УДК 621.391.8

Коновалов О. Ю., к.т.н. *(Київський коледж зв'язку)*

СЕРВІС-ОРІЄНТОВАНА АРХІТЕКТУРА У РОЗПОДІЛЕНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Коновалов О. Ю. Сервіс-орієнтована архітектура у розподілених обчислювальних системах. У роботі розглянута концепція сервіс-орієнтованої архітектури **(**СОА), зв'язаність програмних систем, підходи до розробки розподілених застосувань, принципи побудови СОА, процеси перетворення бізнес-логіки в логіку додатків і реалізація сервісів на основі цих вимог, а також механізми використання сервісів в розподілених обчислювальних системах і організація розподілених застосувань. Виконаний огляд основних підходів до організації і реалізації сервісів у розподілених обчислювальних системах.

Ключові слова: розподілена обчислювальна система, СОА, сильнозв'язані системи, слабозв'язані системи, веб-сервіс, сервіс-орієнтований процес

Коновалов А. Ю. Сервис-ориентированная архитектура в распределенных вычислительных системах. В работе рассмотрена концепция сервис-ориентированной архитектуры **(**СОА), связанность программных систем, подходы к разработке распределенных приложений, принципы реализации СОА, процессы преобразования бизнес-логики в логику приложений и реализация сервисов на основе этих требований, а также механизмы использования сервисов в распределенных вычислительных системах и организация распределенных приложений. Выполнен обзор основных подходов к организации и реализации сервисов в распределенных вычислительных системах.

Ключевые слова: распределенная вичислительная система, СОА, сильносвязанные системы, слабосвязанные системы, веб-сервис, сервис-ориентированный процесс

Konovalov O. Yu. Service-oriented architecture in the distributed calculable systems. The paper considers the service-oriented architecture (SOA) concept, connectivity software systems approaches to the development of distributed applications, the implementation principles of SOA, business processes transformation logic in the logic of the application and implementation of services based on the requirements and mechanisms for the use of services in distributed computing systems and the organization of distributed applications. A review of the main approaches to the organization and realization of services in distributed computing systems.

Keywords: distributed computing system, SOA, strong coupling, loose coupling, web-service, service-oriented process

Вступ. Багато сучасних інформаційних систем (ІС) побудовано на основі триланкової архітектури, ключовим елементом якої є корпоративний сервер додатків [1, 2]. При цьому уся бізнес-логіка додатка розташована на сервері додатків, завдяки чому знижуються вимоги до апаратних засобів клієнтського комп'ютера і, як наслідок, до клієнтського застосування. Враховуючи масштаби сучасних інформаційних систем, вибір і реалізація гнучкої архітектури сервера додатків і самої інформаційної системи є ключовою проблемою, що виникає при створенні програмного забезпечення, і організації розподілених систем.

Характеристика сервіс-орієнтованої архітектури. Сервіс-орієнтована архітектура (Service-Oriented Architecture-SOA) – це парадигма організації і використання розподілених можливостей, які можуть належати різним власникам [3], в основі якої знаходяться принципи виконання певних дій. Складовими сервіс-орієнтованої архітектури являються: *сервісні* компоненти (сервіси); *контракти* сервісів (інтерфейси); *з'єднувачі* сервісів (транспорт); *механізми* виявлення сервісів (регістри).

Сервісні компоненти (чи сервіси) описуються програмними компонентами, що забезпечують прозору мережеву адресацію. Описуючи процеси в інформаційних системах дотримуватимемося визначення, що сервісами називаються відкриті, самовизначальні компоненти, що підтримують швидку побудову розподілених застосувань.

При цьому немає чіткого визначення, який об'єм послуг повинен надавати окремий сервіс, а ці послуги в мережах можуть розрізнятися наступним чином:

– *Програмне забезпечення* як послуга (SaaS, англ. Software-as-a-Service) – модель, в якій споживачеві надається можливість використання прикладного програмного забезпечення провайдера, що працює в хмарній інфраструктурі і доступного з різних клієнтських пристроїв або за допомогою тонкого клієнта, наприклад, з браузеру (веб-пошта) або інтерфейс програми. Контроль і управління основною фізичною і віртуальною інфраструктурою хмари, у тому числі мережі, серверів, операційних систем, зберігання, або навіть індивідуальних можливостей додатка (за винятком обмеженого набору призначених для користувача налаштувань конфігурації додатка) здійснюється хмарним провайдером [4].

– *Платформа* як послуга (PaaS, англ. Platform-as-a-Service) – модель, коли споживачеві надається можливість використання хмарної інфраструктури для розміщення базового програмного забезпечення для наступного розміщення на нім нових або існуючих застосувань (власних, розроблених на замовлення або придбаних тиражованих застосувань). До складу таких платформ входять інструментальні засоби створення, тестування і виконання прикладного програмного забезпечення – системи управління базами даних, єднальне програмне забезпечення, середовища виконання мов програмування - надаються хмарним провайдером. Контроль і управління основною фізичною і віртуальною інфраструктурою хмари, у тому числі мережі, серверів, операційних систем, зберігання здійснюється хмарним провайдером, за винятком розроблених або встановлених застосувань, а також, по можливості, параметрів конфігурації середовища (платформи) [5].

– *Інфраструктура* як послуга (IaaS, англ. IaaS or Infrastructure-as-a-Service) – надається як можливість використання хмарної інфраструктури для самостійного управління ресурсами обробки, зберігання, мереж і іншими фундаментальними обчислювальними ресурсами, наприклад, споживач може встановлювати і запускати довільне програмне забезпечення, яке може включати операційні системи, платформене і прикладне програмне забезпечення. Споживач може контролювати операційні системи, віртуальні системи зберігання даних і встановлені застосування, а також обмежений контроль набору доступних сервісів (наприклад, міжмережевий екран, DNS). Контроль і управління основною фізичною і віртуальною інфраструктурою хмари, у тому числі мережі, серверів, типів використовуваних операційних систем, систем зберігання здійснюється хмарним провайдером [6].

Необхідно враховувати, що при реалізації дрібномодульного сервісу для отримання бажаного результату необхідно забезпечити координовану роботу декількох сервісів. В цьому випадку можна надавати елементарний об'єм функціонального навантаження і забезпечувати високу міру повторного використання. А використання крупномодульних сервісів, дозволяє забезпечити хорошу інкапсуляцію функціональності, але ускладнює повторне використання цих вузькоспеціалізованих сервісів.

Інтерфейс забезпечує опис можливостей і якості послуг, що надаються, конкретним сервісом. У інтерфейсі визначається формат повідомлень, використовуваний для обміну інформацією, а також вхідні і вихідні параметри методів, підтримуваних сервісним компонентом. Від вибору мови і способу опису інтерфейсу залежать можливості програмної сумісності різних реалізацій сервіс-орієнтованої архітектури.

Транспорт забезпечує обмін інформацією між окремими сервісними компонентами. Разом з відкритими стандартами опису інтерфейсів, використання гнучких транспортних протоколів для обміну інформацією між сервісними компонентами дозволяє підвищити програмну сумісність сервіс-орієнтованої системи.

Регістри сервісів використовуються для пошуку сервісних компонентів, що забезпечують необхідну функціональність. Серед усієї множини різних систем, що забезпечують виявлення сервісів [7], можна виділити дві основні категорії: системи *динамічного* виявлення і системи *статичного* виявлення.

Статичні системи виявлення сервісів (наприклад, UDDI) орієнтовані на зберігання інформації про сервіси в системах, що рідко змінюються.

Динамічні системи виявлення сервісів [8] орієнтовані на системи, в яких допустима часта поява і зникнення сервісних компонентів. На сьогодні веб-сервіси найчастіше використовуються для реалізації сервіс-орієнтованої архітектури. Веб-сервіси розроблені для підтримки взаємодії між розподіленими системами за допомогою обчислювальної мережі.

Характеристики веб-сервісів. У [9] дається таке визначення веб-сервісів – це слабозв'язані програмні компоненти, що підтримують багатократне використання, які семантично інкапсулюють окремі функціональні можливості і програмним чином доступні по стандартних протоколах Інтернету. Веб-сервіси незалежні від платформи і мови програмування, оскільки базуються на стандарті XML.

Більшість веб-сервісів використовують HTTP для передачі повідомлень. Це дає значну перевагу при розробці розподілених застосувань в масштабі Інтернет, оскільки зазвичай брандмауери і проксі-сервери без обмежень пропускають HTTP-трафік, і в процесі взаємодії не виникає несподіваних труднощів (які можуть виникнути, наприклад, при використанні технології CORBA).

Розглянемо таке поняття як зв'язаність – міру знання і залежності одного об'єкту від внутрішнього змісту іншого. Відповідно до цього визначення, програмні системи можна розділити на 2 типи: *сильнозв'язані* системи (Strong coupling): Java RMI, .NET Remoting і т.п.; *слабозв'язані с*истеми (Loose coupling): COA.

Сильна зв'язаність виникає, коли залежний клас містить посилання безпосередньо на певний клас, що надає деякі можливості. Слабка зв'язаність виникає, коли залежний клас містить посилання на інтерфейс який може бути реалізований одним або декількома конкретними класами. Основна мета використання концепції слабозв'язаних програмних систем – зменшення кількості залежностей між компонентами. При зменшенні кількості зв'язків, зменшується об'єм можливих наслідків, що виникають у зв'язку із збоями або системними змінами [10, 11].

В табл. 1 зведені характеристики різних типів розподілених систем.

Характеристики типів розподілених систем **Табл. 1**

	Сильнозв’язані системи	Слабозв'язані системи
Фізичні з'єднання	Точка-точка	Через посередника
Стиль взаємодій	Синхронні	Асинхронні
Модель даних	Загальні складні типи	Прості типи
Зв'язування	Статичне	Динамічне
Платформа	Сильна залежність від базової платформи	Незалежність від платформи
Розгортання	Одночасне	Поступове

Традиційний підхід до розробки розподілених застосувань, підтримуваний технологіями розподілених об'єктів, грунтується на тісному зв'язку між усіма програмними компонентами. Слабозв’язаність програмних компонентів, підтримувана технологією веб-сервісів, дозволяє значно спростити координацію розподілених систем і їх реконфігурацію [9, 12]. Важливою особливістю розробки інформаційних систем (чи їх компонентів), покликаної забезпечити їх взаємодію, з метою вирішення певної задачі і отримання певної інформації, являється інтероперабельність. Це визначення об'єднує в собі два поняття: *технічна* інтероперабельність означає сумісність систем на технічному рівні, включаючи протоколи передачі даних і формати їх представлення; *семантична* інтероперабельність – властивість інформаційних систем, що забезпечує взаємну використовуваність отриманої інформації на основі загального розуміння системами її значення.

Особливості проектування СОА. СОА не прив'язане до жорсткої методології проектування, впровадження або управління IT-інфраструктурою. СОА обмежується лише рядом принципів, що характеризують кожен з цих процесів; тому її іноді називають не архітектурою, а архітектурним стилем. Розглянемо деякі з цих принципів [10]:

– ***Розподілене проектування.*** Рішення відносно внутрішніх особливостей інформаційних систем приймаються різними групами людей, що мають власні організаційні, політичні і економічні мотиви.

– ***Постійність змін.*** Окремі ділянки архітектури можуть зазнавати зміни у будь-який момент часу.

– ***Послідовне вдосконалення.*** Локальне поліпшення компонентів архітектури повинне призводити до вдосконалення усієї архітектури в цілому – до зростання сумарної корисності компонентів того ж рівня, що і змінюваний, так само як і компонентів нижчого і більш високого рівня. Наприклад, відомий веб-сервіс Google Translate постійно зазнає зміни. Спочатку, він забезпечував тільки веб-інтерфейс для перекладу і обмежений набір мов. Поступово збільшувалися функціональні можливості сервісу: розширювався набір мов, з'явилася можливість голосового відтворення перекладу, при перекладі окремого слова почали видаватися словникові статті з декількома результатами перекладу і т. п. При цьому API і інтерфейс мінявся настільки мало, що клієнти могли використовувати нові можливості за допомогою

старого API (наприклад, отримання словникових статей) або ж не міняти використовуваний інтерфейс до тих пір, поки не буде потрібно використання нових можливостей.

– ***Рекурсивність.*** Однотипні рішення мають місце на різних рівнях архітектури. З точки зору інформаційних технологій, логіка підприємства [13, 14] може бути розділена на декілька шарів, як показано на рис. 1.

Рис. 1. Еталонна модель SOA foundation

Кожен шар існує окремо від іншого. Структура рішень SOA, показана на рисунку, є еталонною моделлю SOA, що відбиває концептуальний (високорівневий) пристрій рішень SOA. Ця модель, яку іноді також називають "Багатошаровою архітектурою SOA", включає такі шари і поняття, як бізнес-процес, сервіс, сервісний компонент, а також взаємозв'язки між ними. Вона не залежить від технологій, на яких побудована.

Бізнес-логіка є документальною реалізацією бізнес-вимог, які виходять з проблемної області, в якій працює підприємство. Бізнес-логіка, як правило, структурована в процесах, які виражають ці вимоги, а також обмеженнях і залежності від зовнішніх впливів. Логіка додатка – це реалізація бізнес-логіки, організована на основі різних технологічних рішень. Логіка додатка виражає процеси бізнес-логіки за допомогою придбаних або спеціально розроблених програмних систем в умовах обмежених технічних можливостей і залежностей від постачальника рішення.

Процес перетворення бізнес-логіки в логіку додатків і реалізація сервісів на основі цих вимог є процесом створення сервісно-орієнтованої інфраструктури для завдань підприємства. Не існує жорстких правил і концепцій принципів побудови СОА, але при реалізації власної інфраструктури бажано дотримуватися деяких основних підходів, описаних нижче [10, 11].

Сервіси повинні підтримувати повторне використання. СОА-системи повинні підтримувати повторне використання усіх сервісів, незалежно від миттєвих вимог до їх функціональних особливостей. Якщо при розробці системи постаратися максимально

врахувати цю вимогу, то підвищуються шанси значно спростити процес рішення завдань, які неодмінно з'являться в майбутньому, при розвитку системи. Також спочатку орієнтований на повторне використання сервіс дозволяє уникнути розробки "обгортки", яка б підлаштовувала старий сервіс для вирішення нових завдань. Оскільки сервіс – набір зв'язних операцій, логіка кожної індивідуальної операції, що надається сервісом, повинна підтримувати повторне використання [10].

Сервіси повинні забезпечувати формальний контракт використання. Контракт сервісу надає наступну інформацію: *кінцеву точку* (service endpoint): адресу, по якій можна звернутися до цього сервісу; усі *операції*, що надаються сервісом; усі *повідомлення*, підтримувані кожною операцією; *правила* і характеристики сервісу і його операцій.

Сервіси мають бути слабозв'язані. Ніхто не може передбачити, в яку сторону розвиватиметься IT-інфраструктура. Рішення можуть розвиватися, взаємодіяти, замінювати один одного. У зв'язку з цим основним завданням є збереження цілісності системи у рамках такого розвитку, незалежно від змін, що відбуваються. Система сервісів є слабозв'язаною, якщо сервіс може отримувати інформацію про інший сервіс, залишаючись незалежним від внутрішньої реалізації логіки цього сервісу. Це досягається за допомогою використання контрактів сервісів. Слабозв'язаність програмних компонентів, що лежить в основі СОА, дозволяє значно спростити координацію розподілених систем і їх реконфігурацію. Основна мета використання концепції слабозв'язаних програмних систем – це зменшення кількості залежностей між компонентами. При зменшенні кількості зв'язків, зменшується об'єм можливих наслідків, що виникають у зв'язку із збоями або системними змінами. Слабозв'язаність – це не синонім "інкапсуляції" об'єктно-орієнтованої концепції побудови програмних систем. Програмна система може повністю відповідати вимогам інкапсуляції, але бути сильнозв'язаною на семантичному рівні.

Сервіси повинні абстрагувати внутрішню логіку. Кожен сервіс повинен діяти як "чорний ящик", що приховує свої деталі від навколишнього світу. Немає чіткого визначення, який об'єм логіки повинен поміщатися в окремому сервісі. Взаємодія на рівні інтерфейсів є однією з вимог для забезпечення слабкої зв'язаності.

Сервіси мають бути сумісні. Сервіс може як самостійно реалізовувати логіку, так і застосовувати інші сервіси для її реалізації. Сервіси мають бути спроектовані так, щоб підтримувати можливість їх використання як елементи іншого сервісу. Принцип сумісності не залежить від того, чи використовує сервіс для виконання своєї роботи інші сервіси. Як приклад такої взаємодії сервісів виступає концепція оркестрації сервісів. В цьому випадку, сервіс-орієнтований процес (який в принципі може бути визначений як композиція сервісів) управляється сервісом батьківського процесу, який включає інші сервіси, що є учасниками цього процесу. Крім того, принцип сумісності також визначає вид сервісних операцій. Сумісність – це по суті, інша форма повторного використання, і тому операції мають бути стандартними, а для найбільшої сумісності повинні мати необхідний рівень деталізації.

Сервіси мають бути автономними. Властивість автономності вимагає, щоб область бізнес-логіки і ресурсів, використовуваних сервісом були обмежені явними межами. Це дозволяє сервісу самому управляти усіма своїми процесами. Також це усуває залежність від

інших сервісів, що звільняє сервіс від зв'язків, які можуть перешкоджати його застосуванню і розвитку. Питання автономності – найбільш важливий аргумент при розподілі бізнес-логіки на окремі сервіси. Зверніть увагу, що автономність не обов'язково надає сервісу виняткове право власності на бізнес-логіку, яку він інкапсулює. Є тільки гарантія того, що під час виконання сервіс контролює будь-яку логіку, яку він реалізує. Тому ми можемо виділити два типи автономності :

– *Автономність на рівні сервісу*: межі відповідальності сервісів відокремлені, але вони можуть використовувати загальні ресурси. Наприклад, сервіс обгортки, який інкапсулює успадковану програмну систему, яка також кимось використовується (незалежно від цього сервісу), має автономність цього типу. Він управляє успадкованою системою, але також спільно використовує ресурси з іншими існуючими клієнтами.

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

Сервіси не повинні використовувати інформацію про стан. Сервіси повинні зводити до мінімуму об'єм інформації про стан, і час, протягом якого вони її мають. Інформація про стан – це певні дані, що характеризують поточну діяльність. Наприклад, поки сервіс обробляє повідомлення, він тимчасово залежить від стану (stateful). Якщо сервіс несе відповідальність за збереження стану протягом тривалішого часу, його здатність залишатися доступним для інших клієнтів буде ускладнена. Незалежність від стану (statelessness) дозволяє підвищити можливості масштабованості і повторного використання сервісів. Щоб сервіс якомога менше залежав від стану, його операції мають бути розроблені з урахуванням міркувань обробки інформації без даних про стан. Основною особливістю СОА, що підтримує незалежність від стану, є використання повідомлень-документів. Чим вище складність повідомлення, тим більше незалежним і самодостатнім воно залишається.

Сервіси повинні підтримувати виявлення. Виявлення сервісів дозволяє уникнути випадкового створення надмірного сервісу, що забезпечує надмірну логіку. Метадані сервісу повинні детально описати не лише загальну мету сервісу, але і функціональність, що реалізовується його операціями. На рівні СОА виявлення характеризує здатність архітектури забезпечити механізми пошуку, такі як реєстр або каталог. На рівні сервісу, принцип виявлення відноситься до процесу проектування окремого сервісу, так щоб цей сервіс настільки піддавався виявленню, наскільки це можливо.

Одна з важливих переваг архітектури SOA – її надійність, особливо якщо мова заходить про її реалізацію на базі веб-сервісів і специфікації SOAP, що найчастіше використовують протокол HTTP. Річ у тому, що спочатку HTTP не планувався для цілей гарантованої доставки, тому сам по собі він не забезпечує надійного виконання критично важливих транзакцій. Для вирішення цієї проблеми були створені два відносно нові конкуруючі стандарти – WS-RM (Web Services Reliable Messaging) і WS-R (Web Services Reliability), але сучасні реалізації SOA для забезпечення гарантованої доставки використовують більш традиційні методи, наприклад, ESB (Enterprise Service Bus).

У життєвий цикл сервісу входять ті стани, в яких сервіси можуть перебувати, а також ті події, які призводять до зміни цих станів. Під час свого "життя" сервіси проходять через

багато етапів і фаз. Можна уявити собі життєвий цикл сервісів як бізнес-машину із станами (позиціями), в яких можуть перебувати сервіси, і переходами, які переводять сервіси з одного стану в інший. SOA-управління пов'язане з плануванням, визначенням, дозволом і вимірами упродовж життєвого циклу сервісів. SOA-управління визначає, які мають бути сервісні стани, які дії повинні здійснюватися для переміщення із стану в стан (переходи), як (процеси і методи) і ким (посади, контролери). Наприклад, SOA-управління може визначати наступні сервісні стани: ідентифікований, профінансований, специфікований, запрограмований, схвалений, робочий, опублікований, схвалений до вилучення, вилучений.

Далі для підтримки сервісів протягом їх життєвого циклу і для стеження за правильністю протікання процесів, потрібна оболонка SOA. Наприклад, сервісні реєстри і сховища повинні дозволяти робити дії, що ведуть до еволюції сервісів упродовж їх життєвого циклу. Щоб користувачі (і усі, у кого є відповідні права) могли приймати рішення про переміщення сервісів з одного стану в інший, і для сповіщення користувачів про необхідність робити дії, потрібні інструменти для управління дозволами і списками.

Іншим важливим компонентом добре спланованої архітектури SOA є централізований репозиторій, який містить довідник усіх доступних сервісів. Тут підійде будь-яка їх класифікація, яка якнайкраще відповідає нашим потребам. Вона повинна забезпечувати групування сервісів по функціях, підрозділах, джерелам даних або навіть за версією початкового коду. Вказану функціональність в SOA забезпечує реєстр, заснований на стандарті UDDI (Universal Description, Discovery and Integration), який можна розглядати як довідник веб-сервісів. У нім перераховані компанії і пов'язані з ними сервіси (усі використовувані тут стандарти – UDDI, WSDL і SOAP – знаходяться у провадженні організації OASIS [7]).

Висновки. Архітектура SOA абсолютно незалежна від мов програмування, платформ або протокольних специфікацій, за допомогою яких сервіси розробляються, а також від того, де і за допомогою чого вони розгорнуті. Практично архітектура SOA вимагає наявності не лише сервісів, але і засобів, за допомогою яких ці сервіси можуть бути виявлені і підключені незалежно від інфраструктури. SOA – це не продукт або специфікація, тому потрібне ретельне вибудовування цієї архітектури, що складається з безлічі компонентів, таких, як сервери додатків, єднальне ПО, репозиторій і навіть спеціалізовані пакети централізованого управління SOA.

Строго кажучи, SOA не можна відносити ні до реалізації CORBA, ні до архітектури RMI (Remote Method Invocation). Ключовий компонент SOA-сервіс. Сервіси тут є бізнес-функціями, призначеними для забезпечення узгодженої роботи великих застосувань, що складаються з безлічі частин. А кінцевим місцем, розташування сервісів, являється сервер додатків, будь то WebLogic від BEA Systems, WebSphere від IBM, Application Server від Oracle або Java AS від Sun Microsystems.

На відміну від звичайних застосувань сервіс в архітектурі SOA призначається для використання усім реалізованим бізнес-функціям. Тоді як звичайні корпоративні застосування містять в собі схожі фрагменти бізнес-логіки або навіть дублюють окремі об'єкти – наприклад, об'єкт клієнтського замовлення, – в архітектурі SOA треба запустити лише єдиний екземпляр такої бізнес-функції. Таким чином, можна повторно використовувати функціональність в середовищі з множинними застосуваннями і швидко

коригувати бізнес-логіку, для того, щоб мати можливість пристосовуватися до умов ринку, що міняються. У цьому і полягає головна перевага SOA [15].

Література

1. Грэхем И. Объектно-ориентированные методы. Принципы и практика / И. Грэхем ; пер. с англ. – [3-е издание]. – М.: Издательский дом «Вильямс», 2004. – 880 с.

2. Фаулер М. Архитектура корпоративных программных приложений / М. Фаулер ; пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 544 с.

3. OASIS Committee Categories: SOA: [Електронний ресурс] // – Режим доступу : https://www.oasis-open.org/committees/tc_cat.php?cat=soa

4. Software as a Service: Strategic Backgrounder [Електронний ресурс] // – Режим доступу: http://www.siia.net/estore/ssb-01.pdf

5. UP2010 Virtual Cloud Conference – Microsoft's PaaS Solution: [Електронний ресурс] // режим доступу: URL: http://channel9.msdn.com/posts/UP2010-Virtual-Cloud-Conference--Microsofts-PaaS-Solution

6. ORACLE Infrastructure as a service (IaaS) with capacity on demand: [Електронний ресурс] // – Режим доступу: http://www.oracle.com/us/products/engineered-systems/iaas/iaas-exec-brief-1908773.pdf

7. Dijkstra E. W. Cooperating Sequential Processes in Programming Languages, ed. F.Genuys, Academic Press, New York. - NY, 1968.

8. Математика и САПР / [П. Шенен, М. Коснар, И. Гардан и др.]. – М.: Мир, 1988.

9. Cebral J. R. ZFEM: Collaborative visualisation for parallel multidisciplinary applications. Parallel CFD '97: Recent development and advances using parallel computes, Manchester, May 19-21, 1997, Preprints.

10. Радченко Г. И. Распределенные вычислительные системы / Г. И. Радченко. – Челябинск: Фотохудожник, 2012. – 184 с.

11. FG-coud-technical-report: [Електронний ресурс] // – Режим доступу : http://www.itu.int/en/ITU-T/focusgroups/cloud/Documents/FG-coud-technical-report.zip

12. Cloud computing in telecommunications [Електронний ресурс] // – Режим доступу: http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2010/cloudcomputing.pdf

13. Обзор терминологии SOA: Часть 1. Сервис, архитектура, управление и бизнес-термины: [Електронний ресурс] // – Режим доступу : http://www.ibm.com/developerworks/ru/library/ws-soa-term1/

14. Network Virtualisation – Opportunities and Challenges: [Електронний ресурс] // – Режим доступу : http://archive.eurescom.eu/~pub/deliverables/documents/P1900-series/P1956/D1/P1956-D1.pdf

15. Журнал "Сети и Системы связи" [Електронний ресурс] // – Режим доступу : http://www.ccc.ru/magazine/depot/ 06_02/читання.html?0104.htm