НАЦІОНАЛЬНА
АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

ПРОБЛЕМИ ПРОГРАМУВАННЯ

науковий журнал

№ 2 ***Квітень – Червень*** ***2022***

Заснований у березні 1999 р.

ЗМІСТ

Свідоцтво про державну реєстрацію КВ № 7490 від 01.07.2003

Науковий журнал "Проблеми програмування" занесений до переліку наукових фахових видань України, в яких можуть публі-куватися основні результати дисертаційних робіт.

ISSN 1727-4907

NATIONAL ACADEMY
OF SCIENCES OF UKRAINE
INSTITUTE OF SOFTWARE SYSTEMS

PROBLEMS OF PROGRAMMING

scientific journal

№ 2 ***April – June*** ***2022***

Founded in March, 1999

CONTENTS

УДК 004.93’1 HTTPS://DOI.ORG/10.15407/PP2022.02.003

С. О. Безпалько, В. М. Шимкович, А. Ю. Дорошенко

МОДЕЛЬ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ІНЕРЦІЙНОГО ВИМІРЮВАЛЬНОГО ПРИСТРОЮ

Проблеми автоматичного регулювання в сучасних технічних системах вимагають вирішення таких задач, як дотримання високої точності регулювання і здатності працювати в реальному часі. В даній роботі ці задачі вирішуються на прикладі розробки моделі та програмного забезпечення модуля стабілізації кута нахилу площини з трьома ступенями свободи, а також програмного забезпечення для отримання даних з акселерометра-гіроскопа MPU-6050 з використанням протоколу I2C. Розроблено програмну реалізацію цифрового пропорційно-інтегрально-диференціального регулятора з алгоритмом автоматичного підбору коефіцієнтів. Побудовано модель гіроскопічного пристрою та проведено тестування створеного рішення.

Ключові слова: гіроскопічна система, модуль стабілізації, інерційний вимірювальний пристрій, регулятор, алгоритми автоматичного підбору коефіцієнтів.

Вступ

Сучасний етап розвитку людської цивілізації передбачає застосування різних механізмів, які потребують можливості точно регулювати кут нахилу своїх елементів у різних площинах. Наприклад, від танків нового покоління вимагається досконалі системи наведення гармат, а отже можливість стабілізації ствола. До даних систем постійно висуваються нові вимоги: такі як збільшення точності стабілізації та швидкості, поліпшення завадостійкості та ін.

Для виконання всіх цих вимог заводами-виробниками і розробниками постійно удосконалюються конструкції і робочі процеси в стабілізаторах озброєння. На даний момент створено нове покоління високоточних датчиків та обчислювальних комплексів, яке об’єднує результати науково-технічного прогресу і практичного досвіду.

Такі системи керування кутом нахилу необхідні не тільки у військовій сфері, а й мають безліч застосувань у комерції. Зокрема, вдало застосовуються для вирішення проблеми стабілізації камери або фотоапарату для фото- чи відеозйомки. Метод, який не вимагає будь-яких додаткових характеристик комплекту «корпус-об’єктив», полягає у стабілізації всього корпусу камери — на противагу використанню внутрішнього механізму стабілізації зображення. Цього досягають завдяки гіроскопу, приєднаного до корпусу камери. Це дозволяє зовнішньому гіроскопу стабілізувати камеру. Використовується в основному для фотографування із транспортних засобів під час руху в тому випадку, коли об’єктиви чи камери із іншим типом стабілізації зображення недоступні або виявляються неефективними.

Нині дуже великої популярності набули квадрокоптери, для яких контроль кута нахилу відносно землі є першочерговою задачею для якісного та контрольованого польоту. Також подібні технології можуть полегшувати щоденне життя людей із хворобою Паркінсона, а також застосовуватись у хірургії для полегшення роботи лікаря та уникнення механічних помилок.

Якісні електронні датчики-гіроскопи стали доступними в роздрібній торговельній мережі зовсім недавно. Електронні гіроскопи – це перетворювачі нахилу і кутової швидкості в електричний струм. Вони служать для контролю положення і кутових швидкостей із коротким часом відгуку. Відмінністю електронних гіроскопів від їхніх механічних побратимів є компактність і мала вага, відсутність зношування елементів, висока швидкість відгуку, низька напруга живлення та малий струм живлення. Діапазон робочих температур дозволяє здійснювати контроль па-

раметрів у широкому спектрі кліматичних умов і географічних поясів.

Під час автоматичного регулювання кута виникають наступні задачі: необхідно отримати високу точність регулювання (< 18 мрад), система має працювати в реальному часі, тобто встигати здійснювати обрахунки в детермінований проміжок часу, необхідний для нормальної роботи об'єкту(час реакції < 0.001 секунди). Наприклад (це особливо важливо), якщо вирішується питання стабілізації польоту квадрокоптера, коли занадто довгий час реакції призведе до падіння. Також необхідно, щоб перерегулюваня було не більше 15%, адже це може впливати на стабільність системи.

1. Апаратна платформа та модель з інерційним вимірювальним пристроєм

В якості інерційного вимірювального пристрою в даній роботі був використаний модуль акселерометра і гіроскопа MPU6050, що підключається до мікроконтролера через шину I2C. Збір даних із MPU6050 можливий двома способами: збір сирих даних прямо з гіроскопічного датчика та акселерометра або використання вбудованого процесора руху, що називається виробником Digital Motion Processor (DMP)[1]. Для розробки коду та тестування отриманих даних було створено схему на макетній платі, що містить Arduino Nano та MPU6050. Це необхідно для дослідження різних способів отримання даних, перевірки працеспроможності інерційної вимірювальної системи та проведення необхідних налаштувань. Завдяки підключенню до персонального комп'ютера через універсальну послідовну шину з'являється можливість будувати графіки залежності положення від часу і, таким чином, візуалізувати бачити різні проблеми роботи датчика.

Після налаштувань запускається головний цикл програми і починається процес зчитування даних. Оскільки гіроскоп надає дані у вигляді кутової швидкості в даний момент, а не власне значення кута, необхідно здійснити розрахунки. Рівняння кутів на момент часу буде мати наступний вигляд:

$$\begin{aligned} x_i &= |x_{i-1} + w_x \times \Delta T|, \\ y_i &= |y_{i-1} + w_y \times \Delta T|, \\ z_i &= |z_{i-1} + w_z \times \Delta T|, \end{aligned} \quad (1)$$

де x_i, y_i та z_i - 3 кута Ейлера в даний момент часу; x_{i-1}, y_{i-1} та z_{i-1} – кути в попередній проміжок часу, w_x, w_y та w_z – кутова швидкість у відповідних площинах, ΔT – час між двома замірами;

Отримання часу здійснюється за допомогою функції millis() та розраховується кількість секунд із моменту минулого обрахунку. Дані датчика гіроскопа розміщені за адресами 0x43, 0x44 для осі x, 0x45, 0x46 для y та 0x47, 0x48 для z (рисунок 3). 2 байти для кожної з осей.

Для отримання всіх даних у коді запрошуються 6 байт з адреси 0x43. Після цього вони записуються у відповідні змінні. Поділ на значення 131 необхідне для переведення даних у градуси/с. Це значення відрізняється в залежності від виставленого діапазону на гіроскопі. Стандартне значення діапазону - 250 град/с. Після цього вираховується стала похибка датчика з заміряних кутових швидкостей та обраховуються кути за формулою (1).

Щоб збільшити точність показників, до розрахунку можна додати кути, розраховані за допомогою акселерометра. З роботи [4] відомо, що:

$$x = \operatorname{atan}\left(\frac{b}{\sqrt{a^2 + c^2}}\right),$$
$$y = \operatorname{atan}\left(\frac{-a}{\sqrt{b^2 + c^2}}\right),$$

де x, y – кути в радіанах, a, b, c – рух у трьох координатах, заміряний акселерометром; за цими формулами розраховуються кути, що переводяться в градуси. Відтак також віднімається статична похибка. Статична похибка обраховується експериментально, поклавши датчик на рівну поверхню та зробивши декілька сотень розрахунків і порахувавши їх середнє значення.

Після обрахування кутів гіроскопу та акселерометра можна отримати кінцевий результат, застосувавши «підсумову-

ючий фільтр», який підсумовує дані в певній пропорції. В даному випадку беруться 96% даних із гіроскопа та 4% даних із акселерометра, далі отримані дані виводяться в послідовний порт для відображення їх на персональному комп'ютері.

Проте, цей метод збору інформації не є ідеальним. Статична похибка досі присутня, хоча і є дуже малою. Під час руху відбуваються певні стрибки, що унеможливлює отримання кута обертання навколо осі Z з акселерометра, бо вона вирівняна з вектором гравітаційної сили. Однак головна проблема в тому, що після руху датчика з'являється велика похибка, яку неможливо усунути. Це пов'язано з тим, що мікроконтролер не встигає обробити всі дані. Тому якщо після руху покласти датчик у те ж положення, в якому він знаходився, буде добре видно, що дані спокою будуть зовсім іншими.

Для усунення таких помилок застосовується процесор руху, вбудований в MPU6050 [5]. Отриманий пакет даних повертає положення сенсора у вигляді кватерніона, який одразу виводиться в послідовний порт для відображення даних на комп'ютері. Виведення положення через кватерніон є перевагою даної бібліотеки, адже проводити обрахунки з використанням кватерніонів набагато легше, ніж з кутами Ейлера. Кватерніони можна перевести в кути Ейлера за допомогою формули:

$$\begin{bmatrix}\alpha\\ \beta\\ \gamma\end{bmatrix} = \begin{bmatrix}\arctan\left(\frac{2(q_0q_1 + q_2q_3)}{1-2(q_1^2+q_2^2)}\right)\\ \arcsin\left(2(q_0q_2 - q_3q_1)\right)\\ \arctan\left(\frac{2(q_0q_3 + q_1q_2)}{1-2(q_2^2+q_3^2)}\right)\end{bmatrix},$$

де α, β, γ – кути Ейлера, q_i - елементи кватерніона [6].

Проте, позаяк функції arctan та arcsin в комп'ютерних мовах програмування імплементовані так, що працюють тільки в межах від $-\pi/2$ до $\pi/2$, це накладає обмеження на всі ступені свободи [17]. Аби мати можливість згенерувати кути для всіх можливих орієнтацій платформи необхідно використовувати функцію atan2 замість arctan.

$$q = \begin{bmatrix}\cos\left(\frac{\gamma}{2}\right)\\ 0\\ 0\\ \sin\left(\frac{\gamma}{2}\right)\end{bmatrix}\begin{bmatrix}\cos\left(\frac{\beta}{2}\right)\\ 0\\ \sin\left(\frac{b}{2}\right)\\ 0\end{bmatrix}\begin{bmatrix}\cos\left(\frac{\alpha}{2}\right)\\ \sin\left(\frac{\alpha}{2}\right)\\ 0\\ 0\end{bmatrix} =$$

$$= \begin{bmatrix}\cos\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right)\\ \sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right) - \cos\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right)\\ \cos\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right)\\ \cos\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right)\end{bmatrix},$$

де α, β,γ – кути Ейлера, q - кватерніон;[6]

$$\begin{bmatrix}\alpha\\ \beta\\ \gamma\end{bmatrix} = \begin{bmatrix}\text{atan2}\left(2(q_0q_1 + q_2q_3), 1-2(q_1^2+q_2^2)\right)\\ \text{asin}\left(2(q_0q_2 - q_3q_1)\right)\\ \text{atan2}\left(2(q_0q_3 + q_1q_2), 1-2(q_2^2+q_3^2)\right)\end{bmatrix},$$

де α, β,γ – кути Ейлера, q_i – елементи кватерніона. В [5] міститься програмна реалізація даної формули в структурі MPU6050. В даному випадку для розрахунку нахилу в осі Y та в осі X використовується вектор гравітації, який можна отримати з акселерометра.

2. Цифровий ПІД регулятор

ПІД регулятор можна виразити математично через наступне рівняння:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt},$$

де K_p – пропорційний коефіцієнт, K_i – інтегральний коефіцієнт, K_d – диференціальний коефіцієнт, e(t) – помилка; це рівняння можна розділити на 3 окремі частини.

$$pOut(t) = K_p e(t),$$

$$iOut(t) = K_i \int_0^t e(t')dt',$$

$$dOut(t) = K_d \frac{de(t)}{dt}.$$

Де pOut – формула пропорційного впливу, iOut – інтегрального та dOut – диференціального[8]. Після цього формулу можна переписати наступним чином:

$$u(t) = pOut(t) + iOut(t) + dOut(t).$$

Підібравши коефіцієнти регулятора, можна отримати майже ідеальний перехідний процес системи. Тому програмна

реалізація має мати наступні особливості: можливість легкої зміни коефіцієнтів, вихід регулятора має регулюватись у певних межах, а також має бути можливість задавання різної позиції.

Нижче на рис.1 показано оголошення класу, який реалізує логіку роботи пропорційно-інтегрально-диференціального регулятора.

Програмний інтерфейс даного класу складається з трьох функцій та одного конструктора. В конструкторі визначаються межі регулювання, setK відповідає за встановлення коефіцієнтів регулятора. SetPos використовується для визначення необхідного положення, calcReg – функція, що прораховує керуючий вплив, на вхід вона приймає значення положення в даний момент часу, а на виході повертає значення впливу.

```
double PID::calcReg(double inp_){
    //calculate time interval
    auto currTime = millis();
    double dT = (double)(preTime - currTime);
    preTime = currTime;

    //get error
    double error = pos - inp_;

    //calculate every part of PID
    double pOut = kp * error;
    double iOut = ki * error * dT;
    double dOut = kd * ((error - preErr) / dT);
```

Рисунок 1. Програмна реалізація цифрового ПІД регулятора

Конструктор класу задає стандартне значення коефіцієнтам та визначає межі виходу регулювання. Для правильної роботи також необхідно визначити час початку роботи. Для цього використовується функція millis, яка є частиною ядра Arduino. Щоб розрахувати керуючий вплив спочатку треба отримати попередні дані. Тому в calcReg, на рис. 1, спачатку здійснюється обрахунок часу, що минув. Для цього через функцію millis отримується поточний час і від нього віднімається preTime. Після обрахування dT в змінну preTime записується оновлене значення теперішнього часу. Для отримання значення помилки від заданого значення необхідного положення віднімається реальне значення, зняте з датчиків, яке ця функція отримує через змінну inp_. Після цієї підготовки починається розрахунок впливу для кожної частини.

3. Керування двигунами гіроскопічної системи

ПІД регулятор не матиме ніякої користі за відсутності можливості керування електромоторами, які безпосередньо впливають на зворотній зв'язок системи. Для керування обраними драйверами двигунів необхідно генерувати сигнал на виведених пінах мікроконтролера. Щоб мати можливість міняти момент обертання електромоторів необхідно міняти напругу на виході. Для вирішення цієї задачі використовується широтно-імпульсна модуляція (ШІМ) [9].

Змінюючи коефіцієнт заповнення можна отримати сигнал із різною середньою напругою. Це добре видно на рис. 2,

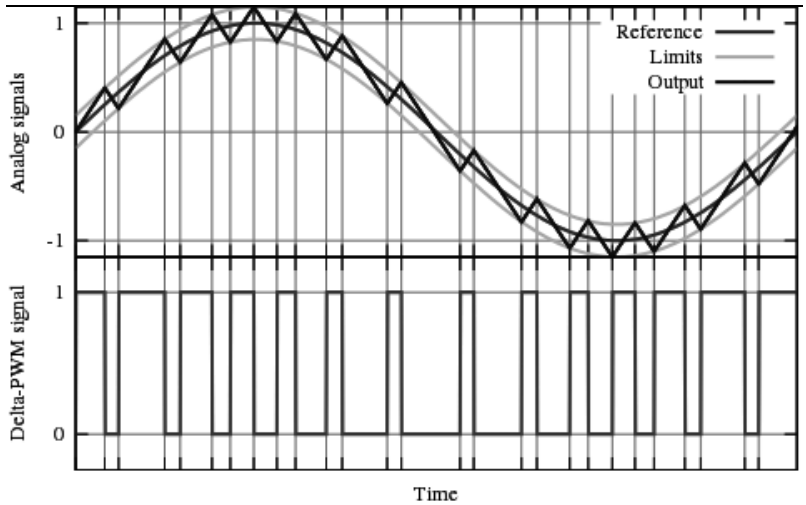


Рисунок 2. Моделювання синусоїдального сигналу в ШІМ[10]

де модулюється синусоїдальний сигнал. Щоб вирівняти вихідний сигнал можна застосувати фільтр низьких частот у вигляді додаткової ємності, проте, позаяк дана система дуже інертна, це не потрібно.

Середню напругу сигналу можна отримати за формулою:

$$\bar{y} = \frac{1}{T}\int_0^T f(t)dt,$$

де $\bar{y}$ – середня напруга, $f(t)$ – функція ШІМ сигналу з періодом T. Оскільки $f(t)$ – це імпульсна хвиля, можна сказати, що її значення буде рівне y_{min}, в періоді, де час $0 < t < D \cdot T$ і y_{max}, коли $D \cdot T < t < T$ відповідно, де D – коефіцієнт заповнення. Тому дане рівняння можна переписати так:

$$\bar{y} = \frac{1}{T}\left(\int_0^{D\cdot T} y_{max}dt + \int_{D\cdot T}^{T} y_{min}dt\right) =$$

$$= \frac{1}{T}(D \cdot T \cdot y_{max} + T \cdot (1-D) \cdot y_{min}) =$$

$$= D \cdot y_{max} + (1-D) \cdot y_{min}.$$

У випадку, якщо y_{min} = 0, то дане рівняння можна спростити навіть більше:

$$\bar{y} = D \cdot y_{max}.$$

Мікроконтролер має можливість генерувати ШІМ сигнал своїх вихідних портах. Оскільки двигуни можуть бути підключені по-різному в залежності від фізичної реалізації, програма має мати можливість легко присвоювати піни для кожного електромотора. Головною метою цього модуля є можливість легко змінювати напругу на входах електродвигунів. Для реалізації цього функціоналу було створено 2 класи – GimbalMot та Motor.

Клас Motor відповідає за керування двигуном. У ньому зберігаються піни, що відведені на підключення цього мотору, напрям обертання та швидкість. Інтерфейс цього класу складається з функції та конструктора. В конструкторі ініціалізуються відведені піни, а функція setO відповідає за встановлення вихідного сигналу на пінах мікроконтролера.

Клас GimbalMot у свою чергу відповідає за керування трьома двигунами гіроскопічної системи. Для цього використовуються три об’єкти вище згаданого класу Motor. Для повороту в осі x, для обертання в осі y та для обертання в осі z. Для ініціалізації необхідно передати на вхід номера шести пінів, по 2 для кожного мотору. Є можливість керувати кожним мотором окремо через відповідні функції setVel[x/y/z].

Збільшення частоти ШІМ необхідне щоб прибрати шум з електромоторів, адже нижчі частоти може почути людське вухо. Після налаштування ШІМ починається налаштування переданих пінів як виходів. Потім із даних виходів створюються по три об’єкти класу Motor для кожної осі. Оскільки в даному випадку зберігається посилання на об’єкт, а не сам об’єкт, в деструкторі необхідно здійснити очищення пам’яті. Тому клас GimbalMot містить деструктор для звільнення пам’яті. Через це відбувається вивільнення пам’яті для кожного з моторів. Таким чином з′являється можливість налаштування потрібного двигуна без зміни роботи інших.

4. Фільтр вхідних даних

Оскільки датчики зазвичай не ідеальні, їхній вихідний сигнал буде під впливом різних шумів, тож необхідно мати фільтр для отримання точніших даних. Для розв′язання даної проблеми було прийнято рішення використати фільтр Калмана. Для того, щоби використовувати фільтр Калмана для оцінювання внутрішнього стану процесу, маючи лише послідовність зашумлених спостережень, необхідно змоделювати процес відповідно до моделі фільтру Калмана. Це означає задання наступних матриць: F_k - моделі переходу станів; H_k - моделі спостереження; Q_k - коваріації шуму процесу; R_k - коваріації шуму спостереження; та іноді B_k - моделі керування для кожного моменту часу, k[12].

$$x_k = F_k x_{k-1} + B_k u_k + w_k,$$

$$w_k \sim N(0, Q_k),$$

де F_k – модель переходу стану для попереднього стану x_{k-1}, B_k - модель впливів керування, що застосовується до вектора керування u_k, w_k – шум процесу, що має

нормальний розподіл з нульовим середнім значенням та коваріацією Q_k;

У момент вимірювання z_k справжнього стану x_k отримується за наступною формулою:

$$z_k = H_k x_k + v_k,$$
$$v_k \sim N(0, R_k),$$

де H_k є моделлю спостереження, що відображає простір справжнього стану у спостережуваний простір, і v_k є шумом спостереження, що, як вважається, є ґаусовим білим шумом з нульовим середнім значенням і з коваріацією R_k.

Фільтр Калмана є рекурсивним, а отже для роботи йому необхідний тільки минулий стан та поточні дані. Це надає перевагу над пакетними фільтрами, які для своєї роботи потребують збір та збереження масиву даних. Щоб зберегти стан фільтру використовуються дві змінні: $\hat{x}_{k|k}$ – оцінка стану в момент часу k при заданих спостереженнях до моменту часу k включно та $P_{k|k}$ – коваріаційна матриця помилок. Алгоритм фільтра Калмана можна розділити на дві частини – стадія передбачення та уточнення.

Стадія передбачення використовує оцінку стану з попереднього моменту часу для отримання оцінки стану на поточний момент часу.

$$\hat{x}_{k|k} = F_k \hat{x}_{k-1|k-1} + B_k u_k,$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k.$$

Розрахунки в цій стадії включають розрахунок передбаченої оцінки стану та коваріацію передбаченої оцінки.

$$\tilde{y}_k = z_k - H_k \hat{x}_{k|k-1},$$

$$S_k = H_k P_{k|k-1} H_k^T + R_k,$$

$$K_k = P_{k|k-1} H_k^T S_k^{-1},$$

$$\hat{x}_{k|k} = \hat{x}_{k-1|k-1} + K_k \tilde{y}_k,$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}.$$

У стадії уточнення необхідно розрахувати наступні значення: відхилення вимірювання, коваріацію відхилення, оптимальний передавальний коефіцієнт Калмана, оновлену оцінку стану, коваріацію оновленої оцінки [12].

Розроблений модуль має наступні функції: getAngle – повертає відфільтроване значення кута (для цього йому необхідно передати значення кута зібране з датчика), кутову швидкість в цій площині та пройдений час в секундах. Функція setAngle відповідає за встановлення початкового кута нахилу, необхідного для майбутніх розрахунків.

Завдяки функції getRate можна також отримати відфільтроване значення кутової швидкості. Для тонкого калібрування фільру використовуються функції setQangle, setQbias та setRmeasure. Q_angle - дисперсія шуму процесу для акселерометра, Q_bias - дисперсія шуму процесу для зміщення гіроскопа, R_measure - дисперсія шуму вимірювання.

У конструкторі класу Kalman здійснюється початкова ініціалізація змінних. Виставляються початкові значення для вищезгаданих Q_angle, Q_bias та R_measure, які в подальшому можна корегувати для кращої роботи. Також заповнюється матриця коваріації помилок P[2][2]. Позаяк на даний момент відоме точне положення та швидкість у початковий момент необхідно заповнити її нульовим значенням.

Алгоритм отримання даних можна умовно поділити на сім кроків. Стадія передбачення включає перші 2 кроки (рис. 3). Перший крок передбачає виконання дискретних рівнянь оновлення часу фільтра Калмана, обраховується змінна rate та angle. Наступним кроком буде обрахування нових значень для матриці коваріації помилок P. Після цього починається стадія уточнення, в якій обчислення здійснюються за формулами, згаданими вище.

Наступною буде стадія уточнення. Крок номер 3 - це обрахунок відхилення вимірювання. Крок 4 включає обрахування коваріації відхилення. Далі у кроці 5 обраховується коефіцієнт Калмана, що на виході являє собою вектор 2 на 1. Після цього необхідно оновити оцінку стану, що на виході дає шуканий кут. Останній крок - це розрахунок коваріації оновленої оцінки, що знову оновлює значення в матриці P для наступного циклу. В кінці функція повертає отриманий кут.

```
float Kalman::getAngle(float newAngle, float newRate, float dt) {

    // Discrete Kalman filter time update equations - Time Update ("Predict")
    /* Step 1 */
    rate = newRate - bias;
    angle += dt * rate;

    // Update estimation error covariance - Project the error covariance ahead
    /* Step 2 */
    P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle);
    P[0][1] -= dt * P[1][1];
    P[1][0] -= dt * P[1][1];
    P[1][1] += Q_bias * dt;
```

Рисунок 3. Фрагмент коду фільтра Калмана, крок 1-2[7]

```
// Discrete Kalman filter measurement update equations - Measurement Update
("Correct")
    // Calculate angle and bias - Update estimate with measurement zk (newAngle)
    /* Step 3 */
    float y = newAngle - angle; // Angle difference

    // Calculate Kalman gain - Compute the Kalman gain
    /* Step 4 */
    float S = P[0][0] + R_measure; // Estimate error
    /* Step 5 */
    float K[2]; // Kalman gain - This is a 2x1 vector
    K[0] = P[0][0] / S;
    K[1] = P[1][0] / S;
    /* Step 6 */
    angle += K[0] * y;
    bias += K[1] * y;

    // Calculate estimation error covariance - Update the error covariance
    /* Step 7 */
    float P00_temp = P[0][0];
    float P01_temp = P[0][1];

    P[0][0] -= K[0] * P00_temp;
    P[0][1] -= K[0] * P01_temp;
    P[1][0] -= K[1] * P00_temp;
    P[1][1] -= K[1] * P01_temp;

    return angle;
```

Рисунок 4. Фрагмент коду фільтра Калмана, крок 3-7[7]

Тестування розробленого фільтра було проведено на генераторі шуму. Як видно з графіку на рис. 4, фільтр Калмана чудово справляється з даним шумом (на графіку – unfiltered) та легко отримує значення (на графіку – filtered), подібне до реального (на графіку – real).

Однак, як бачимо на рис. 5, фільтр може додавати затримку сигналу, тому для отримання точних даних необхідно здійснювати точне калібрування і підбір значень змінних Q_angle, Q_bias та R_measure. Оскільки обрано інерційний вимірювальний пристрій, датчик MPU6050 під час роботи із вбудованим цифровим процесором руху (DMP) сам фільтрує вихідні дані.

5. Тестування

Розроблена керуюча програма мікроконтролера складається з чотирьох відокремлених частин:

модуль зв’язку з інерційним вимірювальним пристроєм;

модуль, що містить реалізацію ПІД регулятора;

модуль керування електродвигунами гіроскопічної системи;

модуль фільтрування вхідних даних.

Для проведення тестування розробленої системи було зібрано робочий макет. Для спрощення створення макету прийнято рішення зробити для тестування систему з одним ступенем свободи.

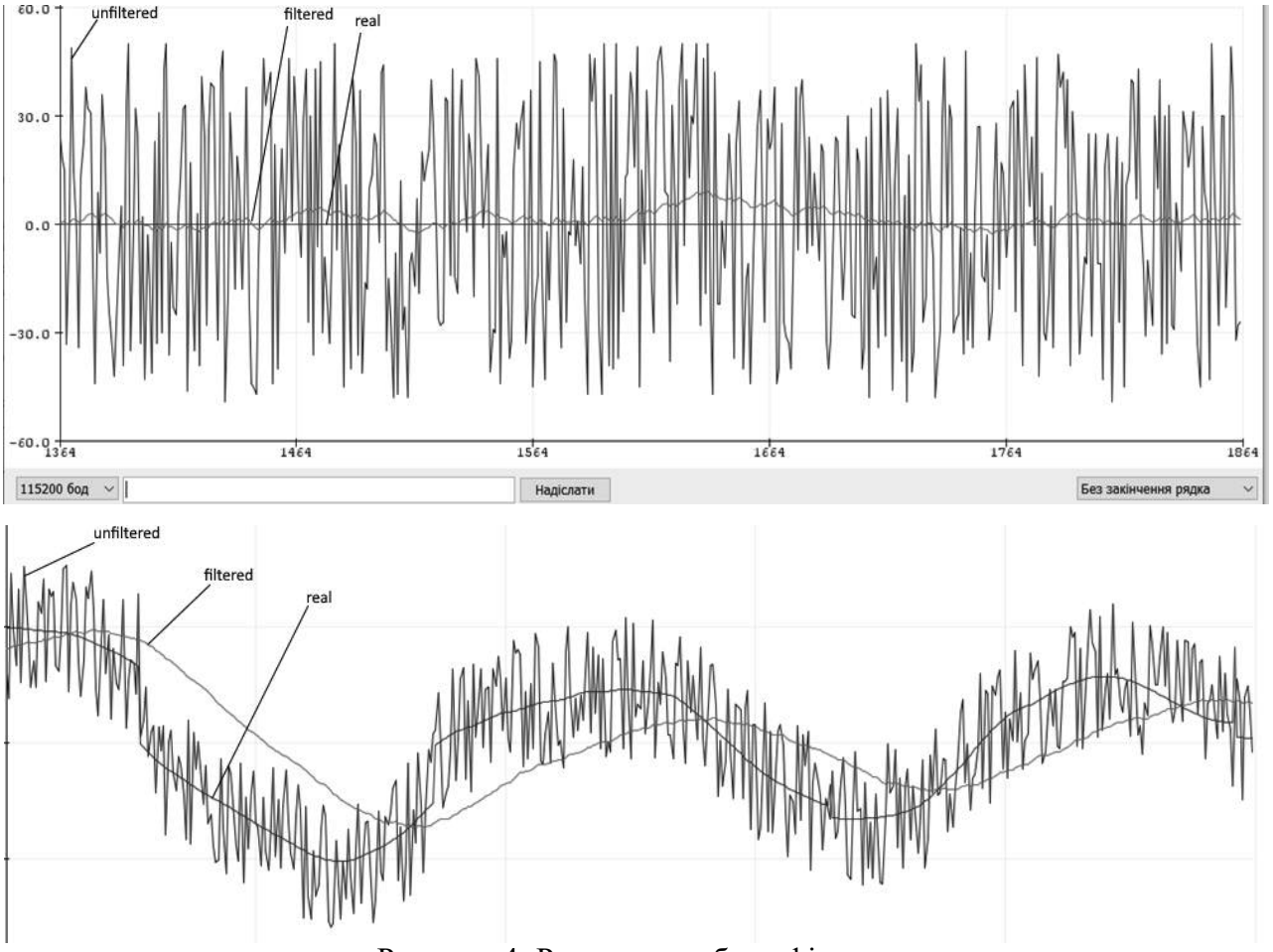


Рисунок 4. Результат роботи фільтру

Для обрахунку пропорційної, інтегральної та диференціальної частин регулятора було вирішено застосувати алгоритм автоматичного підбору. Для цієї задачі обрано релейний метод у парі з методом Зіглера-Нікольса[12]. Метод Зіглера-Нікольса передбачає використання таблиці для підбору коефіцієнтів.

Тип регулятора	K_p	T_i	T_d
Класичний ПІД	$0.6K_u$	$0.5T_u$	$0.125T_u$
Інтегральне правило Песена	$0.7K_u$	$0.4T_u$	$0.15T_u$
Невелике перерегулювання	$0.33K_u$	$0.5T_u$	$0.33T_u$
Без перерегулювання	$0.2K_u$	$0.4T_u$	$0.33T_u$

Підбір коефіцієнтів регулятора проходитиме за відносно простим алгоритмом, що включає поєднання двох методів. Спершу необхідно виставити максимальний вихід регулятора, та очікувати отримання заданого цільового положення, після чого вимкнути вихідний сигнал. Коли появиться похибка, ще раз налаштувати вихід контролера. Повторити ці кроки декілька разів. Далі необхідно обрахувати ультимативний множник, використовуючи значення амплітуди виходу регулятора, та обрахувати період коливань регулювання. Отримавши ці дані можна обрахувати коефіцієнти з використанням метода Зіглера-Нікольса.

Таким чином, було отримано наступний результат:

На даному графіку(рис. 5) зображено перехідний процес системи при наданні короткотривалого фізичного імпульсу на макет системи. З даного графіку можна визначити наступні характеристики перехідного процесу: час перехідного процесу - 0.44с, перерегулювання - 6.2%.

Результати тестування моделі задовільняють поставлені технічні вимоги до її роботи.

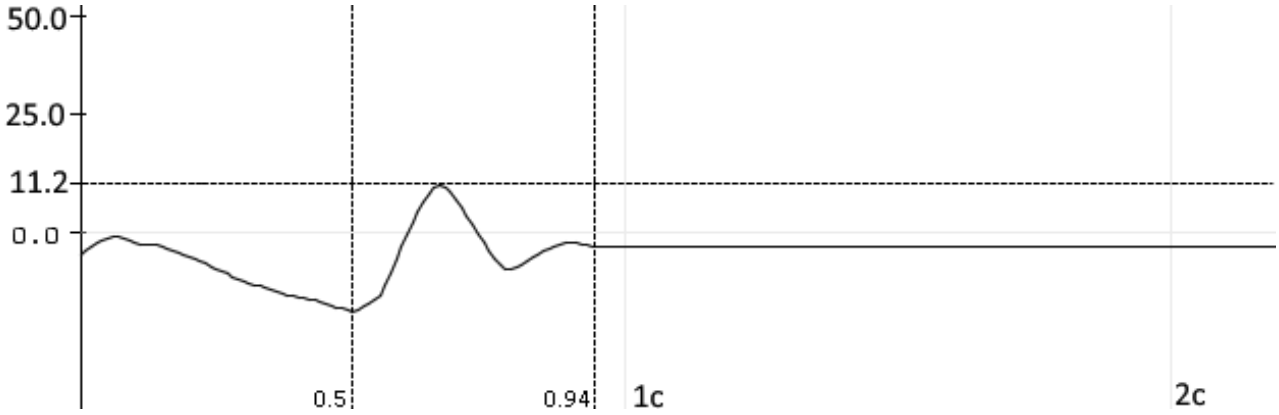


Рисунок 5. Перехідний процес при впливі на систему

Висновки

Розробка модулю стабілізації кута нахилу площини з трьома ступенями свободи була виконана із урахуванням новітніх досягнень у сфері збору даних про положення об'єкта в просторі. Даний модуль був розроблений так, щоб легко замінювати частин модуля та використовувати різні вузли, які відрізняються від описаних у роботі. Для того, щоб поміняти вузол, необхідно змінити модуль у програмі мікроконтролера, відповідальний за нього. За умови дотримання заданого інтерфейсу для зміни буде достаньо поміняти лише один файл. Дані характеристики уможливлюють гнучке застосування даного модуля в багатьох різних сферах.

Перевагами даного модуля є простота реалізації, легкість підключення, гнучкість програми мікроконтролера та можливість використання дешевих та поширених компонентів. Використання новітніх інерційних вимірювальних систем дозволяє легко отримувати точні дані про положення регульованої площини. Щоб мати можливість виконувати роботу навіть у випадку шуму показників був реалізований фільтр даних, що дозволяє працювати із більш давніми моделями датчиків. Для налаштування регулятора було запропоновано використання популярних методик підбору коефіцієнтів ПІД, що значно спрощує запуск системи, яка буде використовувати даний модуль.

У процесі роботи були розроблені структурна, принципова електрична схема, схема алгоритму програми мікроконтролера, обрані та описані основні вузли й елементи.

Для підтвердження дієвості даного модуля було створено розділ, присвячений розробці фізичної моделі. В ньому була сформована структура моделі та продемонстрована можливість стабілізації площини навіть у разі використання неідеальних компонентів.

References

1. *Nyberg, L., & Tjellander, M.* (2017). Camera stabilization (Dissertation). p. 29
2. *John Pardue* (2005) C Programming for Microcontrollers, Knoxville.: Smiley Micros. p. 300
3. MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2 – URL: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf (дата звернення: 05.04.2021)
4. *Welch, Greg & Bishop, Gary* (2006) An Introduction to the Kalman Filter. Proc. Siggraph Course. 8. pp. 1-16 jrowberg/i2cdevlib The I2C Device Library – URL: https://github.com/jrowberg/i2cdevlib (дата звернення: 21.04.2021)
5. *Hamilton, William Rowan* (2000) On quaternions, or on a new system of imaginaries in algebra. Philosophical Magazine. 1844. pp. 489-495
6. Pandanom/StabilizationModule Module for plane stabilization with three degrees of freedom – URL: https://github.com/Pandanom/StabilizationModule (дата звернення: 25.05.2021).
7. *Samotyy V., Telenyk S., Kravets P., Shymkovych V., Posvistak T.* (2018) A real time

control system for balancing a ball on a platform with FPGA parallel implementation. Technical Transactions. Vol. 5. pp. 109-118
8. *Yurkevich, Valery* (2009) PWM PI/PID/PIDF Control for Nonlinear Nonaffine Systems via Singular Perturbation. 2009 International Forum on Strategic Technologies. pp. 185-190.
9. Barr, Michael (2001) Pulse Width Modulation. Embedded Systems Programming. September. pp. 103-104
10. ESP8266 Arduino Core's documentation – URL: https://arduino-esp8266.readthedocs.io/en/latest/ (дата звернення: 07.05.2021)
11. *J. B. Ziegler and N. B. Nichols.* «Optimum settings for automatic controllers» ASME Transactions v64. 1942; pp. 759-768.

Отримано: 24.05.2022

***Про авторів*:**

Безпалько Станіслав Олегович, магістрант 1 року навчання Національного Технічного Університету України «КПІ імені Ігоря Сікорського». https://orcid.org/0000-0001-8550-1792
Шимкович Володимир Миколайович, кандидат технічних наук, доцент кафедри інформаційних систем та технологій Національного технічного університету України «КПІ імені Ігоря Сікорського». Кількість наукових публікацій в українських виданнях – понад 30. Кількість наукових публікацій в зарубіжних виданнях – понад 10. Індекс Хірша – 4. https://orcid.org/0000-0003-4014-2786
Дорошенко Анатолій Юхимович, доктор фізико-математичних наук, професор, завідувач відділу теорії комп'ютерних обчислень, професор кафедри інформаційних систем та технологій Національного технічного університету України «КПІ імені Ігоря Сікорського». Кількість наукових публікацій в українських виданнях – понад 190. Кількість наукових публікацій в зарубіжних виданнях – понад 80. Індекс Хірша – 6. http://orcid.org/0000-0002-8435-1451

Місце роботи авторів:

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», проспект Перемоги 37 та Інститут програмних систем НАН України, 03187, м. Київ-187, проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559
E-mail:
stas110922@gmail.com,
v.shymkovych@kpi.ua,
doroshenkoanatoliy2@gmail.com

УДК 004.43::004.94 https://doi.org/10.15407/pp2022.02.013

А.Ю. Дифучин

ТРАНСЛЯТОР МОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ ПЕТРІ-ОБ'ЄКТНИХ МОДЕЛЕЙ

Високорівневі засоби програмування спрямовані на підвищення швидкості розробки складних програм за рахунок автоматизації рутинних дій програміста, зменшення кількості помилок при кодуванні та зменшення коду в цілому. Візуальне програмування передбачає кодування на основі візуального представлення завдання на виконання обчислень замість текстового. Запропонований транслятор мови візуального програмування Петрі-об'єктних моделей створений відповідно до визначеної формальної граматики мови та формалізму Петрі-об'єктної моделі. Він виконує перетворення візуального представлення моделі в обчислення алгоритму імітації. Перевагами розробленої мови є невеликий алфавіт символів, реалізація можливостей для тиражування об'єктів та зв'язків між ними, універсальність застосування для розробки моделей дискретно-подійних систем.

Ключові слова: транслятор, формальна граматика, стохастична мережа Петрі, алгоритм імітації, Петрі-об'єктна модель

Вступ

Візуальне програмування є напрямком розробки засобів програмного забезпечення, спрямованим на застосування візуальних об'єктів для опису завдання на виконання обчислень. Для того, щоб візуальне представлення стало візуальним програмуванням необхідно гарантувати однозначне перетворення візуального представлення в обчислення. Таке перетворення забезпечує транслятор мови.

Мережі Петрі є ключовим формалізмом для моделювання дискретно-подійних систем, який покриває широкий клас систем від автоматних до стохастичних [1]. У контексті інженерії програмного забезпечення формалізм мереж Петрі є важливим ще й тому, що є загальноприйнятим для розробки паралельних та розподілених обчислень відповідно до стандарту ISO/IEC 15909-1:2004 [2].

Найбільш відомим засобом моделювання мережами Петрі на сьогоднішній день є CPNTools [3]. CPNTools вирішує проблему великої кількості елементів, необхідних для моделювання складних систем, через розширення типів маркерів, які символізують стан системи, та через використання функціональної мови програмування CPN ML для опису стану системи [4]. Комбінування графічного представлення мережі Петрі з елементами функціональної мови програмування зробило представлення моделі складним для сприйняття, оскільки основний зміст функціональності моделі часто прихований за складними функціональними виразами.

Технологія Петрі-об'єктного моделювання розвивається з 2011 року, коли була опублікована праця [5], в якій були розроблені теоретичні засади цього методу моделювання. Перевагою методу є можливість тиражування елементів моделі з однаковою поведінкою та конструювання моделі з фрагментів. У процесі констурювання забезпечуються умови, коли побудована модель має опис стохастичною мережею Петрі, отриманою об'єднанням мереж Петрі фрагментів моделі. Цей доведений теоретично факт є важливим для забезпечення обчислюваності Петрі-об'єктної моделі. Тобто, на відміну від інших існуючих способів конструювання моделі з фрагментів мереж Петрі, зберігається представлення мережею Петрі всієї моделі, не виникають сторонні елементи у представленні моделі, а також необхідність переходити до багаторівневого представлення моделі.

Технологія застосовувалась для моделювання систем, що містять сотні об'єктів. Вона показала достатньо високу швидкодію як у процесі розробки моделі, так і в процесі експериментування з нею. Проте використання виключно редактора мережі Петрі у програмному забезпечен-

ні виявило, що необхідний потужний візуальний інструментарій для конструювання Петрі-об'єктної моделі, оскільки зв'язування Петрі-об'єктів текстовою мовою програмування потребувало значних зусиль, концентрації уваги і часу на налагодження.

Формальна граматика мови Петрі-об'єктного моделювання розроблена у вигляді правил виведення в праці [7] і є контекстно-вільною (або граматикою типу 2 за ієрархією Хомського), однозначною і приведеною.

У даному науковому дослідженні представлена розробка транслятора мови візуального програмування Петрі-об'єктних моделей, наведений опис семантики граматичних виразів мови та визначений спосіб їх перетворення в обчислення.

1. Конструювання Петрі-об'єктної моделі

Петрі-об'єктні моделі застосовують для конструювання моделей дискретно-подійних систем. Під час побудови складної системи, її розбивають на структурні частини, що взаємодіють між собою, розробляють фрагменти моделі і поступово з'єднують їх.

Кожній елементарній події, що відбувається в системі, ставиться у відповідність перехід мережі Петрі. Умови для виникнення події символізують маркери в позиціях. Петрі-об'єктна модель складається з конструктивних елементів Петрі-об'єктів. Кожний Петрі-об'єкт є об'єктом суперкласу, що відтворює функціонування Петрі-об'єкта відповідно до заданої в об'єкті мережі Петрі. Оскільки алгоритм імітації стохастичної мережі Петрі є універсальним, то побудована модель одразу може запускатись на імітацію та виконання експериментальних досліджень. А зусилля, витрачені на побудову моделі, будуть компенсовані зменшенням витрат на написання та налагодження алгоритму імітації.

Детально поняття Петрі-об'єктної моделі викладено у публікації [8]. Петрі-об'єктом є об'єкт-нащадок суперкласу, що містить мережу Петрі та методи для відтворення функціональності об'єкта відповідно до заданої мережі. Зв'язки між Петрі-об'єктами задаються парами ототожнюваних позицій. Усі пари ототожнюваних позицій двох Петрі-об'єктів утворюють конектор, що з'єднує їх.

У візуальному представленні моделі семантичне значення мають триплети елементів Петрі-об'єкт – конектор – Петрі-об'єкт, які повністю визначають Петрі-об'єктну модель:

$$m = \{(o_i, c, o_j) |\, o_i, o_j \in O, i \neq j, \quad c = \{(o_i.p_a, o_j.p_b)\}\},$$

де o_i – j-ий Петрі-об'єкт, c – конектор відповідних Петрі-об'єктів.

Обчислюваність Петрі-об'єктної моделі випливає з того факту, що її функціонування визначається мережею Петрі, отриманою об'єднанням мереж Петрі, з яких вона складена:

$$m.net = \bigcup_j o_j.net,$$

де m – модель, —◇ – символ операції агрегування, $m.net$ – мережа Петрі моделі, $o_j.net$ – мережа Петрі-об'єкта o_j.

2. Формальна граматика мови Петрі-об'єктного моделювання

Для текстових мов програмування теорія формальних мов розроблена достатньо детально [9, 10]. У застосуванні цієї теорії до візуальних мов основним проблемним питанням є спосіб визначення відношення попередній-наступний елемент у граматичному виразі. Іншим проблемним питанням граматики візуальної мови програмування є визначення слова як лексичної одиниці виведення граматичного виразу.

У випадку графового представлення дискретно-подійної моделі відношення попередній-наступний логічно визначається відповідно до розповсюдження подій. Основною змістовою конструкцією, що формується з візуального представлення моделі, є триплети взаємопов'язаних елементів: позиція-дуга-перехід, перехід-дуга-позиція, об'єкт-конектор-об'єкт, об'єкт-конектор-група об'єктів. Тому лексичною одиницею (словом) обрано триплети елементів.

У публікації [7] визначена формальна граматика мови. Алфавіт термінальних символів граматики визначений такими символами (вказані короткі позначення символів):

p – позиція мережі Петрі;

t – перехід мережі Петрі;

a – дуга мережі Петрі;

o – Петрі-об'єкт;

g – група Петрі-об'єктів;

f – група колекцій Петрі-об'єктів;

s – відкрита для з'єднання позиція Петрі-об'єкта;

l – ототожнювач позицій Петрі-об'єктів.

Алфавіт нетермінальних символів:

E – триплет термінальних символів *pat*, що задає одну вхідну дугу переходу;

Q - триплет термінальних символів *tap*, що задає одну вихідну дугу переходу;

N – комбінація кількох нетермінальних символів *E* та *Q*, що може бути інтерпретована в мережу Петрі;

W – комбінація символів, що визначає пару з'єднаних Петрі-об'єктів;

Y – комбінація символів, що визначає з'єднання Петрі-об'єкта з групою Петрі-об'єктів;

X – комбінація символів, що визначає з'єднання Петрі-об'єкта з групою колекцій;

C – конектор Петрі-об'єктів, що складається з кількох ототожнювачів позицій.

Для кожного символа граматики визначене відповідне візуальне його представлення [7].

Правила виведення граматики мови:

(1) $I \longrightarrow NI \mid N$,
(2) $N \longrightarrow EQ \mid QE$,
(3) $N \longrightarrow EN \mid QN$,
(4) $E \longrightarrow pat$,
(5) $Q \longrightarrow tap$,
(6) $N \longrightarrow p \mid pN$,
(7) $I \longrightarrow WI|YI|XI$,
(8) $Y \longrightarrow oCg \mid g$,
(9) $X \longrightarrow oCf \mid f$,
(10) $W \longrightarrow oCo \mid o$,
(11) $I \longrightarrow oI|gI$,
(12) $C \longrightarrow slsC|sls$.

На рисунку 1 наведений приклад візуального представлення моделі, в результаті розбору якого як граматичного виразу отримуємо:

$$I \longrightarrow YI \longrightarrow oCgI \longrightarrow oslsgI \longrightarrow oslsgNI \longrightarrow$$
$$\longrightarrow oslsgNI \longrightarrow oslsgEQI \longrightarrow oslsgEQQI$$
$$\longrightarrow oslsgpatQQI \longrightarrow oslsgpattapQI \longrightarrow$$
$$\longrightarrow ooslsgpattaptapNI \longrightarrow \cdots$$
$$\longrightarrow ooslsgpattaptapEEQQEEQQ \longrightarrow \cdots$$
$$oslsgpattaptaptaptappatpattaptappatpat$$

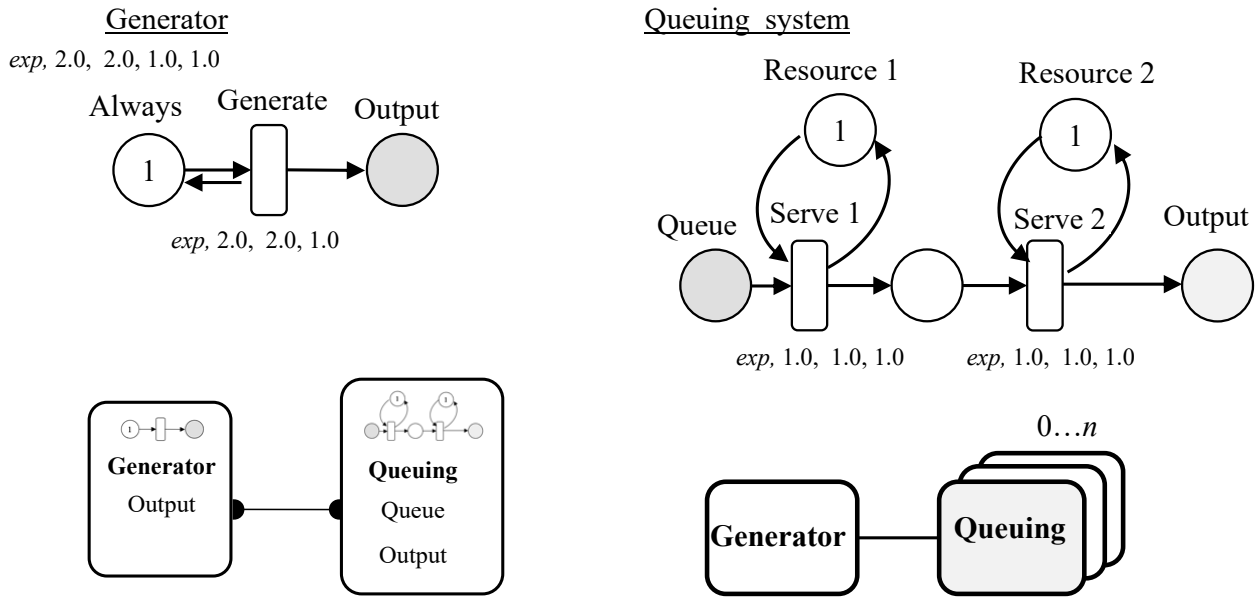


Рис. 1. Візуальне представлення Петрі-об'єктної моделі масового обслуговування: мережі Петрі-об'єктів, ототожнення позицій Петрі-об'єктів, триплет об'єкт-конектор-група.

Тобто модель складається з триплета Петрі-об'єкт – конектор – група Петрі-об'єктів. Мережа Петрі об'єкта Generator складається з одного триплета позиція – дуга – перехід та одного триплета перехід– дуга – позиція. Мережа Петрі об'єкта Queuing з групи складається з 4 триплетів позиція – дуга – перехід та 4 триплетів перехід– дуга – позиція.

3. Транслятор мови

Мова візуального програмування Петрі-об'єктних моделей реалізована у вигляді веб-застосування. Сучасні веб технології надають гнучкості у реалізації візуального редактора Петрі-об'єктних моделей, а віддалене виконання процесу імітації позбавляє залежності від обмеженого локального ресурсу користувача. Візуальний редактор мови програмування Петрі-об'єктних моделей реалізований у клієнтському застосуванні і дозволяє будувати візуальне представлення імітаційної моделі. Перетворення візуального представлення моделі у текстову інтерпретацію та запуск обчислень алгоритму імітації виконується серверним застосуванням. Для цієї мети розроблено транслятор мови.

Транслятор мови програмування включає в себе три послідовні фази: лексичний аналіз, синтаксичний аналіз та семантичний аналіз. Лексичний аналіз візуального представлення моделі - це виявлення у візуальному представленні лексем, тобто наборів візуальних символів. Елементами лексики є візуально представлені символи алфавіту мови: позиція, перехід, дуга, Петрі-об'єкт, група Петрі-об'єктів, відкрита для з'єднання позиція, ототожнювач позицій, конектор. Результат класифікації лексем є вхідною інформацією для синтаксичного аналізатора. Задачею синтаксичного аналізу є побудова синтаксичного дерева, яке являє собою розташування всіх синтаксичних елементів візуального представлення моделі відповідно до правил виведення граматики. На цьому етапі елементи лексики візуальної мови інтерпретуються як трійки термінальних елементів, що визначають зв'язки між елементами мережі Петрі, а також зв'язки між Петрі-об'єктами.

Семантичний аналіз та генерація коду є заключною фазою трансляції, що інтерпретує смисловий зміст та правила виконання конструкцій виділених синтаксичним аналізатором. Оскільки візуальний редактор являє собою клієнтську частину веб застосування, а виконання процесу імітації відбувається на сервері, транслятор виконує перетворення візуального представлення моделі у текстовий формат JSON. Формат передачі даних між клієнтським застосуванням і серверним застосуванням JSON є загальноприйнятим і являє собою текст, що містить структуру даних «ключ – значення». Процес трансляції починається вже під час побудови візуального представлення моделі. В момент створення користувачем зображення елемента мережі Петрі або Петрі-об'єкта, створюється його текстова інтерпретація. В момент з'єднання візуальних елементів між собою будується синтаксичне дерево, що містить трійки елементів. У таблиці 1 наведена структура вихідного JSON мовою TypeScript. Вихідний JSON є результатом роботи транслятора на клієнтській частині та містить інтерпретацію символів алфавіту мови. Під час конструювання моделі формується кінцева текстова інтерпретація моделі у форматі JSON, що відповідає синтаксичному дереву і відправляється на сервер.

Наступний етап трансляції відбувається у серверному застосуванні. Отримавши текстову інтерпретацію моделі серверне застосування здійснює обхід синтаксичного дерева та створення відповідних об'єктів, що передаються в конструктор класу `PetriObjModel` бібліотеки Петрі-об'єктного моделювання [8]. Фінальний етап включає в себе запуск процесу імітації викликом методу `go(t)` класу `PetriObjModel`, аргумент `t` якого вказує на час виконання імітації.

У конекторі важливим є порядок, в якому вказуються з'єднувані об'єкти, оскільки програмно з'єднання реалізується суміщенням адрес пам'яті відповідних позицій об'єктів. Якщо у конструюванні мо-

Таблиця 1

Текстова інтерпретація символів алфавіту мови візуального програмування Петрі-об'єктних моделей мовою TypeScript

Графічне представлення символу	Скорочене позначення символу	Інтерпретація мовою TypeScript

Name
k
	p	`type p = {`
		`  id: number`
		`  name: string`
		`  mark: number`
		`}`

Name
d, r, v, f
	t	`type t = {`
		`  id: number`
		`  name: string`
		`  distribution: string`
		`  delay: number`
		`  deviation: number`
		`  probability: number`
		`  priority: number`
		`}`

q	*a*	`type a = {`
		`  id: number`
		`  name: string`
		`  quantity: number`
		`  placeId: number`
		`  transitionId: number`
		`  info: boolean`
		`}`

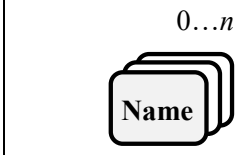
1 — 2	*E*	`type E = [p, a, t]`
1	*Q*	`type Q = [t, a, p]`

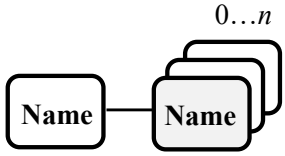
Name
k
	N	`type N = {`
		`  id: number`
		`  name: string`
		`  listE: Array<E>`
		`  listQ: Array<Q>`
		`  sharedPlaces: Array<number>`
		`}`

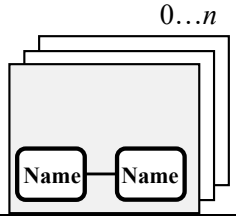
Name
placeName
placeName
placeName
	o	`type o = {`
		`  id: number`
		`  name: string`
		`  petriNetId: number`
		`}`

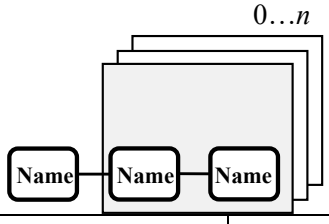
placeName	*s*	`type s = {`
		`  petriObjectId: number`
		`  placeId: number`
		`}`

pName — pName	*sls*	`type sls = [s, s]`
	C	`type C = Array<sls>`
Name — **Name**	*W*	`type W = [o, C, o]`

	*g*	`type g = {`
		`  id: number,`
		`  name: string`
		`  petriObj: o`
		`  quantity: number`
		`}`

	*Y*	`type Y = [o, C, g]`

	*f*	`type f = {`
		`  id: number`
		`  name: string`
		`  listW: Array<W>`
		`  quantity: number`
		`}`

	*X*	`type X = [o, C, f]`

-	*I*	`type model = {`
		`  id: number`
		`  name: string`
		`  time: number`
		`  listObj: Array<o>`
		`  listW: Array<W>`
		`  listY: Array<Y>`
		`  listN: Array<N>`
		`}`

делі використано з'єднання `Link(obj_a, P_a, obj_b, P_b)`, то позиція `P_a` об'єкту `obj_a` буде ототожнена позицією `P_b` об'єкта `obj_b` . Якщо наступним кроком виконано з'єднання `Link(obj_a, P_a, obj_c, P_c)`, то позиція `P_a` об'єкта `obj_a` буде ототожнена позицією `P_c` об'єкта `obj_c`. Тепер адреса позиції `P_a` збігається з позицією `P_c` і попередньо виконане з'єднання з `P_b` втрачено. Щоб організувати з'єднання усіх трьох позицій в одну, потрібно виконати ототожнення позиції `P_b` з `P_a` і в наступному кроці позиції `P_c` з `P_a`. В такому разі всі три позиції вказуватимуть на позицію `P_a` (рис. 1). Саме тому в процесі приєднання до групи в конекторі першим вказується об'єкт групи, а другим – об'єкт, з яким з'єднуються усі об'єкти групи. У конструкторі групи конектор тиражується для всіх об'єктів групи і всі об'єкти будуть отримувати інформацію від одного об'єкта. Отримуємо зв'язок типу one-to-many.

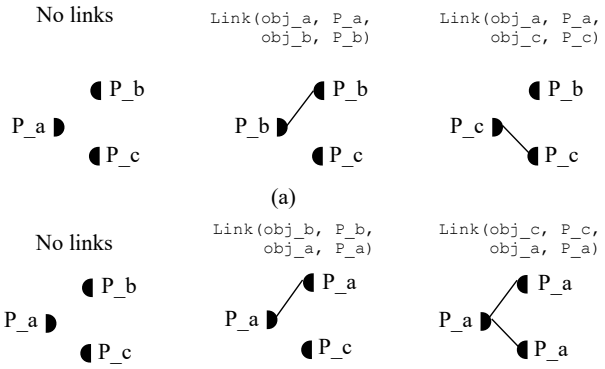


Рис. 2. Встановлення зв'язків між позиці-ями Петрі-об'єктів:
а) послідовність, яка призводить до знищення попереднього зв'язку,
б) послідовність, яка призводить до зв'язку one-to-many

Перетворення Петрі-об'єктної моделі в модель обчислень відбувається з використанням алгоритму імітації на останньому етапі. Оскільки функціонування моделі визначається мережею Петрі, отриманою об'єднанням мереж Петрі усіх об'єктів моделі, то алгоритм імітації виконує обчислення за тими ж правилами, що і алгоритм імітації стохастичної мережі Петрі з часовими затримками з багатоканальними переходами. Проте є суттєвий виграш у складності обчислень через те, що пошук найближчої події здійснюється не по всій мережі, а по моментам найближчої події Петрі-об'єктів. По-друге, через розповсюдження подій в межах одного Петрі-об'єкта (доки не буде досягнута позиція-ототожнювач) перевірка умов виконання події та здійснення події відбувається переглядом тільки елементів одного Петрі-об'єкта замість перегляду всієї мережі Петрі [6].

Імітація моделі здійснюється виконанням таких дій [11]:

- виконати вхід маркерів в переходи для всіх об'єктів;
- доки не вичерпаний час моделювання:
  - визначити момент найближчої події;
  - просунути час у момент найближчої події;
  - змінити стан моделі у відповідності до події.

Зміна стану моделі складається з таких дій:

- виконати вихід маркерів з усіх каналів переходу, для яких час виходу з каналу співпадає з поточним моментом часу;
- визначити переходи, для яких виконана умова входу маркерів, і, за необхідності, розв'язати конфлікт переходів;
- виконати вихід маркерів з обраного на попередньому кроці переходу;
- виконати вхід маркерів в переходи для усіх об'єктів.

Отже, перетворення відбувається через послідовні кроки: 1) транслятор виконує аналіз візуальних символів, які додаються, і визначає лексеми, які утворюються, 2) правильно утворені граматичні вирази (тобто такі, що відповідають визначеним правилам граматики) зберігаються, 3) інтерпретація набору виявлених лексем та формування набору даних моделі у форматі JSON, 4) перетворення моделі з формату JSON у об'єкт класу `PetriObjModel`, 5) запуск обчислень викликом методу `go(t)` класу `PetriObjModel`, аргумент `t` якого вказує на час виконання імітації.

4. Приклад розробки та запуску моделі на обчислення

Через обмежений обсяг матеріалу, який можна розмістити у статті, перетворення візуального представлення моделі у її обчислення наведемо на простому прикладі моделі масового обслуговування (рис. 1). Зауважимо, що завдяки використанню групи об'єктів побудована модель в залежно від параметру *n* відтворює функціонування системи з різною кількістю обробників замовлень, які надходять на обслуговування з генератора - 1. Під час використання звичайних блочних редакторів, які широко використовують у симуляторах, довелось би з'єднувати вручну у візуальному редакторі усі *n* об'єкти. Очевидно, що при великих n це потребує значних зусиль. Окрім того, перегляд усіх *n* зв'язків з метою їх перевірки чи корегування також потребуватиме значних зусиль.

Результат перетворення транслятором візуального представлення моделі у JSON наведений на рисунку 3. Результат перетворення JSON представлення моделі у java код та запуск моделі на імітацію представлений на рисунку 4.

При запуску імітації можемо пересвідчитись у правильності отриманих результатів і переконатись, що час за умови збільшення складності моделі збільшується поліноміально. Оскільки модель допускає теоретичний розрахунок, то за результатом порівняння отриманих результатів імітації з теоретичними робимо висновок про коректність виконаних перетворень. Помилка при часі моделювання 1000000 не перевищувала 5%.

```
Model = {
  "id": 1,
  "name": "SMO",
  "listNet": [{
    "id": 1,
    "name": "Generator",
    "listE": [[{
      "name": "Always",
      "mark": 1,
    }, {
      "quantity": 1,
      "placeId": 1,
      "transitionId": 1
    }, {
      "name": "Generate",
      "distribution": "exp",
      "delay": 2,
      "probability": 1
    }]],
    "listQ": [[{
      "name": "Generate",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "quantity": 1,
      "placeId": 2,
      "transitionId": 1
    }, {
      "name": "Always",
      "mark": 1,
    }], [{
      "name": "Generate",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "quantity": 1,
      "placeId": 2,
      "transitionId": 1
    }, {
      "name": "Output",
      "mark": 0,
    }]]
  }, {
    "name": "Queuing",
    "listE": [[{
      "id": 1,
      "name": "Queue",
      "mark": 0,
    }, {
      "id": 1,
      "quantity": 1,
      "placeId": 1,
      "transitionId": 1
    }, {
      "id": 1,
      "name": "Serve 1",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }], [{
      "id": 2,
      "mark": 0,
    }, {
      "id": 2,
      "quantity": 1,
      "placeId": 2,
      "transitionId": 2
    }, {
      "id": 2,
      "name": "Serve 2",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }], [{
      "id": 3,
      "name": "Resource 1",
      "mark": 1,
    }, {
      "id": 3,
      "quantity": 1,
      "placeId": 3,
      "transitionId": 1
    }, {
      "id": 1,
      "name": "Serve 1",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }], [{
      "id": 4,
      "name": "Resource 2",
      "mark": 1,
    }, {
      "id": 4,
      "quantity": 1,
      "placeId": 4,
      "transitionId": 2
    }, {
      "id": 2,
      "name": "Serve 2",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }]],
    "listQ": [[{
      "id": 1,
      "name": "Serve 1",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "id": 5,
      "quantity": 1,
      "placeId": 2,
      "transitionId": 1
    }, {
      "id": 2,
      "name": "Unnamed",
      "mark": 0,
    }], [{
      "id": 1,
      "name": "Serve 1",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "id": 6,
      "quantity": 1,
      "placeId": 3,
      "transitionId": 1
    }, {
      "id": 3,
      "name": "Resource 1",
      "mark": 0,
    }], [{
      "id": 2,
      "name": "Serve 2",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "id": 6,
      "quantity": 1,
      "placeId": 4,
      "transitionId": 2
    }, {
      "id": 4,
      "name": "Resource 2",
      "mark": 0,
    }], [{
      "id": 2,
      "name": "Serve 2",
      "distribution": "exp",
      "delay": 1,
      "probability": 1
    }, {
      "id": 7,
      "quantity": 1,
      "placeId": 5,
      "transitionId": 2
    }, {
      "id": 5,
      "name": "Output",
      "mark": 0,
    }]]
  }],
  "listObj": [{
    "id": 1,
    "name": "Generator",
    "petriNetId": 1,
  }, {
    "id": 1,
    "name": "Queue",
    "petriNetId": 2,
  }],
  "listW": [],
  "listY": [{
    "id": 1,
    "name": "Generator",
    "petriNetId": 1,
  }, [{
    "petriObjId": 1,
    "placeId": 2
  }, {
    "petriObjId": 2,
    "placeId": 1
  }], {
    "id": 1,
    "name": "Queues",
    "petriObj": {
      "id": 2,
      "name": "Queue",
      "petriNetId": 2,
    },
    "quantity": 100
  }]
}
```


Рис. 3. Результат перетворення транслятором візуального представлення моделі у тексто-вий формат JSON.

```
ArrayList<P> listP = new ArrayList<>();
ArrayList<T> listT = new ArrayList<>();
ArrayList<E> listE = new ArrayList<>();
ArrayList<Q> listQ = new ArrayList<>();
listP.add(new P("Always",1));
listP.add(new P("Queue",0));
listT.add(new T("1", "Generate", "exp", 2.0, 1.0, 1.0, 1));
listE.add(new E(listP.get(0),listT.get(0),new Arc(1)));
listQ.add(new Q(listT.get(0),listP.get(1),new Arc(1)));
listQ.add(new Q(listT.get(0),listP.get(0),new Arc(1)));
PetriSim generator = new PetriSim(new Net("Generator",listP,listE,listQ));

listP = new ArrayList<>();
listT = new ArrayList<>();
listE = new ArrayList<>();
listQ = new ArrayList<>();
listP.add(new P("Queue", 0));
listP.add(new P("Resource 1", 1));
listP.add(new P("", 0));
listT.add(new T("1","Serve 1", "exp", 1.0, 1.0, 1.0, 1));
listE.add(new E(listP.get(0), listT.get(0), new Arc(1)));
listE.add(new E(listP.get(1), listT.get(0), new Arc(1)));
listQ.add(new Q(listT.get(0), listP.get(1), new Arc(1)));
listQ.add(new Q(listT.get(0), listP.get(2), new Arc(1)));
listP.add(new P("", 1));
listP.add(new P("Resource 2", 0));
listT.add(new T("2","Serve 2","exp", 1.0, 1.0, 1.0, 1));
listE.add(new E(listP.get(2), listT.get(1), new Arc(1)));
listE.add(new E(listP.get(3), listT.get(1), new Arc(1)));
listQ.add(new Q(listT.get(1), listP.get(3), new Arc(1)));
listQ.add(new Q(listT.get(1), listP.get(4), new Arc(1)));
PetriSim smo = new PetriSim(new Net("Queueing", listP, listE, listQ));

Group group = new Group(smo, numObjs);
Connector connector = new Connector(new Link(smo, 0, generator, 1));
ArrayList<PetriSim> listO = new ArrayList<>();
ArrayList<W> listW = new ArrayList<>();
ArrayList<Y> listY = new ArrayList<>();
listY.add(new Y(group, generator,  connector));
PetriObjModel model = new PetriObjModel("SMO",listO, listW, listY);
model.go(timeModeling);
model.printResultsForAllObj();
```

Рис. 4. Результат перетворення транслятором текстового формату JSON у java код.

Висновки

Розроблений транслятор мови візуального програмування Петрі-об'єктних моделей, який виконує перетворення візуального представлення Петрі-об'єктної моделі у текстову мову програмування та запускає на обчислення. Лексичний аналіз виконується під час створення візуального представлення моделі у клієнтському застосуванні. Семантичний аналіз і виконання обчислень моделі виконується серверним застосуванням.

Поняття Петрі-об'єктної моделі для тиражування зв'язків розширено поняттями конектор Петрі-об'єктів, група Петрі-об'єктів, колекція Петрі-об'єктів.

Наведено приклад, в якому представлено усі етапи перетворення віуального представлення моделі транслятором мови програмування. Порівняння результатів імітації з теоретичними доводить коректність виконаних перетворень.

References

1. Petri nets : fundamental models, verification, and applications. Michel Diaz (ed.) Willey, 2009. 581 p.
2. ISO/IEC 15909-1:2004 Systems and software engineering — High-level Petri nets — Part 1: Concepts, definitions and graphical notation. [Online] – Available from: https://www.iso.org/standard/38225.html, last accessed 2022/07/2.
3. CPNTools [Online] – Available from: https://cpntools.org/, last accessed 2022/07/2.
4. Jensen, K., Kristensen, L.M. (2009). CPN ML Programming. In: Coloured Petri Nets. Springer, Berlin, Heidelberg. https://doi.org/10.1007/b95112_3.
5. Стеценко И.В. (2011) Теоретические основы Петри-объектного моделирования систем. Математичні машини і системи, 136-148.
6. Stetsenko I. V., Dorosh V. I., Dyfuchyn A. Petri-object simulation: Software package and complexity. 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015, pp. 381-385. (Scopus) https://doi.org/ 10.1109/ IDAACS.2015.7340762.
7. Дифучин А.Ю., Стеценко І.В., Жаріков Е.В. Граматика мови візуального програмування Петрі-об'єктних моделей // Проблеми програмування. – Київ, 2021. - №4. – С.82-94. https://doi.org/10.15407pp2021.04.082
8. Stetsenko, I.V., Dyfuchyn, A.: Petri-object Simulation: Technique and Software. Information, Computing and Intelligent Systems 1, 51-59 (2020). https://doi.org/10.20535/2708-4930.1.2020.216057
9. Kimball J. P. The Formal Theory of Grammar Prentice-Hall, 1973 - 127 p.
10. Becerra-Bonache L., Bel-Enguix G., Jiménez-López M. D., Martín-Vide C. (2018). Mathematical Foundations: Formal Grammars and Languages. In: The Oxford Handbook of Computational Linguistics, Second Edition (2nd edn) Ruslan Mitkov (ed.) [Online] – Available from: https://academic.oup.com/edited-volume/42643/chapter-abstract/358148992?redirectedFrom=fulltext, last accessed 2022/07/2.
11. Stetsenko I.V. (2017) Parallel Algorithm for Petri Object Simulation. Cybernetics and Systems Analysis. 53(4), 605–614.

Отримано 16.07.2022

***Про автора**:*

Дифучин Антон Юрійович,
аспірант 4 року навчання кафедри
інформатики та програмної інженерії
НТУУ "КПІ імені Ігоря Сікорського".
Кількість наукових публікацій
в українських виданнях – 3.
Кількість наукових публікацій
в іноземних виданнях – 3. Індекс Хірша
– 1. https://orcid.org/0000-0002-1722-8840

Місце роботи автора:

Національний технічний університет
України «Київський політехнічний
інститут імені Ігоря Сікорського»,
03056, м. Київ,
проспект Перемоги 37.
Тел.: (044) 236-9651
e-mail: difuchin@gmail.com

УДК 502:004.45 (075.8) http://doi.org/10.15407/pp2022.02.022

М.О. Сидоров

ДИСЕРТАЦІЯ МАГІСТРІВ З ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ – ОБЬЄКТ, ПРЕДМЕТ, ЗМІСТ ДОСЛІДЖЕНЬ

На відміну від закордонних університетів, в Україні протягом багатьох років склалася й існує практика формулювання мети дослідження, застосовуючи поняття об'єкта і предмета дослідження. І це, як показав час, важливий етап роботи, від виконання якого залежить результативність дослідження. Загальне й тотальне проникнення програмного забезпечення в життя з одного боку, а з іншого - поява, спеціалізацій інженерії програмного забезпечення значно ускладняють умови виконання цього етапу магістерського дисертаційного дослідження. Крім цього, в інженерії програмного забезпечення застосовуються наукові методи досліджень, які бажано мати на увазі і використовувати для виконання магістерського дисертаційного дослідження. Тому актуальними є питання щодо формулювання об'єкта і предмета досліджень та застосування доказових методів їх виконання. За мету цієї статті ставилося передовсім, надання рекомендації на основі фундаментальних положень інженерії програмного забезпечення щодо формулювання об'єкта і предмета магістерської дисертації зі спеціалізації в контексті інженерії програмного забезпечення. А також, вказати на застосування в магістерської дисертації наукових методів доказових досліджень і звернути увагу на оформлення результатів. Досягнення мети статті розглядається в контексті причин і умов, які історично склалися в навчанні і відповідних проблем, які виникли в навчальних планах, виборі тем і змісті досліджень магістерських дисертацій. Одна з причин, яка має об'єктивний характер, виникає в контексті сполучення домену інженерії програмного забезпечення з прикладними доменами, яке є основою відповідної спеціалізації. Стаття розрахована на магістрів зі спеціалізацій інженерії програмного забезпечення та їхніх наукових керівників.

Ключові слова: інженерія програмного забезпечення, навчання, магістерські дослідження, науково-експериментальні методи досліджень.

Вступ

Основи інженерії програмного забезпечення почали закладатися 1968 року після першої конференції, що відбулася в Гармиші (Німеччина) [1]. Вже на цій конференції було звернуто увагу на гостру необхідність підготовки кадрів, нестача яких була однією зі складових кризи програмного забезпечення, що на той час мала місце [2].

В Україні до 2006 року фахівці з інженерії програмного забезпечення не готувалися. У рамках бакалаврату комп'ютерні науки вивчали програмісти, які після закінчення університету переважно самостійно або на курсах набували додаткові знання та вміння, щоб більшменш відповідати вимогам не тільки, щодо кодувальника програм, а і щодо інженера з програмного забезпечення. На підприємствах тоді вже створеної Асоціації «Інформаційні технології України» (https://itukraine.org.ua/) відчувався гострий брак справжніх фахівців, на ринку праці їх було вкрай мало. Системного розуміння причин ситуації не було. Вважалося, що університети погано готують фахівців. Тому великі компанії самі почали підготовку кадрів.

Слід зазначити, що в Україні, в академічному середовищі ще раніше були створені передумови для вирішення проблеми в контексті інженерії програмного забезпечення. До цих передумов слід віднести роботи, які велися в Інституті кібернетики АН УРСР, здійснювалися дослідження і створювалися програмні засоби для реалізації так званої технології програмування, під якою розумілася саме інженерія програмного забезпечення [3]. Глушков В.М. підтримував ці роботи [4]. До Києва приїжджали відомі вчені, які працювали у цій галузі, як радянські, так і закордонні. Зокрема, А. Єршов, Є. Дейкстра. 1978 року проведено Першу Всесоюзну конференцію з технології програмування [5]. 1995

року в Інституті кібернетики АН УРСР була захищена перша в пострадянській Україні докторська дисертація, присвячена вирішенню актуальних на той час завдань інженерії програмного забезпечення, що належали до повторного використання програмного забезпечення [6].

2000 року в Національному авіаційному університеті було відкрито першу в Україні кафедру інженерії програмного забезпечення [7], а в Асоціації «Інформаційні технології України», до якої тоді належали тільки підприємства індустрії програмного забезпечення України, була введена позиція і обрано віце-президента для зв'язку з університетами. Як виявилося пізніше, він мав добре розуміння зв'язку проблем Асоціації з відсутністю підготовки фахівців саме з інженерії програмного забезпечення. Тоді разом зі створеною кафедрою і Асоціацією почалися роботи як у викладацькому середовищі, так і в індустрії з роз'яснення необхідності підготовки кадрів з інженерії програмного забезпечення. Таким чином було розпочато підготовку до відкриття в Україні бакалаврату з інженерії програмного забезпечення [8].

У контексті цих робіт у Національному авіаційному університеті кафедрою інженерії програмного забезпечення почав видаватися науковий журнал «Інженерія програмного забезпечення» [9], проводилася щорічна науково-технічна конференція студентів та аспірантів з однойменною назвою. Також кафедрою інженерії програмного забезпечення та Асоціацією «Інформаційні технології України» було видано публікації та проведено низку семінарів і конференцій [10 - 12].

Навесні 2006 року бакалаврат «Програмна інженерія» було відкрито, а восени здійснено перший набір студентів. До відкриття у Національному авіаційному університеті було розроблено навчальний план, на основі шаблону N2S-1c з [13]. План було прийнято на конференції завідувачів кафедр, які готували бакалаврів та фахівців із комп'ютерних наук. Ці кафедри, відповідно до рекомендації міністерства освіти були зобов'язані переходити на підготовку бакалаврів з напряму «Програмна інженерія» (інженерія програмного забезпечення). Процедура обговорення і прийняття цього плану висвітлила значні проблеми (що є і будуть в майбутньому), які пов'язувалися з відсутністю педагогічних кадрів, із розумінням, як готувати саме інженерів з програмного забезпечення.

Зараз більшість класичних та політехнічних університетів України готують бакалаврів за спеціальністю 121 – «Інженерія програмного забезпечення» та магістрів за відповідними спеціалізаціями. В Україні створено освітні стандарти для підготовки бакалаврів та магістрів [14]. На основі спеціальності 01.05.03 – «Математичне та програмне забезпечення обчислювальних машин, комплексів та мереж» створено докторантуру філософії спеціальності «Інженерія програмного забезпечення».

Але проблеми 2006 року залишилися і ненайкраще впливають на підготовку фахівців і саме магістрів.

Проблеми підготовки магістрів з інженерії програмного забезпечення

2006 року у Харкові на конференції з введення бакалаврату інженерії програмного забезпечення в Україні, групою з Києва (Національний авіаційний університет та Асоціація «Інформаційні технології України») було запропоновано навчальний план підготовки (Табл. 1, наведено англійською мовою [13]).

Учасники конференції зустріли план без ентузіазму. Пояснити це можна трьома причинами:

1. Завідувачі кафедр розуміли, що на кафедрах немає викладачів, спроможних викладати нові дисципліни за планом, і їх була переважна більшість;

2. У плані було показано різке скорочення кредитів (навчальних годин) у порівнянні із планами комп'ютерних наук, на такі дисципліни як математика, фізика, відсутня також низка дисциплін, зокрема, з електротехніки, нарисної геометрії тощо;

Таблиця1.

Software engineering, Fundamental and professional subjects

	Name of subjects	***Credits ECTS***
2.1	Mathematical Analysis	3,00
2.2	Linear Algebra and Analytic Geometry	3,00
2.3	Mathematical Statistics	3,00
2.4	Basics of Discrete Mathematics	5,00
2.5	Discrete Structures	3,50
2.6	Physics	4,00
2.7	Empirical Methods of Software Engineering	4,00
2.8	Fundamental of Programming	7,00
2.9	Object Oriented Programming	7,00
2.10	Algorithm and Data Structures	5,00
2.11	Software Construction	4,00
2.12	Software Architecture and Design	
2.13	Computer Architecture	5,00
2.14	Software Requirements Analysis	5,00
2.15	Social Safety and Labor Protection	4,00
2.16	Software economic	1,50
2.17	Software Modeling	5,00
2.18	Computer Networks	3,00
2.19	Operating Systems	6,00
2.20	Group Dynamics and Communications	2,00
2.21	Introduction to Software Engineering	7,50
2.22	Database	6,00
2.23	Human Computer Interaction	3,50
2.24	Software Quality and Testing	4,00
2.25	Program and Data Security	3,50
2.26	Professional Software Engineering Practice	2,50
2.27	Design Practice	1,50
2.28	Software Project Management	3,00
		####

3. У плані також були відсутні дисципліни, які традиційно викладалися в комп'ютерних науках, як-от, чисельні методи, основи прийняття рішень, дослідження операцій, інформаційні технології і т. і.

Проте групі з Києва вдалося знайти підтримку деяких завідувачів кафедр, і запропонований план було взято за основу. Пізніше цей план було рекомендовано Навчально-методичною підкомісією до застосування в стандарті при відкритті бакалаврату «Програмна інженерія».

Але, позаяк, план мав рекомендаційний характер, то і застосовували його в повному обсязі тільки окремі університети, наприклад, Національний авіаційний університет. На жаль, пізніше, коли почали прийматися стандарти навчання як першого, так і другого рівня в них були відсутні навчальні плани. Їх опосередковано представляли переліки компетенцій, вмінь та результатів навчання і освітні програми. Реалізувати їх можна було формально, шляхом створення будь-якого плану, зручного кафедрі. І зараз, зазвичай, ці плани найчастіше віддалено пов'язані з інженерією програмного забезпечення. Пояснюється це тим, що стан з відсутністю компетентних викладачів, на жаль, не змінився.

Наразі залишаються умови, через які виникли проблеми і з підготовкою магістрів. Проблеми виявилися знову, ще і в навчальних планах і виборі тем, і в змісті досліджень магістерських дисертацій. Ситуація, що склалася, має три причини. Дві є наступними (про третю трохи згодом):

1. Підготовка бакалаврів тільки частково відповідає спеціальності;

2. Некомпетентність в інженерії програмного забезпечення викладачів–наукових керівників магістрів.

Як бачимо, обидві причини сягають корінням у минуле. Наслідком першої причини є неготовність студентів до грамотного розв'язання завдань інженерії програмного забезпечення. Наслідком другої причини є вибір тем магістерських досліджень, далеких від інженерії програмного забезпечення. Ці ж причини «працюють» у бакалаврських дипломних роботах. Наприклад, типові теми для бакалаврських робіт - це будь який Web-сайт, зокрема, засоби моніторингу продажу квитків, контролю артеріального тиску, серцево-судинної системи, розташування сервісів у місті тощо.

Розглядаючи навчання магістрів, звернемо увагу на питання вибору об'єкта та предмета дослідження, а також

на структуру та зміст дисертації. Сподіваємося, що відповіді на ці питання, які пропонуються в статті, допоможуть науковим керівникам і студентам відповідних спеціалізацій Інженерії програмного забезпечення.

Аналіз літератури

Усі праці, близькі за темою цій статті, можливо поділити на три групи.

До першої групи віднесемо праці, які стосуються становлення навчання інженерії програмного забезпечення як окремої спеціальності [14 - 19]. До другої групи - праці, які стосуються змісту навчання, саме інженерії програмного забезпечення як, скажімо, окремого напряму підготовки (бакалаврату) [20 - 27]. До третьої групи віднесемо праці, що стосуються підготовки магістрів [28 - 34]. Зрозуміло, що найбільш близькими до теми цієї статті є праці третьої групи.

Зміст праць першої групи найчастіше зосереджувався на ознаках інженерії програмного забезпечення як інженерної діяльності [17-19]. Ознаки зазвичай показувалися на прикладі відмінності інженерії програмного забезпечення від комп'ютерних наук [15]. Це пов'язано с відокремленням свого часу, інженерії програмного забезпечення як освітньої і виробничої спеціальності саме від комп'ютерних наук.

Визначається, що, по-перше, ці дві галузі діяльності відрізняються за методом творчого мислення. Для розв'язання завдань комп'ютерних наук застосовується метод конвергентного мислення, коли для розв'язання передбачаються точні (часто математичні) інструкції, тож мається на увазі, що існує єдине рішення. Це вказує на те, що задачі комп'ютерних наук мають переважно теоретичне підґрунтя. Для вирішення завдань інженерії програмного забезпечення застосовується метод дивергентного мислення, коли для розв'язання пропонується декілька рішень. Серед рішень обирається одне й до його реалізації застосовується метод конвергентного мислення. Водночас на вибір єдиного рішення впливають чимало факторів, більшість з яких пов'язані а практичним підґрунтям розв′язуваного завдання.

По-друге, результат розв'язання завдання інженерії програмного забезпечення завжди є продуктом, у якого є користувач. Продукт створюється в контексті життєвого циклу і має задовільняти низку вимог, характерних для інженерії (конструкція продукту, якість, стандарти, документованість, супровід, вплив на зовнішнє середовище). Одночасно, частина робіт першої групи присвячена з'ясуванню поняття «інженер із програмного забезпечення» [17, 18].

Праці другої групи насамперед було присвячено з'ясуванню навчання як переходу від окремої або декількох дисциплін до системно сформованого переліку дисциплін, спрямованого на навчання спеціальності за відповідним змістом. Дискусії навколо змісту навчання засновані на обговоренні або використанні SWEBOK [21]. Для побудови навчальних планів і програм переважна більшість університетів почала застосовувати рекомендації SE2004 [14].

На сьогодні інженерія програмного забезпечення - це систематизований, регламентований і квантифікований підхід до реалізації процесів створення, експлуатації, супроводу та утилізації програмного забезпечення. До того ж процеси, ресурси та програмні продукти інженерії програмного забезпечення мають відповідати заданим технічним, економічним, соціальним, правовим вимогам, а також вимогам стійкого розвитку, зокрема, «зеленим» [35].

Праці третьої групи спрямовані на підготовку з інженерії програмного забезпечення з урахуванням існуючих практик - прикладних доменів [28-33]. Хоча дослідження в інженерії програмного забезпечення, безумовно, також потребують підготовки магістрів, саме наявність прикладних доменів враховується в підготовці магістрів переважною кількістю університетів, і вона приводить до появи відповідних спеціалізацій спеціальності Інженерія програмного забезпечення. Водночас існують роботи, де ці практики класифіковані, зокрема,

шляхом аналізу понад десяти тисяч виконаних проєктів [31]. Подібні класифікації забезпечують аргументовану базу для вибору спеціалізацій щодо підготовки магістрів. Але, як показує аналіз і досвід автора, наявність прикладних доменів часто призводить до абсолютного зсуву акцентів досліджень магістерських дисертацій з об'єктів і предметів інженерії програмного забезпечення в прикладні домени. Тому зауважимо, що саме зсув змісту навчання магістрів і, зокрема, досліджень магістерських дисертацій з інженерії програмного забезпечення на окремі прикладні домени є третьою причиною появи питань, які розглядаються в статті. Таким чином, до проблем підготовки студентів і некомпетентності викладачів додається проблема сполучення домену інженерії програмного забезпечення з прикладними доменами, наприклад [29], військовим – застосунки для військових; з доменами систем – застосунки для управління пристроями; комерційним – застосунки для обробки інформації різноманітного представлення (текстового, графічного, табличного, планування ресурсів тощо); аутсорсинговим доменам – застосунки, які виконуються, мають відмінності в зв'язку з контрактними відношеннями; інформаційних систем – застосунки для управління бізнес функціями.

Разом с тим, огляд літератури показує, що в умовах, які склались, відсутні роботи щодо особливостей підготовки магістерських дисертацій з інженерії програмного забезпечення в умовах, які склалися. Водночас, якщо перши дві причини проблем підготовки магістрів можна і потрібно усувати, то третя причина - об'єктивна і її треба враховувати під час навчання магістрів.

Це стосується визначення об'єкта й предмета дисертації, назви, змісту та опису досліджень і результатів з урахуванням наявності прикладного домену. На відміну від закордонних університетів, в Україні протягом багатьох років склалася й існує практика формулювання цілей дослідження, застосовуючи поняття об'єкта і предмета дослідження. І це, як показав час, важливий етап, від виконання якого залежить результативність дослідження.

Магістерська дисертація. Об'єкт та предмет

Основи наукових досліджень вчать, що, перш ніж братися безпосередньо за виконання досліджень необхідно визначити об'єкт і предмет дослідження.

Об'єкт дослідження визначається шляхом встановлення найбільш загального явища в домені, в контексті якого буде виконуватися дослідження. Це даність, яка легко встановлюється, якщо у дослідника достатньо знань про сутність дослідження і досліджуваного домену. Для спеціалізацій, що розглядаються, це знання з домену - Інженерія програмного забезпечення.

Предмет дослідження належить до одного або декількох аспектів явища, яке визначено в якості об'єкта дослідження і визначається метою дослідження, на основі особливостей прикладного домену. Для спеціалізацій, що розглядаються, цей домен може бути, зокрема, наступним - Інформаційні системи і обробка даних (SAS.inf Information systems and data processing [13]).

Правильність вибору об'єкта і предмета дослідження впливає з одного боку на результативність дослідження, а з іншого - на можливість правильної оцінки отриманих результатів.

Якщо об'єкт і(або) предмет дослідження обрано невірно, тоді мають місце два випадки.

У першому випадку виявляється, що отримані результати аж ніяк не є тими, що мають бути отримані в досліджуваному домені.

У другому випадку, експерти (екзаменаційна комісія), увага яких спрямовуватиметься на оцінку результатів дослідження, можуть виявитися не спеціалістами в домені, в якому, за припущенням дослідника, проводилося дослідження. Як наслідок - оцінити результати експерти не зможуть.

Зв'язок об'єкта і предмета мусить бути безпосереднім і вимагати, аби

Рисунок 1. Життєвий цикл

об'єкт і предмет були на перетині доменів спеціалізації, а саме, як-от Інженерія програмного забезпечення й Інформаційні системи і обробка даних. Запропонуємо рекомендації щодо визначення цього зв'язку.

Для визначення зв'язку між об'єктом і предметом необхідна інформація для уточнення поняття об'єкта дослідження. Будемо це робити, використовуючи поняття життєвого циклу як системоутворюючого в інженерії програмного забезпечення (Рис.1).

Одночасно, використовуємо те, що всі фази життєвого циклу мають три складові - продукт, процес і ресурс (Рис. 2).

Таким чином, конкретизуючи можливий об'єкт дослідження, будемо розглядати наступне (рис.3): продукти (робочі продукти - результати виконання процесів фаз життєвого циклу; програмні продукти - результати виконання життєвого циклу при створенні програмного забезпечення); процеси (процеси фаз життєвого циклу); ресурси, розділяючи їх на дві частини, ресурси I - програмно-реалізовані ресурси (інструменти і артефакти, наприклад, тести, представлення стилів, документи) і ресурси II. До ресурсів II віднесемо підходи і методи, використовувані в інженерії програмного забезпечення. Наприклад, підхід компонентного створення і супроводження, метод Scram, щодо організації команд (організацій) за методологію Agile. В процесі конкретизації обов'язково маємо на увазі наступні розділи інженерії програмного забезпечення: різновид інженерії – пряма, зворотна, емпірична (метричне забезпечення); горизонтальні процеси і відповідні продукти - документування, валідація, верифікування, керування, гарантування якості; економіка програмного забезпечення; зелене програмне забезпечення і сталий розвиток; культура інженерії програмного забезпечення і правові питання; екосистеми програмного забезпечення.

Наведені уточнення об'єкта дослідження слід розглядати в контексті чотирьох під доменів спеціалізації інженерії програмного забезпечення:

- прикладному, для якого створюється програмне забезпечення (програмний продукт). Наприклад, згідно з спеціалізацією, це інформаційні технології (представлена відповідними знаннями XBoK, як-от, для інформаційних систем і технологій це CBoK [36]);

- проблемному, в якому розв'язуються нові, чи удосконалюються рі-

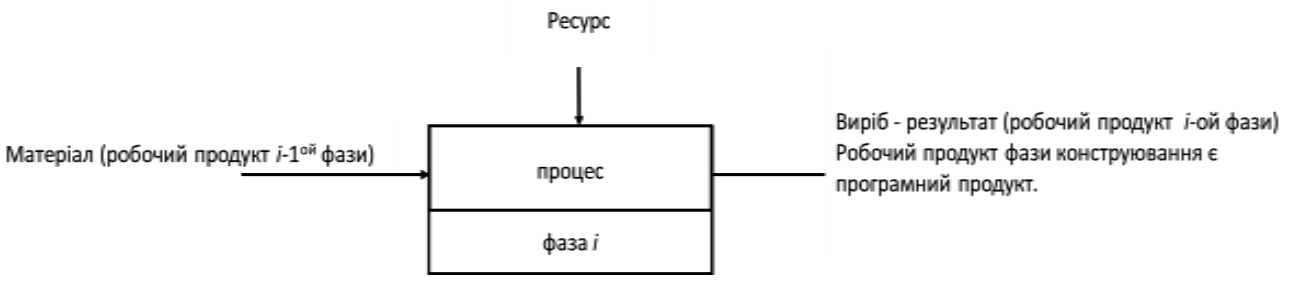


Рисунок 2. Складові фаз життєвого циклу

Рисунок 3. Об'єкт дослідження. Класифікація

шення існуючих, наукові завдання щодо створення і супроводження програмного забезпечення (програмний продукт) - наукова частина домену інженерії програмного забезпечення (представлена SWEBoK [21]);

- реалізаційному, в якому створюється програмне забезпечення (програмний продукт) – інженерна частина домену інженерії програмного забезпечення (представлена SWEBoK);

- домену оцінювання і впровадження результатів, де перевіряються окремі властивості отриманих результатів та досліджуються питання їх впровадження – практична частина домену інженерії програмного забезпечення.

Зв'язки цих доменів визначають структуру магістерського дослідження (рис. 4).

Таким чином, для формулювання об'єкта, предмета та назви дисертації необхідно виконати наступні три кроки (рис. 5):

Аналізуючи завдання дослідження, яке сформульовано в прикладному домені, обираємо в домені інженерії програмного забезпечення об'єкт дослідження,

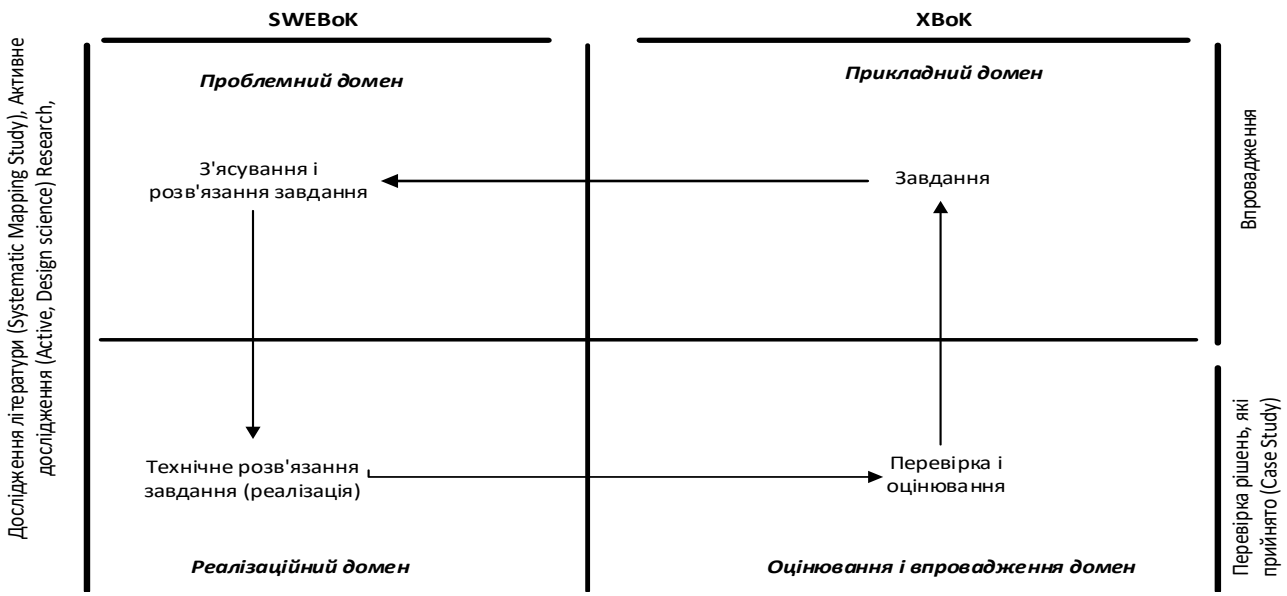


Рисунок 4. Структура магістерського дослідження

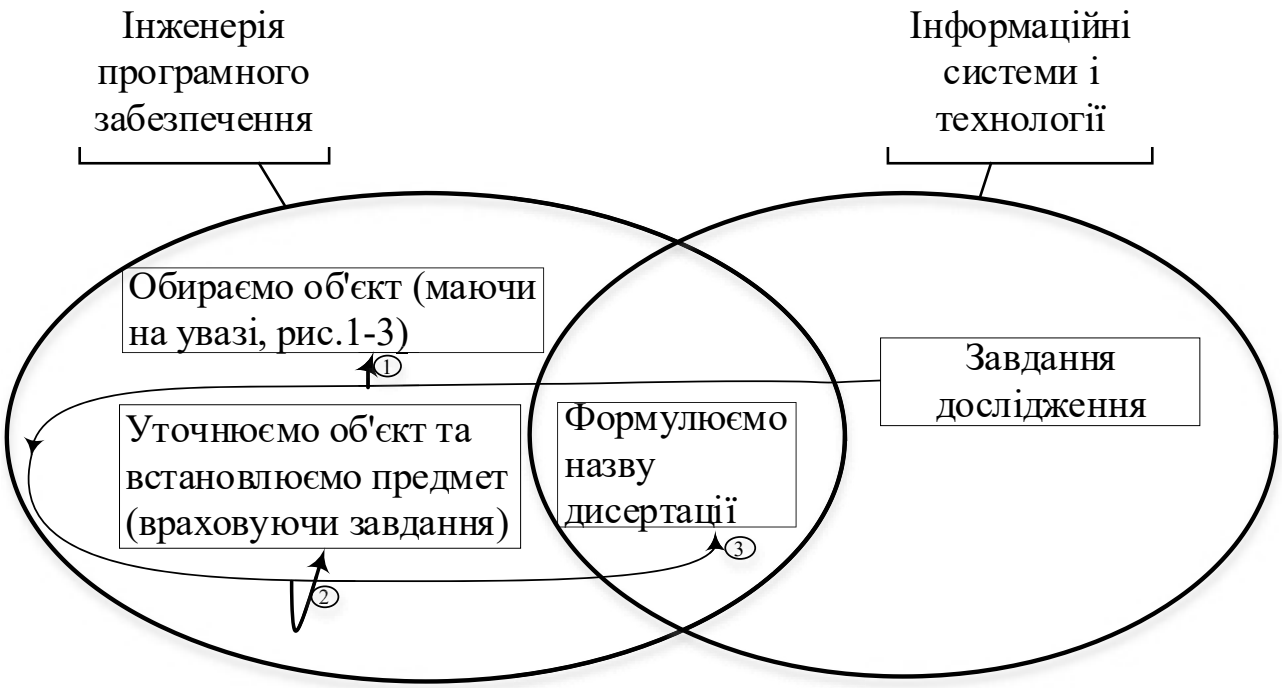


Рисунок 5. Зв’язок завдання-об’єкт-предмет-назва дисертації

спираючись на класифікацію, наведену на рис. 1-3.

Продовжуючи аналіз завдання, уточнюємо (якщо потрібно, шляхом виконання розвідувального дослідження літератури, рис. 6) об’єкт дослідження та встановлюємо предмет дослідження, який визначається об’єктом та цілями (завданням) дослідження. Як правило, це методології, методи, моделі і засоби створення і супроводу програмного забезпечення або відповідне уточнення (продукти, процеси, ресурси) згідно із наведеними рисунками 1-3 для прикладного домену.

Нарешті формулюємо назву дисертації. Здебільшого назва дисертації співпадає з формулою предмета дослідження.

Після виконання цих кроків можна починати магістерське дослідження згідно із структурою, яку наведено на рис. 4.

Застосування рекомендацій

Розглянемо такий випадок. Спеціалізація - інженерія програмного забез-

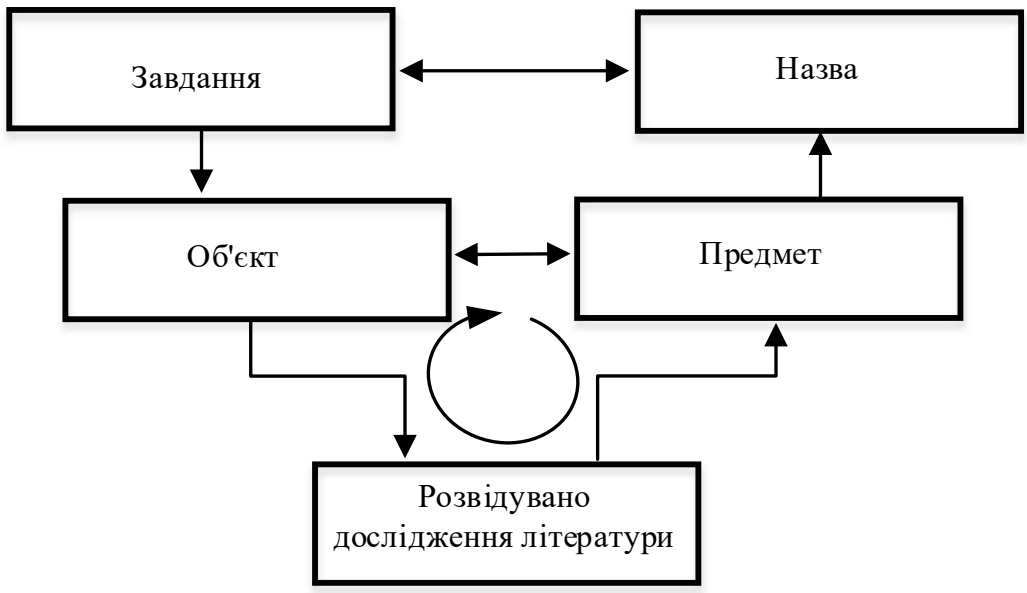


Рисунок 6. Уточнення об’єкта та встановлення предмета

печення інформаційних систем і технологій, домен - інженерія програмного забезпечення, прикладний домен – інформаційні системи і технології. Завдання – створити і дослідити Web - сервіс, що забезпечує прийняття рішень на основі неповної інформації.

Досвід показує, що зазвичай за об'єкт дослідження обирають програмне забезпечення Web – сервісу.

В цьому випадку предметом дослідження визначають метод і алгоритми прийняття рішень на основі неповної інформації (незалежно від того, існують вони чи ні), які необхідно встановити і реалізувати в рамках програмного забезпечення Web - сервісу.

Отож, якщо об'єкт дослідження визначено майже відповідно до домену, то предмет не має ніякого відношення, ні до інженерії програмного забезпечення, ні до перетину цього домену і прикладного домену. Метод і алгоритми прийняття рішень на основі неповної інформації як предмет дослідження цілком належить до домену інформаційних систем і технологій.

Таким чином, отримані результати (метод і алгоритми прийняття рішень на основі неповної інформації) будуть тільки частково належати до спеціалізації, тому експерти з цієї спеціалізації не зможуть їх визначити. Потрібні також експерти з іншого домену, а саме з домену інформаційних систем і технологій. Хоча дослідник повністю переконаний у правильності визначення об'єктом програмне забезпечення, предмет дослідження може бути з ним і не пов'язаний.

Скористаємося вище згаданими кроками (рис.5, 6). На виконання кроку 1 об'єктом дослідження для наведеного випадку (беручи до уваги, що Web – сервіс реалізується шляхом створення відповідного програмного забезпечення, а інших уточнень у завданні немає) може залишатися програмне забезпечення Web - сервісу, що уможливлює прийняття рішень на основі неповної інформації, або відповідне уточнення згідно з наведеними рисунками 1-3.

На другому кроці (рис. 5) цілями (завданням) дослідження визначається предмет дослідження. Як правило, це методології, методи, моделі й засоби створення й супроводу програмного забезпечення, або відповідне уточнення (якщо є в завданні). А саме продукти, процеси, ресурси відповідно до наведених рисунків 1-3. Наприклад, якщо програмне забезпечення розглядається в контексті прийняття рішень на основі неповної інформації, предметом може бути наступне: компоненти неодноразового використання (необов'язково програмні) відповідної форми (робочий продукт фази доменного аналізу) для створення програмного забезпечення Web - сервісу, або процес(и) (фаза(и) життєвого циклу) створення і супроводження програмного забезпечення. Або ж представлення і перетворення знань для створення і супроводження програмного забезпечення (ресурс I – артефакт) або підхід (метод) для створення і супроводження програмного забезпечення (ресурс II) .

В такому разі перевизначимо предмет дослідження наведеного раніше прикладу наступним чином, - методи (і, можливо, засіб), моделі, інструменти створення (супроводження) програмного забезпечення Web - сервісу, що забезпечує реалізацію методу прийняття рішень на основі неповної інформації.

Можливі й інші визначення предмету дослідження залежно від уточнень, які можуть бути в завданні:

- методи, моделі, інструменти створення і супроводження компонентів неодноразового використання для Web - сервісу, що забезпечує прийняття рішень на основі неповної інформації;
- методи, моделі, інструменти для реалізації процесу(ів) створення і супроводження програмного забезпечення Web - сервісу, що уможливлює прийняття рішень на основі неповної інформації;
- методи, моделі, інструменти для представлення і перетворення знань щодо програмного забезпечення Web - сервісу, яке гарантує прийняття рішень на основі неповної інформації.

Відповідно, предметом дослідження буде не метод і алгоритми прийняття рішень на основі неповної інформації, як

це визначено раніше, а метод створення (далі за об'єктами наведених прикладів) програмного забезпечення Web - сервісу, в якому застосовуються метод і алгоритми прийняття рішень на основі неповної інформації.

Як бачимо, об'єкт дослідження - це програмне забезпечення, а, визначаючи предмет, уточнюємо - робочий продукт, процес, або артефакт для прикладного домену - інформаційні системи і технології, де передбачається використання певних методів і алгоритмів прийняття рішень. Водночас використання методу і алгоритмів потребує наступного: розробки нового, або суттєвого удосконалення існуючого підходу до створення або супроводження програмного забезпечення; створення і супроводження компонентів неодноразового використання; реалізації процесу(ів) створення і супроводження програмного забезпечення; створення інструментів для представлення і перетворення знань щодо програмного забезпечення.

Звісно, використання методу і алгоритмів прийняття рішень у рамках програмного забезпечення Web - сервісу має вимагати розв'язання певного науково-практичного завдання, що обов'язково належить до інтересів інженерії програмного забезпечення. Для розв'язання цього завдання в процесі дослідження і буде створюватися метод та, ймовірно, засіб. Саме ці метод і засіб мають бути результатами дисертаційного дослідження, а не програмне забезпечення Web - сервісу, що забезпечує прийняття рішень на основі неповної інформації.

Виконуючи крок 3, беремо до уваги, що визначення предмету дослідження практично завжди буде назвою дисертації.

Так, зокрема, маємо наступні назви:

Метод і засіб створення і супроводження компонентів неодноразового використання щодо розробки і супроводження програмного забезпечення Web - сервісу, який забезпечує прийняття рішень на основі неповної інформації.

Метод і моделі щодо реалізації процесу специфікування вимог програмного забезпечення Web - сервісу, який забезпечує прийняття рішень на основі неповної інформації.

Моделі і інструменти для представлення і перетворення знань щодо створення і супроводження програмного забезпечення Web - сервісу, яке уможливлює прийняття рішень на основі неповної інформації.

Магістерська дисертація. Зміст

Після визначення об'єкта і предмета дослідження можна виконувати дослідження. Магістерська дисертація як основне дослідження (звичайно, це action research [15]) має наступні складові:

- розвідувальне дослідження для з'ясування об'єкта та предмета основного дослідження. Як правило, його не буває, якщо досвіду наукового керівника і знань студента достатньо. Але через умови, про які йшла мова раніше, це дослідження необхідно проводити;
- вивчення стану об'єкта (предмета), шляхом дослідження літератури. Міститься в першому розділі дисертації;
- основне (первинне) дослідження (action research) – дослідження предмета і створення теорії, підходу, методу, засобу, моделі або іншого для вирішення практичної задачі (відповідно назви дисертації). Міститься в другому та третьому розділах дисертації;
- дослідження теорії, підходу, методу, засобу, моделі або іншого, які створено в основному дослідженні (вторинне дослідження). В деяких випадках може сполучатися з контрольованим експериментом. Міститься в четвертому розділі дисертації.

Отож, зазвичай, текст магістерської дисертації описує результати трьох досліджень і містить чотири розділи.

З'ясування та вивчення стану об'єкта і предмета здійснюються шляхом дослідження літератури. Для виконання цих досліджень в інженерії програмного забезпечення найчастіше застосовують метод Систематичного огляду літератури - Systematic mapping study [37]. Це метод вторинного дослідження,

яке використовує чітко визначену методологію для виявлення, аналізу та інтерпретації всіх наявних доказів, пов'язаних із конкретним, первинним дослідженням, тож, є неупередженим та (певною мірою) повторюваним, а тому доказовим.

Основними етапами процесу дослідження за методом Systematic mapping study є визначення дослідницьких питань, проведення пошуку відповідних джерел, відбір джерел, дослідження анотацій та безпосередньо текстів, котрі знайдено за ключовими словами, збір і представлення даних [37]. Кожен етап процесу має результат. Кінцевим є графічне представлення результатів дослідження та обговорення. Основною метою Systematic mapping study є огляд літератури щодо досліджуваної галузі, визначення кількості та типу досліджень, а також наявних у галузі результатів. Крім цього, існує ще дві цілі досліджень:

- перша - відображення частоти публікацій і їх хронології, щоб побачити тенденції досліджень;
- друга - визначення джерел, на які спиралися дослідження в цій галузі.

Для візуалізації результатів зазвичай застосовують діаграму бульбашок з MS Excel.

У частині розділу – обговорення стану досліджень предмету дисертації з повним та інтенсивним застосуванням посилань на знайдені джерела. Після чого формуються підстави для проведення первинного (основного) дослідження.

Первинне дослідження (action research) – дослідження предмета і створення теорії, підходу, методу, засобу, моделі тощо для вирішення науково-практичної задачі, яку поставлено в дисертації.

Активне (дієве) дослідження (Action research) – це дослідження, з метою розв'язання науково-практичної задачі науковими методами, шляхом впливу на явище з будь-якого аспекту або зміною явища, що є фокусом дослідження [38]. Рішення задачі здебільшого потребує вивчення відповідного явища в аспекті предмета дослідження. Тож, активне (дієве) дослідження поєднує наукові методи з практичними задачами, які потребують розв'язання. Водночас дослідник повинен втручатися (впливати, діяти) в явище, змінюючи і перетворюючи його або його контекст. Активне дослідження має глибокий гносеологічній характер, тому, на жаль відсутні методики його проведення. Тут важливу роль відіграє науковий керівник здобувача. Саме від нього залежить, чи буде досягнута мета, чи будуть відповідати результати активного дослідження темі дисертації. Відповідність буде перевірено шляхом виконання вторинного дослідження.

Вторинне дослідження, метод дослідження результату (action research - Case Study). Case study - це метод виконання наукових досліджень, зокрема, в інженерії програмного забезпечення, невеликих явищ. Здебільшого застосовується для обґрунтування результатів первинного дослідження наукових статей, бакалаврських робіт, магістерських дисертацій. Тому засвоєння цього методу і застосування в магістерської дисертації додає дисертації наукового, ґрунтовного і доказового характеру. Case Study - це метод досліджень результату Action research, для розуміння та пояснення явища чи перевірки теорії, методу або засобу, отриманих у результаті виконання основного дослідження, використовуючи переважно (але необов'язково) якісний аналіз [39]. Case Study - це емпіричний метод дослідження. Це не підмножина, або варіант інших методів, таких як, контрольований експеримент, огляд чи історичне дослідження [40]. Case Study спрямоване щодо об'єкта та предмета основного дослідження на наступне:

1. Детально відповідає на питання «як і навіщо»;
2. Забезпечує глибоке розуміння причин і наслідків основного дослідження;
3. Забезпечує тестування теорій і методів, засобів, запропонованих у первинному дослідженні, коли змінні недостатньо керовані.

Існують наступні типи Case Study [39]:

- пояснювальний – досліджує і пояснює «як і чому так?». Наприклад, показує різницю між теоріями, підходами, методами або сутність теорії, підходу, ме-

тоду, або переваги теорії, підходу, методу, які створено;

- описовий – досліджує і показує «як це діє?» - можна віднести до засобів, які створено;

- причинний – досліджує і показує «яким чином?». Наприклад, як створений стиль кодування впливає на розуміння вихідного коду. Цей тип можна визначати шляхом оцінювання ефективності отриманих результатів;

- розвідувальний (попередній) – досліджує і показує «який стан?». Наприклад, дослідження літератури для з'ясування стану дослідження об'єкта або предмета основного дослідження, зокрема, за методом Systematic mapping study.

Може бути Case study змішаного типу. Case study є добре регламентованим методом. Існують докладні методики щодо виконання того чи іншого типу Case study [39].

Case study також має об'єкт та предмет, які формулюють у контексті результатів основного дослідження. Наприклад, основне дослідження – Метод і засоби редокументування успадкованого програмного забезпечення; Основне дослідження – об'єкт - успадковане програмне забезпечення; предмет – процеси, моделі, методи, засоби редокументування успадкованого програмного забезпечення. Тоді Case study - об'єкт - процеси, моделі, методи, засоби редокументування успадкованого програмного забезпечення; предмет – метод і засіб редокументування успадкованого програмного забезпечення, розроблені в основному дослідженні. Питання Case study – Чи працездатні та ефективні метод і засіб редокументування успадкованого програмного забезпечення, створені в основному дослідженні?

У виконанні досліджень щодо інженерії програмного забезпечення дуже важливо застосування вимірювань [40]. Для цього існують метрики і відповідні вимірювачі. Але трапляються випадки, найчастіше саме у проведенні основних досліджень і Case study, коли потрібних метрик або не існує або не зрозуміло які метрики слід застосовувати в конкретному випадку. Тоді використовують широко відомий в інженерії програмного забезпечення метод ціль-питання-метрика - GQM [41].

Метод GQM базується на припущенні, що для цілеспрямованого вимірювання, спочатку необхідно вказати ціль, а далі простежити цю ціль до метрик, які призначені для кількісного визначення цілі і, зрештою, забезпечити основу для інтерпретації метрик щодо заявленої цілі. Для полегшення простеження цілі до метрик застосовують питання, які пояснюють ціль. Таким чином модель GQM має три рівні: концептуальний (ціль); операційний (питання); кількісний (метрики).

Концептуальний рівень (Goal) визначає мету для об'єкта вимірювань з різних причин і точок зору, щодо конкретного середовища. Об'єкти вимірювання такі: продукти, процеси, ресурси.

Операційний рівень (Question) визначає набір питань, які використовуються для характеристики способу оцінки/досягнення конкретної мети. Формулювання питань має здійснюватися на основі характеристик об'єкта. Питаннями намагаються охарактеризувати об'єкт вимірювання (продукт, процес, ресурс) і визначити його з обраної точки зору.

Кількісний рівень (Metric) - це набір даних, метрика. Пов'язаний з кожним запитанням для відповіді на них кількісно. Дані можуть бути наступними:

- об'єктивні - якщо вони залежать лише від об'єкта, властивість якого вимірюється, а не від точки зору. Наприклад, кількість версій документа, кількість годин, витрачених на виконання завдання, розмір програми.

- суб'єктивні - якщо вони залежать і від об'єкта, властивість якого вимірюється, і від точки зору. Скажімо, читабельність тексту, рівень задоволеності користувачів.

Наприклад, об'єкт – стандарт кодування (ресурс), призначення – підвищення ефективності кодування, фокус – програмування, точка зору – менеджера, середовище – мова програмування.

Оформлення результатів (Good papers). Результати досліджень пред-

ставляються відповідними дослідницькими текстами. Найпоширенішими формами є стаття та дисертація. В інженерії програмного забезпечення дослідницькі тексти є звичними засобами для донесення науковій спільноті результатів дослідження [42]. У дослідницькому тексті автор пояснює зацікавленим читачам, як і яких результатів він досягнув, чому читачу слід читати цей текст. Якісний дослідницький документ має відповісти на ряд питань [42]:

- Яким саме є ваш результат?
- На яке запитання ви відповіли (предмет дослідження) ?
- Чому для читача важливий ваш результат?
- У контексті якого ще більшого питання поставлено питання, на яке ви відповіли (об'єкт дослідження)?

Щодо отриманого результату слід дати відповіді на наступні запитання:

- Які нові знання вашого результату читач може використати?
- Яку попередню роботу (вашу чи чужу) ви намагаєтесь вдосконалити?
- Чим ваш результат відрізняється і є кращим за цю попередню роботу?
- Яким конкретно є ваш новий результат?
- Чому читач має вірити вашому результату?
- Який стандарт (методику) слід використовувати для оцінки вашого результату?
-Які конкретні докази свідчать, що ваш результат задовольняє вашу вимогу?

Якщо автор чітко відповідає на ці запитання, він, імовірно, добре усвідомлює і може донести до читача свій результат. Якщо до того ж результат цікавий, зрілий та вагомий для прикладного домена в контексті інженерії програмного забезпечення, то це хороший шанс для позитивної ухвали на захисті дисертації.

Висновки

Метою цієї статті було узагальнення базових положень щодо формулювання об'єкта і предмета магістерської дисертації зі спеціалізації інженерії програмного забезпечення в умовах які склалися в Україні. По-друге, надання рекомендацій щодо застосування в магістерській дисертації наукових методів досліджень і оформлення результатів.

На відміну від закордонних університетів, в Україні протягом багатьох років склалася практика формулювання цілей дослідження, застосовуючи поняття об'єкта і предмета дослідження. І це, як показав час, важливий етап, від виконання якого залежить результативність дослідження. Загальне й тотальне проникнення програмного забезпечення в життя, з одного боку, і поява спеціалізацій інженерії програмного забезпечення - з іншого, значно ускладнюють виконання цього етапу дисертаційного дослідження. Тому актуальними є питання щодо формулювання об'єкта і предмета досліджень та застосування доказових методів їх виконання.

Отож, розглянуті умови, в яких за яких в Украйні здійснюється навчання магістрів, наведено рекомендації щодо формулювання об'єкта та предмета дисертаційного дослідження на основі фундаментальних положень інженерії програмного забезпечення, розглянуто методи доказових досліджень.

Література

1. Report on a conference sponsored by the NATO science committee, Garmisch, Germany, 7th to 11th October 1968, Editors: Peter Naur and Brian Randell.
2. Boehm B., 2006, A View of 20th and 21st Century Software engineering [Text]. ICSE'06. May 20–28. China. 2006. P. 12–29.
3. Вельбицкий И.В. Технология программирования. [Текст]. Техника. Киев. Украина. 1984. 279 с.
4. Глушков В.М., Вельбицкий И.В. Технология программирования и проблемы ее автоматизации. [Текст]. УСИМ. Киев. № 6. 1976. С. 75–93.
5. Технология программирования: Тез. докл. 1-й Всесоюз. конф., Киев, октябрь 1978 г. Пленарные докл. И общ. материалы. – Киев: Ин-т кибернетики АН УССР, 1979.
6. Сидоров М.О. Інженерія утилізації програмного забезпечення систем управлін-

ня та ЕОМ.- Автор. Дис.на здобуття наук. ступ. докт. техн. наук..- Інститут кібернетики імені В.М.Глушкова НАН України.- 1995 р. 25 с.
7. Сидоров М.О. Кафедра інженерії програмного забезпечення. Компьютер-Клас! 2001. № 8. С. 17–19.
8. Бондаренко М., Сидоров М., Морозова Т., Мендзебровський І. Модель випускника бакалаврату «Програмна інженерія», Вища школа. 2009. № 4. С. 50–61.
9. https://nau.edu.ua/ua/menu/science/faxovi-vidannya/
10. Сидоров Н.А. Инженерия программного обеспечения – дисциплина или бакалаврат? Матеріали міжнародної науково-практичної конференції «Розробка систем програмного забезпечення: виклики часу та роль інформаційному суспільстві». Київ, 27-28 січня 2005 р.
11. Сидоров Н.А. Инженерия ПО - дисциплина или бакалаврат? Корпоративные системы. № 2. 2005. С. 22–27.
12. Сидоров Н.А. Инженерия программного обеспечения – учебная дисциплина или подготовка бакалавра? Управляющие системы и машины. 2006. № 2. С. 25–34.
13. Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering A Volume of the Computing Curricula Series, August 23, 2004
14. Накази Міністерства освіти і науки України від 29.10.2018 № 1166, 17.11.2020 р. № 1424
15. Parnas D. Software Engineering Programmes are not Computer Science Programmes// Annals of Software Engineering.- April 9, 1998, P1-16.
16. Boehm B. k "The IEEE-ACM Initiative on Software Engineering as a Profession", IEEE Computer Society Software Engineering Technical Council Newsletter, Vol. 13, No. 1, Sept. 1994, page 1.
17. Cheston, G.A. and Tremblay, J., Integrating software engineering in introductory computing courses. IEEE Softw.,2002, 19(5) 64–71.
18. Michael Davis Defining "Engineer" — How to Do It, and Why It Matters/Journal of Engineering Education, Vol. 85, No. 2. 1996
19. Gary Ford, Norman Gibbs, A Mature Profession of Software Engineering/Technical Report CMU/SEI-96-TR-004 ESC-TR-96-004 September 1996
20. George Hurlburt and Jeffrey Voas, Software Is Driving Software Engineering?/ IEEE Software, January/February 2016, 101-104p
21. P. Bourque and R.E. Fairley, eds., Guide to the Software Engineering Body of Knowledge, ver. 3.0, IEEE CS, 2014; www.swebok.org.
22. https://www.acm.org/binaries/content/assets/education/se2014.pdf
23. A. Mishra et all, Software engineering education: Some important dimensions/European Journal of Engineering Education · June 2007
24. Dart, P., Johnston, L., Schmidt, C. and Sonenberg, L., Developing an accredited software engineering program. IEEE Softw., 1997, 14(6), 66–70.
25. Ford, G.A., Progress Report on Undergraduate Software Engineering Education. CMU/SEI-94-TR-11. Software Engineering Institute, Carnegie Mellon University, 1994.
26. Moore, M.M., Software engineering education. IEEE Softw., 2002, 19(5), 103.
27. Saiedian, H., Software engineering education and training for the next millennium. J. Syst. Softw., 1999, 49, 113–115.
28. Current State of Software Engineering Master's Degree Programs In the United States Donald J. Bagert and Xiaoyan Mu in Proceedings - Frontiers in Education Conference · November 2005
29. V.R. Basilli, "Software Engineering: State of the Art and Practice," Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research, R.W. Selby, ed., John Wiley & Sons, 2007, ch. 8.
30. Capers Jones, Variations in Software Development Practices, November/December 2003 IEEE SOFTWARE
31. C. Landwehr et al. Software Systems Engineering programmes a capability approach/ The Journal of Systems and Software 125 (2017) 354–364
32. A. Mishra et all, Industry Oriented Advanced Software Engineering Education Curriculum, Croatian Journal of Education Vol: 14 (3/2012), pages: 595-624
33. C. Wohlin *, B. Regnell, Strategies for industrial relevance in software engineering

education, The Journal of Systems and Software 49 (1999) 125±134
34. Graduate Software Engineering 2009(GSwE2009) Curriculum Guidelines for Graduate Degree Programs in Software Engineering, 2009, Stevens Institute of Technology
35. Н. А Сидоров, Экология программного обеспечения, Інженерія програмного забезпечення, Вип.1, 2010, С53-61
36. The ACS Core Body of Knowledge for ICT Professionals (CBOK), Australian Computer Society Inc, Sydney NSW 2000
37. K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, Proceedings of the .-12th International Conference on Evaluation and Assessment in Software Engineering, 2008, pp. 1-10.
38. Paulo Sérgio Medeiros Dos Santos, Guilherme Horta Travassos Action Research Can Swing the Balance in Experimental Software Engineering.- Advances in Computers · December 2011, P78.
39. P. Runeson, M. Case study research in software engineering, Guidelines and Examples.- 2012 by John Wiley & Sons, Inc, P241.
40. Forrest Shull • Janice Singer • Dag I.K. Sjøberg Guide to Advanced Empirical Software Engineering.- Springer-Verlag London Limited 2008, P394.
41. Basili,V.R, et al, Goal Question Metric Approach, John Wiley&Sons,1994.
42. Mary Shaw Writing Good Software Engineering Research Papers.- Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, 2003, pp. 726-736.

Отримано: 25.05.2022

Про автора:

Сидоров Микола Олександрович,
доктор технічних наук,
професор.
Кількість наукових публікацій у українських виданнях – 140.
Кількість наукових публікацій у зарубіжних виданнях – 22.
http://orcid.org/0000-0002-3794-780X

Місце роботи автора:

Національний Технічний Університет України «Київський політехнічний інститут імені Ігоря Сікорського», факультет інформатики та обчислювальної техніки, кафедра інформатики та програмної інженерії, ІПІ, професор.
02000, Київ,
вул. Політехнічна, 41.
Моб. тел.: 067 7980361.
E-mail: nyksydorov@gmail/com

УДК 004.42 http://doi.org/10.15407/pp2022.02.037

Ю. Дивак, Т. Мамедов

ІНТЕГРАЦІЯ І КОМПОЗИЦІЯ ВЕБ-СЕРВІСІВ НА ОСНОВІ МОДЕЛІ

Анотація. В цій роботі я хотів би представити новий підхід для розробки застосунку і архітектури для роботи із семантичними веб-сервісами який інтегрує декілька корпоративних застосунків та поєднує результати з різних сервісів через застосування технік, методологій та анотацій, забезпечених технологіями розробки програмних систем і моделюванням бізнес-процесів. Я пропоную застосовувати існуючі техніки моделювання бізнес-процесів (BPMN, BPLWS, OWL-S, WSDL, WebML) для моделювання процесів, які проходять через декілька застосунків, для інтеграції програмних систем та композиції веб-сервісів. Головна мета - знайти методологію для розробки семантично багатих веб-застосунків з напівавтоматичним визначенням семантичного опису сервісу для моделі бізнес процесів. Це підвищить якість проєктування та зменшить обсяг додаткової роботи при розробці застосунків з обробкою семантично анотованої інформації, що проходить через корпоративні застосунки.

Ключові слова: Інтеграція програмних систем, Композиція веб-сервісів, Моделювання бізнес-процесів, Семантичні веб-сервіси, Проєктування, засноване на моделюванні, Методології

Вступ

Працюючи інтеграційним інженером і інтегруючи застосунки з усього світу, я виділив декілька проблем, з якими мав справу щодня. Серед них інтеграція двох незалежних застосунків між собою, інтеграція веб-сервісів із застосунками, і композиція веб-сервісів для отримання більш відповідного результату за менший проміжок часу. Звісно, ці проблеми можна розглядати і вирішувати окремо одна від одної, але, якщо розглядати їх разом, можна знайти нові шляхи вирішення. Дана стаття – це спроба знайти новий підхід для вирішення поставлених задач.

Усі аспекти корпоративного світу розроблені на основі бізнес-процесів, і часто ці бізнес-процеси проходять через декілька корпоративних застосунків, а іноді ці застосунки належать різним вендорам. Робота через декілька застосунків потребує асинхронного зв′язку між ними і здійснюється за часто змінюваними сценаріями. Такі інтеграційні рішення, як описані в [1] : **Файл трансфер**, **Спільна база данних**, **Віддалений виклик процедур** та **Надсилання повідомлень** – це опис підходів у загальному плані, конкретні реалізації інтеграцій. Такі, зокрема, як **мости**, **інтеграція через дані**, **надсилання повідомлень**, **інтеграція на рівні компонентів**, **інтеграція через інтерфейси**, **брокери**, **(ESB) enterprise service buses**, **інтеграційні сервери** та **інтеграційні архітектури**. Всі вони мають обмеження та застосовуються в конкретних зважених випадках. Якщо ми хочему обійти деякі обмеження ми можемо застосувати **семантичні веб сервіси, саме семантика дозволить обійти деякі обмеження**. Семантичні веб сервіси - це парадигма програмування яка базується на анотації процесів та само-описуючої реалізації, яка може бути застосована для побудови крос-корпоративних за стосунків, що вирізняються **гнучкістю**, **автоматичним виявленням ресурсів**, та **динамічним розвитком застосунку**. Одна з найважливіших задач це адаптувати семантичні технології та семантичні анотації до вже розроблених коммпонентів та застосунків. Напівавтоматичний підхід може бути застосований для виділення семантичного опису веб-сервісів.

Запропонований підхід базується на принципах розробки за стосунків, які працюють на основі моделей. Це означає, що спочатку ми маємо тільки модель, а згодом на основі цієї мо-

делі застосунок будує схему виконання процесів, в якій будує композицію веб-сервісів, шукаються описи веб-сервісів у спеціальних репозиторіях. Далі згідно зі схемою виконуються запити до веб-сервісів, збираються і компонуються прийнятні відповідно до критеріїв результати. Результат отримує користувач програмної системи. Інший підхід, побудований на основі роботи з моделлю процесів, описаний в [2] і [4], пропонується для додаткового вивчення.

Під час розробки нового підходу треба реалізувати декілька аспектів проєктування:

- Розробка продукту, який працює на вищому рівні абстракції глобальної хореографії веб-сервісів для взаємодії між застосунками і веб-сервісами.
- Робота згідно із запропонованою бізнес-моделлю. Модель не прив'язана до конкретних сервісів чи компонентів. Опис композиції веб-сервісів, забезпечений за допомогою анотацій, дає додатковий рівень абстракції моделі.
- Архітектура пропонує рішення для роботи декількох корпоративних за стосунків, працюючих одночасно.
- Архітектура пропонує рішення для роботи з декількома веб-сервісами чи акторами.

Приклади застосування

У цій статті ми спробуємо застосувати такий підхід до реального випадку, де ми маємо за стосунок, який працює із сервісами. Цей застосунок має комбінувати результати виконання декількох сервісів в одну задовільну відповідь кінцевому користувачеві згідно з наданої бізнес-моделі. В загальному плані композиція виглядає, як на рисунку 1.

Схема запитів на рисунку 2 показує взаємодію між сервісами та місця, де запит може бути перервано. Під час дослідження ми маємо брати до уваги, що наша система не повинна обмежуватися тільки двома взаємодіючими сервісами. Вона має працювати із сотнями і тисячами різних сервісів, налаштовувати взаємодію правильно та підбирати найбільш відповідні сервіси для композиції та об'єднувати результати їх виконання.

Ми можемо застосувати однаковий підхід для будь-якого випадку, де нам потрібна композиція декількох сервісів. Другий приклад, зображений на рисунку 3 та 4, описує композицію сервісу бронювання квитків на літак і бронювання номера в готелі.

Загальні принципи побудови системи

У загальному прикладі SOAP архітектури ми маємо декілька рівнів:

Рівень користувачів сервісів – в нашому конкретному випадку користувачі сервісів - це за стосунки, розроблені різними способами (B2B, Portals, .NET, JS, Java, Salesforce, Microsoft Dynamics CRM, etc.), які користуються сервісами через медіатор і отримують скомпонований результат виконання запиту з декількох сервісів. Це можуть бути як портали, готові рішення на основі CRM систем, або написані програми, які можуть бути взагалі без серверної частини і забезпечувати до-

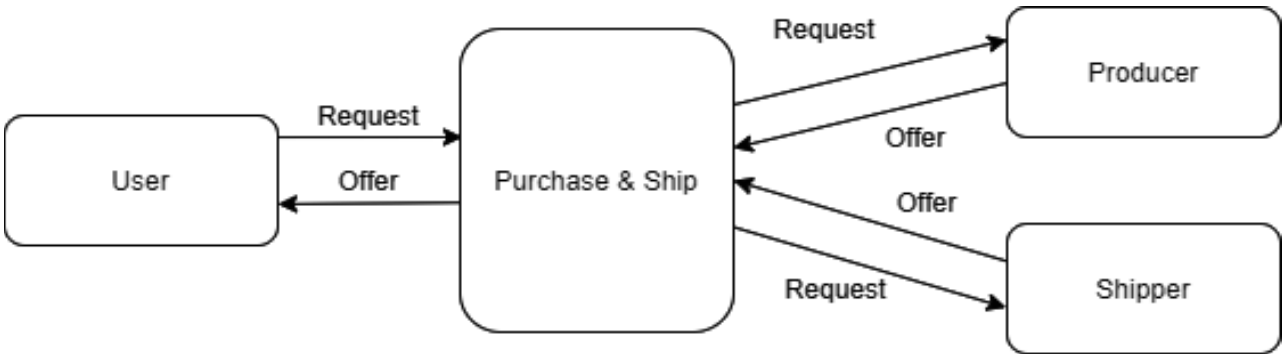


Рисунок 1. Схема композиції сервісів (приклад 1)

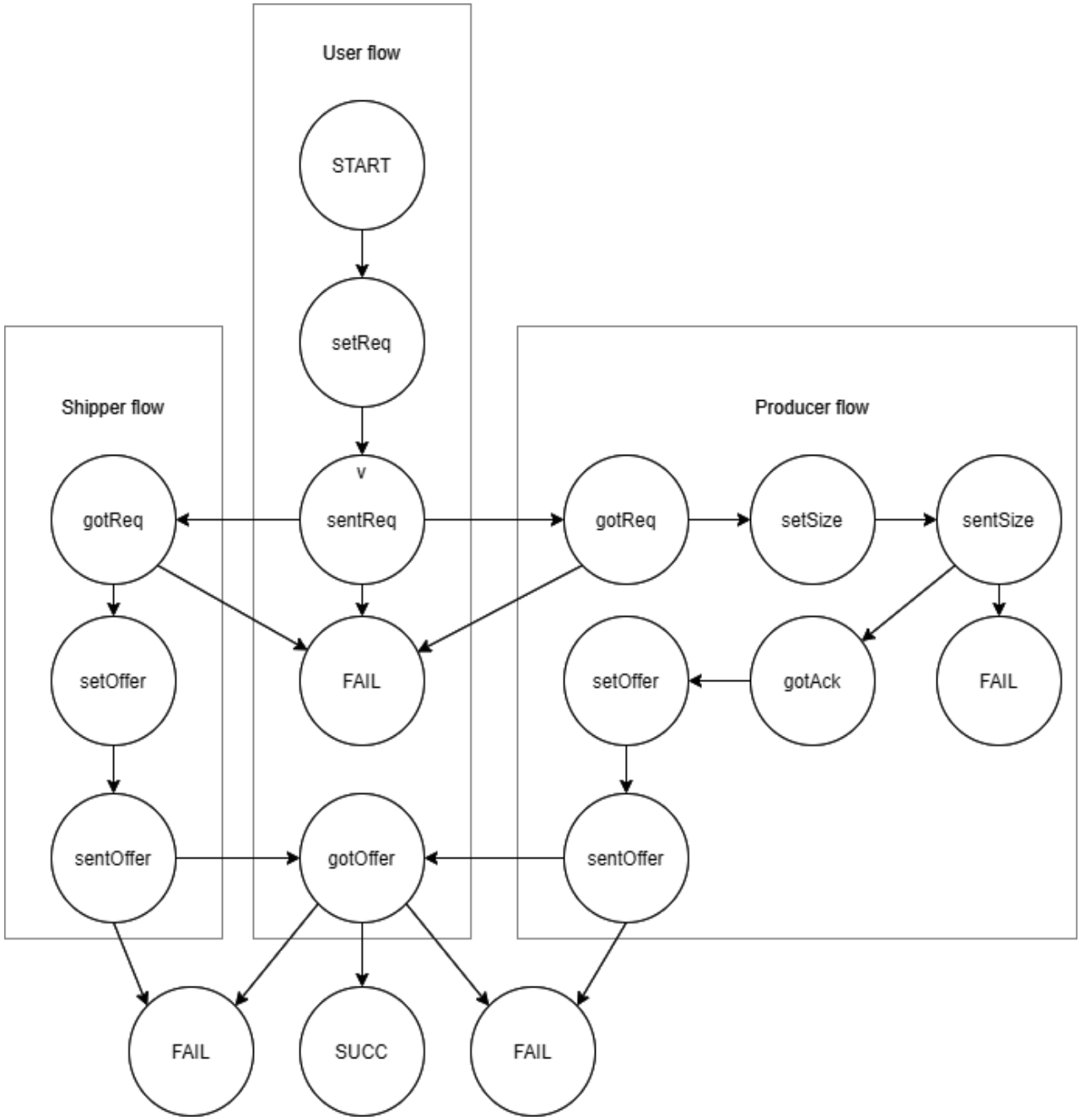


Рисунок 2. Схема виконання (приклад 1)

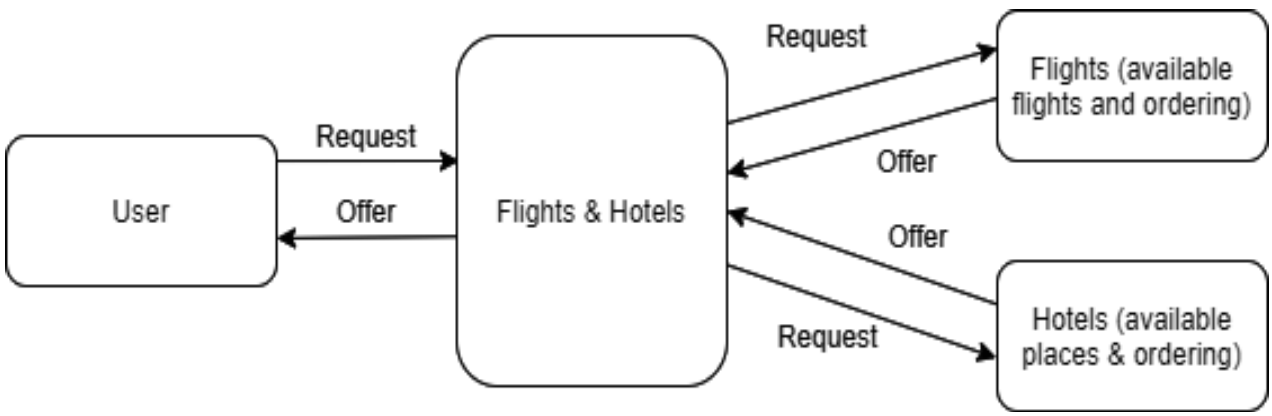


Рисунок 3. Схема композиції сервісів (приклад 2)

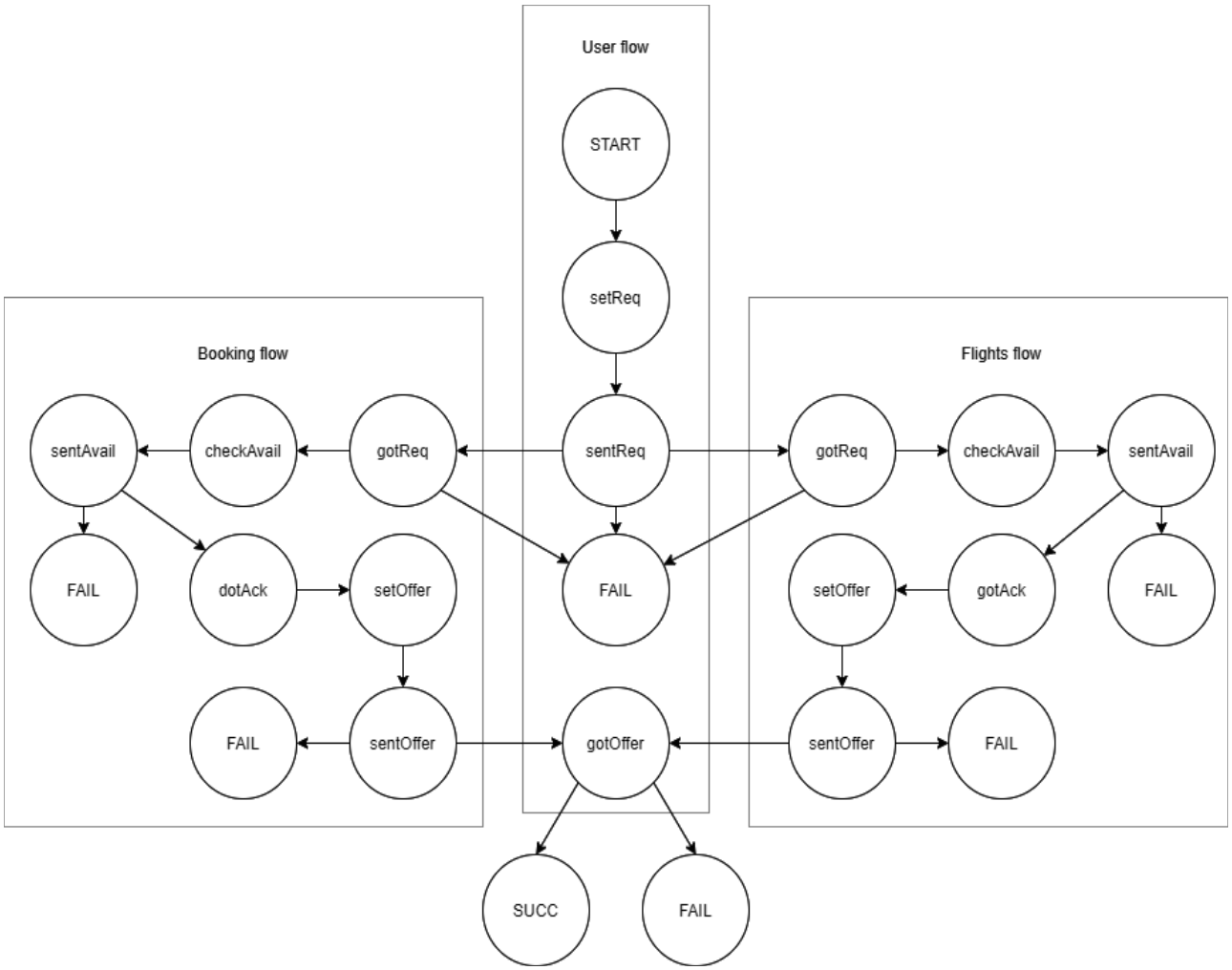


Рисунок 4. Схема виконання композиції (приклад 2)

ступ через медіатор до сервісів через розроблений UI, або системи, в яких тільки частина серверного застосунку замінена на роботу через медіатор із зовнішніми сервісами.

Рівень бізнес-процесів – В нашому конкретному випадку це документ (скрипт, модель бізнес-процесів, BPEL процеси), який описує бізнес-процеси в семантичній манері. Ми отримуємо цей документ від клієнта і згідно з ним забезпечуємо композицію сервісів і повертаємо задовільний результат. Клієнт може не турбуватися про пошук і композицію свервісів, про все піклується медіатор.

Рівень сервісів (композиція) – в нашому випадку ми маємо сховище, або декілька окремих сховищ, де зберігаються описи сервісів в форматах: WSDL, XML, SCHEMA, або WS-Policy. Сховища для описів сервісів можуть бути різні. Наприклад, файлове сховище, SQL, або NoSQL база даних, UDDI репозиторії, де зберігається необмежена або обмежена кількість описів для сервісів, які ми можемо використовувати. Медіатор підключений до сховищ і здійснює пошук у сховищах сервісів, які найбільш відповідають описові сервісу в документі [4].

Рівень компонентів та операційних систем – цей рівень виходить за межі дослідження в цій статті і реалізація покладається на розробників сервісів.

Агрегатори – це за стосунки, які компонують результати виконання запитів у декілька сервісів, зазвичай служать спільній меті - комбінувати відповіді з декількох сервісів та трансформувати в загальну відповідь, яка підходить для кінцевого користувача. Зокрема, це може бути пошук певного продукту та додаткової інформації про нього. Водночас ми повинні агрегувати результати з різних сервісів (онлайн-магазинів) та комбіну-

вати їх в якусь структуровану відповідь. Інший агрегатор може знаходити рейси літаків від різних компаній, номери різних готелів та сортувати за прийнятною ціною або за датами, чи пересадками та поєднувати рейси з доступними номерами в готелях. Це все збирається та компонується у відповідний формат, зрозумілий кінцевому користувачеві (кінцевим користувачем в нашому прикладі є так само застосунки).

Агрегатори в свою чергу не можуть забезпечити дослідження простору інтернету для виявлення нових сервісів та використання їх у системі. Для цього існують спеціальні кравлери, такі ж, що використовуються, наприклад Гуглом для індексації веб-сторінок. У нашій системі може використовуватися кровлер, що збирає описи сервісів, а також ми можемо використовувати сховище, налаштоване вручну.

У цій статті ми поговоремо про декілька компонентів і розробок щодо побудови посередника, який може використовуватися замість частини бекенду.

Медіатор під'єднаний до сервісів або апішок та отримує дані, що відповідають переданим параметрам. Однак тут маємо проблему: якщо один сервіс стає недоступним, ми не можемо отримати з нього відповідь. Тому медіатор може отримати дані з інших сервісів, система динамічна.

Витрати на розробку сервісів або інтеграцій з уже існуючими сервісами, можуть бути знижені завдяки такій системі, адже вона може взяти на себе частину роботи з динамічної композиції вебсервісів.

За основу для нової архітектури системи для інтеграції із сервісами і композиції веб-сервісів взято стандартну архітектуру SOAP за стосунку, яка зображена на рисунку 5, і перероблено згідно з нашими вимогами до системи.

Вже існує схоже рішення і розробка схожого за стосунку, який забезпечує композицію сервісів. Цей застосунок розроблявся 2013 року групою дослідників, і принципи його роботи описані в [5]. Архітектуру системи iServe зображено на рисунку 6.

Проблеми

Маємо зважати на проблеми, що виникають під час розробки подібних систем, які стосуються виконання, проєктування та розробки системи.

Під час виконання ми можемо стикнутися з такими проблемами як: **Доступність сервісу; надійність системи,**

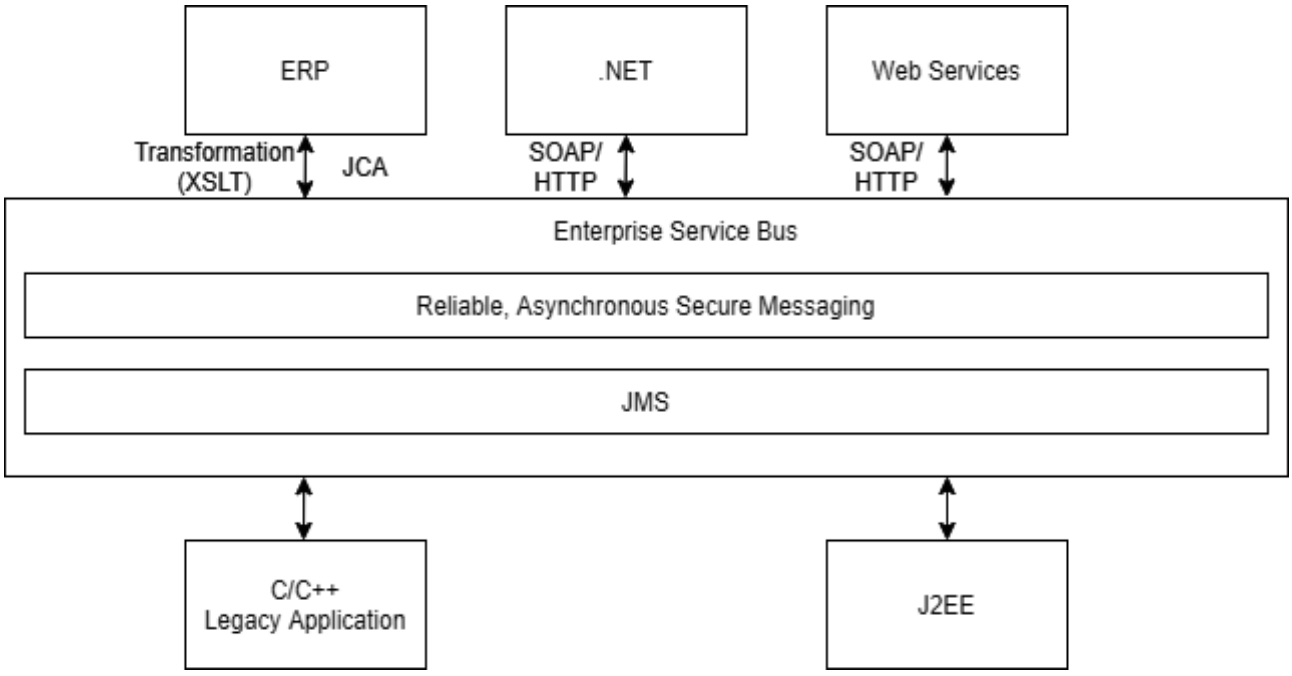


Рисунок 5. SOAP Архітектура

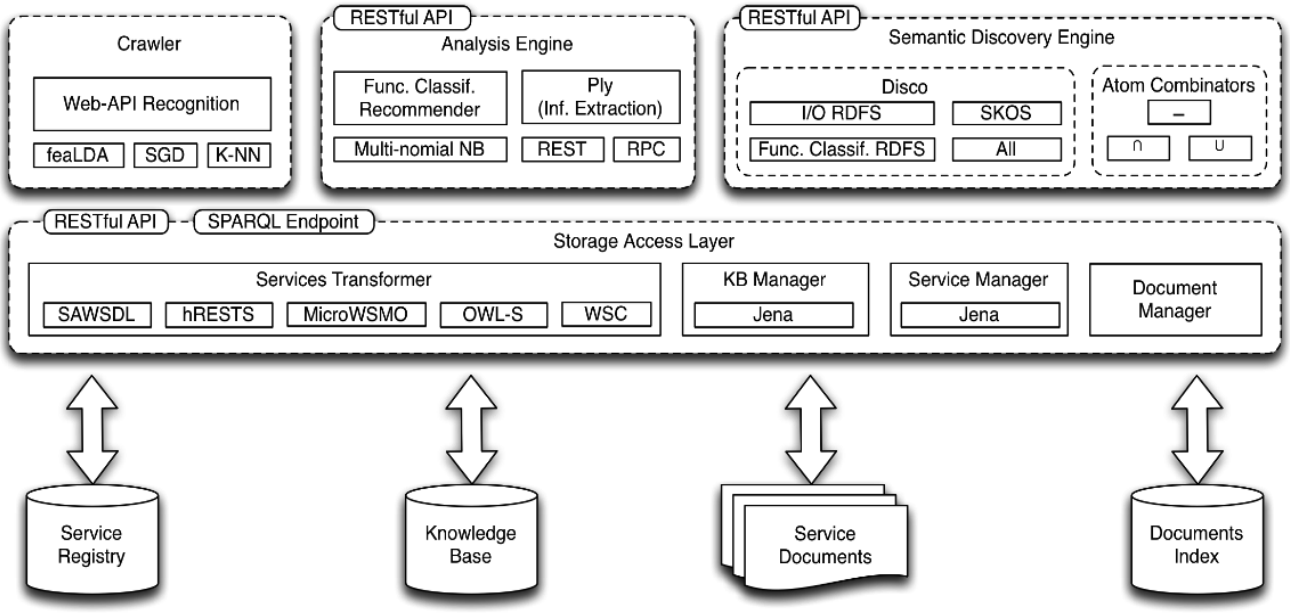


Рисунок 6. iServe архітектура

довговічність (як довго ми можемо її використовувати), масштабованість, зручність використання, безпека, можливість налаштування, продуктивність, обмеження.

Під час проєктування і розробки маємо інші проблеми: **можливість повторного використання, розширюваність, портативність, сумісність, модульність, тестованість, локалізація та інтернаціоналізація**.

Затримки

У подібних системах, де існує декілька компонентів, які взаємодіють через протокол HTTP, цей протокол є найслабшим місцем. Навантаження на систему може перевищити допустимі ліміти пропускання даних, що призведе до затримки відповіді на запити. Вирішенням такої проблеми може бути робота декількох таких систем одночасно на кількох серверах і балансування запитів між застосунками за допомогою наприклад NGINX або на рівні застосунків (балансування між застосунками відбувається на рівні компонентів). Можна також мати систему яка буде автоматично збільшувати кількість застосунків і серверів залежно від навантаження системи. Позаяк наша система складається з декількох компонентів, то їх можна запускати на окремих серверах, тим самим розподіляти навантаження між серверами.

Також можна розбивати нашу композицію на загальні частини і тоді результати виконання деяких частин, які використовуються найчастіше, зберігати в кеші застосунку для прискорення доступу до результатів виконання (Таким чином ми можемо отримати композицію композицій веб-сервісів).

Надійність

Інша проблема може виникнути, якщо один з сервісів, опис якого зберігається в сховищі, стає недоступним. Можливий випадок, коли ми безмежно довго чекаємо на відповідь сервісу. Але в нашому випадку медіатор автоматично перевіряє доступність сервісів для композиції. Якщо ж сервіс недоступний – шукає заміну з інших описів сервісів, які зберігаються в сховищі.

Відповідність результату

Проблема відповідності полягає в тому, чи відповідає результат виконання композиції тим параметрам і вимогам, які ставить користувач системи. Відповідність повністю лягає на бік медіатора.

Для забезпечення більшої відповідності результату виконання композиції вхідним параметрам, в нашій системі не потрібно кешувати окремі запити

до сервісів, бо нам потрібна тільки актуальна інформація, і ми легко можемо замінити один сервіс іншим у побудові маршруту, один результат виконання заміниться іншим.

Кешування може бути корисним, якщо ми хочемо прискорити виконання композиції. До прикладу, ми можемо зберігати на рівні кеша застосунку певну вже готову композицію і повертати результат виконання з кеша, водночас оминаючи повторну композмцію 2 - 3 - 4 сервісів (Композиція композицій).

Доступність

Інша проблема полягає в тому, що всі сервіси різні, і ми мусимо по-різному з ними взаємодіяти. Вирішенням може бути – взаємодія через єдиний інтерфейс, що забезпечить взаємозаміну сервісів іншими, зручність взаємодії із сервісами, у разі, якщо їх велика кількість і є швидкий перехід від одного сервісу до іншого.

В єдиному інтерфейсі також має бути передбачатися можливість перевірки сервісу на доступність і маркування його як недоступний або доступний, і періодичну перевірку доступності сервісів.

Компоненти

1. Користувачі сервісів

Найпростішим користувачем системи може бути звичайна HTML сторінка, що автоматично надсилає стандартний документ, який описує бізнес-процеси і композицію сервісів та написаний, наприклад, за допомогою BPLWS або OWL-S (модель бізнес-процесів) до нашої системи. Цей документ отримує відповідь і рендерить її на сторінці. Основна задача документу, який ми надсилаємо до медіатора - це опис станів, умов та параметрів для сервісів. За допомогою цього підходу ми зберігаємо час на розробку взаємодії із сервісами і доручаємо всю роботу медіатору. Той робить композицію в автоматичному режимі, повертаючи клієнту результат, який відповідає переданим параметрам. Обов'язковою є фронт-енд частина, коли клієнт може обійтися без серверної частини застосунку.

2. Модель бізнес-процесів

Документ (Execution script), який передається від клієнта до медіатора. Вочевидь цей документ - звичайний текстовий файл, складений за допомогою певних правил і принципів, в якому є параметри такі, як передумови та умови, перевірки, сутності та моделі даних.

У цьому документі ми можемо описати дані (Модель даних), очікуючи, що вони будуть повернуті з медіатора до клієнта, а також які між ними зв′язки.

3. Медіатор

Медіатор приймає запити від клієнтів, контролює вхідні параметри, підбирає потрібні сервіси за параметрами, складає маршрут виконання композиції та виконує запити, компонує результат і відповідає клієнту. Це центральний і найважливіший компонент нашої системи, який отримує багато запитів на композицію, працює зі сховищами описів сервісів, перевіряє сервіси на доступність та працює із сервісами, збираючи відповіді. Коли медіатор парсить документ розбираючи всі вхідні і визідні параметри сервісів, він підбирає сервіси з бази і знаходить найкоротші маршрути. Медіатор у постійному контакті із сервісами.

Ми можемо маніпулювати параметрами також на рівні побудови маршруту для отримання найкращого результату (преобробка), та обробляти дані, отримані із сервісів вже згодом (постобробка) таким чином досягяючи найкращого результату виконання композиції. Важливо чітко розуміти, який результат ми хочемо отримати, який результат композиції для досягнення більш точного результату. Це дозволить підібрати відповідні сервіси з більш надійними результатами, скласти більш вдалий маршрут виконання композиції сервісів, а на етапі пост обробки - узагальнити результати так, щоб задовольнити вимоги клієнтів застосунку.

4.Сховище описів сервісів

Опис веб сервісу може зберігатися по-різному. Наприклад, це може бути

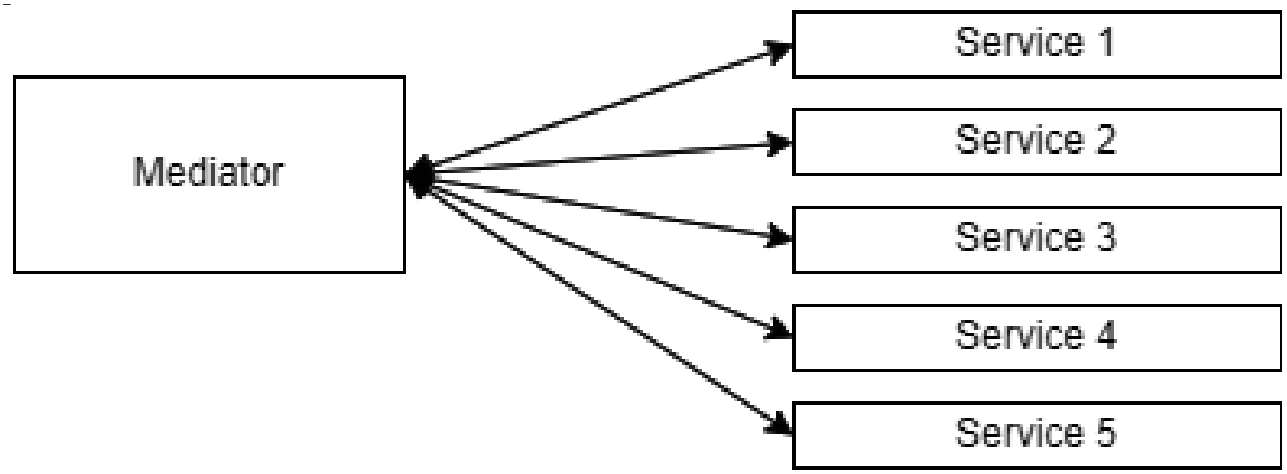


Рисунок 7. Взаємодія медіатора з сервісами

файлове сховище, де зберігаються текстові документи у форматі WSDL (як зроблено частково в застосунку iServe [5]) з описами сервісів. Це можуть бути описи, які зберігаються в реляційній базі даних, з якої ми отримуємо опис в зібраному вигляді з декількох таблиць. Це може бути також документоорієнтована база, яка використовує JSON як документ (NoSql). Описи зберігають інформацію про ендпоінти, URLs, вхідні параметри сервісу, вихідні параметри сервісу, кондішени та прекондішени, моделі тощо. Також ми можемо використовувати UDDI репозиторії як сховища для описів сервісів (як зроблено частково в застосунку iServe[5]), публічні або приватні репозиторії. В цьому випадку наш медіатор стає користувачем репозиторію [3].

5. Веб-кравлер

Що ж до веб-кравлера, ми маємо на увазі окремий за стосунок, який в автоматичному режими здійснює пошук мережею інтернет доступних веб-сервісів. Прикладом застосування такого типу систем є індексація гуглом сторінок в інтернеті. Таким чином наша база даних описів веб-сервісів для подальшої взаємодії може весь час доповнюватися, а це забезпечить кращий результат виконання композиції і взаємодії із сервісами.

Архітектура

На рисунку нижче ми можемо бачити, як виглядає вся система в загальному плані. Всі компоненти системи працюють разом у тісній взаємодії для досягнення єдиного результату. А саме:

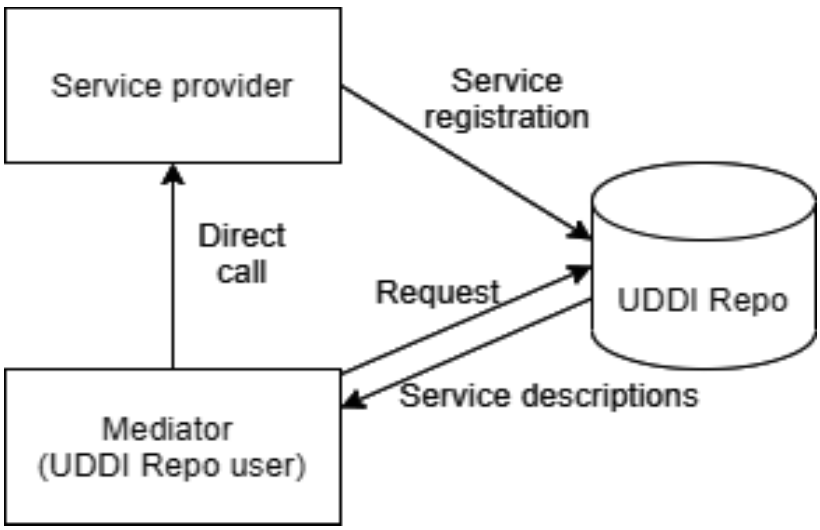


Рисунок 8. Взаємодія медіатора з репозиторіями і конкретним сервісом

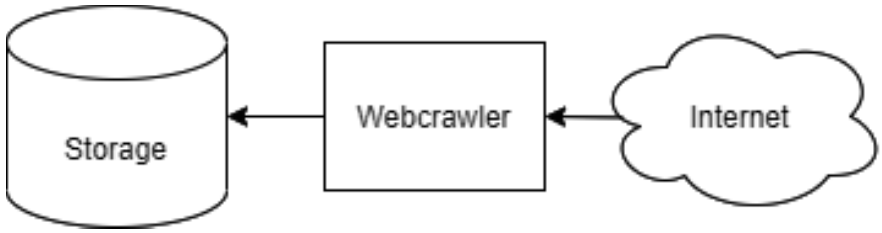


Рисунок 9. Додатковий веб-кровлер для автоматичного пошуку описів сервісів та додавання в базу

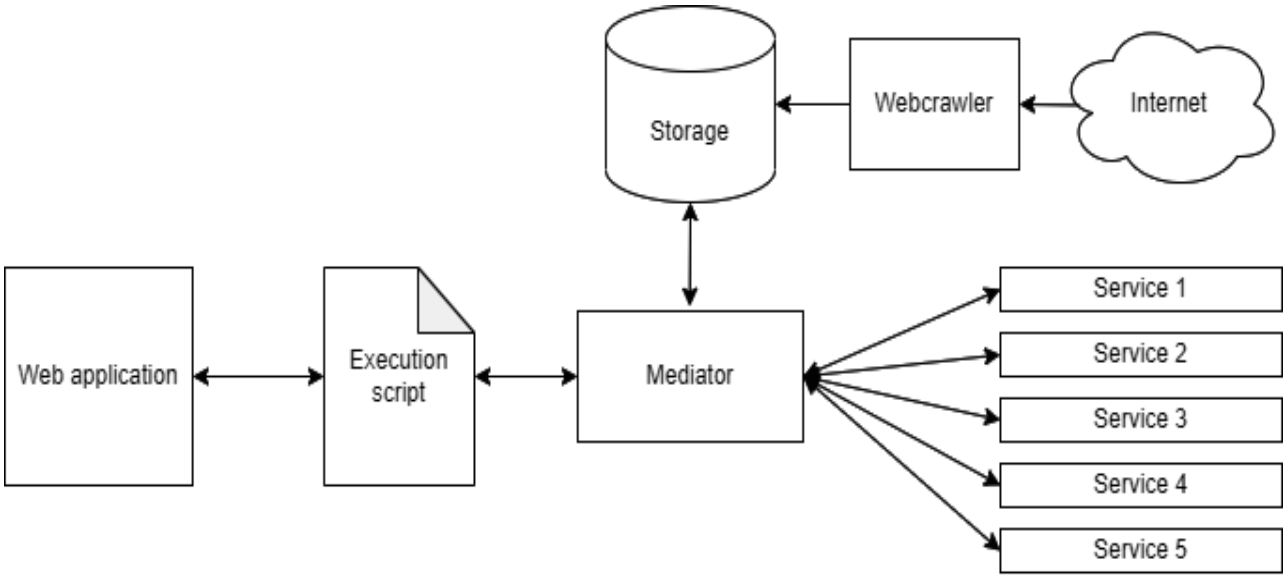


Рисунок 10. Архітектура для композиції сервісів

збір даних із зовнішніх джерел, композиція результатів виконання сервісів та конструювання загального результату виконання композиції, який відповідає вхідним параметрам і вимогам для результату. Одна з основних переваг такої системи - це (як ми бачимо на рисунку 10) гнучкість. Усі компоненти працюють незалежно і можуть бути покращені откремо один від одного без втручання у роботу інших компонентів. Клієнти застосунку можуть бути розроблені за допомогою будь-якої з відомих технологій розробки і проєктування програмних систем.

Висновок і майбутня праця

Нова архітектура може допомогти кінцевим користувачам отримати кращий результат за коротший проміжок часу і водночас значно скоротити витрати на розробку окремих компонентів, інтеграції та композиції сервісів. Цей підхід покращує вже існуючі підходи до композиції веб-сервісів й інтеграції із зовнішніми сервісами, а також дає можливість застосувати до цієї архітектури принципи антології (тому що інтеграція і композиція заснована на моделі). Наступні роботи будуть пов′язані із конкретною реалізацію медіатора і застосування алгоритмів для отримання кращого результату композиції та інтеграції, що забезпечить підвищення продуктивності та покращить інші характеристики системи. Ми спробуємо виміряти продуктивність системи із застосуванням різних підходів до композиції і оберемо найкращий варіант реалізації медіатора.

Алгоритми, які можуть бути застосовані при композиції веб-сервісів в рамках цієї архітектури.

- Євристичні алгоритми пошуку,
- A* Алгоритм,
- Мурашиний алгоритм,
- Генетичний алгоритм,

• Алгоритми динамічного програмування,

• Метод гілок і меж.

Залежно від вимог ми можемо обрати відповідніший алгоритм.

Література

1. Dyvak Y. A. (2021) Analytical review of existing approaches to integration of program systems.
2 Brambilla M., Celino I., Ceri S., Cerizza D., Della Valle E., Michele Facca F., (2006) A Software Engineering Approach to Design and Development of Semantic Web Service Applications.
3. Andon P. I. (2014) The Problems of programming in the semantic web environment. UkrPROG`2014
4. Arpinar B., Aleman-Meza B., Zhang R., Maduko A., (2012) Ontology-driven Web services composition platform.
5. C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert (2014) iServe: a Linked Services Publishing Platform

Отримано: 04.07.2022

Про авторів

Дивак Юрій, аспірант
Мамедов Турал, аспірант
ORCID: 0000-0001-6016-999X

Місце роботи:

Інститут програмних систем
НАН України, 03187, м. Київ-187,
проспект Академіка Глушкова, 40.
yurii.dyvak@gmail.com

УДК 004.031'43 https://doi.org/10.15407/pp2022.02.047

А.Д. Тарасенко, А.Ю. Дорошенко

ВЕБ-СЕРВІС ДЛЯ СИСТЕМ УПРАВЛІННЯ РЕСУРСАМИ ПІДПРИЄМСТВА

Розроблено веб-сервіс для систем управління ресурсами підприємства, що дозволяє створювати та управляти швидкими захищеними системами управління ресурсами підприємства на основі необхідних користувачеві модулів. Унікальність сервісу полягає в тому, що окрім наявності більшості вже існуючих стандартів та можливостей інших ERP систем від інших IT-компаній, він має суттєву перевагу – в його використанні відсутня необхідність звертання до IT-спеціалістів задля розробки та налаштування ERP системи під потреби кінцевого користувача. Здійснено розробку головного веб-сайту веб-сервісу, який передбачає автентифікацію та авторизацію; додатку інсталятору для створення і запуску ERP системи; додаток серверу ERP системи, та клієнтського додатку для взаємодії із сервером ERP системи. Передбачено створення облікових записів працівників компанії у ERP системі, їх розподіл за відділами для доступу до встановлених модулів, а також створення системи як у локальному дата-центрі, так і у хмарному середовищі. Проведено функціональне тестування розробленого веб-сервісу через тестування кожного додатку окремо та у спільному взаємозв'язку.
Ключові слова: система управління ресурсами підприємства, ERP система, модульність, мережева взаємодія.

Вступ

Процес організації бізнесу все більше компаній та корпорацій починають із встановлення системи управління її ресурсами та загальним функціоналом автоматизації бізнес-процесів усередині компанії. Відповідно, для вирішення даного питання бізнес звертається до послуг IT-спеціалістів для створення даної системи. В більшості випадків ці системи мали схожий функціонал, схожі сервіси й можливості, тому із часом у деяких провідних IT-компаній виникла ідея написання даної системи із усіма основними функціями та можливостями та постачання її як IT-послуги. Але оскільки ніколи не відомо, який функціонал буде необхідним потенційному клієнтові, від процесу створення програмних доповнень до вже існуючої системи постачальники даного сервісу вирішили не відмовлятися. В результаті маємо ERP систему із базовим функціоналом, яка за необхідності доповнюється потрібними клієнтові механізмами, та постачається даному клієнтові у вигляді IT-послуги. Клієнтові це вигідно, бо йому не потрібно витрачати час та ресурси (в основному грошові – на найман-ня IT-спеціалістів на розробку системи), а постачальнику системи дає прибуток за користування цією системою клієнтом.

У даній роботі натомість реалізовано систему, призначену для використання як IT-послуги на потенційних комерційних засадах. Кінцевими користувачами даної системи є адміністратори або IT-спеціалісти компаній, яким поставлена задача впровадження ERP системи у даній компанії.

1. Стратегії та основні компоненти

ERP система – інформаційна система управління ресурсами підприємства, належить до програмного забезпечення, що їх компанії, корпорації та організації використовують для управління повсякденними процесами всередині своєї бізнес-структури, такими як бухгалтерський облік, управління проєктами, закупівлею, операціями з ланцюгами поставок тощо. «Під інформаційною розуміють будь-яку систему, яка за допомогою технічних засобів виконує одну, або кілька функцій, як збирання, передавання, перетворення, накопичення, зберігання, та обробка інформації» [1-2].

Повний пакет ERP системи також включає управління ефективністю діяльності підприємства, а також програмне забезпечення, яке полегшує процеси планування, складання бюджету, прогнозу-

вання та звітності про фінансові досягнення компанії [3].

До основних характеристик стратегії ERP належать:

- централізація: використання єдиної системи транзакцій для переважної більшості операцій та бізнес-процесів підприємства. Усі операції об'єднані в єдину базу даних для подальшої обробки та збалансованих планів у реальному часі;

- відтворюваність: забезпечення можливості використання одного пакету програмного забезпечення для різних організацій (можливо, з різними налаштуваннями та розширеннями);

- локалізація: підтримка в єдиній системі декількох валют та мов;

- підтримка декількох юридичних осіб, кількох підприємств, декількох облікових політик та різних схем оподаткування в єдиній системі; це необхідно для використання у корпораціях, зокрема, транснаціональних.

Окрім цього, є ключові компоненти сучасної ERP-системи, які мають вирішити головні проблеми більшості компаній у розгляді питання про перехід до хмари. Є сім компонентів, які поділяються на дві категорії – сучасні параметри платформи та сучасний дизайн бізнес-додатків. Разом ці сім компонентів визначають стандарти сучасної ERP-системи. До того ж, сучасні хмарні додатки мають масштабуватися відповідно до вимог бізнесу та підтримувати новітні цифрові технології для задоволення потреб організації [4].

До вищезгаданих компонентів сучасної ERP-системи належать:

- безпека. Багатошаровий підхід до захисту даних на кожному шарі стеку є головним для максимального захисту даних [5]. Використання безпечної архітектури ізоляції даних у хмарі зменшує ризик та забезпечує швидший доступ до даних та їх обробку;

- інтеграція. Хмарні рішення ERP мають безперешкодно поєднувати бізнес, людей та процеси. Також має бути продумана та реалізована можливість підключення до інших хмар, локальних систем та сторонніх рішень. Для забезпечення сумісності та масштабованості слід вибрати рішення, яке використовує загальну структуру, засновану на галузевих стандартах;

- персоналізація. Хмарні рішення, побудовані на платформі, що базується на стандартах, пропонують персоналізацію та конфігурацію в додатку, що призводить до покращень, безпечних для оновлення. Якщо локальні налаштування клієнта потрапляють у такі сфери, як робочі процеси, інтеграція та звітування, є велика ймовірність вирішення вимог бізнесу за допомогою хмарних рішень;

- повнота. Вбудовані найкращі практики підтримують стандартизацію, що дозволяє знизити витрати та збільшити продуктивність. Навіть якщо хмарний перехід поступовий, доступ до повного набору інтегрованих передових бізнес-процесів забезпечує стандартизацію підприємств;

- глобалізація. Вихід на нові ринки та географії створюють складність, вимагаючи будь-якого хмарного рішення ERP для підтримки декількох дочірніх компаній та локалізації країн. Часто місцеві центри обробки даних повинні відповідати вимогам щодо збереження даних. Правильний хмарний ERP має забезпечувати безперебійний обмін корпоративною інформацією між операціями, бізнес-підрозділами та штаб-квартирою;

- аналіз, заснований на аналізі. Хмарне ERP-рішення повинно мати надійний доступ до даних у реальному часі у своїй основі для забезпечення єдиного джерела істинності між ролями, звітами та аналізом. Це забезпечує своєчасну доставку точних показників ефективності для передових менеджерів та суттєво спрощує процес у разі його залежності від періоду закриття або окремого витягу сховища даних;

- цифрові можливості. Для сучасного бек-офісу цифрові технології маютьбути інтегровані в бізнес-процеси та операції для створення цілісного, продуктивного та інтуїтивно зрозумілого досвіду. Користувацький досвід повинен бути цікавим і включати мобільну доступність для постійних співробітників, власну соціальну інтеграцію для безпечної спіль-

ної роботи в контексті та нестандартне оптичне розпізнавання символів для зображення рахунків-фактур.

2. Огляд аналогів

Одним із найбільш популярних та передових сервісів ERP систем є розробка від технологічного гіганта Microsoft – MS Dynamics 365 BC. Це програмне рішення управління для малого та середнього бізнесу з метою автоматизації бізнес процесів. Даний продукт є частиною сімейства Microsoft Dynamics [6]. Насамперед цей продукт є веб-орієнтованим SaaS (англ. «Software as a Service» – «програмне забезпечення як сервіс») рішенням. До версії #14 була доступна локальна версія, у складі налаштувань якої була можливість використання спеціально налаштованого клієнта. Починаючи із версії #15, доступ до системи здійснюється лише через Інтернет на веб сайті dynamics.com.

Іншим не менш популярним та не менш передовим сервісом ERP систем відносно іншого технологічного гіганта – компанії Oracle – є система Oracle ERP [7]. Це інтегрований хмарний комплекс додатків для управління фінансами, закупівлею і портфелем проєктів для малого та середнього бізнесу. Даний продукт побудований на міцному об'єднанні платформи, додатків та підходу з використанням фінансових, закупівельних, портфельних проєктів та управління ефективністю діяльності підприємств.

На відміну від MS Dynamics 365 BC, дана система доступна лише у вигляді хмарного сервісу, без можливості локального встановлення або гібридного розгортання.

В обох даних аналогах присутній суттєвий недолік – кінцевий користувач має звертатися до постачальника даної системи (Microsoft, Oracle тощо) задля налаштування системи під свої потреби. Розроблена система даного недоліка позбувається.

3. Розробка API

API сервер даної системи – її головний компонент. Взаємодія із зовнішніми запитами у даному проєкті реалізована у вигляді API контролерів – набору методів, які приймають HTTP GET, POST, PUT і DELETE запити, та виконують певну бізнес-логіку.

У даному API присутні три контролери:

- Modules: даний контролер відповідає за роботу із репозиторієм модулів із БД. В ньому присутні наступні методи:

1) GET api/modules?moduleID= повертає детальну інформацію про модуль за його ідентифікатором;

2) POST api/modules/all = повертає масив усіх модулів із врахуванням фільтру (фільтр передається через Body у запиті);

3) POST api/modules/non-default = повертає масив усіх модулів не за замовчуванням із врахуванням фільтру (фільтр передається через Body у запиті);

4) POST api/modules/marketplace = повертає масив усіх невстановлених на вказаній ERP системі (через Body у запиті передається ідентифікатор системи та фільтр);

5) POST api/modules/marketplace/add = запит на додавання модулю до вказаної ERP системи (через Body у запиті передаються ідентифікатори системи та модулю);

6) POST api/modules/marketplace/remove = запит на видалення модулю із вказаної ERP системи (через Body у запиті передаються ідентифікатори системи та модулю);

- ErpSystems: основний контролер у даному додатку. Відповідає за роботу із репозиторієм ERP систем із базою даних. Даний контролер захищений авторизацією (для виклику будь-якого методу необхідно спочатку успішно авторизуватися у системі) через токени, час дії яких – 20 хвилин.

У ньому присутні наступні методи:

1) GET api/erp-systems/check-company?company= перевіряє наявність у головній базі даних сервісу ERP системи із вказаною назвою компанії;

2) GET api/erp-systems/installed-modules?erpID= повертає масив встановлених у вказаній ERP системі модулів за ідентифікатором ERP системи;

3) POST api/erp-systems/user-erp-systems?userID= повертає відфільтрований масив ERP систем, які належать вказаному користувачеві за ідентифікатором користувача (фільтр передається через Body у запиті);

4) POST api/erp-systems виконує запит на створення ERP системи користувачем;

5) DELETE api/erp-systems/erp-systemID= виконує запит на видалення ERP системи користувачем за ідентифікатором ERP системи.

- Users – даний контролер відповідає за роботу із репозиторієм користувачів у базі даних. Аналогічно контролеру «ErpSystems», даний контролер також захищений авторизацією. Володіє наступними методами:

1) POST api/users/login виконує запит на авторизацію користувача. Модель на авторизацію передається через Body запиту;

2) POST api/users виконує запит на реєстрацію користувача. Модель на реєстрацію передається через Body запиту;

3) GET api/users/confirm-email?userID= виконує запит на підтвердження електронної пошти за ідентифікатором користувача;

4) GET api/users/forgot-password?email= виконує запит на надсилання листа для відновлення паролю за вказаною електронною поштою;

5) POST api/users/restore-password виконує запит на відновлення паролю. Модель на відновлення паролю передається через Body запиту;

6) POST api/users/change-password виконує запит на зміну паролю. Модель на зміну паролю передається через Body запиту;

7) PUT api/users виконує запит на оновлення даних про користувача (електронна пошта, логін тощо). Модель із новими даними передається через Body запиту;

8) DELETE api/users виконує запит на видалення облікового запису користувача. Модель із користувачем передається через Body запиту.

4. Розробка головного веб-сайту

Головний веб-додаток повинен слугувати відразу як ознайомчий веб-сайт, на якому користувач може ознайомитися із усіма послугами даного сервісу (модулями (їхніми описами та деталями) та серверним і клієнтським додатками), так і функціональний веб-застосунок.

Вибираючи потрібну сторінку, користувач може знайти необхідну інформацію про модулі, серверний додаток ERP системи та клієнтський додаток.

Для початку було розроблено 3 модулі: модуль управління (встановлений у системі за замовчування та надає доступ до таблиць працівників компанії, відділів та власне модулів), модуль фінансів (надає базовий функціонал управління бухгалтерським обліком) та модуль логістики (надає базовий функціонал для управління оборотом товарів, транспортування тощо).

5. Розробка інсталятора

Інсталятор – головний інструмент користувача для створення ERP системи. Головна мета даного застосунку – дати користувачеві механізм попередніх налаштувань серверу перед його початком роботи.

Важливо зазначити, що перед встановленням системи, необхідно налаштувати правила фаєрволу, щоб зробити комп'ютер, на якому буде встановлений сервер, доступним за його публічною IP адресою.

Задля забезпечення коректної роботи реєстрації систем, даний додаток взаємодіє із головним API сервером. Це відбувається на трьох етапах:

- Авторизація – для запуску системи користувач повинен мати обліковий запис у системі;

- Перевірка наявності системи за назвою компанії – в процесі успішного створення системи їй буде наданий домен. Цей домен генерується залежно саме від назви компанії власника системи за формулою 1:

Домен = «erp-»+назва_компанії+ «.com» (1)

Даний домен слугуватиме ключем шифрування адреси серверу системи (його публічного IP та порту), результатом шифрування буде ключ організації, який надалі використовуватиметься працівниками для успішної авторизації у системі через клієнтський додаток;

- Реєстрація – власне процес відправлення запиту на API для реєстрації нової системи.

6. Розробка серверу ERP системи

Головний компонент – сервер ERP системи – є, по суті, веб-додатком, призначеним для взаємозв'язку між робітниками компанії та базою даних. Мережева взаємодія буде досягнута шляхом використання API контролеру серверу ERP системи.

Сам додаток працює в об'єднанні із інсталятором. Після успішної реєстрації системи через інсталятор генерується JSON файл із налаштуваннями системи. Даний JSON передається у виконуючий файл серверу, де на основі даних налаштувань і конфігурується система. До даних налаштувань належать:

- публічна та приватна IP адреси;
- порт самого додатку, за яким у веб-браузері можна відкрити веб-інтерфейс серверу;
- ключі організації та шифрування;
- домен та ідентифікатор ERP системи;
- перелік модулів;
- шлях підключення до бази даних;
- назва компанії.

Додаток складається із набору сервісів, кожен із яких виконує певну задачу, та головного сервісу, котрий вирішує, який сервіс (або сервіси) необхідно використати залежно від запиту користувача.

Веб-інтерфейс серверу є його візуальною частиною. Він створений для полегшення роботи адміністратора системи і надає весь необхідний інструментарій.

Єдиний спосіб для працівників отримати доступ до функціоналу модулів - це бути частиною відділу, до якого прикріплені необхідні модулі. По суті, відділ є логічним об'єднанням працівників із метою надання їм певних модулів. Тому для полегшення роботи адміністраторів під час запуску серверу буте автоматично створений відділ із доступом до всіх встановлених модулів. Для початку найму нових працівників достатньо створити відділ із доступом до модулю управління (який, як вже було зазначено, встановлено в системі за замовчуванням), та обліковий запис головного HR (англ. «Human Resources») менеджера і закріпити його за даним відділом.

Створюючи обліковй запис нового працівника, необхідно вказати адресу його електронної пошти, на яку надійде лист із його логіном, паролем (котрий генерується системою закрито від адміністратора та буде відомий лише самому працівнику в листі) та ключем організації. Логін працівника генерується за формулою 2:

$$\text{Логін} = \text{нікнейм\_пошти} + \text{«@»} + \text{домен\_системи}, \quad (2)$$

де нікнейм_пошти це набір символів із електронної пошти працівника до символу «@» виключно.

Використовуючи дані із електронного листа та клієнтський додаток, працівник компанії може успішно авторизуватися у системі та починаючи роботу.

Аналогічно користувач отримуватиме листи при оновленні або видаленні його облікових записів.

Головний ERP сервіс є основним компонентом, котрий має доступ до усіх інших сервісів та здійснює на них керуючий вплив. Саме даний сервіс обробляє запит користувача та виконує певні дії для виконання необхідної операції.

Не менш ваговим є сервіс взаємодії із базою даних, оскільки саме через нього відбувається взаємодія із базою даних ERP системи. Алгоритм дій залежить від операцій у БД, які потребує користувач. Таких операцій може бути сім:

- GetByID – пошук за ідентифікатором;
- Select – повна вибірка із таблиці;
- Insert – запис до таблиці;
- Update – оновлення запису у таблиці;
- Delete – видалення із таблиці;

- SelfUpdate – аналогічно оновлення у таблиці, проте дана операція викликається лише у разі оновлення користувачем власного профілю у клієнтському додатку;

- CreateDB – створення БД. Дана операція викликається лише один раз – на самому початку роботи сервера для створення усіх таблиць на основі встановлених модулів.

Сервіс зовнішньої інтеграції є обов'язковим згідно з технічними та функціональними умовами, та слугує основною ланкою у комунікації між працівниками та системою.

Даний інструмент реалізований у вигляді API контролеру, який має один єдиний HTTP POST метод. Даний метод приймає модель запиту у Body HTTP запиту. Варто зазначити, що відповідно до технічних вимог, запити не передаються мережею в чистому вигляді. Тому на даний сервіс надходять лише зашифровані повідомлення. Після цього, згідно з блок-схемами алгоритму дешифрування повідомлення, відбувається процес декодування (рисунок 1):

Блок 1 – прийом закодованого повідомлення;

Блок 2 – Перевірка, чи починається повідомлення на спеціальний роздільник. Якщо так, то перейти до блоку 4, інакше – перейти до блоку 3;

Блок 3 – Дешифрування повідомлення доменом системи;

Блок 4 – Видалити роздільник із початку повідомлення;

Блок 5 – Дешифрування повідомлення головним секретним ключем;

Блок 6 – Перевірка успішності дешифрування повідомлення. Дана перевірка спрацьовує, якщо під час декодування не виникає жодного виключення (Exception). Якщо ні, то у моделі відповіді позначається, що вона не була успішна та вказується повідомлення про помилку, інакше – перейти до блоку 7;

Блок 7 – При успішному дешифруванні результатом виходить модель запиту у форматі JSON, тому для подальшої роботи необхідно конвертувати даний формат у об'єкт класу.

Після цього об'єкт запиту передається на головний ERP сервіс, де виконується уся необхідна бізнес-логіка. Після цього модель відповіді шифрується згідно алгоритму кодування (рисунок 2):

Блок 1 – надходить модель відповіді;

Блок 2 – модель відповіді серіалізується у формат JSON;

Блок 3 – перевірка операції запиту. Якщо відбувалась операція авторизації у системі (Login), перейти до блоку 5, інакше – перейти до блоку 4;

Блок 4 – зашифрувати JSON головним шифрувальним ключем;

Блок 5 – зашифрувати JSON доменом системи;

Блок 6 – дописати на початок повідомлення роздільник.

У кінці процедури зашифрована відповідь повертається користувачеві.

Задля коректної роботи додатку був використаний стандартний механізм проєктування додатків Dependency Injection. Для цього під час запуску додатку генерується колекція абстракцій сервісів, які будуть використані у роботі серверу та їх відповідних реалізацій. Приклад такого налаштування наведено далі.

```
public void ConfigureServices
(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddServerSideBlazor();
    services.AddHttpClient(); services.
    AddHttpContextAccessor(); var
    json = Configuration.GetValu
    e<string>(ConfigurationMak
    erService.ServerApplication);
    _erpSystem = Converter.
    FromJson<ErpSystem>(json);
    services.AddSingleton(_erpSystem);
    services.AddTransient<Core.
    Domain.Configuration.
    IConfigurationService,
    ConfigurationService>(); services.
    AddTransient<IServerConfig
    urationService,Configuration
    Service>(); services.AddTran
    sient<IRequestSenderService,
    RequestSenderService>();
```

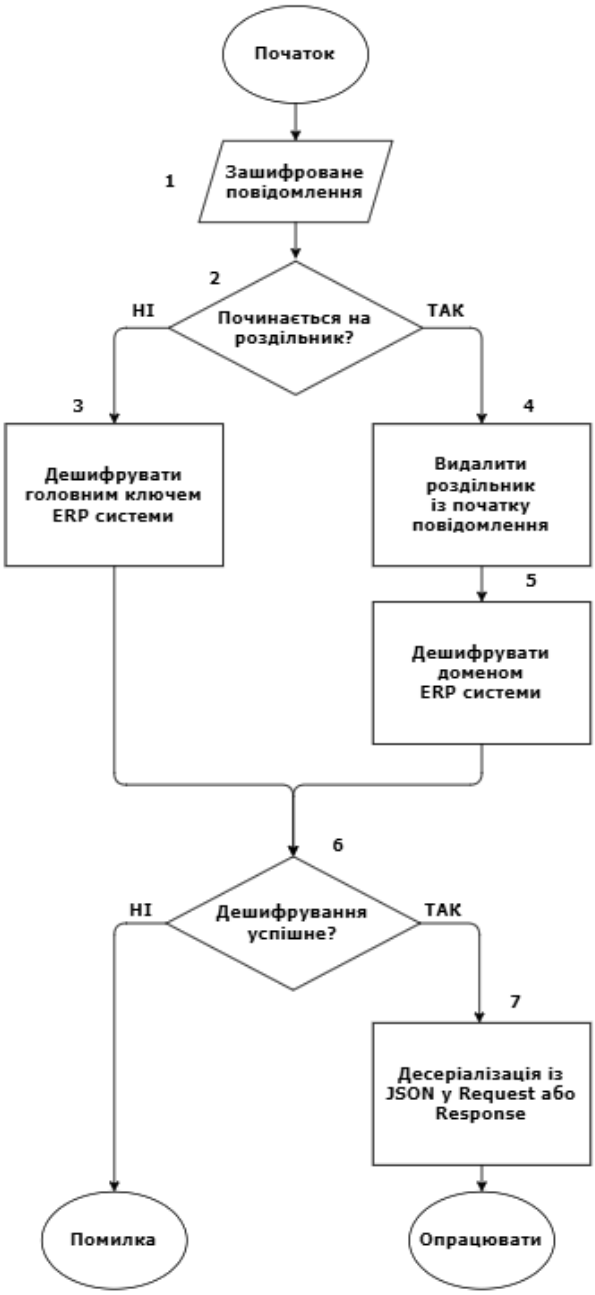


Рис.1 – Блок-схема алгоритму декодування

Рис.2 – Блок-схема алгоритму кодування

```
switch (_erpSystem.DatabaseType)
{
    case DatabaseType.SQL:
    services.AddTransient<IDatabas
    eService,RelationalDatabaseServ
    ice>();
    break;

    case DatabaseType.NoSQL:
    switch(_erpSystem.
    DatabaseProvider)
    {
    caseDatabaseProvider.
```

```
DynamoDb:
services.
AddTransient<IDatabaseService,
DynamoDatabaseService>();
break;
case DatabaseProvider.MongoDb:
case DatabaseProvider.
DocumentDb:
MongoDatabaseService.
RegisterGuidSerializer();
services.
AddTransient<IDatabaseService,
MongoDatabaseService>();
```

```
            break;
            }
            break;
        }
        services.AddTransient<IEmailSende
rService, EmailSenderService>();
        services.
AddTransient<ISystemTerminator,
SystemTerminator>();
        services.AddTransient<ISettingsSav
erService, SettingsSaverService>();
        services.
AddTransient<IMainErpService,
MainErpService>();
        services.AddTransient<IErpService
Encryption, ErpServiceEncryption>();
        services.
AddTransient<IScreenService,
ScreenService>();
        services.AddTransient<ILocalizatio
nService, LocalizationService>();
        services.
AddScoped<ICookieService,
CookieService>();
        services.
AddSingleton<IStateService,
ServerStateService>();
        services.
AddSingleton<IEventBusService,
EventBusService>();
        }
```

7. Розробка клієнтського додатку ERP системи

Працівники компанії також повинні мати можливість працювати із ERP системою своєї компанії. Даний додаток є програмними засобом, який надає цю можливість. Цей застосунок є кросплатформним, що робить можливим запускати його на будь-якій операційній системі (Windows, Linux або MacOS).

Аналогічно до серверу ERP системи, даний додаток також поділений на сервіси. Всього їх два:

- сервіс зовнішньої інтеграції: аналогічно до серверу, це API сервіс, який протоколом HTTP та аналогічними алгоритмами шифрування та дешифрування надсилає на сервер системи запити та отримує відповіді;

- головний сервіс, який по суті є обгорткою API сервісу, з метою приховати деталі реалізації зовнішньої взаємодії та не працювати безпосередньо із API сервісом, що полегшить тестування та дасть можливість розширювати його додатковим функціоналом.

Стартовим вікном у даному додатку є сторінка авторизації. Під час авторизації користувач має ввести свої логін, пароль та ключ організації (усі ці дані будуть відомі у електронному листі, який йому надійде після створення його облікового запису). Оскільки ключ організації - це зашифровані доменом системи, її публічна IP адреса та порт, а домен буде частиною логіну користувача, то шляхом дешифрування ключа організації отримується загальна адреса API серверу ERP системи, із яким налаштовується на взаємодію API сервіс клієнтського додатку. На нього і надсилається запит на авторизацію. У разі успішної авторизації, користувач переходить до головної сторінки. А ні – користувач отримує повідомлення про проблему авторизації.

Після авторизації користувачеві надається можливість до всіх оперувати даними кожного доступного модуля. Для кожної таблиці кожного модуля постачальнику необхідно на програмному рівні реалізувати візуальні форми для заповнення та інші графічні елементи.

У даному додатку для кожної таблиці створюється три візуальні компоненти:

- DataGridView – відображення усіх даних таблиці у вигляді самої таблиці;

- CardView – картка конкретного запису таблиці із короткими відомостями про елемент та з переходом до форми елементу;

- FormView – форма елементу, де повною мірою розкривається інформація про нього та можливе доповнення додатковими даними. Також саме через форму відбувається створення нового елементу, його редагування або видалення.

Окрім цього, присутні дві окремі сторінки:

- DepartmentView – сторінка відділу працівника;

- ProfileView – профіль працівника, де він матиме можливість редагувати такі свої персональні дані як логін, пароль, номер телефону або адресу електронної пошти.

У момент закриття додатку система надсилає запит до серверу із операцією Disconnect, під час чого на сервері даний користувач видаляється із переліку онлайн користувачів.

8. Експеримент

Для визначення ефективності пере-твореної програми проведені експери-менти щодо кожного додатку системи.

Передовсім було проведено додаток головного веб-сайту. Процес реєстрації відбувається із використанням логіну, адреси електронної пошти та паролю. Одночасно відбувається валідація на наявність у системі користувача із вказаним логіном або імейлом. Процес авторизації відбувається із введенням логіну або імейлу та паролю. За умови успішної авторизації користувач потрапляє на сторінку свого облікового запису, де він може перейти до налаштувань свого профілю або до переліку своїх ERP систем.

На сторінці «Your ERP systems» користувач знайде усі свої створені ERP системи. У даній таблиці подається лише основна інформація (назва компанії, ключ організації, домен, тип бази даних). Розкриваючи кожний запис таблиці, можна додатково дізнатися адресу даної системи або перейти до веб-консолі вибраної ERP системи.

Наступним перевіряється додаток інсталятора. Налаштувати та запустити сервер системи можливо лише із використанням додатку інсталятора. Сам процес інсталяції складається із 4 кроків:

– Авторизація – користувач має бути зареєстрований у системі;

– Перевірка назви компанії – назва компанії має бути унікальна в межах сервісу;

– Налаштування серверу– необхідно вказати порт роботи серверу, а також вказати адресу підключення до бази даних;

– Вибір модулів, необхідних для встановлення.

Далі, переглянувши свої налаштування та підтвердивши їх, система реєструється на API та запускається на даному сервері. Після успішного запуску вказує користувачу відповідним повідомленням.

Потім відбувається перевірка роботоздатності сервера ERP системи. Перейшовши на порт веб-інтерфейсу, користувач переходить на його головну сторінку. Згідно з функціональними умовами, користувачу надається можливість оперувати відділами та обліковими записами працівників, а також переглядати записи усіх таблиць кожного модуля.

Останньою здійснюється перевірка роботоздатності клієнтського додатку для працівників. Запустивши клієнтський додаток, користувач має спочатку авторизуватись із використанням логіну, паролю та ключа організації. Успішно авторизувавшись, користувач потрапляє на головну сторінку, де можна побачити доступні йому модулі.

Перейшовши на сторінку відповідного модуля, користувач може відразу почати оперувати даними даного модуля.

Звісно, кількість даних при тестуванні була мала, але варто зазначити, що і реальні сервери будуть набагато потужніші за комп'ютер, на якому проводилися тестування (процесор Intel Core i7-8550K, 8 гігабайт оперативної пам'яті, 256 гігабайт SSD).

Висновки

Реалізовано сервіс систем управління ресурсів підприємства, що відповідає усім висунутим технічним та функціональним вимогам. На основі інструментарію та функціональних можливостей систем аналогів був сформований перелік технічних та функціональних вимог до проєкту. На базі даних вимог були обрані технології та принципи проєктування, які стануть у пригоді та полегшать процес розробки та тестування системи. За допомогою сучасних шаблонів проєктування була розроблена архітектура системи, яка в перспективі може стати основою справжнього комерційного сервісу. Завдяки вищезазначеній концепції модульності дана система легко доповнюється та розширюється. В перспективі

сервіс можна доповнити впровадженням ERP систем не лише на сервері клієнта, а й у хмарі, як це пропонують компанії Oracle та Microsoft, а також доповнювати його новим функціоналом, залежно від вимог бізнесу.

1. *Жураковський Ю.П.* Теорія інформації та кодування: Підручник / *Ю.П. Жураковський, В.П. Полторак*. - К.: Вища шк, 2001. - 255 с.
2. *Полторак, В.П.* Інформаційна безпека та захист даних в комп'ютерних технологіях і мережах [Електронний ресурс]: навчальний посібник для студентів спеціальності 126 «Інформаційні системи та технології» / В. П. Полторак ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,77 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 78 с.
3. Top 3 Essential ERP Modules [Електронний ресурс]. – режим доступу: https://www.chetu.com/blogs/technical-perspectives/oracle-erp-modules.php
4. Что такое ERP и зачем это нужно? [Електронний ресурс] – Режим доступу до ресурсу: https://dynamics.microsoft.com/ru-ru/erp/what-is-erp/.
5. What is AES Encryption and How Does it Work? [Електронний ресурс]. – режим доступу: https://searchsecurity.techtarget.com/definition/AdvancedEncryption-Standard
6. Microsoft Dynamics 365 Business Central [Електронний ресурс]. – режим доступу: https://dynamics.microsoft.com/en-cy/
7. Oracle ERP [Електронний ресурс]. – режим доступу: https://www.oracle.com/erp/

References

1. *Zhurakovskiy Y.P., Poltorak V.P.* Theory of information and coding: Textbook, 2001. – p.10 (in Ukrainian).
2. *Poltorak V.P.* Information security and data protection in computer technology and networks [Online]: textbook for students majoring in 126 "Information Systems and Technologies", 2020. – p.78 (in Ukrainian).
3. Top 3 Essential ERP Modules [Online] – Available from: https://www.chetu.com/blogs/technical-perspectives/oracle-erp-modules.php.
4. What is ERP and why it is necessary? [Online] – Available from: https://dynamics.microsoft.com/ru-ru/erp/what-is-erp/ (in Ukrainian).
5. What is AES Encryption and How Does it Work? [Online]. – Available from: https://searchsecurity.techtarget.com/definition/AdvancedEncryption-Standard (in Ukrainian).
6. Microsoft Dynamics 365 Business Central [Online]. – Available from: https://dynamics.microsoft.com/en-cy/
7. Oracle ERP [Online]. – Available from: https://www.oracle.com/erp/

Отримано: 17.05.2022

Про авторів:

Тарасенко Артем Дмитрович,
магістрант Національного Технічного Університету України «КПІ імені Ігоря Сікорського». https://orcid.org/0000-0003-2631-2558,
Дорошенко Анатолій Юхимович, доктор фізико-математичних наук, професор, завідувач відділу теорії комп'ютерних обчислень, професор кафедри інформаційних систем та технологій Національного Технічного Університету України «КПІ імені Ігоря Сікорського». Кількість наукових публікацій в українських виданнях – понад 190. Кількість наукових публікацій в зарубіжних виданнях – понад 80. Індекс Гірша – 6. http://orcid.org/0000-0002-8435-1451

Місце роботи авторів:

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», проспект Перемоги 37 та Інститут програмних систем НАН України, 03187, м. Київ-187, проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559
E-mail:
artemissterio@gmail.com,
doroshenkoanatoliy2@gmail.com

УДК 004.94 http://doi.org/10.15407/pp2022.02.057

В.А. Резніченко

60 РОКІВ БАЗАМ ДАНИХ (четверта частина)

Наводиться огляд досліджень і розробок баз даних із моменту їх виникнення в 60-х роках минулого століття і до сьогодні. Виділяються наступні етапи: виникнення і становлення, бурхливий розвиток, епоха реляційних баз даних, розширені реляційні бази даних, постреляційні бази даних і великі дані. На етапі становлення описуються системи IDS, IMS, Total і Adabas. На етапі бурхливого розвитку висвітлені питання архітектури баз даних ANSI/X3/SPARC, пропозицій КОДАСИЛ, концепції і мов концептуального моделювання. На етапі епохи реляційних баз даних розкриваються результати наукової діяльності Е. Кодда, теорія залежностей і нормальних форм, мови запитів, експериментальні дослідження і розробки, оптимізація та стандар¬тизація, управління транзакціями. Етап розширених реляційних баз даних присвячений опису темпоральних, просторових, дедуктивних, активних, об'єктних, розподілених та статистичних баз даних, баз даних масивів, машин баз даних і сховищ даних. На наступному етапі розкрита проблематика постреляційних баз даних, а саме NOSQL-, NewSQL- і онтологічних баз даних. Шостий етап присвячений розкриттю причин виникнення, характерних властивостей, класифікації, принципів роботи, методів і технологій великих даних. Нарешті, в останньому розділі подано короткий огляд досліджень і розробок із баз даних у Радянському Союзі.

Ключові слова. Типи баз даних: ієрархічна, мережева, реляційна, навігаційна, темпоральна, просторова, просторово-темпоральна, просторово-мережева, об'єктів, що переміщуються, дедуктивна, активна, об'єктно-орієнтована, об'єктно-реляційна, розподілена, паралельна, масивів, статистична, багатовимірна, машина баз даних, сховища даних, NoSQL, ключ-значення, стовпчикова, документно-орієнтована, графова, мультимодельна, хмарна, наукова, багатозначна, XML, NewSQL, онтологічна, великі дані.

Етап 5. Постреляційні бази даних (2000 – 2010+)

Проникнення Інтернету в усі сфери нашого життя викликало суттєве зростання кількості джерел даних, неймовірне збільшення їх обсягу та інтенсивності використання. Це призвело до проблем зберігання, обробки і до труднощів з оперуванням неструктурованою інформацією. Класичні реляційні СУБД, які вірно служили людству протягом 40 років, виявилися неспроможними впоратися із таким новим викликом. У зв'язку з цим з'явився новий напрямок у БД, який дістав назву NoSQL, в результаті з'явилися БД ключ – значення, документні, колончаті, графові. Однак прихильники реляційних БД вирішили не здаватися без бою і спрямували свої зусилля на таку модернізацію реляційної концепції, за якої вдалося б подолати цей виклик початку нового століття. Так виник напрямок NewSQL. Тоді ж виникла концепція семантичного вебу, мета якого – підвищити семантику вебу задля створення механізмів більш релевантного пошуку. Найважливішою складовою такого вебу стало поняття онтології. Одним із варіантів збереження онтологій стали онтологічні бази даних. Зрештою в цей період почали бурхливо розвиватися повнотекстові БД та електронні бібліотеки. Стислому аналізу цих складових постреляційних БД і присвячено даний розділ.

NoSQL – бази даних

У 2000-і роки з появою вебресурсів із величезними сховищами різнорідної інформації дослідники почали частіше аналізувати нові структури даних. Реляційний підхід заснований на чіткому структуруванні даних і строго формалізованому доступі до них. Це робить БД негнучкими і тим самим уповільнює швидкість роботи. Новий підхід базувався на відмові від фіксованої схеми даних і мови SQL.

NoSQL – термін, що означає низку підходів, спрямованих на реалізацію СУБД, які мають суттєві відмінності від

моделей, котрі використовуються в традиційних реляційних СУБД із доступом до даних засобами мови SQL.

Історія терміну NoSQL. Термін NoSQL уперше з'явився 1998 року й використовувався для опису реляційної БД, розробленої Карлом Строцці (Carlo Strozzi) [755]. Вона не використовувала SQL як мову запитів. Це початкове використання даного терміну нічого спільного не має із сучасною технологією NoSQL. Водночас у першому десятиріччі 21-го століття з'явилися Neo4j (2000), Google BigTable (2004), CouchDB (2005), Amazon Dynamo (2007), Hypertable (2007), Hbase (2007), Dynomite (2008), Voldemort (2009), Cassandra (2009), MongoDB (2009), що були нереляційними СУБД.

Карло Строцци

2009 року в Сан-Франциско Йохан Оскарссон (Johan Oskarsson) організував семінар для обговорення нових технологій із збереження й обробки даних [756]. Головним стимулом зустрічі була поява на ринку розподілених нереляційних продуктів. Ерік Еванс (Eric Evans) запропонував зробити яскравою вивіскою семінару ємний і лаконічний термін "NoSQL" [757]. Термін планувався лише на одну зустріч і не мав якогось глибокого смислового навантаження. Але сталося так, що він розповсюдився світовою мережею і став де-факто назвою цілого напряму в IT-індустрії. Водночас термін NoSQL не означає якусь одну конкретну технологію чи продукт. Наімовірніше він характеризує вектор розвитку IT у бік від реляційних баз даних.

Ерік Еванс

Властивості NoSQL баз даних.

Існує багато різних типів NoSQL БД, та для більшості з них характерні такі властивості:

- **Гнучкість.** Зазвичай бази даних NoSQL пропонують гнучкі схеми, що дозволяє здійснювати розробку швидше й уможливлює поетапну реалізацію. Завдяки використанню гнучких моделей даних БД NoSQL добре підходить для частково структурованих і неструктурованих даних.

- **Горизонтальна (еластична) масштабованість.** Бази даних NoSQL розраховані на масштабування з використанням розподілених кластерів апаратного забезпечення, а не за рахунок додавання дорогих надійних серверів. Висока доступність за рахунок слабкої узгодженості (за рахунок спрощеної семантики ACID).

- **Висока продуктивність.** Бази даних NoSQL оптимізовані для конкретних моделей даних і механізмів доступу, що дозволяє досягнути вищої продуктивності порівняно із реляційними базами даних.

- **Широкі функціональні можливості.** БД NoSQL надають API та типи даних із широкою функціональністю, які спеціально розроблені для відповідних моделей даних.

- **Слабоструктуровані (schemaless) дані.** Структура даних не регламентована. Її можна динамічно змінювати.

- **Підтримка агрегатів.** NoSQL сховища оперують не лише атомарними, а й агрегатними об'єктами. В такому разі не потрібні нормалізовані відношення.

- **Розподіленй системи,** зазвичай без централізованого управління (децентралізовані).

- **Підтримка розподілених систем** без спільно використовуваних ресурсів (share nothing).

Аби зрозуміти, чого можна досягти внаслідок розробки і використання баз даних, було сформульовано твердження Брюера.

Теорема CAP, відома також як теорема Брюера (Brewer), - евристичне ствердження того, що в будь-якій реалізації розподілених обчислень можливо забезпечити не більше двох із трьох наступних властивостей:

- **узгодженість даних (consistency) -** в усіх обчислювальних вузлах у певний момент часу дані не суперечать одне одному (семантика ACID);

- **доступність (availability)** – будь – який запит до певної системи завершується коректною відповіддю, однак без гарантії, що відповіді всіх вузлів системи співпадають;

- **стійкість до розділення (partition tolerance)** – поділ розподіленої системи на кілька ізольованих секцій не призводить до некоректності відгуку від кожної із секцій.

Принцип був запропонований професором Каліфорнійського університету в Берклі Еріком Брюером (Eric Brewer) і Армандо Фоксом (Armando Fox) 1999 року [758], а 2002 року це твердження довели Сет Гільберт (Seth Gilbert) і Ненсі Лінч (Nancy Lynch) [759].

Ерік Брюер

Згодом теорема здобула широку популярність і визнання серед спеціалістів із розподілених обчислень. Концепція NoSQL, у межах якої створюються розподілені системи управління базами даних, часто-густо використовує цей принцип як обгрунтування неминучості відмови чи-то від узгодженості даних, чи від доступності.

На рис. нижче наводиться графічне представлення, на якому сторони трикутника відповідають парам властивостей теореми CAP, поруч наводяться приклади систем, що відповідають цим властивостям.

Класифікація систем згідно CAP

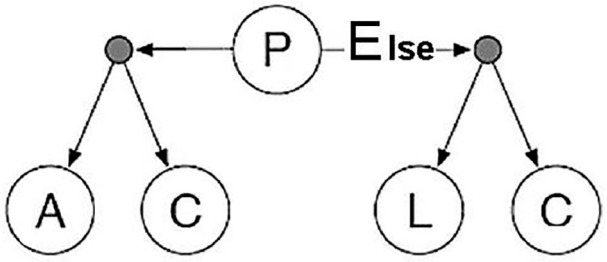
Теорема PACELC. Ця теорема, як і CAP, описує, які саме обмеження й компромісні рішення мають розподілені системи щодо узгодженості, доступності й стійкості до розподілення.

Проте, теорема PACELC додатково стверджує, що необхідно йти на компроміс між затримками отримання відповіді та узгодженням навіть за відсутності стійкості до розділення, що забезпечує краще уявлення про можливі компроміси для розподілених систем.

У згаданій теоремі використовуються не просто трикутник CAP, а наступний умовний вислів:

якщо (**P**)artition tolerance
то(**A**)valability або(**C**)onsistency
інакше (**L**)atency чи (**C**)onsistensy

Простіше кажучи, за умови стійкості до розділення (P) можна обрати (A) або узгодженість (C) (це теорема CAP), інакше (E)lse, якщо стійкості до розділення немає, можна вибрати між часом затримки (L) або узгодженістю ©. Ця ситуація графічно представлена на рис. нижче.



У такий спосіб ця теорема дає чотири різновиди розподілених систем:

PA/EL – висока доступність (A) за умови стійкості до розподілення (P) інакше (E), висока швидкість відповіді (L) (Dynamo, Cassandra, Cosmos DB, Riak);

PC/EC – в обох випадках обирається висока узгодженість (C) (Couchbase, VoltDB/H-Store, Megastore);

PC/EL – висока узгодженість (C) за умови стійкості до розподілення (P) інакше (E) висока швидкість відповіді (L) (PNUTS);

PA/EC – висока доступність (A) за умови стійкості до розподілення (P) інакше (E) висока узгодженість (C) (MongoDB).

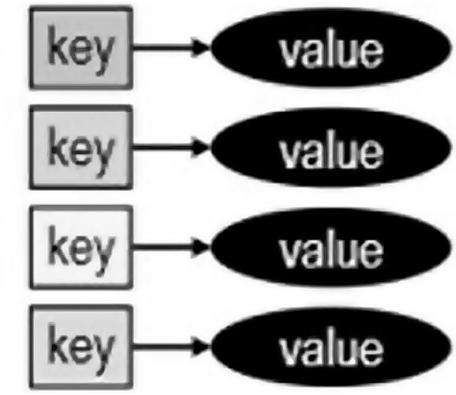
Теорема PACELC уперше була представлена в Інтернеті Даніелем Дж. Абаді (Daniel J. Abadi) з Йельського університету 2010 року, а згодом опублікована у вигляді статті 2012 року [760]. 2015 року він був нагороджений премією Very Large Data Base Endowment Inc. (VLDB) за найкращу статтю за попередні 10 років. 2020 року Даніель Дж. Абаді отримав звання «Дійсний член ACM» (ACM Fellow) за «великий внесок у потокові, розподілені, графові й колончаті бази даних».

Даніель Дж. Абаді

Типи NoSQL баз даних. Далі наводяться різні типи NoSQL баз даних, вказується на кілька з них, що належать кожному з типів. У [761] наводиться велика кількість різних систем класифікації NoSQL із широким зазначенням NoSQL баз даних, приналежних кожному з типів.

Модель асоціативного масиву (associative array model). Являє собою пару «ключ – значення». Асоціативний масив відображає ключ на значення, тобто асоціює значення з ключем. Значенням може бути як атомарна одиниця даних, так і складніша конструкція, як-от список. Формальний опис цієї моделі й алгебри наведено в [762, 763]. Праця [764] присвячена дослідженню застосування цієї моделі в базах даних.

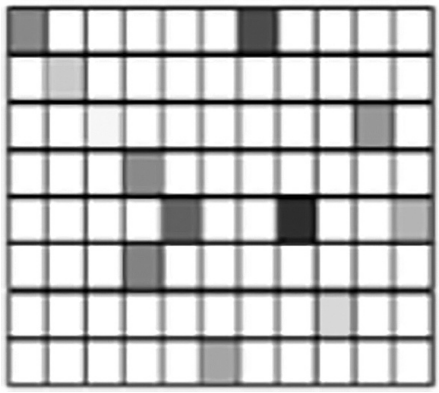
БД ключ – значення (key – value data base). Асоціативні масиви лягли в основу так званих баз даних «ключ – значення». Вважається, що ключ має бути унікальним.



Згідно із [765] першою системою баз даних типу «ключ – значення» була створена 1986 року система GT.M (Greystone Technology M) [766], призначена для високопродуктивної обробки транзакцій. За час, що відтоді минув, було створено безліч систем баз даних цього типу. На сайті DB – engines наведено 57 систем баз даних типу «ключ – значення» [767] і станом на січень 2022 року до трійки найпопулярніших належали Redis, Amazon Dynamo DB, Microsoft Azure Cosmos DB. Додаткова інформація щодо використання баз даних «ключ – значення» міститься в розділі «Великі дані».

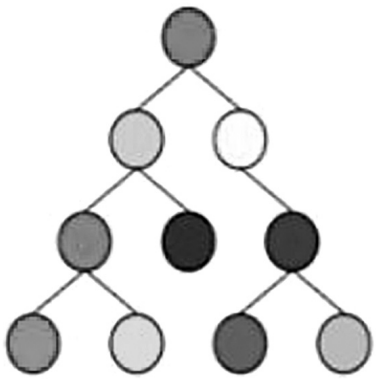
Модель триплетів (triple store model). Розширенням асоціативного масиву стала модель триплетів – три елементи, з'єднані у вираз «суб'єкт – предикат - об'єкт» [768], або в класичний елемент інформації «об'єкт – атрибут – значення». По суті триплет – це одна комірка класичного відношення. Завдяки зазначенню властивості, для якої наведено значення, триплети є інформативнішими за пари «ключ – значення». Кожному об'єкту може відповідати стільки триплетів, скільки властивостей має даний об'єкт. За адресою https://en.wikipedia.org/wiki/Comparison_of_triplestores наведено широкий список баз даних різних сховищ триплетів.

Колончаті БД (column – oriented databases, wide column store / column families). Бази даних, засновані на триплетах, дістали назву колончатих (поколончатих або баз даних, що складаються із сімейства стовпців).



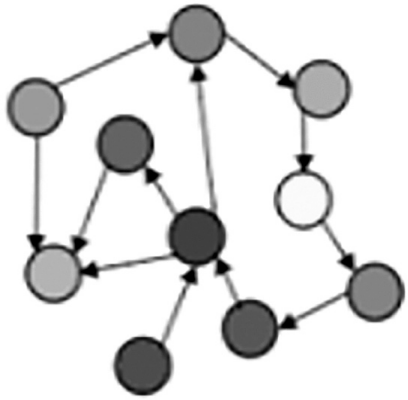
Назва пояснюється тим, що, зібравши разом усі триплети з однаковою властивістю (атрибутом), отримуємо одну колонку відношення. Представниками колончатих СУБД є DynamoDB, Google BigTable Cassandra, Scylla, HBase, Hypertable Mulgara, PNUTS тощо. Детальніше розкриття NoSQL баз даних цього типу наводиться в розділі «Колончаті бази даних».

Документно – орієнтовані БД (document – oriented databases). Об'єднання триплетів, що описують один об'єкт, називається документом. Значеннями можуть бути рядки, числа, масиви та інші вкладені триплети. Значення можуть вкладатися багаторазово. Бази даних, засновані на документах, дістали назву документно – орієнтованих. Зразками СУБД цього типу є IBM Domino, RavenDB, CouchDB, ThruDB, MongoDB, DocumentDB тощо. Детальніше про NoSQL баз даних цього типу можна ознайомитися в розділі «Документно – орієнтовані бази даних».



Графові БД (graph databases). Триплету також надається семантика «об'єкт – відношення - об'єкт». Такий триплет призначений для зберігання інформації,

яка в традиційних базах даних називається зв′язком. Представлення зв′язків між конкретними об′єктами дає змогу описати предметну область у вигляді семантичної мережі або графа, в якому об′єкти утворюють вузли, а відношення – дуги або ребра.



Такі БД дістали назву графових (graph database). Варто підкреслити, що графові моделі не новина: опис схожих баз даних можна знайти в монографії [2], виданій ще 1985 року. Там вони згадуються як бінарні бази даних. Завдяки інтуїтивній простоті й придатності для опису слабо структурованої інформації графові СУБД (Neo4j, AllegroGraph, InfiniteGraph, HyperGraphDB, OrientDB та інші) активно завойовують ринок. Детальніше розкриття NoSQL баз даних цього типу наводиться в розділі «Графові бази даних».

Структура GLOBAL

Зазначимо, що об′єкти, асоціативні масиви, триплети й документи – поняття схожі. В [769] розглядається універсальна структура GLOBAL, за допомогою якої можна представити будь – який із цих елементів. GLOBAL використовується у відомій СУБД Cashe.

Згідно з інформацією nosql-database.org, на січень 2022 року в світі існує понад 225 систем управління NoSQL – базами даних. На сайті наведено список цих СУБД із доволі повною їх класифікацією. Ця класифікація, окрім наведених вище типів NoSQL – баз даних, містить також:

- мультимодельні БД;
- об′єктні БД;
- БД мереж;
- хмарні БД;
- БД XML;
- багатовимірні БД;
- багатозначні БД;
- БД джерел подій (event sourcing);
- БД часових послідовностей / потокові БД (time series/streaming);
- Наукові й спеціалізовані БД.

Мови запитів. Була запропонована велика кількість NoSQL мов запитів, як-от AQL - ArangoDB Query Language, CQL - Cassandra Query Language, HQL - Hypertable Query Language, Cypher – мова запитів графічної бази даних Neo4j. Окрім того, компанія Couchbase, що розвиває такі системи, як Couch DB, Memcached і Membase, анонсувала створення нової мови запитів – UnQL (Unstructured Data Query Language) [770]. Вона являє собою надмножину класичної SQL, тобто багато в чому з нею сумісна, однак орієнтована на роботу з неструктурованими даними. Проєкт виконано спільними зусиллями Річарда Гіппа (Richard Hipp), творця SQLite, та Деміена Каца (Damien Kats), засновника проєкту Couch DB. Із оглядом усіх цих мов можна ознайомитися в [771]. Запропоновано чимало графових моделей баз даних, огляд яких наведено в [772].

Документно – орієнтовані бази даних.

Документно – орієнтовані бази даних (ДОБД) (document – oriented database) – це база даних, призначена для зберігання й маніпулювання документами. Документ – це деревовидний направлений граф із позначеними вершинами. Листові вершини представляють дані документа, а позначки (імена) решти вершин представляють властивості (атрибути) відповідних даних документа. Документи можуть об′єднуватися у колекції в певному сенсі однотипних документів, але водночас ніякі вимоги щодо однаковості складу атрибутів документів можуть і не ставитися. Колекції можуть містити інші колекції.

Документна структура належить до класу так званих безсхемних (schemaless)

самоописуваних (selfdescribing) структур. Як правило, в документній структурі відсутній розподіл на схему й екземпляр. Схеми немає, а всі необхідні структурні елементи схеми присутні в екземплярі, й у цьому сенсі дані є самоописуваними. Документно – орієнтовані дані також дістали назву слабоструктурованих / напівструктурованих (semistructured data).

Слабоструктуровані дані. Слабоструктурована модель даних заснована на ідеї представлення даних без явного і окремого визначення їхньої схеми. Натомість окремі фрагменти інформації перемежовуються структурними / семантичними тегами, що визначають їхню структуру, вкладеність та інші характеристики. Таке представлення забезпечує гнучкішу обробку й обмін даними.

Термін «слабоструктуровані» дані ввів Лунєвський та інші (Luniewski) 1993 року в системі Rufus [775]. 1995 року Папаконстантину та інші (Papaconstantinou) визначив модель для слабо структурованих даних OEM у рамках системи інтеграції гетерогенних баз даних TSIMMIS [776, 777]. 1996 року Бунеман та інші (Buneman) визначив модель слабоструктурованих даних [778]. 1999 року Дойч та інші (Deutch) описав зв′язок між слабо структурованими даними й XML [780].

Мови запитів. Для слабоструктурованих даних були визначені мови запитів, які дозволяють добувати дані з цієї структури або перетворювати одну слабо структуровану структуру в іншу. Ці мови з′явилися практично одночасно із самою структурою. 1995 року була розроблена система і мова Lorel [781] – одна з перших мов запитів для слабоструктурованих даних, де було введено поняття регулярного вираження шляху для навігації шляхами з частково відомою структурою.В мові UnQL [779] увага акцентується на трансформаціях запитів і вводиться структурна ракурсія як центральна парадигма перетворення слабоструктурованих даних. Мова XML – QL [780] стала першою мовою, де принципи мов слабо структурованих даних були застосовані до XML.

З точки зору внутрішньої структури представлення документна структура є різновидом структури NoSQL ключ – значення, коли значення, в свою чергу, може бути парою «ключ – значення» тощо, представляючи таким чином багаторівневу ієрархію. Для форматування документних даних використовуються стандартні мови JSON, BSON, XML, YAML та інші.

Основні операції практично всіх ДОБД позначаються абревіатурою CRUD і означають Create, Retrieve, Update, Delete.

Припускається, що в будь-якій ДОБД існує механізм задавання унікальних ідентифікаторів документів, за якими відбувається індексація, що суттєво прискорює пошук документів.

Системи ДОБД. Була розроблена чимала кількість систем ДОБД. Так на сайті [783] наведено короткий опис понад 60 систем ДОБД, серед яких на 2022 рік кращими вважаються: Amazon Dynamo DB, Mongo DB, Mongo DB Atlas, Couchbase Server, Google Cloud Firestore, Percona Server for Mongo DB, Inter System IRIS, Asango DB, Database Management, Azure Cosmos DB.

Колончаті бази даних.

Колончата NoSQL база даних (КБД) – це така база даних, де дані зберігаються згуртованими по колонках таблиці, а не рядками, як у реляційних базах даних. У ній «сусідами» є не дані з двох стовпчиків того самого рядка, а дані одного і того ж стовпчика, але з різних рядків.

Зазначимо, що термін «колончата БД» має два значення. (1) *Колончато – орієнтована* – це БД, не обов′язково NoSQL, яка зберігає дані таблиці не рядками, а стовпчиками. Такі системи зазвичай використовуються в аналітичних інструментальних засобах, зокрема, HPE Vertica. (2) *Сімейство колонок* (column family) або *широка колонка* (wide-column) представляють тип NoSQL баз даних, які підтримують таблиці, що мають різну кількість, імена й формати/типи колонок у різних рядках таблиці. В даному розділі мова піде про колончаті БД другого типу.

Колончаті сімейства можуть складатися практично з необмеженої кількості колонок, які можуть створюватися динамічно. Читання й записування відбуваються з використанням колонок, а не рядків.

Колонка може бути представлена у вигляді великої кількості пар ключ – значення, де ключ - ім'я колонки. Таким чином успадковуються властивості сховищ типу ключ – значення.

У деяких випадках розрізняють колончаті сховища й сховища сімейства колонок. У першому випадку мається на увазі, що кожна колонка зберігається самостійно, не залежно від інших колонок, а в другому – всі колонки сімейства запам'ятовуються разом.

Супер – колонка – це колонка, що складається з інших колонок. До прикладу, супер – колонкою є ПІБ, яка складається з колонок Прізвище, Ім'я та по-Батькові. З іншого боку, Прізвище, Ім'я та по-Батькові – це сімейство колонок. Отож, сімейство супер – колонок – це сімейство, складене з сімейств колонок. Така вкладеність може бути багаторазовою. З точки зору концепції сховищ ключ – значення ми маємо ситуацію, коли значення в свою чергу є парою ключ – значення. З іншого боку, користуючись термінологією реляційних баз даних, можна сказати, що сімейство супер – колонок – це в певному сенсі аналог поняття «погляд» (view).

Історія КДБ

Етап 1. Транспоновані файли (1969 – 1985). Вважається, що історія колончатих баз даних починається з кінця 60-х років із появою так званих транспонованих файлів (transposed files), в яких рядки табличних даних переводяться у стовпчики, а стовпчики – у рядки. Першою підтримуючою транспоновані файли СУБД вважається TAXIR (1969) [784], орієнтована на зберігання й пошук біологічних даних. До цього класу належить також розроблена 1975 року медична система TOD [785] і система RAPID [786], створена 1976 року компанією Statistics Canada для пошуку й обробки статистичних даних. Згодом вона використовувалась у багатьох статистичних організаціях до кінця 90-х років. Варто також згадати про створену 1977 року SCSS – колончатий варіант системи SPSS (Statistical Package for the Social Sciences) – статистичний пакет для соціальних наук [787]. Однією з найбільш ранніх систем, що мала риси сучасних колончатих СУБД, вважається Cantor [788]. Були здійснені дослідження з організації пошуку в транспонованих файлах [789]. 1975 року в праці [790] досліджено питання декомпозиції записів на підзаписи з наступним їх зберіганням в окремих файлах.

Етап 2. Модель декомпозованої пам'яті – DSM (1985 – 2000). Наступний етап пов'язаний із появою методів вертикального розбиття, що передбачає поатрибутну кластеризацію таблиць. До цього в базах даних панувала так звана модель N-арної пам'яті NSM (N-ary Storage Model). Але 1985 року була опублікована стаття [791], де як альтернатива NSM була запропонована модель декомпозованої пам'яті (Decomposition Storage Model, DSM). У DSM кожна колонка таблиці запам'ятовується окремо, а щоб знати, якому саме рядку таблиці належить значення в колонці, разом із цим значенням запам'ятовується унікальний ідентифікатор рядка. Здебільшого це сурогатний первинний ключ (дивись рис. нижче).

ID	Товар	Ціна	Дата
1	ПК	500	07.10.21
2	Монітор	150	07.10.21
3	Мишка	15	09.10.21
4	Прінтер	170	11.11.21

а) Модель пам'яті /NSM

	Товар
1	ПК
2	Монітор
3	Мишка
4	Прінтер

	Ціна
1	500
2	150
3	15
4	170

	Дата
1	07.10.21
2	07.10.21
3	09.10.21
4	11.11.21

б) Модель пам'яті DSM

Протягом наступних 20 років використовувались саме терміни NSM/DSM для зазначення рядковості й по колонкової моделей пам′яті. Наступні дві статті, присвячені дослідженню проблеми розпаралелювання операцій роботи з DSM [792], а також використанню індексів для виконання операцій з′єднання й проєкції [793], продемонстрували безперечну перевагу DSM перед NSM під час виконання операцій вибірки даних із БД.

Етап 3. Бурхливий розвиток (2000+). До початку 2000-х років концепція DSM не була затребувана, бо були відсутні класи задач, котрі гостро потребували КБД. І лише з появою статистичних і аналітичних баз даних, сховищ даних, технологій OLAP і великих даних, КБД стали доволі затребуваними. Першими дослідницькими прототипами КБД, реалізованими в 2000 – 2005 рр., які значно вплинули на подальший розвиток комерційних КБД, були MonetDB [794, 795], VectorWise (MonetDB/X 100) [796] и C-Store [798].

Першими комерційними КБД вважаються Sybase IQ (1996) і KDB (1998). Друга половина 2000-х років відзначилася бурхливим ростом кількості колончатих СУБД. У цей час були реалізовані Vertica (Vertica Analytic Database),, Exasol, ParAccel, Kognito, InfoBright, SAND, Ingres VectorWise, Kickfire, Paraccel. Прикладами колончатих баз даних є також Apache Cassandra, Scylla, Apache HBase, Google BigTable, Microsoft Azure Cosmos DB. Колончато – орієнтовані засоби впроваджені в такі реляційні СУБД, як Oracle, SQLServer, PostgreSQL, IBM BLU.

На початку 2000-х років виникла ідея гібридних сховищ, що підтримують як рядкові, так і колончаті зберігання даних. Так 2001 року було розроблено сховище PAX (Partition Attributes Across), 2002-го запропоновано гібридну модель “Fractured Mirrors”. Пізніше були реалізовані комерційні гібридні сховища SAP HANA, Infini DB, Greenplum.

На момент написання статті в базі даних https://dbdb.io/ (Database of Databases) була представлена 51 СУБД, що мала відношення до колончатої моделі даних.

Характерні риси і сфера застосування. Характерними рисами КБД є:

- висока швидкість виконання операцій пошуку/доступу, особливо під час виконання запитів з агрегатними функціями;
- висока горизонтальна масштабність завдяки повній свободі у розподілення колонок між вузлами мережі;
- висока ефективність декомпозиції й стискання даних;
- можливість функціонування у безсхемному варіанті.

Колончаті СУБД знаходять застосування переважно в аналітичних схемах класу business intelligence, аналітичних OLAP – сховищах даних (data warehouses) і системах класу Big Data.

Методи реалізації й оптимізація. Хоча сучасні комерційні КБД широко використовують принципи й методи, запропоновані для транспонованих файлів і декомпозованих структур, однак вони мають можливості, які на початкових етапах не були передбачені. Особливо це стосується питань, пов′язаних із аспектами оптимізації і підвищення продуктивності їх функціонування. Наведемо їх.

Віртуальний ключ. Якщо значення колонки є фіксованої довжини, то можна не зберігати сурогатний ключ разом із кожним значенням колонки, а обчислювати як значення ключа, так і розміщення відповідного значення колонки на основі зміщення. Такий принцип був використаний у Monet DB [795]. Ним можна користуватися у разі відсутності сортування в колонках.

Блокова організація й векторна обробка. Для КБД були запропоновані методи блокової організації даних та їх векторної обробки [796, 800], які суттєво підвищують їхню продуктивність. Блокова ітерація [797] передбачає, що велика кількість значень колонки передається у вигляді одного блоку від одного оператора до наступного.

Пізня матеріалізація (late materialization). Пізня матеріалізація

або пізнє відтворення кортежа, потрібного для запиту, має на увазі максимально можливе відтермінування виконання операцій з′єднання колонок [801, 802]. Пізня матеріалізація значно підвищує ефективність використання пропускної здатності пам′яті.

Стискання колонок. Стискання колонки з використанням найефективнішого для нього методу дає суттєве зменшення розмірів файлів колонок [800, 803]. Через те, що колонки містять дані одного типу (атрибуту), досягаються хороші показники стискання за допомогою простих алгоритмів. Було запропоновано багато алгоритмів стискання для колончатих сховищ [804].

Оперування стиснутими даними. У багатьох сучасних колончатих сховищах розпакування даних відтермінovується, допоки це не буде абсолютно необхідним [800, 801]. В ідеальному варіанті – поки не з′явиться необхідність представити результати користувачеві. В зв′язку з цим вирішується задача оперування стиснутими даними. Пізня матеріалізація дозволяє колонкам залишатися стиснутими доти, доки не з′явиться потреба формувати з них кортежі (виробляти їх з′єднання).

Ефективна реалізація операції з′єднання. Оскільки КБД передбачають колончате представлення даних, важливим є питання використання різних стратегій виконання операції з′єднання. В КБД використовуються як класичні алгоритми, так і специфічні [800, 805, 806].

Надлишкове представлення окремих стовпців із різним сортуванням. По колонках, відсортованих відносно конкретного атрибуту, можна здійснювати швидкий пошук. Збереження кількох копій конкретної колонки, відсортованих за різними атрибутами, може суттєво підвищити продуктивність виконання запитів. До прикладу, система C – Store [798] фізично зберігає колекції колонок, кожна з яких відсортована за певним атрибутом. Групи колонок, відсортовані за певним атрибутом, називаються проєкціями. Одна й та ж колонка може знаходитися в різних проєкціях. Наявність різних сортувань сприяє оптимізації функціонування системи.

Крекінг і адаптивне індексування баз даних. Крекінг (Cracking) – це принципово новий підхід у базах даних, який базується на принципі, що створення й ведення індексу є продуктом діяльності з обробки запитів, а не створення й оновлення бази даних. Кожен запит інтерпретується не лише як запит на отримання частки бази даних, а й як вказівка на необхідність «відколювання» від неї невеликих шматочків, описуваних цим запитом. Із цих шматочків практично будується крек – індекс, аби збільшити швидкість подальшого пошуку. Крек – індекс створюється й підтримується динамічно залежно від обробки запитів і адаптується з урахуванням зміни робочих навантажень запитів. Щодо КБД загальна схема функціонування механізму крекінгу наступна [809]. Щоразу, як запит уперше формулюється щодо атрибуту А, механізм крекінгу створює копію колонки атрибуту А, яка називається крекінг – колонкою А. В процесі виконання подальших запитів розпізнаються ті, які звертаються до атрибуту А, й відбувається подальше настроювання крекінг – колонки А та її індексу. Більше того, крекінг – колонка А використовується для збільшення швидкості подальшого пошуку за атрибутом А. Monet DB була однією з перших КБД, яка підтримувала механізм крекінгу. Детальніше ознайомитися з механізмами крекінгу й адаптивного індексування можна в монографії [804].

Ефективне завантаження й оновлення. У зв′язку з тим, що в КБД дані декомпозовані по колонках і активно використовується механізм стискання даних, завантаження й оновлення БД відбувається значно повільніше, ніж у рядкових БД. Тому були досліджені питання оптимізації виконання цих функцій [798, 810]. Так, зокрема, в S-Store [798] дані спочатку записуються в оптимізований для запису буфер, що нести скається. А потім періодично «скидаються» у великі пакети, які стискаються.

Циклотрон даних. Одним із головних завдань розподіленої обробки запитів є розробка самоорганізованої архітектури, здатної оптимально використовувати всі апаратні ресурси для оперативного управління базою даних. Це необхідно для мінімізації часу відгуку на запити й максимізації пропускної здатності за відсутності єдиної глобальної точки координації. Було запропоновано технологію *Циклотрона даних* (Data Cyclotron) [804a], яка вирішує цю проблему використанням турбулентного переміщення даних через кільце зберігання, створене з розподіленої оперативної пам′яті з використанням функціональних можливостей, які пропонуються сучасними мережевими засобами віддаленого прямого доступу до пам′яті (Remote Direct Memory Access – RDMA). Запити, ініційовані в окремих вузлах мережі, постійно взаємодіють із кільцем зберігання, збираючи фрагменти даних, які постійно циркулюють у кільці.

Здійснено порівняльні дослідження рядкових і колончатих сховищ [806 - 808]. Зокрема, результати досліджень [806] показали, що оптимізований варіант колончатого сховища працює вп’ятеро швидше за комерційні рядкові сховища.

За адресою https://www.predictiveanalyticstoday.com/top-wide-columnar-store-databases/ можна ознайомитися із такими 9 найпопулярнішими КБД 2021 року згідно із PAT Research: MariaDB, CrateDB, ClickHouse, Greenplum Database, Apache Hbase, Apache Kudu, Apache Parquet, Hypertable, MonetDB.

Так само відповідно до сайту https://www.g2.com/ наступні 7 КБД були найпопулярнішими 2021 року: Amazon Redshift, Snowflake, ClickHouse, Druid, Hbase, Apache Kudu, Apache Parquet (https://www.g2.com/categories/columnar-databases).

Для глибшого ознайомлення із КБД рекомендуємо монографію [804], а також статті [811 - 813]. Чудовим навчальним матеріалом є посібник [814].

Завершимо виклад колончатих СУБД таким висловом, узятим із [804]: «Сучасні колончаті сховища вийшли за межі простого поколончатого зберігання даних, вони пропонують цілком нову архітектуру баз даних і механізми роботи з ними, адаптовані для сучасних технічних засобів і методів аналітичної обробки даних».

Графові бази даних.

Графова база даних (ГБД) – це БД, у якої модель даних має графову структуру певного виду, включно зі схемою та її екземплярами, а маніпулювання даними здійснюється за допомогою мов, що мають граф – орієнтовані оператори.

Взаємозв′язок між графовою структурою і базами даних відслідковується від моменту виникнення баз даних у 60-і роки минулого століття. Структура даних ієрархічних і мережевих баз даних, а також ER-мови, нагадують графову структуру, однак вони не є ГБД.

ГБД виявляються доволі корисними, або й незамінними у випадку, коли інформація про взаємозв′язок даних є важливішою або настільки ж важливою, як і власне дані. В такому разі дані й взаємозв′язки між ними перебувають на одному логічному рівні.

Згідно з [823a] виділяються дві категорії ГБД: *транзакційні* й *нетранзакційні* ГБД. Перші мають відношення до великої сукупності невеликих графів, як-от лінгвістичні дерева. Характерними операціями для них є пошук суперграфів, підграфів, подібних графів. До другої категорії належать спільні великі ГБД, наприклад, соціальні мережі, які можуть складатися з кількох компонент. Характерні операції – пошук (найкоротшого) шляху, пошук сусідів, пошук компонент із заданими властивостями.

Можна виділити два етапи досліджень і розробок в галузі ГБД. На першому етапі (80-і – початок 90-х років) активно проводилися дослідження у сфері моделей даних і мов запитів ГБД. Пізніше інтерес до них суттєво знизився через появу геопросторових, темпоральних, слабоструктурованих і XML – баз

даних. Однак на початку цього століття ці дослідження і розробки поновилися в зв′язку з появою семантичного ВЕБу, зв′язаних даних і соціальних мереж. Цей тип характеризується передусім тим, що крім досліджень було розроблено багато промислових ГБД.

Графові моделі даних. У ГБД не існує канонічної моделі даних, як-от, скажімо, в реляційній моделі. Це пояснюється гнучкістю графової структури, яка дозволяє нарощувати її складність, а також визначає різні мови маніпулювання і запитів. З огляду на це було запропоновано чимало моделей і мов запитів ГБД. Усе ж виділяють такі типи моделей графових баз даних [825].

Базова графова модель – мічені графи. Складається з вершин (вузлів) і направлених дуг (ребер), котрі з′єднують вершини. Вершини представляють концепти (об′єкти), а дуги - зв′язки між цими концептами. Графова структура застосовується для завдання як схеми БД, так і її екземплярів. Вершини/дуги мають унікальні ідентифікатори й нуль або більше міток (labels) [835, 839, 840, 848]. Зазвичай міткам даються імена відповідних концептів, а дугам – імена відповідних зв′язків. Мітки дають змогу вказати класи, до яких належать вершини/дуги. Складніший варіант базової структури – мультограф, у якому пара вершин може бути зв′язана кількома дугами.

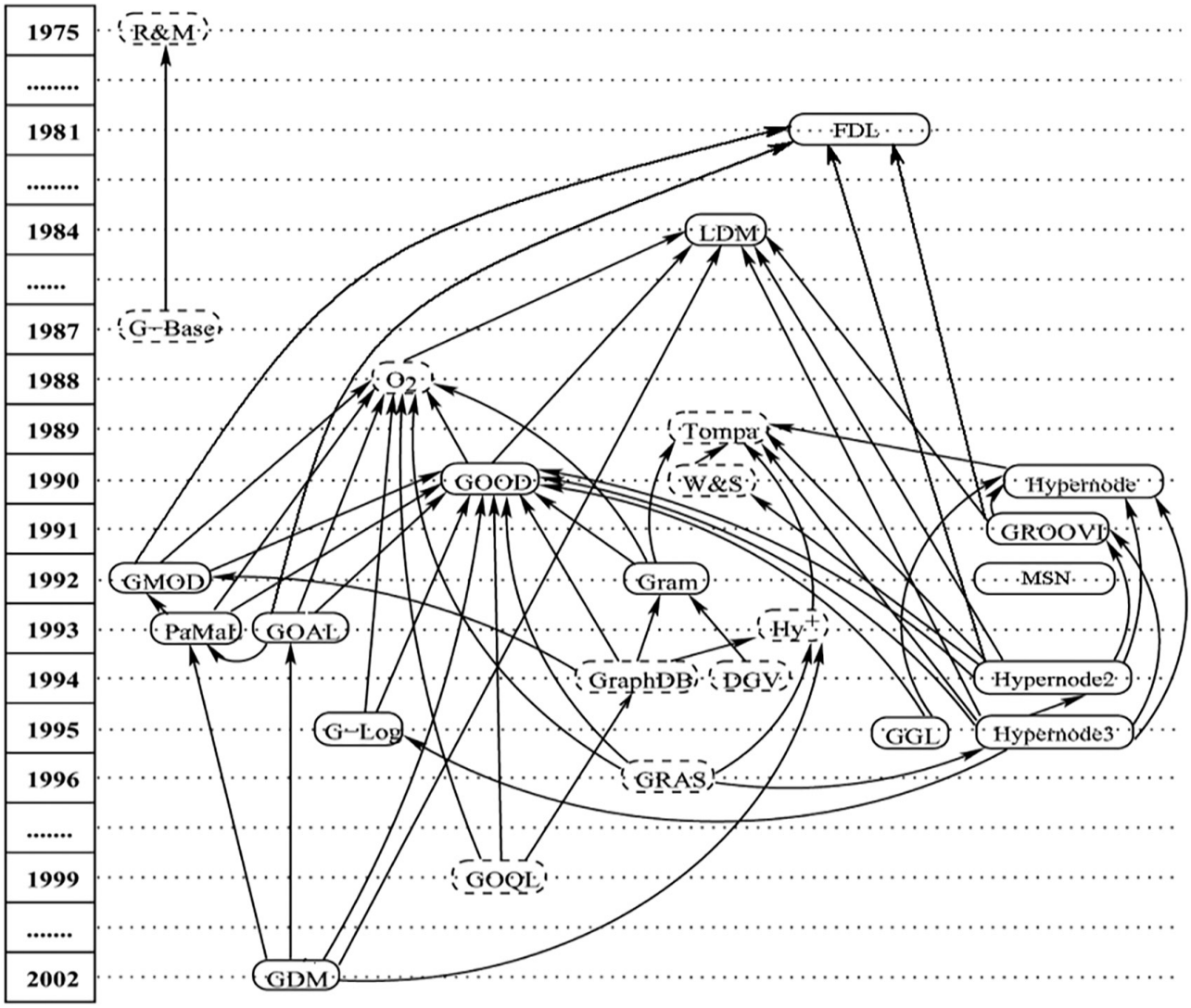
Графова модель із властивостями (Property Graph Model). Граф із властивостями – це спрямований мічений атрибутивний мультиграф. Поняття графу з властивостями введене в [827], а його формальне визначення дано в [824]. Вершини/дуги можуть мати багато властивостей (атрибутів), які дозволяють вказати їхні характеристики. Властивості задаються у вигляді пар ключ – значення. Мовами графових моделей із властивостями є G-CORE [823], PROPER [854], Gremlin [885], Cypher [887], PGQL [891].

Гіперграфова модель – складні дуги. Гіперграф (hipergraph) – це узагальнення графової моделі даних, в якій дуги можуть з′єднувати будь-яку кількість вершин, як на початку дуги, так і в її кінці [828]. Такі дуги називаються гіпердугами (hiperedge). Гіперграфи виявляються корисними, коли дані містять зв′язки типу «багато до багатьох». Гіперграфові моделі представлені в працях [834, 836, 838, 844].

Модель гіпервершин – вкладені графи. Модель гіпервершин (hipernodes) – це спрямований граф, у якому вершини самі можуть бути графами. Такі вершини називаються гіпервершинами, утворюючи структуру вкладених графів (nested graphs). Модель дозволяє наочно і природно представляти об′єкти довільної складності. Модель гіпервершин уперше була визначена 1990 року в [837], а згодом уточнена цими ж авторами в [847]. Даній моделі присвячені також статті [838, 849, 850]. Аналогічна ідея запропонована в мультимасштабних мережах (multi – scaled networks) [841].

Граф даних веба – модель RDF. RDF – це мова представлення взаємопов′язаних ресурсів у вебі з використанням графічної структури даних триплетів, у якій використовуються спрямовані дуги й мічені вершини та дуги. Використовувана в RDF графічна структура є найбільш загальною в тому сенсі, що в ній дуги є також вершинами. Це дозволяє підтримувати принцип самоописуваності (реїфікації), тобто формувати твердження відносно тверджень. У RDF – графі одночасно присутні схема і її екземпляри, для відокремлення екземплярів від схем використовуються мічені спеціальним іменем дуги (ім′я – type). Модель RDF власну мову запитів SPARQL. До баз даних, що підтримують модель даних RDF, належать: 4Store, AllegroGraph, BigData, Jena TDB, Sesame, Stardog, OWLIM, uRiK.

У праці [819] подається детальний огляд графових моделей даних до 2003 року. На рисунку нижче, взятому з [819], представлені найвідоміші графові моделі баз даних, впорядковані за роками їх публікації. Тут овали представляють моделі, стрілки вказують на цитування, а пунктирні овали свідчать про те, що відповідні роботи мають відношення до графової моделі БД.



Далі наводиться короткий опис кожної з моделей із зазначенням статей їх публікації.

- R&M [829] – введено поняття семантичної мережі для зберігання інформації про дані бази даних.
- FDL [830] – у мережах функціональної моделі даних неявно визначена графова структура даних, метою якої було забезпечення «концептуального природного» інтерфейса бази даних.
- LDM [831] – у межах логічної моделі даних явно визначена графова модель бази даних, мета якої – узагальнення реляційної, ієрархічної й мережевої моделей даних.
- G-Base[832] – запропонована графова модель даних, названа G-Base, для представлення складних структур.
- O2 [833] – визначена об′єктно – орієнтована модель даних O2 на основі графової структури.
- Tompa [834] – модель даних для гіпертекстових баз даних.
- GOOD [835] – граф – орієнтована об′єктна модель, призначена для систем із графо подібними засобами представлення і маніпулювання даними.
- W&S [836] – гіперграфова модель для доступу до даних у базі даних.
- Hypernode [837] – граф – орієнтована модель із гіпервершинами, що являють собою вкладені графи.
- GROOVY [838] - об′єктно – орієнтована гіперграфова модель даних.
- GMOD [839] – висунуто ряд концептуальних пропозицій щодо інтерфейсів користувача в граф – орієнтованій базі даних.

- Gram [840] – граф – орієнтована модель для представлення гіпертекстів.
- MSN [841] – графова модель мультимасштабних мереж, яка об′єднує основи теорії графів і об′єктно – орієнтовану парадигму.
- PaMal [842] – є розширенням GOOD з явним представленням картежів і множин.
- GOAL [843] – граф – орієнтована об′єктна модель із асоціативними вершинами.
- Hy+ [844] – гіперграфова модель з мовами запитів і візуалізації.
- Graph DB [845] – графова модель даних і мова запитів для баз даних.
- DGV [846] – графова модель для визначення і маніпулювання графами різного виду, які зберігаються в реляційних або об′єктних базах даних.
- Hypernode2 [847] – модель із вкладеними графами, яка є розвитком Hypernode.
- G-Log [848] – граф – орієнтована модель і декларативна мова запитів.
- GGL [849] – теоретико-графова модель даних баз даних карт генома.
- Hypernode3 [850] – графова модель даних, яка є розвитком Hypernode.
- GRAS [851] – графова атрибутивна модель для представлення складної інформації.
- GOQL [852] - об′єктно-орієнтована графова модель даних і графова мова запитів.
- GDM [853] – граф-орієнтована модель із n-арними семитричними зв′язками.

Графові мови запитів (ГМЗ).

Модель ГБД, окрім структури і обмеженої цілісності, має високорівневу графову мову запитів (ГМЗ), в якій можна формулювати специфічні для графів операції. В [819] подається огляд ГМЗ періоду першої хвилі досліджень і розробок у царині ГБД. На підставі цієї праці в статті [855] було досліджено питання виразної потужності й обчислювальної складності деяких із цих мов. У статті [821] подається порівняльний аналіз мов, функціонуючих на той час ГБД. У статті [856] досліджується проблема формулювання запитів до ГБД і, зокрема, виразність і складність навігаційних мов запитів. І, нарешті, в праці [822] подано огляд основоположних особливостей сучасних ГМЗ, а в [825] наводиться аналітичний огляд ГМЗ станом на 2018 рік.

Як уже згадувалося, ГМЗ мають специфічні для графів операції. Коротко опишемо їх.

Суміжність (adjacency) і окіл (neighborhood). Дві вершини суміжні, якщо вони з′єднуються дугою, дві дуги суміжні, якщо вони мають спільну вершину. В статті [857] досліджується питання ефективності виконання операції суміжності у великих динамічних розряджених графах. Більш загальним поняттям є «окіл». N – околом заданої вершини є множинність вершин, які є досяжними із заданої вершини за допомогою шляху, що охоплює не більше n – дуг. Дослідженню проблеми суміжності/околу в базах даних присвячена монографія [858].

Співставлення зі зразками (pattern matching). Знаходження множини під графів заданого графа бази даних, які відповідають заданому графу – зразку. Завдання пошуку за зразком характерне для багатьох класів прикладних завдань. Скажімо, розпізнавання образів, ідентифікація співтовариств у соціальних мережах. Проблема співставлення зі зразком досліджується в теорії баз даних [861, 862], біоінформатиці [863], семантичному вебі [864]. Як показано в [859, 860] проблема співставлення зі зразком має відношення до проблеми інтелектуального аналізу графів даних (data graph mining).

*Досяжність / зв'язність (reachability / connectivity).*Проблема досяжності полягає у встановленні, чи існує шлях, що веде від однієї вершини до другої. В цьому контексті розрізняють два види шляхів: шляхи фіксованої довжини

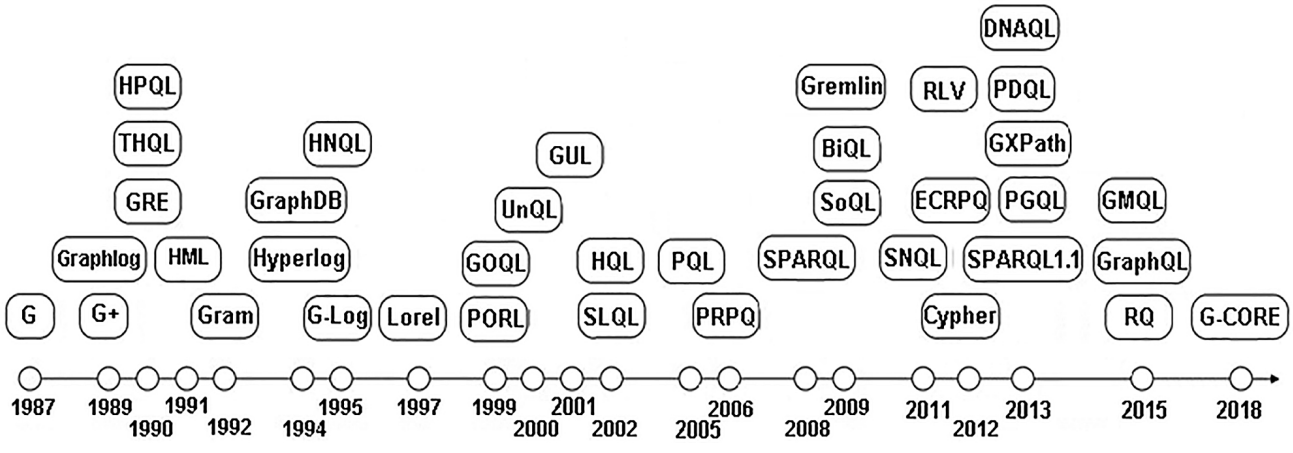
(fixed length paths), які містять фіксовану кількість вершин і дуг, а також регулярні прості шляхи (regular simple paths), в яких накладаються обмеження (регулярні вирази) на вершини / дуги. В статті [861] подано огляд різних теоретико – графових задач, пов′язаних зі шляхами, які мають відношення до баз даних разом із обчисленням транзитивного замикання, виконанням рекурсивних запитів і складного пошуку шляхів включно. В [865] вводиться поняття запиту з регулярним шляхом (regular path query – RPQ) як способу вираження запитів на досяжність, а в [866] аналізується проблема досяжності з урахуванням наявності регулярних виразів. Цей тип запитів досліджується також у працях [856, 867]. Одним із різновидів задачі досяжності є знаходження найкоротшого шляху, якщо їх, скажімо, кілька [868].

Аналітичні запити. Агрегування. Запити цього виду не працюють зі структурного графу, а надають кількісну інформацію (здебільшого в агрегатному вигляді) щодо топологічних властивостей графа бази даних. Аналітичні запити зазвичай містять спеціальні агрегатні оператори типу count, sum, min, max, average, які підсумовують результати запиту у вигляді кількості вершин, кількості сусідів, довжини шляху, відстані між вершинами, найкоротшого шляху між вершинами тощо. Як показано в монографії [859], складні аналітичні запити тісно пов′язані з алгоритмами інтелектуального аналізу графа даних.

Приблизне співставлення і ранжування (approximate matching and ranking). Можливі ситуації, коли користувачі не знають структуру графа, щодо якого формулюються запити, а також, існуючі в ньому обмеження і правила. Як наслідок, вони можуть формулювати запити, котрі не даватимуть результати, або ж відповіді не відповідатимуть очікуваним результатам. У такому разі бажано мати можливості отримання неточних результатів і їх ранжування відповідно до встановлених критеріїв. Однією з перших праць, щодо формулювання «гнучких» запитів до слабо структурованих текстів, була [869], де досліджується питання неточного відпрацювання запиту на основі спеціального виду відповідності між запитом і графом. У праці [870] дається більш узагальнене поняття приблизного співставлення шляхів графа, коли результати пошуку можуть бути ранжовані згідно із їхньою «близькістю» до вихідного запиту. На рисунку нижче, який є дещо зміненим і доповненим варіантом із праці [825], наведені в хронологічному порядку так звані «чисті» ГМЗ, тобто ті, що призначені для роботи з графічними моделями даних.

Ці мови описуються в наступних статтях:

G [865], G+ [871], Graphlog [872], HPQL [837], THQL [836], GRE [873], HML [838] Gram [840], Hyperlog [847], GraphDB [845], G−Log [848], HNQL [850], Lorel [874], PORL [875], GOQL [852], UnQL [876], GUL [877], SLQL [878], HQL [879], PQL [880], PRPQ [881], SPARQL [882], SoQL [883], BiQL [884], Gremlin [885], SNQL [886] Cypher [887], ECRPQ [888], RLV [889], SPARQL 1.1



[890], PGQL [891], GXPath [892], PDQL [826], DNAQL [893], RQ [894]. GraphQL [895], GMQL [896], G-CORE [823].

Для ГБД є принциповим ефективне виконання запитів [897, 898]. З цією метою було розроблено різні методи їх індексування [899 - 901]й оптимізації [902 - 904].

Графові бази даних (ГБД). За останні 20 років було реалізовано понад 60 графових баз даних. Їх списки присутні на багатьох сайтах.

https://hostingdata.co.uk/nosql-database/, на цьому сайті, присвяченому NoSQL – базам даних, є розділ зі списком графових баз даних з адресами в Інтернеті, за якими можна з ними ознайомитися.

На сайті https://dbdb.io/, який являє собою «базу даних баз даних» і містить інформацію про понад 760 СУБД, наведений перелік понад 40 систем БД, що базуються на графовій моделі даних (https://dbdb.io/browse?data-model=graph) із зазначенням основних відомостей і адрес веб – сайтів, за якими можна з ними ознайомитися.

https://sourceforge.net/software/graph-databases/?page=1 за цією адресою наводяться відомості щодо 50 графових систем баз даних, адреси веб – сайтів, на яких можна з ними ознайомитися.

Відзначимо, що в усіх трьох джерелах дати реалізації систем наведені після 2000 року.

У списку нижче поданий узагальнений перелік ГБД у хронологічному вигляді.

Існує велика кількість статей із порівняльним аналізом різних ГБД, як-от: [817, 821, 905 - 907].

Нижче наведено таблицю порівняльного аналізу 20 ГБД взяту з [817].

На закінчення підкреслимо великий внесок у розвиток теорії й методології графових баз даних доктора кафедри комп'ютерних наук Університету Талька (Чилі) Ренцо Англеса (Renzo Angles). Даний розділ написано здебільшого за матеріалами його праць.

Ренцо Англес

2002	InfinityDB
2005	AllegroGraph
2006	Blazegraph, Sparksee
2007	DEX, Neo4J, HyperGraphDB, AnzoGraph, sones GraphDB
2008	InfoGrid
2009	VertexDB, Pregel, SylvaDB, IBM System G
2010	HyperGraphDB, InfiniteGraph, Sones, OrientDB, FlockDB, Filament, G-Store, Redis_graph, Horton, CloudGraph, Stig
2011	ArangoDB, Trinity, OrigoDB, ArangoDB, Fallen-8
2012	GraphChi-DB, GrapheneDB, SparkleDB, Sqrrl, TigerGraph, FaunaDB, JanusGraph
2013	Bitsy, imGraph, AgensGraph, Galaxybase
2014	Cayley, GraphDB, GrapheekDB, GUN
2015	Cosmos DB, DegDB, DGraph
2016	IndraDB, EliasDB, Memgraph, VertexDB, TypeDB
2017	JanusGraph, Amazon Neptune, Fluree
2018	AnzoGraph DB, Nebula Graph, Neptune
2019	TerminusDB

NewSQL – бази даних

2007 року Майкл Стоунбрейкер, розробник систем баз даних Ingres і Postgres, майбутній лауреат премії Тьюрінга, очолив дослідницьку групу, яка опублікувала основоположну статтю [908], де підкреслювалося, що апаратні припущення в основі реляційної архітектури більше не придатні. Стоунбрейкер і його команда запропонували ряд варіантів перспективних проєктних рішень щодо СУБД, два з яких стали особливо важливими для подальшого розвитку напрямку, який згодом дістав назву NewSQL. Це H-Store [909] – розподілена база даних, яка повністю міститься в пам′яті, та C-Store [910] – колончата база даних. 2010 року Рік Кеттелл (Rick Cattell) опублікував статтю [911], де використав термін «масштабований SQL», і проаналізував такі відомі на той час масштабовані реляційні бази даних, як MySQL, Cluster, VoltDB, Clustrix, ScaleDB, ScaleBase, NimbusDB, а також порівняв підходи масштабованої SQL і NoSQL.

Термін NewSQL запропонований 2011 року аналітиком 451 Group Метью Аслетом (Matthew Aslett) [912, 913]. Відтоді він застосовується на означення масштабованих реляційних систем управління базами даних нового покоління з оперативною обробкою транзакцій (OLTP), які мають здатність горизонтальної масштабованості NoSQL і підтримки ACID, характерної для традиційних (SQL) систем баз даних.

Метью Аслет

Пропоновано також варіанти класифікації NewSQL – баз даних [915, 916].

Пізніше були здійснені дослідження, в результаті яких з′явилося чимало публікацій щодо порівняльного аналізу технологій SQL, NoSQL та NewSQL. Фундаментальною працею в цьому напрямку стала монографія Гая Харрісона (Guy Harrison) [12].

На сьогодні NewSQL – технологія знайшла свою нішу на ринку баз даних і широко використовується в промисловості. До цього класу належать наступні системи: MemSQL, VoltDB, Spanner, Calvin, CockroachDB, FaunaDB, YugabyteDB.

Гай Харрісон

Онтологічні бази даних

Із появою семантичного вебу з′явилося поняття онтології. Було висунуто кілька визначень онтологій. Найповніше й загальноприйняте наступне [917]:

Онтологія – це очевидна формальна специфікація погодженої концептуалізації. Детальніше про це визначення в [918].

Сьогодні онтології широко використовуються в різних галузях. Вони стали важливою складовою семантичного вебу. Розроблено інструменти маніпулювання онтологіями, як-от Protege. Однак вони не надають тих можливостей, які дають БД, і, передовсім, постійне зберігання, маніпулювання й формулювання запитів до структури онтології та її даних. У зв′язку з цим виникло по-

няття **онтологічної бази даних (ОнБД).** Це база даних, яка дає можливість зберігати й маніпулювати онтологіями, включно з онтологічною структурою і даними цієї структури. В контексті семантичного вебу було запропоновано декілька підходів створення ОнБД [919 - 922]. З їх оглядом можна ознайомитися в [923]. Найчастіше ОнБД створюються переважно на базі реляційних баз даних. Окрім того, мовою представлення онтологій нижнього рівня обирається RDF.

Пропонується декілька моделей ОнБД, найпопулярніші з яких наведені далі.

Безсхемна модель (schema-oblivious), названа також вертикальною (vertical). В цій моделі онтологія зберігається в єдиній тернарній таблиці у вигляді RDF-триплетів < суб′єкт – предикат - об′єкт>. Ця таблиця містить і структуру онтології, і її дані. Модель представлена в Jena [924, 925], 3store [922], Rstar [926], Virtuoso [927], Oracle [928]. Суттєва перевага – простота підтримки моделі.

Схемна модель (schema-aware) також має назву бінарної (binary). Кожен клас і кожна властивість онтології (RDF/S-схеми) мають власну таблицю [920, 921, 929, 930]. Класи містяться в унарних таблицях, а властивості – в бінарних. Таблиця властивості об′єднує індивіди різних класів, що мають ці властивості. Перевага цієї моделі – підтримка багатозначних властивостей. Модель використовується в системі управління даними SOR IBM [931].

Розширеним варіантом схемної моделі є дуальна (dual) модель, яка може містити не лише схеми класів і властивостей, а й схеми (мета-схеми) структури онтології. Одним із варіантів є мета-схеми IS-A включення одних класів або властивостей в інші, описуючи таким чином таксономічну ієрархію класів і властивостей. У варіанті ISA-схеми задається таксономічна таблиця для класів/властивостей, екземплярами якої є пари класів/властивостей, що перебувають у цьому відношенні.

Крім того, мета-схема може включати завдання області визначення (domaim) і області значення (range) властивостей. Такий підхід застосовується в ОнБД OntoDB [932].

Гібридна модель (hybrid) [929] поєднує властивості двох попередніх. Використовується тернарна таблиця для кожного типу області значення властивості й бінарна таблиця для всіх екземплярів усіх класів.

Горизонтальна модель. Онтологія представляється у вигляді реляційних таблиць, коли властивості класу стають атрибутами таблиці класу. Якщо ж властивість є багатозначною, то вона представляється бінарною таблицею. За умови, що всі властивості багатозначні, то ця модель перетворюється в бінарну. Дана модель використовується в OntoMS [933], OntoDB [932] и Jena2 [934], Існують два різновиди цієї моделі:

- *Таблиця кластеризації за властивостями* (clustered property table): виділяється група властивостей і створюється таблиця із усіма індивідами (екземплярами) онтології, які мають ці властивості не залежно від їх приналежності до класу. Тобто таблиця може містити індивіди різних класів.

- *Таблиця властивість – клас* (property-class table): таблиця має всі індивіди одного класу із заданим набором властивостей. Одна й та ж властивість може бути у різних таблицях.

В обох різновидах за умови існування індивідів, що не потрапляють у жодну із цих таблиць, вони розміщуються у вертикальні таблиці.

Усі ці моделі ОнБД були реалізовані в існуючих RDF-сховищах (RDFSuite, Jena, Sesame, DLDB, RStar, KAON, PARKA, 3Store, Oracle), вичерпний огляд яких подано в [935].

Виведення. В онтологіях існує проблема виведення, коли за таксономічною ієрархією включення класів/властивостей необхідно створити їх транзитивне замикання. В ОнБД пропонується два підходи вирішення цієї задачі: а) або попередньо вираховувати й матеріалізувати їх (під час компіляції) – цей підхід названо MatView і використовується у без схемному підході; б) або вирахову-

вати їх за необхідності (під час виконання). Другий варіант використовується у схемному й гібридному підході.

Мови. Була запропонована велика кількість мов вебу й семантичного вебу, огляд яких наведено в статті [936]. Серед них виділяється клас мов, що працюють із форматом RDF. До них належать: мови сімейства SPARQL (SquishQL, RDQL, SPARQL, TriQL.), мови сімейства RQL (RQL, SeRQL, eRQL), мови з реактивними правилами (Algae, iTQL, WQL), дедуктивні мови запитів (N3QL, R-DEVICE, TRIPLE, Xcerpt). Усі вони так чи інакше можуть бути застосовані для ОнБД, які базуються на RDF. Водночас в [937] описується мова OntoQL, розроблена для ОнБД OntoDB. Крім того, в цій статті визначена алгебра онтологій, на базі якої створена мова OntoQL.

Повнотекстові бази даних

Повнотекстова база даних (ПТБД) – це база даних, що містить повнотекстові документи і уможливлює їх відшукувати. Документи можуть містити звичайний текст, структурований, слабо структурований або неструктурований. Може мати спеціальні описуючі елементи, так звані метадані, а також мати мультимедійні компоненти.

Історично проблемою зберігання текстових документів у комп′ютерах почали перейматися практично одночасно із питанням зберігання числових даних. Перші документальні системи давали можливість редагувати й форматувати тексти й були частиною видавничих систем. Електронний документообіг став невід′ємною частиною автоматизації діловодства, яка почала активно розвиватися в 1970-х і 1980-х роках. Зберігання повнотекстових документів у комп′ютерах спричинило появу гіпертекстової та гіпермедійної технології, що нині є ядром всесвітньої павутини.

Однією з перших систем управління повнотекстовими документами була STAIRS (STorage And Information Retrieval System – система зберігання й пошуку інформації), розроблена в IBM 1969 року для власних внутрішніх потреб. А 1973 року була представлена як комерційний продукт. Однак ПТБД почали широко використовуватися лише на початку 90-х років, коли комп′ютерні технології зберігання зробили їх економічно вигідними й технологічно можливими. Існує два основні класи ПТБД: розширення класичних бібліографічних баз даних до ПТБД і ПТБД на основі Інтернету (на основі пошукових систем чи XML).

Метадані. В ПТБД повнотекстові документи можуть містити описи, такі, як автори, назва, анотації, ключові слова й УДК для наукових статей. Такі описи дістали назву метаданих. Різні типи інформаційних ресурсів можуть мати різні набори метаданих, які дістали назву схем метаданих. До прикладу, розроблені схеми метаданих для опису персон і організацій (vCard і FOAF), бібліографічних ресурсів (MARC, UNIMARC, DC), музичних та історичних цінностей (CDWA), архівів і електронних ресурсів (GILS, EAD) та багато інших. В електронних бібліотеках найбільш використовуваною є схема Дублінського ядра (Dublin Core).

Індексація. За умови невеликої кількості документів незначного розміру під час пошуку здійснюється послідовний огляд усього документа. Але за умови великих розмірів послідовний перегляд неефективний. Тому в цьому випадку попередньо відбувається індексування – побудова списку пошукових термінів із зазначенням місць їхньої зустрічі. В загальному випадку пошуковими термінами стають усі слова тексту документа. Крім тексту власне документа індексації можуть піддаватися метадані. Для забезпечення витонченіших пошукових можливостей процес індексації може включати додаткові можливості, як-от:

- не враховувати незначущі слова такі, як займенники , прийменники, сполучники, вигуки тощо;
- окрім місця знаходження слова запам′ятовувати його порядкове місце в тексті (пошук поблизу слів у фразі);

- разом зі словом запам′ятовувати його контекст, тобто – фрагмент тексту, де воно знаходиться;

- всі однокореневі слова наводяться в індексі лише раз у вигляді одного нормованого слова.

Мови запитів. Мови запитів сучасних пошукових систем в ПТБД дають розвинуті засоби точнішого формулювання того, що саме треба знайти, включають:

- специфікацію простору пошуку (тобто підмножини ПТБД), в якому слід здійснювати пошук;

- використання логічних виразів;

- специфікацію як окремих пошукових термінів, так і фраз;

- специфікацію відстані між словами, на якій вони перебувають в документі;

- використання регулярних виразів;

- специфікацію поточного пошуку, тобто знаходження документів, які містять слова, в певному сенсі близькі до вказаних, у пошуковому виразі.

Всі ці й багато інших можливостей реалізуються як за допомогою розвинутих засобів індексації, згаданих вище, так і за допомогою розробки сучасних механізмів пошуку, огляд яких наводиться далі.

У подальшому ми будемо використовувати наступні терміни, усталені в публікаціях за алгоритмами пошуку:

- **зразок** (sample) – термін або фраза, що шукаються;

- **текст** (text) – документ, в якому здійснюється пошук;

- **співставлення** (matching) – процедура знаходження зразка в тексті.

Існують різні способи класифікації алгоритмів співставлення, наприклад:

- точне/неточне співставлення;

- співставлення зліва направо, справа наліво у вказаному порядку, в довільному порядку;

- з урахуванням або без урахування стоп – слів;

- знаходження в тексті одного чи всіх зразків;

і багато інших прикладів. За основу ми беремо перший варіант, а в межах кожного із цих класів будемо наводити їх різновиди.

Алгоритми точного співставлення

Вони передбачають точну відповідність тексту зразкові. Відомі наступні п′ять варіантів: символьний, з хешируванням, автоматний, біт – паралельний, гібридний.

Символьний підхід. Символьний підхід є класичним підходом співставлення рядків, який передбачає по символьне порівняння зразка з текстом.

Найпростішим варіантом є повний перебір (brute-force). Порівняння зразка починається від самого початку тексту. Порядок посимвольного порівняння не визначається. У разі невдалого порівняння зразок зсувається на одну символьну позицію праворуч, і процес порівняння повторюється. Від моменту його появи було докладено чимало зусиль із розробки ефективніших алгоритмів, основні з яких коротко описані нижче.

Алгоритм Кнута – Морріса – Пратта ***(Knuth–Morris–Pratt – KMP)*** – ефективний алгоритм, здійснюючий пошук підрядка в рядку. Час роботи алгоритму лінійно залежить від об′єму вхідних даних. Спочатку алгоритм був запропонований 1970 року Джеймсом Моррісом (James H. Morris) і Воганом Праттом (Vaughan Pratt) [946] й незалежно від них досліджений Дональдом Кнутом (Donald Knuth) [947]. Зрештою ці троє вчених опублікували спільну статтю 1977 року [948]. Незалежно від них 1971 року радянський вчений Юрій Матіясевич запропонував аналогічний алгоритм під час дослідження завдання співставлення рядків у двійковому алфавіті [949].

Алгоритм Бойєра – Мура ***(Boyer-Moore – BM).*** 1977 року було висунуто пропозицію алгоритму співставлення Бойєра – Мура [950]. Розроблений Робертом Бойєром (Robert S. Boyer) і Джеєм Муром (J Strother Moore), він є алгоритмом загального призначення і став стандартним й еталонним алгоритмом цього класу. Алгоритм передбачає дві фази:

- фаза попередньої обробки. Згідно із певним правилом створюється таблиця, що визначає величини, на які слід робити зсуви зразка праворуч у разі його невдалого порівняння з текстом.

- Фаза порівняння. Зразок порівнюється із текстом справа наліво. В процесі порівняння вираховується значення зсуву за замовчуванням.

Алгоритм ВМ є найефективнішим алгоритмом пошуку для багатьох додатків, зокрема, текстових редакторів. Він добре працює для алфавітів помірного розміру і довгих шаблонів. Проте розміри алфавіту й зразків суттєво впливають на час попередньої обробки. Алгоритм ВМ докладно описано в [952]. Згодом з'явилося багато алгоритмів, які або модифікують ВМ, або об'єднують його з іншими підходами. До модифікуючих варіантів ВМ, поліпшуючих деякі його характеристики, належать алгоритми Хорспула (Horspool) 186], Чжу-Такаока (Zhu-Takaoka) [954], Turbo-BM [955], Апостоліко - Джанкарло (Apostolico and Giancarlo) [957] Сміта (Smith) [958], Райта (Raita) [959], Крочемора [960], Беррі-Равіндрана [961]. Так само гібридні ВМ – підходи передбачають інтеграцію ВМ – алгоритму з іншими методами для підвищення продуктивності. Вони описані в статтях [962 - 968].

Символьний підхід найбільше придатний для додатків, де шукаються великі текстові зразки.

Підхід із хешируванням. Хеш – підхід передбачає попередньо перед порівнянням застосувати хеш – функцію до зразка й порівнюваного рядка, аби порівнювати не символи, а числа для підвищення швидкості. Цей підхід був розроблений 1987 року Майклом Рабіном (Michael O. Rabin) и Річардом Карпом (Richard M. Karp).[969]. Передбачає дві фази:

- попередня обробка – це однократне обчислення хеш – функції зразка й багатократне обчислення хеш – функцій порівняльних підрядків тексту;

- порівняння хеш – функцій зразка й підрядка. У разі їх рівності проводиться додаткове посилювальне порівняння зразка із підрядком через те, що хеш – функція може видавати однакові числа для різних рядків.

Проблема даного підходу полягає в перерахуванні хешу для кожного підрядка. А її вирішення – у використанні для підрахунку наступного хеш – значення поточного хешу. Це досягається шляхом використання так званого кільцевого хешу. Робін і Карп запропонували використовувати поліноміальний хеш – різновид кільцевого.

Хороший опис алгоритму Робіна – Карпа наведений у статті [970]. Було висунуто ще кілька алгоритмів пошуку з хешируванням, описані в [971 - 973].

Оригінальним різновидом хеш – підходу є так званий q-gram – підхід, який передбачає розбиття зразка на частини, обчислення хеш – функцій кожної частини і подальше їх використання у порівнянні зразка із підрядком тексту. За приклад можуть бути алгоритми, описані в [974, 975].

Цей підхід найвдаліше використовується там, де необхідна швидкість співставлення рядків.

Автоматний підхід. Автоматний підхід у вирішенні задачі співставлення рядків передбачає використання концепції автомату. Одним із варіантів є так званий підхід із використанням суфіксного автомату. Він використовує два взаємозв'язані, але різні автомати: детермінований ациклічний кінцевий автомат (deterministic acyclic finite state automaton) [976], який представляє кінцеву множину рядків, і суфіксний автомат, виконуючий функцію суфіксного індексу [977].

Суфіксний автомат – це найменший частковий детермінований кінцевий автомат, який розпізнає набір суфіксів даного рядка. Граф станів суфіксного автомату має назву «орієнтований ациклічний граф слів» (directed acyclic word graph - DAWG). Суфіксний автомат уперше був описаний групою вчених із Денверського і Колорадського університетів 1983 року [978]. Існує три різновиди автоматичного підходу, котрі коротко подані далі.

1) Орієнтований ациклічний граф слів (DAWG). DAWG – це структура даних, яка забезпечує швидкий пошук слів. У DAWG вершина представляє символ. Спеціальна вершина представляє початковий символ. Частина вершин є кінцевими. Вершини мають спрямовані

зв′язки. Переміщення від початкової вершини до інших уздовж зв′язків вирішує задачу співставлення. Були досліджені такі різновиди DAWG-співставлень: зворотній не детермінований (backward non-deterministic) DAWG [979], подвійний прямий (double-forward) DAWG [980], алгоритм співставлення на основі зворотнього прогнозу (Backward oracle matching - BOM) [981] і його різновидів [982, 983].

2) Широке вікно. Широке вікно – це вікно, розмір якого перевищує розмір зразка, через що в результаті переміщення зразка в процесі пошуку вікна перекриваються, а це дає певні переваги. Вперше ідея широких вікон була висунута 2005 року в праці [984]. Ця особливість широких вікон враховується під час використання суфіксного автомату. З часом цей алгоритм був удосконалений у працях [985, 986]. Цей підхід зараховують також до класу біт паралельних алгоритмів.

3) Автоматний підхід із пропуском непотрібних спроб співставлення. Автоматний підхід до співставлення темпоральних (тобто залежних від часу) зразків із можливим пропуском непотрібних спроб співставлення був запропонований у праці [987]. Темпоральний підхід до проблеми співставлення характерний для систем реального часу і веб – додатків.

Біт – паралельний підхід. Побітові операції над комп′ютерними словами типу NOT, OR, AND, XOR мають властивий їм паралелізм. Вважається [988, 993], що бітовий алгоритм точного пошуку винайдений угорським ученим Балінтом Дьомьолкі (Domolki) [989? 990] 1964 року й розширений Шьямасундаром (Shyamasundar) [991] 1977 року перш, ніж він був заново винайдений Рікардо Баеза-Йейтсом и Гастоном Гонне (Ricardo Baeza-Yates and Gaston Gonnet) 1992 року [992] і названий Shift OR. Цей алгоритм з часом був розширений і удосконалений [994, 998].

Після Shift OR 1998 року було запропоновано біт – паралельний алгоритм BNDM (Backward Non Deterministic Matching) [999, 1000], що належить до класу Shift AND алгоритмів. Тобто використовуються операції Shift і AND для паралельного пошуку. Згодом були висунуті удосконалені варіанти BNDM: TNDM (Two way Non Deterministic Matching) [1001], SBNDM (Simplified BNDM) [1001], BNDMq и SBNDMq (q-gram BNDM/SBNDM) [1002], MBNDMq (Multiple BNDMq) [1003], LBNDM (long BNDM) [1001], FBNDM (Forward BNDM) [1004].

Одним із різновидів біт – паралельного пошуку є апаратний підхід, який передбачає використання комп′ютерної SIMD-архітектури. SIMD-алгоритми розроблені для пришвидшення виконання співставлення рядків із використанням апаратного підходу. А саме – функціональних можливостей SIMD-команд. До них належать алгоритми, описані в [1005 - 1007]. Вдалий огляд біт – паралельних алгоритмів наведено у праці [1008].

Цей підхід особливо швидкий, коли зразок не перевищує розмір комп′ютерного слова.

Гібридний підхід. Він добре підходить для вирішення важких задач, адже поєднує переваги різних алгоритмів. Багато алгоритмів використовують гібридну концепцію. Скажімо, алгоритми, описані в [1009 - 1011] поєднують символьний і автоматний підходи, а [1012 - 1013] – символьний і хешированій.

Алгоритми неточного співставлення

Пропонувалися і досліджувалися також алгоритми неточного (приблизного) співставлення.

Неточні алгоритми передбачають введення поняття відстані між рядками і знаходження тих, що містяться на відстані, яка не перевищує задану відносно вихідного рядка.

В зв′язку з цим було введене поняття відстані реагування як способу кількісної оцінки того, наскільки два рядки (не) схожі один на одного, через підрахунок мінімальної кількості операцій, необхідних для перетворення одного рядка в інший.

У статті «String metric» у Вікіпедії (https://en.wikipedia.org/wiki/String_metric) представлені 20 видів метрик на рядках. Наведемо ті з методів визначення ступеню схожості, які найбільше згадувані в науковій літературі.

Відстань Хеммінга (Hamming distance). Введена Річардом Хеммінгом (Richard Hamming) [1014] 1950 року, визначає число позицій, в яких відповідні символи двох слів однакової довжини є відмінними. В більш загальному випадку відстань Хеммінга застосовується для рядків однакової довжини і є метрикою відмінності об′єктів однакової розмірності.

Відстань Левенштейна. Введена радянським ученим Владіміром Левенштейном [1015, 1016] 1965 року і визначається для рядків різної довжини як кількість односимвольних операцій вставки, видалення і заміни, що переводять один рядок в інший. Внесок у вивчення цієї відстані зробив також Ден Гасфілд (Dan Gusfield) [1017].

Відстань Дамерау – Левенштейна (Damerau–Levenshtein). Є узагальненням відстані Левенштейна, запропонована Фрідеріком Дамерау (Frederick J. Damerau) [1018]. Це міра різниці двох рядків символів, що визначається як мінімальна кількість операцій вставки, видалення, зміни й транспозиції (перестановки двох сусідніх символів), необхідних для переведення одного рядка в інший.

Відстань Джаро – Вінклера (Jaro–Winkler distance). Подана 1990 року Уільямом Е. Вінклером (William E. Winkler) [1019] на основі відстані Джаро (Matthew A. Jaro) [1020], запропонованої 1989 року. Неформально відстань Джаро між двома словами – це мінімальна кількість односимвольних перестановок, необхідна для зміни одного слова в інше.

Найбільша загальна послідовність (longest common subsequence) [1021] визначає відстань із використанням операцій вставки й видалення (без заміни).

На основі перелічених вище відстаней була запропонована велика кількість алгоритмів неточного співставлення рядків, із якими можна ознайомитися в оглядах [1022 - 1028].

Відстань при впорядкованому алфавіті. Всі перелічені вище алгоритми неточного співставлення рядків не враховують факт можливого впорядкування символів алфавіту. В свою чергу впорядкований алфавіт дозволяє вводити відстань між символами алфавіту, між двома рядками й на їх основі – відповідні методи співставлення рядків. Так, зокрема, в праці[937a] визначаються Δ-відстань, Г-відстань і (Δ; Г)-відстань між рядками й відповідні методи неточного співставлення рядків.

Огляд літератури. На закінчення огляду алгоритмів співставлення наведемо ще кілька статей.

Вдалий огляд точного співставлення рядків дано в [939]. У статті подана широка класифікація методів співставлення, опис понад 50 алгоритмів і їх порівняльний аналіз. Обговорюються нові напрямки, можливі проблеми, поточні тенденції в сфері алгоритмів співставлення рядків з основним наголосом на алгоритми точного співставлення.

У монографії французьких учених Крістіан Чаррас (Christian Charras) і Тьєррі Лекрок (Tierry Lecrock) [941] подано детальний опис 35 алгоритмів точного співставлення рядків.

Достойний цикл статей представив італійський учений Сімоне Фаро (Simone Faro) та його колеги. В статті [942] наводиться класифікація алгоритмів точного співставлення рядків і 124 алгоритми зі стислою анотацією, які пропонувалися протягом 1970 – 2016 років, з наведенням бібліографії. В праці [943] описується ефективний і гнучкий інструментальний засіб SMART (String Matching Algorithms Research Tool), призначений для розробки, тестування, порівняння і оцінки алгоритмів співставлення рядків. Наводиться також список 124 алгоритмів співставлення, розроблених за період 1970 – 2016 років і представлених в SMART. У [944] наводиться огляд алгоритмів точного співставлення рядків, розроблених після 2000 року, експериментальні дані оцінки деяких із цих алгоритмів.

Електронні бібліотеки

Електронна бібліотека – це розподілена документальна інформаційно – пошукова система, створена на базі ПТБД, яка надає різнорідні колекції електронних документів у глобальній мережі комп'ютерів у зручному для користувачів вигляді.

Однією з ранніх праць на тему електронних бібліотек була монографія Ліклайдера (Joseph Carl Robnett Licklider) [1029], написана 1965 року. В ній передбачалося існування всесвітньої мережі комп'ютерів, яка містить оцифровані версії будь – коли написаної літератури, до якої надається безпосередній доступ.

Джозеф Ліклайдер

До середини 70-х років здійснювалися дослідження й розробки у сфері онлайнових інформаційних систем і сервісів із ЕБ включно. Їх детальний аналіз здійснено у монографії [1030]. Серед перших проєктів, які можна віднести до ЕБ, вважається Проєкт «Гутенберг» [1031] – громадська некомерційна ініціатива, спрямована на створення й розповсюдження цифрової колекції творів, які є суспільним надбанням. Проєкт було створено 4 липня 1971 року, коли студент Іллінойського університету Майкл Харт (Michael Stern Hart) вручну передрукував текст Декларації незалежності США і відправив його іншим користувачам своєї мережі. Станом на 2021 рік у колекції Проєкту було понад 60000 книг.

Майкл Харт

Термін «електронна бібліотека» (digital library) почав широко використовуватися з 1991 року в зв′язку із проведенням серії семінарів, фінансованих Національним науковим фондом США(US National Science Foundation NSF). Того ж року була створена система “e-print archive”, яку згодом перейменували на arXiv.

У середині 90-х була усвідомлена важливість створення ЕБ на державних рівнях. 1997 року в США стартував національний проєкт «Ініціатива електронних бібліотек» (Digital Library Initiative - DLI), що став серйозним поштовхом у розвитку як ЕБ, так і національних програм, проєктів і організаційних структур в інших країнах: Великобританії, Канаді, Австралії, Японії, ЄС тощо. 1996 року в спеціальному випуску журналу EEE Computer [1032], присвяченому побудові великих ЕБ, підбивалися підсумки виконання проєктів аж до 1996 року.

У зв′язку із активним зростанням кількості ЕБ до кінця минулого століття наукова громадськість дійшла розуміння необхідності створення концептуальних положень і моделей ЕБ вцілому і для різних предметних галузей зокрема. Тож у

наступному десятилітті було висунуто багато таких моделей.

- У 1991 – 1997 роках Міжнародною федерацією бібліотечних асоціацій і закладів була розроблена ER-модель «Функціональні вимоги до бібліографічних записів» (Functional Requirements for Bibliographic Records, FRBR) [1033] як узагальнене представлення бібліографічного універсаму, незалежного від будь-якого варіанту каталогізації або реалізації.

- Концептуальна еталонна модель CIDOC CRM [1034] Міжнародного комітету із документації Міжнародної ради музеїв, призначена для інтеграції, посередництва і обміну інформацією в сфері світової культурної спадщини й пов′язаних сфер.

- 1991 року був представлений Загальноєвропейський дослідницький інформаційний формат CERIF (Common European Research Information Format - CERIF) [1035] як всеохоплюючої інформаційної моделі предметної галузі наукових досліджень. Модель CERIF є стандартом в ЄС.

- Вже багато років досліджується проблема аналізу семантики зв′язків між науковими матеріалами. Системним узагальненням цих результатів став комплекс антологій SPAR (Semantic Publishing and Referencing) [1036], який забезпечував доволі детальну категоризацію відношень, що можуть виникати між науковими матеріалами в електронному вигляді, і втілюючих їх зв′язків.

- Група спеціалістів DELOS Європейського дослідного консорціуму з інформатики й математики ERCIM у 2006 – 2007 роках на основі аналізу бібліотечних систем [1037], де велику увагу приділялося функціональним можливостям сучасних ЕБ, спочатку сформулювали маніфест ЕБ [1038]. У ньому були висвітлені наріжні концептуальні принципи ЕБ, а потім розроблено еталонну модель ЕБ DLRM (Digital Library Reference Model) [1039], яка стала де-факто стандартом у Європейському Союзі.

- Гонсалвес (Goncalves) та інші запропонували модель 5S [1040, 1041] як теоретичної бази ЕБ, що сприяло появі великої кількості мета моделей для різних типів ЕБ.

Поява тисяч ЕБ в Інтернеті викликала необхідність їх інтеграції. Серед цілої низки пропозицій у цьому напрямку найефективнішим й загальноприйнятим став протокол OAI PMH – протокол міжбібліотечного обміну метаданими, створений 2001 року [1042]. Його створення привело до появи сайтів – інтеграторів (харвесторів), які об′єднують велику кількість ЕБ й надають спільну точку пошуку й доступу до них. Прикладами таких харвесторів можуть бути наступні:

- Bielefeld University Library - http://www.base-search.net/ - пошукова система Білефельдського університету інтегрує ресурси понад 9000 провайдерів даних;

OpenAire – інформаційні ресурси відкритого доступу Європейського Союзу (56 мільйонів документів) - https://www.openaire.eu/;

Core – найбільша в світі колекція (понад 135 мільйонів) наукових статей відкритого доступу **-** https://core.ac.uk/.

Було розроблено понад 30 інструментальних засобів створення ЕБ (див.: http://roar.eprints.org/). Найпопулярнішими є DSpace, EPrints, OJS, Bepress, OPUS, Fedora, Greenstone.

References

755. Strozzi C. NoSQL – A relational database management system. 2007–2010. – http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page
756. Evans E. NoSQL 2009. May 2009. – Blog post of 2009-05-12. - http://blog.sym-link.com/posts/2009/12/nosql_2009/
757. Evans E. NoSQL: What's in a name? October 2009. – Blog post of 2009-10-30. - http://blog.sym-link.com/posts/2009/30/nosql_whats_in_a_name/
758. Fox A, Brewer E. Harvest, yield and scalable tolerant systems. In: Proceedings of Workshop on Hot Topics in Operating Systems; 1999. p. 174–178.
759. Seth Gilbert, Nancy Lynch. Brewer's conjecture and the feasibility of consistent,

available, partition-tolerant web services. ACM SIGACT News, Volume 33 Issue 2, June 2002, pp. 51-59,

760. Abadi D. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. Computer 45(2), 37-42 (2012)
761. Strauch Ch. "NoSQL Databases". - http://www.christof-strauch.de/ nosqldbs.pdf
762. Kepner J., Chaidez J., Gadepally, Jansen H. "Associative arrays: Unified mathematics for spreadsheets, databases, matrices, and graphs," New England Database Day, 2015.
763. Kepner J., Chaidez J., "The Abstract Algebra of Big Data and Associative Arrays," SIAM Meeting on Discrete Math, Jun 2014, Minneapolis, MN.
764. Jeremy Kepner, Vijay Gadepally, Dylan Hutchison, Hayden Jananthan, Timothy Mattson, Siddharth Samsi, Albert Reuther. Associative Array Model of SQL, NoSQL, and NewSQL Databases. 2016 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–9. IEEE (2016)
765. A Brief History of NoSQL. - http://blog.knuthaugen.no/2010/03/a-brief-history-of-nosql.html
766. GT.M - https://en.wikipedia.org/wiki/GT.M
767. DB-Engines Ranking of Key-value Stores. - db-engines.com/en/ranking/key-value+store
768. Rusher J., Networks R. Triple Store. - https://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html
769. Tweed R., James G. A Universal NoSQL Engine, Using a Tried and Tested Technology. - http://www.mgateway.com/docs/universalNoSQL.pdf, 2010. - 25 p.
770. Welcome to the UnQL Specification home - http://www.unqlspec.org/display/UnQL
771. Bach M., Werner A. Standardization of NoSQL Database Languages. In: Kozielski S., Mrozek D., Kasprowski P., Małysiak-Mrozek B., Kostrzewa D. (eds) Beyond Databases, Architectures, and Structures. BDAS 2014. Communications in Computer and Information Science, vol 424. Springer, Cham. 2014, pp. 50–60
772. Angles R., Gutierrez C. Survey of graph database models. ACM Comput. Surv. 40, 1, Article 1, 2008, 39 p.
773. Suciu D. Semi-structured Data Model. In Encyclopedia of Database Sys-tems, Ling Liu, M. Tamer Özsu Editors, pp. 3446-3451
774. Suciu D. Semi-structured Query Languages. In Encyclopedia of Database Sys-tems, Ling Liu, M. Tamer Özsu Editors, pp. 3457-3459
775. Luniewski A., Shoens K., Schwarz P., Stamos J., Thomas J. The Rufus system: information organization for semi-structured data. In: Proceedings of the 19th International Conference on Very Large Data Bases; 1993. p. 97–107.
776. Papakonstantinou Y., Garcia-Molina H., Widom J. Object exchange across heterogeneous information sources. In: Proceedings of the 11th International Conference on Data Engineering; 1995. p. 251–260.
777. Garcia-Molina H., Papakonstantinou Y., Quass D., Rajaraman A., Sagiv Y., Ullman J., Widom J. The TSIMMIS project: integration of heterogeneous information sources. J Intell Inf Syst. 1997;8(2):117–132.
778. Buneman P., Davidson S., Hillebrand G., Suciu D. A query language and optimization techniques for unstructured data. In: Proceedings of the ACM SIGMOD International Conference on Management of Data; 1996. p. 505–516.
779. Buneman P., Fernandez M., Suciu D. UNQL: a query language and algebra for semistructured data based on structural recursion. VLDB J. 2000;9(1): 76–110.
780. Deutsch A., Fernandez M., Florescu D., Levy A., Suciu D. A query language for XML. In: Proceedings of the 8th International World Wide Web Conference; 1999. p. 77–91.
781. Abiteboul S., Quass D., McHugh J., Widom J., Wiener J. The Lorel query language for semistructured data. 1996. http://www-db.stanford.edu/lore/.
782. Papakonstantinou Y., Abiteboul S., Garcia-Molina H. Object fusion in mediator systems. In: Proceedings of the 22th International Conference on Very Large Data Bases; 1996. p. 413–424.

783. Best Document Databases. - https://www.g2.com/categories/document-databases
784. Estabrook G F., Brill R.C. The Theory of the TAXIR accessioner. Mathematical Biosciences, 1969, Vol. 5, No 3–4, pp. 327-340.
785. Weyl S. Fries J.F. Wiederhold G., Germano F."A Modular Self-describing Clinical Databank System". Computers and Biomedical Research. 1975. 8 (3): 279–293.
786. Turner M.J., Hammond R., Cotton P. A DBMS for Large Statistical Databases. VLDB '79: Proceedings of the fifth international conference on Very Large Data Bases - Vol. 5, 1979, pp. 319–327
787. "SCSS from SPSS, Inc". ComputerWorld. September 26, 1977. p. 28.
788. Karasalo I., Svensson P. An overview of cantor: a new system for data analysis. SSDBM'83: Proceedings of the Second International Workshop on Statistical Database ManagementSeptember, 1983, pp.315–324
789. Don S. Batory. On searching transposed files. ACM Transactions on Database Systems, 4(4):531–544, 1979.
790. Hoffer J.A. , Severance D.G. The use of cluster analysis in physical data base design. In VLDB '75: Proceedings of the 1st International Conference on Very Large Data BasesSeptember 1975 Pages 69–86, 1975.
791. Copeland G.P., Khoshafian S.N. . A decomposition storage model. In Proceedings of the ACM SIGMOD Conference on Management of Data,1985, pp. 268–279
792. Khoshafian S., Valduriez P. Parallel execution strategies for declustered databases. In Proceedings of the International Workshop on Database Machines, pages 458–471, 1987.
793. Khoshafian S., Copeland G., Jagodis T., Boral H., Valduriez P. A query processing strategy for the decomposed storage model. In Proceedings of the International Conference on Data Endineering (ICDE), pp. 636–643, 1987.
794. Boncz P. Monet: A next-generation DBMS kernel for queryintensive applications. University of Amsterdam, PhD Thesis, 2002.
795. Idreos S., Groffen F., Nes N., Manegold S., Mullender S., Kersten M.L MonetDB: Two Decades of Research in Column-oriented Database Architectures. IEEE Data Eng. Bull., 35(1):40–45, 2012.
796. Boncz P., Zukowski M., Nes N. MonetDB/X100: Hyperpipelining query execution. In Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR), 2005, pp. 225-.237
797. Zukowski M., Boncz P.A., Nes N, Heman S. MonetDB/X100 - A DBMS In The CPU Cache. IEEE Data Engineering Bulletin, 28(2): 17–22, June 2005.
798. Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel R. Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alexander Rasin, Nga Tran, and Stan B. Zdonik. C-Store: A Column-Oriented DBMS. In Proceedings of the International Conference on Very Large Data Bases (VLDB), pages 553–564, 2005.
799. Lamb A., Fuller M., Varadarajan R., Tran N., Vandiver B., Doshi L., Bear C. The Vertica analytic database: C-store 7 years later. Proceedings of the VLDB Endowment, Vol. 5, No 12, 2012 pp 1790–1801.
800. Abadi D.J., Madden S.R., Ferreira M. Integrating compression and execution in column-oriented database systems. In Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 671–682, 2006.
801. Abadi D.J., Myers D.S., DeWitt D.J., Madden S.R. Materialization strategies in a column-oriented DBMS. In Proceedings of the International Conference on Data Endineering (ICDE), pp. 466–475, 2007.
802. Idreos S., Kersten M.L., Manegold S. Self-organizing tuple reconstruction in column stores. In Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 297–308, 2009.
803. Zukowski M., Heman S., Nes N., Boncz P. Super-Scalar RAM-CPU Cache Compression. In Proceedings of the 22nd International Conference on Data Endineering (ICDE), 2006. pp. 59-71

804. Abadi D.J., Boncz P., Harizopoulos S., Idreos S., Madden S. (2013), "The Design and Implementation of Modern Column-Oriented Database Systems", Foundations and Trends® in Databases: Vol. 5: No. 3, pp 197-280.

804a. Goncalves R., Kersten M.L. The Data Cyclotron Query Processing Scheme. ACM Transactions on Database Systems, Vo. 36. No 4. December 2011, Article No. 27, pp. 1–35

805. Manegold S., Boncz P., Nes N., Kersten M.. Cache-conscious radixdecluster projections. In Proceedings of the International Conference on Very Large Data Bases (VLDB), pages 684–695, 2004.

806. 22) Abadi D.J., Madden S.R., Hachem N. Column-stores vs. row-stores: how different are they really? In: Proceedings of the ACM SIGMOD International Conference on Management of Data; 2008. p. 967– 980.

807. Halverson A., Beckmann J.L., Naughton J.F., DeWitt D.J. A Comparison of C-Store and Row-Store in a Common Framework. Technical Report TR1570, University of Wisconsin-Madison, 2006. - https://minds.wisconsin.edu/bitstream/handle/1793/60514/TR1570.pdf?sequence=1

808. Harizopoulos S., Liang V., Abadi D.J., Madden S.R. Performance tradeoffs in read-optimized databases. In VLDB, pages 487–498, 2006.

809. Idreos S., Kersten M., Manegold S. Database Cracking. Conference: CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, 2007, pp. 68-78

810. Héman S., Zukowski M., Nes N.J., Sidirourgos L., Boncz P. Positional update handling in column stores. In Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 543–554, 2010.

811. Pingpeng Yuan and Hai Jin. Column Stores. In: Encyclopedia of Database Systems, Ling Liu, M. Tamer Özsu Editors. pp. 518-523.

812. Abadi D.J., Boncz P.A. Harizopoulos S. Column-oriented database systems. Proceedings of the VLDB Endowment, Vol. 2, No. 2, 2009, pp. 1664–1665

813. Kanungo A. Column oriented databases. International Journal of Advanced Computational Engineering and Networking, 2017, Vol. 5, No 8, pp. 10-13

814. Abadi D., Boncz P., Harizopoulos S. VLDB 2009 Tutorial on Column-Stores. - https://www.slideshare.net/abadid/vldb-2009-tutorial-on-columnstores

815. Robinson I., Webber J., Eifrem E. Graph Databases, 2nd Edition. O'Reilly Media, Inc. 2015, 218 p.

816. Wood P.T. Graph Database. In Encyclopedia of Database Systems, Ling Liu, M. Tamer Özsu Editors, pp. 1639-1643

817. Angles R. Graph Databases - http://renzoangles.net/gdm/

818. Angles R., Gutierrez C Querying RDF data from a graph database perspective European semantic web conference, 2005, pp. 346-360

819. Angles R., Gutierrez C. Survey of graph database models. ACM Computing Surveys, Vol. 40, No. 1, Article 1, 2008, pp. 1-39.

820. Angles R., Gutierrez C. The expressive power of SPARQL International Semantic Web Conference, 2008, pp.114-129

821. Angles R. A comparison of current graph database models. IEEE 28th International Conference on Data Engineering Workshops, 2012, 171-177

822. Angles R., Arenas M., Barceló P., Hogan A., Reutter J., Vrgoč D. Foundations of modern query languages for graph databases. ACM Computing Surveys (CSUR), 2017, Vol. 50, No 5, Article No.: 68, pp. 1–40

823. Angles R., Arenas M., Barceló P., Boncz P., Fletcher G., Gutierrez C. G-CORE: A core for future graph query languages. Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 1421-1432

823a. Sakr S. Pardede M. (Eds.). Graph Data Management: Techniques and Applications. IGI Global, 2011, 502 p.

824. Angles R. The property graph database model. In Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, CEUR Workshop Proceedings. CEUR-WS.org, 2018, [Online] URL: http://ceur-ws.org/Vol-2100/paper26.pdf

825. Angles R., Gutierrez C. An Introduction to Graph Data Management: Fundamental Issues and Recent Developments. in Graph Data Management, Springer Publishing Companyt, 2018, pp.1-32
826. Angles R., Barcelo P., Rios G. A practical query language for graph DBs. In: 7th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW), 2013
827. Rodriguez M.A., Neubauer P. Constructions from dots and lines. Bulletin of the American Society for Information Science and Technology, 2010, 36,(6), pp. 35-41
828. Berge C. Graph and Hypergraphs. North-Holland Publishing Company, Amsterdam, 1973
829. Roussopoulos N., Mylopoulos, J. Using semantic networks for database management. In Proceedings of the International Conference on Very Large Data Bases (VLDB). ACM, 1975, 144–172
830. Shipman D.W. The functional data model and the data language DAPLEX. ACM Transactions on Database Systems, vol. 6, No. 1, 1981, pp. 140–173
831. Kuper G.M., Vardi M.Y. A new approach to database logic. In Proceedings of the 3th Symposium on Principles of Database Systems (PODS). ACM Press, 1984, pp. 86–96
832. Kunii H.S. DBMS with graph data model for knowledge handling. In Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: Today and Tomorrow. IEEE Computer Society Press, 1987, pp. 138–142
833. Lecluse C., Richard P., Velez F. O2, an object-oriented data model. In Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM Press, 1988, pp. 424–433.
834. Tompa F.W. A data model for flexible hypertext database systems. ACM Transactions on Information Systems, Vol. 7, No 1, 1989, pp. 85–100
835. Gyssens M., Paredaens J., Den Bussche J.V., Gucht D.V. A graph-oriented object database model. In Proceedings of the 9th Symposium on Principles of Database Systems (PODS). ACM Press, 1990, pp. 417–424
836. Watters C., Shepherd M.A. A transient hypergraph-based model for data access. ACM Trans. Inform. Syst. 8 (2), 1990, pp. 77–102
837. Levene M., Poulovassilis A. The Hypernode model and its associated query language. In Proceedings of the 5th Jerusalem Conference on Information technology. IEEE Computer Society Press, 1990, pp. 520–530
838. Levene M., Poulovassilis A. An object-oriented data model formalised through hypergraphs. Data Knowl. Eng. 6 (3), 1991, pp. 205–224
839. Andries M., Gemis M., Paredaens J., Thyssens I., Den Bussche J.V. Concepts for graph-oriented object manipulation. In Proceedings of the 3rd International Conference on Extending Database Technology (EDBT). LNCS, vol. 580. Springer, 1992., pp. 21–38
840. Amann B., Scholl M. Gram: A Graph Data Model and Query Language. In European Conference on Hypertext Technology (ECHT). ACM, 1992, pp. 201–211
841. Mainguenaud M., Simatic X.T. A data model to deal with multi-scaled networks. Computers, Environment and Urban Systems, 1992, vol.16, No 4, pp. 281–288
842. Gemis M., Paredaens J. An object-oriented pattern matching language. In Proceedings of the First JSSST International Symposium on Object Technologies for Advanced Software. Springer-Verlag, 1993, pp. 339–355
843. Hidders J., Paredaens J. GOAL, A graph-based object and association language. Advances in Database Systems: Implementations and Applications, CISM, 1993, pp. 247–265
844. Consens M., Mendelzon A. Hy+: A hygraph-based query and visualization system. ACM SIGMOD Record, Vol. 22, No 2, 1993, pp. 511–516
845. Guting R.H. GraphDB: modeling and querying graphs in databases. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB). Morgan Kaufmann, 1994, pp. 297–308
846. Gutierrez A., Pucheral P., Steffen H., Thevenin J.-M. Database graph views: A practical model to manage persistent

graphs. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB). Morgan Kaufmann, 1994. pp. 391–402

847. Poulovassilis A., Levene M. A Nested-Graph Model for the Representation and Manipulation of Complex Objects. ACM Transactions on Information Systems (TOIS) 12(1), 1994, pp. 35–68
848. Paredaens J., Peelman P., Tanca L. G-Log: A graph-based query language. IEEE Trans. Knowl. Data Eng. 7, 3, 1995, pp. 436–453
849. Graves M., Bergeman E.R., Lawrence C.B. A graph-theoretic data model for genome mapping databases. In Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS). IEEE Computer Society, 1995, pp. 32-41
850. Levene M., Loizou G. A graph-based data model and its ramifications. IEEE Trans. Knowl. Data Eng. 7, 5, 1995, pp. 809–823
851. KIesel N., Schurr A., Westfechtel B. GRAS, graph-oriented software engineering database system.Information Systems, Vol. 20, No 1, 1995, pp. 21-51
852. Sheng L., Ozsoyoglu Z. M., Ozsoyoglu G. A graph query language and its query processing. In Proceedings of the 15th International Conference on Data Engineering (ICDE). IEEE Computer Society, 1999, pp. 572–581.
853. Hidders J. Typing graph-manipulation operations. In Proceedings of the 9th International Conference on Database Theory (ICDT). Springer-Verlag, 2002. pp. 394–409
854. Spyratos N., Sugibuchi T. (2016) PROPER - A Graph Data Model Based on Property Graphs. In: Grant E., Kotzinos D., Laurent D., Spyratos N., Tanaka Y. (eds) Information Search, Integration, and Personalization. ISIP 2015, pp. 23-35
855. Wood, P.T.: Query languages for graph databases. ACM SIGMOD Record, 2012, Vol. 41, No 1, pp. 50–60.
856. Barceló P. Querying graph databases. In: PODS '13: Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems, 2013, pp. 175–188
857. Kowalik L Adjacency queries in dynamic sparse graphs. IInformation Processing Letters, 2007, vol. 102, pp. 191–195
858. Papadopoulos A.N., Manolopoulos Y. Nearest neighbor search - a database perspective. Series in computer science. Springer, Berlin, 2005, 170 p.
859. Aggarwal C.C., Wang H. (eds) Managing and mining graph data. Advances in database systems. Springer Science – Business Media, Berlin, 2005
860. Washio T.,Motoda H. State of the Art of Graph-based Data Mining. SIGKDD Explorer Newsletter, 2003, vol. 5, no. 1, pp. 59–68
861. Yannakakis M. Graph-theoretic methods in database theory. In: Proceedings of the symposium on principles of database systems (PODS). ACM, New York, 1990, pp 230–242
862. Barcelo P., Libkin L., Reutter J. Querying graph patterns. In Proc. of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), 2011, pp. 199–210
863. Wang X. Finding patterns on protein surfaces: Algorithms and applications to protein classification. IEEE Transactions on Knowledge and Data Engineering, 2005, vol. 17, pp. 1065–1078
864. Carroll J. Matching RDF Graphs. In Proceedings of the International Semantic Web Conference (ISWC), 2002, pp. 5-15
865. Cruz I.F., Mendelzon A.O., Wood P.T. A graphical query language supporting recursion. ACM SIGMOD Record, Vol. 16, No 3, 1987, pp 323–330
866. Fan W., Li J. Ma S., Tang N., Wu Y. Adding regular expressions to graph reachability and pattern queries. iin Proc. of the IEEE 27th International Conference on Data Engineering (ICDE), 2011, pp. 39–50
867. Mendelzon A.O., Wood P.T. Finding regular simple paths in graph databases. SIAM J Comput, 1995, 24(6), pp. 1235–1258
868. Zhu A.D., Ma H., Xiao X., Luo S,. Tang Y., Zhou S. Shortest path and distance queries on road networks: towards bridging theory and practice. In: Proceedings of the international conference on man-

agement of data (SIGMOD). ACM, New York, 2013, pp. 857–868

869. Kanza Y., Sagiv Y. Flexible queries over semistructured data. PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 2001, pp. 40–51.
870. Hurtado C.A., Poulovassilis A., Wood P.T. Ranking approximate answers to semantic web queries. Ranking Approximate Answers to Semantic Web Queries. In: Aroyo L. et al. (eds) The Semantic Web: Research and Applications. ESWC 2009. Lecture Notes in Computer Science, vol 5554. Springer, Berlin, Heidelberg. 2009, pp. 263–277
871. Cruz, I.F., Mendelzon, A.O., Wood, P.T. G+: Recursive Queries without Recursion. In: Proceedings of the 2th International Conference on Expert Database Systems (EDS). 1989, pp. 645–666
872. Consens, M.P., Mendelzon, A.O. GraphLog: a Visual Formalism for Real Life Recursion. In: Proceedings of the 9th ACM Symposium on Principles of Database Systems. 1990, pp. 404–416.
873. Wood, P.T.: Factoring Augmented Regular Chain Programs. In: Proceedings of the 16th International Conference on Very Large Data Bases (VLDB). 1990, pp. 255– 263. Morgan Kaufmann Publishers Inc.
874. Abiteboul S., Quass D., McHugh J., Widom J., Wiener J.L. The Lorel query language for semistructured data. International Journal on Digital Libraries, 1997, 1(1), pp. 68–88
875. Flesca, S., Greco, S.: Partially Ordered Regular Languages for Graph Queries. In: Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP). LNCS, 1999, pp. 321–330
876. Buneman P., M. Fernandez, Suciu D. UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. The VLDB Journal, 2000, 9(1), pp. 76-110
877. Hidders A.J.H. A Graph-based Update Language for Object-Oriented Data Models. Thesis (doctoral)--Technische Universiteit Eindhoven, 2001, 217 p. - https://pure.tue.nl/ws/files/2236754/200142116.pdf
878. Cardelli L., Gardner P., Ghelli G.: A Spatial Logic for Querying Graphs. In: Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP). 2002, pp. 597–610. LNCS, Springer
879. Theodoratos D. Semantic Integration and Querying of Heterogeneous Data Sources Using a Hypergraph Data Model. In: Proceedings of the 19th British National Conference on Databases (BNCOD), Advances in Databases. 2002, pp. 166– 182. LNCS, Springer
880. Leser U. A query language for biological networks. Bioinformatics, 2005, 21(2), pp. 33-39
881. Liu Y.A., Stoller S.D. Querying complex graphs. In: Proc. of the 8th Int. Symposium on Practical Aspects of Declarative Languages. 2006, pp. 16–30
882. Prud'hommeaux, E., Seaborne, A. SPARQL Query Language for RDF. W3C Recommendation. (January 15 2008)
883. Ronen R., Shmueli O. SoQL: a language for querying and creating data in social networks. In: Proceedings of the international conference on data engineering (ICDE). IEEE Computer Society, New York, 2009, pp 1595–1602
884. Dries A, Nijssen S., De Raedt L. A query language for analyzing networks. Proceedings of the 18th ACM conference on Information and knowledge, 2009, pp. 485-494
885. Rodriguez M.A. The Gremlin graph traversal machine and language (invited talk). In: DBPL 2015: Proceedings of the 15th Symposium on Database Programming Languages. ACM, New York, 2015, pp 1–10
886. San Martin M., Gutierrez C., Wood P.T. SNQL: A social networks query and transformation language. In: Barcelo, P. and Tannen, V. (eds.) Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management. CEUR Workshop Proceedings. CEUR-WS.org. 2011.
887. Cypher - Graph Query Language - http://neo4j.com/developer/cypher-query-language/

888. Barcelo P., Libkin L., Lin A.W., Wood P.T. Expressive languages for path queries over graph-structured data. ACM Transactions on Database Systems, 2012, Vol. 37, No 4, pp. 1–46
889. Santini S.: Regular Languages with Variables on Graphs. Information and Computation, 2012, Vol. 211, pp. 1–28
890. Feigenbaum L , Williams G.T., Clark K.G., Torres E. SPARQL 1.1 Protocol. W3C Recommendation. http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/, March 21, 2013.
891. van Rest O., Hong S., Kim J., Meng X., Chafi H. PGQL: a property graph query language. In: Proceedings of the international workshop on graph data management experiences and systems (GRADES), 2013
892. Libkin L., Martens W., Vrgoc D. Querying Graph Databases with XPath. In: Proceedings of the 16th International Conference on Database Theory (ICDT), 2013, pp. 129–140
893. Brijder R., Gillis J.J.M., Van den Bussche J. (2013) The DNA query language DNAQL. In: ICDT '13: Proceedings of the 16th International Conference on Database Theory, 2013, pp. 1–9
894. Reutter J.L., Romero M., Vardi M.Y.: Regular queries on graph databases. In: Proceedings of the 18th International Conference on Database Theory (ICDT). 2015, pp. 177–194
895. GraphQL: A data query language. - https://code.fb.com/core-data/graphql-a-data-query-language/
896. Masseroli M., Pinoli P., Venco F., Kaitoua A., Jalili V., Paluzzi F., Muller H., Ceri S. GenoMetric Query Language: A novel approach to large-scale genomic data management. Bioinformatics, 2015, 31(12), pp. 1881-1888
897. Giugno R., Shasha D. GraphGrep: a fast and universal method for querying graphs. In: Proceedings of the 16th International Conference on Pattern Recognition, 2002. pp. 112–115.
898. He H., K. Singh A. Graphs-at-a-time: query language and access methods for graph databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data; 2008. p. 405–418.
899. Milo T., Suciu D.. Index structures for path expressions. In: Proceedings of the 7th International Conference on Database Theory; 1999. pp. 277–295
900. Picalausa F., Luo Y., Fletcher G.H.L., Hidders J, Vansummeren S. A structural approach to indexing triples. In: Proceedings of the 9th Extended Semantic Web Conference; 2012. p. 406–421
901. Trißl S., Leser U. Fast and practical indexing and querying of very large graphs. In: Proceedings of the ACM SIGMOD International Conference on Management of Data; 2007. p. 845–856.
902. Calvanese D., De Giacomo G., Lenzerini M., Vardi M.Y. Reasoning on regular path queries. SIGMOD Rec. 2003;32(4):83–92.
903. Fernandez M., Suciu D. Optimizing regular path expressions using graph schemas. In: Proceedings of the 14th International Conference on Data Engineering; 1998. p. 14–23.
904. Goldman R., Widom J. DataGuides: enabling query formulation and optimization in semistructured databases. In: Proceedings of the 23rd International Conference on Very Large Data Bases; 1997. p. 436–445.
905. Urbón P. NoSQL graph database matrix. - http://nosql.mypopescu.com/post/619181345/nosql-graph-database-matrix
906. Deepak Singh Rawat, Navneet Kumar Kashyap. Graph Database: A Complete GDBMS Survey.International Journal for Innovative Research in Science & Technology (IJIRST), 2017, Vol. 3, No 12, pp. 217-226
907. Pradeep Jadhav, Ruhi Oberoi. Comparative Analysis of Different Graph Databases, International Journal of Engineering Research & Technology (IJERT), Vol. 3, No 9, 2014, pp. 820-824
908. Stonebraker M., Madden S.R., Abadi D.J., Harizopoulos S., Hachem N.I. The End of an Architectural Era (It's Time for a Complete Rewrite). - VLDB '07: Proceedings of the 33rd international conference on Very large data basesSeptember 2007 Pages 1150–1160

909. R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg & D. Abadi, "H-store: a high-performance, distributed main memory transaction processing system," Proceedings of the VLDB Endowment, Volume 1 Issue 2, August 2008, pages 1496-1499.
910. M. Stonebraker, D. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N.Tran & S. Zdonik, "C-store: a column-oriented DBMS," Proceedings of the 31st International Conference on Very Large Data Bases (VLDB '05), 2005, pages 553 – 564.
911. Cattell Rick. "Scalable SQL and NoSQL data stores," ACM SIGMOD Record 39.4 (2011): 12-27.
912. Matthew A. (2011). "How Will The Database Incumbents Respond To NoSQL And NewSQL?". 451 Group - https://www.cs.cmu.edu/~pavlo/courses/fall2013/static/papers/aslett-newsql.pdf
913. Matthew A. (2011)." What we talk about when we talk about NewSQL". 451 Group - https://blogs.451research.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsql/
914. Stonebraker Mil. NewSQL: An Alternative to NoSQL and Old SQL for New OLTP Apps. Communications of the ACM Blog. - https://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext
915. Pavlo A., Aslett M. What's Really New with NewSQL?. SIGMOD Record, June 2016, Vol. 45, No. 2. pp. 45-55
916. Venkatesh, Prasanna (January 30, 2012). "NewSQL - The New Way to Handle Big Data". - https://www.opensourceforu.com/2012/01/newsql-handle-big-data/
917. Studer R., Benjamins R., Fensel D. Knowledge engineering: Principles and methods. Data & Knowledge Engineering, 25(1–2):161–198, 1998.
918. Guarino N., Oberle D., Staab S. What is an ontology? In Handbook on ontologies, pages 1–17. Springer, 2009.
919. Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., Tolle K.: The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In: SemWeb'01: Proceedings of the Second International Conference on Semantic Web - Volume 40 May 2001, pp. 1–13
920. Broekstra J., Kampman A., van Harmelen F. Sesame: A generic architecture for storing and querying RDFand RDF schema. In Proc. of the First Inter. Semantic Web Conf., pp. 54–68, 2002.
921. Pan Z., Heflin J.: Dldb: Extending relational databases to support semantic web queries. In: Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03). 2003, pp. 109–113
922. Harris S., Gibbins N. 3store: Efficient bulk RDF storage. In Proc. of the 1st Intern. Workshop on Practical and Scalable Semantic Systems (PSSS'03), 2003. pp. 1-15
923. Theoharis Y., Christophides V., Karvounarakis G. (2005) Benchmarking Database Representations of RDF/S Stores. In: Gil Y., Motta E., Benjamins V.R., Musen M.A. (eds) The Semantic Web – ISWC 2005. ISWC 2005. Lecture Notes in Computer Science, vol 3729. Springer, Berlin, Heidelberg. pp. 685-701
924. McBride B. Jena: Implementing the RDF Model and Syntax Specification. SemWeb'01: Proceedings of the Second International Conference on Semantic Web - Volume 40, May 2001, pp, 23–28
925. Agrawal R., Somani A., Xu Y. Storage and querying of e-commerce data. In: VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc. (2001) 149–158
926. Ma L., Su Z., Pan Y., Zhang M, Liu M. Rstar: an rdf storage and query system for enterprise resource management. thirteenth ACM international conference on Information and knowledge management, 2004:484 – 491.
927. Erling O., Mikhailov I.: RDF Support in the Virtuoso DBMS. In: Conference on Social Semantic Web (CSSW'07). Volume 113. (2007) 59–68
928. Wu Z., Eadon G., Das S., Chong E.I., Kolovski, V., Annamalai, M., Srinivasan,

J.: Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In: Proceedings of the 24th International Conference on Data Engineering (ICDE'08). (2008) 1239–1248

929. Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., Tolle K. On storing voluminous rdf descriptions: The case of web portal catalogs. In Proceedings of the Fourth International Workshop on the Web and Databases, WebDB 2001, Santa Barbara, California, USA, May 24-25, 2001, in conjunction with ACM PODS/SIGMOD 2001: 43-48
930. Abadi D.J., Marcus A., Madden S.R., Hollenbach K. Scalable Semantic Web Data Management Using Vertical Partitioning. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07). (2007) 411–422
931. Jing L., Li M.,Lei Z., Jean-Sébastien B., Chen W., Yue P., Yong Y., 2007. SOR: A Practical System for Ontology Storage, Reasoning. In VLDB 2007, 33rd Very Large Data Bases Conference,pp 1402-1405.
932. Dehainsala H., Pierra G., Bellatreche L. (2007) OntoDB: An Ontology-Based Database for Data Intensive Applications. In: Kotagiri R., Krishna P.R., Mohania M., Nantajeewarawat E. (eds) Advances in Databases: Concepts, Systems and Applications. DASFAA 2007. Lecture Notes in Computer Science, vol 4443. Springer, Berlin, Heidelberg. pp 497-508
933. Park M.J., Lee J.H., Lee C.H., Lin J., Serres O., Chung C.W.: An Efficient and Scalable Management of Ontology. In: Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07). (2007) 975–980
934. Wilkinson K., Sayers C., Kuno H., Reynolds D. 2003. Efficient RDF storage and Retrieval in Jena2. Proceedings of the 1st International Workshop on Semantic Web Database (SWDB'03). pp. 131–150.
935. SWAD-Europe Deliverable 10.2: Mapping Semantic Web Data with RDBMSes. - https://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/
936. Bailey J., Bry F., Furche T., Schaffert S. (2005) Web and Semantic Web Query Languages: A Survey. In: Eisinger N., Małuszyński J. (eds) Reasoning Web. Lecture Notes in Computer Science, vol 3564. Springer, Berlin, Heidelberg, 2005, pp. 35–133
937. Jean S., Aït-Ameur Y., Pierra G. (2006) Querying Ontology Based Database Using OntoQL (An Ontology Query Language). In: Meersman R., Tari Z. (eds) On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. OTM 2006. Lecture Notes in Computer Science, vol 4275. Springer, Berlin, Heidelberg. pp 704-721

937a. Melichar B., Holub J., Polcar T. Text searching algorithms Volume I: Forward string matching. Czech Technical University in Prague, 224 p. - /http://www.stringology.org/athens/TextSearchingAlgorithms/tsa-lectures-1.pdf

938. Melichar B., Holub J., Polcar T. Text searching algorithms Volume II: Backward string matching. Czech Technical University in Prague, 61 p.- http://www.stringology.org/athens/TextSearchingAlgorithms/tsa-lectures-2.pdf
939. Hakak S., Kamsin A., Shivakumara P., Gilkar G., Khan W.Z., Imran M. Exact String Matching Algorithms: Survey, Issues, and Future Research Directions. IEEE Access, 2019, Vol. 7, pp 69614-69637

941. Christian Charras, Thierry Lecroq. Handbook of exact string matching algorithms. College Publications (February 27, 2004), 256 p. - http://www-igm.univ-mlv.fr/~lecroq/string/string.pdf
942. Faro S. Exact Online String Matching Bibliography. 2016, 23 p. - https://arxiv.org/abs/1605.05067
943. Faro S., Lecroq T., Borzi, Di Mauro S., Maggio A.. The String Matching Algorithms Research Tool. Stringology 2016: 99-111
944. Faro S. Lecroq T,"The exact online string matching problem: A review of the most recent results", ACM Comput. Survey, Article 13, 42 pages, February 2013.
945. Koloud Al-Khamaiseh, Shadi AL Shagarin. A Survey of String Matching Algo-

rithms. Int. Journal of Engineering Research and Applications, 2014, vol. 4, No 7, pp.144-156

946. Morris, J.H., Jr; Pratt, V. (1970). A linear pattern-matching algorithm (Technical report). University of California, Berkeley, Computation Center. TR-40.
947. Knuth, Donald E. (1973). "The Dangers of Computer-Science Theory". Studies in Logic and the Foundations of Mathematics. 74: 189–195.
948. Knuth D., Morris J.H., Pratt V. (1977). "Fast pattern matching in strings". SIAM Journal on Computing. 6 (2): 323–350.
949. Matiyasevich, Yuri (1973). "Real-time recognition of the inclusion relation". Journal of Soviet Mathematics. 1: 64–70
950. Boyer R.S., Moore J.S. A fast string searching algorithm. Communications of the ACM. 1977, vol. 20, No 10. pp. 762—772. — doi:10.1145/359842.359859.
951. Baeza-Yates R., Gonnet G.H. A new approach to text searching. Communications of the ACM, Vol. 35, No 10,1992 pp 74–82. - https://doi.org/10.1145/135239.135243
952. Алгоритм Бойера-Мура - https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%BE%D0%B9%D0%B5%D1%80%D0%B0_%E2%80%94_%D0%9C%D1%83%D1%80%D0%B0
953. Horspool R.N. Practical fast searching in strings, Software - Practice & Experience, 1980, 10(6) :501-506.
954. Zhu R.F., Takaoka T., 1987, On improving the average case of the Boyer-Moore string matching algorithm, Journal of Information Processing 10(3):173-177.
955. Turbo-BM algorithm - http://www-igm.univ-mlv.fr/~lecroq/string/node15.html
956. CROCHEMORE, M., CZUMAJ A., GASIENIEC L., JAROMINEK S., LECROQ T., PLANDOWSKI W., RYTTER W., 1992, Deux méthodes pour accélérer l'algorithme de Boyer-Moore, in Théorie des Automates et Applications, Actes des 2e Journées Franco-Belges, D. Krob ed., Rouen, France, 1991, pp 45-63, PUR 176, Rouen, France.
957. Apostolico A., Giancarlo R. "The Boyer-Moore-Galil String Searching Strategies Revisited," (in English), SIAM Journal on Computing, vol. 15, No. 1, pp. 98-105, Feb 1986.
958. Smith P.D., "Experiments with a very fast substring search algorithm," Software-Practice and Experience, vol. 21, no. 10, pp. 1065-1074, 1991.
959. Raita T. Tuning the Boyer-Moore-Horspool string searching algorithm. Software-Practice and Experience, vol. 22, no. 10,pp. 879-884, 1992.
960. Crochemore M., Czumaj A., Gasieniec L., Jarominek S., Lecroq T., Plandowski W. Rytter W. "Speeding up two string-matching algorithms," Algorithmica 12(4-5):247-267, 1994.
961. Berry T., Ravindran, S. (2001) A Fast String Matching Algorithm and Experimental Results. Proceedings of the Prague Stringology Club Workshop '99, Collaborative Report DC-99-05, Czech Technical University, Prague, 16-26.
962. Sunday D.M. " A very fast substring search algorithm," Communications of the ACM, Vol. 33, No 8, 1990 pp 132–142. - https://doi.org/10.1145/79173.79184.
963. Colussi L. Correctness and efficiency of pattern matching algorithms. Information and Computation, vol. 95, no. 2, pp. 225-251, 1991.
964. Xian-feng H., Yu-bao Y., Xia L. "Hybrid pattern-matching algorithm based on BM-KMP algorithm." (ICACTE) 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), 2010, pp. V5-310-V5-313, DOI: 10.1109/ICACTE.2010.5579620.
965. Cao Z., Zhenzhen Y., Lihua L. "A fast string matching algorithm based on low-light characters in the pattern." In Advanced Computational Intelligence (ICACI), 2015 Seventh International Conference on, pp. 179-182. IEEE, 2015.
966. Hakak S., Kamsin A., Shivakumara P., Idna Idris M.Y., Gilkar G.A. "A new split based searching for exact pattern matching for natural texts." PloS ONE 13, no. 7 (2018): e0200912. Skid
967. Hakak S., Amirrudin K., Shivakumara P., Idna Idris M.Y. "Partition-Based Pattern Matching Approach for Efficient Retrieval Of Arabic Text." Malaysian

Journal of Computer Science 31, no. 3 (2018): 200-209.
968. Franek F., Jennings C.G., Smyth W.F. A simple fast hybrid pattern-matching algorithm. J. Discrete Algorithms, 5(4):682–695, 2007.
969. Rabin M.O., Karp R.M. Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development. 1987, vol. 31, No 2, pp. 249–260. — doi:10.1147/rd.312.0249.
970. Rabin–Karp algorithm. - https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm
971. Wu S., Manber U. "A fast algorithm for multi-pattern searching," Department of Computer Science, University of Arizona, Tucson, AZ, Report TR-94-171994.
972. Kim S., Kim Y., "A fast multiple string pattern matching algorithm," in Proceedings of 17th AoM/IAoM Conference on Computer Science, 1999, pp. 44-49.
973. Simone F. "A very fast string matching algorithm based on condensed alphabets." In International Conference on Algorithmic Applications in Management, pp. 65-76. Springer, Cham, 2016.
974. Lecroq T. "Fast exact string matching algorithms,"Information Processing Letters, vol. 102, no. 6, pp. 229-235, Jun 15 2007.
975. Kalsi P., Peltola H., Tarhio J. "Comparison of exact string matching algorithms for biological sequences," in Proceedings of the Second International Conference on Bioinformatics Research and Development, BIRD, 2008. pp. 417-426.
976. Daciuk J., Mihov S, Watson B., Watson R. Incremental construction of minimal acyclic finite state automata. Computational Linguistics, 2000, 26(1), pp.3-16.
977. Yang. W. Mealy machines are a better model of lexical analyzers. Computer Languages, Vol. 22, No 1, 1996, pp. 27-38
978. Blumer A., Blumer J., Ehrenfeucht A., Haussler D., McConnel R. Linear size finite automata for the set of all subwords of a word: an outline of results. Bull. European Assoc. Theoret. Comput. Sci., 21:12-20, 1983
979. Commentz-Walter B. A string matching algorithm fast on the average. Proceedings of the 6th Colloquium, on Automata, Languages and Programming, 1979, pp. 118–132
980. Allauzen C., Raffinot M. Simple optimal string matching algorithm, Journal of Algorithms, Vol. 36, No 1, 2000, pp. 102–116
981. Allauzen C., Crochemore M., Raffinot M. Factor oracle: A new structure for pattern matching. In 26th Seminar on Current Trends in Theory and Practice of Informatics (SOFSEM'99), Nov 1999, Milovy, Czech Republic, Czech Republic. pp.291-306.
982. Faro S., Lecroq T. Efficient variants of the Backward-Oracle-Matching algorithm. In Proceedings of the Prague Stringology Conference, Czech Republic, 2008, pp. 146-160: Czech Technical University.
983. Fan H., Yao N., Ma H. Fast variants of the backward-oraclemarching algorithm. In Fourth International Conference on Internet Computing for Science and Engineering, 2009, pp. 56-59.
984. He L., Fang B., Sui J. The wide window string matching algorithm. Theoretical Computer Science, vol. 332, no. 1-3, 2005, pp. 391-404
985. Liu C., Wang Y., Liu D., Li D. Two improved single pattern matching algorithms. In ICAT Workshops, Hangzhou, China,, 2006, pp. 419-422: IEEE Computer Society.
986. Hongbo F., Shupeng S., Jing Z., Li D. Suffix Type String Matching Algorithms Based on Multi-windows and Integer Comparison. In International Conference on Information and Communications Security, pp. 414-420. Springer, Cham, 2015.
987. Masaki Waga, Ichiro Hasuo, Kohei Suenaga. "Efficient online timed pattern matching by automata-based skipping." In International Conference on Formal Modeling and Analysis of Timed Systems, pp. 224-243. Springer, Cham, 2017.
988. Bitap algorithm. - https://en.wikipedia.org/wiki/Bitap_algorithm
989. Bálint Dömölki, An algorithm for syntactical analysis, Computational Linguistics 3, Hungarian Academy of Science pp. 29–46, 1964.

990. Bálint Dömölki, A universal compiler system based on production rules, BIT Numerical Mathematics, 8(4), pp 262–275, 1968. doi:10.1007/BF01933436
991. Shyamasundar R.K. Precedence parsing using Dömölki's algorithm, International Journal of Computer Mathematics, 6(2) pp 105–114, 1977.
992. Baeza-Yates R., Gonnet G.H. A new approach to text searching. Communications of the ACM, Vol. 35, No 10,1992 pp 74–82
993. Ricardo A. Baeza-Yatesm Gaston H. Gonnet. A New Approach to Text Searching. Communications of the ACM, 1992, vol. 35, No 10, pp. 74-82 - ПОВТОРЕНИЕ 5a)
994. Fredriksson K., Grabowski S. Practical and optimal string matching. In SPIRE'05: Proceedings of the 12th international conference on String Processing and Information Retrieval, 2005, pp. 376–387. - https://doi.org/10.1007/11575832_42
995. Salmela L., Tarhio J., Kytojoki J. Multi pattern string matching with q-grams. Journal of Experimental Algorithms, 2006, Vol. 11, pp. 1-19
996. Udi Manber, Sun Wu. "Fast text search allowing errors." Communications of the ACM, 35(10): pp. 83–91, October 1992, doi:10.1145/135239.135244.
997. R. Baeza-Yates and G. Navarro. A faster algorithm for approximate string matching. In Dan Hirchsberg and Gene Myers, editors, Combinatorial Pattern Matching (CPM'96), LNCS 1075, pages 1–23, Irvine, CA, June 1996.
998. G. Myers. "A fast bit-vector algorithm for approximate string matching based on dynamic programming." Journal of the ACM 46 (3), May 1999, 395–415.
999. Navarro G., Raffinot M. A Bit-parallel Approach to Suffix Automata: Fast Extended String Matching. In Proc CPM'98, Lecture Notes in Computer Science 1448: 14-33, 1998.
1000. Navarro G., Raffinot M. Fast and flexible string matching by combining bit-parallelism and suffix automata. ACM Journal. Experimental Algorithmics,2000, 5(4): 1-36
1001. Peltola H., Tarhio J. Alternative Algorithms for Bit-Parallel String Matching. In String Processing and Information Retrieval, Spire Springer, LNCS 2857, pp. 80-93, 2003.
1002. Branislav Durian, Jan Holub, Hannu Peltola and Jarma Tarhio,"Tuning BNDM with q-grams", In the proc. Of workshop on algorithm engineering and experiments, SIAM USA, pp. 29-37, 2009.
1003. Miao C., Chang G., Wang X. Filtering Based Multiple String Matching Algorithm Combining q-Grams and BNDM. In ICGEC '10: Proceedings of the 2010 Fourth International Conference on Genetic and Evolutionary Computing, 2010, pp. 82–585. - https://doi.org/10.1109/ICGEC.2010.149
1004. Faro S., Lecroq T. Efficient variants of the backward-oracle-matching algorithm. International Journal of Foundations of Computer Science, vol. 20, no. 6, pp. 967–984, Dec. 2009,
1005. Peltola H., Tarhio J. (2003) Alternative Algorithms for Bit-Parallel String Matching. In: Nascimento M.A., de Moura E.S., Oliveira A.L. (eds) String Processing and Information Retrieval. SPIRE 2003. pp. 80-93. Lecture Notes in Computer Science, vol 2857. Springer, Berlin, Heidelberg.
1006. M. Oguzhan Külekci, "Filter based fast matching of long patterns by using SIMD instructions," in Proceedings of the Prague Stringology Conference, Prague, Czech Republic, 2009. pp. 118--128
1007. M. Oguzhan Külekci, "A method to overcome computer word size limitation in bit-parallel pattern matching," in Proceedings of the 19th International Symposium on Algorithms and Computation, ISAAC, 2008. pp. 496--506
1008. Gupta S., Rasool A. Bit Parallel String Matching Algorithms: A Survey. International Journal of Computer Applications, 2014, vol. 95, No 10, pp. 27-32
1009. M. Crochemore, A. Czumaj, L. Gaĺ˘gsieniec, T. Lecroq, W. Plandowski, and W. Rytter, "Fast practical multi-pattern matching," Information Processing Letters, vol. 71, no. 3-4, pp. 107-113, Aug 27 1999.
1010. G. Navarro, Nrgrep: A fast and flexible pattern matching tool. Software—Prac-

tice & Experience, Vol. 31, No 13, 2001, pp. 1265–1312. - https://doi.org/10.1002/spe.411.

1011. F. Franek, Jennings, C. G., and Smyth, W. F., "A simple fast hybrid pattern-matching algorithm," J. Discret. Algorithms, pp. 682-695, 2007.
1012. S. Deusdado and P. Carvalho, "GRASPm: an efficient algorithm for exact pattern-matching in genomic sequences," Int J Bioinform Res Appl, vol. 5, no. 4, pp. 385-401, 2009.
1013. P. Shivendra Kumar, H. K. Tiwari, and P. Tripathi. "Hybrid approach to reduce time complexity of string matching algorithm using hashing with chaining." In Proceedings of International Conference on ICT for Sustainable Development, pp. 185-193. Springer, Singapore, 2016.
1014. Hamming R. W. "Error detecting and error correcting codes". The Bell System Technical Journal. 1950, 29 (2): 147–160
1015. Levenshtein V.I. Binary codes with correction of dropouts, insertions and substitutions of symbols (RUS). Reports of the Academy of Sciences of the USSR, 1965. 163.4: 845-848.
1016. Levenshtein, Vladimir I. (February 1966). "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady, 1966. 10 (8): 707–710.
1017. Dan Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology ACM SIGACT News, Vol. 28, No 4, Dec. 1997, pp. 41–60. - https://doi.org/10.1145/270563.571472
1018. Damerau F.J. A technique for computer detection and correction of spelling errors. Communications of the ACM, 1964, vol. 7, No 3, pp 171–176
1019. Winkler, W. E. (1990). "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage" (PDF). Proceedings of the Section on Survey Research Methods. American Statistical Association: 354–359.
1020. Jaro, M. A. Advances in record linkage methodology as applied to the 1985 census of Tampa Florida Journal of the American Statistical Association. 1989, Vol. 84, No. 406, pp. 414-420
1021. Longest common subsequence problem. - https://en.wikipedia.org/wiki/Longest_common_subsequence_problem
1022. Hall P., Dowling G. Approximate string matching. ACM Computing Surveys, 12(4) :381-402, 1980.
1023. Sankoff D., Kruskal J., editors. Time Warps. String Edits, arid Macro molecules: The Theory arid Practice of Sequence Comparison. Add is on-Wesley, 1983.
1024. Apostolico A., Galil Z. Combinatorial Algorithms on Words. NATO ISI Series. Springer-Verlag, 1985.
1025. Galil Z., Giancarlo R. Data structures and algorithms for approximate string matching. Journal of Complexity, Vol. 4, No 1, 1988, pp. 33-72
1026. Jokinen P, Tarhio J, Ukkonen E. A comparison of approximate string matching algorithms. Software Practice arid Experience, 26(12): 1439-1458,1996.
1027. Navarro G. A guided tour to approximate string matching. ACM Computing Surveys, Vol. 33, No , 1, 2001, pp 31–88
1028. Syeda Shabnam Hasan, F. Ahmed, Rosina Surovi Khan. Approximate String Matching Algorithms: A Brief Survey and Comparison. International Journal of Computer Applications, 2015, Vol. 120, No. 8, pp. 26-31
1029. Licklider J.C.R. Libraries of the future. Cambridge, MA: The MIT Press; 1965.
1030. Charles P. Bourne, Trudi Bellardo Hahn. A History of Online Information Services, 1963-1976. MIT Press, 2003, 496 p
1031. Project Gutenberg. - https://en.wikipedia.org/wiki/Project_Gutenberg
1032. Schatz B. (1996). Chen H. (ed.). "Building large-scale digital libraries". IEEE Computer. 29 (5): 22–25.
1033. Functional Requirements for Bibliographic Records, Final Report / IFLA Study Group on the Functional Requirements for Bibliographic Records. – München: K.G. Saur, 1998.
1034. Crofts N., Doerr M., Gill T., Stead S.,, Stiff M. (editors), Definition of the CIDOC Conceptual Reference Model, January 2008. Version 4.2.4.
1035. CERIF in Brief. - https://www.eurocris.org/eurocris_archive/cerifsupport.org/cerif-in-brief/index.html

1036. David Shotton. Introduction the Semantic Publish-ing and Referencing (SPAR) Ontologies. October 14, 2010. http://opencitations.wordpress.com/ 2010/10/14/introducing-the-semantic-publishing-and-referencing-spar-ontologies/
1037. Candela L., Castelli D., Fuhr N., Ioannidis Y., Klas C.-P., Pagano P., Ross S., Saidis C., Schek H.-J., Schuldt H., Springmann M. Current Digital Library Systems: User Requirements vs Provided Functionality. IST-2002- 2.3.1.12. Technology-enhanced Learning and Access to Cultural Heritage. March 2006.
1038. Candela L., Castelli D., Ioannidis Y., Koutrika G., Pagano P., Ross S., Schek H.J., Schuldt H. Setting the foundations of digital libraries: the DELOS manifesto. D-Lib Mag. 2007;13(3/4)
1039. Candela L., Castelli D., Dobreva M., Ferro N., Ioannidis Y., Katifori H., Koutrika G., Meghini C., Pagano P., Ross S., Agosti M., Schuldt H., Soergel D. The DELOS Digital Library Reference Model Foundations for Digital Libraries. IST-2002-2.3.1.12. Technology-enhanced Learning and Access to Cultural Heritage. Version 0.98, December 2007.
1040. Goncalves M.A., Fox E.A.., Watson L.T. and Kipp N.A. Streams, structures, spaces, scenarios, societies (5S): A formal model for digital libraries. ACM Transactions on Information Systems. 22(2), 2004, p. 270–312.
1041. Isah A., Serema B. C., Mutshewa A., Kenosi L. (2013). "Digital Libraries: Analysis of Delos Reference Model and 5S Theory". Journal of Information Science Theory and Practice. 1 (4): 38–47
1042. Open Archives Initiative Protocol for Metadata Harvesting. - https://en.wikipedia.org/wiki/Open_ArchivesInitiative_Protocol_for_Metadata_Harvesting

Отримано: 17.06.2022

Про автора:

Резніченко Валерий Анатолієвич,
кандидат фізико-математичних наук,
заступник завідувача відділом.
Кількість публікацій
в українських виданнях– 61.
Кількість зарубіжнихих публікацій – 4.
Індекс Хірша – 12.
http://orcid.org/0000-0002 4451-8931.

Місце роботи автора:

Інститут програмних систем
НАН України, 03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559.
E-mail: reznich@isofts.kiev.ua

ПРОБЛЕМИ ПРОГРАМУВАННЯ.
2022 – № 2

УДК 004.93’1

Модель та програмне забезпечення для інерційного вимірювального пристрою / С.О. Безпалько, В.М. Шимкович, А.Ю. Дорошенко.

Розроблено модуль стабілізації кута нахилу площини з трьома ступенями свободи, програмне забезпечення для отримання даних з MPU-6050 використовуючи протокол I2C, розроблено програмну реалізацію цифрового пропорційно-інтегрально-диференціального регулятора з алгоритмом автоматичного підбору коефіцієнтів. Побудовано модель гіроскопічного пристрою для проведення тестування створеного рішення. Виходячи з часових та ресурсних обмежень, було обрано ПІД регулятор для керування напругою на двигуні, ШІМ як спосіб створення керуючого сигналу, фільтр Калмана для фільтрування даних інерційного вимірювального пристрою та релейний метод у парі з методом Зіглера-Нікольса для підбору коефіцієнтів регулятора. Результати тестування дали наступні характеристики перехідного процесу: час перехідного процесу - 0.44с, перерегулювання - 6.2%. Подальше поліпшення якості перехідного процесу можливе за рахунок використання більш високоякісних інерційних вимірювальних пристроїв, високоточних електродвигунів із великим моментом кручення.

Ключові слова: гіроскопічна система, модуль стабілізації, інерційний вимірювальний пристрій, 6 – осьова інерційна вимірювальна система, регулятор, широтно-імпульсна модуляція, I2C.

UDC 004.93’1

A model and software for the inertial measurement unit / S. Bezpalko, V. Shymkovysh, A. Doroshenko.

The module of stabilization of an inclination of the plane with three degrees of freedom, the software for data acquisition from MPU-6050 using the I2C protocol, and the program of realization of the digital proportional-integral-differential regulator with algorithm of subtomatic factor is developed. A model of a gyroscopic device for testing the created solution was built. Considering time and resource constraints, the following solutions were chosen: PID controller for motor voltage control, PWM as a way to create a control signal, Kalman filter for processing output of the IMU, device and relay method in pairs with the Ziegler-Nichols method for the selection of the coefficients of the regulator. Test results or the following characteristics of the transition process: the time of the transition process - 0.44s, overregulation - 6.2%. Further improvement of the quality of the transient process is possible through the use of high-quality inertial measuring devices, high-precision electric motors with high torque.

Keywords: gyroscopic system, stabilization module, inertial measuring unit, 6 axis inertial measurement system, controller, pulse width modulation, I2C.

УДК 004.43::004.94

Транслятор мови візуального програмування Петрі-об'єктних моделей / А.Ю. Дифучин.

Високорівневі засоби програмування спрямовані на підвищення швидкості розробки складних програм за рахунок автоматизації рутинних дій програміста, зменшення кількості помилок при кодуванні та зменшення коду в цілому. Візуальне програмування передбачає кодування на основі візуального представлення завдання на виконання обчислень. Транслятор мови візуального програмування Петрі-об'єктних моделей, який розроблений, побудований відповідно до визначеної формальної граматики мови. Транслятор виконує перетворення візуального представлення Петрі-об'єктної моделі у текстову мову програмування та запускає на обчислення. Лексичний аналіз виконується під час створення візуального представлення моделі у клієнтському застосуванні. Семантичний аналіз і виконання обчислень моделі виконується серверним застосуванням. Наведено приклад, в якому представлено всі етапи перетворення віуального представлення моделі транслятором мови програмування. Перевагами розробленої мови є невеликий алфавіт символів, реалізація можливості для тиражування об'єктів та зв'язків між ними, універсальність застосування для розробки моделей дискретно-подійних систем.

Ключові слова: транслятор, формальна граматика, стохастична мережа Петрі, алгоритм імітації, Петрі-об'єктна модель.

UDC 004.43::004.94

The translator of Petri-object model visual programming language / A. Dyfuchyn.

High-level programming tools are aimed at increasing the development speed of complex programs due to automation within the routine actions of the programmer, reducing the number of coding errors and reducing the code in general. Visual programming involves coding based on a visual representation of a task for the computational process instead of a textual one. The translator of the visual programming language of Petri-object models is designed and built according to the defined formal grammar of the language and Petri-object model formalism. It performs the transformation of the visual representation of the model into the calculation of the simulation algorithm. The advantages of the developed language are a small alphabet of symbols, the opportunity for replication of objects and connections between them, and the versatile application for the discrete-event systems models development.

Keywords: visual programming, translator of the visual programming language, Petri-object model, formal grammar of the language, alphabet of symbols.

УДК 502:004.45 (075.8)

Дисертація магістрів з інженерії програмного забезпечення - об′єкт, предмет, зміст досліджень / М.О. Сидоров.

На відміну від закордонних університетів, в Україні протягом багатьох років склалася й існує практика формулювання мети дослідження, застосовуючи поняття об’єкта і предмета дослідження. І це, як показав час, важливий етап роботи, від виконання якого залежить результативність дослідження. Загальне, тотальне проникнення програмного забезпечення в життя з одного боку, а з іншого - поява спеціалізацій інженерії програмного забезпечення значно ускладняють умови виконання цього етапу магістерського дисертаційного дослідження. Окрім цього, в інженерії програмного забезпечення застосовуються наукові методи досліджень, які доцільно мати на увазі і використовувати для виконання магістерського дисертаційного дослідження. Тому актуальними є питання щодо формулювання об’єкта і предмета досліджень та застосування доказових методів їх виконання. Метою цієї статті було надання рекомендації на основі фундаментальних положень інженерії програмного забезпечення щодо формулювання об’єкта і предмета магістерської дисертації зі спеціалізації в контексті інженерії програмного забезпечення. А також, вказати на застосування в магістерської дисертації наукових методів доказових досліджень і звернути увагу на оформлення результатів. Досягнення мети розглядається в контексті причин і умов, які історично склалися в навчанні, і відповідних проблем, що виникли в навчальних планах, виборі тем і змісту досліджень магістерських дисертацій. Одна з причин, яка має об’єктивний характер, виникає за сполучення домену інженерії програмного забезпечення з прикладними доменами, що є основою відповідної спеціалізації. Стаття розрахована на магістрів зі спеціалізацій інженерії програмного забезпечення та їхніх наукових керівників.

Ключові слова: інженерія програмного забезпечення, навчання, магістерські дослідження, науково-експериментальні методи досліджень.

UDC 502:004.45 (075.8)

Master′s Thesis in Software Engineering – Object, Subject, Content of Research / M. Sydorov.

Nowadays scientific researches In Ukraine are based on formulating the goals of research, using the concept of object and subject of research. Moreover, as time has shown, this is an important stage of work on which the effectiveness of the study depends. Both the total use of a software in different arias of life and improvement of software engineering prepare better conditions for this stage of the dissertation research. Modern software engineering uses scientific research methods that should be used to perform dissertation research. The purpose of this article is the following: to provide recommendations for the formulation of the object and subject of the master’s thesis. Special phocus is made on specialization in the software engineering fundamentals. It also aimed at describing the scientific methods of evidence-based research in the master’s thesis. It pays attention to the results presentation. These points are considered in the context of the conditions that have developed in education and the relevant problems that have arisen in the Universities′ curriculum. There are some problems in choosing the topics of master′s thesis. The one is objective in nature and arises from the combination of the domain of software engineering with application domains in a context of specialization. The article is aimed at masters in software engineering and their supervisors.

Keywords: software engineering, education, master’s research, scientific and experimental research methods.

УДК 004.422

Інтеграція та композиція сервісів на основі моделі / Ю. А. Дивак, Т. Мамедов.

У цій роботі я хотів би представити новий підхід для розробки застосунку і архітектури для роботи із семантичними веб-сервісами, який інтегрує декілька інтерпрайзів та поєднує результати з різних сервісів через застосування технік, методологій та анотацій. Підхід формулюється разом із розробкою програмного забезпечення та моделюванням бізнес процесів. Ми пропонуємо застосовувати існуючі техніки моделювання бізнес процесів (BPMN, BPLWS, OWL-S, WSDL, WebML) для моделювання процесів, які проходять через декілька застосунків, для інтеграції програмних систем та веб-сервіс композиції. Головна мета - знайти методологію для розробки семантично багатих веб-застосунків із напівавтоматичним визначенням семантичного опису сервісу з моделі бізнес процесів. Це підвищить якість проєктування та зменшить обсяг додаткової роботи у розробці застосунків з обробкою семантично анотованої інформації, що проходить через підприємства.

Ключові слова: Інтеграція програмних систем, Композиція веб-сервісів, Моделювання бізнес процесів, Семантичні веб-сервіси, Розробка через моделювання, Методологія.

UDC 004.422

Integration and service composition based on model / Y. Dyvak, T.Mamedov.

In the paper we would like to introduce a framework for designing and developing Semantic Web Service application that integrates a several enterprises and composes results from a different services by applying techniques, methodologies and notations provided by Software Engineering and Business process modeling. We propose to use existing business modeling techniques (BPMN, BPLWS, OWL-S, WSDL, WebML) for modeling the cross enterprise processes, for enterprise integrations and web service compositions. The main purpose is to designing and developing semantically rich Web applications, with semiautomatic defining semantic descriptions from the business process model that increase the efficiency of design and decrease the extra work necessary for processing semantically annotated information which passing through the enterprises.

Keywords: Integrations of Program Systems, Web Service Composition, Business Process modeling, Semantic web services, Model Driven Design, Methodologies.

УДК 004.031’43

Веб-сервіс для систем управління ресурсами підприємства / А.Д. Тарасенко, А.Ю. Дорошенко.

Розроблено веб-сервіс для систем управління ресурсами підприємства, що дозволяє створювати та управляти швидкими захищеними системами управління ресурсами підприємства на основі необхідних користувачеві модулів. Унікальність сервісу полягає в тому, що окрім того, що в ньому присутня більшість вже існуючих стандартів та можливостей інших ERP систем від інших IT-компаній, він має суттєву перевагу – в його використанні відсутня необхідність звертання до IT-спеціалістів задля розробки та налаштування ERP системи під потреби кінцевого користувача. Здійснено розробку головного веб-сайту веб-сервісу, який передбачає автентифікацію та авторизацію; додатку інсталятору для створення і запуску ERP системи; додаток серверу ERP системи та клієнтського додатку для взаємодії із сервером ERP системи. Проведено функціональне тестування розробленого веб-сервісу шляхом тестування кожного додатку окремо та у спільному взаємозв’язку. Завдяки концепції модульності, дана система легко доповнюється та розширюється. В перспективі сервіс можна доповнювати новим функціоналом залежно від вимог бізнесу.

Ключові слова: система управління ресурсами підприємства, ERP система, модульність, мережева взаємодія.

UDC 004.031’43

Web service for enterprise resource management systems / A. Tarasenko, A. Doroshenko.

A web service for enterprise resource management systems has been developed, which allows creating and managing fast secure enterprise resource management systems based on user-required modules. The uniqueness of the service is that in addition to the fact that it contains most of the existing standards and capabilities of other ERP systems from other IT companies, it has a significant advantage - there is no need to address IT-specialists for the development and configuration of ERP systems for the needs of the end user in its use. The main website of the web service has been developed, which provides authentication and authorization; installer application for creating and running ERP system; ERP server application, and client application for interaction with ERP system. Functional testing of the developed web service was carried out by testing each application separately and in a joint relationship. Thanks to the concept of modularity, this system is easily supplemented and expanded. In the future, this service can be supplemented with new functionality, depending on business requirements.

Keywords: management system, enterprise resources, ERP system, modularity, network interaction.

УДК 004.94

60 років базам даних / В.А.Резніченко.

Наводиться огляд досліджень і розробок баз даних з моменту їх виникнення в 60-х роках минулого століття і по сьогодні. Виділяються наступні етапи: виникнення і становлення, бурхливий розвиток, епоха реляційних баз даних, розширені реляційні бази даних, пост-реляційні бази даних і великі дані. На етапі становлення описуються системи IDS, IMS, Total і Adabas. На етапі бурхливого розвитку висвітлені питання архітектури баз даних ANSI/X3/SPARC, пропозицій КОДАСИЛ, концепції і мов концептуального моделювання. На етапі епохи реляційних баз даних наводяться результати наукової діяльності Е. Кодда, теорія залежностей і нормальних форм, мови запитів, експериментальні дослідження і розробки, методи оптимізації та стандартизації, управління транзакціями. Етап розширених реляційних баз даних присвячений опису темпоральних, просторових, дедуктивних, активних, об'єктних, розподілених та статистичних баз даних, баз даних масивів, а також машин баз даних і сховищ даних. На наступному етапі розкрита проблематика постреляційних баз даних, а саме, NOSQL-, NewSQL- і онтологічних баз даних. Шостий етап присвячено розкриттю причин виникнення, характерних властивостей, класифікації, принципів роботи, методів і технологій великих даних. Нарешті в останньому розділі подається короткий огляд досліджень і розробок із баз даних у колишньому Радянському Союзі.

Ключові слова: Типи баз даних: ієрархічна, мережева, реляційна, навігаційна, темпоральна.

UDC 004.94

60 Years of Databases / V. Reznichenko.

The article provides an overview of research and development of databases since their appearance in the 60s of the last century to the present time. The following stages are distinguished: the emergence formation and rapid development, the era of relational databases, extended relational databases, post-relational databases and big data. At the stage of formation, the systems IDS, IMS, Total and Adabas are described. At the stage of rapid development, issues of ANSI/X3/SPARC database architecture, CODASYL proposals, concepts and languages of conceptual modeling are highlighted. At the stage of the era of relational databases, the results of E. Codd's scientific activities, the theory of dependencies and normal forms, query languages, experimental research and development, optimization and standardization, and transaction management are revealed. The extended relational databases phase is devoted to describing temporal, spatial, deductive, active, object, distributed and statistical databases, array databases, and database machines and data warehouses. At the next stage, the problems of post-relational databases are disclosed, namely, NOSQL-, NewSQL- and ontological databases. The sixth stage is devoted to the disclosure of the causes of occurrence, characteristic properties, classification, principles of work, methods and technologies of big data. Finally, the last section provides a brief overview of database research and development in the former Soviet Union.

Keywords: Database types: hierarchical, network, relational, navigational, temporal, spatial. spatio-temporal, spatio-network, moving objects, deductive, active, object-oriented, object-relational, distributed, parallel, arrays, statistical, multidimensional, database machines, data warehouse, NoSQL, key-value , triple store, column-oriented, document- oriented, graph-oriented, multimodal, cloud, scientific, multi-valued, XML, NewSQL, ontological, Big Data.

ДО УВАГИ АВТОРІВ!

У журналі «Проблеми програмування» публікуються наукові матеріали, які раніше не публікувалися в інших виданнях.

Мова статті: українська, англійська.* Обсяг статті - від 6 до16 сторінок формату А4.

Документ зберігається у форматі doc або docx. Ім'я подається транслітерацією, як прізвище автора (авторів), наприклад, "Petrenko.doc".

Автори можуть користуватися електронною поштою і також телефаксом для ділової переписки та передачі до редакції тексту статті та правки при коректурі. E-mail редакції: alengoro@isofts.kiev.ua. FAX: +380 (44) 526 6263, Телефон: 526 5065.

1. Оформлення файлу з текстом статті.

При підготовці файлу використовуються: стиль нормальний (звичайний) або normal; шрифт Times New Roman, розмір шрифта 12 пт.; міжрядковий інтервал – 1,0; абзацний відступ -1,25 см; вирівнювання – по ширині. У тексті не допускається вирівнювання пропусками; розстановка переносів – автоматична. Формат паперу А4, розміри полів документа – 20 мм. Текст статті після анотації має бути оформлений **у 2 колонки,** ширина яких – 7,86 см, а пробіл між ними – 1,27 см.

2. Послідовність розміщення та оформлення матеріалу статті.

УДК: індекс за універсальною десятковою класифікацією.

Автори: ініціали та прізвища авторів, курсив (світлий).

Заголовок 1 **(назва статті):** не містить абревіатур та строго відповідає змісту статті. Шрифт 15 пт, напівжирний, регістр верхній.

Анотація **(мовою статті):** 50-100 слів, не містить абревіатур, зрозумілих із змісту статті. Шрифт 10 пт, звичайний.

Ключові слова **(мовою статті):** не більше10 слів, не містить абревіатур, зрозумілих із змісту статті, подаються в називному відмінку, розділені комами. Шрифт 10 пт, звичайний.

Заголовок 2 **(назва розділу):** шрифт 14 пт, напівжирний; абзац із центральним вирівнюванням, без переносів. Заголовки нижчого рівня (пункти і т.п.) у самостійний абзац не виділяються і проходять першим реченням текстового абзацу, шрифт 12 пт, напівжирний.

Основний текст статті має такі необхідні елементи:

постановка проблеми в загальному вигляді і її зв'язок з важливими науковими або практичними завданнями;

аналіз останніх досліджень і публікацій, у яких розпочато рішення даної проблеми і на які спирається автор, виділення невирішених раніше частин загальної проблеми, яким присвячується дана стаття;

формулювання цілей статті (постановка задачі);

виклад основного матеріалу дослідження з повним обґрунтуванням отриманих наукових результатів;

висновки з даного дослідження і перспективи подальших розробок у даному напрямку;

подяка (за наявності такої).

Формули створюються в редакторі Microsoft Equation 3.0 або MathType. Формули, на які є посилання в тексті, повинні мати наскрізну нумерацію. Номер формули друкується в круглих дужках біля краю правого поля. Розмір основного шрифту редактора формул – 12 пт. Розміри символів у формулах: звичайний – 12 пт, великий індекс – 9 пт, дрібний індекс – 7 пт, великий символ – 18 пт, дрібний символ – 11пт. Не допускається масштабування формульних об´єктів.

Рисунки мають бути створені вбудованим редактором Word Picture або експортовані з прикладних програм Windows у графічних форматах (bmp, pcx, gif, jpg або tif). Рисунки розташовуються по центру. Нумерація рисунків здійснюється відповідно до порядку

згадування у тексті. Нумеровані підписи розміщуються під рисунком з позначенням «Рис. », далі вказується номер рисунка і текст підпису.

Таблиці мають бути підготовлені стандартним вбудованим в Word інструментарієм "Таблиця". Таблиці нумеруються за порядком згадування. На номер таблиці повинно бути посилання в тексті. Номер таблиці вказується в окремому рядку з вирівнюванням по правій стороні (наприклад, «Таблиця 1»). Назви таблиць розміщуються над таблицею з вирівнюванням по центру. Мінімальний розмір шрифту в таблицях – 11 пт.

Література*:** нумерований список джерел згідно ДСТУ 8302:2015 від 01.07.2016 р., шрифт 11 пт, відступ: спеціальний, навислий, 0,63 см. Джерела з заголовками на латиниці наводяться без перекладу. Інші джерела подаються мовою оригіналу. Приклади оформлення бібліографічних посилань згідно з вимогами ***Harvard Style наведені в багатьох публікаціях, наприклад: http://www.staffs.ac.uk/assets/harvard_referencing_examples_tcm44-39847.pdf

***Дані про авторів*:** мають починатися рядком "Про авторів:", напівжирний курсив. Далі вказуються для кожного з авторів ПІБ повністю, наукове звання, посада, адреса, кількість публікацій в українських виданнях (приблизна), кількість публікацій в зарубіжних індексованих виданнях (приблизна), індекс Хірша (за наявності), обов'язково номер ORCID (сайт ORCID http://orcid.org/).

***Дані про місце роботи авторів*:** починаються рядком "Місце роботи авторів:", напівжирний курсив. Далі вказуються місце роботи, адреса, телефон, факс, електронна пошта, контактний телефон.

3. Оформлення файлу з анотаціями.

Файл з анотаціями містить інформацію двома мовами (наприклад, якщо стаття написана українською мовою, то анотація та ключові слова – англійською і українською мовами) та має бути оформлений у дві колонки: УДК (шрифт – 8 пт); назва статті (шрифт – 12 пт, напівжирний); прізвища та ініціали авторів (шрифт – 12 пт); текст анотації, ключові слова (шрифт – 10 пт).

Вимоги до анотації англійською мовою: обсяг від 100 до 250 слів, інформативність, оригінальність (не є калькою української анотації), змістовність (відображає основний зміст статті і результати досліджень), структурованість (дотримується логіки опису результатів у статті).

Документ зберігається у форматі doc або docx. Ім´я подається транслітерацією, як прізвище автора (авторів), наприклад, "Petrenko_Annot.doc".

*16.07.2020 р. набули чинності положення Закону України «Про забезпечення функціонування української мови як державної». Відповідно до статті 22 «Державна мова у сфері науки» у наукових виданнях не повинно бути вміщено матеріалів іншими мовами, окрім державної, англійської та мов ЄС.

Примітка: Підписний індекс журналу «Проблеми програмування» – **90853**.