МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кваліфікаційна наукова
праця на правах рукопису

ПЕТРЕНКО ОЛЕКСІЙ ОЛЕКСІЙОВИЧ

УДК 004.046: 004.896: 004.942: 004.021:042

ДИСЕРТАЦІЯ
СТРАТЕГІЇ РОЗВИТКУ СЕРВІС-ОРІЄНТОВАНИХ СИСТЕМ У ХМАРНОМУ СЕРЕДОВИЩІ

спеціальність 05.13.12 – системи автоматизації проектних робіт галузь знань:
0501 «Інформатика та обчислювальна техніка»

Подається на здобуття наукового ступеня кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____

(О.О. Петренко)

Науковий керівник: Кисельов Геннадій Дмитрович,
кандидат технічних наук, старший науковий співробітник

Київ – 2018

АНОТАЦІЯ

Петренко О.О. Стратегії розвитку сервіс-орієнтованих систем у хмарному середовищі - Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук (доктора філософії) за спеціальністю 05.13.12 «Системи автоматизації проектних робіт». - Національний технічний університет України «Київський політехнічний інститут імені ігоря Сікорського», Київ, 2018.

Зараз настав час переходу на нову парадигму програмування, пов'язану ні з об'єктами, а з бізнес-процесами і їх складовою частиною - бізнес-функціями. Її ідея - компоновка застосувань шляхом виявлення і виклику сервісів, доступних в мережі, для виконання певної задачі. В результаті можна визначити сервіс-орієнтовану архітектуру (SOA) наступним чином: це архітектура застосувань, побудована на основі формалізованих бізнес-процесів, функції яких представлені у вигляді багаторазово використовуваних сервісів з прозорими описаними інтерфейсами. При цьому створення, впровадження або зміна бізнес-процесу пов’язані з композицією (оркестровкою, хореографією) раніше розроблених сервісів, призначених для автоматизації бізнес-функцій.

Для бізнесу SOA означає прискорене задоволення потреб клієнтів, реальну гнучкість бізнесу, швидкий час виходу на ринок, простоту співпраці і низьку вартість бізнесу. Для IT-організацій SOA означає більшу продуктивність, зниження витрат на IT-рішення за рахунок прискорення розробки застосувань, більш м'якого повторного використання сервісів, більш високої якості застосувань, і в цілому більш швидкого реагування на запити бізнес-клієнтів для поліпшення і модифікації системи.

Дисертація присвячена дослідженню необхідних змін в сервіс-орієнтованих архітектурах (SOA) у зв’язку з загальною тенденцією переносу прикладних застосувань у хмарне середовище, зокрема, в Європейську відкриту наукову хмару. Досліджені базові відмінності традиційних SOA першого покоління (на основі веб-сервісів з уніфікованими протоколами зв’язку) і хмарних MSA нового покоління (на основі мікросервісів з контейнерами), показані доцільні шляхи їх конвергенції з ціллю забезпечення ефективної віртуалізації хмарних апаратних і системних ресурсів. Програмні застосування, що розроблені відповідно до сервіс-орієнтованих технологій, сьогодні реалізуються як набір сервісів (мікросервісів), інтегрованих за

допомогою відомих стандартних протоколів SOAP, REST та інших, як в SOA, чи контейнерної взаємодії сервісів через API Gateway, як в MSA. Головна відміна в реалізації SOA або MSA полягає в тому, що мікросервіси при застосуванні контейнерів не взаємодіють між собою безпосередньо, а тільки через API шлюз, бо кожен мікросервіс містить свою базу даних.

а сьогодні існує два напрями: орієнтація на потреби наукових спільнот, мотивована початком створення з року Європейської відкритої наукової хмари, і орієнтація на потреби бізнесу, пов'язана з стрімким становленням сервісного сектору економіки. першому випадку сервіси, необхідні для створення застосувань, сьогодні переважно формуються на базі успадкованих наукових застосувань. В роботі узагальнюються дані про такі макросервіси, що опубліковані на сайтах EGI, I IGO, E AT, GIA T.

Поки їх небагато, і користувач в змозі за потребі знаходити їх досить просто.

другому випадку сервіси, необхідні для створення застосувань, розробляються переважно самими користувачами за допомогою ефективного і різноманітного інструментарію провідних фірм, таких як Mic o o t, O ac e, P, SAP тощо. При цьому сервіси, як правило, створюються у вигляді веб-сервісів з семантичним описом (повним чи тільки анотованим), що позволяє впровадити методи автоматичного відкриття необхідних сервісів у дуже

Проведений порівняльний аналізу технологій і концепцій SOA і MSA дає підстави визнати більш притаманним для умов хмарного середовища (з точки зору досягнення максимальної віртуалізації ресурсів) використання мікросервісів і контейнерів і організації їх взаємодій. Тому здається конче доцільним рекомендувати надалі (як для наукових, так бізнесових додатків) застосовувати веб-технології для формування та опису мікросервісів і їх композицій, а також контейнерне управлення ними. Перехід на контейнерне управління сервісами за допомогою технології Ope API є по суті альтернативною реалізацією сервісної шини підприємства, що не тільки спрощує виклик сервісів додатками (або їх частинами), але і допомагає їм

передавати дані і поширювати повідомлення про події з позицій оброблення складних подій.

Тому в роботі рекомендується в SOA переходити на контейнерну взаємодію між сервісами, контролюючи розмір веб-сервісів, а в MSA застосовувати семантичне представлення мікросервісів, щоб автоматизувати їх пошук у великих за розміром реєстрах (новий алгоритм пошуку пропонується). Доводиться, що застосування SOA ефективно для інтеграції додатків і інформаційних систем, а MSA ефективно діє на рівні окремого застосування, фокусуючись на його структурі та компонентах.

ікросервісні застосування набагато простіше, ніж традиційні SOA з їх стандартами, включаючи складні сервісні шини підприємства (ESB) у формі проміжного програмного забезпечення, необхідні для спілкування між усіма сервісами. об зробити таке контейнерне управління сервісами найбільш ефективним, досліджені можливі структури (топології) сервісних застосувань. Всі відомі чотири архітектури систем сервісів притаманні SOA і MSA (синхронна SOA подійно-керована E A об'єднана сервіс-орієнтована, керована подіями E SOA мультиагентна), але в їх реалізації є суттєві відмінності.

априклад, із-за відсутності безпосередньої взаємодії мікросервісів утруднена реалізація шаблону запит-відповідь . Для архітектурних шаблонів MSA в роботі сформульовані рекомендації щодо їх застосування та проведено порівняльний аналіз їх переваг і вад. е дозволило запропонувати критерії доцільного вибору архітектури MSA в залежності від призначення і функціональності MSA, при цьому об’єднана EDSOA архітектура рекомендована як основна для побудови складних сучасних систем сервісів.

Певною перевагою MSA додатків є усування труднощів, пов’язаних з використання складної і примхливої сервісної шини підприємства ESB, що

організує і забезпечує взаємодію сервісів в SOA. Замість неї використовується гнучкий I інтерфейс з можливостями автоматичної конфігурації, само-оптимізації, самостійної адаптації та самозахисту функціональності. Іншою перевагою складових MSA додатків є їх здатність до самостійного розгортання за допомогою повністю автоматизованого механізму розгортання і взаємодії між собою через API Gateway. Вони можуть бути написані на різних мовах програмування і використовувати різні типи технологій зберігання даних. Для того, щоб збільшити доступність мікросервісу, можна використати кілька екземплярів примірника мікросервісу разом з балансувальником навантаження. Певною вадою MSA додатків є значне зростання комунікаційного навантаження при їх виконанні завдяки часу, що витрачається на виконання запитів і відповідей при звертанні до мікросервісів, що може серйозно вплинути на час відгуку на запит користувача та його продуктивність.

Зважаючи на стрімке зростання кількості мікросервісів у MSA реєстрах, процедура відкриття (пошуку) сервісів в них ускладняється. Показано, що усі існуючі інженерні методи пошуку і відкриття сервісів об’єднані використанням оцінок задовольняння сервісами реєстру функціональних і нефункціональних вимог, що містяться у запиті користувача. Вони різняться залежно від того, які нефункціональні вимоги враховуються (контекстні, мовної схожості, категорії сервісів, якості сервісів oS), як вони враховуються (поодинці чи у зв’язках, одноразово чи ієрархічно, наперед перевірки функціональних вимог чи паралельно з ними).

зв’язку з цим запропонований гібридний метод пошуку веб-сервісу, який відрізняється тим, що спочатку за категорією сервісу формується звужена область пошуку шляхом відсіювання непричетних до запиту сервісів (виділенням одного з кластерів), а потім проводиться уточнення попереднього вибору застосуванням семантичного порівняння описів відібраних сервісів із запитом, при цьому перевірка відповідності

функціональних і нефункціональних вимог виконується паралельно, завдаки чому скорочується час пошуку сервісів.

а базі об’єднання онтолого-керованого (О) і модельно-керованого (М) підходів розроблено методику проектування систем сервісів як процесу послідовного перетворення концептуальних моделей (IM, PIM, PSM) з поступовим збільшенням впливу параметрів сервісної платформи, яка застосовується для реалізації системи сервісів. При цьому добре визначена доменна онтологія безпосередньо перетворюється на код застосування, на схему баз даних, у абстрактні інтерфейси користувача та інші. кщо при традиційному проектування відповідальними виконавцями є розробники ПЗ, що більшу частину часу працюють з кодом, то при запропонованому підході відповідальними виконавцями є архітектори ПЗ, які більшу частину часу працюють з онтологіями і моделями спільно з експертами в предметній галузі. Використання онтології дозволяє здійснювати ефективну інтеграцію різних додатків й інформаційних систем навіть у випадках, коли ці системи (додатки) будувалися відокремлено, а також генерувати І системи (також шляхом перетворення мов опису).

писано сервіс-орієнтовану архітектуру і перелік сервісів для мобільної медичної системи моніторингу стану і допомоги літнім хронічно хворим людям, які базується на результатах поведених досліджень. і пропозиції зроблені в рамках реалізації національної реформи системи охорони здоров’я в країні.

***Ключові слова*:** сервіс-орієнтована архітектура (SOA), мікросервіс, віртуалізація, контейнер, хмара, інтерфейс API, мікросервісна архітектура (MSA), онтологія, семантика, модельно-керований підхід, Європейська відкрита наукова хмара (EOS).

***Список публікацій здобувача*:**

1. Petrenko A.I. Service-oriented Mobile Healtcare / Petrenko A.I., Petrenko O.O. // LAMBERT AcademicPublishing, Germany, - 2018, - 64 p.
2. Петренко . . ели и об ект науки о сервисах Петренко . . истемні дослідження і інформаційні технології.- 2.2015 - . -82.
3. исельов Г.Д. аука про сервіси, менеджмент та інжиніринг як основа інноваційної діяльності исельов Г.Д., Петренко . . Вісник ніверситету країна , ерія Інформатика, обчислювальна техніка та кібернетика , ( ), , стр. -36.
4. Петренко, . . Порівняння типів архітектури систем сервісів истемні дослідження і інформаційні технології - . . - . -62.
5. Петренко І. . втоматизовані методи пошуку і відкриття необхідних сервісів Петренко І. ., Петренко . . Вісник ніверситету країна , ерія Інформатика, обчислювальна техніка та кібернетика , ( ), , . -64.
6. Петренко . . Підготовка кадрів для індустрії сервісів . . Петренко ISSN 1998- . Інформаційні технології в освіті.- 2015.- . - . - 165.
7. Петренко . . емантическое модельно-управляемоем оделирование архитектур систем сервисов Петренко . ., Петренко . . лектронное моделирование. - , , .
8. Петренко . . собливості реалізації сервіс-орієнтованих додатків у хмарі истемні дослідження і інформаційні технології - . , . - 33.
9. Petrenko O.O. Cyber-Physical Medical Platform for Personal Health Monitoring /Petrenko OO and Petrenko AI. //JournalofScientificAchievements, August 2017; 2 (8): 24-28.
10. Petrenko O.O, Petrenko A.I. A Model-driven Ontology Approach for Developing Service System Applications / Petrenko O.O. Petrenko A,I. // Journal of Computer Science Applications and Informftion Technology, 2017, 2(4): 1-7.

11. O.O. Petrenko. A loosely integrated suite of services for mHealth formed by wireless sensor networks / O.O. Petrenko, A.I. Petrenko //2016 IEEE First International Conference on Data Stream Mining& 23-27 August 2016, Lviv, Ukraine, Lviv Polytechnic Institute.- pp. 3-8. -IEEE CatalogNumber: CFP16J13-POD.- ISBN: 978-1-5090-3737-7.

12. Петренко О.О. оделювання архітектури системи сервісів Петренко О.О., Петренко .І. Праці міжнародної науково-технічної конференції оделювання- , м. иїв (країна), травень .

13. Петренко . . емантичний пошук зображень за їх змістом. истемний аналіз та інформаційні технології: -я міжнародна науково-технічна конференція ІТ-2014», 26- травня року, иїв, країна: матеріали. - .: ІП Т ПІ , . . -313.

14. Петренко . . ластеризація зображень за їх анотаціями.- истемний аналіз та інформаційні технології: -я міжнародна науково-технічна конференція ІТ-2014», 26- травня року, иїв, країна: матеріали. — . –307.

15. Петренко . . аука про сервіси, управління і інжиніринг // истемний аналіз та інформаційні технології: 17-я міжнародна науково-технічна конференція « ІТ-2015», 2015 року, иїв, країна: матеріали. -- .: "ІП " Т " ПІ", 2015. . 321-322.

ANNOTATION

Petrenko O.O. Strategy development of service-oriented systems in a cloud environment.– Manuscript

Thesis for candidate of science degree in the specialty 05.12.13–System of automation of design works.–National Technical University of Ukraine "Kyiv Polytechnic Institute named after Igor Sikorsky", Kyiv, 2018.

Now is the time to move to a new programming paradigm, which is not related to objects, but deals with business processes and their constituent parts - business functions. Its idea is to arrange applications by discovering and calling services available on the network to perform a specific task. As a result, Service Oriented Architecture (SOA) can be defined as follows: this is an application architecture which is built on the basis of formalized business processes whose functions are presented in the form of reusable services with transparent, described interfaces. At the same time, the creation, implementation or change of the business process are associated with the composition (orchestration, choreography) of previously developed services designed to automate business functions.

For business SOA means accelerated customer satisfaction, real business flexibility, fast entry times, ease of cooperation, and low business value. For IT organizations SOA means greater productivity, lower costs for IT solutions by accelerating application development, milder reuse of services, higher quality applications, and generally more responsive to business customer requests for system upgrades and modifications .

Software applications developed in accordance with service-oriented technologies are now implemented as a set of services (microservices) integrated with known standard SOAP, REST and other SOA protocols, or container-based services interactions via the API Gateway, as in MSA. The main abandonment in the implementation MSA is that the microservices in the application do not interact directly, but only through the API gateway, because each microservice contains its database.

Today, there are two directions: *the orientation towards the needs of scientific communities*, motivated by the start of the creation of the European open science cloud since 2016, and t*he orientation towards business needs*, which is connected with the rapid formation of the service sector of the economy. In the first case, the services for creating applications today are mostly formed on the basis of inherited scientific applications. The dissertation summarizes data on such macro-services published on EGI, INDIGO, EUDAT, GIANT sites.While they are

a little bit, the user is able to find them quite simply as needed. In the second case, the services needed for applications developing are developed mainly by the users themselves through the efficient and diverse tools of leading firms such as Microsoft, Oracle, HP, SAP, etc. In this case, services are usually created in the form of web services with a semantic description (full or only annotated), what allows the introduction of methods for automatically opening the necessary services in very large in scope of repositories services.Therefore, it is recommended in SOA to switch to container interaction between services and to controla size of web services, and in MSA - to apply a semantic description of the micro-services to automate their discovery in large-sized registries (theexpedient search algorithm is suggested).

Micro-service applications are much simpler than traditional SOA with their standards, including sophisticated enterprise bus services (ESBs) in the form of intermediate software, necessary for communication between all services. To make such container management services the most effective, possible structures (topologies) of service applications were investigated. All known four service system architectures are inherent in SOA and MSA (synchronous SOA; event-driven EDA; combined service-oriented, EDSOA-driven, multiagents MA), but there are significant differences in their implementation.

For example, due to the lack of direct micro-service interaction the synchronous SOA with query-response is difficult for realization. For all architectural templates MSA in the dissertation recommendations for their application areformulated and a comparative analysis of their advantages and disadvantages is given. This allowed to propose the criteria for an appropriate choice of MSA architecture, depending on the purpose and functionality of the MSA, while the combined EDSOA architecture is recommended as the basic for building complex modern service systems.

A certain advantage of MSA applications is the elimination of the difficulties associated with the use of the sophisticated and capricious ESB, which

organizes and provides interoperability services in the SOA. Instead, a flexible API interface with automatic configuration, self-optimization, self-adaptation and self-protection functionality is used. Another advantage of the components of MSA applications is their ability to self-deployment using a fully automated deployment mechanism and interoperability through API Gateway. They can be written in different programming languages and use different types of data storage technologies. In order to increase the availability of the microservice, several instances of the copy of the microservice together with the load balancer can be used. A particular disadvantage of MSA applications is the significant increase in communication load due to the time spent on micro-services' requests and responses, which can seriously affect the response time to the user's request and its performance.

Due to the rapid growth in the micro-services number in MSA registers, their discovering becomes more difficult. It was shown that all existing methods of service search and discovery are base on estimation of marching functional and non-functional requirements contained in user queries and service descriptions. They vary depending on what non-functional requirements are taken into account (contextual, linguistic similarities category of services, quality of service QoS), or how they are captured (alone or in connection once or hierarchically, before or together with checking functional requirements).

In this connection, the proposed hybrid method of searching for a web service is characterized by the fact that the service clustersare firstly formed to narrow the search area by discarding unrelated services requests, and then refining the pre-selection is provided by using the semantic comparison of descriptions selected services with the request, while checking the compliance of functional and non-functional requirements is *performed in parallel* resulting in reducing the time of service search.

Based on the combination of Ontologically Managed (OD) and Model-Managed (MD) approaches, a methodology for designing service systems is proposed as a process for the sequential transformation of conceptual models

(CIM, PIM, and PSM) with a gradual increase in the influence of the parameters of the service platform that is used to implement the developed services system. In this case, a well-defined domain ontology directly transforms into a executing code, a database schema, abstract user interfaces, and others. When in the traditional design responsible implementers are software developers who work most of the time with the code, then under the proposed approach responsible implementers are architects of software, which most of the time working with ontologies and models together with experts in the subject field. The use of ontology allows to simplify integration of various applications and information systems, even when these systems (annexes) were built separately, as well as to generate APIs of the system (also by transforming the description of models).

The service-oriented architecture and the list of services for the mobile medical system for state monitoring and assistance to elderly chronically ill people based on the results of the conducted research are described. These proposals are made within the framework of the implementation of the national reform of the health care system in Ukraine.

Keywords: service-oriented architecture (SOA), microservice, virtualization, container, cloud, API interface, microservice architecture (MSA), ontology, semantics, model-driven approach, European open science cloud (EOSC).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

CA – Contextual Algorithm

CDI – Collaborative Data Infrastructure

CIM - Computing Independent Model

CLM - Closed Loop Marketing

GEO – Group on Earth Observations

GEOSS – Global Earth Observation System of Systems

GUI – Graphical User Interface

EDA – Even Driven Architecture

EUDAT – European Data Infrastructure

EGI – European Grid Initiative

ESB – Enterprise Service Bus

FaaS – a function as a service

FIPA – Foundation for Intelligent Physical Agents

IaaS – infrastructure as a service

INDIGO – INtegrating Disrtibuted data Infrastructures for Global ExplOitation;

IWMAS – Internet Wide Multi-Agent System

JSON - JavaScript Object Notation, текстовий формат обміну даними між комп'ютерами

JSP - Java Server Pages

JVM - Java Virtual Machine

MAS – Multi-Agent System

MSA – microversice-oriented architecture

ODIS – Ontology Driven Information System

ODMAS – Ontology Driven Multi-Agent System

OWL – Web Ontology Language

OWL-S – Ontology Web Language for Services

QoS – Quality of Service

PaaS – a platform as a service

PIM - Platform Independent Model

PSM - Platform Specific Model

RA – Registry Agent

RDF – Resource Description Framework

RDFS – Resource Description Framework Schema

SaaS – software as a service

SAS – Sensor Alert Service

SAWSDL – Semantic Annotations for WSDL

SensorML – Sensor Model Language

Service discovery – the act of finding the network location of a service instance for further use.

Serverless computing – A cloud computing model in which a provider manages the allocation of servers and resources

SLA – Service Level Agreement

SOAP – Simple Object Access Protocol

SOA – Service-oriented architecture

SWAGGER – A definition format used to describe and document RESTful APIs

SWE – Sensor Web Enablement

SWS – Semantic Web Services

UDDI – Universal Description, Discovery and Integration

UML – unified modeling language

UNSPSC - United Nations Standard Products and Services Code

URI – Uniform Resource Identifier

WNS – Web Notification Service

WSML – Web Service Modeling Language

WSMO – Web Service Modeling Ontology

WSMX – Web Service Modeling eXecution environment

XML – Extensible Markup Language

ВСТУП

Актуальність теми дослідження визначається тим, що сфера послуг є найбільш *швидкозростаючим сегментом світової економіки,* в якому вже сьогодні зайнято більше половини працездатного населення планети [1,2,6]. Системи сервісів є динамічними конфігураціями людей, технологій, організацій та засобів обміну інформацією, які створюють і забезпечують цінність для користувачів, постачальників та інших зацікавлених сторін.

Сфера послуг потребує *створення своєї наукової бази*, яка може містити наступні чотири основні блоки: *основи науки про сервіси, інженерію сервісів, менеджмент сервісів і технологію реалізації сервісів.* Створення такої наукової бази розпочалося в 2008 році з ініціативи фірми IBM (автори P. Maglio, C. Kieliszewski, J. Spohrer, https://paulallen.ca/documents/2015/ 08/maglio-pp-and-spohrer-j-fundamentals-of-service-science-2008.pdf). Нова наука базується на відомій інновації в IT-технологіях: ***сервіс-орієнтованій архітектурі SOA*** (Service-Oriented Architecture) як стилю архітектури програмних систем. Дана архітектура представлена у вигляді набору сервісів і процесів, які можна комбінувати, а також змінювати з часом відповідно до змін вимог з допомогою планувальників потоку завдань (workflows). Значна частина досліджень з SOA виконана в рамках європейського науково-дослідницький проектів (наприклад, ADVANCE, SENSORIA, ASSERT4SOA, IPCIS, DEPLOY, інші), що проводилися в 2005-2016 рр. за підтримки Європейських рамкових програм FP6 і FP7, в публікаціях B. Stauss, K. Engelmann, A. Kremer, A. Luhn, H. Demirkan, V. Krishna, C. Mohan, Laura C. Anderson, Martin Fowler, James Lewis та інших. Є серед досягнень і скромний внесок вітчизняних фахівців з Інституту програмних систем НАНУ (П. І. Андон, В. Дерецкий) та НТУУ «Київський політехнічний інститут імені Ігоря Сікорського» (М. З. Згуровський, А. І. Петренко, Л.С. Глоба, Б.В. Булах).

Саме на сервіс-орієнтованому підході базується сьогоденний найбільший європейський проект зі створення ***Європейської відкритої науково-дослідницької хмари*** (European Open Science Cloud for Research, EOS), що

розпочався у 2017 році і який мотивує дослідження технологій розміщення численних SOA прикладних додатків у хмарі, яка незабаром буде обслуговувати 1,7 млн. вчених і 80 млн. професіоналів з різних галузей науки і технологій.

Дослідження базових відмінностей традиційних SOA першого покоління (на основі веб-сервісів з уніфікованими протоколами зв'язку) і хмарних SOA нового покоління (на основі мікросервісів з контейнерами), а також можливостей їх конвергенції є актуальною задачею, яка на сьогоднішній день потребує вирішення.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалась на кафедрі системного проектування НТУУ «КПІ» згідно з планами дослідження за темами: НДР «Дослідження нової концепції побудови динамічної архітектури проблемно-орієнтованого програмного забезпечення в грід-хмарному середовищі з елементами постбінарних обчислень» (№ 2710-ф, номер державної реєстрації 0114U003449, терміни виконання: 2014-2016 рр.) і НДР «Проектування сучасних систем сервісів на прикладі мобільної медичної системи для мешканців прифронтових районів в зоні АТО» (№2020-п, номер державної реєстрації 0117U002435, терміни виконання 2017-2020 рр.).

Мета та задачі наукового дослідження. Метою дисертаційної роботи є теоретичне обґрунтування стратегій і напрямків розвитку сервіс-орієнтованої архітектури і моделей систем сервісів для створення конкретних прикладних додатків у зв'язку з загальною тенденцією переносу IT-рішень у хмарне середовище. Ця *робота є однією з перших вітчизняних робіт з розвитку науки про сервіси* і ставить собі за мету ретельний розгляд і вирішення наступних задач, що були сформовані на базі розглянутого сьогоденного стану досліджень і розробок в галузі сервісних хмарних технологій:

1) дослідження спадкоємності між поколіннями SOA, базованими на композиції веб-сервісів і на поєднанні мікросервісів з контейнерами; для доведення того, що ці покоління поділяють спільні принципи і створені для

досягнення спільних цілей, при цьому мікросервісна архітектура (MSA) може розглядатися в якості "тонкоструктурної (fine-grained) SOA" або " продукту еволюції SOA";

2) порівняльного узагальнення можливостей, властивостей і обмежень макросервісів, які формуються зараз з існуючих монолітних рішень і рекомендуються для включення до 2020 року в каталог Європейської відкритої наукової хмари;
3) порівняльного оцінювання можливостей застосування і сумісності (інтероперабельності) існуючого інструментарію провідних фірм для створення мікросервісних реєстрів і забезпечення тим самим реалізації хмарних SOA;
4) дослідження доцільності застосування семантичних описів мікросервісів для організації автоматичного пошуку (відкриття) необхідних сервісів в суттєвого збільшених за розмірами хмарних репозитаріях сервісів;
5) дослідження підходів до автоматизації проектування структури SOA, а значить, і обґрунтування доцільного вибору необхідних для її реалізації сервісів, виходячи з доменних онтологій предметної галузі, або бази знань;
6) дослідження можливостей з удосконалення процедури семантичного пошуку сервісів в репозитаріях за запитом користувача на базі порівняння їх семантичних описів, а також раціонального використання властивостей і відмінностей різних сервіс-орієнтованих архітектур;
7) дослідження еквівалентності функцій сервісної шини підприємства в SOA і API керування в контейнерному підході з точки зору оброблення складних подій;
8) розроблення структури і базових сервісів для хмарної системи;
9) мобільної медичної допомоги пацієнтам з хронічними захворюваннями.

Об'єктом дослідження є процес проектування сервіс-орієнтованих систем для хмарного середовища.

Предметом дослідження є архітектура сервіс-орієнтованих систем, що базується на онтології предметної галузі.

Методи дослідження. Досягнення поставленої мети реалізовано з використанням методів семантичного веб, методів інженерії знань, теорії прийняття рішень, дескриптивної логіки, машинного навчання, багатокритеріального аналізу.

Наукова новизна дисертаційної роботи полягає в тому, що:

1) Проведено аналіз сучасного стану сервісних технологій, базованих на композиції веб-сервісів з уніфікованими протоколами зв'язку (перше покоління SOA) і на поєднанні мікросервісів з контейнерами (друге покоління MSA) та *вперше* показано, як їх властивості підлягають ***конвергенції***, оскільки кожен з них має свої переваги і недоліки. При цьому хмарне середовище і вимоги до віртуалізації його апаратних і системних ресурсів *сприяють переходу до контейнерного виконання веб-сервісів і їх взаємодії через API* замість традиційних уніфікованих протоколів зв'язку, до зменшенню розмірів самих веб-сервісів з ціллю розгортання кількох з них в одному контейнері, що призводить до економії апаратних ресурсів (кількості задіяних віртуальних машин ВМ*).*

2) ***Вперше доведено***, що для MSA можна реалізувати всі відомі архітектури (сервіс-орієнтовану з синхронними зв'язками між сервісами, подійно-керовану EDA; гібридну сервіс-орієнтовану, керовану подіями EDSOA; агентно-керовану), що відомі з практики SOA першого покоління. Показано, що для MSA базовою є подійно-керована модель, в той час як для SOA – модель з синхронними зв'язками між сервісами.

3) ***Вперше*** на базі узагальнення існуючих методів семантичного відкриття сервісів ***запропоновано*** будувати сервіс пошуку сервісів у вигляді гібридної системи шляхом попереднього звуження за категорією сервісу області пошуку і відсіювання непричетних до запиту сервісів (тобто виділення відповідного кластера), і наступної оцінки відповідності функціональних вимог (входів та виходів, передумов та ефектів) і нефункціональних вимог до сервісу (контекстних, мовної схожості, категорій сервісів, якості сервісів QoS), взятих з запиту користувача. При цьому така оцінка завдяки базовій

подійно-керованій архітектурі MSA виконується ***паралельно***, а не поодинці чи роздільно, як то реалізовано сьогодні. Це призводить до скорочення часу пошуку сервісів і робить можливим застосування семантичних описів мікросервісів і семантичних методів їх відкриття в умовам стрімкого зростання розмірів хмарних реєстрів таких сервісів.

4) ***Удосконалено*** методику розробки систем сервісів, яка базується на об'єднанні онтолого-орієнтованого (OD) та модельно-керованого (MD) підходів до моделювання архітектури бізнес-процесів, включенням в неї процесу перетворення бізнес-логіки у логіку додатків. Цей процес супроводжується перетворенням онтології бізнес-процесів у послідовні моделі (CIM, PIM, PSM) з поступовим збільшенням впливу параметрів сервісної платформи, яка застосовується для реалізації системи сервісів.
5) Сформульовані рекомендації з вибору сервіс-орієнтованої архітектури і самих сервісів для мобільної медичної системи моніторингу стану і допомоги літнім хронічно хворим людям в рамках реалізації національної програми перебудови системи охорони здоров'я в Україні. Запропонований репозитарій сервісів відрізняється тим, що має універсальний характер (особливо в галузі сервісів управління платформою і сервісів додатків лікаря і пацієнта), тому на його базі можна з мінімальними витратами розгортати мобільні системи різного призначення і адаптувати їх до завдань підтримки конкретних планів лікування хворих шляхом масштабування сервісів: виключення деяких з них, додавання нових, заміни одних на інші такого ж призначення.

Обґрунтованість і достовірність наукових положень, основних висновків і результатів дисертаційної роботи забезпечується аналізом стану досліджень в проблемній області, узгодженістю теоретичних висновків з результатами експериментальної перевірки моделей на наборі сценаріїв, а також апробацією основних теоретичних положень дисертації в друкованих працях та доповідях на міжнародних наукових конференціях.

Практичне значення отриманих результатів

Практична цінність результатів роботи полягає в наступному:

- Поведено теоретичне обґрунтування, розробка та розвиток сервіс-орієнтованої архітектури і моделей систем сервісів для вирішення завдань пошуку, композиції і оркестрування сервісів та побудови їх робочих потоків (workflows) для підтримки певних бізнес-процесів. Це певний внесок даної роботи в нову прикладну науку про сервіси, їх менеджмент та інжиніринг.
- Розроблені рекомендації користувачам, які бажають створювати свої невеличкі власті хмарні додатки, з застосування технологій MSA, базованих на використанні мікросервісів і контейнерів, включаючи вибір архітектури мікросервісної системи.
- Сформовані рекомендації користувачам, які бажають перенести в хмару свої попередні напрацювання, з попереднього перетворення своїх наявних монолітних додатків у сукупність веб-сервісів і наступного розгортання їх в контейнерах хмари з можливістю автоматичного пошуку.
- Обґрунтовані рекомендації підприємствам, які бажають перетворити всі свої IT-системи на хмарні додатки в інформаційному середовищі підприємства, з одночасного використання MSA і SOA підходів: SOA переважно для інтеграції додатків і інформаційних систем, а MSA на рівні окремого додатка.
- Запропоноване застосування сервіс-орієнтованого підходу до проектування мобільної медичної платформи персонального моніторингу і лікування пацієнтів з хронічними захворюваннями і сформовано перелік сервісів пацієнта, лікаря і підтримки самої платформи як пропозицію до третього етапу національної реформи медицини.

Виконана дисертаційна робота допоможе зорієнтуватися фахівцям предметних галузей, *яким чином найкраще наповнити своїми конкретними прикладними додатками Європейську відкриту наукову хмару (EOSC)*, що

розпочато створювати у 2016-17 роках, зробити свій вибір в залежності від масштабів задач, які їм потрібно вирішувати.

Особистий внесок здобувача. Дисертаційна робота є узагальненням результатів теоретичних і експериментальних досліджень, проведених автором самостійно. Базові роботи [2, 4, 6, 8, 13, 14, 15] написані автором дисертації особисто. У роботах, опублікованих із співавторами, дисертанту належать: [1] – систематизація заходів з проектування сервісів; [3] – аналіз методів науки при сервіси, менеджмент та інжиніринг; [5] – аналіз методів відкриття сервісів; [7] – розвиток онтологічного підходу до моделювання систем сервісів; [9] - розробка наукових основ проектування медичних мобільних сервісів; [10] – поєднання модельно-керованого і онтолого-орієнтованого підходів до проектування систем сервісів; [11] – розробка наукових основ проектування медичних мобільних сервісів; [12] – систематизація заходів з проектування сервісів.

Апробація результатів дисертації

Результати досліджень, які включено в дисертацію, доповідалися та обговорювалися на наступних міжнародних науково-технічних конференціях: Міжнародній науково-технічній конференції «Data stream mining&processing» (’DSMP’2016), м. Львів (Україна), вересень 2016; Міжнародній науково-технічній конференції «Моделювання-2016», м. Київ (Україна), травень 2016; Міжнародній науково-технічній конференції «Системний аналіз і інформаційні технології», м. Київ (Україна), 2014, 2015, 2016 рр.

Публікації

За результатами досліджень опубліковано 15 наукових праць, у тому числі 1 монографія, 9 статей у наукових виданнях (з них 2 статті у виданнях іноземних держав, а 7 у фахових виданнях України, які включені до міжнародних науко-метричних баз), 5 тез доповідей в збірниках матеріалів конференцій. Сім робіт з 15-ти написані автором дисертації особисто.

РОЗДІЛ 1. СЬОГОДЕННИЙ СТАН ДОСЛІДЖЕНЬ ТА РОЗРОБОК У ГАЛУЗІ СЕРВІСНИХ ТЕХНОЛОГІЙ

1.1. Визначення термінів «сервіс» і «наука про сервіси»

Класичне визначення терміну «сервіс» дано фірмою ІБМ у вигляді: «сервіс - це видимий ресурс, що виконує повторюване завдання і описаний зовнішньої інструкцією» [1]. Цей «сервіс» має два основні загально прийняті значення: *економічне*, бізнес-орієнтоване значення, як, наприклад, у виразі "сервісний сектор економіки», та *IT – орієнтоване* значення, як у виразах "веб-сервіс" або "сервіс-орієнтована архітектура SOA". У першому випадку акцент робиться на взаємодію при обміні і на нематеріальний характеру сервісу, а у другому – на технології програмного забезпечення, які дозволяють підтримувати сумісність різних програмних модулів.

В сучасних системах сервісів для економіки ці два значення зливаються, оскільки впровадження сучасної системи сервісів передбачає також інтеграцію інформаційних систем як підсистем організаційної системи систем [1]. В той же час зростає сервісний домен з орієнтацією на сучасні виробничі процеси, наприклад, через «сервісацію» виробництва [2, 3]. Орієнтація на сервіси, наприклад, дозволяє гнучко реалізовувати і пов'язувати сервісами бізнес-процеси з однієї галузі бізнесу, різних галузей, а також бізнес-партнерів. Сервіси також сприяють інтеграції виробничих процесів з інформаційними системами і конкретними IT-технологіями.

Поширення інтегрованого Інтернету, корпоративні додатки, користувальницький досвід, мережі та мобільні сервіси надають більше можливостей для підвищення якості сервісів. Тенденція зосередження на діяльності у сфері сервісів у всіх галузях бізнесу сприяє інновації сервісів, що, в свою чергу, призвело до появи ***науки про сервіси, управління та інжиніринг (SSME***), яка стає основною частиною нових бізнес-моделей в останні роки [3-7]. В результаті SSME в якості дослідницької дисципліни набула глобального впливу. Теми досліджень у цій галузі включають в себе: теорії і концепції науки

про сервіси; методології досліджень, застосування, моделювання та аналізу; розвиток систем сервісів; перетворення бізнес-процесів; проектування сервісів та їх інновації; процес синтезу сервісів, надання сервісів клієнтам; організаційні структури систем сервісів; тематичні дослідження; поведінкові дослідження та розробку навчальних програм. Під *науковою складовою* SSME при цьому розуміють узгоджені методи і стандарти, суворе використання яких науковою спільнотою дозволяє розроблювати сукупність знань про сервіси; під *інженерною складовою* – спосіб застосування знань для створення систем сервісів; під бізнесом – спосіб застосування знань для отримання прибутку від систем сервісів, *під менеджментом* – спосіб покращення процесів створення і розповсюдження систем сервісів.

З точки зору SSME сервіс, що генерується в системі сервісів, пропонує *ціннісні пропозиції* іншим системам сервісів. Таким чином, можна розглядати *сервіс в якості підсистеми системи сервісів* і вивчити його взаємопов'язані компоненти (наприклад, заходи оцінки сервісів, результати обслуговування, вартості пропозиції, сервісні взаємодії та інші) або *як бізнес-процес*, який сам є динамічною системою з дискретними подіями, коли вивчається потік взаємодій і активностей. Інтеграція теоретичних перспектив забезпечується системним підходом до сервісів, що пов'язує концепцію системи сервісів з більш загальною концепцією організаційних систем, складених з систем.

Наука про сервіси може бути визначена як узгодження методів і стандартів, застосовуваних науковим співтовариством для відображення знань, які включають класи спостережуваних явищ, у вигляді концепцій, теорій, моделей і законів, які можуть бути емпірично перевірені і застосовані на благо суспільства.

Наука про сервіси може розглядатися як композиція або інтеграція багатьох галузей досліджень або дисциплін, таких як менеджмент сервісів, маркетинг сервісів, сервісні операції, інженерія сервісів, обчислення сервісів, менеджмент сервісів кадрової служби, сервіси економіки, менеджмент інновацій сервісів, джерела поставок сервісів і укладення контрактів (eSourcing), та інші. Можна зробити чотири попередніх зауваження з приводу цих багатьох дисциплін [8-10].

По-перше, у тій чи іншій спосіб вони всі *зв'язані з ресурсами різних типів*. Наприклад, менеджмент сервісів кадрової служби має справу з інформацією та інформаційними технологіями, а джерела поставок таких сервісів і укладення контрактів – з людськими ресурсами взаємодіючих організацій. Люди, інформація, технології та організації можуть розглядатися в якості різних ресурсів з власними регламентами щодо шляхів їх застосування або конфігурування для створення цінності.

По-друге, деякі з дисциплін більше, ніж інші *намагаються інтегрувати і координувати ресурси* для конкретних цілей, наприклад, менеджменту сервісів, інженерії сервісів та менеджменту інновацій послуг.

По-третє, *вимірювання* важливо для більшості дисциплін, хоча критерії адекватного виміру можуть варіюватися серед різних дисциплін, наприклад, економікою сервісів, менеджментом сервісів та інженерією сервісів.

По-четверте, кожна дисципліна отримала у додаток слово «сервіс». Суспільство переходить від бачення цінності товарів (фізичних речей) до бачення цінності в наданні сервісів (застосування знань через типи відносин на основі взаємної вигоди). Це особливо проблематично, тому що термін "сервіс" викликає багато неправильних уявлень і стереотипів. Наприклад, державна статистика, яка показує зростання в сервісному секторі, базується на підрахунку кількості робочих місць виробничих компаній в самих компаніях і не враховує частину робочих місць, розташованих поза компаніями і пов'язаних з обслуговуванням цієї компанії. З іншого боку, пересічний мешканець сьогодні, швидше за все, асоціює поняття "працівник сервісу" з кимось, хто працює в ресторані швидкого харчування, ніж з тим, хто працює в якості викладача в університеті.

Наука про сервіси, розвиток якої досить інтенсивний, має за своєю суттю *міждисциплінарні ознаки* з довгостроковою метою стати дійсно міждисциплінарною галуззю. Міждисциплінарна мета буде реалізована тільки інтегруванням окремих дисципліни у нове ціле. Ключовою рушійною силою науки про сервіси є бажання промисловості отримати (у потенційно величезних кількостях) новий тип сервісних професіоналів. Новий професіонал повинен мати

глибокі знання в кількох існуючих дисциплінах, а також мати навички з інтегрованої науки і мистецтва дизайну сервісів та реалізації цінностей шляхом об'єднання технологій, бізнес-моделі і соціально-організаційних інновацій для поліпшення бізнес-та соціальних систем.

1.2. Бізнес-орієнтований погляд на сервіси

У всіх країнах світу відбувається макроекономічний перехід від виробництва фізичних речей (сільське господарство і промислові товари) до виконання сервісів з обслуговування населення. Не існує ніяких сумнівів, що сервісна економіка і застосування сервісів швидко зростають не тільки в розвинених країнах, але також у деяких країнах, що розвиваються, таких як Китай та Індія. А в найбільш розвинених країнах більше 70% ВВП формується індустрією сервісів, в якій зайнято сьогодні (за інформацією Міжнародної організація праці) більше половини людства.

Термін "сервісний сектор" колись в основному був пов'язаний з некваліфікованою працею, з низьким рівнем оплати трудомістких видів праці в таких галузях, як оптова та роздрібна торгівля, готелі, пункти громадського харчування. Але сучасні галузі сфери сервісів тісно зв'язані з технологічними і наукомісткими робочими місцями, на яких працюють добре освічені, висококваліфіковані, високооплачувані фахівці і які характеризуються розширеним використанням інформаційних та комунікаційних технологій (ІКТ).

Зазвичай бізнес-процес визначається як ряд дій або кроків у напрямку досягнення певної мети або як спосіб організації роботи та ресурсів (люди, устаткування, інформації і т.д.) для своїх бізнес-цілей [11]. Бізнес-процес представлено у [12] як "набір заходів, які разом виробляють результат, цінний для клієнта". За ще одним визначенням [11] бізнес-процес – це "сукупність взаємопов'язаних завдань роботи, розпочатої у відповідь на подію, досягнення конкретного результату для клієнта та інших зацікавлених сторін процесу." Подія представляє собою конкретний запит на результат, вироблений процесом. Замовник процесу є одержувачем або бенефіціаром продукції, виробленої бізнес-процесом. Потік інформації та управління бізнес-процесом зорганізується як

потік завдань (workflow). Прикладами бізнес-процесів можуть бути розробка програмного продукту, наймання працівника, усунення наслідків зливи.

В роботі [13] бізнес-процес визначається як "набір завдань з можливістю формування структури потоку завдань, який відображає, як організація досягає своєї мети". Мета кожного бізнес-процесу полягає у запропонуванні кожному споживачеві необхідного продукту або сервісу з високим ступенем якості обслуговування. Коаліція менеджменту робочих потоків WfMC (Workflow Management Coalition), що створена в серпні 1993 року для розробки та просування стандартів робочих потоків, визначає бізнес-процес як "набір з однієї або більше пов'язаних процедур або діяльностей, які в сукупності реалізують бізнес-завдання або цілі політики, як правило, в контексті організаційної структури, що визначає функціональні ролі і відносини".

Узагальнюючи, ми можемо визначити бізнес-процес як спосіб організації роботи і ресурсів, який дозволяє досягти мети за допомогою набору заходів, які проводитимуться у певному порядку, визначеному потоком завдань. Останнє фактично визначає головні властивості процесу: мету процесу (що саме?), діяльність (як?), ресурси (яким чином?) і порядок діяльності (коли?).

В [14] процеси розділяються на три групи:

- Основні процеси. Процеси, які пов'язані з зовнішніми запитами від споживачів. Основні процеси безпосередньо додають цінність у спосіб, який сприймається замовником бізнесу.

- Допоміжні процеси. Процеси, які концентруються на задоволення внутрішніх клієнтів. Допоміжні процеси можуть додавати цінність для клієнтів не напряму, підтримуючи основні бізнес-процеси, або ж вони можуть додати цінність безпосередньо.

- Процеси управління (менеджменту). Процеси, які управляють основними та допоміжними процесами, або управляють плануванням на бізнес рівні.

Завдання основних процесів – підвищення задоволеності клієнтів; завдання допоміжних процесів – підвищення ефективності роботи організації; завдання процесів управління – покращення організаційної структури.

Потік завдань (**workflow**) визначається як "процес автоматизації бізнес-процесу у цілому або частково, протягом якого документи, інформація або завдання передаються від одного учасника до іншого для дій, відповідно до набору процедурних правил" [13]. Моделі робочого потоку використовуються для проектування інформаційних систем, необхідних для підтримки бізнесу. Вони визначають ***ролі виконавців*** процесу, які беруть участь у процесі (тобто, ресурси); ***обов'язки та індивідуальні завдання***, за які відповідає кожен ресурс, і ***маршрути потоків управління***, які з'єднують завдання разом, і, отже, визначають шляхи для кожного індивідуального завдання, що бере участь в цьому процесі.

Зростання робочих місць в обслуговуванні йде паралельно зростанню інформаційної економіки, і багато робочих місць є наукомісткими, заснованими на широкому використанні інформаційних технологій як в традиційних зонах обслуговування (у тому числі комунальних послуг, технічного обслуговування будівель, фінансів, охорони здоров'я, освіти), так і нових: електронної комерції, дистанційному навчанню, колективному проектуванню і електронному уряду тощо, що робить сектор сервісів більш привабливим і придає йому більш технічний характер.

1.3. IT-орієнтовані перспективи розвитку систем сервісів

Слід особливо підкреслити, що розвиток науки про сервіси спирається на два таких відомих технічних нововведення в інформаційних технологіях, як *Softwareas a Service* (SaaS), коли програмне забезпечення використовується і орендується через Інтернет, і *Service-Oriented Architecture* (сервіс-орієнтована архітектура, SOA) як архітектурний стиль для створення IT-архітектури підприємства, що використовує принципи орієнтації на сервіси для досягнення тісного зв'язку між бізнесом і інформаційними системами, що його підтримують.

Сервіс-орієнтована архітектура еволюційним шляхом (на противагу "революційному") впроваджує в промисловість нові можливості, нові шляхи для співпраці, нові інфраструктури та нові типи програмних додатків.

Зростання складності сучасних інформаційних систем, включаючи грід/хмарні-інфраструктури, зумовило поширення модульного підходу до розробки їх програмного забезпечення з використанням стандартизованих за можливості інтерфейсів між частинами, як це передбачено концепцією сервіс-орієнтованої архітектури. Теоретичні переваги SOA полягають у підвищенні гнучкості при одночасному зниженні операційних витрат на створення системи сервісів. Дана архітектура представлена у вигляді набору сервісів і процесів, які можна комбінувати, а також змінювати з часом відповідно до змін вимог з допомогою планувальників потоку завдань (workflows). Однак практичне застосування всього потенціалу SOA ускладняється через часту необхідність використовувати окремі сервіси з несумісними інтерфейсами.

Зараз настав час переходу на *нову парадигму програмування*, зв'язану ні з об'єктами, а з бізнес-процесами і їх складовою частиною – бізнес-функціями. Її ідея – це компоновка додатків шляхом виявлення і виклику сервісів, доступних в мережі, для виконання певної задачі. Цей підхід не залежить від конкретних мов програмування і операційних систем. Можна визначити SOA ПЗ наступним чином: це архітектура додатків, побудована на основі формалізованих бізнес-процесів, функції яких представлені у вигляді багаторазово використовуваних сервісів з прозоро описаними інтерфейсами.

В цілому, SOA являє собою *модель взаємодії компонент*, яка пов'язує різні функціональні модулі додатків (сервіси) між собою за допомогою чітко визначених інтерфейсів. Інтерфейси самі по собі не залежать від апаратних платформ, операційних систем або мов програмування, використовуваних для розробки цих додатків. Це дозволяє окремим сервісів взаємодіяти між собою одним і тим же стандартним, але в той же час універсальним способом. Така особливість використання інтерфейсу, незалежна від оточення і платформи, отримала назву моделі "*слабкий зв'язок*". На слабо пов’язаних сервісах будуються

додатки для бізнес-процесів, що об’єднують різні сервіси. Необхідність виклику декількох сервісів для задовольняння потреби вимагає певних координаційних заходів, які необхідно вирішувати. Основні підходи до координації веб-сервісів відомі як *оркестровка* та *хореографія*. При пошуку одного сервісу оркестровка пов’язана з реалізацією синхронного шаблону «запит-відповідь», а хореографія – з реалізацією асинхронної взаємодії між сервісом та споживачем сервісу за шаблоном «публікація-підписка».

Зростання значення сфери сервісів у промислово розвинених країнах робить дуже важливим ***вивчення взаємодії між сервісами.*** Необхідно оцінити існуючі системи, щоб проектувати майбутні мережі розподілених сервісів, і зрозуміти вплив цих мереж в економіці. Відсутність досліджень в області сервісних мереж і брак доступних описів сервісів гальмує вирішення цієї задачі. Тому конче перспективною виглядає спроба розроблення відкритої семантичної мережі обслуговування (OSSN), яка може потенційно забезпечити отримання цінних знань з світової економіки сервісів і підтримати здійснення аналізу мережі сервісів, менеджменту і контролю [8].

Необхідні дослідження з виявлення можливого ***вибору інваріантних сервісів*** для систем, що фокусуються на людської діяльності (електричні мережі; системи водопостачання; транспортні системи; система охорони здоров'я; система освіти; банківсько-фінансові системи; мережі роздрібної торгівля; системи туризму, медіа та розваг й ін.). Це дозволило б створити репозитарій міждисциплінарних інваріантних сервісів як будівельних блоків відповідних систем сервісів. Обсяг досліджень настільки великий, що потребує колективних зусиль багатьох партнерів та співучасників.

Сервісне середовище створюється як хмарний сервіс. Воно буде в подальшому використовуватися для управління сервісами і системами сервісів з врахуванням кращих можливостей для інновацій бізнес-сервісів, що надаються хмарними технологіями за допомогою віртуалізації і поліпшення інформаційного обслуговування наукових кіл, промисловості, а також інших зацікавлених сторін.

Наука про сервіси покликана дослідити основні принципи функціонування складних систем сервісів, шляхи створення, масштабування і вдосконалення таких систем. Окрім того, останнім часом відчувається необхідність інтеграції та взаємодії додатків в рамках сукупності великої кількості інформаційних систем підприємства або декількох підприємств.

1.4. Загальна схема формування прикладних додатків на замовлення (on-demand)

На базі SOC можна побудувати мережу світового масштабу слабко зв'язаних сервісів, які можуть бути без особливих зусиль скомпоновані користувачами за своїми сценаріями у гнучкі прикладні програми з динамічною структурою, що виконуються у розподіленій обчислювальній інфраструктурі.

Сучасні дослідження базується на використанні SOC для відображення у вигляді сервісів окремих проектних процедур (функціональних модулів), а не всього прикладного ПЗ в цілому, що забезпечує можливості модифікації і адаптації такого ПЗ, підтримки репозитарію таких сервісів різними розробниками на різних мовах програмування.

У широкому сенсі SOA – це підхід до розробки додатків, при якому додаток розщеплюється на окремі частини. Ці частини, як правило, розподілені за всією системою і взаємодіють одна з одною через мережу або через API. При цьому до розробки програмного забезпечення, його доставки і розгортання використовується підхід **DevOps** [16], коли команда розробників відповідає за всі аспекти програмного забезпечення, яке вона будує, навіть за постановку свої продукції на ринок та за прибуток і збитки. DevOps (акронім від англ. Development і operations) – це набір практик, націлених на активну взаємодію та інтеграцію фахівців з розробки і фахівців з інформаційно-технологічного обслуговування. Базується на ідеї про тісну взаємозалежність розробки та експлуатації програмного забезпечення, і націлений на те, щоб допомагати організаціям швидше створювати і оновлювати програмні продукти і сервіси.

Протягом недовгої історії розвитку науки про сервіси, їх менеджмент і інжиніринг запропоновано загальну схему формування прикладних додатків «on-demand», яка включає трьох учасників: провайдера сервісів, брокера і споживача сервісів (рис. 1.1).

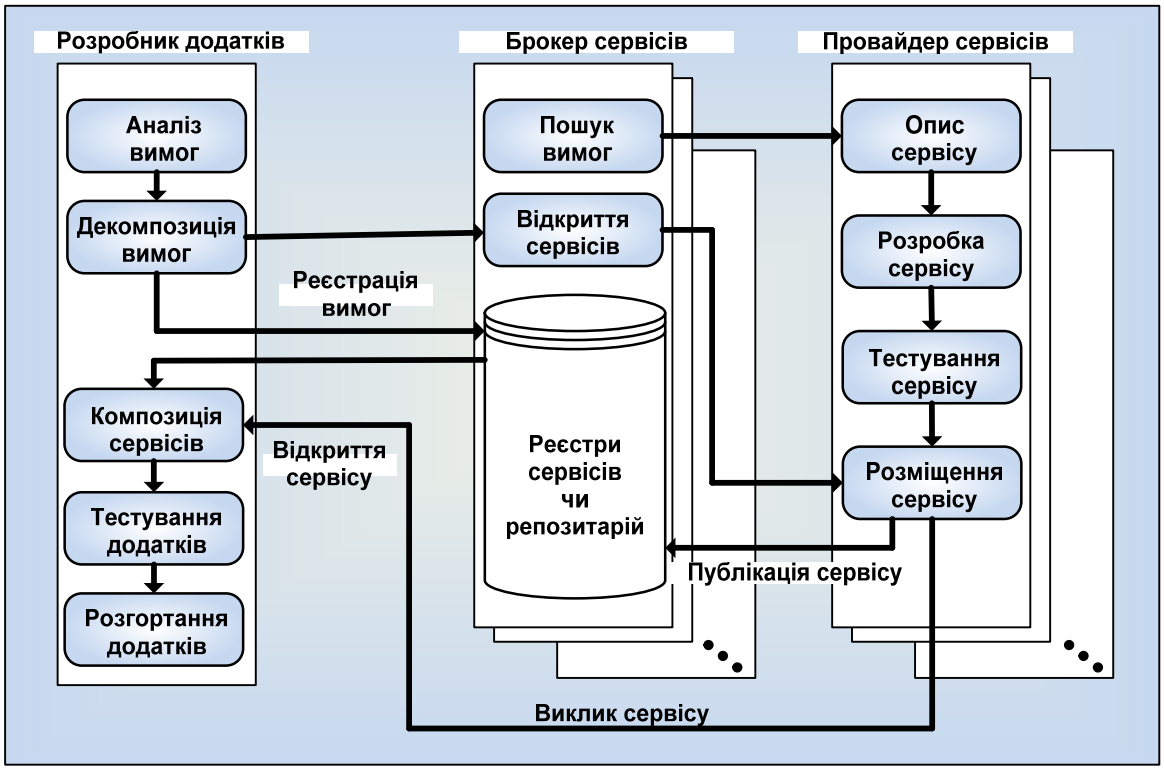


Рис. 1.1 Учасники використання SOC

Щодо сервісів, розміщених в реєстрах, то з ними, зазвичай, виконуються наступні процедури [17-20]**:**

- Знайти різні сервіси, які підходять для даної задачі (відкриття).
- Вибрати найбільш підходящі сервіси серед доступних (вибір).
- Об'єднати сервіси для досягнення мети (композиція).
- Усунити невідповідності (дані, протокол, процес) серед комбінованих сервісів (посередництво).
- Викликати сервіси згідно програмних конвенцій (виконання).
- Контролювати процес виконання (моніторинг).
- Сприяти заміні сервісів на еквівалентні (заміщення).

- Забезпечити підтримку транзакцій і скасувати або пом'якшити небажані наслідки (*компенсації*).

Базова процедура відкриття сервісу ілюструється окремо на рис. 1.2, де показано порівняння запиту користувача і опису сервісу в реєстрі, що рекламується провайдером [21]

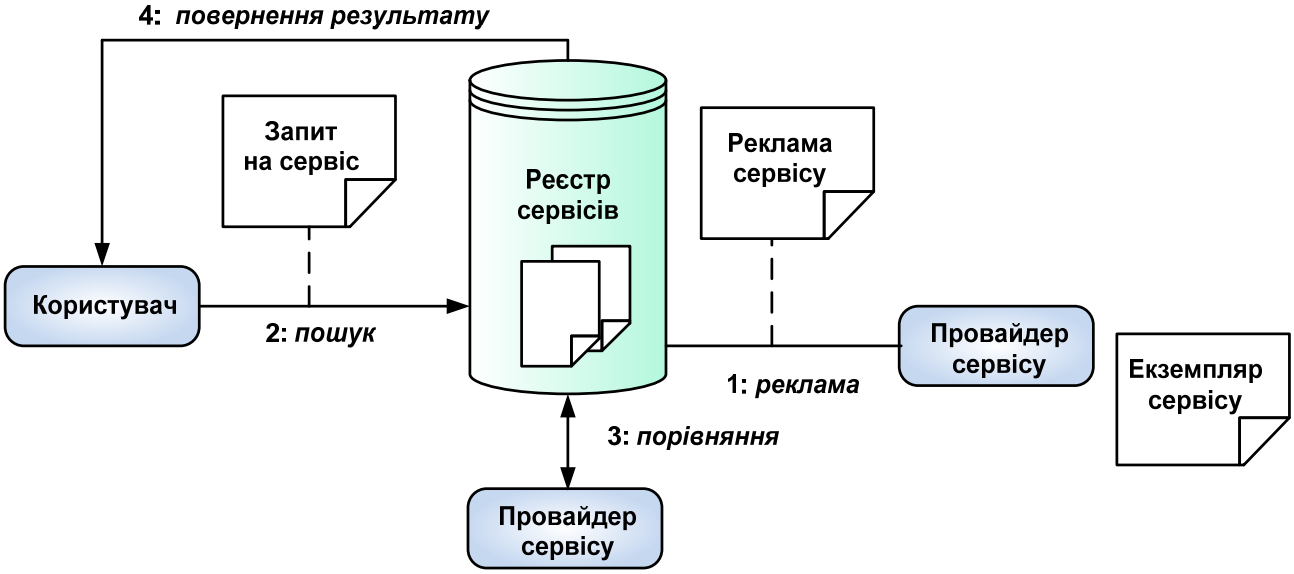


Рис. 1.2. Схема відкриття сервісу

Проблемно-орієнтовані сервіси використовуються, як правило, для вирішення завдань і розробки додатків певного класу. Запити клієнтів на пошук необхідних сервісів, що направляються у реєстр сервісів, містять параметризовані описи завдань у вигляді кінцевого набору вхідних параметрів. Сервіс після обробки запитів клієнтів повертає клієнту результат, оформлений у вигляді кінцевого набору вихідних параметрів.

Інформація про вхідні та вихідні параметри сервісу, яка необхідна клієнту для взаємодії з сервісом, міститься в опублікованому описі сервісу або надається сервісом за запитом клієнта. Сервіс може використовуватися кількома різними додатками. З іншого боку, в одному додатку можуть бути використані кілька сервісів.

1.5. Вплив степені грануляції сервісів на методику реалізації сервісних додатків

Грануляція (зернистість) сервісів – це рівень деталізації обслуговування. Зазвичай у SOA використовують модулі бізнес-логіки досить високого рівня, завдяки чому взаємодія між ними зводиться до обмеженого числа повідомлень і знижується навантаження на мережу. На рис. 1.3 показано, як в якості сервісу можна вибрати всю програму (сервіс 1), підпрогаму (сервіс 2) чи окрему процедуру (сервіс 3). Але останнім часом застосовуються *мікросервіси*, які реалізують тільки поодинокі функції. Мікросервіси повинні представляти бізнес-функції, а не загальні функції програмного забезпечення, наприклад, пошук у базі даних. Тобто мікросервіси належать до атомарних (простих) сервісів.

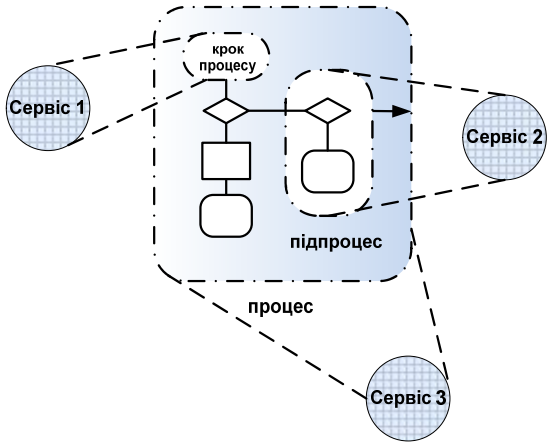


Рис. 1.3. Розділ монолітної програми на сервіси

Властивості, що характеризують сервіс, включають:

- можливість багаторазового застосування;
- можливість визначення сервісу одним або декількома технологічно незалежними інтерфейсами;
- можливість виклику сервісу, слабо зв'язаного з іншими сервісами, що забезпечують можливість взаємодії сервісів між собою.

Сервіси не запам'ятовують інформацію або не зберігають стан від одного виклику до іншого. Сервіси мають бути незалежними, автономними, що не вимагають обміну інформацією чи станом від одного запиту до іншого або залежать від контексту або стану інших сервісів. Коли такі залежності необхідні, вони зазвичай зазначаються у специфікаціях бізнес-процесів, функцій і моделей

даних. Звичайно, замовники додатків зацікавлені у підтримці стану між викликами сервісів, але це не означає, що це має робити постачальник сервісів.

У цілому SOA являє собою модель взаємодії компонентів, яка пов'язує різні функціональні модулі додатків (сервіси) між собою за допомогою чітко визначених інтерфейсів. Інтерфейси самі по собі не залежать від використовуваних апаратних платформ, операційних систем або мов програмування, використовуваних для розробки цих сервісів. Це дозволяє окремим сервісів взаємодіяти між собою одним і тим же стандартним, але в той же час універсальним способом. Така особливість використання інтерфейсу, незалежного від оточення і платформи, отримала назву моделі "слабкого зв'язку". Її очевидною перевагою є підвищена гнучкість і адаптованість, оскільки заміна або модернізація одного з компонентів системи не позначається на інших.

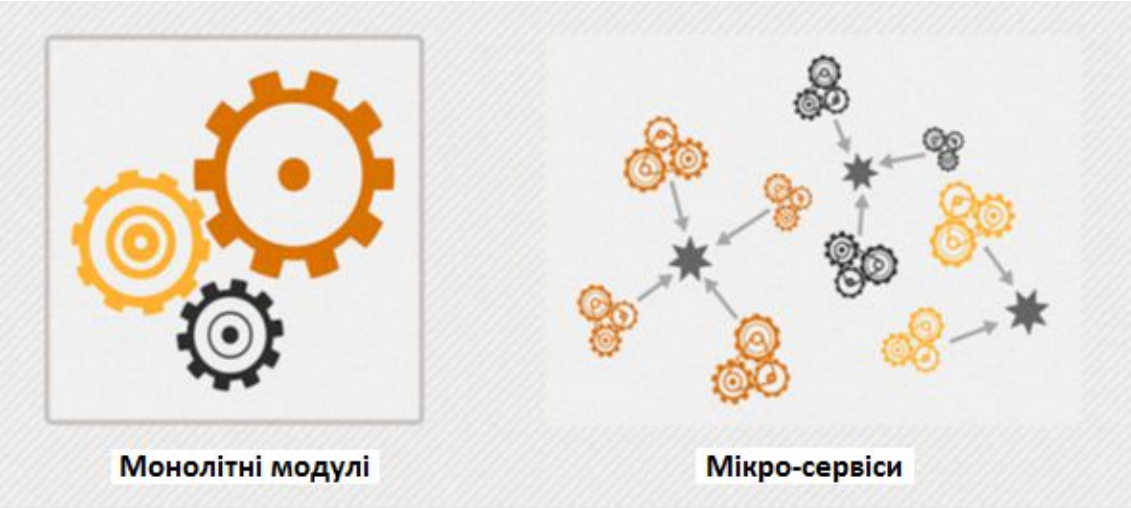


Рис. 1.4. Порівняння монолітних і мікросервісів

1.6. Перше покоління SOA, яке базується на композиції веб-сервісів

Функціональність SOA простіше всього реалізується за допомогою веб-сервісів (служб). Під веб-сервісами розуміються програмні системи, які використовують формат даних *XML (Extensible Markup Language)*, стандарти *Web Services Description Language (WSDL)* для опису своїх інтерфейсів, *Simple Object Access Protocol (SOAP)* для опису формату прийнятих повідомлень і повідомлень, що посилаються, та стандарт *Universal Description Discovery and Integration*

(UDDI) для створення каталогів доступних сервісів [21,22]. І хоча принципи сервіс-орієнтованої архітектури створення інформаційних систем не обов'язково вимагають використання технологій веб-сервісів, зв'язок між цими двома напрямками у розвитку інформаційних технологій є досить тісним. При цьому веб-сервіси є технологічними специфікаціями, у той час як сервіс-орієнтована архітектура (SOA) є принципом проектування архітектури програмних систем. Поєднання WSDL, UDDI, SOAP полегшує застосування сервіс-орієнтованих концепцій в Інтернеті, зокрема, для публікування, знаходження, пов'язування і виконання сервісів.

Базовою процедурою, як відмічалося вище, є відкриття сервісу, що відповідає процесу знаходження відповідного сервісу за його описом у наборі сервісів у центральному або розподіленому репозитарії сервісів (наприклад, UDDI). Однак механізм пошуку UDDI підтримує лише прямі порівняння описів. У тих випадках, коли немає прямих посилань і необхідно скласти набір сервісів для виконання запиту, UDDI не забезпечує ніяких результатів пошуку, тому що він не може вийти за рамки прямих порівнянь описів сервісів на базі порівняння ключових слів. Складність відкриття веб-сервісу у цьому випадку підсилюється застосуванням розповсюдженого опису сервісу мовою WSDL, яка є існуючим базовим стандартом опису веб-сервісів.

Мові WSDL не вистачає семантики, яка необхідна для того, щоб комп’ютери зрозуміли, яким є призначення сервісу. Пошук сервісів заснований на описі сервісних компонентів, наприклад, інтерфейсів або операцій. Але успішність застосування та використання веб-технологій у пошуку сервісів залежить від здатності постачальників сервісів описувати сервіси, здатності користувачів формувати запити і від *порівнювача* описів і вимог (matchmaker), який реалізує алгоритми, що враховують запит і знаходять найбільш підходящі сервіси з набору сервісів репозитарія. Порівнювач фокусується на виборі необхідних елементів з опису сервісу, показниках подібності (similarity metrics) і комбінації отриманих значень подібності [22].

Тому робилися і продовжуються робитися спроби удосконалення існуючої технології WSDL і UDDI, надавши веб-сервісам комплексні семантичні анотації. Семантика до веб-сервісів додається відображенням описів сервісу, його операцій, входу і виходу з урахуванням концептів онтології предметної галузі [23]. Мовою семантичного анотування для WSDL і XML-схеми є *SAWSDL (Semantic Annotations for WSDL and XML Schema)*.

Онтологія – це формальний явний опис понять у даній предметній галузі. У центрі більшості онтологій знаходяться *класи.* Класи описують поняття предметної галузі. Наприклад, клас Action може представляти усі процедури (запуску завдань, передачі даних, контролю потоку даних та інше). Конкретні процедури – це *екземпляри* цього класу. Процедура передачі даних чи запуску завдання – це екземпляри класу Action. Клас може мати підкласи, які представляють більш конкретніші поняття, ніж надклас. Наприклад, ми можемо розділити клас всіх Action на Action_Group (послідовність процедур) і на Task (одне завдання). *Слоти* описують властивості класів і екземплярів: процедура Task може містити файл (containsFile), створювати ресурс (createsResource) чи залежити від певних умов (hasDependency). Слоти можуть мати різні *фацети,* які описують тип значення, дозволені значення, число значень (потужність) та інші властивості значень, які може приймати слот. Онтологія разом з набором індивідуальних екземплярів класів утворює *базу знань.* Насправді, важко визначити, де закінчується онтологія і де починається база знань.

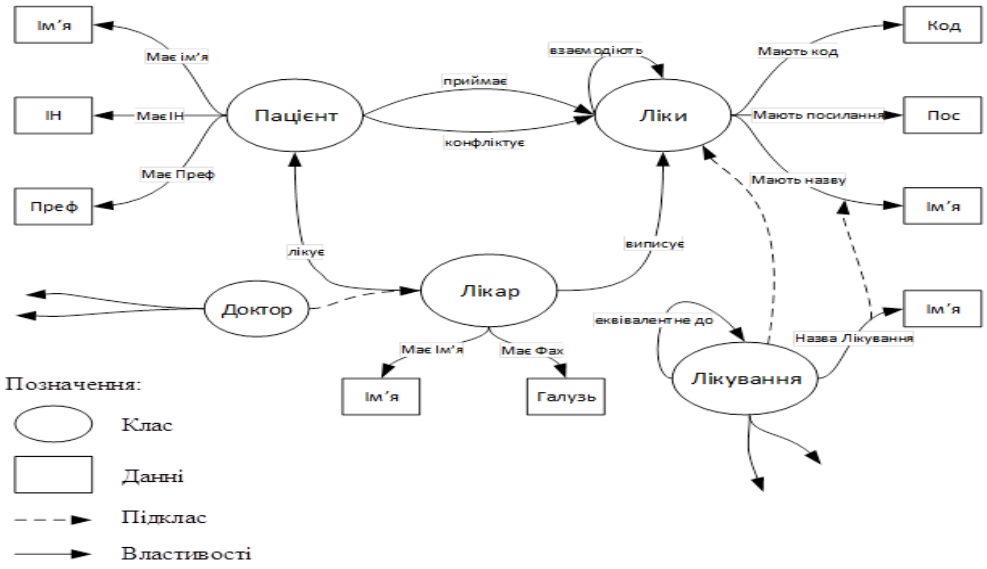


Рис 1.5. Часткова онтологія у галузі охорони здоров'я [112]

На рис. 1.5 надена часткова онтологія у галузі охорони здоров'я. Як показано на рис. 1.5, ***Пацієнт*** (Patient) має ім'я, ідентифікаційний номер (ІН) і деякі преференції (Преф) , зв'язані з віком у здоровому стані, інші. ***Ліки*** (Drug) мають назву, Код і посилання (Пос), інше. ***Лікар*** (Physician) має ім'я і фах в деякій галузі. Пацієнт приймає ліки і може конфліктувати з деякими ліками відповідно до його преференцій. Лікарі лікують пацієнтів і призначають ліки пацієнтам. Концепт ***Лікування*** (Medication) є підкласом іншого концепту Лікар (відповідно, Ліки). Відношення *НазваМед* є підвластивістю іншої властивості *ім'я ліків*.

На слабо пов'язаних сервісах будуються додатки для бізнес-процесів, що об'єднують різні сервіси. Необхідність виклику декількох сервісів для задовольняння потреби вимагає певних координаційних заходів, які необхідно вирішувати. SOA почав цю тенденцію за допомогою *технології дистанційного виклику процедури* (RPC, Remote Procedure Call) з мережі, зберігаючи при цьому контроль над компонентом доступу. Основні підходи до координації веб-сервісів відомі як *оркестровка* та *хореографія*, які згадувалися вище і реалізація яких нагадує процес програмування. Сьогоденним стандартом де-факто серед мов оркестровки є мова *WS-BPEL (Web Services Business Process Execution Language)*, раніше відома як BPEL4WS [25, 26]. Для оркестрування робочих потоків

(workflow) існує велика кількість інших конкуруючих мов: WSFL, XLANG, PDL, XPDL, ССДО, EDOC, BPML, WSCI, BPML, Staffware, FLOWer та інші.

Результатом хореографії є опис і перелік додаткових кроків, які необхідні для формування композиції (комбінації) сервісів, що мають бути виконані. Координаційний підхід є одноранговою мережею (peer-to-peer) і може визначатися за допомогою мови *WS-CDL (Web Service Choreography Description Language)*. Хореографія задається у вигляді набору взаємодій, кожна з яких складається з низки повідомлень, якими обмінюються учасники під час подій, що відбуваються у системі.

Для забезпечення комунікації та інтеграції сервісів у великомасштабних гетерогенних прикладних додатках найбільш зручним є використання сервісної шини підприємства *Enterprise Service Bus (EBS)*, яка діє в якості проміжного шару (або посередника). Існує велика кількість відкритих та комерційних ESB, які розроблялися багатьма постачальниками і налаштовані відповідно до їх потреб. До найбільш відомих реалізацій ESB можна віднести *Open ESB* [30], *ApacheServiceMix* [31], *JBoss ESB* [32], *IBM WebSphere ESB* [33], *Celtix ,IONA* і *ObjectWeb* [34]).

ESB включає у себе бізнес-процеси, сервіси та події, які споживаються або виробляються (рис. 1.6). Використання ESB в якості проміжного ПЗ (middleware) забезпечує наступні групи сервісів [27, 28]:

- *Транспортування:* транспортний сервіс гарантує доставку повідомлень та їх обмін між різними прикладними процесами. ESB може застосувувати динамічну маршрутизацію (також на основі змісту) і відправку запитів на декількох напрямках. Це дозволяє ESB балансувати навантаження або механізми відмовостійкості.
- *Оброблення подій*: сервіс подій дозволяє ESB обробляти події, можливо, аналізувати та контролювати складні серії взаємопов'язаних подій.
- *Оркестрування*: сервіс оркестрування заснований на використанні BPEL технологій, дає змогу ESB організовувати виконання ряду взаємозв'язаних

сервісів. При роботі з подіями на кожному етапі бізнес-процесу сервіси вищого рівня організують роботу сервісів нижчого рівня.

- *Пошуку*: сервіс пошуку, включений у ESB, сприяє тому, що прикладні процеси виявляють відповідні сервіси, з якими вони взаємодіють.
- *Посередництва*: сервіс посередництва є фундаментальним для галузі бізнесу інтеграції. Справді, такий сервіс відповідає за (а) перетворення одного протоколу зв'язку на інший для того, щоб зробити можливим спілкування між гетерогенними середовищами, (б) перетворення змісту будь-якого повідомлення, а також збагачення повідомлення додатковою інформацією для того, щоб будь-які дані, що передаються, були зрозумілими для будь-якого споживача. Крім перетворень сервіс посередництва є відповідальним як за безпеку, яка має вирішальне значення в міжорганізаційних взаємодіях, так і за вимоги якості QoS (наприклад, вимірювань, кешування, виявлення відмови і наступне відновлення).

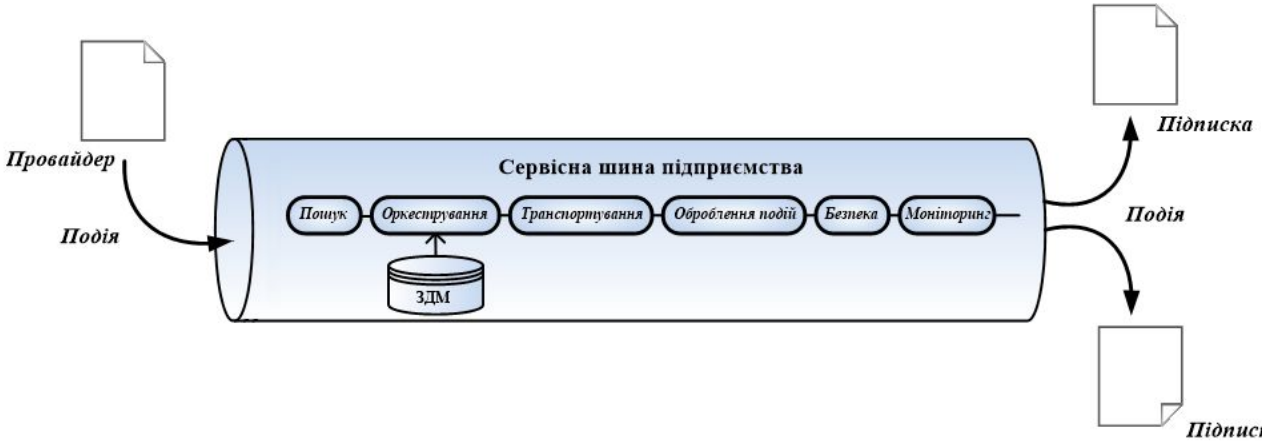


Рис. 1.6. Підтримка менеджменту подій з боку ESB

Основні обмеження ESB пов’язані з їх застосуванням тільки у строго контрольованих умовах, де адміністратори можуть правильно конфігурувати, розгортати і, нарешті, тонко налаштовувати проміжне ПЗ, щоб отримати максимальну продуктивність і максимальний рівень надійності. Як тільки взаємодії мають перетнути кордони контрольованого середовища (яке, можливо, охоплює кілька незалежних адміністративних доменів), можливості цих платформ

можуть непередбачувано деградувати і загальний рівень обслуговування не буде більше легко керованим.

На відміну від багатьох попередніх підходів до підключення інформаційних систем, ESB забезпечує функціонування програмного забезпечення, яке працює на різних платформах, написаних на різних мовах програмування, і використовує різні технології. ESB є архітектурним шаблоном, який полегшує і спрощує бізнес-інтеграцію через посередницькі і транспортні сервіси. ESB підключає і виступає посередником для усіх комунікацій та взаємодій між різнорідними вузлами, як в сервіс-орієнтованої архітектурі (синхронний підхід «один-до-одного»), так в подіємо-орієнтованій архітектурі (асинхронний підхід «багато-до-багатьох»). ESB є сьогоденним найбільш ефективним засобом вирішення складних проблем інтеграції та є технічним рішенням, яке забезпечує найбільшу гнучкість бізнесу та ефективне з'єднання між різнорідними додатками. ESB використовується для інтеграції, оркестровки, маршрутизації, моніторингу обробки подій і кореляції ділової активності. У його складі є механізм оброблення подій *CEP (Complex Event Processing)*, в якому реалізовані такі методи, як виявлення складних візерунків багатьох подій, кореляції подій і абстракції, ієрархії подій і виявлення відносин між подіями, таких як причинність, підпорядкованість, синхронізація, зв'язок з процесами, зв’язаними з подіями. Концепція CEP полягає в обробці множини подій, що трапляються на всіх рівнях організації, при цьому ідентифікуються лише найбільш важливі та аналізується їхній вплив у режимі реального часу, що дає змогу приймати певні рішення.

Слід відмітити, що підходи до реалізації ESB за останній час суттєво змінюються. ESB більше не рекомендується в якості центрального, негнучкого хребта для всього підприємства. Сьогодні "ESB" більш розглядається як розподілена інфраструктура, що масштабується, в якій можна створювати, розгортати і контролювати будь-які сервіси гнучким і ефективним засобом. За його допомогою можна створювати прикладні додатки із сервісів, які реалізують вимоги споживачів і рішення їх бізнес-завдань. Розгортання цих сервісів виконується автоматично незалежно один від одного зі стандартизованим

інтерфейсом для масштабованої платформи під час виконання. При цьому для оброблення бізнес-сервісів передбачається використовувати ESB в сервіс-орієнтованому вигляді як платформу інтеграції [28].

В рамках першого покоління SOA успішно продовжуються дослідження з покращення методів відкриття семантичних сервісів на базі ефективної *оцінки семантичної близькості* запиту і опису сервісів у реєстрі [21, 36], доцільного вибору архітектури системи сервісів, тобто шаблона (паттерна) її проектування [34], а також переносу (migration) компонентів SOA у хмарне середовище [40].

1.7. Друге покоління SOA, яке базується на поєднанні мікросервісів та контейнерів

На сьогоднішній день існує кілька способів доставки сервіс-орієнтованих додатків для кінцевих користувачів – традиційно за допомогою е-інфраструктури підприємства, мобільних пристроїв і через хмару. Як раз додатки, розміщені у хмарі, змінюють усі традиційні правила, на яких будуються SOA першого покоління. Щоб максимізувати переваги хмарних технологій, необхідно *змінити моделі сервісних додатків* і оптимізувати хмару відповідно до цих змін. Починаючи з 2017 року, мікросервіси у поєднанні з контейнерами дозволили прискорити розробку сервісних хмарних додатків і підвищити ефективність їх розгортання; забезпечити переміщення сервісів і їх перезапуск в умовах відмови; забезпечити масштабування сервісів зі зміною навантаження [38]. Мікросервіси виглядають так само, як веб-API в тому, що вони можуть бути доступні за допомогою простих HTTP протоколів запиту і форматів представлення даних JSON.

Однією з проблем, пов’язаною з традиційним розгортанням мікросервісів у хмарі, є затримка доступу до них. Кожен мікросервіс є реалізацією процедур «запит-відповідь», і якщо до мікросервісів часто звертатися в ході виконання додатку, затримки, які накопичуються, можуть серйозно вплинути на час відгуку і продуктивність роботи користувача. Друга проблема на шляху використання мікросервісів полягає у марній траті ресурсів. Мікросервіси, як правило, мають

розміри, набагато менші, ніж традиційні компоненти SOA програмних додатків. Ресурси, що необхідні для розгортання у хмарі контейнера з мікросервісом, можуть скласти біля 90% від ресурсів, що витрачаються на підтримку віртуальної машини (ВМ), але контейнери відрізняються від віртуальних машин у тому, що додатки, що працюють в контейнерах, *розділяють спільно операційну систему* і велику частину проміжного шару (рис. 1.7).

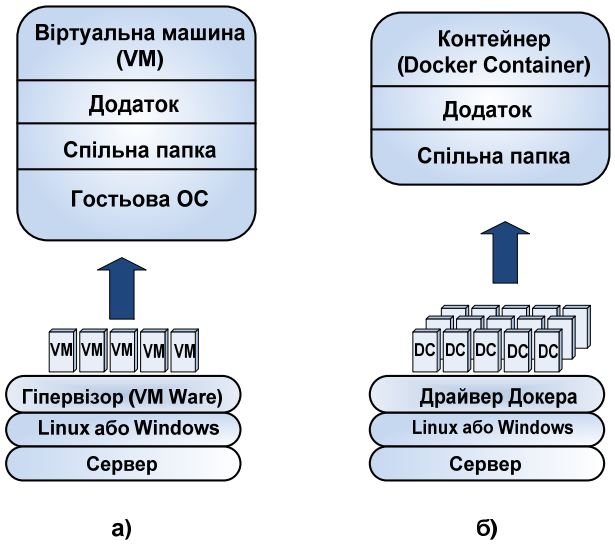


Рис. 1.7 Порівняння процесів віртуалізації (а) і контейнеризації (б)

Усунення дублювання таких великих програмних елементів, як гостьова ОС (Gest OS), дозволяє набагато більшій кількості контейнерів з мікросервісами працювати *на одному сервері* у порівнянні з кількістю віртуальних машин. Ця різниця досягає зазвичай значень від 5 до 10 разів, хоча є повідомлення про рекордні значення у 30 разів.

Мікросервіси також розгортаються швидше в контейнерах, ніж на віртуальних машинах. Це може бути дуже корисним під час горизонтального масштабування сервісів зі зміною навантаження або при перерозподілі мікросервісів у зв’язку з відмовою мережевого серверу. Якщо кожен мікросервіс розмістити в окремій віртуальній машині з незалежної ОС, то буде витрачено багато пам’яті та інших ресурсів.

Ефективність використання ресурсів і прискорення розгортання додатку не є єдиною перевагою контейнерів при переході на мікросервіси. Хмарні

контейнери (наприклад, Docker) здатні природно групувати компоненти додатків на певні кластери, розміщуючи, наприклад, спільні програмні елементи (мікросервіси, API) на тому же самому хості, що і компоненти, які їх використовують. Це призводить до зниження додаткових затримок в мережі, що можуть накопичуватися.

Контейнери, ймовірно, є найбільш швидко зростаючою і найбільш захоплюючою галуззю розвитку віртуалізації і хмарних обчислень. Відносини між контейнерами та мікросервісами не є простими. Контейнери не є необхідними для розгортання мікросервісів, але при цьому мікросервіси необхідні для обґрунтування доцільності використання контейнера. Мікросервісами є одним із способів підвищення маневреності додатків і повторного використання коду. Контейнери є *формою віртуалізації*, яка дозволяє уникнути дублювання операційної системи у машинному образі, забезпечуючи загальну платформу, яка поділиться між всіма компонентами додатку [40].

Підсумовуючи, можна стверджувати, що контейнери пропонують ***новий спосіб реалізації і для SOA,*** який є більш ефективним у багатьох відношеннях. Наприклад, з Docker можна запустити кожен веб-сервіс всередині контейнера. Потім можна використовувати контейнерний оркестратор (скажемо, ***Kubernetes***) для управління зв’язком між контейнерами та гарантувати, що кожний сервіс всередині програми працює стабільно. Контейнери, в свою чергу, можна розгорнути в кластери, при цьому контейнери можуть бути попередньо вбудованими як компоненти платформи, що дозволить розмістити їх разом, щоб створити *образи додатків*. Такий підхід відповідає концепції *DevOps* (рис. 1.8) про швидке розгортання нових можливостей програмного забезпечення безпосередньо в операційному середовищі [16].

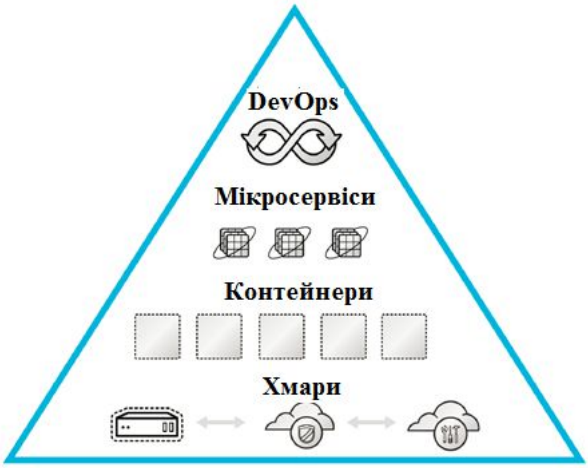


Рис. 1.8. Ієрархія нових концепцій SOA другого покоління

З 2017 року розпочався справжній бум зі створення нового покоління MSA [40], або SOA другого покоління. Саме на розглянутому вище новому сервіс-орієнтованому підході базується найбільший європейський проект зі створення ресурсів ***Європейської відкритої науково-дослідницької хмари (European Open Science Cloud for Research, EOSC***), підготовкою якою займаються зараз усі відомі європейські організації і інфраструктури (EUDAT, CERN, OpenAIRE (Open Access Infrastructure for Research in Europe), GÉANT, EGI, ESFRI (European Strategy Forum on Research Infrastructures), e-IRG (e-Infrastructure Reflection Group), ESA(European Space Agency) та інші [38,39].

EOSC являє собою федеративне, глобально доступне, міждисциплінарне середовище, у якому дослідники, новатори, організації та громадяни можуть публікувати, знаходити, використовувати і повторно використовувати дані кожного з них, інструменти, публікації та інші матеріали для наукових досліджень, інновацій і освітніх цілей. EOSC буде обслуговувати 1,7 млн. вчених і 80 млн. професіоналів з різних галузей науки і технологій.

Загальне бачення більшості відповідних зацікавлених сторін для EOSC полягає у тому, що ця хмара повинна:

- бути екосистемою сервісів, які надаються різними постачальниками сервісів;

- спиратися на існуючі електронні інфраструктури, тому зусилля розробників мають бути спрямовані на інтеграцію/сумісність своїх сервісів;
- постійно розвиватися і інтегрувати нові сервіси і нові технології, як тільки вони стають доступними, готовими до використання (категорії TRL8 +) користувачами;
- вважати потреби користувачів рушійною мотивацією розвитку EOSC (рис. 1.9.)

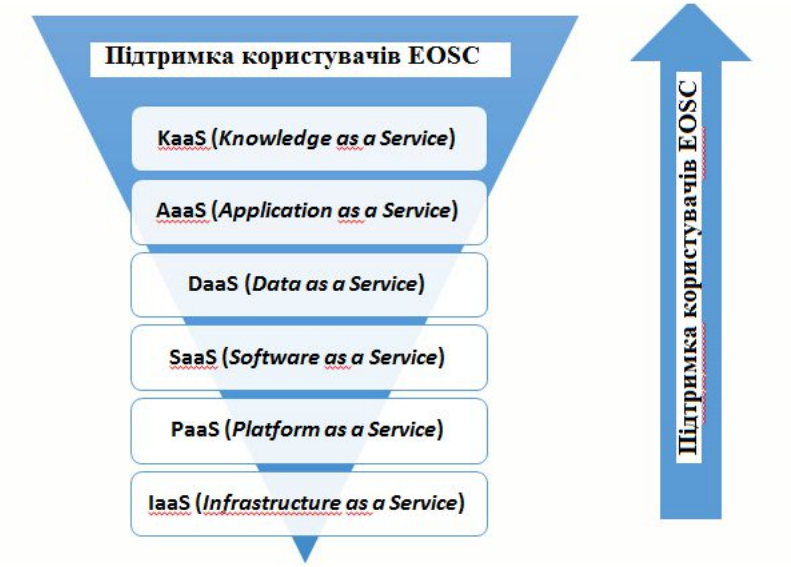


Рис. 1.9. Класифікація сервісів

Відкритий каталог сервісів планується створити до 2019-2020 років в межах нового EOSC-hub проекту [39], що стартував на початку 2018 року і який призначений для забезпечення беззбиткового і відкритого доступу до місцевих, регіональних та національних колекцій сервісів, створених в Європі та у всьому світі.

Передбачається також творення єдиної точки входу на порталі ***eInfraCentral*** для кінцевих користувачів, щоб можна було переглянути каталог сервісів, а також посилити контроль ключових показників ефективності (KPI): доступності, сумісності і фрагментації, зрозумілості і ясності сервісів каталогу.

Основними розробниками реєстрів сервісів EOSC на сьогодні є організації EGI, INDIGO, EUDAT, FI FARE, AMAZON. Інформація про їх продукти носить переважно *рекламний характер*. Здається, що вони базуються на своїх існуючих

монолітних програмних рішеннях, перетворюючи їх на декілька десятків *макросервісів* (замість мікросервісів).

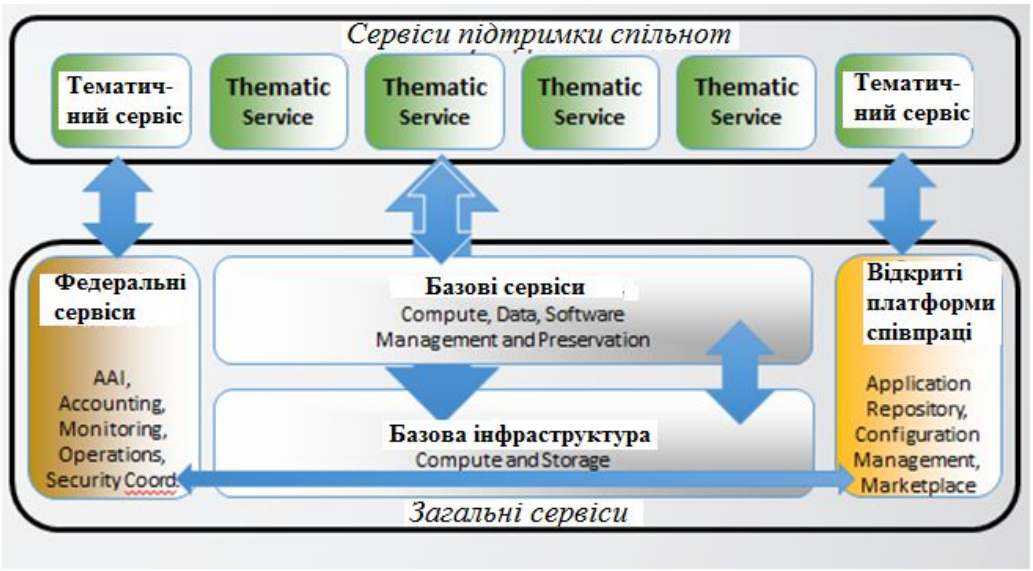


Рис. 1.10. Сервісна архітектура EOSC-hub

Сьогодні такі макросервіси розміщуються в контейнери разом з *екземпляром ОС* і своєю базою даних для виконання додатку в хмарі [38]. А що робити іншим потенційним користувачам хмари, якщо згадані макросервіси не підходять за функціями і розмірами для складання їх індивідувальних прикладних застосувань?

1.8. Виклики сьогодення в розвитку SOA

Недавня поява і стрімке розповсюдження альтернативного напрямку розвитку SOA мотивує до проведення порівняльного дослідження двох базових підходів (веб-сервісів з уніфікованими протоколами зв'язку і мікросервісів з контейнерами) із точки зору їх ***можливої конвергенції,*** оскільки кожен з них має свої переваги і недоліки. Дослідженню підлягають питання:

1) порівняльне узагальнення можливостей і властивостей макросервісів, які формуються зараз з існуючих монолітних рішень і рекомендуються для включення до 2020 року у каталог Європейської Відкритої Наукової Хмари;
2) порівняльне оцінювання можливостей застосування і сумісності (інтерабільності) існуючого інструментарію провідних фірм для створення

мікросервісних реєстрів і забезпечення тим самим реалізації у хмарних MSA віртуалізації як серверів, так і операційних систем;

3) дослідження можливості поєднання веб-сервісів з контейнерними технологіями, що відкриває шляхи до суттєвого збільшення розмірів хмарних репозитаріїв сервісів і організації в них семантичних методів пошуку (відкриття) необхідних сервісів;
4) дослідження підходів до автоматизації проектування структури SOA, а значить, і обґрунтування доцільного вибору необхідних для її реалізації сервісів, виходячи з доменних онтології предметної галузі, або бази знань;
5) дослідження можливостей з удосконалення процедури семантичного відкриття сервісів в репозитаріях за запитом користувача на базі порівняння їх семантичних описів, а також раціонального використання властивостей і відмінностей різних сервіс-орієнтованих архітектур;
6) порівняльне дослідження еквівалентності функціональності сервісної шини підприємства і API управління контейнерами з точки зору оброблення складних подій в SOA і MSA;
7) доведення спадкоємності між SOA і MSA, схожих принципів досягнення спільних цілей, доцільність використання SOA для інтеграції *додатків*, а MSA – для інтеграції компонентів в межах *окремого додатку.*
8) Відповіді (повні чи частинні) на ці запитання будуть певним внеском цієї дисертаційної роботи у розвиток науки про сервіси, їх менеджмент і інженерію.

Висновки до розділу 1

Сектор сервісів є найбільш *швидкозростаючим сегментом світової економіки,* в якому вже сьогодні зайнято більше половини працездатного населення планети. Системи сервісів є динамічними конфігураціями людей, технологій, організацій та засобів обміну інформацією, які створюють і забезпечують цінність для користувачів, постачальників та інших зацікавлених сторін. Індустрія послуг потребує *створення своєї наукової бази*, що може

містити наступні чотири основні блоки: *основи науки про сервіси, інженерію сервісів, менеджмент сервісів і технологію реалізації сервісів.*

Ця робота ставить собі за мету дослідити ряд положень нової науки, що виникли сьогодні у зв’язку із загальною тенденцією переносу IT-рішень у хмарне середовище. На базі розглянутого сьогоденного стану досліджень і розробок в галузі сервісних технологій сформовані науково-практичні задачі, які потребують ретельного розгляду і вирішення.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ БАЗОВИХ ПРИНЦИПІВ І ТЕХНОЛОГІЙ SOA ТА MSA

2.1. Мікросервіси

Мікросервіс є додатком з однією функцією, такою, наприклад, як маршрутизація мережевого трафіку, онлайн-платіж або аналіз медичного тесту. Така концепція не нова: вона базується на концепції веб-сервісів, а об'єднання мікросервісів (атомарних процесів) у функціональні додатки є еволюцію сервіс-орієнтованої архітектури (SOA) бізнес-процесів, яку вже використовують протягом кількох років. Мікросервісні додатки набагато простіше, ніж традиційні SOA з їх стандартами, включаючи складні сервісні шини підприємства (ESB) у формі проміжного програмного забезпечення, необхідні для спілкування між усіма сервісами [40].

Розпочалася епоха мікросервісів: вони розробляються, розгортаються і масштабуються незалежно, швидко адаптуються до змін, порівняно просто реалізують функції малого бізнесу. Компанії самі можуть з чистого аркуша розробляти мікросервіси, які при необхідності можна швидко і легко змінювати, тому що вони представлені меншим за розміром кодом. Це зовсім не те саме, як мати справу з традиційними монолітними додатками з кодом довжиною у мільйони рядків, що розроблялися раніше. Мікросервіси призначені для роботи у хмарних середовищах, обчислювальні ресурси яких, що необхідні для функціонування певних додатків, можуть збільшуватися і зменшуватися за бажанням. Мікросервіси розробляються так, щоб їх бути легко замінити, якщо трапляються негаразди в інфраструктурі чи якийсь конкретний сервіс припиняє роботу.

Організації повинні створити портфель мікросервісів, які вони можуть повторно використовувати у декількох додатках MSA, а також і для традиційних SOA. Так, але є й істотні відмінності:

- Немає зобов'язань щодо використання якоїсь унікальної технології (скажемо, певної мови опису мікросервісу, наприклад, *WHDL*).

- Реалізується більша гнучкість архітектури додатку.
- Мікросервіси управляються як продукти з своїми власними життєвими циклами, при цьому логіка сервісу фактично закладається в оболонку конкретного інтерфейсу API.
- Забезпечується автоматичне розгортання сервісів.

З усіма перевагами, які мікросервіси приносять, вони також створюють кілька складнощів, а саме:

- вимагають інтеграції і оркестрування, оскільки прикладні додатки містять сотні / тисячі мікросервісів;
- потребують більш досконалих засобів автоматизації розгортання і конфігурації, вибору та моніторингу із-за більшої кількості мікросерсівів, зт яких складається додаток.

Можна виділити шість основних заходів з подолання зазначених складнощів і використання переваг мікросервісів в повному обсязі:

- вибір контракту (або внутрішнього зв'язоку) з мікросервісом;
- вилучення (відокремлення) мікросервісів з існуючих монолітних додатків;
- пошук (відкриття) сервісів;
- координація взаємодії;
- управління складністю розгортання сервісів і їх масштабованістю;
- видимість сервісів.

Контракт з сервісом в SOA першого покоління визначався інтерфейсом SOAP, притаманним для веб-сервісів, з підтримкою в той же час найбільш важливих WS-* стандартів, таких як WS-Security або WS-Policy. У світі мікросервісів REST (точніше: RESTful HTTP) є стандартом де-факто. І причина цього є не стільки більш висока продуктивність, але й проста архітектура, розділ концептів, відсутність стану і єдиний інтерфейс. Це ідеальний варіант, особливо для мобільних пристроїв і Інтернету речей (IoT), двох з основних джерел і драйверів для використання мікросервісів.

З REST можна також використовувати різні формати даних, наприклад, завжди можна вибрати між JSON і XML. У той час як легкий формат JSON ідеально підходить для мобільних пристроїв, формат XML (включаючи XML-схему) є найкращим форматом для бізнес-процесів підприємств. Можна визначити схеми, виконувати перетворення і перевірку з набагато меншими зусиллями і менш досконалим оснащенням. Продуктивність була аргументом проти XML у минулому. Сьогодні це вже не є величезним недоліком у більшості випадків.

Зв'язок сьогодні часто здійснюється за допомогою протоколу HTTP. Стандарт обміну повідомлень, такий як *JMS*, є хорошим варіантом для бізнес-процесів, керованих подіями. *WebSockets, MQTT* та інші стандарти також корисні у зв'язку з наявністю мільйонами пристроїв, задіяних в Інтернеті речей. Таким чином, важливо, щоб мікросервісна архітектура MSA підтримувала різні формати даних і транспортні протоколи без повторної побудови мікросервісів.

Мікросервісний підхід до розбиття існуючих додатків (legacy) на мікросервіси відповідно до потреб бізнесу доповнює можливості створення абсолютно нових мікросервісів (рис. 2.1). Потім мікросервіси можуть бути повторно використані у різних контекстах, і це означає, що і для різних комунікаційних потреб. Відокремлення транспортної логіки від логіки мікросервісу є успішною практикою підтвердження життєво важливого значення контексту мікросервісів. При побудові логіки мікросервісу розробник не мусить думати про те, як мікросервіс взаємодіє з кінцевою точкою, якою може бути сервер підприємства (XML / SOAP), сервіс хмари (XML / HTTP), мобільний пристрій (JSON / HTTP) або пристрій IoT (протокол TCP низького рівня, MQTT, або, можливо, навіть власний протокол).

Розробник додатку також повинен бути в змозі виявити і використовувати інші мікросервіси, які можуть бути опубліковані через *сервісний шлюз (API gateway)* [42-46]. Шлюз забезпечує споживання сервісів, забезпечує їх масштабування і надійність і дозволяє повторно їх використовувати у різних контекстах без змін.

Сервісний шлюз робить мікросервіси доступними. Він використовує відкриті стандарти, такі як SAML, Kerberos, OAuth, WS- * або XACML – залежно від вимог [44]. Крім того, розробникам потрібен простий спосіб відкрити необхідних сервісів та їх контрактів. Зазвичай для надання каталогу сервісів й інформації про їх контракти використовується портал самообслуговування.

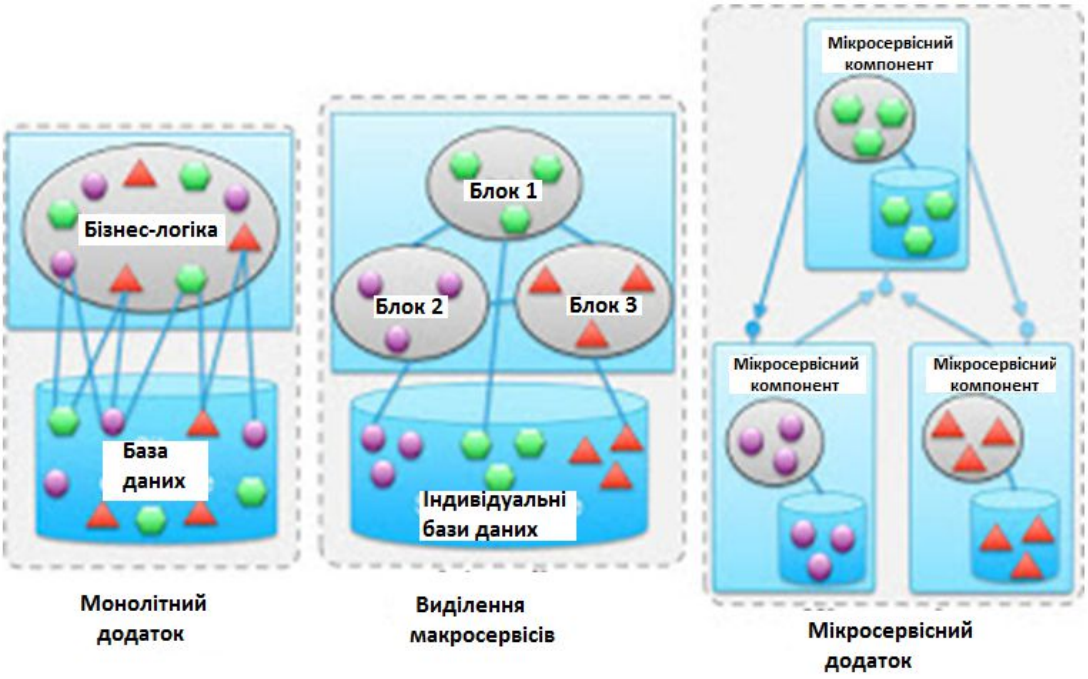


Рис. 2.1. Шлях від монолітних додатків до мікросервісів

У той час, як SOAP підтримується кількома фреймворками та інструментами протягом багатьох років, фреймворк API *Swagger*, здається, стає стандартом за замовчуванням для реалізації, виявлення і тестування REST сервісів. Він також інтегрований у багатьох проміжних ПЗ, або middleware [44].

Коли мова йде про відкриття сервісів, то згадують про ***API, Open API і API керування.*** Це просто терміни, як і попередній термін «сервісна шина» для SOA. Згадану процедуру відкриття сервісів можна було би називати «Реєстром мікросервісів», бо з її допомогою вирішується важлива бізнес- проблема.

Термін API раніше мав на увазі низькорівневі інтерфейси, реалізовані у програмному коді. В останні роки термін змінив значення, і зараз під ним маються на увазі прості інтерфейси, побудовані поверх протоколу HTTP. Зазвичай це еквівалентно REST-інтерфейсам, які надають дані в форматі JSON

(іноді – XML) і використовують команди HTTP: PUT, GET, POST і DELETE для опису дій "створити", "прочитати", "змінити" і "видалити". Дані протоколи і формати даних простіше у використанні, ніж стандарти веб-сервісів, засновані на протоколі SOAP в SOA першого покоління. Так само вони більш підходять для таких мов програмування, як JavaScript, які активно використовуються для здійснення запитів до API.

API керування стає важливим у багатьох галузях, незалежно від того, чи то комунікації бізнес для бізнесу (B2B) або бізнес для клієнта (B2C). API стали окремим продуктом, що продається, і їх виробники конкурують один з одним за увагу розробників. Широко відомі наступні засоби API керування: *TIBCO API Exchange, IBM, Apigee, 3scale, WSO2, MuleSoft, Mashery, Layer 7, Vordel* [46]. Для API потрібні складні портали, на яких розробники додатків могли б знаходити і експериментувати з даними програмними інтерфейсами. Провайдерам потрібна можливість встановлювати плани оплати для розміщення різних показників використання API. Оскільки програмні інтерфейси виставляються публічно, то шлюзи, через які надається доступ, повинні забезпечувати найвищий рівень безпеки.

Координація взаємодії мікросервісів може здійснюватися різними засобами: зі збереженням стану (stateful) або без збереження (stateless), з орієнтацією на повідомлення сервісів або на події, що виникають. Координація взаємодії без збереження стану є найкращою практикою для одного сервісу у більшості випадків, конкретна координація *композитного сервісу* може працювати краще з урахуванням стану процесу. Графічний інструмент може бути використаний як для створення поодинокого мікросервісу, так для створення композитних сервісів.

Контекст використання сервісів буде залежати багато від чого. Без безперервної інтеграції / безперервної доставки не можна ефективно реалізувати концепцію мікросервісів. Тому здійснюється постійне розгортання, налаштування і управління додатками і проміжним ПЗ на підприємстві або у хмарі за

допомогою інструментальних панелей, а також контроль якості розгорнутого додатку, управління портами та балансуванням навантаження.

Мікросервіси можуть розгортатися всюди, у тому числі в приватних центрах обробки даних, на віртуальних машинах і в хмарних середовищах, таких як *Amazon Web Services, VMWare* або *OpenStack*. Важливо розуміти, що кожен мікросервіс будується і розгортується незалежно один від одного. Уніфіковане управління є ще одним ключовим фактором успіху для мікросервіної архітектури. Навіть якщо мікросервіси розробляються за допомогою різних технологій (наприклад, мов Java, Scala, Python, або графічного інструментарію), вони можуть керуватися і контролюватися за допомогою єдиного інтерфейсу, призначеного для користувача.

Кореляція подій є методом осмислення великої кількості подій і ідентифікації кількох подій, які дійсно важливі в цьому обсязі інформації. Тому важливо вміти здійснювати складну обробку подій і потокову аналітику в режимі реального часу.

2.2. Контейнери

Контейнер є програмним забезпеченням для автоматизації розгортання й управління сервісами у *середовищі віртуалізації на рівні операційної системи*. Він дозволяє «упакувати» сервіс з усім його оточенням і залежностями в контейнер, який може бути перенесений на будь-яку Linux-систему, а також надає середовище з управління контейнерами.

Контейнеризація, по суті, реалізується на рівні віртуалізації операційної системи (на відміну від віртуальних машин (VM), кожна з яких оснащена повною вбудованою ОС). Контейнери легко упаковані, легкі і призначені для роботи в будь-якому місці. Кілька контейнерів можуть бути розміщені в одній віртуальній машині.

Контейнери та мікросервіси не те ж саме. Мікросервіс може працювати у контейнері, але він також може працювати і на виділеній VM. Проте, контейнери є хорошим способом розроблення і розгортання мікросервісів, згрупованих на

певні композиції, а інструменти і платформи для запуску контейнерів є хорошим способом для управління мікросервісними додатками [46-48].

Сьогодні є багато постачальників контейнерів, але ніхто не сперечається, що продукція фірми Docker лідирує на ринку, задовольняючи на 40% існуючий попит. Google давно використовує контейнери. Компанія є одним з основних вкладників у різні контейнеро-орієнтовані проекти з відкритим вихідним кодом, включали оркестратор *Kubernetes* і *Google Container Engine*, що функціонують в Google Cloud Platform [48].

Microsoft підтримує контейнеризацію за допомогою *Windows Server Containers,* що дозволяє спільне використання ядра ОС між хостом і контейнерами. *Hyper-V Контейнери* розширюють цю послугу, запускаючи кожен контейнер в оптимізованій віртуальній машині. Для хмари є також *Azure Container Сервіс* (ACS), розроблений спільно з Докер, який може керувати підтримкою інших інструментів оркестровки, таких як Kubernetes.

Клієнти AWS можуть здійснити розгортання контейнери на своїй EC2 платформі, забезпечуючи управління кластерами і планування двигуном Docker за допомогою *EC2 Container Service (*ECS). Ця можливість підтримується *EC2 Container Registry (*ECR) шляхом зберігання,

Компанія VMware в минулому році випустила *vSphere Integrated Containers* для перетворення віртуальних машин на Докер-подібні контейнери, з яких створюються платформи відкритого управління системою сервісів для реалізації бізнес-процесів у середовищі Photon OS. Інші приклади включають IBM Контейнери для Bluemix, *Rackspace Carina* (на основі OpenStack Magnum, вбудована підтримка для контейнерів та оркестровка). Ще є одна ініціатива з відкритим вихідним кодом *Deis*, платформа-як-сервіс (PaaS) на основі CoreOS. Слід визнати, що ринок контейнеризації стрімко змінюється [50].

Перелік програмної подукції для контернерів узазальнений у Додатку 1, серед якої найбільш відомими є *Докер-двигун* (драйвер) для побудови та експлуатації Докер контейнерів разом з засобами *Docker Swarm* (групування та планування хостингу) і *Docker Compose* (багатоконтейнерні додатки); операційна

контейнерна система *CoreOS* та альтернативний формат *rkt*, що пропонує зовсім іншу архітектурну філософію контейнерів; система керування кластером *Mecoc* для ефективного розподілу обчислювальних ресурсів між платформами для доставки додатків; система оркестрування *Mesosphere* на базі *Mecoc* як альтернатива *Kubernetes* та *Amazon's Elastic Container Service*. Близько року тому була започаткована *Open Container Initiative*, спрямована на заохочення відкритих стандартів для форматів контейнерів, що розробляються, та часу роботи.

2.3. Контейнери та SOA

Контейнери пропонують новий спосіб реалізації SOA, який є більш ефективним у багатьох відношеннях. За допомогою контейнера можна запустити кожен сервіс (частину програми) всередині контейнера. Контейнери є кращими будівельними блоками для SOA ніж традиційні сервіси з наступних причин [40,46]:

- *Контейнери легко переміщаються між хостами* (хмарами, вузлами). При використанні традиційних SOA сервіси залежать від конкретних умов хостів. Наприклад, якщо мережевий файл файлової системи встановлюється на сервері CentOS Linux для надання сервісу зберігання даних для SOA, зміна хоста NFS на сервер Ubuntu потребує значних зусиль. На відміну від цього, переміщення контейнера від одного хоста до іншого значно простіше, оскільки контейнерне середовище хоста завжди однаково.
    - *Контейнери мають високу масштабованість*. Оскільки контейнери додатків можуть легко переміщатися між вузлами, контейнерна інфраструктура також легко масштабується шляхом додавання екземплярів певного сервісу, повязаних спільною базою даних. Масштабування в традиційних розподілених системах не реалізується так просто.
    - *Контейнери легко оновити*. Хочете оновити додаток, що працює в контейнері? Просто треба відновити *зображення контейнер*а, на якому заснований додаток, а потім перезапустити контейнер. Цей процес є

більш простим і безвідмовним у порівнянні з традиційним оновленням сервісу в розподіленої системі. Так же просто відмінити зміни, якщо це необхідно.

- *Контейнери можуть бути використані повторно*. Однією з основних цілей SOA є повторне використання сервісів, їх розділення і зменшування кількості надлишкових сервісів. Контейнери просто використовувати повторно, оскільки зображення контейнерів може бути легко скопійовані і контейнери можуть швидко переміщатися між вузлами.

До складу програмних засобів входить *сервер контейнерів*, *клієнтські засоби*, що дозволяють з інтерфейсу командного рядка управляти *зображеннями контейнерів*, а *також API*, що дозволяє в стилі *REST управляти контейнерами програмно*.

Сервер забезпечує повну ізоляцію контейнерів, що запускаються на хості, при цьому кожен контейнер має доступ тільки до прив'язаному до нього мережевого простору імен і відповідним віртуальним мережевим інтерфейсам. В дійсності використовуються кілька хостів для того, щоб створити легко доступну систему контейнерів, коли запускаються багато екземплярів контейнерних зображень, розповсюджуваних між кількома хостами, з'єднаними через мережу.

Набір клієнтських засобів дозволяє запускати процеси в нових контейнерах, зупиняти і запускати контейнери, припиняти і відновлювати процеси в контейнерах. Набір команд дозволяє здійснювати моніторинг запущених процесів. Нові зображення контейнера можливо створювати зі спеціального сценарного файлу, можливо записати усі зміни, зроблені в контейнері у нове зображення. Крім того, в інтерфейсі командного рядка вбудовані можливості взаємодії з публічним репозиторієм, в якому розміщені попередньо зібрані зображення контейнерів [50].

Кілька контейнерів розгортаються в кластерах, багато з них будуть попередньо вбудовані як компоненти, які можуть бути розміщені поруч, щоб створити *зображення* додатків. Є чотири положення, які необхідно враховувати:

- ***Операційні системи контейнерів.*** Навіть якщо контейнери не мають вбудованих ОС, одна спільна ОС, як і раніше, необхідна. Будь-яка стандартна ОС буде працювати, в тому числі Linux або Windows. Проте, фактично необхідні ресурси ОС, як правило, обмежені, тому звичайної операційної системи може бути занадто. Це призвело до розробки спеціальних контейнерних операційних систем, таких як *Rancher OS, CoreOS, VMware Photon, Ubuntu Snappy, Red Hat Project Atomic і Microsoft Nano сервер* [51, 52].
- ***Драйвер (двигун) контейнеру.*** Тут, як відмічалося, домінує Docker, але є конкуренти, такі як CoreOS Rocket (Rkt). Двигуни поставляються з допоміжними засобами, наприклад, *Docker Toolbox*, який спрощує настройку Docker для розробників, і *Docker Trusted Registry* для зображень управління.
- ***Оркестровка контейнерів.*** Контейнери повинні бути розумно згруповані для забезпечення функціонування додатків, і це вимагає оркестровки. Двигуни забезпечують основну підтримку для визначення простих мульти-контейнерних застосувань, наприклад, *Docker Compose*. Проте, повна оркестровка включає в себе планування того, як і коли контейнери мають працювати, управління кластером і надання додаткових ресурсів, часто інших хостів. Інструменти оркестровки включають *Docker Swarm*, Google *Kubernetes* і *Apache Mesos* [52, 54]. Можна використовувати інструменти конфігурації загального призначення, такі як *Chef* and *Puppet* (обидва з відкритим вихідним кодом) або комерційні пропозиції, такі як *HashiCorp Atlas* або *Electric Cloud ElectricFlow*.
- ***Переміщення додатку з однієї хмарної платформи на іншу.*** Постачальникам програмного забезпечення буде потрібно послідовно оновляти свої додатки для розгортання їх у користувачів.

Необхідні контейнери будуть копіюватися і додатки будуть побудовані з цих контейнерів, так що повне робоче середовище буде відтворене, в тому числі і самі контейнери, баланси навантаження, мережі і так далі. У 2015 році компанія

Docker випустила *Docker Networking* для включення віртуальних з'єднань між контейнерами. Англійська фірма Weaveworks також створила *WeaveNet*, а *Micro-Software-Calico* збільшує безпеку контейнерної мережі через динамічну побудову firewalls (брандмауерів). Docker теж розробляє нові інструменти для підтримки життєвого циклу контейнерних додатків. *Docker Universal Control Plane* (*UCP)* забезпечує можливості створення, розгортання та управління додатками на вимогу [55]. Продукт ще повинний піти в виробництво. Weaveworks також має інструмент під назвою *WeaveScope* для моніторингу виробництва контейнерних додатків.

Для того, щоб автоматизувати керування мікросервісами й ініціювати хмарну інфраструктуру і для їх підтримки, треба значно змінити обчислювальний процес. Перш за все, бажано виділити у додатку, що проектується, 50-100 незалежних сервісів, створених за допомогою процедури композиції з окремих мікросервісів. Потім мікросервісне програмне забезпечення, або шар програмного забезпечення, що керує їм, треба гармонізувати з доступними ресурсами ЦП, виміряними у циклах центрального процесора (CPU), з вживаною мережею і засобами зберігання інформації. Це потребує використання програмно-визначеної IT-інфраструктури, яка лежить в основі управління хмарними ресурсами. Наприклад, компанії, починаючи від IBM до OpenStack, пропонують IT-середовища, в яких ресурси обчислення, мережі і зберігання даних доступні і управляються за допомогою ПЗ інтерфейсів прикладного програмування (API), а не з системи управління або командного рядка. Можна, звичайно, інтегрувати існуючу конкретну інфраструктуру з іншою, що вже має плагіни для створення і розгортання додатків і вимагає певної форми програмного контейнера. Мікросервіси можуть бути розгорнуті на сервері з іншими мережними драйверами, що притаманні іншій ОС (наприклад, Linux) або іншій мові програмування (наприклад, версії Python).

2.4. Open API і сервісна шина

В розділі 1 розглядалася сервісна шина підприємства (ESB) як платформа доставки сервісів в традиційних SOA. Чи втратила свою чинність концепція ESB сьогодні для MSA? Відповідь, звичайно, ні! [55]. Однак, ESB більше не рекомендуються в якості центрального, негнучкого хребта для всього підприємства. На заміну сьогодні приходить гнучка, розподілена інфраструктура, що масштабується, де можна створювати, розгортати і контролювати будь-які мікросервіси в гнучкий і ефективний спосіб. При цьому розробка і впровадження можуть бути здійснені як на підприємстві, так і в хмарі, або суміші обох (наприклад, з використанням хмари тільки для тестових середовищ або для обробки споживчих піків).

Сьогодні під ESB розуміються засоби, побудовані для інтеграції, оркестровки, маршрутизації, моніторингу, обробки і кореляції подій, ділової активності. Крім того, можна створювати додатки за допомогою мікросервісів, які реалізують необхідні бізнес-завдання. Ці мікросервіси розгортаються автоматично незалежно один від одного зі стандартним інтерфейсом для масштабованої платформи під час їх виконання. Можна назвати таку псевдо-ESB як завгодно: *платформа інтеграції, мікросервісна платформа*, тощо.

У доповнення до цього інструменту використовується сервісний шлюз (service gateway) для забезпечення безпеки, дотримання політики і надання через API мікросервісів зовнішнім споживачам. Сервісний шлюз керує сервісами інтеграції (побудованими за допомогою псевдо-ESB), прикладними сервісами (побудованими за допомогою псевдо-ESB або будь-якої іншої технології), а також зовнішніми хмарними сервісами (не важливо, як вони побудовані, важливо, що є домовленість про постачання сервісів).

Щоб корелювати події, які відбуваються в різних мікросервісах, необхідно зберігати ці події в пам'яті, зробити їх видимими в режимі моніторингу в реальному часі, доступними для аналітики та активних прогнозованих дій.

2.5. Приклади реєстрів сервісів для EOSC

European Open Science Cloud (EOSC) – це міждисциплінарне середовище для наукових досліджень, інновацій і освітніх цілей, рис. 2.2 [38, 39, 56].

Основними розробниками реєстрів сервісів для EOSC витупають організації EGI, INDIGO, EUDAT, FI FARE, AMAZON. Розглянемо їх вклади більш детально.

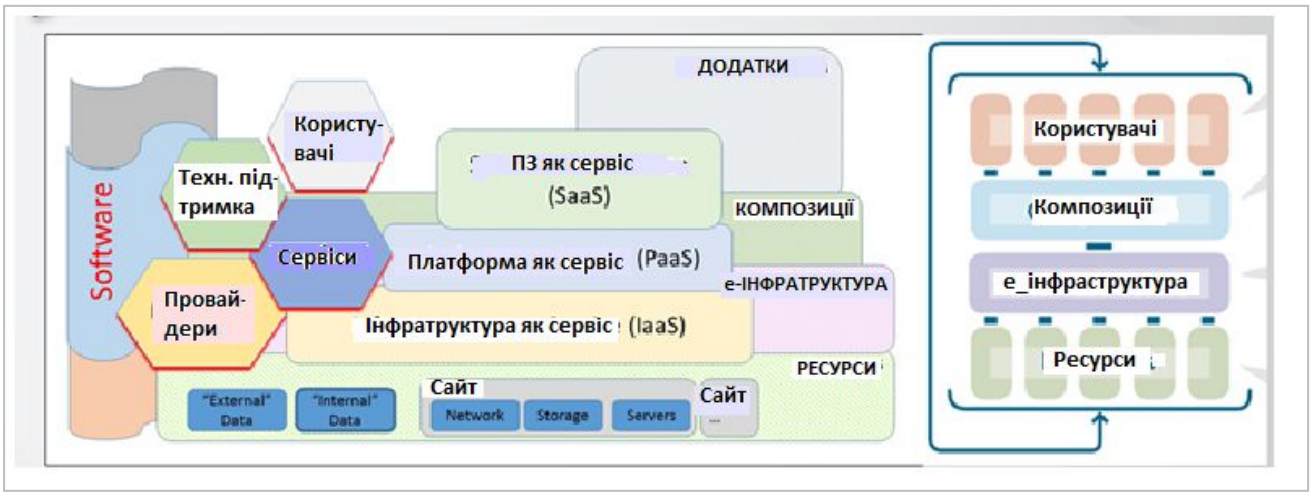


Рис. 2.2. Європейська Відкрита Наукова Хмара

2.5.1. EGI

EGI (European Grid Infrastructure) є федерацією із 300 обчислювальних центрів і центрів даних, розташованих в 50 країнах, яка займається створенням і доставкою через національні кордони відкритих рішень для науки і досліджень шляхом федерації цифрових можливостей, ресурсів і досвіду. Сервіси, які вона надає [57], зведені у табл. 2.1.

Таблиця 2.1

EGI сервіси

№	Назва сервісу	Сервіс забезпечує або підтримує
1	2	3
Обчислювальні сервіси		
1	**Cloud Compute**	• виконання обчислень і оброблення даних; • підтримку тривалих сервісів (наприклад, веб-серверів або баз даних);

		• створювання середовищ розробки і виконання тестування; • здійснення вибору конфігурації віртуальної машини відповідно до вимог користувача; • управління ресурсами хмари на гнучкій основі з комплексним моніторингом і можливостями обліку.
2	**Cloud Container Compute**	• підготовку додатка на вимогу; • середовище для максимального продуктивності; • стандартний інтерфейс для розгортання сервісів від кількох постачальників; • розгортання і виконання Dacker контейнеів в віртуальному середовищі; • сумісність (Interoperable) і прозорість; • усунення тертя між середовищами розробки та експлуатації.
3	**High-Throughput Compute**	• доступ до високоякісних обчислювальних ресурсів; • інтегровані засоби моніторингу та обліку для надання інформації про доступність і споживанні ресурсів; • інструменти для робочого навантаження і управління даними для всіх обчислювальних задач; • велику кількість переробних завдань протягом тривалого періоду часу.
Сервіси зберігання і навчання		
4	**Online Storage**	• призначення глобальних ідентифікаторів файлам; • доступ до високо-масштабуємих сховищ у будь-якій точці світу; • контроль за даними, що використовуються зумисно; • організацію даних, використовуючи гнучку ієрархічну структуру.
5	**Data Transfer**	• ідеально підходить для дуже великих файлів; • здатний обробляти великі обсяги файлів; • підтримує процес передачі з автоматичною повторною спробою.

6	**Arhive Storage**	• збереження даних на тривалий термін; • збереження великої кількості даних; • звільнення свого інтернет-сховища.
7	**FitSM Training**	• збільшення досвіду в галузі управління IT-сервісами; • підвищення професійного профілю визнаної сертифікації.
8	**Training Infrastructure**	• цільові конкретні курси та додана вартість для наукових співтовариств; • легке розгортання курсів і їх повторне використання

2.5.2. INDIGO

INDIGO (INtegrating Distributed data Infrastructures for Global ExplOitation) об'єднує 26 європейських партнерів з 11 країн Європи, у тому числі розробників розподіленого програмного забезпечення, промислові підприємства, дослідницькі інститути, університети, *e*-інфраструктури. Координація діяльності здійснюється італійським Національним інститутом ядерної фізики (INFN). Головні напрямки діяльності:

- розроблення відкритих, сумісних рішень для оброблення наукових даних;
- підтримка відкритої науки з організації європейського простору даних;
- забезпечення співпраці через різні наукові співтовариства в усьому світі.

Сервіси, які надає INDIGO, представлені у табл. 2.2 [58].

Таблиця 2.2

INDIGO сервіси

№	Назва сервісу	Сервіс забезпечує або підтримує
1	2	3
1	**IAM** (Identity and Access Management)	• реєстрацію сервісу; • менеджмент спільних рішень VO (віртуальних організацій); • постачання сертифікованих OIDC/сервер авторизації OAuth; • надання APIs, базованих на стандарті SCIM; • забезпечення ідентифікації користувачів і надання інформації про політики сервісам, щоб можна було

		приймати послідовні рішення щодо авторизації в умовах розподілених сервісів.
2	**Dynpart** (Partition Director Service for Batch and Cloud resources)	• управління гібридним центром обробки даних, який може надавати сервіси як для пакетної обробки даних, так і хмарні розподілені сервіси.
3	**Cloud Provider Ranker**	• автономний WEB REST сервіс, який виконує ранжування постачальників хмар за пріоритетами користувачів з вибору ресурсів; • використання драйвера системи управління бізнес-правилами (BRMS, Business Rules Management System).
4	**OneData** (Global Data Access)	• глобальну систему управління даними, що забезпечує легкий доступ до розподілених ресурсів зберігання даних і підтримує широкий спектр прецедентів, починаючи з управління даними для наукових обчислень з інтенсивним обміном даними.
5	**Orchestrator** (PaaS Orchestrator)	• основний компонент прошарку PaaS; • збирання запитів на розгортання з прошарку (рівня) програмного забезпечення; • координацію ресурсів або сервісів розгортання динамічних Mesos кластерів або безпосередньо на базі IaaS платформ.

6	**Udocker** (Userspace Container Support)	• інструмент для виконання простих контейнерів Docker в просторі користувача; • можливість завантаження і виконання кінцевим користувачем; • bdocker і udocker: два взаємодоповнюючі підходи для виконання контейнерів у пакетних системах оброблення даних.
7	**Ophidia** (Data Mining and Analytics for eScience Server)	• фреймворк (структуру, яка підтримує аналіз даних, інтенсивно використовуючи паралельні методи обчислень і смарт-методи розподілу даних); • ієрархічну організацію зберігання для поділу і розподілу багатовимірних масивів наукових даних на декілька вузлів.

8	**IM** (Infrastructure Manager)	• розгортання складних й індивідуальних віртуальних ресурсів на різних платформах IaaS хмари (таких як AWS, OpenStack); • надання оркестранту обчислювальних ресурсів можливості використання стандартних мов OASIS (наприклад, протоколу TOSCA).
9	**Kubernetes Monitoring Accounting**	• забезпечення основних функціональних можливостей та інструментів, що підтримують роботу усіх сервісів PaaS, доступних в інфраструктурі.
10	**fgAPIServer, fgAPIServerDaemon, fgPortalSetup, fgTools** (Future Gateways (Programmable Scientific Portal)	• програмований інтерфейс з RESTful API сервера; • набір програмних компонентів, здатних створити або допомогти існуючим веб-порталам стати сервісними Шлюзами; • доступ до розподілених обчислювальних ресурсів, таким як грід, хмари і високопродуктивні обчислення (HPC).
11	**indigoclient, indigoKepler** (INDIGO Plug-ins for scientific workflow systems)	• систему управління робочими потоками (Workflow) • забезпечення інфраструктури установкою, запуском і моніторингом певної послідовності завдань, що створює робочий потік, використовуючи ресурси та сервіси, надані постачальниками і іншими спільнотами.
12	**NOW** (Network Orchestration Wrappercomponent for OpenNebula-based clouds).	• приймання запитів для маніпуляції віртуальної мережі; • здійснення глобальної попередньої авторизації і виконання запитів через API OpenNebula.
13	**Orchent** (the orchestrator client)	• командний рядок додатку для керування його розгортанням і вибору ресурсів (оркестратор) у швидкій і простий спосіб.

Нещодавно INDIGO випустив продукт **Electric Indigo** для полегшення збірки прикладних застосувань за бажанням шляхом включення в нього інтерфейсів рівня хмарних провайдерів і можливості автоматичної композиції сервісів. Продукт містить 170 пакетів програмного забезпечення (сервісів), 50 готових до виконання контейнерів Docker. Він призначений для роботи з

операційними системи: CentOS 7, Ubuntu 16,04 в умовах хмар OpenStack Ньютон та OpenNebula 5.x і завантажується з сайту DataCloud Software Repository [58].

2.5.3. European Data Infrastructure

Місія ***European Data Infrastructure*** **(*EUDAT*)** полягає в проектуванні, розробці, реалізації загальних сервісів передачі даних, підтримці кількох наукових співтовариств, а також осіб через географічно розподілену відмовностійку мережу 35 європейських організацій [59]. Ці загальні сервіси і ресурси зберігання даних розподіляються по 15 європейським країнам, і дані зберігаються разом з деякими з найпотужніших суперкомп'ютерів Європи.

За допомогою EUDAT сервісів можна:

- Скористатися перевагами простих, ефективних, надійних, доступних і сумісних за даними інфраструктур.
- Налагодити зв'язок з самим потужними суперкомп'ютерами Європи.
- Отримати підтримку експертів для вирішення проблеми з даними.
- Прослідити походження даних трасуванням в каталозі метаданих.

Згідно зі звітом «Riding the Wave report» [60] сервіси EUDAT пропонуються через інфраструктуру *Collaborative Data (CDI*), яка в даний час визначена на рівні місцевих громад, науково-дослідницьких організацій та прикордонному рівні (дисциплін і країн). Загальні сервіси передачі даних мають відношення до кількох спільнот і доступні на європейському рівні, при цьому характеризуватися:

- високим ступенем відкритості;
- відкритим доступом за умовчанням;
- незалежністю від конкретних технологій, оскільки вони змінюватися часто;
- гнучкістю, щоб дозволити новим громадам інтегруватися, не зважаючи на різнорідність і фрагментацію ландшафту даних.

Сервіси, які надає EUDAT, наведені в таблиці 2.3 **(**https://eudat.eu/services-support):

Таблиця 2.3

EUDAT сервіси

№	Назва сервісу	Сервіс забезпечує або підтримує
1	2	3
1	**B2DROP** (Sync and Exchange Research Data)	• рішення для персональної хмари, засноване в довіреному EUDAT CDI домені, для зберігання та спільного використання наборів даних на початку дослідження.
2	**B2SHARE** (Store and Share Research Data)	• зручний, надійний сервіс, що заслуговує довіри дослідницьких спільнот, для зберігання та обміну дрібномасштабними обсягами даних, що надходять з різних контекстів.
3	**B2FIND** (Find Research Data)	• простий, зручний портал для пошуку колекцій наукових даних, які зберігаються в EUDAT центрах обробки даних т інших сховища даних.
4	**B2SAFE** (Replicate Research Data Safely)	• надійний, безпечний і дуже доступний для управління даними і їх реплікації сервіс, що дозволяє спільноті і реєстрам дублювати і зберегати наукові дані у вузлах EUDAT.
5	**B2STAGE** Get Data to Computation)	• надійний, ефективний, простий у використанні сервіс для переміщення великих обсягів наукових даних між вузлами EUDAT і робочим простором обчислювальних систем високої продуктивності.
6	**B2ACCESS** (Indentity and Authoresation)	• підтримку кількох методів аутентифікації ідентичності користувачів (OpenID, SAML, x.509); • постачання посвідчень і підтримка ідентичності користувачів по всьому світу; • забезпечення унікальних ідентифікаторів EUDAT.

7	**B2HANDLE** (Register your research data)	• забезпечення доступу до даних і їх публікації; • використання сервісу для створення і управління PIDs для об'єктів даних; використання механізму B2HANDLE для вилучення і посилання на об'єкти.

2.5.4. Інші сервіси

Звичайно, що є і інші реєстри сервісів (*Amazon Web Services*, *Google Apps*, *Microsoft Azure*, *GÉANT* або *European National Research* and Education Networks (NRENs) [62], які містять чимало цікавих і корисних сервісів.

Наприклад, Amazon Web Services включає сервіси [63]:

- ресурси віртуальних і приватних хмарних серверів;
- сервіси для контейнерів;
- сервіси AWS Lambda і прямого з'єднання;
- зберігання та кеш; обмін сервісами, робочі області інших додатків;
- керовані сервіси бази даних;
- сервіси машинного навчання та Інтернету речей (IoT);
- адміністрування та безпеку (управління доступом ідентичності);
- розгортання і управління;
- сервіси додатків (гнучкі платежі і т.д.);
- підтримки, довіреного радника, Marketplace (інтернет-магазину).

Сервіс *AWS Step Functions* містить графічну консоль, за допомогою якої можна створювати додаток у вигляді композиції мікросервісів, що виконують окремі функції, крок за кроком, запускати кожен крок, відстежувати його виконання і при наявності помилок перезапускати його. Щоб змінити крок або додати новий, не потрібно навіть писати код.

AWS Step Functions є частиною платформи безсерверних обчислень *AWS Lambda* [68], яка реалізує сервіс «функція як сервіс» ("function as a service (Faas)"), який відрізняється тим, що під час складання і експлуатації додатку

користувачеві не потрібно турбуватися про сервери, віртуальні машини (VM) або базові обчислювальні ресурси, необхідні для підтримки величезного обсягу подій, задіяних в його додатку. Оплачувати йому прийдеться лише за час використання цього сервісу Faas, а не за час оренди хмарного обладнання, яке дуже часто простоює. Безсерверні обчислення в дійсності не усувають сервери, просто хмарний провайдер бере на себе обслуговування фактично задіяних обладнання та інфраструктури.

Зараз відмінності сервісів у різних реєстрах, наведених вище, суттєві і зводиться до наступного:

- різні стандарти і фрейворки, які доступні для управління сервісами;
- різноманітна лексика про управління сервісами, яка використовується спільнотами;
- різні процеси управління портфелем сервісів у різних місцях е-інфраструктури на різних рівнях реалізації практики управління сервісами;
- немає стандартизованого способу, щоб визначити і описати сервісну е-інфраструктуру з різними описами сервісів або переліками сервісів в реєстрах;
- різні пропозиції сервісів від різних постачальників, що потенційно частково перекриваються;
- різні сервіси, що пропоновані на різних рівнях: централізовано в е-інфраструктурі, на національному або регіональному рівні (наприклад, за допомогою NREN) або навіть на місцевому рівні (університет).

Тому з початку 2018 в Європі розпочатий проект *EOSC-hub*, згаданий в розділі 1, з об’єднання існуючих реєстрів у *загальний каталог (перелік) сервісів*. Що таке загальний каталог сервісів? Це структурована інформація про всі сервіси електронної інфраструктури від незалежних постачальників, що відображає точку зору користувача: функціональні можливості сервісів, вартість, терміни та умови поставки, процес замовлення, оплати моделі, контактні точки.

Це також спільна веб-платформа, яка буде основною точкою входу в інформацію про європейські сервіси електронної інфраструктури. Передбачено, що пошук і вибір необхідних сервісів з такого каталогу користувач буде виконувати сам у ручному режимі.

2.6. Інструментарій для створення веб-сервісів

Перехід до загального репозитарію з безліччю альтернативних варіантів мікросервісів від різних провайдерів, у якому відкриття необхідних мікросервісів буде виконуватися *автоматично за запитом користувача*, потребує, перш за все, узгодження описів мікросервісів як веб-сервісів з семантичною інформацією.

При оновленні контенту такого репозитарію можна скористатися чи реєстром сервісів *FIWARE* , побудованих для Інтернету речей (IoT) [66], чи напрацюваннями провідних компаній з розроблення інструментарію для створення сервіс-орієнтованих додатків.

2.6.1. Каталог *FIWARE*

Каталог *FIWARE* містить багату бібліотеку сервісів (Generic Enablers, GE) з посиланнями, які дозволяють розробникам впроваджувати такі їх функціональні можливості, як підключення до Інтернету речей або аналіз Big Data, що значно полегшує програмування (табл.2.4). Всі вони публічні, безкоштовні та відкриті. Поєднуючи їх, можна розпочати розробку свого додатку відразу.

Каталог FIWARE включає посилання на інші каталоги, в яких міститься інформація про спеціальні ресурси для різних доменів (DSE), що може бути корисними для тих, хто планує розробляти програми у галузях енергетики, творчих ЗМІ, розумного виробництва, охорони здоров'я або для сектора сільського господарства.

Таблиця 2.4

FIWARE сервіси

№	Назва сервісу	Сервіс забезпечує або підтримує
1	2	3

1	**Domibus** Electronic Data Exchange	Сервіс реалізує стандартний протокол обміну повідомленнями (на основі профілю AS4), який забезпечує сумісні, надійні та безпечні дані.
2	**AEON** Cloud Messaging	Сервіс у режимі реального часу надає хмарні сервіси (канали) для передачі необмеженої кількості сутностей, обмінюючись необмеженим обсягом інформації, а також сервіси для управління суб'єктами, що беруть участь у хмарному середовищі.
3	**Orion Context Broker** Publish/Subscribe Context Broker	Це реалізація контекстного брокера *Публікація / Підписка*, що забезпечує інтерфейси NGSI9 та NGSI10 для виконання декількох операцій: реєстрації додатків для виробників контексту, оновлення контекстної інформації, отримання сповіщення, коли відбуваються зміни в контекстній інформації.
4	**Cosmos** BigData Analysis	Сервіс призначений для розгортання засобів аналізу як пакетних, так і поточних даних. Пакетна частина пакету передбачає використання драйвера *Hadoop* як сервісу HAAS.
5	**Complex Event Processing (CEP)** - Proactive Technology Online	CEP аналізує дані про події у режимі реального часу, реагуючи на ситуації, а не на окремі події. Ситуації включають композитні події (наприклад, послідовні), розподіл операторів за подіями (наприклад, агрегування) та оператори відсутності.
6	**IoT Discovery**	Сервіс забезпечує взаємодію виробників контексту IoT з споживачами, які шукають необхідні сервіси, використовуючи протокол обміну повідомленнями OMA NGSI-9.
7	**IoT Broker**	Компонент проміжного програмного забезпечення, який дозволяє програмам отримувати сукупну інформацію про базові пристрої і сервіси.
8	**IDAS** Backend Device Management	Сервіс служить для підключення IoT пристроїв / шлюзів до екосистем на основі FIWARE, переводячи протоколи, специфічні для IoT, у протоколи контекстної інформації NGSI, які є стандартною моделлю обміну даними в FIWARE.

9	**Interface Designer**	Сервіс забезпечує простий у використанні повноцінний маніпулятор / редактор 3D-об'єктів на сцені.
10	**Cloud Rendering**	Сервіс визначає загальний спосіб запитування, отримування та керування відеопотоком віддаленої 3D-програми.
11	**Knowage** Data Visualization -	Пакет сучасної бізнес-аналітики для традиційних джерел та систем великих даних.
12	**Store - WStore**	Еталонна реалізація універсального доступу до сховища FIWARE. Зокрема, WStore реалізує API та відкриті специфікації, пов'язані з FIWARE Business Framework.
13	**Biz** Business API Ecosystem	Спільний компонент інтеграції FIWARE Business Framework з набором стандартних API-інтерфейсів, наданих TMForum.
14	**Wirecloud** Application Mashup	Нова платформа mashup, орієнтована на кінцевих користувачів, яка дозволяє створювати веб-додатки та інформаційні панелі / кабіни.
15	**WMarket** Marketplace	Набір базових сервісів для реєстрації суб'єктів господарювання, публікації та отримання пропозицій та вимог, пошуку та виявлення пропозицій відповідно до конкретних вимог споживачів, а також бічні функції, такі як перевірка, оцінка та рекомендації.
16	**Repository** - Repository RI	Послідовний уніфікований API для опису сервісів на мові USDL та пов'язаних мультимедійних файлів для додатків бізнес-структури, який використовується для опублікування опису на єдиній мові різних аспектів сервісів.
17	**Docker**	Екосистема докерів-контейнерів і управління відповідними ресурсами в Центрі обробки даних для створення композиційних сервісів, що складаються з сервісів FIWARE, та для розгортання їх у постачальниках хмар, сумісних з FIWARE Cloud.
18	**Sagitta** Software Deployment & Configuration	Засіб автоматичного розгортання (встановлення та налаштування) з використанням API або через Cloud Portal програмного забезпечення для запуску віртуальних машин.
19	**Pegasus**	Сервіс надає необхідні віртуальні ресурси на рівні IaaS, а

	PaaS Manager	також встановлює та розгортає ці ресурси на одному сервері, на декількох серверах або на еластичній архітектурі з використанням балансування навантаження.
20	**IaaS GE** - FIWARE Reference Implementation	Сервіс, реалізований на базі OpenStack, забезпечує проміжне програмне забезпечення хмарної інфраструктури для віртуальних машин, а також пов'язаних з ними обчислень і зберіганням даних.

2.6.2. Windows Communication Foundation

Однією з технологій, яка дозволяє будувати сервіс-орієнтовані додатки, є ***Windows Communication Foundation (WCF)*,** відома через *.**NET Framework*** [67]. WCF реалізує сучасні галузеві стандарти WS* для сумісності з веб-сервісами. WCF дозволяє використовувати SOAP, JSON, XML і т.і.

ASP.NET – це єдина модель розробки веб-додатків корпоративного класу з мінімальними зусиллями, яка включає сервіси, необхідні для розробника. ASP.NET є частиною *.NET Framework*, тому при написанні додатків розробник має доступ до класів в .NET Framework. Він може розроблювати свої додатки на будь-якій мові програмуванні, сумісній з CLR (Common Language Runtime), у тому числі Microsoft Visual Basic і C#.

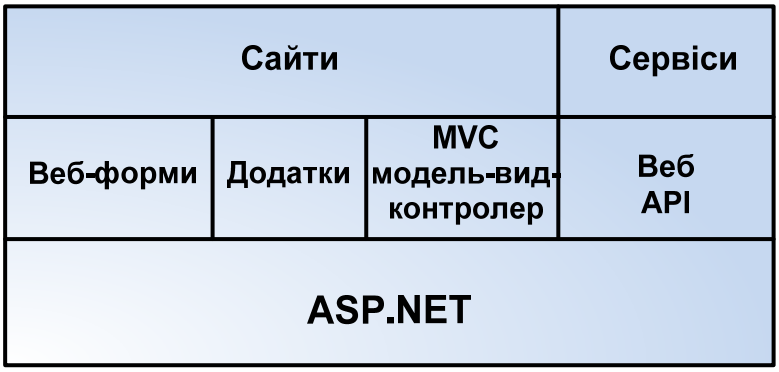


Рис. 2.3. Структура ASP.NET

ASP.NET складається з трьох основних компонентів: Веб-форми (ASP.NET Forms), інтерфейсу користувача (ASP.NET MVC) і програмного інтерфейсу (ASP.NET Web API) (рис. 2.3). Кожен з даних компонентів створено для різних цілей: Forms – для створення простих додатків з мінімальними затратами, проте вони погано розширюються, MVC – для створення великих додатків, коли код

бізнес-логіки відокремлений від коду і розмітки, так що розробники можуть працювати над бізнес-логікою, а дизайнери окремо фокусуються на розмітці і Java Script коді. Найпопулярнішим для розробки сервіс-орієнтованих додатків є ASP.NET Web API, який використовує REST, тобто HTTP і запити GET, POST, PUT, DELETE і т.і., для зміни даних в БД та обчислень. Сервер з розгорнутим ASP.NET Web API додатком може *виконувати роль мікросервісу,* який взаємодіє з іншими такими ж мікросервісами, використовуючи JSON, XML і інші. Однак обов'язковою умовою розробки є розгортання ASP.NET на ОС Windows, що є значним обмеженням

2.6.3. Технология *Enterprise SOA*

Enterprise SOA компанії SAP включає не тільки інструментарій для створення веб-сервісів, але і керовані моделі для побудови сервісів і додатків [68]. На відміну від інших провайдерів SAP не тільки надає інструменти розробки, а й створює підтримуючу екосистему, що включає сертифікацію, партнерську мережу і співтовариство корпоративних сервісів.

Для корпоративної SOA потрібно враховувати більше аспектів, ніж просто розкривати функціональність через сукупність сервісів та забезпечити доступ до них з інших місць у вашому додатку. SAP рекомендує певну методологію для розробки у середовищі SOA підприємства, яка підтримує додаткові аспекти SOA для підприємства. Щоб максимально використовувати переваги SOA підприємства, необхідно чітко, перш за все, визначити свої бізнес-вимоги (рис. 2.4).

Це включає визначення бізнес-процесу, який буде побудовано; необхідних для цього сервісів і призначених для користувача інтерфейсів; а також необхідного захисту для середовища SOA. Як тільки це визначення буде завершено, необхідно перевірити через моделювання свої вимоги до існуючих сервісів, щоб виявити, чи забезпечують вони весь необхідний функціонал. Якщо ні, то наступні кроки на рис. 2.4 показують, яким чином виправити це положення.

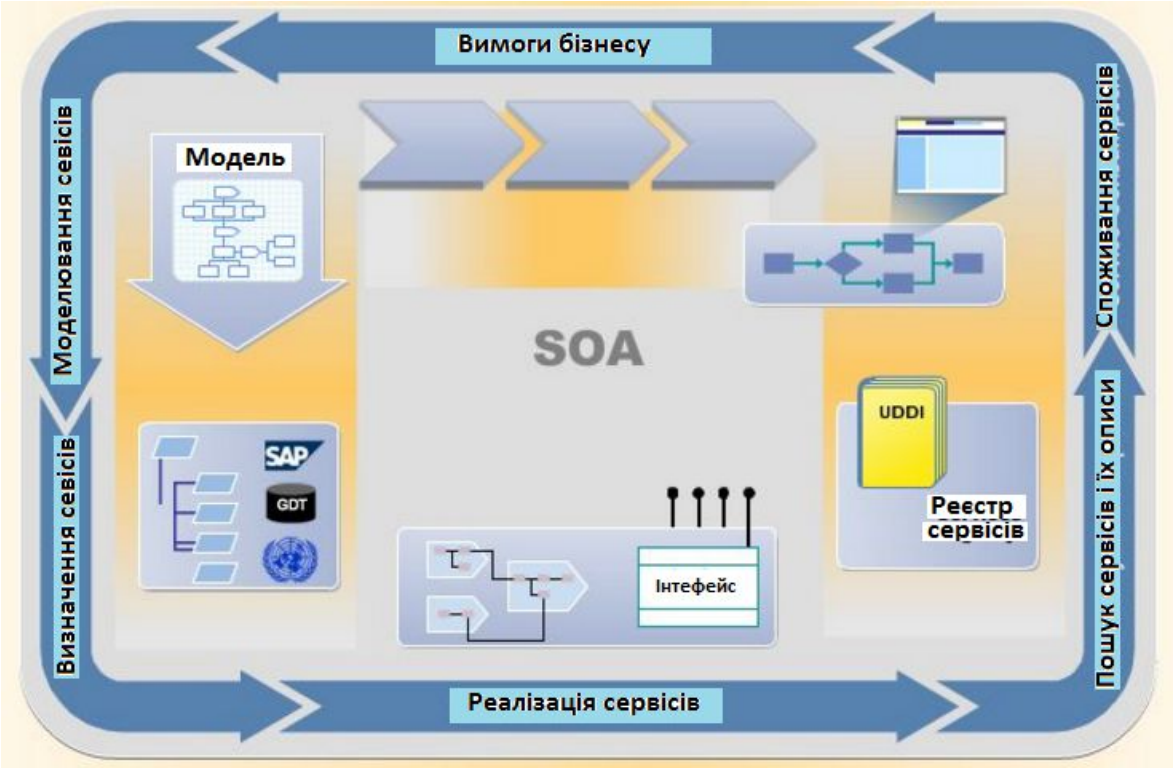


Рис. 2.4. Технология Enterprise SOA

Створення нових сервісів можна розділити на три різні етапи. Починати треба з моделювання цих сервісів, щоб чітко визначити їх бізнес-контекст і бізнес-семантику, включаючи визначення шаблонів зв'язку (синхронний і асинхронний) між задіяними сервісами. Потім потрібно отримати точні визначення сервісів, спираючись моделі бізнес-процесів, які були створили раніше. Ґрунтуючись на бізнес-семантиці, необхідно визначити сигнатури виклику і сигнатури реалізації функцій сервісів, тобто необхідних властивостей API. При цьому можливе повторне використання глобально узгоджених типів даних у розробленому додатку.

Після того, як формування і перевірка вимог до додатку, що створюється, будуть завершені, необхідно розпочати реалізувати відповідну функціональність. ***SAP NetWeaver*** підтримує технологію SAP SOA двома продуктами. Один з них (S*AP NetWeaver Process Integration (PI) 7.1)* забезпечує можливість послідовного вибору кооперативних сервісів з локального реєстру, а другий (SAP NetWeaver Composition Environment (CE) 7.1) дає змогу споживати, розробляти і складати SOA-додатки підприємства. *Композитні* додатки створюються за допомогою

сервісів і можуть (але не обов'язково) використовувати їх на кількох рівнях. Сервіси включають як корпоративні сервіси SAP, так і інші сервіси. Репозиторій корпоративних сервісів є центральним сховищем SOA і має вирішальне значення для використання методології SAP і для забезпечення управління часом розробки.

2.6.4. Oracle SOA Suite 12c

Комплект ***Oracle SOA Suite 12c*** [69] дозволяє створювати і впроваджувати сервіс-орієнтовані архітектури (SOA), а також керувати ними. Компоненти пакета використовують ряд загальних функцій, серед яких узгоджений інструментарій, єдина модель впровадження та управління, комплексні функції безпеки і єдиний центр контролю метаданих.

Простий у використанні, орієнтований на багаторазове використання, єдиний інструментарій розробки додатків і всеохоплююча підтримка управління життєвим циклом дозволяють суттєво знизити витрати і складність розробки та обслуговування (рис. 2.5).

Рис. 2.5. Функціональні компоненти Oracle SOA Suite 12c

Обробка подій у реальному часі вимагає як інфраструктурної складової, так і середовища розробки додатків. Ці вимоги часто включають можливість масштабування від простих повсякденних прикладів з десятками подій на секунду до випадків, коли за секунду надходить безліч подій, і одночасно стоїть вимога реагувати з затримкою у мікросекунди. Крім того, додатки для обробки складних подій повинні уміти виявляти складні шаблони в потоці цих подій. *Обробка подій*

(CEP) – це рішення для створення додатків фільтрації, кореляції та іншої обробки подій у реальному часі таким чином, щоб залежні додатки керувалися аналітикою, що працює дійсно у реальному часі. *Сервісна шина* повністю відповідає вимогам високої доступності і масштабованості. Шина, що розгорнута на декількох серверах додатків у кластерній конфігурації, має високий ступінь відмово стійкості і може бути розширена для виконання великого обсягу операцій шляхом додавання нових серверів в кластер.

Менеджер процесів BPEL – це засіб для створення і виконання бізнес-процесів. Даний продукт надає комплексний і заснований на відкритих стандартах підхід до створення і управління наскрізними бізнес-процесами, які можуть містити в собі як автоматичні дії, так і завдання для користувачів. Це потужний засіб для інтеграції у великих компаніях. Він може бути використаний для інтеграції нових додатків з існуючими системами, створення слабкою зв'язності у сервісній інфраструктурі, створення композитних додатків, автоматизації бізнес-процесів і створення додатків документообігу зі складною структурою маршрутизації і експлуатації. *Моніторинг бізнес-діяльності* складається з сервісів на основі моделей процесів, які написані на мовах, таких як BPMN або BPMN 2.0, що використовуються як інструменти чисто функціональних аспектів бізнес-процесів.

2.6.5. HP SOA Center

Інтегроване рішення для управління та обслуговування програмними комплексами ***HP SOA Center*** компанії Hewlett Packard містить усі елементи, необхідні для успішного створення і експлуатації сервіс-орієнтованої архітектури на підприємстві [70]. З його допомогою можна побудувати і обслуговувати сервіс-орієнтовану архітектуру на підприємстві, ефективно управляти нею і підтримувати необхідні рівні якості і безпеки. HP SOA Center містить усі необхідні інструменти для керівництва та управління на всіх етапах впровадження архітектури SOA, починаючи з тестового запуску і до повного її розгортання на всьому підприємстві. HP SOA Center надає можливість:

- прискорити впровадження SOA і забезпечити необхідну кількість використовуваних сервісів;
- обслуговувати і контролювати при впровадженні SOA ускладнену IT-інфраструктуру підприємства;
- масштабувати SOA під зростаючу кількість сервісів і користувачів.

До складу HP SOA Center входять наступні компоненти.

1) ***HP Systinet*** – готова до роботи платформа, що сприяє якнайшвидшому переходу до сервіс-орієнтованої архітектури та зниженню ризиків для бізнесу, зв'язаних з цією процедурою. Вона включає систему управління *реєстром сервісів* та усі необхідні засоби для впровадження SOA. HP Systinet забезпечує:

- реалізацію набору зрозумілих для користувача сервісів, засоби пошуку сервісів з можливістю всебічного огляду їх стану на кожній фазі життєвого циклу;
- високу надійність і узгоджену взаємодію сервісів, а також формалізовані угоди між їх провайдерами та користувачами;
- управління життєвим циклом сервісів на основі чіткого розуміння вимог організації, включаючи оцінку потреб і створення сервісів, а також їх зміни і видалення в подальшому.

2) ***HP SOA Manager*** – модуль, що дозволяє перейти від пробного застосування SOA до повномасштабної експлуатації. Для побудови централізованої системи керування SOA користувачу надається базовий набір необхідних засобів управління продуктивністю систем і інструментарій усунення проблем і управління політиками.

3) ***HP Business Availability Center for*** *SOA*, що дає змогу скоротити час простою та підвищити якість досвіду користувачів за рахунок оптимізації доступності та продуктивності додатків і бізнесу послуги.

4) ***HP Quality for SOA*** – модуль управління тестуванням, що забезпечує швидкий та наочний тест компонентів додатку, а також дає змогу слідкувати за прогресом тестування.

Перелічені вище інструменти (ASP.NET, Enterprise SOA, Oracle SOA Suite 12c, HP SOA Center) успішно застосовуються у цілому ряді галузей протягом останніх років. На відміну від EGI, INDIGO, EUDAT, зв'язаних з *обслуговуванням потреб науки*, розробники цих інструментів зробили наголос на *обслуговування бізнес-процесів*.

Варто відмітити, що системи, SOA, засновані на веб-сервісах, можуть бути незалежними від технологій розробки і платформ (таких як Java, .NET і т. і.). Наприклад, сервіси, написані на C#, що працюють на платформах .NET, і сервіси на Java, що працюють на платформах Java EE, можуть бути з однаковим успіхом викликані загальним додатком. Програми, що працюють на одних платформах, можуть викликати сервіси, які працюють на інших платформах, що полегшує повторне використання компонентів.

Висновки до розділу 2

1. Ключ до успіху SOA полягає у повторному використанні існуючих IT-ресурсів, наприклад, успадкованих додатків. Однак хмарні MSA мають відмінності від SOA, побудованих на ізольованих е-інфраструктурах, перш за все, в ефективному використанні віртуалізації як апаратних (серверів), так і програмних (ОС) ресурсів.

2. На сьогодні існує два напрями: орієнтація на потреби наукових спільнот, мотивована початком створення з 2016 року Європейської відкритої наукової хмари, і орієнтація на потреби бізнесу, пов'язана з стрімким становленням сервісного сектору економіки.

3. У першому випадку сервіси, необхідні для створення додатків, сьогодні переважно формуються на базі успадкованих наукових застосувань. В роботі узагальнюються дані про такі макросервіси, що опубліковані на сайтах EGI, INDIGO, EUDAT, GIANT. Поки їх небагато, і користувач в змозі за потребі знаходити їх досить просто.

4. У другому випадку сервіси, необхідні для створення додатків, розробляються переважно самими користувачами за допомогою ефективного і різноманітного інструментарію провідних фірм, таких як *Microsoft, Oracle, HP, SAP* тощо. При цьому сервіси, як правило, створюються у вигляді веб-сервісів з семантичним описом (повним чи тільки анотованим), що позволяє впровадити методи автоматичного відкриття необхідних сервісів у дуже великих за обсягом репозитаріях сервісів. Принаймні, веб-сервіси можна створювати і без розглянутого інструментарію провідних фірм, користуючись лише відкритими технологіями і мовами семантичного Веб.

5. Проведений порівняльний аналізу технологій і концепцій SOA і MSA дає підстави визнати більш притаманним для умов хмарного середовища (з точки зору досягнення максимальної віртуалізації ресурсів) використання мікросервісів і контейнерів і організації їх взаємодій. Тому здається конче доцільним рекомендувати надалі (як для наукових, так бізнесових додатків) застосовувати веб-технології для формування та опису мікросервісів і їх композицій, а також контейнерне управлення ним.

6. Перехід на контейнерне управління сервісами за допомогою технології Open API є по суті альтернативною реалізацією сервісної шини підприємства, що не тільки спрощує виклик сервісів додатками (або їх частинами), але і допомагає їм передавати дані і поширювати повідомлення про події з позицій оброблення складних подій.

7. Щоб зробити таке контейнерне управління сервісами найбільш ефективним, треба дослідити можливі структури (топології) сервісних додатків. Неможливе прийняття уніфікованої структури мікросервісу, тому в наступному розділі розглядається архітектури мікросервісів у вигляді серії патернів, які слід вибирати залежно від проектної ситуації.

РОЗДІЛ 3. АРХІТЕКТУРНІ ПАТТЕРНИ СЕРВІС-ОРІЄНТОВАНИХ ДОДАТКІВ

3.1. Монолітна архітектура

Прикладні програми для підприємств розробляються, щоб задовольнити численні вимоги бізнесу. Така прикладна програма пропонує сотні функцій, і усі такі функції, як правило, складаються в **єдиний монолітний додаток**. Поширені *ERP* (Enterprise Resource Planning System) – система планування ресурсів підприємства), ***CRM*** (Customer Relationship Management) – система управління взаємовідносинами з клієнтами) та інші різноманітні програмні системи є хорошими цьому прикладами: вони побудовані як монолітні з декількома сотнями функцій. Розгортання, виправлення неполадок, масштабування та модернізація таких програм дуже ускладнені.

Сервіс-орієнтована архітектура (SOA) була розроблена для подолання труднощів, що виникають внаслідок монолітних застосувань, шляхом введення концепції "сервісу" [70]. Поняття «сервіс» і «бізнес-процес» є взаємозалежними і їх можна використовувати на різних рівнях узагальнення. Наприклад, невеликий процес може бути організований у вигляді окремого сервісу, в тому випадку, якщо його можна типізувати. У той же час процес може бути розбитий на окремі сервіси, які взаємодіють між собою у рамках процесу. Якщо сервісів дуже багато, то не буде досягнута необхідна типізація, але якщо сервіси будуть занадто високорівневими (крупнозернистими), то їх застосування буде утруднено через специфічні відмінностей у бізнес-процесах. Тому сервіс-орієнтовані додатки, перш за все, розрізняються застосуванням технологій SOA и MSA, основні відмінності яких узагальнені у табл. 3.1.

Таблиця 3.1.

Основні відмінності технологій SOA і MSA

Ознака	SOA	MSA
Розмір компоненту	Великі частини бізнес-процесу	Одна функція чи малі частини бізнес-процесу
Зв’язність	Взагалі слабо зв'язані	Завжди слабо зв'язані
Організаційна структура	Будь-яка	Маленькі міжфункціональні команди
Управління	Увага на централізоване управління	Увага на децентралізоване управління
Мета	Забезпечення взаємодії програм, їх інтеграції	Швидке розгортання нових функцій та масштабів розвитку організацій

Отже згідно концепції SOA (MSA) програмний додаток розробляється як сукупність сервісів. Однак використання найпопулярніших веб-сервісів у SOA першого покоління, які складаються з грубозернистих веб-сервісів, сприяє застосуванню програмного забезпечення як квазі-моноліту, якщо вони розгорнуті і виконуються **на одному і тому ж сервері.** Незважаючи на наявність модульного дизайну, програма розгортається як моноліт. Наприклад, якщо використовується Java, то програма складається з одного WAR-файлу, який працює на веб-контейнері, такому як Tomcat. Ускладнення веб-сервісів із-за накопичення додаткових функцій перетворюють SOA першого покоління на монстрів, які не відрізняються від звичайних монолітних застосувань.

На рис. 3.1 показано програмне забезпечення для мобільної медицини, яке складається з кількох сервісів. Передбачається реалізація усіх сервісів додатку як єдиної програмної системи. Тобто усі сервіси реалізовані за допомогою одного набору технологій (наприклад, мови програмування) і використовують загальні бібліотеки коду. Усі сервіси працюють з одним сервером баз даних, що дозволяє кожному сервісу звертатися до бази даних безпосередньо. Усі ці сервіси

розгортаються в один і той самий час виконання програми. Тому програмний додаток демонструє характеристики монолітного застосування: він складний, розроблений і розгорнений як єдине ціле. Але на відміну від справжнього моноліту *кожна зміна коду не призводить до повної перебірки і розгортання нової версії серверної частини програми, а тільки оновленню відповідного йому веб-сервісу.*

Рис. 3.1. Квазі-монолітна архітектура

Існує ще декілька інших недоліків такого підходу. Квазі-моноліт потрібно масштабувати як єдине додаток, і його важко масштабувати в умовах суперечливих вимог до ресурсів (наприклад, для одного сервісу потрібно більше процесорів, а для іншого – більше пам'яті). Один несталий сервіс може призвести до «падіння» всієї програми, і взагалі важко інноваційно впроваджувати у цьому випадку нові технології та фреймворки. Однак в той же час розглянутий SOA підхід забезпечує ефективну інтеграцію різних програм і інформаційних систем.

Головна ідея, що стоїть за MSA, полягає у тому, щоб будувати великі, складні та довготривалі програми як сукупність узгоджених мікросервісів, які розвиваються з часом. Термін "мікросервіс" говорить про те, що сервіси повинні бути невеликими. Користувач має намагатися розбити свою систему на сервіси, щоб вирішити задачі, пов'язані з розробкою та розгортанням. Деякі сервіси можуть бути крихітними, навпаки, інші можуть бути досить великими. В MSA розгортання мікросервісів відіграє ключову роль і задовольняє таким ключовим вимогам:

- Можливість розгортання незалежно від інших мікросервісів.
- Здатність до масштабування на кожному рівні мікросервісу (обраний сервіс може отримати більше трафіку, ніж інші сервіси).

- Швидке розгортання мікросервісів.
- Несправність одного мікросервіса не повинна впливати на будь-які інші сервіси.

Програмне забезпечення для керування полегшує адміністрування програмних модулів, розміщених у розподіленому середовищі. Модулі, які називаються контейнерами, є основними будівельними блоками мікросервісної архітектури. Контейнери використовуються для створення хмарних розподілених програм та перетворення монолітних успадкованих пакетів програм, які не були розроблені для віртуальних середовищ. Програмне забезпечення для керування контейнерами спрощує процес додавання або заміни контейнерів, коли виникає потреба, і полегшує організацію взаємодії великої кількості контейнерів, автоматизуючи більшу частину ручної роботи.

Docker (драйвер з відкритим кодом, який дозволяє розробникам та системним адміністраторам розгортати самодостатні контейнери додатків в середовищах Linux), є відмінним способом розгортання мікросервісів, що відповідають наведеним вище вимогам. Основні кроки, що виконуються, наступні:

- Упакування мікросервісу як зображення Docker контейнера.
- Розгортання кожного екземпляру сервісу як контейнер.
- Масштабування шляхом зміни кількості екземплярів контейнера.

Побудова, розгортання та запуск мікросервісів будуть набагато швидшими, якщо використовувати Docker контейнери, а не традиційні віртуальні машини VM. Kubernetes розширює можливості Docker, дозволяючи керувати кластером контейнерів Linux у вигляді єдиної системи, запускати контейнери Docker на кількох хостах, пропонуючи спільне розміщення контейнерів, відкриття сервісів і управління реплікацією [71]. Отже, використання Kubernetes (на вершині Docker) для розгортання мікросервісів стало надзвичайно потужним підходом, особливо для широкомасштабних розгортань мікросервісів. Перелік базових APIs, ESBs, платформ і фреймфорків наведено у додатку 2.

3.2. Відмінності взаємодії сервісів в архітектурі SOA і MSA

Сервісно-орієнтована архітектура (SOA) передбачає блокову систему з взаємодіючих компонентів, в якій різні функціональні модулі додатків (сервіси), призначені для *управління бізнес-процесами* (Business Process Management, BPM), пов'язані між собою за допомогою чітко визначених інтерфейсів [72,73]. Інтерфейси самі по собі незалежні від оточення і платформи, і тому така модель отримала назву моделі "*слабкою зв'язку*". Фактично суть концепції SOA полягає в уніфікації та автоматизації бізнес-процесів за допомогою типових компонентів – сервісів (наприклад, веб-сервіси використовують один стандарт – розширювану мову розмітки XML). При цьому створення, впровадження або зміна бізнес-процесу пов’язана з компоновкою (оркеструванням) раніше розроблених сервісів, призначених для автоматизації бізнес-функцій.

Мікросервісна архітектура (MSA) – це архітектура на основі вільно зв'язаних мікросервісів з обмеженими контекстами, коли єдиний програмний додаток також розбивається на набір невеликих сервісів, кожен з яких незалежний і виконується у своєму власному контейнері та спілкується з іншими сервісами через API REST (див. Розділ 2). При цьому кожен мікросервіс відображає окрему бізнес-функцію і розгортається незалежно з використанням повністю автоматизованого середовища.

Мікросервіси повинні бути дрібнозернистими, а протоколи – легкими. Тоді їх застосування простіше зрозуміти, їх легше розробляти і тестувати. *Кожен мікросервіс реалізований як окрема програмна система, часто у кожного сервісу є своя база даних.* Оскільки мікросервіси не мають доступу до бази даних іншого мікросервісу – доступ до таких даних здійснюється викликом інших мікросервісів ресурсу. Часто мікросервіси групують, якщо вони реалізують схожий або тісно зв'язаний функціонал. Сьогодні архітектурний стиль мікросервісів приймається багатьма ключовими технологічними гравцями, такими як Netflix, Amazon, The Guardian.

Структура програмного забезпечення для онлайн-роздрібної торгівлі (рис. 3.1) може бути реалізована за допомогою MSA, як показано на рис. 3.2, де на основі бізнес-вимог введений додатковий мікросервіс, створений з оригінального набору сервісів просто розщепленням першого сервісу.

Рис. 3.2. Приклад мікросервісної архітектури

Мікросервісна архітектура зазвичай використовується в одному із двох випадків: коли мова йде про створення програмного забезпечення з нуля або про перетворення існуючих програми / сервісів на сукупність мікросервісів. У будь-якому випадку, важливо правильно визначити розмір, обсяг та можливості мікросервісів. Це, мабуть, найважче, з чим стикаються розробники, коли реалізують MSA на практиці. У випадку MSA вихідний код кожного окремого сервісу не повинен перевищувати декількох сотень рядків коду. SOA ж такого обмеження начебто не накладає, тому в кожному сервісі можуть бути десятки або сотні тисяч рядків коду. Можна вважати, що MSA – це окремий випадок SOA (рис. 3.3).

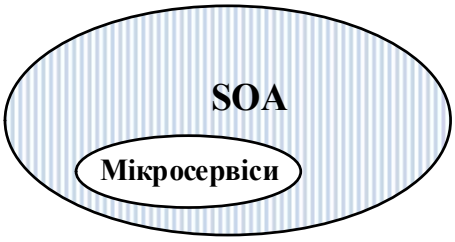


Рис. 3.3. Співвідношення між SOA і MSA

Мікросервісну архітектуру можна розглядати як дрібнозернисту SOA без застосування багажу WS * і ESB. Але вона відрізняється від традиційних SOA і, що більш важливо, вирішує багато проблем, від яких сьогодні страждають багато організацій.

"Мікро" є не зовсім визначеним терміном: більшість розробників схильні думати, що вони повинні намагатися зробити цей сервіс максимально малим за розміром. Доцільність використання мікросервісної архітектури залежить від конкретної ситуації. Мікросервісна архітектура має доволі значний недолік – збільшення значення затримок при комунікації між сервісами. Тобто, при ідентичній реалізації сервісів, квазі-монолітна архітектура може мати кращі показники швидкодії, ніж мікросервісна. З іншого боку, мікросервісна архітектура дає більше можливостей для масштабування, що є суттєвою перевагою для систем, що швидко розвиваються.

Перед використанням мікросервісної архітектури доцільно провести аналіз системи, що розроблюється, і визначити, наскільки прийнятними є показники затримки при комунікації між мікросервісами, і чи не призведуть вони до низької швидкодії системи і неможливості подальшого її масштабування. У MSA бізнес-функції розгортаються декількома мікросервісами, при чому кожному мікросервісу потрібно мати власну базу даних (рис. 3.4).

Для кожного мікросервісу можна вибрати свою мову і бібліотеки, які підходять саме для розв'язуваного цим сервісом завдання.

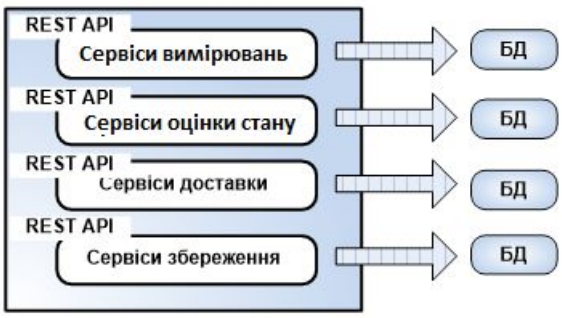


Рис. 3.4. Мікросервіси зі своїми базами даних

Якщо потрібна швидкість, беремо C ++, якщо просто ганяємо туди-сюди дані, беремо Erlang, якщо потрібна якась незвичайна бібліотека, беремо Clojure [74].

Наведемо ключові аспекти впровадження децентралізованого управління даними в MSA:

- Кожен мікросервіс може мати приватну базу даних, щоб зберегти дані, необхідні для реалізації пропонованих бізнес-функцій.
- Даний мікросервіс може отримати доступ лише до приватної бази даних, але не до баз даних інших мікросервісів.
- В деяких бізнес-сценаріях може знадобитися оновити кілька баз даних для однієї транзакції. У таких сценаріях бази даних інших мікросервісів слід оновлювати тільки через їх власні службові API (не дозволяється безпосередньо отримувати доступ до бази даних).

Наявність шлюзу API дозволяє кожному мікросервісу використовувати механізм зв'язку, який підходить до нього, без внесення будь-яких змін до клієнта. Не може бути універсального і фіксованого шаблону для кожного мікросервісу у його взаємодії з іншими сервісами.

3.3. SOA і MSA, керовані запитами

Веб-сервіси є технологію реалізації концепції дизайну SOA. Існує велика кількість досліджень, присвячених вимогам надійності SOA і, більш конкретно, веб-сервісів. Веб-спільнота розробила ряд специфікацій, які підтримають надійний обмін повідомленнями, управління транзакціями, реплікації і безпеку. Сервіси взаємодіють один з одним за допомогою шаблону синхронного запиту і відповіді, що забезпечує щільне зчеплення між компонентами системи (рис. 3.5). Тобто реалізується зв'язок «один-на-один» – один з конкретних сервісів викликається у часі тільки одним із споживачів, але зв'язок є двонаправлений [73].

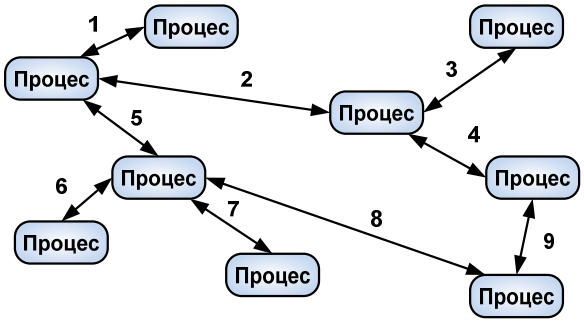


Рис. 3.5. Архітектура SOA, керована запитами

Орієнтація на сервіси та SOA виправдовується краще тоді, коли процеси або їх частини є стандартними, коли вони часто повторюються без змін або коли декільком користувачам потрібний той самий компонент процесу для виконання своїх завдань. Виклик (відкриття) сервісів у SOA реалізується за допомогою віддаленого виклику процедури ***RPC (Remote Procedure Calls***) на вимогу споживача сервісів. Синхронна SOA добре зарекомендувала себе для побудови великих корпоративних програмних додатків. Цілий ряд розробників та інтеграторів пропонують інструменти і рішення на основі SOA (наприклад, платформи ***Intel SOA Expressway***, ***JBoss SOA Platform***, ***IBM WebSphere***, ***Software AG webMethods, Oracle / BEA Aqualogic, Microsoft Windows Communication Foundation, SAP NetWeaver, TIBCO***) [75-79].

У шаблоні "***запит-відповідь***" вся логіка маршрутизації повідомлень розташована на кожній кінцевій точці, і сервіси можуть безпосередньо спілкуватися між собою. Підтримка синхронізації даних між мікросервісами стає набагато складнішою у випадку мікросервісних додатків, оскільки кожен мікросервіс має свої власну базу даних, яка може бути доступна / модифікована тільки за допомогою API REST (рис. 3.6).

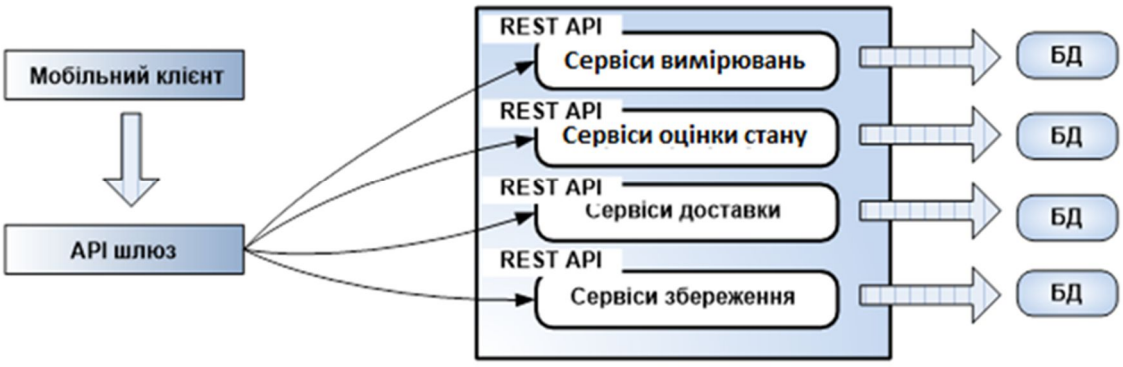


Рис.3.6. Спілкування мікросервісів через API-шлюз

Очевидно, що ця модель придатна для порівняно простих додатків, заснованих на мікросервісах, оскільки зі збільшенням кількості сервісів вона стане більш громіздкою і складною в обслуговуванні. Тому для випадку складних мікросервісних додатків замість шаблону «запит-відповідь» можна

використовувати API шлюз в якості єдиної точки входу у систему, побудовану на базі мікросервісів. По суті, це поєднання управління MSA і API (рис.3.6).

В сценарії мобільної медицини, як показано на рис. 3.6, всі мікросервіси виставляються через API-шлюз, який є єдиною точкою входу для всіх клієнтів. Якщо мікросервіс хоче взаємодіяти з іншим мікросервісом, то він повинний це зробити через API-шлюз.

Шаблон API-шлюзу має наступні переваги:

- Можливість надавати необхідні абстракції на рівні шлюзу для існуючих мікросервісів, наприклад, замість того, щоб надати універсальний API-інтерфейс, що відповідає всім можливим вимогам, API-шлюз може запропонувати різні API для кожного клієнта.
- Полегшена маршрутизація повідомлень / перетворень на рівні шлюзу: можна виконувати основне фільтрування повідомлень, маршрутизацію та перетворення у шлюзовому прошарку.
- Централізоване застосування на рівні шлюзу нефункціональних можливостей, таких як безпека, моніторинг та дроселювання замість реалізації цих загальних функцій на кожному рівні мікросервісу.

Шаблон API-шлюзу стає найбільш поширеним шаблоном у більшості реалізацій мікросервісної архітектури. Ряд розробників та інтеграторів пропонують архітектури і рішення на основі MSA [80-82].

3.4. Оркестрування мікросервісів при їх взаємодії

Створюючи прикладну програму, яка використовує мікросервіси, потрібно визначитися з організацією взаємодії мікросервісів. Коли сервіс-орієнтована архітектура використовує схему оркестрування для взаємодії сервісів, то між сервісами встановлюється зв'язок «запит-відповідь». Тобто один сервіс запитує API іншого сервісу, в результаті чого створюється мережа шляхів зв'язку між усіма сервісами. Інтеграцію, зміну або видалення сервісів із цієї мережі здійснити важко, оскільки необхідно знати про кожне з'єднання між сервісами.

Оркестрування – це традиційний спосіб взаємодії між різними сервісами у сервіс-орієнтованої архітектурі (SOA) [83-85]. Для реалізації оркестрування, як правило, є один контролер, який виступає у ролі "диригента" (оркестратора) загальної взаємодії сервісів за шаблоном «запит / відповідь» (рис.3.7). Наприклад, якщо потрібно викликати у певному замовленні три сервіси, оркестратор звернеться до кожному з них, чекаючи відповіді, перш ніж викликати наступний.

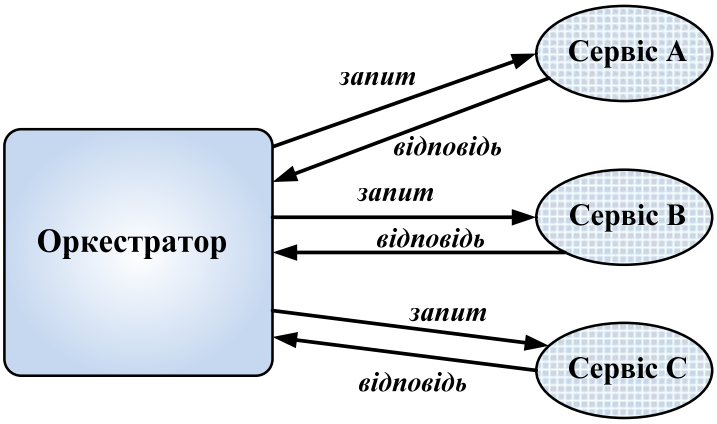


Рис. 3.7. Взаємодія сервісів за шаблоном типу «запит / відповідь»

Переваги і вади оркестрування приведені в табл.3.2.

Таблиця 3.2

Характеристики оркестрування сервісів

Переваги	Вади
Надає ефективний засіб контролю потоку програми, коли відбувається синхронна обробка. Наприклад, якщо Сервіс А необхідно успішно завершити, перш ніж буде викликаний сервіс В.	Об'єднує сервіси разом, створюючи залежності між ними. Якщо сервіс А вимкнений, сервіси В та С ніколи не будуть викликані.
	Якщо для всіх запитів є центральний загальний екземпляр оркестратора, то він створює одну точку загрози збою програми.
	Використовує синхронні обробки, які блокують запити. У цьому прикладі загальна тривалість обробки – це сума часу, необхідного для виклику сервісу А + сервісу В + сервісу С.

Випадки доцільного використання оркестрування сервісів:

- Якщо усі або більшість кроків потрібно робити послідовно з мінімальними можливостями для паралельної обробки.
- Якщо є бажання централізувати загальний контроль потоку. Зокрема, якщо можливість бачити повний об'єм потоку в одному місці має високий пріоритет у порівнянні з часом розробки або часом виконання. Однією із умов, коли це може статися, є випадок, коли існують сотні сервісів, кожен з яких має декілька різних потоків, які базуються на даних, що обробляються. У цьому випадку може бути простіше керувати потоком у центральній частині, а не розподіляти його.
- Роз'єднання (декомпозиція) не є пріоритетом.

3.5. SOA і MSA, керовані подіями (EDA)

У бажанні створення підприємств реального часу, які постійно підключені і завжди доступні в Інтернеті, організації стикаються з більш різноманітними бізнес-сценаріями і виявляють потреби у альтернативних шаблонах проектування як доповнення до синхронної SOA (MSA), керованої запитом. Подійно-керована архітектура (EDA, Even Driven Architecture) включає в себе три базові компоненти: генератор події (датчик), оброблювач події і менеджер подій (відповідач) [86-90]. Менеджер подій реєструє усі виникаючі в системі події (зовнішні і внутрішні), відповідним чином ідентифікує і передає оброблювачу подій. Якщо потрібний оброблювач з яких-небудь причин не доступний, подія залежно від конкретної реалізації або ставиться у чергу, або передається іншому оброблювачу. Завдяки такій схемі система стає максимально гнучкою і чуйною на зміни інформаційного середовища – при виникненні принципово нової події досить підключити новий оброблювач, не зачіпаючи ядра системи. Подібні системи зазвичай будуються на тригерах, які реагують на певні події і ініціюють відповідні веб-сервіси, рис. 3.8 [73].

Обидві нові архітектури синхронна SOA (MSA), керована запитом, і EDA відрізняються від традиційної монолітної архітектури тим, що бізнес-процес

розбивається на малі процедури, що повторно використовуються. Ці процедури формалізуються і реалізуються у вигляді IT-компонентів, які слабо зв'язані і можуть бути інтегровані у динамічний і гнучкий спосіб.

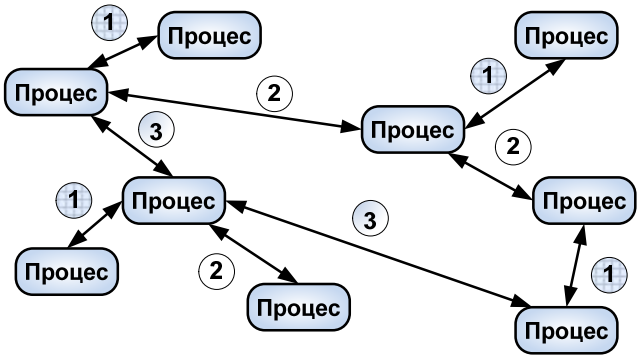


Рис. 3.8. EDA архітектура з трьома подіями

Синхронна SOA (MSA) і EDA намагаються замінити і розбити старі і великі успадковані системи і архітектури на більш гнучкі і багаторазово використовувані бізнес- та IT- компоненти.

Джерело події знає тільки створену подію і не має ніякого поняття про спосіб її подальше оброблення або про зацікавлені сторони. Цей принцип використовується при інтеграції роз’єднаних додатків і бізнес-процесів. Процес складається з декількох етапів. У EDA етапи не залежать фізично або у логічний спосіб один від одного, так що кожен може бути змінений, не викликаючи побічних ефектів, на інші, поки повідомлення про події не змінюються. Реалізується з***в'язок «один-до-багатьох»***, тобто на одну конкретну подію може відгукнутися багато передплатників. При цьому підтримуються асинхронні операції через повідомлення про події.

Для оброблення подій та ініціалізації веб-сервісів можуть бути використані три стратегії: ***проста***, ***потокова*** і ***комплексна.*** Просте оброблення (Simple Event Processing) полягає в ініціалізації веб-сервісів по мірі реєстрації відповідних подій. Основна перевага такої системи – це робота в режимі реального часу. Мінус очевидний – не враховується фактор пріоритету. Потокове оброблення (Event Stream Processing) враховує існуючі залежності веб-сервісів і об'єднує кілька сервісів у один загальний потік. Даний підхід виправдовує себе в системах

з великими накладними витратами на пошук і вилучення інформації з бази даних. І, нарешті, комплексне оброблення (CEP, Complex Event Processing) – це так зване управління за відхиленнями. Будь-яка подія розцінюється як вихід системи зі стану рівноваги. Ініціалізація веб-сервісів переслідує єдину мету – повернути систему у стан рівноваги (між справою задовольняючи потреби клієнтів системи).

Новітній хмарний механізм оброблення подій **CEP** знаходиться в експлуатації з 2007 року [78]. В ньому реалізовані такі методи, як виявлення складних візерунків багатьох подій, кореляції подій і абстракції, ієрархії подій і виявлення відносин між подіями, таких як причинність, підпорядкованість, синхронізація зв'язок з процесами, зв’язаними з подіями. У той час як синхронна SOA (MSA) і оркестрування автоматизують процеси, CEP автоматизує інтелектуальну кореляцію і співвідношення між рішеннями, прийнятими персоналом. Система CEP дозволяє користувачам оброблювати випадкові, не пов'язані події, визначати тенденції і передбачувати результати. Заходи можуть бути вжиті для запобігання негативного наслідку від виникнення події. Шаблони реакцій також можуть бути проаналізовані, щоб поліпшити основні процеси та інформаційні системи з ціллю запобігання майбутніх помилок.

Бізнес-аналітики та керівники середньої ланки добре розуміють процеси, керовані подіями, оскільки вони зв'язані з стандартними бізнес-процедурами, такими як відносини замовника і постачальника, операційні процедури, процеси і потоки завдань.

EDA і синхронна SOA (MSA) можуть бути реалізовані і функціонувати паралельно без будь-яких суперечностей. Існує три основні передумови для цієї природньої співпраці. Перша – це спільні та визначені бізнес-цілі, друга – використання роз’єднаних компонентів і спільної моделі даних, третя – використання спільної інфраструктури і технологій. EDA і синхронна SOA (MSA) є взаємодоповнюючими у багатьох аспектах. Додавши EDA на вершину синхронної SOA (MSA) архітектури, можна отримати нові можливості. Обидві архітектури синхронна SOA (MSA) і EDA є досить новими концепціями і знаходяться на ранніх стадіях становлення. Багато компаній сьогодні здійснюють

впровадження синхронних SOA та реалізацію очікуваних користі та цінностей [89,91,92]. EDA протягом багатьох років використовувалася в якості технічного шаблону проектування, який забезпечує масштабованість і високу продуктивність транзакцій. Обидві синхронна SOA (MSA) і EDA націлені на узгодження бізнесу та IT-технології, але за допомогою різних засобів. У той час, як синхронна SOA (MSA) використовує бізнес-запити/відповіді, EDA використовує бізнес-події для подолання розриву між бізнесом та IT.

Подійно-керований шаблон EDA сприяє декомпозиції зв'язаної системи (або бізнес-процесу) у порівнянні з вимогою слабко-пов’язаної системи для SOA. Тобто функціональні процеси в системі відправника, де відбулася бізнес-подія, не залежать від наявності та завершення віддалених процесів у розподілених нижчестоящих системах. У той час, як в архітектурі, керованій запитом, система відправника повинна точно знати, які розподілені сервіси відгукуються на виклик, тобто залежить від наявності та завершення цих віддалених сервісів.

Архітектура EDA, керована подіями, має значення тільки, коли події опубліковані, споживаються і поширюються ***в режимі реального часу***. Якщо цей підхід застосовується у пасивних сценаріях оброблення подій, наприклад, орієнтованих на фіксовану графіку, то його ефективність не дуже відрізняється від традиційних методів інтеграції даних, що використовуються сьогодні.

Оскільки кожен мікросервіс має свої власну базу даних, яка може бути доступна / модифікована тільки через його API / кінцеву точку, то ***MSA притаманна більш подійно-керована архітектура.*** У цій архітектурі мікросервіс публікує подію, коли щось значне відбувається, наприклад, коли він оновлює запис в базі даних. Інші мікросервіси підписатися на ці події. Коли мікросервіс отримує інформацію про подію, він може оновлювати відповідні записи у своїй власній базі даних, що може привести до появи більшої кількості подій, які публікуються і споживаються іншими сервісами.

Події можуть управляти наступними способами:

- використанням черг повідомлень (messaging queues), таких як RabbitMQ або ZeroMQ;

- використанням бази даних в якості проміжного шару між сервісами, коли один сервіс створює і записує подію в БД. Інші сервіси постійно запитують цю БД і починають діяти, коли виявляють запис про подію.

Варто відзначити, що подійно-керована архітектура при переході до MSA стає майже базовим типом архітектури додатків на відміну від SOA першого покоління, що означає більш високу складність таких додатків. Це призводить, як відмічалося, до значного зростання комунікаційного навантаження у порівнянні з транзакціями в пам'яті монолітних додатків. Мікросервісам потрібна мова опису інтерфейсу, так щоб з її допомогою можна було легко реалізувати клієнтів, з одного боку, а з другого, вона була типізованою звичною мовою (Swagger [93], proto [95], WSDL).

Швидше за все, не у всякій задачі можна або потрібно використовувати мікросервісную архітектуру. Але там, де є така можливість, отримані переваги істотно перевершують недоліки.

3.6. Хореографія мікросервісів при їх взаємодії

При створенні мікросервісної архітектури бажано уникнути появи залежностей між мікросевісами. При подійно-керованому підході немає центрального оркестратора, який контролює логіку того, які кроки відбуваються, бо вся ця логіка вбудовується у кожний мікросервіс заздалегідь [96]. Така взаємодія більш схожа на хореографію, коли один сервіс не вказує іншому сервісу, що йому робити. Натомість кожен сервіс спостерігає за своїм оточенням і реагує на автономні події. У реальному житті це виглядає так: мікросервіси підключаються до потоку повідомлень про події і підписуються на події, у яких вони зацікавлені (рис. 3.9).

Після того, як відбувається серія подій, що стосуються мікросервісу, сервіс виконує відповідні дії. Тепер легко додати нові мікросервіси до архітектури: просто потрібно підключити їх до потоку подій. У гіршому випадку необхідно переконатися, що інші мікросервіси генерують події, які потребують новий

сервіс. Але додавання додаткових подій або розширення корисної завантаженості існуючими подіями не порушить існуючої логіки.

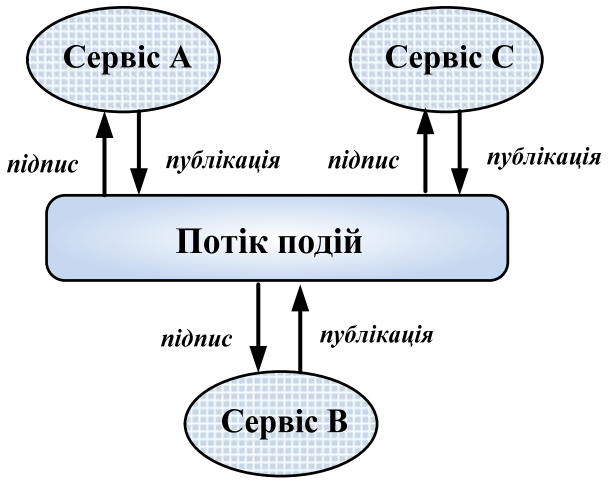


Рис. 3.9. Взаємодія сервісів за шаблоном типу «публікація / підпис».

Мікросервіси знають, як відреагувати на події, використовуючи потік подій для асинхронного спілкування. Кілька мікросервісів можуть споживати однакові події, виконувати певну обробку, а потім генерувати власні події у потоці повідомлень, може, навіть одночасно. Потік події не має ніякої логіки і призначений лише для того, щоб бути середовищем (трубою) обміну повідомленнями.

Асинхронний характер подійно-керованої архітектури усуває блокування або очікування, що відбувається при обробці типу оркестрування (запит / відповідь). Мікросервіси можуть створювати події та продовжувати обробку. Використання потоку подій для цього дозволяє відокремити зв'язок між виробником і споживачами – виробнику не потрібно знати, чи споживач почав працювати перед тим, як він створив подію, або після отримання події, яка була їм зроблена. Крім того, споживачі / виробники можуть споживати / генерувати події з потоку подій.

Переваги і вади хореографії приведені в табл.3.3.

Таблиця 3.3

Характеристики хореографії сервісів

Переваги	Вади
Дозволяє швидше обробляти дані, оскільки сервіси можуть виконуватися паралельно / асинхронно	Складність переміщення. Замість того, щоб керування потоком було централізовано оркестатором, керування потоком розділено і розподілено за окремими сервісами
Легше додавати / оновлювати сервіси, оскільки вони можуть бути просто підключені до / з потоку подій	Кожен сервіс матиме власну логіку потоку, і ця логіка визначить, коли і як він повинний реагувати на конкретні даних у потоці подій.
Ефективний підхід до реалізації швидкої доставки, оскільки події (команди) можуть зосередитись на конкретних сервісах, а не на цілій програмі.	
Контроль розподіляється, тому більше не існує жодного оркестратора, який виступає як центральна точка ризику щодо збою.	
Кілька заходів може бути використано у подійно-керованій архітектурі для надання додаткових переваг. Наприклад, якщо зберегти потік подій, то потім можна відтворити їх. Таким чином, якщо сервіс «впав», коли події все ще генеруються, то під час свого повернення до Інтернет він може відтворити ці події, щоб продовжити обробку.	

Випадки доцільного використання лише подійно–керованого підходу:

- Якщо усі або більша частина необхідної обробки може бути виконана асинхронно. Шаблон подійно-керованої архітектури чудово підходить для обробки, яка має відбуватися паралельно. Якщо тільки мінімальна частина вашої обробки може бути виконана асинхронно, то подійно-керований шаблон може бути надмірним.

- Якщо децентралізація потоку у кожному сервісі є керованою. Кореляційні ідентифікатори можуть бути використані для відтворення централізованого представлення даних для моніторингу та аудиту.
- Якщо швидкість появи на ринку є пріоритетом. Подійно-керована мікросервісна архітектура сприяє розробкам вийти на ринок швидше.

3.7. Мультиагентна архітектура системи сервісів

SOA і MSA можуть отримати вигоду зі застосування технологій багатоагентних систем (Multi-Agent systems, MAS), зокрема, шляхом запозичення їх координаційних механізмів, протоколів взаємодії та інструментарію прийняття рішень. Багатоагентні системи – це особливий погляд на програмування, що розширює сприйняття за межі звичного об'єктно-орієнтованого програмування (ООП). Основна ідея агентів – *це делегування*. Власник або користувач агента делегує йому деяку задачу, і агент автономно виконує цю задачу від імені користувача. Агент повинен бути здатний зв'язатися з користувачем для отримання інструкцій та забезпечення користувача результатами. Нарешті, агент повинен бути здатний контролювати стан свого довкілля і у разі необхідності вживати дії, спрямовані на виконання делегованого йому завдання.

Ідея про делегування складних завдань програмним системам (агентам) дозволяє програмно представляти і більш природним чином вирішувати складні задачі, що не формалізуються. Мультиагентна система являє собою програмну систему, у якій кілька агентів співпрацюють, щоб досягти певної мети. Для кожного окремого агента поставлене перед системою завдання було б нездійсненим, тому агенти повинні вирішувати її разом. При вирішенні завдання агенти, у межах свого простору, можуть обмінюватися інформацією, делегувати один одному підзадачі.

З точки зору сервіс-орієнтованої архітектури традиційні агентні підходи необхідно адаптувати до наступних часто виникаючих вимог: (а) вони повинні забезпечити значну масштабованість застосувань, (б) вони мають бути більш Інтернет-базованими, і (в) вони повинні інтегрувати корпоративні додатки. Тобто

слід розглядати *агентно-керовані обчислення (ADC, Agent-driven computing*) на ряду із сервіс-орієнтованими обчисленнями (SOC). Взаємодія між агентом і сервісом може поліпшити функції і діапазон дії агента; в той час як діалог між ADC і SOC (MSA) може забезпечити двосторонні вигоди для них обох, які сприяють підвищенню ефективності інтеграції корпоративних додатків. Технологія SOC добре притаманна для інтеграції і управління Інтернет-базованими додатками і забезпечення інтероперабельності ПЗ підприємства. Проте, SOC виявилися слабкими у підтримці якості таких властивостей систем сервісів, як їх автономія і гнучкість.

Взагалі ***агент*** і ***сервіс*** – це два незалежні обчислювальні поняття. Але можливості ADC і SOC цілком доповнюють одна одну. Агенти і ADC краще можуть забезпечити внутрішню якість додатків, у той час як сервіси і SOC є більш придатними для реалізації послуги «Software-as-service» і архітектури «додаток-додаток». Різниця між об'єктами, агентами і сервісами не є абсолютною. З одного боку, агент пропонує інтерфейс повідомлень, який може бути доступним через веб-браузер. З іншого боку, реалізація сервісу може бути інтерпретована як неавтономний агент, який реагує на запити. Сервіс як абстрактне поняття може бути реалізований за допомогою конкретного агента. Такий агент може бути часткою програмного або апаратного забезпечення, яка посилає і приймає повідомлення, у той час як сервіс є ресурсом, що характеризується абстрактним набором наданої функціональності. Інтеграція агентів і сервісів, а також інтеграція з ADC і SOC може принести користь кожному підходу до створення потужної обчислювальної системи. Це спонукає на розроблення інтегрованого концепту агентного сервісу і агентної сервіс-орієнтованої архітектури.

Процедури пошуку (ServiceDiscovery) і композиції сервісів (ServiceComposition) можуть скористатися перевагами програмних агентів. З одного боку, програмні агенти можуть сприймати сервіси для того, щоб визначити можливу координацію сервісів (ServiceCoordination). З іншого боку, вони можуть грати активну роль у композиції сервісів (ServiceComposition) для

того, щоб адаптувати різні політики композиції, ввести в дію додаткового партнера або зрозуміти контекст нефункціональних атрибутів.

На рис. 3.13 наведена структурна схема середовища ***ACTAS (Adaptive Composition and Trading with Agents for Services***) [99], призначеного для адаптивної композиції і пошуку сервісів за допомогою агентів. Тобто композитні сервіси можуть адаптуватися до зовнішнього середовища і до змін складових під час виконання.

ACTAS використовує наступні агенти: агент запиту *ReA* (Request Agent), агент об'єкту FA (Facility Agent), трейдер-агент *TRA* (Trader Agent) і агент композиції *CoA* (Composition Agent). Агентне середовище також містить агент особи *PA* (Personal Agent), що представляє користувача додатків, тобто потенційного клієнта обслуговування. Агент *PeA* реалізує інтерфейс для користувальницького додатка.

Агент *FA* виконує кілька завдань: (1) публікацію Service Offer в Service Provider, (2) управління доступністю до Service Modes, (3) потенційне резервування обраної Service Mode, (4) управління ресурсами, (5) погодження з іншими агентами *FA* вибору Component Services і (6) розгортання сервісу, коли він не є Abstract Service (реферативним сервісом).

ACTAS передбачає генерацію *Service Request* всередині *Application Environment*, що веде до адресації агента *PeA*. Згодом *PeA* створює *CoA* для цього запиту. У простому випадку веб-браузер може взяти на себе роль *PeA* в Application Environment. При успішній операції Service Discovery агент CoA повернеться з набором сервісних кандидатів. Оскільки агент ReA зв'язаний з певним додатком, алгоритм *CoA* може бути адаптований до цього запиту. Таким чином, *CoA* може, наприклад, повернути тільки один відповідний сервіс або повертати декілька кандидатів, залежно від політики застосування Завдання на композицію агента *CoA* порівнюється з конкурентними (паралельними) завданнями на композицію сервісів. На відміну від методів координації конкурентних процесів методи композиції можуть адаптуватися. Агенти системи

ACTAS будують проміжне ПЗ для декларативного середовища ACTAS, яке реалізує Service Discovery Service і Service Composition.

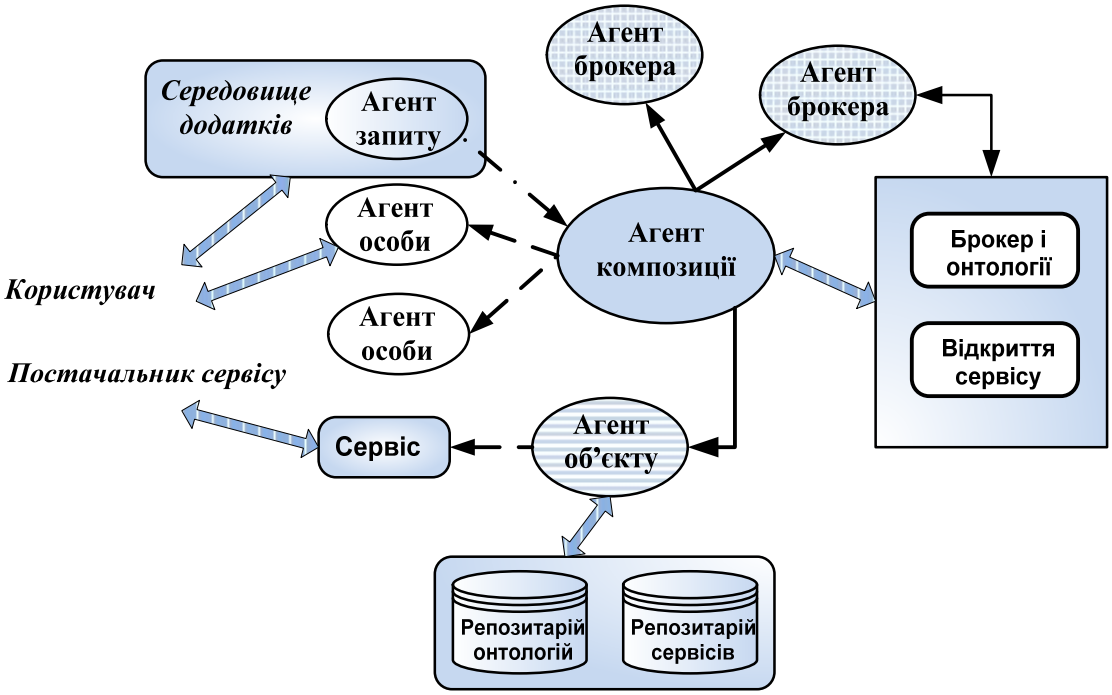


Рис. 3.12. Агентне середовище ACTAS

Розглянуті агентні реалізації процедур пошуку і композиції сервісів технічно складніші, ніж аналогічні їх реалізації засобами мови BPMN або WS-BPEL і робочих потоків (workflow) для SOA. Тому вони поки не знайшли широкого застосування у системах сервісів, при проектуванні яких наголос робиться на забезпечення можливості самим користувачам приймати активну участь в цьому процесі.

Можливий варіант MSA з агентною координацію сервісів (ServiceCoordination) представлений на рис.3.13. Він реагує на події між сервісами та використовує координатор для керування потоком, використовуючи дані про запити і події. Запити (команди) – це дії, які ще потрібно виконати, а події – це те, що вже було зроблено. Координатор описує агентну (або сервісну) функціональність, необхідну для реалізації внутрішніх міжагентних або міжсервісних ролей і їх властивостей а також міжагентні або міжсервісні взаємодії сервісів D, E та F, які попередньо запрограмовані під команди, що генерують агенти координатора. У наведеному прикладі, сервіси A та C

починають виконуватися одночасно. Координатор споживає події з потоку подій і реагує на події, якими він піклується.

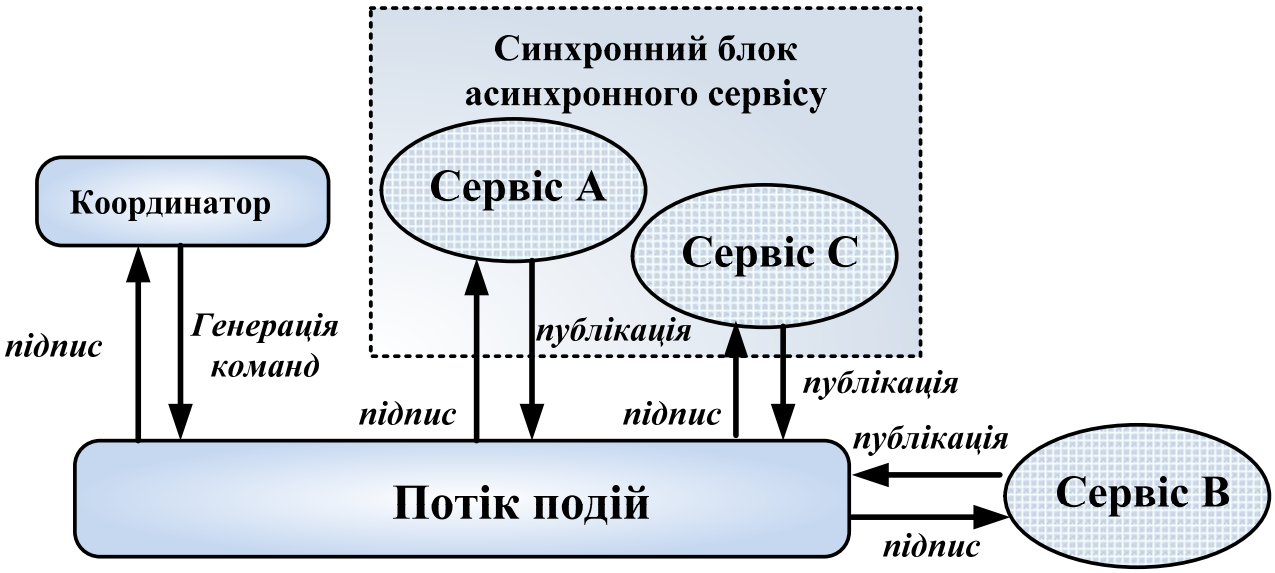


Рис. 3.13. Агентний шаблон взаємодії сервісів в MSA

Переваги і вади агентної архітектури приведені в табл.3.5.

Таблиця 3.5.

Характеристики агентної архітектури сервісів

Переваги	**Вади**
Сервіси ізольовані (але існує певна ступінь зчеплення між сервісами та координатором)	Координатор має зв'язок із сервісами, зокрема, йому необхідно знати, яка команда потрібна кожному сервісу для реагування
Асинхронна обробка здійснюється за рахунок використання подій між сервісами	Якщо координатор «впаде», то це може вплинути на всю подійно-керовану систему
Загальний потік можна побачити в одному місці: у координаторі. Вади	

Випадки доцільного використання агентної архітектури сервісів:

- Якщо є синхронні блоки асинхронної обробки.
- Якщо загальний потік може змінюватися залежно від даних, що оброблюються, і цей потік розрахований на кілька сотень мікросервісів.
- Якщо є необхідність бачити загальний потік від кінця до кінця під час проектування додатку та часу його виконання.
- Якщо є необхідність реалізувати якомога більшу декомпозицію, щоб уникнути залежності між сервісами.

Слід відмітити, що об’єднання агентного і сервісних підходів має серйозну перспективу. Якщо розглядати підприємство як скелет, то SOA (MSA) відіграє роль кісток, а агентні технології роль хрящів, які гарантують, щоб суглоби були підключені правильно, навіть якщо просто кістки можуть виглядати абсолютно несумісними, коли просто дивитися на них.

Таким чином, агент не може безпосередньо допомогти реалізувати основну функціональність сервісу, але він може суттєво сприяти, щоб ці функціональність стала дуже гнучкою, більш розумною, більш здатної до тісної співпраці та самоорганізації. Замінність, сумісність, і процес перевірки на відповідність завдання можуть бути виконані агентами на більш високому рівні, ніж просто перевірка опису інтерфейсів сервісів.

3.8. Об'єднана (гібридна) архітектура системи сервісів

З порівняльного розгляду синхронних SOA (MSA) і EDA очевидний тісний зв'язок між двома їх основними компонентами (подіями та сервісами). Взаємодія між синхронними SOA (MSA) і EDA при цьому, в основному, відбуваються на двох різних рівнях (рис. 3.10):

1. На першому рівні події з’єднують сервіси за допомогою передачі стану процесу від одного сервісу, який визначає і публікує події, до інших сервісів, які запускаються конкретними подіями.

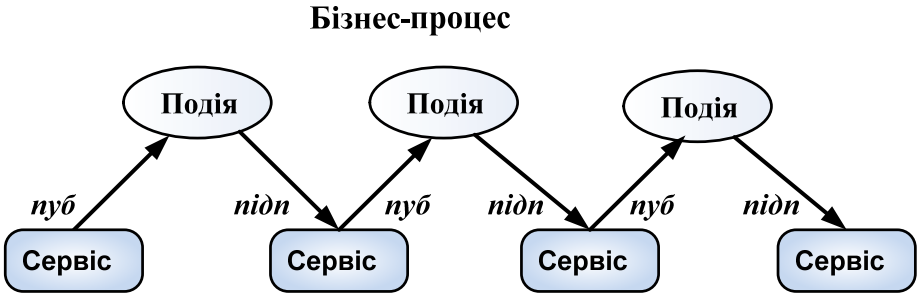


Рис. 3.10. Взаємодія SOA і EDA

2. На другому рівні сервіси об’єднують події за допомогою передачі даних про перехід одного стану процесу до іншого (споживач отримує подію і вживає необхідні заходи, які можуть призвести до виклику нових сервісів і формування нових подій).

Важливо також відзначити, що взаємодія між синхронними SOA (MSA) і EDA також може відбуватися на рівні інформаційної системи, а не тільки на рівні бізнес-процесів. В об’єднаних SOA і EDA сервіси більше не є тільки споживачем подій. Зовнішні / внутрішні інформаційні системи можуть також вживатися і виробляти події.

Об'єднання концепцій синхронних SOA (MSA) і EDA запропоновано називати ***сервіс-орієнтованою архітектурою, керованої подіями*** **(*EDSOA*)** [73,92], або ***гібридною архітектурою.*** Тільки формування бізнес-сервісів і бізнес-подій, а потім їх цільова ув'язка для рішень завдань бізнесу дозволяє домогтися стратегічних переваг для підприємства. При цьому EDA використовується для того, щоб «накинути» на всю програмну інфраструктуру організації «мережу» програмних датчиків і програмних агентів, а також апаратних датчиків, які стежать за подіями у всіх апаратних і програмних компонентах, відстежують значущі для бізнесу події і передають у центр прийняття рішень асоційовані з цими подіями сигнали. Це дозволяє технологічно управляти бізнесом не наосліп, а маючи чітку картину усього, що відбувається на даний момент на підприємстві.

Як відмічалося раніше, важливо при аналізі відносини між EDA і SOA враховувати рівень деталізації (зернистості) подій і сервісів. У добре визначеній

архітектурі рівень деталізації бізнес-подій збігається з зернистістю бізнес-сервісів. Хоча рівень деталізації обох сервісів та подій мають велике значення, поки не зовсім зрозуміло, як вибирати потрібній рівень деталізації на етапі визначення сервісів і подій. І навіть менш очевидно досі, як переконатися, що сервіси і події вибрані на одному рівні.

Обговорюючи SOA і EDA конвергенцію, можна вважати найбільш зручним рішенням цієї задачі використання ***сервісної шини підприємства EBS*** (Enterprise Service Bus), яка діє в якості проміжного шару (або посередника) для забезпечення комунікації та інтеграції великомасштабних гетерогенних прикладних процесів. ESB підтримує усі можливості як SOA, так і EDA, оскільки обслуговує як синхронні, так і асинхронні взаємодії між однією або багатьма зацікавленими сторонами. Це не стандарт, ні специфікація; скоріше, існує велика кількість відкритих та комерційних ESB, які розроблялися багатьма постачальниками і налаштовані відповідно до їх потреб (див. § 1.4).

Можлива також об'єднана (гібридна) мікросервісна архітектура, коли потрібна, наприклад, суміш синхронної та асинхронної обробки або синхронні блоки асинхронної діяльності, чи навпаки. Основним принципом дизайну, який слід враховувати при використанні гібридного підходу, є збереження основної взаємодії сервісів для асинхронної обробки. При оберненому підході (використанні шаблонів оркестровки між сервісами та асинхронних шаблонів всередині сервісу) обмежується загальний обсяг асинхронної обробки, яку можна виконати [97-100]. Гібридний шаблони взаємодії сервісів успішно реалізовано в системі для обміну повідомленнями *Apache Kafka* [97].

Можливий гібридний шаблон MSA використовує події між сервісами та оркестрування у межах сервісу. У цьому прикладі міросервіси A, B та C реагують через події один на одного (рис. 3.11). Мікросервіс A споживає подію, яка ініціює звернення до додаткових сервісів D, E і F. Ці додаткові виклики сервісів можуть бути асинхронними або синхронними. Сервіс A потім генерує подію за результатом цих трьох викликів сервісів.

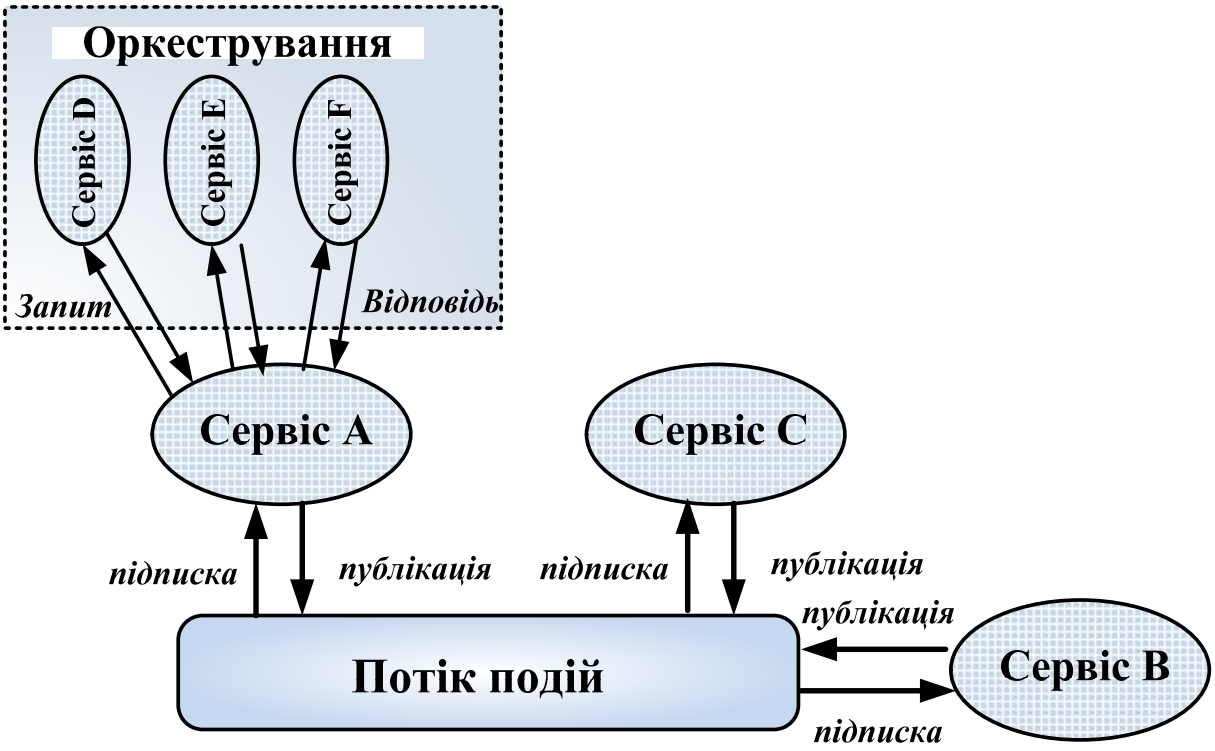


Рис. 3.11. Гібридний шаблон взаємодії сервісів в MSA

Переваги і вади об’єднаної архітектури приведені в табл.3.4.

Таблиця 3.4.

Характеристики об’єднаної архітектури сервісів

Переваги	**Вади**
Сервіси відокремлені (але не сервіси D, E і F у межах сервісу A)	В межах сервісу A існує зв'язок між сервісами D, E та F.
Асинхронна обробка здійснюється за рахунок використання подій між сервісами	Залежно від дизайну в межах сервісу A може бути синхронна обробка, яка блокує запити
Загальний потік подій розподіляється. Кожен сервіс містить свою власну логіку потоку	

Випадки доцільного використання об’єднаної архітектури сервісів:

- Якщо вся або більша частина необхідної обробки може бути виконана асинхронно.
- Якщо децентралізація потоку в кожному сервісі є керованою.
- Якщо швидкість появи на ринку є пріоритетом.
- Якщо є послідовні кроки, які застосовуються лише у межах сервісу, а не між сервісами.

Наприклад, використовується композитний сервіс, який складається з кількох атомарних процесів.

3.9. Традиційна SOA та контейнеризована інфраструктура

Більшість поточних реалізацій веб-сервісів побудовані за допомогою існуючої інфраструктури HTTP-серверів, значна частина з яких є простими *системами генерації сервлет коду* (за принципом «запит-відповідь»). Сьогодні більшість веб-сторінок має розмір 10-20 кбайт. Однак, коли веб-сервери розгортаються за всім підприємством, вони мають справу з повідомленнями розміром 10 або 100 мегабайт в обох напрямках і надають існуючі функції новим партнерам. Веб-сервісні платформи повинні підтримувати швидке безперебійне розгортання нових або покращених веб-сервісів. Крім того, найбільш надійні платформи підтримуватимуть декілька версій одного веб- сервісу.

Веб-сервіси SOA першого покоління вибираються із реєстрів за допомогою інструментарію, розглянутого вище. Це, мабуть, найвагоміша перевага від застосування WSDL і SOAP. Складові цих реєстрів накопичувались протягом кількох років, і IT-організації вручну наповнювали їх, що потребувало від них високого рівня кваліфікації і знань найсучасніших технологій. Зато тепер стало можливим виконувати успішні розробки та розгортання SOA додатків для багатьох інших IT-організацій.

Традиційна SOA та контейнеризована інфраструктура мають багато спільних цілей, але контейнери пропонують кращу ефективність та керованість. Як підприємства можуть об'єднати ці дві технології? Як зробити, щоб парадигма

SOA стала загальною інфраструктурою? Що контейнери додатків можть додати до сервіс-орієнтованої архітектури? Відповіді полягають у тому, що контейнери є будівельними блоками інфраструктури наступного покоління, а це спрощує як ніколи запуск нових програм у SOA та міграцію існуючих. Для цього необхідно створити прості у користуванні інструменти графічного інтерфейсу та достатню документацію, які полегшать успішне використання парадигми веб-сервісів, не вимагаючи глибоких знань про композиції сервісів та нові API.

Найбільш серйозними із існуючих реалізацій SOAP на ринку сьогодні є Java-базовані і побудовані із залученням усіх відповідних частин платформи Java. Це J2EE або, більш конкретно, сервлет-додатки. Тому ці реалізації SOAP представлені користувачам як додатки J2EE, а не як корпоративні платформи для розподілених обчислень. Один сервлет, або JSP, отримує вхідні запити SOAP. Запити аналізуються та надсилаються до логіки додатків, реалізованої у локальних класах Java, віддалених EJB або інших платформах. Інколи створюються унікальні сервлети для кожного веб-сервіса. Сервлет JSP і відповідна бібліотека виконання мають бути розгорнуті у середовищі хосту [168].

Як визначалося раніше, платформа веб-сервісів на рівні підприємства може базуватися на використанні контейнера веб-сервіса. Віртуальну машину Java можна розглядати як апаратну платформу. Сервери додатків працюють на цій платформі. Контейнер веб-сервісу аналогічний мережевому рівню, що полегшує спілкування між різними системами. Особливістю контейнерів веб- сервісів є підтримка багатьох середовищ для хостів. Контейнер веб-сервісів не є просто розширенням для J2EE, веб-сервіс не є спеціальним сервлетом.

Хости, зазвичай, включатимуть основні реалізації J2EE, але контейнери також мають працювати на корпоративних серверах інтеграції і брокерах, при цьому веб-сервісні контейнери можуть встановлюватися безпосередньо на запущені сервери додатків без зміни середовища хосту. До речи, платформа J2EE має кілька типів контейнерів: веб-контейнери для сервлетів і контейнери JSP, EJB для EJB, а також клієнтські контейнери для додатків на автономних терміналах, які використовують компоненти J2EE.

Розгорнуті веб-сервіси стануть найважливішими елементами IT- операцій, всепроникливими у корпоративних обчисленнях, такими, що знижують витрати на технічне обслуговування та зменшують час простою системи, що є критичним для успіху веб-сервісної технології. Якщо програма була розроблена для роботи з SOA, то її відносно легко перенести у контейнери. SOA-додатки вже написані як серія мікросервісів, а не як моноліт. *Для переходу в контейнери основними зусиллями буде зміна способу з'єднання різних мікросервісів, але не зміна характеру самих сервісів*.

3.9..1 Контейнер, який керує та контролює веб-сервіси

Вимоги до контейнеру веб-сервісів включають [169]:

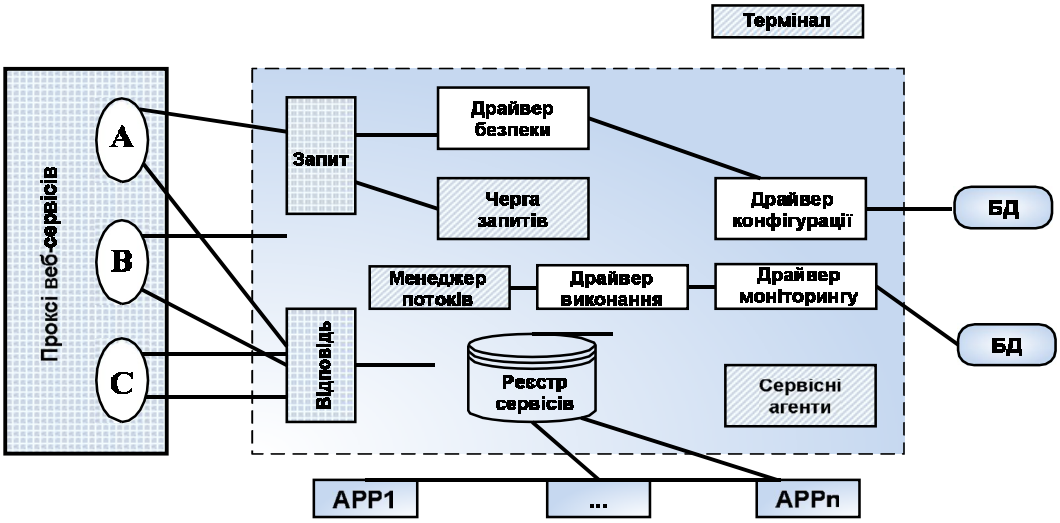
- Контроль доступності веб-сервісу.
- Контроль балансу завантаження веб-сервісів.
- Контроль і, якщо можливо, керування ефективністю веб-сервісу.

Як правило, веб-сервіси являються обгортками навколо існуючих додатків EAI (Enterprise Application Integration) або існуючих (legacy) систем. Концепція контейнера веб-сервісів – це прошарок абстракції, що містить усі корпоративні послуги, які виставляються як веб-сервіси. На рис. 3.14 показано можливу архітектуру типового контейнера веб-сервісів, основними компонентами якого є:

- драйвер конфігурації;
- драйвер безпеки;
- драйвер виконання сервісу;
- драйвер моніторингу сервісу.

Драйвер конфігурації допомагає налаштувати різні параметри, зв’язані з продуктивностю, реєстрацією, моніторингом та безпекою веб-сервісів. Досягнення високої доступності та балансування завантажень веб-сервісів є надзвичайно важливим для успіху будь-якої платформи керування веб- сервісами. Веб-сервіси повинні бути доступні для споживання лише авторизованими клієнтами, які надають захищені запити щодо сервісів. Це забезпечує однаковий

рівень безпеки доступу до сервісів завдяки використанню списків контролю



доступу (ACL) або цифрових підписів.

Рис. 3.14. Архітектура контейнера веб-сервісів

Драйвер безпеки допомагає налаштувати параметри безпеки для певного веб-сервісу. Сервісні запити вводяться у черги обслуговування. Менеджер потоку робіт виділяє потоки на основі конфігурації, вказаної для конкретного сервісу.

Драйвер моніторингу оновлює інформацію про наявність сервісу за допомогою сервісних агентів, які розташовані на вершинах усіх програм, що виставляються у вигляді веб-сервісів. Ця інформація зберігається у базі даних. Після того, як поступає запит на конкретний сервіс, драйвер виконання сервісу разом з драйвером моніторингу перевіряє його наявність у реєстрі веб-сервісів.

У випадку, якщо сервіс не обслуговується («падає»), користувач отримує повідомлення про його недоступність. Збільшення доступності досягається балансуванням навантаження веб-сервісів, коли запитаний сервіс може мати декілька екземплярів в реєстрі сервісів.

Слід зауважити, що якщо контейнер з розглянутою архітектурою фізично *розгорнути на одній з вітуальних машин*, то хмара, у якій розміщені такі контейнери, досягне своїх обмежень, коли ця машина задіє усі свої ресурси, навіть якщо інші машини будуть недовантажені. Тобто у кожній хмарі буде створюватися вузьке місце, якщо для сервісних додатків будуть

використовуватися контейнери, які *можуть обробляти велику кількість сервісів одночасно і відповідати на запити клієнта у прийнятному часі.* Тому краще контейнери розглянутого типу *розміщати на сервері*. Тоді у випадках, коли контейнер досягає своїх обмежень, можна почати розгортати сервіси в іншому контейнері, потім – у третьому і так надалі.

3.9.2. Мікро-контейнер, який керує та контролює один веб-сервіс

Якщо кожний сервіс буде розгорнутий окремо у мікро-контейнері, то хмару може реально масштабувати, доки вона не досягне реальних обмежень, які притаманні усім ресурсам хмари. Кожен сервіс середнього розміру (тобто складений із кількох сотен мікро-сервісів) може бути розгорнутий будь-де у хмарі з мінімальним використанням її ресурсів. Для цього доцільно створити спеціальний мікро-контейнер, який вміщує лише один веб-сервіс і забезпечує мінімальні функціональні можливості керування життєвим циклом розгорнутого веб-сервісу. Тоді можна *розгорнути чимало мікро-контейнерів на будь-якій віртуальній машині, а не тільки на будь-якому сервері.*

До списку основних компонентів, які входять до архітектури мікро-контейнера, можна віднести:

- Транспортний модуль для прийому запитів користувачів і для відправки відповідей.
- Модуль обробки SOAP і XML для аналізу SOAP-повідомлення та інтерпретування / створювання XML контенту.
- Процесний модуль для виклику та обробки розгорнутого веб-сервісу.
- Модуль розгортання веб-сервісу у мікро-контейнері.

Тоді архітектура мікро-контейнера приймає вигляд, приведений на рис. 3.16, і вміщує:

- Платформу розгортання для обробки вихідних архівів і WSDL описів і для генерації відповідних мікро-контейнерів та розгортання сервісів.

- Мікро-контейнер для розміщення веб-сервісу і оброблення клієнтських запитів.
- Тонкий клієнт для надання послуг користувачеві.

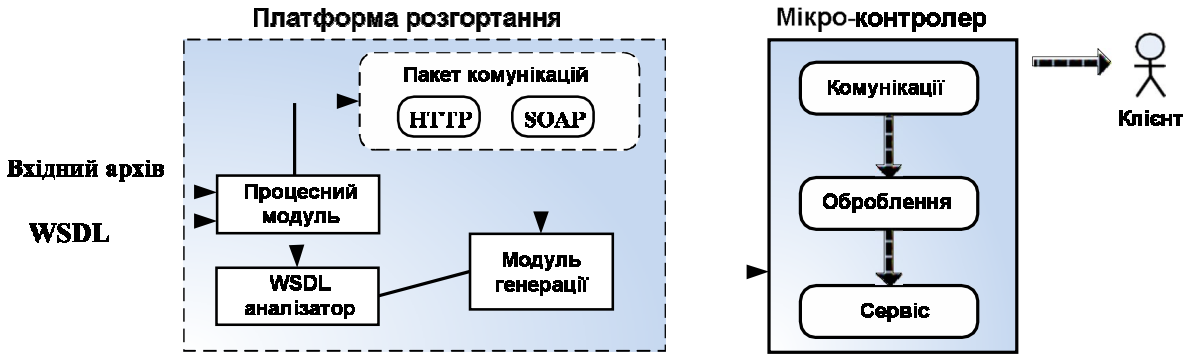


Рис. 3.15. Архітектура мікро-контейнера

Платформа розгортання відповідає за генерацію контейнерного компоненту. Зокрема, після аналізу сервісного WSDL-файлу *WSDL-аналізатором* і після отримання усіх вхідних архивів *модуль обробки* вибирає із загального *пакету комунікацій* тип зв'язку і переключає рішення на *модуль генерації* мікро-контейнера (рис. 3.15). Мікро-контейнерний компонент відповідає за управління спілкуванням з клієнтом, підтримку сервісу та оброблення усіх повідомлень, що надходять або виводять до мікро- контейнера. Він складається із наступних необхідних модулів для розгортання сервісу:

- комунікаційний модуль для встановлення зв'язку та підтримки протоколів з'єднання;
- модуль обробки вхідних та вихідних даних сервера (пакування та розпакування даних);
- сервісний модуль для зберігання та виклику запитуваного сервісу.

Функціонування системи (рис. 3.15) відбувається у чотири етапи: (1) отримання запиту клієнта, (2) вилучення HTTP SOAP обгортки, (3) виклику запитуваного веб-сервісу та (4) побудави відповідного повідомлення та відправлення його клієнту. Модуль генерації може підтримувати декілька протоколів комунікацій між мікро-контейнером та клієнтом.

Зрозуміло, що час відгуку (тобто час між отриманням контейнером запиту і відправленням відповіді клієнту) і споживання пам'яті (розмір пам'яті, необхідний для завантаження і оброблення розгорнутого у контейнері сервісу) будуть збільшуватися пропорційно кількості розгорнутих сервісів для випадку використання контейнера, зображеного на рис. 3.15.

Це час, необхідний для завантаження сервісу, оновлення індексації та контекстів і виконання сервісу. Час відгуку мікро-контейнера з розміщеним одним веб-сервісом, скоріше за все, залишатиметься завжди майже однаковим, тому що кожен екземпляр мікро-контейнера незалежний від інших, отже, можна розгорнути так багато мікро-контейнерів, як тільки це можливо згідно наявних ресурсів, не впливаючи на значення часу відгуку.

Споживання пам'яті у обох випадках збільшується з кількістю розміщених і виконаних сервісів. Але для мікро-контейнера слід очікувати певне заощадження пам’яті, оскільки за допомогою промислового контейнера, який використовується в якості контейнера веб-сервісів (рис. 4.10), може створюватися велика кількість додаткових файлів для кожного розгортаємого веб-сервіса (архіви, індексація, тимчасові файли, контекстні файли, інше). При цьому розмір цих файлів може бути більшим, ніж розмір самого мікро- контейнера, показаного на рис. 3.15.

3.10. Інструментарій розробки мікросервісних систем

До платформи розробки мікросервісних систем можна сформувати такі вимоги:

- Платформа повинна абстрагуватися від архітектурних паттернів сервіс-орієнтованих систеи і допомагати розробникам створювати код.
- Містити хороші моделі метаданих для опису можливостей мікросервісів.
- Підтримувати синхронні (загальнодоступні API) та асинхронні виклики (внутрішні події), обмін повідомленнями, включаючи підтримку протоколів, таких як черга, «публікація - підписка» та інші візерунки.
- Підтримувати декілька мов реалізації мікросервісів та еластичні режими роботи.

- Підтримувати збереження даних.
- Мати захищений шлюз API.
- Забезпечувати автоматизацію розгортання з підтримкою інструментів для контейнерів та їх оркестрування.
- Відображати відносини між мікросервісами для управління додатками з сотен чи тисяч мікросервісів, зокрема, оброблювати складні події (CEP).

Такі платформи зараз розробляються і їх прикладом може бути платформа корпоративного рівня *Macaw,* оптимізована для створення та перетворення MSA додатків, створена у другій половині 2017 року [98]. Macaw пропонує унікальний підхід до модернізації традиційних застосувань при використанні контейнерів та технології мікросервісів. Вона також пропонує підхід "під ключ" для проектування, розробки, створення, публікації, керування та експлуатації сервіс-орієнтованих додатків для широкомасштабних застосувань мікросервісів. Macaw призначений для підприємств, які прагнуть прискорити їхню модернізацію завдяки застосуванню гібридних хмар та мікросервісних додатків.

Платворма надає різноманітні вбудовані та легко використовувані сервери для створення додатків, такі як база даних, безпека, обмін повідомленнями, балансування навантаження для прискореного розвитку та операцій, а також проінтегрована з Kubernetes. Kubernetes застосовується для керування контейнерами та їх оркеструванням і здатний підтримувати різні контейнерні технології (Docker, CoreOS тощо) та хмарну інфраструктуру (публічні та приватні хмари).

Але для повного вирішення задач, пов'язаних із хмарними сервісними потребами підприємств, Macaw містить потрібний прошарок для розробки, запуску та виконання мікросервісів, упакованих у контейнери. Цей прошарок доглядає за потребами рівня застосування, тоді як Kubernetes забезпечує сервіси, необхідні для запуску та управління контейнерами.

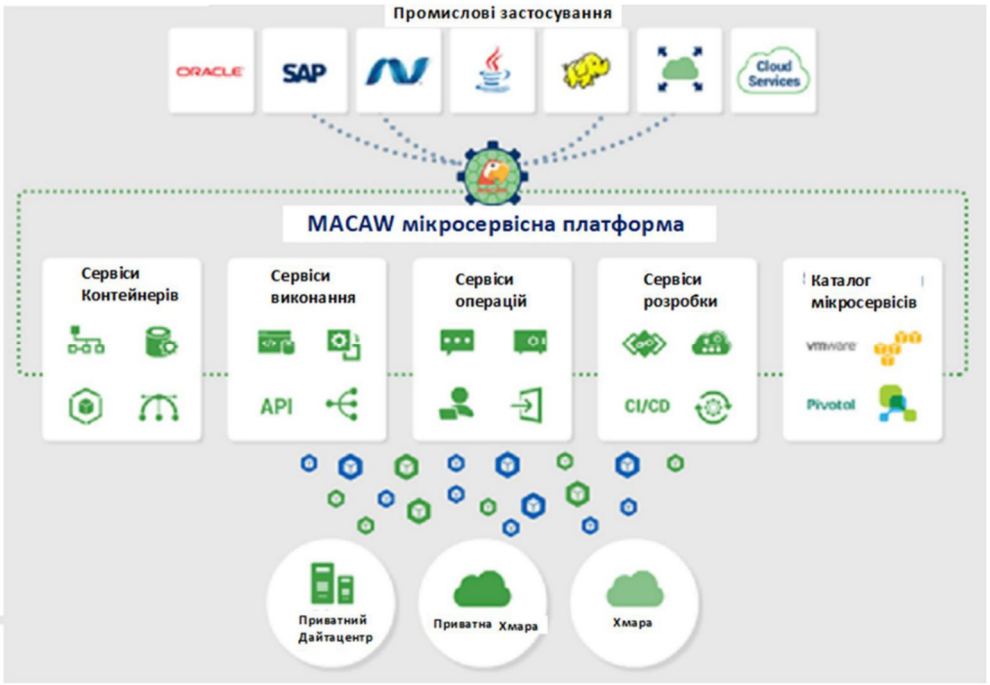


Рис.3.16. Структура платформи Macaw [98]

Висновки до розділу 3

1) Основна причина появи і поширення сервіс-орієнтованих додатків – це старозавітна мрія індустрії програмування про заміну «кустарного» кодування програм «від і до» на «промислову» збірку додатків із «стандартних комплектуючих», як в автомобільній, або інших «традиційних» галузях промисловості. Сервіс-орієнтована архітектура замість монолітної системи пропонує блокову із взаємодіючих компонент, в якій різні функціональні модулі додатків (сервіси чи мікросервіси), які розроблені множиною незалежних виробників, призначені для управління бізнес-процесами (Business Process Management, BPM), зв'язані між собою за допомогою чітко визначених інтерфейсів. Інтерфейси самі по собі незалежні від оточення і платформи, і тому така модель отримала назву моделі "слабкою зв'язку". При цьому створення, впровадження або зміна бізнес-процесу пов’язані з композицією (оркестровкою, хореографією) раніше розроблених сервісів, призначених для автоматизації бізнес-функцій.

2) Програмні додатки, що розроблені відповідно до сервіс-орієнтованих технологій, сьогодні реалізуються як набір сервісів (мікросервісів), інтегрованих

за допомогою відомих стандартних протоколів SOAP, REST та інших, як в SOA, чи контейнерної взаємодії сервісів через API Gateway, як в MSA. Головна відміна в реалізації SOA або MSA полягає в тому, що мікросервіси при застосуванні контейнерів не взаємодіють між собою безпосередньо, а тільки через API шлюз, бо кожен мікросервіс містить свою базу даних.

3) Всі відомі архітектури систем сервісів притаманні SOA і MSA (синхронна SOA; подійно-керована EDA; об'єднана сервіс-орієнтована, керована подіями EDSOA; мультиагентна MA), але в їх реалізації є суттєві відмінності. Наприклад, із-за відсутності безпосередньої взаємодії мікросервісів утруднена реалізація шаблону «запит-відповідь».

4) За традицією IT- додатки і процеси орієнтовані на використання передбачуваних і повторюваних подій. Коли ж відбуваються помітні події, які відрізняються винятковістю, то це створює складності для значної частини бізнесу. Це робить доцільним впровадження об’єднаної EDSOA архітектури як основної для побудови складних сучасних систем сервісів.

5) Дуже важливо правильно виділити частину бізнесу організації, якій буде користь від впровадження EDSOA. Цей крок вимагає формування структури підприємства, де буде визначена сфера використання задіяних сервісів і подій. Тільки формування бізнес-сервісів і бізнес-подій, а потім їх цільова ув'язка для рішень завдань бізнесу дозволяє домогтися стратегічних переваг для підприємства. Бізнес-сервіси і події повинні бути об'єднані таким чином, щоб забезпечити гнучку реконфігурацію процедур оброблення, можливо, шляхом використання декількох реакцій на подію.

6) Певною перевагою MSA додатків є усування труднощів, пов’язаних з використання складної і примхливої сервісної шини підприємства ESB, що організує і забезпечує взаємодію сервісів в SOA. Замість неї використовується гнучкий API інтерфейс з можливостями автоматичної конфігурації, само-оптимізації, самостійної адаптації та самозахисту функціональності.

7) Іншою перевагою складових MSA додатків є їх здатність до самостійного розгортання за допомогою повністю автоматизованого механізму

розгортання і взаємодії між собою через API Gateway. Вони можуть бути написані на різних мовах програмування і використовувати різні типи технологій зберігання даних. Для того, щоб збільшити доступність мікросервісу, можна використати кілька екземплярів примірника мікросервісу разом з балансувальником навантаження.

8) Певною вадою MSA додатків є значне зростання комунікаційного навантаження при їх виконанні. Кожен мікросервіс – це комбінація запитів і відповідей на запити, і якщо під час роботи частими є звертання до мікросервісів, то затримки, що накопичуються, можуть серйозно вплинути на час відгуку на запит користувача та його продуктивність. Ці затримки посилюються завдяки використанню для міжвзаємодії сервісів інструменту керування API.

9) MSA не є панацеєю – вона не вирішить усі IT-потреби підприємства, і тому повинна використовуватися разом з іншими SOA архітектурами у сучасному інформаційному середовищі підприємства. Більшість підприємств не зможуть перетворити усі IT-системи, що діють на підприємстві, на MSA додатки. Тому SOA додатки бажано застосовувати на рівні підприємства, де різні програми змушені взаємодіяти одна з одною через стандартні інтерфейси між додатками. А MSA додатки матимуть місце на рівні окремих застосувань, фокусуючись на їх структурі та компонентах.

10) Досліджені чотири типових архітектурних шаблонів MSA, сформульовані рекомендації щодо їх застосування та проведено порівняльний аналіз їх переваг і вад.. Це дозволило запропонувати критерії доцільного вибору архітектури MSA в залежності від призначення і функціональності MSA.

11) Зважаючи на стрімке зростання кількості мікросервісів у MSA реєстрах, процедура відкриття (пошуку) сервісів в них ускладняється. Тому представляється доцільним для автоматизації цієї процедури дослідити можливість переходу до семантичного опису мікросервісів у реєстрах на мовах WHDL чи OWL, як то зроблено в SOA. Крім того, слід визнати перспективними подальші дослідженні мультиагентних MSA.

РОЗДІЛ 4. ВИКОРИСТАННЯ СЕМАНТИКИ В СИСТЕМАХ СЕРВІСІВ

4.1. Веб-сервіси, їх описи та моделі

Пошук, або відкриття (discovery), сервісів є однією з найбільш невід'ємних властивостей сервіс-орієнтованих систем. Пошук полягає в ідентифікації сервісу з безлічі описів сервісів, що відповідають вимогам запиту на пошук. Зі збільшенням кількості сервісів, розроблених і запропонованих у системі сервісів підприємства або в Інтернеті, користувачі навряд чи в змозі вручну перевірити свої вимоги зі знаходження необхідних сервісів. Користувачам можуть допомогти автоматизовані методи відкриття сервісів. Здатність ефективно знаходити сервіси залежить від того, як сервіси описані і як вони рекламуються, як можуть бути сформовані вимоги до сервісів і як ці вимоги перевіряються. Останнє спонукає до розроблення методів пошуку сервісів, які можуть бути застосовані у різних бізнес-процесах, забезпечуючи при цьому приємний компроміс між виразністю і ефективністю.

Технології веб-сервісів на даний час є найбільш поширеним засобом реалізації сервіс-орієнтованих концепцій і призводять до становлення "Інтернету сервісів", де концентруються сервіси для всіх галузей життя і бізнесу. Сервіс є описом реалізації функціональності, що пропонується програмними компонентами. Оскільки сервіси слабко зв'язані і самодостатні, можна динамічно викликати і замінювати сервіси, наприклад, у бізнес-процесі, навіть через межі однієї компанії або організації. Зокрема, одним з конкретних застосувань є використання сервісів у робочих потоках (workflows), що підтримують бізнес-процеси.

Щоб знайти веб-сервіс, необхідно спочатку описати елементи запропонованих і запитуваних сервісів та їх можливості як можна більш точно. Для цього використовуються відомі технології веб-сервісів, такі як розширювана мова розмітки повідомлень XML (Extensible Markup Language), мова опису веб-сервісів WSDL (Web Service Description Language), протокол простого доступу до об'єктів SOAP (Simple Object Access Protocol). Пошук сервісів засновано на описі

сервісних компонентів, наприклад, інтерфейсів або операцій, але успішність застосування та використання веб-технологій у пошуку сервісів залежить від здатності постачальників сервісів описувати сервіси, здатності користувачів формувати запити і від порівнювача описів і вимог (matchmaker), який реалізує алгоритми, що враховують запит і знаходять найбільш підходящі сервіси з набору сервісів компанії. Порівнювач фокусується на виборі необхідних елементів з опису сервісу, показниках подібності (similarity metrics) і комбінації отриманих значень подібності.

Модель сервісу може бути представлена кортежем

$$C= \{T, M, P, O, B, S, E\},$$

який містить наступні елементи зазначених множин згідно опису сервісу на мові WSDL:

- ***Типи (Types):*** окремі типи даних, що описують повідомлення.
- ***Повідомлення (Messages):*** використовується як механізм комунікацій веб-сервісу.
- ***Типи портів (PortTypes):*** відображення логічного групування операцій, яке визначає веб-сервіси з точки зору операцій, що можуть бути виконані, і повідомлень, які залучені для їх виконання.
- ***Операції (Operations):*** використовується для виконання дій з передачі даних і дозволяють веб-сервісам інтерпретувати отримані дані та визначити, які дані будуть повернуті.
- **Зв'язування (*Binding*):** визначає деталі формату повідомлення та протоколу для операцій і повідомлень, визначених конкретним PortType.
- **Обслуговування (*Service*):** охоплює один або кілька типів портів PortTypes.
- **Кінцеві точки (*Endpoints*):** використовуються для показу набору операцій або PortTypes.

Базовою процедурою є ***відкриття сервісу***, що відповідає процесу знаходження відповідного сервісу за його описом у наборі сервісів в центральному або розподіленому репозитарії сервісів (наприклад, UDDI). Однак механізм пошуку UDDI підтримує лише прямі порівняння описів. У тих випадках, коли немає прямих посилань і необхідно скласти набір сервісів для виконання запиту, UDDI не забезпечує необхідних результатів пошуку, тому що він не може вийти за рамки прямих порівнянь описів сервісів на базі порівняння ключових слів. Складність відкриття веб-сервісу у цьому випадку підсилюється застосуванням розповсюдженого опису сервісу мовою WSDL, яка є існуючим базовим стандартом опису веб-сервісів. Мова WSDL базується на XML-основі і дозволяє описи сервісів, які розповсюджуються через Інтернет у вигляді набору кінцевих операцій з повідомленнями, що містять або документо-орієнтовану або процедурно-орієнтовану інформацію. Інші аспекти, як, наприклад, політика (правила), повинні бути включені в опис сервісу шляхом використання одного з численних WS-стандартів, приклади яких наведені нижче:

- *WS-ResourceLifetime* визначає механізм припинення існування WS-Resource (включаючи обмін повідомленнями), що дозволяє негайно або через вказаний час видалити ресурс.
- *WS-ResourceProperties* визначає, як WS-Resource може бути асоційований з інтерфейсом, що описує веб-сервіс (включаючи обмін повідомленнями) і дозволяє витягати, змінювати та знищувати властивості ресурсу WS-Resource.
- *WS-Notification* визначає механізм обробки повідомлень про події, що базується на принципах передплати / публікації.
- *WS-RenewableReferences* визначає механізм розширення звичайної системи адресації WS-Addressing, прийнятої у веб-сервісах.
- *WS-ServiceGroup* визначає інтерфейс до набору гетерогенних веб-сервісів.
- *WS-BaseFaults* визначає механізм обробки повідомлень про помилки.

Мові WSDL не вистачає семантики, яка необхідна для того, щоб комп'ютери зрозуміли призначення сервісу. Можна удосконалити існуючої технології WSDL і UDDI, надавши веб-сервісам комплексні семантичні анотації. Семантика до веб-сервісів додається відображенням описів сервісу, його операцій, входу і виходу із врахуванням концептів онтології предметної галузі. Структура відображення для цього прикладу приведена на рис. 4.1 [109].

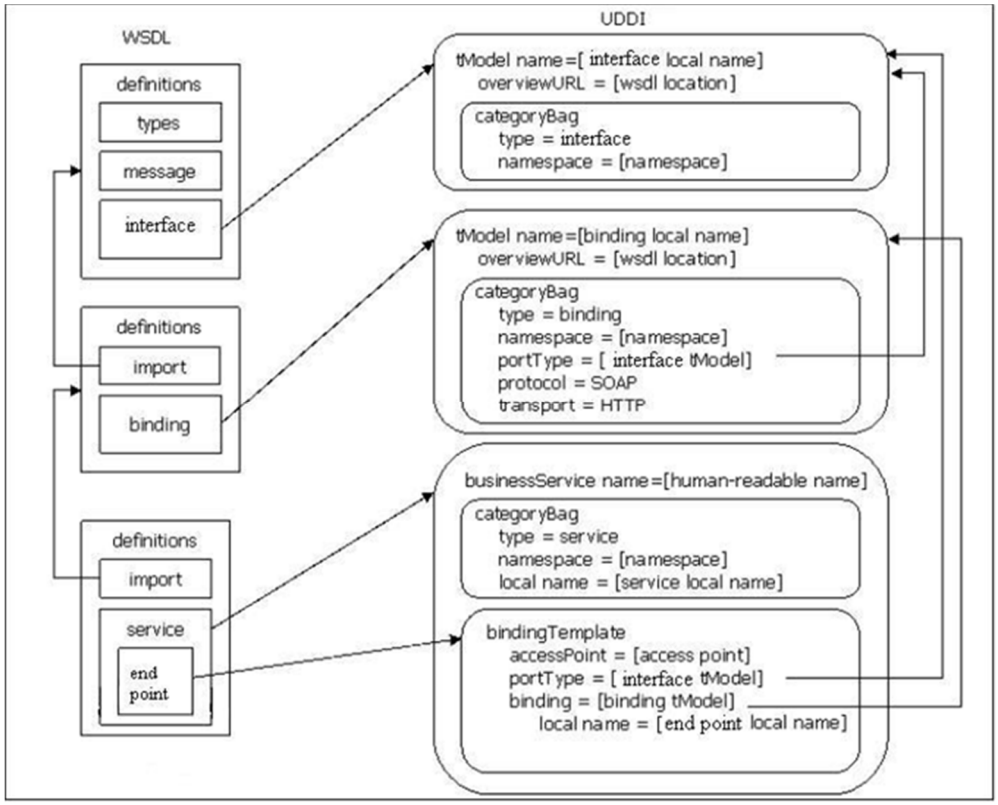


Рис. 4.1. Розширення структури UDDI для семантичних анотацій

Створені семантичні анотації зберігаються у структурах UDDI й існує інтерфейс, що дає можливість робити запити, використовуючи семантичні анотації. Чотири різні типи TModels створюються у UDDI реєстрі: один для представлення функціональної таксономії операцій у конкретній галузі, другий – для вхідних концептів, третій – для вихідних концептів і, нарешті, четвертий – для угруповання операцій з їх входами і виходами в *keyed Reference Groups*.

UDDI дозволяє шукати веб-сервіси за ключовими словами, категоріями, TModels і постачальниками. Також дає змогу шукати постачальників на основі ключового слова, категорії або *TModels*. Тому, коли використовується WSDL, то за допомогою UDD можна організувати пошук:

- сервісів, що описані у визначеннях WSDL;
- сервісів, що реалізують специфічну зв'язність (binding);
- сервісів, які реалізують специфічний інтерфейс;
- сервісів, які підтримують певні протоколи.

Треба пам’ятати, що UDDI, коли використовується відокремлено, реалізує лише метод ключових слів. Однак при використанні WSDL із семантичними анотаціями забезпечується можливість виявлення сервісу на основі більш розширених можливостей. Мовою семантичного анотування для WSDL і XML-схеми є мова SAWSDL *(Semantic*Annotationsfor WSDL and XML Schema), рис. 4.2.

```
<wsdl:types>
 <xs:schema targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#" elementFormDefault="qualified">

    <xs:element name="OrderRequest"
          sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderRequest"
          sawsdl:loweringSchemaMapping="http://www.w3.org/2002/ws/sawsdl/spec/mapping/RDFOnt2Request.xml">
          <xs:complexType>
              <xs:sequence>
              <xs:element name="customerNo" type="xs:integer" />
              <xs:element name="orderItem" type="item" minOccurs="1" maxOccurs="unbounded" />
             </xs:sequence>
         </xs:complexType>
   </xs:element>

<xs:complexType name="item">
     <xs:all> <xs:element name="UPC" type="xs:string" /> </xs:all>  <xs:attribute name="quantity" type="xs:integer" />
</xs:complexType>

<xs:element name="OrderResponse" type="confirmation" />
    <xs:simpleType name="confirmation"
          sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderConfirmation">
         <xs:restriction base="xs:string">
         <xs:enumeration value="Confirmed" />
         <xs:enumeration value="Pending" />
         <xs:enumeration value="Rejected" />
   </xs:restriction>
   </xs:simpleType>
</xs:schema>
</wsdl:types>
```

Рис. 4.2. Приклад семантичної анотації елементів WSDL у SAWSDL.

Існує також можливіть створення опису OWL-S для сервісів зі станом і без стану, для яких відомий відповідний опис WSDL [110]. Процедура перекладу досить складна і починається з розгляду конфігурації та URL-адреси документа WSDL. Транслятор аналізує документ WSDL, вилучає операції, типи портів, входи, виходи, а також властивості ресурсів. Потім транслятор створює для кожної WSDL операції скелет документа OWL-S, а також формує in-входи, виходи, передумови, ефекти та зіставляє ці елементи з онтологічними концепціями, викладеними у конфігурації. Ці концепції можуть описувати

інформацію про сервіс (наприклад ім'я сервісу, провайдера), а також складні входи і виходи сервісів, такі як географічне розташування, географічну інформацію, басейни річок та інше.

Переклад WSDL опису метеорологічного сервісу показано на рис. 4.3. Оскільки кожен сервіс містить декілька операцій, семантичний опис генерується для кожної операції, для того, щоб дати можливість створювати робочі потоки сервісних операцій.

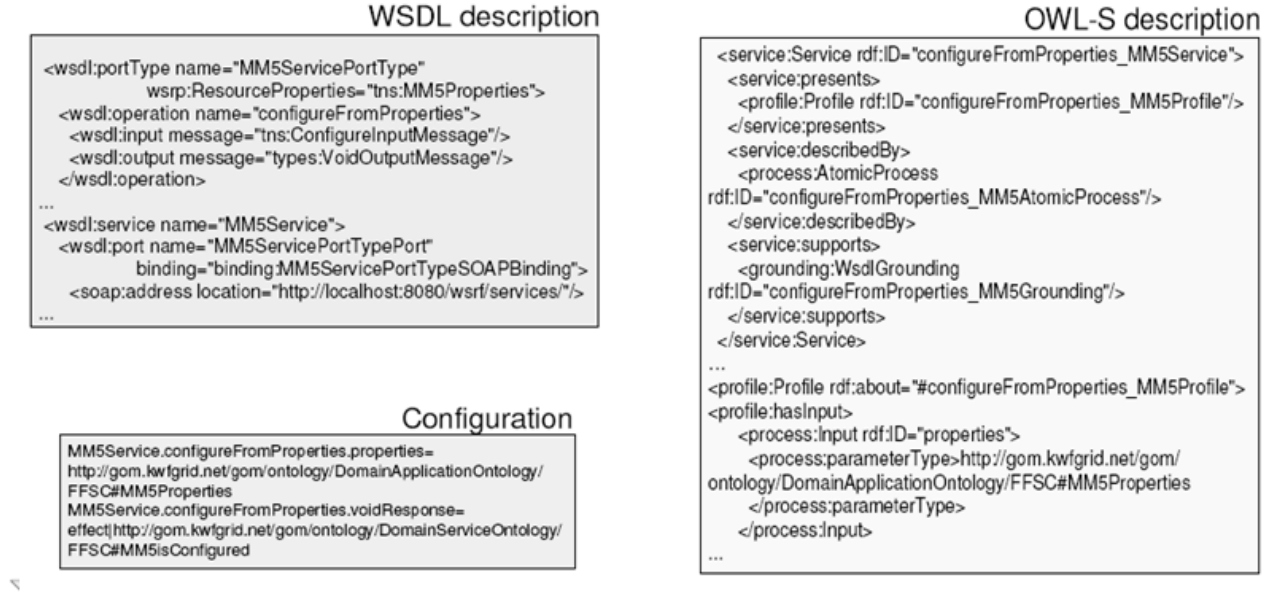


Рис. 4.3. Переклад WSDL опису метеорологічного сервісу

4.2. Семантичні сервіси, їх описи через онтології

Окрім введення в описи семантичних анотацій можна безпосередньо використовувати семантичну інформацію у веб-сервісах, внаслідок чого з’явилося поняття семантичних веб-сервісів (SWS) зі своєю мовою опису **OWL-S** (Web Ontology Language for Web Services) [114]. SWS створюються на базі існуючої технології веб-сервісів для забезпечення динамічного пошуку сервісів, їх композиції, виклику веб-сервісів, результатом чого є можливість автоматизації цих процедур, які в даний час вимагають втручання розробників програмного забезпечення. Але як це може бути досягнуто? Підхід до рішення задачі полягає у використанні більш точних описів семантичних веб-сервісів, які є зрозумілими для комп’ютера. Як правило, семантичний веб-концепт можна розділити на ***дані*** і

метадані [106]. У той час, як дані відображають фактичний контекст (зміст), метадані можуть бути визначені як "дані або інформація про дані". У цілому застосування метаданих має два аспекти: з одного боку, деталізується формат і організація представлення інформаційного змісту, а з іншого боку, – доменні знання зв'язуються з даними, які дають змогу будувати інтерфейси. Ключем до досягнення змісту, зрозумілому для комп’ютерів, є групування термінів, що використовуються у описах метаданих, на чітко визначені, стандартизовані словники. Для цього, використовуються ***онтології***, що мають стандартний формат.

Семантичні технології SWS сприяють інтеграції сервісів за допомогою семантичних описів сервісів і методів штучного інтелекту. З одного боку, можна стверджувати, що труднощі для оброблення сервісів і їх продуктивності виникають від складності семантичних мов, а також методів інтеграції, які залежать від логічних висновків (reasoning). З іншого боку, труднощі менеджменту опису сервісів (Service Descriptions) різко зростуть із зростанням їх складності, при цьому немає засобів автономного контролю. Логічні висновки можуть ефективно допомогти вирішити протиріччя у описі сервісів, а також підтримувати сумісність, коли ці описи змінюються. Таким чином, SWS, здається, відкривають шлях до побудови **автономної SOA**, до прозорого надання сервісів за прикладом, як надається *електричне живлення сьогодні*. Слід підкреслити, що на даний час сервіси складаються дизайнерами на етапі проектування сервісів. Автономні SOA можуть внести певну автоматизацію у цю процедуру. Хмарні обчислення є наступною ідею, реалізація якої залежить від успішного застосування методів розподіленого програмного забезпечення до створення повністю автономної SOA з розширеною інтеграцією сервісів та підтримкою сервісів як програмної парадигми.

Останніми роками розробка онтології – формального явного опису термінів предметної галузі і відносин між ними – перейшла зі світу лабораторій зі штучного інтелекту на робочі місця експертів з предметних галузей. У багатьох дисциплінах зараз розробляються стандартні онтології, які можуть

використовуватися експертами з предметних галузей для сумісного використання і анотування інформації у своїй галузі.

У самому загальному випадку під онтологією розуміють формальний і явний опис ***спільної концептуалізації домену***. Таке визначення підкреслює головні відмінні риси онтології: (1) необхідність певної мови для опису онтологій; (2) використання абстракцій галузі, які включають в себе відповідні поняття; (3) узгоджену точку зору на представлення основ знань домену. Тобто онтологію слід трактувати як структурну специфікацію деякої предметної галузі, її формалізоване уявлення, яке включає словник (або імена) покажчиків на терміни цієї галузі і логічні вирази, які описують, як вони співвідносяться один з одним. Такий словник підтримує обмін знаннями про деяку предметну галузь і безліч зв'язків, встановлених між термінами у цьому словнику.

Поняття і відносини в доменній онтології розроблені відповідно як екземпляри загальних понять "домен-поняття" і "домен-відношення". У центрі більшості онтологій знаходяться ***класи***. Конкретні процедури – це ***екземпляри (Instances)*** цього класу. ***Слоти*** описують властивості класів і екземплярів і можуть мати різні ***фацети***, які описують тип значення, дозволені значення, число значень (потужність) й інші властивості значень, які може приймати слот. На практиці розробка онтології включає:

- визначення класів у онтології;
- розташування класів на таксономічну ієрархію (підклас-надклас);
- визначення слотів і опис допустимих значень цих слотів;
- заповнення значень слотів для екземплярів.

Після цього ми можемо створити базу знань, визначивши окремі екземпляри введених класів, увівши у певний слот його значення і додаткові обмеження. При цьому слід дотримуватися наступних фундаментальних правил розробки онтології:

- Не існує єдиного правильного способу моделювання предметної галузі – завжди існують життєздатні альтернативи. Краще рішення майже завжди залежить від передбачуваного застосування і очікуваних розширень.
- Розробка онтології – це обов'язково ітеративний процес.
- Поняття в онтології повинні бути близькі до об'єктів (фізичних або логічних) і відносин у тій предметній галузі, для якої створюється онтологія. Швидше за все, це іменники (об'єкти) або дієслова (відносини) у пропозиціях, які описують вибрану предметну галузь.

Відображення сукупності понять предметної області і їх відносин в основному реалізується в сучасних онтологічних системах на основі моделі семантичної мережі фреймів. Вузли мережі представляють окремі поняття предметної області, дуги - відносини між поняттями. Окреме поняття в цій моделі представляється фреймом, слоти якого містять атрибути поняття. Похідні (дочірні) поняття успадковують атрибути базових (батьківських) понять. На етапі визначення понять онтології для їх атрибутів зазвичай задається ім'я і тип атрибута. Конкретні значення ці атрибути отримують при створенні на основі понять онтології примірників (об'єктів). Операції по створенню примірників понять підтримує більшість онтологічних систем. При цьому екземпляри частіше відповідають поняттям нижніх рівнів онтологічної ієрархії. Таким чином, онтологія є ієрархією понять, які характеризують предметний світ, об'єкти якого відповідають переважно поняттям нижніх рівнів онтології, а проміжний і верхній її рівні представляють, як правило, абстракції різного ступеня узагальнення.

Реалізація сервісу забезпечує його виконання шляхом перекладу абстрактних концепцій OWL-S профілю і процесу на конкретні повідомлення. Наприклад, на рис.4.4 приведений фрагмент медичної онтології, побудованої автоматично з допомоглю редакторі Protégé [106]. Основу фрагменту онтології складають п’ять взаємозв’язаних класів, що мають назви «прилади» (Device), «лікування» (Treatment), «дані» (Data), « життєво-важливі ознаки» (Vital Signs) і «симптони» (Symptoms), рис. 4.4. Класс «лікування», наприклад, використовується для визначення того, що робить лікар для пацієнта і містить

наступні екземпляри: терапію, ліки, опромінювання чи проводить хірургічну операцію. Клас «життєво-важливих ознак» описує, «результати яких вимірювань враховує лікар» і включає екземпляри: серцевий ритм, показники температури і кров'яного тиску.

Рис 4.4. Фрагмент медичної онтології

Клас «симптонів» з екземплярами (кашель, висока температура, головна біль, загрудина біль) відображує стан і самопочуття пацієнта. Екземпляр цього класу «температура» має три слоти (низька, висока, середня), властивості яких визначаються через співвідношення «is-a», відомого як відношення «батько-дитина».

Із поширенням семантичних технологій та їх упровадженням, зросла і кількість інструментів для менеджменту їх центрального елементу – онтології. Загалом, існує біля 100 розробок, серед яких найпопулярнішими є *Protégé 2000* /

Jena, PCPACK, Ontolingua, OntoStudio,Ontosaurus / PowerLoom, Apollo CH, CEO [106]. Ці інструменти дозволяють користувачам:

- Завантажувати і зберігати OWL і RDF онтології.
- Змінювати і візуалізувати класи, властивості та SWRL-правила.
- Визначати логічні характеристики класу як OWL вирази.
- Виконувати механізми виведення.
- Форматувати екземпляри класів для семантичного веб.

Семантика у веб-сервісах використовується для придання деяким компонентам сервісів певного значення, наприклад, забезпечення детальної інформації про те, яку операцію, інтерфейс або повідомлення містить опис сервісу. Можна виділити чотири різних видів семантики для веб-сервісів:

- ***семантика даних*** формально визначає дані у вхідних і вихідних повідомленнях веб-сервісів;
- ***функціональна семантика*** формально визначає можливості веб-сервісів, тобто, визначає передумови та наслідки і семантичні анотації інтерфейсів і операцій;
- ***нефункціональна семантика*** описує посилання на вимоги до якості QoS і загальні політичні вимоги / обмеження;
- ***семантика виконання*** описує виконання сервісів і операцій.

Основу онтології веб-сервісу складають три взаємозв'язані під-онтології, що мають назви «*профіль*» (ServicePorfile), «*модель сервісу*» (ServiceModel) та «*реалізація сервісу*» (ServiceGrounding), рис. 4.5. Онтологія профілю використовується для визначення того, «що робить сервіс» для потреб рекламування, побудови запитів, пошуку відповідників на запит. Модель сервісу описує, «як сервіс працює» для забезпечення активації, взаємодії, компонування, моніторингу, відновлення. Реалізація сервісу відображує компоненти моделі сервісу з врахуванням детальної специфікації форматів повідомлень, протоколів і т.і. (зазвичай, виражених за допомогою мови OWL-S).

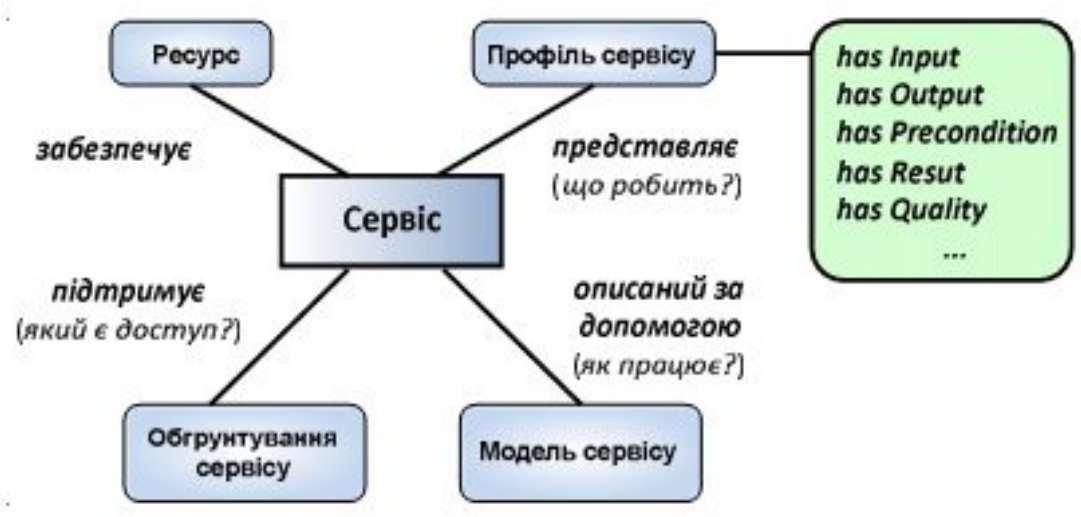


Рис. 4.5. Шаблон онтології семантичного веб-сервісу

Прив'язка відображує компоненти моделі процесу з врахуванням детальної специфікації форматів повідомлень, протоколів й іншого (зазвичай, виражених за допомогою мови OWL-S). Усі ці підонтології поєднані у концепт верхнього рівня «***Сервіс» (Service),*** що посилається на вищезгадані три складові (Service Porfile, Service Model, Service Grounding) відповідно через властивості *«представляє»* (presents), *«описаний»* (describedBy), *«підтримує»* (supports). Профіль сервісу забезпечує шляхи опису сервісів, які пропонуються їх постачальниками, та сервісів, яких потребують замовники. За допомогою ієрархії класів можливо створити спеціалізовані представлення про сервіси, які можна використати як профіль. Сервіс, визначений через профіль, моделюється як функція трьох базових типів інформації:

- Організація, що надає сервіс: контактна інформація, яка стосується суб'єкта, що надає сервіс (наприклад, оператор технічної підтримки та супроводу, відповідальний за роботу сервісу або представник замовника, який може надати додаткову інформація про сервіс, тощо).
- Функції, які виконує сервіс: вхідна інформація, яку потребує сервіс, та вихідні параметри, які він генерує; передумови, які вимагає сервіс, та очікуваний ефект від запуску сервісу;

- Характеристики хосту, що можуть визначати характеристики сервісу: категорія сервісу (наприклад, за класифікацією UNSPSC), оцінка якості сервісу (деякі сервіси можуть бути надійними, з низькою затримкою відповіді, інші – ні), список параметрів сервісу, які можуть містити будь-яку інформацію.

Найбільш важливим типом інформації, яку представляє профіль і яка відіграє ключову роль у процесі відкриття сервісу, є ***специфікація*** функціональності сервісу. OWL-S підкреслює два аспекти функціональності сервісу:

- перетворення інформації, що представлена входами та виходами сервісу,
- зміна станів сервісу під час виконання, яка представлена передумовами та очікуваним ефектом.

Таким чином, функціональні властивості описують входи, виходи, передумови й ефекти сервісу. Нефункціональні властивості описують напівструктуровану інформацію, призначену для людини-користувача, наприклад, ім'я сервісу і параметри сервісу, що включають вимоги до безпеки і якості сервісу, географічні дані й інше. У профілі OWL-S не надаються схеми для опису екземплярів входів, виходів, передумов та ефектів (тобто ***IOPE*** – inputs / outputs / preconditions / effect), однак, така схема існує в онтології процесу. Очікується, що IOPE, опубліковані у профілі, є підмножиною тих, що опубліковані у онтології процесу. Тобто очікується, що усі екземпляри IOPE будуть створені у цій другій частині опису, а профіль має просто посилатися на них. Властивості класу профілю, або його слоти, які онтологія OWL-S визначає для посилання на IOPE, можна підсумувати так:

- *hasParameter* (має параметр): діапазоном є екземпляр класу ***Parameter*** (параметр) у онтології процесу, основне завдання – визначити обрану область знань;
- *hasInput* (має вхід): діапазоном є екземпляри класу Input (вхід), що визначені у онтології процесу;

- *hasOutput*(має вихід): діапазон значень складають екземпляри класу Output (вихід), які визначено у онтології процесу;
- *hasPrecondition* (має передумову): вказує одну або кілька умов роботи сервісу, діапазон складає екземпляр класу Precondition (передумова), визначений за схемою у онтології процесу;
- *hasResult* (має результат): вказує один з результатів роботи сервісу, визначений класом Result (результат) у онтології процесу; він визначає умови генерації виходів та змін, що відбуваються під час виконання сервісу.

Оскільки профіль OWL-S лише описує загальну функцію сервісу, потрібна також детальна інформація про те, як взаємодіяти з сервісом. Взаємодію можна розглядати як процес, і OWL-S визначає підклас *ServiceModel* (модель сервісу) для того, щоб надати засоби для визначення процесів. Концепція процесу у розумінні OWL-S – це не обов'язково програма для виконання, а скоріше – специфікація способів, у які споживач може взаємодіяти з сервісом. Процес може генерувати та повертати деяку нову інформацію, яка базується на наданій йому інформації та стану його навколишнього середовища. Створення інформації описується входами та виходами, а зміни у навколишньому середовищі – передумовами та очікуваними ефектами процесу.

Неформально можна пояснити [107], що кожен процес має будь-яку кількість входів, представляючи інформацію, яка за певних умов вимагається для старту процесу. Процеси можуть мати будь-яку кількість виходів, за якими процес постачає інформацію замовнику на його запит. Входи та виходи представлені підкласами загального класу ***Parameter*** (кожен параметр має тип, вказаний за допомогою *URI (Uniform Resource Identifier).* Може існувати будь-яка кількість передумов, усі з яких мають бути дотримані для успішного запуску процесу. Процес також може мати будь-яку кількість ефектів. Виходи та ефекти можуть залежати від умов навколишнього світу під час виконання процесу. Передумови та ефекти представлені логічними формулами, OWL-S трактує їх як літерали (або текстовий рядок символів або ж XML-літеру, останній варіант

пропонується для мов типу SWRL, RDF, а перший – для мов типу KIF, PDDL). Процеси поєднуються з їх IOPE, використовуючи наступні властивості:

- *hasParticipant* (має учасника);
- *hasLocal* (має локальні змінні);
- *hasInput, hasOutput, hasPrecondition, hasResult*.

Процес включає, як мінімум, дві сторони: споживача, з точки зору якого описується процес, і сервіс, з яким споживач має справу. Як споживач, так і сервіс позначаються як учасники; вони прямо з'єднані з процесом за допомогою властивості *hasParticipant*. Входи та виходи, що описують перетворення інформації, приєднуються до процесу через властивості *hasInput* та *hasOutput* відповідно. Входи можуть походити або від споживача напряму, або з попередніх кроків того самого процесу. Результати та ефекти приєднуються до опису процесу не напряму, а через властивість *hasResult*.

Хоча вищезгадані властивості спільні для усіх процесів, які визначаються у OWL-S, розрізняють три типи процесів:

- *Atomic (атомарні*): описують сервіси, що очікують одного (можливо – складного) повідомлення і повертають одне (можливо – складне) повідомлення у відповідь.
- *Composite (складені*): процеси, що підтримують певний стан.
- *Simple (прості):* процеси, що використовуються як елементи абстракції або для представлення атомарних процесів, або для спрощеного представлення складених процесів.

Тобто, атомарні процеси подібні до дій, які виконує сервіс за один крок взаємодії, складені процеси відповідають діям, що потребують багатокрокової взаємодії, а прості процеси надають механізм абстрагування для уможливлення кількох представлень того ж самого процесу. Атомарний процес активується прямо, і не складається з жодних підпроцесів; його виконання складає лише один крок – він приймає повідомлення, щось робить та надсилає повідомлення у

відповідь. Не важко бачити, ***що атомарні процеси співпадають з мікросервісами.***

З іншого боку, складені процеси можуть складатися з інших (атомарних, складених, простих) процесів; ця декомпозиція може бути вказана з використанням конструкцій управління:

- *Sequence* (послідовність): набір процесів для послідовного виконання;
- *Split*(розгалуження): множина процесів, які виконуються паралельно; завершується тоді, коли усі з цих процесів будуть заплановані до виконання;
- *Split* + *Join* (розгалуження та злиття): складається з послідовного виконання набору процесів із синхронізацією; завершується тоді, коли усі ці процеси завершаться;
- *Choice* (вибір): виклик на виконання одної з даних конструкцій управління;
- *Any-Order* (у будь-якій послідовності): дозволяє виконання компонентів процесу у деякій невизначеній послідовності, але не паралельно;
- *If-Then-Else* (якщо-то-інакше): виконання обраних конструкцій за певних умов;
- *Iterate* (ітераційно): виконує процес кілька разів (без уточнення про кількість ітерацій та умови завершення);
- *Repeat-While* (повторювати за умови) та *Repeat-Until* (повторювати, доки умова не справдилась): ітераційне повторення залежно від умов виходу з циклу.

Знову не важко бачити, що перелічені ***конструкції управління визначають взаємодії між мікросервісами.*** Реалізація сервісу забезпечує його виконання шляхом перекладу абстрактних концепцій OWL-S профілю і процесу на конкретні повідомлення.

Відносини між семантичними поняттями зазвичай визначаються за правилами логічного виводу. Консорціумом W3C (World Wide Web Consortium)

розроблені два основних стандарти для представлення онтології, а саме ***RDF Schema*** (RDFS) і ***мова OWL*** (Web Ontology Language). RDFS і OWL засновані на ***RDF*** (Resource Description Framework), стандарту на основі XML, який дозволяє визначити ***трійки*** (триплети). Трійки визначають відношення між довільними *суб'єктом* і *об'єктом* за допомогою так званого *предикату*. В RDF і OWL суб'єкти, об'єкти і предикати визначаються через однакові ***ресурсні ідентифікатори URI*** або відповідно багатомовні ресурси ідентифікаторів (Iris). Таким чином, стандарт RDF забезпечує дуже універсальний і гнучкий механізм для представлення знань. Однією з форм URI є URL (Uniform Resource Locator), уніфікований покажчик ресурсу. URL – це адреса, що повідомляє вашому комп'ютеру, де можна знайти конкретний ресурс. Оскільки Веб дуже велика система для того, що одна яка-небудь організація керувала їм, уся безліч URI є децентралізованою. Жодна людина і жодна організація не стежить за тим, хто їх створює або як їх можна використовувати. Тоді як одні схеми URI (такі, як http:) залежать від централізованих систем (наприклад, систем DNS), інші схеми (такі, як freenet:) є повністю децентралізованими.

Оскільки стандарти веб-сервісів розробляються на основі мови XML, то вона також є одним з будівельних блоків Семантичного Веб. Це дозволяє з легкістю поєднувати веб-сервіси і технології Семантичного Веб для створення семантичних веб-сервісів (SWS), які визначені як веб-сервіси, властивості, можливості, інтерфейси яких кодуються в однозначній комп’ютерно-інтерпретованій формі. Починаючи з 2001 року, було проведено багато ініціативних розробок стандартів для SWS, серед яких найбільш відомі наступні: мова опису сервісів DARPA Agent Markup Language for Services (***DAML-S***), її наступник – мова опису онтологій веб-сервісів (***OWL-S***), мова моделювання онтологій веб-сервісів ***WSMO*** (Web Service Modeling Ontology), мови моделювання веб-сервісів ***WSML*** (Web Service Modeling Language) і семантичних веб-сервісів (***WSDL-S)*** [108].

Оскільки OWL-S в основному зосереджується на описі функціональних аспектів веб-сервісів, було запропоноване просте онтологічне *розширення OWL-S*

для додавання до веб- сервісу нефункціонального опису, що відноситься до QoS [110].

4.3. Процедури відкриття веб-сервісів

Базовою процедурою SOA і MSA є відкриття сервісу, що відповідає процесу знаходження відповідного сервісу у наборі сервісів, які можуть бути надані центральним або розподіленим репозитарієм сервісів. Після виявлення кількох відповідних сервісів можна вибрати найбільш підходящий сервіс і запустити або посилатися на нього.

Використання онтології веб-сервісу (у вигляді трьох взаємопов'язаних під-онтологій «профіля», «моделі сервісу» та «реалізації сервісу») полегшує процес його відкриття, оскільки процедура порівняння зводиться до категоризації взаємовідношень між поняттями під-онтологій. Однак розроблення і підтримка такої всеосяжної онтології веб-сервісу дуже ускладнена. Крім того, можуть бути різні онтології, розроблені різними провайдерами для одного і того самого домену.

Оцінкою близькості між описом сервісу (документом) і запитом є числове значення, яке виражає ступінь подібності між ними; оцінка близькості називається *оцінкою семантичної близькості*, якщо і тільки якщо вона визначена на основі семантики документів і запитів. Облік структури онтології і семантики відносин дозволяє обчислювати оцінки семантичної близькості між елементами онтології (поняттями, екземплярами, предикатами). Ці оцінки близькості називаються елементарними оцінками близькості, на основі яких визначаються близькості між триплетами. Оцінки близькості між триплетами потім використовуються для визначення близькості між метаописами.

Виходячи зі структури триплетів, для їх порівняння потрібні обчислення наступних видів елементарних оцінок близькості між: 1) поняттями; 2) поняттями і екземплярами; 3) екземплярами; 4) предикатами. Методи обчислення кожного типу елементарної оцінки близькості, що узагальнюють роботи [138-158], приведені в Додатку 2. Сервіс А вважається релевантним

заданому запиту R, якщо і тільки якщо оцінка семантичної близькості між ними перевищує деяку порогову величину.

Оскільки практична реалізація підходів, засованих на семантичної близькості між триплетами онтології, достатньо складна, для інженерної практики використовують різні оцінки близькісті між описами веб-сервісів і запитами при їх відкритті, які набагато простіші, але ефективні і до того ж враховують, що у більшості випадків описи веб-сервісів задані на WSDL з семантичними анотаціями. При цьому базуються на інформації *P, IOPE* і *QoS*, що визначається, зокрема, через шаблон онтології семантичного веб-сервісу (рис.4.5).

Якщо позначити сервіс запиту користувача через R в процесі відкриття сервісу A,то відповідну процедуру можна визначити формулою

$$Sim(R,A) = a_1 Sim_P(R,A) + a_2 Sim_{IOPE}(R,A) + a_3 Sim_Q(R,A) \qquad (1)$$

де $\sum_i a_i = 1,\ a_i$ – вагові коефіцієнти, $0 \le a_i \le 1, i = 1,2,3$.

4.3.1. Врахування контексту (загальних даних про сервіс і запит)

Контект у запиті на відкриття веб-сервісу формально визначається як будь-яка інформація, що може явно і неявно впливати на генерацію запита споживачем. Контекст ділиться на дві категорії: явний і неявний. Явний контекст прямо надається споживачем у процесі порівняння сервісів. Так перший член в формулі (1) визначає порівняння загальних даних про сервіси і включає співвідношення (2) на рис. 4.6.

Неявний контекст збирається будь-яким автоматичним чи напівавтоматичним засобом, використовуючи сенсори та інтелектуальну обробку даних (Data Mining). Контекст розділяють на чотири категорії залежно від того, як збирається контекст [111]. Це такі категорії: особистий профільований контекст (personal profile-oriented), предісторично-орієнтований (history- oriented), процесо-орієнтований (process-oriented) і інший (other).

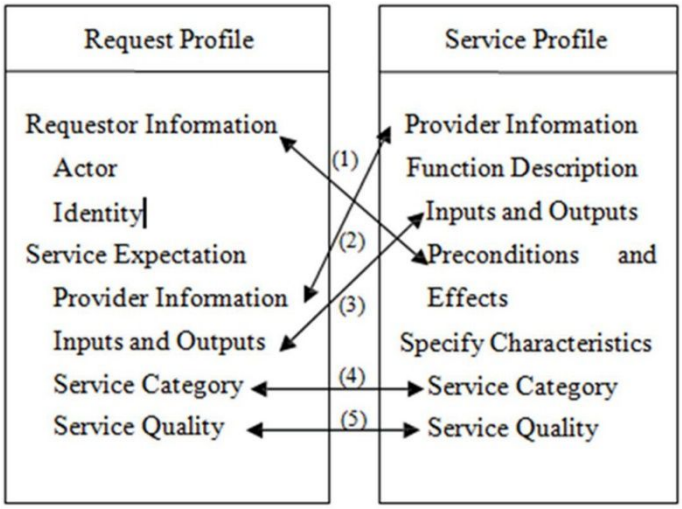


Рис. 4.6. Процес виявлення сервісу за запитом

Результат порівняння даних провайдера для сервісу A з даними запиту R користувача можна оцінити з допомогою вираза:

$$Sim_p = \sum_{i=1}^{\max(i,j)} \mu_i Sim_{P(i)}(R, A) \tag{2}$$

де i – індекс, пов'язанний з інформацією в запиті користувача (Request Profile), j - індекс, пов'язаний з інформацією провайдера сервісу (Service Profile), $\sum_i \mu_i = 1, 0 \le \mu_i \le 1, i = 1, 2, 3, \dots, n$.

Компонента $Sim_{P(i)}(R, A)$ в формулі (2) обчислюється при синтаксичному порівнянні інформаційних описів сервісу A і запиту R через співвідношення потужності |M1| множин M1 підборів певної кількості літерів в рядках даних про R і потужності множини |M2| таких підборів в рядках опису A, а також потужності |M3| множини M3 пересічення множин M1 та M2:

$$Sim_{P(i)}(R, A) = 2\frac{2|M3|}{(|M1| + |M2|)} \tag{3}$$

Слід зазначити, що існує альтернативна оцінка відповідності двох описів (документів), відома як косинусна оцінка, у вигляді:

$$Cos(w_i, w_k) = \frac{w_i^T w_k}{\sqrt{w_i^T w_i \sqrt{w_k^T w_k}}} \tag{4}$$

$w_i = (w_{i1}, w_{i2}, w_{i3}, \dots, w_{im}), w_k = (w_{k1}, w_{k2}, w_{k3}, \dots, w_{km})$ - вектори коефіцієнтів ваги для окремих слів рядків описів *R* і *A* (не штучно виділених підборів певної кількості літерів), компоненти яких визначаються як

$$w_{ij} = \frac{n_{ij} \times n}{n_j \times |d_i|},$$

де n_{ij}- частота входження терміна (слова) t_j в рядок d_i опису сервісу A; n_j – кількість рядків опису сервісу A, що містять термін t_j щонайменше один раз, $a|d_i|$ - визначає загальну кількістю слів в рядку d_i, при цьому m - кількість рядків в описі A. Аналогічно визначається компонента w_{kj} вектора коефіцієнтів ваги w_k для опису *R* [117, 120].

Якщо виявлена відповідність даних, то веб-сервіс вибирають для участі у наступному етапі процедури відкриття.

4.3.2. Врахування функціональності сервісу і запиту

Ступень відповідності функціональності сервісів підраховується другим членом формули (1) при визначенні відповідності функціональностей сервісів. Атрибути – *hasInput, hasOutput, hasPrecondition і hasEffects* зіставляються з тими, що містяться у запрошеному користувачаем сервісі. Ці атрибути включає співвідношення (1), (3) і (4) на рис. 4.6.

Почнемо з спрощеного алгоритму оцінки відповідності сервісів тільки за *IO* (вхід-вихід) порівнянням [138]. Якщо *OutR* представляє вміст (концепт) виходу замовленого сервісу, і *OutA* – концепт виходу наявного сервісу з репозитарію, то можливі чотири ступені співпадіння (збігу), визнані цим алгоритмом:

- *Точний збіг (exact)*: якщо *OutR* і *OutA* такі ж самі або *OutR* є безпосереднім підкласом *OutA*. Це найкращий рівень порівняння: $R = A$, $Sim(R, A) = 1$.
- *Підключення (plug-in)*: якщо *OutA* включає *OutR*, то *OutA* може бути підключений замість *OutR*. Це наступний кращий рівень порівняння

$Sim(R, A) = 1$, оскільки повернутий вихід може бути використаний замість того, що очікується згідно запиту $R < A$.

- *Неясність (subsume)*: якщо *OutR* включає *OutA*, але можливості провайдера щодо спроможності повністю задовольнити відповідним вимогам запиту чітко не визначені $R < A$. В цьому разі

$$Sim(R, A) = np(X \cap Y)/np(X \cup Y) \quad (6)$$

де $np(X \cap Y)$ - кількість ознак (number of propoties), однокових для виходів сервісів R і A, $np(X \cup Y)$ - кількість ознак, різних для виходів сервісів R і A.

- *Невдача (fail)*: якщо немає співвідношення між *OutA* і *OutR*, $(R \neq A)$, $Sim(R, A) = 0$.

Порівнювач онтологій попередньо обчислює співвідношення між концептами, що представляють входи і виходи усіх наявних сервісів у базі даних онтологій при їх реєстрації. Коли приходить запит, відповідний механізм (драйвер) просто перевіряє зв'язки між концептами виходів/входів наявних сервісів і запитаного сервісу. Найгірший рівень між рівнями збігу між виходами/входами обирається як глобальний рівень порівняння між виходами/входами замовленого і наявного сервісів. Наявні у репозитарії сервіси упорядковуються за найбільшою кількістю балів у виходах. Входи порівнюються тільки тоді, коли існують однакові бали для виходів.

В більш загальному випадку використання всього *IOPE* для порівняння сервісів (входів,виходів, передумов і ефектів) застосовується формула:

$$Sim_{IOPE}(R, A) = \beta_1 Sim_{Input}(R, A) + \beta_2 Sim_{Output}(R, A) + \beta_3 Sim_{Precond}(R, A) + \beta_4 Sim_{Effects}(R, A), \sum \beta_i = 1, 0 \leq \beta_i \leq 1, i = 1, 2, 3, 4. \quad (7)$$

Оцінка відповідності здійснюється з допомогою виразу [117,120]

(8)

$$Sim_{IOPE} = \sum_{k=1}^{\max(i,j)} \beta_1 Sim_{IOPE(i)}(R, A)$$

де, як і раніше, i та j – індекси, пов'язані з інформацією в запиті користувача (Request Profile) та з інформацією провайдера про сервіс (Service Profile),

$$\begin{aligned} &Sim_{Input(i)}(R, A) = \{Exact, Plug\ in, Subsume, Fail\}, \\ &Sim_{Output(i)}(R, A) = \{Exact, Plug\ in, Subsume, Fail\}, \\ &Sim_{Precond(i)}(R, A) = \{Exact, Plug\ in, Subsume, Fail\}, \\ &Sim_{Effects(i)}(R, A) = \{Exact, Plug\ in, Subsume, Fail\}, \end{aligned} \quad (9)$$

4.3.3. Врахування вимог до якості сервісів QoS

Існує багато публікацій ***з*** *розширеного порівняння*, коли при порівненні опису сервісу і запиту використовується комбінація функціональних і нефункціональних вимог. Третій член формули (1) обчислює ступень відповідності якостей сервісів (QoS), йому відповідає співвідношення (5) на рис.4.6.

До атрибутів QoS, перш за все, відносять вартість обслуговування, потужність, цілісність, продуктивність, надійність, стійкість, масштабованість і безпеку, при цьому:

- *Вартість послуг* CS (Cost of Service) є вартістю запиту на сервіс або його виклик.
- *Ємність* C (Capacity) є межею одночасних запитів з гарантованим виконанням.
- *Цілісність* I (Integrity) є мірою здатності сервісу до запобігання несанкціонованому доступу і зберігання його цілісності даних.
- *Продуктивність* P (Performance) визначається пропускною здатністю, часом відгуку і затримкою.
- *Пропускна здатність T (Throughput) є кількість запитів, що обслуговуються за певний період часу.*

- *Час відгуку RT* (Response Time) визначається як затримка від запиту на отримання відповіді від сервісу.
- *Затримка* L (Latency) є часом між запитом клієнта і початком відповіді йому.
- *Надійність* R (Reliability) є гарантією того, що сервіс не зламається протягом певного періоду часу, доки користувач (людина або інший сервіс) не розпізнає відмову у роботі.
- *Відмовноздатність* РБ (Robustness) є стійкість до якості вхідних даних і неправильних послідовностей при запиті сервісу.
- *Масштабованість* S (Scalability) визначає, чи може ємність C бути збільшеною у випадку потреби.
- *Безпека* ST (Security) включає наявність і тип механізмів аутентифікації, які сервіс пропонує конфіденційно; цілісність даних повідомлень, якими сервіси обмінюються; невідмова від запитів або повідомлень; стійкість до нападу на сервіси. ST включає наступні поняття аутентифікації і шифрування.
- *Аутентифікація* A (Authentication) є сервісом, який або вимагає аутентифікації користувача, або дає дозвіл анонімним користувачам.
- *Шифрування* E (Encryption) визначає тип і міцність технології шифрування, яка використовується для зберігання та обміну повідомленнями.

Вважається, що провайдер сервісів забезпечує значення цих параметрів у сертифікатах QoS, які при відкритті сервісу порівнюються зі значеннями, які містяться у запиті [115]. При цьому оцінки відповідностей $Sim_Q(R, A)$ виконуються за формулами, аналогічними формулі (6). Основна відмінність полягає в тому, що відхилення нефункциональних вимог, що мають розмірність, нормуються. (10)

$$Sim_Q = \sum_{k=1}^{\max(i,j)} b_k Sim_{Q(k)}(R, A)$$

де

$$Sim_{Q(k)} = \sqrt{Sim_{min} Sim_{avg} Sim_{max}}; \quad (11)$$

$$Sim_{min} = \{1 - abs[Q_R(k)_{min} - Q_A(k)_{min}]\}/Q_A(k)_{min}; \quad (12)$$

$$Sim_{avg} = \{1 - abs[Q_R(k)_{avg} - Q_A(k)_{avg}]\}/Q_A(k)_{avg}; \quad (13)$$

$$Sim_{max} = \{1 - abs[Q_R(k)_{max} - Q_A(k)_{max}]\}/Q_A(k)_{max}; \quad (14)$$

Двома основними елементами у запропонованій процедурі є *перевірка* і *сертифікація*. Провайдер сервісу відповідає за реєстрацію, оновлення та видалення інформації про веб-сервіси у UDDI із значеннями QoS, зв'язаними з бізнесом та продуктивністю веб-сервісів. Перевірка та сертифікація цих значень QoS виконується при пошуку веб-сервісу. QoS-перевірка – це процес підтвердження інформації, описаної в інтерфейсі сервісу. Процес сертифікації використовує цей результат перевірки. При позитивних результатах порівняння значень QoS далі розглядається і порівнюється функціональність видідених сервісів, наприклад, за приведеною в параграфі 4.4.2. процедурою [119].

4.4. Гібридний метод пошуку веб-сервісів

Відкриття веб-сервісу в розглянутих раніше методах складається з трьох процесів: *порівняння, відбору* та *аранжування*. Можна додати ще один, пов'язаний з визначенням категорій (category-based search) сервісів в UDDI чи репозитарії до семантичного пошуку неохідного сервісу, базуючись на співвідношенні (4) на рис. 4.6. Він виконує категоризацію і повертає всі ті сервіси, які підпадають під зазначений набір категорій вибраної таксономії. Тобто пошук веб-сервісів виконується в в два етапи (рис.4.7):

- *на першому етапі* за допомогою синтаксичного порівняння сервісів звужується область пошуку шляхом відсіювання непричетних до запиту сервісів або виділенням одного з кластерів, сформованих застосуванням кластерного методу;
- *на другому етапі* проводиться уточнення попереднього вибору застосуванням семантичного порівняння описів відібраних сервісів із

запитом, для чого складаються семантичні анотації для веб-сервісів або описи онтологій для семантичних веб-сервісів.

Категоризація веб-сервісів полягає у виділенні серед них подібних груп (кластерів), що може значно зменшити простір пошуку для відкриття сервісу і тим самим поліпшити продуктивність пошуку.

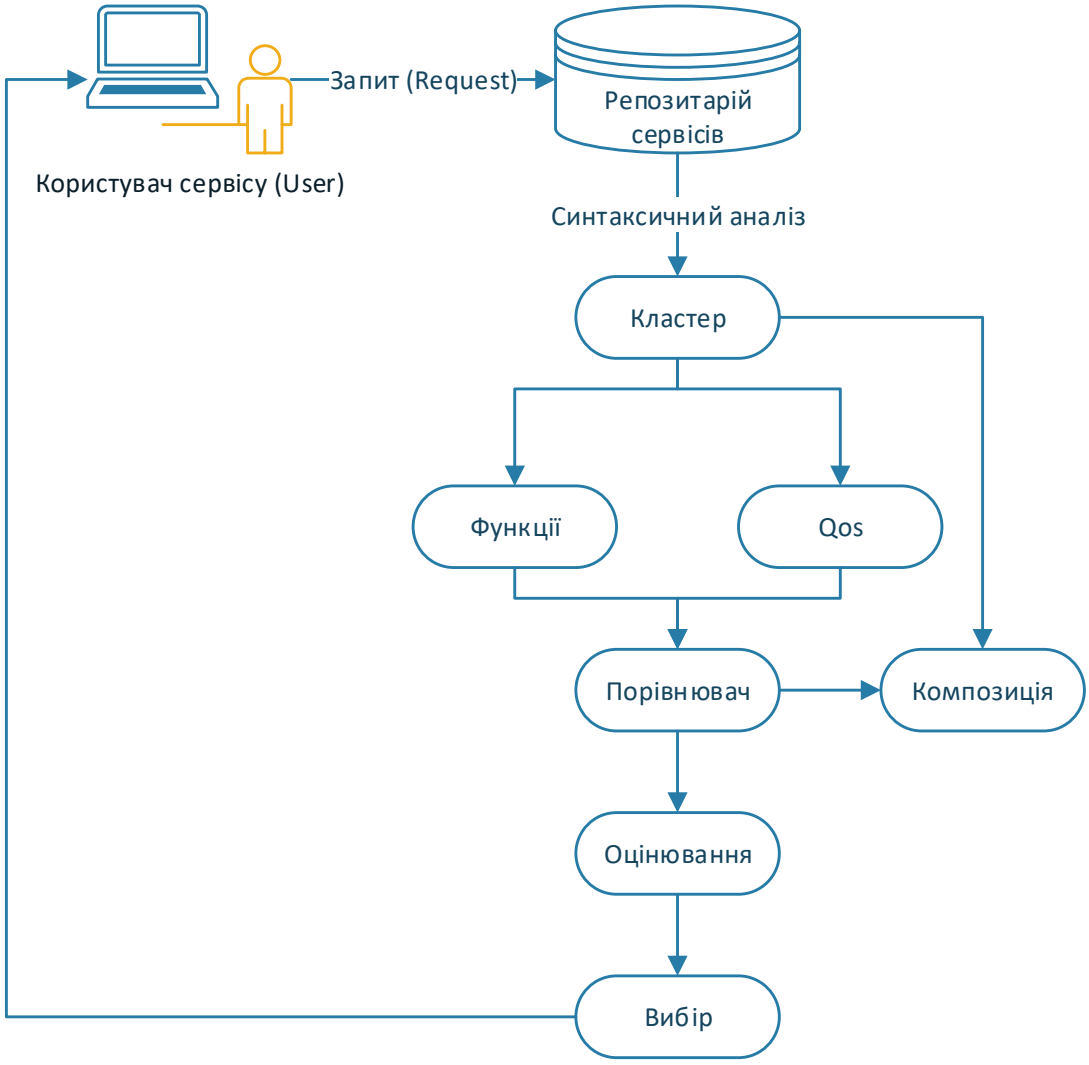


Рис.4.7. Структура гібридного методу пошуку веб-сервісів

Кластеризація веб-сервісів дозволяє споживачу шукати необхідні сервіси згідно його вимог, виключаючи розгляд потенційних сервісів-кандидатів за межами відповідного кластера і тим самим обмежуючи простір пошуку тільки цим кластером. Кластерний підхід спочатку організовує усі наявні доступні опубліковані сервіси на групи (кластери) із подібних сервісів. Кожен кластер зв'язаний з центром кластера, який є представником усіх сервісів у цьому

кластері. Після того, як кластери утворені, при подачі запиту запускається процедура порівняння, яка полягає в обчисленні відстані між запитом і центром кожного кластеру. Потім буде достатньо провести повне семантичне порівняння властивостей сервісів тільки для сервісів обраного кластеру, використовуючи їх функціональні атрибути (такі як вхід, вихід, попередумови і ефекти), а також вимоги до нефункціональних атрибутів (контекстної інформацією і якості обслуговування). Оскільки простір пошуку скорочується, то час, необхідний для знаходження відповідного сервісу для даного запиту, буде дорівнює часу, який витрачається на *знаходження підходящого кластеру*, який відповідає категорії запиту, і часу, який займає порівняння на *відповідність запиту і усіх членів кластеру*. Останнє потребує застосування процедур обчислення подібності оцінювання схожості сервісів, розглянутих в параграфі 4.4, коли підраховується подібність атрибутів сервісів, а потім подібність самих сервісів розраховується як сукупність індивідуальних подібностей атрибутів об'єктів.

Зазвичай кластеризація сервісів виконується синтаксисним методом оцінки мовної схожості текстових описів категорій сервісу і запиту. Пропонується для цього застосувати *латентно – семантичний аналіз (LSA),* що аналізує взаємозв'язок між колекцією документів і термінами, що в них зустрічаються [112, 113]. При цьому при відборі сервісів враховується не тільки наявність в їх анотаціях слів із запиту, але й смислова близькість до запиту (яка вимірюється за допомогою методу LSA). Латентно – семантичний аналіз відображає сервіси і запити у так званий «семантичний простір», в якому і проводяться всі наступні порівняння. Спочатку необхідно скласти частотну матрицю індексованих слів. У цій матриці рядки відповідають індексованим словам, а стовпці – описам сервісів. У кожному осередку матриці вказано, скільки разів слово зустрічається у відповідному описі сервісу.

Далі проводиться сингулярне розкладання отриманої матриці. Сингулярне розкладання – це математична операція, за допомогою якої матрицю розкладають на три складові: $M = U \cdot W \cdot V^t$, де U і V^t - ортогональні матриці, а W діагональна матриця, з яких отримаємо координати сервісів і запитів у семантичному

просторі. На останньому етапі порівнюються між собою координати сервісів і/або запитів: ті, які з них знаходяться ближче один до одного, утворюють кластери, а ті, відстань між якими значно більша, відповідно менш релевантні.

Є ще одна особливість запропонованого гібридного методу пошуку сервісів. Порівняння є ключовим етапом у відкритті веб-сервісу, воно зазвичай полягає в здійсненні двох *послідовних процесів*: порівняння функціональних властивостей сервісів і порівнянні нефункціональних їх властивостей [116, 129-134]. Потім відібрані сервіси фільтрується в процесі відбору та аранжування для отримання бажаного сервісу.

Особливості мікросервісної архітектури з базовою подійно- орієнтованою організацією обчислень дозволяють проводити порівняння обох функціональних і нефункціональних властtвостей сервісів *паралельно* і тим самим враховувати в асинхронному режимі нефункціональні атрибути запиту споживача.

Якщо час передачі функціональних властивостей конкретного запиту дорівнює часу передачі нефункціональних властивостей цього запиту, то дос порівнювач складає ці два типи атрибутів в одну композицію; якщо ні, то функціональні властивості конкретного запиту чекають на отримання нефункціональних властивостей. Складані функціональні та нефункціональні властивості порівнюються з описами веб-сервісів, відобраними в кластер, репозиторії, в результаті чого відкривається бажаний сервіс. Якщо один веб-сервіс не може задовольнити функціональність запиту, яка вимагається споживачем, то з’являється необхідність об'єднувати наявні сервіси кластепа разом.

4.5. Оцінка точності відкриття веб-сервісу

Для оцінки ефективності функціонування інформаційних пошукових систем зазвичай застосовуються поняття *точності, вибору* та *оцінки F-міри.*

Нехай, Qi – кількість відкритих сервісів, що мають відношення до запиту, NRi – кількість відкритих сервісів, що не мають відношення до запиту, тоді маємо

$Ri = Qi + NRi$ – загальна кількість отриманих сервісів. Тоді математично введені оцінки визначаються таким чином:

$$\text{Точність} = \mathrm{Q_I}/\mathrm{R_I},$$

$$\text{Вибір} = \mathrm{R_I}/\mathrm{N_I},$$

$$\mathrm{F} = (2 \cdot \text{Точність} \cdot \text{Вибір})/(\text{Точність} + \text{Вибір}).$$

Для демонстації корисності урахування нефункціональних вимог наведемо порівняльні результати пошуку і відкриття з реєстру UDDI п’яти популярних сервісів: *Готель (Hotel)*, *Оренда авто (Car Brokerage)*, *Авіарейс* ***(Airline)****, Турагенство* ***(Travel Agent)*** *і Лікарня (****Hospital****)*. Для сервісів використовувалася наступна інформація: ім'я сервісу, URL і WSDL опис сервісу. Пошук виконувався методом порівняння ключових слів для WSDL опису і для WSDL опису з семантичною анотацією, яка врахувала лише контекстну інформацію про дату, час і місце генерації запиту [116].

Результати покращення точності пошуку сервісу завдяки урахуванню нефункціональних вимог

Таблиця 4.1

Сервіси	***WSDL опис***			***WSDL опис з анотацією***		
	Ri	***Qi***	***Точн.***	***Ri***	***Qi***	***Точн.***
Готель	9	6	0.66	3	2	0.67
Оренда авто	19	12	0.63	5	4	0.80
Авіарейси	18	11	0.61	4	3	0.75
Турагенство	16	9	0.5	5	4	0.8
Лікарня	22	9	0.4	9	6	0.67

Як свідчать дані таблиці 4.1, точність відкриття сервісу за умови урахування контексної інформації значно підвищується. Результати можно ще суттєво покращити, якщо додатково урахувати й інші нефункціональні вимоги, а також перейти до прямих семантичних описів сервісів на OWL-S.

Схожі до табл.4.1 результати були отримані і для семантичних веб- сервісів, описаних на мові OWL-S, при використанні порівняння запиту і опису сервісу за різними даними під-онтології ServiceProfile [120]. Вони відображені на рис. 4.8

для протестованих 200 сервісів. Перший метод враховував лише ***P***-порівняння, тобто інформацію провайдера.

Другий метод базується на ***IOPE***- порівнянні без врахування семантики передумов, бо на практиці це відбувається часто. І, нарешті, третій метод враховує всі наявні дані і описи ***P+IOPE+QoS*** з профілю сервісу (рис.4.6).

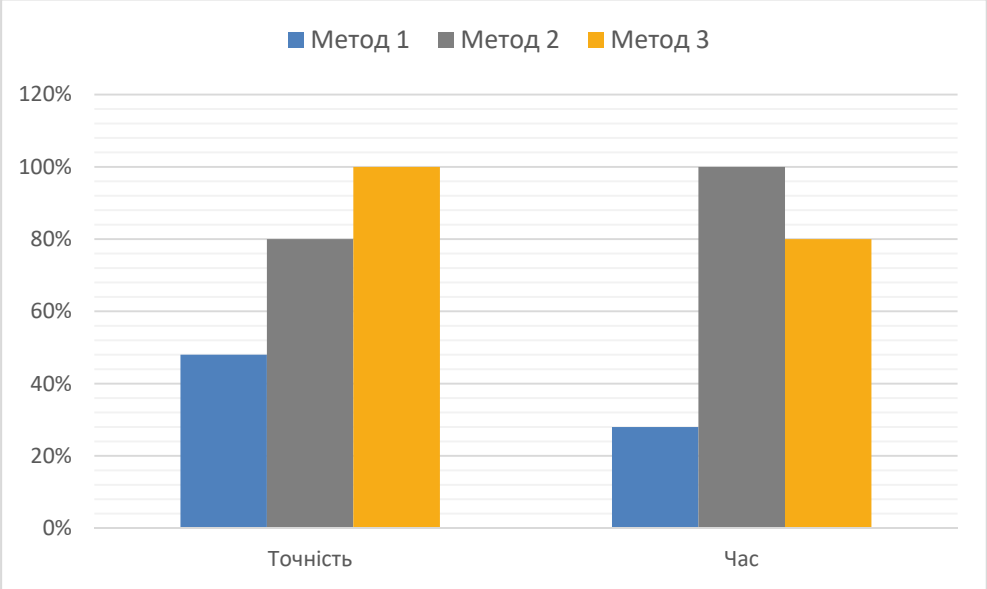


Рис.4.8. Порівняльні результати відкриття семантичних сервісів

Результати представлені в процентному відношенні і містять дані про точність пошуку і час пошуку (позначка 100% на осі ординат відповідає значенню 120 мсек)..

4.6. Композиція сервісів

Автоматизація складання веб-сервісу – це механізм створення нових додаткових веб-сервісів із наявних сервісів автоматично. Це потребує семантичної підтримки в описі та застосування логічних висновків. Суттєву роль в автоматизації складання сервісу грає семантика, а саме: семантичні описи функціональних можливостей сервісів та таких процесів як відкриття, відбір і композиція. На сьогодні існує кілька інструментів складання композиційних сервісів, серед яких слід вказати *METEOR-S, DynamiCos, CoSMos, SeGSeC, SHOP2* [161].

Семантичні веб-сервіси різняться своїми внутрішніми та зовнішніми описами на мовах з чітко визначеною семантикою. Це дозволяє комп'ютерам розуміти описи можливостей сервісів для полегшення автоматизація таких процедур, як вибір сервісу та складання сервісів.

Стуктура системи загальної семантичної композиції представлена на рис. 4.8.

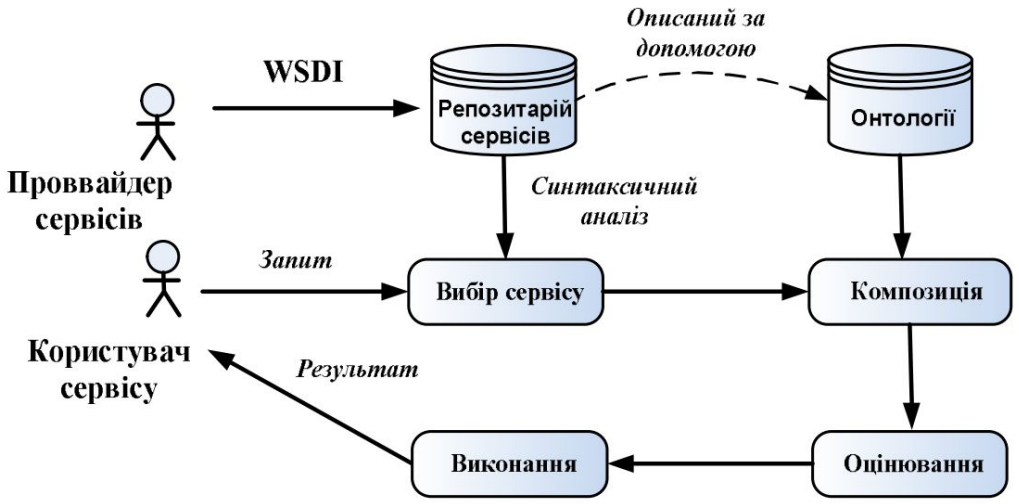


Рис. 4.8. Структура складання семантичних веб-сервісів

Провайдери сервісів постачають описи сервісів (WSDL) у репозитарій, до якого можуть звертатися усі об'єкти. Файл WSDL аналізується для отримання даних щодо сервісу, які включають місце сервісу, якість, функціональні аспекти тощо. Семантична інформація може бути представлена семантичними анотаціями у файлах WSDL. Вони мають бути узгоджені з онтологією домену. На базі запиту користувача обираються сервіси, які повинні забезпечити необхідну функціональність. Якщо один веб-сервіс не може задовольнити функціональність, яка вимагається споживачем, то з'являється необхідність об'єднувати наявні сервіси разом. Тому після вибору сервісів може розпочатися генерація можливих їх композицій. Кожна композиція оцінюється такими параметрами, як якість

сервісу, семантична схожість (відповідність), час виконання та інші. Побудова найкращої композиція виконується драйвером виконання і результат надається споживачу сервісу.

4.6.1. Модельно- керована композиція

Цей підхід має два аспекти: складання сервісу та управління складанням сервісів. При складанні сервісу формується бізнес-процес взаємодії з розробником сервісу. При управлінні складанням сервісу встановлюється взаємодія розробника додатку з системою композиції. Саме він несе відповідальність за виконання та управління композиціями. Розглядаються чотири етапи композиції: визначення, планування, побудови та виконання.

Вимоги до абстрактного складеного сервісу визначаються на початковому етапі. На етапі планування встановлюються послідовність і час виконання сервісів. Неоднозначні композиції сервісів, обрані зі списку порівняних сервісів, визначаються на стадії побудови. На останньому етапі композиція представлена як виконуваний веб-сервіс на мові опису веб-сервісів [162].

4.6.2. Онтологічна керована композиція

У роботі [163] описано декілька підходів до композиції сервісів шляхом узгодження інтерфейсів, зв'язаних з людиною, і одноранговою (peer-to-peer) композицією. Композиція, на основі порівняння інтерфейсів, виконується автоматично, а доменна онтологія використовується для встановлення послідовностей операцій. Процес починається з вхідних параметрів користувача і сервіси змінюються доти, доки не будуть досягнуті необхідні вихідні параметри. Метою композиційного процесу є встановлення найкоротшої послідовності сервісів. Вага складових визначається значеннями семантичної подібності. Вони оцінюються на основі значень тривалості, обчислювальної вартості, надійності та доступності.

Композиційний підхід із залученням людини підключає користувачів до вибору потрібних веб-сервісів і побудови композиції. Набір вибраних веб-сервісів класифікується на основі семантичного значення відповідності.

Свобода вибору веб-сервісів залишається за користувачем. В одноранговій композиції кожен учасник (однорангова особа) зв'язаний з сервісами, що належать до певного домену. Кожен учасник має також належати до певної спільноти, яка є кластером учасників, що надають сервіси для певного домену. У кожній спільноті є майстер-учасник і резервний учасник. У майстер- учасника є дані про усіх майстрів і резервістів усіх спільнот, при цьому резервні учасники віддзеркалюють майстер-учасників. Запит на обслуговування від користувача отримує одноранговий суб'єкт, якщо він не є майстром-учасником спільноти, то він посилює запит до свого майстра. Потім майстер знаходить спільноту, що відповідає запиту, і пересилає запит до майстра цієї спільноти. Майстри встановлюють, які сервіси в їх спільноті забезпечують користувачеві необхідний результат.

Підхід у [164] використовує формальну модель композиції з використанням матриці причинно-наслідкових зв'язків (CLM). CLM є моделлю для оперативного рівня композиції, де сервіси зв'язуються за їх семантичним описом. Семантичні зв'язки між параметрами веб-сервісів є важливими при формуванні нового веб-сервісу. CLM обчислює причинні зв'язки та зберігає їх у вигляді матриці. Мета процесу – це виявити найбільш підходящу композицію, залежнуо від семантичних зв’язків.

Процес композиції веб-сервісу включає три основні кроки. Це модулі відкриття сервісів, функціональності і процесу композиції [165]. Модуль пошуку веб-сервісів знаходить відповідні сервіси для зазначеного запиту. Модуль функціональності зв'язаний з порівнянням входів / виходів сервісів з входами та виходами шуканого веб-сервісу. Задача спрямована на вибір найкращого набору сервісів із реєстру з точки зору задовольнення вимог запиту. Модуль композиції на функціональному рівні вибирає сукупність поєднаних сервісів, здатних задовольнити запит. Цей процес переважно зв'язаний з модулем пошуку сервісів і

спрямований на пошук відповідного сервісу. На рівні композиції сервіси, що складені з існуючих севісів, мають виконуватися безпосередньо для досягнення бажаного результату.

4.6.3. Композиція на основі семантичних анотацій

Семантичні описи на мові WSDL з семантичними анотаціями на SAWSDL забезпечують ідентифікацію відповідних веб-сервісів, що дозволяє автоматично відкривати сервіси під час їх пошуку. У [166,167] представлений підхід для автоматичного відкриття та складання семантичних веб-сервісів.

За допомогою SAWSDL можна описати семантику елементів, визначених в описі WSDL. Кожен елемент зв'язаний з властивостями і поняттями онтології. Анотації SAWSDL можна застосувати до інтерфейсів, операцій, вхідних і вихідних параметрів. Процес відкриття починається, коли викликається запит, який виконує роль веб-сервісу композитора. Запит містить такі параметри, як список доступних входів, список бажаних результатів, список потрібних операцій, максимальна глибина композицій, тайм-аут і дозволи на перебудову композицій. Семантичні веб-сервіси будуть автоматично скомпоновані завдяки використанню механізмів відкриття сервісів, при чому поцедура композиціїя визначається шляхами на графі з сервісами як вузлами та семантичними узгодженнями (посиланнями) як ребрами.

Висновки до розділу 4

1. Показано, що стрімке розповсюдження у всіх галузях сучасного суспільства індустрії сервісів, що базуються на використанні сервіс- орієнтованої архітектури, передбачає створення найближчим часом розподілених сховищ з мільйонами сервісів, які можна було би знаходити і відкривати на основі вимог замовників прикладних додатків. Найкращим варіантом для реалізації таких сервісів є веб-сервіси завдяки їх стандартизації і прив’язку до Інтернет.

2. Універсальна система URL-адресів і технологія гіпертексту у поєднанні з пошуковими програмами утворили середовище, де інформація не

тільки передається, а й інтелектуально обробляється. В розділі описано, як організувати пошук і витяг веб-сервісів, описаних за допомогою спеціальних мов (наприклад, WSDL), з універсального реєстру UDDI за синтаксичною інформацією описів при зіставленні ключових слів. Але все одно потрібна людина-програміст, щоб остаточно вибрати потрібний сервіс із підмножини сервісів з подібними наборами ключових слів.

3. Доведено, що семантичні веб-сервіси усувають цей недолік, і це великий крок уперед в напрямку спрощеного доступу до сервісів через мережу. Вони мають свої мови опису (наприклад, OWL-S) і повністю незалежні від платформи реалізації, оскільки базуються на використанні онтології – свого роду словника понять предметної області і сукупності явним чином визначених припущень щодо змісту цих понять. У найпростішому випадку онтології використовуються для підвищення точності пошуку сервісу і відкриття такого сервісу, у якому згадується в точності шукане поняття, а не довільних сервісів, в текстах опису яких зустрілося задане ключове слово.

4. Загальновідомо, що у різних предметних областях одні й ті самі поняття можуть бути представлені різними термінами. Механізм онтології у цих випадках дозволяє формувати осмислені ієрархічні взаємозв'язки між сервісами, тобто реалізувати композицію сервісів, здатну задовольнити запит користувача, хоча в її описі може і не бути деяких ключових слів з вхідного запиту. Але для цього потрібно навчитися формувати семантичні анотації до опису веб-сервісів.

5. Досліджденні методи пошуку семантичних веб-сервісів, засновані на оцінках концептуальної близькості між елементами онтології сервісів (розміщених у реєстрі) і пошуковому запиті, заданому у вигляді множини триплетів. Оцінки близькості між триплетами можуть використовуються для визначення близькості між метаописами сервісів. Механізми пошуку можуть застосовувати онтології для вибірки сервісів з синтаксично різними, але семантично однаковими словами. Онтології також можуть використовуватися для організації обміну даними та інтеграції програм.

6. Показано, що усі існуючі інженерні методи пошуку і відкриття

сервісів об’єднані використанням оцінок задовольняння сервісами реєстру функціональних і нефункціональних вимог, що містяться у запиті користувача. Вони різняться залежно від того, які нефункціональні вимоги враховуються (контекстні, мовної схожості, категорії сервісів, якості сервісів QoS), як вони враховуються (поодинці чи у зв’язках, одноразово чи ієрархічно, наперед перевірки функціональних вимог чи паралельно з ними). У зв’язку з цим запропонований гібридний метод пошуку веб-сервісу, який відрізняється тим, що спочатку за категорією сервісу формується звужена область пошуку шляхом відсіювання непричетних до запиту сервісів (виділенням одного з кластерів), а потім проводиться уточнення попереднього вибору застосуванням семантичного порівняння описів відібраних сервісів із запитом, при цьому перевірка відповідності функціональніх і нефункціональних вимог виконується паралельно, завдаки чому скоротчується час пошуку сервісів.

РОЗДІЛ 5. АВТОМАТИЗАЦІЯ ПРОЕКТУВАННЯ СИСТЕМ СЕРВІСІВ

5.1. Генерація бізнес-логіки на базі онтології бізнес-процесу

Як було зазначено, SOA пропонує IT-підрозділам можливість розробки нових бізнес-додатків за рахунок повторного використання компонентів із існуючих програм у рамках підприємства, а не написанням функціонально надлишкового коду з нуля і розробки нової інфраструктури для їх підтримки. Конфігурація та координація сервісів у архітектурі, заснованій на сервісах, і композиція сервісів і процесів однаково важливі у сучасних системах сервісів.

У Семантичному Веб онтології забезпечують основу для формування комп’ютерно-зрозумілих даних і для обміну інформацією між людьми і комп’ютерами з врахуванням її синтаксичних і семантичних особливостей. Розрізняють наступні види онтології [171]:

- **Онтології високого рівня (*High-level (upper) ontologies***). Цей вид онтології описує найзагальніші поняття, такі як простір, час, матерія, предмет, дія та інші, які не залежать від конкретної проблеми чи галузі.
- **Галузеві онтології (*Domain ontologies***). Цей вид онтології описує лексику, пов'язану із загальною галуззю (наприклад, менеджмент систем IT (ITSM) або бізнес-процеси), базуючись на термінах, введених у онтології верхнього рівня.
- **Онтології задач (*Task ontologies***). Цей вид онтології описує загальні задачі або етапи діяльності (наприклад, моніторинг або вимірювання), базуючись на термінах, введених у онтології верхнього рівня.
- **Онтології додатків (*Application ontologies*).** Цей вид онтології описує поняття (концепції), що залежать як від певного домену, так і завдання, тому вони часто базуються на обох пов'язаних онтологіях. Ці поняття часто відповідають ролі, яку відіграють доменні особи (об’єкти) при виконанні діяльності певного виду.

При цьому поняття в онтології мають бути близькі до об'єктів (фізичних або логічних) і відносин у тій предметній галузі, для якої створюється онтологія. Швидше за все, це іменники (об'єкти) або дієслова (відносини) у пропозиціях, які описують вибрану предметну галузь.

Сервіси мають властивості з трьох напрямків: параметри сервісів, *метрика оцінювання сервісів* та взаємодії сервісів. Параметрами сервісів можуть бути ефективність, етичні чи естетичні показники, але усі вони є властивостями для людини, що притаманне загальній системі (наприклад, системі сервісів). Оцінювання сервісів включає ефективність та продуктивність, які є об'єктивними показниками і частиною керованості будь- якої загальної системи.

Нарешті, сервісна взаємодія може бути потоком дій, енергії, матеріалів або предметів, інформаційних даних. Галузева онтологія, таким чином, підтримує 3-мірне уявлення про те, що таке є сервіс. З людино- значущої точки зору сервіс є властивість системи, зв’язана з ефективністю, етичними і естетичними показниками.

З точки зору об'єктивного результату сервіс є ефективним системним фактом або подією, а з точки зору потоку дій сервіс є виходом системи. Сервіс може генерувати людино-значущі результати, які є ефективними, етичними чи естетичними.

Сервіс має S - основні атрибути, де один з них керованість. Цей атрибут складається з трьох концепцій, а саме: об'єктивні метрики оцінювання, входи і виходи. Входами і виходами можуть бути енергія, матеріали, дані-інформація-знання або потік актів. Властивості елементів *бізнес-моделі* представлені у табл. 5.1 [173].

Властивості елементів ***бізнес-моделі*** представлені у табл. 5.1 [173].

Таблиця 5.1

Елемент онтології бізнес-моделі

Назва ВМ елемента	**Призначення**
Визначення	Опис елемента бізнес-моделі

Частина чогось	Визначає, до якого стовпця онтології належить елемент або до якого елементу належить цей суб-елемент
Пов'язані з	Описує, з якими іншими елементами онтології зв'язаний елемент
Набір	Вказує, на які суб-елементи може бути розкладений елемент
Входження елементу	Визначає кількість дозволених входжень елемента або суб-елемента всередину онтології
Атрибути	Списки атрибутів елементу або суб-елементу. Допустимі значення атрибуту у межах {значення1, значення2}. Частота їх появлення указані в дужках (наприклад, 1-n). Кожен елемент і суб-елемент мають два стандартних атрибути, які є назва (NAME) і опис (DESCRIPTION), що містять ланцюжок символів {abc}

Властивості елементів ***спрощеної бізнес*-моделі,** наданої у табл. 5.1 описуються таким чином. По-перше, *рядок "визначення"* використовується для опису елементу бізнес-моделі. По-друге, *рядок " частина чогось"* використовується, щоб встановити відносини елемента з чотирма базовими класами (стовпці бізнес-моделі*):* ***Service, Management, Actors, Cost-Benefit*** [172]. По-третє, *поле "пов'язані з"* вживаються для опису відносин поточного елемента з іншими елементами онтології. По-четверте, деякі елементи бізнес-моделі розкладається на суб-елементи, які описані в *полі "Набір"*. По-п'яте, описані "*атрибути"* про цінності та входження кожного елементу.

5.2. Ієрархічна система рівнів опису семантики системи сервісів

Групою управління об'єктами OMG (Object Management Group) запропонована ієрархічна система рівнів опису семантики системи сервісів при її моделюванні (рис. 5.1) [180].

5.2.1. M0: Інформаційний рівень

Коли моделюється бізнес-процес, екземплярами M0 рівня є елементи реального світу бізнесу (наприклад, IT-сервіси, люди, рахунки-фактури, а також продукти). Коли моделюється програмне забезпечення, екземплярами у рівні M0 є програмні відображення об’єктів реального світу (наприклад, комп'ютерна версія рахунку-фактури або замовлення, інформація про продукт, персональні дані персоналу). Таким чином, цей рівень містить фактичні дані, для управління якими розробляється програма.

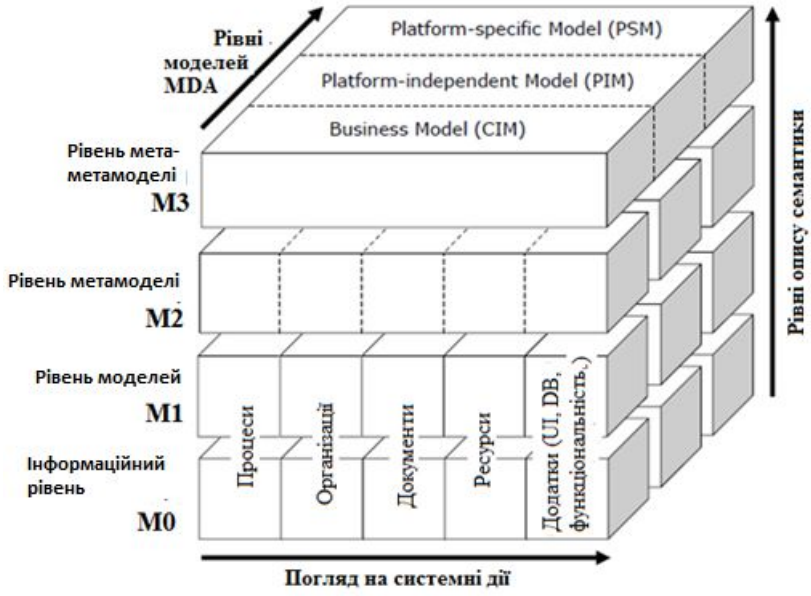


Рис. 5.1. Ієрархічна система рівнів опису семантики системи сервісів [176]

5.2.2. M1: Рівень моделей

Рівень моделей складається з *метаданих*, що описують дані інформаційного рівня. Метадані неофіційно агрегатуються у якості моделей (наприклад, моделі UML). Це рівень, на якому розміщені моделі, і він містить «модель» даних. Поняття на рівні M1 є категоризаціями або класифікацію екземплярів M0 рівня. Точно також кожен елемент рівня M0 завжди є екземпляром елемента в рівні M1.

5.2.3. M2: Рівень метамоделі

Рівень метамоделі складається із описів, які визначають структуру і семантику мета-метаданих. Мета-метадані неформально агрегатуються як

метамоделі, які описують різні види даних без конкретного синтаксису або позначення (наприклад, UML). Цей рівень має свою «модель» на рівні M1 (тобто модель моделей на M1). Елементи, що існують на M1 рівні (класи, атрибути та інші елементи моделі), самі по собі є екземплярами елементів на рівні M2.

Елемент рівня M2 вказує на елементи рівня M1. Рівень M1 містить поняття, необхідні для логічних висновків відносно екземплярів рівня M0, у той же час рівень M2 містить поняття, що потрібні для логічних висновків про поняття рівня M1.

5.2.4. M3: Рівень мета-метамоделі

Рівень мета-метамоделі складається із описів, які визначають структуру і семантику мета-метаданих. Іншими словами, це абстрактна мова для визначення різних видів метаданих. Цей рівень має свою «модель» на рівні M2 і, з історичних причин, він часто згадується як ***Meta Object Facility*** **(MOF**), що є стандартною мовою рівня M3. Таким чином, кожен елемент в M2 є екземпляром елемента M3, і кожен елемент на рівні M3 класифікує M2 елементи. M3 визначає поняття, необхідні для логічних висновків про поняття з рівня M2. Метамоделі забезпечують механізм, який сумісний із різними платформами, і визначає: (а) загальну структуру, синтаксис і семантику бізнес-процесів і платформи як метамоделі; (б) загальну модель програмування для будь-яких результуючих метаданих (наприклад, за допомогою Java, IDL та інших); і (в) загальний формат обміну даними (наприклад, XML).

Є кілька визначень для метамоделей, що надаються різними джерелами в інженерії. Усі вони зводяться переважно до визначення метамоделі як "моделі мови моделювання", коли метамодель визначає "конструкції мови моделювання та їх взаємозв'язки, а також обмеження і правила моделювання, але не конкретний синтаксис мови (рис.5.2). Іншими словами, метамодель визначає "абстрактний синтаксис і статичну семантику мови моделювання." Метамодель діє як фільтр, щоб «витягти» відповідні елементи з системи сервісів, для того, щоб побудувати відповідну модель. Будь-яка особливість (поняття або відносини), якої немає в

метамоделі, буде ігноруватися, коли будується модель, що представляє систему сервісів.

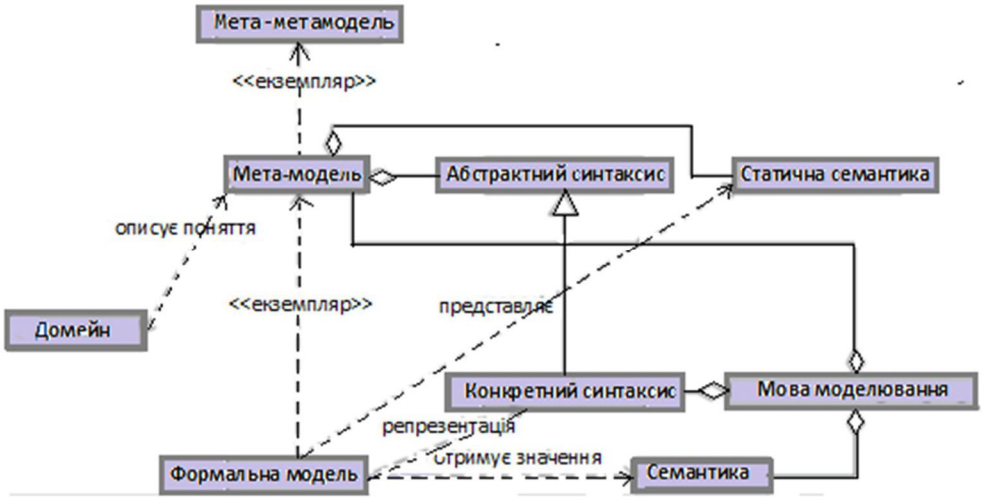


Рис. 5.2. Взаємозв'язки між семантикою, синтаксисом і мовами моделювання

Конкретний синтаксис або позначення мови полегшує презентацію і побудову моделей на мові для людини. Різні конкретні форми синтаксису можуть мати загальний абстрактний синтаксис. Наприклад, метамодель може бути виражена через різні позначення (наприклад, у графічно-базованій нотації або у текстово-базованій нотації), або навіть багато різних графічно-базованих позначень можуть використовувати одну і ту саму метамодель. Таким чином, конкретний синтаксис може бути визначений за допомогою мови моделювання, але він не є частиною метамоделі. Крім того, відмінності між абстрактним синтаксисом і конкретним синтаксисом дуже важливі, метамодель (а не конкретний синтаксис) є основою для автоматизованого інструментарію підтримки обробки моделей систем сервісів. Таким чином, абстрактний синтаксис мови моделювання має справу зі структурою концепцій у мові, не беручи до уваги їх презентацію і зміст.

Важливо відзначити, що *статична семантика* досить суттєво відрізняється від семантики, яка включена в абстрактний синтаксис метамоделі. Статична семантика визначає концепції в мові, які завдають обмеження і правила для оцінки того, добре чи ні що сформований вираз мови. Але семантика, яка є частиною абстрактного синтаксису моделі, не несе або мало містить інформації

про значення концепцій мови. З іншого боку, *метамодельна семантика* визначає, як кожну концепцію у метамоделі слід тлумачити і як саме це поняття трансформуються при перетворенні моделей. Усі перетворення однієї вихідної моделі на інші цільові моделі повинні підтримувати ті же значення метамоделі, яким відповідає вихідна модель.

Таким чином, слід зазначити, що метамодельна семантика не є частиною абстрактного синтаксису мови, хоча модель абстрактного синтаксису є попередньою умовою для визначення семантики метамоделі, оскільки метамодельна семантика додає зміст до понять, визначених в абстрактному синтаксисі. Тобто *метамодель є моделлю мови моделювання*, а інтерпретація метамоделі є відображенням елементів метамоделі на елементи мови моделювання таким чином, що ми можемо визначити значення істинності твердження у метамоделі для будь-якої моделі, описаної на мові моделювання. Оскільки метамодель є специфікацією, модель на мові моделювання є дійсною тільки, якщо жодне з цих тверджень не є помилковими. Тобто, може бути кілька інтерпретацій для і тієї самої моделі, і термін «метамодельна семантика» буде використовуватися відносно до інтерпретації моделі, яка визначає операції перетворення моделі.

І, нарешті, зазначимо, що моделі відповідає її метамодель замість того, щоб бути її «екземпляром». Для того, щоб бути дійсними, моделі мають відповідати своїй метамоделі. Моделі, сумісні у загальних метамоделях, можуть бути використані в якості повторно використовуваних артефактів для майбутніх проектів програмного забезпечення.

5.3. Трансформації моделі системи сервісів

Процес проектування системи сервісів може бути змодельований як набір перетворень моделей, які приймають початкову модель в якості вхідних даних, використовуючи набір правил перетворення [179, 180]. Перетворення неявно або явно визначають зв'язок між початковою і цільовими моделями, коли

перетворення саме по собі є також моделлю. Ці співвідношення можуть визначати собою *модель перетворення* (відносини між моделями на тій же самій мові) або *перетворення мов* (відносини між моделями, які описуються на різних мовах).

Модельні перетворення, як правило, засновані на початковій метамоделі і правилах перетворення, при цьому правила можуть бути засновані тільки на конструкціях метамоделі. Моделі перетворення можуть мати різні галузі застосування:

- Створення моделей нижнього рівня і в підсумку коду, від моделей більш високого рівня.
- Відображення і синхронізація між моделями на одному рівні або на різних рівнях абстракцій.
- Створення уявлення про систему на базі запитів.
- Еволюція завдань моделі, таких як рефакторинг моделі.
- Зворотне проектування від моделей нижнього рівня або коду до моделі більш високого рівня.

Для того, щоб виконати перетворення моделі, необхідно мати чітке уявлення про абстрактний синтаксис і семантику обох моделей, тобто початкової і цільової моделей. Коли визначають перетворення, то існує три різні архітектурні підходи:

1) Пряма модельна маніпуляція. Вона визначає доступ до представлення внутрішньої моделі і здатність маніпулювати представленням за допомогою набору процедурних інтерфейсів прикладного програмування (API). Мова, що використовується для доступу і маніпулювання API, зазвичай загального призначення, наприклад, Visual Basic або Java. Проте, моделі UML і мова дії UML, (xUML) можуть бути використані також.

2) *Проміжне представлення*. Воно визначає експорт моделі у стандартному форму (як правило, XML), яка може бути перетворена зовнішніми інструментами.

Наприклад, багато інструментів UML дозволяє експортувати й імпортувати моделі до і з XMI (в XML-based стандарті для обміну моделями UML).

3) *Підтримка декількох мов трансформації.* Вона визначає мову, яка надає набір конструкцій для явного вираження, складання і застосування перетворень.

Бажані характеристики для мов трансформації моделі будуть виглядати так:

- *Передумови*. Вони описують умови, при яких перетворення дає значущий результат;
- *Композиція*. Оскільки зазвичай легше складати компоненти, ніж будувати «об'єкти» з основних частин, комбінування існуючих перетворень для побудови нових композитів є бажаною властивістю;
- *Форма*. Доступність і прийняття мови залежить від її форми, важливо, що графічне представлення моделей краще ніж повністю текстуальні представлення;
- *Зручність і простота*. Сильно впливає те, чи мова є декларативною (що спрощує алгоритм перетворення) або імперативною (що пропонує знайому парадигму складання правил перетворення, тобто процедури послідовності, вибору і ітерацій), важливо також чи перетворення потребує участі крім мови самих користувачів, які могли б збалансувати легкість у розумінні, точності, стислості і простоті модифікацій по-різному.

На рис. 5.3 наведена структура класичної метамоделі для бізнес-процесу. Відповідно до визначень, наведених вище рівнів, M0 представляє екземпляри середовища виконання процесу (наприклад, клієнт з ідентифікатором $id = AAA$, який робить заказ $order\,id = 111$ у торговому центрі $cart\_id = 222$), M1 представляє модель (наприклад, бізнес-процесу), описану у BPMN, а мета-модель на мові BPDM знаходиться на рівні M2, де задані правила перетворення, тобто правила, визначені з використанням мови OVT. Ці мета-моделі завжди залежить від загальної мета-метамоделі (MOF), яка представлена на рівні M3. Будь яка мета-модель MOF складається із чотирьох приведених мета-рівнів.

MOF, як визначалося раніше, представляє набір елементів мови моделювання, що використовуються у описі та під час розробки мета-моделі у конкретному середовищі моделювання, і існують на рівні M3. Мова UML-сумісна з MOF і також базується на чотиришаровій архітектурі. Як графічна мова моделювання, UML підтримує MOF основними конструкціями при визначенні та візуалізації мета-моделей.

Перетворення моделей є ключовим елементом у моделюванні систем сервісів, забезпечуючи плавний засіб оброблення вхідних онтологічних моделей з ціллю генерування, фільтрації і поновлення цільових моделей.

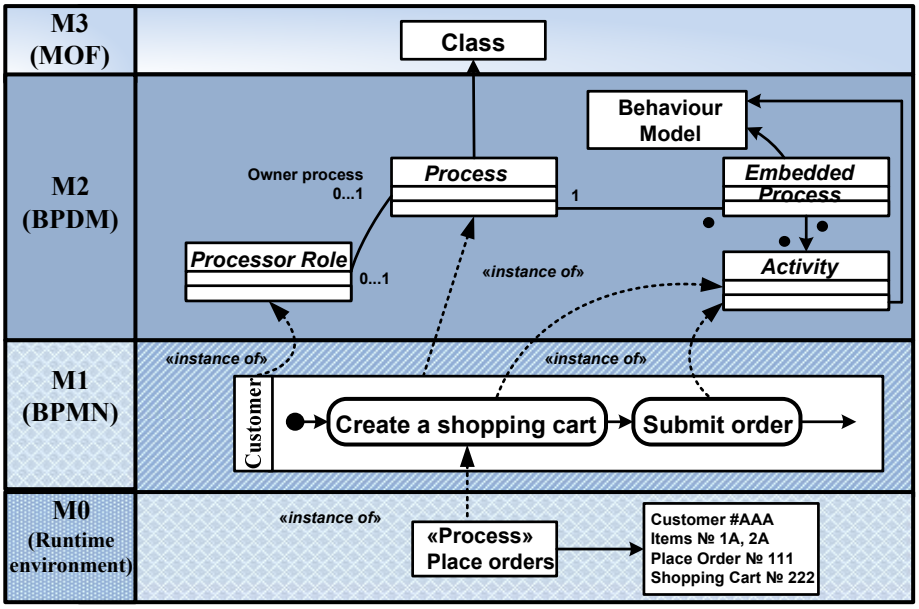


Рис. 5.3. Структура мета-моделі [191]

Мови перетворення моделей реалізують різні типи перетворення, такі як *модель-модель* (ММ) або *модель-код* (МК). Перетворення завжди залежить від моделі, яка вміщує набір тверджень про деякі конкретні системи.

Оскільки застосовність перетворення моделей зросла, було запропоновано багато мов трансформації, які пропонують безліч функцій як з відкритим вихідним кодом, так і на основі комерційних ліцензій. Ці мови трансформації та підходи включають в себе ***QVT*** (Query/Views/Transformations), ***ATL*** (ATLAS Transformation Language), ***Epsilon*** (Extensible Platform of Integrated Languages for

model management), ***KerMeta*** і ***XSLT*** (XML Stylesheet Language Transformations). Огляд і порівняння цих мов приведено в [179].

5.4. Модельно-керований підхід до проектування бізнес-процесів (MD)

При моделюванні системи сервісів як бізнес-процесів, слід розглядати три аспекти: ***об'єкти***, з якими оперує бізнес; ***процеси,*** які він виконує; ***події***, керуючі змінами процесів і об'єктів. Відповідно, можна визначити три типи моделювання: інформаційне, подійне і функціональне. Усі три типи моделювання об'єднуються в одній проектній процедурі, яка заснована на використанні ***модельно-керованого підходу*** (MD, model-driven approach), добре відомого в інженерії [181,193]. MD можна використовувати, щоб отримати контроль і систематично покращувати увесь життєвий цикл IT-рішень: від моделювання загального бізнесу (сприяння ефективному зв'язку між бізнес-аналітиками й IT розробниками і виконання конкретних вимог рішення) до розробки, впровадження, інтеграції та управління різними програмними артефактами.

MDA на основі тришарового підходу забезпечує:

- ***Обчислювальну незалежну модель CIM*** (Computing Independent Model), що описує систему з обчислювальної незалежної точки зору, висвітлюючи структурні аспекти системи, змінюючи фокус з моделювання домену на моделювання архітектури.
- ***Незалежну від платформи модель PIM*** (Platform Independent Model), яка може розглядатися як визначення системи з погляду технології нейтральної віртуальної машини або обчислювальної абстракції.
- ***Модель для конкретної платформи PSM*** (Platform Specific Model), яка, як правило, охоплює технічні концепції та сервіси, що складають платформу реалізації. Тобто ця модель спрямована на конкретну технологію реалізації системи сервісів.

У нашому контексті обирається *платформа на основі веб-сервісів*. Постачальник сервісів робить абстрактний опис інтерфейсу сервісу, який може бути використаний потенційними користувачами сервісів, щоб знайти і

викликати сервіс за допомогою повідомлень на основі XML. Можливості пошуку веб-сервісів при зверненні до їх репозитарію, в якому потенційні користувачі можуть шукати відповідні сервіси, та вибору протоколів для зв’язку сервісів та їх клієнтами , утворюють платформу реалізації як сервіс-орієнтовану архітектуру. Різні типи платформи можуть відрізнятися.

Загальна властивість платформи є її сервіс-орієнтована архітектура, конкретними технологіями реалізації – веб-сервіси і сервісні механізми *Apache Axis* або *Oracle BPEL* [183]. Поведінка і взаємодії процесів є найважливішими складовими моделювання і розуміння архітектури програмного забезпечення. *Зокрема семантичне моделювання та онтології схожі за своєю метою.*

Мови опису онтології, що базуються на лозиці, підходять для підвищення ефективності традиційних мов моделювання для модельно-керованої сервісної архітектури. Тому семантичне моделювання на основі онтології доцільно використовувати для підтримки керованої моделями архітектури систем програмного забезпечення, основаної на сервісах.

Мета-модель описує властивості і конструкції кожної з CIM, PIM і PSM моделей точно. Зв'язок між названими моделями показаний на рис. 5.4 [180].

Як вже говорилося раніше, *метамоделювання* є одним з найбільш важливих понять MDA. Онтології і метамоделі мають різні цілі: онтології є описовими і вони відносяться до структури домену (тобто, до реального світу), у той час, як макромоделі є розпорядчими і належать до MD рішень.

Онтології розглядаються як CIM моделі, тому цільові (концептуальні) моделі PIM і PSM також відображають семантику для даного домену додатків, які повинні захопити семантики про домени реального світу (Real World) незалежно від конкретних потреб додатків. Таким чином, онтологія виступає в якості основи метамоделі для того, щоб генерувати концептуальні моделі для реалізації конкретних інформаційних систем. Екземпляри є конкретними об'єктами певної концепції. Їх властивості визначаються відносинами між концептами та їх властивостями.

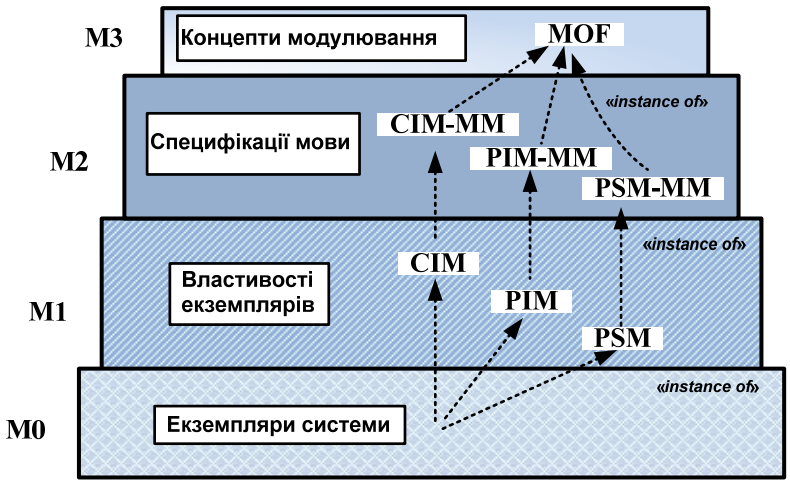


Рис. 5.4. Зв'язок між моделями MD і метамоделями

У цьому онтології подібні до мов моделювання, таких як UML. Однак онтології об'єднують моделювання з логічними висновками. Властивості концепцій визначаються у термінах (універсальних чи екзистенціальних) кількісних показників відношення до інших концептів.

5.5. CIM – Обчислювальна незалежна модель

Мета обчислювальної незалежної моделі (CIM) є опис домену і його концепцій та властивостей. Як правило, відрізняють дві точки зору на моделювання домену. Концепції, представлені у вигляді ієрархії, називають ***інформаційною точкою зору*** в MD. Поведінку представлену у формі, що базується на процесах, називають ***підприємницькою (або процесною) точкою зору*** в MDA, заснованою на концепціях розподіленої обробки. Можна додати третій аспект – ***структурну точку зору***, коли враховуються структурні властивості об'єктів і процесів. Мета полягає у забезпеченні єдиного представлення онтології, яке може охопити усі три точки зору [189]. Процесно-орієнтована онтологія повинна відображати такі типи сутностей домену:

- Два типи концептів (понять): об'єкти, які є статичними сутностями, і процеси, які динамічними сутностями.
- Три типи відносини: "***is a***" (підклас відносини), "***has part***" (компонентні відносини), і "***depends***" (відношення залежності).

Таким чином генерація CIM вимагає наступних кроків: визначення задачі та опис системних вимог, які досягаються визначенням функціональності системи (важливо, що робиться, а не як робиться). Будується діаграма для уточнення основних функцій системи і сценарій їх взаємозв'язків.

Обмеження на властивості, поняття і відносини можна визначити як логічні формули. Відносини підкласів є класичною формою співвідношень понять в онтології. Для доменних систем програмного забезпечення композиція об'єктів і процесів із перспективних компонентів є додатковою, але також і необхідною інформацією. Залежності корисні для опису вхідних-вихідних відносин між об'єктами і заходами, які обробляють їх. Особливі вимоги замовлення на складені процеси можуть бути визначені через обмеження.

Необхідно визначити чи ідентифікувати мову онтології, яка може забезпечити необхідну основу позначень. Мови OWL, базовані на онтології, з підтримкою відносини між компонентами і залежностями задовольняють цим вимогам для позначень у вибраному підході до моделювання. Але є і інші підходи, коли модель CIM описується на мові BPMN [184,185] і навіть UML [186,187], коли вона формується з бізнес-процесів без залучення доменної онтології. Недавно з'явилася мова SoaML (Service-oriented architecture Modeling Language), з допомогою якої описується профіль UML та метамодель для моделювання та дизайну сервісів в рамках сервісної архітектури [188]. Модель CIM на мові SoaML представляє собою відображення загальних бізнес-процесів, складене лише виходячи з сервісів, учасників та їх відносин. Мова SoaML дозволяє індивідуально представити кожний сервіс як сукупність різних дій (дія є основною одиницею хореографії сервісу SoaML), що сприяє визначенню шлюзів у цій моделі [197]. При цьому на виході кожної дії додається об'єкт даних, що містить стан об'єкту.

5.6. PIM - Незалежна від платформи модель

Незалежна від Платформи Модель (ПІМ) зміщує фокус від розрахункової незалежності домену на архітектурні обмеження, що накладаються

обчислювальним середовищем. Архітектура та процеси є ключовими аспектами на цьому рівні моделювання сервісів. Архітектура впливає на сервіси, їх архітектурні конфігурації і процеси взаємодії. Архітектурні конфігурації визначають процеси взаємодії (віддалений виклик, моменти активації сервісів, їх взаємодії) між різними сервісами прикладного додатку. Знову ж таки онтологія використовується для визначення цих аспектів.

Сервіси є компонентами архітектури системи. Вони визначають початкову точку моделювання архітектури. На сьогодні запропоновані різні підходи до онтології сервісів, які розрізняються за способом відображення в онтології сервісів і процесів. Так що важливо ураховувати не тільки властивості сервісів, але також їх конфігурації і композиції у бізнес-процесах.

Мова OWL-S, що була розглянута у Розділі 1, підтримує складені сервіси, але мова WSPO більш сфокусована на підтримку складених сервісів після їх композиції. Сервіси (і процеси) в WSPO не представлені як поняття, а як відносини, що визначають співвідношення між станами системи з метою реалізації узгодженої процесно-орієнтованої структури, яка дозволяє робити логічні висновки про поведінку програмного забезпечення.

WSPO надає шаблон для опису сервісу і сервісного процесу. Інформація про синтаксичні параметри, що мають відношення до активності, повинні бути реалізовані за допомогою сервісних операцій. А семантична інформація, така як попередні умови, прикріплюється до кожної активності, як визначено у шаблоні. Цей шаблон сервісного процесу в моделі PIM визначає основну структуру станів і сервісних процесів. На рис. 5.6 шаблон застосовується до сервісу відправлення даних і відображає такі відносин:

- поняттями онтології при такому підході є стани (pre / post-states), параметри (in / out-parameters) й умови (pre / post-conditions).
- сервіси і самі процеси відповідають перехідним відносинам, а їх синтаксичні та семантичні описи надані в двох формах: об'єктів параметрів (syntax) і умов (semantics), зв'язаних через описи відносини.

Мова WSPO відрізняється від традиційних мов онтологій сервісів двома специфічними властивостями. По-перше, на основі розширення опису логіки вона сприяє визначенню процесу на основі ітерації, послідовної і паралельної композиції та вибору операторів. По-друге, дані надаються процесам у вигляді параметрів, які вводяться як постійні елементи процесу.

WSPO фактично формалізує наше розуміння центральних сервісів і процесів в контексті ПЗ сервіс-орієнтованих систем. Онтології дозволяють робити логічні висновки про специфікації. WSPO дає змогу також робити логічні висновки про композицію сервісів в архітектурі.

Очевидно, що архітектура знаходиться у центрі цієї моделі, але розглянутий підхід не є мовою опису архітектури, хоча переслідує мету відділення обчислень (у межах сервісів) від комунікації (процеси взаємодії між сервісами). Мови опису архітектури зазвичай передбачають опис компонентів (тут сервісів), міжз'єднань (каналів між сервісами) і конфігурацій (об'єднання компонентів і міжз'єднань). Розглянутий підхід наближається до цієї мети, розглядаючи сервіси у якості компонентів і описи (expressions) для процесів у якості конфігурацій.

5.7. PSM – Модель конкретної платформи

Платформа веб-сервісів складається з мов, протоколів і програмних засобів. Для моделі конкретної платформи (PSM) важливі два аспекти: сама модель платформи та модель реалізації конкретних платформ. Модель платформи базується на технології веб-сервісів і на принципах сервіс-орієнтованої архітектури. Моделі реалізації конкретних платформ характеризується основними моделями уживаних мов опису платформ. Платформа у нашому випадку відрізняється від типових MD платформ, таких як Java, .NET, або CORBA. Платформою веб-сервісів є опис сервісів з абстрактним синтаксисом (WSDL), описом анотацій семантичних сервісів (наприклад, SAWSDL, WSMO або OWL-S), і визначенням сервісного процесу (наприклад, WS-BPEL або WS-CDL). Тому перетворення на цьому рівні помітно відрізняється ***від традиційних перетворень PIM-на-PSM моделей***. Ми орієнтуємося на моделі платформ з описом на WSMO і

WS - BPEL мовах, хоча кінцева мета модельно-керованої розробки є забезпечення перетворення моделей для різних цільових мов.

- *Інтероперабельність* сервісів є ключовим завданням платформи веб-сервісів. Два застереження визначають методи, що працюють на цьому рівні: абстрактний опис сервісів для підтримати їх відкриття і віддалений виклик та стандартизований монтаж сервісів і процесів. Дві різні моделі, що підтримуються різними мовами, розглядаються нижче:
- *Опис і відкриття*. Повинні підтримуватися абстрактні синтаксичні та семантичні інтерфейси сервісів. Мова WSDL підтримує синтаксичну інформацію, необхідну для виклику сервісу. Сервіси як основні компоненти бізнес-процесів можуть бути представлені у вигляді понять в онтології. Цей підхід базується на широкому використанні мов опису онтології, таких як OWL-S і WSMO.
- *Процеси і композиція*. Мова виконання бізнес-процесів для веб-сервісів (WS-BPEL) є однією із пропонованих мов сервісів. Специфікації WS-BPEL можуть бути отримані шляхом перетворення виразів процесів на WSPO.

Переваги від використання онтології для опису та відкриття, які базувалися на порівнянні функціональних і нефункціональних властивостей запитів і сервісів, описаних семантично на OWL-S і синтаксично на WSDL, були розглянуті у попередньому розділі.

Концепти WSMO є концептами центральних сервісів та допоміжного опису сутностей, тобто виразів різних видів. Відносини у шаблоні представляють властивості сервісів двох видів. Такі властивості, як ***preCond, postCond***, припущення (***assumption***) і ефекти (***effects***), пов'язані з семантикою сервісів і називаються *можливостями* (***capabilities***). Такі властивості, як ***messageExchange***, є синтаксично-орієнтованими аспектами інтерфейсу.

Стандартизовані формати опису і виклику забезпечують сумісність (інтероперабільність). Необхідну функціональність для конкретного процесу можна отримати з інших місць. Прикладом є функція аутентифікації для онлайн-

медичної системи моніторингу. Сервіс аутентифікації, інтегрований у процес моніторингу, може бути наданим віддаленим постачальником.

WS-BPEL, у свою чергу, пропонує широкий вибір операторів управління робочим потоком, включаючи послідовності (sequence), паралельної композиції (flow), перемикач (switch, choice) й ітерації (while, iteration). Це прямі аналоги WSPO для комбінаторних відносин.

5.7.1. Семантика моделей перетворення

Метамодельна семантика для кожного із прошарків може бути сформована на основі стандартних розглянутих підходів:

- *Онтологія предметної області* (CIM прошарок) може бути визначена у термінах понять (концепцій) і відносин. Абстрактний синтаксис онтології на рівні CIM моделі виглядає так:

Domain ontology (OWL-style) – CIM рівень
model ::= relatioship | constraint_
concept::= object | process
relationship type::= is a | has part | depends
relationship ::= concept relationship type concept
constraint::= concept Constraint relationship Constraint

- *Архітектурні та технологічні аспекти* (PIM прошарок) можуть бути визначені у термінах концепцій, що представляють стани, а відносини – переходи між станами. WSPO також визначається у термінах опису логіки з деякими розширеннями, які підтримують посилання на динамічну логіку. Абстрактний синтаксис онтології на рівні PIM моделі виглядає так:

Service process ontology (WSPO) – PIM рівень

model ::= pre process post _

process::= preState procExpr postState

pre::= preState preCond condition | preState inObj syntax

post ::= postState postCond condition | postState outObj syntax

procExpr ::= process | !procExpr | procExpr; procExpr | procExpr + procExpr | procExpr || procExpr

- *Аспекти інтероперабільності* (PSM прошарок) можна розділити на інтерфейс (визначений у термінах понять і відносин) та конфігурацію і поведінку процесу (визначених у термінах механізмів перехідного стану). Запропонований опис сервісу на WSMO є також строго визначеним у термінах логіки. BPEL є мовою документообігу і бізнес-процесів, центральні поняття можуть бути визначені у термінах обчислення процесу разом із виразами для процесу на підмові WSPO. Абстрактний синтаксис онтології на рівні PSM моделі виглядає так:

Service ontology (WSMO) – PSM рівень

model ::= service

service::= interface capabilities

interface::= messageExchange nonFunctProp

capabilities ::= preCond postCond assumption effect nonFunctProp

На рис. 5.5 представлені моделі MD з різним абстрактним рівням відображення системи. Використання CIM для моделювання бізнес-систем спрощує розуміння цих систем, усуває проблеми та їх джерела. Підтримка процесів інформаційних систем відповідає потребам бізнес-систем. Інформаційні системи зосереджуються на методах обробки інформації та інформаційних процесах, тому ці системи технічно визначені і можуть моделюються за допомогою PIM. Інформаційні системи підтримуються системами прикладного

програмного забезпечення, останні використовують потужність різних технологій для забезпечення можливостей обробки інформації для інформаційних систем. Технологічні рішення систем бізнес-програмного забезпечення моделюються за допомогою PSM. *Платформо-залежна модель* (PDM) представляє собою набір технічних концепцій і специфікацій, що визначають різні компоненти, які складають платформу, та сервіси, які надаються цією платформою.

Без можливості виконувати автоматично перетворення моделей кожна існуюча MD модель повинна бути розроблена та досліджена окремо і / або має бути вручну перетворена на інші моделі. Це часто вимагає багато зусиль.

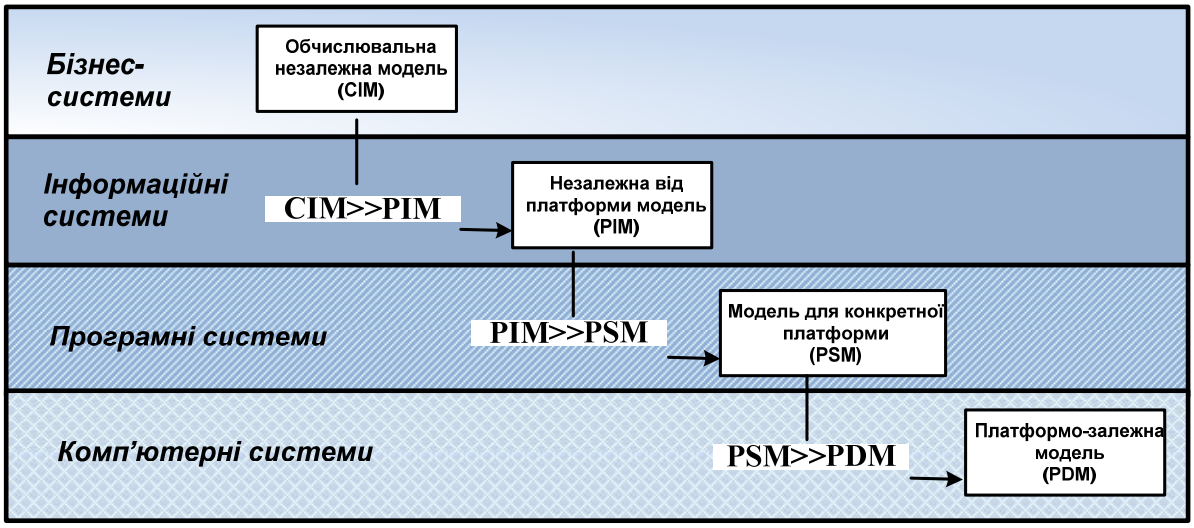


Рис. 5.5**.** Відображення MD на різні системи [197]

Проте, коли використовуються автоматичні перетворення моделі, то відношення між різними концепціями повинні бути розроблені лише один раз для пари мета-моделей, а не для кожного екземпляра моделі. Більш того, автоматичні перетворення моделей дозволяють скоротити час і витрати, необхідні як для розробки моделі, так і виправлення помилок розробників.

5.8. Автоматизація перетворення моделей

Для визначення відображення між двома принципово різними моделями потрібна загальна основа, яка описує як вихідні, так і цільові області перетворення, а також словник трансформації. Такою спільною базою є мета-модель. Автоматичні перетворення моделей зосереджується на побудові моделей,

специфікації трансформаційних правил, підтримці інструментів та автоматичній генерації коду та документації.

На сьогодні запропоновано велику кількість підходів до трансформації моделей [176,179, 197], які можна відобразити загальною схемою перетворення на (рис. 5.6).

На рис. 5.6 відображені базові процедури перетворення (*Мова перетворення*) і (*Правила перетворення*), які передбачають застосування різних мов опису для вхідної та вихідної моделей. Правила перетворення визначаються на рівні метамоделей, тоді як їх виконання (*Виконання перетворення*) відбувається на рівні моделей.

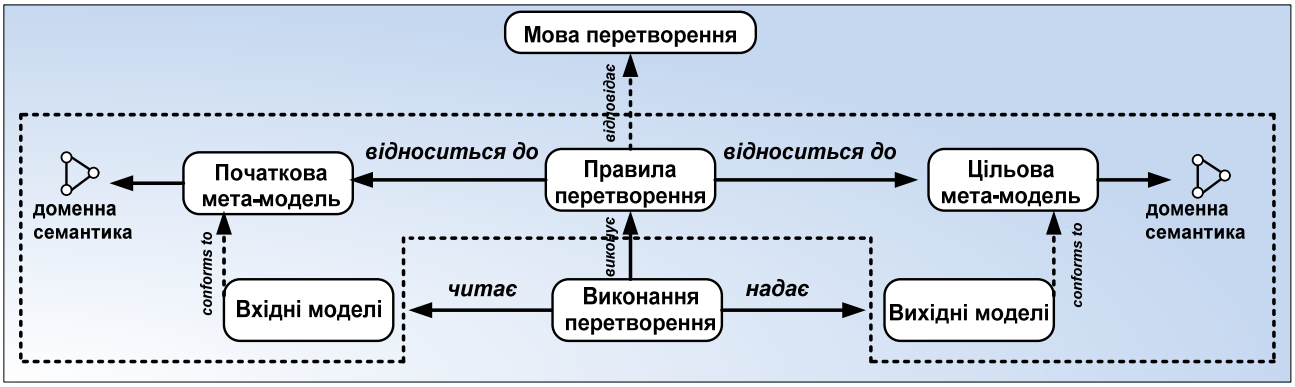


Рис. 5.6. Загальна схема перетворення моделей

Визначимо правила перетворення, засновані на моделях і шаблонах, на прикладі трансформації моделей різних рівнів, приведеному на рис. 5.7 :

- перетворення CIM-на-PIM змінює фокус з моделювання домену на моделювання архітектури, яке може зажадати додаткову інформацію. Однак, враховуючи детальну домену модель, розглянуту з трьох точок зору, вважаємо, що вся необхідна інформація для шаблону PIM доступна.
- перетворення PIM-на-PSM потребує додаткової інформації, зокрема, для комплексного абстрактного опису функціональних і нефункціональних властивостей. Обидва перетворення можуть у будь-якому випадку автоматично генерувати структуровані шаблони і скелети, які містять основні елементи.

5.8.1. Перехід від CIM до PIM

CIM-прошарок підтримує абстрактне, обчислено-незалежне моделювання домену. Ця модель перетворюється (відображається) на обчислено-орієнтовану, але платформо-незалежну сервіс-орієнтовану модель**.**

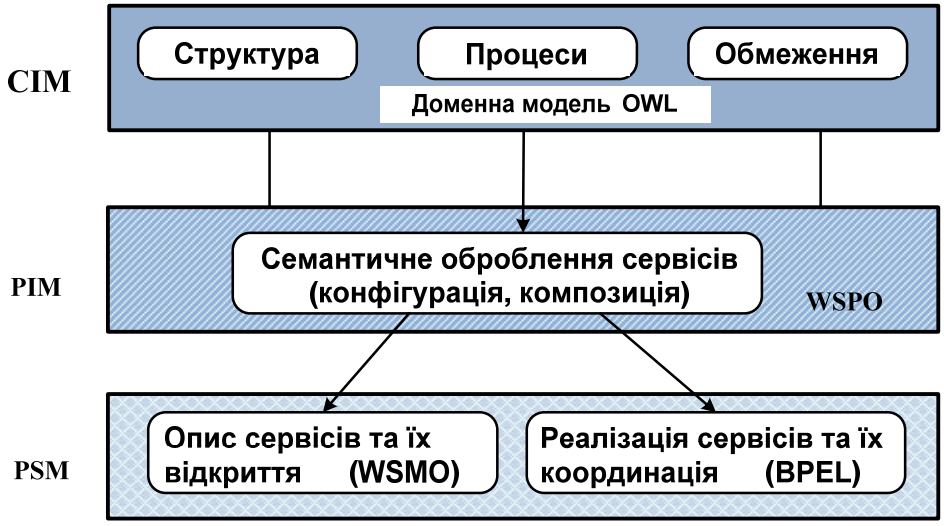


Рис. 5.7. Приклад перетворення моделей різних типів

5.8.2. Перехід від CIM до PIM

PIM-рівень підтримує аналіз та обґрунтування архітектури та технологічних аспектів, таких як конфігурація і композиція, на абстрактному рівні. Отже, інформація повинна бути додана тільки до CIM, щоб забезпечити достатній рівень опису структури на PIM-рівні, якщо процесна (підприємницька) точка зору неадекватно змодельована.

Правила перетворення CIM-на-PIM залежать від способу формування вхідної моделі CIM і мови її опису. Якщо модель CIM формується на базі доменної онтології і прив’язана до рівнів опису семантики системи сервісів M0-M3 (рис. 5.5), то правила для перетворення CIM-на-PIM визначаються перетвореннями мов опису онтології OWL і опису процесів веб-сервісів WSPO (табл. 5.2).

У MDA кроки перетворення визначаються у термінах модельної розмітки і застосувань шаблонів. Розмітка анотації (або метадані) об’єктів в оригінальній моделі підтримує відображення, яке вказує, як ці об’єкти перетворюються на

цільової моделі. Мітки можуть підтримувати визначення шаблону, який має бути розгорнутий. Прикладом цього є правило перетворення CP0, яке визначає створення PIM-шаблону для CIM-концепцій.

Таблиця 5.2

Правила перетворення для моделей CIM-на-PIM або переходу від описів на мові OWL до описів на мові WSPO [189].

Правила	**Аспект**	**Опис**
CP0	Шаблон (template)	Створення для кожного елементу процесу у CIM шаблону в PIM
CP1	Елемент процесу (process element)	Визначення в PIM елементу процесу через елемент процесу в CIM
CP2	Стани (states)	Створення концептів за замовчуванням для перед / пост-станів
CP3	Синтаксис (syntax)	Створення для кожного in / out-параметрів процесів окремого елементу об’єкту
CP4	Семантика (semantics)	Створення пре- і пост- умов залежно від наявності зовнішньої додаткової інформації у вигляді обмежень
CP5	Вирази для процесу (process expressions)	Створення складеного процесу, якщо вирази для процесу доступні у вигляді обмежень, використовуючи вирази для відносин у WSPO

Детальна модель CIM з обмеженнями насправді містить усю інформацію, необхідну для заповнення шаблон WSPO. Загалом, не вся інформація CIM використовується. Наприклад, структурні аспекти відносяться тільки до прошарку конкретних платформ.

5.8.3. Перехід від PIM до PSM

Модель для конкретної платформи (PSM) визначає, як відмічалося раніше, насправді дві окремі моделі: *сервісні метадані* на основі описів онтології, які використовуються для відкриття сервісів, і описи процесу оркестровки і хореографії, які використовуються для композиції сервісів. Відповідні правила перетворення для цих двох аспектів (при виборі WSMO для опису онтології і WS-BPEL – для оркестровки сервісів) представлені у таблиці 5.3.

Таблиця 5.3

Правила перетворення моделей PIM-на-PSM, або переходу від описів на мові WSPO до описів на мові WS-BPEL [190]

Правила	**Аспект**	**Опис**
PP1	WSMO	Відобразити відносини процесів із моделі PIM, базованій на WSPO, на сервісні концепти WSMO і заповнити властивості messageExchange і Pre / postCond відповідно
PP1.1	WSMO Message Exchange	Відобразити WSPO in / out-об'єкти у WSMO описи повідомлень обміну
PP1.2	WSMO pre / post conditions	Відобразити WSPO пре / пост умови у WSMO пре / пост умови
PP2	WS-BPEL	Відобразити відносини композиційного процесу WSPO у BPEL-процесах
PP2.1	WS-BPEL process partners	Створити для кожного процесу BPEL партнерський процес
PP2.2	WS-BPEL orchestration	Перетворити вираз для кожного процесу у BPEL-запиті на стороні клієнта, а BPEL процеси receive і -reply перенести на сторону сервера
PP2.3	WS-BPEL Process activities	Перетворити процесні комбінатори ’;’,’+’,’!’,’||’ на відповідні BPEL комбінатори: sequence, pick, while, flow (послідовність, вибір, тоді як, потік)

Відображення WSPO-на-WSMO копіює у PSM функціональні властивості як синтаксичні, так і семантичні. Подібно станам, які додаються до CIM, щоб забезпечити структуру, що визначає поведінку процесу, до PIM додаються нефункціональні аспекти, щоб підтримати подальший опис для відкриття сервісу.

Відображення WSPO до WS-BPEL перетворює вирази для процесів у BPEL на скелет бізнес-процесів. WS-BPEL є мовою реалізації для виконання процесу у вигляді процесу оркестровки. Реалізації WS-BPEL підтримуються сервісними механізмами, доступними від різних постачальників. Оскільки WSPO включає підмову для виразів процесу, схожу на WS-BPEL, модель WSPO може бути повністю переведена у WS-BPEL. Для того, щоб сформувати повні, здатні для виконання специфікації WS-BPEL, додаткові елементи, що включають

інформацію про простір імен й інформацію про типи партнерів, мають бути визначені. Це платформо-специфічна інформація і, отже, не включена у WSPO.

Головна перевага MD – це забезпечення декількох перетворень для даної CIM для підтримки різних платформ. Наприклад, можна забезпечити перетворення для ***OWL-S*** і ***WS-CDL*** в якості альтернативи для ***WSMO*** і ***WS-BPEL***, дозволяючи користувачеві перемикатися легко між мовами платформи. Ці автоматизовані перетворення також дозволять обійти обмеження реалізації, надаючи багатий набір технологічних комбінаторів на рівні PIM для підтримки перетворення.

Взаємодія та інтеграція моделей є центральним питанням у модельно-керованій методології проектування. Хоча UML і мови онтології це не одне і те саме, але вони істотно перекриваються, і тому дозволяють виконувати перетворення між UML-моделями і онтологіями на OWL основі.

На рис. 5.8 відображені перспективи подальшого розвитку технології MDA з урахуванням нового стандарту мови опису онтології метамоделі ODM (Ontology Definition Metamodel), який розроблений OMG [195].

Згідно [195] ODM забезпечує MD формальним обґрунтуванням для подання, управління, взаємодії і застосування бізнес-семантики. ODM описує, наприклад, перетворення моделей OWL на MOF-модель на мові UML. Схема трансформації моделей на рис. 5.10 адаптована до трансформації аксіом онтології до моделі правил *(Rule model*), що є важливою та невід'ємною частиною кожної концептуальної моделі даних.

Мова UML 2.0 дозволяє описати концептуальні моделі, в яких відносини між підкласами можуть бути семантично визначені. Побудова аксіом, правил виводу і теорем, однак, підтримується мовами онтології, але не UML.

Мова обмеження об'єктів OCL (Object Constraint Language) є розширенням мови UML, яке додає формальні специфікації для UML, але *до-* і *після* умови не описують поведінку сервісу та процесу адекватно. Вимоги тут виходять за рамки можливостей UML і OCL.

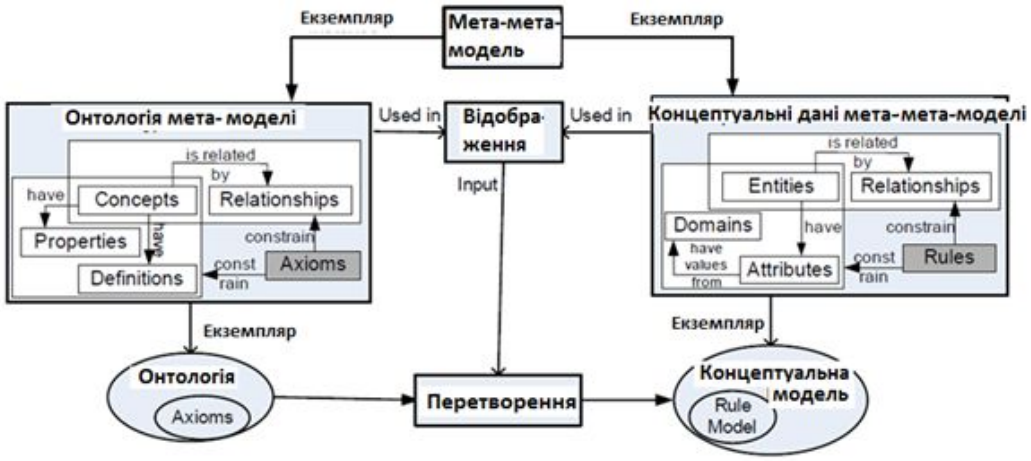


Рис. 5.8. Автоматизоване перетворення онтології на концептуальну модель MOF [196]

Ці вимоги записуються на новій мові визначення онтології метамоделі ODM (Ontology Definition Metamodel). *Виявилося, що використання мов онтології насправді навіть більше підходить для контексту сервіс-орієнтованої архітектури*, ніж UML / OCL, через зв'язок між описом логіки (на мові онтології, таких як OWL) і динамічної (модальної) логіки, яка дозволяє робити логічні висновки про процеси.

Можливе злиття онтологічного підходу та MD призводить до наступного процесу розробки прикладного програмного забезпечення [179]:

- Побудувати незалежні загальну (GO) і доменну (DO) онтології, при цьому частина DO перетворюється на обчислювальну незалежну модель (CIM).
- Перетворити модель CIM у незалежну від платформи модель (PIM) згідно з правилами таблиці 5.2.
- Перетворити модель PIM на модель конкретної платформи (PSM) згідно з правилами таблиці 5.3, що справедливі при виборі WSMO для опису онтології і WS-BPEL – для оркестровки сервісів), і на залежні моделі ADM (наприклад, у схему бази даних), визначити методику порівняння і анотації сервісів.
- Перетворити частини онтологій DO і GO на онтологію контексту (CO), що використовується зовнішнім додатком.

- Застосувати перетворення PSM і ADM на коди додатків і програмних артефактів.

Таким чином, здається, що застосування підходу MD у поєднанні з онтологією може допомогти інженерам програмного забезпечення розробляти та управляти складними системами. З одного боку, використання моделей і метамоделей для розробки програмного забезпечення є загальноприйнятою практикою у програмній інженерії, а з іншого боку, використання онтології як платформи моделювання та логічних висновків для управління моделями успішно використовувалися дослідниками за останнє десятиліття. Крім того, як було зазначено раніше, онтології забезпечують загальні концептуальні знання домену, що дає змогу інженерам програмного забезпечення моделювати як предметну галузь, так і рішення домену.

5.9. Генерування API шляхом перетворення описів онтології

Виявляється можливим також пряме перетворення описів онтології з мови OWL на мову Java [202]. Основна ідея полягає у створенні набору Java інтерфейсів та класів онтології на мові OWL такими, що екземпляр класу Java представляв би обраний екземпляр онтології з більшістю його властивостей, відносинами між класами і обмеженнями. Хоча існує деяка фундаментальна семантична різниця між дескрипною логікою (DL) та об'єктно-орієнтованими (ОО) системи, у першу чергу, зв'язана з повнотою та здійсненністю. Але можна мінімізуємо вплив цих розбіжностей і відобразити більшу частину набагато багатшої семантики OWL у семантику Java. ***В результаті Java API, створений з перетворення онтології, можна використати для створення додатків***, функціональність яких відповідає специфікаціям етапу проектування, що розглядається. Мова OWL має три підмови (OWL Lite, OWL DL і OWL Full) і представлене у [202] рішення для OWL Full є досить повним, хоч були вжиті деякі компроміси при переході до домену ОО. Нижче у таблицях приведені основні правила перетворення двох описів.

Таблиця 5.4

Відображення OWL класів

Класи	OWL	Java
Basic Class	*A*	*interface IntA* *Class A implements IntA*
Class Axioms	*A equivalentClass B*	*interface IntAB extends IntA, IntB* *Class A/B implements IntAB*
	B subClassOfA	*interface IntB extends IntA*
Class Description	*A = intersectionOf(B,C)*	*interface IntA extends IntB, IntC*
	A = unionOf(B,C)	*interface IntB/IntC extends IntA*
	A = complementOf | disjoint With B	*interface IntA {IntA ABBlocker()}* *interface IntB {IntA ABBlocker()* *(overridden blocking method – ABBlocker*
	A = oneOf(I1,I2)	*Enum A {I1,I2}*

Таблиця 5.5

Відображення OWL властивостей

Властивість	OWL	Java
Property Associations	*P domain A*	*interface IntA* *{set(List);List getP()}*
	Range	*VetoableChangeListener* *(range – instanceOf,* *Functional – equal elements)*
Property Descriptions	*Functional*	
	Inverse Functional	*–*
	Symmetric	*PropertyChangeListener* ***(call appropriate accessor functions*** *inside PropertyChange())*
	Transitive	
Property Relationships	*EquivalentProperty*	
	SubPropertyOf	
	InverseOf	
Property Restrictions	*Cardinality*	*VetoableChangeListener* ***(check constraint satisfaction*** *inside VetoableChange())*
	HasValue	
	SomeValuesFrom	
	AllValuesFrom	

Таблиця 5.6

Відображення фрагментів еквівалентного коду Java

Java Element	Code
Interface ***IntProduct***	{.. void setProdutName (List); List getProdutName (List);.. void setHasMaker (List); List getHasMaker (List);.. ..}
Interface **I*ntAutomobileProduct*** **Extends IntProduct**	{.. // Java class implementing this interface registers the **SVFAAutomobileCompanyChecker** Listener on property HasMaker ..}

Class
SVFAAutomobileCompanyChecker
Implements *VetoableChangeListener*
	{..
	void VetoableChange (PropertyChangeEvent evt);
	List newValue = evt. get newValue();
	for (int i = 0; i< newValue.size (); i++)
	if (newValue.elementAt(i)
	instanceof AutomobileCompany)
	return;
	throw SomeValuesFromException();
	..}

З таблиці 5.6 видно, наприклад, що інтерфейс Java *IntProduct* відповідає класу OWL *Produc*t і містить відповідні методи доступу (set / get) для OWL властивостей *ProductName*, *hasMaker* та інші. Ці методи доступу приймають та повертають аргументи типу List. У Java 1.5 можна визначити такі списки *String* та *Company* відповідно. Можна бачити, що потрібно мати окремий клас Java, що визначає кожен інтерфейс, для того, щоб створювати корисні об'єкти Java (екземпляри); у цьому випадку, клас Java *Product* реалізує інтерфейс *IntProduct.*

Відомі й інші публікації з перетворення мов програмування.

5.10. Персоналізація охорони здоров’я на базі сервіс-орієнтованої платформи мобільної медицини для потреб мешканців зони АТО

У світі наростає бум електронної охорони здоров'я (***e-Health***) *як* ***нового стилю надання пацієнтам широкого спектру медичних послуг з профілактики, діагностики, лікування та моніторингу*** *[203-206], базовими* складовими якого є:

- ***Електронна медична карта*** **(**Electronic health record, EHR**),** яка дозволяє організувати передачу даних пацієнтів між різними медичними працівниками (лікарями загальної практики, фахівцями з певних захворювань та іншіми);
- ***Комп'ютеризація роботи лікаря*** **(**Computerized physician order entry): складання в електронному вигляді маршрутів лікування пацієнтів та запитів на діагностичні тести з отриманням відповідей-результатів;
- ***Електронний рецепт*** **(**ePrescription**):** виписка рецептів пацієнтам з одночасною їх електронною передачею фармацевтам;
- ***Клінічна система підтримки прийняття рішень*** (Clinical decision support system**):** надання інформації для фахівців в електронному вигляді про регламенти і стандарти для проведення діагностики та лікування хворих;
- ***Телемедицина*** **(**Telemedicine): фізична і психологічна діагностика та лікування пацієнтів на відстані, у тому числі телемоніторинг функцій пацієнтів;
- ***Інформатика про здоров'я*** (Consumer health informatics): використання інформаційних ресурсів для пацієнтів з медичної тематики про здоровий спосіб життя чи або лікування певних захворювань;
- ***Удосконалення знань про здоров'я*** **(**Health knowledge management**)**: надання лікарям оглядів останніх медичних журналів, керівних вказівок з найкращих практик лікування або епідеміологічних спостережень, включаючи ресурси Medscape і MDLinx та інші;

- ***Віртуальні лікарські бригади*** **(**Virtual healthcare teams**),** що складаються з фахівців різного профілю, які співпрацюють і обмінюються інформацією про пацієнтів за допомогою цифрового обладнання;
- ***Мобільна медицина*** (m-Health**)**, що включає використання мобільних пристроїв для збору агрегованих даних про стан здоров'я пацієнта, надаючи цю інформацію фахівцям-практикам, дослідникам і самим пацієнтам, а також моніторинг у режимі реального часу життєво важливих органів пацієнта і пряме надання медичної допомоги (за допомогою мобільної телемедицини);
- ***Медичні дослідження*** **(**Medical research**)** з використанням високопродуктивних розподілених інфраструктур, потужні обчислювальні ресурси яких та можливості управління даними дозволяють оброблення великих обсягів різнорідних даних;
- ***Інформаційні системи охорони здоров'я*** **(**healthcare information systems), що використовують ПЗ для планування графіків прийому лікарів, управління даними пацієнта, управління розкладом роботи лабораторій, взаємодії зі страховими агенціями та виконання інших адміністративних завдань, зв'язаних з організацією охорони здоров'я.

Підраховано, що завдяки новим інформаційним технологіям можна знизити витрати на медичне обслуговування літніх людей на 25 %, а материнську і дитячу смертність – на 30 %. Крім того, ці технології дозволять підвищити ККД лікарів у сільській місцевості: кожен лікар зможе обслуговувати вдвічі більше пацієнтів, що при гострій нестачі лікарів на периферії серйозно поліпшить показники здоров'я населення.

Теперішня концепція медичної реформи в Україні базується головним чином на останній з перелічених вище складових e-Health, а саме: на інформаційних системах охорони здоров’я [206]. Зараз здійснюється реалізація першої черги медичної реформи під назвою MVP (minimum viable product), яка покликана, по-перше, впровадити базовий необхідний функціонал для підтримки реформи первинної ланки, а, по-друге, закласти підвалини та створити

інфраструктуру для e-Health на національному рівні. Створюються реєстри пацієнтів, ЗОЗ (засобів охорони здоров’я)та медичних працівників. Маючи дані у цих реєстрах, система може зв’язати пацієнта, сімейного лікаря та ЗОЗ у відповідний контракт у реєстрі контрактів. Маючи інформацію з реєстру пацієнтів, реєстру медичних працівників та реєстру лікарських засобів, система може формуватиме електронний рецепт (e-prescription) для певного пацієнта з певною діючою речовиною та дозуванням, а також формувати платежі відповідно до кількості пацієнтів, що обслуговуються.

Передбачається подальше поглиблення реформи. Зокрема, задіяти у сфері моніторингу здоров'я мобільні телефони, які забезпечують нові засоби збору інформації як вручну, так і автоматично у багатьох галузях. Багато з нинішніх телефонів мають вбудованими цілий ряд датчиків, таких як мікрофон, камера, гіроскоп, акселерометр, компас, датчики наближення, GPS та ліхтарик. Наприклад, новітний модуль MediaTek Sensio, вьудований в смартфон, забезпечує 6 функцій [217]: виміряйте частоту серцевих скорочень,

мінливість серцевого ритму, артеріальний тиск, насичення периферичного кисню (SpO2) і записує електрокардіограму (ЕКГ) та фотоплетизмограму (ППГ). Він використовує комбінацію світлодіодів та світлочутливий датчик для вимірювання поглинання червоного та інфрачервоного світла кінчиками пальців. Для запису ЕКГ і Для запису ЕКГ і ППГ користувачеві потрібно буде помістити пальці з кожної руки на електроди, встановлені з боків телефону.

Нові покоління професійних медичних сенсорів можуть легко підключатися до смартфонів і безпосередньо передавати результати зондування. Це створило більш ефективний і зручний спосіб збору інформації про здоров'я людей у вигляді даних про артеріальний тиск, насичення киснем, рівень глюкози в крові, пульс, електрокардіографію (ЕКГ) та електроенцефалограму (ЕЕГ).

Тому на третьому етапі проведення медичної реформи в Україні (2018-2020 роки) необхідно буде вирішувати в галузі m-Health наступні задачі:

- ***Створення мобільної платформи***, що забезпечує прийняття обґрунтованих рішень щодо лікування хворих шляхом моніторингу

лікарями зі своїх смартфонів або планшетів електронних медичних записів пацієнтів.

- ***Розроблення базових складових платформи***: блоку сервісів пацієнтів, що включає портативні пристрої діагностики і вимірювання, які підключені до смартфонів (планшетів) пацієнтів; блоку передачі, зберігання і обробки даних з центральним сервером і блоку сервісів лікарів і консультантів, оснащених засобами мобільного зв'язку та обчислювальної техніки.
- ***Модифікація структури охорони здоров'я*** з урахуванням засобів m-Health, базованих на застосуванні технологій Інтернету речей (Internet of Things), що передбачають підключення до мережі безлічі приладів і тим самим збільшення її вразливості з точки зору безпеки.
- ***Впровадження сервіс-орієнтованого підходу*** до композиції прикладних додатків для пацієнта, лікаря і функціонування усієї мобільної платформи, що прискорює процес їх створення і робить його гнучким, таким, що подаються модернізації та адаптації до завдань підтримки конкретних маршрутів лікування окремих хворих шляхом масштабування сервісів: виключення деяких з них, додавання нових, заміни одних на інші того самого призначення.
- ***Перенесення усіх медичних сервісів і додатків у хмару***, що призводить до зменшення капітальних витрат та використання наявних активів, підвищення швидкості та гнучкості розробки та надання нових (диференційованих) сервісів, ефективного управління відносинами з клієнтами у хмарі (наприклад, білінг).
- Сумісність медичних послуг для груп пацієнтів з різними захворюваннями можна досягнути за допомогою медичної платформи, яка складається із:
    - Блоку сервісів пацієнта та портативних діагностичних пристроїв.

- Хмарної платформи для зберігання та обробки медичних даних (зокрема PHR).
- Блоку сервісів медичних працівників (рис. 5.9).

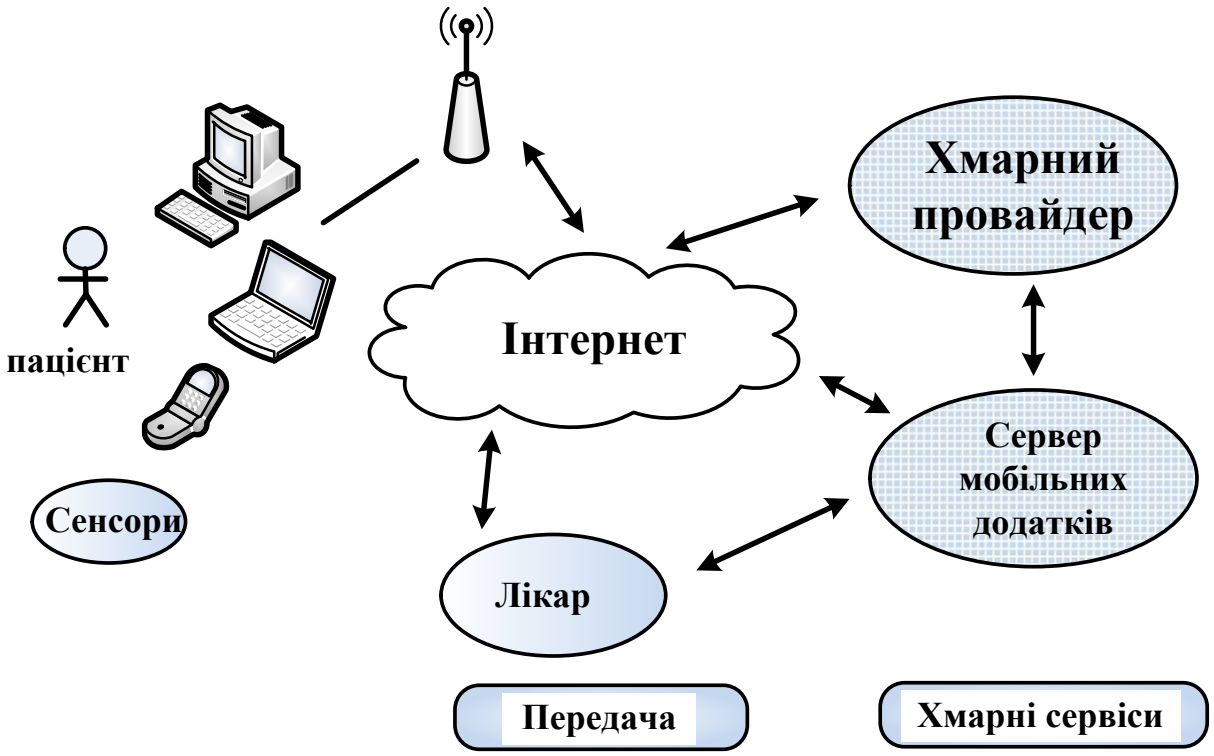


Рис. 5.9. Організація медичних послуг у хмарі

Мережа натільних сенсорів використовується для постійного моніторингу та реєстрації життєво важливих параметрів пацієнтів, які страждають хронічними захворюваннями, такими як діабет, астма, серцеві напади тощо. Пацієнт може попереджатися, наприклад, про серцеві напади ще до їх появи через вимірювання змін його життєво важливих ознак.

Хмарна платформа використовує мережі віддалених серверів, розміщених у Інтернеті для зберігання, керування та обробки даних, а не локальний сервер або персональний комп'ютер. Вона також містить сховище PHR та дані про здоров'я пацієнтів, які отримані від різних джерел.

Постачальник послуг охорони здоров'я (Health Care Service Provider): – це особа або установа, які забезпечують профілактичну, лікувальну або реабілітаційну медичну допомогу систематично індивідуумам, сім'ям або громади.

5.10.1. Блок сервісів пацієнта

Реалізує сервіси обслуговування, які максимально відповідають рекомендаціям фахівців з лікування захворювання, при цьому до них відносяться:

- Можливість виконання на дому вимірювань показників захворювання за допомогою приладів (глюкометра, тонометра, інсулінової помпи тощо), підключених до смартфону або планшету пацієнта, збір і агрегування даних та їх бездротова передача на смартфон (планшет) пацієнта, відправка даних на сервер, де вони одразу ж стануть доступні лікуючому лікарю через медичну карту пацієнта.
- Віддалений моніторинг стану пацієнта, своєчасні нагадування лікуючим лікарем (медсестрою) пацієнту про необхідність прийняти ліки або пройти відповідну процедуру, а також про його дії у невідкладних ситуаціях, що виникають при загостренні хвороби. Якщо життєві показники, за якими ведеться моніторинг, наблизяться до небезпечної межі, лікар (та / або співробітники швидкої допомоги) тут же будуть сповіщені про це.
- Інтегрування у соціальні мережі, де пацієнт може отримувати всебічну інформацію про методи лікування захворювання, ділитися своїм досвідом у цій сфері. Пацієнт може створювати свій обліковий запис, вступати в групи за інтересами, розміщувати свої матеріали і коментувати публікації інших учасників, завантажувати фотографії, обмінюватися електронними листами з іншими членами мережі .
- Надання на смартфон медичної інформації про лікування захворювання, організації правильного харчування, безпечне та ефективне використанні ліків та їх дозування. Поради про дії у невідкладних ситуаціях, коли потрібна консультація спеціаліста або необхідно зв'язатися з медичними центрами та лікарнями, готовими прийти на допомогу, а комунікація з лікуючим лікарем відсутня.

5.10.2 Блок сервісів медичних працівників

Забезпечує ефективну взаємодію медичного працівника (лікуючого лікаря з пацієнтом через смартфон (планшет) лікаря і реалізує сервіси лікування, перераховані нижче:

- Віддалений моніторинг стану пацієнта за даними медичної карти пацієнта, можливість бачити результати аналізів пацієнта, історію хвороби, призначене раніше лікування; оперативно вносити і редагувати інформацію у карту пацієнта. Оцінювати динаміку змін у стані здоров'я пацієнта (чи спостерігається прогрес чи регрес у лікуванні?) І вносити змін до плану його лікування. Отримати від сервера за результатами обробки даних екстрені повідомлення про наближення життєвих показників пацієнта до небезпечної межі.
- Складання плану лікування (roadmap) при постійному доступі до сервера з медичними картами пацієнтів, формування рекомендацій з правильного лікування, харчування та ведення здорового способу життя у вигляді плану лікування і розміщення їх на сервері, контроль їх виконання. Поради щодо застосування інших препаратів для лікування і дій пацієнта у невідкладних ситуаціях, коли потрібна консультація спеціаліста або необхідно зв'язатися з медичними центрами та клініками, готовими прийти на допомогу хворому.
- Можливість проведення консультацій з колегами і фахівцями у режимі телесеансів і відеоконференцій.
- Можливість підтримки контакту з пацієнтом, можливість домовитися про особисту зустріч (про відвідування пацієнта вдома або у медичному закладі). Своєчасні нагадування пацієнтові про необхідність прийняти ліки або пройти відповідну процедуру, а також про його дії у невідкладних ситуаціях.

5.10.3 Сервіси хмарної платформи для зберігання та обробки даних

Забезпечує функціонування системи віддаленого моніторингу стану пацієнтів і реалізує такі сервіси управління:

- Доступ до даних портативних діагностичних приладів та узгодження форматів їх даних при передачі на сервер з медичними картами.
- Підтримка персональних медичних карт пацієнтів, заповнення їх полів і зчитування даних з карт, у яких містяться оброблені дані та сформовані лікарем рекомендації з правильного лікування, харчування та ведення здорового способу життя.
- Композиція сервісів депозитарію та редагування формального опису бізнес-процесу лікування, опис архітектури мобільного додатку на мові опису бізнес-процесів.
- Формування потоку завдань (послідовності програмних модулів окремих сервісів) для конструювання додатків з опису бізнес-процесу лікування з урахуванням даних, необхідних для реалізації окремих сервісів, і використанням прийнятих стандартів і протоколів передачі інформації.
- Екстремальні попередження для лікаря і пацієнта за несприятливими результатами аналізу поточних даних, коли життєві показники пацієнта, за якими ведеться моніторинг, наближаються до небезпечної межі.
- Виклик швидкої допомоги за поточним місцезнаходженням пацієнта (GPS), якщо серверу не вдалося установити екстрений зв'язок з лікарем.
- Формування рахунку за надані пацієнту послуги у лікуванні, використовуючи записи персональної медичної карти, про тиск, і направлення рахунку у банк пацієнта.
- Підтримка електронного рецепта (ePrescription), що прописує необхідні ліки пацієнту і який був складений лікарями та медичними працівниками на основі результатів аналізу даних електронних медичних карт і заміряних медичних показниках пацієнтів на дому. Пацієнти замовляють ці медикаменти на дому у будь-якій аптеці, при цьому не буде

необхідності пред'являти рецепт від лікаря, тому що аптеки будуть мати доступ до ePrescription.

5.10.4 Організація надання медичних послуг

Оскільки у випадку mHealth комунікаційне навантаження на мережу значно зростає, то доцільним представляється ще більше розгалуження національної служби охорони здоров’я. Наприклад, регіональні медичні центри можна перетворити на Центри громадського здоров’я (ЦГЗ) з мережею Сервісних центрів здоров’я (СЦЗ), зв'язаних з місцевими лікарнями, госпіталями, сімейними лікарями, службами швидкої медичної допомоги і службою ліквідації наслідків надзвичайних ситуацій. У свою чергу, регіональні центри ЦГЗ взаємодіють з Інформаційним центром охорони здоров’я (National Health Informational Centre) при МОЗУ, який виступає в якості оператора Національного центру охорони здоров'я.

Зацікавлені місцеві мешканці (наприклад, літні люди з хронічними хворобами) можуть укласти з ЦГЗ Угоди на надання сервісів постійного моніторингу здоров'я, доглядання і мобільної медичної допомоги і бути приписані до одного із СЦЗ, де на сервері будуть накопичуються дані про історію їх хвороби у вигляді персональної електронної медичної карти (PHR). В Угоді чітко оговорюються заходи, які будуть проведені на допомогу пацієнту, при яких симптомах і в які терміни (наприклад, при тяжкому сердечному нападі для мешканця Києва необхідна операція повинна бути здійснена після отримання сигналу тривоги протягом двох годин в Інституті серцевої хірургії).

Для організації моніторингу стану здоров’я пацієнт оснащується бездротовими датчиками, які кріпляться на його тілі і підключені до його смартфону, який використовується як персональна система моніторингу здоров’я (ПСМЗ). Його будинок також оснащується фіксованими датчиками, підключеними бездротово до смартфону, які здатні збирати контексту інформацію, таку як, наприклад, про час, місце знаходження пацієнта (через GPS смартфону і вбудованої RFID-мітки), положення пацієнта (стоїть, сидить, лежить,

впав), діяльність пацієнта у даний час (ходьба, фізичні вправи, водіння, приймання ліків, спить), з ким пацієнт взаємодіє або хто поруч (особливо, доглядач), параметри середовища (температура, атмосферний тиск). ПСМЗ налагоджується на персональний медичний стан пацієнта; при цьому параметри моніторингу однозначно оптимізуються під пацієнта за допомогою алгоритму машинного навчання, реалізованого у пристрої.

Якщо життєві показники, за якими ведеться моніторинг, наблизяться до небезпечної межі, черговий лікар СЦЗ повідомляється про це через ПСМЗ пацієнта і Портал спостереження СЦЗ. Він негайно реагує на тривогу наступним чином:

- дистанційно налаштовує ПСМЗ і датчики пацієнта на максимальну пропускну здатність і на найвищу якість інформації, що знімається з датчиків, щоб перевіритися у попередньому діагнозі причини сигналу тривоги, і перемикає смартфон пацієнта на голосовий зв'язок, щоб не турбувати пацієнта, надаючи йому перші поради (типу зупинитися, сісти, лягти та інші), виходячи з отриманої контекстної інформації про місце розташування пацієнта; активну діяльність пацієнта, історії приймання ліків, знаходження доглядача поруч та інше);
- повідомляє про ситуацію найближчу швидку допомогу, сімейного лікаря і лікарню (постачальника медичних послуг), куди швидка доставить пацієнта, а також родичів пацієнта (якщо він був один під час кризи хвороби). При цьому за час, коли швидка допомога буде у зворотній дорозі, ПСМЗ пацієнта буде автоматично повторно переконфігурована на безперервну відправку життєве важливих даних у лікарню, куди везуть пацієнта.

Цей сценарій показує, що скоординована стратегія електронної охорони здоров'я, використовуючи системи моніторингу особистого здоров'я, може допомогти реалізації адресних пріоритетів ЦГЗ у поліпшенні доступу до охорони здоров'я, профілактиці хронічної хвороби та моніторингу і догляду за літніми людьми зі складними умовами. Сприяючи догляду на дому для літніми

пацієнтами з хронічними захворюваннями, вона також може додати позитивний внесок у зменшення часу очікування в лікарнях.

5.10.5 Використання онтології в проекті

Використання онтологій у медицині в основному зв’язано з представленням та (ре)організацією медичної термінології. Лікарі розробили власні спеціалізовані мови та лексики, які допомагають їм ефективно зберігати загальнодоступні медичні знання та інформацію про пацієнта. Така термінологія, оптимізована для обробки людиною, характеризується значною кількістю неявних знань. Медичні інформаційні системи, з іншого боку, повинні мати можливість однозначно повідомляти про складні та детальні медичні концепції (можливо, виражені різними мовами). Це, очевидно, складне завдання і вимагає глибокого аналізу структури та концепцій медичної термінології. Але це може бути досягнуто шляхом побудови онтологій медичних доменів для представлення медичних термінологічних систем. При цьому онтології можуть [208-212]:

- допомогти у створенні більш потужних та більш сумісних інформаційних систем у сфері охорони здоров'я;
- підтримувати необхідні процедури передачі, повторного використання та обміну даними пацієнта;
- надавати семантичні критерії для підтримки різних статистичних агрегацій даних для різних цілей;
- підтримувати необхідну інтеграцію знань та даних у предметній галузі.

Окрім застосування онтології для вибору початкової моделі CIM при проектуванні архітектури прикладного додатку і отримання його вихідного коду, як то було показано вище, передбачається розробити розподілені сховища сервісів, пов'язаних із процесами збору та зберігання даних з усіх джерел даних, і розробити онтологію цих сервісів. Ці сховища включатимуть такі спеціальні можливості:

- онтолого-базовані сервіси для забору даних з різних портативних носіїв та сенсорів:

- сервіси, що базуються на онтології, для управління даними PHR, клінічними записами, даними медичних зображень тощо;
- онтологічні сервіси для збору інформації про поведінку пацієнтів;
- сервіси ідентифікації ризиків пацієнтів, які страждають на захворювання, або стан яких серйозно погіршився, шляхом постійного моніторингу пацієнтів та персоналізації медичної допомоги.

Можна, користуючись онтологічним підходом, також об'єднати дані різних СЦЗ (рис. 5.10) у єдине віртуальне сховище даних, якщо побудувати спільну антологію [212].

Рис. 5.10. Структура онтологій для семантичної інтеграції інформації

Це дуже дієвий засіб інтеграції різних додатків і інформаційних систем у випадках, коли ці системи (додатки) будувалися відокремлено. Звичайно, що така запропонована перебудова системи охорони здоров'я в Україні вимагає значних сил і коштів і має реалізовуватися на державному рівні. У силах дисертанта було дослідити методологію і сформувати рекомендації до онтолого-базованої сервісної реалізації її базових компонентів, а також впевнитися у їх принциповій здійсненності на прикладах вирішення окремих невеликих завдань [209].

5.10.6 Порівняння із соціальною медичною платформою Wiki-Health

Немає у світі країни, в якій би сьогодні не розроблялась чи вже не функціонувала система m-Health, починаючи з Китаю [216] і Кореї [214] і закінчуючи Катаром [2013] і Сербією [212]. Із існуючої множини виділимо *сервісну платформу* під назвою *Wiki-Health (Англія*), яка використовує хмарні

обчислення та Інтернет речей для контролю особистих медичних станів [215]. Wiki-Health – це соціальна платформа для обслуговування спільнот громадян у сферах здоров'я, фітнесу та благополуччя, що базується на даних і контексті. Wiki-Health пропонує нові засоби зберігання, позначення, отримання, аналізу, порівняння та пошуку даних від датчиків здоров'я. Це робить знання, пов'язані із здоров'ям, доступними для окремих користувачів у масовому масштабі. Wiki-Health ґрунтується на концепції «вікі» та масового співробітництва, згідно з якою агрегований користувальницький контент формується спільно, створюючи небачені обсяги знань у багатоперспективний та соціально привабливий спосіб, рис. 5.111.

Одне із ключових дослідницьких завдань для WikiHealth полягає у тому, щоб розробити нові алгоритми для пошуку необхідних даних у великих масивах даних з мобільних приладів та інших джерел медичної інформації. Дані часових рядів, що надходять від датчиків у реальному часі, оброблюються для визначення спільних рис захворювань у різних людей та виявлення спільнот, у межах яких пацієнти могли б скористатися обміном інформацією або порівнювати наслідки конкретних видів лікування.

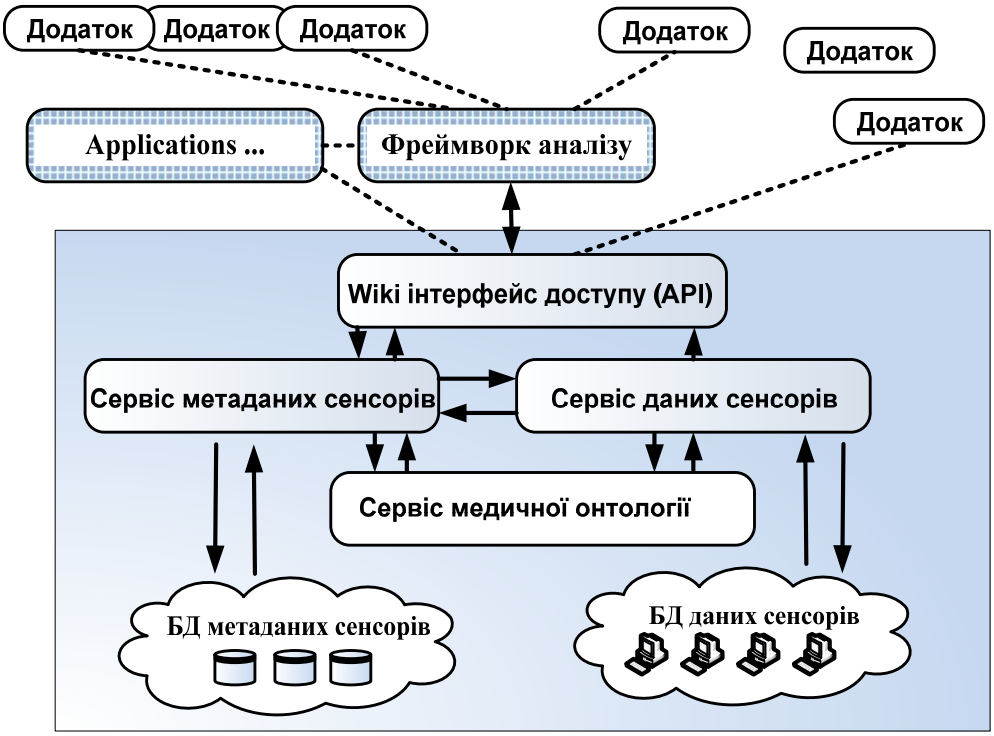


Рис. 5.11. Архітектура Wiki-Health

Наприклад, цікавою роботою може бути моделювання моделей сну різних людей для різних параметрів, таких як їх стилі життя, активність, вплив навколишнього середовища тощо.

SOA концепція створює принципову можливість для налагодження співпраці і обміну даними між національними електронними системами охорони здоров'я.

Висновки до розділу 5

1. При моделюванні системи сервісів як бізнес-процесів слід розглядати три аспекти: об'єкти, з якими оперує бізнес; процеси, які він виконує; та події, шо керують змінами процесів і об'єктів. Усі три типи моделювання об'єднуються в єдиній методології проектування систем сервісів, запропонованій в роботі на базі сумісного використання онтолого-орієнтованого (ОО) і ***модельно-керованого (MD) підходів.*** При цьому ***добре визначена доменна онтологія може бути безпосередньо перетворена на код додатк***у, на схему баз даних, у абстрактні інтерфейси користувача та інші. Якщо при традиційному проектування відповідальними виконавцями є ***розробники ПЗ***, що більшу частину часу працюють з кодом, то при запропонованому підході відповідальними виконавцями є ***архітектори ПЗ***, які більшу частину часу працюють з онтологіями і моделями спільно з експертами в предметній галузі.

2. На етапі аналізу ТЗ (технічного завдання) запропоновано проводити переважно аналіз домену розробниками додатків і існуючих репозитаріїв сервісів, які відповідають бізнес / технічним характеристикам системи, замість традиційного формування системними аналітиками вимог до системи з ціллю їх перетворення в технічні характеристики для наступного дослідження розробниками / дизайнерами.

3. На етапі проектування запропоновано проводити вибір або побудову онтології проектованого додатку і концептуальних моделей, представлених у формі, яка дозволяють розробникам системи сервісів, перетворюючи мову опису, автоматично перевести їх в виконуваний код (CIM-> PIM -> PSM-> UML). Це

суттєво відрізняється від того, як сьогодні виконується вибір будівельних блоків і уточнення структури класів вживаних функцій, проектування архітектури системи (структури її компонентів і модулів), при чому наступне оновлення їх з урахуванням технічних змін є досить складним завданням.

4. На етапі реалізації запропоновано здійснювати пошук і композицію сервісів з існуючих репозитаріїв, розгортати їх в контейнерах і організовувати їх взаємодії в залежності від вибраного шаблону (запит-відповідь, публікація-підписка, їх гібриди) на відміну від прийнятого . кодування на конкретній мові програмування (Java, C ++, C #) в межах однієї команди, яка створює функції, класи, модулі.

5. Запропонований підхід до проектування суттєво спрощує також етап тестування і підтримки, оскільки сервіси можуть бути перевірені ще до їх реєстрації та використання постачальниками сервісів, брокером і клієнтом, практично без взаємодії між собою. В існуючій методиці проектування тестування, зазвичай, проводиться в тій же проектній організації на основі вихідного коду і функціональних специфікацій, а тестові сценарії визначаються розробниками / тестувальниками.

6. Таким чином, програмні проекти визначаються не тільки вимогами і моделями, але й онтологіями (або за допомогою онтологій), які утворюють базу знань для предметної області і розділяються багатьма проектами. Тому отримані крос-проектні рішення обслуговують кілька доменів і призначені для повторного довгострокового спільного використання багатьма проектантами і установами. Проектно-залежні рішення, які отримуються при традиційному проектуванні, в основному не використовуються повторно і мають відносно коротко-строкове ізольоване застосування в обраному домені.

7. Використання онтології дозволяє здійснювати ефективну інтеграцію різних додатків й інформаційних систем навіть у випадках, коли ці системи (додатки) будувалися відокремлено, а також генерувати API системи (також шляхом перетворення мов опису).

8. Запропоновані у рамках національної медичної реформи охорони здоров’я структура хмарної платформи і перелік сервісів для мобільної медичної системи моніторингу стану і допомоги літнім хронічно хворим людям, які базується на результатах поведених досліджень.

9. Зараз настав час переходу на нову парадигму програмування, пов'язану ні з об'єктами, а з бізнес-процесами і їх складовою частиною - бізнес-функціями. Її ідея - компоновка додатків шляхом виявлення і виклику сервісів, доступних в мережі, для виконання певної задачі. В результаті можна визначити SOA (MSA) наступним чином: це архітектура додатків, побудована на основі формалізованих бізнес-процесів, функції яких представлені у вигляді багаторазово використовуваних сервісів з прозорими описаними інтерфейсами.

ВИСНОВКИ

Орієнтація світової економіки на індустрію сервісів, поява міждисциплінарної науки про сервіси, поширення сервісних підходів на технічні системи (зокрема, структури програмного забезпечення) мотивують фахівців компютерних наук, математиків, програмістів, економістів, соціологів і законодавців співпрацювати для досягнення дуже важливої мети: аналізу, побудови, управління та розвитку складних систем сервісів. Головне завдання полягає в тому, щоб виявити логіку складних систем сервісів і впровадити загальну методологію їх моделювання і загальні рамки для інновацій сервісів.

Мета цієї дисертаційної роботи: теоретичне обгрунтування, розробка та розвиток сервіс-орієнтованої архітектури і моделей систем сервісів для вирішення завдання пошуку веб-сервісів сервісів в репозитариях, композиції та оркестровки сервісів і побудови їх робочих потоків (workflows) для підтримки певних бізнес-процесів в хмарному середовищі.

В дисертації вперше отримані такі нові наукові результати:

1. Проведено аналіз сучасного стану сервісних технологій, базованих на композиції веб-сервісів з уніфікованими протоколами зв’язку (перше покоління SOA) і на поєднанні мікросервісів з контейнерами (друге покоління MSA) та показано, що вони поділяють спільні принципи побудови і їх властивості підлягають конвергенції, оскільки кожен з них має свої переваги і недоліки. При цьому хмарне середовище і вимоги до віртуалізації його апаратних і системних ресурсів сприяють переходу до контейнерного виконання веб-сервісів і їх взаємодії через API замість традиційних уніфікованих протоколів зв’язку, зменшенню розмірів самих веб-сервісів для забезпечення можливості розгортання кількох з них в одному контейнері, що призводить до економії апаратних ресурсів (кількості задіяних віртуальних машин ВМ).

2. Досліджені можливі архітектури мікросервісних систем і підходи, як саме реалізувати для MSA всі можливі архітектури (сервіс-орієнтовану з синхронними зв’язками між сервісами, подійно-керовану EDA; гібридну сервіс-орієнтовану, керовану подіями EDSOA; агентно-керовану), які відомі з практики

SOA першого покоління. Показано, що для MSA базовою є подійно-керована модель, в той час як для SOA – модель з синхронними зв'язками між сервісами. При цьому застосування SOA є ефективним для інтеграції додатків і інформаційних систем, MSA ефективно діє на рівні окремого додатка, фокусуючись на його структурі і компонентах.

3. Вперше на базі узагальнення існуючих методів семантичного відкриття сервісів запропоновано будувати сервіс пошуку сервісів у вигляді гібридної системи шляхом попереднього звуження за категорією сервісу області пошуку і відсіювання непричетних до запиту сервісів (тобто виділення відповідного кластера), і наступної оцінки відповідності функціональних вимог (входів та виходів, передумов та ефектів) і нефункціональних вимог до сервісу (контекстних, мовної схожості, категорій сервісів, якості сервісів QoS), взятих з запиту користувача. При цьому така оцінка завдяки базовій подійно-керованій архітектурі MSA виконується паралельно, а не поодинці чи роздільно, як то реалізовано сьогодні. Це призводить до скорочення часу пошуку сервісів і робить можливим застосування семантичних описів мікросервісів і семантичних методів їх відкриття в умовам стрімкого зростання розмірів хмарних реєстрів таких сервісів.

4. Удосконалено методику розробки систем сервісів, яка базується на об'єднанні онтолого-орієнтованого (OD) та модельно-керованого (MD) підходів до моделювання архітектури бізнес-процесів, включенням в неї процесу перетворення бізнес-логіки у логіку додатків. Цей процес супроводжується перетворенням онтології бізнес-процесів у послідовні моделі (CIM, PIM, PSM) з поступовим збільшенням впливу параметрів сервісної платформи, яка застосовується для реалізації системи сервісів.

5. Сформульовані рекомендації з вибору сервіс-орієнтованої архітектури і самих сервісів для мобільної медичної системи моніторингу стану і допомоги літнім хронічно хворим людям в рамках реалізації національної програми перебудови системи охорони здоров'я в Україні. Запропонований репозитарій

сервісів має універсальний характер (особливо в області сервісів управління платформою і багатьох сервісів додатків лікаря і пацієнта), тому на його базі можна з мінімальними витратами розгортати мобільні системи різного призначення і адаптувати їх до завдань підтримки конкретних планів лікування хворих шляхом масштабування сервісів: виключення деяких з них, додавання нових, заміни одних на інші такого ж призначення.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. SOA terminology overview, Part 1: Service, architecture, governance, and business terms", https://www.ibm.com/developerworks/library/ws-soa-term1/
2. "Succeedin g through Service Innovation", Ifm.eng.cam, University of Cambridge, July 2007,https://www.ifm.eng.cam.ac.uk/uploads/Resources/Reports/080428cambridge_ssme_whitepaper.pdf
3. Haluk Demirkan. "Service-oriented technology and management: Perspectives on research and practice for the coming decade" / H.Demirkan, R. J. Kauffman, J. A. Vayghan, Hans-Georg Fill, D. Karagiannis, P. P. Maglio // Electronic Commerce Research and Applications, ELSERVIER, 7, 2008, pp. 356–376
4. Stauss B., Engelmann K. Kremer A., Luhn A. (Eds.). "Services Science: Fundamentals, Challenges and Future Developments" //Springer, Berlin, 2007
5. Fitzsimmons J., F Fitzsimmons M. "Service management: Operations, strategy in formation technology" //NewYork: McGraw-Hill., Heskett, JL, 6th edition, TO 2007.
6. Pinhanez C and Murphy W. "Service Science Handbook" // Springer, NewYork, 2011.
7. P. Maglio, C. A: Kieliszewski, J Spohrer. "Handbook of Service Science (Service Science: Research and Innovations in the Service Economy)" //Springer, NewYork, 2010.
8. H Demirkan, J Spohrer, V Krishna. "Service Systems Implementation (Service Science: Research and Innovations in the Service Economy)" // Springer, NewYork, 2011.
9. "A service perspective for education, research, business and government", ISBN: 978-1-902546-65-0 //University of Cambridge Institute for Manufacturing (IfM) and International Business Machines Corporation (IBM), April 2008, 30 p.
10. IBM Corporation, "Service-oriented architecture, special issue", IBM Systems Journal 2005,44(4), www.ibm.research.com/journal/sj44-4.html
11. Vitharana P. "Service-oriented enterprises and architectures: state of the art and research opportunities" / Vitharana P, Bhaskaran K, Jain H, Wang H, Zhao L. // In: Hoxmeier J, Hayne S. editors. Proceedings of the Americas Conference on Information Systems, Keystone, CO, August 9–12, 2007.
12. Sharp A., McDermott P. "Workflow Modelling. Tools for Process Improvement and Application Development" // Artech House, 2001.
13. "Workflow Management Coalition. Terminology and Glossary English", WfMC-TC-1011, v3, Technical Report, April 2011, //http://www.wfmc.org/Glossaries-FAQs/
14. "Service Science, Management, and Engineering (SSME)". Курс лекций. IBM Almaden Services Research, IBM, 2006 – 2007, https://www.ibm.com/ developerworks/ wikis/display/ssme/Introductory+modules
15. Thomas Erl. "Service-Oriented Architecture, Concepts, Technology, and Design", Prentice Hall, ISBN : 0-13-185858-0, 2015
16. ***DevOps: How to utilize it in your IT workspace***?, Media.Techtarget, June 2017, http://media.techtarget.com/facebook/downloads/DevOps-G6FE462376. pdf
17. "International Conference on Exploring Service Science", Amazon.com? Amazon, 4-6 February 2015, Porto, http://www.amazon.com/Exploring-Services -Science-International-Proceedings/dp/3319149792
18. Петренко А. А. "Объекты и методы науки о сервисах" // Системні дослідження і інформаційні технології.-Київ, №2, 2015.

19. Кисельов Г.Д. “Наука про сервіси, менеджмент та інжиніринг як основа інноваційної діяльності” /Кисельов Г.Д., Петренко О.О. // Вісті університету розвитку людини «Україна». Вип.. , 2015.
20. Петренко О.О. **“**Підготовка кадрів для індустрії сервісів**”** // Інформаційні технології в освіті.-Херсон, випуск 23, 2015.
21. Петренко І.А. “Автоматизовані методи пошуку і відкриття необхідних сервісів” / Петренко І.А., Петренко О.О. // Вісник Університету «Україна», Серія «Інформатика, обчислювальна техніка та кібернетика», №1(17), 2015, С. 55-64.
22. Hui Peng, Zhongzhi Shi, Lirong Qiu. “Matching Request Profile and Service Profile for Semantic Web Service Discovery”, Research gate, https://www.researchgate.net/publication/239813737_Matching_Request_Profile_and_Service_Profile_for_Semantic_Web_Service_Discovery
23. **“**Semantic Annotations for WSDL and XML Schema”, w3.org, 2002, https://www.w3.org/2002/ws/ sawsdl/spec/examples/
24. “OASIS Web Services Business Process Execution Language**”**, Oasis-open.org, OASIS, 2008, http://www.oasis-open.org/committees/ tc_home. php?wg_abbrev=wsbpel
25. Recker J. *“*Business process modeling: A comparative analysis?”/ Recker J.C., Rosemann M., Indulska M., Green P. // Journal of the Association for Information Systems, 10(4): pp. 333-363.
26. Mili H. “Business Process Modelling Languages: Sorting Through the Alphabet Soup” / Mili H., Guy T., Guitta B., Jaoude G.B., Lefebvre É., Elabed L., ElBoussaidi G **.**// ACM Computing Surveys, 43(1), Article 4,2010.
27. **“**Patterns: Service Oriented Architecture and Web Services”, ibm.com, IBM, Junusary 2004, https://www.researchgate.net/publication/200167132_IBM_ Patterns_service-oriented_architecture_and_web_services
28. **“**ESB (Enterprise service bus)”, voxxed.com**,** Voxxed, 2015**,** https://www.voxxed.com/blog/2015/01/good-microservices-architectures-death-enterprise-service-bus-part-one/
29. “Speaking on Advanced Software Architecture”, service-architecture.com, Service Architecture, https://www.service-architecture.com/speaking.html
30. “Open ESB. The Open Source ESB for SOA Integration”**,** OpenESB community, 2008, https://open-esb.dev. java.net
31. “Apach eSoftware Foundation. Service Mix*”*, Apach Software Foundation, 2008. - http://servicemix.apache. org/home.html
32. “Web Sphere Enterprise Service Bus”, ibm.com, IBM, 2008, http://www-01.ibm.com/software/ integration/ wsesb/
33. “Celtix: The Open Source Java Enterprise Service Bus**”**, Objectweb.org, Object Web, 2008, http://celtix.objectweb.org
34. Петренко, О.О. “Порівняння типів архітектури систем сервісів” // Системні дослідження і інформаційні технології - № 4.2015. - С. 48-62.
35. Chen, Y., & Tsai, W. T. “Distributed service-oriented software development “ , Iowa, Kendall/Hunt Publishing, 2008, 465 p., http://citeseerx.ist.psu.edu/viewdoc/download?doi= 10.1.1.459.6427&rep=rep1&type=pdf
36. Нгуен Ба Нгок. “Модель информационного поиска на основе семантических метаописаний” / Нгуен Ба Нгок, Тузовский А. Ф**.** //Управление большими системами. Выпуск 41, 2014, С.51-92.
37. Jürgen Kress. “Oracle Industrial SOA” / Jürgen Kress, Berthold Maier, Hajo Normann, Danilo Schmeidel, Guido Schmutz, Bernd Trops, Clemens Utschig-Utschig, Torsten

Winterberg; oracle.com, Oracle, 2016, http://www. oracle.com/technetwork/articles/soa/ind-soa-toc-1934143.html
38. Tiziana Ferrari. “EGI towards the European Open Science Cloud”, egi.eu, EGI, 2017, https://indico.egi.eu/indico/event/3249/session/24/contribution/10
39. Iulia Popescu. “What is the European Open Science Cloud?”, Inspired(EGI),Issue #28, September 2017, pp.2-4.
40. Ньюмен С. “Создание микросервисов” // СПб., Питер, 2016, 304 с. (Серия «Бестселлеры O’Reilly»). ISBN 978-5-496-02011-4
41. Петренко О.О. “Особливості реалізації сервіс-орієнтованих додатків у хмарі” // Системні дослідження і інформаційні технології - № 3.2017, С.29-38
42. Выростков Д. “Обзор способов и протоколов аутентификации в веб-приложениях”, habrahabr.ru, Хабрахарб,16 July 2015, https://habrahabr.ru/company/ dataart/blog/262817/
43. Бондаренко В. “Три лучших инструмента для описания RESTful API”, habrahabr.ru, Хабрахарб,3 March 2015, https://habrahabr.ru/post/252237/
44. Wilson Mar. “API Management Evaluation”, github.io, https://wilsonmar. github.io/api-management-evaluation/
45. “Picking The Right Cloud Container Platform”, TechTarget, 2017, http://www.communicationstoday.co.in/images/reports/20170501-container-deployment-g6gc442244-report.pdfKai
46. “Controlling software through containers and microservices”, sdtimes.com, Software Development, http://sdtimes.com/controlling-software-containers-microservices/ #sthash.rTBFTNw1.dpuf
47. “Docker - Build, Ship, and Run Any App, Anywhere”, docker.com, Docker, https://www.docker.com/
48. “Containers as a Service: Comparing Providers and Evaluating the State of the Market”, sandhill.com, Sand Hill, April 2015, http://sandhill.com/article/containers-as-a-service-comparing-providers-and-evaluating-the-state-of-the-market/
49. “The Shortlist of Docker Hosting”, codeship.com, CODESHIP, October 2016, https://blog.codeship.com/the-shortlist-of-docker-hosting/
50. “Repositories on Docker Hub”, docker.com, Docker, https://docs.docker.com/ docker-hub/repos/
51. ” Operating System Containers vs. Application Containers”, risingstack.com, Resing Stack, https://blog.risingstack.com/operating-system-containers-vs-application-containers/
52. “Rise of the container-focused operating systems”, thenewstack.io, THENEWSTACK, https://thenewstack.io/docker-fuels-rethinking-operating-system/
53. “Swarm mode overview”, docker.com, Docker, https://docs.docker.com/engine/ swarm/
54. “Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications”, kubernetes.io, https://kubernetes.io/
55. Wähner. “A Good Microservices Architecture = Death of the Enterprise Service Bus (ESB)?”, dzone.com, Dzone, 2017, https://dzone.com/articles/good-microservices
56. “Realising the European Open Science Cloud”, europa.eu, EC, https://ec. europa. eu/research/openscience/pdf/realising_the_european_open_science_cloud_2016.pdf

57. “Fi-Ware”, fiware.org, FIWARE, https://www.fiware.org/developers-entrepreneurs/
58. “Online EGI Service Catalogue”, egi.eu< EGI, https://www.egi.eu/services/
59. “INDIGO services”, indigo-datacloud.eu, INDIGO, https://www.indigo-datacloud.eu/service-component
60. “EUDAT site”, eudat.eu, EUDAT, www.eudat.eu
61. “Riding the Wave report”, google.com, Google, https://www.google.com.ua/?gws_rd=ssl#q=riding+the+wave+report
62. “EUDAT services”, eudat.eu, EUDAT, https://eudat.eu/services-support)
63. “GEANT Site”, geant.org, GEANT, https://clouds.geant.org/
64. “Amazon Web Services’, google.com, Google, https://www.google.com.ua/?gws_rd=ssl#q=riding+the+wave+report
65. ”European E-Infrastructure Services Gateway”, efiscentre.eu, EFIS, http://www.efiscentre.eu/portfolio-item/european_e-infrastructure-services-gateway/
66. “FIWARE”, fiware.org , https://catalogue.fiware.org/
67. “ .NET Framework”, .wikipedia.org, https://en.wikipedia.org/wiki/.NET_Framework
68. “Web Services as a Basis for SOA”, sap.com, SAP, https://blogs.sap.com/ 2014/04/02/sap-and-soa-the-business-process-platform/
69. “Oracle SOA”, oracle.com, ORACLE, http://www.oracle.com/technetwork/articles/soa/ind-soa-toc-1934143.html
70. “Web Services as a Basis for SOA”, sap.com, , SAP, https://blogs.sap.com/2014/04/02/sap-and-soa-the-business-process-platform/
71. “Production-Grade Container Orchestration”, kubernetes.io, Kubernetes, https://kubernetes.io/
72. Service Oriented Architecture : SOA Features and Benefits”, opengroup. org, OpenGroup, https://www.opengroup. org/soa/source-book/soa/soa_features.htm
73. Петренко, О.О. “Порівняння типів архітектури систем сервісів” // Системні дослідження і інформаційні технології - № 4.2015. - С. 48-62.
74. “Почему стоит изучить Clojure?”, habrahabr.ru , Хабрахабр, 2013, https://habrahabr.ru/post/173071/
75. “The Open Source ESB for SOA & Integration”, 2008, https://open-esb. dev.java. net
76. “ServiceMix”, apache. org, Apache Software Foundation 2008, http://servicemix. apache.org/home.html
77. “JBossESB”, jboss.org, JBoss, 2008, http://www.jboss.org/community/docs/ DOC-10326
78. “WebSphere Enterprise Service Bus”, ibm.com, IBM, 2008, http://www01.ibm. com/software/integration/wsesb/
79. Jeff Hanson. “Design an event-driven and service-oriented platform with Mule”, javaworld.com, http://www. javaworld.com/article/2072262/soa/event-driven-services-in-soa.html
80. Chris Richardson. “Pattern: Microservice Architecture”, microservices.io, Microservice Architecture, http://microservices.io/patterns/microservices.html
81. What is Microservices Architecture?: https://smartbear.com/learn/api-design/what-are-microservices/

82. Andrew Bonham. “Microservices—When to React Vs. Orchestrate”, medium.com, https://medium.com/capital-one-developers/microservices-when-to-react-vs-orchestrate-c6b18308a14c
83. Kasun Indrasiri. “Microservices in Practice - Key Architectural Concepts of an MSA”, wso2.com, WSO2, http://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/
84. “Pattern: Command Query Responsibility Segregation (CQRS)”, microservices.io, http://microservices.io/patterns/data/cqrs.html
85. “Microservices: “Decomposing Applications for Deployability and Scalability”, infoq.com, InfoQ,2014, https://www.infoq.com/articles/microservices-intro
86. Jean-Louis Marechaux. “Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus”, ibm.com, IBM, 2006, http://www.ibm.com/developerworks/library/ws-soa-eda-esb/
87. Черняк Л. “ EDA как очередная инкарнация SOA”, ecm-journal.ru, 2007, http://ecm-journal.ru/post/EDA-kak-ocherednaja-inkarnacija-SOA.aspx
88. R. Zicari “Advancing soa with an event-driven architecture”, odbms.org, 2011, http://www.odbms.org/2011/01/advancing-soa-with-an-event-driven-architecture/
89. “Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus (IBM)”, ibm.com, IBM, - http://www.ibm.com/developerworks/library/ws-soa-eda-esb/
90. Krissi Danielson. ‘Mixing Event Driven Computing, SOA, BPM and BI for Instant Responsiveness”, ebizq.net, ebiz, 20097, http://www.ebizq.net/blogs/firstlook/2007/10/post_17.php
91. Guido Schmutz. “Event-Driven SOA: Events Meet Services”, oracle.com, Oracle, 2011, http://www.oracle.com/technetwork/articles/soa/schmutz-soa-eda-405955.html
92. Roy W. Shulte. “Clarifying the Terms ‘Event-Driven’ and ‘Service-Oriented’ Architecture”, gartner.com, Gartner, October 2007, http://www.gartner.com/DisplayDocument?doc_cd=126127&ref=g_fromdoc
93. Randy Heffner. “The Unification of SOA and EDA and More”, forrester.com, Forrester, http://www.forrester.com/Research/Document/Excerpt/ 0,7211, 35720, 00.html
94. “The world's most popular api tooling”, swagger.io, Swagger, https://swagger.io/
95. “Object.prototype.__proto__, mozilla.org, https://developer.mozilla.org/uk/docs/Web/JavaScript/Reference/Global_Objects/Object/proto
96. “Event-Driven Data Management for Microservices”, nginx.com, Nginx, https://www.nginx.com/blog/event-driven-data-management-microservices/
97. “Apache Kafka - a distributed streaming platform”, apache.org,2017, https://www.linkedin.com/pulse/apache-kafka-distributed-streaming-platform-kulwant-yadav/
98. “Conductor is a microservices orchestration engine”, github.io, Netflex, 2017, https://netflix.github. io/conductor/\
99. Kloos Reinhold. “Service Discovery with Semantic Characteristics: ACTAS”/ Kloos Reinhold, Rainer Unland, Cherif Branki // Services (SERVICES-1), 2010 6th World

Congress on DOI: 10.1109/SERVICES.2010.106 : http://ieeexplore.ieee.org /document/5575482/

100. “OCI (Open Container Initiative)”, opencontainers.org, https://www. opencontainers.org/

101. Martin Junghans and Sudhir Agarwal. “Efficient search for Web browsing recipes”/ Martin Junghans and Sudhir Agarwal // Proc. of IEEE International Conference on Web Services (ICWS), Santa Clara, CA, USA, June 27-July 2, 2013, IEEE Computer Society, 2013.

102. “LOG4SWS.KOM”/ S.Schulte, U.Lampe, J. Eckert, and R. Steinmetz // Self-Adapting S (SEASS ’10) at 2010 IEEE 6th World Congress on Services (SERVICES 2010), IEEE Computer Society, Washington, DC, USA, 2010, pp. 511–518.

103. Петренко А.І.“Основи сематричного Веб”/ Петренко А.І., Булах Б.В., Хондюл В. В. // Вид-во «Політехніка», 2012, 169 С.

104. “Workflow Management Coalition. Terminology and Glossary English”, WfMC-TC-1011 v3. TechnicalReport. Retrieved April, 2011, //http://www.wfmc.org/ Glossaries-FAQs/

105. “WSMO-Lite Service Model and Semantics Ontology”, w3.org, w3, 2010, http://www.w3.org/ Submission/2010/SUBM-WSMO-Lite-20100823/

106. Jacek Kopecký, “SAWSDL: Semantic Annotations for WSDL and XML Schema”/ J. Kopecký, T.Vitvar, C.Bournez, and J. Farrell. -// IEEE Internet Computing, 11(6), 2007, pp. 60–67.

107. Marian Babik. “Generating semantic descriptions of web and grid services” / 110. Marian Babik, Ladislav Hluchy,Jacek Kitowski, Bartosz Kryza; psu, 2004, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.7984&rep=rep1&type=pdf

108. Matthias Klusch. “A Hybrid Sematic Web Service Matchmaker”/ Matthias Klusch, F. Kaufer // in: Jiming Liu, Ning Zhong (eds.): International Journal of Web Intelligence and Agent Systems,volume 7, IOS Press, 2009, pp. 23-42.

109. B.Aishwarya. “Improving Web Service Discovery by Using Semantic Models”/ B. Aishwarya,Nayak, Richi and Bruza P. //. Proceedings 9th International Conference on Web Information Systems Engineering (WISE 2008*)* 5175/2008, Auckland, New Zealand, pp. 366-380.

110. Петренко О.О. Кластеризація зображень за їх анотаціями.- // Системний аналіз та інформаційні технології: 16-я міжнародна науково-технічна конференція «САІТ-2014», 26-30 травня 2014 року, Київ, Україна: матеріали. -– С. 305–307.

111. Tim Berners-Lee. “The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities”/ Tim Berners-Lee, James Hendler, and Ora Lassila. // Scientific American, May 2001, pp. 34–43.

112. Rajendran, T. “An Optimal Agent-Based Architecture for Dynamic Web Service Discovery with QoS” / Rajendran, T., Balasubramanie, P. //Second International Conference on Computing, Communication and Networking Technologies. IEEEXplore (2010)

113. Guo W.-Y. “Semantic web service discovery algorithm and its application on the intelligent automotive manufacturing system”/- Guo, W.-Y., Qu, H.-C., Chen, H.:// International Conference on Information Management and Engineering. IEEEXplore (2010)

114. Katia P. Sycara. “Automated discovery, interaction and composition of Semantic Web services”/ Katia P. Sycara, M. Paolucci, A. Ankolekar, and Naveen Srinivasan // Journal of Web Semantics, 1(1), 2003, pp. 27–46.

115. M. Klusch. “Retrieval Performance Evaluation of Matchmakers for Semantic Web Services (S3 Contest)”/ M. Klusch, A. Leger, D.Martin, M. Paolucci, A.Bernstein, and U. Küster // Third International Workshop SMR2 2009 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 8th International Semantic Web Conference (ISWC 2009), 2009

116. P. Dharanyadevi. “Proficient Discovery of Service in Event Driven Service Oriented Architecture”/-P. Dharanyadevi, P. Dhavachelvan, S.K.V. Jayakumar, R. Baskaran , and V.S.K. Venkatachalapathy, uottawa.ca, uOttawa, 2009, http://www.ruor.uottawa.ca/handle/10393/28086

117. Hui Peng. « Matching Request Profile and Service Profile for Semantic Web Service Discovery» / Hui Peng, Zhongzhi Shi, Lirong Qiu, researchgate.net, 2014, https://www.researchgate.net/profile/Zhongzhi_Shi/publication/239813737_Matching_Request_Profile_and_Service_Profile_for_Semantic_Web_Service_Discovery/links/542035900cf203f155c2c002/Matching-Request-Profile-and-Service-Profile-for-Semantic-Web-Service-Discovery.pdf

118. I. Pansa. “A Domain Ontology for Designing Management Services’ / I.Pansa, M Reichle, C . Leist, S. Abeck. // Proc. the Third International Conferences on Advanced Service Computing “SERVICE COMPUTATION”, 2011, pp.11-18.

119. S. Pakari. “Web service discovery methods and techniques: a review” / S.Pakari, E.Kheirkhah and M. Jalali //International Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol. 4, No.1, February 2014, 14 p.

120. Hui Peng. “Matching Request Profile and Service Profile for Semantic Web Service Discovery”/ H.Peng, Z.Shi, L.Qiu, aaai.org, 2007, https://www.aaai.org/Papers/Workshops/2007/WS-07-11/WS07-11-010.pdf

121. Alireza Zohali. “Matching model for semantic web services discovery”/ A. Zohali, K. Zamanifar. // Proceeding WAINA '09 Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops, May 26 - 29, 2009, pp. 50-54.

122. Shou-jian Yu. “Semantics Based Web Services Discovery”/ Shou-jian Yu, Jing-zhou Zhang, Xiao-kun Ge, Guo-wen Wu. / /Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications & Conference on Real-Time Computing Systems and Applications, PDPTA 2006, Las Vegas, Nevada, USA, June 26-29, 2006, v. 1, pp.156-159.

123. K.Sivashanmugam. “Adding Semantics to Web Services Standards”/ K. Sivashanmugam, K.Verma, A.Sheth, J.Miller; wright.edu, Wright State University, 2003, http://corescholar.libraries.wright.edu/knoesis/687

124. Priyadharshini. G. “A Survey on Semantic Web Service Discovery Methods”, Priyadharshini G., Gunasri R., Saravana B. B. //International Journal of Computer Applications (0975 – 8887), Volume 82 – No 11, November 2013, pp.8-11.

125. Петренко О.О. Семантичний пошук зображень за їх змістом.- // Системний аналіз та інформаційні технології: 16-я міжнародна науково-технічна конференція «САІТ-2014», 26-30 травня 2014 року, Київ, Україна: матеріали. - К.: ННК "ІПСА" НТУУ "КПІ", 2014. -– С. 312-313.

126. B.T. G. S. Kumara. “Improving web service clustering through post filtering to bootstrap the service discovery”/ B.T. G. S. Kumara, I. Paik, Koswatte R. C. Koswatte, W.i Chen. -//International Journal of Services Computing (ISSN 2330-4472),Vol. 2, No. 3, July – Sept.2014, pp.89-92/

127. Yu Yue Du. “A Semantic Approach of Service Clustering and Web Service Discovery”/ Yu Yue Du, Yong Jun Zhang and Xing Lin Zhang //Information Technology Journal 12 (5): 2013 ISSN 1812-5638, I DOI 10.3923/itj.2013.967.974, pp. 967-974.

128. Dong Shou. “Effective Web Service Retrieval Based on Clustering”/ Dong Shou and Chi-Hung Chi //Fourth International Conference on Semantics, Knowledge and Grid, 2008,pp. 469- 472.

129. Huiying Goa. ”Web Services Classification Based on Intelligent Clustering Techniques”/ Huiying Goa, Wolffried Stucky and Lei Liu //International Forum on Information Technology and Applications, Vol. 3, 2009, pp. 242-245.

130. Peng Liu. “Clustering-Based Semantic Web Service Matchmaking with Automated Knowledge Acquisition/ Peng Liu, Jingyu Zhang and Xuel Yu //International Conference on Web Information Systems and Mining, Vol. 5854, 2009, pp. 261-270.

131. Rong, W. “A Survey of Context Aware Web Service Discovery: From User’s Perspective”/ Rong, W., Liu, K. // Fifth IEEE International Symposium on Service Oriented System Engineering, 2010, pp. pp.15-22.

132. M. Barhamgi. “A queryre writing approach for web service composition”/ M. Barhamgi, D. Benslimane, and B. Medjahed // IEEE Trans. Services Comput.,vol. 3, no. 3, Jul.-Sep. 2010, pp. 206–222.

133. Matthias Klusch. “Hybrid Adaptive Web Service Selection with SAWSDL-MX and WSDL-Analyzer”/ Matthias Klusch, Patrick Kapahnke, Ingo Zinnikus //Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications, 2009. Pp. 550-564.

134. Massimo Paolucci. “Semantic Matching of Web Services Capabilities”/ Massimo Paolucci and Takahiro Kawamura and Terry R. Payne and
Katia Sycara // I. Horrocks and J. Hendler (Eds.): ISWC 2002, LNCS 2342, 2002, Springer-Verlag Berlin Heidelberg 2002133, pp. 333–347.

135. Крюков К.В. “Меры семантической близости в онтологиях”/ Крюков К.В., Панкова Л.А., Пронина В.А., Шипилина Л.Б. // Проблемы управления. – 2010. – №2. – С. 2–14.
136. Нгуен Ба Нгок. “ Модель информационного поиска на основе семантических метаописаний” Нгуен Ба Нгок, Тузовский А. Ф. //Управление большими системами. Выпуск 41.-2014.- С.51-92.
137. Rada R. “Development and Application of a Metric on Semantic Nets “/ R. Rada, H. Mili, E.Bicknell, and M. Blettner // IEEE Trans. on Systems, Man and Cybernetics. 1989. V. 19, pp.17-30.
138. Haase P., Siebes R., “Harmelen F. Peer selection in peer-to-peer networks with semantic topologies” / Haase P., Siebes R. // Proc. of Int. Conf. on Semantics in a Networked World, Paris, 2004, pp. 108–125.
139. Черний А.В. “ Развитие информационной системы организации с использованием семантических технологий” / Черний А.В., Тузовский А.Ф. // Материалы Всерос. конф. с междунар. участи- ем «Знания – Онтологии – Теория». – Новосибирск: ЗАО «РИЦ Прайс-Курьер», 2009. – Т.2. – С. 52–59.
140. Haase P. “ Peer selection in peer-to-peer networks with semantic topologies”/ Haase P., Siebes R., Harmelen F. // Proc. of Int. Conf. on Semantics in a Networked World, Paris, 2004, pp.108–125.
141. Лукашевич Н.В. “Тезаурус русского языка для автоматической обработки больших текстовых коллекций”/ Лукашевич Н.В., Добров Б.В. // Компьютерная лингвистика и интеллектуальные технологии: Труды Международного семинара Диа- лог'2002 / Под ред. А.С. Нариньяни. – М.: Наука, 2002. – Т. 2. – С. 338–346.
142. A. Heand. “ ASSAM: A tool for semi- automatically annotating semantic web services” / A. Heand E. Johnstonand N. Kushmerick //Proc.of the 3rd International Semantic Web Conference, 2004, Springer
143. Ehrig M. ”Ontology mapping – an integrated approach”/ Ehrig M., Sure Y. // The Semantic Web: Research and Applications. Proc. 1st European Semantic Web Symposium, Berlin, Springer, Vol. 3053, pp. 76–91.
144. Andreasen T. “Domainspecific similarity and retrieval”/ Andreasen T., Knappe R., Bulskov H**.** // 11th Int. Fuzzy Systems Association World Congress,Vol. 1, pp. 496–502.
145. Bondy J.A. “Graph Theory” / Bondy J.A., Murty U.S.R. // N.Y.: Springer, 2008, 651 p.
146. Christopher D.M. “ Introduction to information retrieval” / Christopher D.M., Prabhakar R., Hinrich S. // N. Y.: Cambridge University Press, 2008, 482 p.
147. Z. Wu. “Verbs semantics and lexical selection” / Z. Wu and M. Palmer // Proc. of the 32nd annual meeting on Association for Computational Linguistics, Association for Computational Linguistics, 1994, June, pp. 133-138.
148. P. Resnik. “Using information content to evaluate semantic similarity” // Proc. of the Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies Conference on Artificial Intelligence, August 20-25; Montréal Québec, Canada, 1995, pp. 448-453.

149. D. Sánchez. “Ontology based semantic similarity: A new feature-based approach” / D. Sánchez, M. Batet, D. Isern, and A. Valls // Expert Systems with Applications, 39(9), 2012, pp. 7718-7728.
150. Jiang J. “Semantic similarity based on corpus statistics and lexical taxonomy” / Jiang J., Conrath D. // Proc. Int. Conf. on Computational Linguistics, Taiwan, 1997, pp. 19–33.
151. Bulskov H. “ Measuring similarity for conceptual querying”/ Bulskov H., Knapp . R., Andreasen T. // Proc. 5th Int. FQAS Conf. LNCS, Berlin, Springer, 2002, Vol. 2522, pp. 100–111.
152. Ю.В. Рогушина. “Онтологическая модель интеллектуализации сервис-ориентированных вычислений в распределенной среде Интернет” / Ю.В. Рогушина, А.Я. Гладун. // Проблеми програмування. 2006 № 2-3. Спеціальний випуск, стр.526-536
153. Knappe R. “Measures of semantic similarity and relatedness for use in ontology-based information retrieval”. PhD thesis. – Roskilde University, 2006. – 143 p.
154. D. Guessoum. “Survey of semantic similarity measures in pervasive computing”/. D. Guessoum, M. Miraoui, and C. Tadj // International journal on smart sensing and intelligent systems, 8(1), 2015,pp.125-158.
155. Castano S. “Semantic information interoperability in open networked systems”/ Castano S., Ferrara A., Montanelli S., Racca G. // Proc. of the Int. Conf. SNW,Paris, 2004,pp. 215–230.
156. Тузовский А.Ф. “Онтолого-семантические модели в корпоративных системах управления знаниями: дис. докт. техн. Наук”,Томск, 2007, 175 С.
157. Губин М.Ю. Методы создания семантических метаописаний документов с применением семантических сетей, фреймовых моделей и частотных характеристик / Губин М.Ю., Разин В.В., Тузовский А.Ф. // Доклады Томского государственного университета систем управления и радиоэлектроники, ,Т. 2, №2, 2010, С. 227–229.
158. Seheon Songa. “A goal-driven approach for adaptive service composition using planning”/.Seheon Songa, Seok-Won Lee // Mathematical and Computer Modelling, Volume 58, Issues 1–2, July 2013, pp.Pages 261-273.
159. Bart Orriens. ‘Model Driven Service Composition”/ Bart Orriens, Jian Yang and Mike P Papazoglou // ICSOC Springer-Verlag, 2003, pp. 75-80.
160. I Budak Arpinar. “Ontology-Driven Web Services Composition Platform”/ I Budak Arpinar, Ruoyan Zhang, Boanerges Aleman-meza and Angela Maduko // Information Systems and e-Business Management, vol. 3, 2004, pp. 6-12.
161. Freddy Lecue. “Semantic Web Service Composition Based on a Closed World Assumption”/ Freddy Lecue and Alain Leger //Proc. Of IEEE 4th European Conference on Web Services, 2006, pp. 233-239.
162. Alain Leger. “Semantic Web Service Composition through a Matchmaking of Domain”/ Alain Leger and Freddy Lecue // IEEE 4th European Conference on Web Services, 2006, pp. 171-178.

163. Guliherme C. “ of Semantic Web Services Compositions Based on SAWSDL Annotations”/ Guliherme C. Hobold and Frank Siqueira // IEEE 19th International Conference on Web Services, 2012, pp. 280.
164. D. S. Pukhkaiev. “Aadvanced approach to web service discovery and selection”/ D. S. Pukhkaiev, O. Oleksenko, T. M. Kot, L. S. Globa, A. Schill // Information and Telecommunication Sciences.- V.5, N 1. 2014.- 42-47.
165. Mohan H. G.“A Survey on Semantic Based Automatic Web Service Compositions/ Mohan H G and Devaraj F V. // European Journal of Advances in Engineering and Technology, 2014, 2(1), pp.73-79.
166. Samir Tata. "Web Service Micro-Container for Service-based Applications in Cloud Environments"/ Samir Tata, Samir Moalla, Mohamed Mohamed, Sami Yangui // WETICE.2011.516 vol. 00, no. , 2011, doi:10.1109, pp. 61-66.
167. R. Venkatavaradan, Arulazi Dhesiaseelan. Managing Web Services: “A Container-Based Approach”/ R. Venkatavaradan, Arulazi Dhesiaseelan, developer.com, 2003, http://www.developer.com/java/web/article.php/2230731/Managing-Web-Services-A-Container-Based-Approach.htm
168. Singh, Y.“Model Driven Architecture: A Perspective”/ Singh, Y., Sood, M. // In Proc. IEEE International Advance Computing Conference, 2009.
169. Ahmed, S. "A Methodology for Creating Ontologies in Engineering Design" / Ahmed, S., Kim, S., and Wallace, K. M. //Transactions of the ASME of Journal of Computing in Information Science in Engineering, **7**(June), 2007, pp.132-140.
170. I. N. Athanasiadis. “Ontologies, JavaBeans and Relational Databases for enabling semantic programming”/ I. N. Athanasiadis, F. Villa, A. E. Rizzoli // Proceedings of the 31th IEEE Annual International Computer Software and Applications Conference (COMPSAC), Vol 2, 2007, pp. 341-346.
171. Manuel Mora. “Onto-ServSys: A Service System Ontology”/ Manuel Mora, Mahesh Raisinghani, Ovsei Gelman, and Miguel Angel Sicilia, econbiz.de, EconBiz, 2011, https://www.econbiz.de/Record/onto-servsys-a-service-system-ontology-mora-manuel/10009127309
172. Egon Lüftenegger. “The Service Dominant Business Model: A ServiceFocused Conceptualization”/ Egon Lüftenegger, Marco Comuzzi, Paul Grefen, Caren Weisleder // BETA publication, WP 402 (working paper), ISBN, NUR982, Eindhoven, January 2013.- 23 p.
173. A. Osterwalder. “Business model generation: a handbook for visionaries, game changers and challengers” /A. Osterwalder and I. Pigneur //Willey, New Jersey, 2010, 288 p.
174. H. B.Ouizghal. “A model-driven approach of ontological components for on- line semantic web information retrieval”/ H. B.Ouizghal, M. Aufaure, N.Mustapha //Journal of Web Engineering, Vol. 6, No.4,2007, pp. 303-329.
175. Troncy R. ”Integrating structure and semantics into audio-visual documents” // Proc. of the Second International Semantic Web Conference (ISWC 2003), Sanibel Island. Lecture Notes in Computer Science, vol. 2870, Springer, Berlin, 2003, pp. 566–581.

176. Sagar Sen.” Reusable model transformations” / Sagar Sen, Naouel Moha, Vincent Mah´e, Olivier Barais, Benoit Baudry, et al. // Software and Systems Modeling (SoSyM), Springer, 2012, 11 (1), pp.111-125.
177. Schlieter H.” Towards Model Driven Architecture in Health Care Information System Development”/ Schlieter, H.; Burwitz, M.; Schönherr, O.; Benedict, M. // Proceedings der 12. Internationalen Tagung Wirtschaftsinformatik ,WI 2015, Osnabrück, pp.497-511.
178. Li Ding. ”Using Ontologies in the Semantic Web: A Survey”/ Li Ding, Pranam Kolari, Zhongli Ding, Sasikanth Avancha, Tim Finin, Anupam Joshi // University of Maryland Baltimore, County Baltimore, MD 21250, 2005. 36 p.
179. Czarnecki. K.” Feature-Based Survey of Model Transformation Approaches”/ Czarnecki. K. and S. Helsen //I BM Systems Journal 45(3), 2006, pp. 621-645.
180. The Open Group . “Service-Oriented Architecture Ontology”, Draft Technical Standard, 208, pp.1–112.
181. S. Alahmari. “ A Model-Driven Architecture Approach to the Efficient Identification of Services on Service-oriented Enterprise Architecture”/ S. Alahmari, D. DeRoure, Ed. Zaluska // Second Workshop on Service oriented Enterprise Architecture for Enterprise Engineering in conjunction with the 14th IEEE International Enterprise Distributed Object Computing Conference, Vitória, Brazil, 2010.
182. María-Cruz Valiente. ”An Ontology-Based and Model-Driven Approach for Designing IT Service Management Systems”/ María-Cruz Valiente, Cristina Vicente-Chicote, Daniel Rodríguez // International Journal of Service Science, Management, Engineering, and Technology, 2(2), April-June 201, pp. 65-81.
183. “Oracle BPEL”, oracle.com, Oracle, 2009, http://www.oracle.com/us/products/middleware/application-server/bpel-process-manager-ds-066554.pdf
184. B. Bousetta. “A methodology for CIM modeling and its transformation to PIM”/ B. Bousetta, O. El Beggar and T. Gadi // Journal of Information Engineering and Applications, vol. 3, no. 2, 2013, pp. 1-21.
185. Y. Rhazali. “Transformation Method CIM to PIM: From Business Processes Models Defined in BPMN to Use Case and Class Models Defined in UML”/ Y. Rhazali, Y. Hadi, A. Mouloudi // International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 8, n. 8, 2014, pp. 1467–1471.
186. Y. Rhazali. “Transformation Approach CIM to PIM: From Business Processes Models to State Machine and Package Models”/ Y. Rhazali, Y. Hadi, and A. Mouloudi // Proceedings of the 1st International Conference on Open Source Software Computing, Amman, Jordan, 2015, pp. 1-6.
187. A. Rodríguez. “CIM to PIM Transformation: A Reality, Research and Practical Issues of Enterprise Information Systems II”/ A. Rodríguez, E. Fernández-Medina, M. Piattini // IFIP International Federation for Information Processing, vol. 255, 2008, pp 1239-1249.
188. “Service Oriented Architecture Modeling Language”, omg.com, Object Management Group, 2012, http://www.omg.org/spec/SoaML/
189. Claus Pahl. “Semantic Model-Driven Architecting of Service-based Software Systems”, Dublin City University, 2009, http://ceur-ws.org/Vol-244/paper3.pdf

190. C. Pahl. “An Ontology or Software Component Matching” //International Journal on Software Tools for Technology Transfer, Special Editionon Component-based Systems Engineering, 7,2006, pp. 1-10.
191. T. Levendovszky. “Model Reuse with Metamodel-Based Transformations”/ T. Levendovszky, G. Karsai, M. Maroti, A. Lede-czi, H. Charaf, C. Gacek (ed.) // Proc. of ICSR-7, Springer Berlin / Heidelberg, 2002, LNCS 2319,2002, pp.166-178.
192. The Open Group. “Service-Oriented Architecture Ontology”, Draft Technical Standard, 208.-pp.1–112.
193. Robert France. “Model-driven Development of Complex Software: A Research Roadmap”/ Robert France, Bernhard Rumpe // Proceeding FOSE '07 2007 Future of Software Engineering, May 23 - 25, 2007, pp. 37-54.
194. Ingo Pansa.“A Domain Ontology for Designing Management Services“/Ingo Pansa, Matthias Reichle, Christoph Leist, Sebastian Abeck // Proc. the Third International Conferences on Advanced Service Computing “SERVICE COMPUTATION”, 2011, pp.11-18.
195. Object Management Group. “Ontology Definition Metamodel - Request For Proposal”, (OMG Document: as/2003-03-40). OMG, 2003.
196. O. Vasilecas. “Applying the meta-model based approach to the transformation of ontology axioms into rule model”/ O. Vasilecas, D.Bugait // Information technology and control, 2007, Vol.36, No.1A, pp.122-125.
197. Egon Lüftenegger. “The Service Dominant Business Model: A Service Focused Conceptualization”/ Egon Lüftenegger, Marco Comuzzi, Paul Grefen, Caren Weisleder // Beta Working Paper series 402, Eindhoven University of Technology, January 2013, 50 p.
198. Henning Agt. “Metamodeling Foundation for Software and Data Integration”/ Henning Agt, Gregor Bauho, Mario Cartsburg, Daniel Kumpe, Ralf Kutsche, Nikola Milanovic.// International United Information Systems Conference UNISCON 2009: Information Systems: Modeling, Development, and Integration, Sydney, Australia, April 21-24, pp. 328-339.
199. Zhang, L. “Transforming business requirements into bpel: A mda-based approach to web application development”/ Zhang, L. and W. Jiang // Semantic Computing and Systems, WSCS'08. IEEE International Workshop on,2008, pp. 61-66.
200. De Castro V. “Applying CIM-to-PIM model transformations for the service-oriented development of information systems”/ De Castro V., E. Marcos, and J.M. Vara // Information and Software Technology, 2011, 53(1), pp. 87-105.
201. Y. Rhazali. “CIM to PIM Transformation in MDA: from Service-Oriented Business Models to Web-Based Design Models”/ Y. Rhazali, Y. Hadi and A. Mouloudi // Intern. Journal of Software Engineering and Its Applications, Vol. 10, No. 4 2016, pp. 125-142.
202. A. Kalyanpur. “Automatic Mapping of OWL Ontologies into Java”/ A. Kalyanpur, D. Jiménez, S. Battle, J. Padget, semanticscholar.org, 2005, https://pdfs.semanticscholar.org/ab2c/4b2ef8066fd7f3a9329a01538d44f077d6e8.pdf
203. “GREEN PAPER on mobile Health” ("mHealth"), ec.europa.eu, EC, 2014, http://ec.europa.eu/digital-agenda/en/news /green-paper-mobile-health-mhealth

204. Personal Health Systems: State of the Art – PHS Foresight, PHS_D1.1_Report_ 20130228, 28 February 2013, http://www.phsforesight.eu/wp-content/uploads/2013 /03/PHS_D1.1_Report_20130228.pdf
205. "M-Health Market in Russia and Worldwide: Main Tendencies and Forecasts", json.tv, J'son& Partners Consulting, 2014, http://json.tv/en/ict_telecom_analytics_ view/m-health-market-in-russia-and-worldwide-main-tendencies-and-forecasts-2014090506213061
206. "Меморандум щодо намірів співпраці у побудови в Україні прозорої та ефективної електронної системи охорони здоров'я", МОЗ України, 2 листопада 2016 року
207. "Open Clinical_ Ontologies", openclinical.org, Open Clinical, http://www.openclinical.org/ontologies.html
208. "An Ontology-based Approach for Data Integration in Regionally Interoperable Healthcare Systems"/ Hongqiao Yang, Weizi Li, http://centaur.reading.ac. uk/32726/1/An%20Ontology-based%20Approach%20for% 20Data%20Integration%20 in%20Regionally%20Inoperable%20Healthcare%20Systems.pdf
209. O.O. Petrenko. "A loosely-integrated suite of services for mhealth formed by wireless sensor networks" / O.O. Petrenko, A.I. Petrenko // Data Stream Mining & Processing. Proc. of the 1 th IEEE International Conference on Data Stream Mining & Processing, 23-27 August 2016, Lviv, Ukraine, pp.3-8.
210. Schlieter H. "Towards Model Driven Architecture in Health Care Information System Development", in: Thomas. O.; Teuteberg, F. (Hrsg.)/ Schlieter, H.; Burwitz, M.; Schönherr, O.; Benedict, M. // Proceedings der 12. Internationalen Tagung Wirtschaftsinformatik WI 2015, Osnabrück, pp. 497-511.
211. A. Katasonov. "Ontology-Driven Software Engineering: Beyond model checking and transformations" // International Journal of Semantic Computing, Vol. 6, No: 2, 2012, pp. 205 – 242.
212. Aziz H.A. "Cloud Computing and Healthcare Services"/ Aziz H.A., Guled A. // Journal of Biosensors & Bioelectronics, 2016, 7:3, pp.1-4. DOI: 10.4172/2155-6210.1000220
213. Long Ri Wen. "Healthcare Platform and Big Data Analysis Based Personal Fitness Healthcare Service Model"/ Long Ri Wen, Seung Min Yang, Byung Mun Lee // International Journal of Bio-Science and Bio-Technology,Vol.8, No.5, 2016, pp. 115-128.
214. D. Milovanovic. "Cloud-based IoT healthcare applications: Requirements and recommendations" /D. Milovanovic, Z. Bojkovic // International Journal of Internet of Things and Web Services, V.2, 2017, pp.60-65.
215. Chao Wu. "WikiHealth:A Big Data Platform for Health Sensor Data Management"/ Chao Wu, Li Guo, Chun-Hsiang Lee, Yike Guo; Cloud Computing Applications for Quality Health Care Delivery, http://wiki-health.org/about/overview.php
216. Zhanlin Ji. "A Cloud-Based X73 Ubiquitous Mobile Healthcare System: Design and Implementation"/ Zhanlin Ji, Ivan Ganchev,Máirtín O'Droma, Xin Zhang, Xueji Zhang // The Scientific World Journal, Volume 2014 ,2014,, Article ID 145803, 14 p. http://dx.doi.org/10.1155/2014/145803

217. MediaTek Sensio MT6381 Biosensor Announced: World's First 6-in-1 Health Monitoring Module For Smartphones, https://www.gizmochina.com/2018/01/18/mediatek-sensio-mt6381-biosensor-announced-worlds-first-6-1-health-monitoring-module-smartphones/

Додаток 1. ПЕРЕЛІК ПРОГРАМНОЇ ПРОДУКЦІЇ ДЛЯ КОНТЕРНЕРІВ

Компанія	Назва продукту	Тип продукту	Вільне впробування	Веб-сайт
Amazon	EC2 Container Registry	Container image registry	Free tier available	aws.amazon.com/ecr
Amazon	EC2 Container Service	Containers-as-a-Service	Free tier available	aws.amazon.com/ecs
Anchore	Anchore	Container image security	Demo available by request	anchore.com
Anchore	Anchore Navigator	Container monitoring	Free solution	anchore.io
Apache Software Foundation	Mesos	Cluster management software	Open source	mesos.apache.org
Apcera	Apcera	PaaS, container platform	Free tier available	apcera.com/platform
Apprenda	Apprenda Platform	PaaS, container platform, Kubernetes-as-a-Service	Available by request	apprenda.com/platform
Aqua	Aqua Container Security Platform	Container image security	Demo available by request	aquasec.com/products/aqua-container-security-platform
Bitnami	Stacksmith	Containers-as-a-Service	Available by request	stacksmith.bitnami.com
CA Technologies	Automic Continuous Service	Service orchestration	Available by request	automic.com/products/automic-service- orchestration
Canonical	Ubuntu Core	Container OS	Open source	developer.ubuntu.com/core
Cisco	Contiv	Container-defined networking	Open source	contiv.github.io
CoreOS	Quay Enterprise	Private container image registry	30 days	coreos.com/quay-enterprise
CoreOS	Clair	Container image security	Open source	coreos.com/clair/docs/latest

CoreOS	Flannel	Container-defined networking	Open source	coreos.com/flannel/docs/latest
Datadog	Datadog	Server monitoring, container	14 days	datadoghq.com
Diamanti	Diamanti	Container Platform	Trial available by request	diamanti.com/products
Docker	Docker	Container platform	Free tier available	docker.com/get-docker
Docker	Docker Swarm	Container orchestration and clustering	Open source	github.com/docker/swarm
Docker	Docker Compose	Multi-container application tool	Free solution	docs.docker.com/compose/install
Docker	Docker Hub	Container image registry	Free for public repos	hub.docker.com
Docker	Docker Trusted Registry	Private container image registry	30 days	docker.com/enterprise-edition#/container_management
Docker	Docker Cloud	Containers-as-a-Service	Free tier available	docker.com/enterprise-edition
Docker	Docker EE	Containers-as-a-Service, container security	Open source	kubernetes.io
Google	Kubernetes	Container orchestration	Free based on use	cloud.google.com/container-registry
Google	Google Cloud Container Registry	Container image registry	$300 free credit	cloud.google.com/container-engine
Google	Google Container Engine	Containers-as-Service, container orchestration	Free tier available	docker.com/enterprise-edition
IBM	Bluemix Containers	Containers-as-a-Service, Kubernetes-as-a-Service	Free tier available	ibm.com/cloud-computing/bluemix/containers
Mesosphere	Enterprise DC/OS	Container orchestration	Available by request	mesosphere.com/product

		and monitoring		
Microsoft	Windows Nano Server	Container OS	180 days	docs.microsoft.com/en-us/windows-server/get-started/getting-started-with-nano-server
Microsoft	Azure Container Service	Containers-as-a-Service, Orchestration-as-a-Service	Free tier available	azure.microsoft.com/en-us/services/container-service
Oracle	Smith	Microcontainer builder	Open source	github.com/oracle/smith
Oracle	CrashCart	Microcontainer debugging tool	Open source	github.com/oracle/crashcart
Oracle	RailCar	Rust-based container runtime	Open source	github.com/oracle/railcar
Oracle	Wercker	CI/CD platform for containers	Free tier available	wercker.com
Packet	Packet	Infrastructure-as-a-Service	Free tier available	packet.net/features
Pivotal	Cloud Foundry	PaaS, container platform	Open source	pivotal.io/platform
Platform9	Platform9 Managed Kubernetes	Kubernetes-as-a-service	Sandbox available	platform9.com/managed-kubernetes
Portworx	Portworx	Container data services and networking	Demo available by request	portworx.com
Prometheus	Prometheus	Container monitoring	Open source	prometheus.io

Додаток 2. ОБЧИСЛЕННЯ СЕМАНТИЧНОЇ БЛИЗЬКОСТІ

Оцінкою близькості між описом сервісу (документом) і запитом є числове значення, яке виражає ступінь подібності між ними; оцінка близькості називається оцінкою семантичної близькості, якщо і тільки якщо вона визначена на основі семантики документів і запитів.

Введемо наступні спрощені позначення компонентів онтології: онтологія O являє собою знакову систему

$$O = \langle C, E, R, T \rangle,$$

де C – безліч класів понять; E – множина екземплярів понять; R - множина предикатів, тобто типів відносин; T – множина відносин, які задають різні види зв'язку між класами, екземплярами, предикатами, включаючи частковий порядок на множинах C і R, що задає відносини is-a – «підклас / суперклас», «вище / нижче».

- Триплет $c1, c2 \in C,\ r1 \in R$ – це відношення між поняттями.
- Триплет $e1, e2 \in E,\ r1 \in R$ – це відношення між поняттями.
- Триплет $r1, r2, \ldots ri \in R$ – це відношення між предикатами.

Припустимо:

- На основі онтології предметної області O для кожного сервісу di реєстру $D = \{di\}$ створені семантичні метаописи $m(di) = \{t1, t2, \ldots tn(i)\}$, де $n(i)$ - кількість триплетів у логічному поданні сервісу di; ti – RDF-триплети-кортежі виду ⟨si, pi, oi⟩, де si та oi включені у об'єднання Ci та Ei, а pi належить R.
- Кожен запит q, даний користувачем із множини запитів Q, також складається з множини триплетів $q = \{t1, t2, \ldots tn(q)\}$, де $n(q)$ – кількість триплетів, що містяться у запиті q.
- Визначено вагову функцію w, яка визначає значимість будь-якого триплета $t \in T$ (T – множина можливих триплетів) при описі сервісу di та запиту q: $0 \le w(t, di) \le 1$, де $t \in T$, $di \in D$, $0 \le w(t, q) \le 1$, де $t \in T$, $q \in Q$.

Завдання, що вирішується, полягає у тому, що для кожного запиту *q* потрібно визначити підмножину *RES* множини сервісів *D*, яка складається з релевантних сервісів для заданого запиту *q*. Сервіс *di* вважається релевантним заданому запиту *q*, якщо і тільки якщо оцінка семантичної близькості між ними перевищує деяке порогове значення. При цьому для обчислення близькості між сервісом і запитом використовуються їх семантичні метаописи.

Облік структури онтології і семантики відносин дозволяє обчислювати оцінки семантичної близькості між елементами онтології (поняття, екземпляри, зв'язку – предикати). Ці оцінки близькості називаються елементарними оцінками близькості, на основі яких визначаються близькості між триплетами. Оцінки близькості між триплетами потім використовуються для визначення близькості між метаописами.

Виходячи зі структури триплетів, для їх порівняння потрібні обчислення наступних видів елементарних оцінок близькості між: 1) поняттями; 2) поняттями і екземплярами; 3) екземплярами; 4) предикатами. Методи обчислення кожного типу елементарної оцінки близькості надаються далі. Представлений огляд є розширенням роботи [138,139], коли наведені більш докладні описи розглянутих підходів і додані нові підходи по концептуальної близькості, тобто семантичної близькості між поняттями. Розглянемо, перш за все, обчислення концептуальної близькості на основі ***таксономії***, коли враховується тільки семантичне відношення «вище-нижче» (**is-a** – також відомо як відношення «батько-дитина»). Розглянуті підходи групуються відповідно до властивостей, які використовуються для обчислення близькості. При цьому найчастіше використовуються наступні характеристики понять: 1) довжина шляху між поняттями; 2) глибина таксономії; 3) інформаційний зміст; 4) безліч батьківських понять. Крім того, розглянуті гібридні підходи, тобто методи комбінування різних заходів близькості для отримання більш ефективного вимірювання близькості**.**

Д2.1. Підходи на основі довжини шляху

У роботі [139] запропоновано визначення семантичної відстані між

поняттями. як кількості ребер шляху між ними у таксономії:

$$\mathtt{distRada(c1, c2)} = \min(|\mathtt{path(c1, c2)}|),$$

де $|\mathtt{path(c1, c2)}$ – кількість ребер шляху від $c1$ до $c2$. Шляхи між поняттями визначені з урахуванням таксономії як неорієнтований граф.

Семантична відстань – це зворотна величина близькості, тобто чим більша семантична відстань, тим меншою є близькість і навпаки.

У [141] представлена міра близькості, за якою обмежуються характеристики шляхів між поняттями. При цьому враховуються тільки ті шляхи, які містять не більш за 5 ребер або відповідають одному із 8 шаблонів, представлених у [142]. Близькість за допустимим шляхом обчислюється таким чином:

$$\mathtt{simHirst(c1, c2)} = \mathtt{C - S - k \cdot N},$$

де $\mathtt{S}$ – довжина шляху від $c1$ до $c2$, N – кількість змін напрямку, C, k –константи.

Отже, чим довшим є шлях і більшою кількість змін у напрямку, тим меншою є близькість між поняттями.

Подібно підходу [143] у [144] обмежується конфігурація шляхів, використовуваних при обчисленні близькості. При цьому розглядаються шляхи, що складаються із сукупності ієрархічних відносин (вище / нижче – is-a, частина / ціле – partOf і несиметрична асоціація) або спрямовані в один бік, або включають тільки один перегин (зміна у напрямку).

У роботі [146] близькість між поняттями x і y обчислюється як максимальний добуток вагових коефіцієнтів ребер шляхів між ними. За цим методом для відносини is-a задаються два параметри gen, $spec \in [0, 1]$, які відповідно виражають близькості у напрямку узагальнення і деталізації.

Д2.2. Підходи на основі довжини шляху і глибини вершин

У роботі [148] представлена міра семантичної відстані між поняттями WordNet (зворотна величина близькості). При цьому семантична відстань між будь-якими поняттями $c1$ і $c2$ обчислюється як сума відстаней між сусідніми поняттями, що входять у шляхи між ними.

Семантична відстань між поняттями $c1$ і $c2$, зв'язана з коефіцієнтом відношенням r, обчислюється за формулою:

$$dist_{sunna}(c1,c2) = \frac{\omega(c1 \to c2) + \omega(c2 \to c1)}{2d},$$

з ваговими коефіцієнтами:

$$\omega(c1 \to c2) = \max_r - \frac{\max_r - \min_r}{n_r(c1)},$$

де d – глибина ребра таксономії, тобто максимальне значення глибини двох понять; $\max_r, \min_r$ – максимальне і мінімальне значення вагового коефіцієнта відношенння r; $n_r(x)$ – кількість виходів з поняття x відносини r.

Обчислення семантичної відстані у таксономії є окремим випадком оригінального алгоритму, тобто враховуються тільки ставлення «вище / нижче» (hypernymy / узагальнення) і його зворотне відношення – відношення деталізації (hyponymy). Згідно до роботи [148] вагові коефіцієнти цих відносин варіюються у діапазоні від 1 до 2.

У роботі [149] концептуальна близькість між поняттями $c1$ і $c2$ визначена наступним чином:

$$dist_{Wu\&Palmer}(c1,c2) = \frac{2N_3}{N_1 + N_2 + 2N_3}.$$

Нехай $c3$ – найближче узагальнення понять $c1$ і $c2$, тоді є наступні визначення параметрів заданої формули: N_1 – кількість вершин шляху від c1 до $c3$ – довжина шляху між ними; N_2 – кількість вершин шляху від $c2$ до $c3$; N_3 – кількість вершин шляху від $c3$ до корінного поняття таксономії – її глибини.

Д2.3. Підходи на основі інформаційного змісту

У роботі [150] запропоновано наступне визначення близькості між поняттями $c1$ і $c2$:

$$sim(c1,c2) = \max\left(IC(c)\right),$$

де $sim(c1,c2)$ – множина найближчих верхніх загальних понять для $c1$ і $c2$; $IC(c)$ – інформаційний зміст поняття c. При цьому вживається наступне визначення величини інформаційного змісту поняття таксономії: нехай C – множина понять таксономії і для кожного поняття $c \in C$ визначена ймовірність $p(c)$ того, що зустрічається екземпляр поняття з навчальної колекції текстів.

У роботі [151] інформаційний зміст поняття c визначений у наступний спосіб:

$$IC(c) = -\log p(c).$$

У роботі [152] запропоновано, що зустрічальність кожного терміна навчальної текстової колекції враховується у підрахунку частоти тих понять таксономій, які включають цей термін. Виходячи з даного правила, частота поняття в колекції визначається наступним чином:

$$freq(c) = \sum freq(t).$$

А імовірність того, що зустрічається поняття c, визначається за такою формулою:

$$p(c) = \frac{freq(c)}{N},$$

де N – кількість анотованих термінів навчальної колекції.

Д2.4. Концептуальна близькість в онтології

Вищеописані методи обчислення концептуальної близькості базуються на відношенні «вище / нижче». Однак немає необхідності обмежити вимірювання близькості у використанні тільки цієї відносини. В онтології кількість можливих семантичних відносин необмежена, тому семантична близькість в онтології є більш широким поняттям, ніж у таксономії.

Оцінка концептуальної близькості в онтології може означати частки загальних частин сутностей і ступінь зв'язаності між ними. Наприклад, поняття «електричний автомобіль» і «автомобіль» є близькими поняттями, однак «бензин» і «автомобіль» є зв'язаними поняттями, і ступінь зв'язаності між ними може бути виражена оцінкою близькості в онтології. Проблема обчислення

оцінки семантичної близькості з урахуванням різних типів семантичних відносин розглянута у роботах [156-158].

У [158] представлена симетрична міра семантичної близькості, яка враховує різні семантичні відносини. При цьому для кожної семантичної відносині задається вага, яка приймає значення у діапазоні [0, 1]. Семантична близькість заданого шляху обчислюється як добуток ваг її ребер.

Д2.5. Семантична близькість різнотипних елементів

У роботі [159] представлені методи визначення близькості між різнотипними елементами, згідно з якими порівняння двох різнотипних елементів онтології можливо лише з певними припущеннями, які виражаються відповідними коефіцієнтами. Для обчислення семантичної близькості між поняттям і екземпляром використовується коефіцієнт $dCI \in (0,1]$, який визначає близькість між батьківським поняттям і екземпляром:

$$sim(c,i) = dCI * \max(sim(c,cx))$$

Д2.6. Узагальнення підходу зваженого найкоротшого шляху

У цьому розділі розглядається модифікація міри близькості, що надана у роботі [154], з урахуванням усіх видів семантичних відносин, яка повинна бути придатною для обчислення усіх видів близькості між компонентами триплетів.

Згідно запропонованої модифікації:

- Для відносини «батько-дитина» (is-a) задаються два коефіцієнти *gen* і *spec*, які відповідно визначають близькість у напрямку узагальнення і деталізації.
- Для відносини instance Of (що пов'язує поняття з екземплярами понять) задаються два параметра *dIC*, $dCI \in [0,1]$, які відповідно визначають близькість примірника (екземпляра) поняттю і близькість поняття примірнику.
- Коефіцієнти близькості для відносини *sameAs* (синоніми) та *invertOf* (зворотні відносини) відповідно вибиають рівними 1 та (-1).

- Для інших семантичних відносин *ri* визначається ваговий коефіцієнт *wr*, який визначає семантичну близькість за цими відносинами.

Для P-шляху між сутностями *x* і *y* (які можуть бути поняттями, екземплярами, або предикатами) визначаються наступні характеристики:

- *s (P)* – кількість ребер у напрямку деталізації;
- *g (P)* – кількість ребер у напрямку узагальнення;
- *ic (P)* – кількість ребер від екземпляра до поняття;
- *ci (P)* – кількість ребер від поняття до примірника;
- *inv (P)* – кількість ребер інверсної відносини;
- *oth (P)* – кількість ребер інших відносин.

Оцінка близькості між сутностями *x* і *y* відповідно до шляху P визначається за такою формулою:

$$sim^P(x,y) = -1^{inv(P)} spec^{s(P)} gen^{g(P)} dic^{ic(P)} dci^{ci(P)} wr^{oth(P)}.$$

Д2.7. Налаштування коефіцієнтів

Параметри алгоритму можуть бути налаштовані ручним або автоматичним шляхом. У літературі описані методи автоматичної настройки оптимальних значень коефіцієнтів за допомогою навченою нейронної мережі або генетичного алгоритму [146]. Далі представлений примітивний спосіб визначення приблизних оптимальних значень коефіцієнтів за методом максимізації оцінки міри близькості.

Основна ідея пропонованого методу полягає у тому, що чим більшим є значення коефіцієнта кореляції між оцінками близькості алгоритму і оцінками близькості експертів, тим ефективнішим вважається алгоритм, і при оптимальних значеннях параметрів алгоритму значення коефіцієнта кореляції є максимальним, тому що відповідно до [157] оцінка близькості має суб'єктивний характер.

Оптимальні значення коефіцієнтів визначаються методом повного перебору за наступної умові:

$$corr\left(k_{i(max)}, \dots, k_{m(max)}\right) = \max(corr),$$

де corr(k1(i) , …, km(i)) – значення коефіцієнта кореляції міри близькості з використанням заданих параметрів; max(corr) – максимальне значення коефіцієнта кореляції для даної міри близькості.

Для обчислення значення коефіцієнта кореляції міри близькості спочатку обчислюються близькості між сутностями навчальної колекції за допомогою алгоритму. Потім обчислюється значення коефіцієнта кореляції з оцінками близькості експертів за відомою формулою Пірсона.

Д2.8. Оптимізація виконання запиту

Можливі два методи підвищення швидкості обробки запитів: фільтрація колекції документів за допомогою інвертованого індексу і застосування статичних оцінок близькості. Ці методи можуть бути застосовані окремо або спільно залежно від доступних ресурсів обчислювальних систем**.**

Перший спосіб оптимізації виконання запиту полягає у визначенні нечіткої підмножини D_q множини сервісів D, екземпляри з якого можуть бути релевантними запиту q. Потім порівняння виконуються тільки у даній множині D_q.

Дана підмножина D_q називається контекстною множиною запиту q. Пропонується визначення контекстної підмножини D_q як об'єднання списків релевантних сервісів наступним чином:

$$D_q = \bigcup_{t \in q} I_t,$$

де I_t – список релевантних сервісів триплета t. На основі індексу I список I_t визначається наступним чином:

$$I_t = \left\{ \frac{\mu_{I_t}(d)}{d} \middle| d \in D \right\}.$$

З використанням контекстної підмножни D_q можливе наступне оптимізоване визначення множини результатів запиту q:

$$RES = \left\{ \frac{\mu_{RES}(d)}{d} \middle| d \in \alpha D_q \right\},$$

де αDq – чітка множина, отримана в результаті операції α-зрізання над нечіткою підмножиною D_q.

При цьому прискорення виходить за рахунок того, що розмір множини αDq менший, ніж розмір колекції сервісів D. Однак знижується повнота результатів у зв'язку з можливими помилками фільтрації. Особливість структури інвертованого індексу, що застосований у методі, полягає у використанні триплетів для складання словника покажчиків. Тому для реалізації такого типу інвертованих файлів потрібен ефективний метод організації словників триплетів, дослідження даної проблеми представлено у роботі [158].

За другим методом швидкість виконання запиту може бути збільшена за рахунок використання статичних оцінок близькості між елементами метаописів. Статичний метод означає, що оцінки семантичної близькості обчислюються заздалегідь до виконання запиту і зберігаються у вигляді двомірного масиву. Для даного методу необхідно великий дисковий простір для зберігання масиву оцінок близькості, тому що метод передбачає визначення оцінки семантичної близькості попарно. Примітивним методом вирішення цієї проблеми є видалення малих елементів масивів, тобто зберігаються тільки ті пари, близькість яких вище, ніж задане граничне значення α. У порівнянні з динамічним методом статичний метод збільшує продуктивність системи, однак потрібує більшого дискового простору і додаткові обчислювальні витрати при індексуванні.

Додаток 3. ДО ПОБУДОВИ ОНТОЛОГІЇ МОБІЛЬНОЇ МЕДИЦИНИ

Таблиця базових концептів

Концепти			
Staff (штат)	Treatment (лечение)	Heartbeat (серцебиття)	Drugs (ліки)
Nurse (сестра)	Symptoms (симптоми)	Device (прилади)	Surgery (хірургія)
Patient (пацієнт)	Diagnosis (діагностика)	Sensor (сенсор)	Headache (головна біль)
Paramedics (парамедик)	Radiation (радіація)	Position Detector (місцезнаходження)	Fever (температура)
Doctor (доктор)	Cough (кашель)	Heartbeat Sensor (сенсор серцевого ритму)	
Surgeon (хірург)	Chest Pain(болі в грудях)	Temperature Sensor (сенсор температури)	Blood Pressure (кров'яний тиск)
Physician (лікар)	Vital Signs (життєво-важливі ознаки)	BP Sensor (датчик BP	Therapy (терапія)

Таблиця базових співвідношень

Relation type	**Source concept**	**Target concept**	**Inverse relation**
Treats (лікує)	the Doctor	Patient	Is treated by
Cares about (піклується)	the Nurse	Patient	Is cared by
Has (має)	Patient	Vital signs, Symptoms	Recorded from
Send data to	Device	Doctor, Nurse, Paramedics	Ask data from
Examines (обстежує)	the Doctor	Patient	Is examined by
Operates (оперує)	Surgeon	Patient	Is operated by
Get (отримує)	Patient	Treatment	Is given to

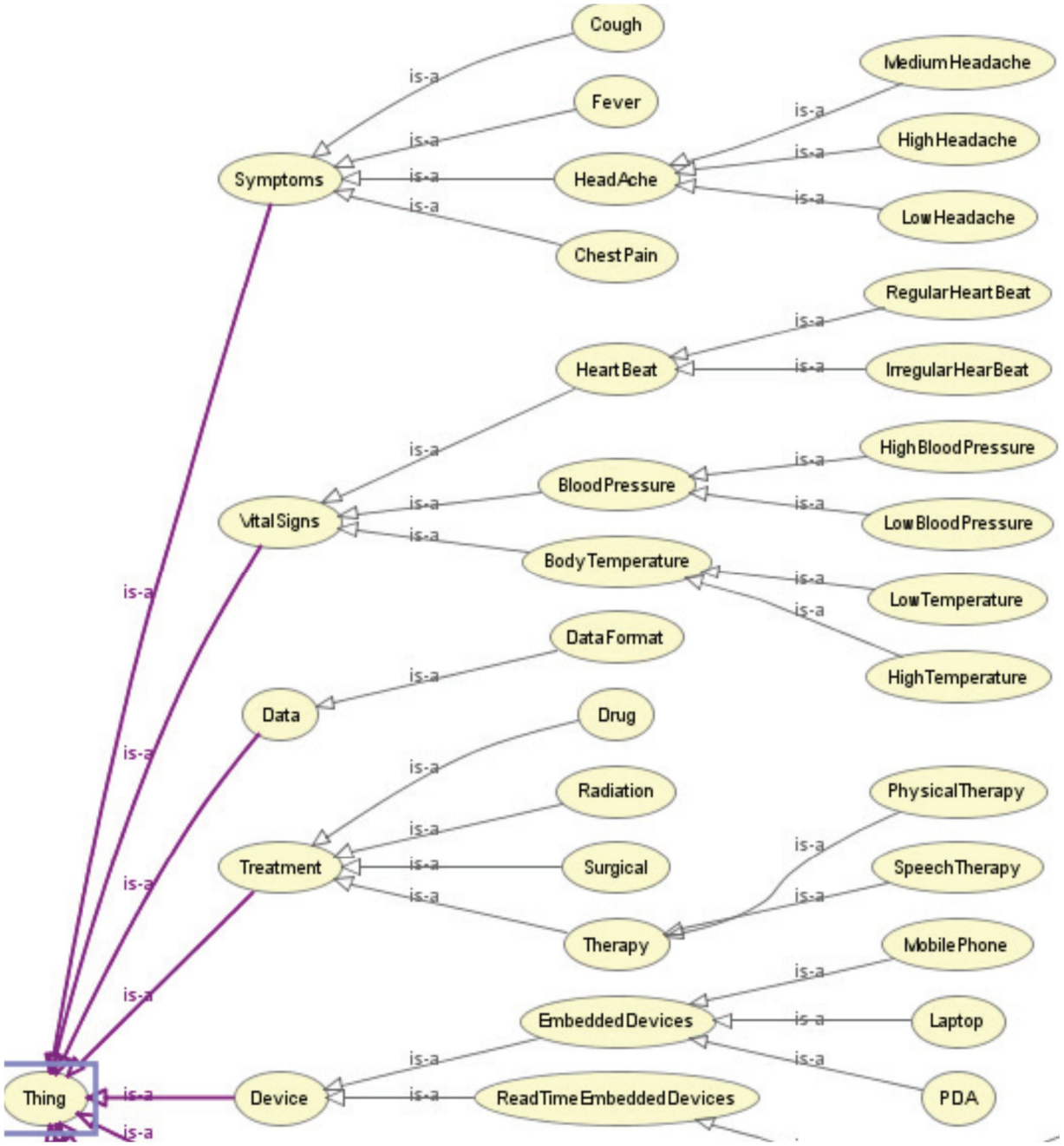


Рис.Д3.1. Медична онтологія (І-частина)

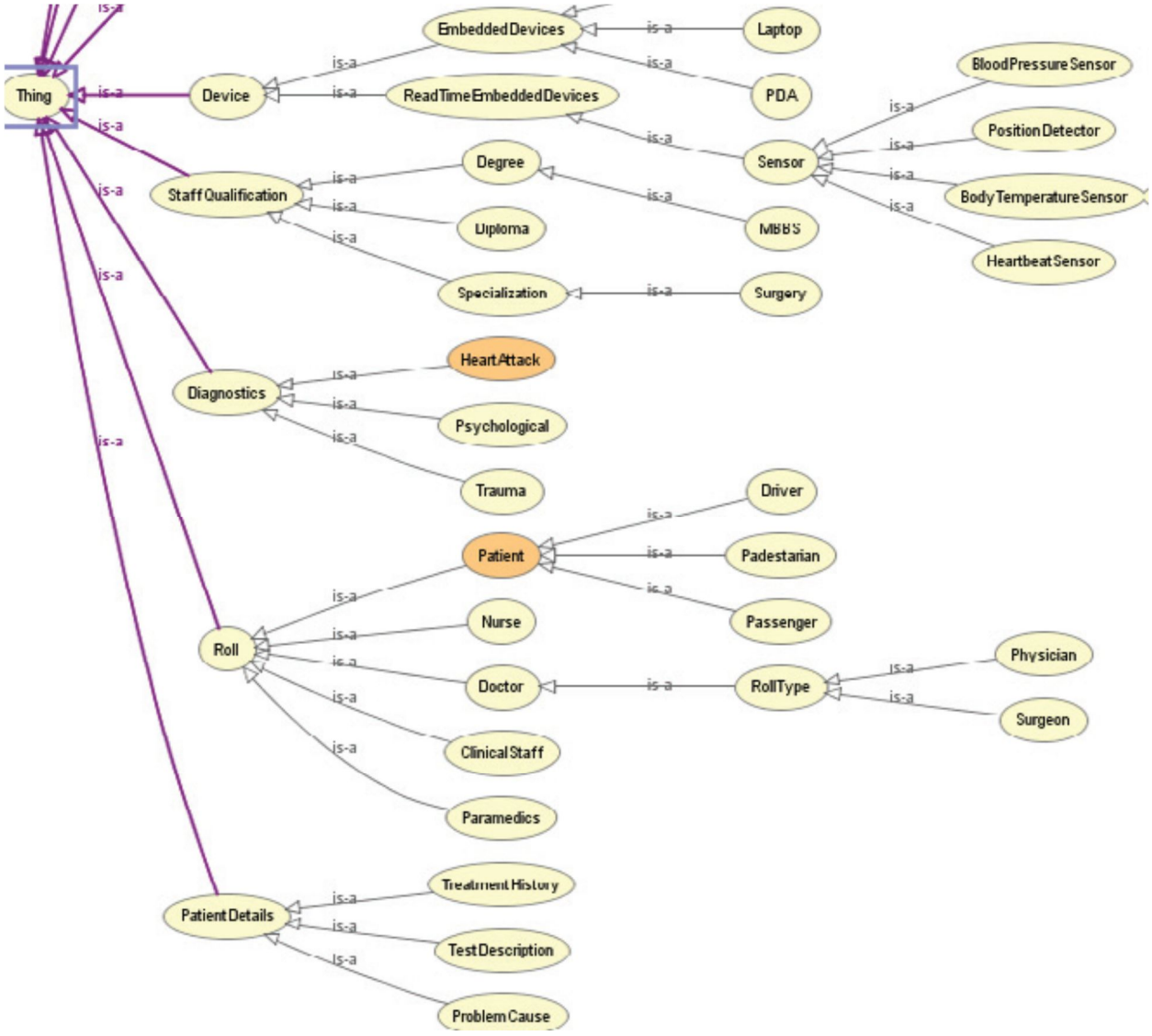


Рис.Д3.2. Медична онтологія (II частина)

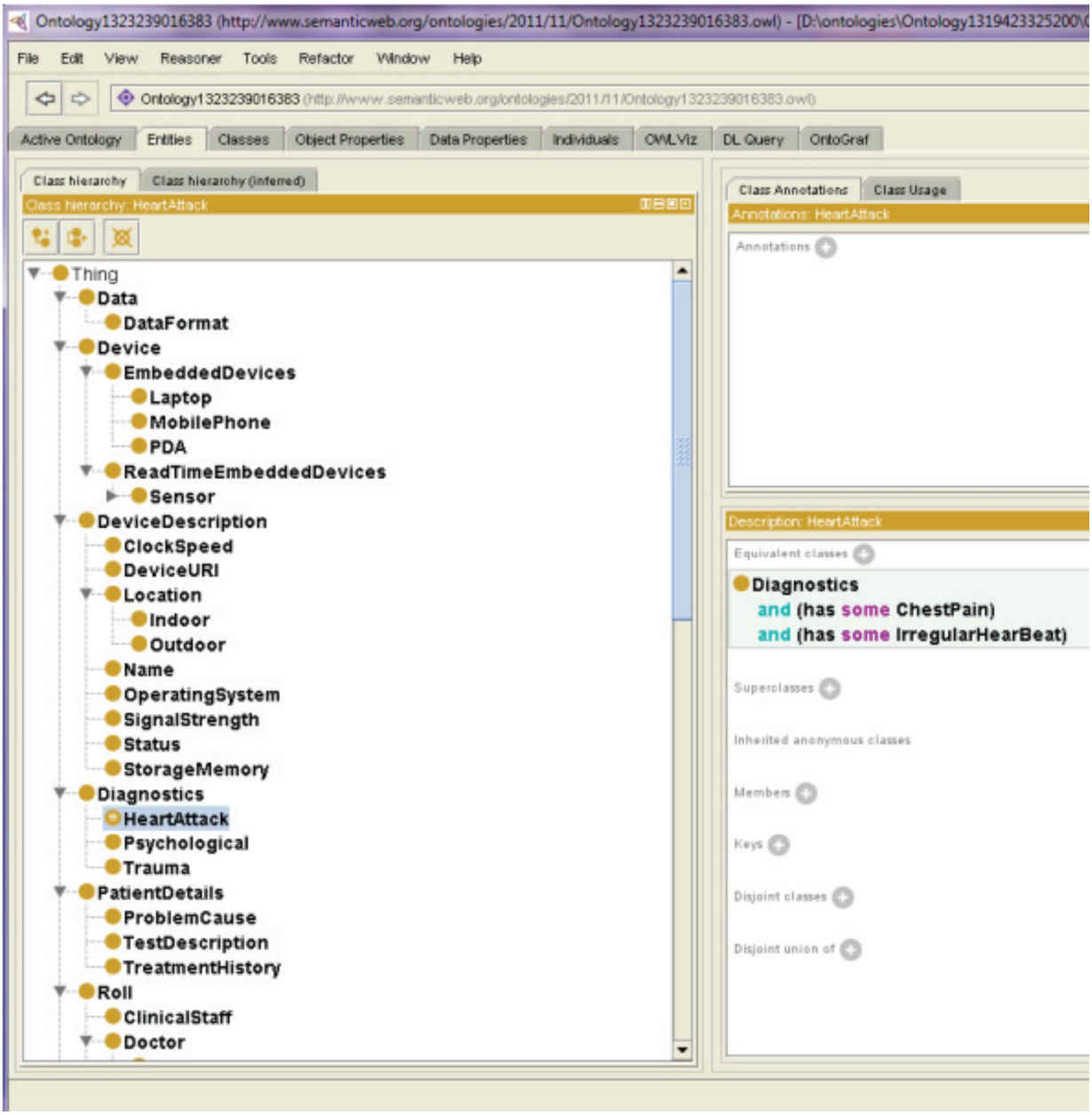


Рис.Д3.3. Дерево класифікатора концепції в редакторі Protégé