Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

# 

# 

Реалізація бази знань на основі системи Protégé

**Текстова частина до курсової роботи**

**за спеціальністю „Програмна інженерія” 6.050103**

**Керівник курсової роботи**

к.ф.-м.н., доц. Жежерун О. П.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(підпис)* “\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2015 р.

Виконала студентка

Яремко С. А.

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2016 р.

Київ 2016

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_ Бублик В.В.

(підпис)

„\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016 р.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на курсову роботу

студентці Яремко Соломії Андріївні факультету інформатики 3 курсу

ТЕМА Реалізація бази знань на основі системи Protégé

Розробити Базу знань за допомогою використання редактора онтологій Protégé та базу даних у СКБД Microsoft Access

Вихідні дані:



Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Календарний план

Анотація

Вступ

РОЗДІЛ 1: Поняття семантичного вебу (Semantic Web)

РОЗДІЛ 2: Онтології та їх реалізація

РОЗДІЛ 3: Створення баз знань за допомогою системи Protégé

Висновки

Список використаної літератури

Дане індивідуальне завдання демонструє практичне використання основних можливостей, які надані редактором онтологій Protégé, на прикладі системи Protégé (версії 4.0), у процесі реалізації бази знань, а також допомагає порівняти можливості баз знань із можливостями баз даних.

Дата видачі „\_\_\_” \_\_\_\_\_\_\_\_\_ 2016 р. Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Завдання отримав \_\_\_\_\_\_\_\_\_\_

(підпис)

**Тема:** Реалізація бази знань на основі системи Protégé

**Календарний план виконання роботи:**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Назва етапу курсової роботи | Термін виконання етапу | Примітка |
| 1. | Отримання теми курсової роботи. | 6.10.2015 |  |
| 2. | Пошук та збір тематичних матеріалів. | 20.11.2015 |  |
| 3. | Ознайомлення з необхідними матеріалами та побудова структури роботи. | 20.12.2015 |  |
| 4. | Написання вступу та плану роботи. | 30.12.2015 |  |
| 5. | Аналіз матеріалів, що необхідні для написання першого розділу роботи, та безпосереднє його написання. | 20.02.2016 |  |
| 5. | Опрацювання необхідних матеріалів та написання другого розділу роботи. | 20.03.2016 |  |
| 6. | Ознайомлення з потрібними джерелами та завершення роботи над третім розділом роботи. | 25.04.2016 |  |
| 7. | Коректне оформлення роботи відповідно до вимог написання курсової роботи. | 25.04.2016 |  |
| 8. | Створення презентації та написання доповіді для захисту роботи. | 4.05.2016 |  |
| 9. | Подання та аналіз попередньої версії роботи з керівником. | 11.05.2016 |  |
| 10. | Корегування роботи згідно із зауваженнями керівника. | 18.05.2016 |  |
| 11. | Захист курсової роботи. | 20.05.2016 |  |

СтудентЯремко С. А.

Керівник Жежерун О. П.

**“\_\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

ЗМІСТ

**Анотація** …………….......................................................................................6

***ВСТУП***...............................................................................................................7

**РОЗДІЛ 1: Поняття семантичного вебу (Semantic Web)**.…...................9

* 1. Базові принципи………………………………………………..9
  2. Логічні рівні семантичного вебу……………………………..10

1.2.1 Уніфікований ідентифікатор ресурсів (URI) …………………11

1.2.2 Розширювана мова розмітки (XML)………………………….12

1.2.3 RDF/RDFS……………………………………………………….13

1.2.4 Мова опису [онтологій](https://uk.wikipedia.org/wiki/%D0%9E%D0%BD%D1%82%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%8F_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) для семантичної павутини OWL …….16

**РОЗДІЛ 2: Онтології та їх реалізація**…………………………………….22

* 1. Поняття онтології…………………………………………….22
  2. Порівняння баз знань та баз даних…………………………24
  3. Загальний опис редактора онтологій Protégé………………27

**РОЗДІЛ 3: Створення баз знань за допомогою системи Protégé**…….30

3.1 Створення властивостей………………………………………30

* + 1. Типи властивостей…………………………………………….30

3.1.2 Характеристики властивостей………………………………..32

3.1.3 Обмеження на властивості…………………………………….34

3.1.3.1 Кванторні обмеження (Quantifier Restrictions: existential restrictions, universal restrictions)……………………………………35

3.1.3.2 Обмеження кардинальності (Cardinality Restrictions)……39

3.1.3.3 hasValue обмеження ( hasValue restrictions )……………..40

* 1. Створення класів…………………………………………….41
     1. Іменовані класи (named classes) та непересічні класи (disjoint classes)………………………………………………………..41
     2. Класи - перелічення (Enumerated Classes)………………...42

3.2.3 Класи – доповнення (Complement Classes )……………….43

* 1. Шаблон програмування Value Partitions (поділ значення) ..45
  2. Використання система логічного виводу (OWL reasoner)..47
  3. Створення індивідів……………………………………………50
  4. Запити до онтології……………………………………………50

**Висновки** ....................................................................................................53

**Список використаної літератури**..........................................................54

Анотація

У курсовій роботі визначенота проаналізованоосновні можливості редактора онтологій Protégé у процесі створення онтології, визначено основні переваги використання баз знань у порівнянні з базами даних. Також у роботі було описано основні принципи проекту Semantic Web, наведено детальний розгляд його основних логічних рівнів та висвітлено спосіб подання знань за допомогою онтологій.

Робота наводить опис розробки та реалізації бази знань.

В роботі використовується система Protégé (версії 4.0).

**Ключові слова:** Семантичний веб, онтологія, база даних, база знань, формальна мова опису онтологій, OWL, RDF, RDFS, XML, редактор онтологій, Protégé, система логічного виводу, reasoner, клас, властивість, екземпляр.

Вступ

**Актуальність теми.** Ідея семантичного вебу, що була запропонована розробником вебу Тімом Бернерсом-Лі у 1998 році та полягала у перетворенні всіх інформаційних ресурсів вебу в єдину базу знань, користуватися якою могли б як люди, так і комп’ютери, базується на розробці мов для вираження інформації в формі, придатній для машинної обробки. Одними з таких мов є формальні мови для визначення та подання веб-онтологій, які стали одним з найперспективніших підходів вирішення проблеми представлення знань, певною спробую всеохоплюючої та детальної формалізації деякої області знань за допомогою визначення базових [об’єктів](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%E2%80%99%D1%94%D0%BA%D1%82) і [відношень](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%BD%D1%8F) між ними. На сьогодні існує значна кількість інструментальних програмних засобів, призначених для проектування, редагування та аналізу онтологій - редакторів онтологій, одним з яких є система Protégé, яка і є предметом дослідження даної курсової роботи. Ця система підтримує використання мови OWL, що є стандартом Консорціуми W3 та певним рішенням проблеми необхідності представлення знань в мережі в єдиному форматі, і надає широкі можливості в процесі роботи над онтологіями, зокрема підтримує мову запитів для пошуку нетривіальних тверджень, надає можливість аналізу цілісності онтології та використання механізму логічного виводу.

**Мета дослідження –** визначитита проаналізуватиосновні функціональні можливості редактора онтологій Protégé у процесі створення онтології (бази знань), а також визначити основні переваги використання баз знань у порівнянні з базами даних.

Досягнення мети передбачає виконання низки **завдань**:

* опис основних цілей проекту Semantic Web;
* розгляд основних технологій реалізації Semantic Web;
* опис способу подання знань за допомогою онтологій;
* порівняння бази даних із базою знань;
* розгляд основних можливостей системи Protégé;
* опис створення бази знань у системі Protégé.

**Об’єкт дослідження** – редактор онтологій з відкритим кодом та фреймворк для створення баз знань Protégé.

**Предметом дослідження** є можливості, що надає система Protégé у процесі створення онтологій з її екземплярами (баз знань).

Логіка дослідження зумовила таку структуру курсової роботи: вступ, 3 розділи, висновки, список використаної літератури.

У першому розділі описано поняття Семантичного Вебу та висвітлено основні принципи цього проекту. Також проведено детальний розгляд логічних рівнів Семантичного Вебу.

Другий розділ присвячено розгляду способу подання знань за допомогою онтологій. Також тут наведені основні функціональні можливості редакторів онтологій та проведено короткий опис редактора онтологій Protégé 4.0.

У розділі три визначенота описаноосновні можливості редактора онтологій Protégé у процесі створення онтології.

1. Поняття семантичного вебу

1.1 Базові принципи

WWW першочергово розроблявся як інформаційний простір, корисний не лише для людей, але й як простір, у якому могли співпрацювати комп’ютери. Проте основною перешкодою для реалізації цієї цілі є те, що інформація у Web є зрозумілою лише для людей, але не для самих комп’ютерів. Для вирішення цієї проблеми у 2001 році винахідник вебу та голова консорціуму W3C Тім Берннерс-Лі розпочав проект сематичного вебу, що за визначенням самого Тіма Бернерса-Лі є розширенням сучасного вебу, в якому інформації надається чітко визначений зміст, який сприятиме спільній роботі людини і комп’ютера.

Підхід семантичного вебу базується на розробці мов для опису інформації у формі, доступній для машинної обробки. Ціллю семантичного вебу є перетворення всіх інформаційних ресурсів вебу в єдину базу знань, тобто створення універсального засобу семантичного пошуку, обробки, збереження та передавання інформації [1]. Для цього необхідно забезпечити кожний інформаційний ресурс описом його семантики і надати засоби для автоматизованої обробки цих описів і представлення знань про них.

Семантичний веб можна розглядати як ефективний спосіб подання даних у Всесвітні павутині або як глобально пов’язану базу даних. За визначенням Консорціуму W3C , СВ являє собою розширенням існуючої мережі Інтернет, що повинно забезпечити «розуміння» інформації комп’ютерами, виділення ними інформації, що найбільше підходить за певними критеріями, і вже після цього – надання інформації користувачам [1].

Як сказано у визначенні, яке надано на домашній сторінці проекту - «Semantic Web є абстрактним поданням даних у Всесвітній павутині, яке базується на стандартах RDF та інших уже існуючих та широковикористовуваних стандартах. Проект розробляється консорціумом W3C у співпраці з великою кількістю дослідників, вчених і промислових партнерів ».

Semantic Web відкриває доступ до чітко структурованої інформації для будь-яких додатків, незалежно від платформи і незалежно від мов програмування. Програми зможуть самі знаходити потрібні ресурси, обробляти інформацію, класифікувати дані, виявляти логічні зв'язки, робити висновки і навіть ухвалювати рішення на основі цих висновків.

Отже, Semantic Web - це динамічна концепція, що постійно розвивається, а не набір комплексних працюючих систем, що полягає в забезпеченні достатньої гнучкості для можливості подання всіх баз даних і правил логіки таким чином, щоб зв'язати їх всі разом, це надбудова над існуючим Web, яка покликана зробити розміщену в ній інформацію придатною для машинної обробки.

* 1. Логічні рівні семантичного вебу

Технологія CВ базується на таких логічних рівнях:

* URI (Universal Resource Identifier ) – уніфікований ідентифікатор, що визначає спосіб запису адреси ресурсу;
* XML (eXtensible Markup Language) – базова форма розмітки, текстовий формат для опису документів довільної структури, що не накладає жодних семантичних обмежень на зміст цих документів;
* RDF (Resource Description Framework)/RDFS (RDF Schema) – механізм опису ресурсів, що дозволяє описати модель даних для ресурсів і відношення між ними, забезпечує просту семантику для цих моделей, подаючи їх у синтаксисі XML/ проста мова опису онтологій, що надає засоби для опису властивостей та класів RDF-ресурсів, а також семантику для ієрархій-узагальнень таких властивостей та класів;
* OWL (Ontology Web Language) – мова мережевих онтологій, призначена для опису класів і відношень між ними, що забезпечує більш повну автоматичну обробку мережевого контексту, ніж XML і RDF [1].

1.2.1 Уніфікований ідентифікатор ресурсів ( URI )

Уніфікований ідентифікатор ресурсів URI - це компактний рядок літер, який однозначно ідентифікує окремий абстрактний чи фізичний ресурс.

URI можна присвоїти будь-чому, і якщо якась сутність має URI, то можна сказати, що ця сутність знаходиться у Web.

Формами URI є URL (Uniform Resource Locator) - уніфікований локатор ресурсу – надає методи знаходження ресурсів у мережі, наприклад, це адреса, по якій завантажується сторінка та URN (Uniform Resource Name) – уніфіковане ім’я ресурсу, що позначає поняття ресурсу, проте не вказує, як дістатися до цього ресурсу. Загальна структура URN має такий вигляд: (urn:namespace-id:resourse-id).

Приклад URI:

http://www.semanticweb.org/solomka/ontologies/dishes.owl#belongsToCountry

Приклад URL: https://en.wikipedia.org/wiki/Semantic\_Web

Приклад URN: urn: ISSN:0167-6423

У 2005 році на зміну URI був запропонований інтернаціоналізований ідентифікатор ресурсу - Internationalized Resource Identifiers (IRI) – формат задавання ресурсу, який дозволяє використовувати Unicode – символів на противагу URI, обмеженою системою кодування символів ASCII. Тобто IRI надає можливість ідентифікувати абстрактний або фізичний ресурс на будь-якій мові світу. URI можуть містити тільки латинські символи та знаки пунктуації з набору символів US-ASCII (в цілому близько 60 символів). Для забезпечення принципів інтернаціоналізму, збереження «читабельності» для людини, в IRI було запропоновано, що ці ідентифікатори можуть містити будь-які символи Юнікоду (Unicode / ISO10646) в чистому вигляді, без будь-якого кодування. IRI ведуть до більш високого ступеня рівноправності користувачів Інтернет. В майбутньому ідентифікатори IRI покликані замінити URI.

Отже, у СВ на зміну зв’язкам між веб-сторінками приходить модель, в якій одиниця даних вказує на іншу, використовуючи глобальні посилання (ідентифікатори), якими і є URI. Також URI використовується для уніфікованого іменування сутностей при побудові RDF-схем [1].

1.2.2 Розширювана мова розмітки ( XML )

Розширювана мова розмітки (XML) - являє собою простий, дуже гнучкий текстовий формат опису документів будь-якої структури, отриманий з SGML (ISO 8879). Спочатку розроблений для вирішення завдань великомасштабного електронного видання, XML також відіграє все більш важливу роль в обміні широкого спектру даних в Інтернеті. Він є підкласом мови SGML (стандартної узагальненої мови розмітки), проте більш простий для розуміння і обробки [6].

XML є важливою складовою СВ, що дозволяє кожному створювати власні теги, а відповідно власні формати документів і потім писати документи в цих форматах. Ці формати документів можуть включати розмітку, яка уточнює зміст контенту документа. Документ з розміткою може бути прочитаним комп'ютером.

Кожен синтаксично коректний XML-документ повинен задовольняти такі вимоги:

* у документі повинен бути лише один елемент верхнього рівня (кореневий елемент);
* кожен елемент повинен мати відкриваючий та закриваючий тег;
* атрибути (пари «ім’я - значення» всередині тегу відкриття елемента) в межах одного елемента мають бути унікальними;
* ім’я елементів та тегів мають задовольняти такі обмеження: першим символом має бути літера, символ нижнього підкреслення чи двокрапка; воно не починається з «xml» у будь-якій комбінації літер маленького чи великого реєстру [1].

Функції XML:

* + надання синтаксису для інших мов розмітки;
  + семантична розмітка Web-сторінок. XML-представлення може використовуватися на Web-сторінці разом з таблицею стилів XSL, що визначає коректний вивід на екран різних елементів;
  + єдиний формат обміну даних. XML-представлення може передаватися між двома застосуваннями як об'єкт даних.

Отже, XML дозволяє користувачам додавати певну структуру у документи, проте не дає можливості зазначити, що ця структура означає.

* + 1. RDF/RDFS

RDF (Resource Description Framework ) – стандартна модель подання метаданих (даних про дані) у мережі, універсальний спосіб розкладання будь-яких знань на маленькі фрагменти, це основна технологія СВ, яка дозволить комп'ютерним програмам користуватися всією структурованою інформацією, розміщеною у вузлах вебу. Ідея полягає в тому, щоб одним простим способом можна було б описати будь-який факт, притому в такому структурованому вигляді, щоб його могли обробляти комп'ютерні програми. В основі цієї технології лежить XML-синтаксис.

Основною відмінністю RDF від XML і інших технологій - це те, що RDF призначений для представлення знань в розподіленому світі, тобто для RDF особливо важливим є сенс. Все, з чим працює RDF, має певний сенс, - посилається на якийсь конкретний об'єкт або на абстрактне поняття, або на якийсь факт. Стандарти, засновані на RDF, описують логічні висновки, що зв'язують ці факти, і вказують, як можна знайти самі факти у величезній базі даних всіх знань, представлених в RDF.

Стандарт RDF був прийнятий у 1998 році консорціумом W3C та має на меті стандартизувати визначення та використання метаданих, які описують ресурси Web. Однак, RDF також добре підходить і для представлення даних, тобто за допомогою RDF можна описувати як структуру ресурсу, так і пов'язану з ним предметну область.

Модель даних RDF є графом з позначеними як вершинами, так і дугами, який дозволяє визначати зв'язки між сутностями.

Стандарт RDF (Resource Description Framework) включає дві основні частини - власне спосіб опису ресурсів, а також спосіб задання схем, за якими ці ресурси описуються.

Перша частина RDF визначає просту модель для опису об'єкта, який розглядається в якості ресурсу, та зв'язків між ресурсами в термінах поіменованих властивостей і значень.

Друга (RDF Schema - RDFS) служить для задання опису семантики властивостей в RDF (опису структури предметної області) і аналогічна діаграмі класів в UML. RDF-схема розроблена як система типів для RDF. Використовуючи схему RDF, ми можемо створювати властивості і класи, а також створювати діапазони і області для властивостей.

Інформаційні ресурси в RDF - це ресурси Web, що ідентифікуються унікальним чином за допомогою їх URI.

Кожне твердження RDF складається з трійки: суб’єкт, предикат, об’єкт. Наприклад, у твердженні «The sky has the color blue» суб’єктом є “sky », предиктом – 'color', обєктом – ‘blue’. Проте кожен елемент трійки повинен мати унікальний ідентифікатор (URI).

RDF описує ресурси у вигляді орієнтованого графа, вершин та звязків між ними : вершина для обєкта, вершина для субєкта та звязок для предиката [1].

Обєктами в RDF-твердженнях можуть бути або URIref-посилання , або літерали (сталі величини), які у свою чергу можуть бути або типізованими (тип – це також URI), або нетипізованими [1].

Основною метою RDF є запропонувати базову модель даних «об'єкт - атрибут - значення» для метаданих. Стандарт RDF надає засоби для запису тверджень щодо ресурсів, проте не надає можливостей для визначення словників, які використовуватимуться в твердженнях .

RDF Schema дозволяє визначати конкретний словник для даних RDF і вказувати види об'єктів, до яких можуть застосовуватися ці атрибути. Іншими словами, механізм RDF Schema надає базову систему типів для моделей RDF.

Основними RDFS властивостями є: subClassOf, subPropertyOf, range, domain, label, comment.

Таким чином, RDF надає можливість формулювати твердження у вигляді, придатному для обробки комп'ютером і це є основою Semantic Web.

Всі терміни для схеми RDF починаються з "http://www.w3.org/2000/01/rdf-schema#". Три найбільш важливих поняття, які дає нам RDF і схема RDF - це "Ресурс" (rdfs: Resource), "Клас" (rdfs: Class) і "Властивість" (rdfs: Property). Ці поняття є "класами" в тому сенсі, що даним класами можуть належати терміни. Наприклад, всі терміни є типами ресурсу. У схемі RDF є також кілька інших властивостей: rdfs: subClassOf і rdfs: subPropertyOf, за допомогою яких можна вказати, що один клас або властивість є підкласом іншого. По суті, RDF-схема повинна визначати онтологію предметної області.

Важливою особливістю стандарту RDF, як і XML, що лежить у його основі, є розширюваність.

За допомогою RDF можна задавати структуру опису джерела, використовуючи і розширюючи вбудовані поняття RDF-схеми, такі як класи, властивості, типи, колекції. Модель схеми RDF включає спадкування; успадковуватися можуть як класи, так і властивості.

Таким чином, RDF цілком підходить на роль універсальної мови опису семантики ресурсів і взаємозв'язків між ними.

До засобів опису змісту і зв'язку між термами разом зі схемами відносяться і онтології.

1.2.4 Мова опису [онтологій](https://uk.wikipedia.org/wiki/%D0%9E%D0%BD%D1%82%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%8F_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) для семантичної павутини OWL

OWL – рекомендована W3C формальна мова для визначення і подання структурованих веб-онтологій, тобто онтологій, які доступні для машинної обробки, що базується на основних елементах RDF і є розширенням RDF/RDFS. Створення OWL це - відповідь на необхідність подання знань в мережі в єдиному форматі. Формальною основою мови опису OWL - онтологій є дескрипційні (описові) логіки (description logics).

Мова OWL має 3 діалекти (підмножини термінів):

- OWL Lite - має найменшу виразну потужність з усіх, але для вирішення простих завдань його може бути досить.

- OWL DL - має виразність еквівалентну виразності дескрипційної логіки (її можна розглядати як розв'язні фрагменти [логіки предикатів](https://uk.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D1%96%D0%BA%D0%B0_%D0%BF%D0%B5%D1%80%D1%88%D0%BE%D0%B3%D0%BE_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BA%D1%83) першого порядку). Для більшості завдань, що зустрічаються при проектуванні онтологій, виразності цього діалекту досить. Цей діалект має дві важливі властивості: повнота (всі логічні висновки, що припускаються тією чи іншою онтологією, будуть гарантовано обчислюваними) і розв'язність (всі обчислення завершуються в кінцевий час). Зокрема в OWL DL - класу заборонено бути екземпляром.

- OWL Full - найбільш виразний діалект, еквівалентний RDF. Однак при використанні OWL Full немає ніяких гарантій по обчислюваності висновків [4].

Кожен з цих діалектів (крім Lite) є розширенням попереднього [4] і як наслідок: будь-яка OWL Lite онтологія є OWL DL онтологією, а будь-яка OWL DL онтологія є OWL Full онтологією.

В основі мови лежить уявлення дійсності в моделі даних «об’єкт - відношення». Кожному елементу опису в цій мові (в тому числі властивостями, що зв'язують об'єкти) ставиться у відповідність [URI](https://uk.wikipedia.org/wiki/URI).

Мова OWL дозволяє описувати класи і відношення і є фактично словником, який розширює набір термінів визначених RDFS. Зокрема, OWL містить терми, що дозволяють визначити :

* Взаємозв’язки між класами (disjointWith, complementOf)
* Рівність класів (sameAs, equivalentClass)
* Додаткові типи властивостей (symmetric, transitive, inverseOf)
* Обмеження на властивості (allValuesFrom, someValuesFrom)

За допомогою цих властивостей формальна семантика OWL описує, як одержати логічні висновки, маючи онтологію, тобто одержати факти, які не подані в онтології буквально, але випливають з її семантики, якщо взяти до уваги уже існуючі факти. Ці наслідки можуть базуватися на одному чи безлічі розподілених документів, які комбінуються з використанням певних механізмів OWL.

Однією з переваг OWL – онтологій буде доступність інструментів, які можуть міркувати. Інструменти забезпечать загальну підтримку, що не є специфічною для певної предметної області [1].

OWL – це спроба зробити веб-ресурс більш доступним для автоматизованих процесів шляхом додавання інформації про ресурси, що описують або забезпечують веб-контент. OWL забезпечує можливість збору інформації з розподілених джерел за допомогою здатності онтологій бути пов’язаними, включаючи прямий імпорт інформації з інших онтологій [1].

Стандартним початковим компонентом будь-якої онтології є набір оголошень XML namespaces, вкладених у відкриваючий тег rdf:RDF, що вказують словники, конструкції яких будуть використовуватися при описі онтології:

<rdf:RDF xmlns="http://www.semanticweb.org/solomka/ontologies/dishes.owl#"

xml:base="http://www.semanticweb.org/solomka/ontologies/dishes.owl"

xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"

xmlns:pizza="http://www.semanticweb.org/solomka/ontologies/pizza.owl#"

xmlns:xsd="http://www.w3.org/2001/XMLSchema#"

xmlns:dishes="http://www.semanticweb.org/solomka/ontologies/dishes.owl#"

xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:owl="http://www.w3.org/2002/07/owl#">

Також кожна онтологія має заголовок і тіло. У заголовку міститься інформація про саму онтологію (версія, коментарі), імпортовані онтології:

<owl:Ontology rdf:about="">

<rdfs:comment

>This ontology contains information about traditional dishes of different countries.</rdfs:comment>

<owl:imports rdf:resource="http://www.semanticweb.org/solomka/ontologies/pizza.owl"/>

</owl:Ontology>

За заголовком іде тіло онтології, що містить опис класів, властивостей і екземплярів онтології , що завершується тегом </rdf:RDF>.

Базовими елементами OWL є класи та властивості.

В OWL введений новий термін клас (owl: Class). Необхідність цього пояснюється тим, що не всі класи діалектів DL і Lite є rdfs класами (в цьому випадку owl: Class є підкласом rdfs: Class). У діалекті Full, подібних обмежень немає, і owl: Class фактично є синонімом rdfs: Class. Для організації класів в таксономії використовується властивість rdfs: subClassOf [1].

Особливе місце займають два взаємодоповнюючих класи Thing і Nothing. Перший з них є суперклассом будь-якого класу OWL, другий -підкласом будь-якого класу OWL. Екземпляр будь-якого класу OWLє підкласом классу класу Thing. Підклас класу Nothing є порожньою множиною [1].

OWL - клас може бути описаний різними способами, які будуть наведені у наступному розділі. Лише один з цих способів визначає іменований OWL - клас. Всі інши способи визначають анонімні класи через обмеження, накладені на властивості.

В OWL визначені ще 3 конструкції, комбінуючи які, можна визначати більш складні аксіоми класів:

* + - rdfs: subClassOf - вказує, що екземпляри одного класу (підкласу) є екземплярами іншого классу (надкласу);
    - owl: equivalentClass - вказує, що екземпляри двох класів збігаються;
    - owl: disjointWith - вказує, що екземпляри, які належать до одного классу, не можуть належати до інщого класу.

В OWL виділяють дві категорії властивостей: властивості - об'єкти і властивості - значення. Перші пов'язують між собою індивіди. Другі пов'язують індивіди зі значеннями даних. Обидва класи властивостей є підкласами класу rdf: Property [1].

Для визначення нових властивостей як екземплярів owl: ObjectProperty або owl: DatatypeProperty використовуються аксіоми властивостей.

Крім того, OWL підтримує такі конструкції для побудови аксіом властивостей:

* + - конструкції RDF Schema: rdfs: subPropertyOf (визначає підвластивість даної властивості), rdfs: domain (визначає домен властивості) і rdfs: range (визначає діапазон свойства)
    - взаємозвязки між властивостями: owl: equivalentProperty (визначає еквівалентну властивість) і owl: inverseOf (визначає зворотню властивість)
    - обмеження кардинальності: owl: functionalProperty (визначає однозначну властивість) і owl: InverseFunctionalProperty (зворотня функціональна властивість, тобто визначає, що властивість, зворотня даній властивості, є однозначною)
    - логічні характеристики властивсоті: owl: symmetricProperty (визначає властивість як симетричну) і owl: TransitiveProperty (визначає властивість як транзитивну).

Індивіди визначаються за допомогою аксіом індивідів (фактів).

Є два види фактів:

* факти про належність індивідів до класів і про значення властивостей цих індивідів.
* Факти про ідентичність / відмінність індивідів, що є необхідними з огляду на те, що в OWL не робиться ніяких припущень про відмінність чи збіг двох індивідів, що мають різні ідентифікатори URI.

Такі переконання виражаються аксіомами ідентичності за допомогою таких конструкцій:

* owl: sameAs - вказує на те, що два посилання URI посилаються на один і той же індивід.
* owl: differentFrom – вказує на те, що два посилання URI посилаються на різні індивіди.
* owl: allDifferent надає засіб для визначення списку попарно різних індивідів.

Основними відмінностями між OWL та RDF/RDFS є те, що OWL надає розширені можливості опису властивостей і класів. OWL забезпечує більш повну автоматичну обробку мережевого контексту, ніж та, яку підтримують XML і RDF, надаючи поряд з формальною семантикою додаткову семантичну підтримку [1]. OWL надає семантику до схеми, додає онтологічну здатність до RDF (який сам по собі забезпечує тільки вкрай обмежені можливості для формального подання знань), шляхом надання механізму автоматичного додавання нових фактів до онтології, за допомогою використання описової логіки. Більше того, логічна модель дозволяє використання мислителя (reasoner), що забезпечує здійснення автоматичної перевірки ваємоузгодженості всіх тверджень та визначень, поданих в онтології, а також забезпечує автоматичну підтримку ієрархії класів, що є особливо корисним у ситуаціях, коли класи мають більше одного батька. Саме в цьому і полягає додаткова семантична підтримка, яку забезпечує OWL у порівнянні з RDF.

Також однією з найбільш важливих можливостей, що надає OWL є саме можливість вказати, що два поняття є різними чи однаковими, що дозволяє об’єднувати дані з різних джерел (Linked Data).

RDF не забороняє створювати беззмістовних тверджень або тверджень, що не узгоджуються з іншими. Тому, у разі викоритсання RDF відсутні гарнтії цілісності та несуперечливості RDF – описів [1]. Викоритсання OWL якраз і є вирішенням цієї проблеми. Якщо RDF визначає, як описувати поняття, то OWL визначає, що саме потрібно описувати, щоб отримати логічно правильну онтологію.

Отже, використання OWL для створення онтологій надає додаткову семантичну підтримку онтології за допомогою розширених можливостей опису класів та властивостей даної онтології у порівнянні з можливостями, наданими RDF/RDFS.

1. Онтології та їх реалізація
   1. Поняття онтології

Онтологія – це система, що складається з набору понять та набору тверджень про ці поняття, на основі яких можна описувати класи, відносини, функції та індивіди [1].

Онтології використовуються для збереження знань з деякої предметної області, це фактично представлення деякою мовою знань про певну предметну область. Вони описують поняття предметної області, а також відношення, що існують між цими поняттями.

Онтології, в загальному вигляді, визначаються як спільно використовувані формальні концепції (системи вихідних теоретичних положень) конкретних предметних областей, інформацією з яких можуть обмінюватися люди і додатки. Вони дозволяють концептуалізувати (визначити базові обєкти та відношення між ними) певну предметну область фіксуванням її сутностей (entities) і зв'язків .

Онтології грунтуються на математичному апараті формальної логіки (descriptive logic), малу підмножину якого охоплює RDF-схема. DL є підмножиною логіки першого порядку, яка є обчислюваною.

Використання онтологій – це перспективний підхід вирішення проблеми семантичної гетерогенності (неоднорідності), що розглядає зміст інформаційних елементів.

ОWL - онтологія складається з екземплярів (індивідів), класів (понять), атрибутів та відношень.

Екземпляри (індивіди) – основні компоненти онтології. Вони можуть представляти як фізичні, так і абстрактні об’єкти. Основною ціллю онтологій є класифікація таких екземплярів [1].

Класи (концепти) - це абстрактні групи, набори об’єктів [1]. Вони описуються за допомогою формальних (математичних) описів, які точно визначають вимоги для членства в класі. Тобто в OWL класи побудовані з описів, які визначають умови, які повинні бути задоволені екземплярами, щоб бути членом класу. Класи також можуть містити екземпляри та інші класи. Вони можуть бути організовані в ієрархію суперклас-підклас, яка також відома як таксономія.

Однією з ключових особливостей OWL-DL є те, що ці ієрархії можуть бути обчислені автоматично за допомогою системи логічного виводу (reasoner).

Об’єкти онтології можуть мати атрибути. Атрибути мають ім’я та їх значення , що використовуюються для збереження інформації, специфічної для певного об’єкта.

Відношення – це атрибут, значенням якого є інший об’єкт, це фактично бінарні відношення між об’єктами онтології. За допомогою них створюються залежності між об’єктами онтології.

Онтології поділяються на предметно-орієнтовані (спеціалізовані) та загальні.

Предметно-орієнтовані онтології (спеціалізовані) — це представлення якої-небудь галузі знань або частини реального світу. У такій онтології містяться спеціальні для цієї галузі знань терміни.

Загальні онтології використовуються для [подання понять](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0%BD%D0%B0%D0%BD%D1%8C), спільних для великої кількості сфер. Такі онтології містять базовий набір термінів, [глосарій](https://uk.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D1%81%D0%B0%D1%80%D1%96%D0%B9) або [тезаурус](https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%B7%D0%B0%D1%83%D1%80%D1%83%D1%81), використовуваний для опису термінів предметних сфер [1].

Отже, онтологія може служити для представлення в базі знань ієрархії понять та відношень між ними. А онтологія, що містить екземпляри об’єктів і є базою знань.

* 1. Порівняння баз знань та баз даних

База знань – це семантична модель, що описує предметну область і дозволяє відповідати на такі питання з цієї предметної області, відповіді на які явно не присутні в базі знань.

Це основний компонент інтелектуальних систем, систем, заснованих на знаннях, що є підкласом інтелектуальних систем та роблять знання предметної області явними і відокремлюють їх від іншої частини системи, та експертних систем, що у свою чергу є підкласом систем, заснованих на знаннях та застосовують експертні знання для вирішення складних задач реального життя [2].

Вона зберігає людські знання і поміщає їх в комп’ютерні системи, де вони використовуються для вирішення складних експертних завдань за допомогою правил логічного вивдення, вирішення яких вимагають високого рівня людського досвіду [2].

База знань зберігає знання в формі, доступній для читання машинами, зазвичай з метою отримання автоматично згенерованих логічних висновків на базі цих знань. В цілому, база знань не є статичним набором інформації, а динамічним ресурсом, що має спроможність вчитися, набувати нових знань як частина штучного інтелекту експертної системи [14].

Основними характерними рисами баз знань є :

* динамічність (база знань постійно змінюється за допомогою нових знань);
* дані бази знань зібрані з багатьох ресурсів;
* ресурси бази знань постіно змінюються;
* бази знань є значно розумнішими, ніж бази даних, бо вони опрацьовують дані і використовують експертні знання для того, щоб дати відповіді, рекомендації, поради.

Результатом запиту до бази знань є не список ресурсів, а відповідь на поставлене запитання.

Системи, що базуються на даних, а не на знаннях, лише опрацьовують дані, але не видають інформації.

Знання певного предмету — це упевнене розуміння предмету, уміння самостійно поводитися з ним, розбиратися в ньому, а також використовувати для досягнення поставленої мети.

Знання у штучному інтелекті – сукупність спеціалізованих фактів, правил їх обробки, умов застосування правил до конкретних фактів, методів отримання нових фактів і способів організації процесу логічного виводу.

Знання з предметної ділянки називається базою знань.

Головна відмінність знань від [даних](https://uk.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D1%96) полягає в їхній активності, тобто поява в базі нових фактів або встановлення нових зв'язків може стати джерелом змін в ухваленні рішень.

Згідно з даними World Wide Web Consortium (W3C), в майбутньому Інтернет може стати великою і складною глобальною базою знань, яка і є Семантичним Вебом ( Semantic Web) [14].

База даних— це поіменована, структурована сукупність взаємопов'язаних даних, які належать до певної предметної області, під якою прийнято розуміти частину реального світу, яка підлягає вивченню для організації, управління і, в кінцевому результаті, автоматизації. Це впорядкований набір логічно взаємопов'язаних даних, що використовується спільно, та призначений для задоволення інформаційних потреб користувачів. В загальному випадку базою даних можна вважати будь-який впорядкований набір даних.

База даних містить інформацію, необхідну для вирішення цілого комплексу задач даної предметної області. Вона може поповнюватися новими даними, а уже внесені дані можуть бути видалені або зредаговані без внесення змін у прикладні програми.

Перш за все, база даних – це конкретне технологічне рішення для зберігання і організації даних. Вона в основному обмежена лише цими функціональними можливостями. На відміну від бази даних, база знань являє собою технологію зберігання "знань". Для цієї мети база знань може використовувати базу даних, але вона не є обмеженою функціями бази даних. База знань може використовувати безліч баз даних, може комбінувати їх або збагачувати їх інформацією з відкритих інформаційних джерел. Таким чином, база знань є радше доповненням до бази даних, оскільки вона має як ті ж самі функціональні можливості, що й база даних, так і додаткові функції, такі як можливість знаходження рішень певних проблем з використанням уже наявних в ній даних за допомогою можливості логічного введення та посилання на інші інформаційні ресурси.

Бази знань, на відміну від баз даних, є складнішими і вимагають набагато більших обчислювальних можливостей, і по суті представляють собою досягнення штучного інтелекту. У той час як бази даних строго корисні для доступу до інформації щодо потрібної теми, бази знань включають в себе екстраполяцію даних для обробки в «знання». Програма, іншими словами, вчиться "думати". У той час як база даних може безперервно розширюватися за допомогою додаткових бітів інформації або ключових слів, вона ніколи не буде нічого більш, ніж база даних - інструмент для доступу до інформації, яка була введена в неї. Система, заснована на знаннях, тим не менш, може виводити додаткову інформацію на основі вже наявної у ній.

Ще однією відмінністю баз знань від баз даних є те, що бази даних містять зв'язки між об'єктами в явній формі, тоді як бази знань можуть виводити інформацію, яка не присутня у ній в явній формі за допомогою можливості логічного виведення.

Отже, основною відмінність між базами даних і базами знань є те, що бази знань потребують семантичної теорії для пояснення свого контенту, в той час як бази даних потребують обчислювальної теорії для їхньої ефективної ( раціональної ) реалізації ( імплементації) на фізичних машинах. Бази знань є сховищем знань у формі, доступні для машинного читання, з метою отримання нових знань на базі уже існуючих. Це «експертна системи», що використовує штучний інтелект поряд із даними, що в ній знаходяться, з метою надання відповідей на поставлені запитання, а не простого списку інформаційних джерел (ресурсів). В той час як база даних є великою, складною структурою даних, яка дає можливість ефективного збереження та пошуку даних, в чому і полягає її основна мета.

* 1. Загальний опис редактора онтологій Protégé

Редактор онтологій – це інструментальний програмний компонент, створений для проектування, редагування та аналізу онтологій [1].

Основною функцією редактора онтологій є підтримка процесу формалізації знань і подання онтології як специфікації (точного і повного опису) [1].

Редактори онтологій зазвичай підтримують роботу з кількома формалізмами, теоретичний базис, що лежить в основі способу представлення онтологічних знань (логіка предикатів першого порядку, дескриптивна логіка ), і форматами (вид зберігання та спосіб передачі онтологічних описів), але часто тільки один формалізм є "рідним" (native) для даного редактора.

Основний набір функцій редактора онтологій:

* робота з одним чи декількома проектами;
* збереження проекту в потрібному форматі;
* відкриття проекту;
* імпорт з зовнішнього файлу;
* редагування метаданих проекту (від налаштування форм редагування та подання даних до підтримки версій проекту);
* редагування онтології (створення, редагування, видалення понять, відношень, аксіом та інших структурних елементів онтології, редагування таксономії) [1].

До додаткових можливостей редакторів відносять підтримку мови запитів (для пошуку нетривіальних тверджень), аналіз цілісності, використання механізму логічного виведення, підтримку багатокористувацького режиму, підтримку віддаленого доступу через Інтернет [1].

Одними з найбільш розповсюджених редакторів онтологій є Protégé, NeOnToolkit, CmapsEditor, HozoEditor.

Дана робота присвячена опису можливостей одного з цих редакторів, а саме редактора онтологій Protégé. Це безплатний, з відкритим вихідним кодом редактор онтологій і фреймворк для побудови інтелектуальних систем [15].

Це вільно розповсюджена Java-програма, призначена для побудови (створення, редагування і перегляду) онтологій певної предметної області. Вона включає редактор онтологій, що дозволяє проектувати онтології, розгортаючи ієрархічну структуру абстрактних і конкретних класів і властивостей.

Даний інструмент підтримує використання мови OWL і дозволяє генерувати HTML-документи, що відображають структуру онтологій. Оскільки він використовує фреймову модель подання знань ОКВС, це дозволяє адаптувати його і для редагування моделей предметних областей, представлених не в OWL, а в інших форматах (UML, XML, SHOE, DAML + OIL, RDF / RDFS і т.д.).

З моменту його створення Protégé багато років використовувався експертами в основному для концептуального моделювання в області медицини. Останнім часом його стали застосовувати в інших предметних областях - зокрема, при створенні онтологій для Semantic Web.

Інтерфейс складається з головного меню і декількох вкладок для редагування різних частин бази знань і її структури. Набір і назви вкладок залежать від типу проекту (мови представлення) і можуть бути налаштовані вручну. Звичайно, є такі основні вкладки: Класи (Classes), Властивості – об’єкти (Object Properties), Властивості – значення (Data Properties), Екземпляри (Individuals) та Запити (DL Query).

Призначення основних вкладок - надати можливості заповнення бази знань.

Практична розробка онтології включає:

* + визначення класів в онтології;
  + розташування класів у таксономічну ієрархію (підклас - надклас);
  + визначення властивостей і опис їх допустимих значень.

Після цього можна створити базу знань, визначивши окремі екземпляри цих класів та надавши значення властивостям цих екземплярів.

Фундаментальні правила розробки онтології, що сприяють прийняттю правильних проектних рішень при розробці онтології:

* + не існує єдино правильного способу моделювання предметної області - завжди існують життєздатні альтернативи.
  + розробка онтології - це обов'язково ітеративний процес.
  + поняття в онтології повинні бути близькі до фізичних або логічних об’єктів і відношень предметної області.

Найбільший вплив на прийняття рішень щодо моделювання онтології має призначення онтології та знання того, наскільки детальною або загальною вона повинна бути.

3. Створення онтологій за допомогою системи Protégé

* 1. Створення властивостей

3.1.1Типи властивостей

OWL властивості являють собою відносини. Є два основних типи властивостей: властивості об'єктів (властивсоті – об'єкти) і властивості типів (властивості - значення).

Властивості об'єктів позначають взаємозв'язки між двома екземплярами, вони пов’язують два екземпляри.

Вони можуть бути створені в Protégé, використовуючи «Object Properies view» у вкладці «Object Properties tab». Для створення нової властивості об’єкта потрібно скористатися кнопкою «Add Property» (рисунок 3.1).

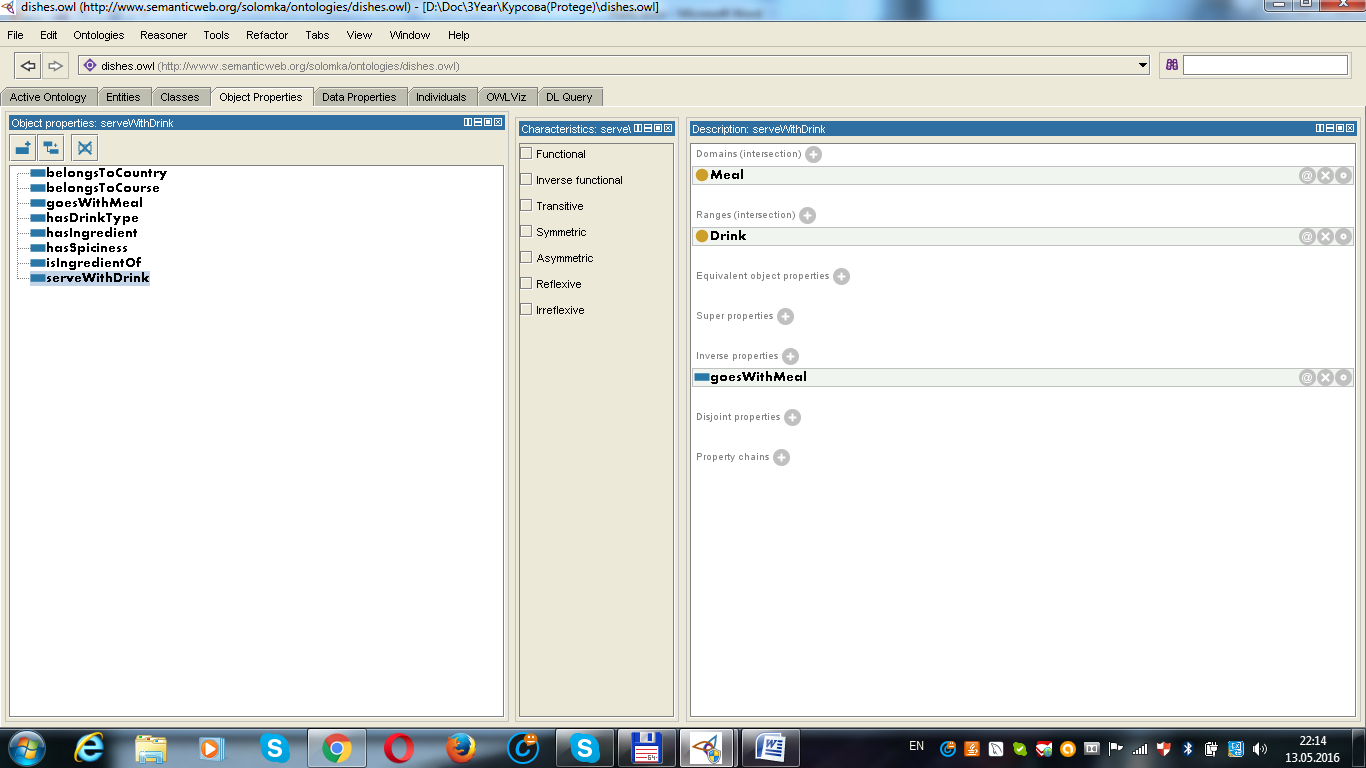
В OWL властивості можуть мати підвластивості, так що можна сформувати ієрархію властивостей. Підвластивості спеціалізують свої супервластивості (так само як підкласи спеціалізують свої суперкласи). Наприклад, властивість «hasMother» може спеціалізувати більш загальну властивість «hasParent».

Кожна властивість об'єкта може мати відповідну зворотню властивість. Якщо якась властивість повязує екземпляр а з екземпляром b, тоді її зворотня властивість повязуватиму екземпляр b з екземпляром a.

Наприклад, властивість «goesWithMeal» є зворотною до властивості «serveWithDrink» у онтології dishes.owl .

Зворотна властивість може бути створена у Protégé, використовуючи inverse properties view (рисунок 3.1).

Рисунок 3.1– Вкладка «Object Properties tab» (зворотна властивість)

Властивості типів даних зв'язають екземпляри з XML Schema типами даних або rdf - літералами. Іншими словами, вони описують відносини між екземпляром і значенням даних, яке можу мати тип або бути безтиповим.

Вони можуть бути створені в Protégé, використовуючи вкладку «Datatype Properties view» у «Data Properties tab». Для стоврення нової типової властивості потрібно скористатися кнопкою «Add Property» (рисунок 3.2) .

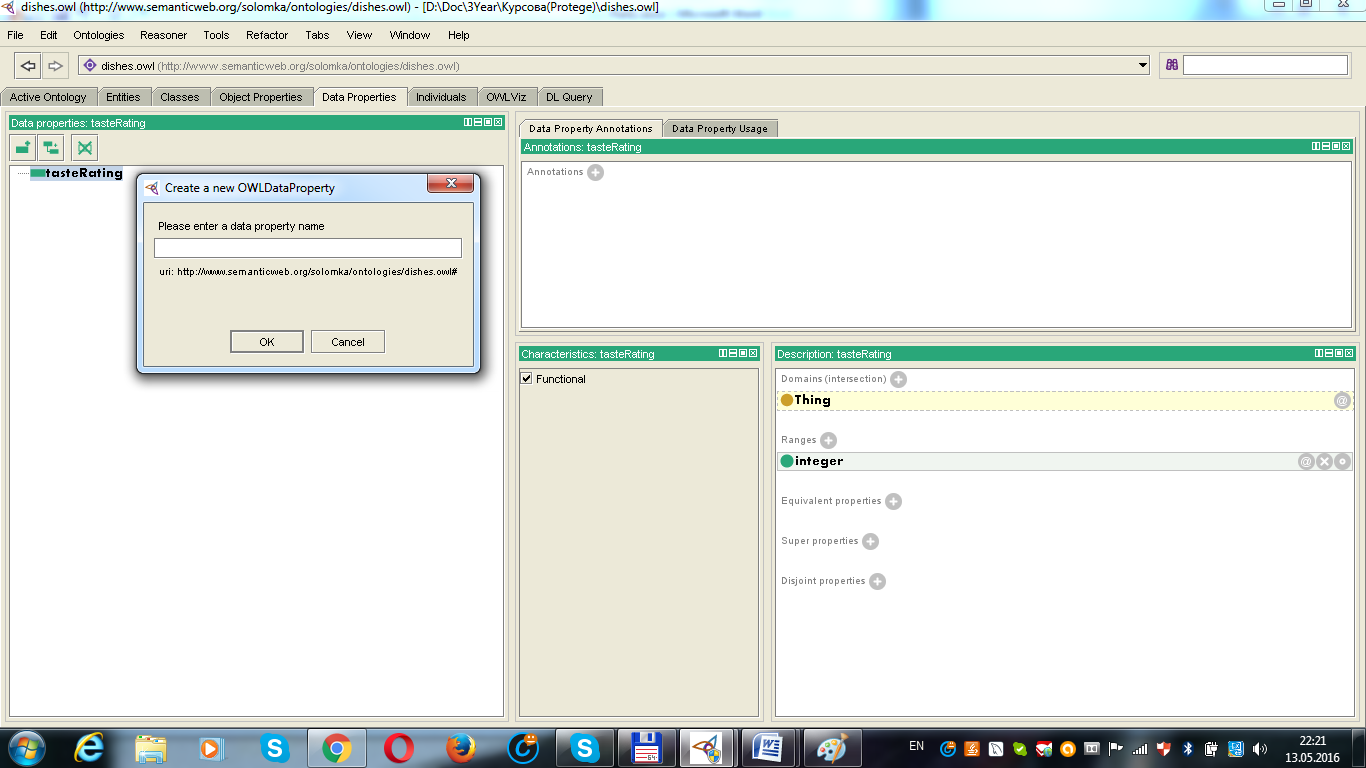


Рисунок 3.2– Створення властивості типів у «Data Properties tab»

Властивості типів даних також можуть бути використані в обмеженнях на властивості, щоб віднести екземпляри до членів даного типу даних.

Вбудовані типи даних вказані в XML Schema словнику і включають в себе цілі числа, раціональні числа, стрічки та логічний тип даних.

Окрім використання попередньо визначеного набору типів даних, ми можемо додатково спеціалізувати використання типу даних за допомогою

визнчення обмежень на можливі значення. Наприклад, можна легко визначити діапазон значень для числового типу даних.

OWL також має третій тип власності – анотації. Вони використовуються для додавання інформації (метаданах - даних про дані) до класів, окремих екземплярів і властивостей.

3.1.2 Характеристики властивостей

ОWL надає можливість збагатити значення властивостей за допомогою використання характеристик цих властивостей.

OWL підтримує такі характеристики властивостей: функціональні (functional), зворотні функціональні (inverse functional), транзитивні (transitive), симетричні (symmetric), асиметричні (asymmetric), рефлексивні (refflexive), іррефлексивні (irrefflexive) (рисунок 3.1).

Якщо властивість є функціональною, то для даного екземпляра може бути щонайбільше один екземпляр, з яким даний екземпляр є пов'язаний за допомогою цієї властивості. Наприклад, властивість «belongsToCourse» є функціональною, тобто кожен екземпляр класу «Meal» може бути пов'язаний лише з одним екземпляром класу «Сourse» за допомогою цієї властивості.

Якщо властивість є зворотною функціональною, то це означає, що зворотна властивість до даної є також функціональною.

Якщо властивість є транзитивною і вона пов’язує екземпляр a з екземпляром b і екземпляр b з екземпляром c, то ми можемо зробити висновок, що екземпляр a є також пов'язаний з екземпляром c за допомогою цієї властивості.

Якщо властивість є транзитивною, то її зворотна властивість також повинна бути транзитивною. Більше того, якщо властивість є транзитивною, то вона не може бути функціональною.

Якщо властивість є симетричною і вона пов’язує екземпляр a з екземпляром b, то ми можемо зробити висновок, що екземпляр b також пов'язаний з екземпляром a за допомогою цієї властивості.

Якщо властивість є асиметричною і вона пов’язує екземпляр a з екземпляром b, тоді екземпляр b не може бути пов'язаний з екземпляром a за допомогою цієї властивості.

Якщо властивість є рефлексивною, то вона пов’язує екземпляра класу з самим собою. Наприклад, якщо ми маємо екземпляра класу «Людина» Андрій і властивість «поважає». Тоді Андрій пов’язує самого себе, так само як Андрій може поважати і інших екземплярів класу «Людина». Таким чином властивість «поважає» є рефлексивною.

Якщо ж властивість є іррефлексивною, То вона може бути описана як властивість, що пов’язує два екземпляри між собою, проте лише за умови, що ці екземпляри є різними. Наприклад, якщо у нас є два екземпляри класу «Людина» і властивість «єСестрою». Тоді ця властивість є іррефлексивною, оскільки вона може пов’язувати між собою лише різні екземпляри класу.

Вказавши, що властивість об’єкта є функціональною, ми стверджували, що будь-який екземпляр можу бути пов'язаний цією властивості щонайбільше з одним іншим екземпляром. Ми також можемо використовувати функціональну характеристику і для властивостей типів даних, що є єдиною характеристикою властивостей цього типу.

Також властивості можуть мати область визначення (domain) та область значення (range) (рисунок 3.1). Властивості повязують екземпялри з області визначення з екземплярами з області значення. Наприклад, для властивості «serveWithDrink» областю визначення є екземпляри класу «Meal», а областю значення є екземпляри класу «Drink».

До області значень (range) може входити декілька класів у Protégé, але тоді областю значень буде вважаться перетин цих класів.

Якщо для властивості, що має зворотню властивість, було визначено її область визначення та область значення, то Protégé автоматично визначить область визначення і область значення для її зворотної властивості. Тоді областю визначення для зворотної властивості буде область значення початкової властивості, а область значення зворотної властивості буде область визначення початкової властивості.

3.1.3 Обмеження на властивості

Властивості можна використовувати для опису та визначення класів за допомогою обмежень, накладених на них.

Як було уже зазначено, властивості описують бінарні відношення, пов’язують екземпляр різних класів між собою. Основною ідеєю використання обмежень на властивості є те, що клас екземплярів може бути описаний або визначений за допомогою відношень, в яких беруть участь екземпляри цих класів. Саме такі класи й описують за допомогою обмежень, що накладаються на відношення (властивості). Іншими словами, обмеження є певним типом класів так само як іменовані класи є певним типом класів [3].

Обмеження описують анонімні класи. Анонімний клас містить усі екземпляри, що задовольняють дані обмеження, тобто всі екземпляри, що мають зв’язки, необхідні для членства у цьому класі. Коли ми описуємо іменований клас, використовуючи обмеження, ми нас правіж описуємо анонімний суперклас іменованого класу [3].

Обмеження використовують для опису OWL – класів, щоб визначити анонімні суперкласи класів, що описуються.

В OWL обмеження на властивості поділяють на 3 основні категорії :

* Кванторні обмеження (Quantifier restrictions), що поділяють на обмеження існування (existential restrictions) та універсальні (загальні) обмеження (universal restrictions);
* Обмеження кардинальності;
* hasValue обмеження.

3.1.3.1 Кванторні обмеження (Quantifier Restrictions)

Обмеження існування описують клас екземплярів що беруть участь щонайменше в одному зв’язку по певній властивості з екземплярами, що є членами певного класу. Напркилад, «клас екземплярів, що мають принаймні один hasIngredient зв'язок з екземплярами класу Sweet». В Protégé 4.0 використовується ключове слово «some» для вираження обмеження існування.

В Protégé 4.0 обмеження для класу є відображені і можуть бути додані чи зредаговані використовуючи «Class Description View», що містить всю інформацію, що є використана для опису класу.

Найбільш поширеним типом обмежень в OWL – онтологіях є саме обмежені існування (Existential restriction/ Some Restrictions/ some values from restrictions).

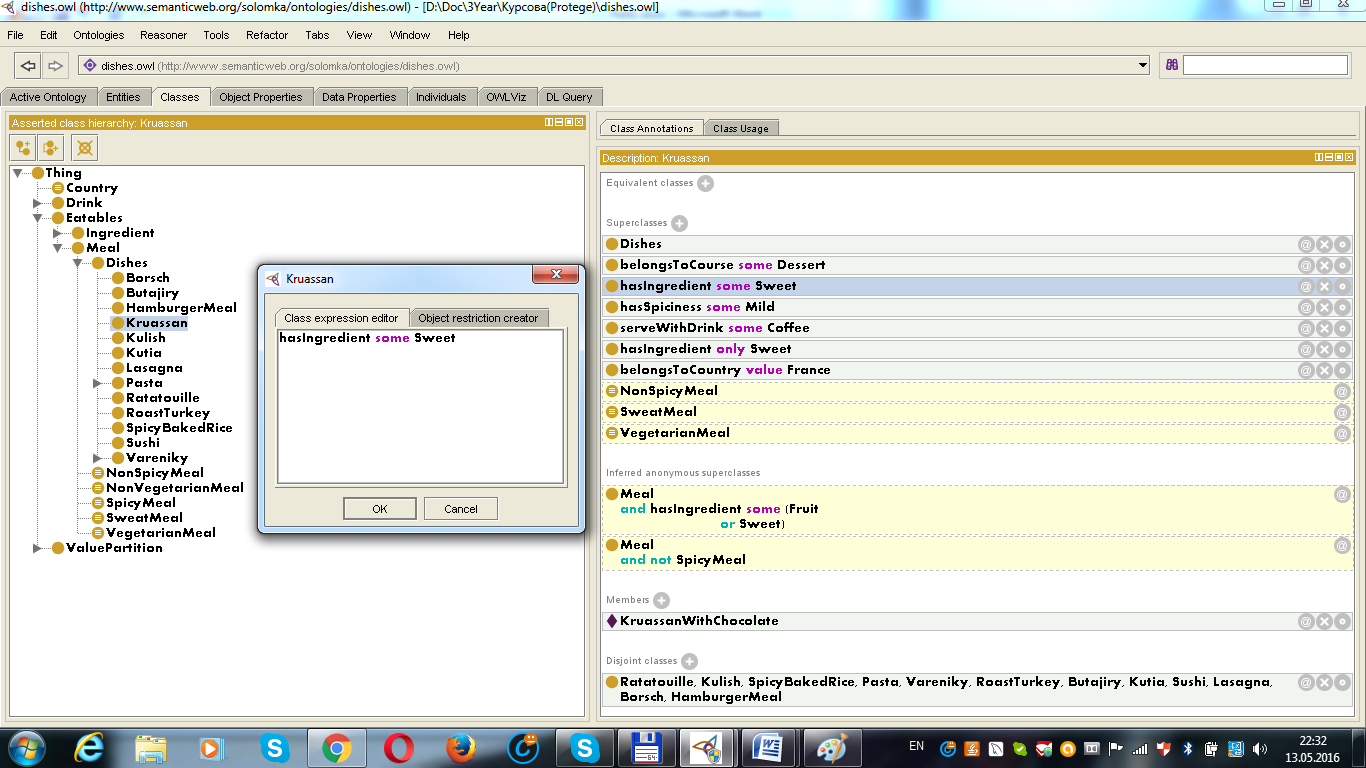
Щоб створити обмеження в Protégé 4.0, потрібно у «Class Description View» натиснути кнопку «Add» біля «Superclasses» заголовка. Після цього буде відкрито текстове поле, де можна ввести саме обмеження. Це текстове поле дозволяє створювати обмеження, використовуючи класи, властивості та екземпляри. Для створення обмеження потрібно спочатку ввести або вибрати властивість, на яку буде накладено обмеження, потім ввести тип обмеження («some» або «only»), після чого ввести назву класів, якими буде обмежена властивість (рисунок 3.3).

Рисунок 3.3 – Створення кванторних обмежень

Зазначимо, що кванторні обмеження визначають необхідні умови. Наприклад, обмеження «клас екземплярів, що мають принаймні один «hasIngredient» зв'язок з екземплярами класу Sweet» визначає таку необхідну умову : «якщо щось є екземпляром класу «Kruassan», то для нього є необхідним мати принаймні один «hasIngredient» зв'язок з екземплярами класу «Sweet».

Клас, що має лише необхідні умови (necessary conditions) називається примітивним класом (Primitive Class).

Проте, використовуючи лише необхідні умови, ми не можемо сказати: «якщо певний екземпляр задовольняє ці умови, то він повинен бути екземпляром даного класу».

Для цього нам необхідно змінити необхідні умови на необхідні та достатні умови (necessary AND sufficient conditions). Це означатиме, що умови, необхідні для членства у класі, є також достатні, щоб визначити, що будь-який екземпляр, що задовольняє ці умови, повинен бути членом даного класу.

Клас, що має щонайменше один набір необхідних та достатніх умов, називається визначеним класом (Defined Class). Вони мають визначення, і будь-який екземпляр, що задовольняє це визначення, є членом даного класу.

Необхідні умови в Protégé 4.0 названі «Superclasses», а необхідні та достатні умови названі «Equivalent classes».

Також у Protégé 4.0 визначені класи позначені як жовтий кружечок з трьома білими горизонтальними лініями, а примітивні класи позначені просто порожнім жовтим кружечком.

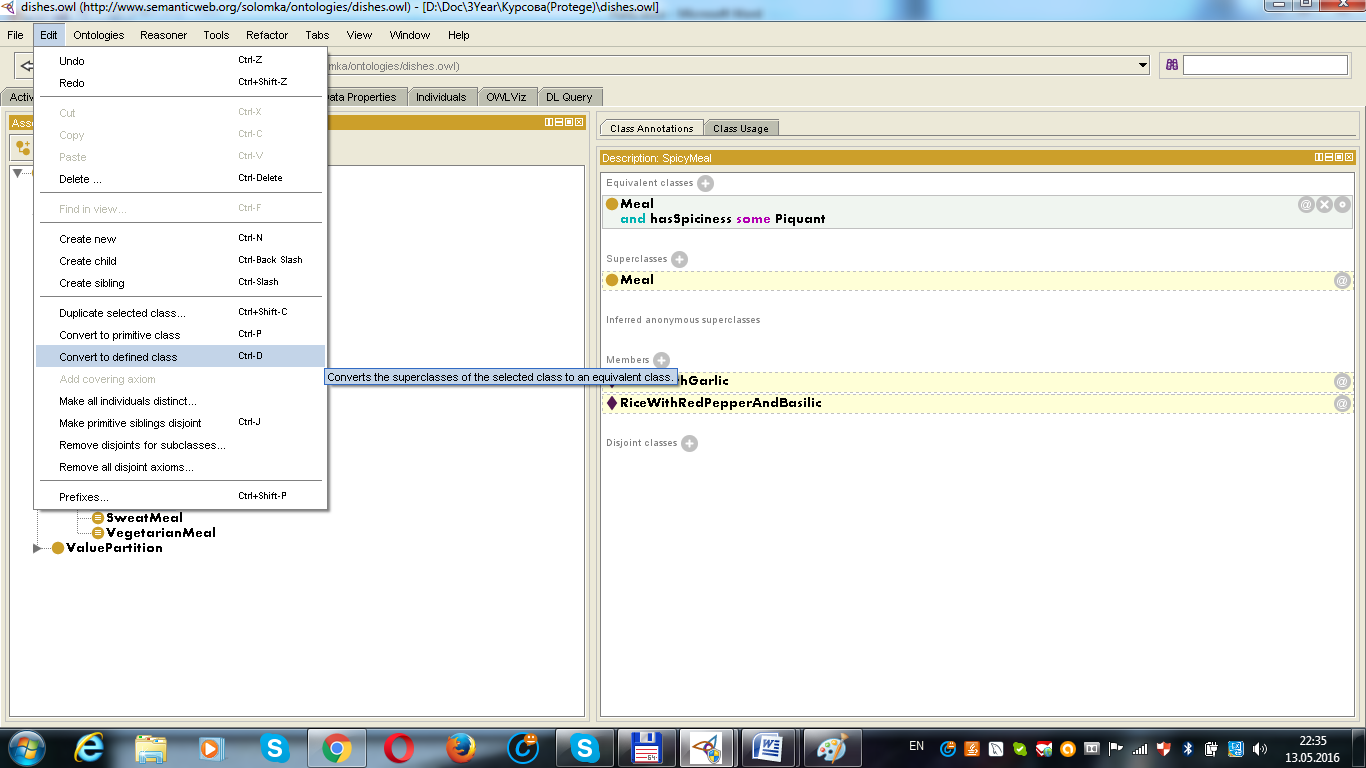
Щоб перетворити необхідні умови в необхідні та достатні умови в Protégé 4.0, умови повинні бути перенесені з «Superclasses» заголовка до «Equivalent classes» заголовка в «Сlass Description view». Це може бути здійснено за допомогою опції «Convert to defined class» в меню «Edit» (рисунок 3.4).

Рисунок 3.4 – Перетворення необхідних умов в необхідні та достатні умови в Protégé 4.0

Таким чином ми перетворюємо опис певного класу у визначення цього класу. Це перетворення є важливим для здійснення автоматичного визначення класової ієрархії, що є ключовим завданням мислителя описової логіки (description logic reasoner). Важливо зазначити, що класи ніколи не будуть визначені як підкласи примітивних класів за допомогою мислителя.

Універсальні обмеження описують клас екземплярів, що мають зв’язки по даній властивості лише з екземплярами певного класу. Наприклад, «клас екземплярів, що мають зв’язки по властивості hasIngredient лише з екземплярами класів Macaroni, HotVeggySauce або Vegetable». В Protégé 4.0 використовується ключове слово «only» для вираження універсальних обмежень.

Варто зазначити, що універсальні обмеження також описують екземпляри, що не мають жодного зв’язку по даній властивості з екземплярами іншого класу, оскільки ці екземпляри відповідно не мають жодного зв’язку по даній властивості з екземплярами інших класів, тому вони й задовольняють дані обмеження. Для певної властивості універсальні обмеження не забезпечують існування зв’язку, вони лише стверджують, що якщо такий зв'язок є по даній властивості, то він має бути лише з екземплярами певного класу [3].

Для того, щоб вказати, що екземпляри певного класу мають зв’язки по певній властивості з екземплярами певних класів і лише з екземплярами даних класів і ніяких інших, в Protégé є можливість визначення аксіоми замикання (closure axiom) для певної властивості.

Аксіома замикання для властивості складається з універсального обмеження, накладеного на цю властивість, щоб визначити те, що екземпляри цього класу можуть мати зв’язки по даній властивості лише з екземплярами чітко визначеного списку класів чи одним конкретним класом і ні з якими іншими.

Варто зазначити, що при описі класу з використанням аксіоми замикання використання лише універсального обмеження без обмежень існування є недостатнім, оскільки, як було уже згадано, універсальні обмеження також описують екземпляри, що не мають жодного зв’язку по даній властивості з екземплярами іншого класу [3].

Додавання аксіоми замикання можна здійснювати шляхом додавання простого обмеження для властивості. Проте Protégé 4.0 надає можливість автоматичного створення аксіоми замикання. Для цього потрібно вибрати обмеження на властивість, натиснути праву клавішу миші і вибрати «Create closure axiom» (рисунок 3.5).

Додавання аксіом замикання до опису класів є необхідним для автоматичного визначення класової ієрархії за допомогою використання мислителя.

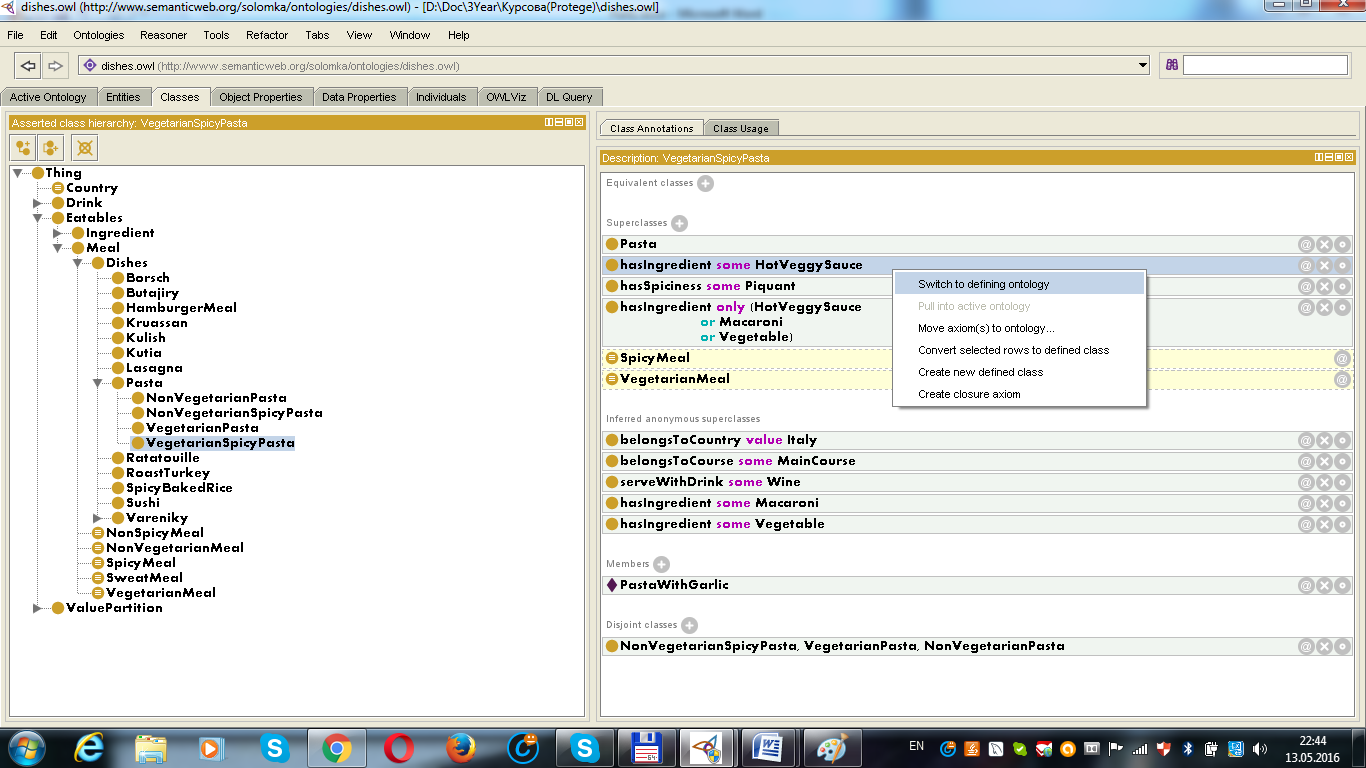


Рисунок 3.5 – Додавання аксіом замикання до опису класів в Protégé 4.0

3.1.3.2 Обмеження кардинальності (Cardinality Restrictions)

Обмеження кардинальності описують OWL – класи, екземпляри яких мають щонайменше/ щонайбільше/ точно певну кількість зв’язків по певній властивості з екземплярами інших класів або значеннями даних [3].

Для певної властивості мінімальне обмеження кардинальності визначає мінімальну кількість зв’язків по цій властивості, в яких повинні брати участь екземпляри даного класу.

Максимальне обмеження кардинальнсті визначає максимальну кількість зв’язків по певній властивості, в яких можуть приймати участь екземпляри даного класу. Відносини (між двома екземплярами) вважається окремим відносинами тільки тоді, коли учасники цих відносин визначені як різні (окремі) екземпляри, тобто вони є екземплярами непересічних класів.

Обмеження кардинальності визначають точну кількість зв’язків по певній властивості, у яких повинні брати участь усі екземпляри певного класу.

Визначені обмеження кардинальності (Qualified Cardinality Restrictions) визначають клас об’єктів, що задовольняють ці обмеження. Для цього після створення новго іменованого класу, потрібно визначити необхідну та достатню умову, що матиме вигляд: «назва властивості» exactly «число, що позначає точну кількість звязків по даній властивості, в яких має приймати участь кожен екземпляр даного класу» назва класу, з екземплярами якого мають бути пов’язані екземпляри даного класу по даній властивості».

3.1.3.3 hasValue обмеження (hasValue Restrictions)

hasValue обмеження описує анонімний клас екземплярів, що мають щонайменше один звязок по певній властивості з певним іншим екземпляром [3]. Наприклад, hasValue обмеження «belongsToCountry value France» описує множину екземплярів, що мають щонайменше один звязок по властивості «belongsToCountry» з екземпляром France (рисунок 3.3). Різниця між hasValue обмеженнями та обмеженнями кардинальності полягає в тому, що обмеження кардинальності описують анонімні класи, екземпляри яких мають зв’язкок по певній властивості з будь-яким екземпляром інших класу.

Варто зазначити, що hasValue обмеження є семантично еквівалентними до обмеження існування, тобто екземпляри анонімного класу можуть мати зв’язки по даній властивості і з іншими екземплярами.

* 1. Створення класів

3.2.1 Іменовані класи (named classes) та непересічні класи (disjoint classes)

Основною складовою частиною OWL – онтології є класи. В Protégé 4.0 стоврення, редагування та видалення класів здійснюється, використовуючи «Classes Tab». Пуста онтологія містить лише один клас «Thing». OWL – класи тлумачаться як набори екземплярів. Клас «Thing» – це клас, що містить умі екземпляри онтології. Саме тому усі інші класи онтології є підкласами класу Thing [3].

Для створення нового класу у Protégé потрібно спочатку вибрати батьківський клас з ієрархії класів , якщо це перший клас онтології, то потрібно вибрати клас Thing, а потім натиснути кнопку «Add subclass» та ввести назву класу (рисунок 3.6).

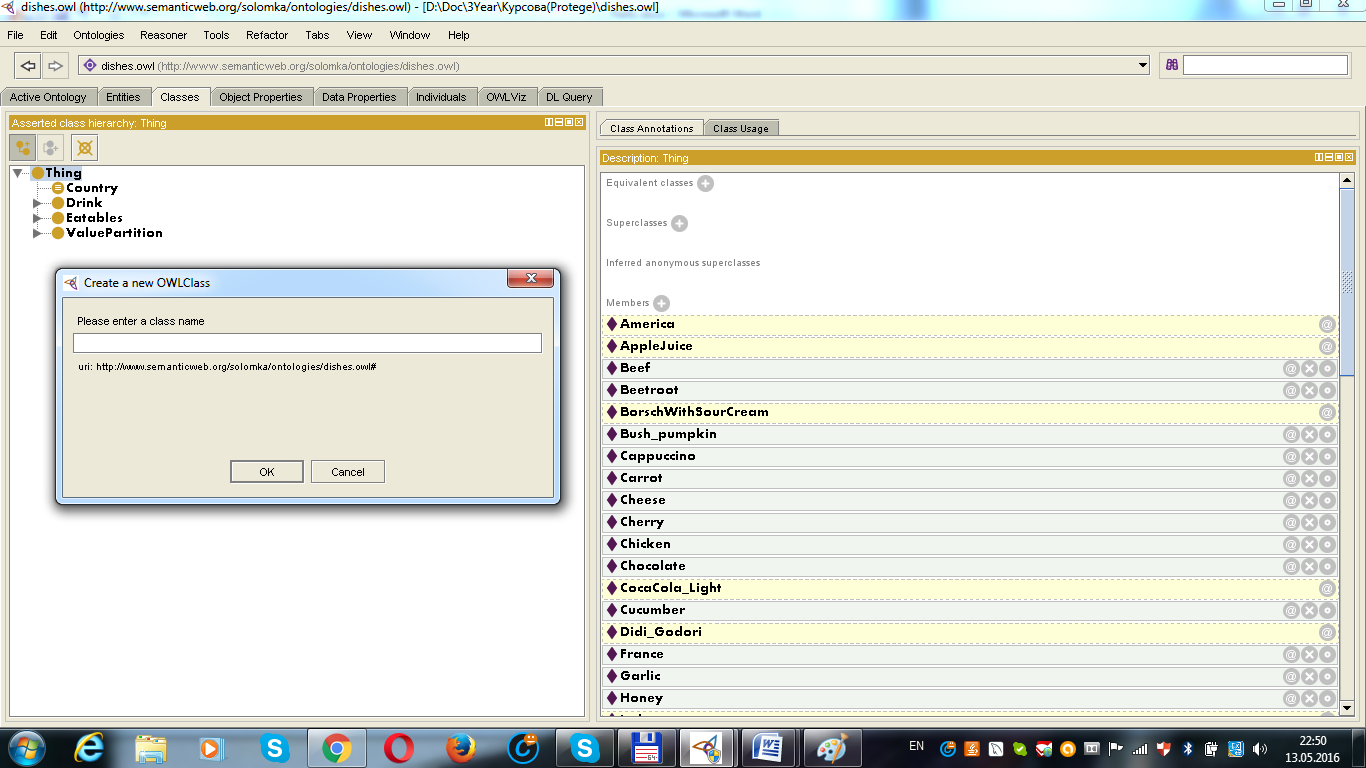


Рисунок 3.6 – Створення нового класу у Protégé 4.0

Protégé 4.0 надає можливість швидкого створення ієрархії класів. У вкладці «Tools» головного меню потрібно вибрати опцію «‘Create Class Hierarchy...». Потім вибрати клас, підкласи якого будуть створені. Тоді у текстовій області вводити назви класів, кожен клас з нового рядка. Для стоврення ієрархії нових клас, класи, що повинні бути підкласами інших кдасів, виділяють за допомогою табулювання. Після введення ієрархії класів потрібно вказати, що класи будуть непересічними, підтвердивши опцію «Make all primitive siblings disjoint».

Непересічність класів (disjoint classes ) означає, що екземпляри (об’єкти) не можуть бути членами більше, аніж одного з цих класів. Щоб визначити класи, що є непересічними з даним класом, потрібно натиснути кнопку «Disjoints classes», що розміщена у нижній частині вікна ‘Class Description’, після чого обрати потрібні класи з класової ієрархії (рисунок 3.3).

3.2.2 Класи - перелічення (Enumerated Classes)

ОWL надає можливість опису класів не лише за допомогою іменованих суперкласів чи анонімних суперкласів, таких як обмеження, а й визначення класів за допомогою явного перелічення всіх екземплярів, що є членами даного класу. Такі класи називається Класи перелічення (Enumerated Classes) [3].

В Protégé 4.0 класи перелічення визначаються за допомогою «Class Description View» редактора тверджень – екземпляри, що визначають клас, мають бути перелічені в фігурних дужках (рисунок 3.7). Проте ці екземпляри спочатку мають бути створені в онтології.

Класи перелічення, визначені в такий спосіб, є анонімними класами, вони є класами лише тих екземплярів, що є перелічені в фігурних дужках. Можна також прикріпити ці екземпляр до іменованого класу, створивши перелічення еквівалентним класом.

Наприклад, створимо клас перелічення «Country», для того щоб, можна було вказати, національною стравою якої країни є дана страва (рисунок 3.7).

Таке визначення класу означає, що екземпляром класу «Country» моє бути лише один з перелічених у списку екземплярів і ніякий інший; клас «Country» є еквівалентний (містить ті самі екземпляри) анонімному класу, що є визначений вичерпним списком країн.

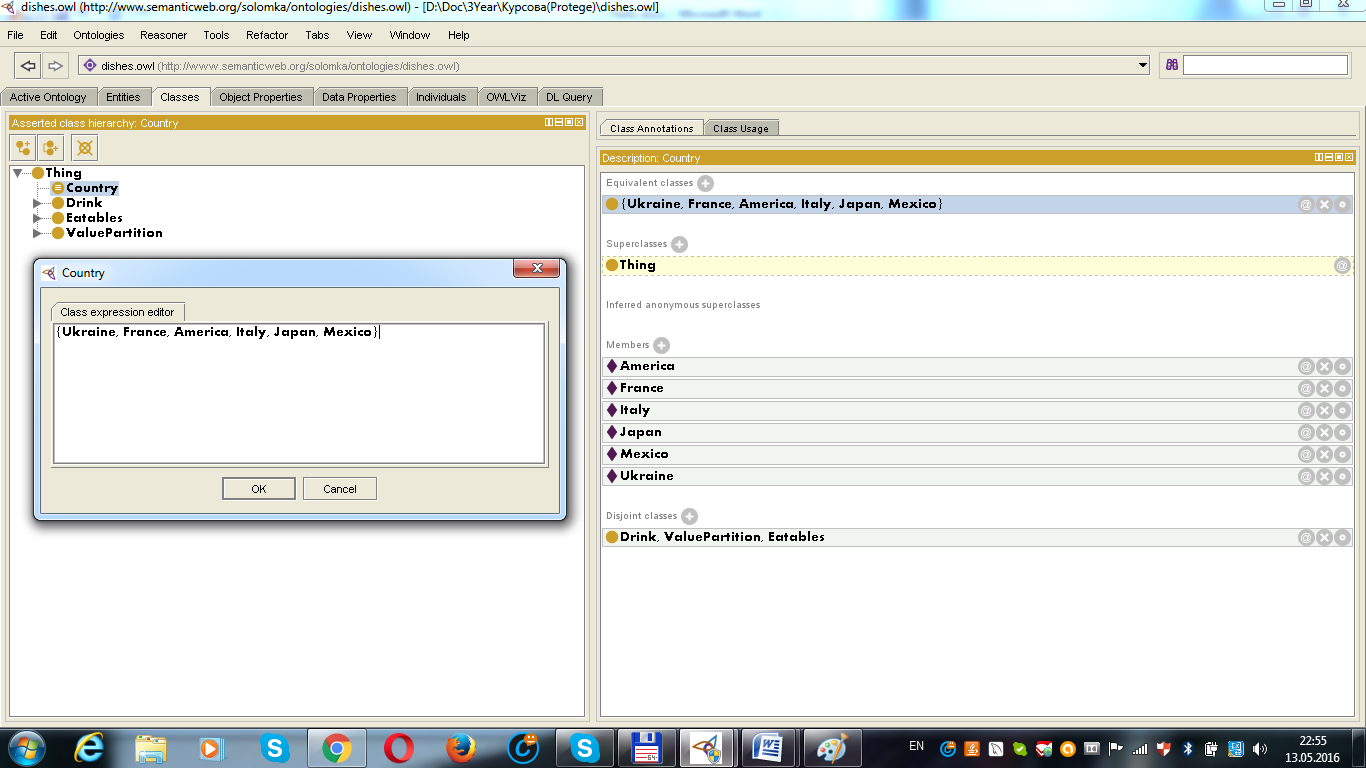


Рисунок 3.7 – Створення класів-перелічень у Protégé 4.0

3.2.3 Класи – доповнення (Complement Classes)

Ще одним типом класів є клас – доповнення (Complement Class) певного класу. До нього належатимуть усі екземпляри, що не є членами класу, для якого даний клас є доповненням.

Зазначимо, що ці два класи: клас – доповнення та клас, доповненням якого є даний клас, мають бути непересічними (disjoint classes), щоб екземпляри одного класу не могли бути одночасно екземплярами іншого класу.

У онтології є клас «SpicyMeal». Створимо клас - доповнення до цього класу «NonSpicyMeal», до якого входитимують усі екземпляри, що не є екземплярами класу SpicyMeal. Для цього після створення класу у класовій ієрархії «NonSpicyMeal» позначимо його несумісним з класом SpicyMeal. Потім створимо обмеження для даного класу у «Class Description View», натиснувши кнопку «Add» біля «Superclasses» заголовка. У текстові області введемо обмеження: « Meal and not SpicyMeal» . Тепер перетворимо необхідну умову у необхідну та достатню умову для визначення класу, що є необхідним для забезпечити можливість автоматичної класифікації онтології. Для цього потрібно обрати у меню «Edit — Convert to defined class» (рисунок 3.4). Після цього клас-доповнення є визначеним (рисунок 3.8).

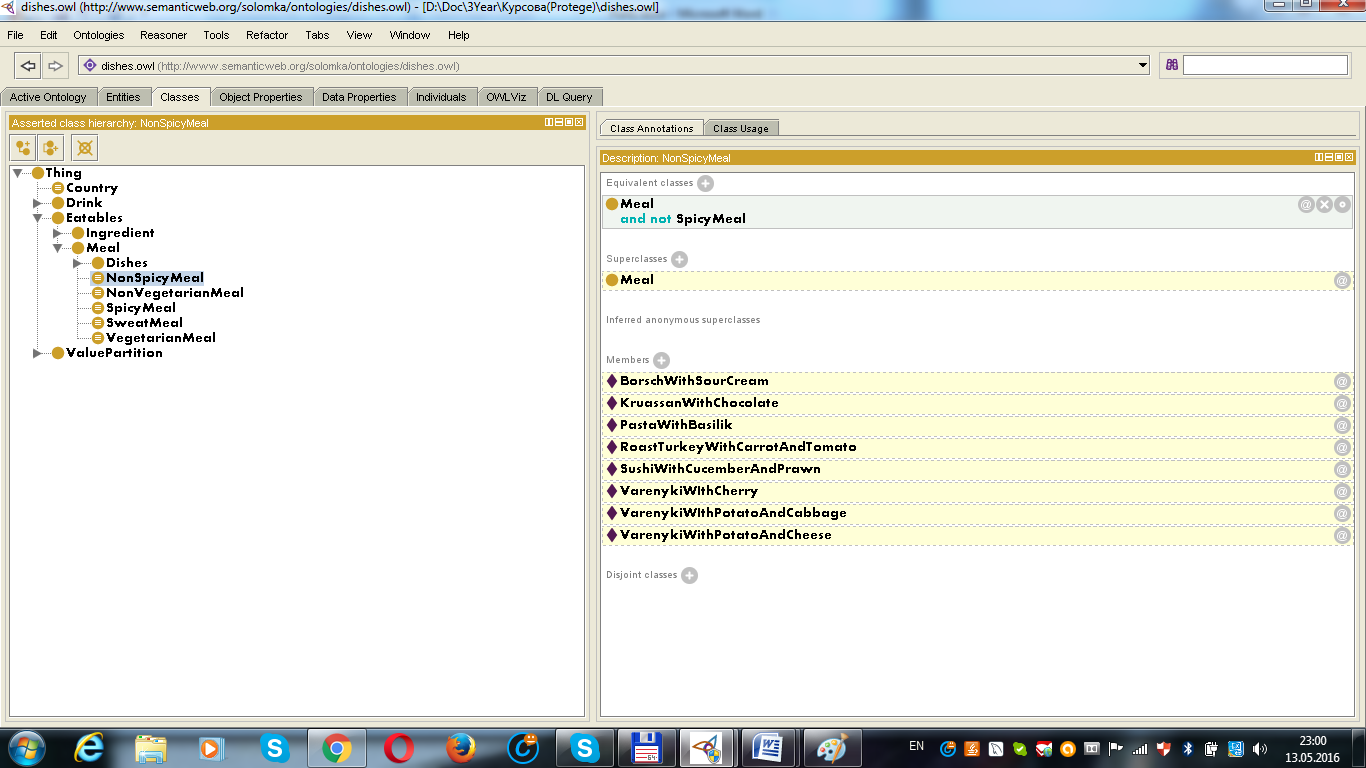


Рисунок 3.8 – Створення класів - доповнень у Protégé 4.0

Отже, OWL – класи у Protégé можуть бути описані одним із наступних способів:

* Опис класів за допомогою іменованих суперкласів;
* Опис класів за допомогою анонімних суперкласів, що є визначені обмеженнями, накладеними на певні властивості;
* Класи - перелічення (Enumerated Classes)
* Класи - доповнення (Сomplement Сlass)

Також OWL надає можливість створення складних описів класів, що складаються з простіших описів класів, об’єднаних разом, використовуючи логічна оператори AND чи OR.

AND – клас, створений за допомогою використання оператора AND є перетином декількох класів, тобто мітить екземпляри, що є одночасно екземплярами декількох класів.

OR – клас, створений за допомогою цього оператор є обєднанням декількох класів, тобто містить екземпляри, що є екземплярами одного з цих класів.

* 1. Шаблон програмування Value Partitions (поділ значення)

Поділ значення (Value Partitions) не є частиною OWL чи іншої онтологічної мови, це шаблон програмування, рішення типових проблем у програмуванні. Цей шаблон застосовується з метою удосконалення опису класу. Value Partitions обмежує діапазон можливих значень вичерпним переліком. Створення Value Partitions в OWL складається з декількох кроків:

* створити класу для представлення ValuePartition;
* створити підкласів класу представлення ValuePartition для визначення можливих значень ValuePartition;
* зробити класи непересічними;
* створити охоплюючу аксіому (covering axiom), щоб зробити список значень вичерпним;
* створити властивість об’єктів для ValuePartition;
* зробити цю властивість функціональною;
* визначити ValuePartition клас областю значень цієї властивості [3].

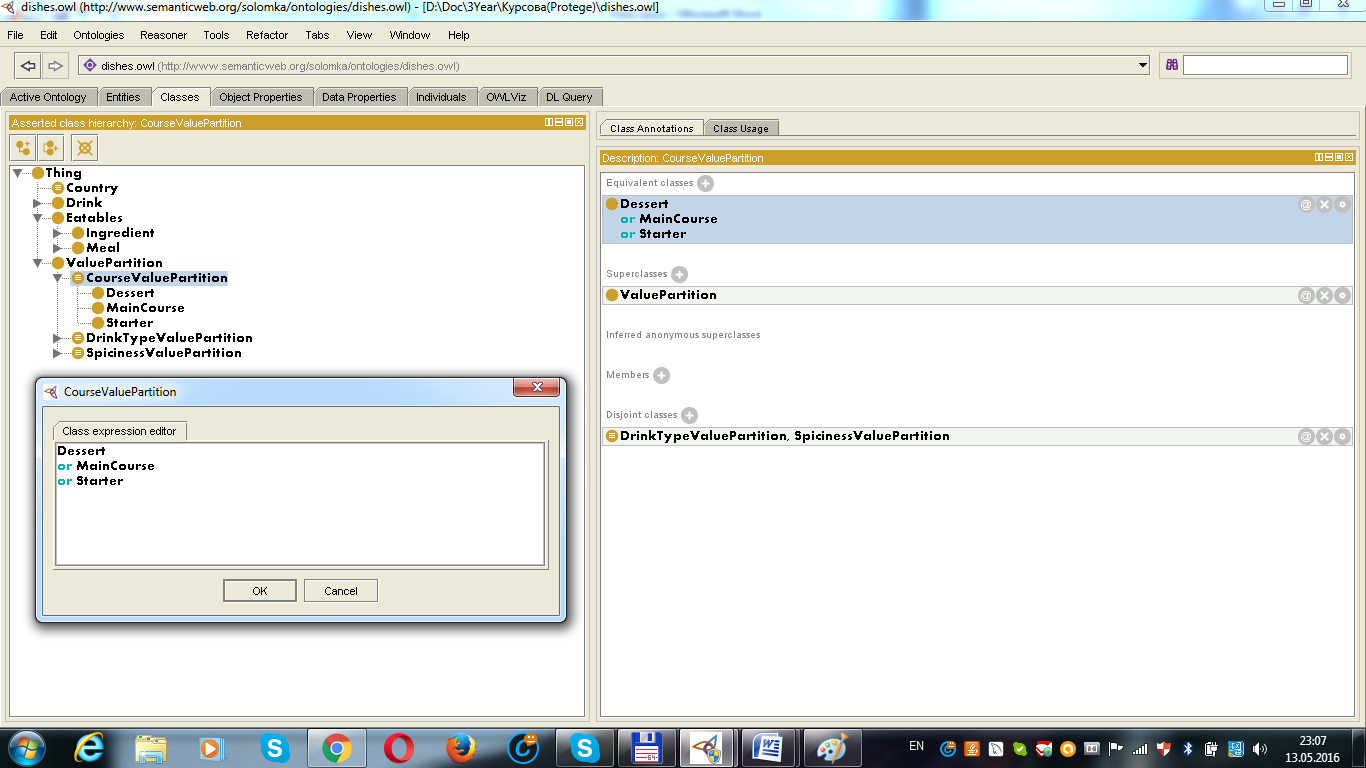
Охоплююча аксіома (covering axiom) є частиною ValuePartition шаблону. Вона складається з 2 частин: клас, який буде «накритий», та клас, який складатиме «накриття».

Нехай є 3 класи: А, В, і С. Класи В і С є підкласами А. Якщо ми маємо охоплюючи аксіому, що стверджує, що клас А є «накритим» класами В і С, то це означає, що екземпляр класу А обов’язково має бути екземпляром класу В і/або екземпляром класу С. Якщо класи В і С є непересічними, тоді екземпляр класу А обов’язково має бути екземпляром класу В або екземпляром класу С. Якщо ж охоплюючи аксіома не визначена, то це означає, що незважаючи на те, що В і С є підкласами А, певний екземпляр може бути членом класу А, не будучи при цьому ні екземпляром класу В, ні екземпляром класу С. В Protégé 4.0 охоплююча аксіома являє собою клас, що є об’єднанням класів, які вона «покриватиме» , яке буде суперкласом класу, що є «накриттям». В такому разі клас А матиме суперклас «В об’єднання С» [3].

Наприклад, нехай ми хочемо віднести кожну страву до певного типу страв (перша страва, друга страва, десерт). Спочатку створимо клас «CourseValuePartition» та його підкласи: «Starter», «MainCourse», «Dessert», визначивши їх непересічними (disjoint). Потім додамо охоплюючу аксіому для класу «CourseValuePartition». Для цього, вибравши «CourseValuePartition» з класової ієрархії, у вікні опису класу біля заголовку «Equivalent classes» натиснемо кнопку «Add» і наберемо « Starter or MainCourse or Dessert». Після цього створимо функціональну властивість об’єктів «belondsToCourse» з областю значень, яку складає клас «CourseValuePartition». Після цього шаблон Value Partitions є реалізованим (рисунок 3.9).

Рисунок 3.9 – Реалізація шаблону Value Partitions у Protégé 4.0

* 1. Використання система логічного виводу (OWL reasoner)



Однією з основних властивостей онтологій, описаних за допомогою OWL-DL діалекту, що є заснована на описовій логіці, є те, що вони можуть бути проаналізовані мислителем (reasoner).

Програмна система (reasoner) забезпечує автоматичне виведення знань з онтології, що випливають із знань, що уже наявні в даній онтології, та забезпечує виконання й інших операцій з онтологіями.

Однією з головних можливостей reasoner є його спроможність визначати, чи є певний клас підкласом іншого класу. На основі цього reasoner може виводити класову ієрархію онтології.

Іншою важливою можливістю reasoner є його здатність перевіряти несуперечливість опису класів. Аналізуючи опис класу (умови приналежності екземпляра до класу), reasoner може перевірити можливість класу мати екземпляри, що до нього належать. Клас вважається суперечливим, якщо він не може мати екземплярів, що до нього належать.

Protégé 4.0 надає можливість підключення різних OWL reasoners, таких як: Pellet, RacerPro, [KAON2](http://kaon2.semanticweb.org/). Reasoner, що початково є встановленим у Protégé, є Fact++, що реалізований на мові програмування С++.

Отже, мислитель може автоматично збудувати класову ієрархію онтології та здійснити перевірку логічної узгодженості онтології.

У Protégé 4.0 стоврена вручну класова ієрархія називається стверджувальною ієрархією «asserted hierarchy», а ієрархія, що автоматично побудована мислителем називається виведеною ієрархією «inferred hierarchy».

Для автоматичної класифікації онтології та перевірки її узгодженості за допомогою використання мислителя у Protégé 4.0 потрібно у вкладці меню вибрати опцію «Reasoner», після чого з випадаючого списку опцій вибрати назву відповідного мислителя та вибрати опцію «Classify...».

Коли класова ієрархію буде побудована, вона з’явиться у новому вікні «Inferred class hierarchy» у лівій частині екрану, поруч з вікном «Аsserted class hierarchy». Якщо певний клас був визначений мислителем як неузгоджений, він буде виділеним у виведеній ієрархії «inferred hierarchy» червоним кольором (рисунок 3.10).

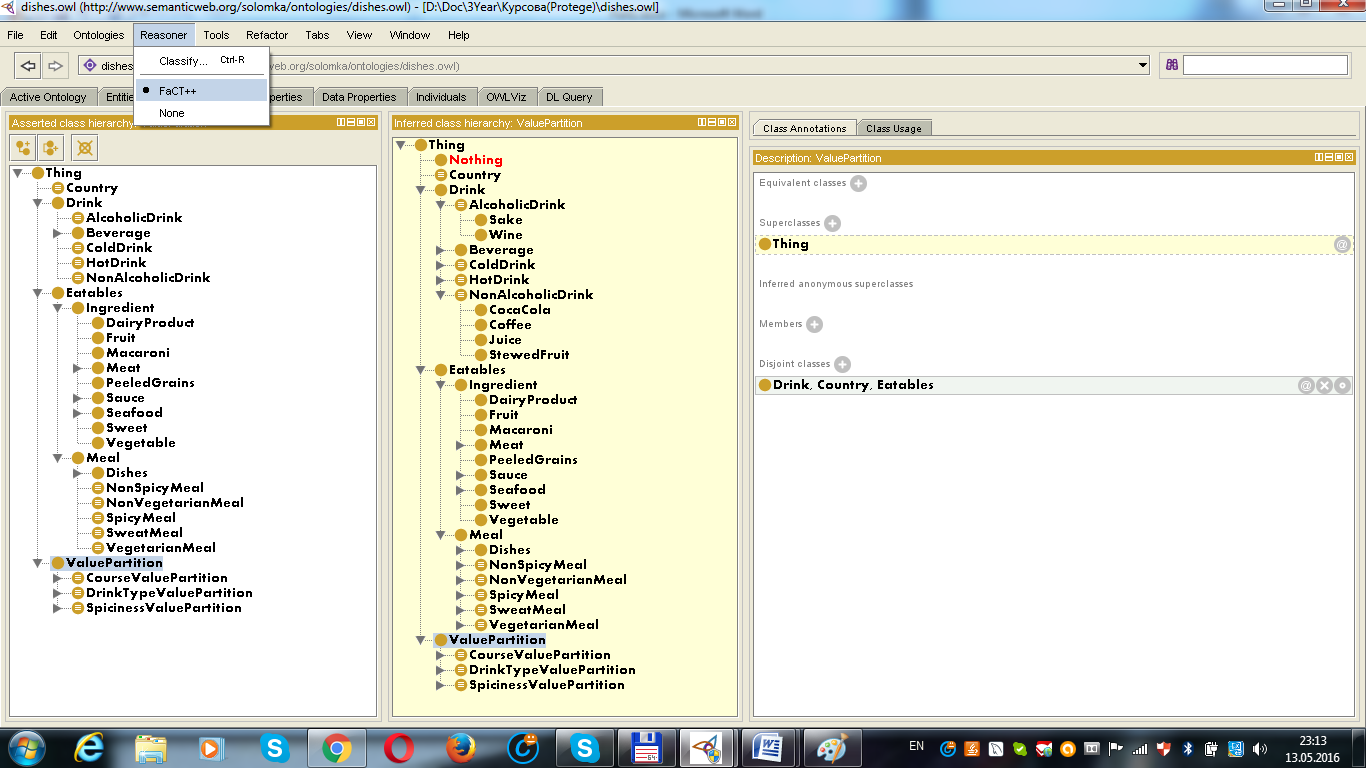


Рисунок 3.10 – Автоматична класифікація онтології та перевірка її узгодженості за допомогою використання системи логічного виводу FaCT++ у Protégé 4.0

Як уже було згадано раніше, додавання аксіом замикання, які стверджують, що екземпляри певного класу мають зв’язки по певній властивості з екземплярами певних класів і лише з екземплярами даних класів і ніяких інших, до опису класів є необхідним для автоматичного визначення класової ієрархії за допомогою використання мислителя. Це пояснюється тим, що логічний вивід в OWL – DL базується на open world assumption (OWA). Це припущення полягає в тому, що ми не можемо стверджувати, що чогось не існує, поки ми явно не визначимо, що цього не існує. Тобто, поки щось явно не визначено, що воно не є істиною, ми не можемо стверджувати, що воно є хибою. У такому разі вважається, що певні знання ще не додані до бази знань [3]. Саме для того, щоб уникнути цієї невизначеності і використовують аксіоми замикання (closure axiom).

* 1. Створення індивідів

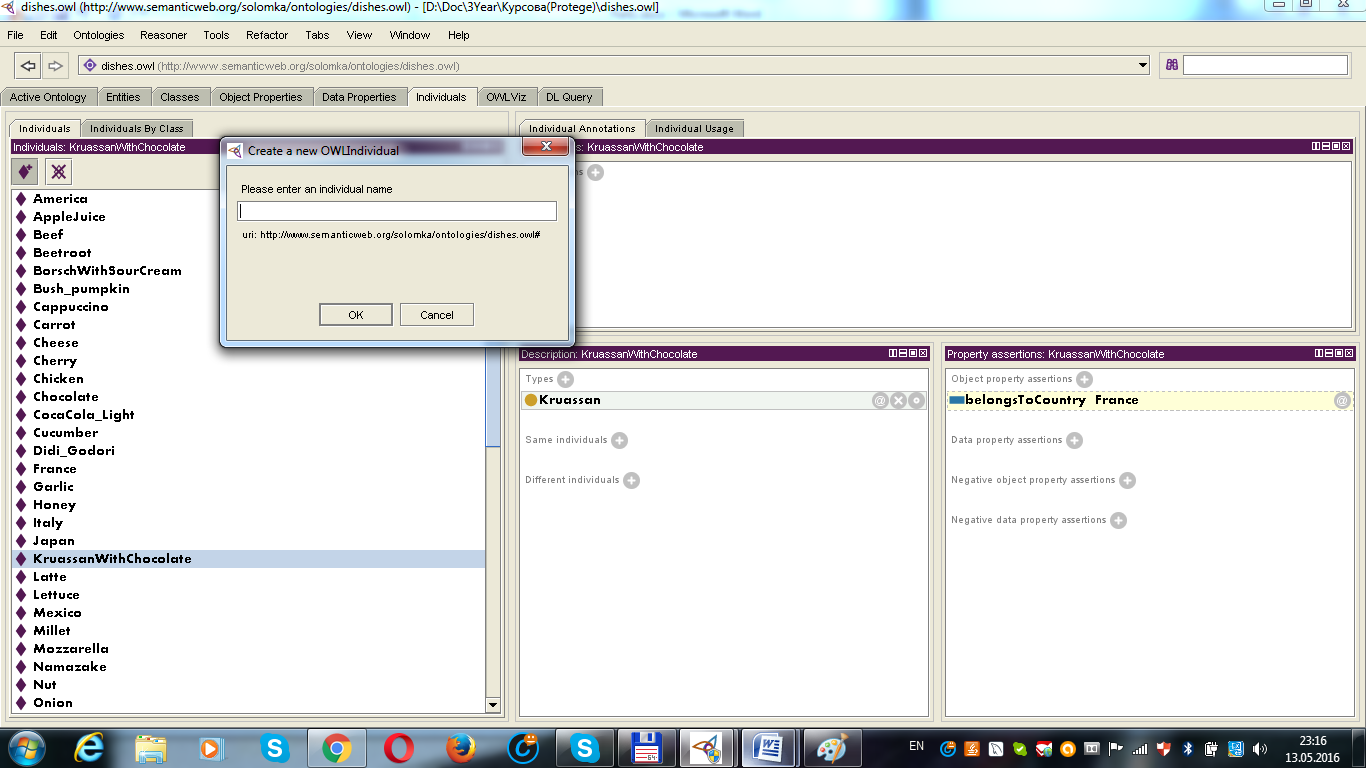
OWL надає можливість створення екземплярів і визначати їх властивості. Також, як уже було вказано раніше, екземпляри можуть бути використані для опису класів, у визначенні hasValue обмежень та створенні класів - перелічень (enumerated classes). Для створення екземплярів та визначення їх властивостей у Protégé 4.0 використовують «Individuals Tab». У цій вкладці для створення нового екземпляра потрібно натисну кнопку «Add individual» і ввести назву нового екземпляра (рисунок 3.11).

Рисунок 3.11 – Створення екземплярів у Protégé 4.0

* 1. Запити до онтології

У Protégé 4.0 для пошуку інформації у класифікованій онтології використовують закладку «DL Query tab» (рисунок 3.12). Вона вбудована в систему Protégé 4.0 Desktop. Мова запитів базується на Manchester OWL синтаксисі. Це зручний синтаксис для OWL - DL, який заснований на зборі всієї інформації про конкретний клас, властивість, або екземпляр в єдину структуру,що називається фреймом [8].

Зазначимо, що пошук інформації за допомогою запитів можна здійснювати лише у систематизованій онтології, тобто онтології, проаналізованій певною програмною системою (reasoner).

Запити можуть бути реалізовані для пошуку окремих екземплярів та класів.

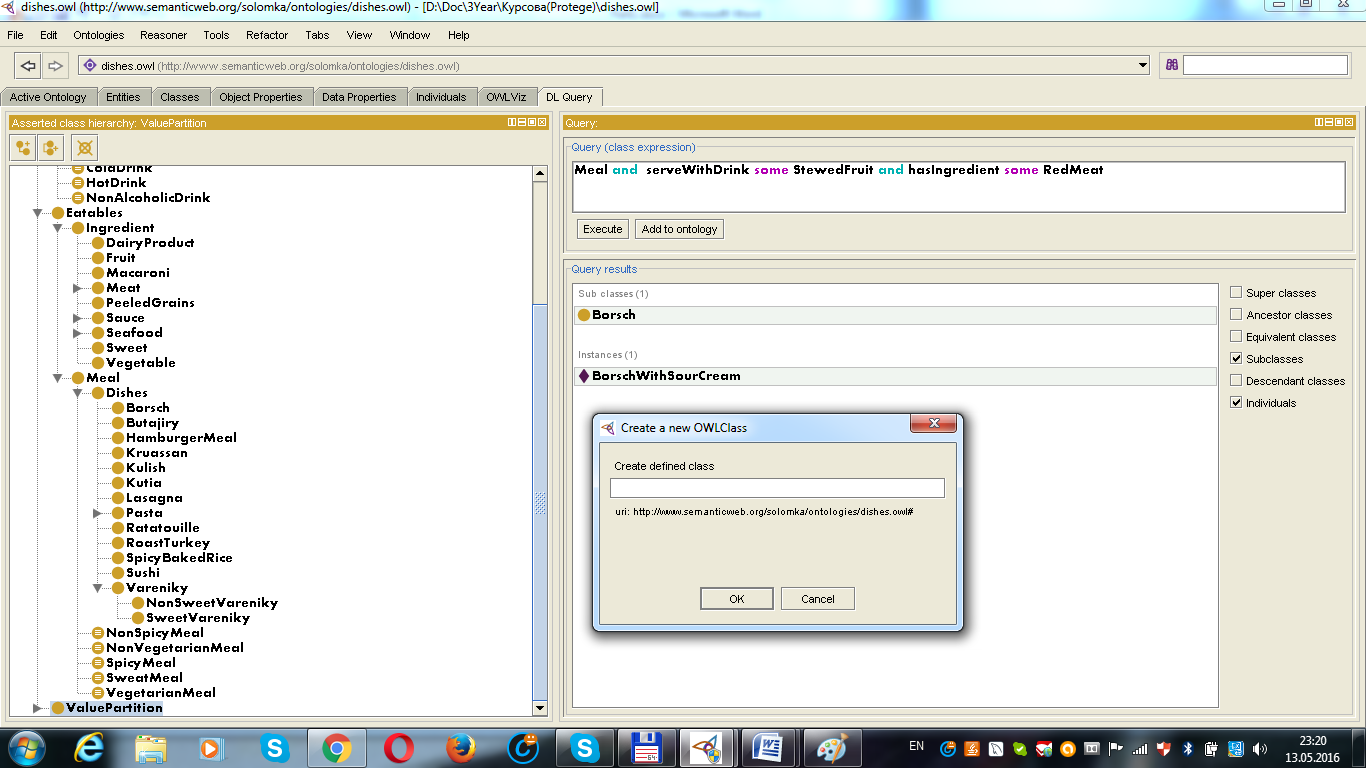
DL Query tab є зручним способом перевірки визначення класу перед його створення. Якщо запит є корисним і є необхідність створення класу з відповідним його визначення, Protégé 4.0 надає таку можливість [8]. Для цього потрібно натиснути кнопку « Add to ontology», після чоговвести назву класу. Виконавши ці дії, в онтології буде створено клас з відповідним його визначенням .

Рисунок 3.12 – Створення запитів до онтології у Protégé 4.0

Отже, редактор онтологій Protégé 4.0 надає широкий спектр можливостей для створення OWL – онтологій у зручний для користувача спосіб з наданням можливості виводу нових знань на базі уже існуючих за допомогою використання програмної системи логічного виводу (reasoner) та можливості пошуку потрібної інформації за допомогою використання «DL Query tab».

Висновки

Отже, проект СВ був розпочатий з метою перетворення інформації, що міститься у Web та є зрозумілою для людей, на інформацію, що є доступною для машинної обробки. Саме для забезпечення цієї цілі і була розроблена формальна мова визначення і подання структурованих веб-онтологій OWL. Вона стала перспективним рішенням проблеми подання усіх знань у Web в єдиному форматі, придатному для сприйняття машинами.

Перетворення ресурсів вебу в глобальну базу знань забезпечує здійснення об’єднання різних видів інформації в єдину структуру, в якій вся інформація подається в чіткому і певному семантичному порядку, що забезпечує надання людині лише тієї інформації, що її справді цікавить.

На основі проведеного порівняння баз даних із базами знань можна зробити висновок, що бази даних, в першу чергу, орієнтовані на здійснення ефективного збереження, організації та пошуку даних, в чому і полягає їх основна мета. База даних ніколи не буде нічим більшим, аніж інструментом доступу до інформації в той час, як бази знань є семантичною моделлю, що надає не лише можливість збереження інформації, а й можливість виводу нової інформацію на основі вже наявної у ній за допомогою використання системи логічного виводу. Саме ця можливість бази знань і забезпечує об’єднання різних ресурсів вебу в одну структуру.

Також, проаналізувавши та практично використавши функціональні можливості редактора онтологій і фреймворку для побудови інтелектуальних систем Protégé у процесі реалізації бази знань, можна стверджувати, що він є зручним та ефективним програмним забезпеченням для створення OWL-онтологій, завдяки зручному інтерфейсу, а також підтримки можливості використання програмної системи логічного виводу, наприклад, системи Fact++, що входить у реалізацію Protégé 4.0, для автоматизованого створення та підтримки класової ієрархії онтології, а також перевірки несуперечливості описів класів OWL - онтології.

Список використаної літератури

1. Глибовець М.М. Інтелектуальні мережі / М.М. Глибовець, А. М. Глибовець, М. В. Поляков. – Дніпропетровськ: ПП Видавництво «Нова ідеологія»,2014. – 462 с.
2. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень / С. О. Субботін. – Запоріжжя: Запорізький національний технічний університет, 2008. – 341 с.
3. Horridge M. A Practical Guide To Building OWL Ontologies Using Prot´eg´e 4 and CO-ODE Tools / M. Horridge. – The University Of Manchester, 2011. – 107 s. – [Електронний ресурс]. – Доступно з: <http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf>
4. OWL Web Ontology Language. – [Електронний ресурс]. – Доступно з: <https://www.w3.org/TR/owl-guide/>
5. <https://www.w3.org/TR/owl-ref/> 4 і 5 посилаання дає один і той самий сайт знайди різницю їх назв

# Extensible Markup Language (XML). – [Електронний ресурс]. – Доступно з:<https://www.w3.org/XML/>

1. Resource Description Framework (RDF). – [Електронний ресурс]. –

Доступно з: <https://www.w3.org/RDF/>

1. DL Query tab. – [Електронний ресурс]. – Доступно з:

<http://protegewiki.stanford.edu/wiki/DLQueryTab>

1. Protege Desktop User Documentation. – [Електронний ресурс]. –

<http://protegewiki.stanford.edu/wiki/Protege4UserDocs>

# How does a reasoner work?– [Електронний ресурс]. – Доступно з: <http://ontogenesis.knowledgeblog.org/1486>

## Ontologies, RDF, and OWL. – [Електронний ресурс]. – Доступно з: http://webdam.inria.fr/Jorge/html/wdmch8.html#x13-1740007.4

1. Michael L. Brodie [On Knowledge Base Management Systems](http://link.springer.com/book/10.1007/978-1-4612-4980-1) / Michael L. Brodie, John Mylopoulos. – [Електронний ресурс]. – Доступно з: http://link.springer.com/chapter/10.1007/978-1-4612-4980-1\_9#page-1

# How does a knowledge base differ from a database? – [Електронний ресурс]. – Доступно з: <http://www.enotes.com/homework-help/how-does-knowledge-base-different-from-database-465999>

# Knowledge base. – [Електронний ресурс]. – Доступно з: (<http://searchcrm.techtarget.com/definition/knowledge-base>)

1. A free, open-source ontology editor and framework for building intelligent systems. – [Електронний ресурс]. – Доступно з: http://protege.stanford.edu/