цьому випадку ми зазвичай говоримо про вибрали *часовий розподіл*. Найяскравішим прикладом застосування ГР можуть служити програмні системи, створені на основі концепції хмарних обчислень.

Тема 3. Організація розподілених обчислень з використанням серверів додатків

Сервери додатків (СД) є однією з ключових складових IT-інфраструктури значної частини сучасних великих підприємств. Якщо компанія має потребу в інтеграції її внутрішньокорпоративних додатків з корпоративним Web-сайтом і Web-додатками, а також в надійному і швидкому доступі до даних і додатків з боку власних співробітників, партнерів і клієнтів, вона рано чи пізно стикається з необхідністю впровадження одного або декількох серверів додатків.

Сервер додатків (англ. *application server*) - це програмна платформа, призначена для ефективного виконання процедур (Програм, механічних операцій, скриптів), які підтримують побудова додатків. Сервер додатків діє як набір компонентів, доступних розробнику програмного забезпечення через API (інтерфейс прикладного програмування), який визначений самою платформою.

Переваги серверів додатків

- ***Цілісність даних і коду*** - виділяючи бізнес логіку на окремий сервер, або на невелику кількість серверів, можна гарантувати оновлення та покращення додатків для всіх користувачів.
- ***Централізована настройка і управління*** - зміни в налаштуваннях програми, таких як зміна сервера бази даних або системних налаштувань, можуть проводитися централізовано.
- ***Безпека*** - СП діє як центральна точка, використовуючи яку, постачальники сервісів можуть управляти доступом до даних і частинам самих додатків. При цьому відповідальність за аутентифікацію можна перемістити з потенційно небезпечного рівня клієнта на рівень СП, додатково приховуючи рівень бази даних.
- ***Підтримка транзакцій.*** *Транзакція* - одиниця активності, під час якої послідовність змін ресурсів (в одному або різних джерелах) виконується атомарно (як неподільна одиниця роботи).

Оскільки СП виконує масу потрібного генерування коду, розробники можуть сфокусуватися на бізнес-логіці, що зменшує тривалість і вартість розробки додатків.

Архітектура сучасних корпоративних додатків

Сервер додатків - це інфраструктурне програмне забезпечення, призначене для створення розподілених інформаційних систем з виділеними службами бізнес-логіки, реалізованими у вигляді компонентів, що виконуються під його керуванням.

Ці компоненти можуть являти собою:

- COM-об'єкти;
- CORBA-об'єкти;
- компоненти Enterprise JavaBeans.

У **функції сервера додатків** входять:

- управління оптимізацією системних ресурсів (пам'ять, потоки та ін.);
- забезпечення зв'язку додатки з зовнішніми ресурсами (включаючи бази даних, мережі та інші додатки);
- забезпечення якісної підтримки сервісів (доступність,
- надійність, безпека, управління, продуктивність, масштабованість);
- розгортання додатків (механізм їх поширення для установки на інших комп'ютерах, пристроях, серверах і в хмарі).

Сучасні сервери додатків дозволяють реалізувати надійні і стійкі до збоїв інформаційні системи за рахунок підтримки створення кластерів і наявності коштів відновлення після збоїв.

В даний час сервери додатків є основою багатьох корпоративних рішень з підвищеними вимогами до надійності, наприклад додатків, пов'язаних з електронною комерцією, що реалізують схеми

- «підприємство - споживач» (*business-to-consumer*, B2C),
- «підприємство - підприємство» (*business-to-business*, B2B),

- «підприємство - співробітник» (*business-to-employee*, B2E).

Як правило, при реалізації подібних рішень сервери додатків розташовуються між сервером баз даних і Web-сервером або між сервером баз даних (СБД) та клієнтськими додатками, при цьому іноді функціональність Web-сервера реалізується і в самому сервері додатків.

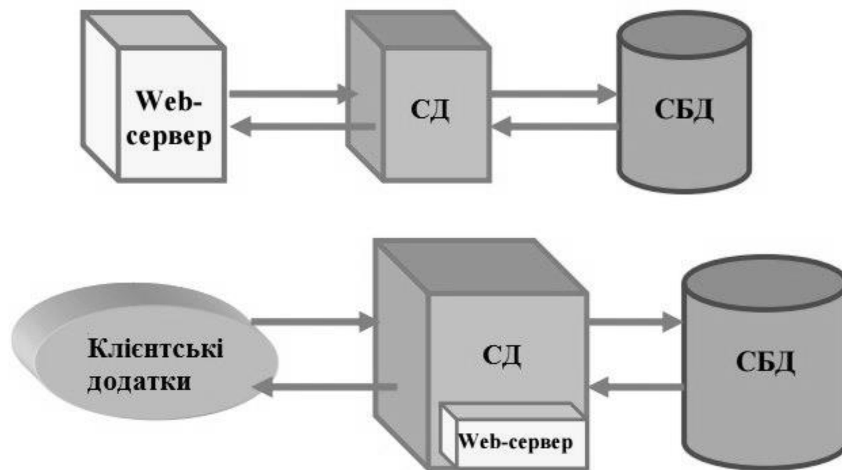
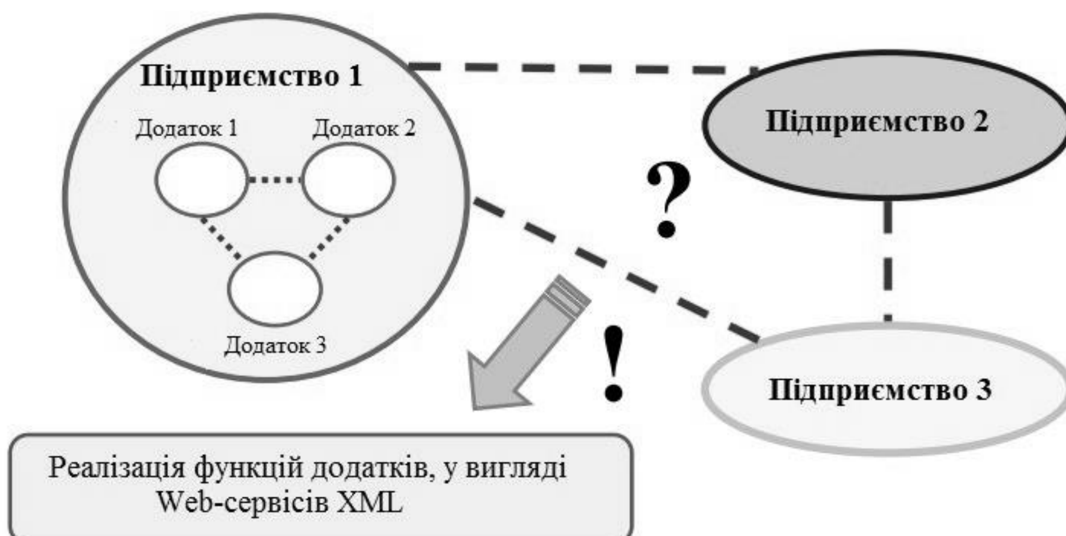


Рис. 15 - Варіанти розташування СД при реалізації корпоративних рішень

Говорячи про корпоративні рішеннях, не можна не відзначити *проблему інтеграції* як різних додатків всередині одного підприємства, так і додатків, що використовуються на різних підприємствах. Одним з загальноприйнятих способів її рішення є реалізація функцій додатків, до яких потрібен доступ ззовні, у вигляді Web-сервісів XML, і більшість виробників СД, СУБД і засобів розробки додатків реалізували підтримку Web-сервісів і пов'язаних з ними технологій.



Технології та стандарти

Сьогодні на корпоративному ринку домінують дві архітектури серверів додатків:

- *.NET* (представлена тільки у виконанні Microsoft);
- *Java EE* (раніше відома як J2EE), що надається безліччю компаній (мультивендорна).

Але при цьому серйозну конкуренцію лідерам складають нові програмні моделі, такі як Spring Framework, PHP, Ruby on Rails, Apex Code, Plain Old Java Object (POJO).

Сервери додатків можуть бути доступні користувачам:

- у вигляді продуктів, що встановлюються власне всередині компаній,
- як хмарних сервісів, що надаються провайдером.

Для визначення розстановки сил на ринку серверів додатків звернемося до даних дослідницької і консалтингової компанії Gartner, що спеціалізується на ринках інформаційних технологій (IT).

Для оцінки постачальників будь-якого сегмента ринку IT, Gartner використовує дві лінійні прогресивні експертні шкали:

- **повнота бачення** (англ. *completeness of vision*),
- **здатність реалізації** (англ. *ability to execute*).

При цьому, повнота бачення відкладається на осі абсцис, здатність реалізації - на осі ординат. Кожен постачальник, таким чином, виявляється в одному з **чотирьох квадрантів площини**, званих:

- лідери (англ. *leaders*) - постачальники з позитивними оцінками як за повнотою бачення, так і по можливості реалізації,
- претенденти (англ. *challengers*) - постачальники з позитивними оцінками тільки за можливості реалізації,
- провидці (англ. *visionaries*) - постачальники з позитивними оцінками тільки за повнотою бачення,
- нішеві гравці (англ. *niche players*) - постачальники з негативними оцінками за обома критеріями.

Gartner називає **магічним квадрантом** конкретний аналіз будь-якого сегмента ринку, з розподілом постачальників за вказаними чвертях.

На думку Gartner, СП можуть застосовуватися **в трьох основних сценаріях:**

- **"тимчасово-орієнтовані" проекти** - швидка розробка і розгортання додатків у відповідь на бізнес-вимоги, які -не можуть чекати. В цьому випадку час реалізації проекту є більш важливим, ніж забезпечення доступності, масштабованості і т. д .;
- **проекти "масового ринку"** - не дуже складні прикладні системи, створювані невеликими IT-компаніями. Тут важливі такі параметри, як низька вартість, простота розгортання, підтримки та управління, надійність. Функціональність, масштабованість і продуктивність не настільки істотні;
- **"систематично-орієнтовані" проекти** - реалізація критично важливих для бізнесу корпоративних систем, розрахованих на довгостроковий період експлуатації (не менше трьох років). Поряд з

різноманітної функціональністю тут необхідні надійність, безпеку, керованість, масштабованість, продуктивність.

Нижче в магічному квадраті розглядаються постачальники продуктів класу СП масштабу підприємства (Enterprise Application Server - EAS) - таких, які можуть застосовуватися в третьому сценарії. При цьому наголошується, що багато такі рішення можна успішно використовувати і в двох інших варіантах.

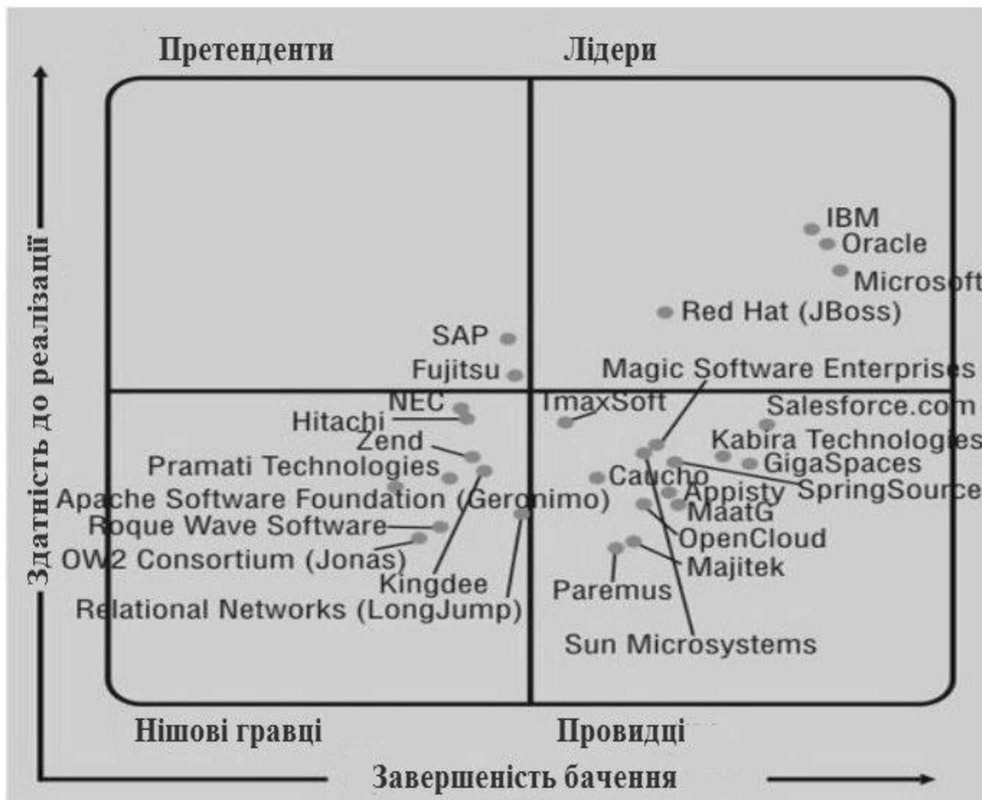


Рис. 16 - Магічний квадрант для СП масштабу підприємства

Як бачимо з квадранта:

По-перше: існує досить велика кількість гравців, **28 компаній**, деякі з яких пропонують рішення **Open Source** (вільне ПЗ, тобто матеріали, використані для його створення, доступні за вільної / відкритої ліцензії).

По-друге: є ряд платформ, спочатку орієнтованих на варіант **хмарної обчислювальної моделі** (зокрема, потрібно звернути увагу на Salesforce.com).

По-третє: є чітко виділена група лідерів: **IBM, Microsoft, Oracle, Red Hat (JBoss).**

По-четверте: видно гостра конкурентна ситуація на ринку, яка надає замовникам широкий спектр вибору найбільш підходящих їм рішень. Gartner вважає, що найближчим часом тут з'являться нові серйозні гравці, такі як **Google** і **Tibco**, чиї рішення (відповідно **App Engine** і **Silver**) знаходяться поки в режимі бета-тестування.

Група лідерів

IBM

Сімейство ПО **IBM WebSphere** включає представницький набір EAS-пропозицій (у т.ч. серію продуктів **WebSphere Application Server - WAS**), який покриває широкий діапазон вимог замовників:

- для -Тимчасові-орієнтірованних‖ проектів (*WebSphere sMash*),
- для масового ринку (*WAS Community Edition, WAS Express*),
- для масштабованих корпоративних рішень:
  - *WAS Network Deployment,*
  - *WebSphere Virtual Enterprise,*
  - *CloudBurst Appliance* і ін.

Деякі з цих засобів доступні в вигляді **хмарних сервісів IBM** і **Amazon Web Services EC2**.

WAS-рішення базуються в цілому на стандартах Java EE і SOA, забезпечуючи при цьому підтримку різних моделей програмування. Слід зазначити, що IBM має в своєму розпорядженні потужні набори засобів розробки (Rational) і управління IT (Tivoli).

У той же час слід зазначити, що присутність IBM на масовому ринку поки невелика. Випущений в середині 2008 року WebSphere sMash поки має відносно невелику инсталлированную базу і досить обмежену підтримку з боку третіх фірм. Продукти для хмар (WAS Hypervisor Edition, CloudBurst Appliance) з'явилися відносно недавно, і поки в цій сфері IBM помітно відстає від лідерів. В цілому Gartner відзначає, що стратегія IBM в області APaaS знаходиться в ранній стадії своєї реалізації.

Microsoft .NET Framework разом з **Internet Information Server** (обидва є інтегрованими компонентами **Windows Server**) представляють собою повний набір функціональності EAS. Продукти сімейства корпоративних серверів **Microsoft Server System** призначені, як і інші СД, для створення і розгортання інтегрованих корпоративних рішень.

При відносно невисокій вартості для цих серверів характерні:

- підтримка XML,
- підтримка стандартів Інтернет,
- підтримка кластерної архітектури,
- високий ступінь взаємної інтеграції.

За функціональністю сімейство серверів Microsoft Server System, в цілому, заповнює практично всі сучасні напрямки застосування СД. Всі сервери сімейства Microsoft Server System підтримують управління COM-, COM + - і .NET-компонентами і доступні для операційних систем сімейства Windows. Для інших платформ ці продукти не випускалися і не випускаються.

З продуктів, що входять в сімейство Microsoft Server System, до *серверів додатків в традиційному розумінні* можна віднести:

- сервер інтеграції додатків *Microsoft BizTalk Server*,
- сервер повідомлень і групової роботи *Microsoft Exchange Server*,
- сервер електронної комерції *Microsoft Commerce Server*,
- масштабований сервер додатків для мобільної телефонії *Microsoft Mobile Information Server*,
- корпоративний портал *Microsoft SharePoint Portal Server*,
- сервер для управління інформаційним наповненням Web-сайтів *Microsoft Content Manager Server*;
- сервер для управління великими корпоративними проектами *Microsoft Project Server*.

Хмарні пропозиції Microsoft реалізовані у вигляді бета-версії **Windows Azure Platform** і недавно анонсованої технології програмованої хмарної платформи (**xRM**).

Перевага Microsoft - величезна інстальована база Windows Server і велике співтовариство розробників ПЗ, що робить її продукти стандартом де-факто на масовому ринку. При цьому компанія постійно нарощує свою присутність в сфері великих корпоративних проектів. У компанії є досить чітка стратегія реалізації хмарної моделі обчислень. Але з достоїнств Microsoft випливають і її **слабкі сторони** - підтримка однієї ОС і використання продуктів від одного постачальника.

Через традиційне фокусування на масовому ринку компанія постійно запізнюється з реалізацією важливих технологічних ініціатив (наприклад, XTP (Extreme Transaction Processing - форма обробки транзакцій в інформаційних технологіях) і SOA). При цьому у Microsoft з'явився ряд дуже серйозних суперників (Google, Salesforce, VMware), також орієнтованих на масовий ринок, але в конкурентній боротьбі використовують інші ділові та технологічні моделі.

Oracle

Основу EAS-пропозицій Oracle становить JEE-сімейство **Oracle WebLogic Server (WLS),** отримане в результаті придбання в 2008-му компанії BEA Systems, і власна розробка **Oracle Application Server** (Вона ще підтримується, але в стратегічному плані не розвивається). Крім того, до групи EAS-продуктів входять:

- засіб розробки *Oracle JDeveloper*,
- інструмент *Oracle TopLink*;
- засіб для моніторингу, адміністрування і управління *Oracle Enterprise Manager*.

В результаті придбання **Sun Microsystems** компанія поповнила свій EAS-арсенал ще й цілим портфелем EAS-технологій (зокрема вільним середовищем розробки **NetBeans**), завдяки чому посіла провідну роль в Java-співтоваристві.

WLS-сімейство в цілому фокусується на підтримці потужних критично важливих бізнес-додатків, але при цьому відповідає вимогам широкого спектра клієнтів, в тому числі з малого і середнього бізнесу. Ці рішення мають велике поширення на ринку і використовують самі сучасні технологічні концепції.

На поточний момент в Oracle слабо представлені EAS-рішення, орієнтовані на масовий ринок, хоча придбання Sun **GlassFish** частково заповнило цей пролом в спектрі ПО Oracle.

Компанію Oracle часто критикують за занадто *активну стратегію придбань*, яка часом ставить у глухий кут замовників Oracle, які перестають орієнтуватися в перспективах розвитку того чи іншого напрямку продуктів компанії.

Red Hat JBoss EAS - це JEE5-сумісний **JBoss Application Server** (с квітня 2013 **WildFly**). Він доступний для безкоштовного завантаження (без технічної підтримки) або в складі пакету для підприємств у вигляді **JBoss Enterprise Application Platform** з оплатою підтримки за передплатою.

Крім того, у Red Hat є набір **JBoss Enterprise SOA Platform**. Сюди входять сервер додатків і цілий ряд технологій підтримки SOA, включаючи рішення **JBoss ESB**, призначене для інтеграції різних систем, в тому числі несумісних.

Є також EAS-пропозиція **JBoss Communications Platform**, орієнтоване на використання в телекомунікаційній галузі.

Для Web-проектів пропонується **JBoss Enterprise Web Server**, що включає сервер додатків Apache Tomcat. У сімейство ПО JBoss входить і цілий ряд інших продуктів і інструментів, в тому числі кошти розробки та управління IT-інфраструктурою.

Все це ПО, *безкоштовне* або *отримується за передплатою*, поширюється за ліцензіями LGPL 2.x або Apache Software і доступно у вигляді вихідних кодів або об'єктних модулів.

JBoss EAS має відмінну технічну репутацію на ринку, Red Hat є явним лідером серед постачальників відкритих EAS-рішень, її продукти мають величезну инсталлированную базу і безліч партнерів і користувачів. Фактично це єдине **Open Source**-сімейство на IT-ринку, яке на рівних конкурує з пропозиціями провідних пропрієтарних вендорів. Але бізнес-стратегія Red Hat, націлена на підвищення прибутковості підрозділу JBoss, іноді має результатом уповільнення впровадження інженерних інновацій. Компанія явно відстає від конкурентів в освоєнні передових технологій, таких як XTP, *покрокове управління і хмарні обчислення.*

Архітектури серверів додатків Java Platform, Enterprise Edition

Java Platform, Enterprise Edition, скорочено **Java EE** (до версії 5.0 - **Java 2 Enterprise Edition** або **J2EE**) - набір специфікацій і відповідної документації для мови Java, яка описує архітектуру серверної платформи для задач середніх і великих підприємств. Специфікації деталізовані настільки, щоб забезпечити переносимість програм з однієї реалізації платформи на іншу.

Основна мета специфікацій - забезпечити масштабованість додатків і цілісність даних під час роботи системи. JEE багато в чому орієнтована на використання еė через веб як в інтернеті, так і в локальних мережах. Вся специфікація создаётся і затверджується через JCP (Java Community Process) в рамках ініціативи Sun Microsystems Inc.

Популярності JEE також сприяє те, що Sun пропонує безкоштовний комплект розробки, **SDK (Software Development Kit)**, дозволяє підприємствам розробляти свої системи, не витрачаючи великих коштів. У цей комплект входить сервер додатків GlassFish з ліцензією для розробки.

Актуальна версія Java EE (JEE) має номер 7.0. При переході на версію 5.0 до набору специфікацій додалося кілька нових технологій і змінилася назва специфікації з J2EE (**Java 2**

Platform, Enterprise Edition), на **Java Platform, Enterprise Edition**, сокращѐнно **Java EE**.

Версія	Повне ім'я	Дата публікації
1.0	Java 2 Platform Enterprise Edition, v 1.0	грудень 1999
1.2	Java 2 Platform Enterprise Edition, v 1.2	2000
1.2.1	Java 2 Platform Enterprise Edition, v 1.2.1	23 травня 2000
1.3	Java 2 Platform Enterprise Edition, v 1.3	24 вересня 2001
1.4	Java 2 Platform Enterprise Edition, v 1.4	24 листопада 2003
5.0	Java Platform, Enterprise Edition, v 5	11 травня 2006
6.0	Java Platform, Enterprise Edition, v 6	6 грудня 2009
7.0	Java Platform, Enterprise Edition, v 7	12 червня 2013

Java EE включає в себе **стандарти наступних технологій**:

EJB - *Enterprise JavaBeans* - специфікація технології написання і підтримки серверних компонентів, що містять бізнес-логіку.

JPA - *Java Persistence API* - надає можливість зберігати в зручному вигляді Java-об'єкти в базі даних.

Сервлет - клас, який розширює функціональні можливості сервера. Сервлет взаємодіє з клієнтами за допомогою принципу запит-відповідь.

JSP - *JavaServer Pages* технологія, що дозволяє веб-розробникам легко створювати вміст, який має як статичні, так і динамічні компоненти. По суті, сторінка JSP є текстовим документом, який містить текст двох типів: статичні вихідні дані, які можуть бути оформлені в одному з текстових форматів HTML, SVG, WML, або XML, і JSP елементи, які конструюють динамічний вміст.

JSTL - *JavaServer Pages Standard Tag Library* - «стандартна бібліотека тегів JSP ». Вона розширює специфікацію JSP, додаючи бібліотеку JSP тегів для загальних потреб, таких як розбір XML даних, умовна обробка, створення циклів і підтримка інтернаціоналізації.

Інтернаціоналізація – технологічні прийоми розробки, спрощують адаптацію продукту (такого як програмне або апаратне забезпечення) до мовних і культурних особливостей регіону (регіонів), відмінного від того, в якому розроблявся продукт.

JSF - *JavaServer Faces* - компонентний серверний фреймворк для розробки веб-додатків на технології Java, призначений для полегшення розробки користувальницьких інтерфейсів (ІІ) для JEE-додатків. Підхід JSF ґрунтується на використанні компонентів.

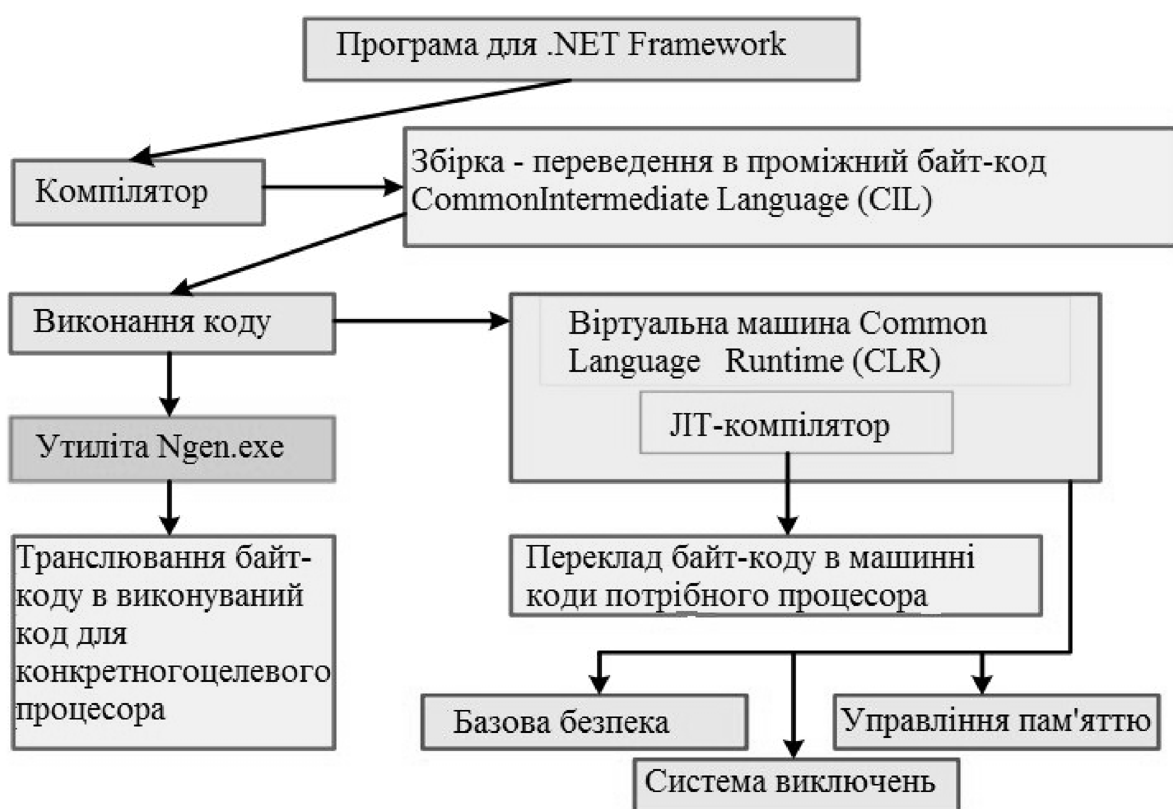
Стан компонентів ПІ зберігається, коли користувач запитує нову сторінку і потім відновлюється, якщо запит повторюється.
JAX-WS - *Java API for XML Web Services* - це прикладний програмний інтерфейс мови Java для створення веб-служб.
JNDI - *Java Naming and Directory Interface* - це Java API, організований в вигляді служби каталогів, який дозволяє Java-клієнтам відкривати і переглядати дані і об'єкти за їхніми іменами.
JMS - *Java Message Service* - стандарт проміжного програмного забезпечення для розсилки повідомлень, що дозволяє додаткам, виконаним на платформі Java EE, створювати, надсилати, отримувати та читати повідомлення.
JTA - *Java Transaction API* - Java API для транзакцій. Визначає взаємодія між менеджером транзакцій і іншими учасниками розподіленої транзакції системи.
JAAS - Java Authentication and Authorization Service - реалізація в мові програмування Java стандарту системи інформаційної безпеки PAM. **Pluggable Authentication Modules** (PAM, підключаючі модулі аутентифікації) - це набір поділюваних бібліотек, які дозволяють інтегрувати різні низькорівневі методи аутентифікації у вигляді єдиного високорівневого API. Це дозволяє надати єдині механізми для управління, вбудовування прикладних програм в процес аутентифікації.
JavaMail - це Java API, призначений для отримання і відправки електронної пошти з використанням протоколів SMTP, POP3 та IMAP.
JACC - *Java Authorization Contract for Containers* - це стандарт, який визначає контракти безпеки між модулями СП і політики авторизації. Ці контракти визначають способи установки, настройки і використання провайдерів авторизації в рішеннях доступу.
JCA - *J2EE Connector Architecture* - рішення на базі технології Java для з'єднання серверів додатків і корпоративних інформаційних систем в рамках рішень інтеграції корпоративних додатків.
JAF - *JavaBeans Activation Framework* - стандартне розширення для платформи Java, що дозволяє скористатися стандартними послугами: визначити тип довільного фрагмента даних; инкапсулировать доступ до нього; виявити доступні для нього операції; і встановити відповідний боб (bean) для виконання операції (й).
StAX - *Streaming API for XML* - інтерфейс прикладного програмування для читання і запису XML-файлів.
CDI - *Context and Dependency Injection* - Впровадження контекстів і залежностей - допомагають зв'язати рівень веб-вузлів і рівень транзакцій платформи JEE. CDI - це набір послуг, які, дозволяють розробникам використовувати JavaBeans з JSF в веб-додатках.

Microsoft .NET Framework

.NET Framework - програмна платформа, випущена компанією Microsoft в 2002 році. Основою платформи є загальномовного середовища виконання **Common Language Runtime (CLR),** яка підходить для різних мов програмування.

Функціональні можливості CLR доступні в будь-якими мовами програмування, що використовують цю середу.
Вважається, що платформа .NET Framework стала відповіддю компанії Microsoft на набрала на той час велику популярність платформу Java компанії Sun Microsystems (нині належить Oracle).
Хоча .NET є патентованою технологією корпорації Microsoft і офіційно розрахована на роботу під операційними системами сімейства Microsoft Windows, існують незалежні проекти (перш за все це **Mono** і **Portable.NET**), що дозволяють запускати програми .NET на деяких інших операційних системах.

Архітектура .NET



Програма для .NET Framework, написана на будь-якому підтримуваному мові програмування, спочатку перекладається компілятором в єдиний для .NET проміжний байт-код **Common Intermediate Language (CIL)** (раніше називався **Microsoft Intermediate Language, MSIL)**. У термінах .NET виходить збірка, англ. assembly.

Потім код або виповнюється віртуальною машиною **Common Language Runtime (CLR)**, або транслюється утилітою NGen.exe в виконуваний код для конкретного цільового процесора. Використання віртуальної машини переважно, оскільки позбавляє розробників від необхідності піклуватися про особливості апаратної частини. У разі використання віртуальної машини CLR, вбудований в неѐ **JIT**-компілятор «на льоту» (just in time) перетворює проміжний байт-код в машинні коди потрібного процесора. Сучасна технологія динамічної

компіляції дозволяє досягти високого рівня швидкодії. Віртуальна машина CLR також сама піклується про базову безпеки, управлінні пам'яттю і системі винятків, позбавляючи розробника від частини роботи. Microsoft почала розробляти .NET Framework в кінці 1990-х під ім'ям «**Next Generation Windows Services**» (NGWS). У 2000 році була випущена перша бета-версія .NET 1.0.

Версія	**Дата виходу**	**Visual Studio**	**За умовчуванням у Windows**
1.0	1.05.2002	Visual Studio.NET	
1.1	1.04.2003	Visual Studio.NET 2003	Windows Server 2003
2.0	11.07.2005	Visual Studio 2005	Windows Vista, Windows 7, Windows Server 2008 R2
3.0	6.11.2006	Visual Studio 2005 + розширення	Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2
3.5	9.11.2007	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	12.04.2010	Visual Studio 2010	Windows 8, Windows Server 2012
4.5	15.08.2012	Visual Studio 2012	Windows 8, Windows Server 2012
4.5.1	17.10.2013	Visual Studio 2013	Windows 8.1, Windows Server 2012 R2

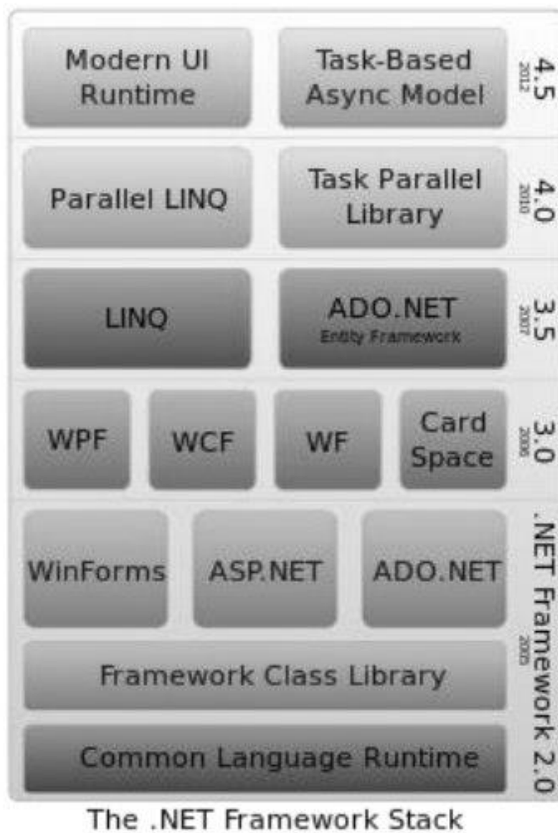
Архітектура .NET Framework описана і опублікована в специфікації **Common Language Infrastructure (CLI),** розробленої Microsoft і утверждѐнной ISO (Міжнародна організація по стандартизації) і Ecma (Раніше ECMA - European Computer Manufacturers Association, організація, що займається питаннями стандартизації інформаційних і комунікаційних технологій).

У CLI описані:

- типи даних .NET,
- формат метаданих про структуру програми,
- система виконання байт-коду та ін.

Об'єктні класи .NET, доступні для всіх підтримуваних мов програмування, містяться в бібліотеці **Framework Class Library (FCL**). Ядро FCL називається **Base Class Library (BCL).**

Windows Forms - API, відповідальний за графічний інтерфейс



The .NET Framework Stack

користувача. **ASP.NET (Active Server Pages)** - технологія створення веб додатків і веб-сервісів. **ADO.NET** - технологія, яка надає .NET-додаткам доступ до даних. **Windows Presentation Foundation (WPF)** - система для побудови клієнтських додатків Windows з візуально привабливими можливостями взаємодії з користувачем. Розроблятися додатки можуть автономно або запускатися в браузері.

Windows Communication Foundation (WCF) - програмний фреймворк, використовуваний для обміну даними між додатками. До випуску в складі .NET Framework 3.0, був відомий під кодовим ім'ям **Indigo**. WCF дозволяє створювати безпечні і надѐжние транзакційні системи через спрощену уніфіковану програмну модель міжплатформної взаємодії. У WCF закладені принципи інтероперабельності, які дозволяють організовувати роботу з іншими платформами.

Windows Workflow Foundation (WF) - технологія для визначення, виконання і управління робочими процесами (англ. *workflow*). (**Робочий процес (потік)** - 1. графічне представлення процесу виконання завдання і пов'язаних з ним підпроцесів; 2. спосіб надходження інформації до різних об'єктів, які беруть участь в процесі). WF орієнтована на візуальне програмування і використовує декларативну модель програмування.

Windows CardSpace (WCS) - це спосіб ідентифікації користувачів при переміщенні між ресурсами Інтернету без необхідності повторного введення імен і паролів. 15 лютого 2011 корпорація Майкрософт оголосила про скасування Windows CardSpace 2.0 і про роботу над заміщує ПО U-Prove.

Language Integrated Query (LINQ) - проект по додаванню синтаксису мови запитів, що нагадує SQL (*structured query language* - «мова структурованих запитів »), в мови програмування платформи .NET.

ADO.NET Entity Framework (EF) - об'єктно-орієнтована технологія доступу до даних. Надає можливість взаємодії з об'єктами як за допомогою LINQ (LINQ to Entities), так і з використанням Entity SQL.

Паралельний LINQ (PLINQ) - паралельна реалізація LINQ to Objects. PLINQ, що реалізує повний набір стандартних операторів запиту LINQ в вигляді розширення для простору імен T: System.Linq і має додаткові оператори для паралельних операцій. PLINQ може значно збільшити швидкість запитів LINQ

to Objects, ефективніше використовуючи всі доступні ядра на головному комп'ютері.

Task Parallel Library (TPL) - бібліотека паралельних задач, що представляє собою поєднання загальних типів і API, які дозволяють реалізовувати паралелізм і узгодженість операцій. TPL є високорівневим каркасом паралельного програмування для .NET коду, що дозволяє максимально використовувати продуктивність багатоядерних процесорів. TPL дозволяє реалізувати логічний паралелізм (вказати те, що потенційно може працювати паралельно) замість жорсткого поділу на потоки. Реальне розпаралелювання бібліотека виробляє під час виконання в залежності від доступних апаратних засобів.

Modern UI Runtime - дизайнерський стиль, орієнтований на друкарське оформлення інтерфейсу користувача.

Task-based Asynchronous Pattern (TAP) - заснована на завданнях асинхронна модель - схема програмування, спрямована на створення асинхронних потоків у виконанні програми і управління ними.

Однією з основних ідей Microsoft .NET є **сумісність програмних частин, написаних різними мовами**. Наприклад, служба, написана на C ++ для Microsoft .NET, може звернутися до методу класу з бібліотеки, написаної на Delphi. Кожна бібліотека (збірка) в .NET має відомості про свою версію, що дозволяє усунути можливі конфлікти між різними версіями збірок.

Середовища розробки, що підтримують .NET:

- Microsoft Visual Studio (C #, Visual Basic .NET, Managed C ++, F #)
- SharpDevelop
- MonoDevelop
- Embarcadero RAD Studio (Delphi for .NET); раніше Borland Developer Studio (Delphi for .NET, C #)
- Zonnon
- PascalABC.NET

Додатки .NET також можна розробляти в текстовому редакторі, просто викликаючи компілятор з командного рядка.

Mono

Mono - проект зі створення повноцінного втілення системи .NET Framework на базі вільного програмного забезпечення. Основний розробник проекту Mono - компанія Xamarin (раніше Novell). Після укладення Microsoft договорѐнності з Novell, платформа Mono була офіційно визнана реалізацією .NET на Unix-подібних операційних системах: Linux, Mac OS X і інших. (Хоча Mono успішно працює і під Microsoft Windows). Однак договорѐнность стосується тільки Novell і клієнтів Novell; також технології ASP.NET, ADO.NET і Windows Forms були стандартизовані ECMA / ISO, і використання їх в Mono знаходиться під загрозою юридичних претензій з боку Microsoft, тому Mono надає реалізацію ASP.NET, ADO.NET і Windows.Forms, але в одночас рекомендує не використовувати ці API.

Тема 4. Віддалений виклик процедур

Ідея виклику віддалених процедур (Remote Procedure Call — RPC) полягає в розширенні механізму передачі управління і даних усередині програми, що виконується на одній машині, на передачу управління і даних через мережу.

Кошти віддаленого виклику процедур призначені для:

- полегшення організації розподілених обчислень;
- створення розподілених клієнт-серверних інформаційних систем.

Найбільша ефективність використання RPC досягається в тих додатках, в яких існує інтерактивний зв'язок між віддаленими компонентами з невеликим часом відповідей і відносно малою кількістю переданих даних. Такі програми називаються ***RPC-орієнтованими***.

Характерними **рисами виклику віддалених процедур** є:

- *асиметричність* – одна з взаємодіючих сторін є ініціатором;
- *синхронність* - виконання виклику процедури припиняється з моменту видачі запиту і відновлюється тільки після повернення з виклику процедури.

Проблеми і завдання, які необхідно вирішити при реалізації RPC:

1. Процедура, яка викликає і яка викликається виконуються на різних машинах і мають різні адресні простори, що створює проблеми при передачі параметрів і результатів, особливо якщо машини знаходяться під управлінням різних операційних систем або мають різну архітектуру.
2. на відміну від локального виклику віддалений виклик процедур обов'язково використовує транспортний рівень мережевої архітектури (наприклад, TCP), проте це залишається прихованим від розробника.
3. у реалізації RPC беруть участь як мінімум два процеси — по одному в кожній машині. У разі, якщо один з них аварійно завершиться, можуть виникнути наступні ситуації:
   - при аварії процедури, яка викликає віддалено викликані процедури стануть «сиротами»;
   - при аварійному завершенні віддалених процедур викликаючі процедури стануть «знедоленими батьками», які будуть марно чекати відповіді від віддалених процедур.
4. Неоднорідність мов програмування і операційних середовищ => проблема сумісності, досі не вирішена ні з допомогою введення одного загальноприйнятого стандарту, ні з допомогою реалізації кількох конкуруючих стандартів на всіх архітектурах і у всіх мовах.

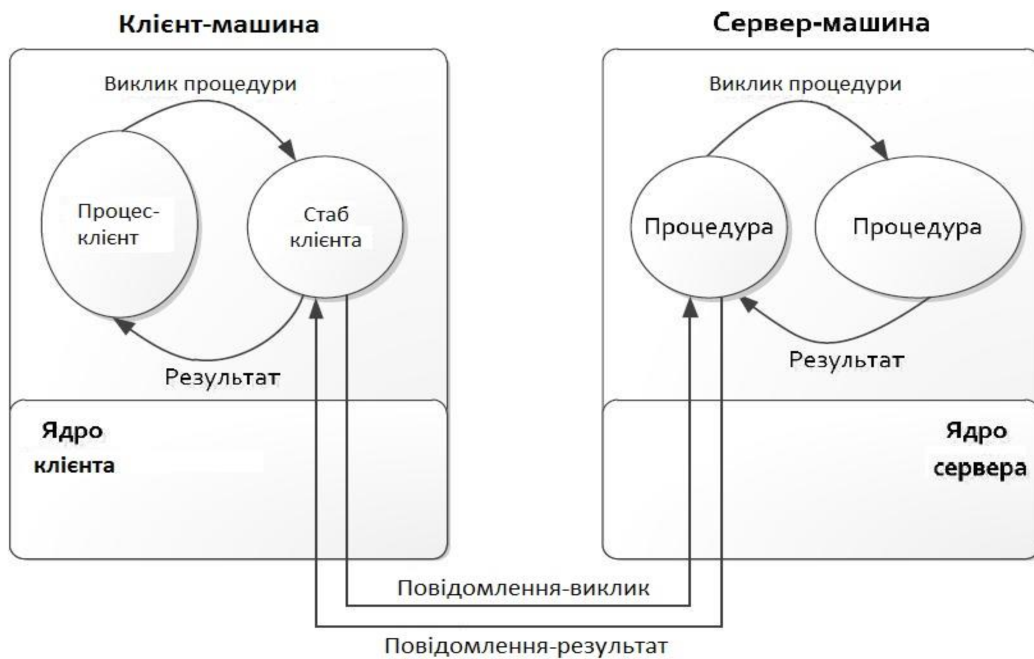
Базові операції RPC

Щоб зрозуміти роботу RPC, розглянемо спочатку **виконання виклику локальної процедури** в звичайному комп'ютері, що працює автономно. Щоб здійснити виклик, викликаюча процедура поміщає параметри в стек в зворотному порядку. Після того, як виклик виконаний, він поміщає обчислене значення в регістр, переміщує адрес повернення і повертає управління викликаючій процедурі, яка обирає параметри з стека, повертаючи його в початковий стан.

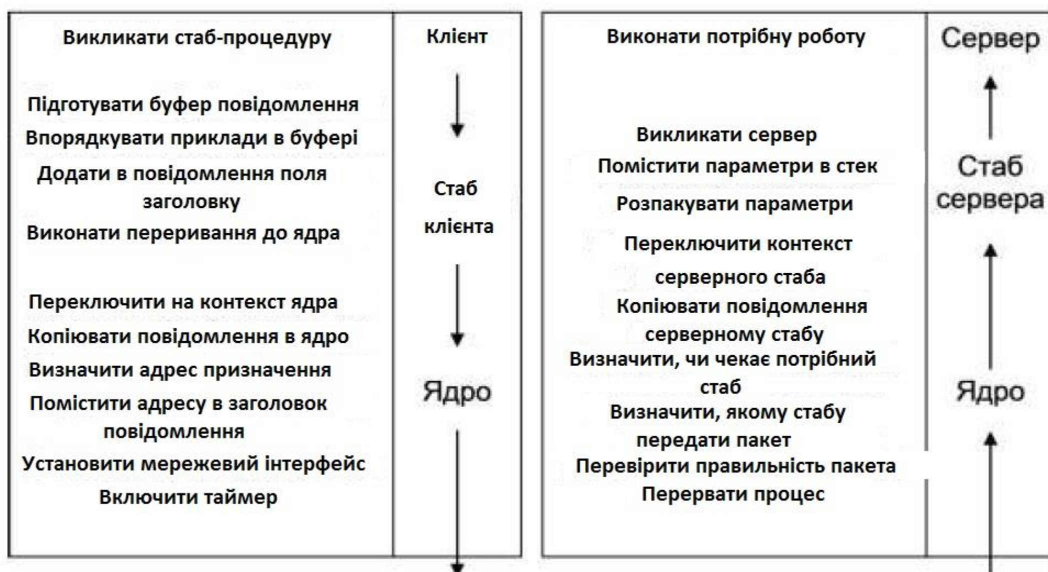
Ідея, покладена в основу RPC, полягає в тому, щоб зробити вигляд виклику віддаленої процедури по можливості так само, як і виклик локальної процедури. Іншими словами - викликаючій процедурі не потрібно знати, що процедура, що викликається знаходиться на іншій машині, і навпаки.

RPC досягає прозорості наступним шляхом. Коли процедура, яка викликається дійсно є віддаленою, в бібліотеку поміщається замість локальної процедури інша версія процедури, звана клієнтським стабом (англ. stub - заглушка). Подібно до оригінальної процедури, стаб викликається з використанням викликающої послідовності, так само відбувається переривання при зверненні до ядра. Тільки на відміну від оригінальної процедури він не поміщає параметри в регістри і не запитує ядра дані, замість цього він формує повідомлення для відправки ядру віддаленої машини.

Взаємодія програмних компонентів при виконанні віддаленого виклику процедури



Етапи виконання процедури RPC



Після того, як клієнтський стаб був викликаний програмою-клієнтом, його першим завданням є ***заповнення буфера повідомлень повідомленням***. В деяких системах клієнтський стаб має єдиний буфер фіксованої довжини, що заповнюється кожен раз з самого початку при надходженні кожного нового запиту. В інших системах буфер повідомлення являє собою пул буферів для окремих полів повідомлення, причому деякі з цих буферів вже заповнені.

Потім виконується ***перетворення параметрів у відповідний формат і їх вставка в буфер повідомлення***. До цього моменту повідомлення готове до передачі, тому виконується ***переривання за викликом ядра***.

Коли ядро отримує управління, воно перемикає контексти, зберігає регістри процесора і карту пам'яті (дескриптори сторінок), встановлює нову карту пам'яті, яка буде використовуватися для роботи в режимі ядра. Оскільки контексти ядра і користувача розрізняються, ядро має скопіювати повідомлення в свій власний адресний простір, запам'ятати ***адресу призначення***, після чого ***передати його мережевому інтерфейсу***.

На цьому завершується робота на клієнтській стороні.

Включається ***таймер передачі***, і ядро може виконувати або циклічне опитування наявності відповіді, або передати управління планувальнику, який обере який-небудь інший процес на виконання.

На стороні сервера ***біти, що надходять поміщаються*** приймаючою апаратурою або у ***вбудований буфер, або в оперативну пам'ять***. Коли вся інформація буде отримана, генерується переривання.

Обробник переривання перевіряє правильність даних пакета і визначає, якому стабу слід їх передати. Якщо жоден з стабів не очікує цей пакет, обробник повинен або помістити його в буфер, або взагалі відмовитися від нього. Якщо є очікуючий стаб, то повідомлення копіюється йому. Нарешті, виконується ***переключення контекстів***, в результаті чого відновлюються регістри і карта пам'яті, приймаючи ті значення, які вони мали в момент, коли стаб зробив виклик.

Тепер починає роботу ***серверний стаб***. Він розпаковує параметри і поміщає їх відповідним чином в стек. По завершенні роботи, виконується виклик сервера.

Після виконання процедури сервер передає результати клієнту. Для цього виконуються всі описані вище етапи, тільки в зворотному порядку.

Реалізації RPC

Різні реалізації RPC мають дуже різну один від одного архітектуру і різняться у своїх можливостях: одні реалізують архітектуру **SOA**, інші **CORBA** або **DCOM**.

Існує безліч **технологій, що забезпечують RPC**, наприклад:

DCE/RPC — Distributed Computing Environment / Remote Procedure Calls (бінарний протокол на базі різноманітних транспортних протоколів, у тому числі TCP/IP);

DCOM — Distributed Component Object Model відомий як MSRPC Microsoft Remote Procedure Call або «Network OLE» (об'єктно-орієнтоване розширення

DCE/RPC, що дозволяє передавати посилання на об'єкти і викликати методи об'єктів через такі посилання);

JSON-RPC— JavaScript Object Notation Remote Procedure Calls- протокол, який використовує JSON (текстовий формат обміну даними, заснований на JavaScript) для кодування повідомлень; JSON-RPC визначає кілька типів даних і команд, підтримує попередження і численні виклики.

.NET Remoting - створений компанією Microsoft компонент, що надає API для міжпроцесної взаємодії. Є частиною пакету .NET Framework 1.0 і фактично являє собою Microsoft-реалізацію протоколу SOAP.

Java RMI — Java Remote Method Invocation — програмний інтерфейс виклику віддалених методів в мові Java, що представляє собою розподілену об'єктну модель, специфіковану, способи викликів віддалених методів, що працюють на іншій віртуальній машині Java.

XML RPC – XML- виклик віддалених процедур – протокол, який використовує XML для кодування повідомлень і HTTP якості транспортного механізму. Є прабатьком SOAP.

SOAP — протокол обміну структурованими повідомленнями в розподіленому обчислювальному середовищі. Спочатку SOAP призначався в основному для реалізації RPC, але зараз використовується для обміну довільними повідомленнями у форматі XML, а не тільки для виклику процедур.

Internet Communications Engine (Ice) – розроблена ZeroC об'єктна система проміжного шару, що використовує механізм віддаленого виклику процедур. Поширюється під подвійною ліцензією: GNU GPL (вільне) або комерційної. Ice підтримує багато платформ програмування, включаючи C++, Java, .NET, Visual Basic, Python, Ruby і PHP. Ice успішно конкурує з SOAP, оскільки використовує бінарний протокол передачі даних, забезпечуючи менше навантаження на мережу і процесор.

Організація зв’язку з використанням віддалених об’єктів

Об'єктно-орієнтована технологія в даний час широко застосовується при розробці додатків, в тому числі і розподілених. Одним з найбільш важливих **властивостей об'єкта** є те, що він *приховує особливості реалізації*, надаючи можливість для взаємодії суворо описаний інтерфейс. Це дозволяє заміняти або змінювати об'єкти, залишаючи незмінним інтерфейс.

Розвиток клієнт-серверної архітектури на початку 1990-х років призвело до формування *об'єктно-орієнтованої концепції* розподілених систем, орієнтованої на інкапсуляцію механізму розподілених взаємодій і зменшення складності розробки розподілених додатків за допомогою методів об'єктно-орієнтованої розробки і *віддалених викликів методів* об'єктів.

Основними перевагами даного підходу стали:

- спрощення розробки розподілених додатків порівняно з класичним клієнт/серверним підходом;

- можливість розробки додатків для гетерогенних обчислювальних середовищ (забезпечувалася застосуванням віртуальних машин, наприклад, Java, і незалежним описом інтерфейсів взаємодіючих компонентів);
- можливість відділення інтерфейсу віддаленого об'єкта від його безпосередньої реалізації.

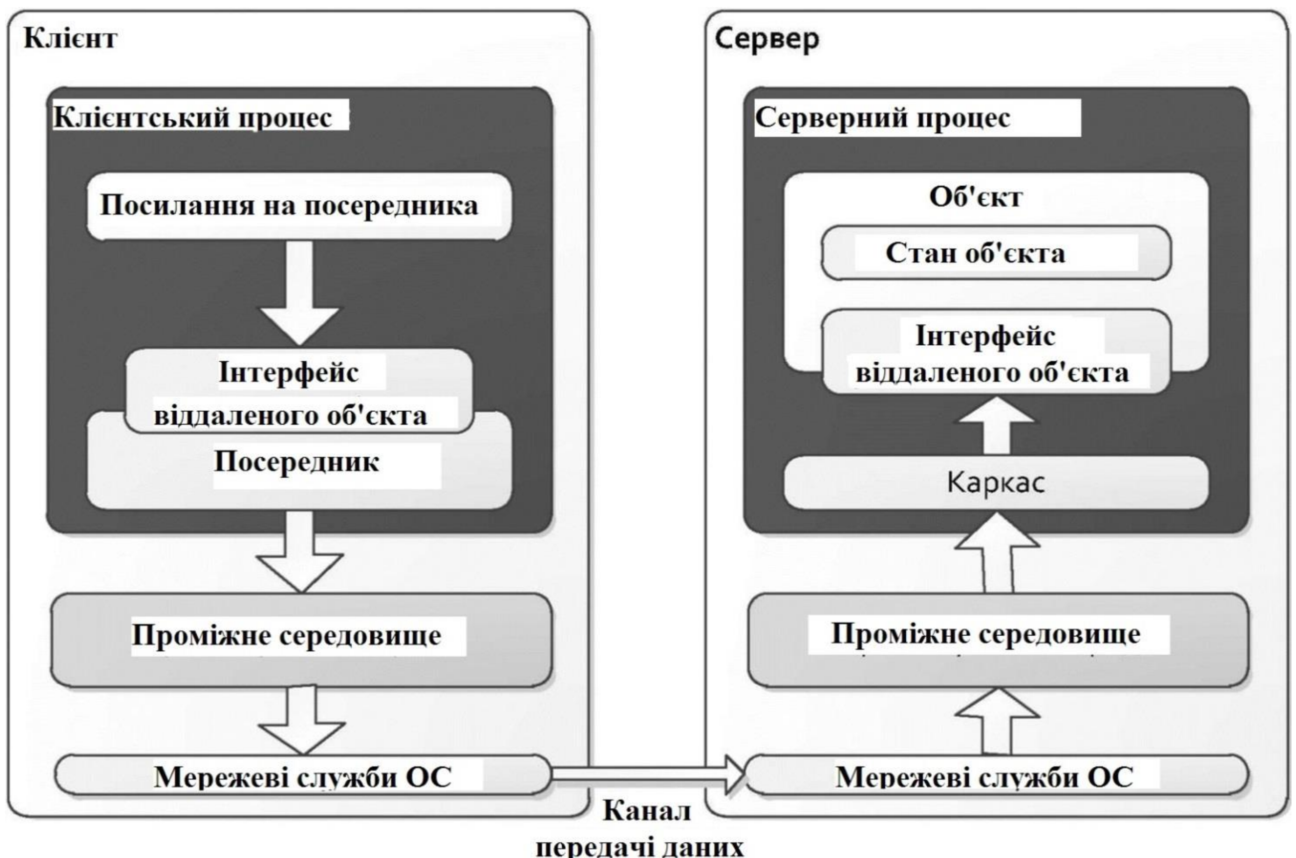
Віддалений об'єкт являє собою деякі дані, сукупність яких визначає його ***стан***. Цей стан можна змінювати шляхом виклику його ***методів***. Якщо можливий прямий доступ до даних віддаленого об'єкта, це відбувається за допомогою неявного віддаленого виклику, необхідного для передачі значення поля даних об'єкта між процесами. Методи і поля об'єкта, які можуть використовуватися через віддалені виклики, доступні через деякий ***зовнішній інтерфейс*** класу об'єкта.

Для передачі параметрів по мережі використовується серіалізація об'єктів і даних.

Серіалізація – це переклад стану об'єкта в послідовність біт (найчастіше, бінарний або XML-файл), після чого його копія може бути передана в інший процес.

Зворотний процес - **десеріалізація** - це відновлення стану об'єкта з прийнятої послідовності бітів.

Використання віддалених об’єктів



У момент, коли клієнт починає використовувати віддалений об'єкт, на стороні клієнта створюється клієнтська заглушка, яка називається ***посередником*** (англ. *proxy*). Посередник реалізує той же інтерфейс, що і віддалений об'єкт.

Зухвалий процес використовує методи посередника, який використовує їх параметри для передачі по мережі, і передає їх по мережі сервера. Проміжне середовище на стороні сервера невикористовує параметри і передає їх заглушці на стороні сервера, яку називають ***каркасом*** (*skeleton*) або, як і в віддаленому виклику процедур, ***заглушкою***. Каркас зв'язується з деякими ***екземпляром*** віддаленого об'єкта. Це може бути як новостворений, так і існуючий екземпляр об'єкта, в залежності від застосовуваної моделі використання віддалених об'єктів.

При використанні віддалених об'єктів виникає ряд проблемних питань, пов'язаних з часом їх життя:

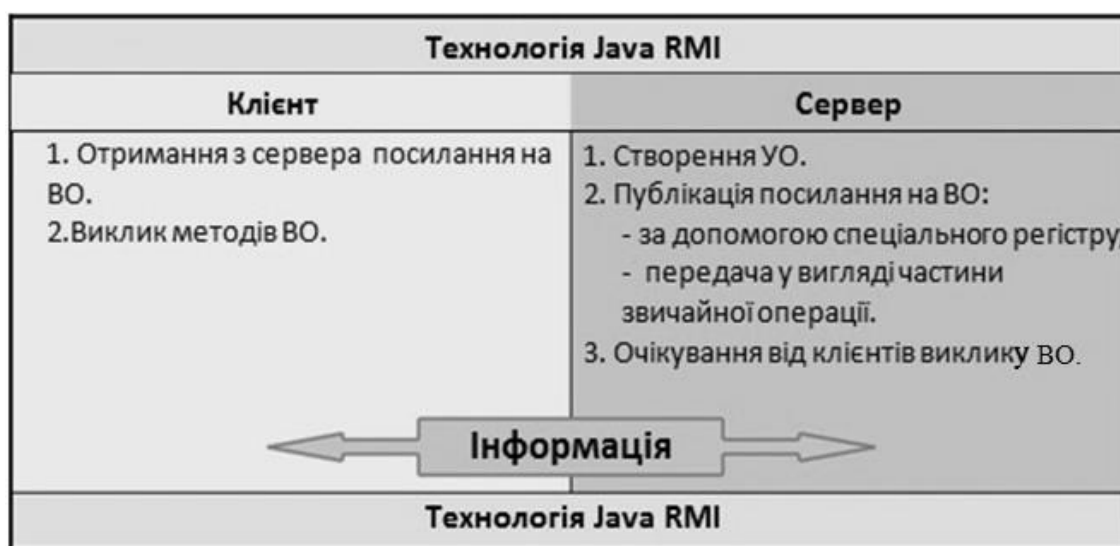
1. коли саме часу створюється екземпляр віддаленого об'єкта;
2. протягом якого проміжку часу він існує.

Для опису життєвого циклу в системах з віддаленими об'єктами використовуються два додаткових поняття:

- **активація об'єкта** - процес перекладу створеного об'єкта в стан обслуговування віддаленого виклику, тобто зв'язування з каркасом і посередником.
- **деактивація об'єкта** - процес переведення об'єкта в невикористовуваний стан.

Java RMI

Технологія ***Java RMI*** (*Remote Method Invocation - виклик віддалених методів*) дозволяє забезпечити прозорий доступ до методів віддалених об'єктів (ВО), забезпечуючи доставку параметрів викликається методу, повідомлення об'єкту про необхідність виконання методу і передачу повертається клієнтові назад.



Переваги:

- можливість розробляти систему цілком грунтуючись на об'єктно-орієнтованої концепції, не заглиблюючись в розробку власних протоколів взаємодії між розподіленими компонентами систем;
- кроссплатформенність, що надається віртуальною машиною Java.

Недоліки:

- сувора обмеженість даної технології платформою Java;
- необхідність обробки з'єднань між розподіленими компонентами програми => обмеження масштабованості.

CORBA

CORBA (**Common Object Request Broker Architecture**- загальна архітектура брокера об'єктних запитів) - це технологія розробки розподілених додатків, орієнтована на інтеграцію розподілених ізольованих систем.

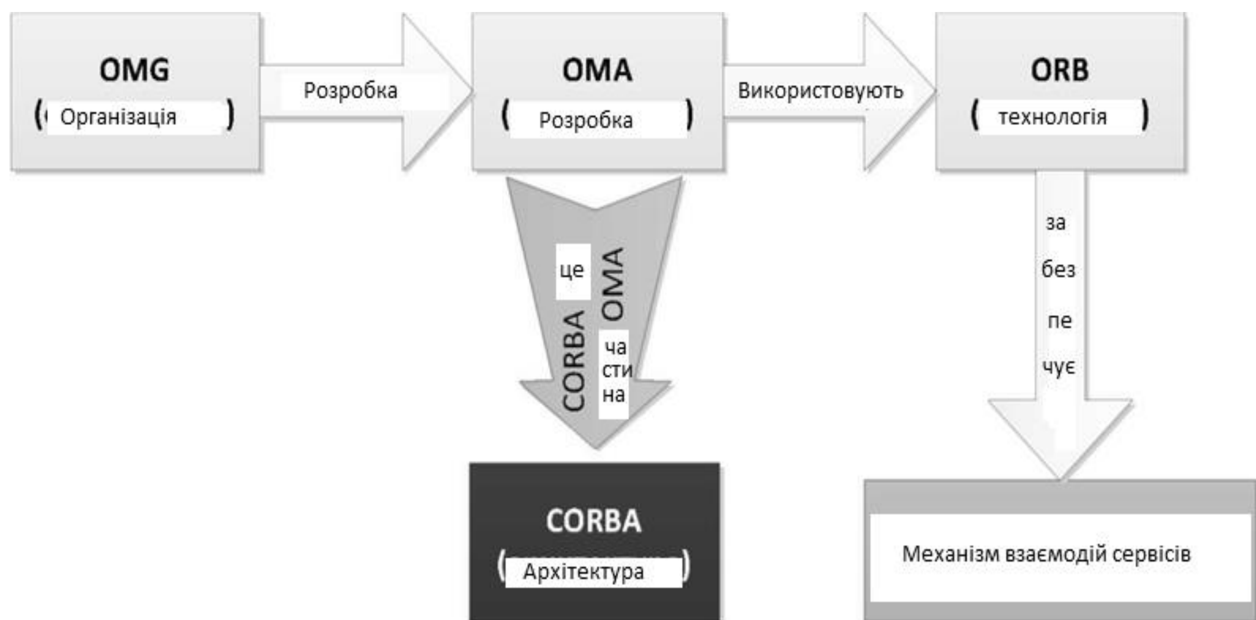
Основне завдання OMG - розробка і просування об'єктно-орієнтованих технологій і стандартів. Це некомерційне об'єднання, що розробляє стандарти для створення корпоративних платформо-незалежних додатків.

Консорціум OMG був створений **11 компаніями**, серед яких були як *розробники*, так і *споживачі* програмного забезпечення. Серед творців OMG слід зазначити такі компанії як **Hewlett-Packard, IBM, Sun Microsystems, Apple Computer, American Airlines** і **Data General**. Довгий час майбутнє стандартів, що просуваються OMG (в першу чергу CORBA), піддавалося сумніву в деяких кругах. Однак зараз консорціум включає близько **800 компаній**.

Концептуальною інфраструктурою, на якій базуються всі специфікації OMG, є **Object Management Architecture (OMA).** До складу OMA входять

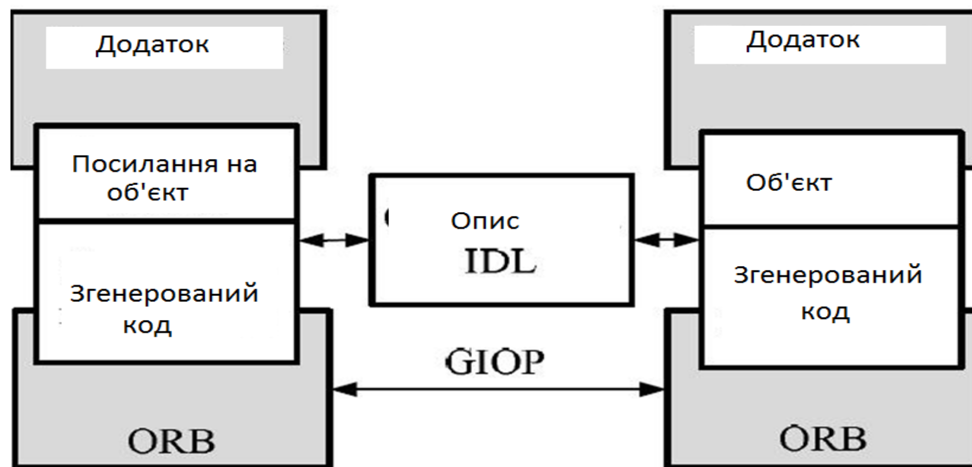
різноманітні стандартизовані або зараз стандартизовані OMG сервіси, програмні зразки і шаблони, мова визначення інтерфейсів розподілених об'єктів **IDL (Interface Definition Language)**, стандартизовані або стандартизовані відображення IDL на мови програмування і, нарешті, об'єктна модель **CORBA**. Головною особливістю CORBA є використання компонента **ORB (Object Resource Broker** - *брокер ресурсів об'єктів*) для створення екземплярів об'єктів і виклику їх методів. Даний компонент формує «міст» між додатком і інфраструктурою CORBA.

Основні поняття технологій CORBA



У 1997 році консорціум OMG опублікував специфікацію CORBA 2.0. У ній визначалися стандартний протокол і відображення для мови С ++, а в 1998 р. було визначено відображення для Java. В результаті розробники отримали інструментальне засіб, що дозволяє їм відносно легко створювати неоднорідні розподілені додатки. CORBA швидко завоювала популярність, і з використанням цієї технології був створений ряд критично важливих додатків.

Технологія CORBA



Ядро архітектури CORBA

Технологічний стандарт CORBA визначає мову **IDL**, застосовуваний для уніфікованого опису інтерфейсів розподілених об'єктів, і його відображення на мови Ada, C, C ++, Java, Python, COBOL, Lisp, PL / 1 і Smalltalk.

Для перетворення опису інтерфейсу на мові IDL на потрібну мову програмування використовується спеціальний ***компілято***р. Надалі побудований з його допомогою програмний код може бути перетворений будь-яким стандартним компілятором в виконуваний код.

Для створення екземплярів об'єктів і виклику їх методів використовується компонент **ORB**, який формує «міст» між додатком і інфраструктурою CORBA. ORB підтримує віддалене взаємодія з іншими ORB, а також забезпечує управління УО, включаючи облік кількості посилань і часу життя об'єкта.

Для забезпечення взаємодії між ORB використовується протокол **GIOP** (**General Inter-ORB Protocol** - *загальний протокол для комунікації між ORB*). Найбільш поширеною реалізацією даного протоколу є протокол **IIOP** (**Internet Inter-ORB Protocol** - *протокол взаємодії ORB в мережі інтернет*), що забезпечує відображення повідомлень GIOP на стек протоколів TCP / IP.

Спочатку, технологія CORBA орієнтована на надання готової проблемно-орієнтованої інфраструктури для створення РВС в рамках певної проблемної області. Тому до **складу CORBA входять набори**:

- ***стандартних об'єктних сервісів*** (CORBA Services):

- *NameService* - сервіс імен;

- *сервіс повідомлень*, що дозволяє CORBA-об'єктів обмінюватися повідомленнями;

- *сервіс транзакцій*, що дозволяє CORBA-об'єктів організовувати транзакції і ін.

- ***загальних засобів*** (Common Facilities):

- *горизонтальні* - загальні для всіх прикладних областей;

- *вертикальн*і - для конкретної прикладної області (наприклад, для медичних організацій, для ряду виробничих сфер і ін.).

Між об'єктними сервісами і загальними засобами CORBA немає чіткої межі. І перші, і другі являють собою CORBA-*об'єкти зі стандартизованими інтерфейсами*. Слід зазначити, що в реальній системі не обов'язково повинні бути присутніми всі сервіси, їх набір залежить від необхідної функціональності. На сьогодні розроблено 14 об’єктних сервісів.

Розробка на основі CORBA

Процес розробки програми з використанням технології CORBA складається з наступних 4-х етапів:

1. Визначення інтерфейсу на IDL.
2. Обробка IDL для створення коду заглушки і скелетона.
3. Створення коду реалізації об'єкта (сервер).
4. Створення коду використання даного об'єкта (клієнт).

Мова визначення IDL дозволяє незалежно від мови програмування створити універсальне опис інтерфейсу майбутньої системи.

Створений на IDL код повинен спеціальним компілятором перетворюватися в код інтерфейсу об'єкта на необхідному мовою програмування. Після чого, на клієнта автоматично генерується заглушка, що перетворює виклики методи даного інтерфейсу в обігу до ORB. На сервері програміст на основі згенерованого інтерфейсу створює власну реалізацію даного класу. Скелетон автоматизує отримання та обробку віддаленого виклику методів, що надходять через ORB.

Приклад опису на мові IDL

```
// Модуль системи цінових пропозицій
module QuoteSystem
{
    // Структура даних ціни
    struct Quote
    {
        string value;
    }
    // Інтерфейс сервера цінових пропозицій
    interface QuoteServer
    {
        // Атрибут, який визначає фондову біржу
        string exchange;
        // Виключення "Невідомий ідентифікатор"
        exception UnknownSymbolException { string message; };
        // Пошук пропозицій по ідентифікатору
        Quote getQuote   (in string symbol)
                        raises   (UnknownSymbolException);
};
```

У порівнянні з класичним клієнт-серверних підходом, використання технології CORBA для розробки розподілених додатків має такі **переваги**:

- використання IDL для опису інтерфейсів, що дозволяє розробляти програмні компоненти незалежно від мови програмування і базової операційної системи;
- підтримка багатою інфраструктурою розподілених об'єктів;
- прозорість виклику віддалених об'єктів.

Однак програмні рішення на базі технології CORBA рідко виходять за рамки окремих підприємств. Розробка великомасштабних міжустановних систем на базі технології CORBA поєднана з такими **труднощами**:

- погана сумісність різних реалізацій технології CORBA від різних постачальників;
- проблеми взаємодії вузлів CORBA через Інтернет;
- неузгодженість багатьох архітектурних рішень CORBA і відсутність компонентної моделі, яка могла б значно спростити розробку.

На зміну технології CORBA, прийшли стандартизовані протоколи веб-сервісів, такі як XML, WSDL, SOAP і ін. В даний час CORBA використовується для реалізації вузького кола успадкованих додатків.

Тема 5. Компонентні системи

Компонентно-орієнтований підхід (КОП) до проектування і реалізації програмних систем і комплексів є в деякому сенсі розвитком о***б'єктно-орієнтованого підходу*** і практично більш придатний для розробки великих і розподілених програмних систем (наприклад, корпоративних додатків).

З точки зору КОП **програмна система** - це набір компонентів з чітко визначеним інтерфейсом. На відміну від інших підходів програмної інженерії, зміни в систему вносяться шляхом створення нових компонентів або зміни старих, а не шляхом рефакторінга існуючого коду.

Програмний компонент - це автономний елемент програмного забезпечення, призначений для багаторазового використання, який може поширюватися для використання в інших програмах у вигляді скомпільованого коду. Підключення до програмних компонентів здійснюється за допомогою відкритих інтерфейсів, а взаємодія з програмним середовищем здійснюється по подіях, при тому в програмі, що використовує компонент, можна призначати обробники подій, на які вміє реагувати компонент.

Застосування компонентного програмування покликане забезпечити більш просту, швидку і прямолінійну процедуру початкової інсталяції прикладного програмного забезпечення, а також збільшити відсоток повторного використання коду, тобто ***посилити основні переваги ООП***.

Властивості компонентів:

- компоненти - це істотно більші одиниці, ніж об'єкти;
- можливість містити множинні класи;
- незалежність від мови програмування.

Слід зазначити, що автор і користувач компонента територіально розподілені і можуть не тільки писати програми, а й говорити на різних мовах.

Можна виділити наступні **основні переваги застосування КОП** при проектуванні і розробці ПО:

- зниження вартості програмного забезпечення;
- підвищення повторного використання коду;
- уніфікація обробки об'єктів різної природи;
- менше людино-залежний процес створення програмного забезпечення.

Так як програмний компонент (ПК) має на увазі повноцінне автономне використання у вигляді «чорного ящика», до розробки ПК ставлять серйозні **вимоги**:

- ***повна документованість інтерфейсу:*** всі методи, що надаються в інтерфейсі ПК, повинні бути якісно задокументовані, з урахуванням всіх можливих варіантів їх використання в сторонніх додатках;
- ***ретельне тестування:*** необхідно врахувати всі можливі і неможливі варіанти використання ПК в сторонніх системах на всіх можливих значеннях вхідних даних;
- ***ретельний аналіз вхідних значень:*** необхідно враховувати можливість передачі в ПК вхідних даних, які не відповідають його специфікації і адекватно обробляти такі ситуації;
- ***повернення адекватних і зрозумілих повідомлень про помилки:*** так як один ПК може бути використаний в великому числі сторонніх програмних систем, необхідно забезпечити стороннім розробникам можливість отримання інформації про помилки ПК і можливі варіанти їх вирішення;
- ***необхідність передбачити можливість неправильного використання.***

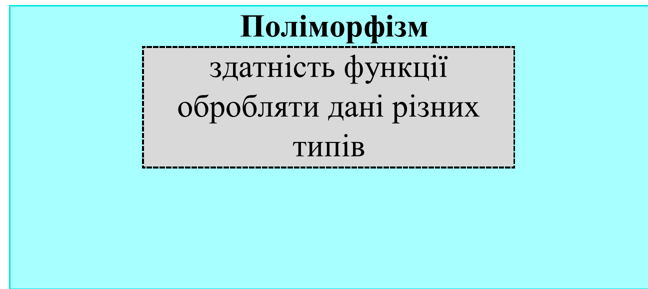
Component Object Model

COM (**Component Object Model** — *об'єктна модель компонентів*) - це технологічний стандарт від компанії Microsoft, призначений для створення ПЗ на основі взаємодіючих компонентів, кожен з яких може використовуватися в багатьох програмах одночасно. Стандарт втілює в собі ідеї ***поліморфізму*** та ***інкапсуляції*** ООП.

Стандарт COM був розроблений в 1993 році корпорацією Microsoft як основа для розвитку технології **OLE (Object Linking and Embedding** - *зв'язування і вбудовування об'єкта*). Існуюча на той момент технологія OLE 1.0 вже дозволяла створювати т.зв. «Складові документи» (*compound documents*) (наприклад в пакеті Microsoft Office, включати діаграми Microsoft Excel в документи Microsoft Word).

Стандарт COM міг би бути *універсальним* і від платформи *незалежним*, але закріпився в основному на ОС сімейства Microsoft Windows.

На основі COM були реалізовані технології: Microsoft OLE Automation, ActiveX, DCOM, COM +, DirectX, а також XPCOM.



Спеціальний (ad hoc)

«багато реалізацій зі схожими інтерфейсами»

```
program Adhoc;

function Add( x, y : Integer ) : Integer;
begin
    Add := x + y
end;

function Add( s, t : String ) : String;
begin
    Add := Concat( s, t )
end;

begin
    Writeln(Add(1, 2));
    Writeln(Add('Hello, ', 'World!'));
end.
```

Параметричний

«одна реалізація з узагальненим інтерфейсом»

```
data List a = Nil | Cons a (List a)

length :: List a -> Integer
length Nil = 0
length (Cons x xs) = 1 + length xs

map :: (a -> b) -> List a -> List b
map f Nil = Nil
map f (Cons x xs) = Cons (f x) (map f xs)
```

Принципи роботи COM

Основним поняттям, яким оперує стандарт COM, є **COM-компонент**.

Програми, побудовані на стандарті COM, фактично не є автономними програмами, а являють собою набір взаємодіючих між собою COM-компонентів.

Кожен компонент має унікальний ідентифікатор **(GUID - Globally Unique Identifier)** і може одночасно використовуватися багатьма програмами.

Компонент взаємодіє з іншими програмами через **COM-інтерфейси** - набори абстрактних функцій і властивостей. Кожен COM-компонент повинен, як мінімум, підтримувати стандартний інтерфейс «**IUnknown**», який надає базові засоби для роботи з компонентом і ***включає в себе три методи:***

- QueryInterface служить для отримання покажчика на інтерфейс COM-компонента;
- AddRef збільшує число посилань на даний COM-компонент на одиницю.

- Release зменшує число посилань на даний COM-компонент на одиницю. Якщо при черговому виклику Release число посилань дорівнювала нулю, COM-компонент повинен знищитися і звільнити всі ресурси.

Разом функції AddRef і Release служать для управління часом життя COM-компонента, що експортує інтерфейси.

Базові функції, що дозволяють використовувати COM-компоненти, надає **Windows API**.

Windows API спроектований для використання в мові C для написання прикладних програм, призначених для роботи під управлінням операційної системи MS Windows, і являє собою безліч функцій, структур даних і числових констант, таких угодам мови C. Всі мови програмування, здатні викликати такі функції і оперувати такими типами даних в програмах, що виконуються в середовищі Windows, можуть користуватися цим API. Зокрема, це мови **C ++, Pascal, Visual Basic** і багато інших.

Існують бібліотеки і середовища програмування, що надають ту чи іншу частину Windows API в більш зручному вигляді, наприклад, розроблені Microsoft: **MFC, ATL / WTL, .Net / WinForms / WPF**.

Microsoft Foundation Classes (MFC) - бібліотека мовою C ++, покликана полегшити розробку GUI-додатків для Microsoft Windows шляхом використання

багатого набору бібліотечних класів. MFC створює каркас додатка - «кісткову» програму, автоматично створювану за заданим макету інтерфейсу і повністю бере на себе рутинні дії з обслуговування додатки (відпрацювання віконних подій, пересилання даних між внутрішніми буферами елементів і змінними і т.п.). Програмісту після генерації каркаса додатки необхідно тільки вписати код в місця, де потрібні спеціальні дії.

Delphi

```
TMyClass = class
private
  FMyField: Integer;
  procedure SetMyField(const Value: Integer);
  function GetMyField: Integer;
public
  property MyField: Integer read GetMyField write SetMyField;
end;
```

Active Template Library (ATL) - набір шаблонних класів мови C ++, розроблених для спрощення написання COM-компонентів. Ця бібліотека дозволяє розробникам створювати різні об'єкти COM, сервери автоматизації OLE і керуючі елементи ActiveX. Середовище розробки Visual Studio включає майстри і помічники для ATL, що дозволяють створити первинну об'єктну структуру практично без програмування вручну. ATL - це в деякій мірі облегчѐнная альтернатива MFC як засіб управління COM.

Windows Template Library (WTL) - вільно розповсюджувана бібліотека шаблонів (шаблонних класів) C ++, призначена для написання стандартних GUI додатків Windows, що є розширенням бібліотеки ATL. WTL є надбудову над

інтерфейсом Win32 API, і в першу чергу розроблялася як облегчѐнная альтернатива бібліотеці MFC. WTL підтримує роботу з вікнами і діалогами, стандартними діалогами Windows, GDI (*Graphics Device Inte*rface - інтерфейс Windows для представлення графічних об'єктів і передачі їх на пристрої відображення (напр., Монітори, принтери)), стандартними контролю, ActiveX і ін. В бібліотеці представлені основні елементи управління: меню, панелі інструментів, кнопки, поля введення, списки і т. д.

Технології, засновані на стандарті COM

OLE (Object Linking and Embedding - зв'язування і вбудовування об'єкта) - технологія зв'язування та впровадження об'єктів в інші документи і об'єкти. OLE дозволяє передавати частину роботи від однієї програми редагування до іншої і повертати результати назад. Наприклад, встановлена на персональному комп'ютері видавнича система може послати якийсь текст на обробку в текстовий редактор, або деяке зображення в редактор зображень за допомогою OLE-технології. У 1996 році була спробувала перейменувати OLE в ActiveX, але це вдалося лише частково і призвело до плутанини в термінах.

Терміном ActiveX зараз називають все, що відноситься до OLE, плюс деякі нововведення, проте згоди з питання про точне визначення ActiveX серед експертів по DCOM (навіть всередині Microsoft) не існує.

COM + (раніше **Microsoft Transaction Server, MTS**) - призначена для підтримки систем обробки транзакцій. Технологія COM + базується на можливостях COM і забезпечує підтримку розподілених додатків на компонентної основі. Об'єкти транзакцій COM + мають основними властивостями об'єктів COM. Крім цього, об'єкти транзакцій реалізують **специфічні можливості**:

- управління транзакціями;
- безпеку;
- пулінг ресурсів;
- пулінг об'єктів.

Особливості COM + об'єктів:

- реалізація в складі внутрішнього сервера (динамічна бібліотека);
- наявність посилання на бібліотеку типів COM +;
- використання тільки стандартного механізму маршалинга COM;
- імплементація інтерфейсу IobjectControl.

Можливі типи об'єктів:

- statefull (зі збереженням інформації про стан об'єкта);
- stateless (без збереження інформації про стан об'єкта).

Принцип роботи COM +

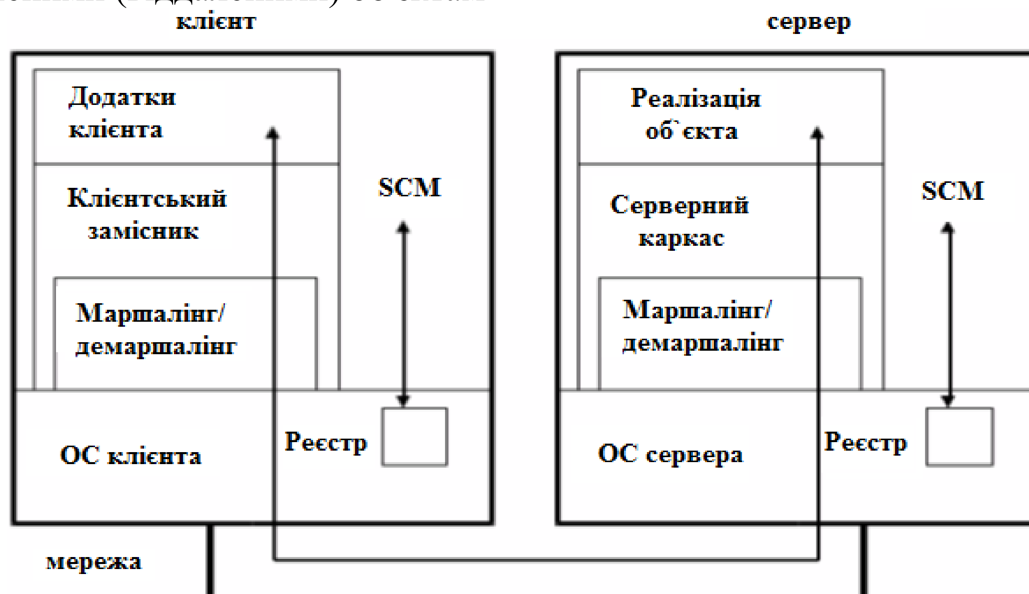
COM + - це сукупність програмних засобів, що забезпечують розробку, поширення і функціонування розподілених додатків для мереж Інтернет і інтранет. До її **складу** входять:

- ПО проміжного рівня, що забезпечує функціонування об'єктів транзакцій під час виконання.
- Утиліта MTS Explorer, що дозволяє управляти об'єктами транзакцій.
- Інтерфейси прикладного програмування.
- Засоби управління ресурсами.

Стандартна програмна модель додатків, що використовують COM +, являє собою триланкову архітектуру розподілених додатків. При цьому бізнес-логіка програми сконцентрована в **об'єктах транзакцій**, а ПО проміжного рівня, що управляє цими об'єктами, побудовано з використанням компонентної моделі. Розробники, що використовують COM + в своїх додатках, створюють об'єкти бізнес-логіки, що задовольняють вимогам до об'єктів COM +; потім компілюють їх і встановлюють в середовищі COM + за допомогою пакетів. **Пакет** COM + являє собою контейнер, що забезпечує угруповання об'єктів з метою захисту даних, поліпшення управління ресурсами і збільшення продуктивності. Управління пакетами здійснюється за допомогою утиліти MTS Explorer.

DCOM

DCOM (Distributed COM) - розширення COM для підтримки зв'язку між розподіленими (віддаленими) об'єктам



Для створення об'єкта на удалѐнной машині, бібліотека COM викликає менеджер управління сервісами (**SCM**) локального комп'ютера, який зв'язується з **SCM** сервера і передаѐт йому запит на створення об'єкта. Ім'я сервера може здаватися при виконанні функції створення об'єкта або зберігатися в реєстрі.

Для виклику удалѐнного об'єкта параметри повинні бути вилучені з ***стека*** (або з ***регістрів процесора***), поміщені в буфер і передані через мережу. Процес вилучення параметрів і розміщення їх в буфер називається **маршалінг**. Цей процес нетривіальний, оскільки параметри можуть містити покажчики на масиви і структури, які, в свою чергу, можуть містити покажчики на інші структури. На сервері проводиться зворотний процес відтворення стека (**демаршалінг**), після чого викликається потрібний об'єкт. Після завершення виклику проводиться маршалинга повертається і вихідних параметрів і відправка їх клієнту.

Для виконання маршалинга і демаршалінга необхідно мати точний опис методу, включаючи всі типи даних і розміри масивів. Для опису використовується мова опису інтерфейсів (IDL), що входить в стандарт DCE RPC. Отримані файли опису компілюються спеціальним компілятором IDL в вихідний код на мові C, що виробляє маршалинга і демаршалінг для зазначених інтерфейсів. Код, що запускається на стороні клієнта, називається «проксі», на стороні об'єкта - «стаб» і завантажується бібліотекою COM за потребою.

Маршалинга VS Серіалізация:

Раунд 1 «Вікіпедія»

У стандартній бібліотеці мови **Python** термін **«маршалінга»** схожий за значенням з терміном «сериализация», але терміни не є синонімами в документі RFC 2713, на який спирається стандарт Java:

«Маршалізувати» об'єкт означає записати його стан і кодову (і) базу (и) таким чином, щоб при зворотній операції виходила копія оригіналу, імовірно за допомогою автоматичного завантаження визначення класу об'єкта. Маршалинга подібний сериализации, за винятком того, що при маршалинга записуються кодові бази. Маршалинга відрізняється від сериализации тим, що маршалинга особливим чином застосовується до віддалених об'єктів.

«Серіалізірувати» об'єкт означає перетворювати його стан в потік байт таким чином, щоб вказаний потік міг бути перетворений назад в копію вихідного об'єкта.

Маршалінг VS Серіалізация:

Раунд 2 «forum.boolean.name» (http://forum.boolean.name/showthread.php?t=16401)

http://www.gunsmoker.ru/2011/09/blog-post_9471.html

Майже синонімом терміна "серіалізація" є маршалінг (від англ. Marshal - упорядковувати). Зворотний процес - демаршалінга (аналог десеріалізації). **Маршалінг - більш загальне поняття, ніж серіалізація**. Будь-яка серіалізація є окремим випадком маршалінга. Відмінність маршалинга від серіалізації в тому, що серіалізація передбачає упаковку лише даних програми. Скажімо, матриці чисел, записи про співробітника в картотеці або таблиці висот ігрового рівня. Коли говорять про маршалінг, мають на увазі серіалізацію не тільки даних програми, але і її програмного стану, мета-інформації. Наприклад, серіалізація об'єкта з полями запише лише дані об'єкта (поля). Маршалінг ж об'єкта запише не тільки інформацію про його даних (полях), але і інформацію по відновленню структури об'єкта - клас об'єкта, або ж його мета-інформацію для реконструювання типу. Здійснюючи маршалінг і демаршалінг об'єкта ви створюєте його повну копію. При цьому приймаюча сторона може нічого не знати про об'єкт.

http://www.gotdotnet.ru/forums/2/28864/143256/

Отже, маршалінг - той великий дядько, який контролює і визначає, як будуть передаватися дані. Серіалізація - це метод, за яким користується маршалінг для здійснення своєї темної справи.

Маршалінг VS Серіалізація:

Раунд 3 «Висновки»

1.В залежності від технології (мови) і рівня абстракції (деталізації) опису того, що відбувається, значення термінів дещо розмазуються аж до їх повного злиття.

2.Серіалізація - процес перетворення об’єкта в послідовність байт.

3. Маршалінг - процес перетворення об'єктів (в їх актуальних станах) для передачі між процесами.

4. Маршалінг експлуатує серіалізацію в тому чи іншому вигляді, виходячи із завдання. Завдання, щовирішується за допомогою маршалінга полягає в тому чи іншому способі обміну об'єктами між процесами. Сама серіалізація не визначає: чи знаходяться у вихідний послідовності тільки дані або додаткова інформація про структуру об'єкта - будь-який метод перетворення об'єкта в серію байт називається серіалізацією.

CORBA VS COM

CORBA	COM/DCOM/COM+
Служба колекцій	Об'єкти даних ActiveX
Служба запитів	-
Служба паралельного доступу	Паралельні потоки управління
Служба транзакцій	Автоматичні транзакції COM +
Служба подій	Події COM +
Служба повідомлень	Події COM +
Служба зовнішніх зв'язків	Утиліти маршалінга
Служба життєвого циклу	Фабрики класів, JIT-активізація
Служба ліцензування	Спеціальні фабрики класів
Служба найменування	Монікери
Служба властивостей	-
Служба обміну	-
Служба безпеки	Спеціальне сховище
Служба відносин	-
Служба захисту	Авторизація
Авторизація	-

Концепція JavaBeans

JavaBeans - класи в мові Java, написані за певними правилам. Вони використовуються для об'єднання декількох об'єктів в один (bean) для зручної передачі даних. JavaBeans забезпечують основу для багаторазово використовуваних, вбудованих і модульних компонентів ПО.

Специфікація Sun Microsystems визначає JavaBeans, як ***«універсальні програмні компоненти, які можуть управлятися за допомогою графічного інтерфейсу»***.

Компоненти JavaBeans можуть приймати різні форми, але найбільш широко вони застосовуються в елементах графічного інтерфейсу. ***Одна з цілей створення JavaBeans - взаємодія зі схожими компонентними структурами.*** Наприклад, Windows-програма, при наявності відповідного моста або об'єкта-обèрткі, може використовувати компонент JavaBeans так, нібито він є компонентом COM або ActiveX.

Правила опису JavaBean

Щоб клас міг працювати як bean, він повинен відповідати определѐнним угодами про імена методів, конструкторі і поведінці. Ці угоди дають можливість створення інструментів, які можуть використовувати, заміщати і з'єднувати JavaBeans.

Правила опису свідчать:

- ***Клас повинен мати public конструктор без параметрів***. Такий конструктор дозволяє інструментам створити об'єкт без додаткових складнощів з параметрами.

- ***Властивості класу повинні бути доступні через get, set і інші методи доступу***, які підкоряться стандартній угоді по іменам. Це легко дозволяє інструментам автоматично визначати і оновлювати зміст bean. Багато інструменти навіть мають спеціалізовані редактори для різних типів властивостей.
- ***Клас повинен бути серіалізованим***. Це дає можливість надійно зберігати, зберігати і відновлювати стан bean незалежним від платформи і віртуальної машини способом.
- ***Клас не повинен містити ніяких методів обробки подій.***

Enterprise JavaBeans

Enterprise JavaBeans - це високорівнева, що базується на використанні компонентів технологія створення розподілених додатків, яка використовує низькорівневий API для управління транзакціями. Перший варіант специфікації Enterprise JavaBeans з'явився в березні 1998 р За час свого існування, технологія пройшла великий шлях і продовжує розвиватися.

Enterprise JavaBeans - більше, ніж просто інфраструктура. Її використання має на увазі ще й технологію (процес) створення розподіленого додатка, нав'язує певну архітектуру програми, а також визначає стандартні ролі для учасників розробки.

Застосування даних технік забезпечує вирішення наступних **стандартних проблем** масштабованих і **ефективних серверів додатків з використанням Java**:

- організація віддалених викликів між об'єктами, що працюють під управлінням різних віртуальних машин Java;
- управління потоками на стороні сервера;
- управління циклом життя серверних об'єктів (створення, взаємодія з користувачем, знищення);
- оптимізація використання ресурсів (час процесора, пам'ять, мережеві ресурси);
- створення схеми взаємодії контейнерів і операційних середовищ;
- створення схеми взаємодії контейнерів і клієнтів, включаючи універсальні засоби створення розробки компонентів і включення їх до складу контейнерів;
- створення засобів адміністрування та забезпечення їх взаємодії з існуючими системами;
- створення універсальної системи пошуку клієнтом необхідних серверних компонентів;
- забезпечення універсальної схеми управління транзакціями;
- забезпечення необхідних прав доступу до серверних компонентів;
- забезпечення універсального взаємодії з СУБД.

Технологія Enterprise JavaBeans визначає набір універсальних і призначених для багаторазового використання компонентів, які називаються **Enterprise beans (компоненти EJB).** При створенні розподіленої системи, її бізнес-логіка буде реалізована в цих компонентах.

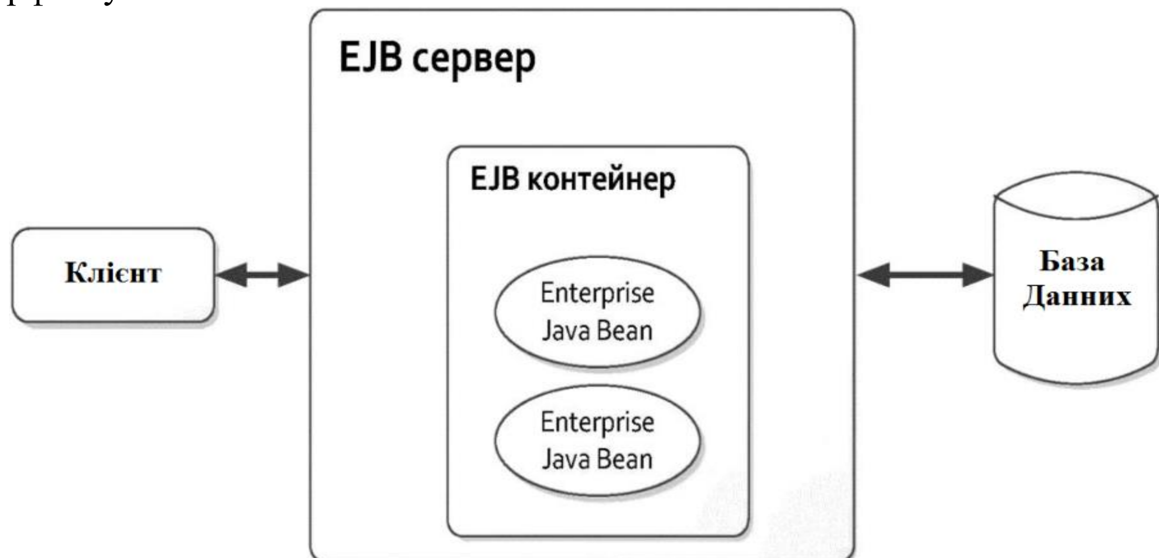
Кожен **компонент EJB складається з**:

➢ ***віддаленого інтерфейсу*** (remote-інтерфейс) - визначає бізнес-методи, які може викликати клієнт EJB;

➢ ***власного інтерфейсу*** (home-інтерфейс) - надає методи

- create для створення нових екземплярів компонентів EJB
- пошуку (finder) для знаходження екземплярів компонентів EJB
- remove для видалення екземплярів EJB.

➢ ***реалізації EJB-компонента*** - визначає бізнес-методи, оголошені в віддаленому інтерфейсі, і методи створення, видалення та пошуку власного інтерфейсу.



Після завершення розробки, набори компонентів EJB поміщаються в спеціальні файли (архіви, jar), по одному або більше компоненту, разом зі спеціальними *параметрами розгортання*. Потім вони встановлюються в спеціальній операційному середовищі, в якій запускається ***контейнер EJB***.

Клієнт здійснює пошук компонентів в контейнері за допомогою *home-інтерфейсу* відповідного компонента. Після того, як компонент створений і / або знайдений, клієнт виконує звернення до його методам за допомогою *remote-інтерфейсу*.

Контейнери EJB виконуються під управлінням ***сервера EJB***, який є сполучною ланкою між контейнерами та операційної середовищем. Сервер EJB забезпечує доступ контейнерів EJB до системних сервісів, таких як управління доступом до баз даних або моніторинг транзакцій. Всі ***екземпляри компонентів EJB*** виконуються під управлінням контейнера EJB, який надає системні сервіси розміщеним в ньому компонентів і управляє їх життєвим циклом.

У загальному **випадку контейнер призначений для вирішення наступних завдань:**

➢ Забезпечення безпеки. Дескриптор розгортання (deployment descriptor) визначає права доступу клієнтів до бізнес-методів компонентів. Забезпечення захисту даних забезпечується за рахунок надання доступу тільки для авторизованих клієнтів і тільки до дозволених методів.

- Забезпечення віддалених викликів. Контейнер бере на себе всі низькорівневі питання забезпечення взаємодії та організації віддалених викликів, повністю приховуючи всі деталі процесу, як від розробника компонентів, так і від клієнтів. Це дозволяє виробляти розробку компонентів точно так же, як якщо б система працювала в локальній конфігурації, тобто без використання віддалених викликів.
- Управління життєвим циклом. Клієнт створює і знищує екземпляри компонентів, однак контейнер для оптимізації ресурсів і підвищення продуктивності системи може самостійно виконувати різні дії, наприклад, активізацію і деактивацію цих компонентів, створення їх пулів і т.д.
- Управління транзакціями. Всі параметри, необхідні для управління транзакціями, поміщаються в дескриптор поставки. Всі питання щодо забезпечення управління розподіленими транзакціями в гетерогенних середовищах і взаємодії з декількома базами даних бере на себе контейнер EJB. Контейнер забезпечує захист даних і гарантує успішне підтвердження внесених змін; в іншому випадку транзакція відкочується.

Типи компонентів EJB

Існують **три типи компонентів EJB:**

- сесійні (Session Beans);
- сутнісні (Entity Beans);
- керовані повідомленнями (Message Driven Beans).

Сесійна компонент являє собою об'єкт, створений для обслуговування запитів *одного* клієнта. У відповідь на віддалений запит клієнта контейнер створює екземпляр такого компонента. Сесійна компонент завжди зіставлений з одним клієнтом, і його можна розглядати як «представника» клієнта на стороні EJB-сервера.

Сесійні компоненти є тимчасовими об'єктами. Зазвичай сесійну компонент існує, поки що створив його клієнт підтримує з ним сеанс зв'язку. Після завершення зв'язку з клієнтом компонент вже ніяк не пов'язана із з ним.

Сесійні компоненти бувають трьох типів:

- stateless (без стану)
- stateful (з підтримкою поточного стану сесії)
- singleton (один об'єкт на всі додаток; починаючи з версії 3.1)

Сутнісні компоненти являють собою об'єктне представлення даних з бази даних. Ключовою відмінністю сутнісного компонента від сесійної є те, що кілька клієнтів можуть одночасно звертатися до одного екземпляра сутнісного компонента. Сутнісні компоненти змінюють стан зіставлених з ними баз даних в контексті транзакцій.

Стан компонентів-сутностей в загальному випадку потрібно зберігати, і живуть вони стільки, скільки існують в базі даних ті дані, які вони представляють, а не стільки, скільки існує клієнтський або серверний процес. Зупинка або крах контейнера EJB не приводить до знищення містяться в ньому сутнісних компонентів.

Керованими повідомленнями компоненти характеризуються тим, що їх логіка є реакцією на події в системі.

Складові частини EJB-компонента

EJB-компонент фізично складається з декількох частин, включаючи сам компонент, реалізацію деяких інтерфейсів і інформаційний файл. Все це збирається разом в спеціальний **jar-файл - модуль розгортання**.

- ***Enterprise Bean*** є Java-класом, розробленим постачальником Enterprise Bean. Він реалізує інтерфейс Enterprise Bean і забезпечує реалізацію бізнес-методів, які виконує компонент. Клас не реалізує ніяких методів авторизації, багатопоточності або підтримки транзакцій.
- ***Домашній інтерфейс***. Кожен створюваний Enterprise Bean повинен мати асоційований домашній інтерфейс. Домашній інтерфейс застосовується як фабрика для компонента EJB. Клієнт використовує домашній інтерфейс для знаходження екземпляра компонента EJB або створення нового екземпляра компонента EJB.
- ***Віддалений інтерфейс*** є Java-інтерфейсом, який відображає через рефлексію ті методи Enterprise Bean, які необхідно показувати зовнішньому світу. Віддалений інтерфейс грає ту ж роль, що і IDL-інтерфейс в CORBA, і забезпечує можливість звернення клієнта до компоненту.
- ***Описувач розгортання*** є XML-файлом, який містить інформацію щодо компонента EJB. Використання XML дозволяє установникові легко міняти атрибути компонента. Конфігураційні атрибути, визначені в описувач розгортання, включають:

- імена родинної злагоди і віддаленого інтерфейсу;
- ім'я JNDI для публікації домашнього інтерфейсу компонента;
- транзакційні атрибути для кожного методу компонента;
- контрольний список доступу для авторизації.

- ***EJB-Jar-файл*** - це звичайний java-jar-файл, який містить компонент (компоненти) EJB, домашній і віддалений інтерфейси, а також описувач розгортання.

Інфраструктура Enterprise JavaBean

Інфраструктура EJB забезпечує віддалене взаємодія об'єктів, управління транзакціями і безпеку додатки. Специфікація EJB обумовлює вимоги до елементів інфраструктури і визначає Java API, проте вона не стосується питань вибору платформ, протоколів та інших аспектів, пов'язаних з реалізацією.

У загальному випадку необхідно гарантувати збереження стану компонентів в контейнерах. Інфраструктура EJB зобов'язана надати можливості для інтеграції додатка з існуючими системами і додатками. Всі аспекти взаємодії клієнтів з серверними компонентами повинні відбуватися в контексті транзакцій, управління якими покладається на інфраструктуру EJB.

Специфікація Enterprise JavaBeans - це суттєвий крок до стандартизації моделі розподілених об'єктів в Java.

Тема 6. Сервіс-орієнтована архітектура

Сервіс-орієнтована архітектура (*СОА, Service-Oriented Architecture - SOA*) - це парадигма організації та використання розподілених можливостей, які можуть належати різним власникам.

СОА - це модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов'язаних замінних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами.

Типові складові СОА:

► сервісні компоненти (сервіси);

► контракти сервісів (інтерфейси);

► з'єднувачі сервісів (транспорт);

► механізми виявлення сервісів (регістри).

Сервісні компоненти (або сервіси) - це відкриті, самовизначаються програмні компоненти, що надають певну функціональність. Залежно від обсягу наданих послуг виділяють сервіси:

► *маломодульні (ММС),* що надають елементарний обсяг функціонального навантаження і забезпечують високий ступінь повторного використання; іноді для отримання бажаного результату, необхідно забезпечити координовану роботу декількох ММС;

► *багатомодульних (БМС),* що дозволяють забезпечити хорошу інкапсуляцію функціональності; повторне використання БМС ускладнене в зв'язку з їх вузькою спеціалізацією.

Контракт сервісу (або інтерфейс) забезпечує опис можливостей і якості послуг, що надаються конкретним сервісом. В інтерфейсі визначається формат повідомлень, що використовується для обміну інформацією, а також вхідні і вихідні параметри методів, підтримуваних сервісним компонентом. Від вибору мови і способу опису інтерфейсу залежать можливості програмної сумісності різних реалізацій СОА.

З'єднувач сервісів (транспортні) забезпечує обмін інформацією між окремими сервісними компонентами. Поряд з відкритими стандартами опису інтерфейсів, використання гнучких транспортних протоколів для обміну інформацією між сервісними компонентами дозволяє підвищити програмну сумісність сервіс орієнтованої системи.

Механізми пошуку послуг (або регістри сервісів) використовуються для пошуку сервісних компонентів, що забезпечують необхідну функціональність. Виділяють 2 основні категорії систем виявлення:

► статичні системи пошуку послуг (наприклад, UDDI) орієнтовані на зберігання інформації про сервіси в рідко змінюються системах;

► динамічні системи пошуку послуг орієнтовані на системи, в яких допустимо часта поява і зникнення сервісних компонентів.

Зв'язаність програмних систем

Пов'язаністю називають ступінь знання і залежно одного об'єкта від внутрішнього змісту іншого.

Програмні системи можна розділити на 2 типу:

► *Сильнозв'язані системи (Strong coupling)*: залежний клас містить посилання безпосередньо на певний клас, який надає деякі можливості (приклади: Java RMI, NET Remoting);

► *слабкозв'язаної системи (Loose coupling):* залежний клас містить посилання на інтерфейс який може бути реалізований одним або декількома конкретними класами (приклад: СОА).

Основна мета використання концепції слабкопов'язаних програмних систем - це зменшення кількості залежностей між компонентами. При зменшенні кількості зв'язків, зменшується обсяг можливих наслідків, що виникають у зв'язку зі збоями або системними змінами.

Традиційний підхід розробки розподілених додатків, підтримуваний технологіями розподілених об'єктів, ґрунтується на тісному зв'язку між усіма програмними компонентами. Слабо зв'язаність програмних компонентів, підтримувана технологією веб-сервісів, дозволяє значно спростити координацію розподілених систем і їх реконфігурацію.

Порівняння сильнозв'язаних і слабозвя'заних систем

	Сильнозв'язані системи	**Слабкозв'язані системи**
Фізичне з'єднання	Точка-точка	Через посередника
Стиль взаємодії	Синхронне	Асинхронне
Модель даних	Загальні складні типи	Прості типи
Зв'язані	Статичні	Динамічні
Платформа	Сильна залежність від базової платформи	Незалежність від платформи
Розкручування	Одночасне	Поступове

Принципи побудови СОА

Інтероперабельність - здатність двох або більше інформаційних систем (або їх компонентів) до взаємодії, з метою вирішення певної задачі і отримання певної інформації.

Це визначення об'єднує в собі два поняття:

► *технічна інтероперабельність* - сумісність систем на технічному рівні, включаючи протоколи передачі даних і формати їх подання;

► *семантична інтероперабельність* - властивість інформаційних систем, що забезпечує взаємну вживаної отриманої інформації на основі загального розуміння системами її значення.

Прикладом семантичної інтероперабельність програмних систем може служити процес передачі певних даних в текстовому вигляді по каналах зв'язку. Наприклад, якщо системи семантично НЕ інтероперабельність, то одержувач не зможе однозначно інтерпретувати отриману рядок «1.23»: це може бути число з плаваючою комою, записане в десятковій або шістнадцятковій системі числення, а може бути дата, яку треба інтерпретувати «23 січня». СОА не наказує жорсткої вертикальної («зверху вниз») методології проектування, впровадження або

управління IT-інфраструктурою. СОА обмежується низкою принципів, які характеризують кожен з цих процесів; тому її іноді називають не архітектурою, а архітектурним стилем. Основні принципи побудови СОА:

Розподілене проектування. Рішення щодо внутрішніх особливостей інформаційних систем приймаються різними групами людей, що мають власні організаційні, політичні та економічні мотиви.

Сталість змін. Окремі ділянки архітектури можуть зазнавати зміни в будь-який момент часу.

Послідовне вдосконалення. Локальне поліпшення компонентів архітектури повинно призводити до вдосконалення всієї архітектури в цілому - до зростання сумарної корисності компонентів того ж 8 рівня, що і змінний, так само як і компонентів нижчого і вищого рівня. Наприклад, відомий веб-сервіс Google Translate постійно зазнає змін. Спочатку, він забезпечував тільки веб-інтерфейс для перекладу і обмежений набір мов. Поступово збільшувалися функціональні можливості сервісу: розширювався набір мов, з'явилася можливість голосового відтворення перекладу, при перекладі окремого слова почали видаватися словникові статті з декількома результатами перекладу і т. П. При цьому API (Application Programming Interface) і інтерфейс змінювався незначно.

Рекурсивних. Однотипні рішення мають місце на різних рівнях архітектури.

Підхід СОА

З точки зору інформаційних технологій, **логіка підприємства** може бути розділена на: бізнес-логіку (БЛ) і логіку додатка (ЛП). **Бізнес-логіка** - документальна реалізація бізнестребованій, які виходять з проблемної області, в якій працює підприємство. БЛ, як правило, структурована в процесах, які виражають ці вимоги, а також обмеження і залежності від зовнішніх впливів. **Логіка програми** - це реалізація БЛ, організована на основі різних технологічних рішень.

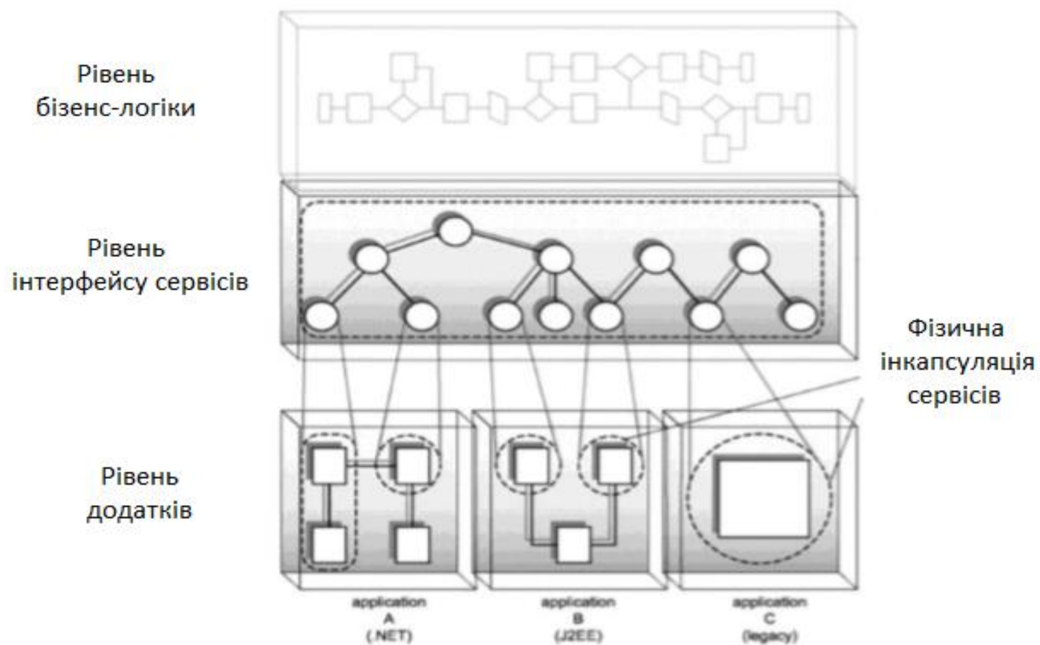


Рис. 15 – Рівні логіки підприємства

ЛП висловлює процеси БЛ за допомогою придбаних або спеціально розроблених програмних систем в умовах обмежених технічних можливостей і залежностей від постачальника рішення.

Процес перетворення бізнес-логіки в логіку додатків і реалізація сервісів на основі даних вимог є процесом створення сервісно-орієнтованої інфраструктури для задач підприємства. Не існує «догматичних» принципів побудови СОА, але при реалізації власної інфраструктури бажано дотримуватися **деяких основних підходів.**

1. Сервіси повинні підтримувати повторне використання. СОА системи повинні підтримувати повторне використання всіх сервісів, незалежно від сьогохвилинних вимог до їх функціональних особливостей. Якщо при розробці системи постаратися максимально врахувати цю вимогу, то підвищуються шанси значно спростити процес вирішення завдань, які неодмінно з'являться в майбутньому, при розвитку системи. Також спочатку орієнтований на повторне використання сервіс дозволяє уникнути розробки «обгортки», яка б підбудовувала старий сервіс для вирішення нових завдань. Так як сервіс - це не що інше, як просто набір зв'язкових операцій, логіка кожної індивідуальної операції, що надається сервісом, повинна підтримувати повторне використання.

2. Сервіси повинні забезпечувати формальний контракт використання. Контракт сервісу надає наступну інформацію:

► кінцеву точку (service endpoint): адреса, за якою можна звернутися до даного сервісу;

► всі операції, що надаються сервісом;

► всі повідомлення, підтримувані кожною операцією;

► правила і характеристики сервісу і його операцій.

3. Сервіси повинні бути слабо зв'язаної. Ніхто не може передбачити, в який бік буде розвиватися ІТ-інфраструктура. Рішення можуть розвиватися, взаємодіяти, заміняти один одного. У зв'язку з цим основним завданням є збереження цілісності системи в рамках такого розвитку, незалежно від змін, що відбуваються. Система сервісів є слабо зв'язаної, якщо сервіс може здобувати знання про іншому сервісі, залишаючись незалежним від внутрішньої реалізації логіки даного сервісу. Це досягається за допомогою використання контрактів сервісів.

Малозв'язаність програмних компонентів, що лежить в основі СОА, дозволяє значно спростити координацію розподілених систем і їх реконфігурацію.

4. Сервіси повинні абстрагувати внутрішню логіку. Кожен сервіс повинен діяти як «чорний ящик», що приховує свої деталі від навколишнього світу. Немає чіткого визначення, який обсяг логіки повинен поміщатися в окремому сервісі. Взаємодія на рівні інтерфейсів є однією з вимог для забезпечення слабкою пов'язаності.

5. Сервіси повинні бути сумісні. Сервіс може як самостійно реалізовувати логіку, так і застосовувати інші сервіси для її реалізації. Сервіси повинні бути спроектовані таким чином, щоб підтримувати можливість їх використання в якості елементів іншого сервісу. Принцип сумісності не залежить від того, чи використовує сервіс для виконання своєї роботи інші сервіси. Сумісність - це, по суті, просто інша форма повторного використання, і тому операції повинні бути стандартними, а для найбільшою сумісності повинні володіти необхідним рівнем деталізації.

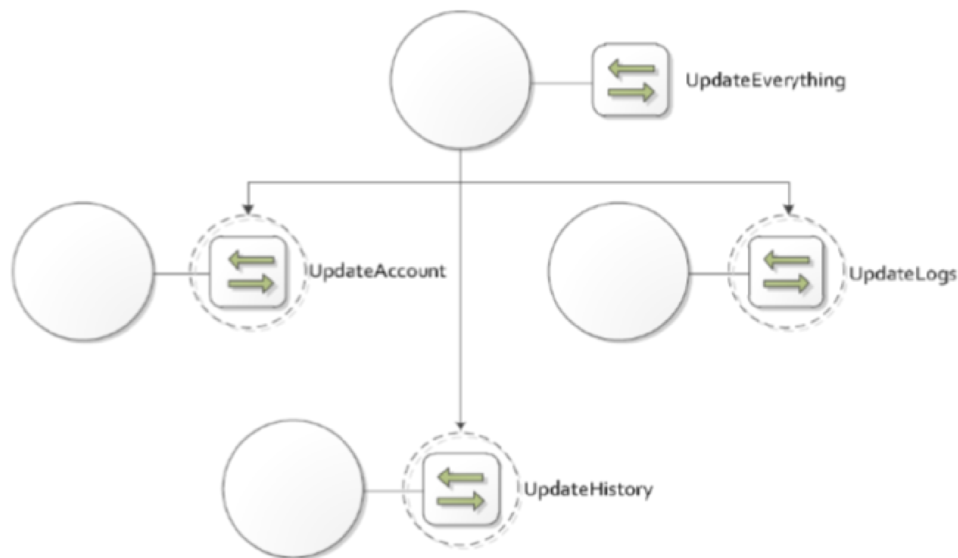


Рис 16 – Сервіси, використані в якості елементів іншого сервісу

6. Сервіси повинні бути автономними. Властивість автономності вимагає, щоб область бізнес-логіки і ресурсів, використовуваних сервісом були обмежені явними межами. Це дозволяє сервісу самому керувати всіма своїми процесами. Також це усуває залежність від інших сервісів, що звільняє сервіс від зв'язків, які

можуть перешкоджати застосуванням і розвитку. Питання автономності - найбільш важливий аргумент при розподілі бізнес-логіки на окремі сервіси.

Автономність не обов'язково надає сервісу виключне право власності на бізнес-логіку, яку він інкапсулює. Є 2 типу автономності:

► Автономність на рівні сервісу: межі відповідальності сервісів відокремлені, але вони можуть використовувати загальні ресурси.

► Чистий автономність: бізнес-логіка і ресурси знаходяться під повним контролем сервісу. Як правило, такий вид автономності використовується, коли для реалізації сервісу бізнес-логіка створюється з нуля.

7. Сервіси не повинні використовувати інформацію про стан. Сервіси повинні зводити до мінімуму обсяг інформації про стан, і час, протягом якого вони нею володіють. Інформація про стан - це певні дані, що характеризують поточну діяльність. Наприклад, поки сервіс обробляє повідомлення, він тимчасово залежить від стану (stateful). Якщо сервіс несе відповідальність за збереження стану протягом більш тривалого часу, його здатність залишатися доступним для інших клієнтів буде утруднена.

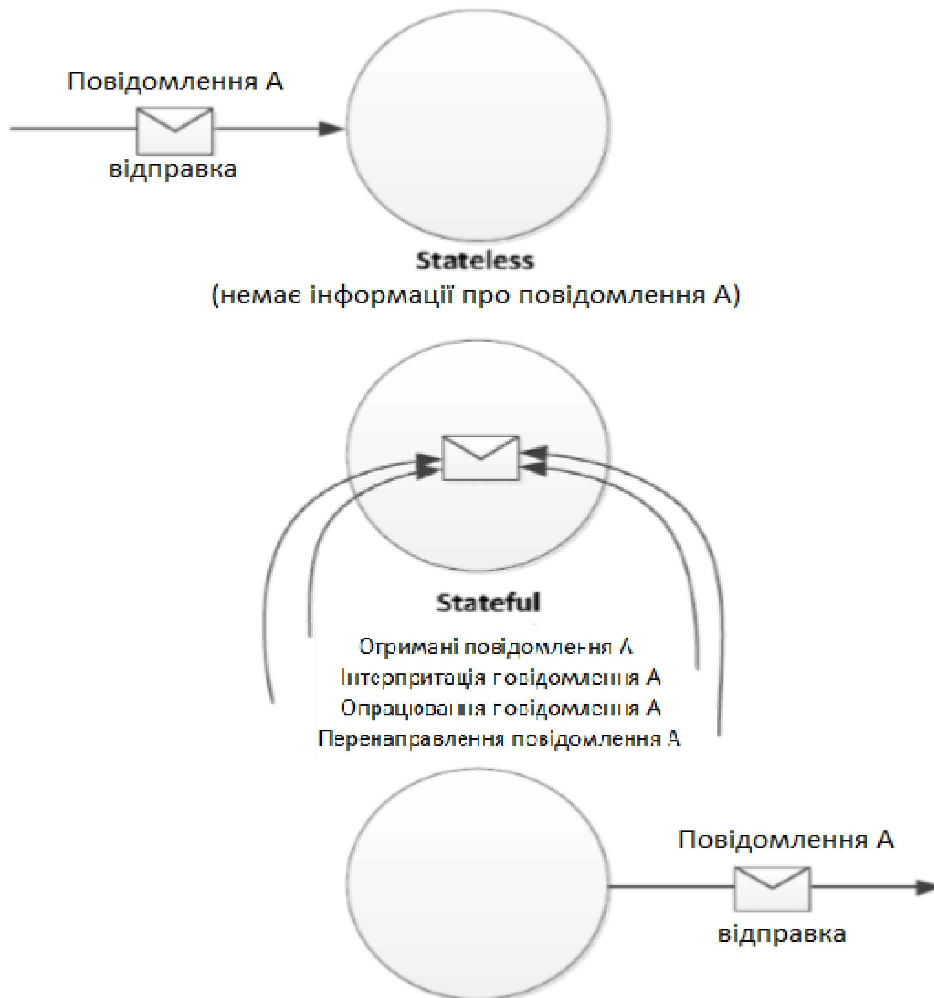


Рис.17 Часова залежність від стану в залежності від оброки повідомлення

Незалежність від стану (statelessness) дозволяє підвищити можливості масштабованості і повторного використання сервісів. Операції сервісу повинні

бути розроблені з урахуванням міркувань обробки інформації без даних про стан. Для підтримки незалежності від стану в СОА використовуються сообщеніядокументи. Чим складніше повідомлення, тим більш незалежним і самодостатнім воно залишається.

8. Сервіси повинні підтримувати виявлення. Виявлення сервісів дозволяє уникнути випадкового створення надлишкового сервісу, що забезпечує надлишкову логіку. Метадані сервісу повинні детально описати не тільки загальну мету сервісу, але і функціональність, реалізовану його операціями. На рівні СОА, виявлення характеризує здатність архітектури забезпечити механізми пошуку, такі як реєстр або каталог. На рівні сервісу, принцип виявлення відноситься до процесу проектування окремого сервісу, так щоб даний сервіс настільки подавався виявлення, наскільки це можливо.

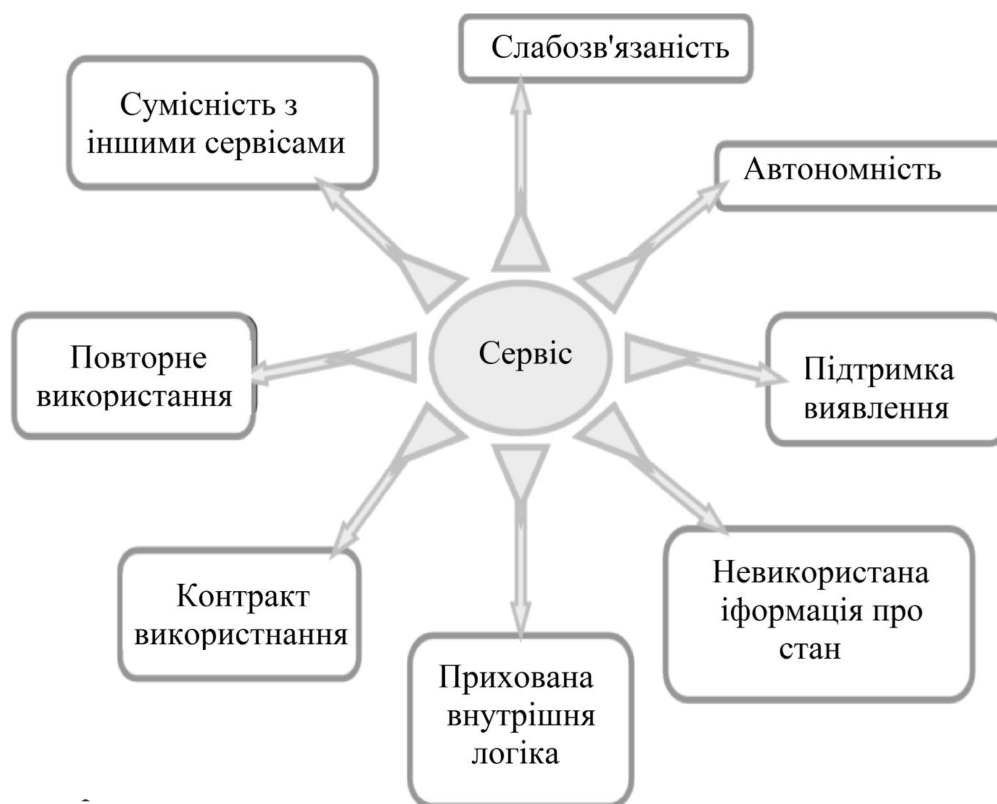


Рис.18 – Властивості сервісів в відповідності до СОА

Тема 7. Веб-сервіси

Веб-сервіси (Web-службы) - це програмні компоненти, з допомогою яких можна створювати незалежні масштабовані слабозв`язні додатки

Веб-сервіс - це ідентифікована веб-адресом програмна система з стандартизованими інтерфейсами. Веб-служба є одиницею модульності при використанні сервіс-орієнтованої архітектури додатку.

Специфікація визначає три основні стандарти, що використовуються для підтримки представлення, пошуку та обміну інформацією між веб-сервісами - це

WSDL, UDDI та SOAP, які створюють так званий ***«Трикутник СОА»:***

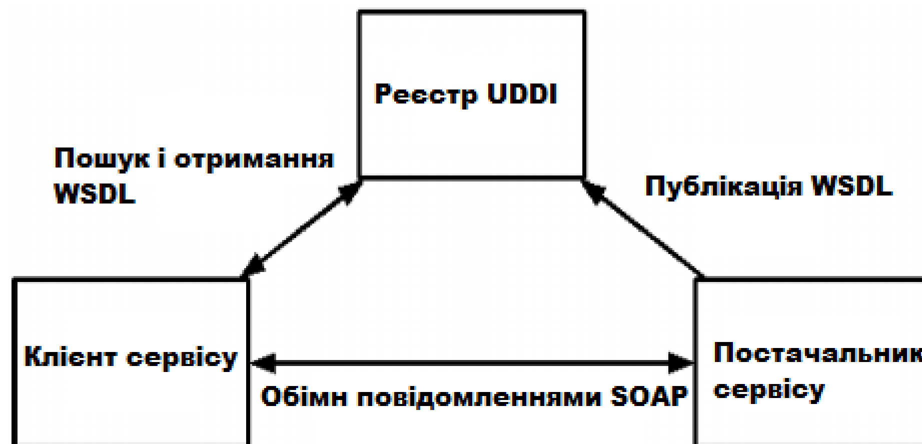


Рис. 19 – Процес взаємодії між клієнтом і постачальником веб-сервіса

► ***замовник*** (клієнт сервісу, service requestor);
► ***виконавець*** (постачальник сервісу, service provider);
► ***каталог*** (реєстр, service broker).

Коли служба розроблена, виконавець реєструє її в каталозі, де її можуть знайти потенційні замовники. Замовник, знайшовши в каталозі відповідну службу, імпортує звідти її **WSDL**-специфікацію і розробляє відповідно до неї своє програмне забезпечення. WSDL описує формат запитів і відповідей, якими обмінюються замовник і виконавець в процесі роботи.

Для забезпечення взаємодії використовуються такі нормативні документи:

- ***XML (eXtensible Markup Language):*** розширювана мова розмітки, призначена для зберігання і передачі структурованих даних;
- ***SOAP (раніше Simple Object Access Protocol):*** спочатку (до версії 1.2) «Простий Протокол Доступу до Об'єктам»; нині протокол обміну повідомленнями на базі XML;
- ***WSDL (Web Services Description Language)***: мова опису зовнішніх інтерфейсів веб-служби на базі XML;
- ***UDDI (Universal Discovery, Description and Integration):*** універсальний інтерфейс розпізнавання, опису та інтеграції. Каталог веб-служб і відомостей про компанії, що надають веб-служби в загальне користування або конкретним компаніям.

XML - розширювана мова розмітки. Специфікація XML описує XML-документи і частково описує поведінку XML-процесорів (Програм, які читають XML-документи і забезпечують доступ до їх вмісту). XML розроблявся як мова з простим формальним синтаксисом, зручним для створення і обробки документів програмами і одночасно зручний для читання і створення документів людиною, яка працює в Інтернеті. Мова називається розширюваним, оскільки він не фіксує розмітку, яка використовується в документах: розробник може створювати розмітку

відповідно до потреб конкретної області, будучи обмеженим лише синтаксичними правилами мови. **SOAP** призначений для організації взаємодії віддалених систем за допомогою асинхронного обміну XML-відформатованими документами,

що складаються з трьох частин: конверта (обгортки), заголовка і тіла. SOAP формує базовий шар стеку протоколів веб-сервісів, забезпечуючи інфраструктуру обміну повідомленнями між ними.

WSDL описує сервіси у вигляді якихось абстрактних ресурсів, здатних приймати на вхід документи певних типів і ініціювати відправлення документів інших типів. WSDL використовується для опису веб-сервісів і для визначення їх розташування. WSDL написаний на XML і є XML-документом.

WSDL визначає сервіс з двох точок зору:

► абстрактної: сервіс задається в термінах, що посилаються і приймаються йому в повідомленнях, які описуються засобами XML Schema у вигляді, незалежному від конкретного транспортного протоколу;

► конкретної: визначаються прив'язки до транспортних форматів і точок фізичного розміщення.

Група специфікацій WSDL 2.0 складається з трьох основних документів:

WSDL Part 1: Core language («Основна мова»),

WSDL Part 2: Message exchange patterns («Шаблони обміну повідомленнями»),

WSDL Part 2: Bindings («Прив'язки»).

UDDI - це стандарт внутрішнього пристрою і зовнішнього інтерфейсу бази даних (сховища), що зберігає опис сервісів. Весь опис в БД зберігається у вигляді XML-записів.

Остання версія UDDI (3.01) забезпечує:

► реплікацію репозиторіїв зі складними моделями їх підпорядкованості один одному,

► побудову сховища з декількох вузлів (і реплікацію даних між ними),

► глобальну унікальність записів і ключів,

► API публікації описів і підписки на зміни,

► засоби забезпечення цілісності даних, інтернаціоналізації

записів, шифрування вмісту.

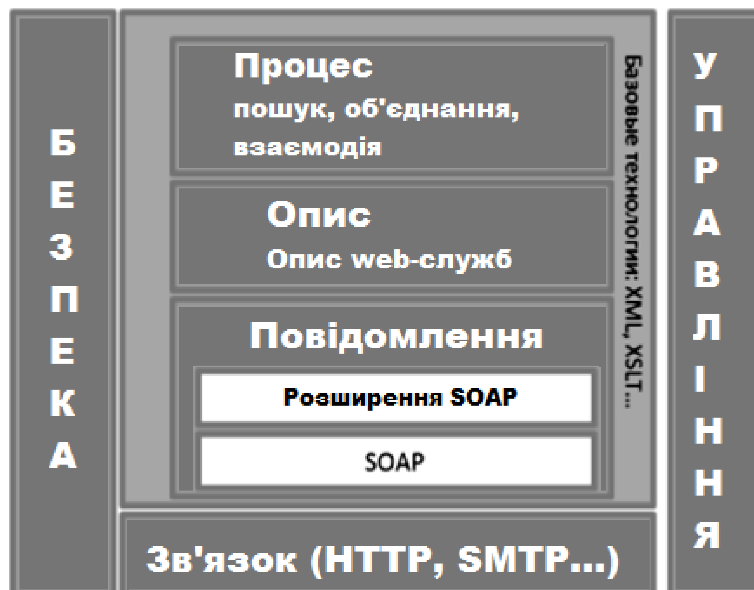


Рис. 19 – Стек протоколів веб-сервісів

Архітектуру веб-сервісів становить маса протоколів і специфікацій. Їх можна розбити на

чотири частини:

► процес,
► опис,
► повідомлення,
► зв'язок,

утворюють стек протоколів, в якому кожен верхній рівень опирається на нижній.

Стандарт WSDL

Розглянемо приклад найпростішого веб-сервісу. Даний сервіс буде забезпечувати піднесення до квадрату переданого числа. На мові Java цей сервіс можна було б описати у вигляді наступного вихідного коду.

Відповідно, нам необхідно сформувати (вручну або автоматично) WSDL-документ і помістити його за адресою http://supercomputer.susu.ru/MyMath.wsdl.

Стандарт WSDL забезпечує опис веб-сервісу у вигляді повідомлень, які може відправити або ж прийняти веб-сервіс. Також, він відповідає за методи зв'язування даних повідомлень з базовим середовищем передачі даних. В зв'язку з цим виділяють наступні **елементи WSDL-документа:**

► Блок types - типи даних, що використовуються веб-сервісом;
► Блок message - повідомлення, що використовуються веб-сервісом;
► Блок portType - методи, які зазвичай пропонуються сервісом;
► Блок binding - протоколи зв'язку, які використовуються веб-сервісом.

Елемент **<portType>** є найбільш важливим елементом WSDL. Він визначає сам веб-сервіс, операції, що ним надаються, і використовувані повідомлення. Цей елемент можна порівняти з бібліотекою функцій, в якій вказані вхідні параметри і результати роботи функції.

Розглянемо, яким чином буде описуватися наш сервіс **"MyMath**".

Інтерфейс сервісу MyMath складається з однієї операції squared, яка повинна виконуватися з одним параметром in0. Ця операція складається з двох повідомлень:

► вхідного (wsdl: input message = "impl: squaredRequest"), яке має передати користувач веб-сервісу в наш сервіс для того, щоб запустити операцію піднесення до квадрату;

► вихідного (wsdl: output message = "impl: squaredResponse"), яке буде повернуто користувачеві після того, як наша операція піднесення до квадрату успішно завершиться.

```
<wsdl:portType name="MyMath">
    <wsdl:operation name="squared" parameterOrder="in0">
        <wsdl:input message="impl:squaredRequest"
      name="squaredRequest"/>
        <wsdl:output message= "impl:squaredResponse"
      name="squaredResponse"/>
    </wsdl:operation>
</wsdl:portType>
```

Рис. 21 – Приклад опису блоку portType

Елемент **<message>** визначає елементи даних операції. Кожне повідомлення може містити одну або кілька частин. Ці частини можна порівняти з параметрами викликаються функцій в традиційних мовах програмування.

```
<wsdl:message name="squaredRequest">

        <wsdl:part name="in0" type="xsd:int"/>

</wsdl:message>

<wsdl:message name="squaredResponse">

        <wsdl:part name="squaredReturn" type="xsd:int"/>
```

Рис. 22 - Приклад опису блоку message

У <message> наводиться опис всіх частин повідомлень "squaredRequest" і "squaredRes-ponse", інтерфейси яких описані в блоці <portType>. Кожна частина повідомлення - це параметр виклику методу сервісу. Для проведення нашої операції, користувач повинен передати 1 вхідний параметр "in0" у вигляді цілого числа, що визначається атрибутом type = "xsd: int". Результатом операції також буде ціле число, що описано в блоці <wsdl: part name = "squaredReturn" type = "xsd: int" />.

Елемент **<binding>** визначає формат повідомлення та деталі протоколу для кожного порту. Він відповідає за те, яким чином елементи абстрактного інтерфейсу в блоці <portType> перетворюються в масиви інформації в форматі протоколів взаємодії, наприклад SOAP.

Основною частиною блоку <binding> є елемент *<soap: binding>*, що визначає конкретний протокол передачі даних. Атрибут style визначає тип запиту і може мати два значення: "rpc" (Remote Procedure Call - віддалений виклик процедур) і "document". Також, в блоці <Soap: binding> оголошується *transport* - визначальний протокол, на основі якого буде проводитися взаємодія (зазвичай HTTP).

```
<wsdl:binding name="MyMathSoapBinding" type="impl:MyMath">
    <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.or
    g/soap/http"/>
    <wsdl:operation name="squared">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="squaredRequest">
            <wsdlsoap:body encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace" use="encoded"/>
        </wsdl:input>
        <wsdl:output name="squaredResponse">
            <wsdlsoap:body encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace" use="encoded"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

Рис. 23 – Приклад опису блоку binding

Усередині *<soap: operation>* знаходиться елемент, який описує значення поля soapAction HTTP-запиту.

Елементи *input* і *output* визначають як будуть декодуватись вхідні і вихідні повідомлення цієї операції.

В блоках <**port**> и <**service**>відбувається визначення, де знаходитсясервіс:

- *port* – описує розташування і спосіб доступу до кінцевої точки,
- *service* – іменована колекція портів.

Рис. 24 – Приклад блоку service

```
<wsdl:service name="MyMathService">
        <wsdl:port binding="impl:MyMathSoapBinding" name="MyMath">
                <wsdlsoap:address location="http://supercomputer.susu.ru/MyMath"/>
        </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Як можна бачити з прикладу, в блоці <port> не відбувається безпосереднього опису методів взаємодії з веб-сервісом. Вони описуються раніше, в блоці <binding>, а в блоці <port> тільки дається посилання на описаний метод зв'язку binding.

Стандарт SOAP

Стандарт SOAP забезпечує взаємодію між веб-сервісами.

Повідомлення SOAP - це одностороння передача інформації від джерела до приймача між вузлами SOAP.

Повідомлення SOAP є основним будівельним блоком, який забезпечує можливості більш складних шаблонів взаємодії: запит / відповідь, режим "діалогу" і т.п.

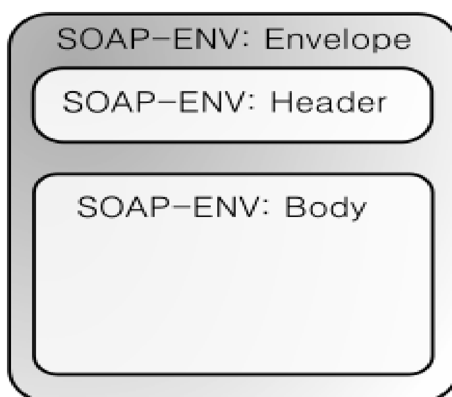


Рис. 25 – Структура SOAP-повідомлення

Повідомлення SOAP складається з 3-х частин: *конверта*, що містить в собі *заголовок* і *тіло* повідомлення. У *тілі* міститься XML-блок з інформацією, яка повинна бути доставлена кінцевому адресату. Тема - необов'язковий елемент, за допомогою якого можна передавати дані, які не є основною робочим навантаженням (напр. інф. для обробки повідомлення). SOAP-повідомлення здатне слідувати за маршрутом, який містить кілька вузлів, кожен з яких може якось його обробляти. Статус цих змін відбивається в блоках заголовка повідомлення.

У **заголовку** SOAP-повідомлення можна ввести нові елементи повідомлення, не передбачені стандартом SOAP. Ці елементи відіграють утилітарну роль по відношенню до основного повідомленням, що міститься в тілі SOAP (напр.: номер транзакції, в рамках якої прийшло повідомлення; інформація для авторизації користувача і ін.).

Для елементів заголовка можна вказати значення наступних атрибутів:

➢ *mustUnderstand* - якщо значення дорівнює 1, одержувач зобов'язаний обробляти цей елемент заголовка. Якщо він це не вміє, то зобов'язаний відкинути дане повідомлення;

➢ *actor* – вказує назву конкретного додатка-одержувача якщо SOAP-повідомлення про-ходить ланцюжок додатків при обробці.

```
<soap:Header>
    <trans:Transaction xmlns:trans="http://www.host.com/namespaces/space/"
    soap:mustUnderstand="1">
    12
    </trans:Transaction>
</soap:Header>
```

Рис. 26 – Приклад заголовку SOAP-повідомлення

Запит

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <getProductDetails
                xmlns="http://warehouse.example.com/ws">
        <productID>12345</productID>
        </getProductDetails>
    </soap:Body>
</soap:Envelope>
```

Відповідь

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

Рис. 27 – Приклад запиту і відповіді за допомогою SOAP-повідомлення

У тілі SOAP-повідомлення проводиться передача повідомлення по формату, визначеному в блоках <portType> і <message> WSDL-документа. Ім'я основного блоку, що знаходиться в тілі SOAP-повідомлення відповідає імені повідомлення, яке визначено в інтерфейсі веб-сервісу.

Оскільки стандарт SOAP забезпечує односторонню передачу повідомлень, для того щоб отримати відповідь від сервера, якому було передано SOAP

повідомлення, може знадобитися щоб він ініціював процес передачі повідомлення і встановив з'єднання з клієнтом. Це може викликати значні труднощі в сучасних умовах організації зв'язку в мережі Інтернет, тому що більшість клієнтів не мають виділених статичних IP-адрес.

Дана проблема вирішена в рамках стандартного зв'язування протоколу SOAP і протоколу HTTP, що реалізує патерн поведінки **«запит-відповідь»** (анг. *SOAP HTTP Binding*).

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap+xml;
charset=utf-8
Content-Length: nnn

<soap:Envelope...
```

```
HTTP/1.1 200 OK
Content-Type: application/soap;
charset=utf-8
Content-Length: nnn

<soap:Envelope...
```

Заголовок «ContentType» для повідомлень HTTP-запиту і HTTP-відповіді вста-новлюється в значення text / xml (application / soap + xml в SOAP 1.2). HTTP-запит повинен використовувати POST (починаючи з SOAP 1.2 можна використовувати GET). HTTP-відповідь повинен вивикористовувати статусний код 200 якщо обробка SOAP-повідомлення пройшла нормально, або 500 якщо в тексті листа міститься помилка SOAP.

Тема 8. Основи хмарних обчислень

У попередніх лекціях ми розглянули дві ключові тенденції, що визначили появу концепції хмарних обчислень. Це консолідація та віртуалізація IT-інфраструктури. Третім ключовим компонентом чи третім китом Cloud Computing є поняття Software as a Service (SaaS).

Рис.28. Приклади застосування концепції SaaS

Перші ідеї про використання обчислень як публічної послуги були запропоновані ще у 1960-х відомим вченим у галузі інформаційних технологій, винахідником мови Lisp, професором MIT та Стенфордського університету Джоном Маккарті (John McCarthy). Реалізація першого реального проекту приписується компанії Salesforce.com, яка була заснована в 1999 році. Саме тоді і з'явилася перша пропозиція нового виду b2b продукту «Програмне забезпечення як сервіс» (Software as a Service, SaaS). Певний успіх Salesforce у цій галузі порушив інтерес у гігантів IT індустрії, які спішно повідомили про свої дослідження у галузі хмарних технологій. І ось уже перше бізнес-рішення під назвою «Amazon Web Services» було запущено в 2005 році компанією Amazon.com, яка з часів кризи дотком активно займалася модернізацією своїх датацентрів. Наступним свою технологію поступово ввела Google, розпочавши з 2006 року b2b пропозицію SaaS сервісів під назвою Google Apps. І нарешті свою пропозицію анонсувала компанія Microsoft, презентувавши її на конференції PDC 2008 під назвою «Azure Services Platform».

Рис.29. SaaS сервіси Google

Сам факт високої зацікавленості найбільших гравців ринку IT демонструє певний статус хмарних обчислень як тренд 2009-2010 років. Крім того, з релізом Microsoft Azure Service Platform багато експертів пов'язує новий виток розвитку веб-технологій та вихід усієї сфери хмарних обчислень на новий рівень.

Нагадаємо, що під хмарними обчисленнями ми розуміємо програмно-апаратне забезпечення, доступне користувачеві через Інтернет або локальну мережу у вигляді сервісу, що дозволяє використовувати зручний інтерфейс для віддаленого доступу до виділених ресурсів (обчислювальних ресурсів, програм та даних).

На даний момент більшість хмарних інфраструктур розгорнуто на серверах датацентрів, використовуючи технології віртуалізації, що фактично дозволяє будь-якому додатку використовувати обчислювальні потужності, абсолютно не

замислюючись про технологічні аспекти. Тоді можна розуміти "хмару" як єдиний доступ до обчислень з боку користувача.

Види хмарних обчислень

З поняттям хмарних обчислень часто пов'язують такі сервіс-що надають (Everything as a service) технології, як:

· « Інфраструктура як сервіс » (“Infrastructure as a Service” або “IaaS”)

· « Платформа як сервіс » (“Plaatform as a Service”, “PaaS”)

· " Програмне забезпечення як сервіс " ("Software as a Service" або "SaaS").

лянемо кожну з цих технологій докладніше.

Інфраструктура як сервіс (IaaS)

Недоліки та проблеми хмарних обчислень

Отже, відзначимо основні недоліки та труднощі використання cloud computing:

Постійне з'єднання із мережею. Cloud Computing завжди майже завжди потребує з'єднання з мережею (Інтернет). Якщо немає доступу до мережі - немає роботи, програм, документів. Багато "хмарних" програм вимагають хорошого Інтернет-з'єднання з великою пропускною здатністю. Відповідно програми можуть працювати повільніше, ніж на локальному комп'ютері. На думку провідних російських IT-компаній, основною перешкодою широкому розвитку хмар є відсутність широкосмугового доступу в Інтернет (ШПД) – насамперед у регіонах.

Безпека.

Безпека даних теоретично може бути під загрозою. Не всі дані можна довірити сторонньому провайдеру в інтернеті, тим більше не тільки для зберігання, але ще й для обробки. Все залежить від того, хто надає "хмарні" послуги. Якщо хтось надійно шифрує Ваші дані, постійно робить їх резервні копії, вже не один рік працює на ринку подібних послуг і має гарну репутацію, то загрози безпеці даних може ніколи не статися. У користувача «хмарних» бізнес-додатків можуть також виникнути і юридичні проблеми, наприклад, пов'язані з виконанням вимог захисту персональних даних.

Держава, на території якої розміщено датацентр, може отримати доступ до будь-якої інформації, яка в ній зберігається. Наприклад, за законами США, де є найбільша кількість датацентрів, у цьому випадку компанія-провайдер навіть не має права розголошувати факт передачі конфіденційної інформації будь-кому, крім своїх адвокатів.

Ця проблема є, напевно, однією з найважливіших у питанні виведення конфіденційної інформації в хмару. Шляхів її вирішення може бути кілька. По-перше, можна шифрувати всю інформацію, що міститься на хмару. По-друге, можна просто її туди не поміщати. Однак, у всякому разі, у компаній, які користуються хмарними обчисленнями, це має бути певним пунктом у переліку питань інформаційної безпеки. Крім того, самі провайдери повинні покращувати свої технології, надаючи деякі послуги із шифрування.

Функціональність «хмарних» програм. Не всі програми або їх властивості доступні віддалено. Якщо порівнювати програми для локального використання

та їх "хмарні" аналоги, останні поки що програють у функціональності. Наприклад, таблиці Google Docs або програми Office Web Apps мають набагато менше функцій і можливостей, ніж Microsoft Excel.

Для розгортання веб-додатків розробнику не потрібно придбати обладнання та програмне забезпечення, немає необхідності організовувати їх підтримку. Доступ клієнта може бути організований за умов оренди.

Такий підхід має такі переваги:

· масштабованість;

· відмовостійкість;

· Віртуалізація;

· Безпека.

Масштабованість PaaS передбачає автоматичне виділення та звільнення необхідних ресурсів залежно від кількості користувачів, що обслуговуються додатком.

PaaS як інтегрована платформа для розробки, тестування, розгортання та підтримки веб-додатків дозволить весь перелік операцій з розробки, тестування та розгортання веб-додатків виконувати в одному інтегрованому середовищі, виключаючи цим витрати на підтримку окремих середовищ для окремих етапів.

Здатність створювати вихідний код і надавати його в загальний доступ усередині розробки значно підвищує продуктивність зі створення програм на основі PaaS.

Найвідомішим прикладом такої платформи є AppEngine від Google, яка пропонує хостинг для веб-застосунків з можливістю купувати додаткові обчислювальні ресурси (наприклад, для тестування високих навантажень). Для запуску програм Google AppEngine на віртуальних кластерних системах була розроблена платформа AppScale, яка не має жодного відношення до Google.

У системах веб-пошуку та контекстної реклами компанії Yahoo використовується платформа Hadoop, орієнтована на передачу великих обсягів даних між мережними серверами. На базі Hadoop побудовані HBase (аналог бази даних Google BigTable), а також HDFS (Hadoop Distributed File System, аналог Google File System).

Ще одним яскравим представником PaaS є продукти компанії Mosso:

- Cloud Sites - веб-хостинг (Linux, Windows, Mail) для навантажувальних веб-проектів з можливістю розширювати базові безкоштовні можливості за додаткову плату (трафік, сховище даних, обчислювальна потужність).

— Cloud Files — файловий cloud-хостинг зі щомісячною погігабайтною оплатою за обсяг файлів, що зберігаються. Управління здійснюється через браузер або за допомогою API (PHP, Python, Java, .NET, Ruby).

- Cloud Servers - погодинна оренда серверів (RAM за годину), з можливістю вибору серверної ОС. Можна змінювати параметри сервера, але не в режимі реального часу. Незабаром розробники обіцяють створити API для управління серверами.

Ну а в центрі всієї хмарної інфраструктури Microsoft – операційна система Windows Azure. Windows Azure створює єдине середовище, що включає хмарні

аналоги серверних продуктів Microsoft (реляційна база даних SQL Azure, що є аналогом SQL Server, а також Exchange Online, SharePoint Online та Microsoft Dynamics CRM Online) та інструменти розробки (.NET Framework та Visual Studio, оснащена в версії 2010 набором Windows Azure Tools). Так, наприклад, програміст, який створює сайт у Visual Studio 2010, може не виходячи з програми розмістити свій сайт у Windows Azure.

Програмне забезпечення як обслуговування (SaaS).

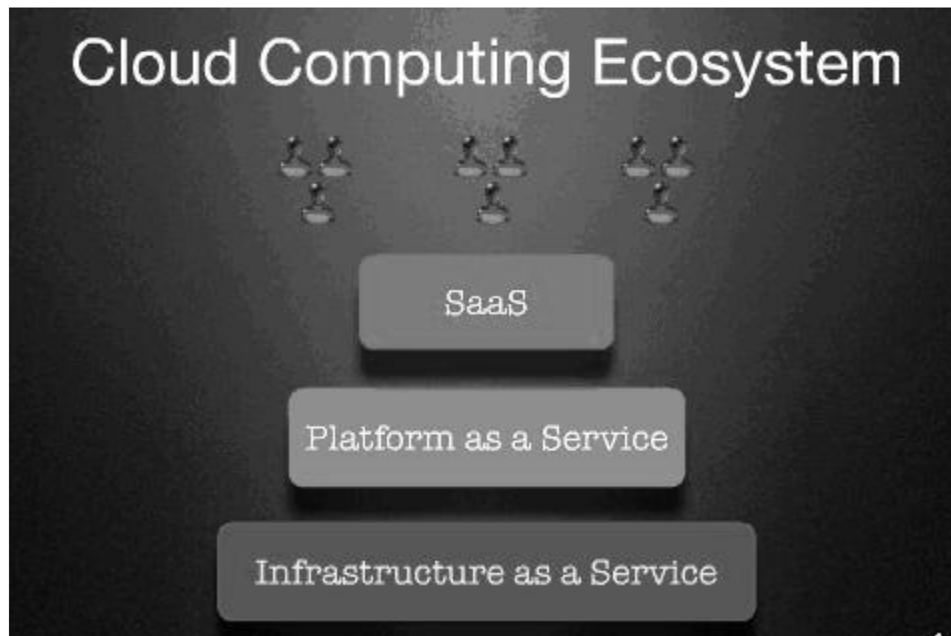


Рис.30. Вершина айсберга хмарних технологій представлена сервісами SaaS

SaaS – модель розгортання програми, яка передбачає надання програми кінцевому користувачеві як послуги на вимогу (on demand). Доступ до такої програми здійснюється через мережу, а найчастіше через Інтернет-браузер. В даному випадку, основна перевага моделі SaaS для клієнта полягає у відсутності витрат, пов'язаних із встановленням, оновленням та підтримкою працездатності обладнання та програмного забезпечення, що працює на ньому. Цільова аудиторія – кінцеві споживачі.

У моделі SaaS:

· Додаток пристосований для віддаленого використання;

· Одним додатком можуть користуватися кілька клієнтів;

· Оплата за послугу стягується або як щомісячна абонентська плата, або на основі сумарного обсягу транзакцій;

· Підтримка програми входить вже до складу оплати;

· Модернізація програми може здійснюватися обслуговуючим персоналом плавно та прозоро для клієнтів.

З точки зору розробників програмного забезпечення модель SaaS дозволить ефективно боротися з неліцензійним використанням програмного забезпечення завдяки тому, що клієнт не може зберігати, копіювати та встановлювати

програмне забезпечення.

По-суті, програмне забезпечення в рамках SaaS можна розглядати як зручнішу і вигіднішу альтернативу внутрішнім інформаційним системам.

Розвитком логіки SaaS є концепція WaaS (Workplace as a Service – робоче місце як послуга). Тобто клієнт отримує своє розпорядження повністю оснащене всім необхідним для роботи ПЗ віртуальне робоче місце.

За нещодавно опублікованими даними SoftCloud попитом користуються наступні SaaS додатки (у порядку зменшення популярності):

· Пошта
· Комунікації (VoIP)
· Антиспам та антивірус
· Helpdesk
· Управління проектами
· Дистанційне навчання
· CRM
· Зберігання та резервування даних

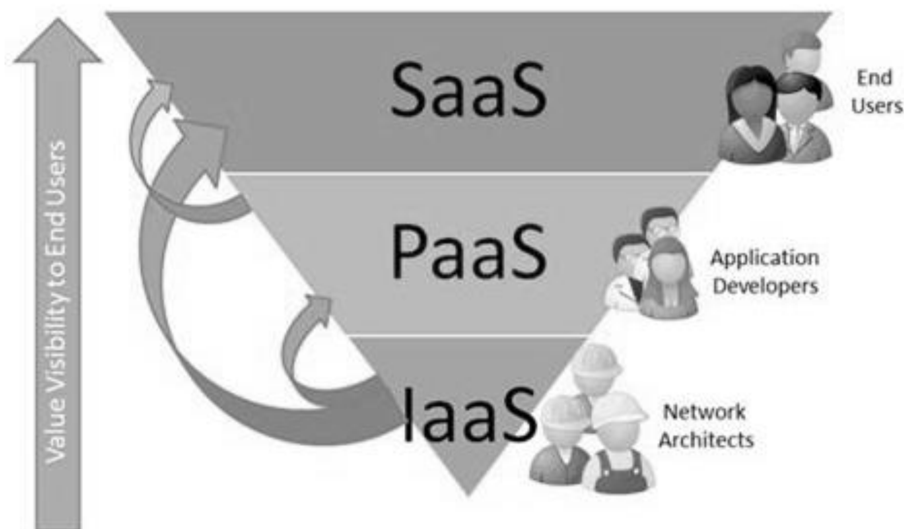


Рис.31. Сервіси SaaS мають найбільшу споживчу базу

Дуже схожими є продукти MobileMe (Apple), Azure (Microsoft) та LotusLive (IBM). Суть даних сервісів у цьому, що вони надають користувачам доступом до зберігання своїх даних (контакти, пошта, файли), і навіть спільної роботи кількох користувачів із документами.

Питаннями зберігання даних в Інтернеті спантеличена і компанія Google, яка розробляє проект GDrive, який буде віртуальним жорстким диском, який визначатиметься ОС як локальний. Також заявлено, що можна буде зберігати необмежену кількість даних, що звучить дуже привабливо.

Зберігання без обмежень також пропонує MediaFire.com. Є як повністю безкоштовне використання (щоправда, з деякими обмеженнями, наприклад, на максимальний розмір файлу, що завантажується), так і покупка преміум-акаунта, що розширює можливості (наприклад, шифрування файлів, отримання прямих

посилань на скачування).

Ще одним цікавим представником виду SaaS є продукт iCloud, що є операційною системою, працювати з якою можна безпосередньо через браузер. Інтерфейс операційної системи виконано у стилі Windows Vista/XP. На сьогоднішній день проект знаходиться на стадії бети і в самій ОС реалізований мінімум додатків.

Також до SaaS відносяться послуги Online backup, або, простіше кажучи, резервного копіювання даних. Користувач просто платить абонентську плату, а послуги самі автоматично в певний час шифрують дані з комп'ютера або іншого пристрою і відправляють їх на віддалений сервер, тим самим дані можуть бути доступні з будь-якої точки земної кулі. Дану послугу зараз надають безліч компаній, у тому числі, такі як Nero та Symantec.

Цікаве застосування cloud-технологіям знайшли і розробники комп'ютерних ігор: тепер сучасним комп'ютерам та ігровим приставкам не будуть потрібні потужні графічні адаптери (відеокарти), адже вся обробка даних та рендеринг будуть проводитися cloud-серверами, а гравці отримуватимуть вже оброблене відео. Одним із перших заявив про себе сервіс OnLive, і зовсім недавно про це заговорила і компанія Sony, яка збирається впровадити цю ідею у Playstation 3.

Відповідно до SaaS-концепції користувач платить не одноразово, купуючи продукт, а бере його в оренду. Причому, використовує ті функції, які йому потрібні. Наприклад, щорічно вам потрібна якась програма. І найчастіше ви її використовувати не збираєтеся. То навіщо ж купувати продукт, який у вас лежатиме без діла? І навіщо витрачати на нього місце (у квартирі, якщо коробка з диском, на вінчестері, якщо це файл)?

Конкуренція у хмарній сфері призвела до появи безкоштовних сервісів. Саме таким шляхом пішли два конкуренти — Microsoft і Google. Обидві компанії випустили набори послуг, що дозволяють працювати з документами. Google це Google Docs, Microsoft — Office Web Apps.

При цьому обидва сервіси тісно взаємопов'язані з поштою (Gmail у першому випадку та Hotmail у другому) та файловими сховищами. Таким чином, користувача як би переводять зі звичного йому офлайн-середовища в онлайн. Важливо, що і Google, і Microsoft інтегрують підтримку своїх онлайн-сервісів у всі програмні середовища як настільні, так і мобільні (нагадаємо, що Google створила ОС Android, а Microsoft Windows Phone 7).

Аналогічну концепцію (але з дещо іншими акцентами) просуває головний конкурент обох компаній — Apple. Йдеться про дуже цікавий сервіс під назвою MobileMe. Сервіс включає поштовий клієнт, календар, адресну книгу, файлове сховище, альбом фотографій і інструмент для виявлення втраченого iPhone. За можливість користуватися всім цим Apple бере приблизно 65 євро (або 100 доларів) на рік. При цьому Apple забезпечує такий рівень взаємодії свого набору інтернет-сервісів та програм на комп'ютері (під керуванням Mac OS X), телефоні, плеєрі та iPad, що потреба у використанні браузера пропадає. Ви користуєтеся звичними програмами на своєму Mac, iPhone і iPad, проте всі дані зберігаються не на них, а в хмарі, що дозволяє забути про необхідність синхронізації, а також

про їх доступність.

Якщо Apple інтегрує веб-сервіси в звичні програми операційної системи, то Google заходить з протилежного боку: операційна система Chrome OS, що розробляється інтернет-гігантом, являє собою фактично один браузер, через який користувач взаємодіє з розгалуженою мережею веб-сервісів. ОС орієнтована на нетбуки, відзначаються дуже низькі системні вимоги та відсутність необхідності самостійної установки програм (оскільки всі програми працюють безпосередньо в Інтернеті). Тобто Google надає переваги хмарної концепції, які зазвичай декламуються при роботі з корпоративними клієнтами, звичайним користувачам. Водночас очевидною є неможливість використання таких нетбуків у країнах з недостатньо широким проникненням широкосмугового інтернету.

Всі три типи хмарних послуг взаємопов'язані, і представляють вкладену структуру.

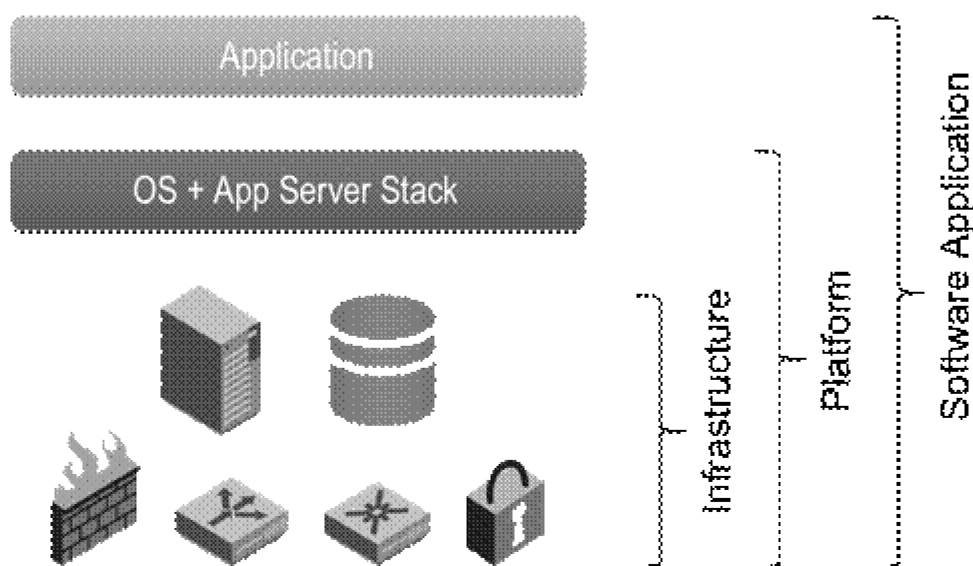


Рис. 32. Взаємозв'язок хмарних сервісів

Крім різних способів надання послуг розрізняють кілька варіантів розгортання хмарних систем.

Приватна хмара (private cloud) **-**використовується для надання сервісів усередині однієї компанії, яка є одночасно замовником і постачальником послуг. Це варіант реалізації «хмарної концепції», коли компанія створює її собі, у межах організації. Насамперед реалізація private cloud знімає одне з важливих питань, яке неодмінно виникає у замовників при ознайомленні з цією концепцією – питання захисту даних з погляду інформаційної безпеки. Оскільки «хмара» обмежена рамками самої компанії, це питання вирішується стандартними методами. Для private cloud характерне зниження вартості обладнання за рахунок використання ресурсів, що простоюють або неефективно використовуються. А також зниження витрат на закупівлю обладнання за рахунок скорочення логістики (не думаємо, які сервери закуповувати, в яких конфігураціях,

По суті, потужність нарощується пропорційно зростаючому в цілому навантаженні, не залежно від кожного завдання, що виникає – а, так би мовити, в середньому. І стає легше і планувати, і закуповувати та реалізовувати – запускати нові завдання у виробництво.

Публічна хмара – використовується хмарними провайдерами для надання сервісів зовнішнім замовникам.

Змішана (гібридна) хмара - спільне використання двох вищезгаданих моделей розгортання

Взагалі одна з ключових ідей Cloud полягає саме в тому, щоб з технологічної точки зору різниці між внутрішніми та зовнішніми хмарами не було і замовник міг гнучко переміщати свої завдання між власною та орендованою ІТ-інфраструктурою, не замислюючись, де саме вони виконуються.

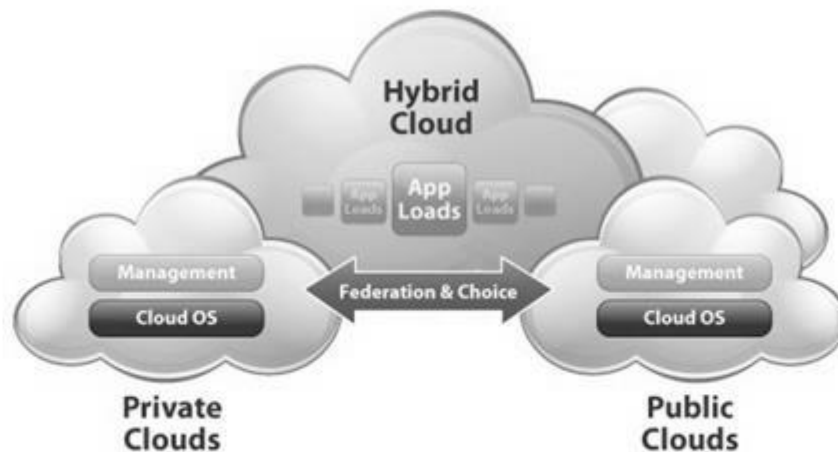


Рис. 33. Взаємозв'язок хмар різних типів

Таким чином, ці технології при спільному використанні дозволяють користувачам хмарних обчислень скористатися обчислювальними потужностями та сховищами даних, які за допомогою певних технологій віртуалізації та високого рівня абстракції надаються їм як послуги.

Тема 9. GRID-технологі

Основні напрямки досліджень у галузі Grid-технологій

Термін "*Grid*-обчислення" (*Computing grid*), де "*grid* " означає "решітка, сітка,*мережа* ",*за* змістом аналогічний виразу "єдина енергосистема". Суть його полягає у прагненні об'єднати всі комп'ютери світу в єдину систему – у віртуальний*суперкомп'ютер* небаченої потужності, що дозволить розподіляти та перерозподіляти ресурси між користувачами відповідно до їх запитів. Саме так людство користується електрикою єдиних енергетичних мереж. Основу такого об'єднання можна розглядати і для транспортних мереж, мереж забезпечення водою, нафтою, газом тощо.

Маючи такий *суперкомп'ютер*, *Користувач* може в будь-який час і в будь-

якому місці запросити стільки обчислювальних ресурсів, скільки йому потрібно для вирішення свого завдання. Більше того, він може замовити її рішення, надіславши *запит*-заявку на *сайт* одного із центрів обслуговування. Так само, як єдина енергосистема локально зазнає змінного навантаження (споживання електроенергії в окремій країні, регіоні чи населеному пункті змінюється залежно від часу доби, року тощо), так і система *Grid* -обчислень здатна перерозподіляти потужності, надсилаючи запити на недовантажені комп'ютери відповідно до збільшених вимог.

Надання обчислювальних ресурсів *на* вимогу пов'язують із відомим поняттям "комунальна послуга", що цілком відповідає тенденції об'єднання та розподілу ресурсів життєзабезпечення.

Однак на *Grid* -технології покладені на завдання та інформаційного характеру: зробити всю інформацію світу оперативно доступною. Це означає, що *користувач* повинен відчувати, що вся *інформація*, за якою він звернувся, знаходиться тут, поруч, не далі, ніж на сервері провайдера, до якого він підключений.

І ми ловимо себе на думці, що говоримо про Інтернет!

Так, по суті, йдеться про сміливий, фантастичний розвиток саме того, що фантастичним здавалося вчора, про розвиток всесвітньої павутини, про впровадження в неї обчислювальних функцій та підвищення якості інформаційного обслуговування. Саме так і намічається подальший*шлях* розвитку Інтернету: від WWW до GRID.

Таким чином, для *Grid*- технології характерні два (традиційно змішуваних) напрями розвитку, що визначаються їх важливістю та відмінностями у дослідженні.

Перший напрямок, пов'язаний з інформаційними завданнями, представляє сьогодні сутність Інтернету. Це обмін новинами, придбання нових знань, програмних засобів, мульти-медіа-розваги, діалогові системи телеконференції, електронна*пошта*, фінансові*операції* тощо. Ці функції" Всесвітньої Павутини" вже є достатніми найширшим верствам населення, покриваючи весь комп'ютерний сервіс. Вони органічно увійшли до базового виховання дітей, визначаючи рівень грамотності сучасної людини.

Проте саме тут є основна проблема. Некерований стихійний, навіть "дикий" розвиток Інтернету призводить до парадоксу: доступність величезних обсягів світової інформації породила значний*вага* відмов, відповідей про дійсну недоступність цієї інформації, про перевищення допустимого часу пошуку, помилки. Пошукові системи блукають*за* ієрархічною, заплутаною системою*Web*- серверів, натикаючись з їхньої навантаження, створюючи трафік зворотного руху інформації. Ми з відчаєм стежимо за замерлою планкою індикатора, чекаючи видачі такого знайомого запису. *Error ...*" — із припущенням про закінчення допустимого часу адресного звернення.

Зробити всю інформацію доступною, встановити гарантований час пошуку, породити у користувача відчуття того, що вся *інформація на* його запит знаходиться поруч, - це фантастичне завдання. Чи можна об'єднати всю

інформацію світу, що блукає в Інтернеті, в одну величезну базу даних, розмір якої навіть важко оцінити? Та ще й з необмеженим багатоканальним доступом?

Рух у напрямі здійснення цієї глобальної ідеї є завданням дослідників в області *Grid*- технологій. Ясно, що ідеального вирішення це завдання може ніколи не отримати. Проте прагнути мінімізації часу виконання запиту, попри структурну " віддаленість " інформації, безсумнівно, слід.

Другий напрямок розвитку *Grid*- технологій пов'язано, як раніше говорилося, з таким розширенням функцій Інтернету, при якому *користувач* може отримувати відповіді запити обчислювального характеру. Йдеться, звичайно, про ті завдання, вирішення яких недоступне широкому користувачеві — вченому, управлінцю, колективу, всім, кому потрібна достовірна *інформація* для найважливіших рішень. *Інтернет* повинен приймати замовлення на роботу із завданнями високої складності, великої розмірності, із завданнями моделювання складних фізичних явищ, таких, наприклад, як точний метеорологічний прогноз, із оптимізаційними завданнями господарського планування, із завданнями статистичної обробки експериментів, із завданнями — запитами *для* управління космічним простором, з імітаційними завданнями для випробування нових технічних засобів тощо.

Це означає, що в складі Інтернету повинні бути потужні обчислювальні центри, забезпечені пакетами прикладних програм, що розвиваються, вирішення складних завдань, оболонками для вирішення класів завдань. Такі центри повинні комплектуватися висококваліфікованими математиками, які розвивають обчислювальну базу, готові приймати замовлення, консультувати користувачів при "доведення" замовлень до необхідного формального представлення, бо мрії про повну автоматизацію процесу їх виконання ще довго будуть нездійсненними.

Необхідно врахувати й селекцію допустимих замовлень: за бажання, наприклад, усунути школяра-разгильдяя, який припиняє виконання уроку домашньому комп'ютеру.

При побудові *Grid*- систем особливу актуальність набувають проблеми захисту. Мета захисту – перекрити*доступ* до ресурсів та інформації для тих користувачів, які не мають відповідних привілеїв. Таким користувачем може стати і власник, який надав свій*комп'ютер* в оренду або*за* договором на час простою.

Базовими елементами захисту є:

- автентифікація - перевірка справжності учасника взаємодії із системою;
- механізм авторизації - *перевірка допустимості* затребуваної операції;
- конфіденційність та цілісність даних – захист від нелегітимного доступу;
- білінг та аудит — здатність контролювати та підраховувати обсяг використаних ресурсів та забезпечуваних ними служб;
- суворе виконання зобов'язань — здатність встановлювати, що цей учасник виконав певне завдання чи погодився її виконання, навіть якщо він це заперечує.

Історична довідка

Перші досліди в галузі *Grid*- технологій пов'язані з розрахунками експериментів у ядерній фізиці. Вважається, що цей*досвід* взагалі став базою формування World Wide*Web*, *WWW* - Всесвітньої павутини. З ним пов'язують ім'я Тім Бернес-Лі. Перед цим ученим було поставлено завдання знайти спосіб, який дозволив би вченим, які беруть участь в експериментах на Великому адронному колайдері, обмінюватися даними та подавати результати їх обробки на загальне обговорення. Багато фізиків, більшу частину часу що у своїх наукових інститутах, теж хотіли повноправно брати участь у аналізі даних.

Тім Бернес-Лі запропонував створити в Європейській організації ядерних досліджень (*CERN*) систему розподіленого інформаційного забезпечення, засновану на використанні *гіпертексту* та здатну об'єднати наукові центри всієї Землі. Були написані спеціальні програми, встановлені на багатьох комп'ютерах світу, розбиті на групи, пов'язані зі своїм сервером. Ці програми могли працювати з єдиною *БД CERN*, за допомогою додаткових серверів переробляючи дані та повертаючи результати в єдину*БД*.

У 1990 році прототип того, що згодом отримало назву Всесвітньої Павутини, був створений у *CERN*, а починаючи з 1991 року, перші *браузери* та *WWW*- сервери з'явилися у розпорядженні ядерних фізиків усього світу. Широке поширення відразу ж набули мови *HTML* і протокол *HTTP*.

Проте теоретичне *узагальнення* досвіду *CERN* та розвиток ідеї *WWW* в область сучасного уявлення про майбутню мережу *Grid*, було зроблено американськими вченими Яном Фостером та Карлом Кессельманом. *За* їхнім уявленням, *Grid* дійсно є "надбудовою" над Інтернетом, призначеною для розподілених обчислень при вирішенні завдань високої складності у галузі науки та технологій.

На відміну від безструктурного павутиння *WWW*, грати *Grid* суворо упорядкована. Характерно, що поділу на інформаційні завдання та завдання обчислювальні автори не припускають. *Користувач*, підключаючись до*Grid*, отримує*доступ* до мільйонів комп'ютерів, як обчислень, так зберігання даних, здобутих у результаті рішення, й у розміщення великих масивів інформації. Першочергова увага приділяється проблемам безпеки, анонімності, таємності.

Grid-технології вирішення інформаційних завдань

Вже відомі розробки, що підтримують складні розподілені *бази даних* багатоканального використання. Одним з проектів є розробка *Oracle 10G*, призначена для реалізації комерційної *Grid*- системи. Її *механізми* підтримують такі підсистеми та функції:

- Grid зберігання даних;
- Grid серверів БД;
- Grid серверів додатків;
- Засоби самоналаштування вузлів БД;
- Систему керування Grid;
- Кошти для *поділу інформації* між вузлами Grid.

Проаналізуємо, що з викладеного в даному курсі матеріалу про паралельні інформаційні технології можна використовувати в рамках напряму досліджень, що розглядається.

Насамперед, припускаючи, що метою досліджень є не просто сервісна *підтримка*, а*оптимізація* інформаційного обслуговування, необхідно сформулювати*критерій оптимізації*. Логічно вибрати таким критерієм *мінімум середнього часу виконання одиничного запиту* з урахуванням часу обробки його пошуковою системою.

Але саме в такому світлі є проблема мінімізації часу звернення до бази даних, викладена в "Застосування SPMD-технології при побудові мережевих баз даних з циркулюючою інформацією"! І там математично, на основі теорії масового обслуговування, доведено, що активізація сукупної пам'яті *БД у* вигляді" зустрічної" циркуляції сегментів як забезпечує синхронізацію незалежного багатоканального доступу, але, головне, сприяє повільному, цілком прийнятному зростанню часу виконання запиту залежно від зростання кількості абонентів. (Інакше цей час швидко прагне нескінченності.)

Було показано, що одним із методів побудови *БД* з циркулюючою інформацією є *об'єднання* серверів в єдину замкнуту систему - в "кільце", *за* яким циркулюють *сегменти* цієї основи. Абоненти ж жорстко пов'язані зі своїми серверами, забезпечуючи розподілені, приблизно рівні, потоки запитів на кожен*сервер*. *По* кожному запиту має враховуватись середнє *час очікування* потрібного сегмента з допомогою його циркуляції.

Тоді можна уявити таку схему функціонування гіпотетичної Асоціації *Web*- серверів (рис. 33), об'єднаних на основі гігантської сукупної *бази даних* , світового (або хоча б корпоративного) масштабу. Враховуючи все зростаючу *потужність* серверів і, головне, - засобів передачі даних, можна вже сьогодні очікувати значного скорочення кількості відмов при виконанні запитів, що вимагають складного багатоступінчастого трафіку, що залежить від пропускної спроможності багатьох проміжних пунктів.

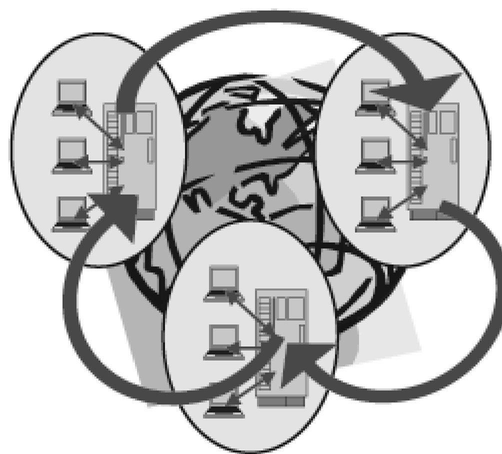


Рис. 34. Асоціація Web-серверів світового інформаційного простору

Відомі проекти Grid-технології вирішення обчислювальних завдань

Grid- технології, швидко поширюючись, завойовують ключові позиції в галузі високих технологій у всьому світі, стверджуючи пріоритетний розвиток усієї широкої галузі інформаційних технологій (ІТ) як локомотива державного процвітання.

Отже, система *Grid* -обчислень - це розподілене програмно-апаратне комп'ютерне середовище, з принципово новою організацією обчислень та управління потоком завдань та даних.

Повчальним прикладом практичного втілення концепції *Grid* може служити глобальний проект *China Grid*, запущений корпорацією *IBM* спільно з міністерством освіти Китаю з метою підвищення ефективності науково-дослідної та освітньої діяльності провідних китайських університетів. *Grid* -система, побудована на операційній платформі *Linux*, забезпечує інтегроване, відкрите, віртуалізоване та автономне робоче середовище. Нею має бути охоплено близько 100 навчальних закладів *по* всій країні, що залучить до проекту понад 200 тисяч студентів та викладачів. Завдяки *China Grid* китайські університети сподіваються скоротити *витрати* на науково-дослідні та *дослідно-конструкторські роботи*.

Прикладом реалізації розглянутої технології може бути експериментальний Grid-сегмент МДУ ім. М.В.Ломоносова.

В основі цього проекту лежить проект *EDG* (*EU Data GRID* – європейський проект для фізики високих енергій, біоінформатики та системи спостережень за Землею). В свою*Насамперед*, ним використовується проект *Globus* (розробник - *Argonne National Lab*.), Що надає інструментальні засоби зв'язку, інформаційного обслуговування, безпеки, управління ресурсами, локального управління ресурсами та завданнями.

Програмне забезпечення Globus доступний і розповсюджується вільно.

Проект *EDG* передбачає формування обчислювальних ресурсів з урахуванням аналізу сайтів організацій, підрозділів, обчислювальних вузлів.

Типовий *сайт* містить наступні *Розділи* інформації.

- *Комп'ютер кінцевого користувача*, що дозволяє користувачеві взаємодіяти з Grid-середовищем, зокрема, запускати завдання та отримувати результати.
- *Обчислювальний елемент*, що представляє посилання на Grid-інтерфейс до локальної системи керування пакетною обробкою (СУПО).
- *Робочі вузли*, що входять до складу обчислювального елемента та керовані за допомогою СУПО, що приховано від користувача.
- *Накопичувач даних*, що управляє одноманітним доступом до будь-яких пристроїв пам'яті.

- *Брокер ресурсів*, що приймає завдання від користувача, що узгоджує вимоги до ресурсів, які містяться в описі завдання, з наявними, та направляє завдання на відповідний сайт обчислювального елемента.
- *Каталог реплік*, який підтримує базу даних про місця зберігання оригінальних файлів та всіх їх копій.

Щоб отримати *доступ* до ресурсів МДУ-*GRID*, необхідний*комп'ютер* із встановленим користувачем *Grid* -інтерфейс. *Доступ* у середовище *Grid* відбувається під ім'ям, яке міститься в сертифікаті, і контролюється за допомогою спеціальної програми-посередника (електронної довіреності), яка створюється на обмежений термін за допомогою персонального ключа користувача.

Файл з описом завдання створюється за допомогою мови опису завдань (*Job Description Language*) і містить необхідні *вхідні дані*, вимоги до ресурсів та відомості про те, куди мають бути записані результати обробки завдання.

Основи проектування Центру Grid-технологій

У підході до реалізації ідеї *Grid-* технологій спостерігається деяка поспішність. Немає формального аналізу можливих видів запитів, слабко виражено поділ на інформаційні та обчислювальні завдання.

Матеріал попередніх лекцій свідчить, що це алгоритми і всі роботи, виконувані з допомогою обчислювальних засобів, *структуровані*. Недостатньо пов'язувати виконання запиту вирішення завдання лише з виділенням обчислювальних ресурсів. Ними треба керувати, а обчислювальний процес – розпаралелювати.

В "Організація та оптимізація паралельних процесів" представлена загальна схема паралельного обчислювального процесу, з якої видно обов'язкову наявність двох керуючих елементів: супервізора та диспетчера. *Супервізор* управляє ходом обчислювального процесу, реалізуючи його функціональну спрямованість. *Диспетчер* керує використанням обчислювальних ресурсів.

На відміну від завдань перерозподілу енергії, які подібні до завдань оптимального використання бригади землекопів, що риють велику яму, завдання розпаралелювання обчислень, як зазначалося вище, стикаються з основною труднощами: з необхідністю дотримання часткової впорядкованості *робіт* з необхідністю вирішення конкретних*задач синхронізації* ("Синхронізація паралельних процесів").

Перші досліди застосування *Grid* -обчислень, подібні до досвіду Тіма Бернес-Лі, дозволяли розподіляти між комп'ютерами світу роботи, мало пов'язані між собою. То були незалежні експерименти, великі масиви даних. Проте результати поверталися та оброблялися централізовано. Це дуже проста схема розпаралелювання*за* інформацією, примітивний аналог *SPMD-технологі*.

У загальному випадку, навіть при реалізації цієї технології, що є втіленням розпаралелювання *за* інформацією, неможливо уникнути синхронізації *за* загальними даними.

Механізми синхронізації повинні працювати швидко, що вимагає конкретного аналізу можливості та доцільності вирішення завдань на основі*віртуального ресурсу Інтернет*. Це зауваження на користь ідеї концентрації обчислювальних ресурсів.

Відповідно до вимог практичної доцільності, необхідно усвідомити, якого типу бувають запити обчислювального характеру. *Grid*- систем (рис. 35).

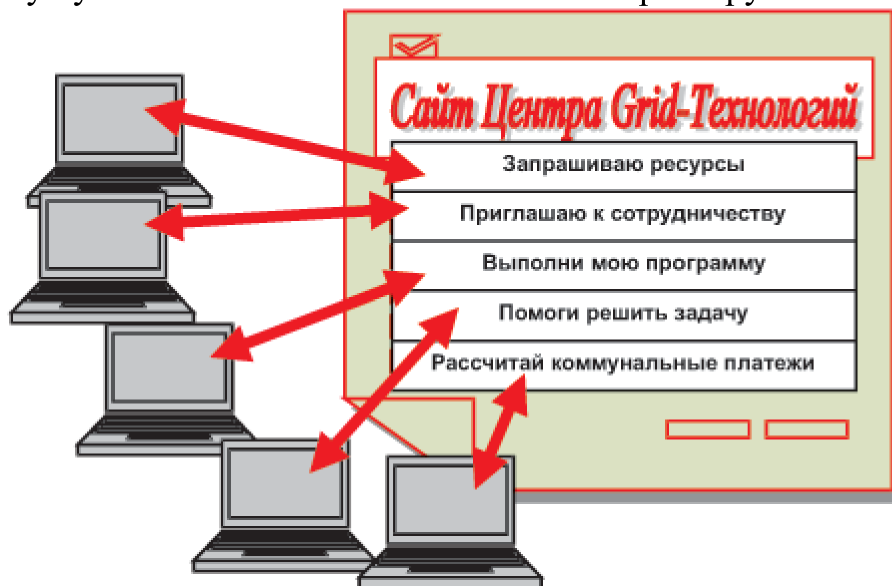


Рис. 35. Типи запитів до системи Grid -обчислень

Уявімо, що *Користувач* заходить на *сайт* великих *GRID*- Компанії. Його бажання можуть бути такими:

1. Він готовий, наприклад, орендувати ресурс для побудови задуманої автоматизованої системи управління залізничним транспортом. Користувач у діалоговому режимі докладно викладає необхідні характеристики, у тому числі кількісні.

Компанія на основі своїх можливостей погодить з користувачем послугу. Далі йдуть домовленості фінансового та юридичного характеру.

Таким чином, користувач, отримавши докладний опис наданих йому ресурсів, сам працює з ними, ввівши за необхідності їх адресацію або застосовуючи принцип віртуальності, - за задумом створюваної системи. При цьому, зрозуміло, він має засоби взаємодії між окремими засобами для організації необхідної йому схеми обчислень.

2. Користувач хоче вирішити або вирішувати періодично, із заданим темпом, за динамічно змінюваними вихідними даними, транспортне завдання (проблему Хічкока) з відомими обмеженнями пропускної спроможності магістралей. Звертаючись до цієї компанії, він передбачає наявність у її пакеті прикладних програм (*ППП*) потрібної програми, та ще вирішуваної швидко на основі застосування методу розпаралелювання. Він вступає у діалог з Компанією, уточнюючи постановку завдання, умови її вирішення, режим

вирішення, обіцяє продовжити стосунки у разі успішного досвіду, погоджує умови оплати тощо.

Якщо умови йому не подобаються, користувач звертається до сайту іншої компанії.

3. Користувач хоче вирішити конкретне завдання знаходження екстремуму функції від 1020 змінних на позитивній ділянці, обмеженої 524 гіперплощинами та заданою системою диференціальних рівнянь у приватних похідних другого порядку. Звертаючись до Компанії, користувач може розраховувати лише наявність оболонок, у яких можна занурювати конкретні умови завдання та описи висловлювання.

У діалоговому режимі висококваліфікований консультант пропонує вирішити завдання у два етапи: спочатку за допомогою наявної оболонки знайти рішення системи диференціальних рівнянь, а потім, за допомогою іншої оболонки, знайти екстремум.

Якщо користувач погоджується на всі умови щодо часу рішення та оплати, укладається договір. Щоправда, у договорі має бути передбачено ситуацію, коли у процесі рішення з'ясовується некоректність завдання вихідних даних, нерозв'язність, неоднозначність рішення тощо. Такі ситуації вимагають подальших консультацій, уточнення постановки завдання та величини оплати, а, можливо, і розірвання договору, після чого можуть наслідувати значні юридичні та фінансові позови.

4. Користувач знає алгоритм розв'язання свого завдання, описаний мовою Паскаль. Він нічого не бажає знати про ресурси, він хоче отримати тривимірний масив результатів рішення або місце його зберігання. Про статистичну обробку та побудову діаграм він поки не думає, залишаючи це питання в рамках рішення про доцільність подальшої співпраці з Компанією.

Обробка такого запиту користувача найбільше приємна Компанії, яка, не відчуваючи відповідальності, може видати відповідь про некоректність запису алгоритму або помилки.

5. Користувач звертається по розв'язання задачі з фізики за IX клас. Компанія може надавати і таку послугу — з неформальної постановки завдання. Висококваліфікований спеціаліст (користуючись досконалим довідником), виконавши нехитрі обчислення, у діалоговому режимі роз'яснить користувачеві необхідність батьківських гарантій у частині передоплати та висловить свою думку про витрати виховання.

6. Домогосподарка вимагає про пільгову допомогу при розрахунку оплати комунальних послуг, виходячи з поточного тарифу, з випередженням зазначених витрат електроенергії, гарячої та холодної води на випадок неодмінного та швидкого його підвищення. Компанія, в особі досвідченого консультанта, що вступив у діалог, маючи можливості фінансових послуг за відомими, постійно оновлюваними даними, пропонує оплатити не 112 квт, а 200, т.к. тариф у наступному місяці підвищиться, а зазначена витрата лежить у межах критично допустимої. У цьому сенсі Grid-технологія справді вторгається у сферу комунальних послуг.

На основі цих прикладів можна зробити висновок, що завдання нашої гіпотетичної *Grid* -Компанії, яку точніше слід назвати "Центр *GRID* -Технологій", дуже широкі і різноманітні. Наведемо низку принципів його діяльності та деякі пункти його обґрунтування.

1. Необхідність (і неминучість) таких підприємств, які мають великим арсеналом обчислювальних засобів, очевидна. Знеособленого звернення до Інтернету не буває. *Хтось* має обробляти запити.
2. Така компанія, мабуть, будучи акціонерним товариством, у свою чергу, може орендувати кошти дрібніших власників, а також мережеве обладнання, системи передачі даних, включаючи космічні.
3. Розглянемо, які обчислювальні засоби мають становити власну технологічну базу Центру.

Центр, поза сумнівом, має об'єднувати рішення інформаційних та обчислювальних завдань. Тому наявність Web-серверів та участь їх у Всесвітній чи Регіональній асоціації є обов'язковою.

Що стосується виключно обчислювальних завдань, то тут, як говорилося вище, ресурс може бути складений на договірній основі, на основі фактично розрізнених обчислювальних засобів організацій та окремих власників.

Проте *основний капітал* Центру GRID-Технологій визначається власними обчислювальними засобами та супутнім обладнанням. Його комплектування та розвиток визначається двома можливими напрямками:

- комплектацією на базі перспективних або розроблюваних суперкомп'ютерів та багатопроцесорних обчислювальних систем на їх основі;
- комплектацією на базі існуючого парку комп'ютерів, робочих станцій і серверів, що стрімко розвивається, які за допомогою супутнього мережевого обладнання об'єднані в локальну обчислювальну мережу, що реалізує розподілені обчислення. Цей напрямок, який не заперечує перший, безсумнівно, більш перспективний.

4. Центр має бути укомплектований висококваліфікованими математиками-програмістами та системотехніками, що працюють за декількома напрямками:

- оперативне консультування користувачів при "доведення" завдань;
- розробка пакета прикладних програм та оболонок;
- співробітництво з користувачами при спільній розробці складних програмних та інформаційних проектів;
- науково-дослідна діяльність з розвитку Grid-технологій та їх застосування;

Проаналізувавши матеріали лекцій, позначимо той пакет прикладних програм та оболонок, який може бути запропонований Центру *GRID* -технологій. Це:

- розв'язання систем лінійних рівнянь;
- завдання сортування та пошуку;
- паралельний логічний висновок;
- паралельні технології вирішення задач лінійного програмування;
- паралельні технології вирішення задач цілісного лінійного програмування;

- паралельні технології вирішення транспортного завдання без обмежень та з обмеженнями пропускної спроможності магістралей;
- паралельні технології знаходження максимальної пропускної спроможності мережі;
- методи точного вирішення задач розпаралелювання як задач мережевого планування та управління;
- завдання оперативного паралельного планування - задачі диспетчування;
- оболонка: звичайно-різносні методи розв'язання систем диференціальних рівнянь;
- оболонка: паралельна технологія розв'язання оптимізаційного завдання нелінійного програмування при лінійних обмеженнях;
- оболонка: паралельна технологія розв'язання "плоської" оптимізаційної задачі нелінійного програмування при нелінійних обмеженнях (легко розвивається на випадки більшої розмірності);
- оболонка: паралельний пролог;
- оболонка: розрахунок величини збудження нейронів нейромережі

Слід зазначити також пропоновані численні алгоритми та схеми організації паралельних обчислень у BC і процесори різної архітектури, а також способи побудови паралельних керуючих процесів з урахуванням їх синхронізації.

Питання для самостійної роботи:

1. Вивчення моделі розподіленої організаційної системи: банківські системи.
2. Обробка інформації в супермережі (Грід). Аналіз архітектури Грід.
3. Вивчення системи грід-освіти.
4. Вичення можливостей GSM мереж для передачі даних в територіально розподілених системах.
5. Інтернет, розподілені мережі VPN.
6. Територіально-розподілена мережа передачі даних.
7. Розподілено-корпоративні мережі.
8. Використання соціальних мереж в розподілених обчисленнях.
9. Вивчення розподіленої бездротової системи WDS (англ. Wireless Distribution System).
10. Теририторіально розподілені мережі звязку.

Список використаної літератури:

1. Andrews, Gregory R. (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Addison–Wesley, ISBN 978-0-201-35752-3.
2. Arora, Sanjeev; Barak, Boaz (2009), Computational Complexity – A Modern Approach, Cambridge, ISBN 978-0-521-42426-4.
3. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (1990), Introduction to Algorithms (1st ed.), MIT Press, ISBN 978-0-262-03141-7.
4. Dolev, Shlomi (2000), Self-Stabilization, MIT Press, ISBN 978-0-262-04178-2.
5. Elmasri, Ramez; Navathe, Shamkant B. (2000), Fundamentals of Database Systems (3rd ed.), Addison–Wesley, ISBN 978-0-201-54263-9.
6. Ghosh, Sukumar (2007), Distributed Systems – An Algorithmic Approach, Chapman & Hall/CRC, ISBN 978-1-58488-564-1.
7. Lynch, Nancy A. (1996), Distributed Algorithms, Morgan Kaufmann, ISBN 978-1-55860-348-6.
8. Herlihy, Maurice P.; Shavit, Nir N. (2008), The Art of Multiprocessor Programming, Morgan Kaufmann, ISBN 978-0-12-370591-4.
9. Papadimitriou, Christos H. (1994), Computational Complexity, Addison–Wesley, ISBN 978-0-201-53082-7.
10. Peleg, David (2000), Distributed Computing: A Locality-Sensitive Approach, SIAM, ISBN 978-0-89871-464-7, archived from the original on 2009-08-06, retrieved 2009-07-16.
11. Java Remote Method Invocation (Java RMI) [Electronic Resource]. – Available from: http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/. – 16.10.2015.
12. Таненбаум, Э. Распределенные системы: принципы и парадигмы / Э. Таненбаум. – СПб: Питер, 2003. – 877 с.
13. Родин, А. В. Параллельные или распределенные вычислительные системы / А. В. Родин, В. Л. Бурцев // Компьютерные системы и технологии. – М.: МИФИ, 2006. - Т. 12. - С. 151 – 153.
14. Grid Computing in Research and Education [Electronic Resource] / L. Ferreira, F. Lucchese, T. Yasuda, C. Y. Lee, C. A. Queiroz, E. Minetto, A. Mungioli. – IBM: RedBooks, 2005. – 180 p. – Available from: http://www.redbooks.ibm.com/redbooks/pdfs/sg246649.pdf. – 16.10.2015.
15. Java EE Compatibility [Electronic Resource]. – Available from: http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp136984.html. – 16.10.2015.
16. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5 / Э. Троелсен. – 6-е изд. — М.: Вильямс, 2013. — 1312 с.
17. Mohamed, A. A history of cloud computing [Electronic resource] / A. Mohamed. - Available from: http://www.computerweekly.com/feature/Ahistory-of-cloud-computing. – 13.10.2015.