МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНБАСЬКА ДЕРЖАВНА МАШИНОБУДІВЕЛЬНА АКАДЕМІЯ

КАФЕДРА «АВТОМАТИЗАЦІЯ ВИРОБНИЧИХ ПРОЦЕСІВ»

КОНСПЕКТ ЛЕКЦІЙ

по дисципліні «РОЗПОДІЛЕНІ КОМП'ЮТЕРНІ СИСТЕМИ ТА МЕРЕЖІ»

для студентів спеціальності 123
«Комп'ютерна інженерія»

Затверджено
на засіданні кафедри АВП
Протокол № 3
від „06" листопада 2017 року

КРАМАТОРСЬК 2017

УДК 004

Конспект лекцій по дисципліні «Розподілені комп'ютерні системи та мережі» .для студентів спеціальності 123 «Комп’ютерна інженерія» / Укладач О.В. Суботін. - Краматорськ: ДДМА, 2017. - 52 с.

Коротко викладено матеріал з основних розділів курсу «Розподілені комп'ютерні системи та мережі» для спеціальності 123 «Комп’ютерна інженерія».

Укладач: О.В. Суботін, доцент

Відповідальний за випуск: О.В. Суботін, доцент

ЛЕКЦІЯ 1.
ЗАГАЛЬНІ ПОЛОЖЕННЯ

Питання до вхідного контролю з дисципліни «Розподілені комп'ютерні системи та мережі»

1 Що таке «розподілена» обчислювальна система?
2 У чому сенс поняття «сервісно-орієнтована архітектура»?
3 Що таке Web-сервіс?
4 Дайте визначення поняттю «гетерогенна комп'ютерна система».
5 Що таке «кроссплатформенность»? Що може бути (або не бути) кроссплатформенную?
6 Що таке интероперабельность?
7 У чому сенс мережевої моделі OSI?
8 Перерахуйте рівні мережевий моделі OSI.
9 Мережевий рівень моделі OSI. Функції та протоколи.
10 Транспортний рівень моделі OSI. Функції та протоколи.
11 Прикладний рівень моделі OSI. Функції та протоколи.
12 Функції та протоколи фізичного і канального рівня моделі OSI.
13 Що таке grid-система?
14 У чому суть «хмарних» технологій?
15 У чому проблеми розподіленого зберігання даних?
16 Що є ресурсами для паралельної обчислювальної системи?

Хмарне сховище даних - модель онлайн-сховища, в якому дані зберігаються на численних розподілених в мережі серверах, що надаються в користування клієнтам, в основному, третьою стороною. На противагу моделі зберігання даних на власних виділених серверах, придбаних або орендованих спеціально для подібних цілей, кількість або будь-яка внутрішня структура серверів клієнту, в загальному випадку, хоч я знаю. Дані зберігаються, а так само і обробляються, в так званому хмарі, яке представляє собою, з точки зору клієнта, один великий віртуальний сервер. Фізично ж такі сервери можуть розташовуватися віддалено один від одного географічно, аж до розташування на різних континентах.

Переваги хмарних сховищ

Клієнт платить тільки за те місце в сховищі, яке фактично використовує, але не за оренду сервера, всі ресурси якого він може і не використовувати.

Клієнту немає необхідності займатися придбанням, підтримкою і обслуговуванням власної інфраструктури зі зберігання даних, що, в кінцевому рахунку, зменшує загальні витрати виробництва.

Всі процедури по резервуванню і збереженню цілісності даних виробляються провайдером хмарного центру, яка не втягує в цей процес клієнта.

Потенційні проблеми

Безпека при зберіганні і пересилання даних є одним з основних питань при роботі з хмарою, особливо щодо конфіденційних, приватних даних.

Загальна продуктивність при роботі з даними в хмарі може бути нижче такої при роботі з локальними копіями даних.

Надійність і своєчасність отримання і доступності даних в хмарі дуже сильно залежить від багатьох проміжних параметрів, в основному таких як канали передачі даних на шляху від клієнта до хмари, питання останньої милі, питання про належну якість роботи інтернет-провайдера клієнта, питання про доступність самої хмари в даний момент часу.

Хмарні шлюзи

Хмарні шлюзи - технологія, яка може бути використана для більш зручного представлення хмари клієнту. Наприклад, за допомогою відповідного програмного забезпечення, сховище в хмарі може бути представлено для клієнта як локальний диск на комп'ютері. Таким чином, робота з даними в хмарі для клієнта стає абсолютно прозорою. І при наявності гарної, швидкого зв'язку з хмарою клієнт може навіть не помічати, що працює не з локальними даними у себе на комп'ютері, а з даними, що зберігаються, можливо, за багато сотень кілометрів від нього.

Показники для оцінки хмарних сховищ

- Ціна / безкоштовність різних видів послуг.
- Обсяг інформації, що зберігається (в різних конфігураціях).
- Стабільність роботи і репутація.
- Простота і швидкість здійснення елементарних операцій по приміщенню файлу в сховище.
- Гнучкість і розвиненість можливостей з управління доступом.
- Простота управління і конфігурації (політика безпеки і розмежування доступу, простота підключення нових послуг).
- Наявність шлюзів і клієнтських додатків під різні ОС.
- Використовувані алгоритми шифрування і інші засоби управління безпекою.
- Наявність (і рівень можливостей) API для розробки своїх додатків.

Питання для контролю та самостійної роботи

1 Що таке «хмарні технології»? Які цілі і причини їх використання?
2 Які переваги та недоліки хмарних сховищ даних?
3 Які хмарні сховища даних ви знаєте?
4 Які можливості надає кожне з них?
5 Яка їхня порівняльна популярність?
6 Що таке Directory Service (Служба каталогів)? Які операції можуть бути доступні через служби каталогів хмарних сховищ даних?
7 Які хмарні сховища даних надають API для роботи зі своїми службами каталогів?
8 Яким чином можуть бути реалізовані ці API? Які мережеві протоколи для цього використовуються? До яких рівнями OSI відносяться ці протоколи? Чи не порушується ідеологія OSI таким їх використанням?

ЛЕКЦІЯ 2.
RPC

Віддалений виклик процедур (або виклик віддалених процедур) (від англ. Remote Procedure Call (RPC)) - клас технологій, що дозволяють комп'ютерним програмам викликати функції або процедури в іншому адресному просторі (як правило, на віддалених комп'ютерах).

Мета - побудова розподілених програмних комплексів на базі гетерогенних мережевих середовищ (з вузлами на базі різних апаратних платформ і операційних систем).

Вимога - інтероперабельність (англ. Interoperability - здатність до взаємодії) - здатність продукту або системи, інтерфейси яких повністю відкриті, взаємодіяти і функціонувати з іншими продуктами або системами без будь-яких обмежень доступу і реалізації.

Ідея- виклик віддаленої процедури повинен виглядати по можливості так само, як виклик локальної процедури. (Викликає процедурі не потрібно знати, що викликається процедура знаходиться на іншій машині, і навпаки).

Специфіка: Відсутність загального адресного простору.

Таблиця 1 - Відмінності RPC від локального виклику

RPC	Локальний виклик
Різні ОС, може бути різного типу.	одна ОС
Різні комп'ютери, можуть бути різні апаратні платформи.	один комп'ютер
Різні адресні простори.	Загальна адресний простір
Не може використовувати зовнішні покажчики в параметрах.	Може використовувати зовнішні покажчики в параметрах.
Для передачі параметрів використовує мережевий транспорт.	Для передачі параметрів використовує пам'ять процесу.

Реалізація віддалених викликів істотно складніше реалізації викликів локальних процедур. Оскільки викликає і викликається процедури виконуються на різних машинах, вони мають різні адресні простори, і це створює проблеми при передачі параметрів і результатів, особливо якщо машини не ідентичні. Так як RPC не може розраховувати на пам'ять, що розділяється, це означає, що параметри RPC не повинні містити покажчиків на вічка не стековой пам'яті і що значення параметрів повинні копіюватися з одного комп'ютера на інший. Наступна відмінність RPC від локального виклику полягає в тому, що він обов'язково використовує нижележащую систему зв'язку, однак це не повинно бути явно видно ні у визначенні процедур, ні в самих процедурах.

Проблема: Неможливість прямого копіювання даних.

Причини:

- різний порядок проходження байт;
- механізми розподілу пам'яті ;
- відмінності уявлення структур даних в мовах програмування;

- існування динамічних структур та об'єктів.

Порядок проходження байтів - метод запису байтів мультибайтних чисел (залежить від архітектури)

Порядок від старшого до молодшого або (англ. Big-endian, дослівно: «тупокінцевий»):, запис починається зі старшого і закінчується молодшим. Цей порядок є стандартним для протоколів TCP / IP, він використовується в заголовках пакетів даних і в багатьох протоколах більш високого рівня, розроблених для використання поверх TCP / IP. Тому, порядок байтів від старшого до молодшого часто називають мережевим порядком байтів (англ. Network byte order).

Порядок від молодшого до старшого або (англ. Little-endian, дослівно: «гострий») Цей порядок записи прийнятий в пам'яті персональних комп'ютерів з x86-процесорами, в зв'язку з чим іноді його називають интеловский порядок байтів.

Перемикається порядок - для деяких процесорів.

Змішаний порядокбайтів (англ. middle-endian) іноді використовується при роботі з числами, довжина яких перевищує машинне слово. Число представляється послідовністю машинних слів, які записуються в форматі, природному для даної архітектури, але самі слова слідують в зворотному порядку.

Рішення: Кошти серіалізації.

Серіалізация (в програмуванні) - процес перекладу будь-якої структури даних в послідовність бітів. Зворотною до операції серіалізації є операція десеріалізациі - відновлення початкового стану структури даних з бітової послідовності.

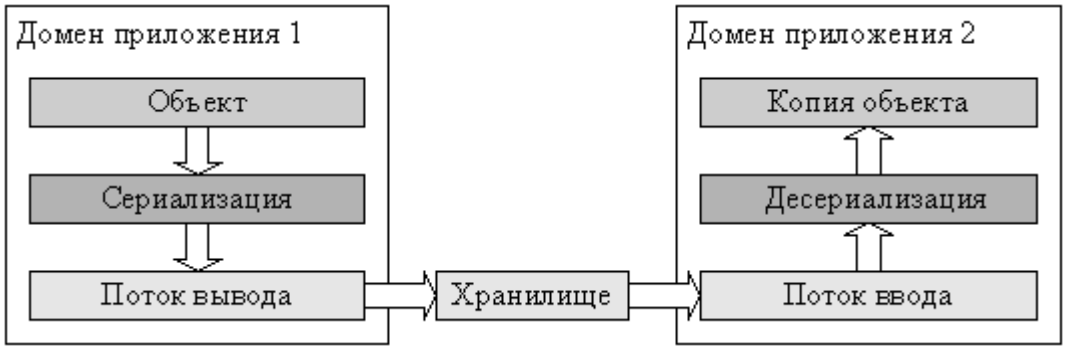

Рисунок 1. - Складна структура (дерево, зв'язний список) в послідовність біт

Призначення:

- збереження об'єктів (в т.ч. в файл);
- передача по мережі;
- в т.ч. здійснення віддалених викликів процедур (в іншому адресному просторі);
- поширення об'єктів, особливо в технологіях компонентно-орієнтованого програмування, таких як COM і CORBA;
- виявлення змін в даних, що змінюються з часом.

Серіалізация повинна бути архітектурно-незалежною.

Маршалинга - схоже з серіалізацією поняття (маються різночитання) + кодові бази і по-іншому з об'єктами. Часто під маршалинга розуміють власне передачу після серіалізації.

Проблема RPC:досягнення прозорості. (Виклику віддаленої функції як локальної).

Рішення: механізм заглушок і мови опису інтерфейсів.

IDL- мова опису інтерфейсів (англ. Interface Description Language або Interface Definition Language) - мова опису специфікацій інтерфейсів програмних компонентів. Містить набір прототипів процедур віддаленого виклику. (Синтаксично схожий на опис класів C ++ або Java).

Заглушка (stab) - об'єкт-посередник використовується для віддаленого виклику процедури відповідно до описаного інтерфейсом. Процедура, що здійснює сериализацию-десеріалізацію даних і передачу-прийом їх по мережі за допомогою звернень до RTL.

Характерні риси виклику віддалених процедур:

- асиметричність, тобто одна з взаємодіючих сторін є ініціатором (клієнт-серверна архітектура);

- синхронність, тобто виконання викликає процедури припиняється з моменту видачі запиту і відновлюється тільки після повернення з викликається процедури.

Взаємодія програмних компонентів при виконанні віддаленого виклику процедури показано на рис. 2.

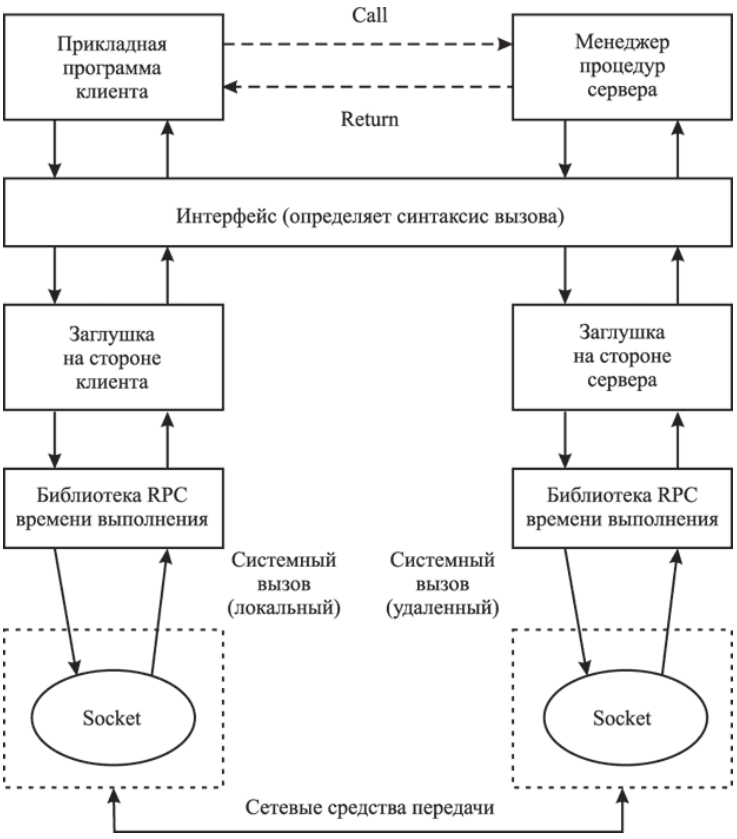

Рисунок 2. - Взаємодія програмних компонентів при виконанні віддаленого виклику процедури

Коли викликається процедура дійсно є віддаленою, замість локальної процедури використовується інша її версія, яка називається клієнтської заглушкою (stub). Подібно локальної процедурі, заглушка викликається з використанням того ж списку параметрів, тільки на відміну від локальної процедури вона відправляє викликає повідомлення на сервер. Зухвала повідомлення містить параметри процедури для процесу на сервері. Далі клієнтський процес очікує від сервера відповідного повідомлення. Процес на серверній стороні очікує надходження викликає повідомлення, дочекавшись якого, витягує параметри процедури, виконує необхідні обчислення і відправляє у відповідь повідомлення клієнту. Далі процес на серверній стороні чекає наступного викликає повідомлення. Процес на клієнтській стороні отримує відповідь було надіслане, витягує з нього результати обчислень і продовжує виконання.

На цьому рисунку зліва зображено клієнтський процес, що містить прикладну програму клієнта, клієнтську заглушку і бібліотеку RPC часу виконання. На серверній стороні (праворуч) показані менеджер процедур сервера (частина процесу сервера), серверна заглушка і бібліотека RPC часу виконання сервера. Пунктиром позначені віртуальні зв'язку, суцільними лініями - реальний маршрут даних.

Модель RPC передбачає активність тільки одного з процесів в кожен момент часу (синхронність).

Тимчасова діаграма виклику віддалених процедур (рис.3).

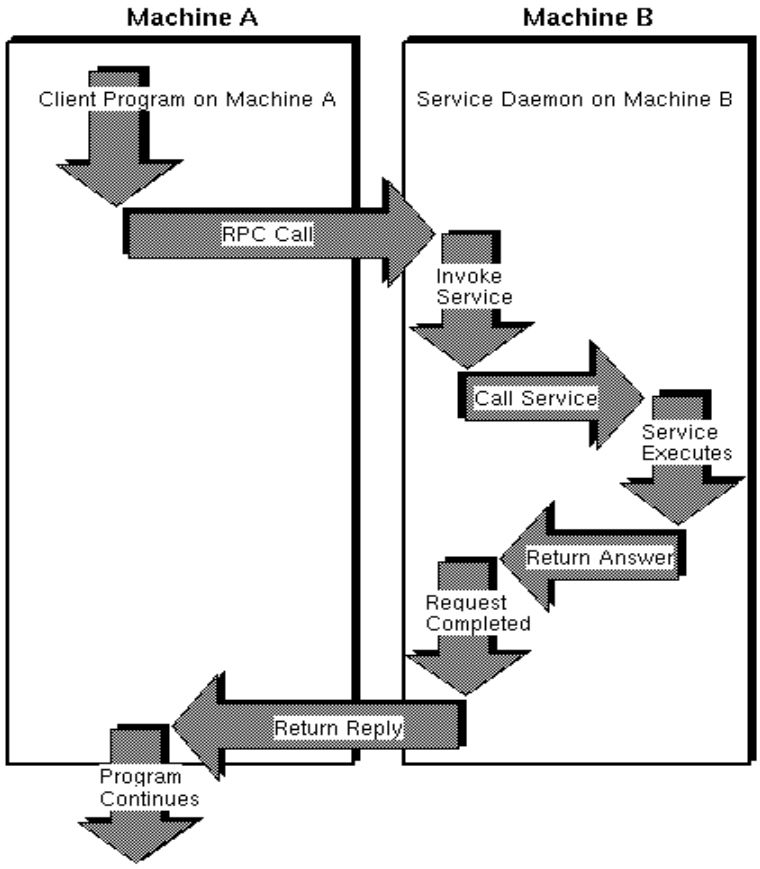

Рисунок 3. - Тимчасова діаграма виклику віддалених процедур

З вищевикладеного випливає, що перед виконанням компіляції і зв'язування RPC-програми необхідно вирішити дуже важливу попередню задачу. Це завдання полягає в створенні і компіляції файлу визначення інтерфейсів, в якому знаходиться опис клієнт-серверних інтерфейсів. Ці інтерфейси визначаються на спеціальній мові визначення інтерфейсів (Interface Definition Language, IDL) і містять набір <прототипів> для викликів віддалених процедур з боку клієнта, які реально повинні бути виконані на серверній стороні. Після створення такого файлу він компілюється спеціальним IDL-компілятором. Виходом цього компілятора є пара об'єктних файлів, один з яких відноситься до серверного модулю, а другий - до клієнтського. Вони містять коди клієнтських і серверних заглушок, де всі деталі віддаленого виконання процедур, передачі даних і т.д. ув'язані з бібліотекою RPC часу виконання.

Виклики віддалених процедур можуть бути реалізовані на будь-якій мові програмування, тому мова визначення інтерфейсів є стандартним для всіх мережевих платформ Ці файли надалі зв'язуються з результатом компіляції програм клієнтської і серверної сторін. Крім того, IDL-компілятор генерує заголовки для підключення його до вихідних файлів клієнта і сервера на етапі їх компіляції. Заголовки містить всі оголошення, отримані з визначень, наявних в IDL-файлі. Зокрема, там знаходиться глобальний унікальний ідентифікатор інтерфейсу (Global Unique Identifier, GUID), який використовується під час виконання для опису і прив'язки певних в RPC-додатку інтерфейсів.

Схема розробки розподіленого додатка за технологією RPC (рис.4).

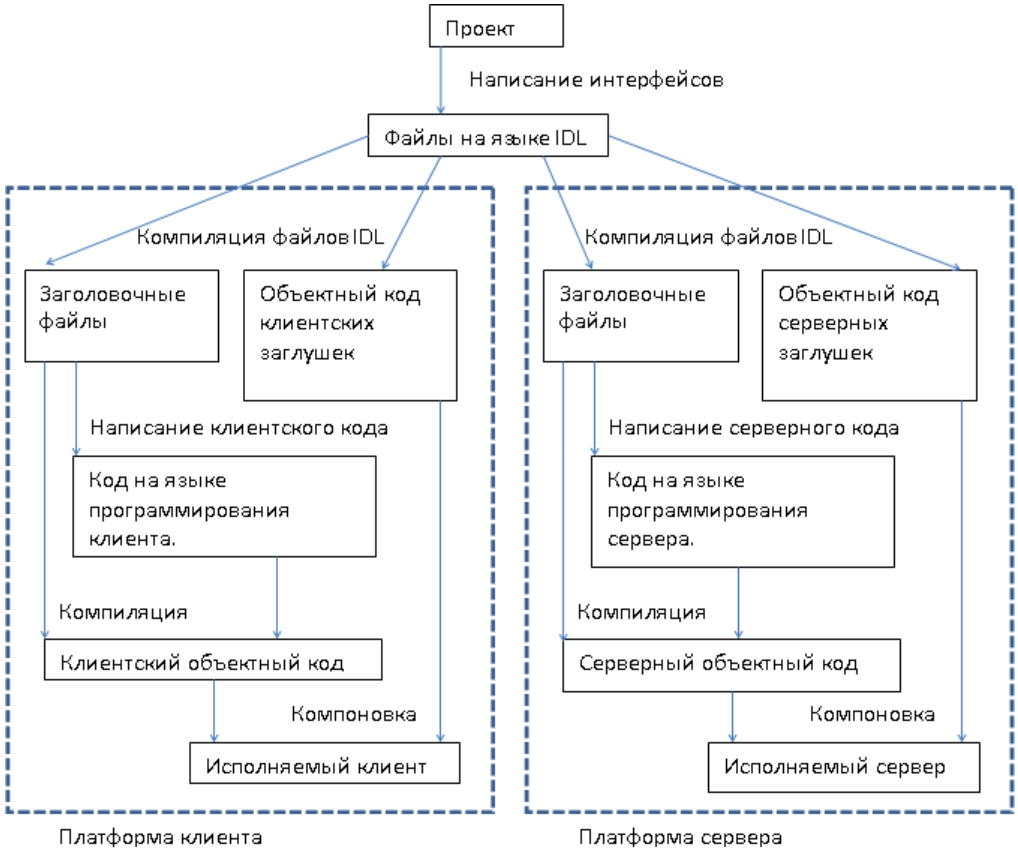

Рисунок 4. - Схема розробки розподіленого додатка за технологією RPC

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

Інтерфейс RPC не залежить від транспортних протоколів. Залежно від ситуації, на транспортному рівні можуть бути використані або протокол TCP, або протокол UDP. У загальному випадку, взаємодіючі процеси користуються механізмом сокетних з'єднань.

Реалізації RPC.

Різні реалізації RPC мають дуже відрізняється один від одного архітектуру і різняться в своїх можливостях :. На транспортному рівні RPC використовують в основному протоколи TCP і UDP, однак, деякі побудовані на основі HTTP (що порушує архітектуру ISO / OSI, так як HTTP спочатку не транспортний протокол).

CORBA Common Object Request Broker Architecture - загальна архітектура брокера об'єктних запитів - технологічний стандарт написання розподілених додатків, які просуваються консорціумом (робоча група) OMG (object management group) і відповідне йому інформаційна технологія.

Технологія CORBA створена для підтримки розробки та розгортання складних (розподілених) об'єктно-орієнтованих прикладних систем.

CORBA є механізмом в програмному забезпеченні для здійснення інтеграції ізольованих систем, який дає можливість програмам, написаним на різних мовах програмування, що працюють в різних вузлах мережі, взаємодіяти один з одним так само просто, як якщо б вони перебували в адресному просторі одного процесу.

RMI(англ. Remote Method Invocation) - програмний інтерфейс виклику віддалених методів в мові Java. Клієнт-сервер для конкретного запиту. Об'єкт-заглушка - представник серверного об'єкта. Метод заглушки здійснює маршалінга, відсилання, прийом, демаршалінг і т. Д.

SOAP (Від англ. Simple Object Access Protocol - простий протокол доступу до об'єктів) - протокол обміну структурованими повідомленнями в розподіленої обчислювальної середовищі.

Спочатку SOAP призначався в основному для реалізації віддаленого виклику процедур (RPC). Зараз протокол використовується для обміну довільними повідомленнями в форматі XML

SOAP може використовуватися з будь-яким протоколом прикладного рівня: SMTP, FTP, HTTP, HTTPS і ін. Однак його взаємодія з кожним із цих протоколів має свої особливості, які повинні бути визначені окремо. Найчастіше SOAP використовується поверх HTTP.

Повідомлення SOAP виглядає так:

SOAP-конверт

SOAP-заголовок

Елемент заголовка 1

Елемент заголовка 2

...
Елемент заголовка N
тіло SOAP
Елемент тіла N
...
Елемент тіла 2
Елемент тіла 1

Приклад SOAP-запиту на сервер інтернет-магазину:

<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>

Приклад відповіді:

<Soap: Envelope xmlns: soap = "http://schemas.xmlsoap.org/soap/envelope/">
<Soap: Body>
<GetProductDetailsResponse xmlns = "http://warehouse.example.com/ws">
<GetProductDetailsResult>
<ProductID> 12345 </ productID>
<ProductName> Стакан гранований </ productName>
<Description> Стакан гранований. 250 мл. </ Description>
<Price> 9.95 </ price>
<Currency>
<Code> 840 </ code>
<Alpha3> USD </ alpha3>
<Sign> $ </ sign>
<Name> US dollar </ name>
<Accuracy> 2 </ accuracy>
</ Currency>
<InStock> true </ inStock>
</ GetProductDetailsResult>
</ GetProductDetailsResponse>
</ Soap: Body>
</ Soap: Envelope>

Недоліки:
- повільно;
- стандарт не завжди строго підтримується;
- реалізація через протоколи прикладного рівня порушує ідеологію OSI.

Предок -XML RPC.
Підручник: http://www.w3.org/2002/07/soap-translation/russian/part0.html
SOAP відповідає архітектурі CORBA.
Передача даних - передача коду.

Розвиток технологій RPC у Microsoft.

OLE (16-bit Windows, 1990) (англ. Object Linking and Embedding) - технологія зв'язування та впровадження об'єктів в інші документи і об'єкти

COM(1993) - (англ. Component Object Model - об'єктна модель компонентів) -технологічний стандарт, призначений для створення програмного забезпечення на основі взаємодіючих компонентів, кожен з яких може використовуватися в багатьох програмах одночасно. Стандарт втілює в собі ідеї поліморфізму і інкапсуляції об'єктно-орієнтованого програмування. Компоненти - шматки функціоналу з інтерфейсами (через ВІНАП)

COM-95 (1995) Доопрацювання COM.

DCOM- (1997) Distributed COM. DCOM дозволяє COM-компонентів взаємодіяти один з одним по мережі. Мережевий рівень DCOM називається ORPC (Object RPC) і є об'єктно-орієнтованим розширенням DCE RPC. Перейменовано в ActiveX

COM + з Microsoft Transaction Server (MTS) (2000) Дополнітельніе готовіе послуги. COM + об'єднує компоненти в так звані додатки COM +, що спрощує адміністрування і обслуговування компонентів.

.NET Promoting- реалізація SOAP від Microsoft - компонент - API для взаємодії між процесами. (2002 .NET Framework 1.0) Виклики методів віддалених об'єктів навіть через гетерогенність мережі.

WCF(.NET Framework 3.0.) Windows Communication Foundation - програмний фреймворк, який використовується для обміну даними між додатками входять до складу .NET Framework. Інтероперабельність. Тобто каркас для взаємодії всього з усім. Служби WCF хостятся на процесах додатків, інших службах або IIS.

Реалізації IDL.

CORBA IDL - мова опису інтерфейсів розподілених об'єктів, розроблений робочою групою OMG. Створено в рамках узагальненої архітектури CORBA.

IDL DCE, мова опису інтерфейсів специфікації міжплатформного взаємодії служб, яку розробив консорціум Open Software Foundation (тепер The Open Group)

MIDL (Microsoft Interface Definition Language) - мова опису інтерфейсів для платформи Win32 визначає інтерфейс між клієнтом і сервером. Запропонована від Microsoft технологія використовує реєстр Windows і використовується для створення файлів і файлів конфігурації додатків (ACF), необхідних для дистанційного виклику процедури інтерфейсів (RPC) і COM / DCOM інтерфейсів.

COM IDL - мова опису інтерфейсів між модулями COM. Є наступником мови IDL в технології DCE (англ. З-поміж розподілених обчислень) - специфікації міжплатформного взаємодії служб, яку розробив консорціум Open Software Foundation (тепер The Open Group).

Поняття і скорочення: RPC SOAP CORBA IDL DCOM сериализация маршалинга заглушка (stub) big-endian little-endian middle-endian.

Питання для контролю та самостійної роботи

1. Модель виклику віддалених процедур.
2. Проблеми при виклику віддалених процедур.
3. Серіалізация - поняття і призначення.
4. Порядок запису байтів - види.
5. Стандарт SOAP - призначення, можливості, недоліки, приклади.

ЛЕКЦІЯ 3.
SOA (частина 1)

Відкриті інформаційні системи

Однією з головних тенденцій сучасної індустрії інформатики є створення відкритих систем.

Властивість відкритості означає:

1 перенесення (мобільність) ПО на різні апаратні платформи,

2 пристосованість системи до її модифікаціям (модифицируемость або власне відкритість)

3 пристосованість системи до комплексированию з іншими системами з метою розширення її функціональних можливостей і (або) надання системі нових якостей (інтегрованість).

Перехід до відкритих інформаційних систем дозволяє істотно прискорити побудову ІС в результаті заміни тривалої і дорогої розробки нових систем по повному циклу їх компонуванням з раніше спроектованих підсистем або швидкої модернізацією вже існуючих систем (реінжиніринг).

Відкритість передбачає виділення в системі інтерфейсної частини (входів і виходів), що забезпечує сполучення з іншими системами або підсистемами, причому для комплексування досить мати дані тільки про інтерфейсних частинах сполучених об'єктів. Якщо ж інтерфейсні частини виконані відповідно до заздалегідь обумовленими правилами і угодами, яких повинні дотримуватися всі творці відкритих систем певної програми, то проблема створення нових складних систем значно спрощується. З цього випливає, що основою створення відкритих систем є стандартизація та уніфікація в області інформаційних технологій.

Значного розвитку концепція відкритості отримала в області побудови обчислювальних мереж, що знайшло відображення в еталонної моделі взаємозв'язку відкритих систем, підтримуваної низкою міжнародних стандартів. Ідеї відкритості широко використовуються при побудові програмного, інформаційного і лінгвістичного забезпечень ІС; в результаті підвищується ступінь універсальності програм і розширюються можливості їх адаптації до конкретних умов.

Аспекти відкритості відображені в стандартизації:

1 API {Application Program Interface) - інтерфейсів прикладних програм з операційним оточенням, в тому числі системних викликів і утиліт операційної системи (ОС), тобто зв'язків з ОС;

2 міжпрограмного інтерфейсу, включаючи мови програмування;

3 мережевої взаємодії;

4 призначеного для користувача інтерфейсу, в тому числі коштів графічного взаємодії користувача з ЕОМ;

5 засобів захисту інформації.

Стандарти, що забезпечують відкритість ПО, в даний час розробляються такими організаціями, як ISO (International Standard Organization), IEEE (Institute of Electrical and Electronics Engineers), EIA (Electronics Industries Association) і ін.

Серед інших стандартів, що сприяють відкритості ПО ІС, слід зазначити стандарти графічного призначеного для користувача інтерфейсу, зберігання і передачі графічних даних, побудови баз даних і файлових систем, супроводу і управління конфігурацією програмних систем і ін.

Важливе значення для створення відкритих систем мають уніфікація і стандартизація засобів міжпрограмного інтерфейсу, або, іншими словами, необхідна наявність профілів ІС для інформаційної взаємодії програм, що входять в ІС. Профілем відкритої системи називають сукупність стандартів та інших нормативних документів, які забезпечують виконання системою заданих функцій.

Так, в профілях ІС можуть фігурувати уніфікована мова SQL обміну даними між різними СУБД, стандарти мережевої взаємодії і т. П.

Мабрук - Короткі основи SOA
(*http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/)*

Введення в SOA

SOA - це архітектурний підхід до визначення, зв'язування і інтеграції повторно використовуваних бізнес-сервісів, що мають чіткі межі і самодостатніх за своєю функціональністю. В рамках такої архітектури можна організовувати бізнес-сервіси в бізнес-процеси. Впроваджуючи концепцію сервісів (більш високого рівня абстракції, що не залежить від додатків і платформи інформаційної інфраструктури, а також від контексту або інших сервісів), SOA переносить інформаційні технології на наступний рівень, більш відповідний для забезпечення функціональної сумісності та реалізації в гетерогенних середовищах.

Оскільки SOA грунтується на стандартах (таких, наприклад, як Web-сервіси), затверджених і підтримуваних основними постачальниками інформаційних технологій, сервіси можна швидко створювати і об'єднувати. Можна організувати взаємодію підприємств незалежно від їх інфраструктури, що забезпечує делегування, спільне застосування, повторне використання та максимальну ефективність існуючих активів.

При впровадженні SOA ви переводите внутрішню інформаційну інфраструктуру на більш високий, більш відкритий і керований рівень. З появою повторно використовуваних сервісів і високорівневих процесів внесення змін стає простіше, ніж коли б то не було, і починає більше бути схожим на розбирання і складання частин (сервісів) в нові процеси, спрямовані на здійснення бізнес-діяльності. Це не тільки сприяє підвищенню ефективності і повторного використання, а й дає можливість змінювати і підлаштовувати інформаційні технології під мінливі бізнес-вимоги.

Що найкраще підходить для SOA?

Існують певні ситуації і бізнес-функції, коли слід негайно звернутися до SOA, оскільки ця архітектура може істотно підвищити конкурентоспроможність і продуктивність і чітко проявити свої переваги. До таких ситуацій головним чином відносяться:

1 Централізовані бізнес-функції, що використовуються декількома суб'єктами. SOA допомагає ідентифікувати ці функції і зібрати їх в повторно використовувані самодостатні сервіси, які не піддаються впливу змін в процесах, їх використовують.

2 Інтеграція з партнерами. SOA сприяє застосуванню стандартів, створюють єдині критерії для роботи всіх зацікавлених сторін. Крім того, забезпечується архітектурою SOA гнучкість покращує процес інтеграції завдяки можливості підключати, змінювати і оновлювати сервіси практично непомітно для ваших клієнтів.

3 Наявність працюючих старих технологій. Деякі організації не бажають відмовлятися від перевірених і надійних старих технологій. Питання безпеки роблять деяких користувачів, особливо в сфері банківського обслуговування, недовірливими до нових програмних систем і їх недослідженим вразливостей. У таких ситуаціях SOA може допомогти одягнутися старі технології в нові стандарти, відобразити їх у заснованої на стандартах середовищі і зробити придатними для інтеграції і повторного використання.

Чим обумовлена швидке налаштування під мінливі бізнес-вимоги?

Через неминучості змін єдиним гарантом забезпечення безперервності бізнес-діяльності є здатність адаптуватися до змін і бути готовим до них (рухливість бізнесу - agility). SOA забезпечує можливість адаптації до бізнес-вимогам (що має вирішальне значення для майбутнього будь-якої діяльності), завдяки наступним факторам:

Слабке зв'язування

1. Усуває жорсткі зв'язку, що перешкоджають змінам.

2. Менше вкладень в реалізацію і більше в повторне використання.

3. Покращує можливості віддаленого доступу до оригінальних джерел інформації, зменшуючи затримки і залежності.

4. Проекти по інтеграції управляються бізнес-вимогами (тобто бізнес-діяльність є основною рушійною силою).

5. Завдяки відображенню і спільного використання інформації, слабке зв'язування дозволяє компаніям отримувати в режимі реального часу більше даних про ефективність бізнес-діяльності.

6. Полегшує партнерам взаємодія з вашою компанією.

7. Сприяє просуванню і публікації ваших сервісів, полегшуючи клієнтам виявлення їх і вашої компанії.

8. Полегшує пошук нових партнерів і сервісів, допомагаючи знайти більш підходить під ваші вимоги сервіс.

Повторне використання

1. Робить процеси більш узгодженими, оскільки вони базуються на одних і тих же компонентах.

2. Сприяє підвищенню якості завдяки конкуренції між провайдерами сервісів.

3. Дозволяє змінювати систему незалежно від змін бізнес-діяльності.

4. Зменшує вплив змін, оскільки вони виконуються централізовано і охоплюють всі беруть участь сторони.

Можливість розширення

1. Робить SOA-рішення доступними організаціям будь-якого розміру.

2. Змінює процес розробки на більш динамічний, більш відповідний для ведення бізнес-діяльності.

3. Прискорює злиття і поглинання.

В яких випадках застосування SOA не обгрунтовано

Архітектура SOA приносить користь бізнес-організаціям практично у всіх ситуаціях. Однак в дуже спеціалізованих випадках вона може перешкодити поліпшенню бізнес-діяльності. До таких ситуацій належать:

1 Коли інформаційне середовище гомогенна. Якщо організація використовує комплекс узгоджених продуктів (що належать, наприклад, одному виробнику), SOA може виявитися перешкодою, а не корисною стратегією.

2 Коли критична продуктивність в режимі реального часу. В силу слабкого зв'язування між різними споживачами та виробниками архітектура SOA залежить від протоколів взаємодії, які за своєю природою є повільними. Вона також схильна застосовувати логіку посередництва і асинхронні протоколи, які не підходять для ефективної роботи в режимі реального часу.

3 Коли нічого не змінюється. Якщо споживач не бачить змін в бізнес-логікою, поданні, потоці даних, процесі або будь-яких інших аспектах додатки, перетворення старих систем в SOA може не виправдати витрачених зусиль.

4 Коли тісне зв'язування не є недоліком. Слабке зв'язування приносить користь, коли воно використовується з компонентом, який вами не управляється і зміни якого ви, отже, не можете контролювати. З іншого боку, коли компонент ваш і знаходиться під вашим контролем, слабке зв'язування може зажадати додаткових накладних витрат, особливо якщо компонент не є повторно використовуваним.

5 Сервіс - це функція, яка є чітко визначеною, самодостатньою і не залежить від контексту або стану інших сервісів.

coupling cohesion

провайдер - реєстр - споживач

Віртуалізація сервісу (Service Mediation).

Поділ сервісу відбувається за допомогою механізму віртуалізації. Віртуальний сервіс є проксі-об'єктом для реального сервісу. Проксі-сервіс являє собою бажаний споживачем сервісу інтерфейс. Споживачі звертаються до проксі-сервісу, який передає повідомлення до дійсного сервісу.

Віртуалізація сервісу забезпечує гнучкість, необхідну при впровадженні SOA. Ця гнучкість заснована на факті, що віртуальний сервіс розділяє постачальника і споживача в термінах розташування, передачі даних і повідомлень.

Незалежність розташування.

Віртуальний сервіс дозволяє приховати дійсне місце розташування сервісу від споживачів. Це дає свободу переміщати реалізацію сервісу без

повідомлення споживачів. Наприклад, ви можете перемістити сервіс на сервера більшої потужності для збільшення продуктивності.

Незалежність передачі даних.

Віртуалізація сервісу дозволяє постачати сервіс декількома засобами передачі даних. Припустимо, ви створили сервіс «CreateOrder», доступний через JMS (Java Message Service). Сервіс став популярний і деякі користувачі бажають розширити функціональність своїх додатків даним сервісом. Складність в тому, що вони можуть використовувати HTTP-протокол. Звичайно потрібно створити іншу реалізацію сервісу "CreateOrder" для підтримки HTTP, але можливості віртуальних сервісів дозволяє створити віртуальний HTTP-сервіс без зміни реалізації. Це прозоро вирішує проблему взаємодії і дозволяє розширювати число користувачів сервісу.

Незалежність повідомлень.

Іноді споживачі сервісу не синхронізовані з постачальниками в сенсі очікуваних сервісом XML-повідомлень. У таких ситуаціях віртуалізація сервісу пропонує трансформувати сполучення між форматами постачальника і споживача. Подібний ефект може бути отриманий, наприклад, при введенні в експлуатацію нової версії сервісу та зміні XML-схем, що визначають параметри повідомлень. Передбачається, що споживачі сервісу повинні завжди дотримуватися очікуваний постачальником формат. Але, при змінах, складно змусити всіх споживачів миттєво пристосуватися.

Типові функції віртуального сервісу

Віртуальний сервіс - найкраще місце реалізації деяких технічних умов або забезпечення якості сервісу (QualityOfService): QoS - ймовірність / контроль за - дотриманням контракту сервісу на різних рівнях.

1. Перевірка XML повідомлень на коректність формату і відповідність інтерфейсу сервісу.
2. Аутентифікація і авторизація: ідентифікація споживача сервісу та перевірка наявності у нього прав для виклику сервісу.
3. Розшифровка повідомлень і перевірка підпису.
4. Балансування навантаження і гарантії наявності ресурсів для роботи сервісу.
5. Маршрутизація повідомлень. Передача повідомлень різним реалізаціям сервісу в залежності від вмісту повідомлень або зовнішніх умов.
6. Моніторинг роботи сервісу, продуктивності, а також перевірка надання постачальникам необхідних послуг (SLA).

Дані вимоги змінюються багато частіше, ніж функціональна логіка сервісів.

Механізм віртуалізації дозволяє реалізувати різну якість сервісу (QoS) для різних клієнтів, наприклад, HTTP-аутентифікацію всередині підприємства і передачу шифрованого XML для зовнішніх клієнтів.

Стандарти

– XML - розширювана мова розмітки, призначений для зберігання і передачі структурованих даних і його розширення:

– WSDL - мова опису зовнішніх інтерфейсів WS на базі XML

– SOAP - протокол обміну повідомленнями на базі XML

– UDDI - універсальний інтерфейс розпізнавання, опису та інтеграції. Каталог веб-служб і відомості про компанії, що їх забезпечують

http://www.webservicelist.com/

WDSL

WSDL (англ. Web Services Description Language) - мова опису веб-сервісів і доступу до них, заснований на мові XML.

Остання офіційна специфікація версія 2.0 (WSDL Version 2.0 від 26 червня 2007 року).

Кожен документ WSDL можна розбити на наступні логічні частини:

– визначення типів даних (types) - визначення виду відправлених і отриманих сервісом XML повідомлень

– елементи даних (message) - повідомлення, що використовуються web-сервісом

– абстрактні операції (portType) - список операцій, які можуть бути виконані з повідомленнями

– зв'язування сервісів (binding) - спосіб, яким повідомлення буде доставлено

```
<message name="GetTermRequest">
 <part name="Term" type="Xs: string"/>
</ Message>

<message name="GetTermResponse">
 <part name="Value" type="Xs: string"/>
</ Message>

<portType name="GlossaryTerms">
 <operation name="GetTerm">
 <input message="GetTermRequest"/>
 <output message="GetTermResponse"/>
 </ Operation>
</ PortType>
```

SOAP

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>
```

Альтернатива - JSON

```
{
 "FirstName": "Іван",
 "LastName": "Іванов",
 "Address": {
 "StreetAddress": "Московське ш., 101, кв.101",
 "City": "Ленінград",
 "PostalCode": 101101
 },
 "PhoneNumbers": [
 "812 123-1234",
 "916 123-4567"
 ]
}
```

JSON (англ. JavaScript Object Notation) - текстовий формат обміну даними, заснований на JavaScript і часто використовується саме з цією мовою. Як і багато інших текстові формати, JSON легко читається людьми.

Джерела:

Виломова Е. А. - Проектування та експлуатація інформаційних систем в медіаіндустрії

Норенков - Основи автоматизованого проектування, 34-36.

Мабрук - Короткі основи SOA (http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/). Секція 2-4 (Секція 4 до «XML в SOA»):

http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/section2.html

http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/section3.html

http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/section4.html

SOA Adoption for Dummies (Англомовний), 29 - 31.

https://docs.google.com/document/d/1LsI3lK4ZZjTsU3xbFyP4Mv8vzZXzkhbAh-9seHD-XUo/edit?hl=en&pli=1 WSDL-UDDI-SOAP

UDDI (англ. Universal Description Discovery & Integration) - інструмент для розташування описів веб-сервісів (WSDL) для подальшого їх пошуку іншими організаціями та інтеграції в свої системи.

ЛЕКЦІЯ 4.
Сервіс-орієнтовані архітектури (частина 2)

http://citforum.ru/internet/webservice/soa/

Шаблон функціонального дизайну (одна частина виконує одну задачу), далі - по ходу зменшення зв'язності.

Процедурна парадигма - підпрограми - на рівні кодування.

Модульність - на рівні компонування - збірка в програму лінковщік - різні мови.

Спільні бібліотеки (Бовтаються в системі окремо) (зв'язування - на рівні виконання).

Компоненти-фреймворки - компоненти існують незалежно від програми (програма вироджується до логіки зв'язку компонентів).

SOA - гетерогенність і кроссплатформенность (протиставлення монолітної архітектури).

Парадигми (зменшення зв'язності coupling, збільшення зачеплення).

Лінійна послідовна

коди-асемблер-високий рівень (фортран, алгол - кінець 50-х)

проблема - спагетті, багато goto

Структурна блокова

(Запропоновано в 70-х роках XX століття Е. Дейкстрой, розроблена і доповнена Н. Віртом.)

Будь-яка програма являє собою структуру, побудовану з трьох типів базових конструкцій:

– послідовне виконання - одноразове виконання операцій в тому порядку, в якому вони записані в тексті програми;

– розгалуження - одноразове виконання однієї з двох або більше операцій, в залежності від виконання деякого заданого умови;

– цикл - багаторазове виконання однієї і тієї ж операції до тих пір, поки виконується деяка задана умова (умова продовження циклу).

У програмі базові конструкції можуть бути вкладені одна в одну довільним чином, але ніяких інших засобів управління послідовністю виконання операцій не передбачається.

Теорема Бома-Якопіні

Будь-яку схему алгоритму можна представити у вигляді композиції вкладених блоків begin і end, умовних операторів if, then, else, циклів з передумовою (while) і може бути додаткових логічних змінних (прапорів).

Ця теорема була сформульована італійськими математиками К. Бомом і Дж. Якопіні в 1966 році і каже нам про те, як можна уникнути використання оператора переходу goto.

Операторні дужки.

Проблема - змінні.

Процедурна - неодноразове використання блоків. заглушки (в одному флаконі з блочною) послід розробки - зверху вниз. Відмирання блок-схем.

Повторювані фрагменти програми (або не повторюються, але представляють собою логічно цілісні обчислювальні блоки) можуть оформлятися у вигляді т. Зв. підпрограм (процедур або функцій). В цьому випадку в тексті основної програми, замість поміщеного в підпрограму фрагмента, вставляється інструкція виклику підпрограми. При виконанні такої інструкції виконується викликана підпрограма, після чого виконання програми триває з інструкції, наступної за командою виклику підпрограми.

Розробка програми ведеться покроково, методом «зверху вниз».

Спочатку пишеться текст основної програми, в якому, замість кожного зв'язкового логічного фрагмента тексту, вставляється виклик підпрограми, яка буде виконувати цей фрагмент. Замість справжніх, які працюють підпрограм, в програму вставляються «заглушки», які нічого не роблять. Отримана програма перевіряється та налагоджували. Після того, як програміст переконається, що підпрограми викликаються в правильній послідовності (тобто загальна структура програми вірна), підпрограми-заглушки послідовно замінюються на реально працюють, причому розробка кожної підпрограми ведеться тим же методом, що і основний програми. Розробка закінчується тоді, коли не залишиться жодної «затички», яка не була б вилучена. Така послідовність гарантує, що на кожному етапі розробки програміст одночасно має справу з доступним для огляду і зрозумілим йому безліччю фрагментів, і може бути впевнений, що загальна структура всіх вищих рівнів програми вірна. При супроводі та внесення змін до програми з'ясовується, в які саме процедури потрібно внести зміни, і вони вносяться, не зачіпаючи частини програми, які безпосередньо не пов'язані з ними. Це дозволяє гарантувати, що при внесенні змін і виправлення помилок не вийде з ладу якась частина програми, яка перебуває в даний момент поза зоною уваги програміста. безпосередньо не пов'язані з ними. Це дозволяє гарантувати, що при внесенні змін і виправлення помилок не вийде з ладу якась частина програми, яка перебуває в даний момент поза зоною уваги програміста. безпосередньо не пов'язані з ними. Це дозволяє гарантувати, що при внесенні змін і виправлення помилок не вийде з ладу якась частина програми, яка перебуває в даний момент поза зоною уваги програміста.

Проблема - занадто багато параметрів. Рішення - структури.

Модульна - роздільна компіляція, збірка, різні мови, бібліотеки - фізична поділ.

ООП (Паралельно з модульної) - інкапсуляція, успадкування, поліморфізм (пізніше зв'язування) - логічне поділ і розмежування доступу

Проблема - тендітні базові класи. Рішення – агрегація.

Компонентна - динамічне підключення.

У 1987 р Ніклаус Вірт, уніфікувавши, запропонував для мови Оберон патерн написання блоків. Блок, що задовольняє вимогам цього патерну, називається компонентом. Даний патерн сформувався при вивченні проблеми тендітних базових

класів, що виникає при побудові об'ємної ієрархії класів. Патерн полягав в тому, що компонент компілюється окремо від інших, а на стадії виконання необхідні компоненти підключаються динамічно.

КОП = ООП

+ Модульність (включаючи упрятиваніе інформації і пізніше зв'язування модулів, тобто можливість довантажувати необхідні модулі в процесі виконання програми, а не заздалегідь, як це зазвичай робиться в старих системах програмування);

+ Безпеку (статичний контроль типів змінних і автоматичне керування пам'яттю);

- успадкування реалізації через кордони модулів (тільки від абстракцій);

- фреймворки.

Сервіс-орієнтована архітектура (гетерогенність, интероперабельность)

(SOA, англ. Service-oriented architecture) - модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов'язаних (англ. Loose coupling) замінних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами.

Програмні комплекси, розроблені відповідно до сервіс-орієнтованою архітектурою, зазвичай реалізуються як набір веб-служб, взаємодіючих по протоколу SOAP, але існують і інші реалізації (наприклад, на базі jini, CORBA, на основі REST).

CORBA (OMG, з 1989) гетерогенність - нішевий продукт, архітектури, стандарти, IDL итд Все вийшло просто і невимушено на практичних завданнях в веб-сервісах - HTTP - SOAP і ін XML-розширення.

Інтерфейси компонентів в сервіс-орієнтованої архітектури інкапсулюють деталі реалізації (операційну систему, платформу, мова програмування) від інших компонентів, таким чином забезпечуючи комбінування і багаторазове використання компонентів для побудови складних розподілених програмних комплексів, забезпечуючи незалежність від використовуваних платформ та інструментів розробки, сприяючи масштабованості і керованості створюваних систем.

SOA - це не технологія, а спосіб проектування і організації інформаційної архітектури та бізнес-функціональності.

SOA: це парадигма, призначена для проектування, розробки та управління дискретних одиниць логіки (сервісів) в обчислювальному середовищі. Застосування цього підходу вимагає від розробників проектування додатків як набору сервісів, навіть якщо переваги такого рішення відразу неочевидні. Розробники повинні "вийти за межі" своїх додатків і подумати, як скористатися вже існуючими сервісами, або вивчити, як їх сервіси можуть бути використані їх колегами.

SOA "підштовхує" до використання альтернативних технологій і підходів (таких як обмін повідомленнями) для побудови додатків за допомогою зв'язування сервісів, а не за допомогою написання нового програмного коду. В цьому випадку, при належному проектуванні, застосування повідомлень дозволяє компаніям своєчасно реагувати на зміну ринкових умов - "налаштовувати" процес обміну повідомленнями, а не розробляти нові програми.

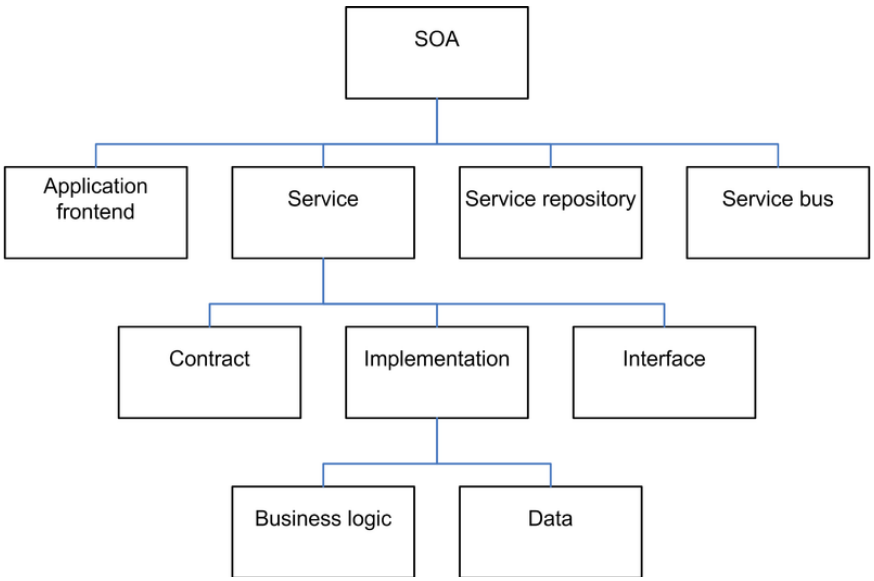

Рисунок 5. – Структура SOA

Standardized servie contract: Services adhere to a communications agreement, as defined collectively by one or more service-description documents.

Service loose coupling: Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.

Service abstraction: Beyond descriptions in the service contract, services hide logic from the outside world.

Service reusability: Logic is divided into services with the intention of promoting reuse.

Service autonomy: Services have control over the logic they encapsulate.

Service statelessness: Services minimize resource consumption by deferring the management of state information when necessary

Service discoverability: Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.

Service composability: Services are effective composition participants, regardless of the size and complexity of the composition.

Some authors also include the following principles:

Service granularity: A design consideration to provide optimal scope and right granular level of the business functionality in a service operation.

Service normalization: Services are decomposed and / or consolidated to a level of normal form to minimize redundancy. In some cases, services are denormalized for specific purposes, such as performance optimization, access, and aggregation.

Service optimization: All else equal, high-quality services are generally preferable to low-quality ones.

Service relevance: Functionality is presented at a granularity recognized by the user as a meaningful service.

Service encapsulation: Many services are consolidated for use under the SOA. Often such services were not planned to be under SOA.

Service location transparency: This refers to the ability of a service consumer to invoke a service regardless of its actual location in the network. This also recognizes the discoverability property (one of the core principle of SOA) and the right of a consumer to access the service. Often, the idea of service virtualization also relates to location tra-

nsparency. This is where the consumer simply calls a logical service while a suitable SOA-enabling runtime infrastructure component, commonly a service bus, maps this logical service call to a physical service.

SOA - це термін, який з'явився для опису виконуваних компонентів - таких як Web-сервіси - які можуть викликатися іншими програмами, які виступають в якості клієнтів або споживачів цих сервісів.

У найзагальнішому вигляді SOA припускає наявність трьох основних учасників: постачальника сервісу, споживача сервісу та реєстру сервісів (див. рис. 6). Взаємодія учасників виглядає досить просто: постачальник сервісу реєструє свої сервіси в реєстрі, а споживач звертається до реєстру із запитом.

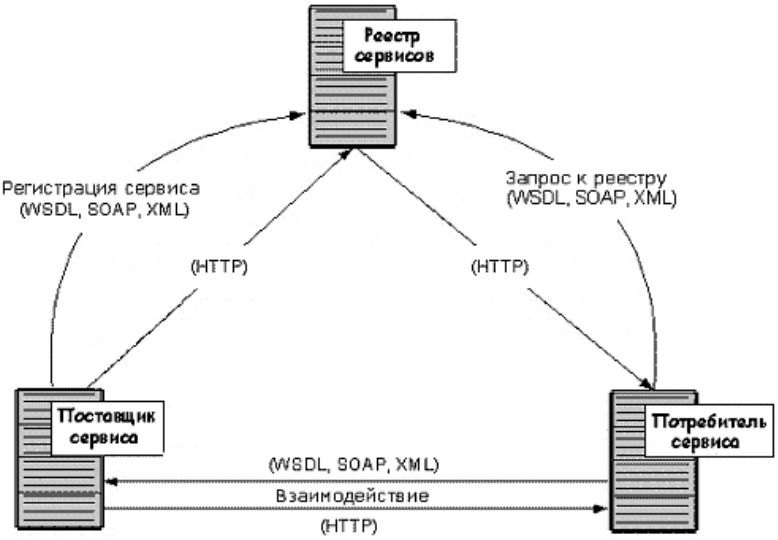

Рисунок 6. - Загальний вигляд SOA

Для використання сервісу необхідно дотримуватися угоди про інтерфейс для звернення до сервісу - інтерфейс повинен не залежати від платформи. SOA реалізує масштабованість сервісів - можливість додавання сервісів, а також їх модернізацію. Постачальник сервісу і його споживач виявляються непов'язаними - вони спілкуються за допомогою повідомлень. Оскільки інтерфейс повинен не залежати від платформи, то і технологія, яка використовується для визначення повідомлень, також повинна не залежати від платформи. Тому, як правило, повідомлення є XML-документами, які відповідають XML-схемі.

Будь-яка розмова про SOA мимоволі переходить на міркування про роль і місце Web-сервісів. Незважаючи на те, що основні положення SOA склалися задовго до появи Web-сервісів, сьогодні Web-сервіси займають центральне місце в SOA (див. рис. 6). Так, Джером Уестерман зазначає, що використання XML і Web-сервісів "піднімає SOA на більш високий рівень".

Дійсно, відкриті стандарти, що описують XML і Web-сервіси, дозволяють застосовувати SOA до всіх технологій і додатків, встановленим в компанії. Як відомо, Web-сервіси базуються на широко поширених і відкритих протоколах: HTTP, XML, UDDI, WSDL і SOAP. Саме ці стандарти реалізують основні вимоги SOA - по-перше, сервіс повинен піддаватися динамічному виявлення і викликом (UDDI, WSDL і SOAP), по-друге, повинен використовуватися незалежний від платформи інтерфейс (XML). Нарешті, HTTP забезпечує функціональну сумісність.

(Проблема OSI застаріла. HTTP - протокол рівня додатка використовується як транспорт).

Переваги використання SOA

Перш ніж, перерахувати переваги використання SOA, буде доречним нагадати, що переваги бувають різні: стратегічні і тактичні. SOA має ряд переваг як стратегічних, так і тактичних.

Стратегічна цінність SOA:

- Скорочення часу реалізації проектів, або "часу виходу на ринок".
- Підвищення продуктивності.
- Більш швидка і менш дорога інтеграція додатків і інтеграція B2B (Business to Business) - зупинимося більш детально на цьому пункті.

Відомо, що реалізація традиційних рішень для інтеграції прикладних програм - непросте завдання, що вимагає істотних капіталовкладень. Крім того, часто при впровадження необхідно написання програмного коду. SOA передбачає розміщення сервісів в мережі в режимі виконання, тобто дозволяє автоматизувати ці ресурсомісткі процеси, завдяки чому істотно скорочуються всі витрати на інтеграцію.

Тактичні переваги SOA:

- Простіші розробка і впровадження програм.
- Використання поточних інвестицій.
- Зменшення ризику, пов'язаного з впровадженням проектів в області автоматизацією послуг і процесів.
- Можливість безперервного поліпшення наданої послуги.
- Скорочення числа звернень до служби технічної підтримки.
- Підвищення показника повернення інвестицій (ROI).

В ідеалі - програмування діаграммкамі мишкою, зв'язування сервісів в бізнес логіку і відразу впровадження без програмування (кодування)

Web-сервіси - це автономні, модульні програми, призначені для реалізації бізнес-процесів. Web-сервіси спираються на ряд галузевих стандартів: WSDL (для опису), UDDI (для інформування та публікації) і SOAP (для обміну повідомленнями). Ці специфікації не залежить від платформи і мови, завдяки чому користувачі можуть пов'язувати різні компоненти з різних організаційних структур.

Питання для контролю та самостійної роботи

- Основні поняття архітектури сервіс - орієнтованого типу.
- Канони і принципи службово - орієнтованої методології.
- Конфігурація і хостинг сервісів WCF
- Сервіс - орієнтовані додатки, як пріложеніяобмена повідомленнями (messagingapplications).
- Надання метаданих -MetadataExchange (MEX)
- Проектування і визначення контрактів
- Робота з каналами. Управління паралельної розробкою.
- Кінцеві точки (EndPoints) і поведінки (Bindings)
- Обробка помилок, збоїв і винятків
- Управління екземплярами (Instance Management)
- Стандарти SO програмування.

ЛЕКЦІЯ 5.
Архітектури розподілених систем REST, SOA, CRUD, RPC.

RPC - сериализация / маршалинга - заглушки - IDL

REST (Representational State Transfer, «передача уявлень станів») - стиль побудови архітектури розподіленого додатка. Був описаний і популяризував 2000 року Роєм Філдінгом (Roy Fielding), одним із творців протоколу HTTP.

Найвідомішою системою, побудованої в значній мірі по архітектурі REST, є сучасна Всесвітня павутина.

Дані в REST повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад HTML, XML, JSON). Мережевий протокол (як і HTTP) повинен підтримувати кешування, повинен бути незалежним від мережевого шару, не повинен зберігати інформацію про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабованість системи і дозволяє їй еволюціонувати з новими вимогами:

- Клієнт-сервер
- Запит-відповідь
- Ресурс-передача
- Ніяких проміжних станів або з'єднань (куки у клієнта)
- Звернення до ресурсів, замість виклику методів.
- Зазвичай прив'язаний до хттп.
- RESTful - ресурсно-орієнтована.

Антиподом REST є підхід, заснований на виклик віддалених процедур (Remote Procedure Call - RPC). Підхід RPC дозволяє використовувати невелику кількість мережевих ресурсів з великою кількістю методів і складним протоколом. При підході REST кількість методів і складність протоколу строго обмежені, через що кількість окремих ресурсів може бути великим.

HTTP - CRUD (create read update delete)
get - r
post - cud
put - cu
delete - d

Зазвичай запит, в якому ресурс - хоч в тому ж XML:

http://blog.meta-systems.com.ua/2009/10/rest-crud-rpc.html

REST і CRUD проти RPC

Розвиток мережевих комунікацій почалося зі спеціалізованих протоколів, заточених під прикладні потреби і повторюють функціональність один одного в багатьох випадках. Концепція ця називається RPC (Remote Procedure Call) або виклик віддалених процедур. Характеризується RPC тим, що функціональність програми повністю відображена в наборі команд протоколу, наприклад: FTP, POP, NNTP, протоколи СУБД, не кажучи вже про численні протоколи систем

промислової автоматизації або прикладних систем. Пізніші RPC стали абстрактними, тобто з'явився якийсь стандарт, що описує протокол, а вже на його основі розроблялися прикладні служби, що володіють конкретним набором команд, наприклад: SunRPC, DCOM, CORBA, Java RMI, і т.д.

Наступний крок полягав в обмеженні набору команд до "універсального мінімуму" і перенесення всієї смислового навантаження в параметри викликів. Підхід цей отримав назву CRUD за першими літерами команд (create, read, update and delete), а архітектура інформаційних систем, побудована на базі підходу - REST (Representational State Transfer). Клеінт-серверні додатки в CRUD / REST архітектурі працюють ресурсом і викликом, як основними абстракціями. Ресурс (найчастіше файл) - це відповідь сервера, що містить дані і поточний стан. Стан - це набір параметрів ресурсу, які змінюються в часі. Запит призводить до передачі ресурсу на клієнтську сторону, де користувач з ним працює і при наступному запиті відправляє ресурс назад на сервер разом зі станом.

Таким чином, архітектура REST повністю відкидає можливість встановлення сесії, а отже не дає можливості зберігання стану на стороні сервера. На відміну від REST, сервера RPC ставлять у відповідність певну область пам'яті сервера з кожним підключеного користувачем, створюють ідентифікатор сесії і встановлюють з'єднання: віртуальне (ідентифікатора) або постійне (не закриваючи сокет між викликами).

Перевагою crud / REST архітектури є просте масштабування на кластери, можливість застосування хмарних обчислень, простота застосування для інформаційних, гіпертекстових і інших типових задач. З недоліків можна привести: неможливість створення систем реального часу, надмірність трафіку, відсутність машини станів, як необхідного засобу при розробці складних систем. Перевагою ж RPC архітектури є відсутність обмежень на набір команд, встановлення з'єднань і зберігання стану на сервері, внаслідок чого, RPC досі є єдиним засобом розробки прикладних, промислових і інших складних систем. А очевидним недоліком RPC є складність розробки, налагодження і впровадження, що природно при більшій гнучкості.

RPC = SOAP
Разом: SOAP
Плюси:
- Незалежні від мови, платформи і транспорту.
- Розроблено для розподілених систем.
- Стандартизований, документований.
- Розвинені інструменти розробки.
- В протоколі визначено обробка помилок.
- Можливість розширення.

Мінуси:
- Концептуально складніше і «важче» ніж REST.
- Великий обсяг службових даних в повідомленнях.
- Розробка без спец. інструментів складна.

Разом: REST

Плюси:

- Незалежні від мови і платформи (транспорт-HTTP).
- Простіше ніж SOAP.
- Швидше вивчення, менше потрібно інструментів.
- короткий, немає проміжного протоколу повідомлень (SOAP).
- За дизайном і філософії ближче до WWW.

Мінуси:

- Передбачає взаємодію тільки клієнт-сервер, що не застосуємо для розробки розподілених систем чи не стандартизовані механізми безпеки, гарантованої доставки повідомлень і т.п. - більше доводиться реалізувати самостійно прикладної системі.
- Жорстко прив'язаний до моделі протоколу HTTP.

http://www.osp.ru/os/2004/11/184785/

ЛЕКЦІЯ 6.
Оркестровка і хореографія Web-сервісів
(Кріс Пелц)

Для об'єднання Web-сервісів при створенні високорівневих бізнес-процесів, в які залучені кілька підприємств, необхідна стандартизація моделей їх взаємодії.

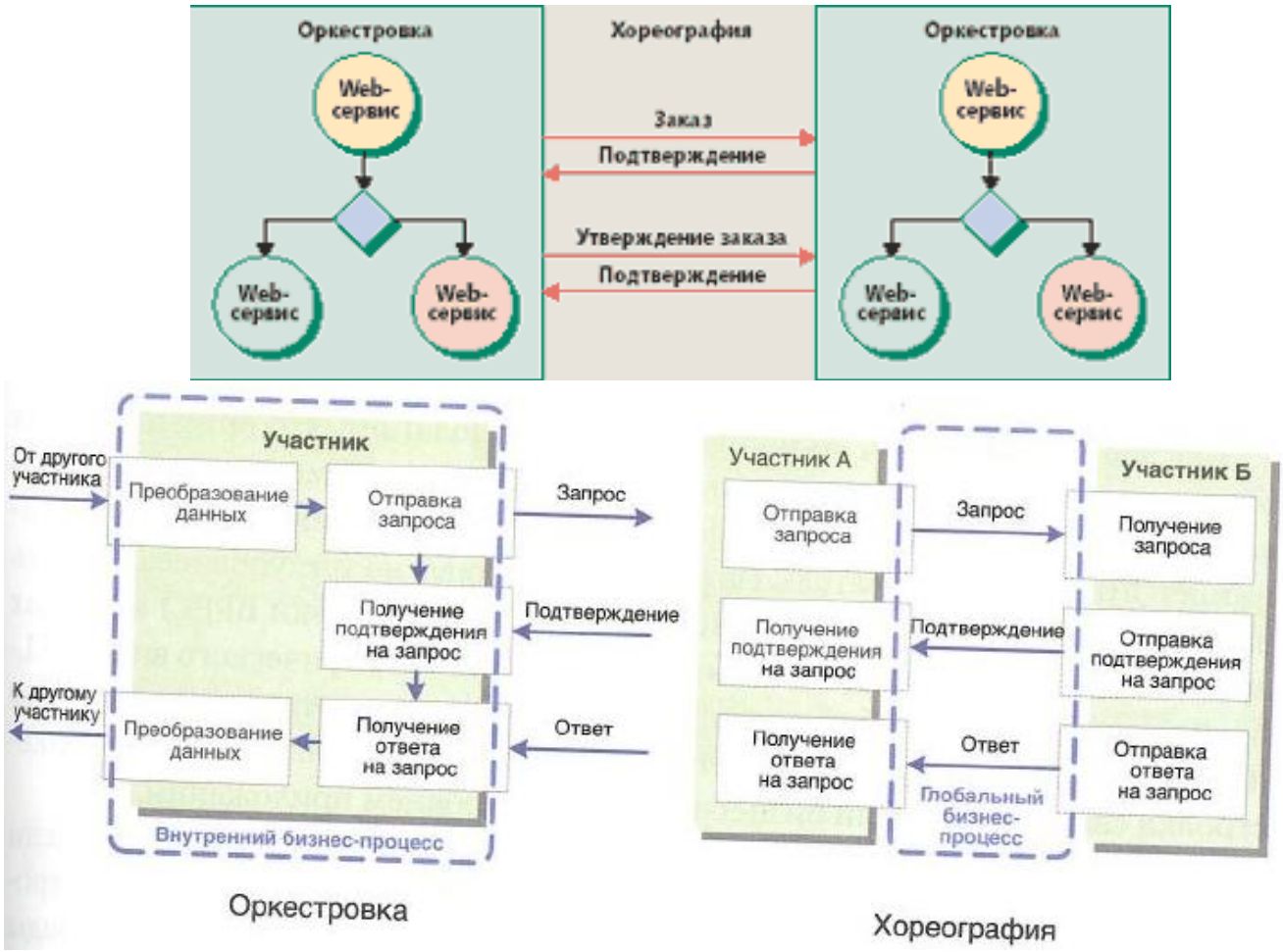

Рисунок 7. – Приклад оркестровки і хореографії

Оркестровка відноситься до виконуваного бізнес-процесу, а хореографія дозволяє відстежувати повідомлення його учасників.

Оркестровка (orchestration) і хореографія (choreography) відповідають двом підходам до опису бізнес-процесів: у вигляді послідовності виконання сервісів або у вигляді їх взаємодії.

Проблематика:

1. Для забезпечення надійності і універсальності, важлива можливість асинхронного звернення до сервісу. Підвищити продуктивність процесу дозволяє можливість одночасного звернення до кількох сервісів. Для реалізації асинхронних Web-сервіси можуть бути доступні механізм кореляції запитів, в якості

якого архітектори програмних систем зазвичай використовують ідентифікатори кореляції.

2. Управління винятковими ситуаціями і цілісністю транзакцій. Доступність ресурсів при виконанні тривалих розподілених транзакцій. (Классич транзакція - відкати, цілісність.)

3. Оркестровка Web-сервісів повинна бути динамічною, гнучкою і адаптивною. Механізм оркестровки обробляє потік робіт бізнес-процесу, викликаючи відповідні Web-сервіси та визначаючи, які кроки слід виконати. Цей підхід дозволяє замінювати сервіси в потоці робіт.

4. Проектувальники повинні мати можливість компоновки високорівневих сервісів з існуючих оркестрованих процесів.

Мова BPEL

У травні 2003 року Microsoft, IBM, Siebel, BEA Systems і SAP спільно розробили версію 1.1 специфікації мови BPEL4WS (Business Process Executive Language). Ця специфікація, для стислості іноді іменована BPEL, дозволяє моделювати поведінку Web-сервісів при взаємодії бізнес-процесів [2]. Її складовою частиною є граматика на основі XML, яка призначена для опису логіки управління при координації Web-сервісів, що беруть участь в потоці робіт бізнес-процесу. Механізм оркестровки відповідно до цієї граматикою може координувати дії і компенсувати процес в цілому при виникненні помилок.

BPEL є розширенням WSDL. Інтерфейс WSDL визначає список можливих операцій, а BPEL задає порядок їх виконання. WSDL описує публічні точки входу і виходу для кожного процесу BPEL, а типи даних WSDL визначають інформацію, якою обмінюються окремі запити бізнес-процесу. Крім того, WSDL дозволяє посилатися на зовнішні сервіси, необхідні процесу BPEL.

BPEL підтримує як виконувані, так і абстрактні бізнес-процеси. Виконуваний процес моделює поведінку учасників певного бізнес-взаємодії, по суті, моделюючи приватний потік робіт. Абстрактний процес, або бізнес-протокол, визначає обмін публічними повідомленнями між учасниками. Бізнес-протоколи не є виконуваними і не відображають внутрішніх подробиць потоку робіт бізнес-процесу. Іншими словами, виконувані процеси моделюють оркестровку, а абстрактні процеси - хореографію Web-сервісів.

Специфікація BPEL4WS підтримує як базові, так і структурні дії. Базове дію - це інструкція по взаємодії з чимось зовнішнім по відношенню до процесу. Наприклад, базові дії обслуговують отримання запиту, відповідь і виклик Web-сервісу (рис.8).

Структурні дії управляють потоком робіт бізнес-процесу в цілому, визначаючи послідовність виклику Web-сервісів. Ці дії також підтримують виконання циклів і динамічне розгалуження. По суті, вони становлять основну логіку програмування в BPEL.

Два інших важливих елемента BPEL - партнерські зв'язки і змінні.

Мінлива ідентифікує певні дані в потоці повідомлень. Коли процес BPEL отримує повідомлення, він привласнює значення відповідної змінної, щоб наступні запити могли звернутися до цих даних. Змінні використовуються для

управління довгостроковим зберіганням даних в ході обробки запитів Web-сервісів.

Партнерська зв'язок може позначати будь-який сервіс, з яким взаємодіє даний процес. Кожній партнерської зв'язку приписана певна роль в бізнес-процесі.

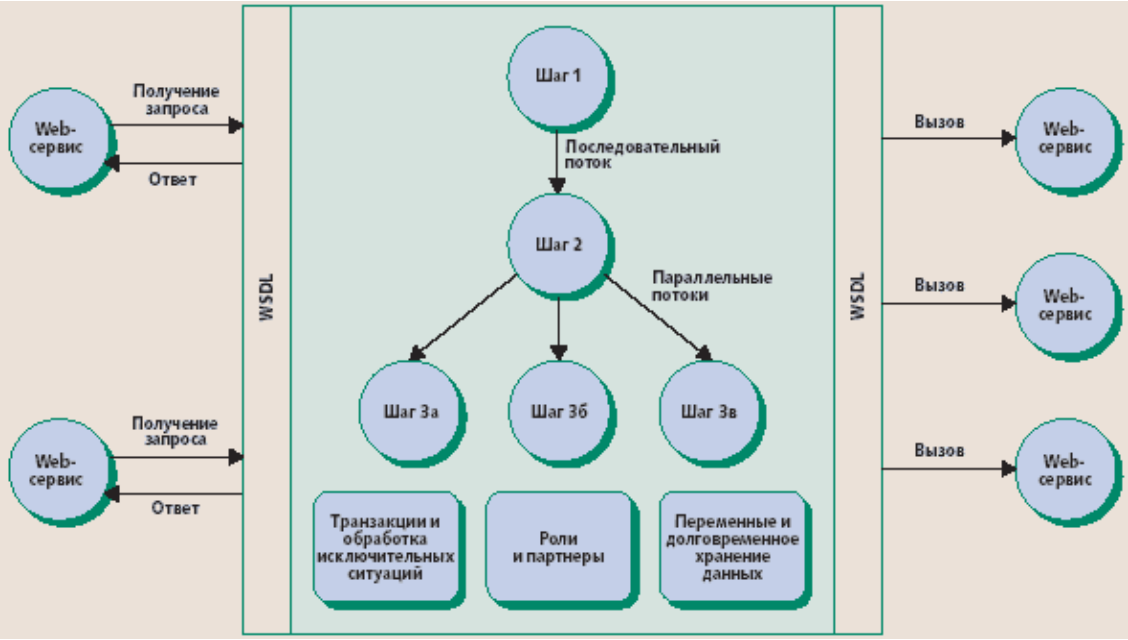

Рисунок 8. - Відповідь і виклик Web-сервісу

Основні файли BPEL-проекту:

.bpel - логічний синтез і координація веб-служб. Фактично, алгоритм виконання бізнес-процесу. (Його графічне представлення нагадує блок-схему і діаграму потоків даних в одній особі).

.wsdl - опис інтерфейсів для обміну повідомленнями. «Як досягти веб-служби» (WSDL).

.xsd - опис структур даних проекту (XML Schema).

Язик - візуальний редактор.

Мова WSCI

Компанії Sun Microsystems, SAP, BEA і Intalio розробили специфікацію мови опису інтерфейсів Web Services Choreography Interface (WSCI), яка визначає розширення мови WSDL, спрямоване на організацію спільної роботи. У загальному вигляді WSCI визначає хореографію, або обмін повідомленнями між Web-сервісами. Специфікація забезпечує кореляцію повідомлень, правила впорядкування, обробку виняткових ситуацій, транзакції і динамічна взаємодія.

BPML - оркестровка.

Мова BPEL підтримує як виконувані процеси, так і організацію взаємодії між ними. BPML і WSCI можуть працювати разом так, щоб BPML моделював виконання бізнес-процесу, а WSCI - хореографію Web-сервісів.

ЛЕКЦІЯ 7.
Хмарні обчислення

Історія і ключові фактори розвитку

Вперше ідея того, що ми сьогодні називаємо хмарними обчисленнями, була озвучена JCR Licklider, в 1970 році. У ці роки він був відповідальним за створення ARPANET (Advanced Research Projects Agency Network). Його ідея полягала в тому, що кожна людина на землі буде підключений до мережі, з якої він буде отримувати не тільки дані на і програми. Інший вчений John McCarthy висловив ідею про те, що обчислювальні потужності будуть надаватися користувачам як послуга (сервіс). На цьому розвиток хмарних технологій було призупинено до 90-х років, після чого її розвитку посприяв ряд факторів.

1. Розширення пропускної здатності Інтернету, в 90-ті роки не дозволило отримати значний стрибок у розвитку в хмарної технології, так як практично жодна компанія не технології того часу не були готові до цього. Однак сам факт прискорення Інтернету дав поштовх швидкому розвитку хмарних обчислень.

2. Одним з найбільш значущих подій в даній області була поява Salesforce.com в 1999 році. Дана компанія стала першою компанією надала доступ до свого додатком через сайт, по суті дана компанія стала першою компанією надала своє програмне забезпечення за принципом - програмне забезпечення як сервіс (SaaS).

3. Наступним кроком стала розробка хмарного веб-сервісу компанією Amazon в 2002 році. Даний сервіс дозволяв зберігати, інформацію і робити обчислення.

4. У 2006, Amazon запустила сервіс під назвою Elastic Compute cloud (EC2), як веб-сервіс який дозволяв його користувачам запускати свої власні додатки. Сервіси Amazon EC2 і Amazon S3 стали першими доступними сервісами хмарних обчислень.

5. Інша віха в розвиток хмарних обчислень сталася після створення компанією Google, платформи Google Apps для веб-додатків в бізнес секторі.

6. Значну роль в розвитку хмарних технологій зіграли технології віртуалізації, зокрема програмне забезпечення, що дозволяє створювати віртуальну інфраструктуру.

7. Розвиток апаратного забезпечення сприяло не стільки швидкому зростанню хмарних технологій, скільки доступності даної технології для малого бізнесу і індивідуальних осіб. Що стосується технічного прогресу, то значну роль в цьому зіграло створення багатоядерних процесорів і збільшення ємності накопичувачів інформації.

Хмарний обчислення в даний час.

Вікіпедія дає таке визначення хмарних обчислень. Хмарні обчислення (англ. Cloud computing) - технологія розподіленої обробки даних, в якій комп'ютерні ресурси і потужності надаються користувачеві як Інтернет-сервіс. Надання користувачеві послуг як Інтернет-сервіс є ключовим. Однак під Інтернет-сервісом не варто розуміти доступ до сервісу тільки через Інтернет, він може

здійснюватися також і через звичайну локальну мережу з використанням веб-технологій.

З визначення та історії видно, що основою для створення і швидкого розвитку хмарних обчислювальних систем послужили великі інтернет сервіси, такі як Google, Amazon і ін, а так само технічний прогрес, що по суті говорить про те що поява хмарних обчислень було всього лише справою часу . Розглянемо яким же чином розвиток перерахованих вище напрямків дозволило хмарним системам стати доступнішими.

1. Розвиток багатоядерних процесорів привело до:

- збільшення продуктивності, при тих же розмірах обладнання;
- зниження вартості обладнання, як наслідок експлуатаційних витрат;
- зниження енергоспоживання хмарної системи, для більшості ЦОД це дійсно проблема при нарощуванні потужностей ЦОД.

2. Збільшення ємностей носіїв інформації, зниження вартості зберігання 1 Мб інформації дозволило:

- безмежно (принаймні так позиціонують себе більшість «хмар») збільшити обсяги інформації, що зберігається;
- знизити вартість обслуговування сховищ інформації, значно збільшивши обсяги даних.

3. Розвиток технології багатопотокового програмування призвело до:

- ефективному використанню обчислювальних ресурсів багатопроцесорних систем;
- гнучкий розподіл обчислювальних потужностей хмар.

4. Розвиток технологій віртуалізації призвело до:

- створення програмного забезпечення дозволяє створювати віртуальну інфраструктуру не залежно від кількості наданих апаратних ресурсів;
- легкість масштабування, нарощування систем;
- зменшення витрат на адміністрування хмарних систем;
- доступність віртуальної інфраструктури через мережу Інтернет.

5. Збільшення пропускної здатності призвело до:

- збільшення швидкості роботи з хмарними системами зокрема віртуальний графічний інтерфейс і робота з віртуальними носіями інформації;
- зниження вартості Інтернет трафіку для роботи з великими обсягами інформації;
- проникненню хмарних обчислень в маси.

Всі перераховані вище фактори призвели до підвищення конкурентоспроможності хмарних обчислень в IT сфері.

Переваги хмарних обчислень

Доступність- хмари доступні усім, з будь-якої точки, де є Інтернет, з будь-якого комп'ютера, де є браузер. Це дозволяє користувачам (підприємствам) економити на закупівлі високопродуктивних, дорогих комп'ютерів. Також співробітники компаній стають більш мобільними так, як можуть отримати доступ до свого робочого місця з будь-якої точки земної кулі, використовуючи ноутбук, нетбук, планшетник або смартфон. Немає необхідності в покупки ліцензійного програмного забезпечення, його налаштування і оновлення, ви просто заходите

на сервіс та користуєтеся його послугами заплативши за фактичне використання.

Низька вартість - основні чинники знизили вартість використання хмар наступні:

- зниження витрат на обслуговування віртуальної інфраструктури, викликане розвитком технологій віртуалізації, за рахунок чого потрібен менший штат для обслуговування всієї IT інфраструктури підприємства;

- оплата фактичного використання ресурсів, користувач хмари платить за фактичне використання обчислювальних потужностей хмари, що дозволяє йому ефективно розподіляти свої грошові кошти. Це дозволяє користувачам (підприємствам) економити на покупці ліцензій до ПЗ;

- використання хмари на правах оренди дозволяє користувачам знизити витрати на закупівлю дорогого устаткування, і зробити акцент на вкладення грошових коштів на наладку бізнес процесів підприємства, що в свою чергу дозволяє легко почати бізнес;

- розвиток апаратної частини обчислювальних систем, в зв'язку з чим зниження вартості обладнання.

Гнучкість - необмеженість обчислювальних ресурсів (пам'ять, процесор, диски), за рахунок використання систем віртуалізації, процес масштабування і адміністрування "хмар" ставати досить легким завданням, так як «хмара» самостійно може надати вам ресурси, які вам необхідні, а ви платите тільки за фактичне їх використання.

Надійність - надійність «хмар», особливо що знаходяться в спеціально обладнаних ЦОД, дуже висока так, як такі ЦОД мають резервні джерела живлення, охорону, професійних працівників, регулярне резервування даних, високу пропускну здатність Інтернет каналу, висока стійкість до DDOS атак.

Безпека - «хмарні» сервіси мають досить високу безпеку при належному її забезпеченні, однак при недбалому ставленні ефект може бути повністю протилежним.

Великі обчислювальні потужності- ви як користувач «хмарної» системи можете використовувати всі її обчислювальні можливості, заплативши тільки за фактичний час використання. Підприємства можуть використовувати цю можливість для аналізу великих обсягів даних.

Недоліки

Постійне з'єднання з мережею- для отримання доступу до послуг «хмари» необхідно постійне з'єднання з мережею Інтернет. Однак у наш час це не такий і великий недолік особливо з приходом технологій стільникового зв'язку 3G і 4G.

Програмне забезпечення та його кастомизация- є обмеження по ПО яке можна розгортати на «хмарах» і надавати його користувачеві. Користувач ПО має обмеження в використовуваному ПО і іноді не має можливості налаштувати його під свої власні цілі.

Конфіденційність - конфіденційність даних, що зберігаються на публічних «хмарах» в даний викликає багато суперечок, але в більшості випадків експерти сходяться в тому, що не рекомендується зберігати найбільш цінні для

компанії документи на публічному "хмарі", так як в даний час немає технології яка б гарантувала 100 % конфіденційність даних, що зберігаються.

Надійність - що стосується надійності інформації, що зберігається, то з упевненістю можна сказати що якщо ви втратили інформацію збережену в "хмарі", то ви її втратили назавжди.

Безпека - "хмара" саме по собі є досить надійною системою, однак при проникненні на нього зловмисник отримує доступ до величезного сховища даних. Ще один мінус це використання систем віртуалізації, в яких в якості гіпервизора використовуються ядра стандартні ОС такі, як Linux, Windows і ін., Що дозволяє використовувати віруси.

Дорожнеча обладнання - для побудови власного хмари компанії необхідно виділити значні матеріальні ресурси, що не вигідно щойно створеним і малим компаніям.

Види послуг що надаються хмарними системами

Що стосується послуг, що надаються, то в даний час концепція хмарних обчислень передбачає надання наступних типів послуг своїм користувачам:

- все як послуга (Everything as a Service);

При такому вигляді сервісу користувачеві буде надано все від програмно апаратної частини і до управлінням бізнес процесами, включаючи взаємодію між користувачами, від користувача вимагається тільки наявність доступу в мережу Інтернет. На мій погляд, даний вид сервісу це більш загальне поняття по відношенню до нижченаведеними послуг, які є окремими випадками.

- інфраструктура як послуга (Infrastructure as a service);

Користувачеві надається комп'ютерна інфраструктура, зазвичай віртуальні платформи (комп'ютери) пов'язані в мережу. Які він самостійно налаштовує під власні цілі.

- платформа як послуга (Platform as a service);

Користувачеві надається комп'ютерна платформа, зі встановленою операційною системою можливо і з ПО.

- програмне забезпечення як послуга (Software as a service);

Даний вид послуги зазвичай позиціонується як «програмне забезпечення на вимогу», це програмне забезпечення розгорнуте на віддалених серверах і користувач може отримувати до нього доступ за допомогою Інтернету, причому всі питання оновлення та ліцензій на дане програмне забезпечення регулюється постачальником даної послуги. Оплата в даному випадку проводитися за фактичне використання програмного забезпечення.

- апаратне забезпечення як послуга (Hardware as a Service);

В даному випадку користувачеві послуги надається обладнання, на правах оренди яке він може використовувати для власних цілей. Даний варіант дозволяє економити на обслуговуванні даного обладнання, хоча за своєю суттю мало чим відрізняється від виду послуги «Інфраструктура як сервіс» за винятком того що ви маєте голе обладнання на основі якого розвертаєте свою власну інфраструктуру з використанням найбільш підходящого програмного забезпечення.

- робоче місце як послуга (Workplace as a Service);

В даному випадку компанія використовує хмарні обчислення для організації робочих місць своїх співробітників, налаштувавши і встановивши все необхідне програмне забезпечення, необхідне для роботи персоналу.

- дані як послуга (Data as a Service);

Основна ідея даного виду послуги полягає в тому, що користувачеві надається дисковий простір, яке він може використовувати для зберігання великих обсягів інформації.

- безпеку як сервіс (Security as a Service).

Даний вид послуги надає можливість користувачам швидко розгортати, продукти дозволяють забезпечити безпечне використання веб-технологій, безпеку електронного листування, а також безпеку локальної системи, що дозволяє користувачам даного сервісу економити на розгортанні та підтримці своєї власної системи безпеки.

Класифікація хмарних сервісів.

В даний час виділяють три категорії «хмар»:

Публічна хмара - це IT-інфраструктура використовується одночасно безліччю компаній і сервісів. Користувачі даних хмар не мають можливості управляти і обслуговувати дане хмара, вся відповідальність з цих питань покладено на власника даного хмари. Абонентом пропонованих сервісів може стати будь-яка компанія і індивідуальний користувач. Вони пропонують легкий і доступний за ціною спосіб розгортання веб-сайтів або бізнес-систем, з великими можливостями масштабування, які в інших рішеннях були б недоступні. Приклади: онлайн сервіси Amazon EC2 і Simple Storage Service (S3), Google Apps / Docs, Salesforce.com, Microsoft Office Web.

Приватна хмара - це безпечна IT-інфраструктура, контрольована і експлуатована в інтересах однієї-єдиної організації. Організація може керувати приватним хмарою самостійно або доручити це завдання зовнішньому підряднику. Інфраструктура може розміщуватися або в приміщеннях замовника, або у зовнішнього оператора, або частково у замовника і частково у оператора. Ідеальний варіант приватного хмари це хмара розгорнуте на території організації, що обслуговує і контрольоване її співробітниками.

Гібридна хмара - це IT інфраструктура використовує кращі якості публічного і приватного хмари, при вирішенні поставленого завдання. Часто такий тип хмар використовується, коли організація має сезонні періоди активності, іншими словами, як тільки внутрішня IT-інфраструктура не справляється з поточними завданнями, частина потужностей перекидається на публічне хмара (наприклад великі обсяги статистичної інформації, які в необробленому вигляді не уявляють цінності для підприємства), а також для надання доступу користувачам до ресурсів підприємства (до приватного хмари) через публічне хмара.

Куди треба розвиватися або на чому можна заробити гроші?

За оцінками експертів потенціал хмарних обчислень дуже високий. А відповідно можна буде потрапити в цей потік і відхопити його частина працюючи в наступних напрямках:

1. Надання послуг хмарних обчислень - дана можливість доступна не всім компаніям так, потрібні значні вкладення в побудові і розробку ЦОД.

2. Розробка ПО для побудови віртуальної інфраструктури, не слід забувати і про тих хто буде впроваджувати і налаштовувати це ПО, т. Е. Будуть потрібні фахівці в цій галузі.

3. Аутсорсінг, адміністрування хмар - будуть потрібні фахівці з адміністрування і консультування в сфері хмарних обчислень.

4. Апаратне забезпечення - компанії займаються розробкою і проектуванням апаратного забезпечення для створення «хмар».

5. Проектування - дана сфера охоплює практично всі перераховані вище сфери починаючи від проектування ЦОД і закінчуючи проектуванням програмного забезпечення.

http://habrahabr.ru/blogs/cloud_computing/111274/
http://habrahabr.ru/company/scalaxy/blog/65228/

ЛЕКЦІЯ 8.
GRID

Що таке грід?

Грід - географічно розподіленої інфраструктури, що об'єднує безліч ресурсів різних типів (процесори, довготривала і оперативна пам'ять, сховища і бази даних, мережі), доступ до яких користувач може отримати з будь-якої точки, незалежно від місця їх розташування. Грід передбачає колективний розділяється режим доступу до ресурсів і до пов'язаних з ними послуг в рамках глобально розподілених віртуальних організацій, що складаються з підприємств і окремих фахівців, які разом використовують спільні ресурси. У кожної віртуальної організації є своя власна політика поведінки учасників, які повинні дотримуватися встановлених правил. Віртуальна організація може утворюватися динамічно і мати обмежений час існування.

Перші повнофункціональні прототипи грід-систем відносяться до проекту Distributed Computing System (DCS) project початку 70-х років, над яким велися роботи в Каліфорнійському Університеті під керівництвом Девіда Фарбера. Технологія описувалася як "Кільце (мережа) працює як одна дуже гнучка система, на якій окремі вузли (комп'ютери) можуть запитувати завдання". Ця схема дуже схожа на те, як обчислювальні можливості іпользуются на гріді. Однак в 80-х роках, ця технологія була практично занедбана, так як труднощі адміністрування і безпеки пов'язані з виконанням вашого завдання на комп'ютері, яким ви не можете управляти, здавалися (і до сих пір здаються деяким) непереборними.

Ідеї грід були зібрані разом Яном Фостером, Карлом Кессельманом і Стівом Тьюк, так званими "батьками грід". Вони стояли і біля витоків розробки першого стандарту конструювання грід-систем, вільно поширюваного програмного інструменту з відкритим кодом Globus Toolkit. Цей стандарт об'єднав не лише управління процесорним часом, але і управління зберіганням даних, забезпеченням безпеки, рухами даних, контроль станів, а також інструментарій для розробки додаткових сервісів. Він заснований на тій же інфраструктурі, включаючи переговори про угоду, механізми повідомлення, тригерні сервіси та агрегування інформації Термін грід має більш далекі наслідки, ніж прийнято вважати. У той час як Globus Toolkit залишається стандартом де-факто для створення рішень на базі грід, була створена велика кількість інших засобів,

Для подальшого розвитку Globus Toolkit в 1999 році була створена спеціальна організація Global Grid Forum (GGF), в яку поряд з академічними організаціями увійшли багато виробників комп'ютерних систем і програмного забезпечення.

У 2002 році GGF і корпорацією IBM в рамках версії Globus Toolkit 3.0 була представлена нова системна розробка Open Grid Services Architecture (OGSA), інкорпоровані в грід поняття і стандарти веб-сервісів. У цій архітектурі грід-сервіс визначається як спеціальний тип веб-сервісу, завдяки чому стає можливою робота з ресурсами грід на базі стандартних інтернет-протоколів.

У своїй статті "Що таке грід? Три критерію "в 2002 році Ян Фостер наводить простий список критеріїв, відповідно до якого, грід - це система яка:

• координує використання ресурсів за відсутності централізованого управління цими ресурсами (Грід інтегрує і координує ресурси і користувачів, які знаходяться в різних місцях, наприклад, персональний комп'ютер користувача і центральний комп'ютер, різні адміністративні відділення однієї компанії або різні компанії, і направляє учасникам повідомлення про гарантії, страховці , платежах, членство і т.д. Якщо це не так, ми маємо справу з локальною системою управління);

• використовує стандартні, відкриті, універсальні протоколи і інтерфейси (Грід будується на базі багатоцільових протоколів і інтерфейсів, що дозволяють вирішувати такі фундаментальні завдання як аутентифікація, авторизація, виявлення ресурсів і доступ до ресурсів. Якщо це не так, ми маємо справу зі спеціалізованою прикладної системою);

• повинна нетривіальним чином забезпечувати високоякісне обслуговування (грід дозволяє використовувати що входять до його складу ресурси таким чином, щоб забезпечувалося високу якість обслуговування, що стосується, наприклад, таких параметрів, як час відгуку, пропускна здатність, доступність і надійність, а спільне використання ресурсів різних типів, готових відповідати складним запитам користувачів, робить вигоду від використання комбінованої системи значно вище, ніж від суми її окремих частин).

В даний час вважається, що ключовими властивостями гріда є:

- грід працює з обчисленнями;
- грід працює з даними;
- грід забезпечує безпеку;
- грід надає інформаційну систему (можливість моніторингу виконання робіт в гріді);
- грід забезпечує підтримку різних середовищ програмування.

Більшість сучасних грід-систем можуть працювати тільки під Linux, використовуючи специфічне програмне забезпечення. Це дещо обмежує можливості їх застосування. Крім цього підтримує можливість запуску складного робочого потоку (workflow), який може бути створений за допомогою мови програмування високого рівня (Java) або с допомогою графічного інтерфейсу. Для подання робочого потоку використовується базується на XML мова JSDL, що є одні з відкритих стандартів в області грід-технологій.

http://jre.cplire.ru/jre/dec03/4/text.html (2003)
http://software.intel.com/ru-ru/blogs/2010/02/09/2003069/
http://www.gridclub.ru/about
http://www.gridclub.ru/library/publication.2004-11-29.7104738919/publ_file/

ЛЕКЦІЯ 9.
Програмне забезпечення інфраструктури GRID

Створені різні інтегровані пакети програм обслуговування GRID, часто звані toolkit. Однією з них, є програма Globus, яка становить частину інфраструктури GRID, вона надає платформу для підтримки віртуальної організації та виконання GRID додатків. У цьому розділі розповідається про цю програму і її компонентах. Хоча кожен пакет програм інфраструктури GRID - Globus, Condor, Legion, Unicore або інші приватні рішення - мають свої особливості і спеціалізацію, компоненти програми Globus повністю відповідають даній ідеології і її основних тез.

1. GLOBUS

Набір програм Globus Toolkit (GT) - програмний продукт з відкритим вихідним кодом і набором бібліотек. Він містить набір стандартних блоків і інструментів, які можуть бути використані розробниками і системними інтеграторами. За кілька років вийшло чотири версії програми Globus: оригінальна - в кінці дев'яностих, GT2 - у 2000, GT3 - в 2003, і GT4 - в 2005. Версія GT2 послужила базисом для безлічі GRID розробників по всьому світу. GT3 - стала першою повноцінною реалізацією інфраструктури GRID, побудованої на технології Web-сервісів, з використанням проміжної ланки GGF's OGSI. GT4 - перша версія, повністю сумісна з основними Web-сервісами так само, як GRID- сервіси засновані на WSDL і WSRF. Більшість систем GRID використовують ОС UNIX.

У наступних версіях програми Globus, очікується подальше поєднання з пакетом специфікацій OGSA, які були визначені в GGF. Протягом всіх етапів розвитку GT, розробники Globus концентрували свою увагу на створенні інструментів, що мають загальний інтерфейс для взаємодії різнорідними компонентами системи. Зокрема, в GT визначені і реалізовані протоколи, API, і інші засоби, що надають загальні рішення проблем використання і сумісності, таких як, ідентифікація, дослідження ресурсів і доступ до них. Вирішення цих проблем, досягається за допомогою механізмів, які забезпечують безпеку, дослідження інформації, управління ресурсами та даними, зв'язок, діагностику помилок і портативність.

Ресурсні протоколи GT, використовуються для ініціювання процесу розрахунку, виявлення і моніторингу ресурсів, а також передачі даних. Пакет програм розміщення і управління ресурсами GRID (GRAM - GRID Resource Allocation and Management), призначений для безпечного управління процесами на віддалених ресурсах. Служба моніторингу та виявлення вільних ресурсів надає єдиний механізм виявлення і доступу до інформації по статусу і конфігурації GRID ресурсів, зокрема, до конфігурації обчислювального сервера, мережевого статусу і можливостям різних сервісів. GridFTP - це розширена версія FTP додатки і протоколу. Розширення включають в себе протоколи безпечного з'єднання, часткового доступу до файлів і управління розпаралелюванням для більш високій швидкості передачі даних.

2. CONDOR

Кластерна система Condor була створена групою розробників Університету Wisconsin-Madison де і була більше 10 років тому розгорнуто першу конфігурація. На сьогоднішній день в університеті є 350 настільних UNIX станцій, які включені в мережу Condor і надають доступ для роботи користувачам з усього світу. У багатьох випадках, особливо якщо мова йде про розрахункових задачах, користувачам набагато важливіше не швидкість виконання одного завдання, а число завдань, які можна виконати за досить тривалий час. Інакше кажучи, число операцій в місяць або рік. Такий постулат лежить в основі технології ефективного використання наявних комп'ютерних ресурсів - High Throughput Computing (HTC). Система Condor - це ПЗ для підтримки середовища HTC, утвореної станціями на платформі UNIX і NT. Незважаючи на те, що Condor може керувати спеціалізованими кластерами з робочих станцій, його ключова перевага - здатність розподіляти звичайні комп'ютерні ресурси, доступні в будь-якій лабораторії або офісі. Іноді ще Condor називають «мисливець за вільними станціями» - замість того щоб запускати завдання на своїй машині, користувач звертається до системи, яка шукає тимчасово вільні машини в мережі і запускає на них завдання. Коли машина перестане бути вільною (який повернувся з обіду користувач натиснув клавішу або кнопку миші, або машина отримала команду вивантаження ОС), Condor перериває виконання завдань, здійснює міграцію на іншу вільну машину і перезапускає завдання на неї з перерваного місця. Якщо немає вільних машин, то завдання поміщається в чергу і чекає вільних ресурсів. Передбачений в Condor механізм управління завданнями,

Крім виконання міграції на вільні машини Condor забезпечує управління розподіленими ресурсами. На відміну від інших систем СУПО, в яких адміністратору або користувачу потрібно вручну редагувати реквізити завдання в черзі, щоб задовольнити вимогам виконує комп'ютера і проштовхнути завдання на рахунок, Condor повністю автоматизує процес розподілу завдань, використовуючи для цього об'єктну технологію ClassAds. Дана технологія працює подібно газетному рекламному класифікатором. Всі машини, доступні пулу Condor оголошують (рекламують) свої можливості в рубриці «Пропозиція»: обсяг пам'яті на диску, тип і швидкодія процесора, обсяг віртуальної пам'яті, середнє завантаження і т.п. З іншого боку, користувач описує потреби свого завдання в рубриці «Запит ресурсів». Condor працює в ролі брокера, задовольняє заявки з рубрики «Запит ресурсів» ресурсами, представленими в рубриці «Пропозиція». Одночасно система забезпечує ведення дисципліни декількох рівнів пріоритетів розгляду заявок: пріоритет призначення ресурсів, пріоритет використання і пріоритет серед машин, які відповідають однаковим заявками.

При роботі Condor від користувача не потрібно спеціальної реєстрації (account або login) на машинах де буде виконуватися його завдання - система сама виконує реєстрацію за допомогою технології віддаленого виклику RSC (Remote System Call), що дозволяє перехоплювати виклики ОС при виконанні операцій читання / запису файлів і пересилати їх на машину звідки було запу-

щено завдання. Таким чином, користувач може не хвилюватися про доступність файлів на віддаленій машині або отриманні для себе account. Система не вимагає використовувати будь-який спеціальний ПО і здатна виконувати звичайні UNIX і NT програми, а користувачеві потрібно тільки пересобрать свою задачу з бібліотеками Condor. Права господарів робочих станцій,

3. GRAM

Модуль Globus Resource Allocation Manager (GRAM) забезпечує виділення ресурсів, створення і моніторинг процесів, а також управління ними. GRAM перетворять запити на мові Resource Specification Language (RSL) в команди, зрозумілі локальним планувальникам. Застосовуючи цю функцію, користувачі мережі розподілених обчислень можуть віддалено задіяти такі утиліти Oracle, як export, import і sqlplus, для виконання необхідних дій в Oracle Database 10g. Цей механізм можна використовувати для налаштування віддалених баз даних в мережі розподілених обчислень.

WS GRAM. Публікує інформацію про стан локальних планувальників завдань, зокрема: поточний стан черги завдань, кількість доступних і число вільних процесорів, число розміщених завдань, інформація про ресурсах оперативної пам'яті.

WS GRAM не є планувальником ресурсів, а представляє з себе реалізацію протоколу для зв'язку різних локальних планувальників ресурсів з використанням повідомлень стандартного формату. WS GRAM надає расширяемую архітектуру, засновану на підключаються модулях для підтримки широкого спектра локальних планувальників.

WS GRAM націлений на коло завдань, де особливо важливі надійність операцій, гнучкий моніторинг, управління доступом і керування даними. WS GRAM не створювалася як інтерфейс для "віддаленого виклику процедур" (RPC) для додатків яким ці особливості не потрібні. Що лежить в основі оточення WSRF, що використовується WS GRAM, потрібно розглядати як сервер додатків для легковажних розподілених додатків. Іншими словами, якщо додаток вимагає лише незначної передачі даних для введення і виведення в незалежній від внутрішнього стану формі (важливі лише вихідні дані або сигнал про помилку) і викликається багаторазово, воно є хорошим кандидатом для реалізації у вигляді web-сервісу.

Для управління завданнями за допомогою WS GRAM використовуються наступні типи сервісів:

1) Сервіси управління завданнями описують, моніторять і управляють повним життєвим циклом завдання. Ці послуги є спеціально спроектовані для управління завданнями компоненти GRAM

2) Сервіси передачі файлів забезпечують переміщення файлів серед обчислювальних ресурсів. WS GRAM використовує існуючі технології, такі як GridFTP і RFT

3) Сервіси управління акредитацією використовується для делегування повноважень серед розподілених елементів архітектури WS GRAM, заснованої

на вимогах призначених для користувача додатків. WS GRAM використовує існуючі технології реалізовані GSI

WS GRAM використовує функціональні можливості WSRF, щоб передбачити ідентифікацію запитів управління завданнями, а так же захистити завдання від зловмисного втручання Щоб захистити користувачів один від одного, завдання виконуються у відповідних контекстах локальної безпеки, наприклад, під певними UID, заснованими на деталях запиту завдання і політики авторизації. Крім того, механізми WS GRAM використовуються для взаємодії з локальними ресурсами спроектовані з урахуванням мінімізації необхідних привілеїв і ризику відмови сервісу. Клієнт може делегувати деякі зі своїх прав сервісу WS GRAM, для надання вищевказаних функцій, наприклад права WS GRAM для доступу до даних на віддаленому елементі зберігання в якості частини виконання завдання. додатково, клієнт може делегувати права для використання самим процесом завдання. WS GRAM забезпечує роздільну реалізацію цих методів делегування. Для реалізації обліку і зниження шкоди від неправильного поводження до служби або збою, WS GRAM використовує широкий діапазон методів аудиту та реєстрації хронології уявлення завдань і критичних системних операцій.

WS GRAM забезпечує "максимум один раз" семантику відправки завдання. Клієнт може перевіряти і при необхідності повторно відправляти завдання, без ризику виконання більше ніж однієї копії завдання в результаті помилок передачі даних. У той час як багатьом завданням дозволено виконуватися до їх природного завершення, WS GRAM забезпечує клієнтів механізмом для скасування їх завдань в довільній точці життєвого циклу завдання.

WS GRAM WS надає надійну, високоефективну передачу файлів між обчислювальними ресурсами і зовнішніми (GridFTP) елементами зберігання даних до і після виконання завдання. WS GRAM підтримує механізм інкрементального виведення вмісту довільного файлу з обчислювального ресурсу, під час виконання завдання.

Деякі завдання є паралельними, це означає, що вони складаються з більше ніж однією одночасної завдання. Ці завдання як правило розміщуються на паралельних обчислювальних апаратних засобах, для забезпечення зростаючої обчислювальної пропускної здатності. Деякі паралельні середовища програмування, типу MPI, забезпечує паралельно що виконуються завданням механізми для отримання інформації про кількість процесів один одного або обміну даними. WS GRAM надає механізм для 'зустрічей' завдань, який може бути використаний додатками за умови, що ті не мають можливість іншого рішення поставленого завдання. Цей механізм може бути безпосередньо використаний додатками, або бібліотеками посередника, спроектованими для подання спрощеного запуску додатків за допомогою WS GRAM. Сам WS GRAM не вимагає щоб запущені завдання координувалися, але це потрібно для забезпечення функціонування широкого спектра додатків. WS GRAM надає протоколи і API для реалізації механізму 'зустрічей'.

4. WSRF

Web Services Resource Framework - це специфікації для оперування ресурсами через стандартні Web-сервіси. ()

4.1 Основи WSRF

Принцип WSRF - рішення для зберігання інформації ресурсів, доступ до якої здійснюється через інтерфейс Web-сервісів. Основною причиною для таких функціональних можливостей є те, що ви можете надати ресурс з постійною інформацією про стан (або іншою інформацією), яка зберігається між зверненнями конкретного додатка. Просто поміщений ресурс стає механізмом для зберігання інформації. У стандартному додатку інформація буде якимось чином зберігатися в програмному коді. А в середовищі Web-сервісів вам необхідно надати механізм для зберігання інформації, тому що час життя Web-сервісу дуже недовго. Якщо ви хочете записати стан додатки, ресурсу або сервіси, то у вас повинен бути механізм для запису цієї інформації.

Ресурс - це тільки елемент, про який ви хочете зберегти інформацію, і може бути як статичним елементом, так і ставитися до групи елементів, якої ви звертаєтеся по імені або ID (ідентифікатор) .Наявність ресурсу і збережених значень дозволяє вам записувати інформацію в середовищі Web СЕРВІС, де немає стану.

WSRF надає стандартизований метод доступу до інформації про ресурс і внесення інформації в ресурс. Так як він заснований на стандартах, то будь-який продукт, що підтримує WSRF, може використовувати або виконувати запити до вашого ресурсу, що полегшує підтримку ресурсу в середовищі SOA.

4.2 Обробка ресурсу

У WSRF ресурс - це ім'я, дане об'єкту, в який ви можете записувати для зберігання і зчитувати інформацію. У додатку, наведеному в цьому посібнику в якості зразка, ви будете обробляти ресурс, що зберігається в базі даних Derby.

Огляд конкретного прикладу ресурсу і, в якійсь мірі всіх середовищ WSRF, наведено на рис.9.

У прикладі ви обробляєте ресурс з назвою Person, який насправді є рядком таблиці, що зберігається в базі даних. У ресурсу буде унікальний ідентифікатор ID (обробляється унікальним полем ID в таблиці ресурсу).

Підтримуючи унікальний ID, ви можете зберігати численні ресурси (і, отже, інформацію про стан множинних ресурсів), змушуючи систему гнучко публікувати інформацію про ресурс Person.

База даних використовується як механізм зберігання і забезпечує довготривале зберігання, необхідне для підтримки стандарту ресурсів Web-сервіс і WSRF. WSRF визначає стандартні методи для отримання інформації про ресурс, яке обробляється як Web-сервіси, показані на рис.9.

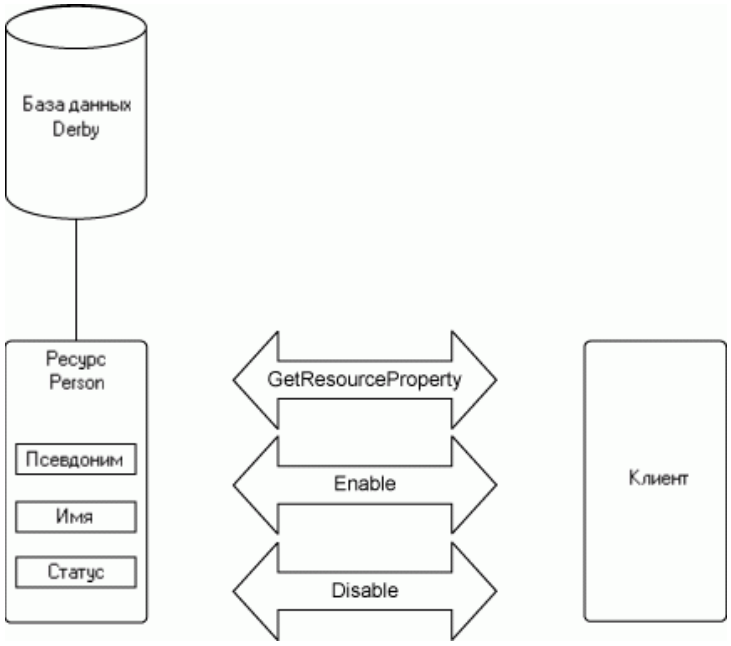

Рисунок 9. - Огляд прикладу ресурсу

Необхідно забезпечити три Web-сервісу: один для отримання значення властивості ресурсу і два для активації і блокування ресурсу.

5. OGSA

Open Grid Services Architecture (OGSA) описує архітектуру для сервісно-орієнтованого мережевого обчислювального середовища для ділового та наукового використання, розробленого в рамках Глобального Grid Forum (GGF). OGSA базується на кількох інших технологіях Веб-сервісу, зокрема WSDL та SOAP, але вона має на меті бути великою мірою агностичною щодо обробки даних на транспортному рівні.

Якщо коротко, OGSA - це розподілена взаємодія та обчислювальна архітектура, що базується на сервісах, що забезпечує взаємодію на неоднорідних системах, щоб різні типи ресурсів могли спілкуватися та обмінюватися інформацією. OGSA описується як удосконалення нової архітектури веб-служб, спеціально розробленої для підтримки вимог Grid. OGSA прийнято як мережева архітектура в ряді сіткових проектів, включаючи Альянс Globus.

"Архітектура служб Open Grid Services, версія 1.5" описує сітку OGSA з точки зору наступних можливостей:

- Інфраструктурні послуги;
- Послуги з управління виконанням;
- Послуги передачі даних;
- Послуги з управління ресурсами;
- Служби безпеки;
- Послуги з самоуправління;
- Інформаційні послуги.

ЛЕКЦІЯ 10.
СЛАР - паралельні методи

Методи рішення систем з розрідженою матрицею

Зазвичай кажуть, що матриця розріджена, якщо вона містить O (n) відмінних від нуля елементів. В іншому випадку матриця вважається щільною.

Очевидно, що будь-яку розріджену матрицю можна обробляти як щільну, і навпаки. При правильній реалізації алгоритмів в обох випадках будуть отримані правильні результати, проте обчислювальні витрати будуть істотно відрізнятися. Тому приписування матриці властивості розрідженості еквівалентно твердженням про існування алгоритму, що використовує її розрідженість і робить операції з нею ефективніше в порівнянні зі стандартними алгоритмами.

Багато алгоритми, тривіальні для випадку щільних матриць, в розріджененом випадку вимагають більш ретельного підходу. У багатьох алгоритмах обробки розріджених матриць можна виділити два етапи: символічний і чисельний. На символічному етапі формується портрет результуючої матриці (тобто визначаються місця ненульових елементів в структурі матриці); на чисельному етапі визначаються значення ненульових елементів результуючої матриці.

Існують різні формати зберігання розріджених матриць. Одні призначені для зберігання матриць спеціального виду (наприклад, льон-точних), інші забезпечують роботу з матрицями загального вигляду.

Найбільш очевидним способом зберігання довільної розрідженій матриці є координатний формат: зберігаються тільки ненульові елементи матриці, і їх координати (номери рядків і стовпців). При цьому підході зберігання матриці A можна забезпечити в трьох одновимірних масивах.

Даний спосіб представлення називають повним, оскільки представлена вся матриця A, і неврегульованим, оскільки елементи матриці можуть зберігатися в довільному порядку - витратний і повільний.

Розріджений рядковий формат - це одна з найбільш широко використовуваних схем зберігання розріджених матриць.

Ця схема ставить мінімальні вимоги до пам'яті і в той же час виявляється дуже зручною для кількох важливих операцій над розрідженими матрицями:

- додавання, множення, перестановок рядків і стовпців, транспонування, рішення лінійних систем з розрідженими матрицями коефіцієнтів як прямими, і ітераційними методами і т . д. ;
- масив ненульових елементів матриці A, в якому вони перераховані по рядках від першої до останньої (позначимо його знову як values);
- масив номерів стовпців для відповідних елементів масиву values (позначимо його як cols);
- масив покажчиків позицій, з яких починається опис чергового рядка (позначимо його pointer). Опис k-го рядка зберігається в пози-ціях з pointer [k] -й по (pointer [k + 1] -1) -у масивів values і cols. Якщо pointer [k] □pointer [k + 1], то k-й рядок порожня. Якщо матриця A складається з n рядків, то довжина масиву pointer буде n + 1.

Це - добре для малих операцій.

Розріджений столбцевий формат.

Граф заповнення для розрідженої матриці.

Рішення систем СЛАР

Система m лінійних алгебраїчних рівнянь з n невідомими (чи, лінійна система, також вживається абревіатура СЛАР) в лінійної алгебри - це система рівнянь виду

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

Ax = b

Методи розв'язання алгебраїчних задач можна розділити на точні і ітераційні. Класи завдань, для вирішення яких зазвичай застосовують методи цих груп, можна умовно назвати відповідно класами завдань із середнім і великим числом невідомих.

Проблеми паралельних розрахунків лінійної алгебри:

- матметоди - точні і ітераційні;
- систематичні помилки - з речами близьких значень і розподіл на них;
- зберігання даних і блоковість операцій - швидкість від кеша - дані повинні зберігатися в блоках з послідовним доступом;
- з довільною матрицею А (в основі - метод Гаусса);
- особливі матриці (дійсної симетричною позитивно певної) – Холецкого;
- стрічкові - прогонки, редукції.
- розріджені (проти щільних) системи багатодіагональні блочно-діагональні і т.д.

Метод Гаусса ґрунтується на можливості виконання перетворень лінійних рівнянь, які не змінюють при цьому рішення розглянутої системи (такі перетворення носять найменування еквівалентних). До числа таких перетворень відносяться:

- множення будь-якого з рівнянь на ненульову константу,
- перестановка рівнянь,
- додаток до рівняння будь-якого іншого рівняння системи.

Метод Гаусса включає послідовне виконання двох етапів. На першому етапі, який називається прямий хід, вихідна система лінійних рівнянь за допомогою послідовного виключення невідомих приводиться до верхнього трикутного вигляду. При виконанні зворотного ходу (другий етап алгоритму) здійснюється визначення значень неіз-Вестн.

Прямий хід складається в послідовному виключенні невідомих в рівняннях розв'язуваної системи лінійних рівнянь.

На ітерації i, 1□i <n, методу проводиться виключення невідомої i для всіх рівнянь з номерами k, великих i (тобто i <k□n). Для цього з цих рівнянь здійснюється віднімання рядка i, помноженої на константу aki / aii з тим, щоб результуючий коефіцієнт при невідомої xi в рядках виявився нульовим

При виконанні прямого ходу методу Гаусса рядок, яка використовується для виключення невідомих, носить найменування ведучої, а діагональ-ний елемент провідного рядка називається провідним елементом. Як можна помітити, виконання обчислень є можливим тільки, ес-ли провідний елемент має нульове значення. Більш того, якщо ведучий елемент aii має мале значення, то розподіл і множення рядків на цей елемент може призводити до накопичення обчислювальної похибки та обчислювальної нестійкості алгоритму. Рішення - знаходиться максимум по модулю.

Зворотний хід алгоритму полягає в наступному. Після приведення матриці коефіцієнтів до верхнього трикутного вигляду стає можливим визначення значень невідомих. З останнього рівняння перетвореної системи може бути обчислено значення змінної xn, після цього з передостаннього рівняння стає можливим визначення змінної xn□1 і т.д.

Трудомісткість Гаусса - O (n3).

Паралельний алгоритм

Прямий хід:

-для кожної змінної;

-пошук максимуму в стовпці змінної (умовно паралл);

-виключення змінної з усіх нижніх рядків (паралельно).

Зворотній хід: як тільки визначається значення однієї змінної - відразу використовується у всіх інших.

Бар'єри.

Проблема балансування - трикутність - більше-менше.

Рішення - **стрічкова схема**.

Кожен потік вважає кілька рядків, розташованих в різних місцях трикутника.

LU - розкладання. (Факторизація) Нижня - з одиницями. верхня - з ненулямі.

Метод Холецкого (відомий також як метод квадратного кореня) - призначений для вирішення систем рівнянь виду Ax = b з дійсної симетричною позитивно певної матрицею. Нагадаємо, що матриця називається позитивно певної, якщо для будь-якого вектора x виконано нерівність

Для стрічкових матриць - метод прогонки. Завдання - напр. сполучення сплайнів. Ширина діагоналі. Прямий хід - зворотний хід. Ліва прогін - права прогін. Метод зустрічної прогонки - распараллеливается на два потоки.

Прискорення - в два рази.

Є блоковий алгоритм - на неопр. кол-во потоків. Однак прискорення все одно - тільки в два рази.

Аналіз формул методу прогонки показує, що в деяких випадках метод може давати велику похибку. Потенційним джерелом похибки є формули для обчислення «прогоночних» коефіцієнтів, які містять операцію ділення на різницю близьких за значенням величин.

Метод редукції, (Ті ж діагональні матриці) який буде розглянуто нижче, вільний від цього недоліку і, крім того, при реалізації на сучасних обчислювальних системах він демонструє високу ефективність у порівнянні з методом прогонки. Основне обмеження методу редукції полягає в тому, що він застосовний лише для матриць розміру, рівного ступеня двійки, на відміну від методу прогонки, що застосовується для матриць будь-якого розміру.

Ідея методу редукції полягає в послідовному виключенні з системи невідомих спочатку з непарними номерами, потім з номерами, кратними 2 (але не кратними 4), і т.д., (прямий хід) і відновленні значень непарних змінних на підставі відомих значень змінних з парними номерами (зворотний хід).

Аналіз обчислювальної схеми методу редукції показує, що кожен наступний крок прямого і зворотного ходу залежить від попереднього. Таким чином, распараллелить цілком прямий або зворотний ходу не виходить.

З іншого боку, виключення невідомих на окремому кроці прямого ходу можна проводити незалежно, тому що в цьому випадку залежностей за даними немає. Аналогічно може бути распараллелен окремий крок зворотного ходу, тому що значення невідомих знаходяться незалежно.

На закінчення відзначимо, що при реалізації розглянутого паралельного алгоритму в системах із загальною пам'яттю не повинно виникати ефект «гонки даних», пов'язаний з доступом потоків до однакових областям пам'яті, тому що распаралелені окремі кроки прямого і зворотного ходу не мають залежностей за даними.

ЛІТЕРАТУРА

1 Соммервилл И. Инженерия программного обеспечения. - М.: Вильямс, 2002. -624 с.
2 Бабенко Л.П., Лавріщева К.М. Основи програмної інженерії. Навчальний посібник. – К.: Знання, 2001. - 269 с.
3 Орлов С.А. Технологии разработки программного обеспечения. – СПб.: Питер, 2003. – 480 с.
4 Иванова Г.С. Технология программирования. Учебник для вузов. -М.: изд-во МГТУ имени Н.Э. Баумана, 2002 - 320 с.
5 Бек К. Экстремальное программирование. – СПб.: Питер, 2002. – 224 с.
6 Астелс Д., Миллер Г. Практическое руководство по экстремальному программированию. – К.: Диалектика, 2002. – 320 с.
7 Крачтен Ф. Введение в Rational Unified Process. -К.: Диалектика, 2002. – 240 с.
8 Кендалл С. Унифицированный процесс. Основные концепции. - К.: Диалектика, 2002. – 160 с.
9 Бек К., Фаулер М. Экстремальное программирование: планирование. Библиотека программиста. – СПб.: Питер, 2003. - 144 с.
10 Уилсон С. Принципы проектирования и разработки программного обеспечения. Учебный курс MCSE. - М.: Русская редакция, 2002.- 736 с.
11 Смит К. У., Уильямс Л. Эффективные решения: практическое руководство по созданию гибкого и масштабируемого программного обеспечения. - К.: Диалектика, 2003. – 448 с.
12 Коберн А. Быстрая разработка программного обеспечения. – М.: Лори, 2002. – 314 с.
13 Коберн А. Современные методы описания функциональных требований к системам. – М.: Лори, 2002. – 263 с.
14 Леффингуэлл Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход. - К.: Диалектика, 2002. – 244 с.
15 Бек К. Экстремальное программирование: разработка через тестирование. – СПб.: Питер, 2003. – 240 с.
16 Ларман К. Применение UML и шаблонов проектирования. - М.: Вильямс, 2002. – 624 с.
17 Фаулер С. UML. Основы. – М.: Символ, 2002. – 190 с.
18 Рамбо Дж., Якобсон А., Буч Г. UML. Специальный справочник. - СПб.: Питер, 2002 – 656 с.
19 Гамма Э., Хелм Р. Приемы объектно - ориентированного программирования. Паттерны. - СПб.: Питер, 2003 – 368 с.
20 Влиссидес Дж. Применение шаблонов проектирования. - К.: Диалектика, 2003. – 144 с.
21 Шаллоуей А., Тротт Дж. Шаблоны проектирования. Новый подход к ООП и анализу. - К.: Диалектика, 2002. – 288 с.
22 Аллен Э. Типичные ошибки проектирования. Библиотека программиста. - СПб.: Питер, 2002 – 424 с.

23 Тейт Б. Горький вкус Java. Библиотека программиста. -СПб.: Питер, 2003. – 333 с.
24 Трофимов С.А. Case – технологии: практическая работа в Rational Rose. – М.: Бином –Пресс, 2002. – 288 с.
25 Фаулер М., Бек К. Рефакторинг: улучшение существующего кода. – М.: Символ, 2003. – 432 с.
26 Канер С., Фолк Дж., Нгуен Е. К. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. - К.: Диа-Софт, 2001. - 544 с.
27 Жоголев Е.А. Лекции по технологии программирования. Учебное пособие. - М.: МГУ, 2001. – 160 с.
28 Береза А.М.. Основи створення інформаційних систем: Навч. посібник. – К.: КНЕУ, 2001. - 214с.
29 Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ: Учебник. – М.: ФиС, 1995.- 288 с.
30 Брукс Ф. Мифический человеко-месяц или как создаются программные системы: Пер. с англ. – СПб.: Символ-Плюс, 1999. – 324 с.
31 Йордон Э. Путь камикадзе. – М.: Лори, 2001. – 244 с.
32 Ющенко Е.Л. и др. Многоуровневое структурное проектирование программ. – М.: ФиС, 1989.
33 Вендров А.М. CASE – технологии. Современные методы и средства проектирования информационных систем. -М.:ФиС, 1998. - 176 с.
34 Бєгун А.В. Технологія програмування: об’єктно – орієнтований підхід. -К.: КНЕУ, 2000. - 200 с.
35 Роберт У. Себеста. Основные концепции языков программирования. - М.: Вильямс, 2001. – 672с.
36 Шафер Д.Ф., Фатрелл Р.Т. Управление программными проектами: достижение оптимального качества при минимуме затрат. – К.: Диалектика, 2003. – 1136 с.
37 Ковалёв А., Курдюмов И. Управление проектом по созданию Интернет-сайта. -М.:Альпина Паблишер, 2001. – 337с.
38 Петцольд Ч. Код. Тайный язык информатики. - М.: Русская редакция, 2001. – 512 с.
39 Коналлен Дж. Разработка Web-приложений с использованием UML. – М.: Вильямс, 2001. – 288 с.
40 Эммерих В. Конструирование распределительных объектов. – М.: Мир, 2002. – 510 с.
41 Рихтер Дж. Программирование на платформе Microsoft .NET Framework. -М.: Русская редакция, 2002. – 512 с.
42 Арчер Т. Основы C#. Новейшие технологии. – М.: Русская редакция, 2001.- 448 с.
43 Петцольд Ч. Программирование для Windows на C#. В 2 т. - М.: Русская редакция, 2002. – 1200 с.