



UMCS

UNIwersytet Marii Curie-Skłodowskiej
w Lublinie

Wydział Matematyki, Fizyki i Informatyki

Kierunek: **informatyka**

Specjalność: –

Kacper Wójcik

nr albumu: 296538

Analiza i projekt systemu rozwoju ontologii

Analysis and design of the ontology development system

Praca licencjacka
napisana w Katedrze Cyberbezpieczeństwa
pod kierunkiem dra Patryka Burka

Lublin 2022

Spis treści

Wstęp	5
1 Ontologia	9
1.1 Języki	9
1.1.1 Języki związane z ontologiami	9
1.1.2 OWL	10
1.2 Metodologie i metodyki	12
1.2.1 Metodologie wysokiego poziomu	12
1.2.2 Metodologie mikro poziomu	14
1.2.3 Metodyki	15
2 ODE	19
2.1 Czym jest ODE	19
2.2 Zastosowania oraz działanie ODE	20
2.3 Problemy istniejących ODE oraz potencjał rynkowy	22
2.4 Ankieta	22
2.4.1 Analiza wyników ankiety	22
3 Zebranie wymagań oraz projekt aplikacji	25
3.1 FURPS	25
3.2 Projekt aplikacji	26
3.2.1 Funkcjonalność	26
3.2.2 Używalność, wydajność, niezawodność i wsparcie	28
3.2.3 Klasy, oraz przypadki użycia	29
3.2.4 Interfejs	30
4 Implementacja oraz przedstawienie wersji demonstracyjnej	35
4.1 Klasy	35
4.2 Wersja demonstracyjna	37
Zakończenie	41

Spis tabel	43
Spis rysunków	45
Spis listingów	47
Bibliografia	47

Wstęp

Stale rozrastające się zasoby wiedzy, katalogowanie jej, uporządkowanie dla systemów informatycznych na całym świecie w sposób zrozumiały, oraz pozwalający na łatwe przeszukiwanie i wykorzystanie, tak dużych ilości informacji powodowało wiele problemów w ostatnich dekadach. W roku 2018, ilość danych w internecie wynosiła 10 jottabajtów (10 000 000 000 000 000 GB) [1], stale zwiększając swój rozmiar po dziś dzień. Naturalną kwestią jest to, że wiele ważnych informacji jest pomijane, zapisane w sposób nieintuicyjny lub niezrozumiały, dla ludzi i maszyn. Ontologie, oraz języki jakimi są zapisywane, takie jak np. Web Ontology Language (OWL) [2], rozwiązują problem standaryzacji danych w celu zwiększenia ich użyteczności.

Cel i zakres

Celem pracy jest przybliżenie tematyki ontologii (w tym języków opisu ontologii a także metodologii i metodyk wytwarzania), oraz wykonanie analizy i projektu systemu tworzenia/zarządzania ontologiami (Ontology development environment - ODE). Została wykonana analiza jednego z najpopularniejszych programów do edycji ontologii - Protégé. Przeprowadzenie ankiety, oraz podsumowanie jej wyników, pozwoliło wyciągnąć wnioski, jak duża jest grupa docelowa, oraz jakie problemy napotkali w przeszłości ankietowani, korzystający z takiego oprogramowania. Zostały zebrane wymagania, stworzone diagramy klas, i przypadków użycia, oraz został przedstawiony interfejs graficzny. Implementacja wcześniej zaprojektowanych klas, oraz wersja demonstracyjna przedstawiająca wygląd oprogramowania, potwierdziła użyteczność zaproponowanych funkcji, oraz estetykę.

Rozdział 1

Ontologia

Ontologia ma wiele różnych definicji. Jak podaje PWN, słowo ontologia pochodzi od greckich słów *óntos*, czyli byt, oraz *lógos*, czyli nauka. Początkowo ontologia była działem filozofii zajmującym się badaniem czym jest byt [3]. Aktualnie w informatyce ontologia, często definiowana jest jako formalna reprezentacja dziedziny wiedzy [4], która opisuje obiekty(byty) wraz z łączącymi je relacjami. Ontologie mogą różnić się ze względu na materiał, czy zakres. Wyróżniamy [5]:

- wysokiego poziomu(z ang. top-level lub upper ontology) zajmujące się pojęciami szerokimi, które można dopasować do każdej dziedziny [6]
- dziedzinowe, zajmujące się opisem bytu w konkretnej dziedzinie

Ontologie, są używane w semantycznych wyszukiwarkach np. filmów [7]. Słowa, które wpisywane są do takiej wyszukiwarki muszą zostać przeanalizowane w celu pokazania użytkownikowi treści, które są najbardziej zbliżone do złożonego zapytania. Wyszukiwarka analizując tekst szuka relacji między danymi wyrazami, przez co może podać dokładniejszy wynik, niż gdyby nie zwracała uwagi na związki między nimi.

Ontologie są również stosowane w medycynie, gdzie dane biomedyczne są zmieniane w praktyczne wskazówki, które pomagają podejmować ważne decyzje [8].

Do tworzenia i zarządzania nimi stosuje się edytory ontologii, które zostały opisane w następnym rozdziale.

1.1 Języki

1.1.1 Języki związane z ontologiami

Jednym z pierwszych języków do zapisu ontologii był Knowledge Interchange Format (KIF) [4], który powstał w latach 90. XX wieku. Jak sama nazwa wskazuje

był formatem, który służył do wymiany wiedzy między np. różnymi programami. Mógł być wykorzystywany do takich szerokich pojęć jak np. to, że każda istniejąca osoba ma matkę (Rys. 1.1). Oparty jest na logice "first order", czyli na węższym rachunku predykatów. Kwantyfikator/predykat może wyrazić wiedzę na temat konkretnej domeny np. jeśli coś jest krajem, to zajmuje jakąś powierzchnię. W zdaniu "Kacper powiedział, że wszystko co ma powierzchnię jest państwem" predykat "powiedział" łączy zdanie z obiektem Kacper. Złożoność obliczeniowa zwiększa się wraz ze wzrostem skomplikowania.

Example KIF Sentences

Nobody can be both a brother and a sister.

```
(forall (?x ?y)
  (=> (brother ?x ?y)
    (not (sister ?x ?y))))
```

Every person has a mother.

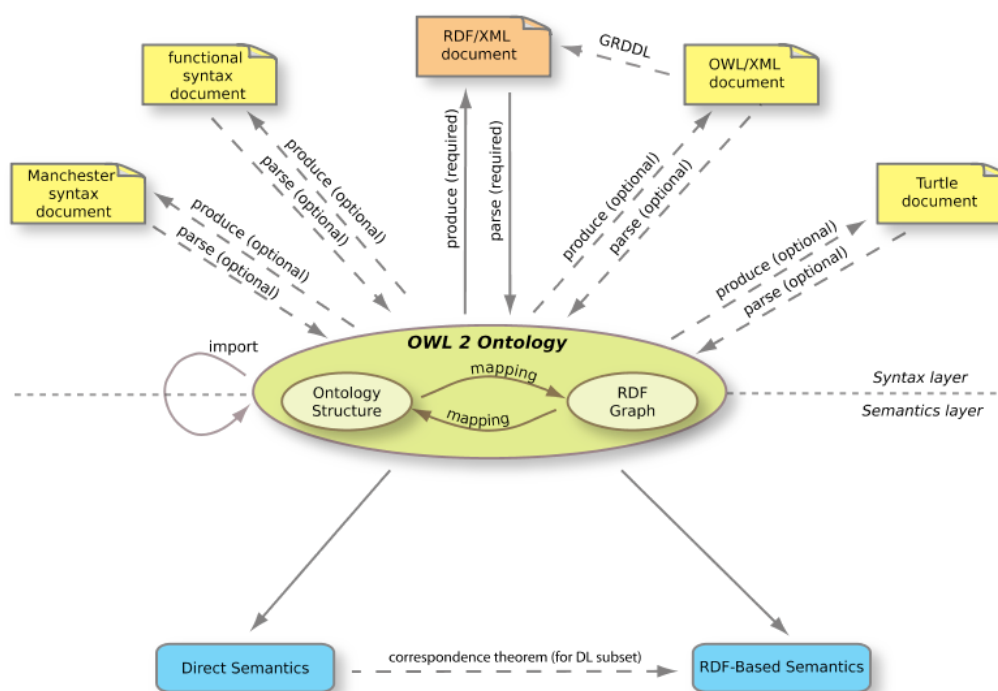
```
(forall (?x)
  (=> (person ?x)
    (exists (?y)
      (and (person ?y)
        (mother ?x ?y)))))
```

Rysunek 1.1: Przykład użycia KIF (źródło: [9])

Kolejnym znanym językiem jest Loom. Ostatnia stabilna wersja oznaczona numerem 4.0 została wypuszczona w 1999 roku. Język ten został stworzony przez grupę badawczą z Uniwersytetu Południowej Kalifornii [10]. Loom w przeciwieństwie do KIF jest językiem opartym na logice opisowej tzn. takiej, która jest o wiele łatwiejsza do zrozumienia przez człowieka, nie odstrasza złożonością i syntaktyką, oraz łatwo ją wdrożyć na dużą skalę. Jest segmentem logiki pierwszego rzędu ze zmianami i ograniczeniami.

1.1.2 OWL

Kolejnym bardzo popularnym językiem do zapisu ontologii jest Web Ontology Language (OWL). OWL został stworzony do reprezentowania bardzo dużych zbiorów wiedzy na wiele tematów [2]. Jego pierwszą oficjalną wersję wydano w 2004 roku, a następnie w 2009 premierę miała jego kolejna duża edycja, którą była OWL2 [12]. Wydanie OWL2 przyczyniło się do wzrostu zainteresowania tym językiem co skutkowało, wzrostem ilości narzędzi dla developerów, oraz zaczęto częściej wprowadzać ontologie do systemów informatycznych. Jednym z najbardziej popularnych języków

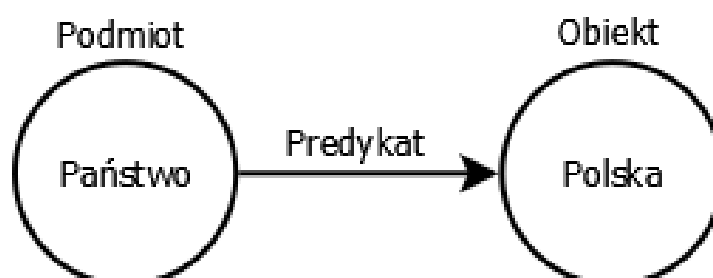


Rysunek 1.2: Struktura OWL2 (źródło: [11])

używanych z ontologiami jest OWL2.

Resource Description Framework(RDF) [2], który pierwszy raz został upubliczniony 28.02.1996 i którego ostatnia aktualna wersja została wydana 25.02.2014 jest językiem zaprojektowanym jako model abstrakcyjny, który standaryzuje jak dane są połączone ze sobą [13]. RDF jest złożony z trójek (Rys. 1.3):

- podmiot
- predykat
- obiekt/przedmiot



Rysunek 1.3: Trójka RDF

Manchester syntax jest składnią stworzoną, w taki sposób, aby ludziom było łatwiej ją rozumieć np. po przez zamianę znaku \vee na **or** lub \neg na **not**, które

ułatwia czytanie osobom niezaznajomionym z logiką opisywaną za pomocą symboli [12].

Nałożone przez XML standardy, które uniemożliwiły kodowanie niektórych grafów RDF [14] (niektóre predykaty Uniform Resource Identifier (URI) [15] zostały zabronione, oraz zakazane zostało kodowanie niektórych punktów kodowych (Unicode codepoint)), wymusiły stosowanie innego języka jakim jest *turtle*, a który nie posiada tych ograniczeń [14]. O ile, jak sama nazwa wskazuje, RDF-based semantic (Rys. 1.2) bazuje na semantyce RDF, tak semantyka bezpośrednia (Direct Semantic) jest własną semantyką OWL, którą można przetransferować do semantyki bazującej na RDF.

1.2 Metodologie i metodyki

W określeniu, czy stworzona przez nas ontologia jest poprawna pomocne są metodyki i metodologie. Metodologie można podzielić na:

- wysokiego poziomu (macro-level), które są ogólne, mówiące o procesie
- mikro poziomu (micro-level), skupiające się na tworzeniu ontologii

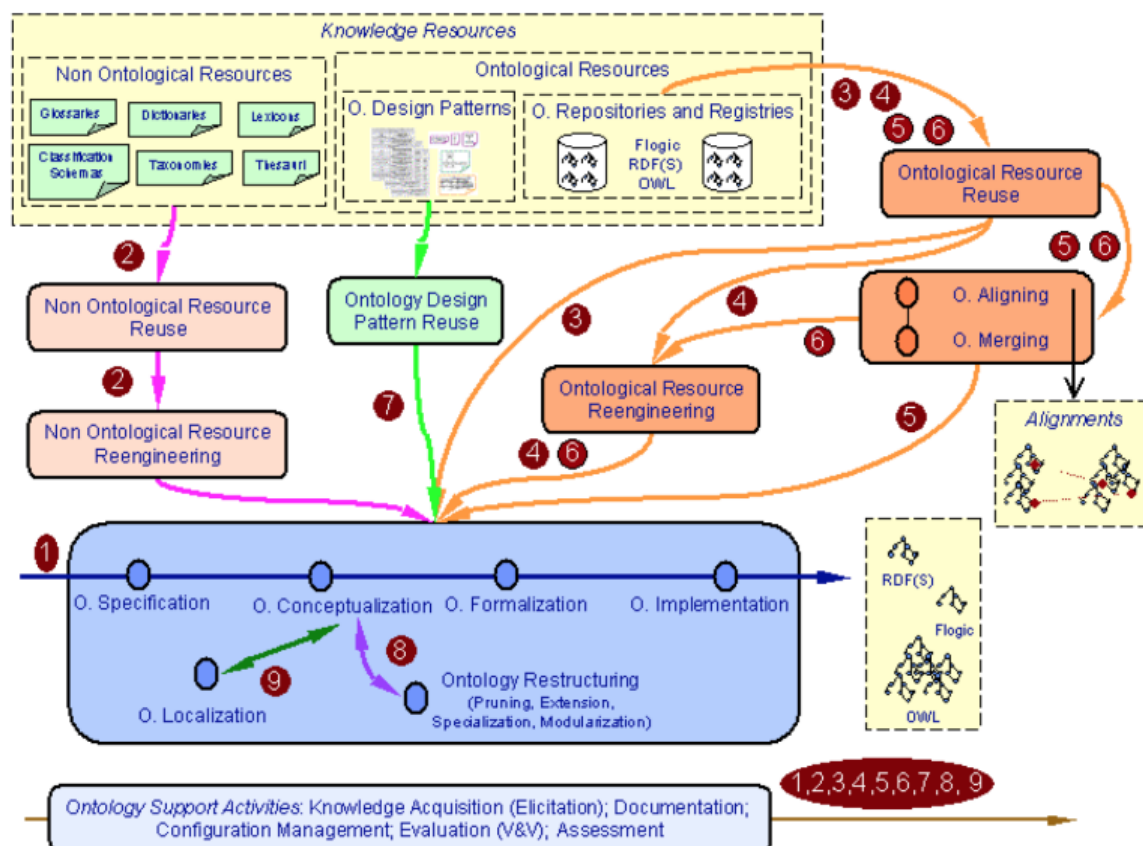
1.2.1 Metodologie wysokiego poziomu

Metodologie są zbiorem pewnych zasad, którymi należy się kierować, aby osiągnąć jakieś zamierzone cele. Metodologie wysokiego poziomu (macro level) pozwalają projektować ontologie w sposób ustrukturyzowany. Głównymi krokami są:

- specyfikacja - kto będzie korzystał, w jaki sposób, zastosowania
- konceptualizacja - analiza ryzyka (np. finansowego), analiza problemu, przygotowanie dokumentu koncepcyjnego [16]
- formalizacja - zamiana modelu konceptualnego na formalny
- implementacja - wdrożenie projektu (np. stworzenie ontologii)
- obsługa - poprawki, aktualizacje

Natomiast istnieje wiele innych kroków, które nie zostały wymienione w tych punktach, a warto o nich wspomnieć. Takimi są np. tworzenie dokumentacji, czy system kontroli wersji. Przez to, że powstawanie ontologii to bardzo złożony proces, może on różnić się ze względu na złożoność problemu, inne podejście lub różne domeny. Z czasem, rozwój ontologii wymuszał coraz to nowsze podejścia do projektowania ich, czego skutkiem jest metodologia NeON [12]. Zamiast 5 kroków wymienionych wyżej,

zaczęto stosować bardziej indywidualne ścieżki tworzenia ontologii (Rys. 1.4) oraz rozwinęto pytania kompetencyjne (CQ - Competency Questions). CQ są to pytania



Rysunek 1.4: Tworzenie ontologii (źródło: [17])

na które ontologia konkretnej domeny powinna odpowiedzieć. Dla przykładu można podać pytanie "Które rzeki płyną do morza" dla ontologii mówiącej o państwie.

Kolejną metodologią jest metodologia koła życia (z ang. lifecycle) [12] (Rys. 1.6), w której przechodzenie z jednej fazy do drugiej następuje płynnie. Można przedstawić ją formie koła, które jest podzielone na część projektowania (strona prawa, czerwona), oraz na część użytkową (strona lewa, niebieska). Wyróżniamy tam takie fazy jak:

- opracowanie wymagań
- analiza ontologiczna
- projektowanie ontologii
- projektowanie systemu
- rozwój i ponowne wykorzystanie ontologii

How many pizzas are available?	Find all vegetarian pizzas	Can you have a pizza with any combination of toppings?
Do pizzas come in different sizes?	How many pizzas in the menu contain meat?	Can I remove toppings from a pizza?
Are there any children's pizzas?	Find pizzas with a single meat ingredient	Find all the pizzas with less than 3 toppings
Are different bases available?	Does this pizza contain halal meat?	If I have 3 ingredients, how many kinds of pizza I would make?
What kind of bases are possible?	Find all the nut free pizzas	Find all pizzas which are sharing 3 or more ingredients
Show me all pizza base options	Which pizzas do not have nuts?	Are we including folded pizzas (calzone) in our domain?
Is it thin or thick bread?	How many pizzas have either ham or chicken topping?	Should we include the oven type in the pizza definition? (e.g. wood fired vs. electric oven)
Is it "deep pan" or "Chicago" style? (deep pan sucks!)	Find all pizzas that have prawns but not anchovy	Explore relationships between common pizza choices to hypothesise about the origins of pizza toppings
Is it stuffed crust and what is it stuffed with?	What is the most popular (used) topping?	How much it will cost me to order all pizzas in the menu?
Which are gluten free bases?	What is the third least popular topping?	How much does Margherita Pizza weight?
Which pizzas are spicy?	Which toppings are allowed for customisation purposes?	How many pizzas did I eat last week?
Which sort of cheese do we have?	Reason backwards and suggest toppings that commonly go with each other (e.g. anchovies and capers)	I want to know whether a pizza is healthy
What sauces are available?	Are toppings organic?	Can I have a menu without pizzas please?
Find pizzas with peppers and olives	Show me the offers of the day	Which is the latest combination of toppings?

Rysunek 1.5: Przykłady pytań koncepcyjnych (źródło: [18])

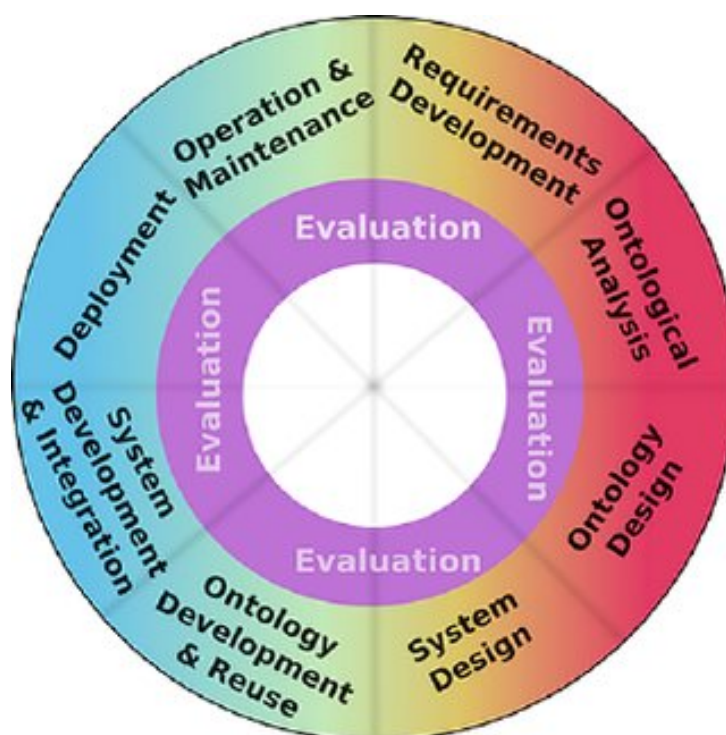
- rozwój i integracja systemu
- zastosowanie
- obsługa i konserwacja

Metodologia dotycząca zarządzania projektami nazywana Agile [20], pomaga po przez samo organizujące się zespoły, zwiększyć komunikację w grupie za sprawą częstszych spotkań, oraz zredukować stres, ponieważ zespoły same decydują o terminach realizacji zadań, które narzucane, często są nierealne do zrealizowania. W tym podejściu ważniejsze jest współdziałanie, oraz dostarczenie działającego produktu, niż rozbudowana dokumentacja, lub trzymanie się sztywno ram projektu.

1.2.2 Metodologie mikro poziomu

Metodologie mikro poziomu, poruszają inne problemy niż wysokiego poziomu. Te metodologie pokazują w jaki sposób przejść z reprezentacji nieformalnej do opartej na logice. Jak i w metodologiach wysokiego poziomu tak i tutaj mamy parę kroków, które pozwalają nam lepiej tworzyć ontologie. Są nimi:

- analiza wymagań, przypadki użycia
- projektowanie architektury ontologii, rozproszonej lub nie



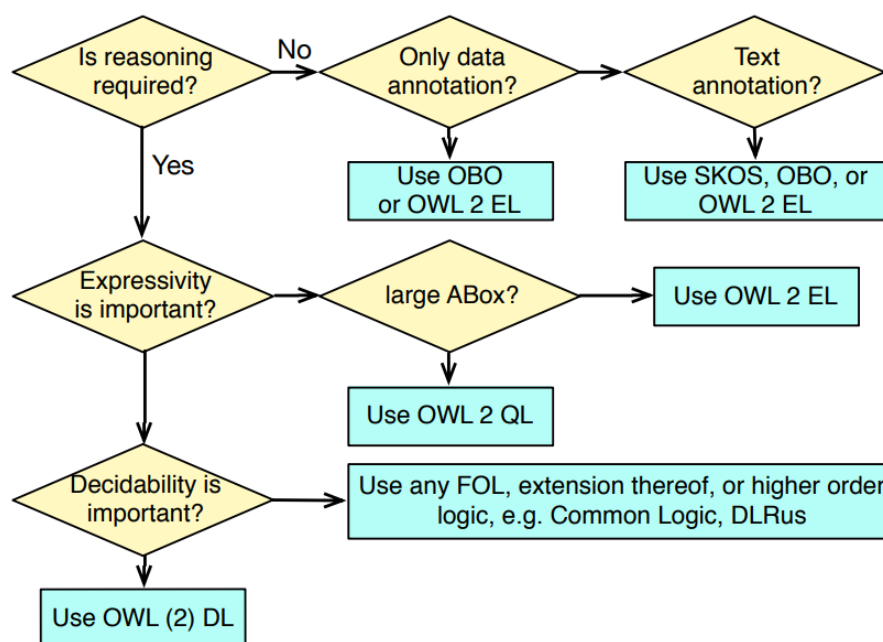
Rysunek 1.6: Metodologia koła życia (źródło: [19])

- wybór języka reprezentacji
- wybór fundamentalnej ontologii
- rozważenie domeny ontologii
- półautomatyczne podejścia oddolne, narzędzia i przekształcenia językowe
- formalizacja
- jeśli potrzeba, generowanie wersji lite
- obsługa i konserwacja

Jednym z najważniejszych zadań jest wybór języka, w którym zamierzamy tworzyć naszą ontologię (Rys. 1.7). Trzeba odpowiedzieć sobie na pytania np. czy wymagane są uzasadnienia, lub jaki typ/typy danych będziemy zbierać. Wyniki pozwolą nam zdecydować jaki język będzie odpowiedni do naszego problemu.

1.2.3 Metodyki

Prawie w każdej dziedzinie życia jest wiele rozwiązań tego samego problemu, tak samo w ontologiach jest wiele różnych metodyk, jedną z nich jest np IDEF5 z 1989 roku [22]. Zakłada ona przedstawienie pewnych informacji takich jak obiekty, relacje



Rysunek 1.7: Diagram pomagający przy wyborze języka (źródło: [21])

między nimi, oraz wartości w sposób uporządkowany składający się z elementów takich jak [23]:

- język graficzny
- język tekstowy
- metoda w jaki sposób przechwytywać ontologie

W każdej z wcześniejszych metodologii można znaleźć wiele metod, które są dopełnieniem procesu tworzenia ontologii. Osoba tworząca ontologie często musi zadać sobie pytanie w jaki sposób pokazywać ją użytkownikowi - może to być za pomocą diagramów, ale również w sposób tekstowy. W Metodykach najczęściej mamy podejścia filozoficzne, logiczne lub mieszane.

OntoClean pomaga w znajdowaniu błędów w taksonomii, czyli kategoryzacji elementów. Wykorzystując pojęcia z filozofii takie jak:

- sztywność (zapis: +R dla sztywności, oraz -R dla braku sztywności)
- tożsamość (zapis: +I dla tożsamości, oraz -I dla braku tożsamości)
- jedność (zapis: +U dla jedności, oraz -U dla braku jedności),

oraz dzięki dopasowanym im elementom składni pomaga w tworzeniu łatwiejszych do zrozumienia ontologii [24][25]. Klasa człowiek jest sztywna, ponieważ przez całe

życie człowiek jest człowiekiem, natomiast klasa student nie jest sztywna, ponieważ studentem jest się przez jakiś okres czasu w życiu. Aby można było odróżnić klasy od siebie ważna jest ich tożsamość. Jeśli występuje niejednoznaczność w odróżnieniu od siebie instancji, należy przyjąć, że są one różne. W OntoClean występują sortale, tzn takie klasy, których byty są identyfikowane jako ten sam obiekt. Klasa, która jest sortalem wymusza, na jej podklasach również bycie sortalami. Jednościami są indywidualności, których wszystkie części są połączone tylko ze sobą, przez wyodrębnioną relację [26].

Przez to, że przy tworzeniu ontologii można popełnić wiele błędów, zostały one zapisane w postaci tzw antywzorców, lub katalogu pułapek [27], z których część można wykryć automatycznie za pomocą OntOlogy Pitfall Scannera!(OOPS!). Jednym z takich błędów jest np. tworzenie synonimów jako klas, lub łączenie różnych pojęć w tej samej klasie.[12]

Human understanding	Modelling issues
<ul style="list-style-type: none"> • P1. Creating polysemous elements • P2. Creating synonyms as classes • P7. Merging different concepts in the same class • P8. Missing annotations • P11. Missing domain or range in properties • P12. Missing equivalent properties • P13. Missing inverse relationships • P19. Swapping intersection and union • P20. Misusing ontology annotations • P22. Using different naming criteria in the ontology 	<ul style="list-style-type: none"> • P2. Creating synonyms as classes • P3. Creating the relationship “is” instead of using "rdfs:subClassOf", "rdf:type" or "owl:sameAs" • P4. Creating unconnected ontology elements • P5. Defining wrong inverse relationships • P6. Including cycles in the hierarchy • P7. Merging different concepts in the same class • P10. Missing disjointness • P17. Specializing too much a hierarchy • P11. Missing domain or range in properties • P12. Missing equivalent properties • P13. Missing inverse relationships • P14. Misusing "owl:allValuesFrom" • P15. Misusing “not some” and “some not” • P18. Specifying too much the domain or the range • P19. Swapping intersection and union • P21. Using a miscellaneous class • P23. Using incorrectly ontology elements • P24. Using recursive definition • P25. Defining a relationship inverse to itself • P26. Defining inverse relationships for a symmetric one • P27. Defining wrong equivalent relationships • P28. Defining wrong symmetric relationships • P29. Defining wrong transitive relationships
Logical consistency	
<ul style="list-style-type: none"> • P5. Defining wrong inverse relationships • P6. Including cycles in the hierarchy • P14. Misusing "owl:allValuesFrom" • P15. Misusing “not some” and “some not” • P18. Specifying too much the domain or the range • P19. Swapping intersection and union • P27. Defining wrong equivalent relationships • P28. Defining wrong symmetric relationships • P29. Defining wrong transitive relationships 	
Real world representation	
<ul style="list-style-type: none"> • P9. Missing basic information • P10. Missing disjointness 	

Rysunek 1.8: Tabela pułapek (źródło: [27])

Rozdział 2

ODE

2.1 Czym jest ODE

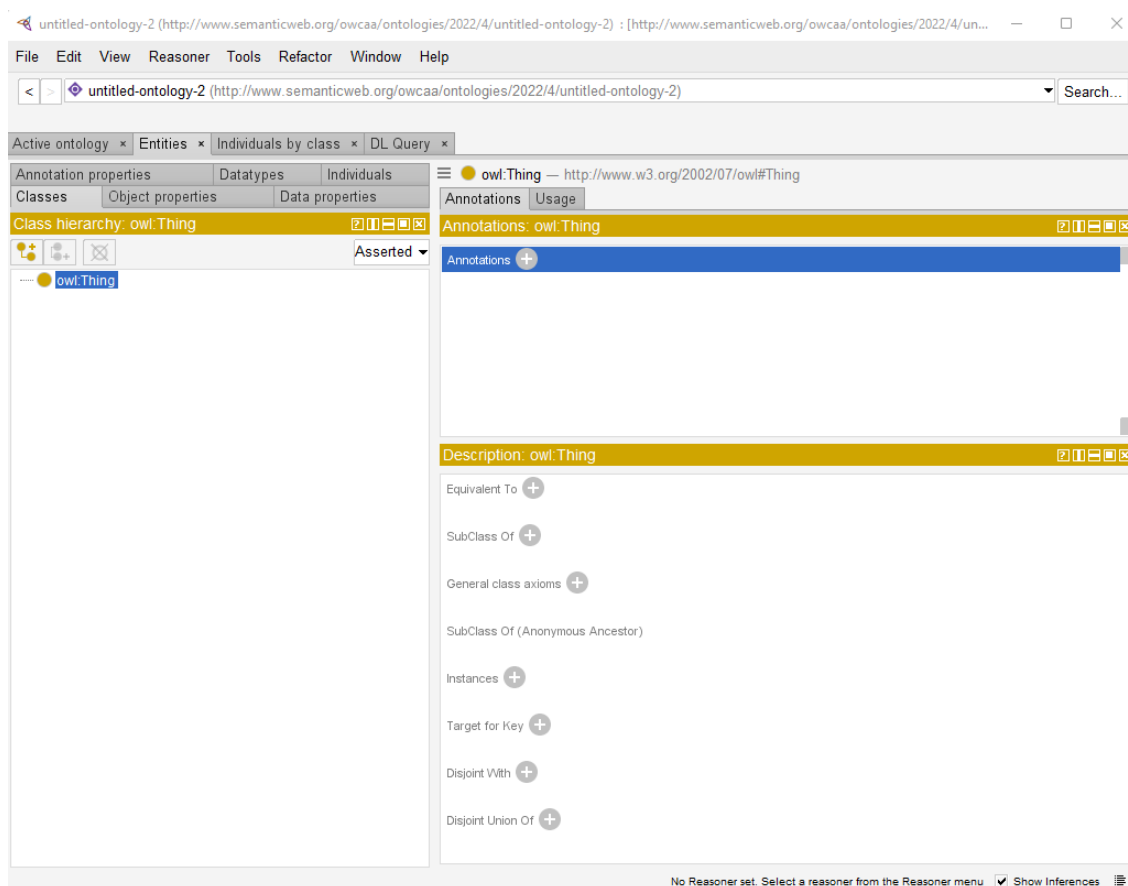
ODE jest środowiskiem programistycznym, który umożliwia oraz pomaga tworzyć własne ontologie. Każdy programista tworzący oprogramowania lub ontologie, wie jak ważne jest odpowiednio przystosowane do tego środowisko, zapewniające odpowiednią ilość funkcji, intuicyjny interfejs, czasami obsługę dodatków, które wprowadzają usprawnienia, czy zmieniają motyw, dzięki czemu osoba korzystająca z niego może czuć się bardziej komfortowo. Szerokie upodobania użytkowników, powodują powstawanie wielu różnych edytorów ontologii, dzięki czemu każdy jest w stanie dopasować je pod własne preferencje wizualne, czy pod określoną funkcjonalność. Edytory te mogą różnić się chociażby językami jakie wspierają. Jednymi z najbardziej znanych oraz mających często bardzo różną funkcjonalność środowisk są:

- Protégé
- NeOn Toolkit
- SWOO.P
- TopBraid Composer
- Vitro
- Knoodl
- OWLGrEd (który jest edytorem graficznym)

2.2 Zastosowania oraz działanie ODE

W tej pracy działanie edytorów ontologii jest przedstawione na podstawie edytora Protégé, który jest jednym z najbardziej znanych, i najczęściej używanych programów do tworzenia i zarządzania ontologiami. Baza zarejestrowanych użytkowników strony oprogramowania, zawiera 366 084 kont [28], mimo iż nie jest to wymagane do pobrania programu (wymagane jest do wersji webowej). Jego popularność może być zasługą tego, że jest otwarto-źródłowy oraz darmowy. Został stworzony w centrum badań informatyki biomedycznej w szkole medycznej Uniwersytetu Stanforda w Kalifornii [8].

Na samej górze interfejsu użytkownika (Rys. 2.1) znajduje się proste menu, w którym można znaleźć takie opcje jak: utworzenie ontologii z pliku w formatach: .pont, .pins, .xml, .owl, .rdf; możliwe jest dodawanie nowych zakładki takich jak widok klas czy OWL Viz, który pokazuje w sposób graficzny klasy i relacje między nimi. W widoku klas można dodawać nowe klasy, oraz konkretnym z nich dodawać



Rysunek 2.1: Protege

opisy, którymi mogą być:

- równoważność do innej klasy

- jest podklasą
- nad klasy
- instancje (obiekty)
- rozłączność z

Kolejną ważną zakładką jest właściwości obiektu w której można tworzyć relacje między obiektami. Każda taka relacja może w swoim opisie zawierać np.

- równoważność
- jest pod własnością
- odwrotność
- domenę
- zakres

W zakładce właściwości danych można tworzyć dane konkretnych typów, oraz nadawać im właściwości takie jak:

- równoważność
- jest pod własnością
- domenę
- zakres

W karcie indywidualność jest możliwość tworzenia instancji konkretnych klas, dodania typów, równoważności, odmienności instancji do innej, oraz wcześniej utworzonych właściwości danych/obiektów. OWLViz pozwala nam oglądać hierarchię klas w postaci grafów, gdzie można za pomocą przycisków sterować tym, które klasy i ich podklasy mają być pokazywane. Podobnie działa OntoGraf.

Jest to tylko część z funkcjonalności jakie posiada Protege. Jest to bardzo rozbudowane środowisko do tworzenia i zarządzania ontologiami, co ma swoje zalety, jak i wady które zostały opisane w następnych rozdziałach.

2.3 Problemy istniejących ODE oraz potencjał rynkowy

Potencjał rynkowy jest to "szacowanie maksymalnej, możliwej do osiągnięcia przez wszystkie firmy na danym rynku wielkości sprzedaży" [29]. Upraszczając definicję można wywnioskować, że produkt który jest tworzony musi być lepszy pod jakimś względem od aktualnie dostępnych, żeby móc trafić w mniej zagospodarowany przez firmy teren, które np. są dłużej na rynku. Gdy obiekt zbytu trafia do wielu konsumentów, łatwiej jest zbadać ich potrzeby, oraz oczekiwania. Ciężko określić potencjał rynkowy programu do tworzenia i zarządzania ontologiami, ze względu na małą ilość osób, które kiedykolwiek miało styczność z takimi edytorami.

2.4 Ankieta

W celu lepszego zrozumienia rynku, oraz potrzeb konsumentów z polski, została przeprowadzona ankieta w formularzach docs od google. Nacisk położono na jak najmniejszą ilość punktów, na to żeby były krótkie, i aby w otwartych z nich dać ankietowanym przykłady odpowiedzi. Opracowane pytania zostały ułożone w sposób logiczny, od najbardziej ogólnych do bardziej szczegółowych:

- Czy wiesz czym są ontologie?
- Czy słyszałeś kiedyś o edytorach ontologii?
- Czy korzystałeś kiedykolwiek z edytora ontologii?
- Jeśli tak to jakie problemy napotkałeś (nieintuicyjny interfejs, brak lub zbyt wiele funkcji itp)
- Co nie podoba Ci się w oprogramowaniu ze zdjęcia (zdjęcie pochodzi z edytora Protege)

Wyniki, mają za zadanie wskazać jak duże jest zainteresowanie ontologiami, wśród ludzi młodych, jakie problemy napotykają w ontologiach, oraz ich spostrzeżenia na temat protege.

2.4.1 Analiza wyników ankiety

Analizując odpowiedzi z ankiety, w której udział wzięło 50 osób, wynika że ponad połowa nie miała styczności z takimi programami, i nie posiadają wiedzy czym są ontologie (Rys. 2.2). Wnioskując po odpowiedziach można wysnuć tezę, że taki projekt nie znajdzie bardzo szerokiego grona odbiorców.

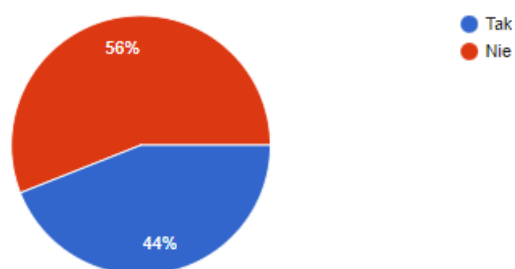
Ankietowani, którzy korzystali z edytorów ontologii, lub którym interfejs Protege nie spodobał się, wymienili problemy takie jak:

- przestarzały interfejs
- brak trybu ciemnego
- brak przechowywania plików w chmurze
- brak języka polskiego
- zbyt wiele funkcji, przez co problematyczne jest wdrożenie się do środowiska
- brak wprowadzenia (poradnika w postaci wyskakujących okien przedstawiających interfejs, przy pierwszym korzystaniu z oprogramowania)

Aby zachęcić użytkowników innych programów do zmiany swoich edytorów na rzecz tego zaprojektowanego w tej pracy, powinien on być pozbawiony większości problemów wymienionych przez ankietowanych.

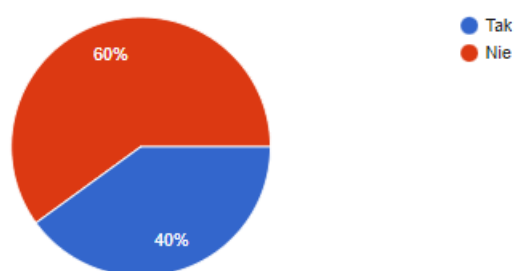
Czy wiesz czym są ontologie?

50 odpowiedzi



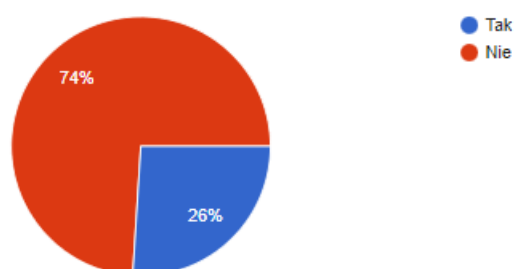
Czy słyszałeś kiedyś o edytorach ontologii?

50 odpowiedzi



Czy korzystałeś kiedykolwiek z edytora ontologii?

50 odpowiedzi



Rysunek 2.2: Wyniki przeprowadzonej ankiety

Rozdział 3

Zebranie wymagań oraz projekt aplikacji

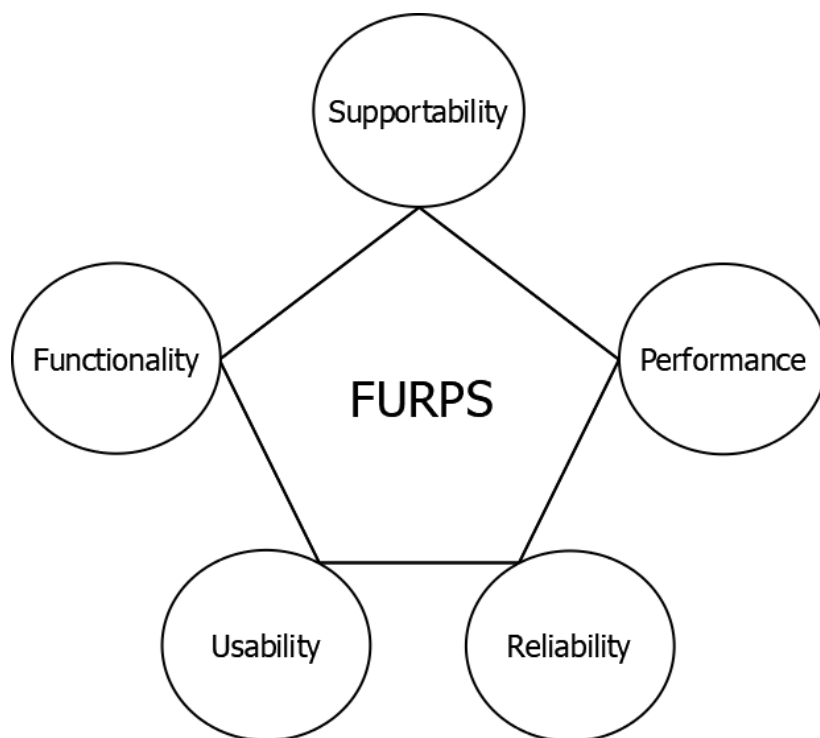
3.1 FURPS

FURPS (functionality, usability, reliability, performance, supportability - funkcjonalność, użyteczność, niezawodność, wydajność, wsparcie) [30] jest to model pomagający zebrać jak największą ilość wymagań jakie projekt powinien spełniać. Został on wymyślony przez Roberta Gready'ego który stworzył listę 5 głównych zagadnień, których przynajmniej w teorii opracowanie powinno pomóc pozbyć się większości nieścisłości i problemów jakie użytkownicy, oraz klienci dla których projektuje się aplikacje mogą napotkać w przyszłości.

Każdy z tych 5 punktów można rozwinąć o kolejne podpunkty (Tab. 3.1).

Funkcjonalność	Używalność	Wydajność	Niezawodność	Wsparcie
Wymagania klienta	Spójność	Przepustowość	Dokładność	Adaptowalność
Zapisywanie logów	Estetyka	Responsywność	Awaryjność	Kompatybilność
Połączenia	Ergonomia	Skalowalność	Samonaprawa	Szybkość napraw
Możliwość rozbudowy	Dostępność			
Licencjonowanie	Lokalizacja językowa			
	Wsparcie użytkownika			

Tabela 3.1: Rozwinięcie pkt. z FURPS (źródło: [30])



Rysunek 3.1: model FURPS

3.2 Projekt aplikacji

3.2.1 Funkcjonalność

Projektując ODE główny nacisk został postawiony na wybranie najbardziej potrzebnych funkcji niezbędnych do tworzenia ontologii i przedstawienie ich w prostej formie, żeby nowy użytkownik czuł się pewnie poruszając się po interfejsie, i żeby nie przytłaczała go ilość funkcji.

Aktorami są użytkownicy, którzy nie mają wielkiej wiedzy z zakresu ontologii, oraz osoby, którzy nie mają wysokich wymagań w stosunku do ilości funkcji jakich potrzebują do tworzenia ontologii.

Kluczowymi opcjami jakie posiada praktycznie każde środowisko programistyczne, i które również to będzie posiadało są:

- otworezenie pliku (z serwera lub komputera)
- utworzenie pliku
- zapisanie (na serwerze lub komputerze)
- usunięcie pliku
- zmiana nazwy pliku

Aby umożliwić użytkownikowi zapisywanie swoich ontologii na serwerze, każdy klient będzie obowiązkowo potrzebował założyć własne konto w ODE, oraz logować się na stronie w celu weryfikacji tożsamości, i przydzielenia dostępu do odpowiednich plików. Przy rejestracji będzie wymagane podanie:

- e-maila
- hasła
- powtórnie hasła

Przy logowaniu natomiast wystarczy:

- e-mail jako login
- hasło

Jeśli użytkownik zapomniałby swojego hasła, będzie możliwość zmiany, po podaniu maila, zostanie wysłany link resetujący hasło. Przy pierwszym logowaniu, użytkownik będzie przeprowadzony przez program dzięki wyskakującym okienkom opisującym podstawową funkcjonalność.

Kolejnymi ważnymi funkcjami, które są już związane głównie z ontologiami są:

- w zarządzaniu klasami, własnościami obiektów, oraz własnościami danych:
 - dodanie podklasy/pod własności obiektu/pod własności danych
 - dodanie równej klasy/własności obiektu/własności danych
 - usunięcie samej klasy, lub wszystkich jej podklas razem z nią
- w zarządzaniu indywidualnościami klas:
 - dodanie indywidualności
 - usunięcie indywidualności

Do każdej z klas można usunąć, lub dodać jako opis:

- równość klasy do innej
- nad klasy(super klasy)
- instancje

Do każdej z indywidualności można usunąć, lub dodać jako opis:

- typ (z jakiej klasy pochodzi)
- równość indywidualności do innej

- przypisanie właściwości danych

Do każdej własności obiektów można usunąć, lub dodać jako opis:

- równość własności do innej
- nazwy własności, których jest pod własnością
- własność, z którą jest odwrotna
- domenę
- zasięg

Do każdej własności danych można usunąć, lub dodać jako opis:

- równość własności do innej
- nazwy własności, których jest pod 3ewłasnością
- zasięg
- domenę

3.2.2 Używalność, wydajność, niezawodność i wsparcie

Zwracając uwagę na wyniki ankiety, które zostały opisane w rozdziale 2.4.1, ODE zostało zaprojektowane w stylu nowoczesnym, który objawia się między innymi tym, że elementy widoczne na ekranie są zaokrąglone, oraz posiadają lekkie cienie co dodaje wrażenia trójwymiarowości. Tryb ciemny, również jest czymś co klient może odebrać jako obcowanie z nowoczesnym oprogramowaniem. Z lokalizacji językowej jaka znajduje się na stronie jest język angielski jako domyślny, oraz polski. Dzięki temu, że kod został udostępniony publicznie na platformie github [31] w ramach otwartej licencji, użytkownicy po przez wątki, które będą mogli poruszać, otrzymują informacje jak rozwiązać ich problemy, a bardziej ambitni z nich, będą mieli możliwość dodawania własnej funkcjonalności, która będzie mogła być w przyszłości(po weryfikacji) dodana do ODE. Ważne jest również to, aby do oprogramowania powstały automatyczne testy. Wiele osób uważa, że większość projektu, zajmują testy jednostkowe, które sprawdzają poprawność napisanego kodu. Aktualnie nie ma w projekcie uwzględnionej dostępności dla osób z niepełnosprawnościami, ani wersji mobilnej, która najprawdopodobniej w tego typu oprogramowaniu była by niemożliwa do stworzenia w sposób ergonomiczny. Strona powinna być kompatybilna z najnowszymi wersjami przeglądarek internetowych takich jak:

- Google Chrome
- Microsoft Edge

- Safari
- Mozilla Firefox
- Opera

Przez to, że grupa do, której kierowany jest produkt jest niewielka, obciążenie serwera również nie będzie duże. Można założyć, że miesięcznie na stronie pojawi się około 50 użytkowników z maksymalnie około 10 w jednym momencie. Każdy z nich powinien mieć własne miejsce do zapisu. Przykładowy plik ontologii o pizzach zajmuje około 130 KB miejsca na dysku [32]. Przyjmując, że każdy użytkownik posiadałby średnio 3 takie pliki, potrzebne by było 6500 KB ($\sim 0,01$ GB) miejsca na dane. Pliki aplikacji, oraz użytkowników będą znajdować się na jednym serwerze, który powinien mieć przynajmniej 0,5 GB miejsca na dane.

W możliwościach rozwoju należy uwzględnić zaprojektowanie motywu wysokiego kontrastu, funkcji czytaj, która zamieniała by treść strony na mowę, dodanie możliwości udostępniania swojej ontologii innym użytkownikom, oraz projekt wizualizacji ontologii za pomocą grafów. W kwestii poprawy bezpieczeństwa będzie wymagane zaprojektowanie uwierzytelniania wielopoziomowego. Gdyby, dysk zaczął się zapełniać, przy 80% należy dokupić 50% więcej miejsca.

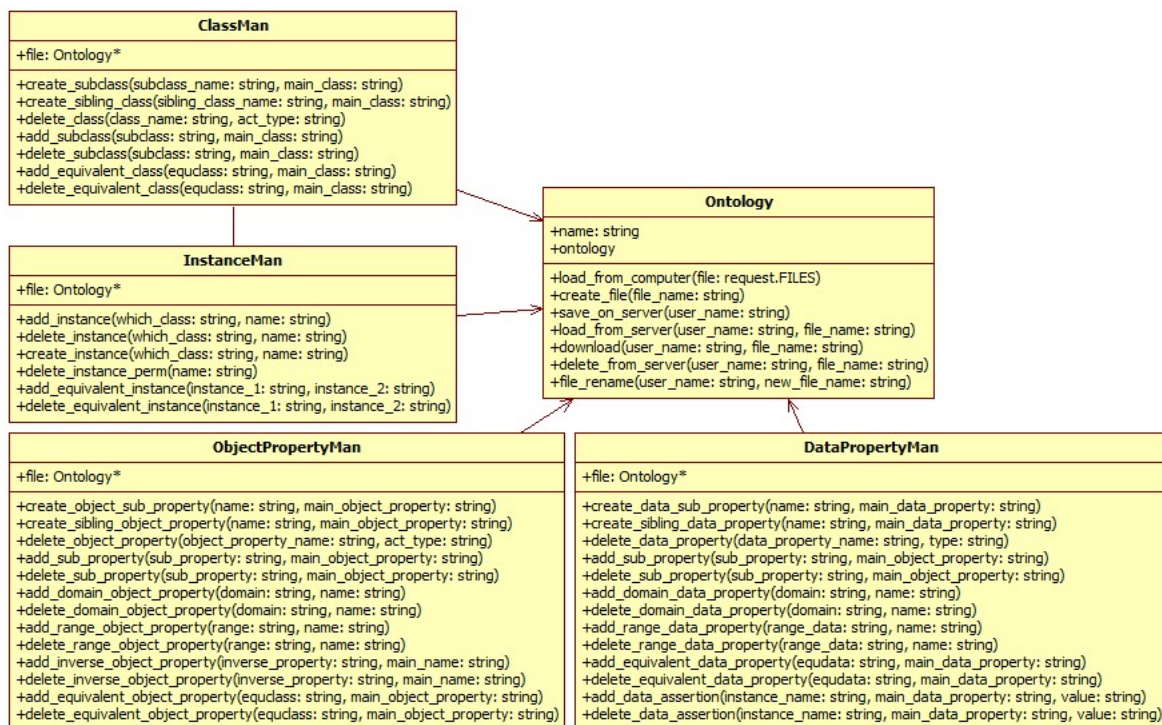
3.2.3 Klasy, oraz przypadki użycia

Projekt zakłada 5 klas, którymi są (Rys. 3.2):

- Ontology
- ClassMan
- InstanceMan
- ObjectPropertyMan
- DataPropertyMan

Klasa Ontology zapewnia obsługę plików ontologii użytkownika, natomiast reszta pozwala na zarządzanie podzielonymi odpowiednio na klasy, instancje, właściwości obiektów i właściwości danych elementami ontologii.

Stworzony diagram przypadków użycia (Rys. 3.3), przedstawia w sposób ogólny interakcje z aktorem, ramy systemu, oraz przypadki użycia. Użytkownik może stworzyć konto. Aktor, żeby zarządzać plikami ontologii będzie musiał się zalogować. Zarządzanie klasami, instancjami, właściwościami obiektów/danych rozszerza zarządzanie ontologiami, które rozszerza operowanie na plikach ontologii.



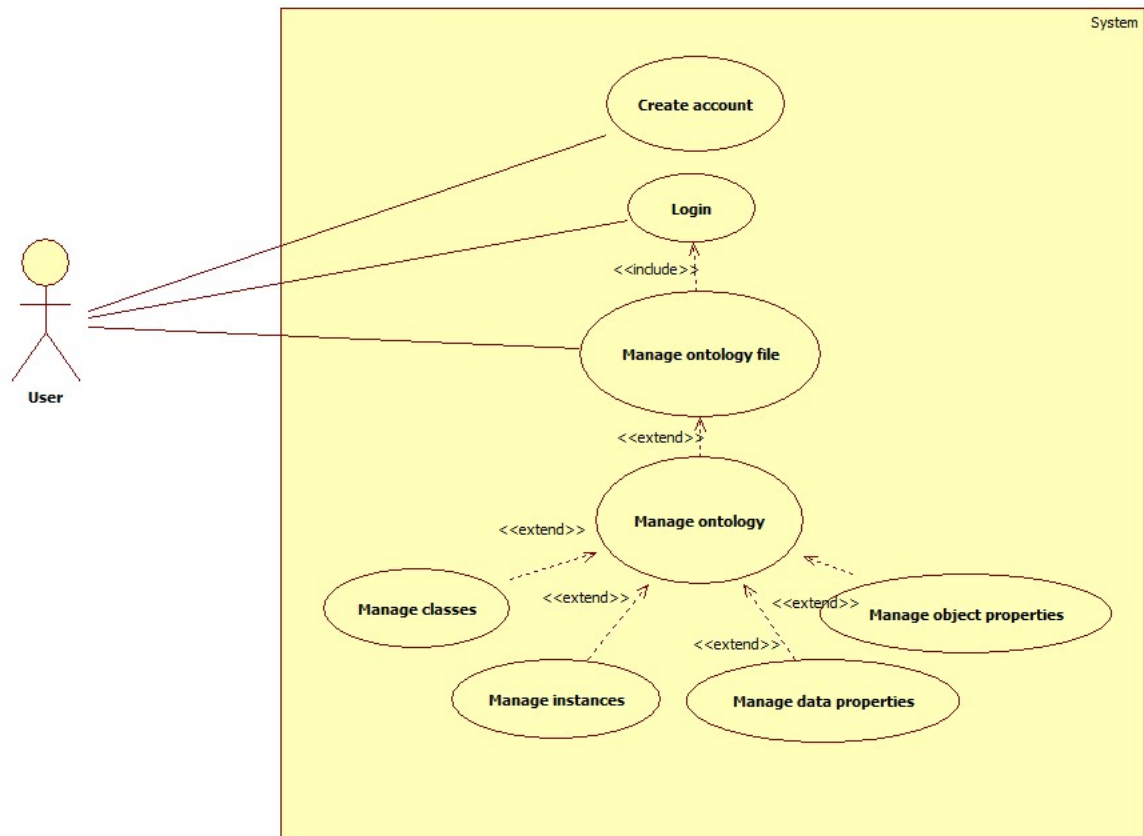
Rysunek 3.2: Diagram klas

Większość zaproponowanych metod powinna być zrozumiała, natomiast należy rozróżnić metody `delete_instance` oraz `delete_instance_perm`, pierwsza metoda usuwa instancje z opisu klasy, natomiast druga odpowiada za całkowite jej usunięcie.

Najbardziej rozbudowaną metodą jest ta, odpowiadająca usuwaniu klas, właściwości obiektów, oraz właściwości danych (Rys. 3.4). Zostaje jej przekazana nazwa usuwanego elementu, oraz typ(w jaki sposób usunąć). Gdy użytkownik usuwa jedną klasę(bez podklas), najpierw podłączone zostają jej podklasy do jej nad klasy, a następnie zostaje usunięta klasa, której nazwa została przekazana do metody. Kiedy użytkownik chce usunąć wszystkie podklasy, metoda przechodzi rekurencyjnie po podklasach przekazując je sobie, do momentu, aż nie będzie żadnych, i usuwa klasę, która była przekazana jako ostatnia. Następnie wraca do wcześniejszych, i usuwa je. Analogicznie działa to dla właściwości obiektów, oraz właściwości danych.

3.2.4 Interfejs

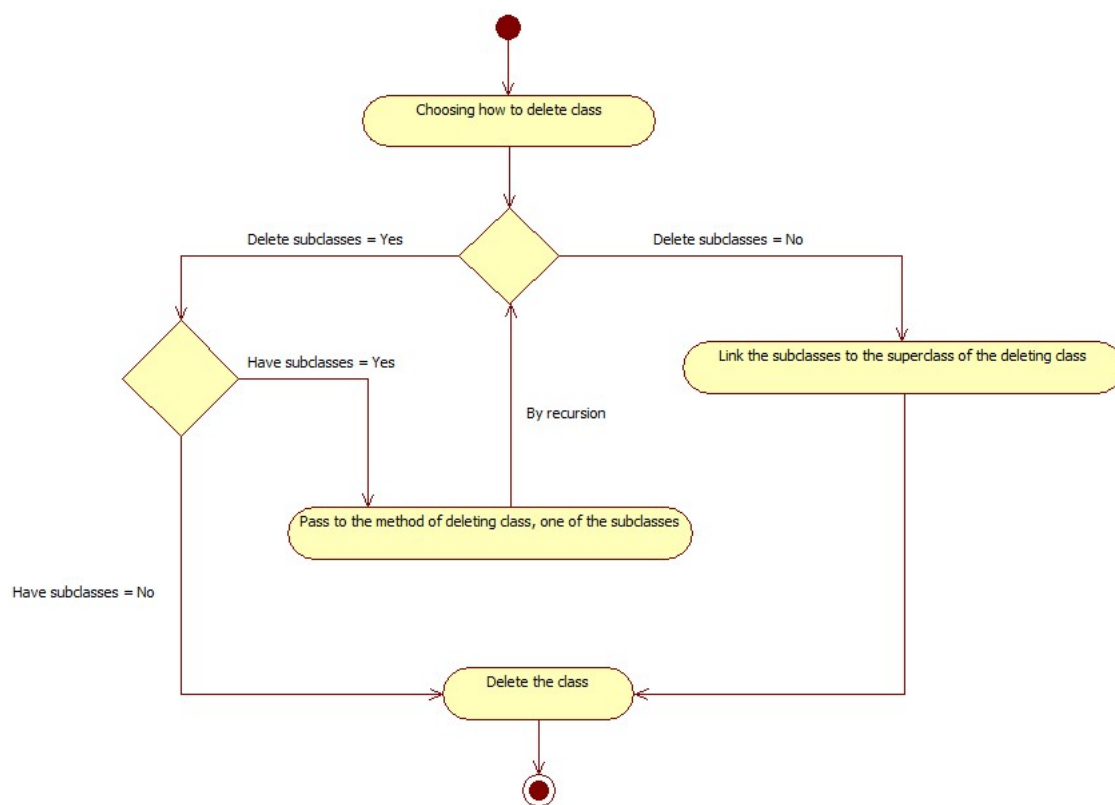
Graficzny interfejs użytkownika(GUI) został zaprojektowany w sposób nowoczesny (Rys. 3.5). Przeważają w nim odcienie białego, pomarańczowego, różowego, oraz niebieskiego. Cienie dodają wrażenia trójwymiarowości. Na górze interfejsu znajdują się przyciski `language`, oraz `change mode`, które pozwolą odpowiednio na zmianę języka i włączenie trybu ciemnego(bądź wyłączenie). W centralnym punkcie znajduje



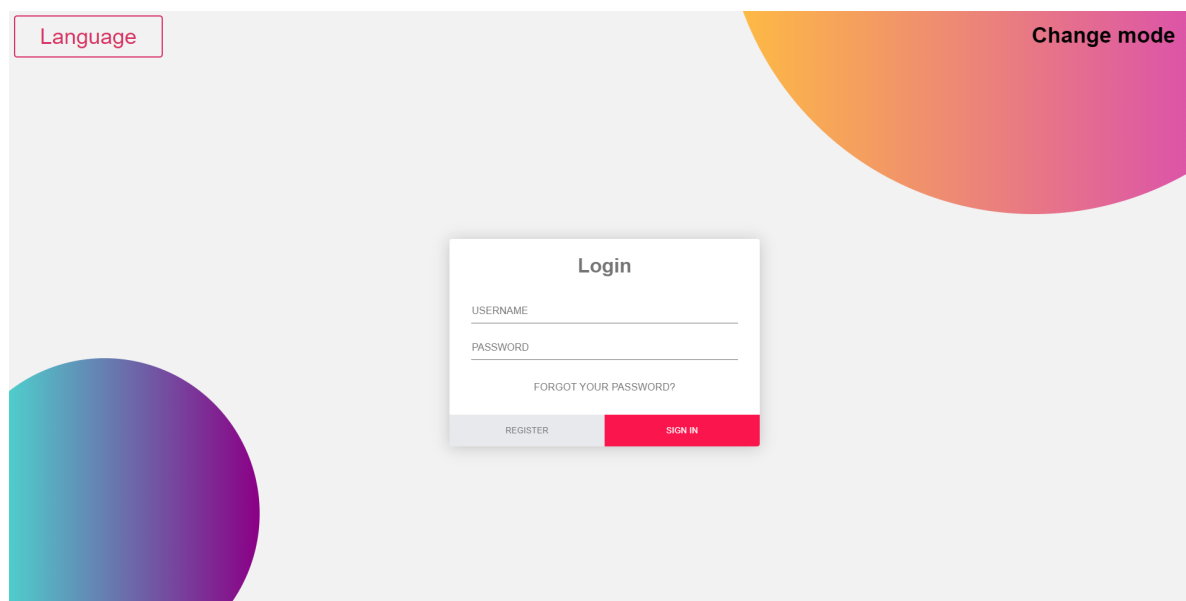
Rysunek 3.3: Diagram przypadków użycia

się miejsce na wprowadzenie nazwy użytkownika, oraz hasła. Strona rejestracji różni się tylko częścią środkową (Rys. 3.6). Ciemny motyw zakłada zmianę barwy tła, z białej, na ciemnoszarą, dopasowanie koloru czcionki, żeby była czytelna, oraz zmianę wypełnień przycisków language na różowy, oraz przycisku register na ciemniejszy szary (Rys. 3.7).

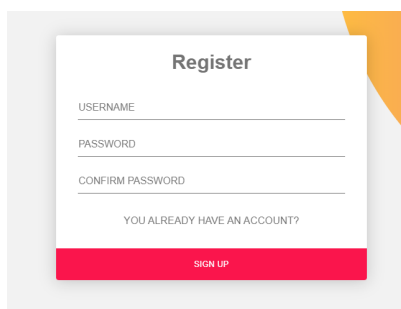
Na stronie głównej (Rys. 3.8) w pasku nawigacyjnym został dodany przycisk file w takim samym stylu jak language, powitanie użytkownika po jego nazwie, oraz wylogowanie. Niżej znajdują się przyciski pozwalające na zmianę aktualnie przeglądanych elementów ontologii. W przestrzeni po lewej stronie, jest wyświetlana hierarchia odpowiadająca aktualnie wybranej opcji np. klas, z rozwijanymi listami tych klas, oraz możliwością dodawania podklas, klas bliźniaczych i usuwania ich. Po prawej stronie opis aktualnie wybranej klasy z możliwością usuwania oraz dodawania kolejnych elementów do opisu.



Rysunek 3.4: Diagram aktywności

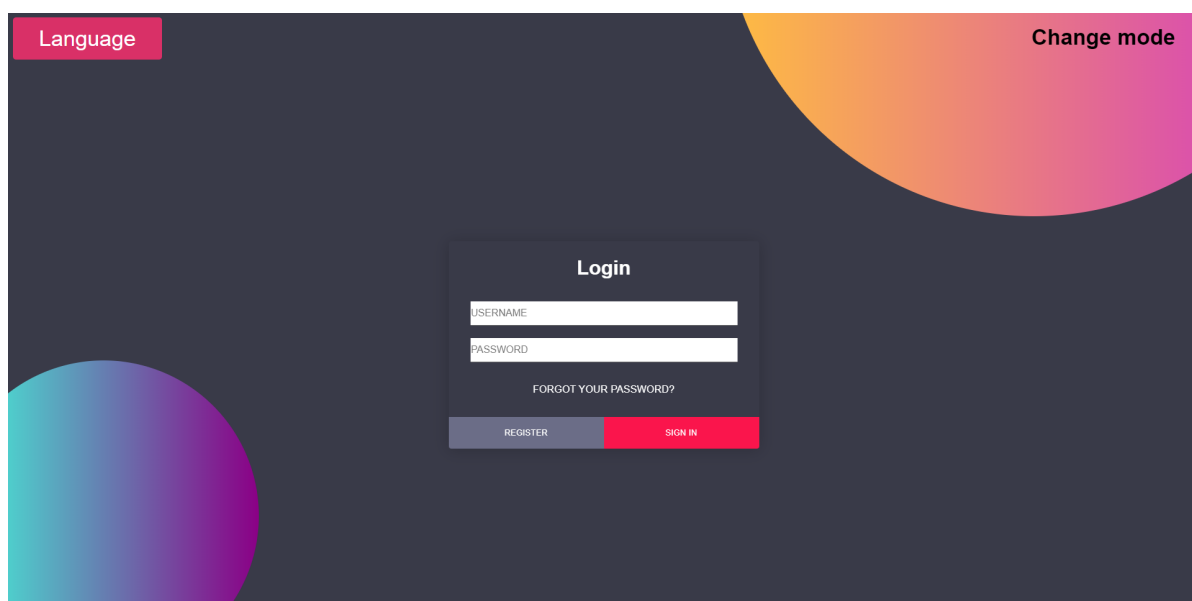


Rysunek 3.5: Strona logowania



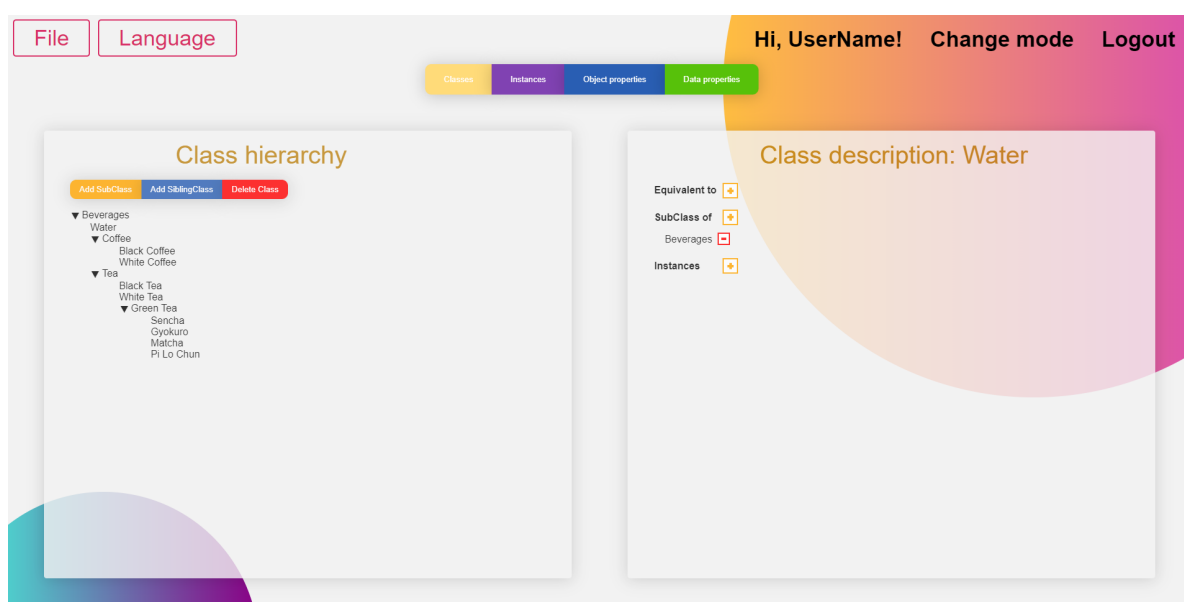
A registration form titled "Register" with three input fields: "USERNAME", "PASSWORD", and "CONFIRM PASSWORD". Below the fields is a link "YOU ALREADY HAVE AN ACCOUNT?". At the bottom is a red "SIGN UP" button.

Rysunek 3.6: Punkt centralny strony logowania



A login form titled "Login" with two input fields: "USERNAME" and "PASSWORD". Below the fields is a link "FORGOT YOUR PASSWORD?". At the bottom are two buttons: "REGISTER" and "SIGN IN". The form is centered on a dark background with colorful circular accents. In the top left corner is a "Language" button, and in the top right corner is a "Change mode" button.

Rysunek 3.7: Strona logowania po angielsku i w trybie ciemnym



A home page with a header bar containing "File", "Language", "Hi, UserName!", "Change mode", and "Logout". Below the header is a navigation bar with "Classes", "Instances", "Object properties", and "Data properties". The main content area is divided into two panels. The left panel, titled "Class hierarchy", shows a tree structure of classes: "Beverages" (Water, Coffee, Tea), "Black Coffee", "White Coffee", "Black Tea", "White Tea", "Green Tea", "Sencha", "Gyokuro", "Matcha", and "Pi Lo Chun". The right panel, titled "Class description: Water", shows relationships: "Equivalent to", "SubClass of", and "Instances".

Rysunek 3.8: Strona domowa

Rozdział 4

Implementacja oraz przedstawienie wersji demonstracyjnej

4.1 Klasy

Klasy zostały wykonane w oparciu o projekt z diagramu klas (Rys. 3.2), przy użyciu języka python3 [33] i biblioteki Owlready2 [34]. W pliku file.py znajduje się klasa `Ontology`, której obiekty posiadają swoją nazwę i ontologię.

Listing 4.1: Przykładowe metody klasy `Ontology`

```
class Ontology:
    def __init__(self, file_name, ontology):
        self.name = file_name
        self.ontology = ontology

    def save_on_server(self, user_name):
        file_path = os.path.join(os.path.abspath(
            os.getcwd()), "data", user_name, "")
        if os.path.exists(os.path.dirname(
            file_path + self.name)):
            return HttpResponse(status=409)

        if not os.path.exists(os.path.dirname(
            file_path)):
            try:
                os.makedirs(os.path.
                    dirname(file_path))
            except OSError as exc:
```

```

        if exc.errno != errno.
            EEXIST:
                raise
    self.ontology.save(os.path.join(
        file_path, self.name), format="rdxml
    ")

    def load_from_server(self, user_name, file_name)
        :
        self.name = file_name
        file_path = os.path.join(os.path.abspath
            (os.getcwd()), "data", user_name,
            file_name)
        self.ontology = get_ontology(file_path).
            load()

    def file_rename(self, user_name, new_file_name):
        file_path_old = os.path.join(os.path.
            abspath(os.getcwd()), "data",
            user_name, self.name)
        file_path_new = os.path.join(os.path.
            abspath(os.getcwd()), "data",
            user_name, new_file_name)
        if not os.path.exists(os.path.dirname(
            file_path_old)):
            self.name = new_file_name
        else:
            os.rename(file_path_old,
                file_path_new)

```

W pliku ontology.py znajdują się wszystkie inne klasy, które zostały zaprojektowane i są potrzebne do zarządzania ontologiami. Każda z nich posiada obiekt klasy Ontology. Oparte są one na dynamicznym tworzeniu nowych klas. W implementacji brakuje usuwania wszystkich podklas, oraz pod własności obiektów/danych.

Listing 4.2: Przykładowe metody klasy ClassMan

```

class ClassMan:
    def __init__(self, file: Ontology):
        self.file = file

    def create_subclass(self, subclass_name,
        main_class):

```

```

        with self.file.ontology:
            if main_class == "Thing":
                new_class = types.
                    new_class(
                        subclass_name, (eval(
                            main_class),))
            else:
                new_class = types.
                    new_class(
                        subclass_name, (self.
                            file.ontology[
                                main_class],))

    def create_sibling_class(self,
        sibling_class_name, main_class):
        with self.file.ontology:
            new_class = types.new_class(
                sibling_class_name, (list(
                    self.file.ontology[main_class]
                        .is_a)[0],))

```

4.2 Wersja demonstracyjna

Do stworzenia wersji demonstracyjnej nowoczesnego ODE dla nowych użytkowników został użyty framework django [35], który jest wykorzystywany do tworzenia stron internetowych (część back-endowa) w języku python3. Strony zostały napisane w języku znaczników jakim jest HTML, kaskadowych arkuszy stylów (CSS), oraz skryptów napisanych w JavaScriptcie. Wersja testowa pokazuje interfejs, oraz system autoryzacji, który jest częścią django - posiada własny plik bazy danych sqllite, oraz metody do logowania, rejestracji, wylogowania i sprawdzania, czy użytkownik jest zalogowany. Aktualna implementacja pozwala na skorzystanie z 5 rozwinięć linków.

Listing 4.3: Paterny URL

```

urlpatterns = [
    path('login/' ,views.loginPage , name="login"),
    path('logout/' ,views.logoutUser , name="logout")
    ,
    path('register/' ,views.registerPage , name="
        register"),
    path('', views.home, name="home"),

```

```

        path( 'login_pl/' , views.loginPagePl , name=
            login_pl" ) ,
    ]

```

Pierwszą stroną jaka pokazuje się po uruchomieniu serwera developerskiego jest logowanie (Rys. 3.5), użytkownik jest w stanie zmienić język z angielskiego na polski(i z polskiego na angielski). Zmiana aktualnie działa tylko na stronie logowania. Po naciśnięciu przycisku rejestracji okno login zamienia się na okno z rejestracją. Funkcja generująca widok logowania, w polu kontekstowym, wysyła wiadomość jak ma być renderowana strona(logowanie lub rejestracja), a następnie dzięki warunkowi if w pliku login.html, wyświetla odpowiedni element na stronie.

Listing 4.4: Przykład okna rejestracji w htmlu

```

{% if page == 'login' %}
.
.
.
{% else %}

<div class="circlebig"></div>
<div class="circlesmall"></div>
<div class="login">
    <form method="POST" action="">
        {% csrf_token %}
        <h1>Register</h1>
        <div class="content">
            <div class="input">
                {{ form.username }}
            </div>
            <div class="input">
                {{ form.password1 }}
            </div>
            <div class="input">
                {{ form.password2 }}
            </div>
            <a href="{% url 'login' %}"
                class="link">You already have
                an account?</a>
        </div>
        <div class="action">
            <div class="register">
                <button>Sign up</button>

```

```
                                </div>
                        </div>
</form>
{% if messages %}
<ul class="messages">
    {% for message in messages %}
    <li>{{ message }}</li>
    {% endfor %}
</ul>
{% endif %}
</div>
```

Po zalogowaniu użytkownik zostaje przekierowany na widok domowy (Rys. 3.8). Na tej stronie jedyną funkcjonalnością jest możliwość wylogowania, która przekieruje na widok logowania.

Listing 4.5: Przykład funkcji wylogowania

```
def logoutUser(request):
    logout(request)
    return redirect('login')
```


Zakończenie

W pracy zostały opisane języki, przy użyciu których tworzy się ontologie, ich historia, metodologie podzielone na mikro, oraz makro, metodyki takie jak np. OntoClean, a także działanie edytora Protege. Dzięki ankiecie, która trafiła do wielu osób(50), można było wyciągnąć wnioski, które pomogły w lepszym zrozumieniu potrzeb klienta. Następnie zostały zebrane wymagania, możliwości rozwoju, wstępnie oszacowany ruch, wraz z ilością danych jakie są wymagane na pliki. Został stworzony diagram klas, ogólny diagram przypadków użycia, oraz diagram aktywności. Wykonany został projekt nowoczesnego gui, oraz implementacja klas, wraz ze stworzeniem wersji demonstracyjnej interfejsu, przy użyciu django.

Stworzenie oprogramowania, które trafia w gusta każdego klienta jest niemożliwe. Zawsze trzeba decydować się na różnego typu rozwiązania, oraz ustępstwa, dzięki czemu jest duża różnorodność na rynku, i każdy użytkownik jest w stanie znaleźć coś dla siebie. Projekt nowoczesnego ODE dla początkujących na pewno jest w stanie trafić w pewną niszę na rynku, gdzie ludzie oczekują ciągłego dostępu do swoich ontologii, i zwracają uwagę na wygląd oprogramowania, natomiast nie potrzebują wielu zbędnych funkcjonalności, których zapewne nigdy nie użyją. Demo, które zostało zaprezentowane potwierdziło użyteczność, oraz przydatność funkcji takich jak np. zmiana języka.

Spis tabel

3.1	Rozwinięcie pkt. z FURPS (źródło: [30])	25
-----	---	----

Spis rysunków

1.1	Przykład użycia KIF (źródło: [9])	10
1.2	Struktura OWL2 (źródło: [11])	11
1.3	Trójka RDF	11
1.4	Tworzenie ontologii (źródło: [17])	13
1.5	Przykłady pytań koncepcyjnych (źródło: [18])	14
1.6	Metodologia koła życia (źródło: [19])	15
1.7	Diagram pomagający przy wyborze języka (źródło: [21])	16
1.8	Tabela pułapek (źródło: [27])	17
2.1	Protege	20
2.2	Wyniki przeprowadzonej ankiety	24
3.1	model FURPS	26
3.2	Diagram klas	30
3.3	Diagram przypadków użycia	31
3.4	Diagram aktywności	32
3.5	Strona logowania	32
3.6	Punkt centralny strony logowania	33
3.7	Strona logowania po angielsku i w trybie ciemnym	33
3.8	Strona domowa	33

Listings

4.1	Przykładowe metody klasy Ontology	35
4.2	Przykładowe metody klasy ClassMan	36
4.3	Paterny URL	37
4.4	Przykład okna rejestracji w htmlu	38
4.5	Przykład funkcji wylogowania	39

Bibliografia

- [1] Ewelina Stój. “Ile miejsca zajmuje dzisiejszy Internet? Ok. 10 bilionów TB”. In: *PurePC* (2018). URL: <https://www.purepc.pl/ile-miejsca-zajmuje-dzisiejszy-internet-ok-10-bilionow-tb>.
- [2] W3. “OWL/Web Ontology Language (OWL)”. In: *W3* (2012). URL: <https://www.w3.org/OWL/>.
- [3] Encyklopedia PWN. *ontologia*. URL: <https://encyklopedia.pwn.pl/haslo/ontologia;3951174.html>.
- [4] Adam Czarnecki. *Technologie informatyczne wykorzystywane w projektowaniu i implementacji ontologii*. Pomorskie Wydawnictwo Naukowo-Techniczne PWNT, 2006.
- [5] Władysław Stróżewski. *Ontologia*. Kompendia filozoficzne. Kraków: Aureus, 2004. ISBN: 8387887447.
- [6] V Mascardi, V Cordi, P Rosso. “A Comparsion of Upper Ontologies”. In: (2007). URL: <https://web.archive.org/web/20171113222220/https://pdfs.semanticscholar.org/4f28/6fdf9280449588b9d3781c9c897da28e0cff.pdf>.
- [7] Rajalaxmi-R. Priya P. “Ontology based semantic query suggestion for movie search”. In: *2013 International Conference on Information Communication and Embedded Systems (ICICES)*. 2013. DOI: 10.1109/ICICES.2013.6508326.
- [8] Protege. “Protege - About”. In: *Protege* (2016). URL: <https://protege.stanford.edu/about.php>.
- [9] Michael Gruninger. URL: https://images.slideplayer.com/24/6956543/slides/slide_1.jpg.
- [10] H. Glaser et al. “CS AKTive Space, or How We Learned to Stop Worrying and Love the Semantic Web”. In: *IEEE Intelligent Systems* 6.03 (2004), pp. 41–47. ISSN: 1941-1294. DOI: 10.1109/MIS.2004.8.

- [11] “OWL 2 Web Ontology Language Document Overview (Second Edition)”. In: (2012). URL: <https://www.w3.org/TR/owl2-overview/#Introduction>.
- [12] C. Maria Keet. *An Introduction to Ontology Engineering*. v1.5. 2020.
- [13] W3. *Resource Description Framework*. URL: <https://www.w3.org/TR/PR-rdf-syntax/Overview.html>.
- [14] Gavin Carothers Eric Prud’hommeaux. “Turtle - Terse RDF Triple Language”. In: (2012). URL: <https://www.w3.org/TR/2012/WD-turtle-20120710/>.
- [15] Tim Berners-Lee. “Uniform Resource Identifiers (URI): Generic Syntax”. In: (1998).
- [16] Anna Węgrzyn. *Konceptualizacja*. URL: <https://mfiles.pl/pl/index.php/Konceptualizacja>.
- [17] Mari Carmen Suarez-Figueroa, Guadalupe Aguado de Cea, Carlos Buil, Klaas Dellschaft, Mariano Fernandez-Lopez, Andres Garcia, Asuncion G’omez-P’erez, German Herrero, Elena Montiel-Ponsoda, Marta Sabou, Boris Villazon-Terrazas, and Zheng Yufei. *NeOn methodology for building contextualized ontology networks*. v1.0. 2008.
- [18] Robert Stevens, Sean Bechhofer. “Competency Questions”. In: (2013). URL: <https://studentnet.cs.manchester.ac.uk/pgt/2013/COMP60421/slides/Week2-CQ.pdf>.
- [19] Amanda Vizedom et al. “Toward Ontology Evaluation across the lifecycle”. In: *Applied ontology* 8 (Oct. 2013), pp. 179–194. DOI: 10.3233/AO-130125.
- [20] Anna Baron-Jaworska. “Co to jest Agile i Scrum i dlaczego warto z nich korzystać przy projektowaniu systemów IT?” In: (2021). URL: <https://kotrak.com/pl/blog/co-to-jest-agile-i-scrum-i-dlaczego-warto-z-nich-korzystac/>.
- [21] C. Maria Keet. *Transforming semi-structured life science diagrams into meaningful domain ontologies with DiDOn*. 2012.
- [22] Douglas B. Lenat, R. V. Guha. *Building large knowledge-based systems; representation and inference in the Cyc project*. 1989.
- [23] Perakath C. Benjamin et al. “IDEF5 Method Report”. In: (1994).
- [24] Nicola Guarino, Chris Welty. *A formal ontology of properties*. 2000.
- [25] Nicola Guarino, Chris Welty. *Identity, unity, and individuality: towards a formal toolkit for ontological analysis*. 2000.
- [26] Guarino, Nicola and Chris Welty. “Evaluating Ontological Decisions with On-toClean”. In: (2002).

- [27] Mar'ia Poveda-Villal'on, Mari Carmen Su'arez-Figueroa, Asunci'on G'omez-Perez. *Validating ontologies with OOPS!* 2012.
- [28] Protege. "Protege - Community". In: *Protege* (2016). URL: <https://protege.stanford.edu/community.php>.
- [29] Waniowski Paweł. *Strategie cenowe*. Warszawa: Polskie Wydawnictwo Ekonomiczne, 2003. ISBN: 83-208-1447-2 (35+10).
- [30] Tomasz Tomaszewski. "Metoda FURPS, czyli 29 rzeczy do przemyślenia w każdym projekcie IT". In: (2014). URL: <https://www.analizait.pl/2014/metoda-furps-czyli-29-rzeczy-do-przemyslenia-w-kazdym-projekcie-it/>.
- [31] Kacper Wojcik. "ODE". In: (2022). URL: <https://github.com/KWojcik243/ODE>.
- [32] URL: <https://protege.stanford.edu/ontologies/pizza/pizza.owl>.
- [33] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.
- [34] Lamy JB. "Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies". In: (). URL: http://www.lesfleursdunormal.fr/static/_downloads/article_owlready_aim_2017.pdf.
- [35] *Django*. URL: <https://www.djangoproject.com/>.