Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: **«МЕТОД ТА ПРОГРАМНИЙ ЗАСІБ ЗАСТОСУВАННЯ МЕТАДАНИХ В ПРОЦЕСАХ ПОШУКУ»**

Виконав: студент 2 курсу, групи 1КІ-18м
спеціальності
123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Тягун Д. Т.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ОТ, Савицька Л. А.
(прізвище та ініціали)
Рецензент: к. т. н., доц. каф.МБІС, Поплавський А. В.
(прізвище та ініціали)

м. Вінниця - 2019 рік

АНОТАЦІЯ

Дана магістерська кваліфікаційна робота присвячена розробці та програмній реалізації методу застосування метаданих в процесах пошуку. Цей програмний засіб дозволить досягти збільшення результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів.

Високий рівень вирішення поставленої задачі досягнуто за рахунок використання сучасної мови програмування Java.

В даній магістерській роботі виконано аналіз сучасних моделей, методів та засобів застосування метаданих в процесах пошуку; розглянуто існуючі аналоги та поточний стан технологій в галузі застосування метаданих в процесах пошуку; запропоновано модель та метод застосування метаданих в процесах пошуку; розроблено ключові процеси роботи методу застосування метаданих в процесах пошуку та на його основі цього створено програмний засіб; запропоновано програмну реалізацію запропонованого методу застосування метаданих в процесах пошуку; проведено тестування програмного продукту та виконано аналіз отриманих результатів.

ANNOTATION

This master's qualification work is devoted to the development and program implementation of the method of applying metadata in the search process. This software tool will allow you to increase the search results of documents that satisfy the request, within a certain static collection of documents.

The high level of the solution to this task was achieved through the use of modern Java programming language.

In this master's thesis an analysis of modern models, methods and means of using metadata in the search processes; examines the existing analogs and the current state of technology in the application of metadata in the search process; the model and method of application of metadata in search processes is proposed; the key processes of the method of using metadata in the search processes have been developed and, based on this, a software tool was created; the program realization of the proposed method of using metadata in the search processes is proposed; Software product testing was carried out and analysis of the results was performed.

ЗМІСТ

					08-23.МКР.014.00.000 ПЗ		
Змн.	Лист	№ докум.	Підпис	Дата			
Розроб.		Тягун Д. Т.			Метод та програмний засіб застосування метаданих в процесах пошуку	Літ.	Арк. 6 / Аркушів 162
Перевір.		Савицька Л. А..					
Реценз.		Поплавський А. В.				ВНТУ, гр. 1КІ-18М	
Н. Контр.		Швець С. І.					
Затверд.		Мартинюк Т. Б.					

ВСТУП

Актуальність теми дослідження. Стрімкий розвиток інформаційних технологій вже не новина, подальша трансформація постіндустріального суспільства тісно пов'язана з набуттям нових знань та ефективним керуванням тими, що вже набуті [1]. Такі інтелектуальні прориви є цілком природніми після появи постіндустріального суспільства.

Цілком очевидним є те, що підвищення ролі знання веде до стрімкого збільшення інформаційних потоків, які людина отримує щодня. За умов ігнорування частини цього потоку, є ризик втрати актуалізації своїх знань. І навпаки, за умов вмілого та ефективного керування потоками інформації дозволяє зберігати та підвищувати професійну кваліфікацію, просуватися вперед у щоденних справах, створювати новий контент тощо. Закономірно, що рано чи пізно, на якомусь етапі інформаційний потік може стати настільки величезним, що навіть при великому бажанні, і витрачаючи на освоєння знань більшу частину свого часу, людина не встигатиме – бо новий контент буде з'являтись швидше, ніж попередній може бути опрацьованим в принципі [2]. Часто нові технології так і не впроваджуються у життя, чи не виходять на ринок, оскільки встигають морально застаріти, і головною причиною такого стану справ було те, що системи пошуку не встигли їх з різних причин адекватно проіндексувати і ці матеріали так і залишилися незнайденими в безкінечних множинах інформаційних потоків [3-4].

Нині дуже популярними є підтримка наукових здобутків науково метричними базами даних. Наука наукометрія дозволяє оцінити глибину науковості статей, дати об'єктивну оцінку новій інформації в тій чи іншій галузі. Наприклад, на відому конференцію ACM Conference on Human Factors in Computer Systems, щороку приходять наукові матеріали на рецензування і кількість їх зросла вдвічі за останні 10 років. Але це лиш ті матеріали, які пройшли рецензування. Якщо ж виміряти ту кількість статей, що поступають на розгляд комісії конференції, то виявимо, що вона виросла в п'ять разів, тобто, реальний приріст інформаційного потоку в тільки одній галузі становить

більше ніж в 5 разів за 10 років [5]. Такий стрімкий, майже експоненційний ріст потоку нових наукових розробок означає, що в адекватні часові терміни практично мало хто може встигнути ознайомитись із цими матеріалами, що й призводить до того, що гіпотетично перспективні технології навіть не були помічені ні науковим суспільством в цілому, ні гіпотетичним ринком користувачів.

Це формує вже досить давно **актуальну задачу** ефективного пошуку інформації в спеціалізованих системах, архівах, і в тому числі, і в мережі Інтернет. Звісно, потужні пошукові системи Google [6], або CiteSeer [7] дозволяють значно полегшити людині пошук і оцінку релевантності знайдених результатів. Однак, на сучасному етапі розвитку досі необхідний науковий та творчий пошук якщо не цілковито нових підходів, то можливо нетривіального використання уже існуючих, що дозволили б покращити ситуацію.

В своїх наукових працях В. Буш аналізував процес мислення людини, він вважав, що дані організуються в класи і підкласи і водночас створюють асоціативні ланцюжки, які він порівнював із т.з. кодовою таблицею [2]. В. Буш передбачав формування trail blazers – реконструкторів цих асоціативних ланцюжків. За допомогою реконструкторів інші люди можуть бачити ланцюжки асоціацій. За таким принципом, власне, і побудовані сучасні пошукові системи. А в якості т.з. реконструкторів служать спеціалізовані програмні засоби, які аналізують тестові дані, виконують їх класифікацію, створюючи індексні масиви, що потім вже і використовуються для вибірки даних на релевантність.

З метою полегшити процеси пошуку і виключити аналіз всього тексту в пошуках ключових слів та виразів, в даній роботі пропонується використовувати так звані «мета-теги», які в стислій формі описують вміст сторінки, текстових даних чи якогось ресурсу. Таким чином, кожна веб-сторінка, чи документ, супроводжуються «хмарою тематичних тегів». Звісно їх кількість, на жаль, кінцева, а із швидким ростом інформаційного потоку, на кожен такий тег припадатиме все більше і більше даних, а це, відповідно, вимагатиме додаткових уточнюючих тегів, і зрештою, кожен документ треба

буде супроводжувати такою кількістю тегів, що їх індексація за своїм обсягом не буде відрізнятись від повнотекстової інформації.

Цю задачу можна вирішити двома способами. Перший підхід полягає у відмові від жорсткої системи тегів та перехід до т.з. семантичного аналізу текстових даних та автоматизованої їх класифікації [8]. Проте, такий підхід має суттєвий недолік, що полягає у автоматичному визначенні ключових фрагментів текстових даних, що можуть скласти основу для запуску прцесу класифікації. Професор Янг з університету Карнегі-Меллона [9] займається порівнянням та аналізом методів класифікації текстів. В рамках своїх досліджень він поділив їх на такі категорії:

1) застосування векторних машин. Це побудова простору таких рішень, які дозволяли би точно класифікувати документи, проте у випадку рідкісних категорій чи двозначних результатів цей метод не є дієвим;

2) метод найближчого сусіда. Тут результат аналізу окремого документа порівнювався із результатами аналізу т.з. «еталонних документів», потім згідно результатів, цьому тексту проставлялась «вага». Цей метод можна реалізувати за допомогою нейромережі, однак потужна залежність від якості еталонних текстів робить його негнучким. При такому підході постійно із ростом та розвитком наукових досліджень прямо пропорційно має рости і сама база таких еталонів, а отже, подібна система класифікації даних буде постійно вимагати все більше і більше ресурсів.

3) лінійно-квадратичний аналіз за Янгом. Цей метод вирішував проблему встановлення ваг і узгодження еталонів. Замість еталонних текстів даний метод використовував вектори ключових слів із відповідними їм вагами. Це дозволяло точніше категоризувати текстові дані, і водночас, не залежати від обсягу еталонних текстів. Проте, в суміжних галузях ключові слова можуть співпадати, і звідси незадовільний результат для суміжних категорій: метод давав збій.

4) ймовірнісний метод, який мав на меті вирішити проблему суміжних категорій шляхом оцінювання ймовірності «попадання» в певну категорію,

шляхом перевірки висунутих гіпотез про «неналежність» окремого ключового виразу до інших виразів.

Альтернативним способом вирішення поставленої задачі стало обмеження по обсягу аналізованого тексту. Наприклад, в працях Нігама та Маккаллума [10] пропонується використовувати лише фрагменти текстових даних, що виділені спеціальними HTML-тегами. Майкл Брент [11] в свій час запропонував аналізувати потік дієслів, групуючи дієслова за тематикою та галуззю знань. І цей метод виявився, на диво, доволі точним, навіть на прикладах художніх текстів цей алгоритм давав лише 3% похибки.

Однак всі ці методики та методи, в будь-якому випадку, вимагали машинного та автоматичного аналізу текстів, а це означає – побудови складних інтелектуальних інформаційних систем. Такі великі затрати можуть собі дозволити потужні пошукові рушії, а для більш локальних задач (скажімо, бібліотечних систем, або баз наукових статей) вони не застосовні зовсім.

Звернімося до іншого класу підходів, що базується на застосуванні метаданих в процесах пошуку, тобто, подальшій поступовій деталізації тегів, що можуть формувати масиви доповняльних даних. Ці дані повинні містити елементи категоризації на базі логічних ланцюжків. Такі ланцюжки і отримали назву «семантичних метаданих». Це є базою для нового семантичного підходу, що є одним із перспективних напрямків розвитку Internet. Аналізу підходу застосування метаданих в процесах пошуку, його дослідженню його застосуванню для індексування користувацьких даних і присвячена дана магістерська кваліфікаційна робота.

Зв’язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри обчислювальної техніки Вінницького національного технічного університету; у рамках Національної стратегії розвитку освіти в Україні на період до 2021 р. (наказ Президента № 334/2012 від 25.06.2013 р.); плану наукової та навчально-методичної роботи кафедри обчислювальної техніки.

Метою дослідження магістерської кваліфікаційної роботи є збільшення результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів.

Для досягнення поставленої у роботі мети необхідно розв’язати такі завдання:

− провести аналіз сучасних моделей, методів та засобів застосування метаданих в процесах пошуку;

− розглянути існуючі аналоги та поточний стан технологій в галузі застосування метаданих в процесах пошуку;

− запропонувати модель та метод застосування метаданих в процесах пошуку;

− розробити ключові процеси роботи методу застосування метаданих в процесах пошуку та на його основі цього розробити програмний засіб;

− виконати програмну реалізацію запропонованого методу застосування метаданих в процесах пошуку;

− провести тестування програмного продукту та виконати аналіз отриманих результатів.

Об’єкт дослідження – процеси роботи з мета-даними.

Предмет дослідження – методи та програмні засоби застосування метаданих в процесах пошуку.

Методи дослідження. Дослідження, виконані під час роботи над кваліфікаційною магістерською роботою, ґрунтуються на теоретико-множинних підходах і принципах кросплатформеного підходу для реалізації програмної моделі керування електронними нотатками; структурному проектуванні програмного забезпечення – для реалізації кросплатформеного програмного забезпечення застосування метаданих в процесах пошуку; методах об’єктно-орієнтованого програмування – для реалізації методу застосування метаданих в процесах пошуку та розробки відповідного програмного забезпечення.

Наукова новизна одержаних результатів полягає в такому:

− вперше запропоновано метод застосування метаданих в процесах пошуку. Запропонований у даній роботі метод застосування метаданих в процесах пошуку дозволяє збільшити результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів;

− вдосконалено модель застосування метаданих в процесах пошуку, що дозволяє створення користувачем власних мета-сутностей;

− вдосконалено процес роботи з мета-даними, який дозволяє створення семантичних ланцюжків між поняттями;

− вдосконалено процес створення мета-сутностей, які дозволяють зменшувати кількість тегів у хмарі тегів.

Практичне значення одержаних результатів полягає у такому:

1. Розроблено новий метод застосування метаданих в процесах пошуку.
2. Вдосконалено процеси роботи з мета-даними та мета-сутностями, які дозволяю створення семантичних ланцюжків між поняттями та водночас зменшувати кількість тегів у хмарі тегів.
3. Вдосконалено алгоритми роботи з мета-даними та мета-сутностями, які дозволяють збільшити результати пошуку.
4. Розроблено програмний засіб застосування метаданих в процесах пошуку.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням методів та інформаційних технологій під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими, та збіжністю результатів дослідження з результатами, що отримані під час впровадження розробленого програмного засобу.

Особистий внесок магістранта. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно [12].

1 ТЕХНІКО-ЕОНОМІЧНЕ ОБГРУНТУВАННЯ

1.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Результатом магістерської кваліфікаційної роботи «Метод та програмний засіб застосування метаданих в процесах пошуку» є розробка «Memoria». Для проведення технологічного аудиту залучено трьох незалежних експертів. У нашому випадку такими експертами є: Трояновська Тетяна Іванівна (к.т.н., доцент кафедри обчислювальної техніки ВНТУ), Савицька Людмила Анатоліївна (к. т. н., доцент кафедри обчислювальної техніки ВНТУ), Кущ Дмитро Сергійович (директор ТОВ «СМЕДИЯ ГРУП»).

Оцінювання комерційного потенціалу буде здійснене за критеріями, що наведені в таблиці 1.1.

Таблиця 1.1 - Критерії оцінювання комерційного потенціалу розробки бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 1.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-тер.	0	1	2	3	4

4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатацій ні витрати значно вищі, ніж в аналогів	Експлуатацій ні витрати дещо вищі, ніж в аналогів	Експлуатацій ні витрати на рівні експлуатаційн их витрат аналогів	Експлуатацій ні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовую ться у військово-промисловом у комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 1.1

11	Термін реалізації ідеї	Термін реалізації ідеї	Термін реалізації ідеї	Термін реалізації ідеї	Термін реалізації ідеї

	більший за 10 років	більший за 5 років. Термін окупності інвестицій більше 10-ти років	від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу зведено в таблицю 1.2.

Таблиця 1.2 - Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1 – Трояновська	2 – Савицька	3 – Кущ
	Бали, виставлені експертами:		
1	4	3	4
Ринкові переваги (недоліки):			
2	3	4	3
3	4	4	4
4	4	3	4
5	3	4	3
Ринкові перспективи			
6	4	3	2
7	3	2	4
Практична здійсненність			
8	4	3	4
9	4	3	3
10	4	4	4
11	3	4	4
12	4	3	3
Сума балів	$СБ_1$=44	$СБ_2$=40	$СБ_3$=42
Середньоарифметична сума балів $\overline{СБ}$	42		

За даними таблиці 1.2 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Зважимо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 1.3.

Таблиця 1.3 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Рівень комерційного потенціалу розробки, становить 42 балів, що відповідає рівню «високий».

Тегування електронних записів, в тому числі і нотаток, що актуально для теми даної магістерської роботи, використовується з метою полегшення процесів пошуку через можливість категоризувати дані всередині електронних записів. Кожному із записів ставиться у відповідність комплект ключових слів (тегів), кожне із яких потім індексується, і отож користувач може переглядати та групувати свої електронні закладки з точки зору різних тематик, чи у відповідності певним задачам. Система тегів була згодом розвинута в модель соціального тегування, яка була покладена в основу Web 2.0.

Першопочатково модель застосування метаданих в процесах пошуку була створена для пришвидшення людського пошуку. Організація контенту за тегами і ключовими словами дозволяла людям самостійно утворювати ієрархії категорій, і отож брати на себе частку функцій та впливати на процеси застосування метаданих в процесах пошуку.

Аналоги можна розглядати 2 типи систем:

1) движки семантичної сітки (чи це можуть бути технології і мови програмування, які підтримують роботу із ними);

2) інтерфейсний прототип стандарту для створення демонстраційного програмного засобу.

Нині є такі движки, які підтримують RDF:

– R2RML Parser – модуль, в мові програмування Java. Дозволяє «розбирати» формат RDF, генерувати нові структури в форматі RDF. До того ж має хоч і обмежену, але підтримку мови запитів в межах простих мета-сутностей;

– dotNetRDF – модуль, в мові програмування C#, та ін. мовах програмування, на основі технології .NET. Призначена для маніпуляцій RDF на простому рівні, а також формулювання простих запитів;

– 4Suite – комплексний пакет роботи з форматом XML мовою програмування Python. Підтримує RDF як один із XML-форматів. Дозволяє обробляти RDF, але не може формулювати запити;

– ActiveRDF – пакет для роботи з форматом RDF для мови програмування Ruby. Вміє обробляти, але не формулювати запити. Є частиною web-движка Ruby on Rails;

– ARC RDF Store – спеціалізований пакет для роботи з форматом RDF безпосередньо з web-сторінки, написаної мовою PHP. Є частиною web-движка;

– Brahms – пакет для роботи з форматом RDF з ередовища мови програмування C++. Швидкий, здатний опрацьовувати великі мета-сутності. Має велику бібліотеку класів (див. табл. 1.4.).

З метою полегшити процеси пошуку і виключити аналіз всього тексту в пошуках ключових слів та виразів, в даній роботі пропонується використовувати так звані «мета-теги», які в стислій формі описують вміст сторінки, текстових даних чи якогось ресурсу. Таким чином, кожна веб-сторінка, чи документ, супроводжуються «хмарою тематичних тегів». Звісно їх кількість, на жаль, кінцева, а із швидким ростом інформаційного потоку, на кожен такий тег припадатиме все більше і більше даних, а це, відповідно, вимагатиме додаткових уточнюючих тегів, і зрештою, кожен документ треба буде супроводжувати такою кількістю тегів, що їх індексація за своїм обсягом не буде відрізнятись від повнотекстової інформації.

Таблиця 1.4 - Основні технічні показники аналогів

Аналоги	Платформи	Web-движок	Створення RDF	Обробка RDF	Запити
R2RML Parser	Всі	Ні	Так	Так	Так
dotNetRDF	Windows	Ні	Так	Так	Так
4Suite	Всі	Так	Так	Так	Ні
ActiveRDF	Всі	Так	Так	Так	Ні
ARC RDF Store	Всі	Так	Так	Так	Ні
Brahms	Windows, Linux	Ні	Так	Так	Так

Нині є безліч реалізацій семантичних сіток і заснованих на їх основі інтернет-сервісів. Проте, фактично немає реалізацій таких систем для побутового використання кінцевим споживачем. Тим не менш, пошук на основі мтаданих може цілком використовуватись в таких щоденних сферах життя, як, скажімо, електронні нотатки в календарі.

Центральною точкою програмної моделі є «хмара тегів», до якої звертаються як панель нотаток, так і панель мета-сутностей. Хмара тегів містить в собі всі семантичні мітки – теги, які використовуються користувачами прпід часи створенні нотаток. На панелі мета-сутностей на їх базі вибудовуються зв’язки, які, потім будуть використані в панелі пошуку.

На даний момент ведуться переговори з інвесторами для фінансування.

Просування даного продукта планується за допомогою сайту та партнерів. Потрібно розробити сайт – візитку, де будуть описані усі переваги продукта та налаштувати рекламу (Google Ads, Yandex Direct, Facebook, Instagram). Продумати фінансову мотивацію для партнерів, щоб вони просували продукт через свої канали продажу.

Ціна розробки, поки що залишається у таємці, відомо що фіксованої ціни не буде, буде підписка на продукт на місяць. Ціна підписки буде оголошена після вдалих переговір з інвесторами.

1.2 Прогнозуваня витрат на виконання наукової роботи та впровадження її результатів

Прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної робіт складається з таких етапів:

1) розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи;

2) розрахунок загальних витрат на виконання даної роботи;

3) прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

Основна заробітна плата розробника (дослідника) $З_о$ розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t \text{ [грн]}, \quad (1.1)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.

$Т_р$ – число робочих днів в місяці; приблизно $Т_р = 22$ дні;

t – число робочих днів роботи розробника (дослідника) - 62.

Для розробника із місячним посадовим окладом у 6800 грн і кількістю робочих днів у місяці – 22, заробітна плата складатиме:

$$З_о = \frac{6800}{22} \cdot 62 = 19163{,}64 \text{ (грн)}.$$

Додаткова заробітна плата розробника розраховувати як 10 % від основної заробітної плати:

$$З_д = З_о \cdot 0{,}10 \text{ [грн]}. \quad (1.2)$$

$$З_д = 19163{,}64 \cdot 0{,}10 = 1916{,}36 \text{ (грн)}.$$

Згідно діючого законодавства нарахування на заробітну плату (ЄСВ) складає 22 % від суми основної та додаткової заробітної плати розробника:

$$Н_{зп} = (З_о + З_д) \cdot 22 / 100\% \text{ [грн]}. \quad (1.3)$$

Обрахуємо нарахування на заробітну плату розробника продукту:

$$Н_{зп} = (19163{,}64+1916{,}36) \cdot 22 / 100\% = 4637{,}60 \text{ (грн)}.$$

Амортизація обладнання, яке використовувалось для проведення розробки, розраховується за формулою (3.4):

$$A = \frac{Ц \cdot H_a}{100} \cdot \frac{T}{12} \text{ [грн]}, \quad (1.4)$$

де Ц – балансова вартість обладнання, грн.;

$Н_а$ – річна норма амортизаційних відрахувань;

Т – термін використання під час розробки, місяців.

$$H_a = \frac{B_п - B_л}{B_п \cdot T_{кв}} \cdot 100 \text{ [грн]}, \quad (1.5)$$

де $В_п$ і $В_л$ – відповідно первісна та ліквідаційна вартість об’єкта основних фондів;

$Т_{кв}$ – строк корисного використання, 5 роки.

$$H_a = \frac{20000 - 2000}{20000 \cdot 5} \cdot 100 = 18\% \text{(грн)}.$$

Розрахуємо амортизаційні витрати на ноутбук, балансова вартість якого становить 18000 грн, а термін використання – 3 місяців:

$$A = \frac{18000 \cdot 18}{100} \cdot \frac{3}{12} = 810 \text{ (грн)}.$$

Витрати на силову електроенергію визначаються на основі витрат на одиницю продукції та тарифів на енергію за допомогою формули:

$$B_e = B \cdot П \cdot Ф \cdot K_П, \text{ [грн]}, \quad (1.6)$$

де В – вартість 1 кВт електроенергії, грн.;

П – установлена потужність обладнання, кВт;

Ф – фактична кількість годин роботи обладнання, яку задіяно на виготовлення 1 шт. програмного забезпечення з інформаційною технологією захисту вмісту контенту, Ф = 8·22·3=528 год;

$К_П$ – коефіцієнт використання потужності.

$$B_e = 2 \cdot 0{,}6 \cdot 528 \cdot 0{,}4 = 253{,}44 \text{ (грн)}.$$

Витрати за доступ до мережі Інтернет розраховуємо за формулою:

$$B_{\partial i} = Ц_{\partial i} \cdot T \text{ [грн]}, \quad (1.7)$$

де $Ц_{дi}$ – це ціна доступу за місяць;

Т – кількість місяців використання доступу до мережі.

$$В_{ді} = 150 \cdot 3 = 450 \text{ (грн)}.$$

Інші витрати доцільно прийняти, як 50% від суми основної заробітної плати розробника:

$$Ів = 19163,64 \cdot 50\% = 9581,82 \text{(грн)}.$$

Загальні витрати на розробку нового програмного продукту – це сума всіх попередніх витрат:

$$В = 19163,64+1916,36+4637,60+810+253,44+450+9581,82 = 27812,86 \text{ (грн)}.$$

Оскільки над роботою задіяна одна людина, якою виконується вся робота, то α становить 1. Тобто, загальні витрати на виконання роботи дорівнюють 27812,86 грн.

Прогнозування загальних витрат на виконання та впровадження результатів розробки обчислюються за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \text{[грн]}, \qquad (1.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи, $В_{заг}$=В. Так як розробка знаходиться на стадії впровадження, то $\beta \approx 0,9$.

$$ЗВ = \frac{27812,86}{0,9} = 30819,17 \text{ (грн)}.$$

Отже, загальні витрати на виконання та впровадження результатів розробки становлять 30819,17 грн.

1.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі проведемо кількісне прогнозування, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. В умовах ринку узагальнюючим позитивним результатом, що його отримує підприємство від впровадження результатів тієї чи іншої розробки, є збільшення чистого прибутку підприємства. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Зростання чистого прибутку забезпечить підприємству надходження додаткових коштів, які дозволять покращити фінансові результати діяльності.

Виконання даної наукової роботи та впровадження її результатів складає приблизно до 1 року.

Позитивні результати від впровадження розробки очікуються вже в перші місяці після впровадження. Проведемо детальніше прогнозування позитивних результатів та кількісне їх оцінювання по роках. Обчислимо збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \cdot \Delta N)_i \ [грн], \qquad (1.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

Припустимо, що внаслідок впровадження результатів наукової розробки чистий прибуток підприємства збільшиться на 150 грн., а кількість одиниць реалізованої послуги збільшиться: протягом першого року – на 500 од., протягом другого року – ще на 350 од., протягом третього року – ще на 780 од.

Орієнтовно: реалізація продукції до впровадження результатів наукової розробки складала 1 шт., а прибуток, що його отримувало підприємство на одиницю продукції до впровадження результатів наукової розробки – 200 грн.

Потрібно спрогнозувати збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_1$ протягом першого року складе:

$$\Delta П_1=200\cdot 1+(150+200)\cdot 500=175200{,}00 \text{ (грн)}.$$

Обчислимо збільшення чистого прибутку підприємства $\Delta П_2$ протягом другого року:

$$\Delta П_2=200\cdot 1+(150+200)\cdot (500+350)=297700{,}00 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_3$ протягом третього року становитиме:

$$\Delta П_3=200\cdot 1+(150+200)\cdot (500+350+780)=570700{,}00 \text{ (грн)}.$$

Отже, розрахунки показують, що комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

1.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

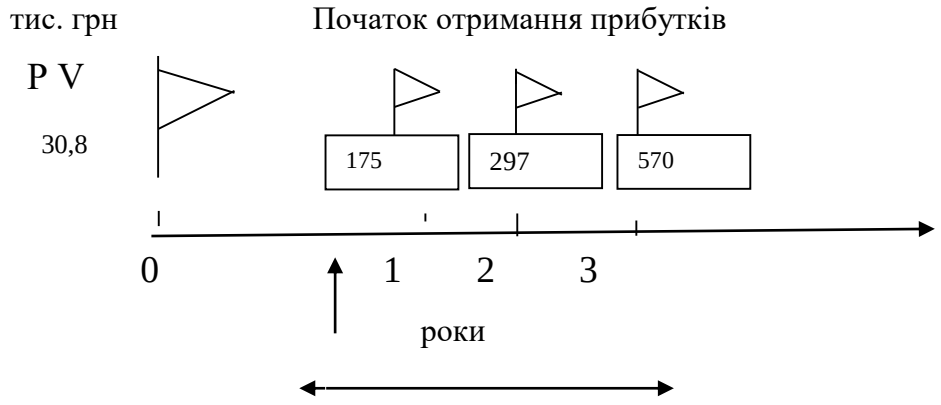
Основними показниками, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності. Розрахунок ефективності вкладених інвестицій передбачає проведення таких робіт:

1-й крок. Розраховують теперішню вартість інвестицій PV, що вкладаються в наукову розробку. Такою вартістю ми можемо вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, тобто будемо вважати, що ЗВ = PV = 55226,94 (грн).

2-й крок. Розраховують очікуване збільшення прибутку $\Delta П_i$, що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження.

3-й крок. Для спрощення подальших розрахунків необхідно будувати вісь часу, на яку нанести всі платежі (інвестиції та прибутки), що мають місце під час виконання науково-дослідної роботи та впровадження її результатів.

Рисунок 1.1 характеризує рух платежів (інвестицій та додаткових прибутків).



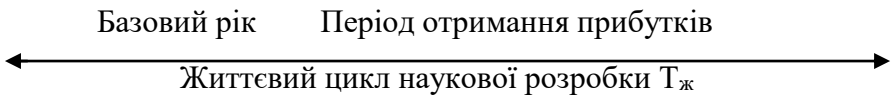


Рисунок 1.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

4-й крок. Розраховують абсолютну ефективність вкладених інвестицій $Е_{абс}$ за формулою:

$$Е_{абс} = (ПП – PV), \quad (1.10)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн.

Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t}, \quad (1.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки „0”;

$$ПП = \frac{175200,00}{(1+0,1)^1} + \frac{297700,00}{(1+0,1)^2} + \frac{570700,00}{(1+0,1)^3} = 840954,62 \text{ (грн).}$$

$$Е_{абс} = 840954,62 – 30819,17 = 810135,45 \text{ (грн).}$$

Оскільки $Е_{абс} > 0$, результат від проведення наукових досліджень щодо розробки програмного продукту та їх впровадження принесе прибуток, тобто є доцільним, але це ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даної програми.

5-й крок. Розраховують відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$E_в = \sqrt[T_ж]{1+\frac{E_{абс}}{PV}} - 1, \quad (1.12)$$

Де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1+\frac{810135,45}{30819,17}} - 1 = \sqrt[3]{7,99} - 1 = 2 \text{ або } 200\%$$

Порівняємо $Е_в$ з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть.

Спрогнозуємо величину $\tau_{мін.}$ У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f, \quad (1.13)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; d = 0,2;

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = 0,05.

$$\tau = 0,2 + 0,05 = 0,25$$

Оскільки $E_в$ = 200% > $\tau_{мін}$ = 25%, то у інвестора буде потенційна зацікавленість у фінансуванні даної наукової розробки.

6-й крок. Розраховують термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ за формулою:

$$T_{ок} = \frac{1}{E_в} \text{[року]}. \quad (1.14)$$

$$T_{ОК} = \frac{1}{0,99} = 0,5 \text{ (року)}.$$

Оскільки термін окупності вкладених у реалізацію наукового проекту інвестицій менше трьох років ($T_{ок} < 3$ років), то фінансування нової розробки є доцільним.

1.5 Висновки

В даному розділі було виконано оцінювання комерційного потенціалу нової розробки. Проведено технологічний аудит з залученням трьох незалежних експертів. Визначено, що рівень комерційного потенціалу розробки вище середнього. Аналіз комерційного потенціалу розробки показав, що програмний продукт за своїми характеристиками випереджає аналогічні програмні продукти і буде затребуваний ринком. Він має кращі функціональні показники, а тому є конкурентоспроможним продуктом на ринку. Існуючі переваги нової розробки дозволять швидко її поширити та популяризувати.

Згідно із розрахунками всіх статей витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи загальні витрати на розробку складають 30819,17 грн.

Розрахована абсолютна ефективність вкладених інвестицій в сумі 810135,45 грн свідчить про отримання прибутку інвестором від комерціалізації програмного продукту.

Щорічна ефективність вкладених в наукову розробку інвестицій складає 200 %, що вище за мінімальну бар'єрну ставку дисконтування, яка складає 25%. Це означає потенційну зацікавленість інвесторів у фінансуванні розробки.

Термін окупності вкладених у реалізацію проекту інвестицій становить 0,5 року, що також свідчить про доцільність фінансування нової розробки.

2 АНАЛІЗ СУЧАСНИХ МОДЕЛЕЙ, МЕТОДІВ ТА ЗАСОБІВ ЗАСТОСУВАННЯ МЕТАДАНИХ В ПРОЦЕСАХ ПОШУКУ

2.1 Методи застосування метаданих в процесах пошуку

Поняття «метаданих» фактично досить широке і точного формулювання не має. У цілому, цим визначенням позначають деяку інформацію, яка характеризує суть якихось інших даних. Так, скажімо, структурні метадані (structural metadata) характеризують формат та деяку взаємодію між різними типами даних в деякій системі. Це, для ясності – така собі, «інформація про зберігання якихось даних». Описові метадані (descriptive metadata) містять у собі інформацію про те, що саме знаходиться в визначеному тексті, містить дані про те, якій темі він присвячений, і яка саме проблематика в ньому розглядається.

У загальному випадку, визначень та класифікацій для широкого поняття метаданих існує немало. Часто , в загальнодоступних джерелах, використовується такий комплект критеріїв, що є спільним для всіх їх типів. Розглянемо для прикладу звичайний текст. Отже, його метадані можуть містити таке:

1) Перелік засобів, що використані і за допомогою яких створено даний текст;
2) Мета цього конкретного тексту;
3) Час та дата створення цього конкретного тексту;
4) Автор цього конкретного тексту;
5) Розташування комп’ютера (гаджета, вузла, хоста), на якому він був створений (розміщений, модифікований тощо);
6) Використані стандарти для відображення цього конкретного тексту.

Часто ці метадані зберігались і знаходилися разом із самим текстом, що вони його характеризують, тобто, в складі формату самого документу. Так, скажімо, відповідні поля для цих дій є в форматах фірми Microsoft, а також у багатьох інших, можливо менш поширених способах зберігання інформації та

даних. Більше того, подібні поля присутні також в графічних форматах (EXIF, JFIF) та, звісно, у звукових форматах (у вигляді т.з. ID3-тегів).

Проте з часом лише цього стало мало, бо цих даних досить для автоматичної категоризації, як вже говорилось вище, але недосить для здійснення складного процесу пошуку і гарної оцінки релевантності текстів. Тому американський фахівець по базам даних (БД) Ральф Кімбелл запропонував розширити систему метаданих загалом, і припустив, що можна поділити матедані на два потужних класи:

- технічні метадані;
- бізнес-метадані [12].

Технічні метадані фактично розташовані всередині самого формату даних, і несуть лиш службову роль. В той час як бізнес-метадані можуть розташовуватись в іншому місці , скажімо, всередині системи БД, або в зовнішньому файлі, тобто, окремо від власне інформації, яку характеризують ці метадані. Отож, він знімав обмеження, яке до цього моменту чітко ставилось форматами файлів, і тим самим відкривав дорогу створенню значно більш складних метаданих, які тепер могли надавати більш детальну характеристику інформації, формувати її окремі структури і отож забезпечувати нові розширені можливості для пошуку.

В літературі одним із перших методів реалізації такої системи метаданих став т.з., метод «Дублінського ядра» (Dublin Core), і це була гнучка система, метаконтент якої був поділений на 2 великі частини, ще до настанов вищезгаданого Кімбелла. Перший комплект отримав назву простого комплекту, і містив базовий комплект із 15 полів, які дозволяли би досить точно ідентифікувати якийсь довільний конкретно взятий текст. Другий, по аналогії, став називатися компетентним комплектом, що тепер застосовувався в складному розширеному процесі пошуку, бо містив не просто низку додатових полів з метаконтентом, а й додатково – самі взаємозв’язки між ними, що зберігались окремо у т.з. «реєстрі метаданих», де додатково знаходились спеціальні «словники», які дозволяли процедуру уніфікації значення відповідних полів.

І уся ця потужна система зводилась до такої досить простої схеми: кожному описуваному ресурсу ставився у відповідність комплект ключових пар по типу «поле-значення» (ці вказані пари є описовими ресурсами), в якому значення обов’язково вибирались би з деякого спеціального словника, асоційованого саме із цим полем. І тоді ці пари могли би міститися в самому документі, або окремо, в БД, чи іншому сховищі даних, на яке обов’язково мало би посилатись ресурс, що характеризується.

Тепер, це значення не лише ділилось за принципом «літерал» чи «не літерал», а й мало свій окремий формат даних та представлення. Така жорстка вимога ставилась для того, аби полегшити створення словників.

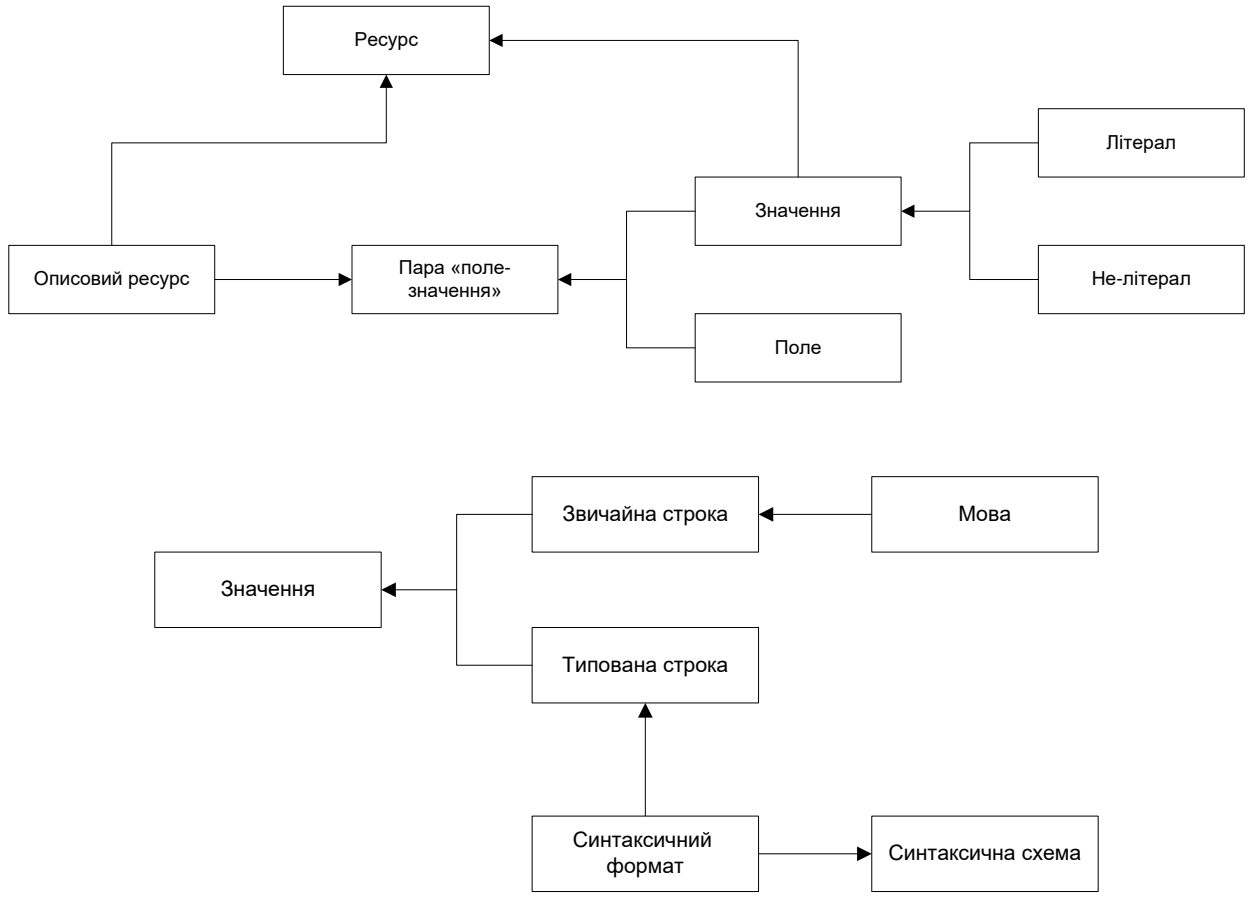


Рисунок 2.1 – Метод Dublin Core

З рисунка 2.1 видно, що в цей метод від початку входила також і деяка мова, якою вказується це значення. Цей параметр, до слова, входить до того ж відомого переліку з 15 базових полів, що мав би містити документ для

задоволення умов «простого» рівня метаданих. А от вже більш складні поля метаданих потребують окремого синтаксису, що може бути реалізовуваним на базовому рівні, або ж тоді містити складні структури, або ж чи ці поля мають входити до ієрархічних та (або) логічних структур вищого рівня [13]. Це зображено це на рисунку 2.2. Тут видно, що будь-який словник містить визначення, кожне з яких може бути значенням якогось окремого поля (а воно, тепер, може докладно деталізувати інше поле – зверніть увагу на додатковий зв'язок цього компоненту). Водночас це визначення має належати до деякого класу, який, відтепер, також може входити до складу деякого іншого, більшого, класу, який і задає категоризацію даного ресурсу по якомусь параметру.

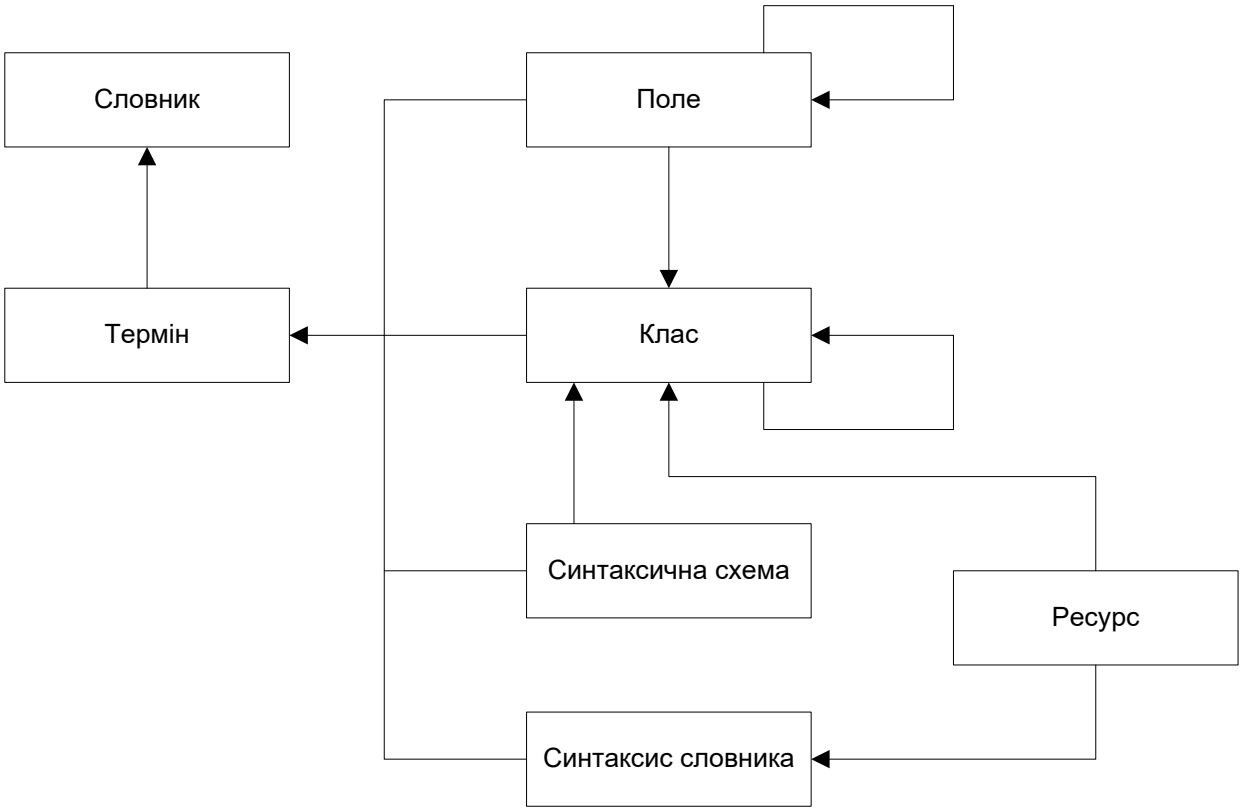


Рисунок 2.2 – Метод ієрархічних структур для систем метаданих

Фізично саме дане визначення має відповідати певній синтаксичній схемі, точно так само, як і словник мав би відповідати певному формату. Зверніть увагу, що зміст викладення самого визначенняу і форми зберігання цих словників розподілені. Викликано це тим, що формат значення фактично

залежить лиш від самого цього визначенняу, або ж класу, до якого цей формат належить. А от формат словника строго залежить від самого ресурсу даних. Бо, скажімо, у випадку web-сторінок він може мати звичний текст-орієнтований формат [14], а от у випадку БД – може зберігатись у вигляді окремої БД із своєю особливою структурою [12].

На основі цього базового методу було побудоване поняття про т.з. семантичні структури – так звані «мета-сутності», які уособлювали логічну організацію визначень в класи та концептуальні схеми. Саме такий підхід мав би забезпечити можливість нечіткого пошуку, і на базі таких порівняно простих логічних висновків дати змогу класифікувати тексти.

Проте, реалізація мета-сутностей була успішно відкладена, оскільки невдовзі виникла інша проблема: як і де саме мають зберігатись метадані у великій інформаційній мережі, якою, зрештою, ставав Internet.

2.2 Моделі та технології застосування метаданих у процесах пошуку в Інтернеті

Вище було описано, що семантичні структури можуть бути реалізовані досить різними способами. Це може бути реалізовано за допомогою структур, що знаходяться усередині самого файлу, або ж це може бути якась окремо створена спеціальна система, призначена для збирання і зберігання саме семантичних даних.

Одною із перших таких спроб створити подібну систему запропонували Рой Голдман та Дженніфер Уідом. Їхня пропозиція зрештою зводилась до необхідності створення такої особливої БД, до створення т.з. «сітки даних» [15], яка би містила форматовані нотатки та довідки про інформаційні матеріали і ресурси, і які би згодом стали основою для формулювання все більш точних запитів та збирання все більшої кількості статистики, що би дозволила оптимізувати процес пошуку. Саме цими авторами було вперше запропоновано застосувати відому деревоподібну модель деталізації метаданих, та ними ж були сформовані базові поняття, на яких створено сучасні семантичні системи.

Нехай існує БД (на прикладі типів закладів харчування) із такою структурою (рис. 2.3).

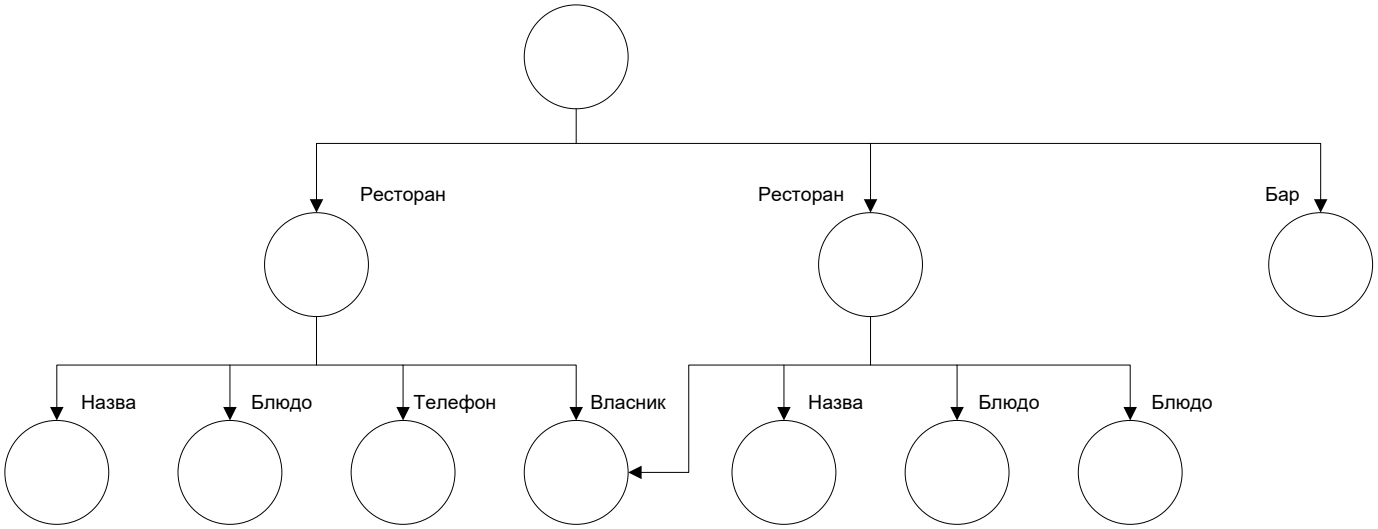


Рисунок 2.3 – Деревоподібна модель БД із нечіткою структурою метаданих

Як видно з рисунка 2.3, в цій БД однакові об’єкти, як не дивно, можуть мати різний перелік властивостей. Причому, деякі із них можуть навіть повторюватись, а деякі – можуть загалі бути відсутніми. З метою впорядкувати все це і дозволити користувачу системи працювати однотипно із усіма елементами системи, незважаючи на їх відмінності, дана БД використовує механізм метаданих, організованих у відповідності до видів і типів елементів. Це може бути окрема структура, що містить список можливих властивостей, які може мати окремий елемент (рис. 2.4).

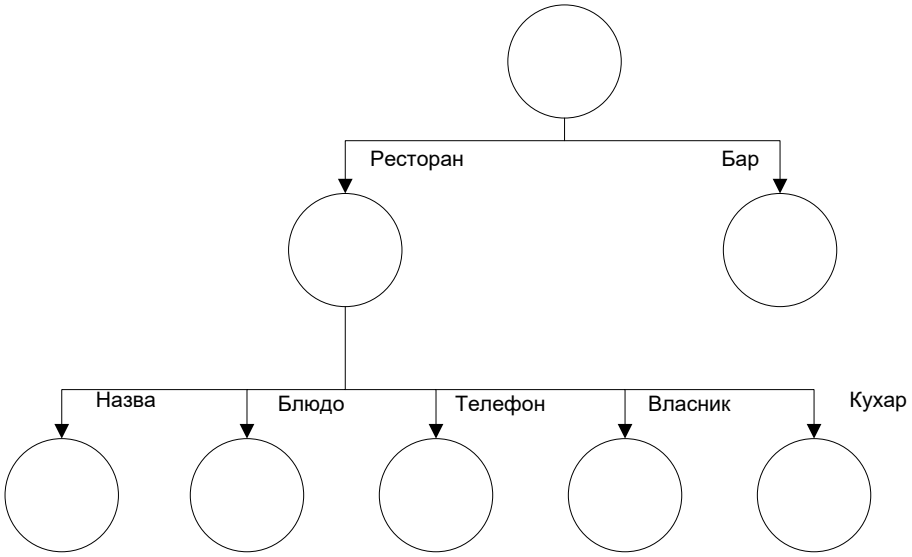


Рисунок 2.4 – Деревоподібна модель БД із чіткою структурою метаданих

Подібний підхід запропонували запровадити і для текстових даних. Згідно цього, в текстовому документі якимось чином виділялися певні «поля» (скажімо, маркувались спеціальними символами, або спеціальними методами розмітки тексту).

Далі створювалась так звана спеціальна «сітка даних», що відповідала показаній на рис. 2.4 структурі метаданих, і водночас містила список та характеристики всіх можливих полів, що можуть бути присутніми в цьому документі. Далі, під час формування звіту користувачу показувалась ця сітка, і тепер в кожну її комірку він міг внести потрібне йому значення. Це можна побачити на прикладі модель пошуку Lore, однієї із перших пошукових систем на базі т.з. сітки даних (рис. 2.5).

Отож, вже нема такої необхідності переглядати весь текст повністю, а можна лиш перевіряти його окремі ключові теги, і ще причому лиш ті, які вказав користувач. Тепер, в поєднанні з процесоми індексування та хешування цих полів, ми можемо значно пришвидшувати пошук, і довести повнотекстовий пошук до тієї ж швидкості, що й БД.

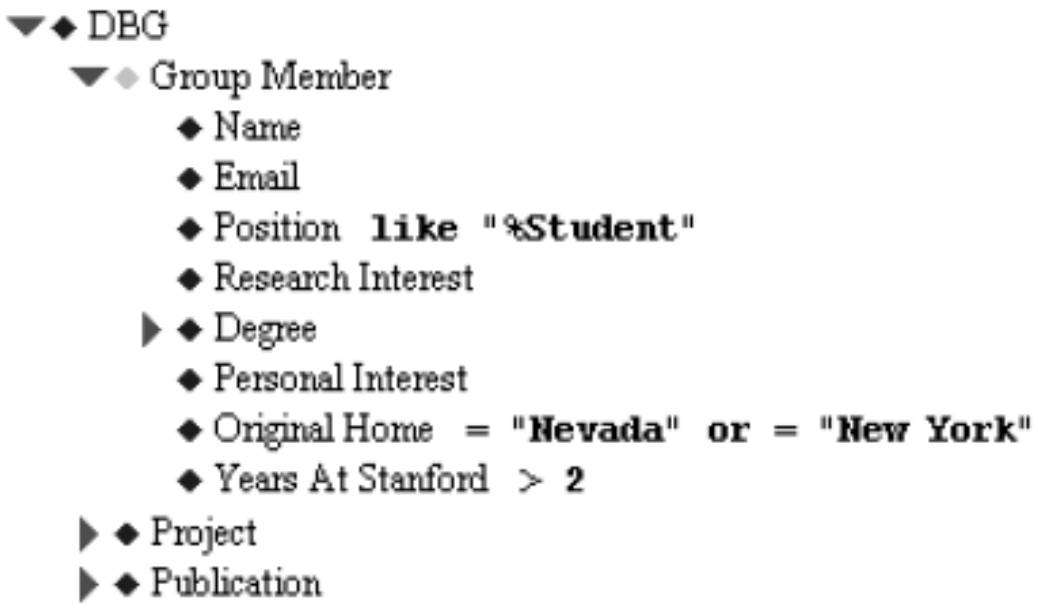


Рисунок 2.5 – Модель пошуку на базі сітки даних

Лишалась не вирішеною лиш одна проблема. Такий підхід, що передбачав виокремлення полів в тексті, а особливо, якщо їх дуже багато, робило текст зовсім нечитабельним. І тут стала у пригоді технологія, яка уже досить давно використовувалась в документообігу, і називалась SGML (Standard Generalized Markup Language). Взагалі це засіб для створення текстових форматів, орієнтованих на автоматичну обробку і дозволяла створювати свої власні теги, якими можна ділити текст не лише на абзаци, а й на іншого типу фрагменти.

Основою технології SGML стала DTD-схема (Document Type Definition). Це є особлива структура, яка міститься в заголовку документа та визначала, на які конкретно фрагменти розділено текст в документі (абзаци, реквізити ділових документів тощо).

В самому документі SGML це виглядало ось так (у прикладі - розклад ТБ).

```
<!Doctype tvschedule [
<!Elem_t tvschedule (channel+)>
<!Elem_t channel (banner,day+)>
<!Elem_t banner (#pcdata)>
<!Elem_t day (date,(holiday|programslot+)+)>
<!Elem_t holiday (#pcdata)>
<!Elem_t date (#pcdata)>
<!Elem_t programslot (time,title,description?)>
```

```
<!Elem_t time (#pcdata)>
<!Elem_t title (#pcdata)>
<!Elem_t description (#pcdata)>
<!Atlist_ tvschedule name cdata #required>
<!Atlist_ channel chan cdata #required>
<!Atlist_ programslot vtr cdata #implied>
<!Atlist_ title rating cdata #implied>
<!Atlist_ title language cdata #implied>
]>
```

Лістинг 2.1 – DTD-схема документу «Розклад ТБ»

Кожен елемент, на який має ділитись даний документ, характеризувався окремим ключовим словом, а в дужках перелічувались всі вкладені елементи, які він міг містити. До того ж, окремо вказувались атрибути та тип елементу.

З метою надати DTD-схемі документу необхідний функціонал, і досить спростити, було розроблено мову програмування XML, яка невдовзі стала технологією семантичної моделі [16]. Сама мова і технологія в основі є простими. Текст ділився на деякі елементи, кожному елементу співставлявся деякий тег, що мав тип і атрибути (майже як в БД):

```
<Installer>
    <Package name="Default">
        <description> Пакет по замовчуванню </description>
        <File name="10 Years Later.txt"
            path="C:\Documents and Settings\Loner\Мои документы\10 Years Later.txt"
size="20353"/>
        <File name="0487.doc"
            path="C:\Documents and Settings\Loner\Мои документы\0487.doc"
size="33792"/>
        <File name="ax_files.xml"
            path="C:\Documents and Settings\Loner\Мои документы\ax_files.xml"
size="1618"/>
</Installer>
```

Лістинг 2.2 – Типовий XML-документ

Як видно з лістинга 2.2, кожний елемент виокремлюється спеціальною розміткою, яка заключена в спеціальні дужки. Кожен елемент у цілому складається із «відкриваючого» тегу та «закриваючого» тегу. Між ними може

міститись текст, або інші дані чи навіть якісь елементи додаткові. Саме таким є елемент description та головний елемент (document element) документа Installer.

Проте він може містити водночас і атрибути, що є парами типу «ключ-значення» і наявність тексту та атрибутів можуть комбінуватись. Всі елементи та атрибути задавались відомим вже нам Document Type Definition (як і в мові SGML), поки не було розроблено стандарт «схема XML».

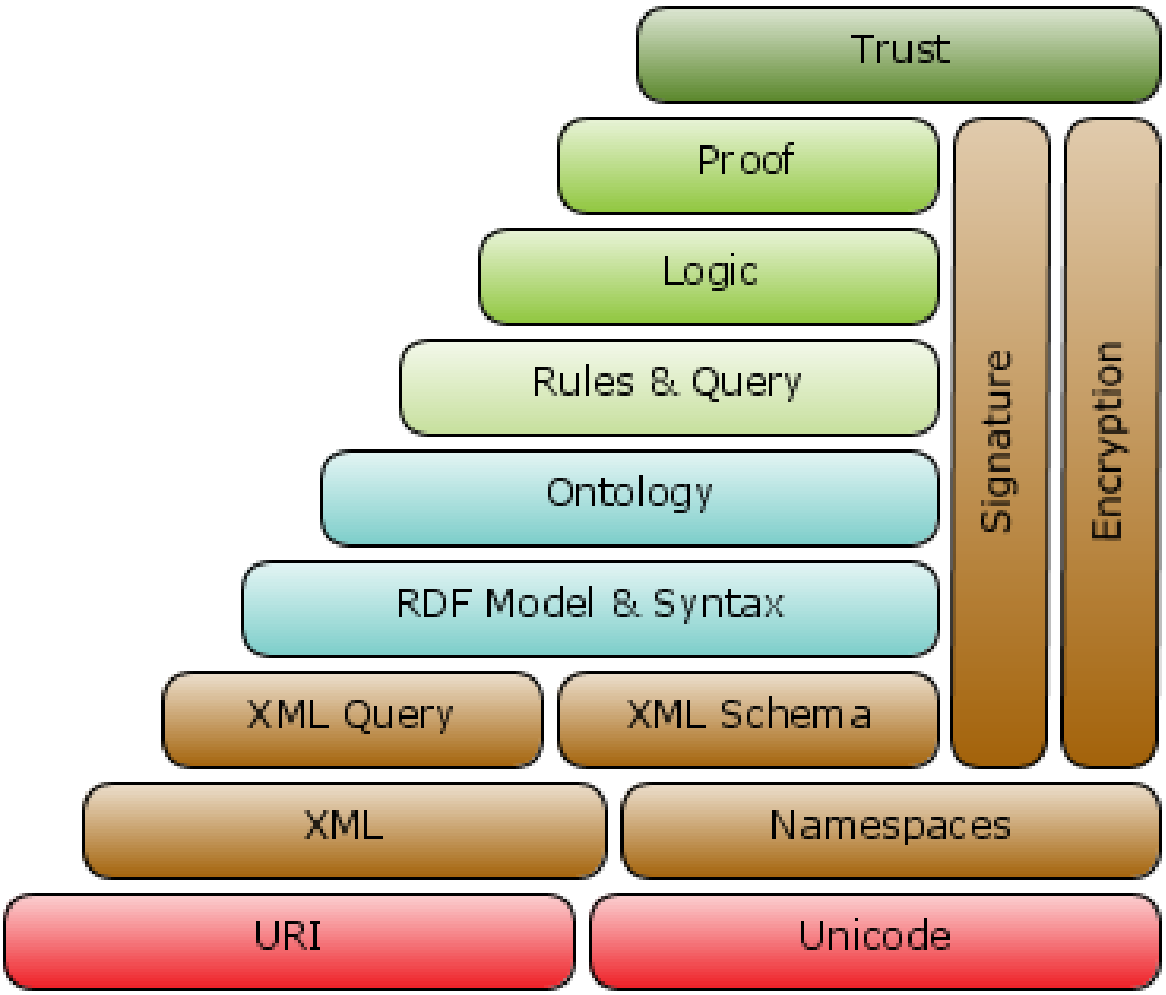


Рисунок 2.5 – Загальна семантична модель для текст-орієнтованих систем (в т. ч. Internet)

Над мовою XML працювали висококваліфіковані світові фахівці і запропонували остаточний варіант нової семантичної моделі, що отримала назву Semantic Web [17]. Створення нової мови програмування XML відкривало нові можливості до семантичної моделі, яка дозволяла описувати

вміст документів і використовувати логічні обчислення для здійснення процесів пошуку. Зовні ця модель виглядала так, як на рисунку 2.5.

ЇЇ ще називають «семантичним стеком» (semantic stack), за аналогією до мережевого протоколу. Розберемо кожен рівень послідовно, і детально ознайомимось із логікою роботи цієї семантичної моделі.

1) Рівень URI та Unicode – вони визначають способи адресування документа та представлення його контенту. URI (Uniform Resource Identifier – Універсальний Ідентифікатор Ресурсу) – це мережева адреса документу.

Саме URI ми вводимо в адресну стрічку браузера, коли хочемо перейти на якусь веб-сторінку. Проте фактично це поняття є значно ширшим. Використовуючі різні ці форми, можна легко проадресувати файл локально, використовуючи мережеві методи [18].

Стандарт Unicode – це низькорівневе представлення тексту на рівні символьних кодів. Кожен символ при цьому кодується 2 байтами, 1 – це № алфавіта, а 2 – № символа, або літери. І так можна представити текст. [19]

2) Рівень XML та XML Namespaces. Про XML говорилось а от про «простори імен» - можна сказати, що це є окремий стандарт і окрема теорія, яка є чільним компонентом моделі семантичного Інтернету. На базі простору імен XML Namespaces будуть розроблені у цій роботі основні алгоритми роботи програмного засобу.

Отже, простор імен XML Namespaces є організаційною структурою, що утворює т.з. семантичну хмару (semantic cloud) даних. Нехай що у нас є документи з різних тематик чи сфер діяльності. Логічно було б, створюючи для них формати, мати конкретну ознаку, яка би дозволяла точно ідентифікувати не лише такі документи, а й їх фрагменти, включені до якогось іншого документу. Ось це й робить механізм «простору імен» XML Namespaces, задаючи тематично прив’язку тега за допомогою префікса [20]:

```
<html:head>
        <html:title>
                        Тест
```

```
        </html:title>
    </html:head>
```

Лістинг 2.3 – Приклад простору імен для HTML

Префікс html (на лістингу 2.3) дозволяє комп’ютеру визначити, що цей і саме цей фрагмент документа слід інтерпретувати як деякий код мовою HTML. Відповідно і залежить відображення на екрані.

Користуючись таким доволі прозорим методом «повідомити» комп’ютеру, що за тип інформації він оброблює, можна частково вирішити задачу уніфікації значної кількості XML-орієнтованих форматів.

3) Рівень XML Schema та XML Query. Відомий вже нам формат схеми XML, який задає сам формат XML-документа, послідовність елементів та їх вкладеність, також належність атрибутів. Він є в XML-форматі, а особливий префікс xs дозволяє програмі відрізнити його від безпосередньо XML-кода [21].

Розглянемо детальніше XML Schema та XML Query, тому що вони будеуть важливими як для подальшого розгляду семантичної моделі Інтернету, так і для розробки програмного засобу в рамках даної магістерської роботи.

```
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
      <xs:Elem_t name="Class">
            <xs:complexType>
                  <xs:sequence>
                        <xs:Elem_t ref="label"/>
                        <xs:Elem_t ref="comment"/>
                        <xs:Elem_t ref="subClassOf" minOccurs="0"/>
                  </xs:sequence>
                  <xs:attribute name="ID" use="required">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:enumeration value="Тест 1"/>
                                    <xs:enumeration value="Тест 2"/>
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:attribute>
            </xs:complexType>
      </xs:element>
    </xs:schema>
```

Лістинг 2.4 – Схема документа за XML Schema

На лістингу 2.4 показано формулювання одного елементу в даному XML-документі. Кожен елемент характеризується спеціальним тегом xs:element, який обов'язково має мати тип. Елемент може також мати атрибути, які обов'язково типуються. Якщо тип елементу не вказанио, тоді вважається, що він текстовий. Це є вжливою частиною поняття метаданих.

Всі типи даних в схемі XML діляться на 2 види:

- простий;
- складний.

Простий тип даних в даному конкретному випадку має атрибут ID, як показано на лістингу 2.4. Перелічуваний тип (counter) відноситься до простого типу також. Складним типом, наприклад, є послідовність якоїсь кількості вкладених елементів. Як показано на лістингу 2.4, елемент Class має саме складний тип, і водночас містить 3 вкладених елементи: label, comment, subClassOf.

Отож, можна задавати необхідний формат документа, одночасно визначати тип. Це такий крок до створення потрібної нам семантичної системи. Для досягнення цієї мети часто застосовують окрему мову запитів XML Query [22]. В основі XML Query лежить програмна реалізація т. з. «сітки даних», такої, що може працювати із будь-яким XML-подібним форматом даних.

4) Рівні номер 4 та 5 мають досить пряме відношення до даної кваліфікаційної магістерської роботи, а тому будуть детально розглянуті пізніше.
5) Рівень номер 6 даної схеми 2.5 і ті рівні, що знаходяться вище – то вони реалізуються в програмній частині, і визначають не сам формат, а основний алгоритм роботи із системою, на основі п'яти вищих рівнів. Основою цього підходу є «модель здобуття знань» (knowledge mining), рсисунок 2.6.

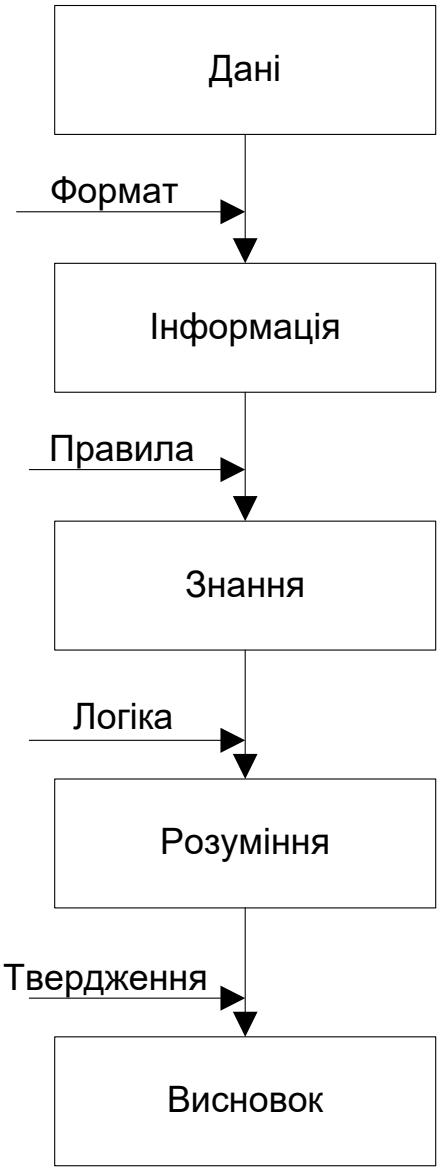


Рисунок 2.6 – Модель здобуття знань

З метою отримати цілісний інформаційний пакет та «повідомити» комп'ютерній системі, з якими саме даними він має справу, можна скористатися простором імен, схем XML та жорсткою типізаціює всіх елементів та атрибутів. Наступним етапом після цього є організація отриманих даних в «знання».

Поняття знання безумовно підкоряється логічним правилам. Часто під час застосування метаданих в процесах пошуку застосовується логіка предикатів, водночас часто можна скористатись масивом правил «якщо... то», які використовуються в дедуктивних БД. Використовуючи такі логічні правила, можна локалізувати певну галузь знань, яка цікавить користувачасистеми.

Отож, подальший науковий пошук зводиться до автоматичного доведення істинності гіпотез, що якийсь матеріал відповідає запитам користувача системи. Всі допущені гіпотези, які пройшли критерій «сітки даних», стають твердженнями (т.з. proof), і видаються у вигляді результатів запитів.

2.3 Застосування метаданих в процесах пошуку на базі технології XML

Як уже зазначалось вище, стек, що застосовує метедані містить два рівні, які мають пряме відношення до семантичних структур:

- RDF-структури;
- мета-сутності.

Це є комплексна система метаданих, що легко може бути адаптована до будь-якої системи (Dublin Core [23], чи якась спеціалізована система). Основою є спеціальний формат RDF (Resource Description Framework), що призначений для описування даних, що існують в мережі, також основою є процеси створення систем метаданих [24]. В основі даного формату RDF стоїть переконання, що кожний об’єкт та його категорія формують логічний предикат типу (рис. 2.7).

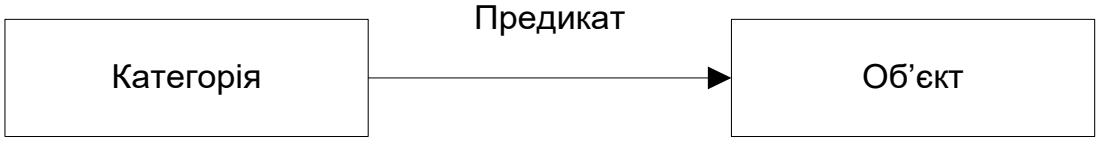


Рисунок 2.7 – Взаємовідносини категорії та об’єкта в RDF

Отож, якщо є задача описати ресурс за допомогою RDF, необхідно підготувати ряд тверджень, або фактів, як вони називаються в теорії аналізу застосування метаданих в процесах пошуку.

Скажімо, є задача описати автора якогось ресурсу, який має web-адресу. Тоді роль категорії виконуватиме web-сторінка. Роль предиката – визначення «автор», а роль об‘єкта – ім’я даного автора.

Так будується структура опису ресурсу за допомогою метаданих. У даному конкретному прикладі на лістингу 2.4 він сам є собі категорія. Проте бувають більш складні випадки, наприклад якщо цей ресурс є частиною якогось іншого, потужнішого ресурсу. Або якщо цей ресурс входить до складу якогось тематичного об’єднання ресурсів, чи є галуззю людських знань (лістинг 2.4).

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Іван Іванченко</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

Лістинг 2.4 – Загальний вигляд RDF-документа

В цьому конкретному випадку на лістингу 2.4 категорією є сам персонаж (Person), і містить такі об’єкти – ім’я, пошту та ступінь (предикати: fullName, mailBox, personalTitle). З метою відрізнити цей формат від інших форматів, використовується спеціальний простір імен, і префікс contact: гарантуватиме, що даний фрагмент система пошуку не сплутає з іншими.

Проте часто буває так, що простого переліку фактів є недостатнім, оскільки потрібні правила аналізу даних і фактів, їх приналежність до категорії. Ці правила містяться в так званих мета-сутностях, які характеризують ієрархічні відносини між предикатами, визначеннями, категоріями.

У цілому, мета-сутності можуть мати різний формат, причому, не лише на основі XML, бо вони, не мають входити до мережевих систем, а можуть бути внутрішнім форматом пошукової системи. До мета-сутностей є 2 вимоги:

1) Мета-дані мають просто формулюватись, і бути легко оброблюваними автоматизованими та автоматичними засобами;
2) Мета-сутність завжди створюється для якоїсь конкретної задачі чи конкретної галузі.

За умов дотримання вищевказаних вимог, мета-сутність буде цілком відповідати стандарту [25] і має містити таке:

- екземпляри,
- поняття, атрибути
- відношення.

Екземпляром називають найнижчий елемент мета-сутності і це є визначення, або конкретне значення якогось поняття. Екземпляри об’єднуються в класи за спільними ознаками. Наприклад, якщо є такі визначення «Паскаль», «C++» та «Ruby», тоді їх можна об’єднати в клас «Мови програмування». Класи можуть містити не лише екземпляри, а й інші класи, формуючи при цьому ієрархію знання.

Тоді кожен екземпляр або сам має бути собі атрибутом, або містити деякий перелік цих атрибутв (проте, і наявність атрибутів не є обов’язковою, хоча з їх допомогою можна встановити непрямі, або агреговані зв’язки, бо атрибут може мати значення, що дорівнює іншому об’єкту мета-сутності).

Зв’язки (або відношення) в мета-сутностях можуть бути вертикальними (відповідно ієрархії), та горизонтальними (відповідно агрегованим зв’язкам). Але в межах формату вони виглядають лінійно. Ось як виглядає мета-сутність, викладена в визначенняах RDF:

```
<?xml version="1.0" encoding="windows-1251"?>
<rdf:RDF>
   <rdfs:Class rdf:ID="Заняття">
      <rdfs:label>Заняття</rdfs:label>
      <rdfs:comment>Перший опис</rdfs:comment>
      <rdfs:subClassOf rdf:resource="Тест 2"/>
   </rdfs:Class>
   <rdf:Property rdf:ID="Тест 1.Работа">
      <rdfs:label>Работа</rdfs:label>
   </rdf:Property>
   <rdfs:Class rdf:ID="Програмування">
      <rdfs:label>Програмування</rdfs:label>
      <rdfs:comment>Другий опис</rdfs:comment>
   </rdfs:Class>
   <rdf:Property rdf:ID="Тест 2.Java">
      <rdfs:label>Java</rdfs:label>
   </rdf:Property>
</rdf:RDF>
```

Лістинг 2.5 – Мета-сутність в RDF-форматі

Як видно з лістинга 2.5, мета-сутність завжди подається в лінійному вигляді, оскільки більшість текст-орієнтованих движків, які не можуть обробляти вкладені елементи (6 рівнів вкладення – це стандарт, скажімо, в HTML-движках), а в мета-сутностях така межа відсутня. І саме тому ієрархічні (вертикальні) відношення в даному випадкуповністю замінені на горизонтальні (агреговані). На ознаку вкладеність класу безпосередньо вказує наявність елемента subClassOf, який вказує на той клас, до змісту якого входить даний клас. Кожен кла, має окремі екземпляри (в даному випадку це – Property). І також, в даному випадку важливим є факт їх наявності, тобто кожен із цих класів є сам собі атрибутом. Модель дерева знань, яке характеризує даний код, має вигляд, як на рисунку 2.8.

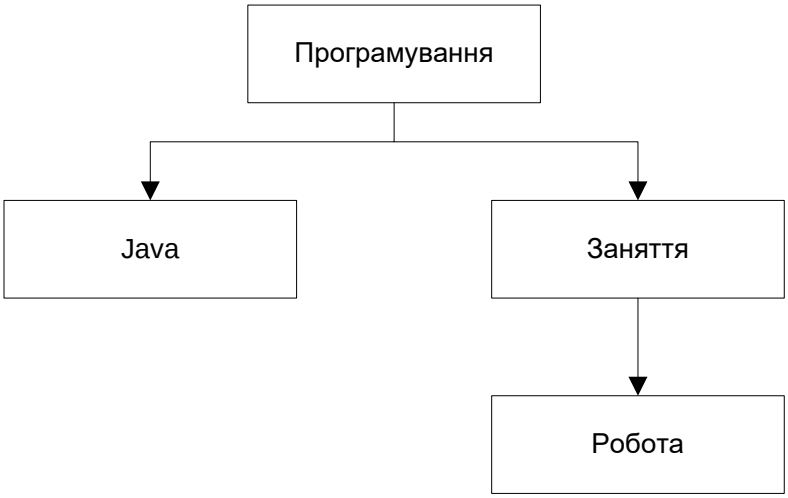


Рисунок 2.8 - Модель дерева знань

Звісно, що такого формату явно недостатньо для реалізації всієї повноти концепції мета-сутності. Тому є спеціальні нестандартизовані (поки що) окремі розширення, що розвиваються в межах стандарту RDF. Ось вони та їх характеристика:

– OWL – Web Ontology Language. Рекомендована W3C, і є логічною мовою програмування. Вона є найбільш комплексним з усіх

стандартів, що має кілька рівнів реалізації – від min до max із гарантованою «вирішуваністю» та «категоризованістю»;

- KIF – Knowledge Interchange Format. Спеціалізований движок, що використовує S-логіку. Є дуже складним у реалізації, але досить ефективний під час аналізу художніх і науково-популярних текстів;
- Common Logic – відгалуження KIF, є стандартизованим нині. Є завершеною логічною мовою програмування;
- CycL – логічна мова програмування, заснована на обчисленнях предикатів.
- DAML – одна із ранніх логічних мов програмування формування запитів на основі XML. Використовується промисловцями та військовими галузями.

В даній магістерській роботі ми будемо використовувати мінімальну форму, бо нашою задачею є реалізація самого принципу застосування метаданих під час процесів пошуку.

2.4 Аналіз існуючих аналогів та поточного стану технологій в галузі застосування метаданих в процесах пошуку

Нині є безліч реалізацій семантичних сіток і заснованих на їх основі інтернет-сервісів. Проте, фактично немає реалізацій таких систем для побутового використання кінцевим споживачем. Тим не менш, пошук на основі мтаданих може цілком використовуватись в таких щоденних сферах життя, як, скажімо, електронні нотатки в календарі.

Виконавши аналіз програм для робочих чи особистих нотаток чи ведення щоденника, було виявлено що майже всі з них містять систему тегування нотаток, однак мають реалізацію на «чистому» HTML. Із збільшенням кількості тем нотаток кількість цих тегів збільшується в геометричній прогресії. Дослідним шляхом можна показати, що якщо застосувати методи застосування метаданих, тобто, семантичної сітки та в результаті створити т.з. «семантичну хмару» для системи підтримки електронних нотаток, тоді необхідна кількість

тегів зменшиться в кілька разів, або ж зменшиться до 2-3 тегів. Решту задачі виконає застосування коректної мета-сутності.

Це буде цілком достойною демонстрацією можливостей застосування мета-даних в процесах пошуку. Оскільки дасть спрощення самого процесу пошуку та дозволить підвищити його ефективність.

Тоді як аналоги можна розглядати 2 типи систем:

3) движки семантичної сітки (чи це можуть бути технології і мови програмування, які підтримують роботу із ними);
4) інтерфейсний прототип стандарту для створення демонстраційного програмного засобу.

Нині є такі движки, які підтримують RDF:

- R2RML Parser – модуль, в мові програмування Java. Дозволяє «розбирати» формат RDF, генерувати нові структури в форматі RDF. До того ж має хоч і обмежену, але підтримку мови запитів в межах простих мета-сутностей;
- dotNetRDF – модуль, в мові програмування C#, та ін. мовах програмування, на основі технології .NET. Призначена для маніпуляцій RDF на простому рівні, а також формулювання простих запитів;
- 4Suite – комплексний пакет роботи з форматом XML мовою програмування Python. Підтримує RDF як один із XML-форматів. Дозволяє обробляти RDF, але не може формулювати запити;
- ActiveRDF – пакет для роботи з форматом RDF для мови програмування Ruby. Вміє обробляти, але не формулювати запити. Є частиною web-движка Ruby on Rails;
- ARC RDF Store – спеціалізований пакет для роботи з форматом RDF безпосередньо з web-сторінки, написаної мовою PHP. Є частиною web-движка;
- Brahms – пакет для роботи з форматом RDF з ередовища мови програмування C++. Швидкий, здатний опрацьовувати великі мета-сутності. Має велику бібліотеку класів.

Зведемо в таблицю 2.1 характеристики знайдених засобів, і оцінимо їх за критеріями, далі визначимо який із критеріїв найбільш підійде для реалізації застосування метаданих до процесу пошуу в галузі електронних нотаток.

Таблиця 2.1 – Порівняльна таблиця движків RDF

	Платформи	Web-движок	Створення RDF	Обробка RDF	Запити
R2RML Parser	Всі	Ні	Так	Так	Так
dotNetRDF	Windows	Ні	Так	Так	Так
4Suite	Всі	Так	Так	Так	Ні
ActiveRDF	Всі	Так	Так	Так	Ні
ARC RDF Store	Всі	Так	Так	Так	Ні
Brahms	Windows, Linux	Ні	Так	Так	Так

Виходячи з даних таблиці 2.1, максимально підходить під умови магістерської роботи 2 движка: R2RML для Java, а також Brahms для C++. З поміж них з економічної точки зору краще обрати Java, бо час розробки програми цією мовою менший, ніж мовою C++ [26] [27].

2.5 Програмні засоби застосування метаданих в процесах пошуку

Є багато програмних засобів для ведення електронних персональних нотаток. Для досягнення поставленої мети, оберемо той програмний засіб, що має необхідний інтерфейс, який не хоче від користувача багато часу на освоєння, і водночас буде досить функціональним аби вести щоденні нотатки.

Нині існує кілька таких програм.

Chrysanth Diary. Дане ПЗ призначено не лише для ведення нотаток, а й для організації приватних даних. Сюди входить підсистема електронних нотаток,

організатор фотографій, система звукозапису та відео. До переваг можна віднести інтуїтивний інтерфейс, таку особливість, що основою є не календар, а планувальник. Типи нотаток розподіляються по окремих вкладках, і таким чином щоденникові записи розміщуються останніми, що знижує навігацію, і ц ізаписи не прив’язані до календаря, що власне ускладнює їх пошук. Нотатки тегуються, проте їх завдання неочевидне, і доступне лиш після вибору необхідного функціоналу.

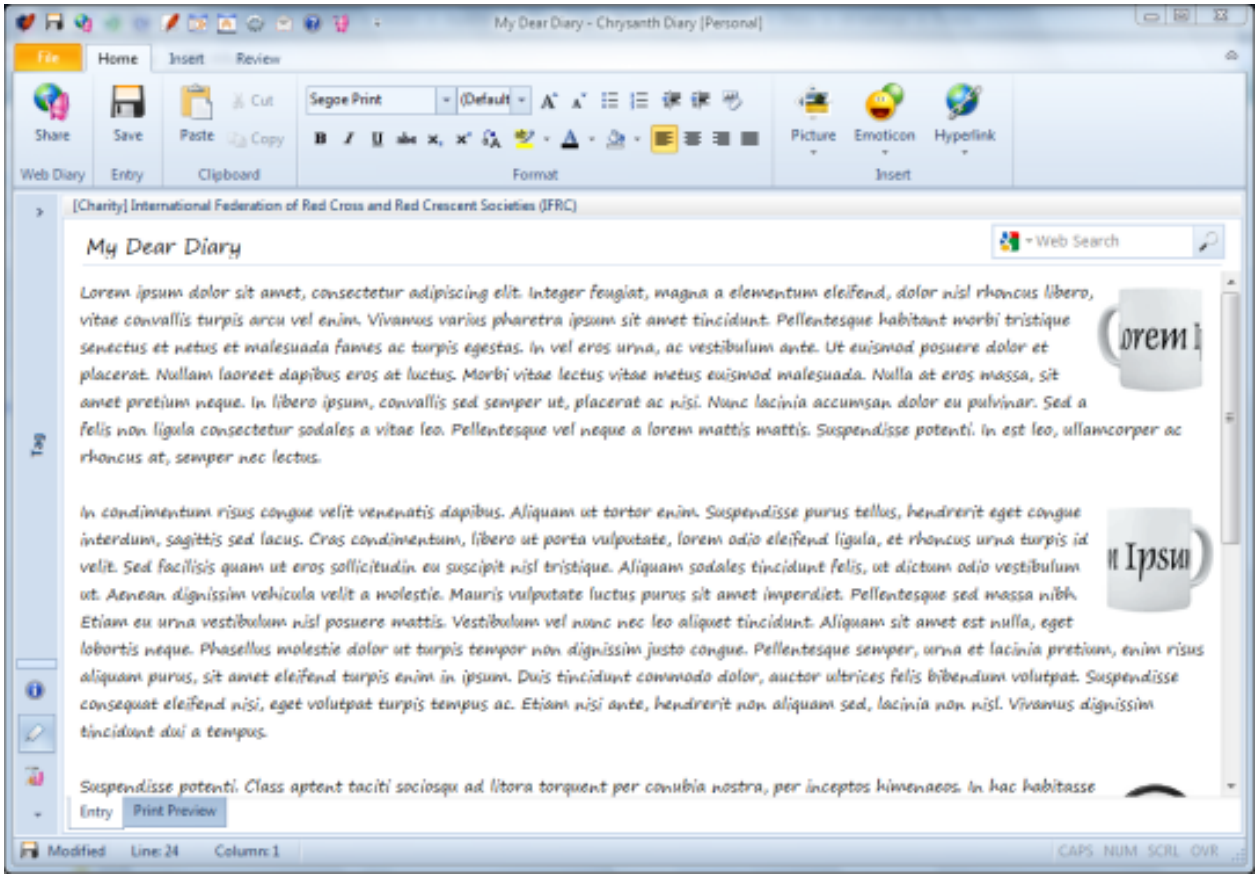


Рисунок 2.9 – Chrysanth Diary

iDailyDiary. Цей програмний засіб здебільшого не призначений для ведення безпосередньо нотаток, оскільки записи прив’язані не до самого календаря, а йдуть загальним текстом. До того ж, тут нотатки не тегуються взагалі. Його перевагами є мінімалістська простота дизайну і поділ всіх нотаток на 3 категорії: власне нотатки, планування та користувацькі документи. Але ці документи, які можна готувати прямо у програмі, не заходячи в офісні пакети

прикладних програм. Гібридні системи подібного типу підтримують лише повнотекстовий пошук.

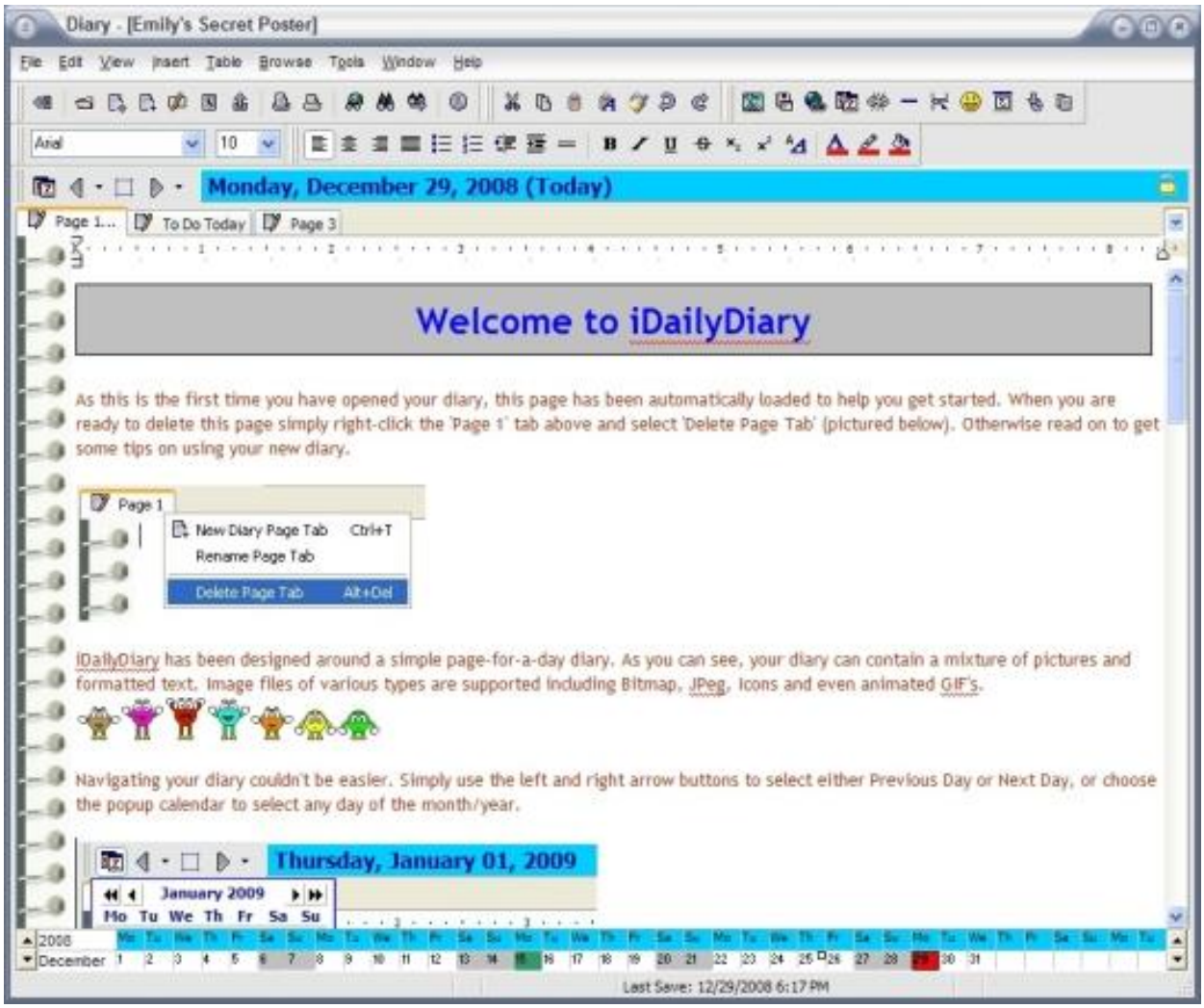


Рисунок 2.10 – iDailyDiary

SmartDiarySuite. Ця програма є комплексним агрегатором підтримки кількох користувачів на 1 робочій станції, плюс захищає їх дані паролями. Тут кожна нотатка одразу тегується, і записується в особливу БД, яка вміє автоматично індексувати та сортувати ці нотатки. Теги при цьому процесі категоризуються. Окрім власне категорій, існують механізми для додаткового створення категорій та тегів із складними умовами. Програма SmartDiarySuite не має розвиненої системи пошуку, зокрема, тут п ошук такий: за календарем, тегами та повним текстом.

Інтерфейс програмного засобу досить простий і побудований за принципом панелей, що спрощує реалізацію її інтерфейсного аналога. Отже, саме вона і буде обрана нами за інтерфейсний прототип.

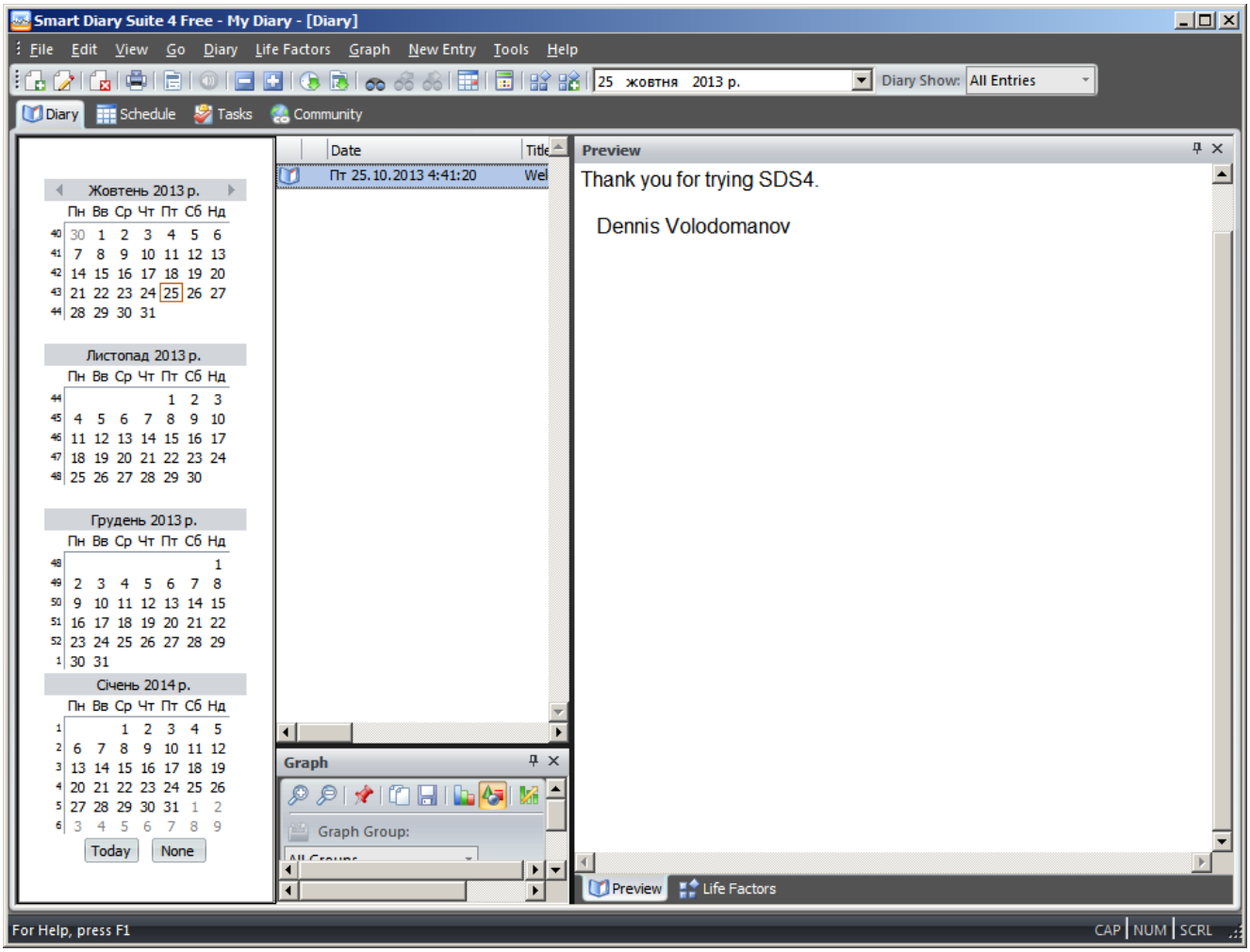


Рис. 2.11 – Smart Diary Suite

Отож, обравши прототипи для реалізації движка та інтерфейсного рішення, можна перейти до формулювання технічного завдання на демонстраційний приклад в даній кваліфікаційній магістерській роботі.

2.6 Формулювання технічного завдання

Виходячи із теми та мети даної кваліфікаційної магістерської роботи, маємо створити демонстраційний програмний засіб для ведення деякої системи

електронних нотаток, сстеми їх тегування, застосування метаданих в процесі пошуку на основі тегів з використанням семантичної хмари для пошуку схожих чи суміжних нотаток.

До програмного засобу, що розробляється, є кілька груп вимог:

1) Технічні умови визначають параметри самої системи, яких має бути достатньо для її стабільної роботи.
2) Функціональні вимоги визначають визначений перелік функцій, які гарантовано мають бути реалізовані в програмі.
3) Ергономічні вимоги визначають вимоги до інтерфейсу програми.

Розглянемо кожну із цих груп. У якості прототипу движка було обрано бібліотеку мови програмування Java, і технічні умови мають відповідати умовам, під час задоволення яких буде працювати середовище Java. Це будуть мінімальні технічні умови для запуску програми. Нині ці вимоги мають такий перелік ресурсів:

– ОС: Будь-яка, що підтримує Java;
– ОЗП: 64 Мб – 128 Мб;
– Жорсткий диск: 125 Мб. вільного місця;
– Інші вимоги: відсутні.

До функціональних вимог слід віднести ті, що стосуються власне виконання головної задачі демонстраційного прикладу, а саме:

– Створення системи електронних нотаток та прив’язка їх до внутрішнього календаря;
– На одну й ту саму дату може бути створено кілька різних нотаток;
– Кожна нотатка має параметри: заголовок, зміст і перелік тегів;
– Допускається, безумовно, необмежена кількість редагувань і можливість видалення нотатки чи групи нотаток;
– Створення, редагування та зберігання мета-сутностей;
– Здійснення процесу пошуку за датою;
– Здійснення процесу пошуку за заголовком;
– Процес повнотекстового пошуку;

– Можливість виконувати нечіткий пошук із використанням мета-сутності, при цьому ця мета-сутність має обиратись незалежно від того, які документи в даний момент відкриті користувачем;

– Видавання користувачеві результатів таким чином, аби він міг їх продивитись і оцінити їх релевантність, не виходячи з вікна пошуку програми;

– Нотатки мають захищатись особистиим паролем користувача;

– Програма має підтримувати кількох користувачів і їх паролі.

До ергономічних умов можна віднести вимоги до інтерфейсу:

– Програма повинна бути легкою в керуванні і досить простою для інтуїтивного керування нею;

– Всі базові функції мають бути продубльовані як в розділі головного меню, так і на додатковій кнопковій панелі;

– Редагування мета-сутностей має бути наглядним;

– Виконання кожної із вищевказаних функціональних вимог має вимагати не більше 2 дій (зайти в меню + вбрати пункт для дії).

Цього цілком досить для створення демонстраційного прикладу, що дозволить показати, як за допомогою семантичної сітки та семантичної хмари можна різко зменшити кількість тегів на електронну нотатку.

2.7 Висновки за розділом

В даному розділі була конкретизована тема необхідності розвитку затосування матаданих в процесах пошуку. Була розглянута ідея цього підходу, основні методи, моделі та поняття теорії метаданих. Проведено короткий опис технологій затосування матаданих в процесах пошуку. В процесі аналізу було зроблено висновок, що запропонована до розгляду технологія в основному застосовується в мережах, хоча і має потужний потенціал застосування в ПЗ для звичайних користувачів. Цей результат аналізу приводить нас до емпіричної ідеї про застосування метаданих у процесах пошуку, що дозволить зменшити тегування електронних нотаток, і підвищити ефективність пошуку потрібної інформації в особистих документах користувача.

З метою перевірити і потвердити отримане емпіричне твердження, було проаналізовано існуючі на ринку семантичні движки, існуючі програмні засоби ведення персональних та ділових електронних нотаток, обрано прототип для формування демонстраційного прикладу, а також сформульовано завдання на розробку.

3 МОДЕЛЬ, МЕТОД ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУВАННЯ МЕТАДАНИХ В ПРОЦЕСАХ ПОШУКУ

3.1 Модель застосування метаданих в процесах пошуку

3.1.1 Загальний принцип застосування метаданих в тегах під час процесу пошуку

Тегування електронних записів, в тому числі і нотаток, що актуально для теми даної магістерської роботи, використовується з метою полегшення процесів пошуку через можливість категоризувати дані всередині електронних записів [12]. Кожному із записів ставиться у відповідність комплект ключових слів (тегів), кожне із яких потім індексується, і отож користувач може переглядати та групувати свої електронні закладки з точки зору різних тематик, чи у відповідності певним задачам. Система тегів була згодом розвинута в модель соціального тегування [28], яка була покладена в основу Web 2.0.

Першопочатково модель застосування метаданих в процесах пошуку була створена для пришвидшення людського пошуку. Організація контенту за тегами і ключовими словами дозволяла людям самостійно утворювати ієрархії категорій, і отож брати на себе частку функцій та впливати на процеси застосування метаданих в процесах пошуку.

Цілком справедливим є факт, що кількість тегів із зростанням кількості матеріалу завжди також росте, причому зростає експоненційно [12]. Тоді виникає т.з. «хмара тегів», яку часто ілюструють діаграмою частотного розподілу їх використання [29]. Фактично «хмара» є видом зваженого списку, де всі теги групуються відповідно їх частотному розподілу. Графічним і візуальним чином це можна показати на рисунку 3.1, де частота позначається розміром шрифта.

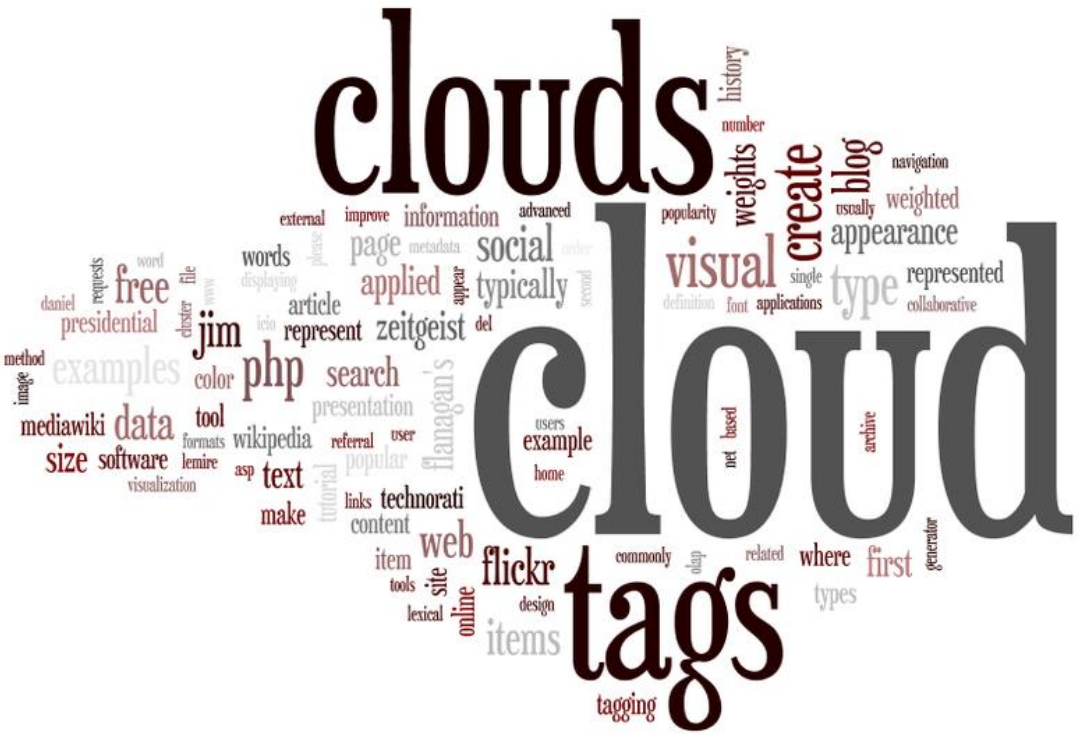


Рисунок 3.1 – Приклад хмари тегів

Далі такий зважений список групується за певними категоріями, далі утворюється так звана «глобальна хмара метаданих» із кластерами, що поділені за елементами та користувачами. Утворені кластери можуть сортуватися за кількістю під категорій [12].

Нечіткий пошук в такій системі із застосуванням метаданих зводиться до гіпотези, що у випадку досить простих тегів вони співпадатимуть із ключовими словами, які задає людина в рядку пошуку, і сам процес пошуку відтак стане більш ефективним. На практиці ця гіпотеза цілковито не потвердилась. Приична в тому, що окрема особа вибудовує власну термінологію ключових слів, керуючись особистою лексикою, яка часто не має відношення до граматичних чи семантичних правил [12].

Скажімо, синонімічний ряд тегів для ОС Linux виглядає так: Linux, GNU/Linux, Линакс, Лінукс, Лінуха, Лялікс, Лунікс. Це не враховуючи різновидів цієї ОС, які теж вживаються як частина синонімічного ряду. Все це формує явище «фолксономії» [29].

Розглянемо приклад. Нехай користувач створив деякий електронний матеріал, розмістив на якомусь реусрсі, і маркував 5 тегами, які відображають різні аспекти матеріалу.

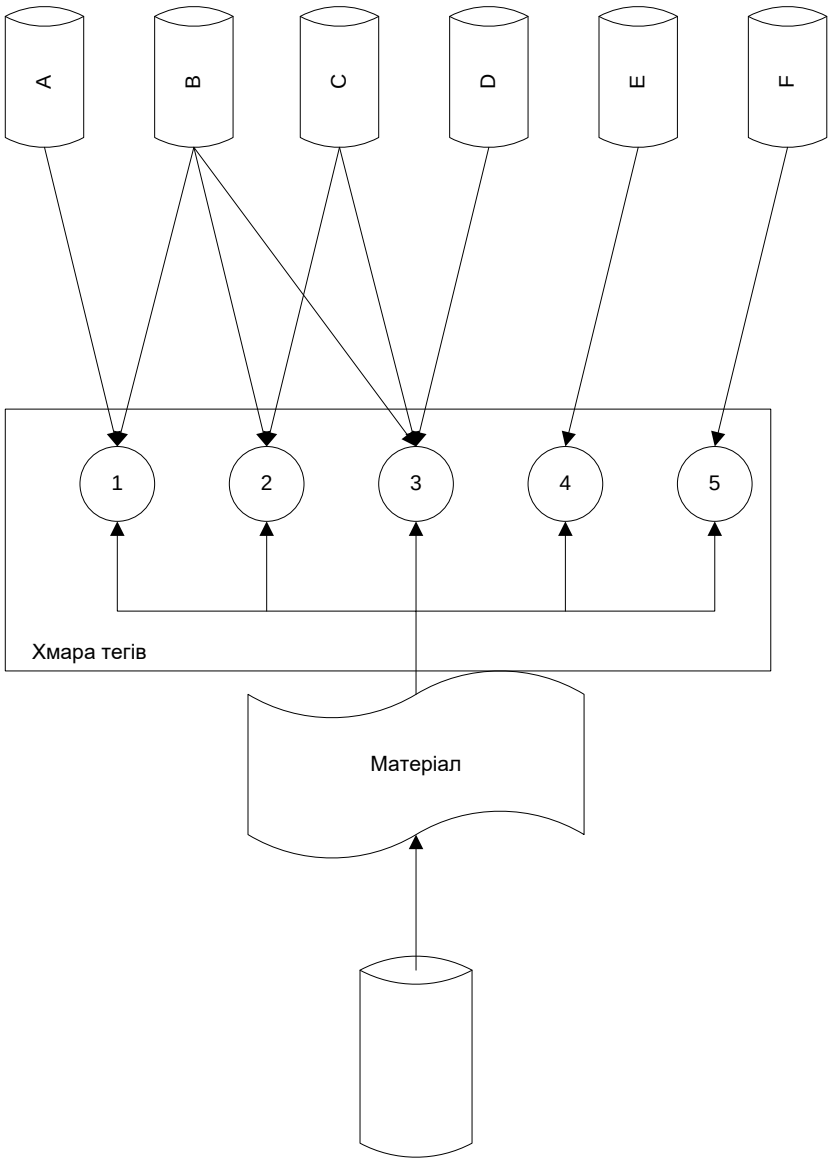


Рисунок 3.2 – Широке фолксономічне тегування

Під час подальшого поширення цього матеріалу, він може бути тегований не всіма цими ключовими словами, а лиш тими, які будуть такими для інших користувачів. І тоді може бути так, що у результаті цей матеріал ніби-то «зникне» із індексів, які відповідають аспектам 4 і 5 (бо вони лишаться в поширеннях значно меншої кількості користувачів), і тоді цей матеріал буде категоризований із більшою ймовірністю в кластери із застосуванням

метаданих, що відповідають аспектам 1, 2 і 3 (умовно). І тоді цей матеріал взагалі ризикує бути «не знайденим», бо логічний ланцюжок, який пов'язує всі 5 маркерів матеріалу, будуть втрачені [12].

Такий фолксономічний підхід дозволяє користувачам систем пошуку формувати свої кластери понять і визначеннь, і навіть свою власну хмару метаданих. Проте при цьому дуже важко знайти об'єкти, які носять загальний чи ну дуже окремий характер. Виникає така собі плутанина, і тоді знайти цей матеріал можна, звернувшись до самого автора, або досить довго заглибившись в його публікації. Звісно, що на це в користувача нема часу, і виникає та сама задача, яка була поставлена ще у вступі до даної роботи – проблема т.з. «незнаходжуваності» матеріалу.

Вирішити цю задачу, можна, зберігаючи на якомусь ресурсі ці логічні ланцюжки, і саме тут і може бути корисною модель застосування метаданих в тегах в процесах пошуку в Інтернет.

Вже згадувалося, що за допомогою мета-сутностей можна будувати ієрархії визначень та їх логічних взаємозв'язків. Є ідея дозволити користувачу створювати свої мета-сутності. Тоді, під час процесів пошуку, інший користувач (або пошукова система в автоматичному режимі) зможе звернутись до мета-сутності автора матеріалу, і таки мчином легко відновити всі необхідні логічні зв'язки. Ступінь знаходжуваності документа при такому підході значно підвищується. Звісно, що зільшиться обсяг самих метаданих, але це не становить вели коїпроблеми із сучасним розвитком обчислювальної та комп'ютерної техніки, де обсяги пам'яті уже не має такого критичного значення, як це було іще 10 років назад.

Підхід цей, використовується в «фасетній класифікації», що полягала у узгодженні різних бібліотечних класифікацій, які існували в світі, і створенні «сітчастої» моделі класифікації, де кожна класифікаційна система – це вісь, яку перетинають інші класифікації в точках, де їх визначення співпадали. Звісно, що сама така сітчаста структура не може точно категоризувати матеріал, проте вона дозволяє переходити від однієї класифікації до іншої, спираючись на

спільні визначення [30]. Такий метод узгодження різних ієрархій визначеннь називається багатоаспектним класифікуванням.

Саме цю модель поклали в основу застосування метаданих до процесів пошуку, і саме наведені елементи фасетної класифікації можна застосувати для реалізації нечіткого пошуку по користувацьким тегам [12].

3.1.2 Математичний опис моделі застосування метаданих в процесах пошуку

Хмара тегів із декількома класифікаційними системами є деякою невпорядковану множину визначеннь. За умови їх організаціїї в ієрархічну структуру, хмара стане схожою на орієнтований граф, в якому відношення стануть ребрами, а їх напрямок позначить належність до якогось із класів. Тоді різні класифікаційні відношення можна позначати ребрами різного кольору, або із різними позначками. Отож, будь-який пошук із застосуванням метаданих за багатоаспектною класифікацією ма би зводитись до кількох досить простих операцій.

Нехай у нас є деякий початковий вислів чи значення, який має відповідний вузол в цьому орієнтованому графі.

Першою операцією стане формулювання списку вузлів, спільних ребер які у даного вузла є вхідними, тобто позначають входження до класу. Отож, ми отримаємо комплект класифікацій за різними т.з. фолксономіями. Далі ми маємо виконати вибірку елементів, які пов’язані вихідними ребрами із вузлів-класів. Всі матеріали, які марковані хоча би одним із вузлів і попали у вибірку, мають бути включеними в пошук.

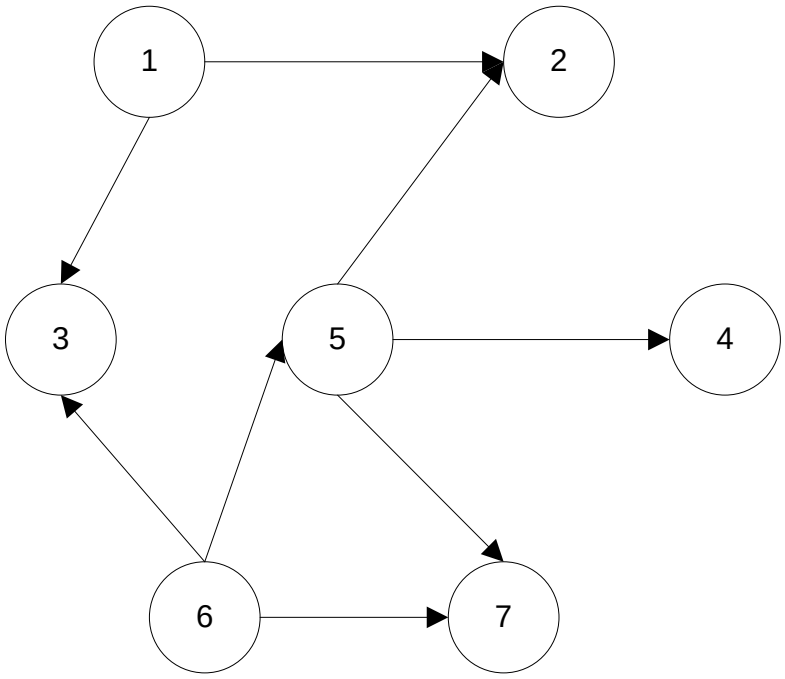


Рисунок 3.3 – Хмара тегів у вигляді орієнтованого графу

Отож можна навіть уникнути помилкового виключення електронних матеріалів із процесів пошуку навіть тому, що він був неправильно чи не повністю класифікований своїм же автором.

Звісно, що такий повномасштабний пошук за допомогою багатоаспектної класифікації, реалізованої як мета-сутності, виходить за рамки даної магістерської роботи вже в силу свого обсягу. Проте, цілком можна реалізувати процес пошуку у менших масштабах, застосувавши технологію RDF, яка містить необхідні для цього інструменти.

З цією метою скористаємось 2 елементами, які входять до його складу:

- Class;
- Property.

Визначення, що входять до складу класів можемо позначити в якості властивості, і перерисуємо вищенаведений орієнтований граф таким чином, як на рисунку 3.4:

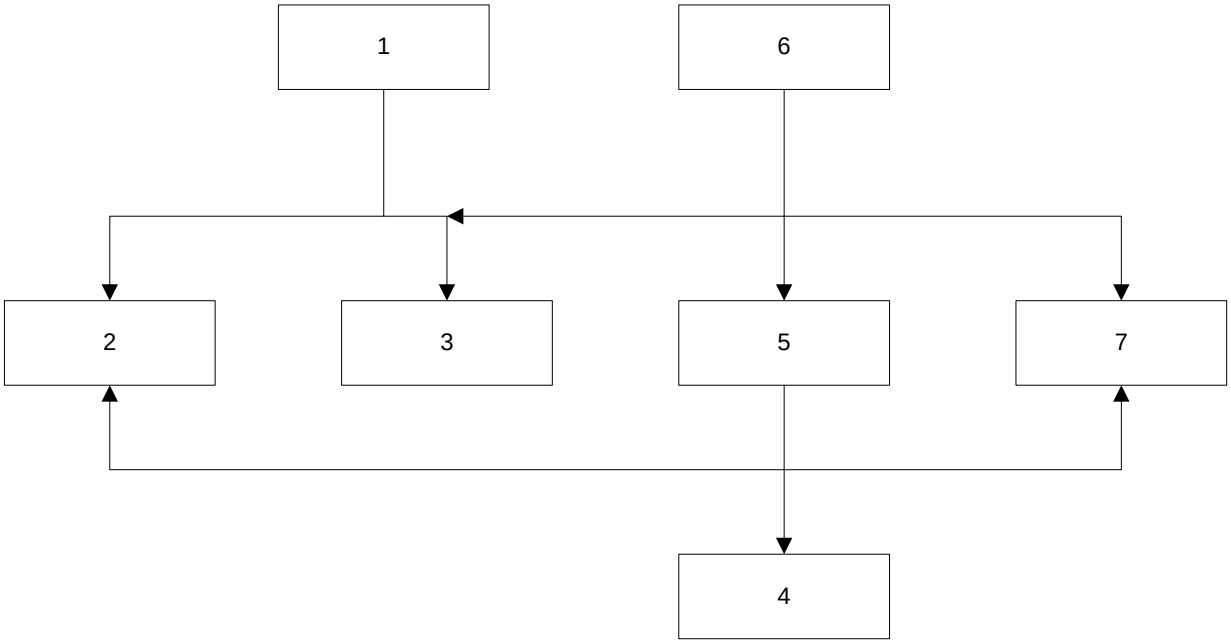


Рисунок 3.4 – Представлення хмари тегів у вигляді структури RDF

Представивши наразі структуру тегів, можна виділити класи (1) та (6), а також (5), який є підкласом по відношенню до (5).

Одночасно доведеться виконати повторення, бо за стандартом RDF не може бути 2 однакових властивостей, бо вони всі префіксуються іменем класу. Такими «вузлами-дублями» можуть бути у даному випадку вузли (2) і (7), які одночасно входять до ієрархій (1) та (5) та (5) та (6). Відповідно, ці пари ієрархій є сіткою Рангатана, де подібні визначення утворюють специфічні зв'язки між класифікаційними системами.

Отже, саме цей підхід дозволив нам створити сітчасту класифікацію за Рангатаном із мінімальними технічними витратами та одночасно спростити алгоритмічну реалізацію. Зокрема, завдяки підходу за Рангатаном можна застосувати теорію множин з метою описати математично запропонований нами метод.

Система Рангатана основана на 2 базових принципах:

1) Поняття, визначення і терміни представлені у вигляді набору фасетних ознак (переліку тегів, які відносяться до деякого класу);

2) Класифікаційні та (або) пошукові індекси синтезуються способом комбінування фасетних ознак відповідно до фасетної формули.

Розглядаючи ці 2 принципи через призму теорії множин, можна сказати, що дана фасетна класифікація стане набором множин, елементи яких теж будуть множинами. А кожний такий класифікатор буде множиною, у якій кожен елемент-множина містить хоча би 1 спільний із іншим елементом-множиною елемент-поняття.

З метою реалізувати у все виглялі алгоритму вищенаписане, потрібна фасетна формула, тобто математичний принцип, по якому можна сформувати індексний масив і надати користувачеві деякий результат. Для цього застосуємо метод теорії множин, при цьому будемо розглядати будь-яку мета-сутність як множину множин (3.1):

$$0 = \{I_1, I_2, \dots, I_n\}, \tag{3.1}$$

де I – це такі класи, що містять окремі визначення-властивості та підкласи, причому визначені вони ось так, як у (3.2):

$$I_0 = \{T_1, T_2, \dots, T_j; I_1, I_2, \dots, I_k\}, \text{ причому } I_k \neq I_0 \tag{3.2}$$

Важливо: випадок, коли множина-клас містить у якості підпорядкованого елемента саму себе, звісно, існує. Тоді такий тип класифікації будемо вважати рекурентною, і вона тоді реального застосування не матиме. Будемо вважати, що 2 класи є подібними, якщо виконується вираз (3.3):

$$I^1 \sim I^2, \text{ за умови } I^1 \cap I^2 = \emptyset \tag{3.3}$$

Множина, що утворюється в результаті перетину батьківських множин, має формуватись в результаті рекурентного процесу. Тобто, подібні визначення варто шукати не лише у відповідній підмножині визначеннь даного класу, а й в цих же множинах підкласів. Це дає можливість з метою пошуку точок перетину в сітці Рангатана застосуватий класичний алгоритм пошуку в глибину, що

дозволить виявити найбільш складні окремі фолксономічні випадки. Відповідно, в ролі фасетної формули виступатиме рекуррентна формула (3.4):

$$\begin{cases} F(0) = I^1 \cap I^2 \neq \emptyset \\ F(1) = I^1 \cap \left(I_1^2, I_2^2, \dots, I_k^2\right) \neq \emptyset \\ \dots \\ F(n) = I^1 \cap (I_1^n, I_2^n, \dots, I_k^n) \neq \emptyset \end{cases}, \quad (3.4)$$

де n – кількість подібних класів.

Маючи таку фасетну класифікацію, процес пошукового індексування можна виконувати за допомогою алгоритмічного розв’язку цієї системи рівнянь. Таких рішень може бути 2:

- пошук в глибину;
- рекурсивний пошук.

В даній роботі було використано другий варіант, оскільки він має більш просту реалізацію, у процес якої включено все, що стосується даної тематики, враховуючи «різнобій» в тегуванні та фолксономіях. Суттєву складність алгоритму становитиме число вкладеності класів, які можуть містити зібрані за фасетною формулою класифікатори. Тоді можна застосувати до вкладених класифікаторів рекурсивні алгоритми таким же чином, як і до основного класифікатора. Приклад однієї із можливих реалізацій застосування мета-даних в процесах пошуку та здійснення на цій основі нечіткого пошуку наведено далі в цій роботі.

3.2 Метод застосування метаданих в процесах пошуку

Загальна суть методу застосування метаданих в процесах пошуку полягає у використанні таких вдосконалених процесів:

- застосування мета-сутностей;
- організації пошуку по тегам.

Для організації роботи методу застосування метаданих в процесах пошуку необхідно забезпечити виконання таких етапів:

1) вхід у систему;
2) робота процесу авторизації та автентифікації:
   – авторизацію користувачів системи, щоб дати їм можливість створювати мета-дані для документів, які надалі будуть використані у хмарі тегів;
   – перевірка авторизації;
   – повторна авторизація (якщо потрібна);
3) процес роботи з мета-даними:
   – наявність програмної моделі для демонстрації роботи методу застосування метаданих в процесах пошуку;
   – запуск програмної моделі для демонстрації;
   – можливість виконувати набір функцій з електронними даними для процесу створення мета-даних та пошуку по тегам;
4) процес створення мета-сутностей;
5) робота процесу пошуку по тегам;
6) отримання результату;
7) вихід з системи.

Загальна схема методу застосування метаданих в процесах пошуку зображено на рисунку 3.5.

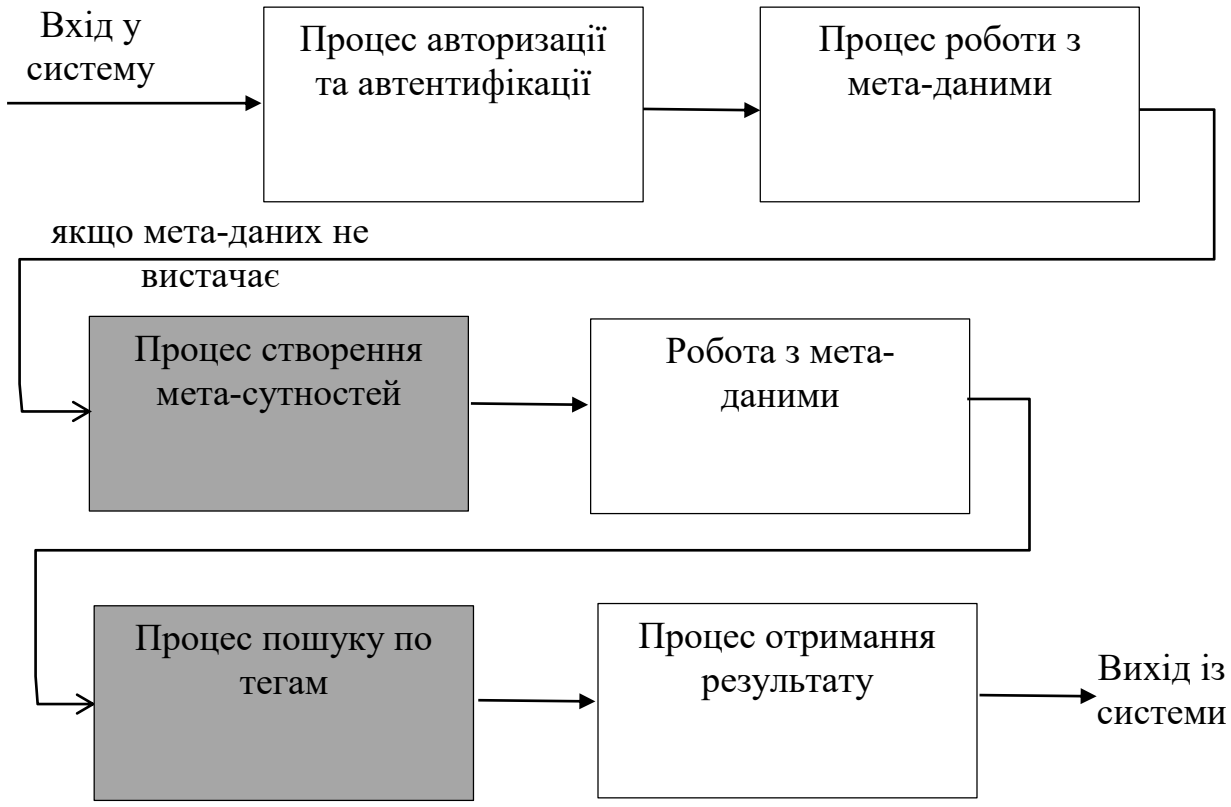


Рисунок 3.5 – Загальна схема методу застосування метаданих в процесах пошуку

3.3 Процес авторизації та автентифікації

Процеси авторизації та автентифікації в даній програмній моделі будується за ілком класичною схемою, де кожен користувач має свій логін і пароль, а також додатково – свій власний файловий простір у вигляді окремого каталога, де зберігається хмара тегів, а також всі створені даним користувачем теги до його документів, що він їх виладає у мережу (у середовищі програмної моделі).

Побудуємо діаграму роботи для процесу авторизації та автентифікації, а далі зобразимо нарисунку 3.6.

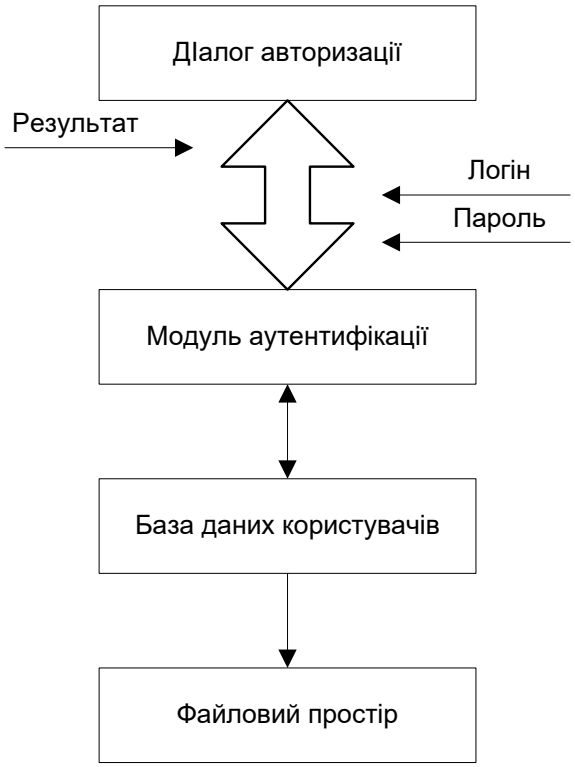


Рисунок 3.6 – Діаграма роботи процесу авторизації та автентифікації

У структурі програмної моделі як БД можна використовувати зовнішній файл. Спроектуємо формат даних для процесу авторизації та автентифікації (рис. 3.7).

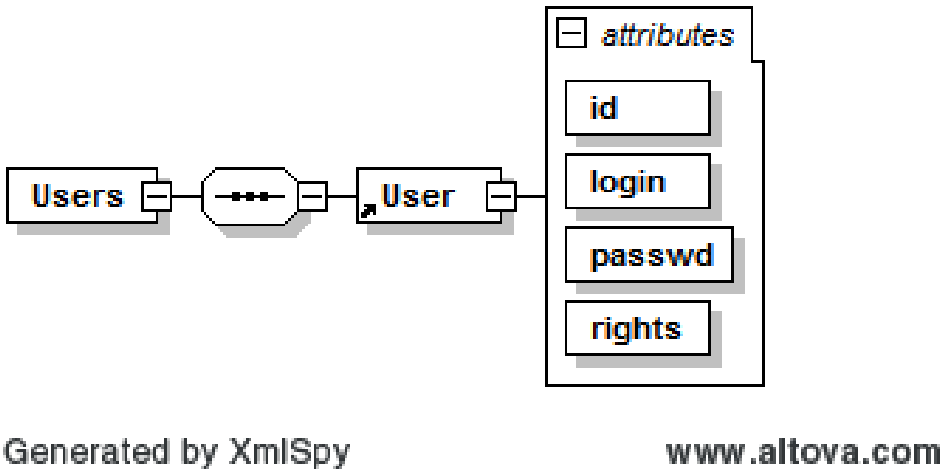


Рисунок 3.7 – Формат даних для процесу авторизації та автентифікації

Тонким місцем даної структури файлу для процесу авторизації та автентифікації є відкрите зберігання пароля. З метою захистити пароль, ми можемо застосувати деякий алгоритм шифрування, наприклад, алгоритм DES, що входить до складу Java Cryptography Extension [2].

Наведемо код, який кодує стрічку тексту за алгоритмом DES, а потім вміє розкодувати. В даному випадку, шифр симетричний, тому для обох цих процесів можна застосувати один і той самий ключ.

```
Cipher ecipher;
Cipher dcipher;
byte[] salt =
{ (byte) 0xA9, (byte) 0x9B, (byte) 0xC8, (byte) 0x32, (byte) 0x56,
   (byte) 0x35, (byte) 0xE3, (byte) 0x03 };
KeySpec keySpec =
      new PBEKeySpec(phrase.toCharArray(), salt, iterationCount);
SecretKey key =
SecretKeyFactory.getInstance("PBEWithMD5AndDES").generateSecret(keySpec);
ecipher = Cipher.getInstance(key.getAlgorithm());
dcipher = Cipher.getInstance(key.getAlgorithm());
AlgorithmParameterSpec paramSpec =
      new PBEParameterSpec(salt, iterationCount);
ecipher.init(Cipher.ENCRYPT_MODE, key, paramSpec);
dcipher.init(Cipher.DECRYPT_MODE, key, paramSpec);
byte[] utf8 = str.getBytes("UTF8");
byte[] enc = ecipher.doFinal(utf8);
String coded = new sun.misc.BASE64Encoder().encode(enc);
byte[] dec = new sun.misc.BASE64Decoder().decodeBuffer(coded);
byte[] utf8 = dcipher.doFinal(dec);
String uncoded = new String(utf8, "UTF8");
```

Лістинг 3.1 – Кодування і декодування за алгоритмом DES

Для спрощення програмної моделі створимо єдиний ключ і для процесу кодування і для процесу декодування всіх паролів, а щоб пароль не зберігався відкрито, «зашиємо» його в сам код програми. Це можна спостерігати у вихідному коді, що наведений в додатках. Маючи такий спосіб захисту пароля, можемо тепер безпечно застосувати формат даних для процесу авторизації та автентифікації який стане БД користувачів, і побудувати алгоритм роботи процесу авторизації та автентифікації.

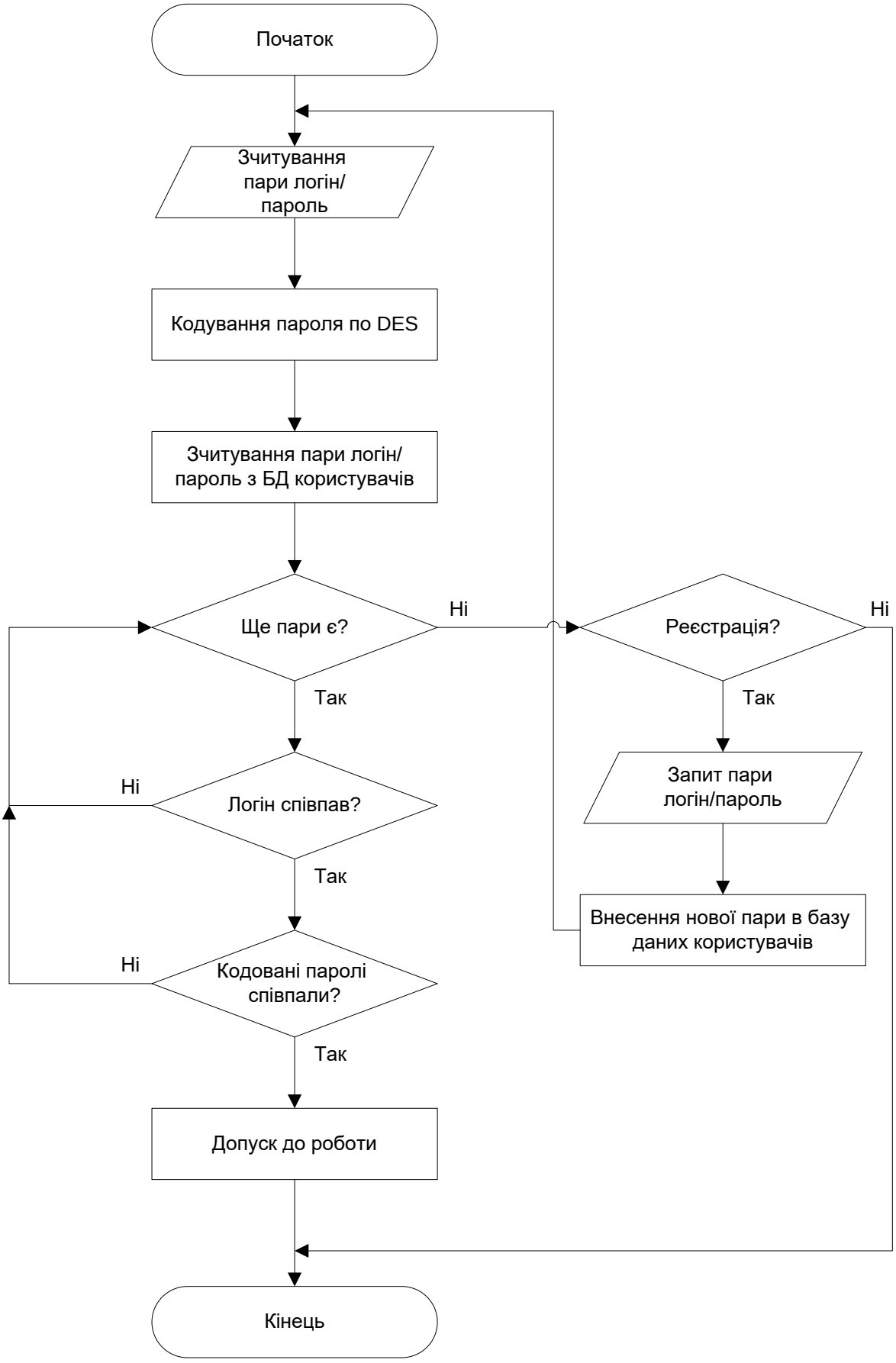


Рисунок 3.8 – Алгоритм роботи процесу авторизації та автентифікації

І завершальним останнім етапом розробки процесу авторизації та автентифікації є його інтерфейсне рішення, яке складається із 2 діалогових вікон, які мають такі функції:

– введення даних;

– реєстрація користувача.

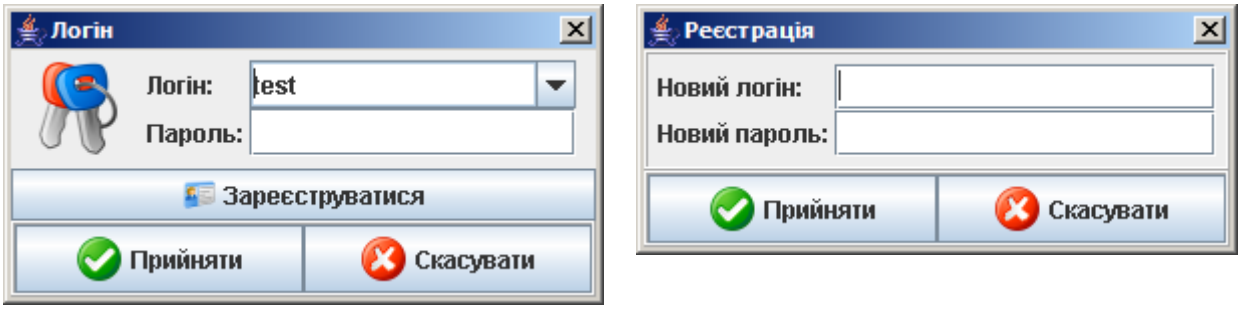


Рисунок 3.9 – Діалогові вікна для авторизації та автентифікації

Повний програмний код підпрограми авторизації та автентифікації, разом із вихідними кодами діалогових вікон, наведено в додатках.

3.4 Процес роботи з мета-даними

Процес роботи з мета-даними передбачає наявність програмної моделі для демонстрації роботи методу застосування метаданих в процесах пошуку. Після запуску цієї програмної моделі для демонстрації має бути можливість виконувати набір функцій з електронними даними для процесу створення мета-даних та пошуку по тегам.

3.4.1 Проектування програмної моделі

Модель застосування метаданих в процесах пошуку є складною для реалізації в повному обсязі. Проте для створення програмної моделі досить обмеженої підмножини, яка дозволяє формувати прості мета-сутності та здійснювати на їх основі пошук із застосуванням метаданих.

Скористаємось як прикладом електронних нотаток, і спроектуємо програмну модель керування електронними нотатками, яка дозволить нам «прикріплювати» до кожної з них мітки-теги, які у поєднанні із мета-сутністю, дозволять формувати «хмари тегів» і, знаходити необхідні нотатки, навіть якщо умова не містить достатньо вичерпного переліку потрібних міток.

Отже, програмна модель, в такому випадку, має відповідати таким основним умовам:

1) Дозволяти користувачу програмної моделі створювати і організовувати електронні нотатки;

2) Дозволяти користувачу програмної моделі створювати мета-сутності, досить складні для нечіткого пошуку;

3) Дозволяти користувачу програмної моделі здійснювати простий, та складний пошук;

4) Програмна модель має мати графічний інтерфейс;

5) Програмна модель має мати систему авторизації та аутентифікації;

6) Програмна модель має дозволяти обробляти одночасно нотатки та мета-сутності;

7) Нотатки та мета-сутності мають створюватись, редагуватись та видалятись лише за допомогою засобів програмного інтерфейсу.

Технології мови програмування Java та метаформату XML як основного носія данихбудуть застосовані для побудови програмної моделі.

Складемо діаграму компонентів програмної моделі, з яких буде складатись сама програма керування електронними нотатками.

Центральною точкою програмної моделі є «хмара тегів», до якої звертаються як панель нотаток, так і панель мета-сутностей. Хмара тегів містить в собі всі семантичні мітки – теги, які використовуються користувачами прпід часи створенні нотаток. На панелі мета-сутностей на їх базі вибудовуються зв’язки, які, потім будуть використані в панелі пошуку.

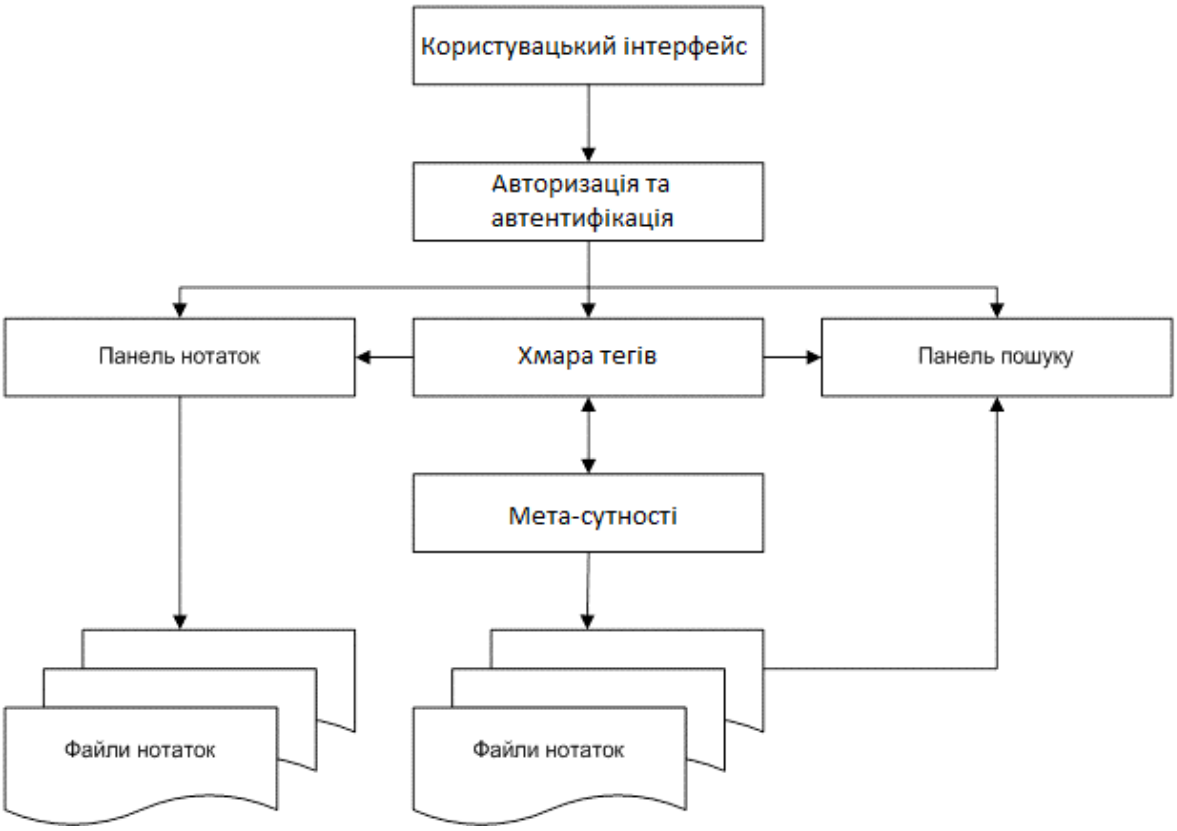


Рисунок 3.10 – Діаграма компонентів програмної моделі

Для того, щоб хмару тегів можна було використовувати для всіх електронних нотаток, необхідно, щоб відповідний компонент програмної моделі витримував такі умови:

1) завжди був один;
2) зберігався на зовнішньому носії, скажімо, у вигляді окремого файлу.

В потужних web-орієнтованих системах так хмари тегів або семантичні хмари зберігаються на спеціальних серверах, і вказуються в особливій частині документів, у якійсь ключовій мета-сутності. Для даної програмної моделі можна обмежитися просто файлом.

В технології Java скористаємось паттерном Singleton, що дозволяє гарантувати, що протягом роботи програмної моделі буде існувати лиш один об’єкт такого типу, і до нього існуватиме лиш одна точка доступу [1]. Глобальним об’єктом, який містить нявні теги, має бути масив елементів, кожен із яких містить одну унікальну мітку. Отже, маємо такий лістинг:

```
public class TagCloud
{
 private static TagCloud cloud = null;
 private Vector tagData = new Vector();

 private TagCloud()
 {
 }
 public static TagCloud getInstance()
 {
  if(cloud == null)
   cloud = new TagCloud();
  return cloud;
 }
}
```

Лістинг 3.2 – Реалізація «хмари тегів» як класу типу Singleton

Тепер побудуємо формат даних «хмари тегів», який дозволить зберегти всі необхідні теги. З цією метою скористаємось метаформатом XML, і на його базі сформуємо такий вектор даних.

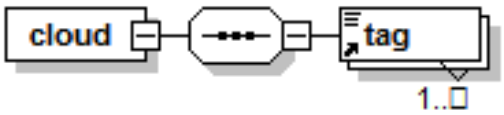


Рисунок 3.11 - Вектор формату даних «хмари тегів» на основі XML

У коді він матиме такий вигляд:

```
<?xml version="1.0" encoding="windows-1251"?>
<cloud>
    <tag>Робота</tag>
    <tag>Програмування</tag>
    <tag>Ruby</tag>
    <tag>C++</tag>
    <tag>Java</tag>
    <tag>Політика</tag>
</cloud>
```

Лістинг 3.3 – Вектор формату даних «хмари тегів»в метаформаті XML

Хмара тегів повинна бути унікальною для кожного користувача, тому ці два компонента тісно взаємопов’язані, як це показано у вихідних кодах класу TagCloud, наведеного в додатках.

3.4.2 Реалізація програмної моделі керування електронними нотатками

Програмна модель керування електронними нотатками має робочий простір, що поділяється на бічну панель, яка містить взаємопов’язані календар і список нотаток, а також центральну зону, де відбувається редагування нотаток та мета-сутностей.

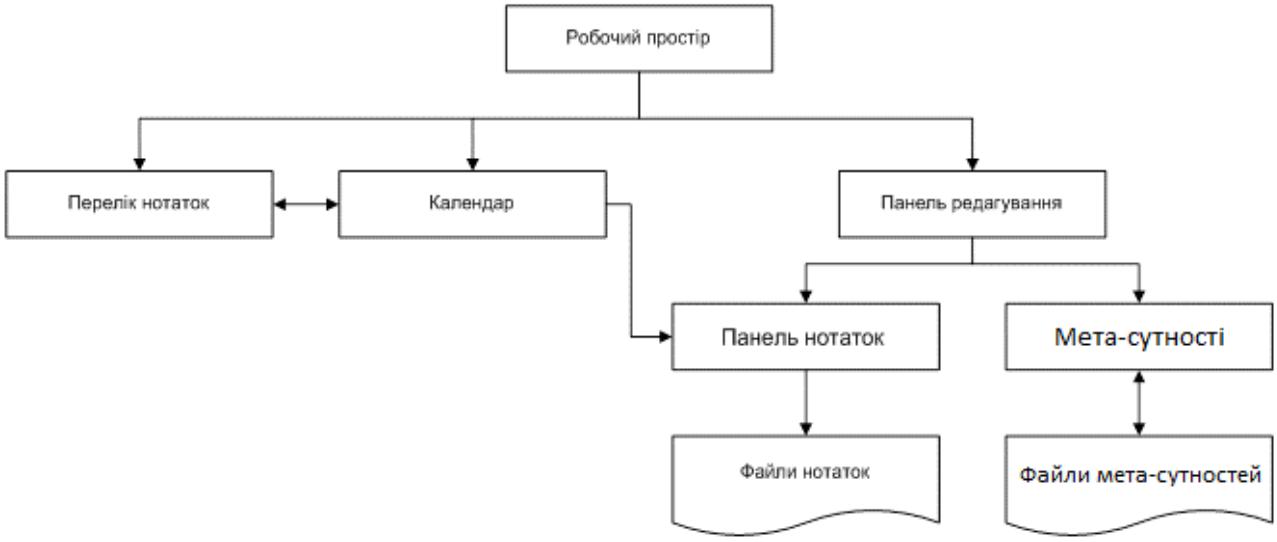


Рисунок 3.12 – Структура програмної моделі керування електронними нотатками

Взаємозв’язки між окремими компонентами програмної моделі керування електронними нотатками були приведені до єдиної ієрархії, що дозволило нам спростити реалізацію з точки зору процесів програмування.

Обидві панелі редагування тісно пов’язані із відповідними форматами даних, проте як видно з рисунка 3.12, панель нотаток пов’язана із календарем, оскільки нотатка може бути відкрита не як файл, а лиш в прив’язці до дати. Такий підхід дає такі переваги:

- дозволяє впорядкувати нотатки;
- при зростанні кількості нотаток – полегшує пошук.

Календар, пов’язаний із переліком нотаток,і це, звісно, дозволяє робити більше ніж одну 1 в добу, і при цьому не вносити плутанини. І під час вибору дати в календарі список автоматично фільтрується і обмежується лиш тими записами, які було зроблено на конктретну дату. Спроектуємо тип даних для нотатки у вигляді XML-схеми на рис. 3.13.

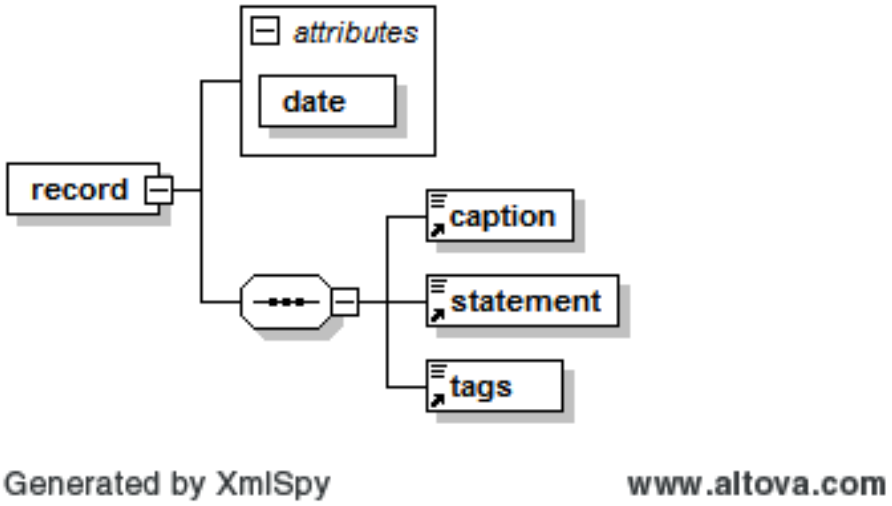


Рисунок 3.13 – Формат даних електронної нотатки

У вигляді реального файлу він буде виглядати так:

```
<?xml version="1.0" encoding="windows-1251"?>
<record date="02.06.2019">
   <caption>Перша нотатка</caption>
   <statement>Сьогодні було написано базовий функціонал.</statement>
   <tags>Робота; Java</tags>
</record>
```

Лістинг 3.4 – Файл електронної нотатки

Виходячи з формату даних електронної нотатки, спроектуємо загальний вигляд робочого простору.

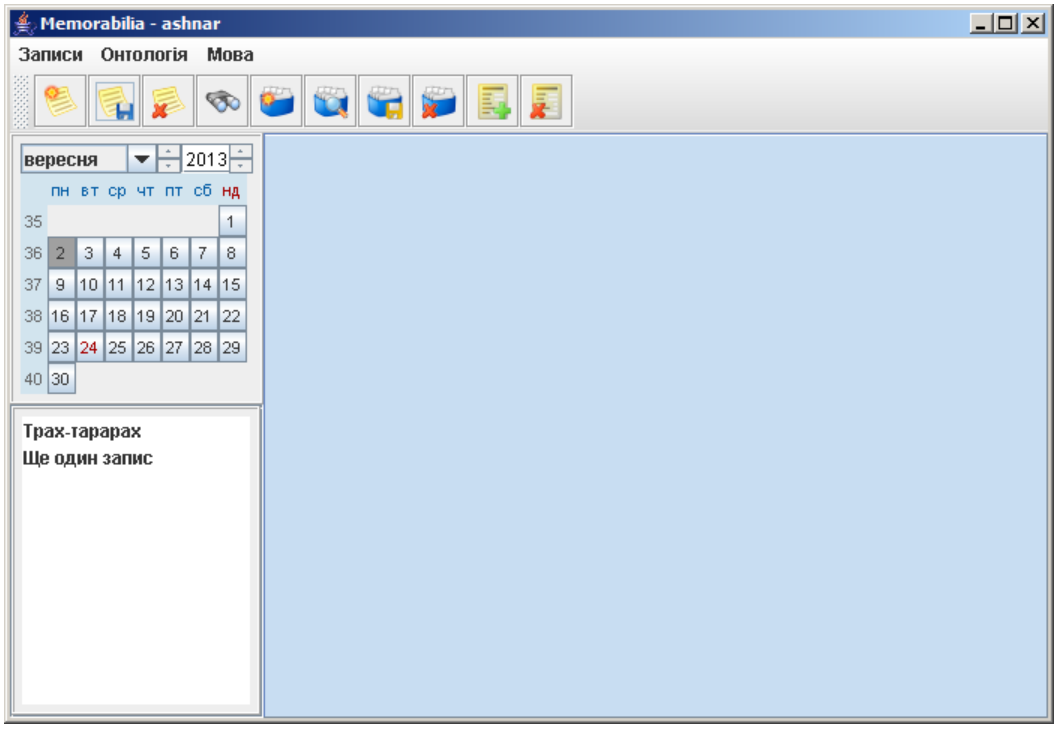


Рисунок 3.14 – Робочий простір електронних нотаток

Так як робочий простір електронних нотаток є головним, то він є одночасно є і головним вікном демонстраційної програми, де є меню та панель швидкого доступу до функцій меню.

Як видно з рисунка, вікно демонстраційної програми розбито на 3 частини:

1) Бічна панель містить календар та перелік нотаток.

2) Центральна частина призначена для редагування даних.

3) Між календарем та переліком є зв’язок на основі відомої схеми «подія-обробник», коли дія користувача над елементом керування (скажімо, вибір дати) викликає виконання відповідного фрагмента програми. В даному випадку завантаження переліку нотаток, які зареєстровані за даною датою.

Складемо алгоритм прив’язування списку нотаток до дати.

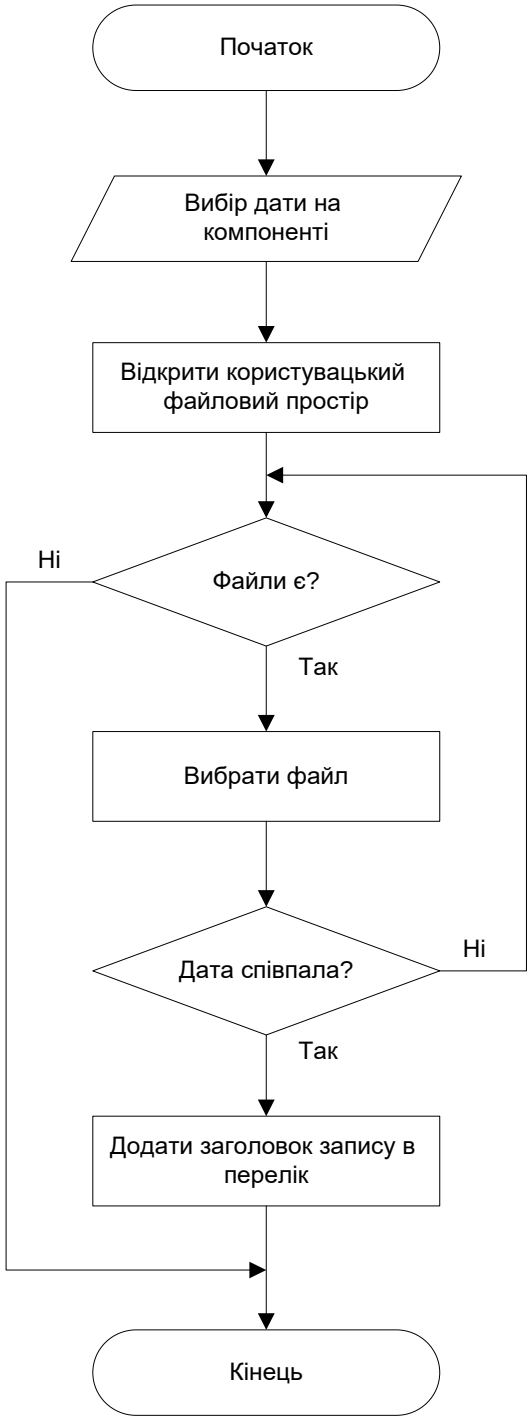


Рисунок 3.15 – Алгоритм прив’язування списку нотаток до дати

Програмний код, який відповідає алгоритму прив’язування списку нотаток до дати, наведено в додатках. Список нотаток пов’язаний із самою панеллю редагування їх. Тепер відслідковуємо не дату нотатки, введену користувачем, а подвійний клік на відповідному заголовку (лістинг, що відображує код зв’язку між переліком нотаток та панеллю редагування знаходиться нижче).

```
public void mouseClicked(MouseEvent e)
 {
  Try   {
   if((e.getClickCount() >= 2) && (e.getSource() instanceof JList))
   {
    File currentDir = new File(".");
    File profileDir = new File(currentDir.getCanonicalPath() + File.separator + "profiles" +
File.separator + Memorabilia.getInstance().getLoginName());
    DateFileFilter filter = new DateFileFilter(dateWidget.getDate());
    File[] records = profileDir.listFiles(filter);
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    for(int i = 0; i < records.length; i++)
    {
     FileInputStream fis = new FileInputStream(records[i].getCanonicalPath());
     Document curDoc = builder.parse(fis);
     fis.close();
     Elem_t documentElem_t = curDoc.getDocumentElement();
     if(Utilities.getTextElement(documentElement, "caption",
"").equals(recList.getSelectedValue().toString()))
     {         int unique = -1;
      for(int j = 0; j < workspace.getTabCount(); j++)
       if(workspace.getComponentAt(j) instanceof RecordPane)
       {
        if( ((RecordPane)workspace.getComponentAt(j))
.getRealName().equals(records[i].getCanonicalPath()) )
         unique = j;
       }
      if(unique == -1)
      {
       RecordPane newPane = new RecordPane();
       newPane.open(records[i].getCanonicalPath());
       workspace.addTab(localize("Record: ") + Utilities.getTextElement(documentElement,
"caption", ""), newPane);
      }
      else
       workspace.setSelectedIndex(unique);       }          }
       }
  } catch (Exception exc)
  {
   System.err.println(exc.getMessage());
   exc.printStackTrace(System.err);
  }
 }
```

Лістинг 3.5 – Лістинг, що відображує код зв’язку між переліком нотаток та панеллю редагування

Лістинг 3.5 характеризує реакцію демонстраційної програми на подвійний клік, саме тому в базовій умові відстежується кількість кліків їх і джерело.

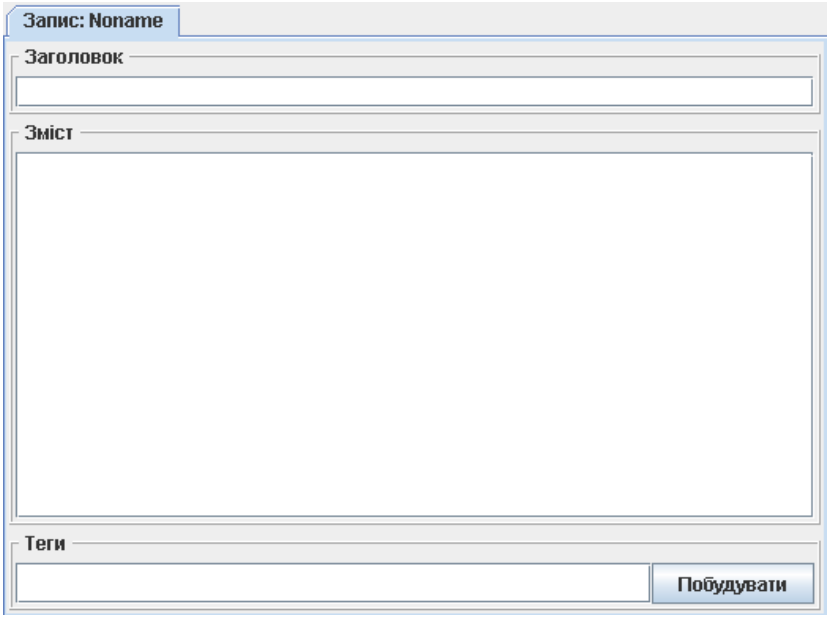


Рисунок 3.15 – Панель редагування нотатки

Панель редагування нотатки виглядає як на рис. 3.15 і містить для поля редаагування. Внизу вказуються теги. Кнопка «Побудувати» будує зв’язок із хмарою тегів, з якої можна обрати 1 або декілька тегів, або ж занести нові.

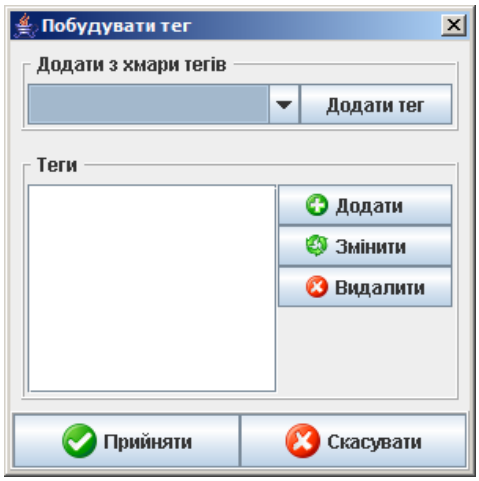


Рисунок 3.16 – Конструктор тегів

Зверніть увагу на комбобокс в верхній частина діалогового вікна конструктора тегів. Він і забезпечує зв’язок із хмарою тегів, завантажуючи в

нього всі елементи, які в ній містяться. Вихідний програмний код, що відповідає діалоговому вікну конструктора тегів, знаходиться в додатках.

3.5 Процес створення мета-сутностей

Для програмної моделі досить обмеженого варіанту семантичних мета-сутностей. Скористаємось для побудови схеми створення мета-сутностей форматом RDF.

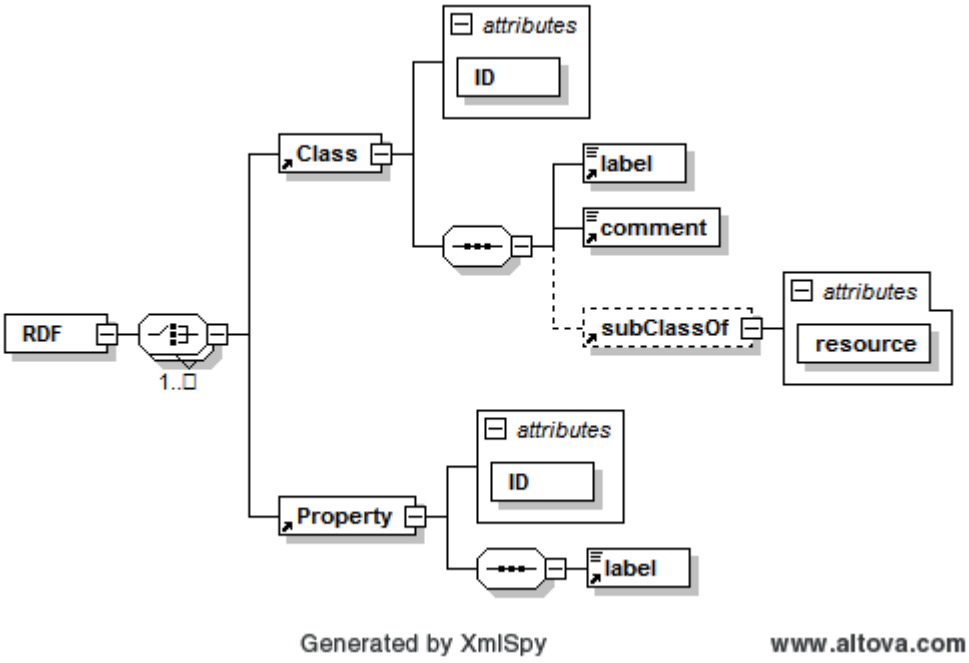


Рисунок 3.17 – Схема RDF для створення мета-сутностей

Відповідно рис. 3.17, підмножина RDF обмежує побудову мета-сутностей одним рівнем логічних структур. Таким чином, елементи хмари тегів характеризується як властивості (Property), що створюють спеціалізовані класи (Class), які можуть складатись лише із властивостей, або містити якісь інші класи. Для створення таких класів необхідна окрема панель редагування.

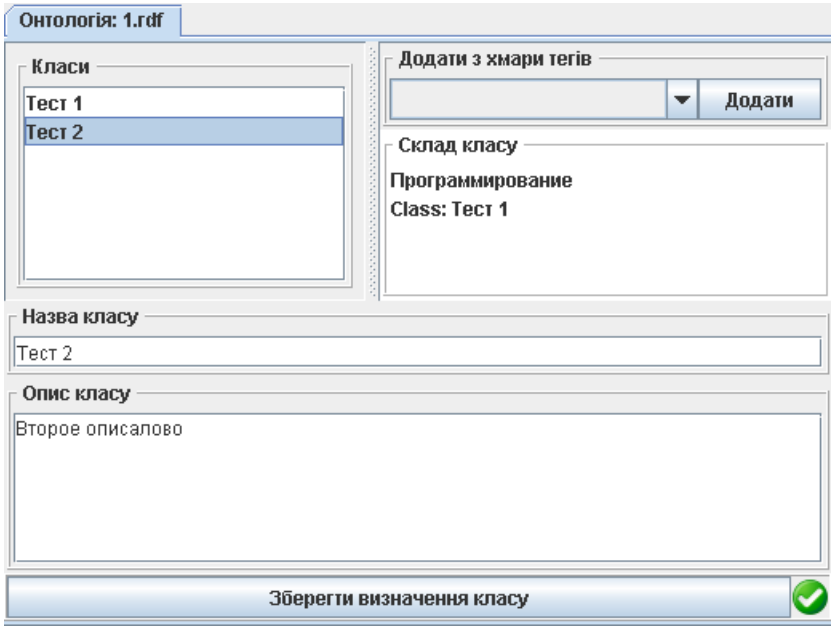


Рисунок 3.18 – Панель редагування мета-сутностей

Панель редагування мета-сутностей також розділена на 3 складових. В верхній частині панелі редагування мета-сутностей є 2 списка:

1) класи, визначені в даній мета-сутності (їх можна редагувати);
2) вміст класу.

На відміну від панелі редагування електронних нотаток, ця не прив’язана до календаря і дат, тому дозволяє створювати й окремо зберігати мета-сутності як окремі файли в будь-якому місці файлової системи.

Файли мета-сутностей вони цілком переносимі, і мета-сутність, створену в просторі одного користувача, можна застосувати в просторі іншого. Під час переносу всі елементи хмари тегів іншого користувача можуть бути додані в хмару тегів даного користувача під час імпорту мета-сутності.

Файл мета-сутностей, який утворюється в результаті зберігання, матиме такий вигляд:

```
<?xml version="1.0" encoding="windows-1251"?>
<rdf:RDF>
   <rdfs:Class rdf:ID="Тест 1">
      <rdfs:label>Тест 1</rdfs:label>
      <rdfs:comment>Первое описалово</rdfs:comment>
      <rdfs:subClassOf rdf:resource="Тест 2"/>
   </rdfs:Class>
```

```
    <rdf:Property rdf:ID="Тест 1.Работа">
      <rdfs:label>Работа</rdfs:label>
    </rdf:Property>
    <rdfs:Class rdf:ID="Тест 2">
      <rdfs:label>Тест 2</rdfs:label>
      <rdfs:comment>Второе описалово</rdfs:comment>
    </rdfs:Class>
    <rdf:Property rdf:ID="Тест 2.Программирование">
      <rdfs:label>Программирование</rdfs:label>
    </rdf:Property>
</rdf:RDF>
```

Лістинг 3.6 – Приклад файлу мета-сутності

Створений автором програмний код для маніпулювання вмістом мета-сутностями та представлення класів RDF, наведений в додатках.

3.6 Процес пошуку по тегам

Мета-сутності, що створюються в режимі панелі редагування мета-сутностей, можна застосувати і для процесу пошуку із застосуванням метаданих. Процес пошуку в даній програмній моделі реалізований за допомогою трьох стратегій:

1) пошук по заголовкам;
2) пошук за змістом;
3) нечіткий пошук за даною мета-сутністю.

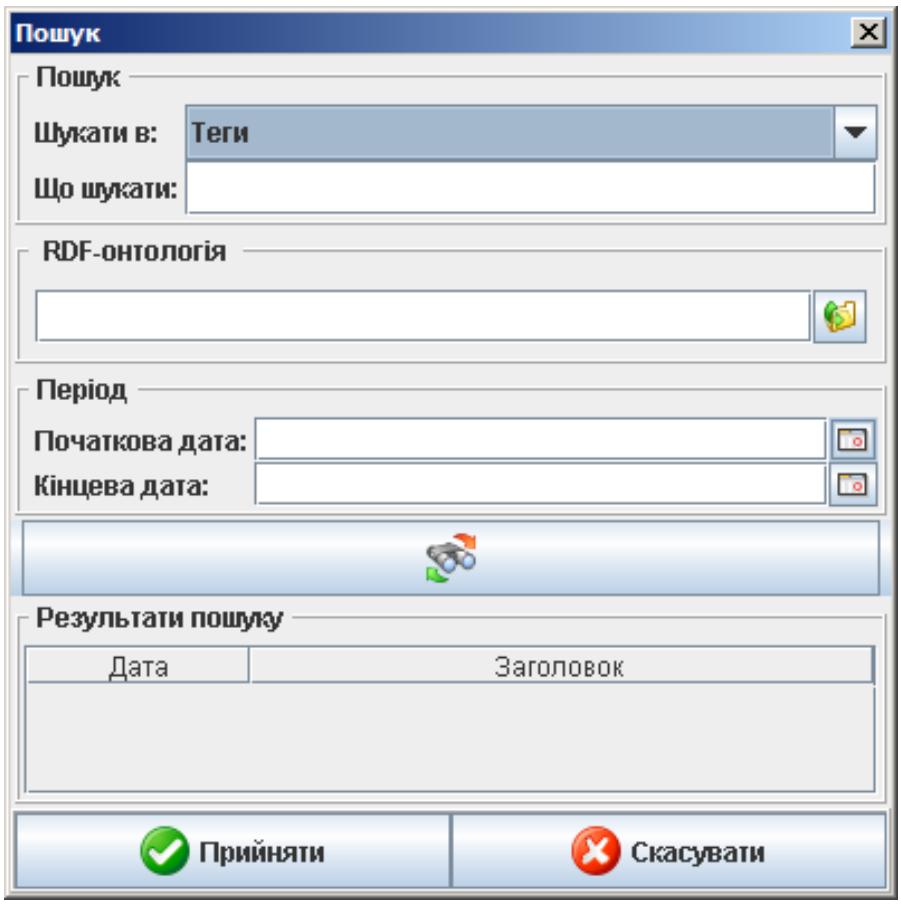


Рисунок 3.19 – Діалогове вікно пошуку

Як видно з даного діалогового вікна пошуку, в програмі створено три типи фільтрування нотаток, що точно відповідають наведеним вище. Крім цих фільтрів існує фільтр дат, який дозволяє шукати нотатки лише в заданому часовому періоді. Якщо пошук відбувається по заголовку, чи тексту нотатки, процес пошуку зводиться до пошуку підстрічки в тексті, то процес нечіткого пошуку має зовсім іншу природу.

Складемо алгоритм пошуку по тегам, який характеризує такий тип пошуку. Задача, яка ставилась при розробці даного програмного засобу є максимальний результат пошуку при мінімумі введених даних, алгоритм буде «жадібним», тобто таким, що навіть при виникненні незначних відповідностей, знайдений факт буде заноситись в результати пошуку.

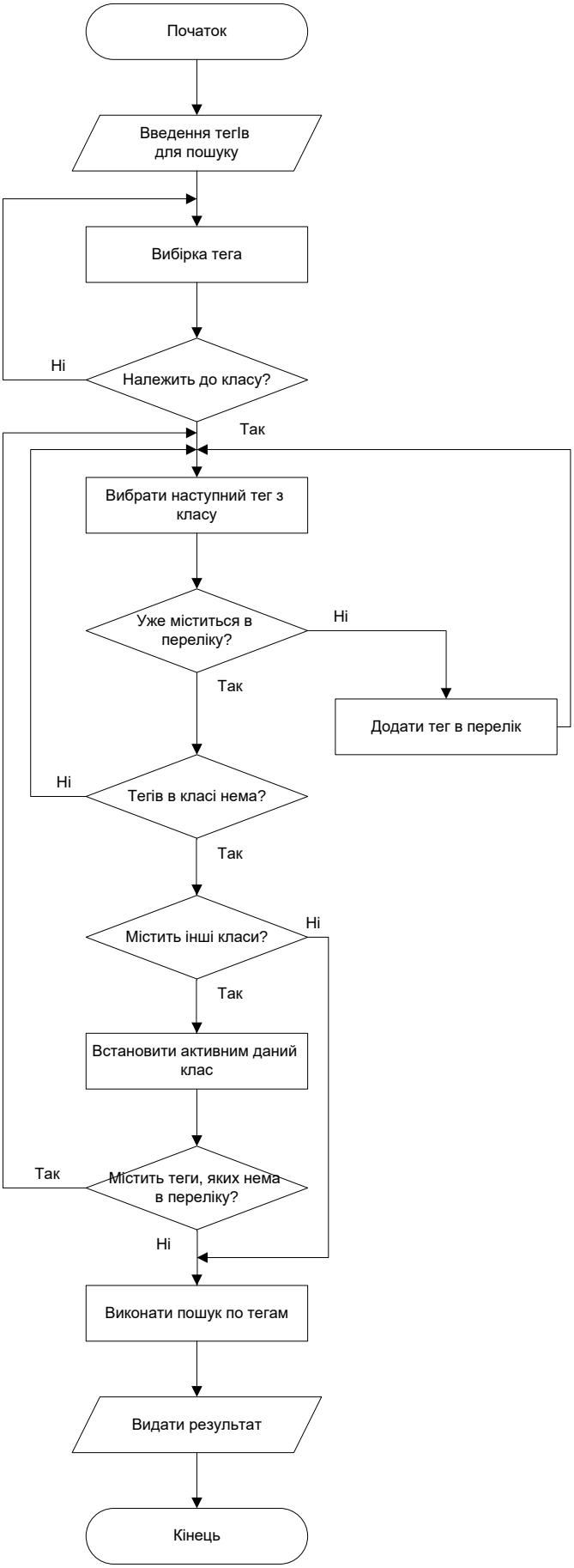


Рисунок 3.20 – Алгоритм пошуку по тегам

Теги вказуються через крапку із комою у вигляді рядка. Можна перетворити рядок тегів на вектор, що значно спростить додання нових тегів до списку тегів. Алгоритм такого розбиття вже є реалізованим мовою програмування Java, наведемо відповідний фрагмент програмного коду.

```
String tagstr = data.toString();
StringTokenizer tokenizer = new StringTokenizer(tagstr, ";");
Vector tags = new Vector();
while(tokenizer.hasMoreTokens())
 tags.add(tokenizer.nextToken().toString().trim());
tagList.setListContent(tags);
```

Лістинг 3.7 – Перетворення рядка на вектор

Створивши таку «пошукову хмару тегів», можна легко розширити БД пошуку, одночасно при цьому не збільшуючи кількість тегів пошуку. З метою полегшення роботи користувача програмною моделлю, забезпечимо можливість відкриття нотаток безпосередньо з діалогового вікна пошуку.

Для цього нам потрібна таблиця результатів пошуку. Зробимо її «read only», інакше подвійний клік буде сприйнятий як сигнал редагування таблиці. Отже. створимо підклас і перевизначити службовий метод isCellEditable так, щоб для кожної комірки поверталось негативне значення, яке переведе таблицю в режим «read only».

```
public boolean isCellEditable(int row, int column)
 {
  return false;
 }
```

Лістинг 3.8 – Метод, необхідний для таблиці «лиш для читання»

Тепер відстежимо подвійний клік мишею:

```
public void mouseClicked(MouseEvent e)
 {
  if(e.getClickCount() >= 2)
  {
   if(e.getSource() instanceof JTable)
   {
    int current = resultsPane.getSelectedRow();
```

```
((Memorabilia)Memorabilia.getInstance()).openFromExternal(foundFiles.get(current).toString());
          }
         }
        }
```

Лістинг 3.9 – Відстеження подвійного кліка в таблиці результатів

З метою забезпечити відкриття електронної нотатки, треба звертатись до головного класу Memorabili, що реалізований як екземпляр виду Singleton, що гарантує, що об’єкт програми буде лиш одині тоді всі ключові функції виносити саме туди.

3.7 Програмний засіб застосування метаданих в процесах пошуку

3.7.1 Алгоритм роботи програми

У даній програмі з метою звертання до функцій меню або для передачі команд з діалогових вікон, треба гарантувати, що екземпляр головного класу програми завжди був єдиним. Для цього використаємо паттер Singleton, побудуємо клас Memorabilia, з приватним конструктором. Це забезпечить перехід до робочого простору, і одночасно підвищить захищеність користувацьких даних (лістинг 3.10).

```
        setLoginAdapter(new LocalLoginAdapter(new LoginDialog(), new RegisterDialog()));
         JMenuBar menuBar = new JMenuBar();
         JToolBar tools = new JToolBar();
         JMenu menu = new JMenu(localize("Records"));
         menu.add(Utilities.createMenuItem(localize("New Record"), "ctrl N", "newRecord", this,
"record_add.png", tools));
         menu.add(Utilities.createMenuItem(localize("Save Record"), "ctrl S", "saveRecord", this,
"record_save.png", tools));
         menu.add(Utilities.createMenuItem(localize("Close Record"), "ctrl W", "closeRecord",
this));
         menu.add(Utilities.createMenuItem(localize("Erase Record"), "ctrl K", "killRecord", this,
"record_erase.png", tools));
         menu.add(new JSeparator());
         menu.add(Utilities.createMenuItem(localize("Search"), "alt F7", "search", this,
"search.png", tools));
         menu.add(new JSeparator());
         menu.add(Utilities.createMenuItem(localize("Quit"), "alt X", "quit", this));
         menuBar.add(menu);
           menu = new JMenu(localize("Ontology"));
```

```
        menu.add(Utilities.createMenuItem(localize("New Ontology"), "", "newSemantic", this, "onto_new.png", tools));
        menu.add(Utilities.createMenuItem(localize("Edit Ontology"), "", "editSemantic", this, "onto_load.png", tools));
        menu.add(Utilities.createMenuItem(localize("Save Ontology"), "", "saveSemantic", this, "onto_save.png", tools));
        menu.add(Utilities.createMenuItem(localize("Close Ontology"), "", "closeSemantic", this, "onto_close.png", tools));
        menu.add(new JSeparator());
        menu.add(Utilities.createMenuItem(localize("New class"), "", "newClass", this, "class_new.png", tools));
        menu.add(Utilities.createMenuItem(localize("Remove class"), "", "killClass", this, "class_kill.png", tools));
        menuBar.add(menu);
          menuBar.add(getLangMenus());
        setJMenuBar(menuBar);
```

Лістинг 3.10 – Об’єднання всіх функцій в меню

З перших стріок лістингу 3.10 видно, що одразу підключається підсистема авторизації та аутентифікації.

Складемо алгоритм роботи головного класу програми (рисунок 3.21).

З рисунку 3.21 можемо бачити, що всі складові програмного засобу застосування метаданих в процесах пошуку спроектовані та реалізовані автором в рамках даного розділу магістерської кваліфікаційної роботи об’єднуються в єдину інформаційну систему, а оскільки паттерн Singleton гарантує лише один екземпляр цього типу, то процес об’єднання складових програми відбувається не лише на інтерфейсному та компонентному рівні, а й на рівні системному, так би мовити, «зшиваючи» програму в єдине ціле.

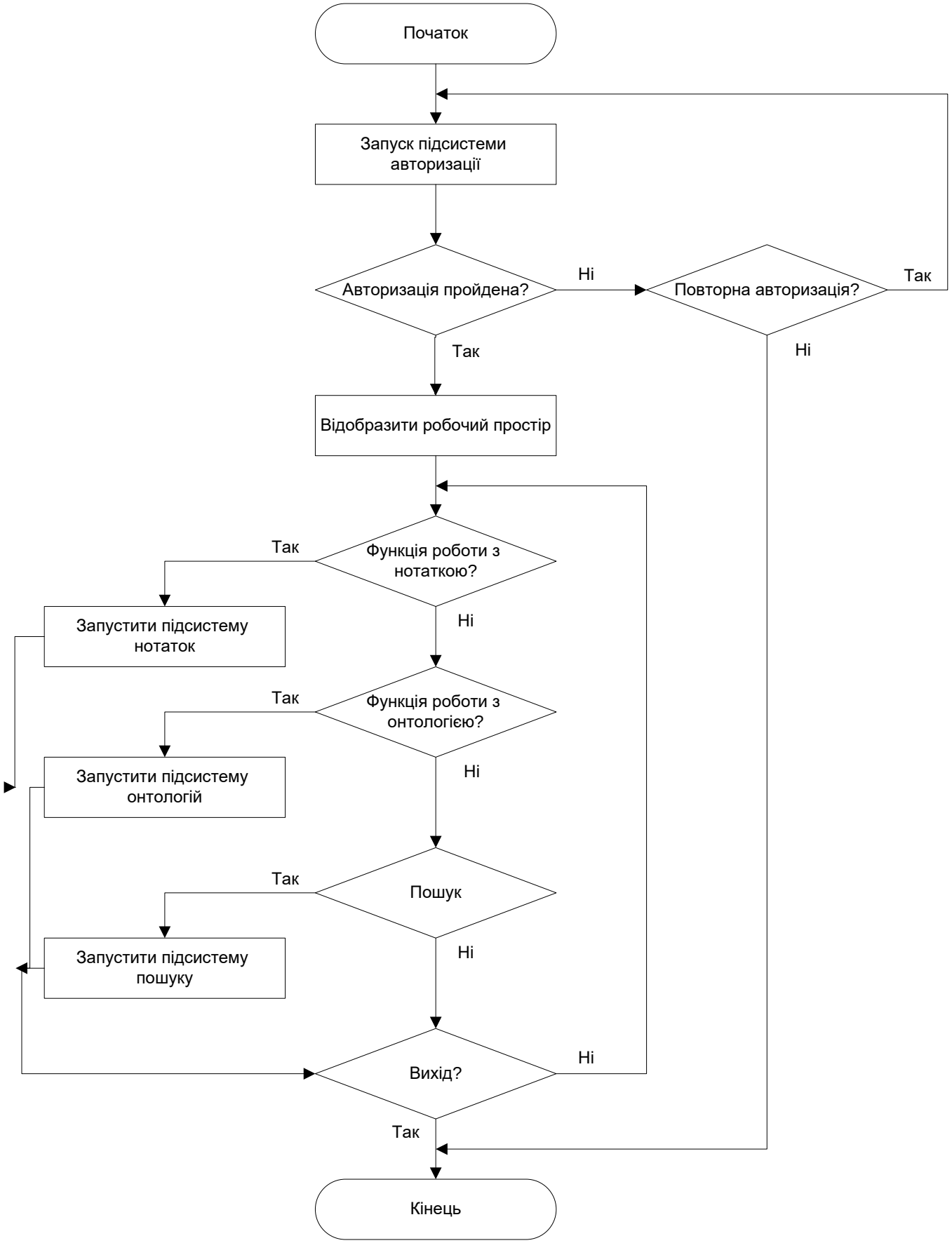


Рисунок 3.21 – Алгоритм функціонування головного модуля програми

3.7.2 Цикл роботи з демонстраційною моделлю в тестовому режимі

Робота із демонстраційною моделлю починається з роботи процесу авторизації. На рис. 3.22 показано відповідне діалогове вікно:

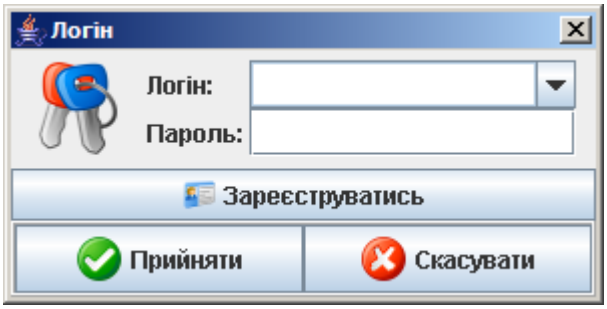


Рисунок 3.22 – Вікно авторизації

Якщо новий логін ще не створено, треба натиснути кнопку Зареєструватись. З’явиться діалог для введення нового логіна та пароля.

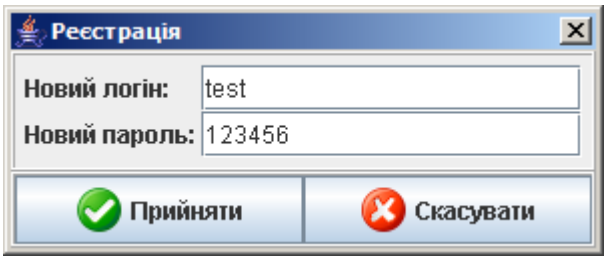


Рисунок 3.23 – Вікно реєстрації нового користувача

Зверніть увагу: якщо у звичайному діалоговому вікні пароль «забивається» спеціальними символами, то тут він показаний прямо. Це єдине місце, де пароль використовується в такій формі. В інших фігурують або кодові відповідники, або пароль «приховується». Після реєстрації створюється не лише обліковий запис користувача, а й його окремий файловий прості, де будуть збережені всі його нотатки.

Якщо процедуру авторизації провалено, система запропонує або пройти її ще раз (кількість спроб необмежена), або вийти з програми взагалі. Якщо процедуру пройдено, на екрані з’явиться робочий простір.

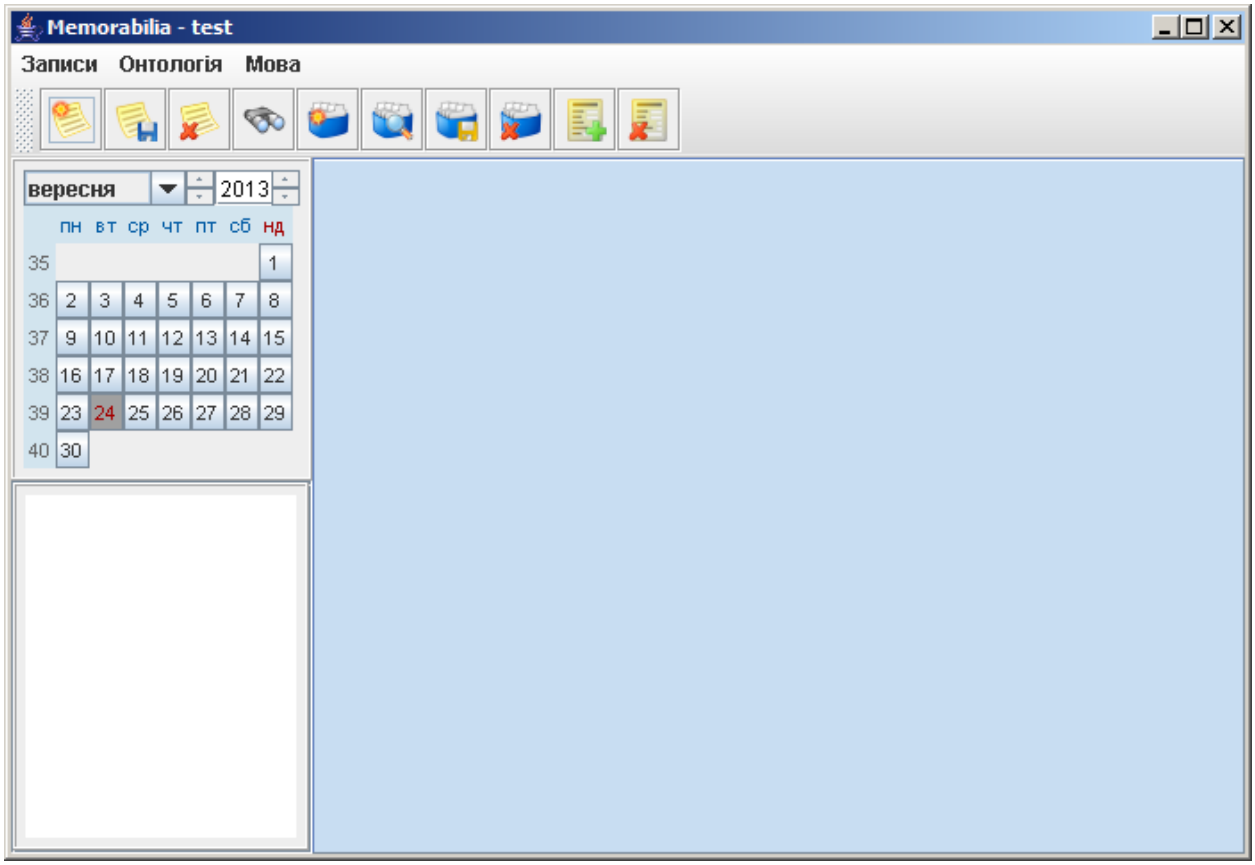


Рисунок 3.24 – Робочий простір програми

Подальша робота програми керується функціями меню та кнопками з лінійки швидкого доступу. Кожна з останніх має спливаючу підказку, що спрощує інтуїтивне освоєння програми.

Рисунок 3.25 – Панель швидкого доступу до функцій (із підказкою)

Оскільки основним призначенням програми є ведення електронних нотаток, перелічимо для початку функції, що відносяться до них:

 - створення нової електронної нотатки;

 - зберігання поточної електронної нотатки;

 - видалення поточної електронної нотатки.

Команда створення викликає появу в робочому просторі панелі редагування нотатки. Одночасно можна створювати кілька таких нотаток.

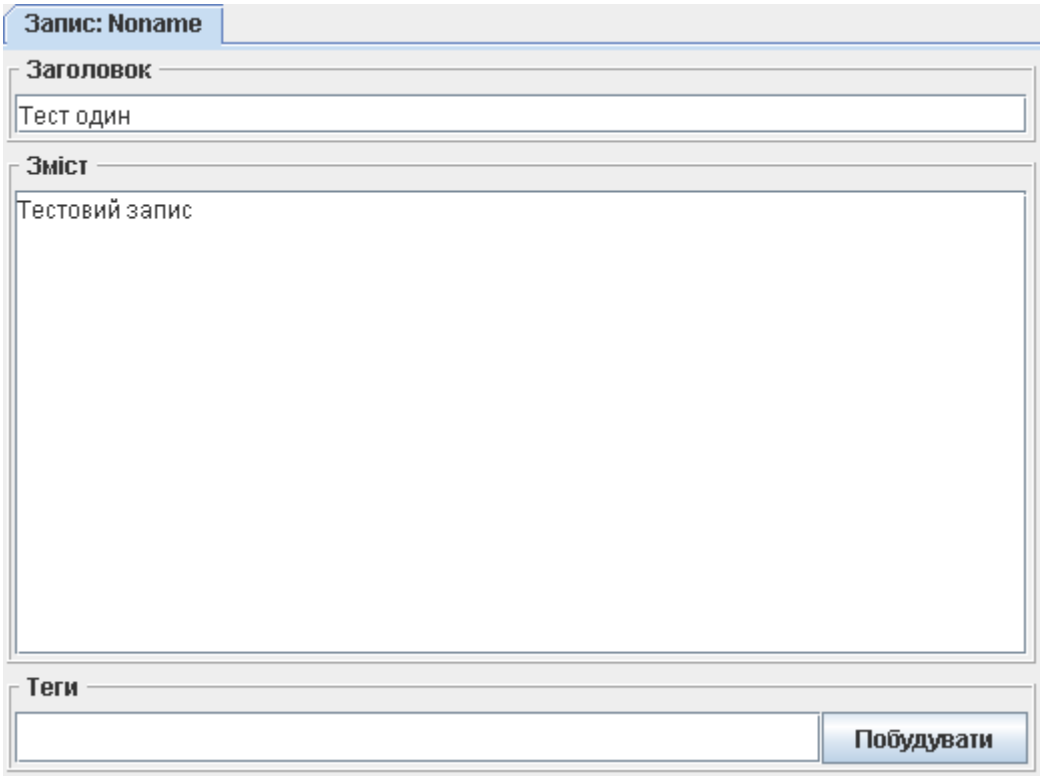


Рисунок 3.26 – Панель редагування нотатки

Зверніть увагу, до зберігання нотатки в файловому просторі користувача в заголовку вкладки відображається службова назва «Noname». Якщо ж виконати зберігання, то назва з’явиться не лише в заголовку, а й в списку записів в робочій області.

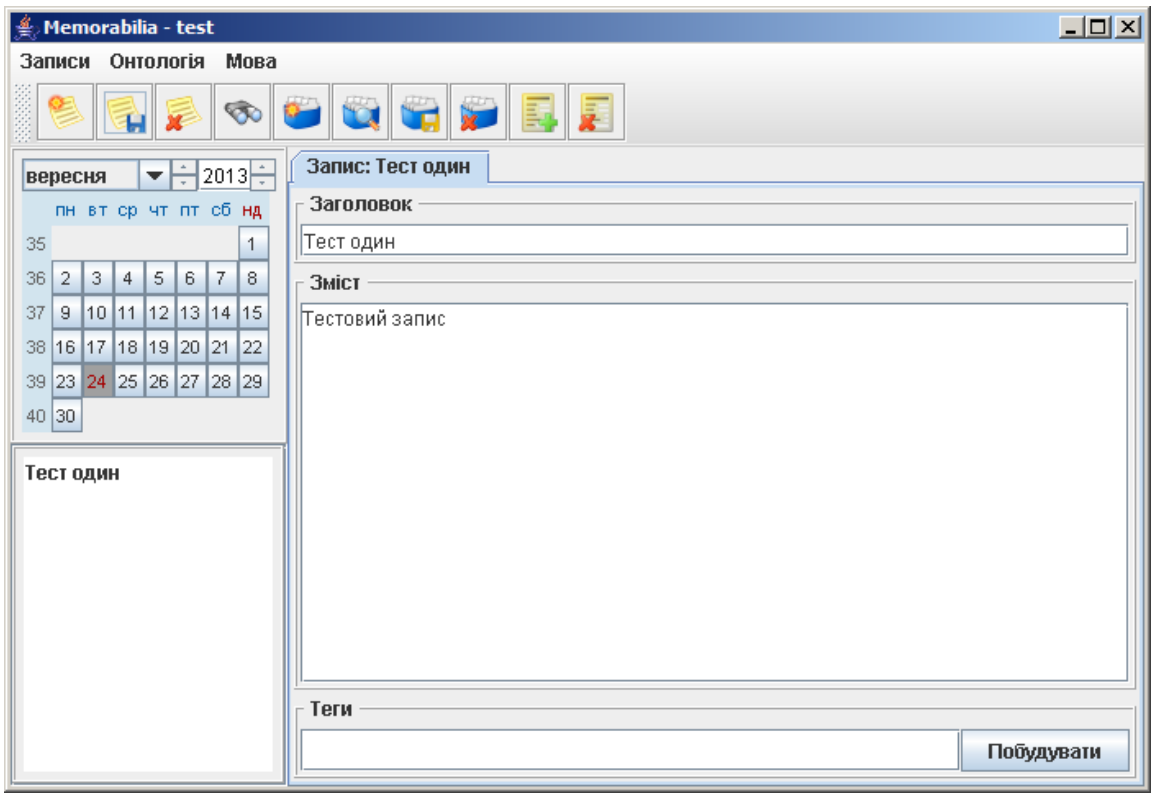


Рисунок 3.27 – Результат зберігання нотатки

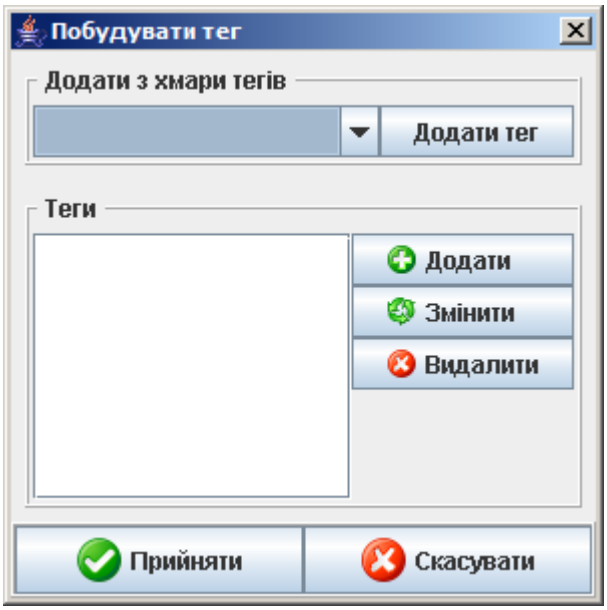


Рисунок 3.28 – Майстер побудови тегів

В подальшому її можна буде відкрити, вибравши в календарі потрібну дату, і клацнувши двічі на заголовку в переліку записів. Функція видалення

нотатки видаляє її фізично. Натомість, якщо треба просто закрити нотатку, досить натиснути «гарячу комбінацію» Ctrl-W, або вибрати відповідну функцію з меню Записи**.**

Для кожної нотатки можна сформувати комплект міток-тегів, які в подальшому сформують загальну хмару тегів і забезпечуватимуть пошук записів нечітким методом. Для цього можна ввести їх або вручну через точку з комою (;), або скористатись майстром побудови.

Всі нові теги, які вводяться користувачем в цьому діалоговому вікні, автоматично будуть внесені в хмару тегів після підтвердження побудови. Для введення переліку нових тегів використовуються бокові кнопки. Можна додати і вже існуючі, з хмари тегів, за допомогою випадаючого меню вгорі.

Наступним видом роботи є створення мета-сутностей. Розглянемо основні функції:

 - створення нової мета-сутності;

 - відкриття і редагування існуючої мета-сутності;

 - зберігання існуючої мета-сутності;

 - закриття мета-сутності.

На відміну від нотаток, окремої функції відкриття яких не існує – вони відкриваються лиш через систему календар-перелік заголовків, мета-сутності можна відкрити з будь-якого місці на користувацькому жорсткому диску. Це пов’язано із тим, що мета-сутність також є способом хмари тегів від одного користувача до іншого, або з однієї робочої станції на іншу. Під час відкриття мета-сутності всі невідомі семантичні мітки автоматично додадуться до поточної семантичної хмари. Створення ж нової мета-сутності викликає на екран панелі редагування.

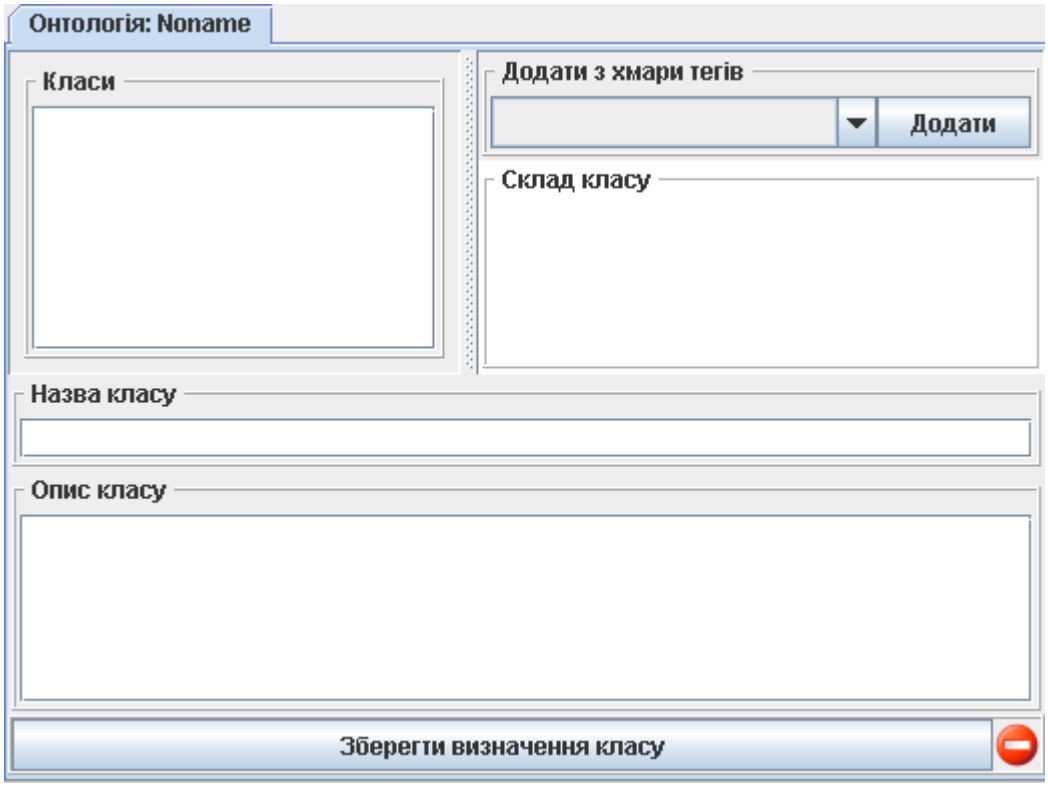


Рисунок 3.29 – Панель редагування мета-сутності

Мета-сутність складається із класів, які репрезентують логічні групи семантичних міток. Створити новий клас можна за допомогою функції Онтологія => Новий клас. Вона також винесена на панель швидкого доступа.

Заповнивши поля в нижній частині діалогового вікна, а також вказавши склад класу (додавати елементи можна лиш із хмари тегів) слід обов'язково натиснути кнопку Зберегти формулювання класу. Збереженість чи незбереженість формулювання класу показує іконка в правому нижньому кутку панелі. Якщо зберегти формулювання, вона зміниться на зелену.

Рисунок 3.30 – Іконка збережності формулювання

До складу нового класу можна додавати не лише теги з хмари, а й вже створені попередньо класи, отож створюючи ієрархії, які дозволятимуть в

подальшому гнучко формувати пошукові хмари. Видалити клас можна за допомогою функції Видалити клас. Вона присутня як в меню Онтологія, так і на панелі швидкого доступа. Наступним етапом роботи є пошук. Діалогове вікно пошуку показано на рис. 3.31.

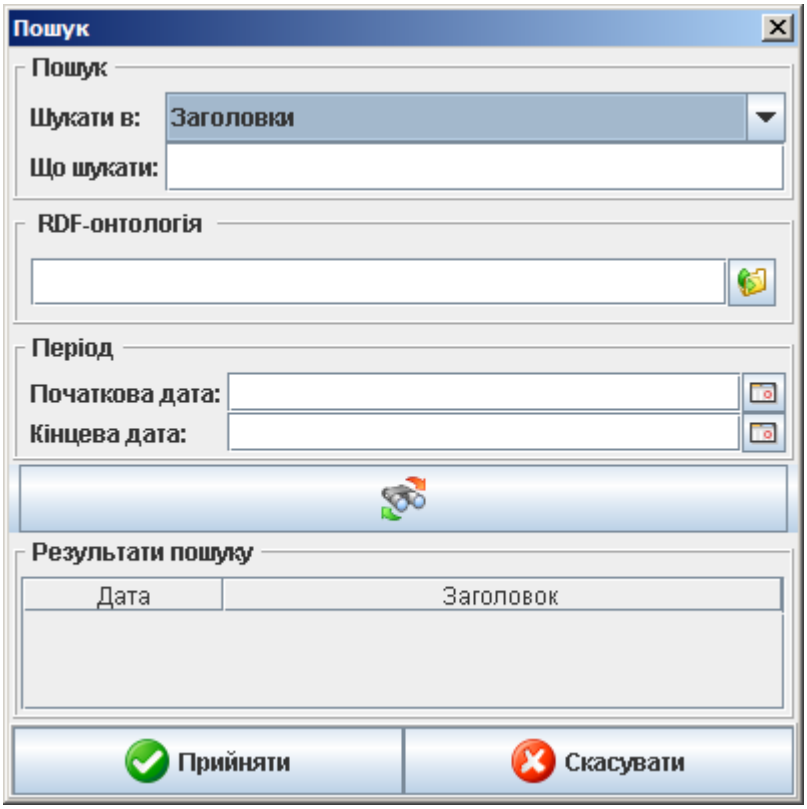


Рисунок 3.31 – Діалогове вікно пошуку

Існує три режими пошуку: а) в заголовку; б) в тексті; в) за мета-сутністю. Щоб вибирати поміж ними, необхідно вибрати відповідне значення у випадаючому меню Шукати в.... Нижче, в полі Що шукати, необхідно ввести шуканий текст. У випадку заголовка чи текста, тут можна вказати частину потрібного слова чи виразу. У випадку мета-сутності необхідно точно вказати тег, який стане основою пошукової хмари.

У обох випадках знайдені записи можуть бути профільтровані відповідно початковій та кінцевій даті періода. Це дозволяє звужувати пошук, якщо умова виявилась надто загальною. Внизу, в табличці будуть висвітлені результати

пошуку. Якщо на рядку із відповідним записом двічі клацнути, він буде відкритий в робочій області для перегляду чи редагування.

Якщо необхідно шукати за мета-сутністю, окрім вищевказаних параметрів, слід вказати файл мета-сутності в полі RDF-онтологія. Варіюючи різні файли мета-сутностей, можна варіювати пошукову хмару, і отож робити пошук максимально гнучким.

Нарешті, останньою важливою функцією в даній програмі є можливість локалізації. За нього відповідає меню Мови. Наразі можливо користуватись нею із російським та англійським інтерфейсом.

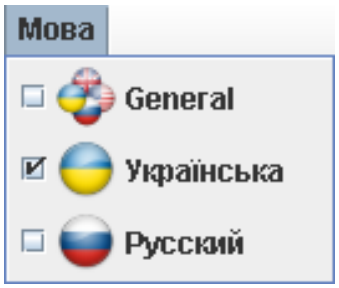


Рисунок 3.32 – Меню мови

Для того, щоб використовувати дану програму іншою мовою, необхідно позначити відповідний пункт в меню, і перезапустити її. В подальшому можлива локалізація на інші мови.

3.8 Оцінка роботи методу на прикладі демонстраційної моделі

Проведемо порівняльний аналіз, що допоможе оцінити результати пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів на прикладі демонстраційної моделі.

Позиції, за якими оцінюватимемо метод застосування метаданих в процесах пошуку занесемо у таблицю 3.1 та емпірично оцінимо по шкалі від 0 до 10 у порівнянні із результатами пошуку засобів-аналогів із застосування

метаданих в процесах пошуку, але без використання методу застосування метаданих в процесах пошуку.

Таблиця 3.1 – Оцінка методу застосування метаданих в процесах пошуку

	Memoria	Smart Diary Suite	iDaily Diary	Chrysanth Diary
питання моделювання	10	10	10	9
класифікація документів	8	7	8	7
фільтрація документів	10	8	9	9
кластеризація документів	8	9	8	7
проектування архітектур пошукових систем і користувальницьких інтерфейсів	9	6	7	8
отримання інформації, зокрема анотування і реферування документів	8	8	7	7
мови запитів	10	9	7	7
формулювання запиту природною мовою	10	9	8	8
вибір пошукових системи і сервісів	9	9	8	7
формалізація запиту	10	8	9	8
проведення пошуку в одній або декількох пошукових системах	8	8	8	7
огляд отриманих результатів (посилань)	9	9	8	6
попередня обробка отриманих результатів:	9	9	8	6
модифікація запиту і проведення повторного (уточнюючого) пошуку з подальшою обробкою отриманих результатів	10	8	7	6

Відобразимо у вигляді графіку результати дослідження (рисунок 3.33).

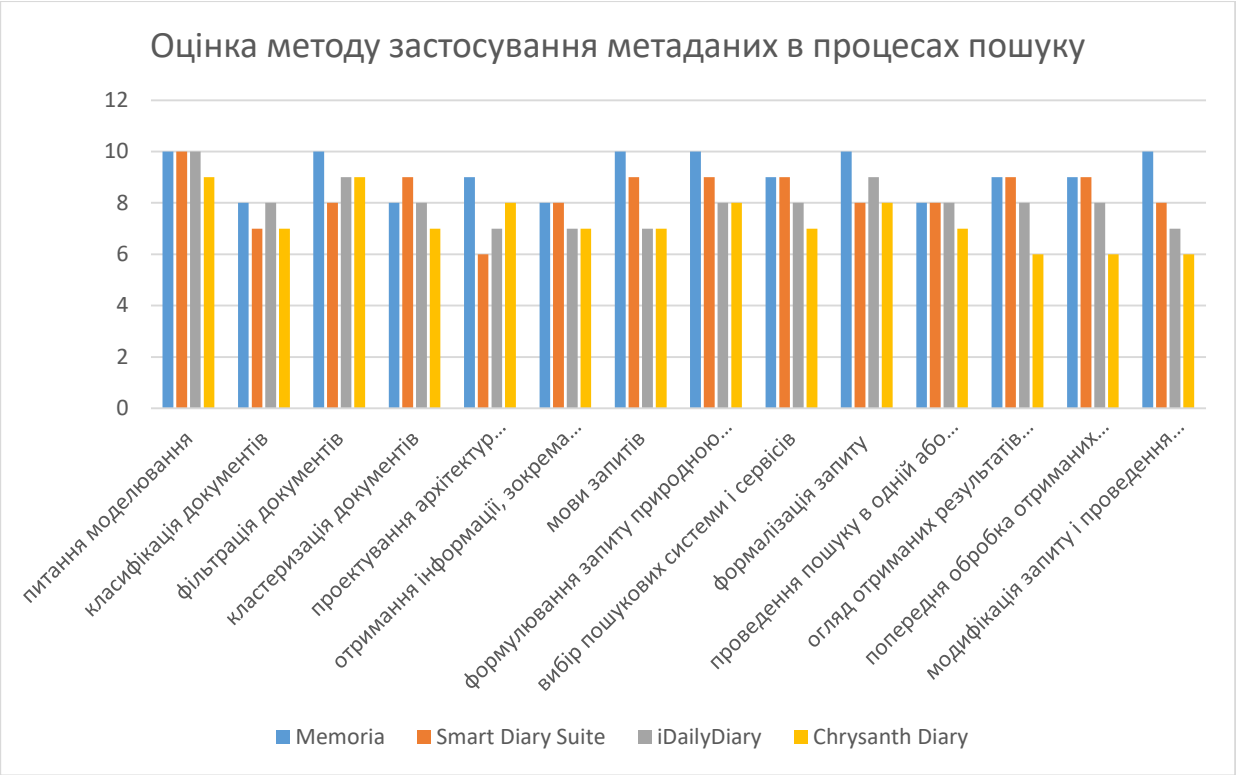


Рисунок 3.33 – Оцінка методу застосування метаданих в процесах пошуку

Запропонований метод застосування метаданих в процесах пошуку, що реалізований у демонстраційній моделі Memoria в сумарному показнику дозволяє збільшити результатів пошук документів, що задовольняють запиту, в рамках деякої статичної колекції документів на такий показник:

- на 8% у порівнянні з Smart Diary Suite
- на 3% у порівнянні з iDaily Diary
- на 8% у порівнянні з Chrysanth Diary

У середньому, запропонований метод застосування метаданих в процесах пошуку, що реалізований у демонстраційній моделі Memoria в інтегральному показнику дозволяє збільшити результатів пошук документів, що задовольняють запиту, в рамках деякої статичної колекції документів на 12,7% у порівнянні із аналогами.

3.9 Висновки за розділом

В даному розділі було проаналізовано практичні аспекти реалізації семантичного підходу «хмари тегів». Зокрема, була розглянута модель частотного розподілу тегів, її недоліки та можливі методи розв’язку основних проблем – таких як фолксономічна класифікація. У якості основи було прийнято сітку Рангатана, яка дозволяє узгоджувати між собою різні класифікаційні системи.

На основі теорії множин було запропоноване математичне представлення рішення, та фасетна формула, яка дозволяє утворити індексний масив для пошуку за допомогою простих алгоритмів – пошуку в глибину та рекурсії. У якості основного методу була обрана рекурсія, через простоту реалізації.

Було спроектовано демонстраційний приклад, який дозволяє показати, яким чином можна здійснювати нечіткий пошук на основі хмари тегів та семантичних мета-сутностей.

Спочатку було висунуто основні вимоги до програми, надалі їх було конкретизовано до окремих процесів, і нарешті – до алгоритмів та вихідних кодів. Для готової програми було складено цикл роботи, так що будь-який користувач може самостійно освоїти її за невеликий проміжок часу і самостійно будувати семантичні структури для пошуку.

Даний демонстраційний приклад дозволяє майже повністю проілюструвати принципи семантичного нечіткого пошуку.

ВИСНОВКИ

З метою полегшити процеси пошуку і виключити аналіз всього тексту в пошуках ключових слів та виразів, в даній роботі пропонується використовувати «мета-теги», які в стислій формі описують вміст ресурсу, які в кінцевому підсумку, супроводжуються «хмарою тематичних тегів».

Результатом виконання магістерської кваліфікаційної роботи є збільшення результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів. Було спроектовано демонстраційний приклад, який дозволяє показати, яким чином можна здійснювати нечіткий пошук на основі хмари тегів та семантичних мета-сутностей.

Наукова новизна одержаних результатів полягає в такому:

- вперше запропоновано метод застосування метаданих в процесах пошуку. Запропонований у даній роботі метод застосування метаданих в процесах пошуку дозволяє збільшити результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів;
- вдосконалено модель застосування метаданих в процесах пошуку, що дозволяє створення користувачем власних мета-сутностей;
- вдосконалено процес роботи з мета-даними, який дозволяє створення семантичних ланцюжків між поняттями;
- вдосконалено процес створення мета-сутностей, які дозволяють зменшувати кількість тегів у хмарі тегів.

Запропонований метод застосування метаданих в процесах пошуку, що реалізований у демонстраційній моделі в інтегральному показнику дозволяє збільшити результатів пошук документів, що задовольняють запиту, в рамках деякої статичної колекції документів на 12,7% у порівнянні із аналогами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Э. Тоффлер Третья волна – М., АСТ, 2016;
2. V. Bush As We May Think – The Athlantic Monthly, 1 July, 2005 - http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/
3. A. Kay The Early History of Smalltalk – ACM, 2013 - http://www.smalltalk.org/smalltalk/TheEarlyHistoryOfSmalltalk_TOC.html
4. Хороший план №9 // Хакер, 29.07.2014 - http://www.xakep.ru/post/23246/default.asp
5. C. Bartneck, J. Hu Scientometric Analysis of The CHI Proceedings - Eindhoven University of Technology, Dept. of Industrial Design, - April, 2019
6. S. Brin, L. Page The Anatomy of a Large-Scale Hypertextual Web Search Engine – Stanford University, Computer Science Dept., 2018
7. H. Li, I. Councill, L. Bolelli, D. Zhou, Y. Song, and others CiteSeer – A Scalable Autonomous Scientific Digital Library – Pennsylvania State University, 2017
8. S. Bergler From Lexical Semantics To Text Analysis – Montreal, Concordia University, Computer Science Dept., 2014
9. Y. Yang A Re-examination of Text Categorization Methods – Carnegie-Mellon University, School of Computer Science, 2018
10. K. Nigam, A. McCallum, S. Thrun, T. Mitchell Text Classification from Labeled and Unlabeled Documents using EM – Carnegie-Mellon University, School of Computer Science, 2016
11. M. Brent Automatic Acquisition of Subcategorization Frames from Untagged Text – MIT, AI Lab, 2015
12. Л. А. Савицька, Т. І. Коробейнікова, В. Д. Тягун. МЕТОД ТА ПРОГРАМНИЙ ЗАСІБ ЗАСТОСУВАННЯ МЕТАДАНИХ В

ПРОЦЕСАХ ПОШУКУ. Інформаційні технології та комп'ютерна інженерія – В. : ВНТУ, 2019 – №3 (46). Подана до друку.

13. R. Kimball, R. Merz The Data Warehouse Toolkit: Building the Web-Enabled Data Warehouse – Wiley, 2012
14. A. Powell, M. Nilsson, A. Naeve, P. Jonhston, T. Baker DCMI Abstract Model – DCM Initiative, 2017 - http://dublincore.org/documents/abstract-model/
15. Open Metadata Framework – Metalab - http://www.ibiblio.org/osrt/omf/
16. R. Goldman, J. Widom DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases – Stanford University, 2017
17. Extensible Markup Language (XML) 1.0 (Fifth Edition) - W3C Recommendation, 26 November 2018
18. T. Berners-Lee The Semantic Web – // The Scientific American, 17 May 2011
19. T. Berners-Lee, R. Fielding, L. Masinter Uniform Resource Identifier (URI): Generic Syntax - Network Working Group, January 2015 (http://tools.ietf.org/html/rfc3986)
20. Unicode 6.3.0 - http://www.unicode.org/versions/Unicode6.3.0/
21. Namespaces in XML 1.0 (Third Edition) - W3C Recommendation, 8 December 2009 (http://www.w3.org/TR/REC-xml-names/)
22. XML Schema Part 0: Primer Second Edition - W3C Recommendation, 28 October 2014 (http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/)
23. XQuery 1.0: An XML Query Language - W3C Recommendation, 14 December 2016 (http://www.w3.org/TR/xquery/)
24. Guidance on expressing the Dublin Core within the Resource Description Framework - http://www.ukoln.ac.uk/metadata/resources/dc/datamodel/WD-dc-rdf/
25. Resource Description Framework (RDF): Concepts and Abstract Syntax - W3C Recommendation, 10 February 2014 (http://www.w3.org/TR/rdf-concepts/)

26. RDF Semantics - W3C Recommendation, 10 February 2014 (http://www.w3.org/TR/2004/REC-rdf-mt-20040210/)
27. G. Phipps Comparing Observed Bug and Productivity Rates for Java and C++ - // Software, Practice and Experience, 2016
28. Comparison of Java and C++ - http://en.wikipedia.org/wiki/Comparison_of_Java_and_C++
29. Models, Social Tagging and Knowledge Management - http://www.column2.com/2017/09/models-social-tagging-and-knowledge-management-bpm2009-bpms209/
30. T. Vander Wal Explaining and Showing Broad and Narrow Folksonomies - http://www.vanderwal.net/random/entrysel.php?blog=1635
31. J. S. Poulin , K. P. Yglesias Experiences with a Faceted Classification Scheme in a Large Reusable Software Library - //17th Annual International Computer Software and Applications Conference, 2013
32. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес Приемы объектно-ориентированного программирования. Паттерны проектирования – СПб, Питер, 2015
33. JCE Encryption – Data Encryption Standard (DES) Tutorial – Mkyong.com, http://www.mkyong.com/java/jce-encryption-data-encryption-standard-des-tutorial/, 2009
34. E. Biham, A. Shamir Differential Cryptanalysis of DES-like Cryptosystems – The Weizmann Institute of Science Department of Applied Mathematics, 2010
35. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт / Уклад. В. О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ

д.т.н., проф. Мартинюк Т.Б.

_____

«04» жовтня 2019року

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської дипломної роботи

«МЕТОД ТА ПРОГРАМНИЙ ЗАСІБ ЗАСТОСУВАННЯ МЕТАДАНИХ В ПРОЦЕСАХ ПОШУКУ»

08-023.МКР.014.00.000.ТЗ

Керівник: к.т.н., доц. каф. ОТ

Савицька Л. А.

(підпис) (прізвище та ініціали)

Виконав: студент 2 курсу, групи 1КІ-18м

спеціальність 123 «Комп'ютерна інженерія»

(шифр і назва напряму підготовки)

Тягун Д. Т.

(підпис) (прізвище та ініціали)

м. Вінниця – 2019 р.

1. Підстава для виконання магістерської дипломної роботи (МДР)

1.1Стрімкий, майже експоненційний ріст потоку нових наукових розробок означає, що в адекватні часові терміни практично мало хто може встигнути ознайомитись із цими матеріалами, що й призводить до того, що гіпотетично перспективні технології навіть не були помічені ні науковим суспільством в цілому, ні гіпотетичним ринком користувачів.

Це формує вже досить давно **актуальну задачу** ефективного пошуку інформації в спеціалізованих системах, архівах, і в тому числі, і в мережі Інтернет. Звісно, потужні пошукові системи Google або CiteSeer дозволяють значно полегшити людині пошук і оцінку релевантності знайдених результатів. Однак, на сучасному етапі розвитку досі необхідний науковий та творчий пошук якщо не цілковито нових підходів, то можливо нетривіального використання уже існуючих, що дозволили б покращити ситуацію.

1.2 Наказ по ВНТУ № 254 від 2 жовтня 2019 р. про затвердження теми магістерської кваліфікаційної роботи.

2. Мета і призначення ДР

З метою полегшити процеси пошуку і виключити аналіз всього тексту в пошуках ключових слів та виразів, в даній роботі пропонується використовувати так звані «мета-теги», які в стислій формі описують вміст сторінки, текстових даних чи якогось ресурсу. Таким чином, кожна веб-сторінка, чи документ, супроводжуються «хмарою тематичних тегів». Звісно їх кількість, на жаль, кінцева, а із швидким ростом інформаційного потоку, на кожен такий тег припадатиме все більше і більше даних, а це, відповідно, вимагатиме додаткових уточнюючих тегів, і зрештою, кожен документ треба буде супроводжувати такою кількістю тегів, що їх індексація за своїм обсягом не буде відрізнятись від повнотекстової інформації.

Метою дослідження магістерської кваліфікаційної роботи є збільшення результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів.

3. Вихідні дані для виконання МДР

Список технічної літератури, аналіз, вивчення та дослідження сучасних моделей, методів та засобів застосування метаданих в процесах пошуку, технічне завдання на магістерську роботу.

4. Матеріали, що подаються до захисту МДР

Пояснювальна записка до МДР, графічні і ілюстративні матеріали, протокол попереднього захисту МДР на кафедрі, відгук наукового керівника, відзив рецензента, протоколи складання державних екзаменів, анотації до МДР українською та іноземною мовами.

5. Техніко-економічні показники

5.1 Витрати на програмні засоби, що використовуються в ході даної розробки, повинні бути мінімальними.

5.2 Лімітна ціна на програмне забезпечення не повинна перевищувати ціну аналога.

6. Порядок контролю виконання та захисту МДР

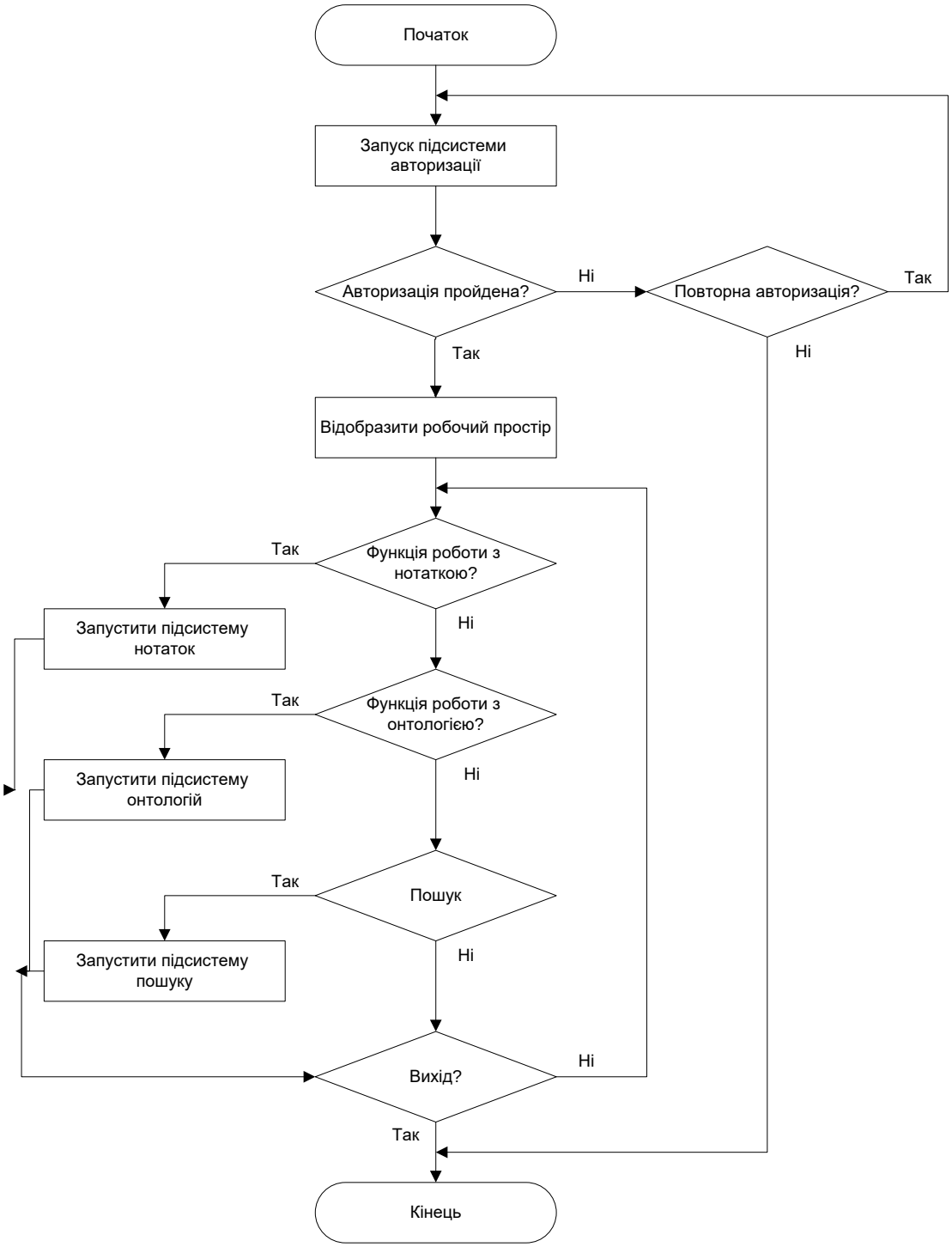
6.1.Робота виконується в три етапи, таблиця 6.1.

Таблиця 6.1 – Етапи виконання роботи.

Етап	Зміст	Початок	Кінець	Результат
1	Інформаційний пошук та огляд літературних джерел. Розробка методу та програмної реалізації застосування метаданих в процесах пошуку.	14.11.2018	14.06.2019	Розділи 1 та 3.

2	Техніко-економічне обґрунтування теми дипломної роботи, розробка програмного засобу.	18.02.2019	25.04.2019	Чернетки матеріалів 1 розділу. Попередній захист.
3	Підготовка матеріалів пояснювальної записки.	12.10.2019	10.12.2019	Пояснювальна записка.

7. Загальний алгоритм роботи програми є таким:



8. Порядок контролю та прийому

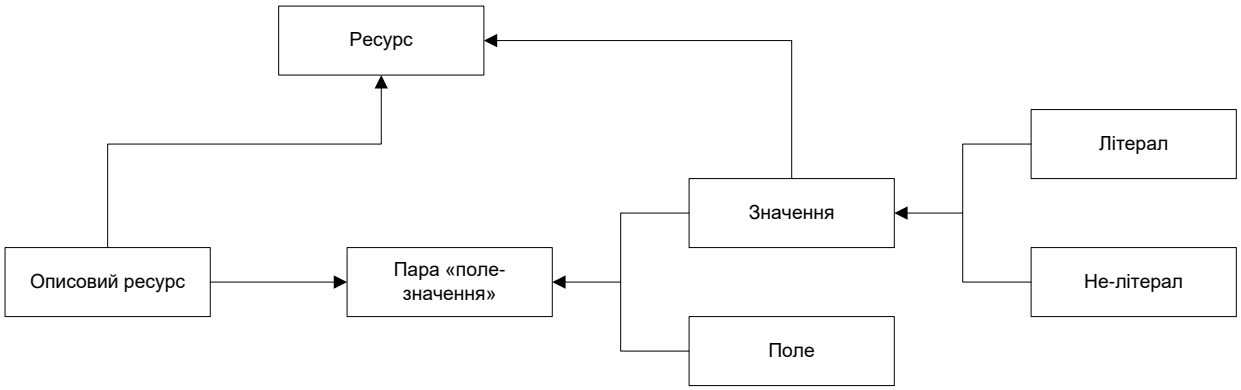
8.1 До приймання дипломної роботи надається:

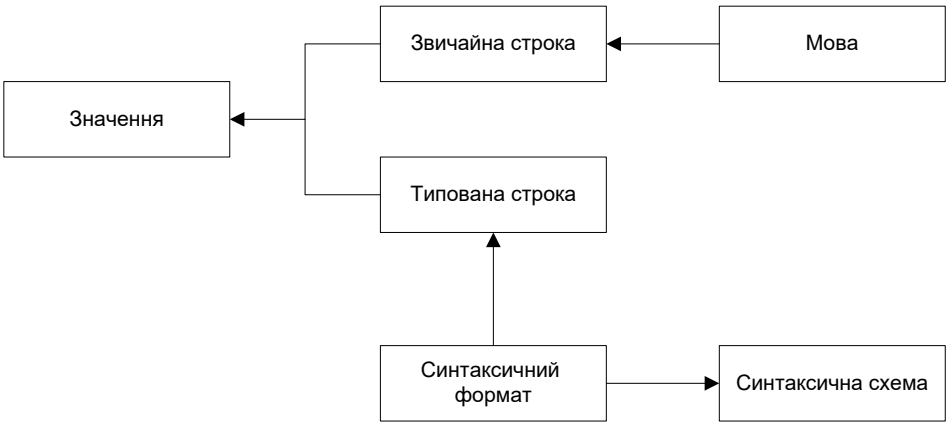
- пояснювальна записка з відповідними узгодженнями;
- демонстрація програмного засобу;
- презентація;
- відгук керівника роботи та рецензента;

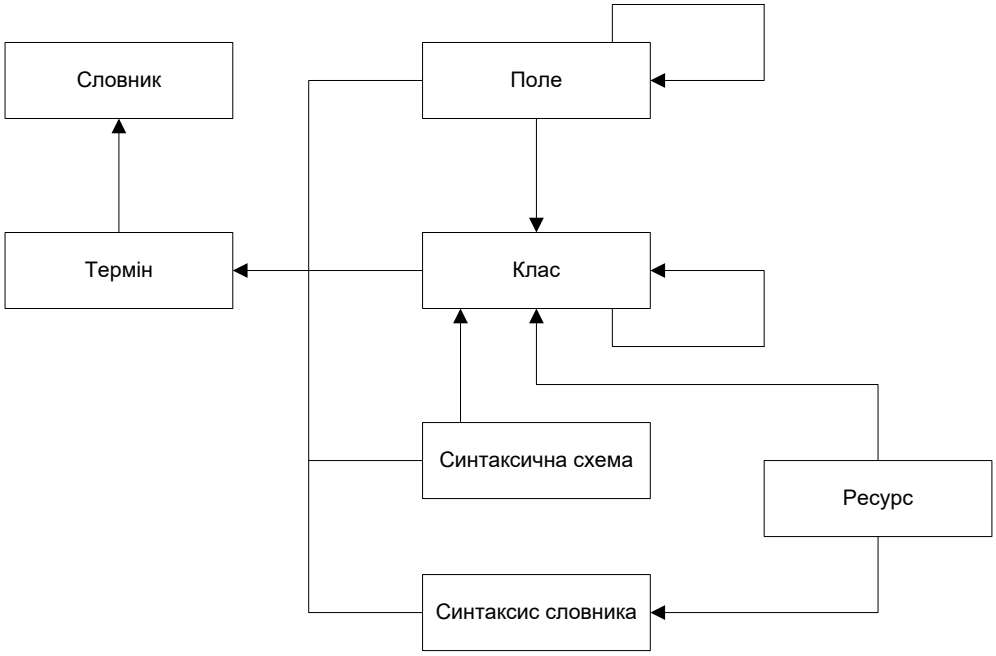
Технічне завдання до виконання прийняв _____ Тягун Д. Т.

Додаток В

Метод Dublin Core та ієрархічних структур для систем метаданих

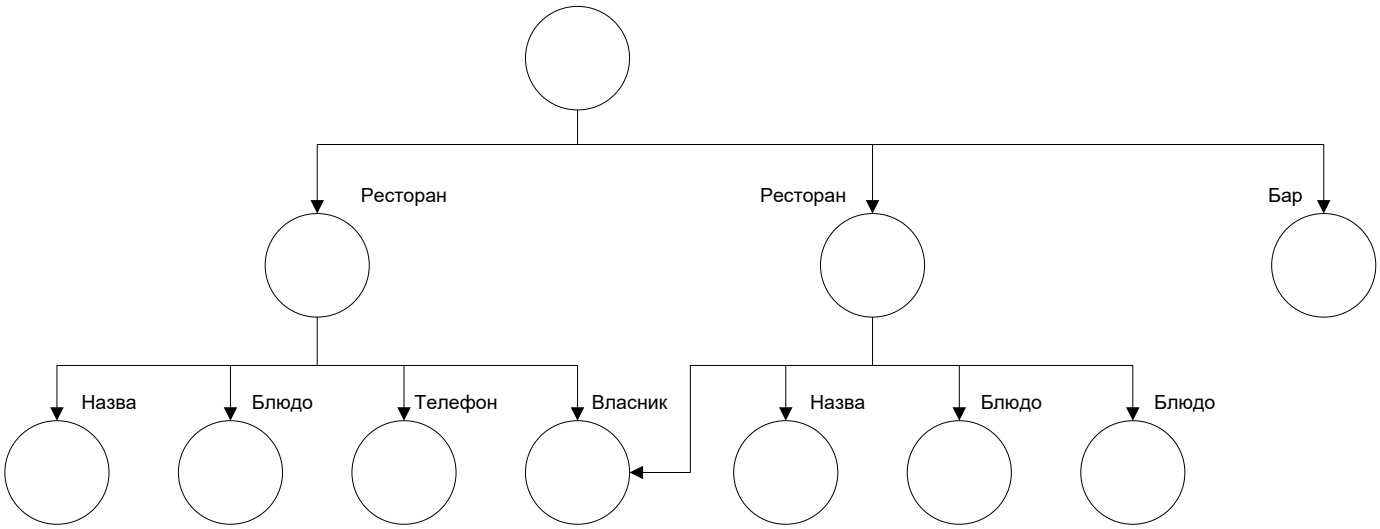


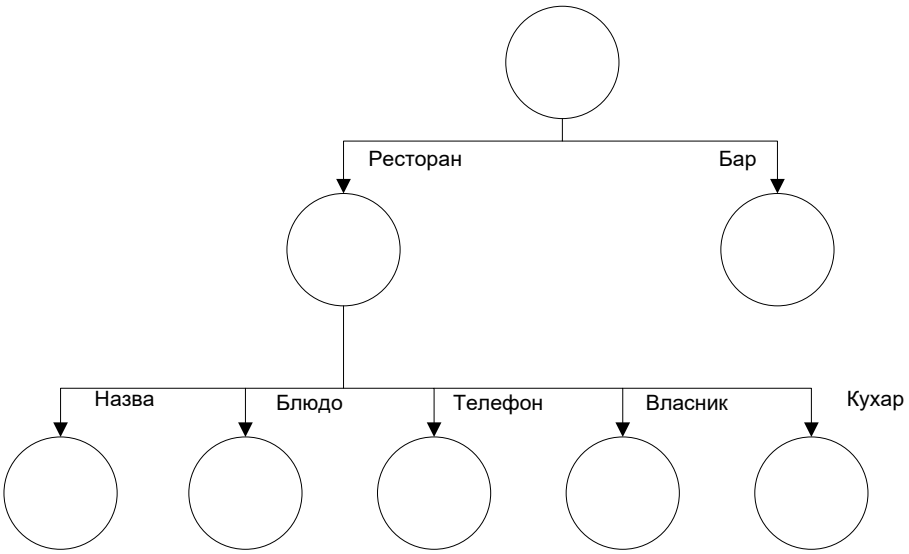




Додаток С

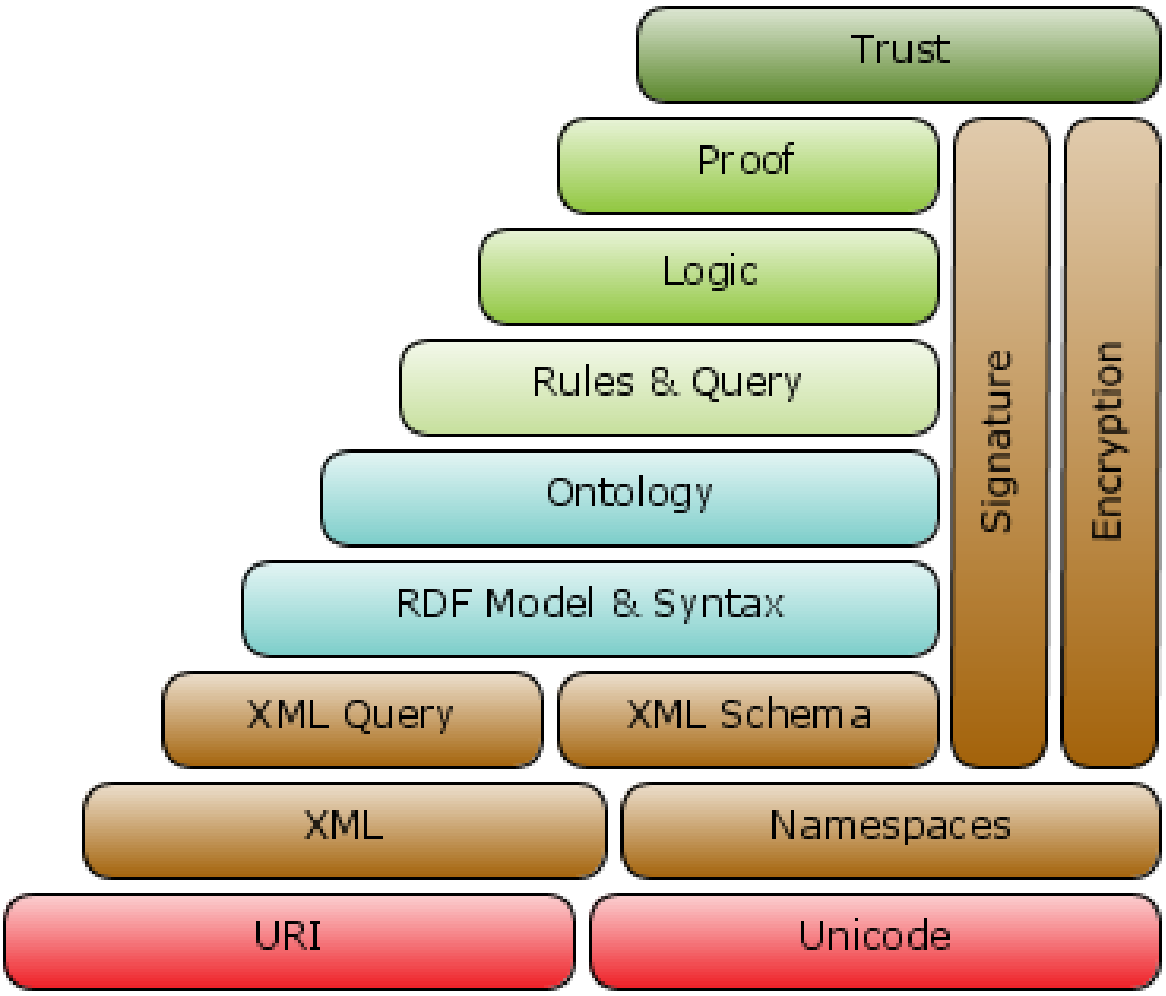
Деревоподібна модель БД із нечіткою та чіткою структурою метаданих





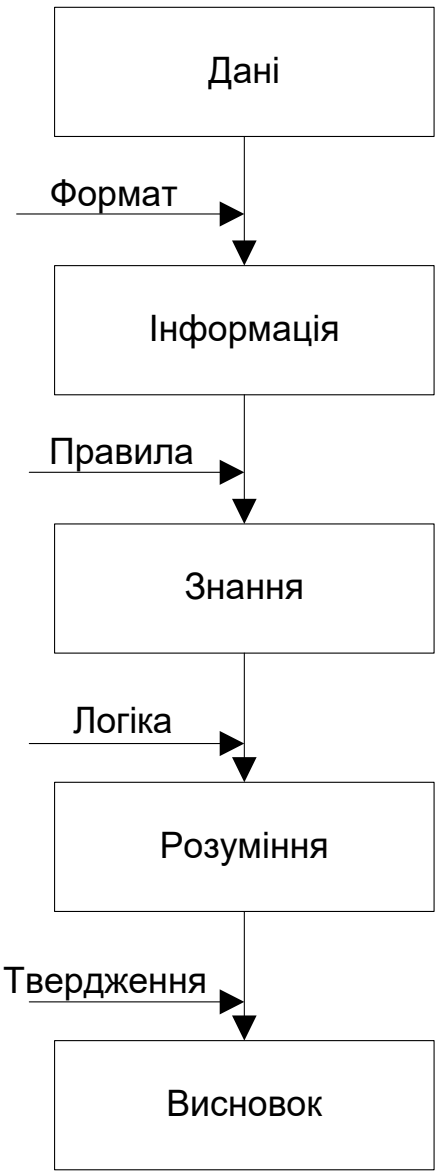
Додаток D

Загальна семантична модель для текст-орієнтованих систем



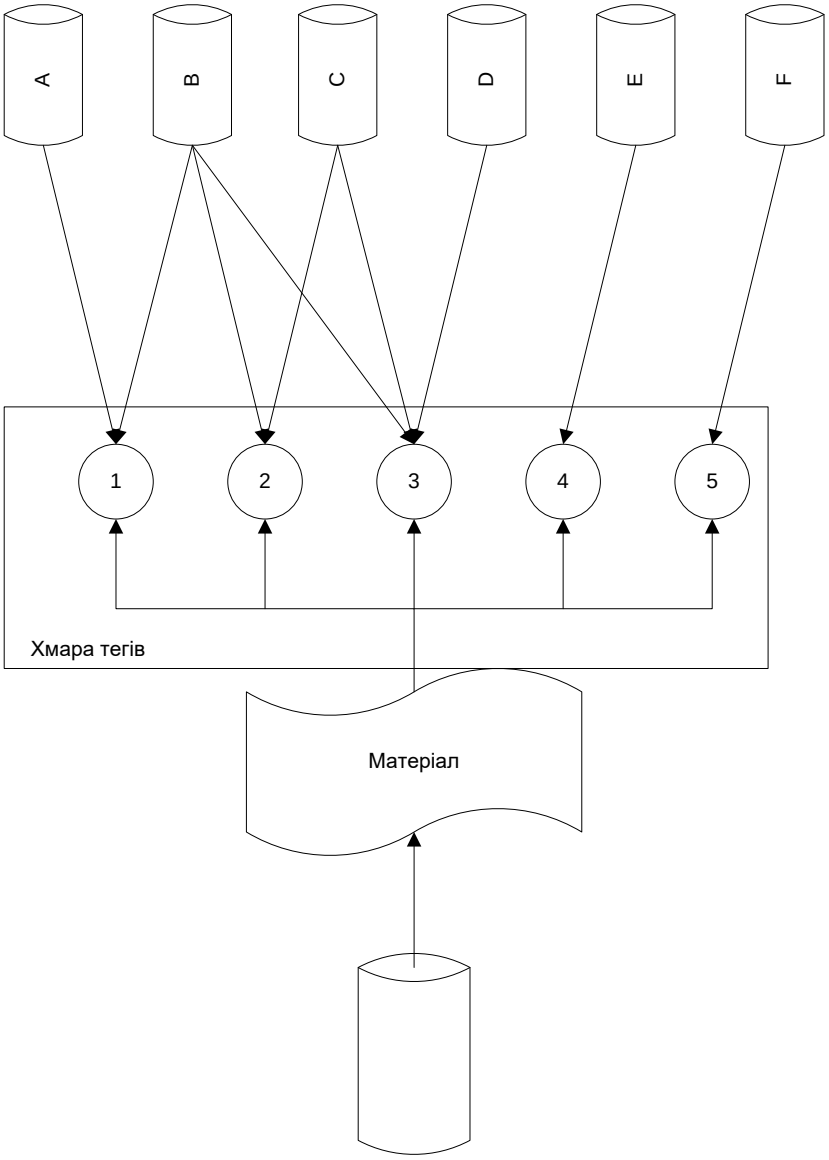
Додаток F

Модель здобуття знань



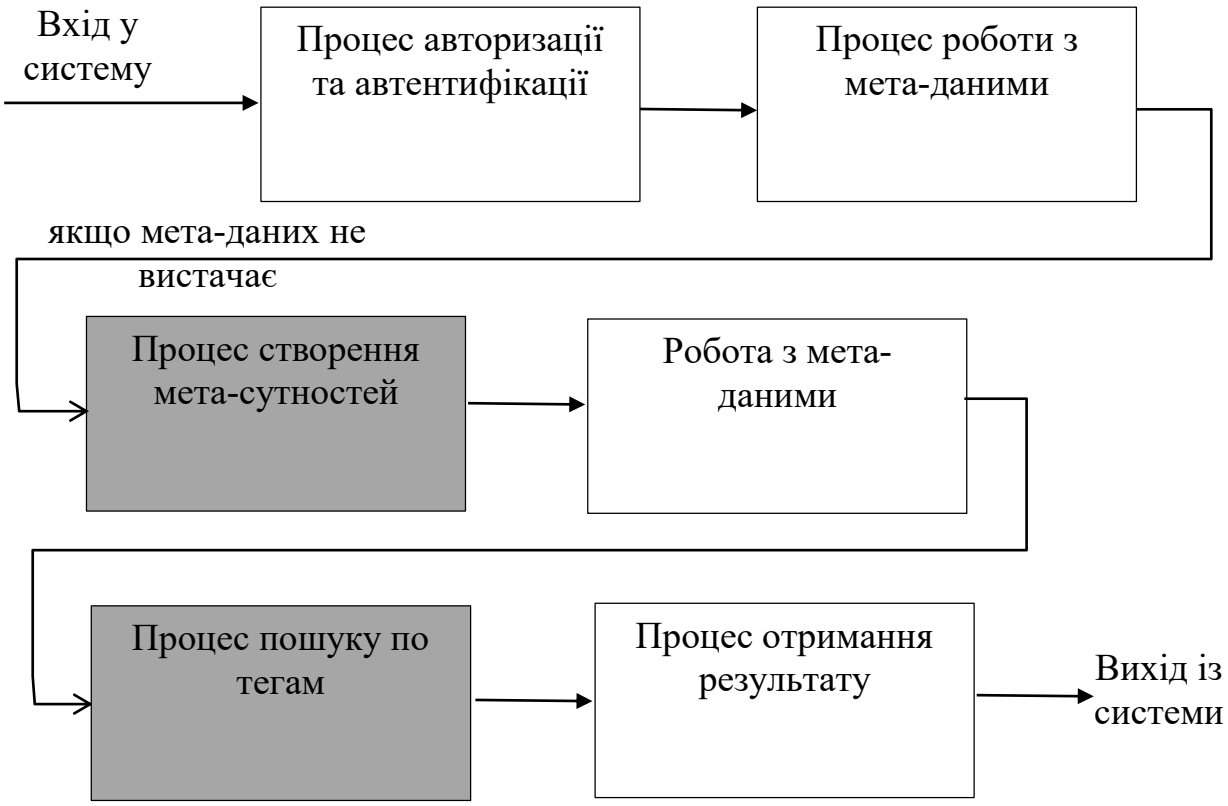
Додаток G

Широке фолксономічне тегування



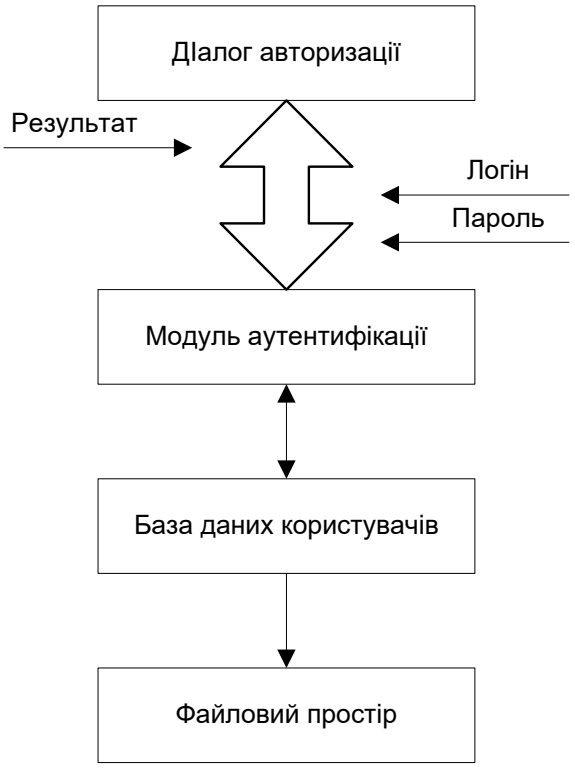
Додаток Н

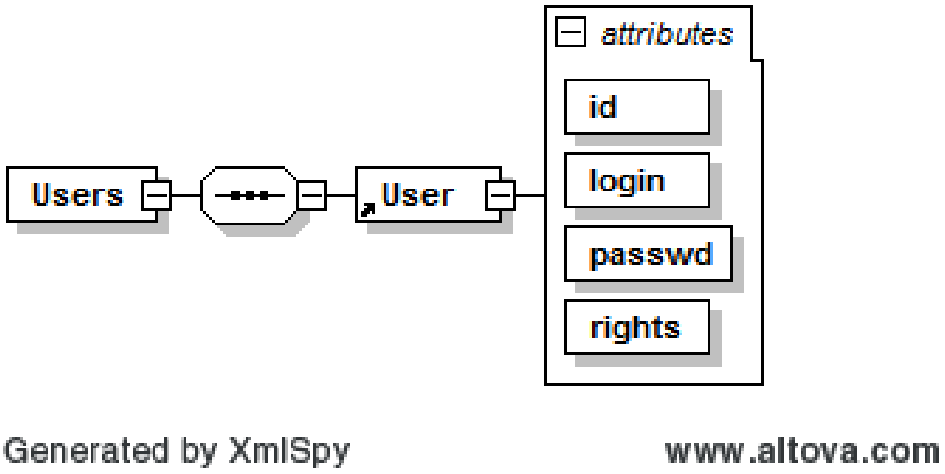
Загальна схема методу застосування метаданих в процесах пошуку



Додаток J

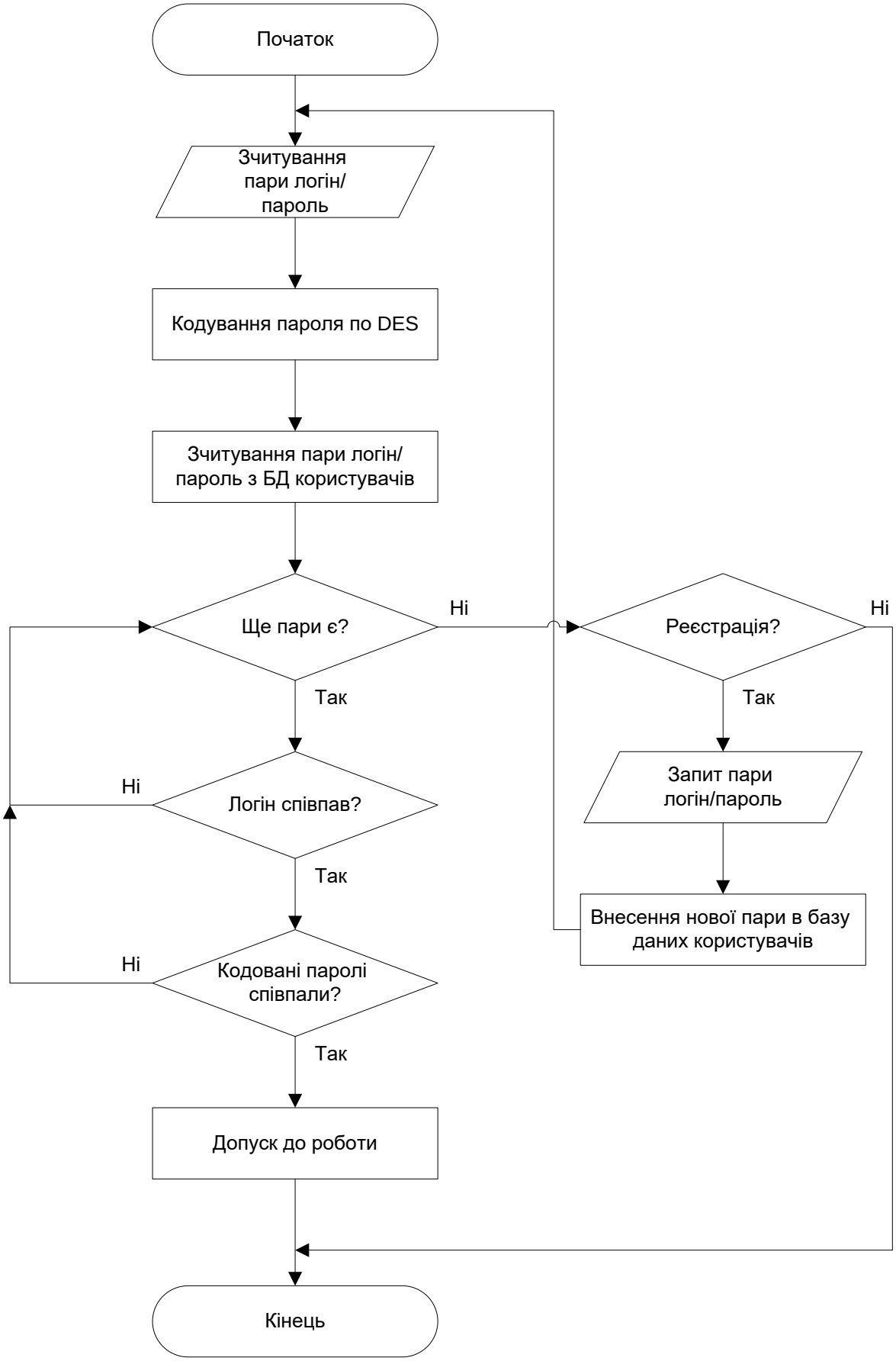
Діаграма роботи процесу та формат даних для авторизації та автентифікації





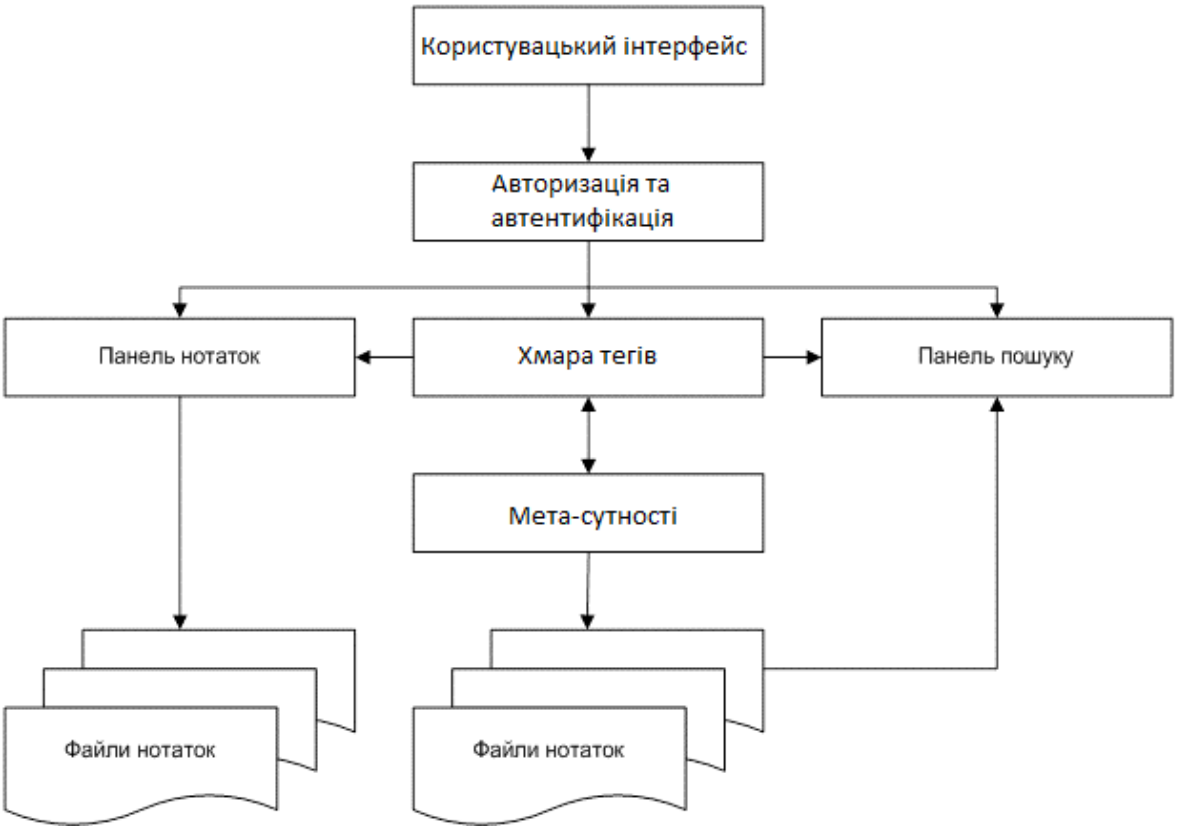
Додаток К

Алгоритм роботи процес авторизації та автентифікації



Додаток L

Діаграма компонентів програмної моделі

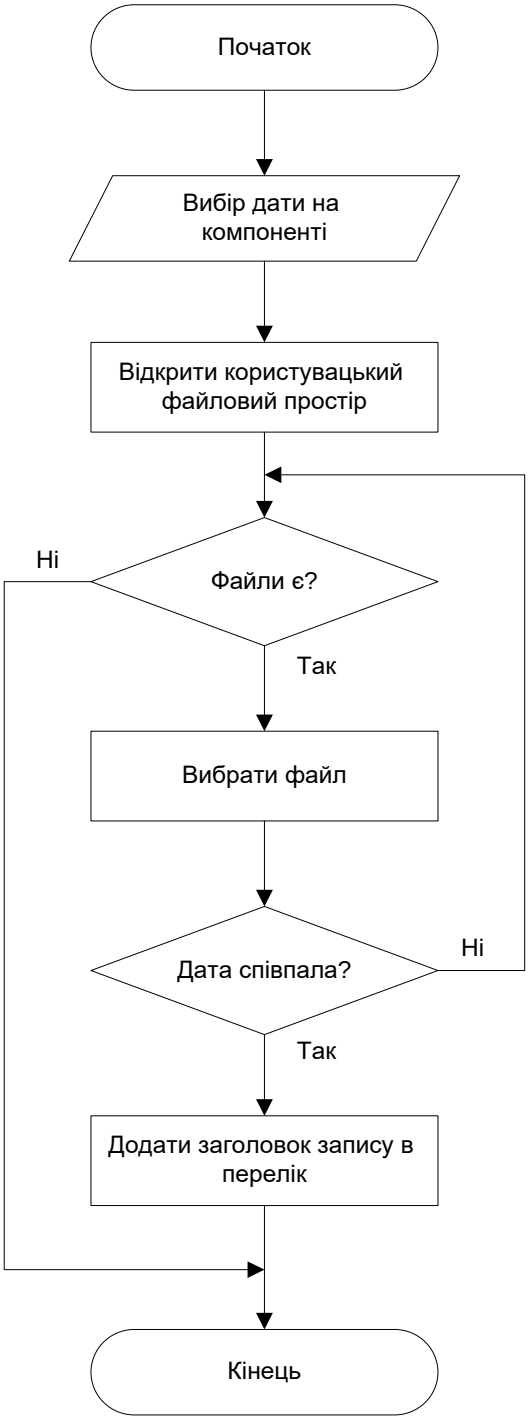


Додаток М

Структура програмної моделі керування електронними нотатками

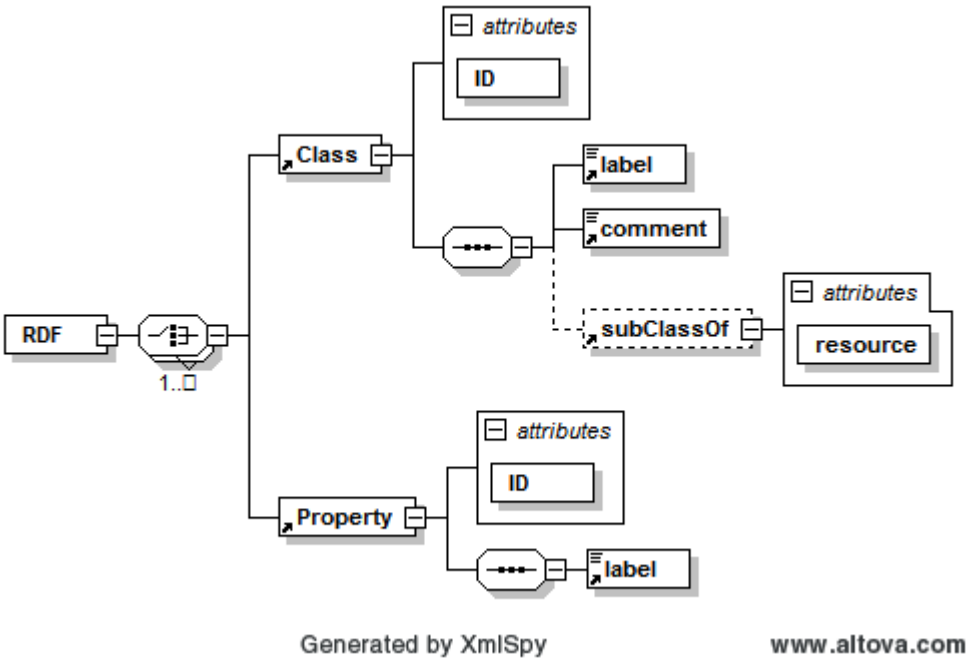
Додаток N

Алгоритм прив’язування списку нотаток до дати



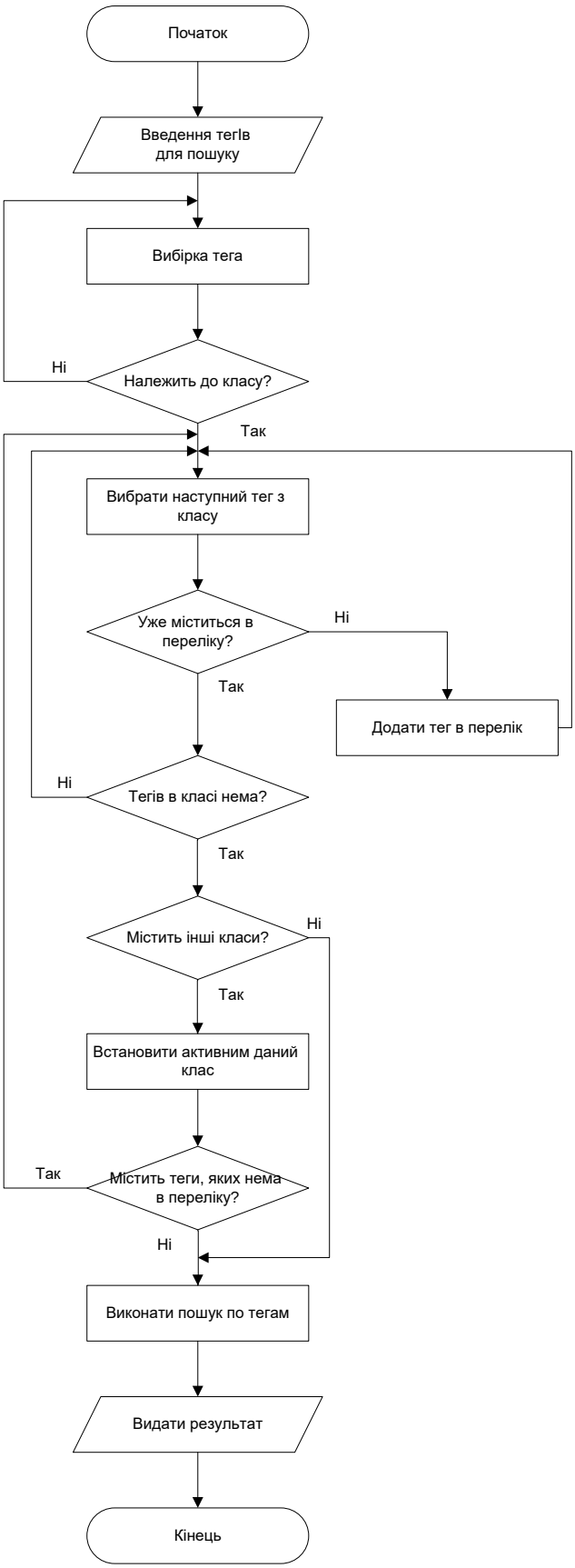
Додаток Р

Схема RDF для створення мета-сутностей



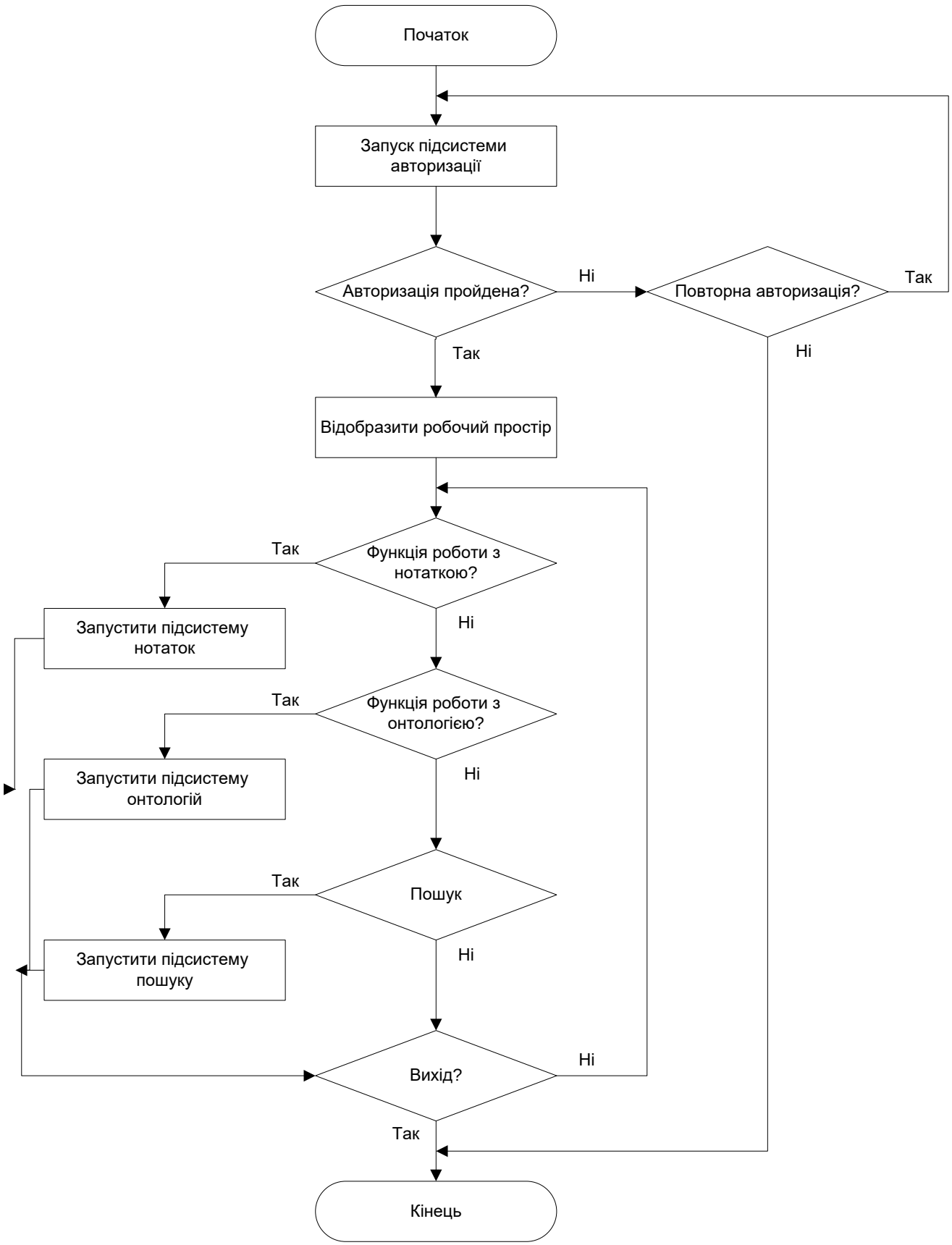
Додаток Q

Алгоритм пошуку по тегам



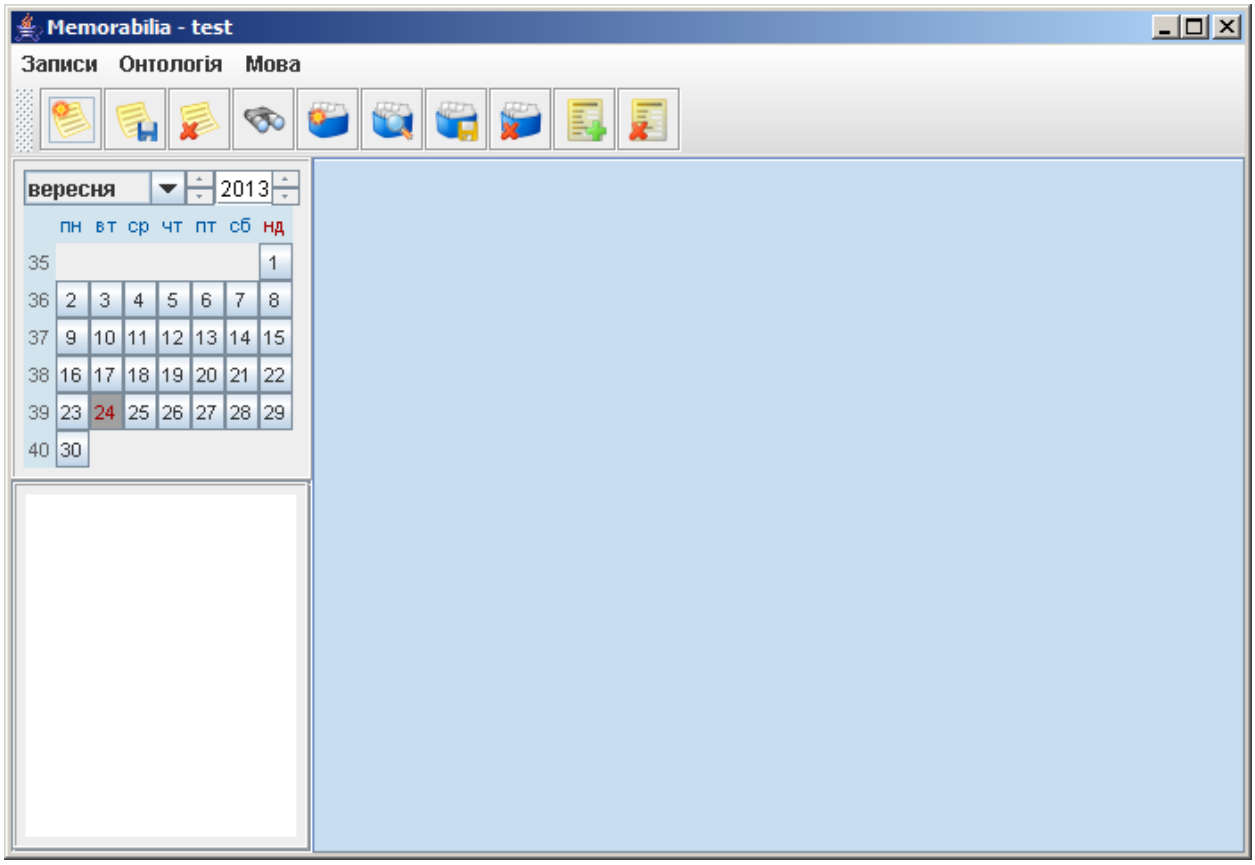
Додаток R

Алгоритм функціонування головного модуля програми



Додаток S

Робочий простір програми



Додаток Т

Лістинг програми

Клас семантичної хмари TagCloud

```
package ua.richesa.memorabilia.tools;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.util.Vector;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import ua.richesa.memorabilia.Memorabilia;
import ua.richesa.tools.Utilities;

public class TagCloud
{
 private static TagCloud cloud = null;

 private Vector tagData = new Vector();

 private TagCloud()
 {
  try
  {
   DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
   DocumentBuilder builder =
factory.newDocumentBuilder();

   File currentDir = new File(".");
   File profileDir = new File(currentDir.getCanonicalPath()
+ File.separator + "profiles" + File.separator +
Memorabilia.getInstance().getLoginName());

   FileInputStream fis = new
FileInputStream(profileDir.getCanonicalPath() + File.separator +
"tagcloud.xml");
   Document curDoc = builder.parse(fis);
   Element documentElement =
curDoc.getDocumentElement();
   NodeList tagNodes =
documentElement.getElementsByTagName("tag");
   for(int i = 0; i < tagNodes.getLength(); i++)
   {
    Element currentTag = (Element)tagNodes.item(i);

tagData.addElement(currentTag.getChildNodes().item(0).getNodeV
alue());
   }
   fis.close();

  } catch (Exception e)
  {
   System.err.println(e.getMessage());
   e.printStackTrace(System.err);
  }
 }

 public void addTag(String newTag)
 {
  if(!tagData.contains(newTag))
   tagData.addElement(newTag);
 }

 public void removeTag(String newTag)
 {
  if(tagData.contains(newTag))
   tagData.remove(newTag);
 }

 public Vector getTagCloud()
 {
  return tagData;
 }

 public void addTags(Vector newTags)
 {
  for(int i = 0; i < newTags.size(); i++)
   addTag(newTags.get(i).toString());
 }

 public void flush()
 {
  try
  {
   DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
   DocumentBuilder builder =
factory.newDocumentBuilder();
   Document newDocument = builder.newDocument();
   Element root = newDocument.createElement("cloud");
   for(int i = 0; i < tagData.size(); i++)

root.appendChild(Utilities.createTextElement(newDocument, "tag",
tagData.get(i).toString()));
   newDocument.appendChild(root);

   File currentDir = new File(".");
   File profileDir = new File(currentDir.getCanonicalPath()
+ File.separator + "profiles" + File.separator +
Memorabilia.getInstance().getLoginName());
   FileOutputStream stream = new
FileOutputStream(profileDir.getCanonicalPath() + File.separator +
"tagcloud.xml");
   OutputFormat output = new
OutputFormat(newDocument);
   output.setEncoding("windows-1251");
   output.setIndenting(true);
   OutputStreamWriter writer = new
OutputStreamWriter(stream, "windows-1251");
   XMLSerializer serial = new XMLSerializer(writer,
output);
   serial.serialize(newDocument.getDocumentElement());
   stream.close();

  } catch (Exception e)
  {
   System.err.println(e.getMessage());
   e.printStackTrace(System.err);
  }

 }

 public static TagCloud getInstance()
 {
  if(cloud == null)
   cloud = new TagCloud();
  return cloud;
 }
}
```

2. Підсистема авторизації та аутентифікації

2.1. Клас авторизації LocalLoginAdapter

```
package ua.richesa.memorabilia.login;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.util.Vector;
```

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import ua.richesa.helpers.CommonDialog;
import ua.richesa.helpers.DESWrapper;
import ua.richesa.helpers.LogStream;
import ua.richesa.helpers.login.LoginAdapter;
import ua.richesa.helpers.login.LoginRecord;
import ua.richesa.tools.Utilities;

public class LocalLoginAdapter implements LoginAdapter
{
  private static final String encryptKey = "h7s2d7h2sJdK";
  private Vector loginList = new Vector();
  private CommonDialog registerDialog = null;
  private CommonDialog loginDialog = null;

  public LocalLoginAdapter(CommonDialog logDialog,
CommonDialog regDialog)
  {
   registerDialog = regDialog;
   loginDialog = logDialog;
   try
   {
    File loginFile = new File((new
File(".")).getCanonicalPath() + File.separator + "profiles" +
File.separator + "userlist.xml");
    System.out.println(loginFile.exists());
    if(loginFile.exists())
    {
     DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
     DocumentBuilder builder =
factory.newDocumentBuilder();
     Document newDocument = builder.parse(new
FileInputStream((new File(".")).getCanonicalPath() + File.separator
+ "profiles" + File.separator + "userlist.xml"));
     Element root = newDocument.getDocumentElement();

     NodeList users = root.getElementsByTagName("User");
     DESWrapper dec = new DESWrapper(encryptKey);
     for(int i = 0; i < users.getLength(); i++)
     {
      Element rec = (Element) users.item(i);
      LoginRecord record = new LoginRecord();
      record.id = rec.getAttribute("id");
      record.login = rec.getAttribute("login");
      record.passwd =
dec.decrypt(rec.getAttribute("passwd"));
      record.rights = dec.decrypt(rec.getAttribute("rights"));
      loginList.addElement(record);
     }

    }

   } catch (Exception e)
   {
    LogStream.getInstance().println(e.getMessage());
    e.printStackTrace(LogStream.getInstance());
   }
  }

  public Vector getLogins()
  {
   Vector data = new Vector();
   for(int i = 0; i < loginList.size(); i++)
    data.addElement(((LoginRecord)loginList.get(i)).login);
   return data;
  }

  public String getLoginID(String login)
  {
   for(int i = 0; i < loginList.size(); i++)
    if(((LoginRecord)loginList.get(i)).login.equals(login))
     return ((LoginRecord)loginList.get(i)).id;
    return "";
  }

  public String getLoginPasswd(String login)
  {
   for(int i = 0; i < loginList.size(); i++)
    if(((LoginRecord)loginList.get(i)).login.equals(login))
     return ((LoginRecord)loginList.get(i)).passwd;
   return "";
  }

  public String getLoginRight(String login)
  {
   for(int i = 0; i < loginList.size(); i++)
    if(((LoginRecord)loginList.get(i)).login.equals(login))
     return ((LoginRecord)loginList.get(i)).rights;
   return "";
  }

  public CommonDialog getRegisterDialog()
  {
   return registerDialog;
  }

  private boolean checkLogin(LoginRecord rec)
  {
   for(int i = 0; i < loginList.size(); i++)
   {
    LoginRecord current = (LoginRecord) loginList.get(i);
    if(rec.login.equals(current.login))
     return true;
   }
   return false;
  }

  public boolean registerLogin(String login, String passwd,
String rights)
  {
   LoginRecord rec = new LoginRecord();
   rec.id = String.valueOf(loginList.size());
   rec.login = login;
   rec.passwd = passwd;
   rec.rights = rights;
   if(!checkLogin(rec))
   {
    loginList.addElement(rec);
    flush();
    return true;
   }
   return false;
  }

  public boolean registerLogin(LoginRecord rec)
  {
   rec.id = String.valueOf(loginList.size());
   if(!checkLogin(rec))
   {
    loginList.addElement(rec);
    flush();
    return true;
   }
   return false;
  }

  public void flush()
  {
   try
   {
    DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
    DocumentBuilder builder =
factory.newDocumentBuilder();
    Document document = builder.newDocument();

    Element root = document.createElement("Users");
    DESWrapper encrypter = new
DESWrapper(encryptKey);
    for(int i = 0; i < loginList.size(); i++)
    {
     Element record = document.createElement("User");
     LoginRecord rec = (LoginRecord) loginList.get(i);
     record.setAttribute("id", rec.id);
     record.setAttribute("login", rec.login);
```

```
                record.setAttribute("passwd",
encrypter.encrypt(rec.passwd));
                record.setAttribute("rights",
encrypter.encrypt(rec.rights));
                root.appendChild(record);
               }

               document.appendChild(root);
               File loginFile = new File((new
File(".")).getCanonicalPath() + File.separator + "profiles" +
File.separator + "userlist.xml");
               FileOutputStream stream = new
FileOutputStream(loginFile.getCanonicalPath());
               OutputFormat output = new OutputFormat(document);
               output.setEncoding("windows-1251");
               output.setIndenting(true);
               output.setIndent(2);
               OutputStreamWriter writer = new
OutputStreamWriter(stream, "windows-1251");
               XMLSerializer serial = new XMLSerializer(writer,
output);
               serial.serialize(document.getDocumentElement());
               stream.close();

              } catch (Exception e)
              {
               LogStream.getInstance().println(e.getMessage());
               e.printStackTrace(LogStream.getInstance());
              }
             }

             public void setRegisterDialog(CommonDialog
registerDialog)
             {
               this.registerDialog = registerDialog;
             }

             public void setLoginList(Vector loginList)
             {
               this.loginList = loginList;
             }

             public Vector getLoginList()
             {
               return loginList;
             }

             public CommonDialog getLoginDialog()
             {
              return loginDialog;
             }
            }
```

Клас діалогового вікна авторизації LoginDialog

```
            package ua.richesa.memorabilia.dialogs;

            import java.awt.BorderLayout;
            import java.awt.GridLayout;

            import java.awt.event.ActionEvent;
            import java.awt.event.ActionListener;
            import java.util.Vector;
            import javax.swing.JButton;
            import javax.swing.JComboBox;
            import javax.swing.JLabel;
            import javax.swing.JPanel;
            import javax.swing.JPasswordField;
            import javax.swing.JTextField;
            import javax.swing.border.EmptyBorder;
            import ua.richesa.helpers.CommonDialog;
            import ua.richesa.helpers.login.RegisterAdapter;
            import ua.richesa.memorabilia.Memorabilia;
            import ua.richesa.tools.Utilities;

            public class LoginDialog
             extends CommonDialog implements ActionListener
            {
             private JComboBox loginText = new JComboBox();
             private JPasswordField passwdText = new
JPasswordField();

             public LoginDialog()
             {
              super();
              setTitle(Memorabilia.getInstance().localize("Login"));
              setSize(300, 150);
              JPanel mainPane = new JPanel(new BorderLayout());
              JPanel innerPane = new JPanel(new BorderLayout());
              JPanel inPane = new JPanel(new GridLayout(2, 1));
              inPane.add(new
JLabel(Memorabilia.getInstance().localize("Login: ")));
              inPane.add(new
JLabel(Memorabilia.getInstance().localize("Password: ")));
              innerPane.add(inPane, BorderLayout.WEST);
              inPane = new JPanel(new GridLayout(2, 1));
              loginText.setEditable(true);
              inPane.add(loginText);
              inPane.add(passwdText);
              passwdText.requestFocusInWindow();
              innerPane.add(inPane, BorderLayout.CENTER);
              JPanel sidePane =  new JPanel(new BorderLayout());
              sidePane.add(new
JLabel(Utilities.getIconResource("login_icon.png")),
BorderLayout.CENTER);
              sidePane.setBorder(new EmptyBorder(0, 10, 0, 10));
              mainPane.add(sidePane, BorderLayout.WEST);
              mainPane.add(innerPane, BorderLayout.CENTER);
              innerPane.setBorder(new EmptyBorder(5, 0, 5, 10));

mainPane.add(Utilities.createButton(Memorabilia.getInstance().loca
lize("Register"),
                                  "registerNew", this,
"register_small.png"), BorderLayout.SOUTH);
              getContentPane().add(mainPane,
BorderLayout.CENTER);
              setAbsoluteCenteredView();
             }

             public void updateLoginNames()
             {
              loginText.removeAllItems();
              Vector newNames =
Memorabilia.getInstance().getLoginAdapter().getLogins();
              for(int i = 0; i < newNames.size(); i++)
               loginText.addItem(newNames.get(i).toString());
              Memorabilia.getInstance().forceRevalidate();
             }

             public void actionPerformed(ActionEvent e)
             {
              if(e.getActionCommand().equals("registerNew"))
              {
               CommonDialog dialog =
Memorabilia.getInstance().getLoginAdapter().getRegisterDialog();

if(!loginText.getSelectedItem().toString().trim().equals(""))

dialog.setInitialData(loginText.getSelectedItem().toString());
               dialog.setVisible(true);
               if(dialog.accepted)
               {
                RegisterAdapter adapter = (RegisterAdapter) dialog;
                adapter.processRegistration();
               }
              return;
              }
              super.actionPerformed(e);
             }

             public Object getData()
             {
              Vector data = new Vector();
              data.addElement(loginText.getSelectedItem().toString());
              data.add(passwdText.getText());
              return data;
             }
            }
```

Клас діалогового вікна реєстрації RegisterDialog

```
            package ua.richesa.memorabilia.dialogs;
```

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import javax.swing.border.CompoundBorder;
import javax.swing.border.EmptyBorder;
import javax.swing.border.EtchedBorder;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import ua.richesa.components.ValueEditable;
import ua.richesa.helpers.CommonDialog;
import ua.richesa.helpers.login.LoginRecord;
import ua.richesa.helpers.login.RegisterAdapter;
import ua.richesa.memorabilia.Memorabilia;
import ua.richesa.tools.Utilities;

public class RegisterDialog extends CommonDialog
  implements ValueEditable, RegisterAdapter
{
  private JTextField loginField = new JTextField();
  private JTextField passField = new JTextField();

  public RegisterDialog()
  {
   super();

setTitle(Memorabilia.getInstance().localize("Registration"));
   JPanel regPane = new JPanel(new BorderLayout());
   JPanel innerPane = new JPanel(new GridLayout(2, 1));
   innerPane.add(new
JLabel(Memorabilia.getInstance().localize("New login: ")));
   innerPane.add(new
JLabel(Memorabilia.getInstance().localize("New password: ")));

   regPane.add(innerPane, BorderLayout.WEST);
   innerPane = new JPanel(new GridLayout(2, 1));
   innerPane.add(loginField);
   innerPane.add(passField);

   passField.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent e)
     {
      if(e.getKeyCode() == KeyEvent.VK_ENTER)
      {
       accepted = true;
       setVisible(false);
      }
     }
    });

   regPane.add(innerPane, BorderLayout.CENTER);
   regPane.setBorder(new CompoundBorder(new
EtchedBorder(EtchedBorder.RAISED), new EmptyBorder(5, 5, 5,
5)));
   getContentPane().add(regPane, BorderLayout.CENTER);
   setSize(300, 125);
   setAbsoluteCenteredView();
  }

  public Object getData()
  {
   LoginRecord rec = new LoginRecord();

   rec.login = loginField.getText();
   rec.passwd = passField.getText();

   return rec;
  }

  public void setInitialData(Object data)
  {
  }

  public boolean isAccepted()
  {
    return accepted;
  }

  public JDialog getDialog()
  {
    return this;
  }

  public void clear()
  {
  }

 public LoginRecord getRecordedLogin()
 {
  LoginRecord rec = new LoginRecord();
  rec.login = loginField.getText();
  rec.passwd = passField.getText();
  return rec;
 }

 public void processRegistration()
 {

Memorabilia.getInstance().getLoginAdapter().registerLogin(getRec
ordedLogin());

((LoginDialog)Memorabilia.getInstance().getLoginAdapter().getLo
ginDialog()).updateLoginNames();
   try
   {
    File currentDir = new File(".");
    File profileDir = new File(currentDir.getCanonicalPath()
+ File.separator + "profiles" + File.separator +
getRecordedLogin().login);
    profileDir.mkdirs();
   } catch (IOException e)
   {
    System.err.println(e.getMessage());
    e.printStackTrace(System.err);
   }
  }
 }
```

Підсистема електронних нотаток

Абстрактний клас панелі редагування (AbstractMemoPane)

```
package ua.richesa.memorabilia.dialogs;

import javax.swing.JPanel;

public abstract class AbstractMemoPane extends JPanel
{

 public static final int MEMOTYPE_RECORD = 0;
 public static final int MEMOTYPE_ONTO   = 1;

 private int memoType = MEMOTYPE_RECORD;

 protected String realName = "";

 public void setMemoType(int memoType)
 {
  this.memoType = memoType;
 }

 public int getMemoType()
 {
  return memoType;
 }

 public String getRealName()
 {
  return realName;
 }
```

```
        }
```

Клас редагування електронної нотатки (RecordPane)

```
package ua.richesa.memorabilia.dialogs;

import java.awt.BorderLayout;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import java.io.OutputStreamWriter;

import java.text.SimpleDateFormat;

import java.util.Date;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import ua.richesa.memorabilia.Memorabilia;
import ua.richesa.tools.Utilities;

public class RecordPane extends AbstractMemoPane
 implements ActionListener
{
 private JTextField topicName = new JTextField();
 private JTextArea topicText = new JTextArea();
 private JTextField tagList = new JTextField();

 public RecordPane()
 {
  setLayout(new BorderLayout());

  JPanel namePane = new JPanel(new BorderLayout());
  namePane.add(topicName, BorderLayout.CENTER);
  namePane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),
                              Memorabilia.getInstance().localize("
Caption ")));
  add(namePane, BorderLayout.NORTH);

  JPanel textPane = new JPanel(new BorderLayout());
  textPane.add(new JScrollPane(topicText),
BorderLayout.CENTER);
  textPane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),
                              Memorabilia.getInstance().localize("
Statement ")));
  add(textPane, BorderLayout.CENTER);

  JPanel tagPane = new JPanel(new BorderLayout());
  tagPane.add(tagList, BorderLayout.CENTER);

tagPane.add(Utilities.createButton(Memorabilia.getInstance().locali
ze("Build"),
                              "buildTags", this), BorderLayout.EAST);
  tagPane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),
                              Memorabilia.getInstance().localize(" Tags
")));
  add(tagPane, BorderLayout.SOUTH);
 }

 private String makeNumber(int index)
 {
  StringBuffer buffer = new StringBuffer(String.valueOf(index));
  for(int i = 0; i < 4 - String.valueOf(index).length(); i++)
   buffer.insert(0, "0");
  return buffer.toString();
 }

 public void open(String fileName)
 {
  try
  {
   DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
   DocumentBuilder builder = factory.newDocumentBuilder();
   FileInputStream fis = new FileInputStream(fileName);
   Document curDoc = builder.parse(fis);
   Element documentElement = curDoc.getDocumentElement();
   topicName.setText(Utilities.getTextElement(documentElement,
"caption", ""));
   topicText.setText(Utilities.getTextElement(documentElement,
"statement", ""));
   tagList.setText(Utilities.getTextElement(documentElement,
"tags", ""));
   fis.close();
   realName = fileName;
  } catch (Exception e)
  {

  }
 }

 public void save(Date date)
 {
  try
  {
   // Stage 1: Pick filename
   SimpleDateFormat dateFormat = new SimpleDateFormat("dd-
MM-yyyy");
   File checkFile = null;
   if(realName.equals(""))
   {
    File currentDir = new File(".");
    File profileDir = new File(currentDir.getCanonicalPath() +
File.separator + "profiles" + File.separator +
Memorabilia.getInstance().getLoginName());

    int num = 0;
    do
    {
     checkFile = new File(profileDir.getCanonicalPath() +
File.separator + dateFormat.format(date) + " " +
makeNumber(num++) + ".xml");
    } while(checkFile.exists());
   }
   else
    checkFile = new File(realName);

   // Stage 2: Save
   DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
   DocumentBuilder builder = factory.newDocumentBuilder();
   Document newDocument = builder.newDocument();
   Element root = newDocument.createElement("record");
   dateFormat = new SimpleDateFormat("dd.MM.yyyy");
   root.setAttribute("date", dateFormat.format(date));
   root.appendChild(Utilities.createTextElement(newDocument,
"caption", topicName.getText()));
   root.appendChild(Utilities.createTextElement(newDocument,
"statement", topicText.getText()));
   root.appendChild(Utilities.createTextElement(newDocument,
"tags", tagList.getText()));
   newDocument.appendChild(root);

   FileOutputStream stream = new
FileOutputStream(checkFile.getCanonicalPath());
   OutputFormat output = new OutputFormat(newDocument);
   output.setEncoding("windows-1251");
   output.setIndenting(true);
```

```
  OutputStreamWriter writer = new OutputStreamWriter(stream,
"windows-1251");
  XMLSerializer serial = new XMLSerializer(writer, output);
  serial.serialize(newDocument.getDocumentElement());
  stream.close();

 } catch (Exception e)
 {
  System.err.println(e.getMessage());
  e.printStackTrace(System.err);
 }
}

public void erase()
{
 File checkFile = new File(realName);
 checkFile.delete();
}

public void actionPerformed(ActionEvent e)
{
 if(e.getActionCommand().equals("buildTags"))
 {
  TagBuildDialog dialog = new
TagBuildDialog(Memorabilia.getInstance());
  dialog.setInitialData(tagList.getText());
  dialog.setVisible(true);
  if(dialog.accepted)
   tagList.setText(dialog.getData().toString());
 }
}

public String getCaption()
{
 return topicName.getText();
}
}
```

Клас діалогового вікна побудови тегів (TagBuildDialog)

```
package ua.richesa.memorabilia.dialogs;

import java.awt.BorderLayout;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.StringTokenizer;
import java.util.Vector;

import javax.swing.JComboBox;
import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.border.CompoundBorder;
import javax.swing.border.EmptyBorder;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

import ua.richesa.components.list.ListPane;
import ua.richesa.helpers.CommonDialog;
import ua.richesa.memorabilia.Memorabilia;
import ua.richesa.memorabilia.tools.TagClassComboBox;
import ua.richesa.memorabilia.tools.TagCloud;
import ua.richesa.tools.Utilities;

public class TagBuildDialog extends CommonDialog
implements ActionListener
{
 private TagClassComboBox pickTag = new
TagClassComboBox(TagCloud.getInstance().getTagCloud(), new
Vector());
  private ListPane tagList = new
ListPane(Memorabilia.getInstance().localize(" Tags "));

  public TagBuildDialog(JFrame parent)
  {
   super(parent);
   setTitle(Memorabilia.getInstance().localize("Tag
Builder"));
   setSize(300, 300);

   JPanel mainPane = new JPanel(new BorderLayout());
   JPanel pickPane = new JPanel(new BorderLayout());
   pickPane.add(pickTag, BorderLayout.CENTER);

pickPane.add(Utilities.createButton(Memorabilia.getInstance().local
ize("Add picked"),
                                    "addPicked", this),
BorderLayout.EAST);
   pickPane.setBorder(new CompoundBorder(new
EmptyBorder(5, 5, 5, 5),
    new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),

Memorabilia.getInstance().localize(" Pick from a cloud "))));
   mainPane.add(pickPane, BorderLayout.NORTH);
   mainPane.add(tagList, BorderLayout.CENTER);

   getContentPane().add(mainPane,
BorderLayout.CENTER);
   setCenteredView();
  }

  public void actionPerformed(ActionEvent e)
  {
   if(e.getActionCommand().equals("addPicked"))
   {
    tagList.addToList(pickTag.getSelectedItem());
    return;
   }
   super.actionPerformed(e);
  }

  public Object getData()
  {
   Vector currentTags = tagList.getListContent();
   StringBuffer buffer = new StringBuffer();
   for(int i = 0; i < currentTags.size(); i++)
   {
    buffer.append(currentTags.get(i).toString());
    if(i != (currentTags.size() - 1))
     buffer.append("; ");
   }
   TagCloud.getInstance().addTags(currentTags);
   TagCloud.getInstance().flush();
   return buffer.toString();
  }

  public void setInitialData(Object data)
  {
   String tagstr = data.toString();
   StringTokenizer tokenizer = new StringTokenizer(tagstr,
";");
   Vector tags = new Vector();
   while(tokenizer.hasMoreTokens())
    tags.add(tokenizer.nextToken().toString().trim());
   tagList.setListContent(tags);
  }
 }
```

Клас редагування онтологій (OntologyPane)

```
package ua.richesa.memorabilia.dialogs;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import java.io.File;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;

import java.util.Vector;
```

```
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.NodeList;

import
ua.richesa.components.combobox.IndentedCellValue;
import ua.richesa.components.list.ListPane;
import ua.richesa.memorabilia.Memorabilia;
import ua.richesa.memorabilia.tools.RDFClass;
import ua.richesa.memorabilia.tools.RDFElement;
import ua.richesa.memorabilia.tools.TagClassComboBox;
import ua.richesa.memorabilia.tools.TagCloud;
import ua.richesa.tools.CustomFileFilter;
import ua.richesa.tools.CustomFileView;
import ua.richesa.tools.Utilities;


public class OntologyPane extends AbstractMemoPane
 implements ActionListener, MouseListener,
DocumentListener,
        ListSelectionListener
{

 private JTextField classLabel = new JTextField();
 private JTextArea classDesc = new JTextArea();
 private ListPane classes = new
ListPane(Memorabilia.getInstance().localize(" Classes "), false);
 private JList tagPane = new JList();
 private TagClassComboBox tagPicker = new
TagClassComboBox(TagCloud.getInstance().getTagCloud(), new
Vector());
 private JLabel saveStatus = new
JLabel(Utilities.getIconResource("icon_notsaved.png"));

 private boolean classSaveStatus = false;
 private boolean turnOff = false;

 public OntologyPane()
 {
  setLayout(new BorderLayout());

  JPanel editPane = new JPanel(new BorderLayout());
  JPanel innerPane = new JPanel(new BorderLayout());
  classLabel.getDocument().addDocumentListener(this);
  innerPane.add(classLabel, BorderLayout.CENTER);
  innerPane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),

Memorabilia.getInstance().localize(" Class Label ")));

  editPane.add(innerPane, BorderLayout.NORTH);
  innerPane = new JPanel(new BorderLayout());
  classDesc.getDocument().addDocumentListener(this);
  innerPane.add(new JScrollPane(classDesc),
BorderLayout.CENTER);
  innerPane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),

Memorabilia.getInstance().localize(" Class Description ")));
  editPane.add(innerPane, BorderLayout.CENTER);
  innerPane = new JPanel(new BorderLayout());

innerPane.add(Utilities.createButton(Memorabilia.getInstance().loc
alize("Save Class Info"), "saveClassInfo", this),
          BorderLayout.CENTER);
  innerPane.add(saveStatus, BorderLayout.EAST);
  innerPane.setBorder(new
EtchedBorder(EtchedBorder.RAISED));
  editPane.setPreferredSize(new Dimension((int)
editPane.getPreferredSize().getWidth(),
    200));
  editPane.add(innerPane, BorderLayout.SOUTH);

  add(editPane, BorderLayout.SOUTH);

  JPanel tagSelector = new JPanel(new BorderLayout());
  innerPane = new JPanel(new BorderLayout());
  innerPane.add(tagPicker, BorderLayout.CENTER);

innerPane.add(Utilities.createButton(Memorabilia.getInstance().loc
alize("Add"),
                "pickTag", this),
BorderLayout.EAST);
  innerPane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),

Memorabilia.getInstance().localize(" Pick from a cloud ")));
  tagSelector.add(innerPane, BorderLayout.NORTH);
  tagPane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),

Memorabilia.getInstance().localize(" Class content ")));
  tagSelector.add(tagPane, BorderLayout.CENTER);

  tagPane.addMouseListener(this);

  classes.setSelectionListener(this);
  classes.removeButtons();
  JSplitPane lists = new
JSplitPane(JSplitPane.HORIZONTAL_SPLIT, classes,
tagSelector);
  lists.setDividerLocation(225);
  add(lists, BorderLayout.CENTER);
 }

 private boolean containsClass(String what)
 {
  Vector data = classes.getListContent();
  for(int i = 0; i < data.size(); i++)
   if( ((RDFClass)data.get(i)).getClassLabel().equals(what)
)
    return true;
  return false;
 }

 private boolean containsCurrent(String what)
 {
  if(classes.getSelectedItem() == null)
   return false;
  RDFClass current = (RDFClass)
classes.getSelectedItem();
  if(current.getClassLabel().equals(what))
   return true;
  return false;
 }

 private Vector getSuperclasses(String className)
 {
  Vector superClasses = new Vector();
  Vector allClasses = classes.getListContent();
  for(int i = 0; i < allClasses.size(); i++)
  {
   RDFClass current = (RDFClass) allClasses.get(i);
   for(int j = 0; j < current.getContentElements().size(); j++)
```

```
            {
             RDFElement element = current.getContentElement(j);
             if((element.type == RDFClass.ELEMENT_CLASS) &&
element.elementName.equals(className))
              superClasses.addElement(current.getClassLabel());
             }
            }
            return superClasses;
           }

           public void openOntology(String fileName)
           {
            try
            {
             DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
             DocumentBuilder builder =
factory.newDocumentBuilder();
             FileInputStream fis = new FileInputStream(fileName);
             Document curDoc = builder.parse(fis);
             Element documentElement =
curDoc.getDocumentElement();

             // Stage 1: Check and load properties on-demand
             Vector oldProps =
TagCloud.getInstance().getTagCloud();
             NodeList nProps =
documentElement.getElementsByTagName("rdf:Property");
             boolean loadProps = false;
             for(int i = 0; i < nProps.getLength(); i++)
             {
              Element curProp = (Element) nProps.item(i);
              String pText = Utilities.getTextElement(curProp,
"rdfs:label", "");
              if(!pText.trim().equals(""))
              {
               if(!oldProps.contains(pText) && !loadProps)
               {
                if(JOptionPane.showConfirmDialog(this,
Memorabilia.getInstance().localize("RDF Ontology contains
unlisted tags. Add them to tag cloud?"),

Memorabilia.getInstance().localize("Warning"),
JOptionPane.INFORMATION_MESSAGE,
JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
                 {
                  loadProps = false;
                 }
               }
              }

              if(loadProps)
               TagCloud.getInstance().addTag(pText);
             }

             // Stage 2: Load classes

             NodeList cProps =
documentElement.getElementsByTagName("rdfs:Class");
             Vector cData = new Vector();
             for(int i = 0; i < cProps.getLength(); i++)
             {
              RDFClass curClass = new RDFClass();
              Element curRec = (Element) cProps.item(i);
              curClass.setClassLabel(Utilities.getTextElement(curRec,
"rdfs:label", ""));
              curClass.setClassDesc(Utilities.getTextElement(curRec,
"rdfs:comment", ""));

              for(int j = 0; j < nProps.getLength(); j++)
              {
               Element curProp = (Element) nProps.item(j);
               String testName =
curProp.getAttribute("rdf:ID").substring(0,
curProp.getAttribute("rdf:ID").indexOf("."));
               if(testName.equals(curClass.getClassLabel()))
               {
                RDFElement elem = new
RDFElement(Utilities.getTextElement(curProp, "rdfs:label", ""),
RDFClass.ELEMENT_TAG);
                curClass.addContentElement(elem);
               }
              }

              for(int j = 0; j < cProps.getLength(); j++)
              {
               if(j == i)
                continue;

               Element current = (Element) cProps.item(j);
               NodeList allsubs =
current.getElementsByTagName("rdfs:subClassOf");
               if(allsubs.getLength() > 0)
               {
                for(int k = 0; k < allsubs.getLength(); k++ )
                {
                 Element cursub = (Element) allsubs.item(k);

if(cursub.getAttribute("rdf:resource").equals(curClass.getClassLabe
l()))
                 {
                  RDFElement elem = new
RDFElement(current.getAttribute("rdf:ID"),
RDFClass.ELEMENT_CLASS);
                  curClass.addContentElement(elem);
                 }
                }
               }
              }

              cData.addElement(curClass);
              tagPicker.addRDFElement(curClass);
              Memorabilia.getInstance().forceRevalidate();
             }
             classes.setListContent(cData);
             setSaved(true);

             realName = (new File(fileName)).getName();
            } catch (Exception e)
            {
             System.err.println(e.getMessage());
             e.printStackTrace(System.err);
            }
           }

           public void saveOntology(boolean force)
           {
            try
            {
             File checkFile = new File(realName);
             if((!force && realName.equals("Noname")) ||
!checkFile.exists())
              force = true;

             if(force)
             {
              JFileChooser chooser = new JFileChooser();
              CustomFileFilter filter = new CustomFileFilter("RDF
Documents (*.rdf)");
              filter.addExtension(".rdf");
              CustomFileView view = new CustomFileView();
              view.addCustomFileView(".rdf", "document_rdf.png");
              chooser.setFileFilter(filter);
              chooser.setFileView(view);
              if(chooser.showSaveDialog(this) ==
JFileChooser.APPROVE_OPTION)
               realName =
chooser.getSelectedFile().getCanonicalPath();

              if(!realName.endsWith(".rdf"))
               realName = realName + ".rdf";
             }

             DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
             DocumentBuilder builder =
factory.newDocumentBuilder();
             Document newDocument = builder.newDocument();
             Element root =
newDocument.createElement("rdf:RDF");

             Vector allClasses = classes.getListContent();
```

```
            for(int i = 0; i < allClasses.size(); i++)
            {
             RDFClass rec = (RDFClass) allClasses.get(i);
             Element rdfClass =
newDocument.createElement("rdfs:Class");
             rdfClass.setAttribute("rdf:ID", rec.getClassLabel());

rdfClass.appendChild(Utilities.createTextElement(newDocument,
"rdfs:label", rec.getClassLabel()));

rdfClass.appendChild(Utilities.createTextElement(newDocument,
"rdfs:comment", rec.getClassDesc()));
             Vector sClasses = getSuperclasses(rec.getClassLabel());
             if(sClasses.size() > 0)
             {
              for(int j = 0; j < sClasses.size(); j++)
              {
               Element subClass =
newDocument.createElement("rdfs:subClassOf");
                subClass.setAttribute("rdf:resource",
sClasses.get(j).toString());
               rdfClass.appendChild(subClass);
              }
             }
             root.appendChild(rdfClass);

             for(int j = 0; j < rec.getContentElements().size(); j++)
             {
              RDFElement curElement = rec.getContentElement(j);
              if(curElement.type != RDFClass.ELEMENT_CLASS)
              {
               Element property =
newDocument.createElement("rdf:Property");
                property.setAttribute("rdf:ID", rec.getClassLabel() +
"." + curElement.elementName);

property.appendChild(Utilities.createTextElement(newDocument,
"rdfs:label", curElement.elementName));
               root.appendChild(property);
              }
             }
            }

            newDocument.appendChild(root);
            FileOutputStream stream = new
FileOutputStream(realName);
            OutputFormat output = new
OutputFormat(newDocument);
            output.setEncoding("windows-1251");
            output.setIndenting(true);
            OutputStreamWriter writer = new
OutputStreamWriter(stream, "windows-1251");
            XMLSerializer serial = new XMLSerializer(writer,
output);
            serial.serialize(newDocument.getDocumentElement());
            stream.close();
           } catch (Exception e)
           {
            System.err.println(e.getMessage());
            e.printStackTrace(System.err);
           }
          }

          public void actionPerformed(ActionEvent e)
          {
           if(e.getActionCommand().equals("saveClassInfo"))
           {
            turnOff = true;
            RDFClass nclass = new RDFClass();
            nclass.setClassLabel(classLabel.getText());
            nclass.setClassDesc(classDesc.getText());
            for(int i = 0; i < tagPane.getModel().getSize(); i++)
             nclass.addContentElement((RDFElement)
tagPane.getModel().getElementAt(i));
            if(containsCurrent(classLabel.getText()))
             classes.replaceToList(nclass);
            else
            {
             classes.addToList(nclass);
             tagPicker.addRDFElement(nclass);
             Memorabilia.getInstance().forceRevalidate();
            }
            Memorabilia.getInstance().forceRevalidate();
            setSaved(true);
            turnOff = false;
           }

           if(e.getActionCommand().equals("pickTag"))
           {
            if(tagPicker.getSelectedItem() != null)
            {
             boolean exists = false;
             for(int i = 0; i < tagPane.getModel().getSize(); i++)
             {

if(tagPane.getModel().getElementAt(i).toString().equals(tagPicker.g
etSelectedItem().toString()))
               exists = true;
             }
             if(!exists)
             {
              ListPane.addToList(tagPane,
((IndentedCellValue)tagPicker.getSelectedItem()).getValue());
              setSaved(false);
             }
            }
           }

          public boolean checkSaved()
          {
           if(!classSaveStatus)
           {
            if(JOptionPane.showConfirmDialog(this,
Memorabilia.getInstance().localize("Current class doesn't saved.
Discard changes?"),
                                             "Warning",
JOptionPane.WARNING_MESSAGE,
                                             JOptionPane.YES_NO_OPTION)
== JOptionPane.YES_OPTION)
             return true;
            return false;
           }
           return true;
          }

          public void createNewClass()
          {
           if(checkSaved())
           {
            classLabel.setText("");
            classDesc.setText("");
            tagPane.setListData(new Vector());
            Memorabilia.getInstance().forceRevalidate();
           }
          }

          public void killCurrentClass()
          {
           ListPane.removeToList(tagPane);
          }

          public void mouseClicked(MouseEvent e)
          {
           if((e.getClickCount() >= 2) && (e.getSource() instanceof
JList))
           {
            ListPane.removeToList(tagPane);
            Memorabilia.getInstance().forceRevalidate();
           }
          }

          public void mousePressed(MouseEvent e)
          {
          }

          public void mouseReleased(MouseEvent e)
          {
          }

          public void mouseEntered(MouseEvent e)
          {
```

```
            }

            public void mouseExited(MouseEvent e)
            {
            }

            public void insertUpdate(DocumentEvent e)
            {
             classSaveStatus = false;

saveStatus.setIcon(Utilities.getIconResource("icon_notsaved.png"))
;
             Memorabilia.getInstance().forceRevalidate();
            }

            public void removeUpdate(DocumentEvent e)
            {
            }

            public void changedUpdate(DocumentEvent e)
            {
            }

            private void setSaved(boolean saved)
            {
             classSaveStatus = saved;
             if(saved)

saveStatus.setIcon(Utilities.getIconResource("icon_saved.png"));
             else

saveStatus.setIcon(Utilities.getIconResource("icon_notsaved.png"))
;
            }


            public void valueChanged(ListSelectionEvent e)
            {
             if(turnOff)
              return;
             if(!classSaveStatus)
             {
              if(JOptionPane.showConfirmDialog(this,
Memorabilia.getInstance().localize("Current class doesn't saved.
Save changes?"),
                                               "Warning",
JOptionPane.WARNING_MESSAGE,
                                               JOptionPane.YES_NO_OPTION)
== JOptionPane.YES_OPTION)
              {
               RDFClass nclass = new RDFClass();
               nclass.setClassLabel(classLabel.getText());
               nclass.setClassDesc(classDesc.getText());
               for(int i = 0; i < tagPane.getModel().getSize(); i++)
                nclass.addContentElement((RDFElement)
tagPane.getModel().getElementAt(i));
               turnOff = true;
               Object oldElement = classes.getSelectedItem();
               classes.replaceItem(nclass, true);
               classes.setSelectedItem(oldElement);
               turnOff = false;
               setSaved(true);
              }
             }

             RDFClass current = (RDFClass)
classes.getSelectedItem();
             classLabel.setText(current.getClassLabel());
             classDesc.setText(current.getClassDesc());
             tagPane.setListData(current.getContentElements());
             Memorabilia.getInstance().forceRevalidate();
             setSaved(true);
            }
           }
```

Клас діалогового вікна пошуку (SearchDialog)

```
           package ua.richesa.memorabilia.dialogs;

           import com.toedter.calendar.JDateChooser;

           import java.awt.BorderLayout;

           import java.awt.GridLayout;

           import java.awt.event.ActionEvent;

           import java.awt.event.MouseEvent;
           import java.awt.event.MouseListener;

           import java.io.File;

           import java.io.FileInputStream;

           import java.text.SimpleDateFormat;

           import java.util.Date;
           import java.util.Locale;
           import java.util.StringTokenizer;
           import java.util.Vector;

           import javax.swing.JButton;
           import javax.swing.JComboBox;

           import javax.swing.JFrame;
           import javax.swing.JLabel;
           import javax.swing.JList;
           import javax.swing.JOptionPane;
           import javax.swing.JPanel;
           import javax.swing.JScrollPane;
           import javax.swing.JTable;
           import javax.swing.JTextField;

           import javax.swing.border.CompoundBorder;
           import javax.swing.border.EmptyBorder;
           import javax.swing.border.EtchedBorder;
           import javax.swing.border.TitledBorder;

           import javax.swing.table.DefaultTableModel;

           import javax.xml.parsers.DocumentBuilder;
           import javax.xml.parsers.DocumentBuilderFactory;

           import org.w3c.dom.Document;
           import org.w3c.dom.Element;

           import org.w3c.dom.NodeList;

           import ua.richesa.components.SelectFilePane;
           import ua.richesa.components.table.ReadOnlyTable;
           import ua.richesa.helpers.CommonDialog;
           import ua.richesa.memorabilia.Memorabilia;
           import ua.richesa.memorabilia.tools.RDFClass;
           import ua.richesa.memorabilia.tools.RDFElement;
           import ua.richesa.memorabilia.tools.TagCloud;
           import ua.richesa.tools.CustomFileFilter;
           import ua.richesa.tools.CustomFileNameFilter;
           import ua.richesa.tools.CustomFileView;
           import ua.richesa.tools.Utilities;

           public class SearchDialog extends CommonDialog
            implements MouseListener
           {
            private SelectFilePane ontoPane = new
SelectFilePane(Memorabilia.getInstance().localize(" RDF Ontology
"));
            private JComboBox searchType = null;
            private JTextField searchText = new JTextField();
            //private JTextField deepScan = new JTextField();
            private JDateChooser beginPeriod = new JDateChooser();
            private JDateChooser endPeriod = new JDateChooser();
            private ReadOnlyTable resultsPane = new
ReadOnlyTable();

            Vector foundFiles = new Vector();

            public SearchDialog(JFrame parent)
            {
             super(parent);
             setTitle(Memorabilia.getInstance().localize("Search"));
             setSize(400, 400);
             setResizable(false);
```

```
        JPanel mainPane = new JPanel(new BorderLayout());

        Vector choices = new Vector();

choices.addElement(Memorabilia.getInstance().localize("Titles"));

choices.addElement(Memorabilia.getInstance().localize("Text"));

choices.addElement(Memorabilia.getInstance().localize("Tags"));
        searchType = new JComboBox(choices);

        JPanel searchPane = new JPanel(new BorderLayout());

        JPanel topPane = new JPanel(new BorderLayout());
        JPanel innerPane = new JPanel(new GridLayout(2, 1));
        innerPane.add(new
JLabel(Memorabilia.getInstance().localize("Search in: ")));
        innerPane.add(new
JLabel(Memorabilia.getInstance().localize("Search what: ")));
        innerPane.setBorder(new EmptyBorder(0, 5, 0, 0));
        topPane.add(innerPane, BorderLayout.WEST);

        innerPane = new JPanel(new GridLayout(2, 1));
        innerPane.add(searchType);
        innerPane.add(searchText);
        topPane.add(innerPane, BorderLayout.CENTER);
        topPane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),

Memorabilia.getInstance().localize(" Search ")));
        searchPane.add(topPane, BorderLayout.NORTH);

        CustomFileFilter filter = new CustomFileFilter("RDF
Documents (*.rdf)");
        filter.addExtension(".rdf");
        CustomFileView view = new CustomFileView();
        view.addCustomFileView(".rdf", "document_rdf.png");
        ontoPane.setCurrentFilter(filter);
        ontoPane.setCurrentView(view);
        searchPane.add(ontoPane, BorderLayout.CENTER);

        topPane = new JPanel(new BorderLayout());
        innerPane = new JPanel(new GridLayout(2, 1));
        innerPane.add(new
JLabel(Memorabilia.getInstance().localize("Begin date: ")));
        innerPane.add(new
JLabel(Memorabilia.getInstance().localize("End date: ")));
        innerPane.setBorder(new EmptyBorder(0, 5, 0, 0));
        topPane.add(innerPane, BorderLayout.WEST);
        innerPane = new JPanel(new GridLayout(2, 1));

        beginPeriod.setLocale(Locale.ENGLISH);
        endPeriod.setLocale(Locale.ENGLISH);

if(Memorabilia.getInstance().getLocaler().getCurrentLocale().equal
s("Русский"))
        {
         beginPeriod.setLocale(new Locale("ru", "RU"));
         endPeriod.setLocale(new Locale("ru", "RU"));
        }

if(Memorabilia.getInstance().getLocaler().getCurrentLocale().equal
s("Українська"))
        {
         beginPeriod.setLocale(new Locale("uk", "UA"));
         endPeriod.setLocale(new Locale("uk", "UA"));
        }

        innerPane.add(beginPeriod);
        innerPane.add(endPeriod);
        topPane.add(innerPane, BorderLayout.CENTER);
        topPane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),

Memorabilia.getInstance().localize(" Date(s) ")));
        searchPane.add(topPane, BorderLayout.SOUTH);

        mainPane.add(searchPane, BorderLayout.NORTH);

        topPane = new JPanel(new BorderLayout());

        JButton sButton = Utilities.createButton("", "search", this,
"start_search.png");
        sButton.setBorder(new CompoundBorder(new
EmptyBorder(0, 5, 0, 5), sButton.getBorder()));
        topPane.add(sButton, BorderLayout.NORTH);

        innerPane = new JPanel(new BorderLayout());

        String heads[] =
{Memorabilia.getInstance().localize("Date"),

Memorabilia.getInstance().localize("Caption")};

        resultsPane.setModel(new DefaultTableModel(heads, 2));

resultsPane.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

resultsPane.getColumnModel().getColumn(0).setPreferredWidth(10
0);

resultsPane.getColumnModel().getColumn(1).setPreferredWidth(27
9);
        clearResults();
        resultsPane.addMouseListener(this);

        innerPane.add(new JScrollPane(resultsPane));
        innerPane.setBorder(new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED),

Memorabilia.getInstance().localize(" Search Results ")));
        topPane.add(innerPane, BorderLayout.CENTER);

        mainPane.add(topPane, BorderLayout.CENTER);

        getContentPane().add(mainPane,
BorderLayout.CENTER);
        setCenteredView();
        }

       private void clearResults()
       {
        DefaultTableModel model = (DefaultTableModel)
resultsPane.getModel();
        int howMany = model.getRowCount();
        for(int i = 0; i < howMany; i++)
         model.removeRow(0);
        Memorabilia.getInstance().forceRevalidate();
       }

      private void addResult(String date, String caption)
       {
        DefaultTableModel model = (DefaultTableModel)
resultsPane.getModel();
        Vector data = new Vector();
        data.addElement(date);
        data.addElement(caption);
        model.addRow(data);
       }

      private Vector recurseRDFClass(Vector all, RDFClass
current)
       {
        Vector listTags = new Vector();
        Vector allTags = current.getContentElements();
        for(int i = 0; i < allTags.size(); i++)
        {
         RDFElement elem = (RDFElement) allTags.get(i);
         if(elem.type == RDFClass.ELEMENT_TAG)
          listTags.addElement(elem.elementName);

         if(elem.type == RDFClass.ELEMENT_CLASS)
         {
          if(!allTags.contains(elem.elementName))
           listTags.addElement(elem.elementName);
          for(int j = 0; j < all.size(); j++)
          {
           RDFClass curClass = (RDFClass) all.get(j);
           if(curClass.getClassLabel().equals(elem.elementName))
           {
            Vector addition = recurseRDFClass(all, curClass);
            for(int k = 0; k < addition.size(); k++)
```

```
                {
                 if(!listTags.contains(addition.get(k).toString()))
                  listTags.addElement(addition.get(k).toString());
                }
               }
              }
             }
            return listTags;
           }

          private Vector getRDFClasses(String where, String
whatToSearch)
           {
           Vector tagCloud = new Vector();
           try
           {
            Vector rdfs = new Vector();

            DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder builder =
factory.newDocumentBuilder();
            FileInputStream fis = new FileInputStream(where);
            Document curDoc = builder.parse(fis);
            Element documentElement =
curDoc.getDocumentElement();

            // Stage 1: Check and load properties on-demand
            Vector oldProps =
TagCloud.getInstance().getTagCloud();
            NodeList nProps =
documentElement.getElementsByTagName("rdf:Property");
            boolean loadProps = false;
            for(int i = 0; i < nProps.getLength(); i++)
            {
             Element curProp = (Element) nProps.item(i);
             String pText = Utilities.getTextElement(curProp,
"rdfs:label", "");
             if(!pText.trim().equals(""))
             {
              if(!oldProps.contains(pText) && !loadProps)
              {
               if(JOptionPane.showConfirmDialog(this,
Memorabilia.getInstance().localize("RDF Ontology contains
unlisted tags. Add them to tag cloud?"),

Memorabilia.getInstance().localize("Warning"),
JOptionPane.INFORMATION_MESSAGE,
JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
               {
                loadProps = false;
               }
              }
             }

             if(loadProps)
              TagCloud.getInstance().addTag(pText);
            }

            // Stage 2: Load classes

            NodeList cProps =
documentElement.getElementsByTagName("rdfs:Class");
            for(int i = 0; i < cProps.getLength(); i++)
            {
             RDFClass curClass = new RDFClass();
             Element curRec = (Element) cProps.item(i);
             curClass.setClassLabel(Utilities.getTextElement(curRec,
"rdfs:label", ""));
             curClass.setClassDesc(Utilities.getTextElement(curRec,
"rdfs:comment", ""));

             for(int j = 0; j < nProps.getLength(); j++)
             {
              Element curProp = (Element) nProps.item(j);
              String testName =
curProp.getAttribute("rdf:ID").substring(0,
curProp.getAttribute("rdf:ID").indexOf("."));
              if(testName.equals(curClass.getClassLabel()))
              {
               RDFElement elem = new
RDFElement(Utilities.getTextElement(curProp, "rdfs:label", ""),
RDFClass.ELEMENT_TAG);
               curClass.addContentElement(elem);
              }
             }

             for(int j = 0; j < cProps.getLength(); j++)
             {
              if(j == i)
               continue;

              Element current = (Element) cProps.item(j);
              NodeList allsubs =
current.getElementsByTagName("rdfs:subClassOf");
              if(allsubs.getLength() > 0)
              {
               for(int k = 0; k < allsubs.getLength(); k++ )
               {
                Element cursub = (Element) allsubs.item(k);

if(cursub.getAttribute("rdf:resource").equals(curClass.getClassLabe
l()))
                {
                 RDFElement elem = new
RDFElement(current.getAttribute("rdf:ID"),
RDFClass.ELEMENT_CLASS);
                 curClass.addContentElement(elem);
                }
               }
              }
             }

             rdfs.addElement(curClass);
            }

            // Stage 3: Find out all tags in ontology
            for(int i = 0; i < rdfs.size(); i++)
            {
             RDFClass curClass = (RDFClass) rdfs.get(i);
             Vector elems = curClass.getContentElements();
             for(int j = 0; j < elems.size(); j++)
             {
              RDFElement curElem = (RDFElement) elems.get(j);
              if(curElem.elementName.equals(whatToSearch))
               tagCloud = recurseRDFClass(rdfs, curClass);
             }
            }

           } catch (Exception e)
           {
            System.err.println(e.getMessage());
            e.printStackTrace(System.err);
           }
           return tagCloud;
          }

         private void performSearch()
         {
          try
          {
           // Get File set to process
           File currentDir = new File(".");
           File profileDir = new File(currentDir.getCanonicalPath()
+ File.separator + "profiles" + File.separator +
Memorabilia.getInstance().getLoginName());

           CustomFileNameFilter xmlFilter = new
CustomFileNameFilter("XML Documents (*.xml)");
           xmlFilter.addExtension(".xml");
           File[] basicList = profileDir.listFiles(xmlFilter);

           String whatSeek = searchText.getText();
           foundFiles.removeAllElements();
           clearResults();
           Vector ontoTags = new Vector();
           if(!ontoPane.getPathName().trim().equals(""))
            ontoTags = getRDFClasses(ontoPane.getPathName(),
whatSeek);

           for(int i = 0; i < basicList.length; i++)
```

```
            {
             File current = basicList[i];

             DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
             DocumentBuilder builder =
factory.newDocumentBuilder();
             FileInputStream fis = new FileInputStream(current);
             Document curDoc = builder.parse(fis);
             Element documentElement =
curDoc.getDocumentElement();
             if(!documentElement.getTagName().equals("record"))
              continue;
             SimpleDateFormat dateFormat = new
SimpleDateFormat("dd.MM.yyyy");

             // Exclude other dates, if they exist
             Date curDate =
dateFormat.parse(documentElement.getAttribute("date"));
             if(beginPeriod.getDate() != null)
             {
              if(beginPeriod.getDate().compareTo(curDate) > 0)
               continue;
             }

             if(endPeriod.getDate() != null)
             {
              if(endPeriod.getDate().compareTo(curDate) < 0)
               continue;
             }


if(searchType.getSelectedItem().toString().equals(Memorabilia.getI
nstance().localize("Titles")))
             {
              String title =
Utilities.getTextElement(documentElement, "caption", "");

              if(title.toLowerCase().indexOf(whatSeek) != -1)
              {
               foundFiles.add(current.getCanonicalPath());
               addResult(documentElement.getAttribute("date"),
title);
              }
             }


if(searchType.getSelectedItem().toString().equals(Memorabilia.getI
nstance().localize("Text")))
             {
              String text =
Utilities.getTextElement(documentElement, "statement", "");
              if(text.toLowerCase().indexOf(whatSeek) != -1)
              {
               foundFiles.add(current.getCanonicalPath());
               addResult(documentElement.getAttribute("date"),
Utilities.getTextElement(documentElement, "caption", ""));
              }
             }


if(searchType.getSelectedItem().toString().equals(Memorabilia.getI
nstance().localize("Tags")))
             {
              // Obtain taglist
              String tagstr =
Utilities.getTextElement(documentElement, "tags", "");
              StringTokenizer tokenizer = new
StringTokenizer(tagstr, ";");
              Vector tags = new Vector();
              while(tokenizer.hasMoreTokens())
               tags.add(tokenizer.nextToken().toString().trim());

              //System.out.println(ontoTags.toString());
              //System.out.println(tags.toString());
              // Cycle all tags and check if any contains in onto taglist
              for(int j = 0; j < tags.size(); j++)
              {
               if(ontoTags.contains(tags.get(j).toString()))
               {
                foundFiles.add(current.getCanonicalPath());
                addResult(documentElement.getAttribute("date"),
Utilities.getTextElement(documentElement, "caption", ""));
                break;
               }
              }
             }
            }

            Memorabilia.getInstance().forceRevalidate();

           } catch (Exception e)
           {
            System.err.println(e.getMessage());
            e.printStackTrace(System.err);
           }
          }

          public void actionPerformed(ActionEvent e)
          {

           if(e.getActionCommand().equals("search"))
           {
            performSearch();
            return;
           }

           super.actionPerformed(e);
          }

          public void mouseClicked(MouseEvent e)
          {
           if(e.getClickCount() >= 2)
           {
            if(e.getSource() instanceof JTable)
            {
             int current = resultsPane.getSelectedRow();

((Memorabilia)Memorabilia.getInstance()).openFromExternal(foun
dFiles.get(current).toString());
            }
           }
          }

          public void mousePressed(MouseEvent e)
          {
          }

          public void mouseReleased(MouseEvent e)
          {
          }

          public void mouseEntered(MouseEvent e)
          {
          }

          public void mouseExited(MouseEvent e)
          {
          }
         }
```

6. Головний клас програми (Memorabilia)

```
package ua.richesa.memorabilia;


import com.toedter.calendar.JCalendar;

import com.toedter.components.JLocaleChooser;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Point;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import java.beans.PropertyChangeEvent;
```

```
import java.beans.PropertyChangeListener;

import java.io.File;

import java.io.FileInputStream;

import java.util.Date;
import java.util.Locale;
import java.util.Vector;

import javax.swing.JFileChooser;
import javax.swing.JList;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JOptionPane;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;

import javax.swing.JTabbedPane;
import javax.swing.JToolBar;

import javax.swing.border.CompoundBorder;

import javax.swing.border.EmptyBorder;
import javax.swing.border.EtchedBorder;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import ua.richesa.helpers.BasicApplication;
import ua.richesa.helpers.login.LoginAdapter;
import ua.richesa.helpers.login.LoginRecord;
import ua.richesa.memorabilia.dialogs.LoginDialog;
import ua.richesa.memorabilia.dialogs.OntologyPane;
import ua.richesa.memorabilia.dialogs.RecordPane;
import ua.richesa.memorabilia.dialogs.RegisterDialog;
import ua.richesa.memorabilia.dialogs.SearchDialog;
import ua.richesa.memorabilia.login.LocalLoginAdapter;
import ua.richesa.memorabilia.tools.DateFileFilter;
import ua.richesa.memorabilia.tools.TagCloud;
import ua.richesa.tools.CustomFileFilter;
import ua.richesa.tools.CustomFileView;
import ua.richesa.tools.Utilities;

public class Memorabilia extends BasicApplication
 implements MouseListener, PropertyChangeListener
{
 private JCalendar dateWidget = new
JCalendar(Locale.ENGLISH);
 private JList recList = new JList();
 private JTabbedPane workspace = new JTabbedPane();

 public Memorabilia()
 {
  setSize(700, 480);
  setTitle("Memorabilia");
  setLoginAdapter(new LocalLoginAdapter(new
LoginDialog(), new RegisterDialog()));

  JMenuBar menuBar = new JMenuBar();
  JToolBar tools = new JToolBar();
  JMenu menu = new JMenu(localize("Records"));
  menu.add(Utilities.createMenuItem(localize("New
Record"), "ctrl N", "newRecord", this, "record_add.png", tools));
  menu.add(Utilities.createMenuItem(localize("Save
Record"), "ctrl S", "saveRecord", this, "record_save.png", tools));
  menu.add(Utilities.createMenuItem(localize("Close
Record"), "ctrl W", "closeRecord", this));
  menu.add(Utilities.createMenuItem(localize("Erase
Record"), "ctrl K", "killRecord", this, "record_erase.png", tools));
  menu.add(new JSeparator());
  menu.add(Utilities.createMenuItem(localize("Search"),
"alt F7", "search", this, "search.png", tools));
  menu.add(new JSeparator());
  menu.add(Utilities.createMenuItem(localize("Quit"), "alt
X", "quit", this));
  menuBar.add(menu);

  menu = new JMenu(localize("Ontology"));
  menu.add(Utilities.createMenuItem(localize("New
Ontology"), "", "newSemantic", this, "onto_new.png", tools));
  menu.add(Utilities.createMenuItem(localize("Edit
Ontology"), "", "editSemantic", this, "onto_load.png", tools));
  menu.add(Utilities.createMenuItem(localize("Save
Ontology"), "", "saveSemantic", this, "onto_save.png", tools));
  menu.add(Utilities.createMenuItem(localize("Close
Ontology"), "", "closeSemantic", this, "onto_close.png", tools));
  menu.add(new JSeparator());
  menu.add(Utilities.createMenuItem(localize("New
class"), "", "newClass", this, "class_new.png", tools));
  menu.add(Utilities.createMenuItem(localize("Remove
class"), "", "killClass", this, "class_kill.png", tools));
  menuBar.add(menu);

  menuBar.add(getLangMenus());
  setJMenuBar(menuBar);

  getContentPane().setLayout(new BorderLayout());
  getContentPane().add(tools, BorderLayout.NORTH);

  JPanel mainPane = new JPanel(new BorderLayout());
  JPanel sidePane = new JPanel(new BorderLayout());

  dateWidget.setBorder(new CompoundBorder(new
EtchedBorder(EtchedBorder.RAISED), new EmptyBorder(5, 5, 5,
5)));
  if(getLocaler().getCurrentLocale().equals("Русский"))
   dateWidget.setLocale(new Locale("ru", "RU"));
  if(getLocaler().getCurrentLocale().equals("Українська"))
   dateWidget.setLocale(new Locale("uk", "UA"));

  sidePane.add(dateWidget, BorderLayout.NORTH);
  JPanel listPane = new JPanel(new BorderLayout());
  listPane.add(recList);
  listPane.setBorder(new CompoundBorder(new
EtchedBorder(EtchedBorder.RAISED), new EmptyBorder(5, 5, 5,
5)));
  sidePane.add(new JScrollPane(listPane),
BorderLayout.CENTER);

  mainPane.add(sidePane, BorderLayout.WEST);
  mainPane.add(workspace, BorderLayout.CENTER);
  getContentPane().add(mainPane,
BorderLayout.CENTER);

  recList.addMouseListener(this);
  dateWidget.addPropertyChangeListener(this);
 }

 public void updateRecordsList()
 {
  try
  {
   File currentDir = new File(".");
   File profileDir = new File(currentDir.getCanonicalPath()
+ File.separator + "profiles" + File.separator +
Memorabilia.getInstance().getLoginName());
   DateFileFilter filter = new
DateFileFilter(dateWidget.getDate());
   File[] records = profileDir.listFiles(filter);
   DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
   DocumentBuilder builder =
factory.newDocumentBuilder();
   Vector fileList = new Vector();
   for(int i = 0; i < records.length; i++)
   {
    FileInputStream fis = new
FileInputStream(records[i].getCanonicalPath());
    Document curDoc = builder.parse(fis);
    Element documentElement =
curDoc.getDocumentElement();

fileList.addElement(Utilities.getTextElement(documentElement,
"caption", ""));
```

```
            fis.close();
            }

            recList.removeAll();
            recList.setListData(fileList);
            forceRevalidate();

           } catch (Exception e)
           {
            System.err.println(e.getMessage());
            e.printStackTrace(System.err);
           }
          }

          public void setAbsoluteCentered()
          {
           Dimension screen =
Toolkit.getDefaultToolkit().getScreenSize();
            setBounds((int)((screen.getWidth() / 2) -
(getBounds().getWidth() / 2)),
                  (int)((screen.getHeight() / 2) -
(getBounds().getHeight() / 2)),
                  (int)getBounds().getWidth(),
(int)getBounds().getHeight());

          }

          public boolean showLogin(LoginAdapter adapter)
          {
           boolean oldstat = super.showLogin(adapter);
           if(!oldstat)
            return oldstat;

           LoginDialog dialog = (LoginDialog)
adapter.getLoginDialog();
            String passwd =
adapter.getLoginPasswd(getLoginName());
            Vector entered = (Vector) dialog.getData();
            if(passwd.equals(entered.get(1).toString()))
             return true;

           return false;
          }

          public static void main(String args[])
          {
           Utilities.MODE_DEPLOYED = 0;
           Memorabilia memo = new Memorabilia();

           LoginAdapter adapter = memo.getLoginAdapter();

((LoginDialog)adapter.getLoginDialog()).updateLoginNames();
            boolean logstat = false;
            while(!logstat)
            {
             logstat = memo.showLogin(adapter);
             if(!logstat)
             {

if(JOptionPane.showConfirmDialog(Memorabilia.getInstance(),
Memorabilia.getInstance().localize("Wrong password! Repeat
login?"),

Memorabilia.getInstance().localize("Request"),
                                  JOptionPane.YES_NO_OPTION,
JOptionPane.WARNING_MESSAGE) ==
JOptionPane.NO_OPTION)
              {
               System.exit(-1);
              }

             }
            }

            memo.setTitle(memo.getTitle() + " - " +
memo.getLoginName());
            memo.updateRecordsList();
            memo.setVisible(true);
            memo.setAbsoluteCentered();
           }

          public void actionPerformed(ActionEvent e)
          {
           if(e.getActionCommand().equals("newRecord"))
           {
            RecordPane newPane = new RecordPane();
            workspace.addTab(localize("Record: ") + "Noname",
newPane);
            return;
           }

          if(e.getActionCommand().equals("saveRecord"))
          {

((RecordPane)workspace.getSelectedComponent()).save(dateWidge
t.getDate());
           workspace.setTitleAt(workspace.getSelectedIndex(),
                        localize("Record: ") +
((RecordPane)workspace.getSelectedComponent()).getCaption());
           updateRecordsList();
           return;
          }

          if(e.getActionCommand().equals("killRecord"))
          {
           if(workspace.getSelectedComponent() != null)
           {

((RecordPane)workspace.getSelectedComponent()).erase();

workspace.remove(workspace.getSelectedComponent());
           }
           else
           {
            if(recList.getSelectedValue() != null)
            {
             String curCaption =
recList.getSelectedValue().toString();
              File checkFile = new
File(findFileByCaption(curCaption));
              if(checkFile.exists())
               checkFile.delete();
            }
           }
           updateRecordsList();
           return;
          }

          if(e.getActionCommand().equals("closeRecord"))
          {
           workspace.remove(workspace.getSelectedComponent());
           return;
          }

          if(e.getActionCommand().equals("newSemantic"))
          {
           OntologyPane newPane = new OntologyPane();
           workspace.addTab(localize("Ontology: ") + "Noname",
newPane);
           return;
          }

          if(e.getActionCommand().equals("saveSemantic"))
          {
           if(workspace.getSelectedComponent() instanceof
OntologyPane)
           {
            OntologyPane onto =
(OntologyPane)workspace.getSelectedComponent();
            onto.saveOntology(false);
           }
           return;
          }

          if(e.getActionCommand().equals("editSemantic"))
          {
           try
           {
            OntologyPane onto = new OntologyPane();
            JFileChooser chooser = new JFileChooser();
            CustomFileFilter filter = new CustomFileFilter("RDF
Documents (*.rdf)");
```

```
                filter.addExtension(".rdf");
                CustomFileView view = new CustomFileView();
                view.addCustomFileView(".rdf", "document_rdf.png");
                chooser.setFileFilter(filter);
                chooser.setFileView(view);
                if(chooser.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION)

onto.openOntology(chooser.getSelectedFile().getCanonicalPath());

                workspace.addTab(localize("Ontology: ") +
chooser.getSelectedFile().getName(), onto);

               } catch (Exception exc)
               {
                System.err.println(exc.getMessage());
                exc.printStackTrace(System.err);
               }
              }

              if(e.getActionCommand().equals("closeSemantic"))
              {
               if(workspace.getSelectedComponent() instanceof
OntologyPane)
               {
                OntologyPane onto =
(OntologyPane)workspace.getSelectedComponent();
                if(onto.checkSaved())
                 workspace.remove(onto);
               }
              }

              if(e.getActionCommand().equals("newClass"))
              {
               if(workspace.getSelectedComponent() instanceof
OntologyPane)
               {

((OntologyPane)workspace.getSelectedComponent()).createNewCla
ss();
               }
              }

              if(e.getActionCommand().equals("killClass"))
              {
               if(workspace.getSelectedComponent() instanceof
OntologyPane)
               {

((OntologyPane)workspace.getSelectedComponent()).killCurrentCl
ass();
               }
              }

              if(e.getActionCommand().equals("search"))
              {
               SearchDialog dialog = new SearchDialog(this);
               dialog.setVisible(true);
               if(dialog.accepted)
               {

               }
              }

              super.actionPerformed(e);
             }

             public void openFromExternal(String what)
             {
              try
              {
               DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
               DocumentBuilder builder =
factory.newDocumentBuilder();
               FileInputStream fis = new FileInputStream(what);
               Document curDoc = builder.parse(fis);
               fis.close();
               Element documentElement =
curDoc.getDocumentElement();
               RecordPane newPane = new RecordPane();
               newPane.open(what);
               workspace.addTab(localize("Record: ") +
Utilities.getTextElement(documentElement, "caption", ""),
newPane);

              } catch (Exception e)
              {
               System.err.println(e.getMessage());
               e.printStackTrace(System.err);
              }
             }

             private String findFileByCaption(String caption)
             {
              try
              {
               File currentDir = new File(".");
               File profileDir = new File(currentDir.getCanonicalPath()
+ File.separator + "profiles" + File.separator +
Memorabilia.getInstance().getLoginName());
               DateFileFilter filter = new
DateFileFilter(dateWidget.getDate());
               File[] records = profileDir.listFiles(filter);
               DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
               DocumentBuilder builder =
factory.newDocumentBuilder();
               for(int i = 0; i < records.length; i++)
               {
                FileInputStream fis = new
FileInputStream(records[i].getCanonicalPath());
                Document curDoc = builder.parse(fis);
                fis.close();
                Element documentElement =
curDoc.getDocumentElement();
                if(Utilities.getTextElement(documentElement,
"caption", "").equals(caption))
                 return records[i].getCanonicalPath();
               }
              } catch (Exception e)
              {
               System.err.println(e.getMessage());
               e.printStackTrace(System.err);
              }
              return "";
             }

             public void mouseClicked(MouseEvent e)
             {
              try
              {
               if((e.getClickCount() >= 2) && (e.getSource() instanceof
JList))
               {
                File currentDir = new File(".");
                File profileDir = new File(currentDir.getCanonicalPath()
+ File.separator + "profiles" + File.separator +
Memorabilia.getInstance().getLoginName());
                DateFileFilter filter = new
DateFileFilter(dateWidget.getDate());
                File[] records = profileDir.listFiles(filter);
                DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
                DocumentBuilder builder =
factory.newDocumentBuilder();
                for(int i = 0; i < records.length; i++)
                {
                 FileInputStream fis = new
FileInputStream(records[i].getCanonicalPath());
                 Document curDoc = builder.parse(fis);
                 fis.close();
                 Element documentElement =
curDoc.getDocumentElement();
                 if(Utilities.getTextElement(documentElement,
"caption", "").equals(recList.getSelectedValue().toString()))
                 {
                  int unique = -1;
                  for(int j = 0; j < workspace.getTabCount(); j++)
                   if(workspace.getComponentAt(j) instanceof
RecordPane)
```

```
                {
                 if(
((RecordPane)workspace.getComponentAt(j)).getRealName().equal
s(records[i].getCanonicalPath()) )
                   unique = j;
                 }
                if(unique == -1)
                {
                 RecordPane newPane = new RecordPane();
                 newPane.open(records[i].getCanonicalPath());
                 workspace.addTab(localize("Record: ") +
Utilities.getTextElement(documentElement, "caption", ""),
newPane);
                }
                else
                 workspace.setSelectedIndex(unique);
               }
              }

             }
            } catch (Exception exc)
            {
             System.err.println(exc.getMessage());
             exc.printStackTrace(System.err);
            }
           }

           public void mousePressed(MouseEvent e)
           {
           }

           public void mouseReleased(MouseEvent e)
           {
           }

           public void mouseEntered(MouseEvent e)
           {
           }

           public void mouseExited(MouseEvent e)
           {
           }

           public void propertyChange(PropertyChangeEvent evt)
           {
            updateRecordsList();
           }
          }
```

Додаток U

Наукова публікація

УДК 004.6

Л. А. Савицька, Т. І. Коробейнікова, В. Д. Тягун

Метод та програмний засіб застосування метаданих в процесах пошуку

Анотація. Нині дуже популярною є підтримка наукових здобутків науково метричними базами даних. Наука наукометрія дозволяє оцінити глибину науковості статей, дати об'єктивну оцінку новій інформації в тій чи іншій галузі. Наприклад, на відому конференцію ACM Conference on Human Factors in Computer Systems, щороку приходить наукові матеріали на рецензування і кількість їх зросла вдвічі за останні 10 років. Але це лише ті матеріали, які пройшли рецензування. Якщо ж виміряти ту кількість статей, що поступають на розгляд комісії конференції, то виявимо, що вона виросла в п'ять разів, тобто, реальний приріст інформаційного потоку в тільки одній галузі становить більше ніж в 5 разів за 10 років. Такий стрімкий, майже експоненційний ріст потоку нових наукових розробок означає, що в адекватні часові терміни практично мало хто може встигнути ознайомитись із цими матеріалами, що й призводить до того, що гіпотетично перспективні технології навіть не були помічені на науковим суспільством в цілому, ні гіпотетичним ринком користувачів. Тому дана робота присвячена розробці та програмній реалізації методу застосування метаданих в процесах пошуку. Цей програмний засіб дозволить досягти збільшення результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів.
Високий рівень вирішення поставленої задачі досягнуто за рахунок використання сучасної мови програмування Java. В даній роботі виконано аналіз сучасних моделей, методів та засобів застосування метаданих в процесах пошуку; розглянуто існуючі аналоги та поточний стан технологій в галузі застосування метаданих в процесах пошуку; запропоновано модель та метод застосування метаданих в процесах пошуку; розроблено ключові процеси роботи методу застосування метаданих в процесах пошуку та на його основі було створено програмний засіб; запропоновано програмну реалізацію запропонованого методу застосування метаданих в процесах пошуку; проведено тестування програмного продукту та виконано аналіз отриманих результатів.

Ключові слова: Метод та програмний засіб застосування метаданих в процесах пошуку, метаданні в процесах пошуку, застосування метаданих в процесах пошуку, метаданні в пошуку.

Abstract. Nowadays, it is very popular to support scientific achievements with scientific metric databases. Scientometric science allows you to assess the depth of scientific articles, to give an objective assessment of new information in a particular field. For example, at the well-known ACM Conference on Human Factors in Computer Systems, every year, scientific papers come for peer review and their numbers have doubled in the last 10 years. But these are only the materials that have been reviewed. If we measure the number of articles submitted to the conference commission, we find that it has increased five times, that is, the real increase in the information flow in only one industry is more than 5 times in 10 years [5]. Such rapid, almost exponential growth in the flow of new scientific development means that in adequate timeframes, almost no one will be able to get acquainted with these materials, which leads to the fact that hypothetically promising technologies have not even been noticed by either the scientific society as a whole or the hypothetical society users. The rapid development of information technology is no longer news, and the further transformation of post-industrial society is closely linked to the acquisition of new knowledge and the effective management of what has already been acquired. Nowadays, it is very popular to support scientific achievements with scientific metric databases. Scientometric science allows you to assess the depth of scientific articles, to give an objective assessment of new information in a particular field. This has been shaping the problem of finding information in specialized systems, archives, including the Internet, for quite some time now. Of course, powerful Google search engines or CiteSeer can make it easier for a person to search and evaluate the relevance of the results they find. However, at the present stage of development, a scientific and creative search is still needed, if not entirely new approaches, it is possible to use non-trivial existing ones, which would improve the situation. Therefore this paper is devoted to the development and software implementation of the method of application of metadata in search processes. This software tool will allow you to increase the search results of documents that meet the query within a static collection of documents. A high level of problem solving has been achieved through the use of modern Java programming language. In this thesis an analysis of modern models, methods and means of using metadata in the search processes, examines the existing analogs and the current state of technology in the application of metadata in the search process, the model and method of application of metadata in search processes is proposed, the key processes of the method of using metadata in the search processes have been developed and, based on this, a software tool was created; the program realization of the proposed method of using metadata in the search processes is proposed, Software product testing was carried out and analysis of the results was performed.

Keywords: Method and software for using metadata in search processes, metadata in search processes, use of metadata in search processes, metadata in search.

Аннотация. Сейчас очень популярны поддержка научных достижений научно метрическими базами данных. Наука наукометрия позволяет оценить глубину научности статей, дать объективную оценку новой информации в той или иной области. Например, на известную конференцию ACM Conference on Human Factors in Computer Systems, ежегодно приходят научные материалы на рецензирование и количество их возросло вдвое за последние 10 лет. Но это лишь те материалы, которые прошли рецензирование. Если же измерить то количество статей, поступающих на рассмотрение комиссии конференции, то обнаружим, что она выросла в пять раз, то есть, реальный прирост информационного потока в только одной отрасли составляет более чем в 5 раз за 10 лет [5]. Такой стремительный, почти экспоненциальный рост потока новых научных разработок означает, что в адекватные временные сроки практически мало кто может успеть ознакомиться с этими материалами, и приводит к тому, что гипотетически перспективные технологии даже не были замечены ни научным обществом в целом, ни гипотетическим рынком пользователей. Поэтому данная работа посвящена разработке и программной реализации метода применения метаданных в процессах поиска. Этот программный средство позволит достичь увеличения результатов поиска документов, удовлетворяющих запросу, в рамках некоторой статической коллекции документов.
Высокий уровень решения поставленной задачи достигнуто за счет использования современного языка программирования Java. В данной работе выполнен анализ современных моделей, методов и средств применения метаданных в процессах поиска; рассмотрены существующие аналоги и текущее состояние технологий в области применения метаданных в процессах поиска; предложена модель и метод применения метаданных в процессах поиска, разработаны ключевые процессы работы метода применения метаданных в процессах поиска и на его основе этого создана программное средство; предложено программную реализацию предложенного метода применения метаданных в процессах поиска; проведено тестирование программного про- дукту и выполнен анализ полученных результатов.

Ключевые слова: Метод и программный способ применения метаданных в процессах поиска, метаданные в процессах поиска, применения метаданных в процессах поиска, метаданные в поиске.

Вступ

Стрімкий розвиток інформаційних технологій вже не новина, подальша трансформація постіндустріального суспільства тісно пов'язана з набуттям нових знань та ефективним керуванням тими, що вже набуті. Такі інтелектуальні прориви є цілком природніми після появи постіндустріального суспільства. Цілком очевидним є те, що підвищення ролі знання веде до стрімкого збільшення інформаційних потоків, які людина отримує щодня. За умов ігнорування частини цього потоку, є ризик втрати актуалізації своїх знань. І навпаки, за умов вмілого та ефективного керування потоками інформації дозволяє зберігати та підвищувати професійну кваліфікацію, просуватися вперед у щоденних справах, створювати новий контент тощо. Закономірно, що рано чи пізно, на якомусь етапі інформаційний потік може стати настільки величезним, що навіть при великому бажанні, і витрачаючи на освоєння знань більшу частину свого часу, людина не встигатиме – бо новий контент буде з'являтись швидше, ніж попередній може бути опрацьованим в принципі. Часто нові технології так і не впроваджуються у життя, чи не виходять на ринок, оскільки встигають морально застаріти, і головною причиною такого стану справ було те, що системи пошуку не встигли їх з різних причин адекватно проіндексувати і ці матеріали так і залишилися незнайденими в безкінечних множинах інформаційних потоків.

Актуальність

З метою полегшити процеси пошуку і виключити аналіз всього тексту в пошуках ключових слів та виразів, в даній роботі пропонується використовувати так звані «мета-теги», які в стислій формі описують вміст сторінки, текстових даних чи якогось ресурсу. Таким чином, кожна веб-сторінка, чи документ, супроводжуються «хмарою тематичних тегів». Звісно їх кількість, на жаль, кінцева, а із швидким ростом інформаційного потоку, на кожен такий тег припадатиме все більше і більше даних, а це, відповідно, вимагатиме додаткових уточнюючих тегів, і зрештою, кожен документ треба буде супроводжувати такою кількістю тегів, що їх індексація за своїм обсягом не буде відрізнятись від повнотекстової інформації.

Мета

Метою дослідження статті є збільшення результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів.

Для досягнення поставленої у роботі мети необхідно розв'язати такі завдання:

- провести аналіз сучасних моделей, методів та засобів застосування метаданих в процесах пошуку;
- розглянути існуючі аналоги та поточний стан технологій в галузі застосування метаданих в процесах пошуку;
- запропонувати модель та метод застосування метаданих в процесах пошуку;
- розробити ключові процеси роботи методу застосування метаданих в процесах пошуку та на його основі цього розробити програмний засіб;
- виконати програмну реалізацію запропонованого методу застосування метаданих в процесах пошуку;
- провести тестування програмного продукту та виконати аналіз отриманих результатів.

1. Методи застосування метаданих в процесах пошуку

У загальному випадку, визначень та класифікацій для широкого поняття метаданих існує немало. Часто , в загальнодоступних джерелах, використовується такий комплект критеріїв, що є спільним для всіх їх типів. Розглянемо для прикладу звичайний текст. Отже, його метадані можуть містити таке:

1) Перелік засобів, що використані і за допомогою яких створено даний текст;
2) Мета цього конкретного тексту;
3) Час та дата створення цього конкретного тексту;
4) Автор цього конкретного тексту;
5) Розташування комп'ютера (гаджета, вузла, хоста), на якому він був створений (розміщений, модифікований тощо);
6) Використані стандарти для відображення цього конкретного тексту.

Часто ці метадані зберігались і знаходилися разом із самим текстом, що вони його характеризують, тобто, в складі формату самого документу. Так, скажімо, відповідні поля для цих дій є в форматах

фірмою Microsoft, а також у багатьох інших, можливо менш поширених способах зберігання інформації та даних. Більше того, подібні поля присутні також в графічних форматах (EXIF, JFIF) та, звісно, у звукових форматах (у вигляді т.з. ID3-тегів).

Проте з часом лише цього стало мало, бо цих даних досить для автоматичної категоризації, як вже говорилось вище, але недостатньо для здійснення складного процесу пошуку і гарної оцінки релевантності текстів. Тому американський фахівець по базам даних (БД) Ральф Кімбелл запропонував розширити систему метаданих загалом, і припустив, що можна поділити метадані на два потужних класи:

- технічні метадані;
- бізнес-метадані.

2. Моделі та технології застосування метаданих у процесах пошуку в Інтернеті

Семантичні структури можуть бути реалізовані досить різними способами. Це може бути реалізовано за допомогою структур, що знаходяться усередині самого файлу, або ж це може бути якась окремо створена спеціальна система, призначена для збирання і зберігання саме семантичних даних.

Одною із перших таких спроб створити подібну систему запропонували Рой Голдман та Дженіфер Уідом. Їхня пропозиція зрештою зводилась до необхідності створення такої особливої БД, до створення т.з. «сітки даних», яка би містила форматовані нотатки та довідки про інформаційні матеріали і ресурси, і які би згодом стали основою для формулювання все більш точних запитів та збирання все більшої кількості статистики, що би дозволила оптимізувати процес пошуку. Саме цими авторами було вперше запропоновано застосувати відому деревоподібну модель деталізації метаданих, та ними ж були сформовані базові поняття, на яких створено сучасні семантичні системи. Нехай існує БД (на прикладі типів закладів харчування) із такою структурою (рис. 2.1).

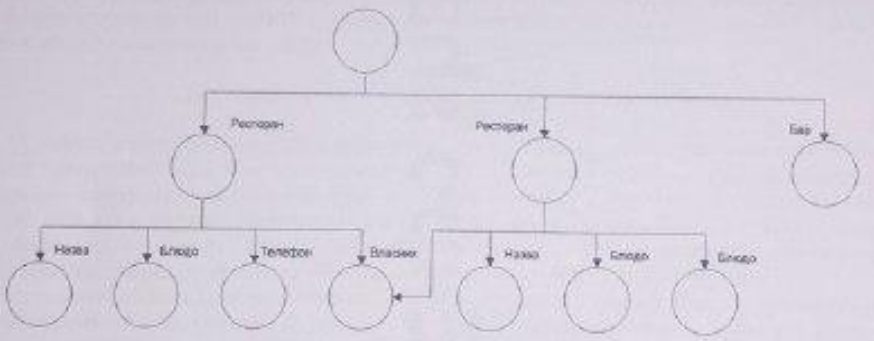


Рисунок 2.1 – Деревоподібна модель БД із нечіткою структурою метаданих

Як видно з рисунка 2.1, в цій БД однакові об'єкти, як не дивно, можуть мати різний перелік властивостей. Причому, деякі із них можуть навіть повторюватись, а деякі – можуть загалі бути відсутніми. З метою впорядкувати все це і дозволити користувачу системи працювати однотипно із усіма елементами системи, незважаючи на їх відмінності, дана БД використовує механізм метаданих, організованих у відповідності до видів і типів елементів. Це може бути окрема структура, що містить список можливих властивостей, які може мати окремий елемент (рис. 2.2).

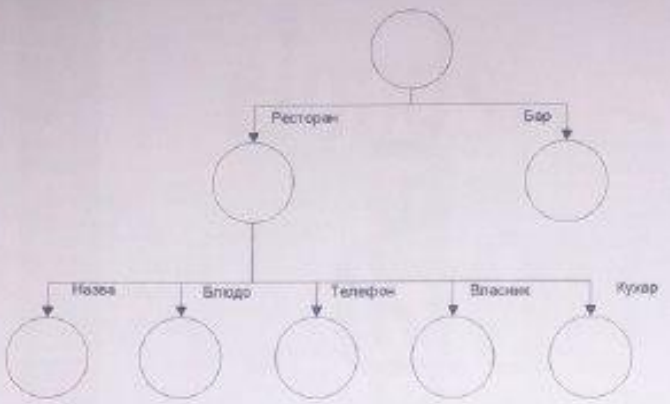


Рисунок 2.2 – Деревоподібна модель БД із чіткою структурою метаданих

Подібний підхід запропонували запровадити і для текстових даних. Згідно цього, в текстовому документі якимось чином виділялися певні «поля» (скажімо, маркувались спеціальними символами, або спеціальними методами розмітки тексту).

Отож, вже нема такої необхідності переглядати весь текст повністю, а можна лиш перевірятн його окремі ключові теги, і ще причому лиш ті, які вказав користувач. Тепер, в поєднанні з процесами індексування та хешування цих полів, ми можемо значно пришвидшувати пошук, і довести повнотекстовий пошук до тієї ж швидкості, що й БД.

3. Модель застосування метаданих в процесах пошуку

Тегування електронних записів, в тому числі і нотаток, що актуально для теми даної магістерської роботи, використовується з метою полегшення процесів пошуку через можливість категоризувати дані всередині електронних записів. Кожному із записів ставиться у відповідність комплект ключових слів (тегів), кожне із яких потім індексується, і отож користувач може переглядати та групувати свої електронні закладки з точки зору різних тематик, чи у відповідності певним задачам. Система тегів була згодом розвинута в модель соціального тегування, яка була покладена в основу Web 2.0.

Першопочатково модель застосування метаданих в процесах пошуку була створена для пришвидшення людського пошуку. Організація контенту за тегами і ключовими словами дозволяла людям самостійно утворювати ієрархії категорій, і отож брати на себе частку функцій та впливати на процеси застосування метаданих в процесах пошуку.

Цілком справедливим є факт, що кількість тегів із зростанням кількості матеріалу завжди також росте, причому зростає експоненційно. Тоді виникає т.з. «хмара тегів», яку часто ілюструють діаграмою частотного розподілу їх використання. Фактично «хмара» є видом зваженого списку, де всі теги групуються відповідно їх частотному розподілу. Графічним і візуальним чином це можна показати на рисунку 3.1, де частота позначається розміром шрифта.

Рисунок 3.1 – Приклад хмари тегів

Далі такий зважений список групується за певними категоріями, далі утворюється так звана «глобальна хмара метаданих» із кластерами, що поділені за елементами та користувачами. Утворені кластери можуть сортуватися за кількістю підкатегорій.

Нечіткий пошук в такій системі із застосуванням метаданих зводиться до гіпотези, що у випадку досить простих тегів вони співпадатимуть із ключовими словами, які задає людина в рядку пошуку, і сам процес пошуку відтак стане більш ефективним. На практиці ця гіпотеза цілковито не потвердилась. Причина в тому, що окрема особа вибудовує власну термінологію ключових слів, керуючись особистою лексикою, яка часто не має відношення до граматичних чи семантичних правил.

Скажімо, синонімічний ряд тегів для ОС Linux виглядає так: Linux, GNU/Linux, Линакс, Лінукс, Лінуха, Ляліке, Лунікс. Це не враховуючи різновидів цієї ОС, які теж вживаються як частина синонімічного ряду. Все це формує явище «фолксономії».

Розглянемо приклад. Нехай користувач створив деякий електронний матеріал, розміщув на якомусь реусрсі, і маркував 5 тегами, які відображають різні аспекти матеріалу.

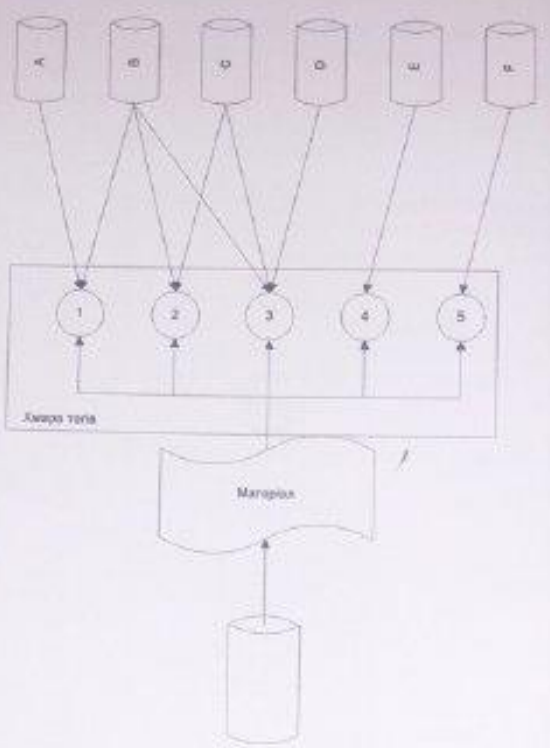


Рисунок 3.2 – Широке фолксономічне тегування

Під час подальшого поширення цього матеріалу, він може бути тегований не всіма цими ключовими словами, а лиш тими, які будуть такими для інших користувачів. І тоді може бути так, що у результаті цей матеріал ніби-то «зникне» із індексів, які відповідають аспектам 4 і 5 (бо вони лишаться в поширеннях значно меншої кількості користувачів), і тоді цей матеріал буде категоризований із більшою ймовірністю в кластери із застосуванням метаданих, що відповідають аспектам 1, 2 і 3 (умовно). І тоді цей матеріал взагалі ризикує бути «не знайденим», бо логічний ланцюжок, який пов'язує всі 5 маркерів матеріалу, будуть втрачені.

Такий фолксономічний підхід дозволяє користувачам систем пошуку формувати свої кластери понять і визначень, і навіть свою власну хмару мета-даних. Проте при цьому дуже важко знайти об'єкти, які носять загальний чи ну дуже окремий характер. Виникає така собі плутанина, і тоді знайти цей матеріал можна, звернувшись до самого автора, або досить довго заглибившись в його публікації. Звісно, що на це в користувача нема часу, і виникає та сама задача, яка була поставлена ще у вступі до даної роботи – проблема т.з. «незнаходжуваності» матеріалу.

Вирішити цю задачу, можна, зберігаючи на якомусь ресурсі ці логічні ланцюжки, і саме тут і може бути корисною модель застосування метаданих в тегах в процесах пошуку в Інтернет.

Вже згадувалося, що за допомогою мета-сутностей можна будувати ієрархії визначень та їх логічних взаємозв'язків. Є ідея дозволити користувачу створювати свої мета-сутності. Тоді, під час процесів пошуку, інший користувач (або пошукова система в автоматичному режимі) зможе звернутись до мета-сутності автора матеріалу, і таки мчином легко відновити всі необхідні логічні зв'язки. Ступінь знаходжуваності документа при такому підході значно підвищується. Звісно, що зільшиться обсяг самих метаданих, але це не становить вели коіпроблеми із сучасним розвитком обчислювальної та комп'ютерної техніки, де обсяги пам'яті уже не має такого критичного значення, як це було іще 10 років назад.

Підхід цей, використовується в «фасетній класифікації», що полягала у узгодженню різних бібліотечних класифікацій, які існували в світі, і створенні «сітчастої» моделі класифікації, де кожна класифікаційна система – це вісь, яку перетинають інші класифікації в точках, де їх визначення співпадали. Звісно, що сама така сітчаста структура не може точно категоризувати матеріал, проте вона дозволяє

переходити від однієї класифікації до іншої, спираючись на спільні визначення. Такий метод узгодження різних ієрархій визначень називається багатоаспектним класифікуванням.

Саме цю модель поклали в основу застосування метаданих до процесів пошуку, і саме наведені елементи фасетної класифікації можна застосувати для реалізації нечіткого пошуку по користувацьким тегам.

Висновки

Результатом статті є збільшення результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекції документів. Було спроектовано демонстраційний приклад, який дозволяє показати, яким чином можна здійснювати нечіткий пошук на основі хмари тегів та семантичних мета-сутностей.

Наукова новизна одержаних результатів полягає в такому:

- вперше запропоновано метод застосування метаданих в процесах пошуку. Запропонований у даній роботі метод застосування метаданих в процесах пошуку дозволяє збільшити результатів пошуку документів, що задовольняють запиту, в рамках деякої статичної колекцій документів;
- вдосконалено модель застосування метаданих в процесах пошуку, що дозволяє створення користувачем власних мета-сутностей;
- вдосконалено процес роботи з мета-даними, який дозволяє створення семантичних ланцюжків між поняттями;
- вдосконалено процес створення мета-сутностей, які дозволяють зменшувати кількість тегів у хмарі тегів.

Запропонований метод застосування метаданих в процесах пошуку, що реалізованої у демонстраційній моделі в інтегральному показнику дозволяє збільшити результатів пошук документів, що задовольняють запиту, в рамках деякої статичної колекцій документів на 12,7% у порівнянні із аналогами.

Список літератури

1. Э. Тоффлер Третья волна – М., АСТ, 2016.
2. V. Bush As We May Think – The Athlantic Monthly, 1 July, 2005 - http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/
3. A. Kay The Early History of Smalltalk – ACM, 2013 - http://www.smalltalk.org/smalltalk/TheEarlyHistoryOfSmalltalk_TOC.
4. Хороший план №9 // Хакер, 29.07.2014 - http://www.xakep.ru/post/23246/default.asp
5. C. Bartneck, J. Hu Scientometric Analysis of The CHI Proceedings - Eindhoven University of Technology, Dept. of Industrial Design, - April, 2019
6. S. Brin, L. Page The Anatomy of a Large-Scale Hypertextual Web Search Engine – Stanford University, Computer Science Dept., 2018
7. H. Li, I. Councill, L. Bolelli, D. Zhou, Y. Song, and others CiteSeer – A Scalable Autonomous Scientific Digital Library – Pennsylvania State University, 2017

Відомості про авторів

Коробейнікова Тетяна Іванівна, к.т.н., доцент кафедри обчислювальної техніки, ВНТУ, кафедра обчислювальної техніки, tetianatroianovska@gmail.com, Вінниця, Хмельницьке шосе, 95

Трояновская Татьяна Ивановна, к.т.н., доцент кафедры вычислительной техники, ВНТУ, кафедра вычислительной техники, tetianatroianovska@gmail.com, Винница, Хмельницкое шоссе, 95

Troianovska Tetiana, PhD, associate professor of computing engineering department, Vinnytsya national technical university, tetianatroianovska@gmail.com, Vinnytsya, Khmelnytsk highway, 95

Савицька Людмила Анатоліївна, к. т. н., доцент кафедри обчислювальної техніки, ВНТУ, кафедра обчислювальної техніки, savytska.liudmyla@vntu.edu.ua, Вінниця, Хмельницьке шосе, 95

Савицкая Людмила Анатольевна, к.т.н., доцент кафедры вычислительной техники, ВНТУ, кафедра вычислительной техники, savytska.liudmyla@vntu.edu.ua, Винница, Хмельницкое шоссе, 95

Savytska Liudmyla, к.т.н., associate professor of computing engineering department, Vinnytsya national technical university, savytska.liudmyla@vntu.edu.ua, Vinnytsya, Khmelnytsk highway, 95

Тягун Дмитро Тарасович, магістр кафедри обчислювальної техніки, ВНТУ, кафедра обчислювальної техніки, dmytriy.tyagun@gmail.com, Вінниця, Хмельницьке шосе, 95

Тягун Дмитрий Тарасович, магистр кафедры вычислительной техники, ВНТУ, кафедра вычислительной техники, dmytriy tyagun@gmail.com, Винница, Хмельницкое шоссе, 95

Tyagun Dmitry Tarasovich, magister of computing engineering department, Vinnytsya national technical university, department of the computer engineering, dmytriy.tyagun@gmail.com, Vinnytsya, Khmelnytsk highway, 95

Л.А. Савицька

МЕТОД ТА ПРОГРАМНИЙ ЗАСІБ ЗАСТОСУВАННЯ МЕТАДАНИХ В ПРОЦЕСАХ ПОШУКУ

Вінницький національний технічний університет, м. Вінниця

L.A. Savytska

METHOD AND SOFTWARE FOR APPLYING METADATA TO SEARCH PROCESSES

Vinnitsa national technical university. Vinnitsa