МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

Пояснювальна записка
до дипломної роботи
бакалавр
(освітньо-кваліфікаційний рівень)

на тему «Розробка програмного забезпечення електронного словника»

Виконав: студент 4 курсу, групи ІТ-141
напряму підготовки 6.040302 „Інформатика”
спеціальності 7.04030201 „Інформатика”

_____ Войтіков А.Ю.
(підпис)

Керівник,
доцент, д.т.н._____ Лифар В.О.
(підпис)

Рецензент,
доцент, к.т.н._____ Фесенко Т.М.
(підпис)

СЄВЄРОДОНЕЦЬК – 2018

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

бакалаврської роботи студента гр. ІТ-141 Войтікова А.Ю..

Науковий керівник Лифар В.О., в.о.завідувача каф. ПМ СНУ ім. В.Даля

ПІБ, посада

Оцінка наукового керівника:

Рецензент Фесенко Т.М., доцент каф. ПМ СНУ ім. В. Даля

ПІБ, місто роботи, посада

Оцінка рецензента:

Кінцева оцінка за результатами захисту:

Голова ЕК,
зав. кафедри ПМ
д.т.н., доцент _____ Лифар В.О.

підпис підпис, дата

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет інформаційних технологій та електроніки
Кафедра програмування та математики
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки 6.040302 „Інформатика”
Спеціальність 7.04030201 „Інформатика”

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМ,
д.т.н., доцент
_____ Лифар В.О.
«___» _____ 2018 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
ВОЙТІКОВУ АНДРІЮ ЮРІЙОВИЧУ

1. Тема роботи Розробка програмного забезпечення електронного словника.
керівник роботи Лифар В.О.
затверджені наказом вищого навчального закладу від “20” лютого 2018 року № 52/48
2. Строк подання студентом роботи 04 червня 2018 р.
3. Вихідні дані до роботи
Об'єктом даної роботи є автоматична обробка природномовних текстів..
3.1 Літературні джерела:

Баранов А.Н. Введение в прикладную лингвистику. Изд. 3-е. / Баранов А.Н. – М.: Издательство ЛКИ, 2007. – 360с.

Беляева Л.Н., Герд А.С., Убин И.И. Автоматизация в лексикографии / Беляева Л.Н., Герд А.С., Убин И.И. – СПб., Питер, 1996. – 318-333с.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
4.1 Вступ
4.2 Аналіз предметної галузі (огляд літератури), з висвітленням наступних питань:
Суть і призначення словників. Лексикографія
Устрій словника
Комп'ютерна лексикографія
4.3 Основна частина, в якої висвітлити:
Технологічні засади розробки електронного термінологічного словника
Розробка програми для автоматизації термінографічних робіт
4.4 Висновки
4.4 Перелік використаних джерел
5. Перелік графічного матеріалу немає
6. Дата видачі завдання 12 лютого 2018 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	**Примітка**
1	Одержання завдання на виконання роботи	12.02.18	
2	Укладання і погодження з керівником плану і етапів виконання роботи	20.02.18	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	1.03.18	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	09.0318	
3	Аналіз технологічних засад розробки електронного термінологічного словника	02.04.18	
5	Укладання та тестування програмного продукту	20.04.18	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	10.05.18	
7	Здача готової пояснювальної записки на кафедру	21.05.18	
8	Укладання доповіді і презентації	01.06.18	

Студент _____ Войтіков А.Ю.
(підпис)

Керівник роботи _____ Лифар В.О.
(підпис)

РЕФЕРАТ

Робота містить: 65 сторінок основного тексту, 9 рисунків, 1 таблицю та 25 використаних джерел.

Дипломна робота складається з чотирьох розділів, вступу, висновків і додатків. Об’єктом дослідження даної роботи є методи та засоби укладання термінологічного словника.

Мета роботи полягає в проектування та розробці компонентів програмних засобів автоматизації термінографічних робіт.

В ході виконання роботи було досліджено методи побудови багатомовних термінологічних словників, визначено необхідні функціональність, структура та засоби розробки. В результаті отримано багатофункціональний програмний засіб для автоматизації термінографічних робіт, який дає можливість не лише використовувати словник, але і створювати або редагувати словникову статтю, форматувати її, додавати нові мови представлення або інтерпретації словникової статті.

Ключові слова: БАГАТОМОВНИЙ ТЕРМІНОЛОГІЧНИЙ СЛОВНИК, КОМПОНЕНТ ПРОГРАМНОЇ СИСТЕМИ, СЛОВНИКОВА СТАТТЯ, ЛІНГВІСТИЧНА БАЗА ДАНИХ, XML-ДОКУМЕНТ, ПАРСЕР, DOM, DTD-МОДЕЛЬ, РЕГУЛЯРНІ ВИРАЗИ, JAVA.

ЗМІСТ

ВСТУП

Актуальність. Розвиток науки і техніки приводить до зміни терміносистеми – з'являються нові терміни, нові значення старих термінів, а отже, виникає необхідність в оновленні словників з метою фіксації змін.

Термінологічні словосполучення, які асоційовані із значущими поняттями предметної області, відіграють велику роль в методах обробки і аналізу текстів. Вони служать основою для формування інформаційно-пошукових тезаурусів, рубрикаторів, а також вирішують задачу поповнення вже існуючого тезауруса і можуть бути застосовані для укладання багатомовного словника.

Розробка термінологічних словників в тій або іншій сфері діяльності є тривалою і складною процедурою, оскільки на ринку не існує програмного продукту, що підтримує всі етапи підготовки словника. Тому представляється актуальним завдання розробки інструменту для створення багатомовного термінологічного словника.

Об'єкт дослідження – методи та засоби укладання термінологічного словника.

Предмет дослідження – засоби автоматизації термінографічних робіт.

Мета дослідження: автоматизація термінографічних робіт.

Задачі дослідження:

- проаналізувати моделі побудови термінографічних словників;
- розробити XML-документ термінологічного словника: основні частини, базові типи, структура;
- зробити порівняльний аналіз DTD-моделі й XML-схем;
- проаналізувати основні програмні інтерфейси. Розібрати побудований XML-документ за допомогою DOM-парсеру;
- розробити програмні засоби для розробки словника та автоматизації пошуку термінів.

РОЗДІЛ 1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО ПОБУДОВУ І СТРУКТУРУ СЛОВНИКІВ

1.1. Суть і призначення словників. Лексикографія

Словник – це організовані збори слів, зазвичай з приписаними їм коментарями, в яких описуються особливості їх структури і функціонування. Найчастіше коментується семантична (смислова) структура слів, тобто словам в словнику зіставляються пояснення (тлумачення) їх значень і вживання, але можливі і багато інших типів коментарів. Окрім слів, об'єктами словникового опису можуть виступати їх компоненти (такі, наприклад, словники морфем), словосполуки різних типів, стійкі сентенції – прислів'я, приказки, цитати і тому подібне. Існують також словники, в яких спеціальні коментарі при кожній одиниці словникового опису відсутні.

У іншому значенні термін «словник» позначає всю сукупність слів деякої мови (інакше кажучи, його лексику) і протиставляється терміну «граматики», що позначає сукупність правил побудови із слів складніших мовних виразів.

Лінгвістична дисципліна, в центрі уваги якої знаходяться методи створення (складання) словників, називається лексикографією (від греч. lexis "слово" і grafia "писання, наука"). У відмінності від лексикології – теоретичної дисципліни, частиною теоретичної семантики, що є, і зайнятою розробкою методів опису плану вмісту лексем (у тому числі методів тлумачення), проблематика лексикографічних робіт лежить саме в словниковій сфері. Центр її інтересу – типи словників і способи організації словникової статті. Зрозуміло, тип словника прямо визначається структурою словникової статті і навпаки.

Лексикографія – це одночасно наука і мистецтво. Лексикограф є вченим настільки, наскільки він прагне обережно тлумачити (пояснити)

слова, і художником в тій мірі, в якій він бачить і задовольняє всілякі потреби своїх читачів. Людська складова завжди була важлива для цього роду діяльності, і деякі укладачі словників заслужили на репутацію людей твердих і незалежних думок.

До компетенції лексикографії відноситься також завдання розробки технологій складання словників. В даний час у зв'язку з розвитком комп'ютерної техніки створюються комп'ютерні технології створення словників, а також всілякі електронні словники, поширені на компакт-дисках або розміщені на серверах комп'ютерних мереж [2].

1.2. Устрій словника

1.2.1. Основні структурні компоненти словника

Кожен словник складається з ряду компонентів, що забезпечують читачеві доступ до інформації, що міститься в ньому. Перший найважливіший компонент – зміст словника. У словник включаються всі одиниці, які формують область опису словника і є входами словникових статей. Фактично словник задає область опису словника. Словник може складатися з морфем (для словників морфем і граматичних словників), лексем (наприклад, для тлумачних словників), словоформ (для граматичних словників) і словосполук (наприклад, для словників фразеологізмів, словників ідіом). Елементарною одиницею словника є словникова стаття – кожен окремо взятий об'єкт опису словника і зіставлені йому словникові характеристики. Безліч словникових статей формує основний текст словника. Окремий структурний компонент утворюють покажчики. У звичайному словнику тлумачень покажчики зустрічаються рідко. Абсолютно інша ситуація із словниками фразеологізмів і словниками ідіом. Оскільки базова форма одиниць фразеологізмів схильна до варіювання, наприклад, покласти зуби на полицю і зуби на полицю покласти, то будь-який вибраний спосіб впорядковування словника не забезпечує легкого пошуку. Для полегшення

процедури пошуку потрібної ідіоми для словників фразеологізмів створюються покажчики, які дозволяють знаходити ідіому по будь-якому з її компонентів. Часто покажчики включаються і в структуру тезаурусів і в структуру двомовних словників. Покажчики тезауруса дають можливість визначити, в які таксони входить те або інше слово, а покажчики двомовних словників частково виконують функції зворотного словника по відношенню до даного [14].

Важливою структурною частиною лінгвістичного словника є список джерел. У європейській словниковій традиції його наявність абсолютно необхідна, оскільки використання будь-яких вже опублікованих текстових матеріалів (у тому числі і в прикладах) вимагає відповідного дозволу від тримача авторського права.

Особливою частиною словника можна вважати статтю вводу, в якій пояснюються принципи користування словником і міститься інформація про структуру словникової статті. Інколи структура словникової статті виноситься в особливий розділ словника. Крім того, лінгвістичні словники, як правило, включають в свій склад список умовних скорочень і алфавіт.

1.2.2. Основні структурні зони словникової статті

Базова одиниця словника – словникова стаття – складається з декількох зон опису. Кожна зона містить особливий тип словникової інформації. Перша зона – лексичний вхід словникової статті, вокабула або лема. Часто у вокабулі вказується наголос. Лексичний вхід зазвичай виділяють напівжирним шрифтом, і тому в жаргоні лексикографів і редакторів ця зона часто називається «чорним словом». У словнику тлумачень після лексичного входу найчастіше слідують зона граматичної інформації і зона стилістичної інформації. Граматична інформація про слово в словниках тлумачення вказує приналежність до частини мови, характерні граматичні форми (наприклад, для іменників – форма родового відмінка однини і вказівка на рід). Комплекс стилістичної інформації дає уявлення про обмеження на вживання слова: літературна мова vs. діалекти, термін vs. не термін; різні стилі літературної

мови [1]. У словниках тлумачення далі слідує зона значення, яка розділяється на окремі підзони:

- номер значення;
- додаткові граматичні і стилістичні додатки;
- зона тлумачення;
- зона прикладів;
- зона відтінків значення.

У словниках тлумачень словникова стаття, як правило, має зону фразеологізмів. Оскільки зона фразеологізмів зазвичай маркірується знаком ромба, на жаргоні лексикографів вона називається «заромбовоною» зоною.

Крім того, для повнішого опису слова в деяких випадках наводиться етимологічна або історична інформація – зона етимології.

На Рис.1.1 приведена типова словникова стаття дієслова «лишить» Малого академічного словника (Словарь русского языка в 4-х тт./под ред. А.П.Евгеньевой).

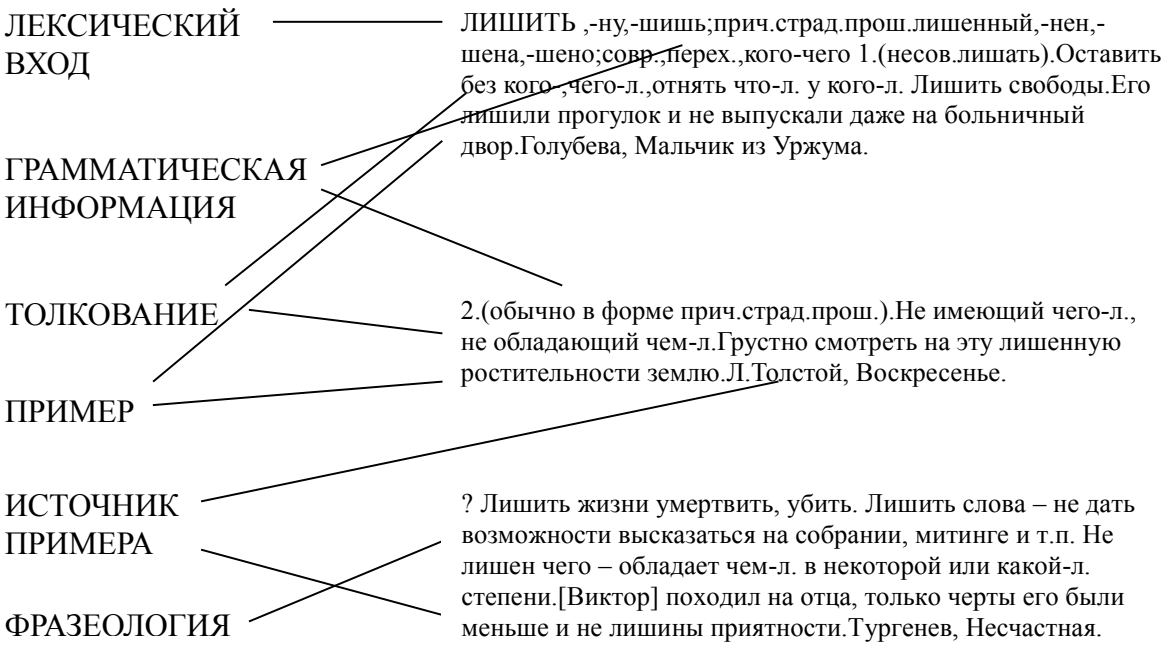

Рис.1.1 – Приклад словникової статті

Кожен тип лінгвістичного словника характеризується своєю структурою словникової статті. Так, нормативні і дескриптивні словники відрізняються не лише вибором матеріалу, але і його організацією в словнику.

Нормативний словник:

1. Лексичний вхід (вокабула).
2. Стилістична інформація.
3. Граматична інформація.
4. Тлумачення.
5. Приклади вживання.
6. Зона ідіоматики (стійкі поєднання, фразеологізми).

Дескриптивний словник:

1. Лексичний вхід (вокабула).
2. Варіанти.
3. Сфери вживання.
4. Граматична інформація.
5. Тлумачення.
6. Приклади вживання.
7. Приклади нестандартних вживань.
8. Зона ідіоматики (стійкі поєднання, фразеологізми).

Тим самим словник дескриптивного типа представляє по можливості всі особливості вживання слова, а нормативний звертає увагу на літературний стандарт, розділяючи лексику на стилістичні області і формуючи стандарт літературної мови.

Абсолютно іншу структуру словникової статті мають частотні словники, словники метафор і епітетів, різні види «словників мовних форм». Зокрема, в них практично відсутня зона тлумачення.

Декілька розмитим поняття словникової статті виявляється для тезаурусів, оскільки в них представлена ієрархія семантичних стосунків усередині лексики. В цьому випадку краще говорити про ієрархічну

побудову словникової статті. Остання з'являється в тезаурусі як сукупність ієрархічно впорядкованих термінальних таксонів, формуючих таксони вищих рівнів. Словникову статтю тезауруса утворює таксон, формально не підпорядкований жодним іншим таксонам [1].

Для маркування зони словникової статті використовуються різні види графічного виділення, що дозволяють читачеві легко знаходити і розділяти типи інформації, що зіставляються в словниковому представленні того або іншого слова. Найбільш типові способи графічного виділення, прийняті в лексикографічній традиції, приведені в таблиці 1.2.

Таблиця 1.2

Типові способи топографематичного виділення словникових зон

Зона словникової статті і / або її компонентів	Виділення
Лексичний вхід	напівжирний шрифт / абзацний відступ / висячий рядок
Граматична інформація	менший кегль / курсив
Стилістична інформація	менший кегль / курсив / початкова прописна буква
Номер тлумачення	напівжирний шрифт
Тлумачення	звичайний прямий
Приклад	курсив / графічний маркер прикладу
Джерело / автор прикладу	менший кегль
Зона фразеологізмів/ідіом	графічний маркер
Фразеологізми	напівжирний шрифт / курсив

Різні графічні виділення тісно зв'язані один з одним. Наприклад, використання курсиву для маркування джерел приклада вживання робить неможливим його використання для виділення самого прикладу. Правила поєднання виділень вивчаються не лише в книжковій справі, але і в лінгвістиці, семіотиці, теорії реклами і теорії дії.

1.3. Комп'ютерна лексикографія

Сучасна лексикографія істотно розширила і підсилила свій інструментарій комп'ютерними технологіями створення і експлуатації словників. Спеціальні програми – бази даних, комп'ютерні картотеки, програми обробки тексту – дозволяють в автоматичному режимі формувати словникові статті, зберігати словникову інформацію і обробляти її. Деякі сучасні словники зобов'язані своїм існуванням саме комп'ютерним технологіям обробки великих корпусів текстів. Наприклад, словникові видання видавництва COLLINS ґрунтуються на Базі даних англійської мови (Бірмінгемський університеті, робоча група COBUILD), яка в даний час налічує близько 200 млн. слововживань. Використання комп'ютерного інструментарію істотно полегшує працю лексикографів і підвищує ефективність лексикографічних робіт [2].

Робота лексикографа безпосередньо пов'язана із словами, прикладами їх вживання і словниковими статтями розробленого словника. Традиційна форма фіксації словникових даних – каталожна картка, в якій вказується слово, приклад вживання, автор, а також різна додаткова інформація. Сучасні комп'ютерні технології дозволяють спростити процес збору і зберігання лексикографічної інформації, використовуючи замість звичайної картотеки базу даних, записи якої є аналогом традиційної каталожної картки. На відміну від звичайної картотеки, записи бази даних дають можливість автоматично сортувати масив по вибраним параметрам, відбирати потрібні приклади, об'єднувати їх в групи і так далі. Спеціалізовані лексикографічні бази даних – маються на увазі спеціальні програмні оболонки – на ринку відсутні. Проте сучасні бази даних типа D-Base, ACCESS, FOX-Base, PARADOX сповна використовуються для ведення електронних словникових картотек [1].

Лексикографічні бази даних фіксують первинний лексикографічний матеріал, який використовується для написання словникових статей словника.

Ще один важливий етап лексикографічної роботи – пошук прикладів вживання слова і формування картотеки прикладів. У традиційній технології збір прикладів проводиться вручну і віднімає величезну кількість часу. Сучасні комп'ютерні програми дають можливість вибирати приклади потрібного слова з корпусів текстів, що зберігаються в машинному форматі на комп'ютері, в автоматичному режимі. Пошук прикладів вживання слова називається побудовою конкордансів. Деякі комп'ютерні програми побудови конкордансів за бажанням користувача можуть перетворювати знайдені контексти в записі бази даних. Наприклад, програма DIALEX дозволяє отримувати конкорданси як в традиційній формі (у вигляді файлу для текстового редактора), так і у форматі бази даних PARADOX.

Після підготовки первинного словникового матеріалу – словникової картотеки – безпосередньо слідує етап складання словникової статті. Технологічний ланцюжок словникових робіт і тут не залишається без комп'ютерної підтримки. Нова словникова стаття вводиться в базу даних, яка стає вихідною базою даних створюваного словника. Редагування словникових статей також відбувається в базі даних, а не в звичайному текстовому файлі. Все це істотно скорочує час розробки словника, оскільки спрощується обробка системи відсилань, в автоматичному режимі відбувається сортування (у тому числі сортування за алфавітом словникових статей), порівняно легко породжуються різні покажчики. Для редагування словника можна залучати комп'ютерні програми перевірки орфографії.

Нарешті, останній етап – формування тексту словника, створення оригінал-макету книги – також істотно полегшується. Технологічний ланцюжок і тут не уривається: існуюче програмне забезпечення дозволяє видати текстовий матеріал відразу з бази даних з розміткою під топографематичне виділення. Після запису бази даних трансформуються в

автоматичному режимі в зони словникової статті з відповідними шрифтами, кеглями, курсивом, підкресленнями та ін.

На Рис.1.3 та Рис.1.4 представлені етапи лексикографічної роботи в традиційному варіанті та комп'ютерна технологія створення словника. Зрозуміло, у кожному конкретному випадку проекти створення словників можуть модифікувати стандартні схеми. Наприклад, в деяких випадках для збору корпусу прикладів можуть використовуватися не лише корпуси текстів, але і лексикографічні бази даних [2]. Так, проект словника Фразеологізмів сучасної російської мови спирається не лише на корпус текстів по сучасній російській мові (що включає тексти художньої прози, публіцистики, детективної літератури), але і на базу даних по сучасній ідіоматиці, що включає в даний час близько 50 тисяч контекстів вживання ідіом. Іншими словами, корпус прикладів формується не лише в результаті

Рис.1.3 – Традиційна технологія створення словника

обробки корпусу текстів, але і бази даних. Для словників, призначених письменникам, може бути передбачений етап формування корпусу текстів письменників-сучасників, необхідний для виявлення відмінностей між

особливостями ідіостиля даного автора і загальними характеристиками мови відповідної епохи.

Особливо слід згадати про існування видавничих систем, використовуваних для створення оригінал-макету (верстки) словників. До них відносяться, наприклад, видавничий пакет програм Quark-X-Press, різні версії програм Page-Marker і WinWord. Для словникової верстки найбільш зручні системи, що мають вбудовані мови, що дозволяють формувати макроси – відносно прості, але технологічно ефективні операції обробки редагованого і верстаного тексту. До них відносяться процедури приписування стилів зонам словникової статті, сортування по алфавіту, створення покажчиків і тому подібне.

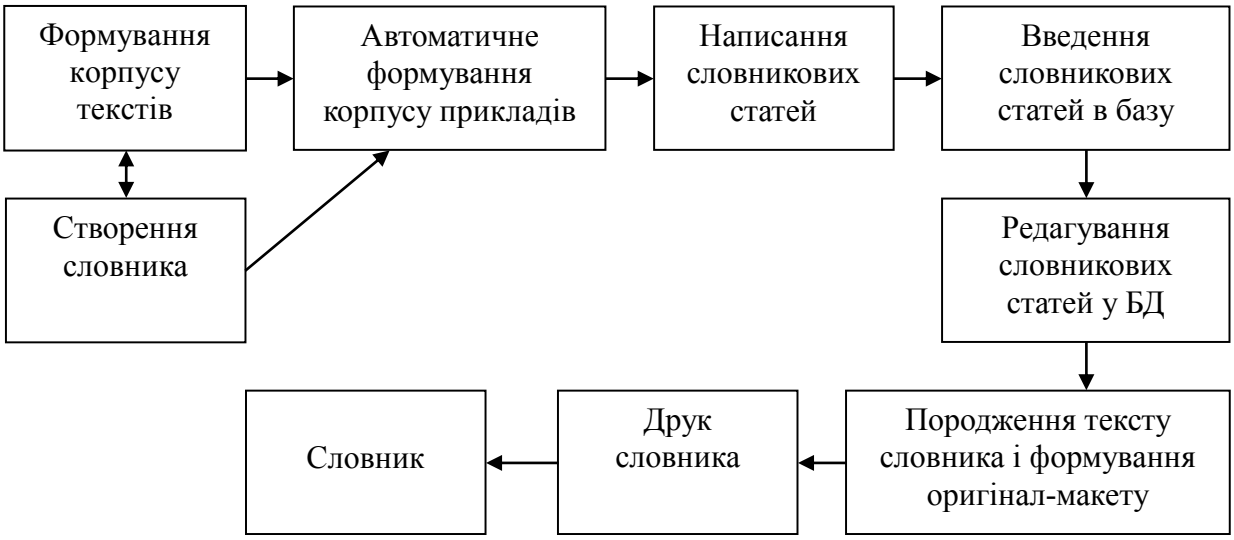

Рис.1.4 – Комп'ютерна технологія створення словника

РОЗДІЛ 2. ТЕХНОЛОГІЧНІ ЗАСАДИ РОЗРОБКИ ЕЛЕКТРОННОГО ТЕРМІНОЛОГІЧНОГО СЛОВНИКА

2.1. Основи XML

Мова розмітки документів – це набір спеціальних інструкцій, тегів, призначених для формування в документах деякої структури і визначення відносин між різними елементами цієї структури. Теги мови, або, як їх іноді називають, управляючі дескриптори, у таких документах якимось чином кодуються, виділяються щодо основного вмісту документа і служать у якості інструкцій для програми, що демонструє вміст документу на стороні клієнта. У самих перших системах для позначення цих команд використовувалися символи "<" і ">", усередині яких містилися назви інструкцій та їх параметри. Зараз такий спосіб позначення тегів є стандартним.

Використання гіпертекстової розбивки текстового документа в сучасних інформаційних системах багато в чому пов'язане з тим, що гіпертекст дозволяє створювати механізм нелінійного перегляду інформації. У таких системах дані представляються не у вигляді безперервного потоку текстової інформації, а набором взаємопов'язаних компонентів, перехід за якими здійснюється за допомогою гіперпосилань.

XML (Extensible Markup Language) – це мова розмітки, що описує цілий клас об'єктів даних – XML–документів.

XML – текстовий формат, призначений для зберігання структурованих даних (замість існуючих файлів баз даних), для обміну інформацією між програмами, а також для створення на його основі більш спеціалізованих мов розмітки (наприклад, XHTML), іноді званих словниками. XML є спрощеним підмножиною мови SGML [10].

Ця мова використовується в якості засобу для опису граматики інших мов і контролю за правильністю складання документів. Тобто сам по собі

XML не містить ніяких тегів, призначених для розмітки, він просто визначає порядок їх створення.

Метою створення XML було забезпечення сумісності при передачі структурованих даних між різними системами обробки інформації, особливо при передачі таких даних через Інтернет. Словники, засновані на XML (наприклад, RDF, RSS, MathML, XHTML, SVG), самі по собі формально описані, що дозволяє програмно змінювати і перевіряти документи на основі цих словників, не знаючи їх семантики, тобто, не знаючи смислового значення елементів.

XML – це ієрархічна структура, призначена для зберігання будь–яких даних, візуально структура може бути представлена як дерево. Найважливіша обов'язкова синтаксична вимога – це те, що документ має тільки один кореневий елемент (альтернативно званий елементом документа). Це означає, що текст або інші дані всього документа повинні бути розташовані між єдиним початковим кореневим тегом і відповідним йому кінцевим тегом.

Перший рядок XML–документа називається рекламою XML (XML declaration) – це необов'язкова рядок, що вказує версію стандарту XML (звичайно це 1.0), також тут може бути вказана кодування символів і зовнішні залежності. Решта XML–документа складається з вкладених елементів, деякі з яких мають атрибути та вміст. Елемент зазвичай складається з відкриваючих і закриваючих тегів, обрамляючи текст та інші елементи. Вмістом елементу (content) називається все, що розташоване між відкриваючим і закриваючим тегом, включаючи текст та інші (вкладені) елементи.

2.2. DTD–модель. Основні риси та перспективи застосування

2.2.1. Цілі DTD–моделі

Звичайно ж, DTD створений не просто так. Коріння даного формату вирушає ще в SGML. Можна навіть сказати інакше: DTD є обов'язковою

частиною застосування SGML (яким є і HTML, наприклад). У нім оголошуються і описуються всілякі правила, застосовні саме до даної мови розмітки. І справді: в HTML свої правила, в XHTML вже трохи інші. Навіть цю різницю необхідно десь відобразити, аби було видно, що це не одне і те ж.

Зрозуміло, DTD є не просто шаром між різними програмістами: DTD – це надійна опора для валідаторів і браузерів. Так, саме DTD використовують валідатори для того, щоб перевірити документ на відповідність всім необхідним вимогам. Браузер у меншій мірі використовує даний документ.

Сформулюємо основні цілі, які переслідує DTD:

- шар між програмістами;
- опис набору правил, відповідних для конкретної мови розмітки;
- самодокументація набору правил;
- відособлення різних застосувань SGML (HTML, XHTML і так далі);
- уніфікація способів запису правил розмітки на різних мовах.

Кожне правило тим або іншим чином перетинається з іншими правилами, таким чином утворюючи одну велику структуру даних описового характеру.

2.2.2. Синтаксис DTD

У XML-документах DTD-модель визначає набір дійсних елементів, ідентифікує елементи, які можуть знаходитися в інших елементах, і визначає дійсні атрибути для кожного з них. Синтаксис DTD вельми своєрідний і від автора–розробника потрібні додаткові зусилля при створенні таких документів(складність DTD є одній з причин того, що використання SGML, що вимагає визначення DTD для будь–якого документа, не набуло настільки широкого поширення як, наприклад, HTML). Як вже наголошувалося, в XML використовувати DTD не обов'язково – документи, створені без цих правил, правильно оброблятимуться програмою–аналізатором, якщо вони

задовольняють основним вимогам синтаксису XML. Проте контроль за типами елементів і коректністю стосунків між ними в цьому випадку повністю покладатиметься на автора документа. До тих пір, поки граматика нашої нової мови не описана, його зможемо використовувати лише ми, і для цього ми будемо вимушені застосовувати спеціально розроблене програмне забезпечення, а не універсальні програми-аналізатори [10].

У DTD для XML використовуються наступні типи правил: правила для елементів і їх атрибутів, описи категорій(макровизначень), опис форматів бінарних даних. Всі вони описують основні конструкції мови – елементи, атрибути, символьні константи зовнішні файли бінарних даних.

Для того, щоб використовувати DTD в нашому документі, ми можемо або описати його в зовнішньому файлі і при описі DTD просто вказати заслання на цей файл або ж безпосередньо усередині самого документа виділити область, в якій визначити потрібні правила. У першому випадку в документі вказується ім'я файлу, опису, що містить, DTD:

```
<?xml version="1.0" standalone="yes" ?>
<! DOCTYPE terminological_dictionary SYSTEM "termin_diction.dtd">
...
```

Усередині ж документа DTD– декларації включаються таким чином:

```
...
<! DOCTYPE terminological_dictionary [
<! ELEMENT interlang (phraseological_units | interpretation | gram_info |
                stylist_moment | leks_login)*>
]>
...
```

В тому випадку, якщо використовуються одночасно внутрішні і зовнішні описи, то програмою–аналізатором спочатку розглядатимуться внутрішні, тобто їх пріоритет вищий. При перевірці документа XML–процесор в першу чергу шукає DTD усередині документа. Якщо правила усередині документа не визначені і не заданий атрибут standalone ="yes", то

програма завантажить вказаний зовнішній файл і правила, що знаходяться в нім, будуть прийняті звідти. Якщо ж атрибут standalone має значення "yes", то використання зовнішніх описів DTD буде заборонено.

2.2.3. Визначення елементів DTD–моделі

Елемент в DTD визначається за допомогою дескриптора !ELEMENT, у якому вказується назва елементу і структура його вмісту.

Наприклад, для елементу <log_info> можна визначити наступне правило:

<!ELEMENT log_info PCDATA>

Ключове слово ELEMENT вказує, що даною інструкцією описуватиметься елемент XML. Усередині цієї інструкції задається назва елементу (log_info) і тип його вмісту.

У визначенні елементу ми вказуємо спочатку назву елементу(log_info), а потім його модель вмісту – визначаємо, які інші елементи або типи даних можуть зустрічатися усередині нього. В даному випадку вміст елементу log_info визначатиметься за допомогою спеціального маркера PCDATA(що означає parseable character data – будь–яка інформація, з якою може працювати програма–аналізатор). Існує ще дві інструкції, що визначають типа вмісту: EMPTY, ANY. Перша вказує на те, що елемент має бути порожнім(наприклад, <log_info/>), друга – на те, що вміст елементу спеціально не описується.

Послідовність дочірніх для поточного елементу об'єктів задається у вигляді списку розділених комами назв елементів. При цьому для того, щоб вказати кількість повторень включень цих елементів можуть використовуватися символи +, *, ? :

<!ELEMENT interpretation (example_source|example|interpret|log_info)*>

В даному прикладі вказується, що усередині елементу <interpretation> мають бути визначені елементи example_source, example, interpret і log_info. В тому випадку, якщо існує декілька можливих варіантів вмісту визначуваного елементу, їх слід розділяти за допомогою символу "|".

Символ * в даному прикладі вказує на те, що визначувана послідовність внутрішніх елементів може бути повторена кілька разів або ж зовсім не використовуватися.

Якщо у визначенні елементу вказується "змішаний" вміст, тобто текстові дані або набір елементів, то необхідно спочатку вказати PCDATA, а потім розділений символом "|" список елементів.

Приклад коректного документа XML:

```
<!ELEMENT terminology_dictionary (word)*>
<!ELEMENT word (interlang)*>
<!ATTLIST word id CDATA #REQUIED>
<!ELEMENT interlang ( leks_login | stylist_moment | gram_info |
                interpretation | phraseological_units)*>
<!ATTLIST interlang language CDATA #IMPLIED >
<!ELEMENT leks_login (#PCDATA)>
<!ELEMENT stylist_moment (#PCDATA)>
<!ELEMENT gram_info (#PCDATA)>
<!ELEMENT interpretation (example_source|example|interpret|log_info)*>
<!ELEMENT log_info (#PCDATA)>
<!ELEMENT interpret (#PCDATA)>
...
<!ELEMENT phraseological_units (example_source | example |
                phras_info)*>
…
```

2.2.4. Визначення атрибутів

Списки атрибутів елементу визначаються за допомогою ключового слова !ATTLIST. Усередині нього задаються назви атрибутів, типи їх значень і додаткові параметри.

Наприклад, для елементу <article> можуть бути визначені наступні атрибути:

```
<!ATTLIST article
id ID #REQUIRED
about CDATA #IMPLIED
type (actual | review | teach )  'actual' ">
```

У даному прикладі для елементу article визначаються три атрибути: id, about і type, які мають типів ID(ідентифікатор), CDATA і список можливих значень відповідно. Всього існує шість можливих типів значень атрибуту:

- CDATA – вмістом документа можуть бути будь–які символьні дані;
- ID – визначає унікальний ідентифікатор елементу в документі ;
- IDREF(IDREFS) – вказує, що значенням атрибуту повинна виступати назва(або декілька таких назв, розділених пропусками в другому випадку) унікального ідентифікатора визначеного в цьому документі елементу;
- ENTITY(ENTITIES) – значення атрибуту має бути назвою(або списком назв, якщо використовується ENTITIES) компонента (макровизначення), визначеного в документі;
- NMTOKEN (NMTOKENS) – вмістом елементу може бути лише одне окреме слово(тобто цей параметр є обмеженим варіантом CDATA);
- Список допустимих значень – визначається список значень, які може мати даний атрибут.

Також у визначенні атрибуту можна використовувати наступні параметри:

- #REQUIRED – визначає обов'язковий атрибут, який має бути заданий у всіх елементах даного типа;
- #IMPLIED – атрибут не є обов'язковим;
- #FIXED "значення" – вказує, що атрибут повинен мати лише вказане значення, проте само визначення атрибуту не є обов'язковим, але в

процесі розбору його значення в будь–якому разі буде передано програмі–аналізатору;

- Значення – задає параметри атрибуту за умовчанням.

2.2.5. Визначення компонентів DTD-моделі

Компоненти (entity) є визначеннями, вміст яких може бути повторно використаний в документі . У інших мовах програмування подібні елементи називаються макровизначеннями. Створюються DTD–компоненти за допомогою інструкції !ENTITY:

<!ENTITY hello ' Ми раді вітати Вас!' >

Програма–аналізатор, переглядаючи в першу чергу вміст області DTD–визначень, обробить цю інструкцію і при подальшому розборі документа використовуватиме вміст DTD–компонента в тому місці, де зустрічатиметься його назва. Тобто тепер в документі ми можемо використовувати вираження &hello;, яке буде замінено на рядок "Ми раді вітати Вас".

У загальному випадку, усередині DTD можна задати три типи макровизначень.

Внутрішні макровизначення – призначені для визначення строкової константи. З їх допомогою можна організовувати заслання на часто змінну інформацію, роблячи документ більш читабельним. Внутрішні компоненти включаються в документ за допомогою амперсанта &.

У XML існує п'ять передвстановлених внутрішніх символьних констант:

- < – символ "<";
- > – символ ">";
- & – символ "&";
- ' – символ апострофа "'";
- " – символ подвійної лапки """.

Зовнішні макровизначення – вказують на вміст зовнішнього файлу, причому цим вмістом можуть бути як текстові, так і двійкові дані. У першому випадку в місці використання макросу будуть вставлені текстові рядки, в другому – бінарні дані, які аналізатором не розглядаються і використовуються зовнішніми програмами.

<!ENTITY logotype SYSTEM "/image.gif" NDATA GIF87A>

Макровизначення параметрів можуть використовуватися лише усередині області DTD-моделі і позначаються спеціальним символом %, що вставляється перед назвою макросу. При цьому вміст компонента буде поміщений безпосередньо в текст DTD–правила.

2.2.6. Типізація даних

Досить часто при створенні XML-елементу розробникові потрібно визначити, дані якого типа можуть використовуватися як його вміст. Тобто якщо ми визначаємо елемент <last-modified>10.10.98</last-modified>, то хочемо бути упевненими, що в документі в цьому місці знаходитиметься рядок, що є датою, а не числом або довільною послідовністю символів. Використовуючи типізацію даних, можна створювати елементи, значення яких можуть використовуватися, наприклад, як параметри SQL-запитів. Програма клієнт в цьому випадку повинна знати, до якого типа даних відноситься поточне значення елементу і в разі відповідності формує SQL-запрос [17].

Якщо як програма на стороні клієнта використовує XML-процессор, то інформацію про тип можна передавати за допомогою спеціально створеного для цього атрибуту елементу, що має відповідне визначення в DTD-моделі. В процесі розбору програма-аналізатор передасть значення цього атрибуту клієнтському застосуванню, яке зможе використовувати цю інформацію належним чином. Наприклад, аби вказати, що вміст елементу має бути довгим цілим, можна використовувати наступне визначення DTD:

```
<!ELEMENT counter (PCDATA)>
<!ATTLIST counter data_long CDATA #FIXED "LONG">
```

Задавши атрибуту значення за умовчанням LONG і визначивши його як FIXED, ми дозволили тим самим програмі-клієнтові отримати необхідну інформацію про тип вмісту даного елементу, і тепер вона може самостійно визначити відповідність типа цього вмісту вказаному в DTD визначенні.

У кінці хотілося б відзначити, що DTD надає нам вельми зручний механізм здійснення контролю за вмістом документа. На сьогоднішній день, практично всі програми перегляду документів в мережі інтернет використовують DTD-правила. Проте, це далеко не єдиний спосіб перевірки коректності документа. Зараз в W3 консорціумі знаходиться на розгляді новий стандарт мови опису структури документів, званий схемами даних.

2.3. Схеми даних. XML-схеми

2.3.1. Основи використання XML-схем

XML-схема володіє потужнішими можливостями, ніж DTD-модель. Для ілюстрації переваг використання механізму XML-схем по-перше у трьох прикладах порівнюються різні способи представлення елементів. У прикладі №1 представлена витримка з XML-документу. У прикладі №2 показано два елементи, оголошені в синтаксисі DTD-моделі, а в прикладі 3 представлений синтаксис, відповідний XML-схемі. Синтаксис в прикладі №3 подібний до синтаксису XML. При використанні схеми, валідуючий парсер може виконати перевірку, чи є елемент InvoiceNo позитивним цілим числом, і чи полягає PRODUCTID із заданого набору символів (шести цифр і однієї букви від A до Z). Парсер, який обробляє DTD-визначення, може лише підтвердити, що дані елементи є рядками.

Приклад №1: Фрагмент XML-документу

```
<InvoiceNo>123456789</InvoiceNo>
<ProductID>J123456</ProductID>
```

Приклад №2: Фрагмент DTD-моделі, що описує елементи з прикладу №1

```
<!ELEMENT InvoiceNo (#PCDATA)>
<!ELEMENT ProductID (#PCDATA)>
```

Приклад 3: Фрагмент XML-схеми, що описує елементи з прикладу №1

```
<element name='InvoiceNo' type='positive-integer'/>
<element name='ProductID' type='ProductCode'/>
<simpleType name='ProductCode' base='string'>
<pattern value='[A-Z]{1}d{6}'/>
</simpleType>
```

2.3.2. Використання просторів імен в XML-схемі

При спільній роботі одна сторона може обробляти документи інших сторін, і різні сторони можуть представляти свої елементи даних по-різному. Більш того, в окремому документі потрібно незалежно один від одного посилатися на елементи з однаковим ім'ям, створені різними сторонами. Використання XML-схем дозволяє розрізняти визначення з одним і тим же ім'ям за допомогою визначення різних просторів імен.

Така XML-схема визначає набір нових імен, таких як імена елементів, типів, атрибутів, груп атрибутів, чиї визначення і оголошення описані в схемі. У прикладі №3 імена визначаються як InvoiceNo, PRODUCTID і ProductCode.

Імена, визначені в схемі належать так званому цільовому простору імен. Само по собі простір імен є фіксованим, довільним ім'ям, яке повинне відповідати синтаксису URL. Наприклад, простір імен для схеми, представленої в прикладі №3, можна задати таким чином: http://www.snu.edu.ua/Account.

Синтаксис оголошення простору імен інколи може збити з пантелику. Оголошення починається з http://, проте воно не посилається на файл з описом схеми. Насправді, заслання http://www.snu.edu.ua/Account взагалі не веде ні на один файл, а лише на призначене ім'я.

Визначення і оголошення в схемі можуть посилатися на імена, які можуть належати іншим просторам імен. У кожній схемі може бути визначений один цільовий простір імен і можлива безліч вихідних просторів імен. Взагалі, кожне ім'я в заданій схемі належить деякому простору імен. Імена простору імен можуть бути досить довгими, проте їх можна скоротити за допомогою синтаксису оголошення xmlns в документі XML-схеми. Всі ці концепції проілюстровані в прикладі №4:

```
<!--XML Schema fragment in file schema1.xsd-->
<xsd:schema targetNamespace=' http://www.snu.edu.ua/Account '
    xmlns:xsd='http://www.w3.org/1999/XMLSchema'
    xmlns:ACC= 'http://www.snu.edu.ua/Account '>
<xsd:element name='InvoiceNo' type='xsd:positive-integer'/>
<xsd:element name='ProductID' type='ACC:ProductCode'/>
<xsd:simpleType name='ProductCode' base='xsd:string'>
  <xsd:pattern value='[A-Z]{1}d{6}'/>
</xsd:simpleType>
```

У XML-схемі, представленій у прикладі №4, простором імен targetNamespace є http://www.snu.edu.ua/Account, воно містить імена InvoiceNo, PRODUCTID і ProductCode. Імена schema, element, simpleType, pattern, string і positive-integer належать вихідному простору імен http://www.w3.org/1999/XMLSchema, яке скорочується як xsd шляхом оголошення xmlns. У псевдонімі xsd немає нічого особливого, можна вибрати і інше ім'я. У прикладі targetNamespace є також одним з вихідних просторів імен, оскільки ім'я ProductCode використовується у визначенні інших імен.

У фрагменті схеми з прикладу №4 не потрібно вказувати розташування вихідних файлів схеми. Для загальної "схеми схем" http://www.w3.org/1999/XMLSchema вказувати дорогу не потрібно, оскільки він добре відомий. Також не потрібно вказувати місце розташування

вихідного простору імен http://www.snu.edu.ua/Account, оскільки воно виступає тут також цільовим простором імен, визначеним в даному файлі.

XML-документ може посилатися на імена елементів з безлічі просторів імен, визначених в безлічі схем. Для звернення до просторів імен використовуються оголошення xmlns. Для завдання розташування файлів використовується атрибут schemaLocation з простору імен XML Schema. Даний атрибут відрізняється від атрибуту schemaLocation простору імен xsd з попередніх прикладів.

Не дивлячись на те, що DTD-моделі служать розробникам SGML і HTML як механізм опису структурованої інформації ось вже впродовж 20-ти років, DTD володіють деякими обмеженнями в порівнянні з XML-схемами.

Згідно DTD-моделі елемент може бути представлений одним з трьох способів:

- текстовий рядок;
- текстовий рядок, змішаний з іншим дочірнім елементом;
- набір дочірніх елементів.

DTD-модель не володіє синтаксисом XML і пропонує лише обмежену підтримку для типів і просторів імен.

2.3.3. Визначення елементів

Визначенням елементу полягає у визначенні його імені і моделі контента. У XML-схемі модель контента елементу визначається його типом. Отже, елементи в XML-документі можуть мати лише значення, які личать типам, визначеним в його схемі.

Специфікація XML-схеми визначає декілька простих типів для значень.

Тип елементу може бути простим або комплексним (складним). Елемент простого типа не може містити інші елементи або атрибути. Комплексний тип може створювати ефект вбудовування елементів в інші елементи або може асоціювати атрибути з елементом. До цього моменту використовувалися лише приклади з простими типами, визначеними

користувачем. У специфікацію XML-схем також включені зумовлені прості типи. Зумовлений простий тип обмежує значення за їх базовим типом. Наприклад, значенням зумовленого простого типа ProductCode є підмножина значень базового типа string.

Елемент, який не містить атрибутів або інших елементів може бути віднесений до простого типа, зумовленого або визначеного користувачем, такому як string, integer, decimal, time, ProductCode і тому подібне. Приклад №5:

```
<element name='age' type='integer'/>
<element name='price' type='decimal'/>
```

Тепер спробуємо додати до простого елементу price з прикладу №5 атрибут currency. Ми не можемо цього зробити, оскільки елемент простого типа не може мати атрибутів. Якщо треба додати атрибут, тоді необхідно визначити price як елемент комплексного типа. У прикладі №6, визначаємо, так званий анонімний тип, в якому комплексному типові не дається явного імені. Іншими словами, атрибут name елементу complexType не визначений. Приклад №6:

```
<element name='price'>
<complexType base='decimal' derivedBy='extension'>
<attribute name='currency' type='string'/>
</complexType>
</element>
```

У XML-документі в елемент можуть бути вкладені інші елементи. Ця вимога виражається безпосередньо в DTD-моделі. XML-схема замість цього визначає елемент і його тип, який може включати оголошення інших елементів і атрибутів. Приклад наведений в прикладі №7 – порівняння комплексних типів даних у DTD-моделі і XML-схемі:

XML-документ:

```
<Book>
```

```
  <Title>Cool XML<Title>
  <Author>Cool Guy</Author>
</Book>
```

DTD-модель:

```
<!ELEMENT Book (Title, Author)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
```

XML-схема:

```
<element name='Book' type='BookType'/>
<complexType name='BookType'>
   <element name='Title' type='string'/>
   <element name='Author' type='string'/>
</complexType>
```

Не дивлячись на те, що XML-код у прикладі №7 відповідає обом фрагментам DTD-моделі і XML-схеми, між ними існує велика різниця. У DTD-моделі всі елементи є глобальними, тоді як XML-схемі в таблиці дозволяє визначити Title і Author локально, для появи їх лише в рамках елементу Book. Для точного повторення ефекту оголошень у DTD-моделі в XML-схемі, елементи Title і Author повинні мати глобальну зону дії, як це показано в прикладі.

2.3.4. Вираження складних обмежень для елементів

XML-схема пропонує велику гнучкість, чим DTD-модель при вираженні обмежень для моделі контента елементів. На простому рівні, такому як в DTD, ми можемо асоціювати з елементом атрибути, а також вказати, що в них може з'являтися послідовність з лише одного, нуля або більш (*), або одного або більш (+) за елементи із заданого набору елементів. У XML-схемі можна виразити додаткові обмеження, використовуючи для цієї мети, наприклад, атрибути minOccurs і maxOccurs для елементу element і елементи choice, group і all.

Приклад №8: Вираження обмежень для типів елементів

```
<element name='Title' type='string'/>
```

```
<element name='Author' type='string'/>
<element name='Book'>
  <complexType>
    <element ref='Title'/>
    <element ref='Author'/>
  </complexType>
</element>
```

У прикладі №8 тег Title є опціональним по відношенню до тегу Book (таке ж правило можна задати і в DTD-моделі). Проте тут також говориться, що в елементі Book має бути хоч би один і не більше двох елементів Author. Значенням атрибутів minOccurs і maxOccurs тега element за умовчанням є 1. Елемент choice вказує на те, що може з'явитися лише один з вказаних дочірніх елементів. Інший елемент all визначає, що всі дочірні елементи можуть з'являтися лише один раз, разом і у будь-якому порядку, або не з'являтися зовсім.

2.3.5. Підведення підсумків

За допомогою простих прикладів були розкриті найбільш фундаментальні концепції, необхідні для визначення структури елементів за допомогою XML-схем. Був зроблений порівняльний аналіз DTD-моделі й XML-схем. Доступно також безліч інших потужних механізмів:

- XML-схеми містять усесторонню підтримку для спадкоємства типів, дозволяючи повторно використовувати визначені раніше структури. Таке використання називають аспектами;
- декілька механізмів, що дозволяють контролювати загальне визначення підтипу або замінювати його в певному документі;
- окрім використання підтипів, можна визначати еквівалентні типи, тобто значення одного типа може бути заміщене значенням іншого;
- XML-схеми забезпечують механізм для заміщення елементу або типа шляхом оголошення їх як абстрактних;

- для більшої зручності можна позначити і задати імена групам атрибутів або елементів. Це дозволяє повторно використовувати їх при подальших зверненнях;
- XML-схеми надають три елементи – appInfo, documentation і annotation – для використання коментарів, як користувачам (documentation) так і застосуваннями (appInfo).

2.4. Введення в DOM

2.4.1. Програмні інтерфейси

Для спрощення написання Java-программ, які обробляють XML, було створено багато програмних інтерфейсів. Це інтерфейси були визначені компаніями, організаціями по стандартизації і групами користувачів, аби забезпечити потреби XML-програмістів. Наступні інтерфейси:

- Document Object Model (DOM);
- Simple API for XML (SAX);
- JDOM;
- Java API for XML Processing (JAXP).

Перші три з цих інтерфейсів (DOM, SAX і JDOM) визначають, як представляється вміст XML-документу і як відбувається доступ до нього. JAXP містить класи для створення об'єктів парсерів. Аби створити парсер DOM або SAX, ми використовуємо JAXP. Коли ми використовуємо JDOM, бібліотека JDOM використовує "під килимом" JAXP для створення парсера [10]. У результаті:

- ми використовуємо DOM, SAX або JDOM, аби працювати з вмістом XML-документу;
- якщо ми використовуємо DOM або SAX, ми використовуємо JAXP для створення парсера;
- якщо ми використовуємо JDOM, бібліотека створює для нас парсер.

2.4.2. Основи парсерів

XML-парсер – це частина коду, яка читає XML-документ і аналізує його структуру. Парсери класифікуються декількома різними способами:

- перевіряючи або неперевіряючи парсери;
- парсери, які підтримують XML-схеми;
- парсери, які підтримують Document Object Model (DOM);
- парсери, які підтримують Simple API for XML (SAX).

Є три різні види XML-документів:

- правильно форматовані документи: вони слідують основним правилам XML (атрибути мають бути в лапках, теги мають бути коректно вкладені і так далі).
- правильні документи: це правильно форматовані документи, які також задовольняють ряду правил, заданих у визначенні типа документа DTD-моделі або в схемі XML;
- неправильні документи: будь-які інші.

Наприклад, якщо ми маємо XML-документ, який слідує всім базовим правилам XML, це правильно форматований документ. Якщо цей документ також слідує всім правилам, які ми визначили для документів нашої програми, він також правильний. Якщо XML-парсер знаходить, що XML-документ не є правильно форматованим, специфікація XML вимагає, аби парсер видавав фатальну помилку. Перевірка правильності, проте, – інший випадок. Перевіряючи парсери перевіряють XML-документи при їх розборі, а неперевіряючи парсери – ні. Іншими словами, якщо XML-документ є правильно форматованим, неперевіряючий парсер не дивиться, чи слідує цей документ правилам, визначеним в DTD-моделі або XML-схемі і навіть взагалі чи є для цього документа такі правила (у більшості перевіряючих парсерів перевірка правильності за умовчанням вимкнена).

Так чому ж використовуються неперевіряючи парсери? Дві причини: швидкість і ефективність. Від XML – парсеру потрібні деякі витрати на читання DTD-моделі або XML-схеми, а потім установка движка перевірки

правил, який обробляє кожен елемент і атрибут в XML-документі відповідно до правил. Якщо ми упевнені, що XML-документ правильний (наприклад, якщо він генерується із запиту до бази даних), ми можемо пропустити перевірку правильності. Залежно від складності правил для документа, це може заощадити значний об'єм часу і пам'яті.

Якщо ж у нас крихкий код, який отримує вхідні дані з XML-документів, і цей код вимагає, аби документи слідували певній DTD-моделі або XML-схемі, нам, можливо, слід перевіряти все, незалежно від того, як дорого або довго це буде.

Вихідна специфікація XML визначила визначення типа документа (DTD) як спосіб визначення того, як повинен виглядати правильний XML-документ. Велика частина синтаксису DTD-моделі відбувалася з SGML і розглядала перевірку правильності з точки зору публікації, а не з точки зору типізації даних. Крім того, DTD-моделі мають синтаксис, відмінний від XML-документів, що робить важким для користувачів розуміння синтаксису правил документа.

Аби здолати ці обмеження, Worldwide Web Consortium (W3C) створив мову схеми XML. Схема XML дозволяє нам визначити, як виглядає правильний документ, з набагато більшою точністю. Мова схеми XML дуже багата, а документи схеми XML Schema можуть бути дуже складними. З цієї причини XML-парсери, які підтримують перевірку правильності за допомогою схеми XML мають тенденцію бути дуже великими. Мова схеми XML W3C не є єдиним гравцем на цьому полі. Деякі парсери підтримують інші мови схеми XML, такі як RELAX NG або Schematron.

2.4.3. Об'єктна модель документа (DOM)

Об'єктна модель документа або DOM є офіційною рекомендацією W3C. Вона визначає доступний для програм інтерфейс доступу і зміни структури XML-документів. Коли XML-парсер заявляє, що він підтримує DOM, це означає, що він реалізує інтерфейси, визначений у стандарті.

Коли ми розбираємо XML-документ за допомогою парсера DOM, ми отримуємо структуру дерева, яка представляє вміст XML-документа. Весь текст, елементи і атрибути (разом з іншими речами) знаходяться в структурі дерева. DOM також надає різні функції, які ми можемо використовувати для дослідження вмісту, структури дерева і маніпулювання ними.

Парсер не відстежує деяких речей. Наприклад, кількості пропусків між атрибутами і їх значення не зберігаються в дереві DOM.

DOM не дає нам жодного способу дізнатися, як був кодований вихідний документ. Інший приклад – XML-парсер не повідомить нас, чи був вихідний елемент закодований одним тегом або двома (наприклад, чи була горизонтальна лінія XHTML закодована як <hr/> або як <hr></hr>?). Інформація, яку повинен відстежувати XML-парсер, називається XML Infoset.

DOM – це інтерфейс для роботи із структурою XML-документів. DOM проект було розроблено для надання ряду об'єктів і методів, покликаних полегшити життя програмістів. У ідеалі ми можемо написати програму, яка використовує один DOM-підтримуючий парсер для обробки XML-документу, а потім перемкнутися на іншій – підтримуючий парсер без зміни нашого коду.

Коли ми розбираємо XML-документ за допомогою парсер DOM, ми отримуємо ієрархічну структуру даних (дерево DOM), яка представляє все, що парсер знайшов в XML-документі. Ми можемо потім використовувати функції DOM, аби маніпулювати деревом, переміщати гілки, додавати нові гілки, видаляти частини дерева.

З точки зору мови Java, Node – це інтерфейс. Node є основним типом даних DOM; все в дереві DOM є Node того або іншого типа. DOM Рівня 1 також визначає декілька підінтерфейсів інтерфейсу Node:

- Element: представляє елемент XML у вихідному документі;
- Attr: представляє атрибут елементу XML;

- Text: вміст елементу. Це означає, що елемент з текстом містить дочірній текстовий вузол; текст елементу не є властивістю самого елементу;
- Document: представляє весь XML-документ. Лише один об'єкт Document існує для кожного XML-документу, який ми розбираємо. Маючи об'єкт Document, ви можемо знайти корінь дерева DOM; від кореня ми можемо використовувати функції DOM для читання дерева і маніпулювання ним.

Додатковими типами вузлів є: Comment, який представляє коментар в XML-файлі; ProcessingInstruction, який представляє інструкцію обробки; і CDATASection, який представляє розділ CDATA. Можливо, нам не знадобляться ці типи вузлів, але якщо вони нам знадобляться, вони є.

Коли ми працюємо з DOM, ми часто використовуємо наступні методи:

- Document.getDocumentElement(): повертає корінь дерева DOM (ця функція є методом інтерфейсу Document; він не визначений для інших підтипів Node);
- Node.getFirstChild() і Node.getLastChild(): повертають першого або останнього нащадка даного Node;
- Node.getNextSibling() і Node.getPreviousSibling(): повертають наступного або попереднього "брата" даного Node;
- Element.getAttribute(String attrName): для даного Element повертає значення атрибуту з ім'ям attrName. Якщо нам потрібне значення атрибуту "id", використовуємо Element.getAttribute("id"). Якщо цей атрибут не існує, метод повертає порожній рядок ("").

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ДЛЯ АВТОМАТИЗАЦІЇ ТЕРМІНОГРАФІЧНИХ РОБІТ

3.1. Вибір середовища розробки

Для реалізації поставленого завдання була використана мова програмування – Java. Розглянемо основні особливості цього середовища розробки.

Java є потужною об'єктно-орієнтованою мовою. Її головний принцип (пишемо один раз – використовуємо скрізь) знімає залежність від конкретної платформи, а його динамічні можливості забезпечать нам максимальну гнучкість.

Незалежність від платформ. Компілятор Java (the Java Compiler) генерує незалежний двійковий код, який коректно виконується на будь-якій машині, де є так звана віртуальна Java-машина. Прийняття платформи Java всіма провідними виробниками операційних систем і комп'ютерної техніки дозволяє нам писати програми мовою Java і використовувати їх практично скрізь. Таким чином, один і той же код можна запускати під управлінням операційних систем Windows, Linux, FREEBSD, Solaris, Apple Mac та інші. Це стає дуже важливим, коли програми завантажуються за допомогою глобальної мережі Інтернет і використовуються на різних платформах.

Простота підтримки. Програми Java можна легко модернізувати, що дозволяє швидко упроваджувати нові версії програмних продуктів, що раз у раз з'являються. Нові застосування і їх нові версії можна завантажити з будь-якої крапки, підключеної до мережі, і використовувати "на ходу", тобто без перекомпіляції. Java звільняє нас від необхідності створення, а користувачів від необхідності установки нових версій програмного забезпечення, оскільки нові функції можна легко вбудовувати у вже наявні програми в міру необхідності.

Повнота. Платформа Java має всю "будівельну цеглу", необхідну для створення Java-програм і аплетів.

Продуктивність. Java налаштована на високу продуктивність. Для ще більшого підвищення продуктивності такі компанії, як Microsoft, Sun, Hewlett-Packard та інші розробляють компілятори Just-In-Time для трансляції критично важливих частин програми в машинні коди в режимі реального часу.

Глобальні можливості. Використовуючи стандарт Unicode 2.0, JDK 1.1 підтримує багатий набір міжнародних інтерфейсів прикладного програмування (API), що ще більш спрощує розробку аплетів і застосувань для глобального використання. JDK 1.1 дозволяє створювати аплети, які автоматично набудовуватимуться і працюватимуть на тій мові, яка вважатиме за краще користувач.

Безпека. Архітектура Java підтримує самі передові функції безпеки, такі як контроль друку, арифметика з відсутністю покажчиків, автоматичне "прибирання сміття", незмінні послідовності символів, перевірка часу компіляції і граничних значень, а також двійковий код і розділення імен в підключеному режимі. JDK Release 1.1 ще більш підсилює ці можливості за рахунок підтримки таких функцій безпеки, як "підписані" аплети, буферизація повідомлень, контроль доступу і управління загальними і приватними ключами кодування даних. В результаті можна легко і просто включати функції безпеки високого і низького рівня в застосування і аплети Java.

Крім того, Java – повністю об'єктно-орієнтована мова, навіть більшою мірою, чим C++. Все в мові Java є об'єктами, за винятком не багатьох основних типів (primitive types), наприклад чисел.

Інтерфейси прикладного програмування Java надають технології доступу до інформаційних ресурсів, у тому числі до баз даних і традиційних застосувань. За допомогою цих інтерфейсів можна створювати розподілені незалежні застосування і аплети для середовища клієнт/сервер з

використанням потужних мережевих можливостей Java. Інтерфейс дає практично все, що можна отримати від множинного спадкоємства, уникаючи при цьому складнощів, які виникають при управлінні ієрархіями класів.

3.2. Базовий XML-файл

3.2.1. Основні частини XML-файлу

Основними частинами XML-файлу є:

- XML-оголошення: основне оголошення <?xml version"1.0"?> визначає цей файл, як XML-документ. Не є загальноприйнятим завдання кодування в оголошенні. Тут не має значення, яку мову або кодування використовує XML-файл, парсер в змозі читати його правильно, поки він розуміє дане кодування;
- оголошення DOCTYPE: XML є зручним способом для обміну даними між людьми і машинами, але аби обмін працював гладко, необхідний загальний словник. Необов'язкове оголошення DOCTYPE може бути використане для вказівки документа – в даному випадку, termin_diction.dtd – з яким файл може бути порівняний, аби переконатися, що в нім немає зайвої або бракуючої інформації (наприклад, відсутній word-id або помилкове ім'я елементу). Документи, оброблені таким чином, називаються правильними (valid) документами. Успішна перевірка правильності не є обов'язковою для XML;
- самі дані: дані в XML-документі повинні міститися усередині єдиного кореневого елементу, такого, як елемент terminology_dictionary. Аби XML-документ міг оброблятися, він має бути правильно форматованим (well-formed).

У DOM робота з XML-інформацією означає розбиття її спочатку по вузлах.

3.2.2. Базові типи вузлів: документ, елемент, атрибут і текст

Найбільш поширеними типами вузлів в XML є:

Елементи: елементи є базовими будівельними блоками XML. Зазвичай елементи мають нащадків, якими є інші елементи, текстові вузли або їх комбінація. Елементні вузли також є єдиним типом вузлів, що мають атрибути:

```
<interpretation>
<phraseological_units>
```

Атрибути. Вузли атрибутів містять інформацію про елементний вузол, але не розглядаються як нащадки елементу, як в:

```
<interlang language="українська">
<interlang language="російська">
<word id="3365">
```

Текст. Текстовий вузол – це саме текст. Він може містити інформацію або лише пропуск:

```
<leks_login>РОЗРОБКА</leks_login>
<log_info> 1.</log_info>
<interpret>Дія за знач. розробити, розробляти.</interpret>
<example>Відкрита розробка – метод добування корисних
        копалин, за якого виймання розкривних порід і
        корисних копалин відбувається у відкритих просторах
        на земній поверхні.
</example>
<example_source/>
```

Документ. Вузол документа є загальним предком для всіх інших вузлів в документі:

```
<terminology_dictionary>
```

Менш поширені типи вузлів: CDATA, коментарі, інструкції обробки і фрагменти документу. Інші типи вузлів використовуються не так часто, але все одно важливі в деяких ситуаціях. До їх числа входять:

- CDATA: скорочення від Character Data (символьні дані);

- коментарі:
- інструкції обробки;
- фрагменти документа.

CDATA – це вузол, що містить інформацію, яка не повинна аналізуватися парсером. Замість цього вона повинна просто передаватися як звичайний текст. Наприклад, в ній може бути записаний HTML для спеціальних цілей. При нормальних обставинах процесор повинен намагатися створити елементи з кожного записаного тега, який навіть може не бути правильне форматованим. Ці проблеми можуть бути обійдені вживанням секцій CDATA. Ці секції записуються в спеціальній нотації:

```
<[CDATA[
<b>Important: Please keep head and hands inside ride at <i>all
    times</i>.</b>
]]>
```

Коментарі включають інформацію про дані і зазвичай ігноруються застосуванням. Вони записуються як:

```
<!-- This is a comment. -->
```

Інструкції обробки – це інформація, спеціально адресована застосуванню. Деякими прикладами є коди, які мають бути виконані, або інформація про те, де знайти таблицю стилів. Наприклад:

```
<? xml-stylesheet type="text/xsl" href="tyle.xsl"?>
```

Аби бути правильно форматованим, документ повинен мати лише один кореневий елемент. Інколи при роботі з XML повинні тимчасово створюватися групи елементів, для яких немає необхідності задовольняти цій вимозі.

Інші типи вузлів включають сутності, вузли заслань на сутності і нотації. Одним із способів додаткової організації XML-даних є вживання просторів імен.

3.2.3. Опис структури словника

Опис структури словника в XML-документі представляється визначенням тегів маркування і правилами їх вкладеності. У XML елементи є лише контейнерами для зберігання даних, а в кожному XML-документі є один головний елемент, який містить в собі всі дані цього документа. Покажемо це на прикладі нашого програмного засобу:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE terminology_dictionary SYSTEM "termin_diction.dtd">
<terminology_dictionary>
…
<word id="14365">
<interlang language="українська">
<leks_login>ПРОГРАМА</leks_login>
<stylist_moment> -и, ж.</stylist_moment>
<gram_info/>
<interpretation>
<log_info> 1.</log_info>
<interpret>Наперед продуманий план якої-небудь діяльності, роботи і т.
        ін..</interpret>
<example>Екологічна програма – порядок контролю за станом
        довкілля..</example>
 …
<interlang language="російська">
<leks_login>ПРОГРАММА</leks_login>
<stylist_moment> -ы, ж.</stylist_moment>
<gram_info/>
<interpretation>
<log_info> 1.</log_info>
<interpret>Содержание и план предстоящей деятельности, работ и т.
        п.</interpret>
<example>Наша страна осуществляет грандиозную программу подъема
        материального благосостояния народа.</example>
<example_source>Катаев, Счастье нашей молодежи.</example_source>
…
<interlang language="англійська">
<leks_login>PROGRAM</leks_login>
<stylist_moment/>
<gram_info/>
<interpretation>
```

<log_info> 1.</log_info>
<interpret>A verbal or written request for assistance or employment or admission to a school.</interpret>
<example>December 31 is the deadline for applications.</example>…
</word>
…
<word id="14366">
…
</terminology_dictionary>

Елемент <terminology_dictionary>...</terminology_dictionary> містить в собі весь вміст даного термінологічного словника. Всі інші елементи, наприклад, <word id="14365">.</word>, є субелементами, вкладеними в елемент <terminology_dictionary>. <!DOCTYPE terminology_dictionary SYSTEM "termin_diction.dtd"> посилається на зовнішній файл, що містить визначення, що діють, для перевірки правильності документа. Всі разом вони створюють структуру електронного словника [10].

Лексикографічна інтерпретація словникової статті представляється у вигляді вузла з ідентифікатором відповідного атрибуту для мови – <Interlang language="українська">, <Interlang language="російська">, <Interlang language="англійська">, або запис інтерпретації на будь-якій іншій мові.

3.3. DOM як структура

3.3.1. Об'єкт DOM Document

Об'єкт DOM Document є колекцією вузлів або порцією інформації, організованою в ієрархію. Ця ієрархія дозволяє рухатися по дереву у пошуках потрібної інформації. Аналіз структури зазвичай вимагає, аби був завантажений повний документ, і ієрархія була побудована до початку роботи. Оскільки DOM грунтується на ієрархії інформації, про неї говорять, що вона деревовидно-базована або об'єктно-базована.

Для великих документів розбір і завантаження повного документа може бути повільним і ресурсоємним, так що для роботи з такими даними

можуть виявитися кращими інші засоби. Подієво-базовані моделі, такі, як Simple API for XML (SAX), працюють на потоці даних, обробляючи його у міру вступу. Подієво-базований API обходить необхідність побудови дерева в пам'яті, але фактично не дозволяє змінювати дані у вихідному документі.

З іншого боку, DOM також забезпечує API, який дозволяє додавати або видаляти вузли в будь-якій точці дерева в належним чином створеному застосуванні.

Парсер – це програмне застосування, яке призначене для того, щоб аналізувати документ, – в моєму випадку XML-файл – і робити щось визначене з його інформацією. У подієво-базованому API, такому, як SAX, парсер події деякому слухачеві. У деревовидно-базованому API, такому, як DOM, парсер будує в пам'яті дерево даних.

3.3.2. DOM API

Починаючи з DOM Рівня 1, DOM API містить інтерфейси, які представляють різні типи інформації, які можуть бути знайдені в XML-документі, такі, як елементи і текст. Він також включає методи і властивості, необхідні для роботи з цими об'єктами.

Рівень 1 включає підтримку XML 1.0 і HTML, в якій кожен елемент HTML представляється як інтерфейс. Він включає методи для додавання, редагування, переміщення і читання інформації, що міститься у вузлах і так далі. Він, проте, не включає підтримку просторів імен XML, які забезпечують можливість сегментувати інформацію усередині документа.

Підтримка просторів була додана в DOM Рівня 2. Рівень 2 розширює Рівень 1, дозволяючи розробникам виявляти і використовувати інформацію просторів імен, яка може бути застосовна до вузла. Рівень 2 додає також декілька модулів підтримки таблиць стилів, Cascading Style Sheets (CSS), подій і розширених маніпуляцій з деревом.

DOM Рівня 3 включає покращену підтримку об'єкту Document (попередні версії залишали це на розсуд застосувань, що робило скрутним

створення родових застосувань), розширену підтримку просторів імен, і нові модулі для завантаження і збереження документів, перевірки правильності і XPath, засобу для вибору вузлів, використовувані в XSL Transformations і інших технологіях XML.

3.3.3. Установки парсера

Одна з переваг створення парсерів за допомогою DocumentBuilder полягає в управлінні різними установками парсера, що створюється за допомогою DocumentBuilderFactory. Наприклад, парсер може бути встановлений на перевірку правильності документа:

```
try {
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  dbf.setValidating(true);
  DocumentBuilder db = dbf.newDocumentBuilder();
  doc = db.parse(docFile);
} catch (Exception e) {
...
```

Java-реалізація DOM Рівня 2 забезпечує управління параметрами парсера через наступні методи:

- setCoalescing(): визначає, чи перетворює парсер вузли CDATA на текст і чи сполучає їх з довколишніми текстовими вузлами (якщо можливо). Значення за умовчанням – false;
- setExpandEntityReferences(): визначає, чи розширюються зовнішні заслання на сутності. Якщо true, зовнішні дані вставляються в документ. Значення за умовчанням – true;
- setIgnoringComments(): визначає, чи ігноруються коментарі у файлі. Значення за умовчанням – false;
- setIgnoringElementContentWhitespace(): визначає, чи ігноруються пропуски між елементами (аналогічно тому, як браузер інтерпретує HTML). Значення за умовчанням – false;

- setNamespaceAware(): визначає, чи звертає парсер увагу на інформацію простору імен. Значення за умовчанням – false;
- setValidating(): за умовчанням парсер не перевіряє правильність документів. Встановлюємо тут true для перевірки правильності нашого XML-документу.

Результат, DomOne, що генерується, включає багато пропусків, включаючи пропуски між елементами. У багатьох випадках можна не звертати увагу на ці пропуски. Наприклад, переклади рядків і пропуски використовувані для відступів для всіх елементів <interpretation>, не мають значення, якщо все, що ми хочемо, – отримати текст рядків.

Дерево DOM заповнюється об'єктами так, що всі вузли пропусків є насправді об'єктами Java. Ці об'єкти займають пам'ять, займають час обробки для створення і знищення, і код повинен знаходити їх і ігнорувати їх. Отже, якщо ми вказуємо парсеру не створювати ці об'єкти взагалі, це збереже час і пам'ять, що завжди добре.

На щастя, DocumentBuilderFactory надає метод для ігнорування не значимих пропусків. Метод setIgnoringElementContentWhitespace(boolean), як випливає з його довгої назви, залишає пропуски за бортом.

3.3.4. Створення об'єкту парсера термінологічний словник

Створюється фабричний об'єкт, а потім фабричний об'єкт створює парсер:

```
public void parseAndPrint(String uri)
{
        Document doc = null;
         try
        {
                DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();
. . .
```

Об'єкт DocumentBuilder є парсером. Він реалізує всі методи DOM, які потрібні для розбору і обробки XML-файлу. Він створюється усередині блоку try; багато чого може виявитися неправильним, коли ми створюємо парсер і розбираємо документ. Так що нам потрібно виловлювати будь-які помилки, які можуть статися. Класи DocumentBuilderFactory і DocumentBuilder визначені JAXP.

Тепер, коли створено парсер, просто необхідно доручити йому розібрати файл (або URI):

```
doc = db.parse(uri);
```

Doc є екземпляром інтерфейсу Document. Іншими словами, розбір XML-файлу дає структуру Document, яку можна вивчати, аби визначити, що містить XML-файл.

3.3.5. Виведення вмісту дерева DOM

Оскільки все в дереві DOM є Node(вузлом) того або іншого типа, треба використовувати рекурсивний метод обходу дерева і друку всього, що є в ньому. Стратегія полягатиме у виклику методу для друку вузла. Метод роздруковуватиме вузол, а потім викликати себе ж для кожного з нащадків цього вузла. Якщо цей нащадок має своїх нащадків, метод викликатиме себе також з нього. Ось приклад коду:

```
if (doc != null)
printDomTree (doc);
. . .
public void printDomTree(Node node)
{
 int type = node.getNodeType();
 switch (type)
 {
        // print the document element
        case Node.DOCUMENT_NODE:
        {
         printDomTree(((Document)node).getDocumentElement(),printtxar);
         break;
```

```
}
```

Метод printDomTree приймає аргумент типа Node. Якщо цей Node є вузлом документу, друкується XML-декларація, а потім викликається printDomTree для елементу документа (елементу XML, який містить останню частину документа). Елемент документа буде, як правило, мати нащадків, так що printDomTree викликатиме сам себе також і для кожного нащадка. Цей рекурсивний алгоритм обходить все дерево DOM і друкує його вміст.

От як printDomTree обробляє елементний вузол:

```
// print element and any attributes
case Node.ELEMENT_NODE:
{
        NamedNodeMap attrs = node.getAttributes();
        for (int i = 0; i < attrs.getLength(); i++)
        printDomTree(attrs.item(i),printtxar);
        if (node.hasChildNodes())
        {
                NodeList children = node.getChildNodes();
                for (int i = 0; i < children.getLength(); i++)
                printDomTree(children.item(i),printtxar);
        }
 break;
}
```

Для елементного вузла потрібно віддрукувати ліву кутову дужку і ім'я вузла. Якщо цей елемент має атрибути, викликається printDomTree для кожного атрибуту (технічно, ми можемо використовувати тут для друку атрибутів Attr.getName() і Attr.getValue()).

Коли надруковані всі атрибути, викликається printDomTree знову для кожного дочірнього елементу, що міститься в цьому вузлі. Останнім кроком є друк кінцевого елементу після того, як всі дочірні елементи оброблені.

Обробка атрибутів і текстових вузлів тривіальна. Для атрибуту виводиться пропуск, ім'я атрибуту, знак рівності і відкриваюча лапка, значення атрибуту і закриваюча лапка (це спрощений спосіб; значення

атрибуту може містити апострофи або лапки).

Обробка текстових вузлів найпростіша, просто необхідно виводити текст:

```
 case Node.TEXT_NODE:
{
        String str,str1;
        str = printtxar.getText();
        str1 = str+node.getTextContent();
        printtxar.setText(str1);
        break;
 }
```

3.3.6. Обробка похибок парсером

При різноманітних можливостях створення парсера багато речей можуть бути неправильно зробленими. Як показано, застосування зводить все це в єдине родове Exception, яке не може бути достатнє корисним в сенсі відладки.

Аби краще діагностувати проблеми, ми виловлюємо специфічні виключення, що відносяться до різних аспектів створення і використання парсера.

DOM визначає інтерфейс ErrorHandler; це один з інтерфейсів, DefaultHandler, що реалізовуються. ErrorHandler містить три методи: warning, error і fatalError:

- warning відноситься до попередження, умови, не визначеної, як помилка або фатальна помилка специфікацією XML;
- error, як виявляється з назви, відноситься до події помилки; наприклад, специфікація XML визначає порушення правил перевірки правильності як помилку;
- fatalError також визначений специфікацією XML; значення атрибуту, не поміщене в лапки, має бути фатальною помилкою.

Трьома подіями, визначеними в ErrorHandler є:

1. warning(). Проблема, що відноситься до попереджень, як визначено в специфікації XML.
2. error(). Проблема, що відноситься до того, специфікація XML розглядає, як помилку.
3. fatalError(). Проблема, що відноситься до фатальних помилок, як визначено в специфікації XML.

Попередження і помилки зазвичай відносяться до таких речей, як сутність і DTD. Тоді як фатальні помилки зазвичай відносяться до порушень основного синтаксису XML (наприклад, теги не вкладені коректно). Ось синтаксис трьох обробників помилок:

```
try
{
 DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
 DocumentBuilder db = dbf.newDocumentBuilder();
  ErrorChecker errors = new ErrorChecker(mw);
 db.setErrorHandler(errors);
 doc = db.parse(uri);
…
 public void error (SAXParseException e)
{
   JOptionPane.showMessageDialog(mw, e.getMessage(),
   "При перевірці правильності xml-документу сталася не критична
   помилка. Програма може продовжувати роботу.",
   JOptionPane.ERROR_MESSAGE);
 }

 public void warning (SAXParseException e)
{
  JOptionPane.showMessageDialog(mw, e.getMessage(),
  "При перевірці правильності xml-документу з'явилося зауваження.
  Програма може продовжувати роботу.",
  JOptionPane.ERROR_MESSAGE);
  }
```

```
public void fatalError (SAXParseException e)
{
  JOptionPane.showMessageDialog(mw, e.getMessage(),
  "При перевірці правильності xml-документу сталася критична
  помилка. Програма припинить роботу.",
  JOptionPane.ERROR_MESSAGE);
  System.exit(1);
 }
```

Клас DOMParseException визначає такі методи, як getLineNumber() і getColumnNumber() для надання рядка і стовпця, де сталася помилка. getLocationString лише форматує цю інформацію в рядок; приміщення цієї коди в окремий метод означає, що не треба включати його в кожен обробник помилки.

Коли парсер створив документ, застосування може проходити через нього для обробки даних.

3.4. Контроль за вмістом документа. Опис DTD-моделі

Для того, щоб використовувати DTD в нашому документі, ми описуємо його в зовнішньому файлі і при описі в DTD-файлі просто вказуємо заслання на цей файл:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE terminology_dictionary SYSTEM "termin_diction.dtd">
<terminology_dictionary>
…
```

Атрибут standalone має значення "no", це говорить про те, що використання зовнішніх описів DTD дозволене.

```
<!ELEMENT terminology_dictionary (word)*>
```

Визначає список елементів для кореневого елементу terminology_dictionary. Це оголошення елементу повідомляє аналізатору про те, що кореневий елемент terminology_dictionary містить елементи-нащадки word.

<!ELEMENT interlang
(phraseological_units|interpretation|gram_info|stylist_moment|leks_login)*>

Визначає список елементів для елементу interlang. Це оголошення елементу повідомляє аналізатору про те, що кореневий елемент interlang містить елементи-нащадки phraseological_units, interpretation, gram_info, stylist_moment і leks_login, і що ці елементи повинні з'являтися в документі в цьому ж порядку. Якщо цей порядок буде порушений, то аналізатор видасть повідомлення про помилку.

<!ELEMENT leks_login (#PCDATA)>

Визначає елемент leks_login і вказує, що цей елемент міститиме символьні дані, які підлягають обробці аналізатором.

<!ELEMENT stylist_moment (#PCDATA)>

Визначає елемент stylist_moment і вказує, що цей елемент міститиме символьні дані, які підлягають обробці аналізатором.

<!ELEMENT gram_info (#PCDATA)>

Визначає елемент gram_info і вказує, що цей елемент міститиме символьні дані, які підлягають обробці аналізатором.

<!ELEMENT interpretation (example_source|example|interpret|log_info)*>

Визначає список елементів для елементу interpretation. Це оголошення елементу повідомляє аналізатору про те, що кореневий елемент interpretation містить елементи-нащадки example_source, example, interpret і log_info, і що ці елементи повинні з'являтися в документі в цьому ж порядку. Якщо цей порядок буде порушений, то аналізатор видасть повідомлення про помилку.

Далі відбувається подібний опис тому, як було приведено вище.

Ось приклад атрибуту для елементу word, який точно визначає, що може містити цей елемент:

```
<!ATTLIST word
id CDATA #REQUIRED
>
```

Перший рядок представленого фрагмента коду служить для оголошення елементу. Остання частина — для оголошення наступного атрибуту. У слова може бути тільки один унікальний номер (id), який однозначно визначає його у словнику. #REQUIRED вказує на те, що значення обов'язкове для даного атрибуту.

Лексикографічна інтерпретація словникової статті представляється у вигляді вузла з ідентифікатором відповідного атрибуту для мови – <Interlang language="українська">, <Interlang language="російська">, <Interlang language="англійська">, або запису інтерпретації на будь-якій іншій мові.

Елемент <terminology_dictionary>...</terminology_dictionary> містить в собі весь вміст даного термінологічного словника. Решта елементів, наприклад, <word id="14365">...</word>, є субелементами, вкладеними в елемент <terminology_dictionary>. Всі разом вони створюють структуру словника [10]:

```
<!ELEMENT terminology_dictionary (word)*>
<!ELEMENT word (interlang)*>
<!ATTLIST word id CDATA #REQUIED>
<!ELEMENT interlang ( leks_login | stylist_moment | gram_info |
                interpretation | phraseological_units)*>
<!ATTLIST interlang language CDATA #IMPLIED >
<!ELEMENT leks_login (#PCDATA)>
<!ELEMENT stylist_moment (#PCDATA)>
<!ELEMENT gram_info (#PCDATA)>
<!ELEMENT interpretation (example_source|example|interpret|log_info)*>
<!ELEMENT log_info (#PCDATA)>
<!ELEMENT interpret (#PCDATA)>
...
<!ELEMENT phraseological_units (example_source | example |
                phras_info)*>
```

Як бачимо, DTD надає нам вельми зручний механізм здійснення контролю за вмістом документа.

3.5. Основи інтерфейсу програми

На початку дослідницької роботи була поставлена мета реалізувати програму для автоматизації термінографічних робіт. Результатом є вихідний jar файл програми, який працює в середовищі Windows, Linux, FREEBSD, Solaris, Apple Mac та інші, оскільки Java, як показувалося раніше незалежна від платформи. JAR файл — це Java архів (скорочення від англ. Java Archive). Є звичайним zip-архівом, в якому міститься частина програми на мові Java. JAR файл є виконуваним, тому в ньому міститься файл MANIFEST.MF у каталозі META-INF. Розглянемо принципи роботи додатка.

Так, як програма в основному призначена для роботи в різних середовищах, то необхідно було в першу чергу реалізувати простий й у той же час приємний для зору інтерфейс. Рішенням цього завдання стало використання стандартних компонентів середовища розробки. Наперед, слід зазначити, що вони впоралися із цим завданням. Візуальні компоненти повністю відтворили інтерфейс програми.

Отже, завдання зовнішнього оформлення програми взяли на себе візуальні компоненти. Розглянемо суть роботи програмного інструмента й поступово розглянемо всі елементи керування ним.

Запустивши програму, ми бачимо головне вікно, на якому розташовані елементи керування (Рис. 3.1).

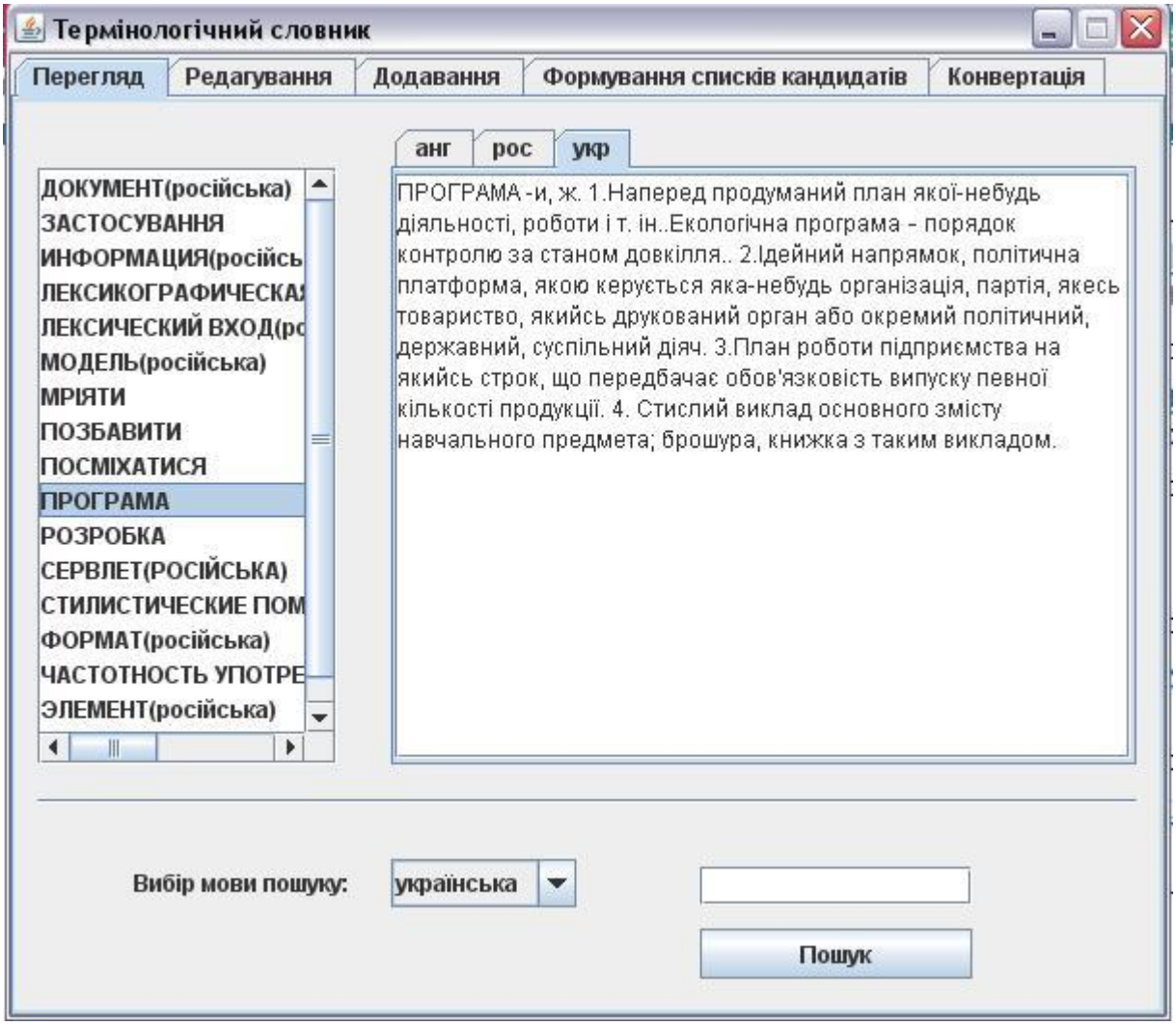

Рис.3.1 – Головне вікно програми

Головне вікно (вкладка перегляд) дає можливість повністю звернутись до потрібної інформації. Ліворуч ми бачимо список усіх термінів даної предметної області, які є у базі даних. Якщо треба подивитися значення конкретного терміна – треба лише вибрати його. Праворуч ми бачимо панель вкладок, на якій відображається згідно вибраної мові інтерпретації словникова інформація, а саме – лексичний вхід, стилістична інформація, граматична інформація, інтерпретація, приклад, джерело прикладу, зона фразеології, якщо вона присутня для даного терміна. Слід сказати, що в списку терміни відсортовані за алфавітом. Це було досягнуто з допомогою класу, який був розроблений, як клас сортування елементів списку – Sortedlistmodel. Базова мова побудови списку термінів – українська. Тому,

якщо термін немає української інтерпретації, тоді в списку його ім'я відображається на тій мові, на якій він має значення.

Якщо потрібно знайти у базі деяку словникову статтю, то вибрав відповідний пункт в головному меню (нижня частина форми) – можна її знайти. Для цього потрібно ввести всього лише слово по якому вести пошук та мову пошуку, яка запропонована у списку (Рис.3.2).

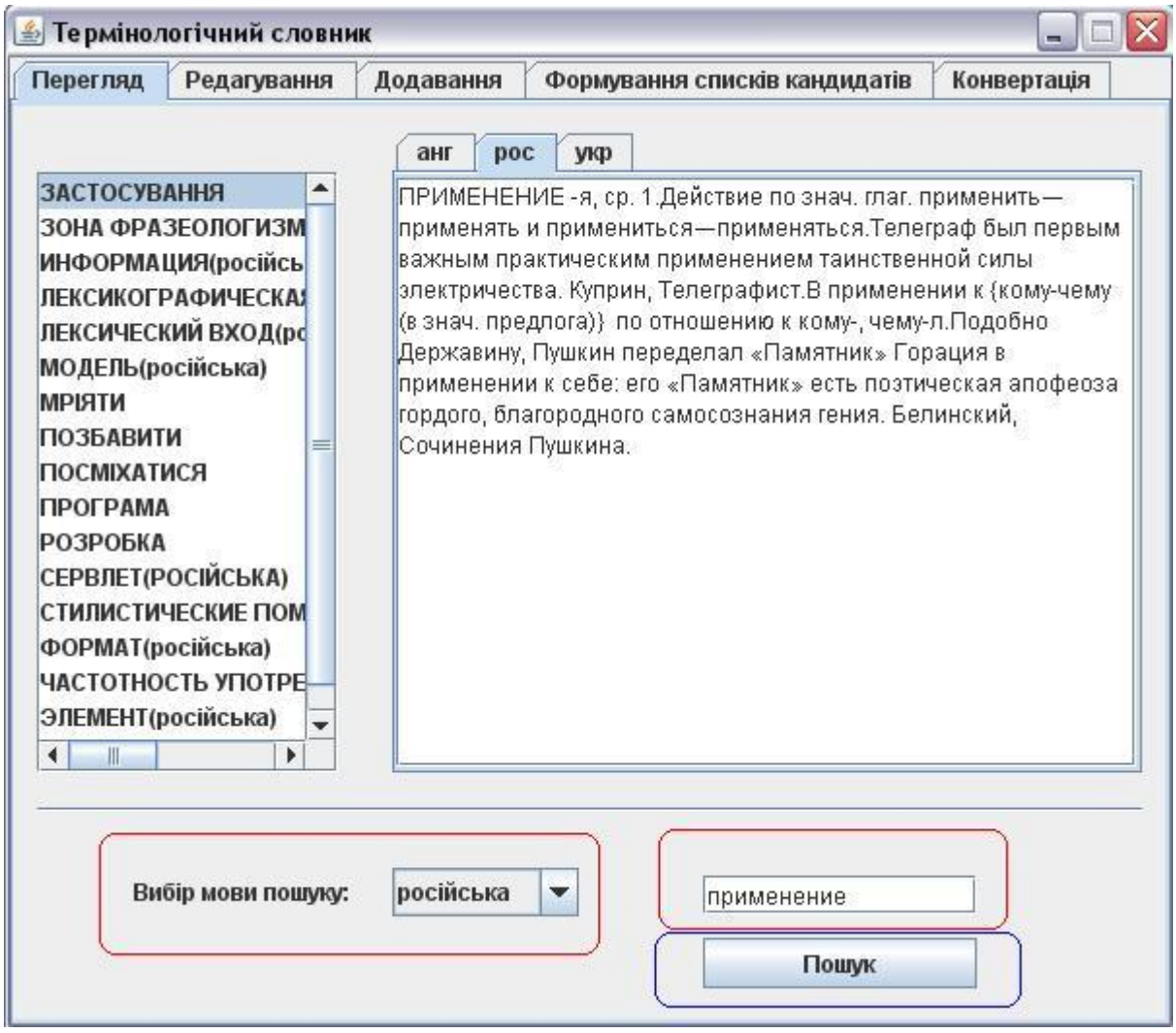

Рис.3.2 – Пошук потрібної словникової статті

Вкладка Редагування (Рис.3.3).

Користувач (експерт) має можливість редагувати, змінювати й видаляти дані, які стосуються певного терміна предметної області. А саме

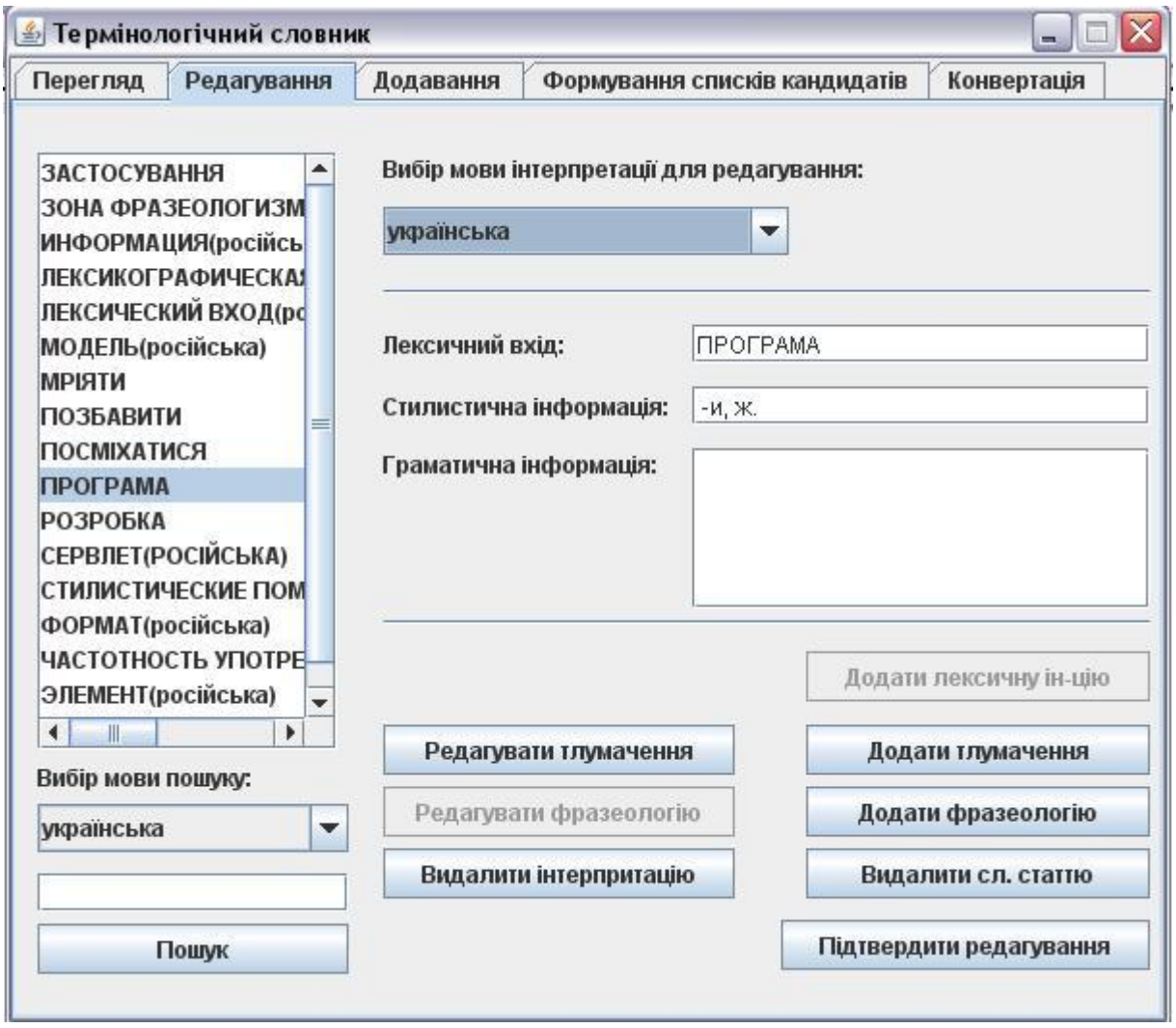

Рис.3.3 – Вкладка редагування

редагувати тлумачення, змінювати текст тлумачення, приклад і джерело приклада, видалити, як певне тлумачення, так і всі тлумачення даної словникової статті (Рис.3.4); додавати фразеологію або редагувати її (Рис.3.5).

Виходячи із цього, йому надані відповідні форми з можливістю здійснення всіх вище перерахованих операцій. Вони складаються із груп меню, які надають повну можливість заповнення бази й редагування.

Розглянемо інтерфейс програми більш докладно.

При редагуванні тлумачення на відповідній формі ми бачимо зліва – тип словникової інформації (наприклад тлумачення, джерело й т.і.), справа – текстове поле з поточними значення, якщо воно не пусте. Нижче на формі

присутні елементи інтерфейсу для переходу до іншого тлумачення, а також для підтвердження редагування, видалення поточного тлумачення та

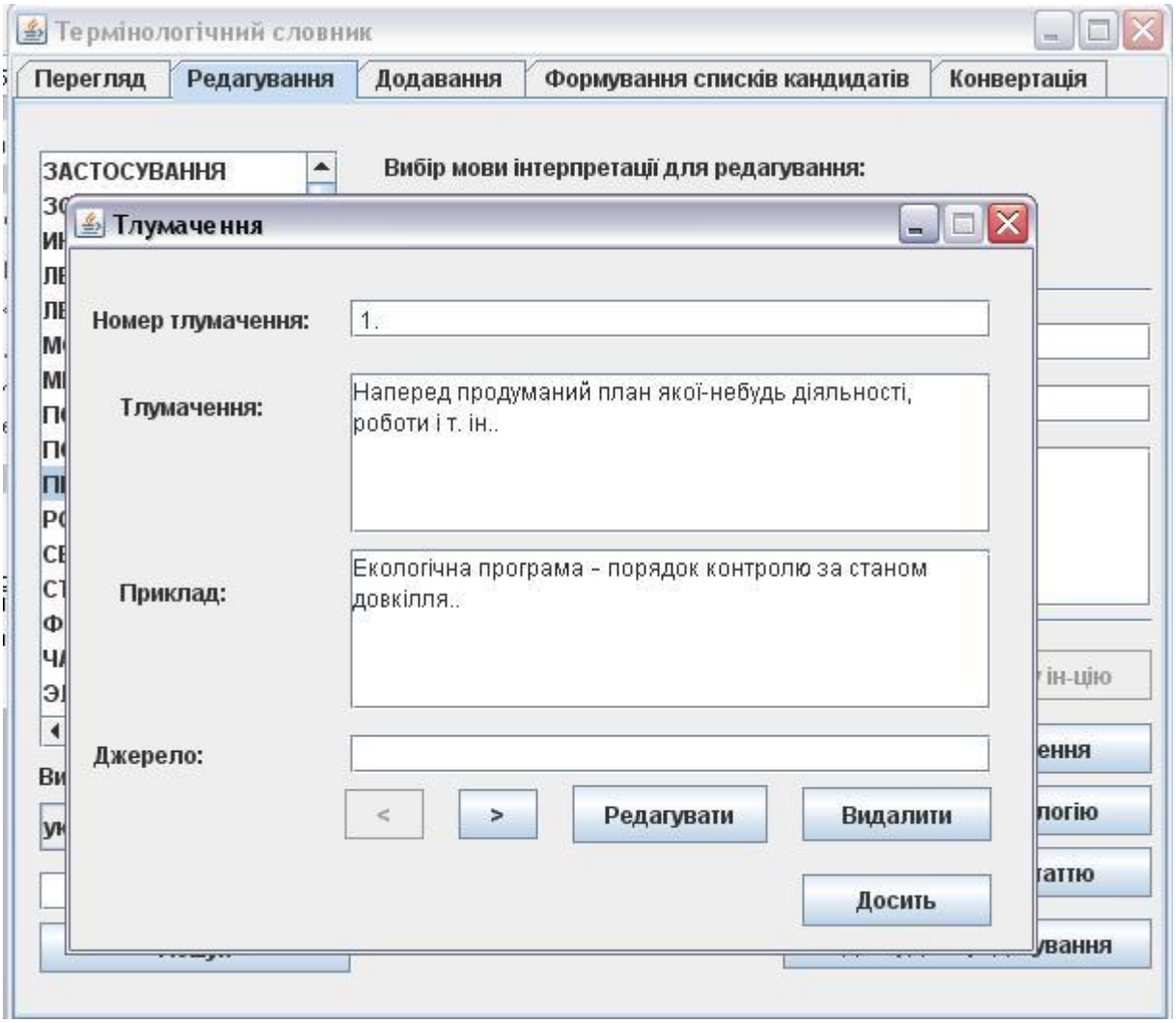

Рис.3.4 – Форма редагування тлумачення

елементу для виходу із режиму редагування тлумачення.

При редагуванні фразеологічної інформації виконуються практично ті ж самі вказівки щодо зміни інформації та інших прийнятих операції у нашій практиці.

Якщо потрібно знайти у базі деяку словникову статтю, щоб далі вести роботу з нею – необхідно скористатися пошуком. Як це зробити було детально описано вище.

Після усіх проведених операцій щодо зміни словникової статті треба підтвердити редагування, натиснувши елемент, який має назву «Підтвердити редагування».

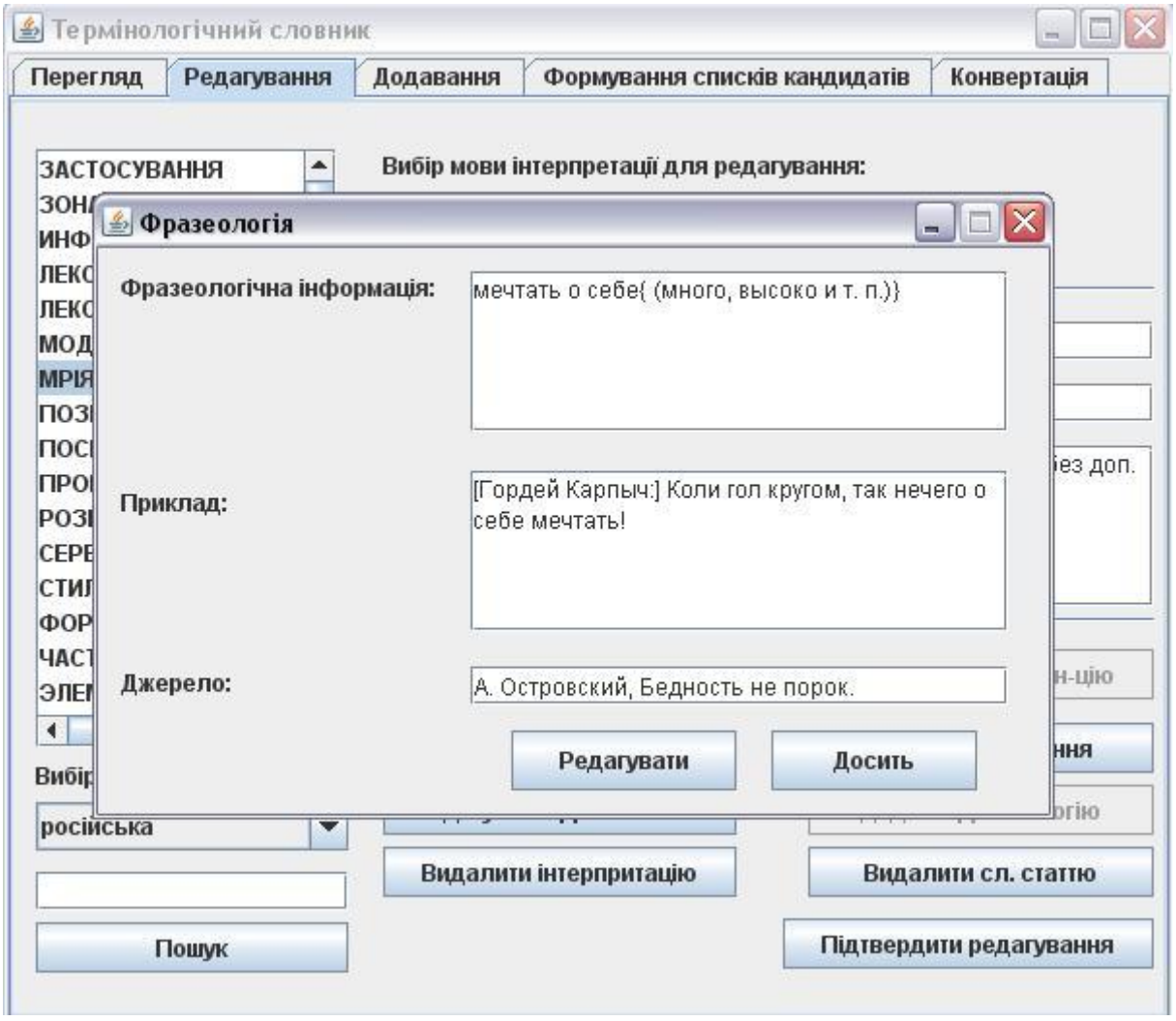

Рис.3.5 – Форма редагування фразеології

Вкладка Додавання (Рис.3.6).

Користувач (експерт) має можливість додавати нові терміни в словник. На формі наданий відповідний інструментарій для цього. Оперуючи потрібними типами даних про термін і вносячи їх до відповідних текстових полів відбувається наповнення бази даних. Слід зазначити, що починати наповнення словникової інформації для нового терміну слід з вибору мови інтерпретації, тобто мови на якому він буде представлений, і введення

лексичного входу. Потім можна приступати до додавання тлумачень і фразеології. Аби здійснити додавання цього ж терміну на іншій мові, які є доступними у випадному списку – потрібно змінити мову і продовжувати аналогічно вводити інформацію. Для остаточного утвердження потрібно натиснути елемент з надписом «Додати сл.статтю».

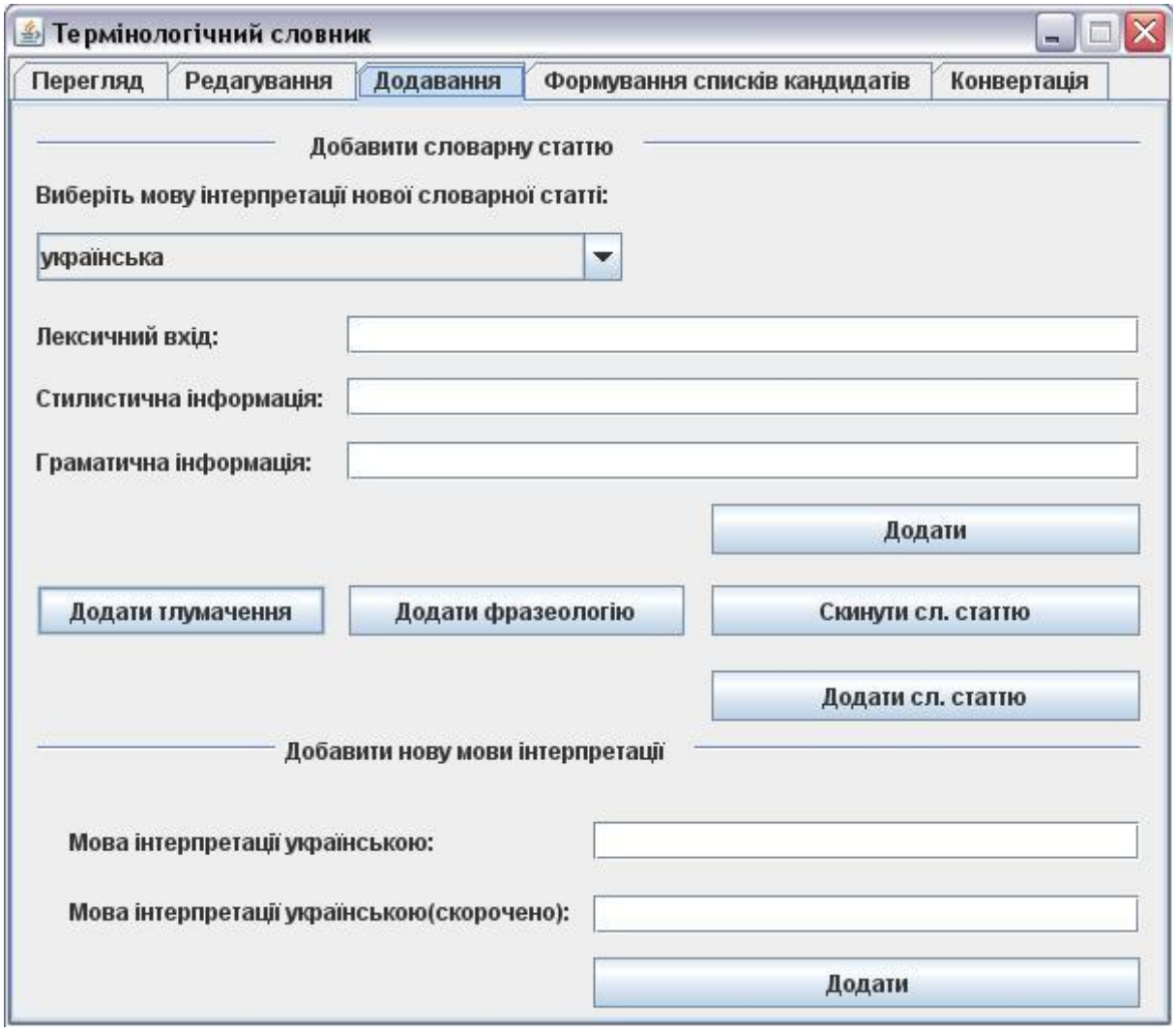

Рис.3.6 – Вкладка додавання

При додаванні чергового терміну програма може видати попередження у вигляді повідомлення, що такий термін вже існує в базі даних. Тоді завдання – спрощується. Потрібно лише перейти у вкладку редагування. За допомогою пошука знайти потрібну словникову статтю та добавити чи змінити словникову інформацію.

Може виникнути така ситуація, що в запропонованому списку немає потрібної експертові мови для інтерпретації. Він може сам додати потрібну мову за допомогою елементів, які знаходяться в нижній частині форми.

ВИСНОВКИ

Незважаючи на численні дослідження і розробки у напрямі автоматизації термінографічних робіт, на ринку немає спеціалізованого інструменту з необхідною функціональністю. Тому актуальним було завдання розробки саме такого програмного засобу.

В ході виконання роботи було досліджено методи побудови багатомовних термінологічних словників, визначено необхідні функціональність, структура та засоби розробки. В результаті отримано багатофункціональний програмний засіб для автоматизації термінографічних робіт, який дає можливість не лише використовувати словник, але і створювати або редагувати словникову статтю, форматувати її, додавати нові мови представлення або інтерпретації словникової статті. Інструмент забезпечує наступні функції: перегляд списку словникових статей в словнику, створення, пошук словникової статті по доступній мові інтерпретації, додавання нової мови, якщо її немає в запропонованому списку, можливість редагування потрібної словникової інформації, включаючи перегляд, виправлення граматичної і стилістичної інформації, можливість додавання нового тлумачення, нового прикладу і джерела прикладу, і таке інше. Тобто, створено компонент середовища розробки багатомовного термінологічного словника.

База даних представляє структурований XML-документ, призначений для зберігання зон опису словникової статті: лексичного входу, граматичної та стилістичної інформації, тлумачень, прикладу тлумачення та його джерела, зони фразеологізмів.

Завантаження повного документа, розбір і аналіз структури ґрунтується на об'єктно-базованій ієрархії (DOM). Розбір включає читання XML-документу для визначення його структури і вмісту. При розборі документа за допомогою парсера DOM, ми отримуємо структуру дерева, яка представляє його вміст. Весь текст, елементи і атрибути знаходяться в структурі дерева.

Валідація в DOM здійснюється за допомогою DTD-моделі, тобто DOM-парсер перевіряє правильність XML-документу. Перевірка правильності дозволяє підтвердити, що XML-дані відповідають заданій структурі. Таким чином, DTD визначає елементи, які можуть з'являтися в документі, порядок їх появи та порядок вкладеності, інші основні деталі структури XML.

Таким чином, забезпечується можливість створення, читання і редагування словника.

У ході реалізації було застосовано такі мови програмування та web-програмування, як HTML, XML, RegExp, Java.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Баранов А.Н. Введение в прикладную лингвистику. Изд. 3-е. / Баранов А.Н. – М.: Издательство ЛКИ, 2007. – 360с.
2. Беляева Л.Н., Герд А.С., Убин И.И. Автоматизация в лексикографии / Беляева Л.Н., Герд А.С., Убин И.И. – СПб., Питер, 1996. – 318-333с.
3. Валиков А.Н. Технология XSLT / Валиков А.Н. – СПб. : Бхв-Петербург, 2002. – 544 с.
4. Добров Б.В., Лукашевич Н.В., Сыромятников С.В., Формирование базы терминологических словосочетаний по текстам предметной области // Пятая Всероссийская научная конференция «Электронные библиотеки: перспективные методы и технологии, электронные коллекции», Санкт-Петербург, 28-31 октября 2003 г. – СПб.: СпбГУ, 2003. – с. 201-210.
5. Эккель Б. Философия Java. Библиотека программиста. 4-е издание/ Эккель Б. – СПб. : Питер, 2009. – 640с.
6. Кузнецов М., Симдянов И. Объектно-ориентированное программирование / Кузнецов М., Симдянов И. – Спб. : Бхв-Петербург, 2007. – 608с.
7. Лукашевич Н.В., Добров Б.В. Тезаурус русского языка для автоматической обработки больших текстовых коллекций // Компьютерная лингвистика и интеллектуальные технологии: Труды Международного семинара Диалог 2002 / Под ред. А.С.Нариньяни. – М. : Наука, 2002. – 338-346с.
8. Лукашевич Н.В., Салий А.Д., Добров Б.В. Использование компьютерных технологий для экспертизы терминологического словника в области государственного финансового контроля // Компьютерная лингвистика и интеллектуальные технологии: Труды международной конференции "Диалог'2005" (Звенигород, 1-6 июня, 2005 г.) / Под ред. И.М. Кобозевой, А.С. Нариньяни, В.П. Селегея. – М.: Наука, 2005.

9. Мельников Г.П. Основы терминоведения / Мельников Г.П. – М. : Изд-во РУДН, 2009. – 116с.
10. Статывка Ю.И., Пономарев С.В. Разработка программы для автоматизации терминографических работ // Сучасні тенденції розвитку інформаційних технологій в науці, освіті та економіці: Матеріали IV Всеукраїнської науково-практичної конференції. 15-17 квітня 2010р., м.Луганськ. – Луганськ : Phoenix, 2010. – 122-124с.
11. Статывка Ю.И., Пономарев С.В. Разработка программы для автоматизации терминографических работ / Статывка Ю.И., Пономарев С.В. // Вестник ВНУ им.В.Даля, 2010. – в печати.
12. Сергеев А. HTML и XML. Профессиональная работа / Сергеев А. – М. : Діалектика, 2004. – 881с.
13. Пономарев С.В. Разработка программы для пополнения терминологического словника // Матеріали наукової конференції студентів Східноукраїнського національного університету імені Володимира Даля, секція інформатики. 17-21 травня 2010р., м. Луганськ. – в печати.
14. Питц-Моудтис Н., Кирк Ч. XML: Пер. с англ. / Питц-Моудтис Н., Кирк Ч. – СПб. : БХВ-Петербург, 2000. – 736с.
15. Поль А. J2EE. Разработка бизнес-приложений / Поль А. - СПб. : DiaSoft, 2006. – 719с.
16. Суперанская А.В., Подольская Н.В., Васильева Н.В. Общая терминология: Терминологическая деятельность / Суперанская А.В., Подольская Н.В., Васильева Н.В. – СПб. : Питер, 2008. – 288 с.
17. Шелов С.Д. Определение терминов и понятийная структура терминологии / Шеллов С.Д. – СПб. : Изд-во СПбГУ, 1998. – 236с.
18. Шилдт Г., Холмс Д. Искусство программирования на Java / Шилдт Г., Холмс Д. – СПб. : Вильямс, 2005. – 336 с.
19. Широков В.А. Інформаційна теорія лексикографічних систем. / Широков В.А. – К.: Довіра, 1998. – 331с.

20. Erik T. Learning XML, 2nd Edition / Erik T. – O'Reilly Media, 2003. – 600p.
21. Frantzi K.T., Ananiadou S. Automatic Term Recognition using Contextual Cues / Frantzi K.T., Ananiadou S. – Nagoya, Japan, 1997.
22. Harold E. Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX / Harold E. – Addison Wesley, 2002. – 1120p.
23. McLaughlin B. Java and XML, 2nd Edition / McLaughlin B. – O'Reilly, 2001. – 528p.
24. Tennison J. Beginning XSLT 2.0: From Novice to Professional / Tennison J. – Apress, 2005. – 825p.
25. Church K., Hanks P. Word association norms, mutual information, and lexicography / Church K., Hanks P. – Vancouver, Canada, 1989. – 76-83p.