POLITECHNIKA KRAKOWSKA
IM. TADEUSZA KOŚCIUSZKI
WYDZIAŁ FIZYKI MATEMATYKI I INFORMATYKI
KIERUNEK INFORMATYKA

MICHAŁ BĄK

APLIKACJA WSPOMAGAJĄCA PRZEPROWADZENIE POWTÓREK W PROCESIE UCZENIA SIĘ STWORZONA NA PODSTAWIE MODELU SIECI SEMANTYCZNEJ

PRACA MAGISTERSKA
STUDIA STACJONARNE

Promotor: dr inż. arch. Paweł Ozimek

Kraków 2012

Spis treści

Wprowadzenie

We wczesnych etapach projektowania systemów informatycznych potrzebne jest dokładne zrozumienie wybranego fragmentu rzeczywistości, który dany system będzie odwzorowywał. Metodyki projektowania systemów zalecają stworzenie modelu biznesowego tej rzeczywistości, który w sposób sformalizowany opisze zdobytą o niej wiedzę. Sieć semantyczna pozwala na stworzenie modelu wybranego fragmentu rzeczywistości. Technologia opracowana przez Rossa Quilliana jest ontologiczną metodą reprezentacji wiedzy. Quillian założył, że pamięć ludzką najlepiej opisuje model asocjacyjny. Oznacza to, że dane terminy mogą być wyjaśnione przez inne terminy. W taki sposób powstaje struktura powiązań. Sieć semantyczna jest zbiorem obiektów powiązanych ze sobą różnymi relacjami. Stanowi ona graficzną reprezentację pewnego rodzaju logiki, gdzie relacje między obiektami są przedstawione w postaci grafu, w którym obiekty to węzły, a relacje to łuki. Zaletą sieci semantycznych jest elastyczność, możliwość przeprowadzenia wnioskowania, tworzenie modelu na podstawie danych pozyskanych z zewnątrz oraz łączenie ze sobą różnych ontologii.

1.1 Cel pracy

Celem pracy magisterskiej jest stworzenie sieci semantycznej języka PHP przy użyciu specjalnie w tym celu wykonanej aplikacji w architekturze MVC wspomagającej proces poznawania języka.

1.2 Zakres pracy

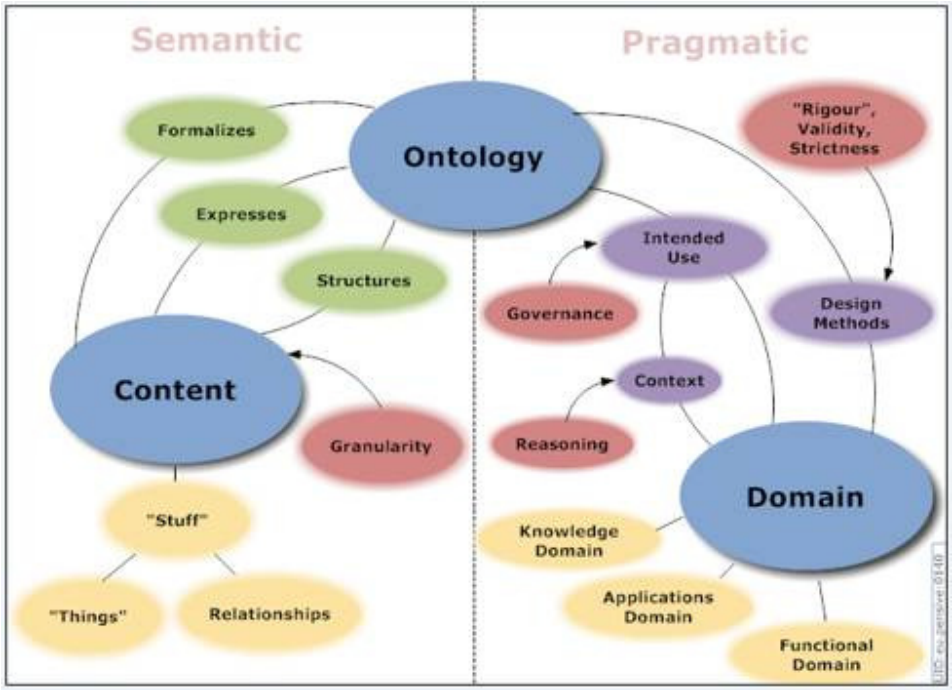
W pracy magisterskiej zostały omówione zagadnienia sieci semantycznej oraz ontologii w informatyce. Przedstawiono języki do ich opisu i aplikacje wspomagające ich tworzenie. Poruszono zagadnienia związane z architekturą MVC i Frameworkiem CakePHP. Jako przykład została zbudowana sieć semantyczna przedstawiająca relację miedzy obiektami w modelu MVC. Następnie stworzono aplikację, która wspomaga przeprowadzenie powtórek w procesie uczenia się. Relacyjna baza danych języka PHP wbudowana do aplikacji stanowiła podstawę do wygenerowania trzonu sieci semantycznej języka. Trzon ten został następnie rozbudowany do wielopoziomowej sieci semantycznej języka dzięki wykorzystaniu aplikacji do powtórek.

2. Ontologia, Sieć semantyczna

2.1 Ontologia w informatyce

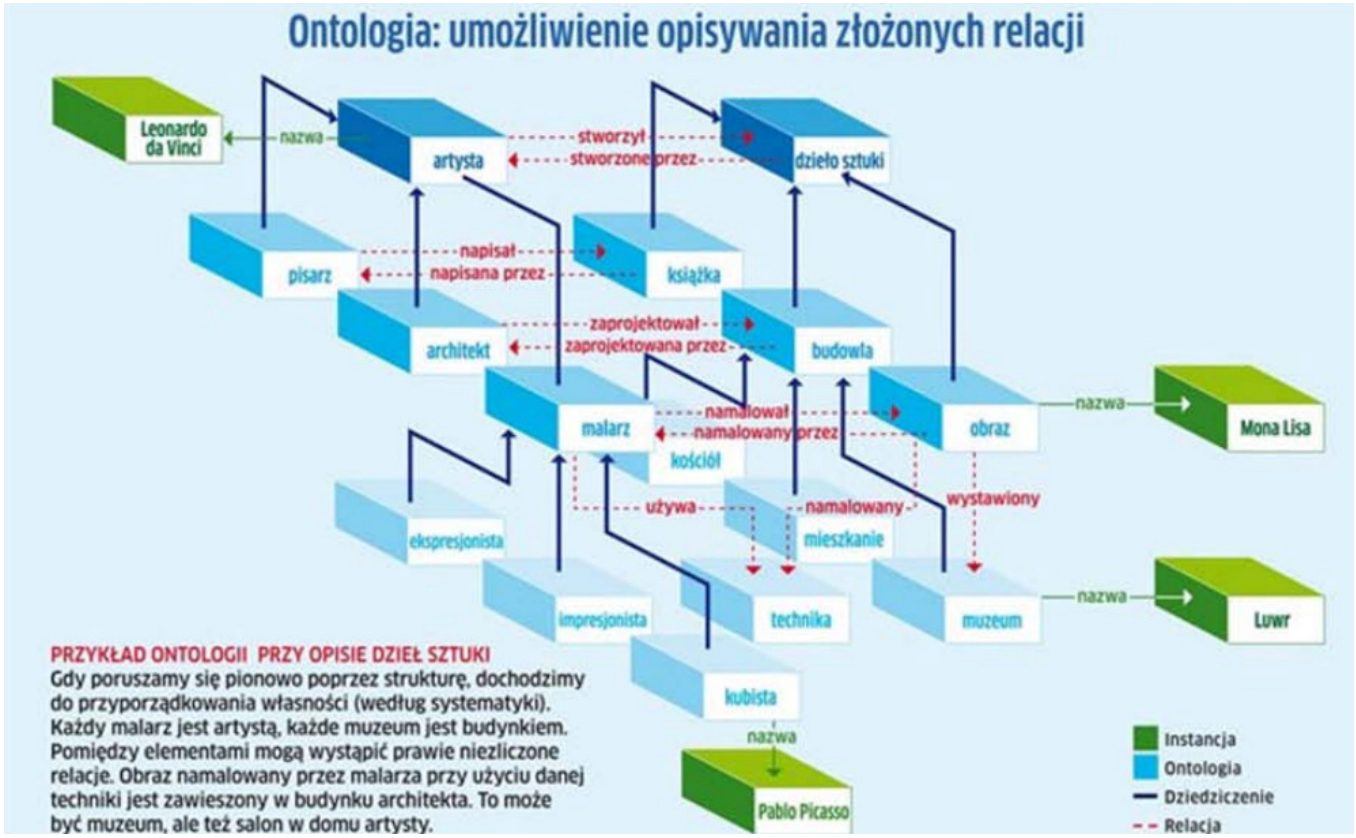
Słowo ontologia wywodzi się z filozofii, stanowi dział zajmujący się badaniem struktury rzeczywistości. W informatyce ma ona inne znaczenie. Najczęściej przytaczaną definicją ontologii jest ta sformułowana przez Thomasa Grubera: „ontologia jest formalną specyfikacją wspólnej warstwy pojęciowej" cytat z książki „Ontologie w systemach informatycznych" [4]. Słowo formalny oznacza, że ontologia powinna być łatwo przetwarzana przez komputer i czytelna dla ludzi. Słowo specyfikacja informuje, że definicje pojęć i relacji muszą być jednoznacznie sformułowane. Wiedza zawarta w ontologii powinna być akceptowana przez użytkowników. Warstwa pojęciowa jest modelem wycinku świata rzeczywistego, w którym występują pojęcia, obiekty i relacje, np. modelem jest wypożyczalnia wideo, a elementami w nim wstępującymi są obiekty: film, klient, i relacje wypożyczenie i zwrot filmów przez klienta. Słowo wspólne oznacza, że ontologie mogą być łączone ze sobą.

Ontologie w informatyce zazwyczaj prezentuje się w postaci schematu blokowego połączonego relacjami. Rysunek poniżej przedstawia ontologię, która ma dwie możliwości: semantyczną i pragmatyczną.



Rysunek 1 Przykład ontologii, ilustracja pochodzi z artykułu „A Basic Guide to Ontologies" [1]

Dobrze zbudowana ontologia pozwala na zrozumienie słów szukanych przez użytkownika. Rysunek poniżej przedstawia przykład otologii o sztuce, która prezentuje proces wnioskowania.

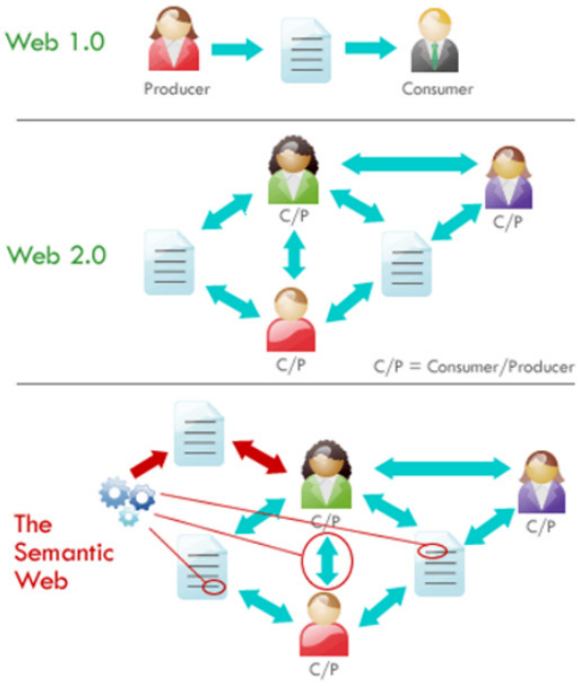


Rysunek 2 Ontologia: umożliwienie opisania złożonych relacji, ilustracja pochodzi z artykułu „Semantyczna Sieć: Boty uczą się kojarzyć" [2]

W informatyce ontologie są coraz bardziej popularne, ale bardzo trudno je stworzyć. Człowiek nie jest w stanie sam ich stworzyć bez odpowiednich narzędzi. Problemem zajmą się aplikacje analityczne. W obecnej chwili bazę danych strony WWW uzupełnia bot, program, który szuka strony. W erze działania sieci semantycznych strony internetowe będą przeanalizowane przez programy analityczne, następnie sklasyfikowane trafią do bazy danych. Słowa szukane za pomocą wyszukiwarki internetowej zostaną poddane analizie w celu zrozumienia ich tematu i wybraniu odpowiedniej ontologii. Zamiast katalogów, które są stosowane obecnie, będą używane konkretne ontologie. W Internecie materiałów ciągle przebywa. Ontologie będą musiały być ciągle aktualizowane. Dla człowieka jest to zadanie praktycznie niemożliwe. Tworzeniem i ich aktualizacją zajmie się program. W ten sposób ontologie rozwijałyby się same. Człowiek kontrolowałby cały proces. Twórcy zasobów mogliby pomagać przez opisanie ontologii metadanymi. Dzięki temu program analityczny otrzymałby orientację tematyczną. Jednak niektórzy użytkownicy opisywaliby swoje zasoby błędnie. Ontologie będą również budowane przez twórców, nie będzie do nich dostępu globalnego albo będą dostępne za opłatą. Budowa ontologii to bardzo poważny problem. Wymyślenie dobrego rozwiązania zajmie dużo czasu.

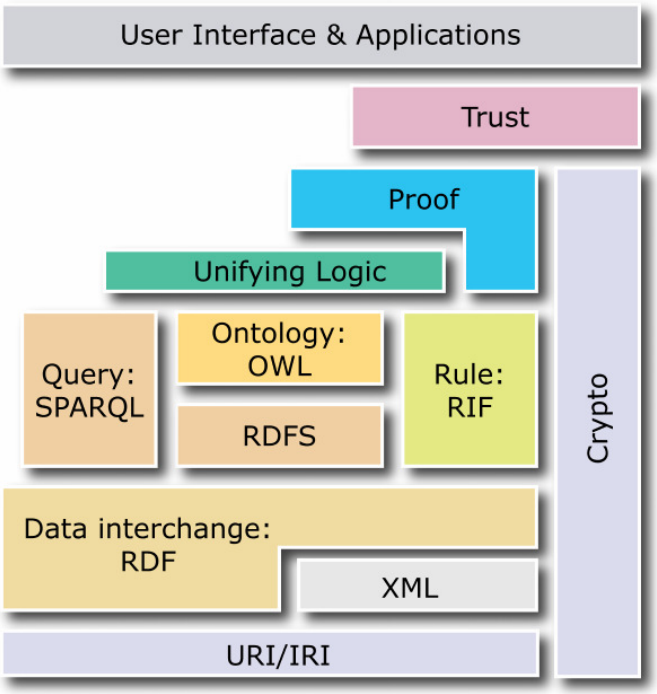
2.2 Sieć Semantyczna

Sieć semantyczna pozwala na nową komunikację w Internecie, która pozwala systemom zrozumieć użytkownika. Rysunek poniżej przedstawia schemat komunikacji w Web 1.0, Web 2.0 i Sieci semantycznej.



Rysunek 3 Schemat komunikacji Web 1.0, Web 2.0 i Sieci semantycznej

Z siecią semantyczną wiążą się technologie, które pozwalają na utworzenie standardów zawartych w sieci. Rysunek poniżej przedstawia semantyczny tort.



Rysunek 4 Semantyczny tor, ilustracja pochodzi z artykułu „Semantyczny tort: jak przełknąć wizję Web 3.0 ” [3]

Podstawą sieci jest technologia, która pozwala na wskazanie dowolnej treści i obiektu oraz opisanie ich w dowolnym języku. Jest to URI (Uniform Resource Identifer). Adres URI jest podobny do URL, różni się od niego tym, że pozwala na wskazanie przedmiotów w świecie rzeczywistym (teoretycznie), również ich opisanie i połączenie z dokumentami w Internecie. Adres IRI (Internationalized Resource Identifier) rozszerza URI o możliwość używania znaków spoza standardu ASCII. Kolejną warstwa to standard XML. W XMLu można zapisać informacje tak, aby maszyna mogła je przetworzyć, ale nie zrozumieć. Warstwą wyżej zdefiniowaną jest standard RDF (Resource Description Framework). Jest on nieodłącznie związany z Siecią Semantyczną. W odróżnieniu od XMLa umożliwia on zapis grafu (nieskierowanego) w postaci przetwarzanej przez maszyny. Kolejne dwie warstwy związane są z definiowaniem znaczenia pojęć wykorzystanych w opisie grafu RDF czyli tzw. ontologii. RDF Schema i OWL pozwala na definiowanie ontologii za pomocą hierarchii klas i właściwości. Warstwa Rule jest odpowiedzialna za reguły, które pozwalają na definiowanie reguł przetwarzania wiedzy zapisanej w RDF i ontologiach. Istotne jest, że ontologie, reguły i logika są jedną warstwą. Język SPARQL przeznaczony jest do wydobywania informacji z semantycznej sieci. Warstwa Crypto zapewnia bezpieczeństwo operacji w Sieci Semantycznej. Ostatnią warstwą jest interfejs użytkownika, który ma ogromne znaczenie w interakcjach użytkownik z systemem opartym o rozwiązania semantyczne.

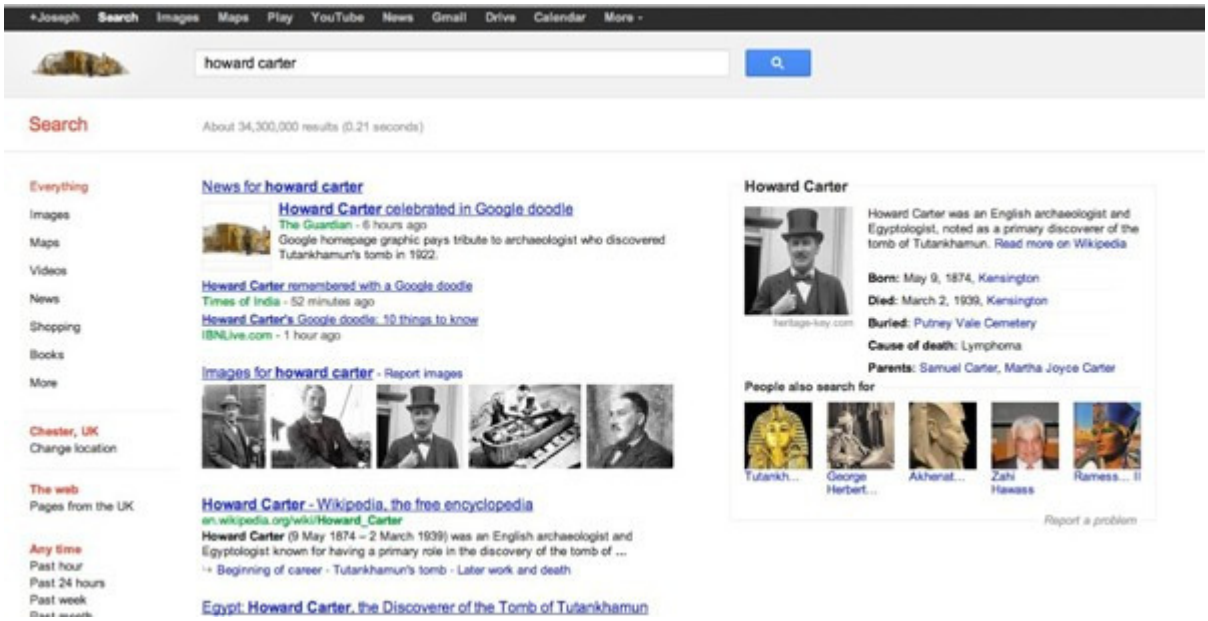
Sieć semantyczna będzie korzystać z ontologii w celu realizacji procesu wnioskowania. Umożliwią one znalezienie powiązań między słowami, które będą zrozumiałe dla komputera. Obecnie powstają wyszukiwarki bazujące na technologii sieci semantycznej. Wyszukiwarka Swoogle wykorzystuje technologie sieci semantycznych. Posiada ona dziesięć tysięcy ontologii. Wyszukiwarka ta nie wyszukuje stron internetowych. Korzysta ona ze swoich zasobów, niestety są one niskie i zapisane są w różnym formacie. Ilustracja poniżej przedstawia interfejs wyszukiwarki.

Ilustracja 1 Wyszukiwarka Swoogle

Lepszą wyszukiwarką opartą na wyszukiwaniu semantycznym jest Baidu Box Computing. Wyszukiwarka chińska przegląda strony internetowe i potrafi udzielić odpowiedzi na proste pytania. Szacuje się, że około 70% wyszukiwań zwraca dopowiedź prawidłową. Ilustracja poniżej przedstawia przykład zapytania w wyszukiwarce Baidu.

Ilustracja 2 Wyszukiwarka zapytana po chińsku zwróciła zadawalającą odpowiedź

Firma Google buduje wyszukiwarkę, która będzie korzystać z technologii semantycznych. Pierwowzór tej wyszukiwarki można zobaczyć już teraz. Na zadane pytanie w odpowiedzi zostaje przedstawiony boks zawierający kluczowe informacje: nazwy, liczby, daty. Ilustracja poniżej przedstawia przykład zapytania w wyszukiwarce.



Ilustracja 3 Wyszukiwarka Google

Semantyczna wyszukiwarka będzie analizowała otocznie słów kluczowych, analizując kontekst, w jakim się one pojawiają a nie tylko pojedyncze wyrazy.

2.3 Języki do opisu sieci semantycznej

W rozdziale zostały opisane języki do tworzenia ontologii oparte na meta języku XML.

2.3.1 XML

Język XML (eXtensible Markup Language) jest standardem W3C, służy do opisu dokumentów za pomocą znaczników. Jest on jedną z warstw semantycznego tortu str.
8 rysunek 4. Dokumenty XML są łatwo przetwarzane przez komputery i są czytelne dla ludzi. Format XML pozwala na łatwe przechowywanie dowolnych danych. Programy mogą dzięki wspólnemu formatowi XML łatwiej wymieniać dane.
Przykładowa składnia języka XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Film>
<tytul>Milczenie owiec</tytul>
</Film>
```

Na początku znajduje się prolog dokumentu, między znakami <? ?>, w którym umieszcza się deklarację, np. wersja XML. W dokumencie XML dane przechowywane są w sposób tekstowy, opisane przez znaczniki. Znacznik rozpoczyna lewy nawias kątowy <, a kończy prawy >.
Kod XML może posiadać także atrybuty:

```
<Film tytul=”Milczenie owiec”> </Film>
```

Klasą jest *Film* a atrybutem jest *tytul.* Wartości atrybutów ujmuje się w cudzysłów. Powyższy przykład można zapisać:

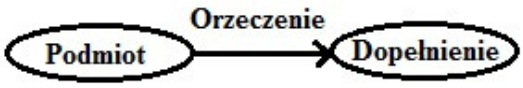
<?xml version="1.0" encoding="UTF-8"?>

<Film tytul="Milczenie owiec"/>

Język XML pozwala na prezentowanie danych w strukturalny sposób. Pozwala na definiowanie i używanie własnych znaczników. Języki ontologii są oparte na znacznikach XML.

2.3.2 RDF

RDF (Resource Description Framework) jest językiem opartym na języku XML. Powstał jako standard kodowania meta danych, czyli danych służących do opisu innych danych. Model RDF przedstawia się za pomocą: podmiotu (predykat), orzeczenia (właściwość), i dopełnienia (wartość). Podmiot jest obiektem, do którego można się odwołać przez URL (Uniform Resource Locator) lub URI (Uniform Resource Identifier). Orzeczenie jest relacją między zasobami. Dopełnienie oznacza wartość dla podmiotu. Zagadnienie jest opisane w dokumencie *Resource Description Framework (RDF):Concepts and Abstract Syntax* [5]. Rysunek przedstawia graf trójki RDF.



Rysunek 5 Model RDF

Poniżej został przedstawiony fragment kodu w języku, który został stworzony na podstawie tabelki:

Tytuł	Rok	Aktor	Reżyser
Ojciec chrzestny	1976	Marlon Brando	Francis Ford Coppola
Leon zawodowiec	1994	Jean Reno	Luc Besson

Tabela 1

<?xml version="1.0"?>

<rdf: RDF

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

xml:film="http://www.film”>

<rdf: Description

rdf:obout="http://www.film / ojciecchrzestny">

<film:rok>1976</ film:rok>

<film:aktor>Marlon Brando</ film:aktor>

<film:rezyser>Francis Ford Coppola</ film:rezyser>

</rdf: Description>

<rdf: Description

rdf: obout ="http://www.film / Leonzawodowiec">

<film:rok>1994</ film:rok>

<film:aktor>Jean Reno</ film:aktor>

<film:rezyser>Luc Besson </ film:rezyser>

</rdf: Description>

</rdf:RDF>

W pierwszej linijce kodu RDF znajduje się wersja XML. W drugim wierszu element rdf:RDF określa typ dokutemu. Wyrażenie *xmlns:rdf* oznacza, że znaczniki z ciągiem znaków *rdf:* pochodzą z przestrzeni nazw *http://www.w3.org/1999/02/22-rdf-syntax-ns#*. Element *xmlns:film* określa, że znaczniki z ciągiem znaków *film*: należą do przestrzeni *http://film.pl*. Są to właściwości: *rok*, *aktor*, *rezyser*. W przykładzie podmiotem jest *http://www.film / ojciec chrzestny*, który zawiera właściwość rok, o wartość 1976, tłumacząc film „Ojciec chrzestny” jest z roku 1976.

Właściwości mogą być zdefiniowane jako atrybuty. Są one umieszczone wewnątrz znaczników.

Przykładowy kod w języku RDF:

<?xml version="1.0"?>

<rdf: RDF

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

xml:film="http://www.film”>

<rdf: Description

rdf:obout="http://www.film / Leonzawodowiec">

film:rok=”1994”

film:actor=”Jean Reno”

film:rezyser=”Luc Besson”>

</rdf: Description>

</rdf:RDF>

Wartości można zapisać za pomocą przestrzeni nazw np. *<film:aktor rdf:abuat=" http://www.film / ojciec chrzestny/aktor/marlonbrando/>*.Właściwość aktor nie posiada wartości, ale odwołuje się do zasobu, w którym znajduje się informacja o aktorze.

RDF pozwalana na kodowanie metadanych. Stanowi on jeden z filarów Semantycznego Internetu.

2.3.2 RDFS

RDFS(RDF Schema) jest rozszerzeniem języka RDF. Pozwala ono na budowanie ontologii. Język umożliwia tworzenie hierarchii klas i właściwości za pomocą wyrażeń *rdf:subClassOf*, *rdf:subPropertOf*. RDFS pozwala na określenie powiązań między klasami, za które odpowiadają wyrażenia *rdf:domain* i *rdf:range*. Możliwe jest także definiowanie zasobów klas oraz określenie typu za pomocą *rdf:type*. Język RDFS został dokładnie opisany w dokumentacji W3C *RDF Vocabulary Description Language 1.0: RDF Schema* [6]. Cechy języka RDFS zostały przedstawione w kolejnym podrozdziale OWL, który jest jego rozszerzeniem.

Przykładowy zapis w RDFS:

```
<?xml version="1.0"?>
<rdf: RDF
xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:film="http://www.wypozyczalnia/osoba”>
<rdf: Description rdf:ID=”Osoba”>
        <rdf:type rdf:resource=”http://www.w3.org/2000/01/rdf-schema#Class”/>
</rdf: Description>
<rdf: Description rdf:ID=”Klient”>
        <rdf:type rdf:resource=” http://www.w3.org/2000/01/rdf-schema#Class”/>
        <rdf:subClassOf rdf:resource=”Osoba”>
</rdf: Description>
</rdf:RDF>
```

Powyższy przykład można skrócić używając *rdf:Class* zamiast *rdf:Description*

```
<?xml version="1.0"?>
<rdf: RDF
xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
```

xml:film="http://www.wypozyczalnia/osoba”>

<rdf: Class rdf:ID=”Osoba”>

<rdf: Class rdf:ID=”Klient”>

<rdf:subClassOf rdf:resource=”Osoba”>

</ rdf: Class >

</rdf:RDF>

W przykładzie *Klient* jest podklasą *Osoba.*

Język RDFS jest bardziej złożony od swojego poprzednika RDF. Za jego pomocą można już tworzyć proste ontologie.

2.3.3 OWL

OWL (Web Ontology Language) jest językiem Ontologii, zaprojektowanym dla aplikacji, które przetwarzają treść informacji, a nie tylko ją prezentują. Język OWL jest częścią wizji projektu Sieci Semantycznej, która jest oparta na meta języku XML do definiowania schematów znakowania, oraz do przedstawienia danych przez języki RDFS i OWL. Można wyróżnić trzy odmiany języka OWL: OWL Lite, OWL DL, OWL Full. Każdy z tych języków jest poszerzeniem swojego poprzednika. Zachodzi lista powiązań między tymi językami: ontologia OWL Lite jest ontologią dla OWL DL, ontologia OWL DL jest ontologią dla OWL Full.

Język OWL jest częścią stosu rekomendacji W3C: XML, Schemat XML, RDF, Schemat RDF (RDFS). OWL został stworzony na podstawie języka RDFS i jest jego rozszerzeniem. Poniżej opisałem cechy OWL i RDFS z przykładami, które zostały utworzone na podstawie dokumentacji: *RDF Vocabulary Description Language 1.0: RDF Schema* i *OWL Web Ontology Language Overview* [6, 7] i projektu „Sieć semantyczna, wypożyczalnia wideo” w programie Protege.

Class

Class (klasa), określa grupę jednostek, które stanowią całość, np. klienci zapisani do wypożyczalni stanowią grupę, która należy do klasy *Klient*.

Przykład klasy w języku OWL:

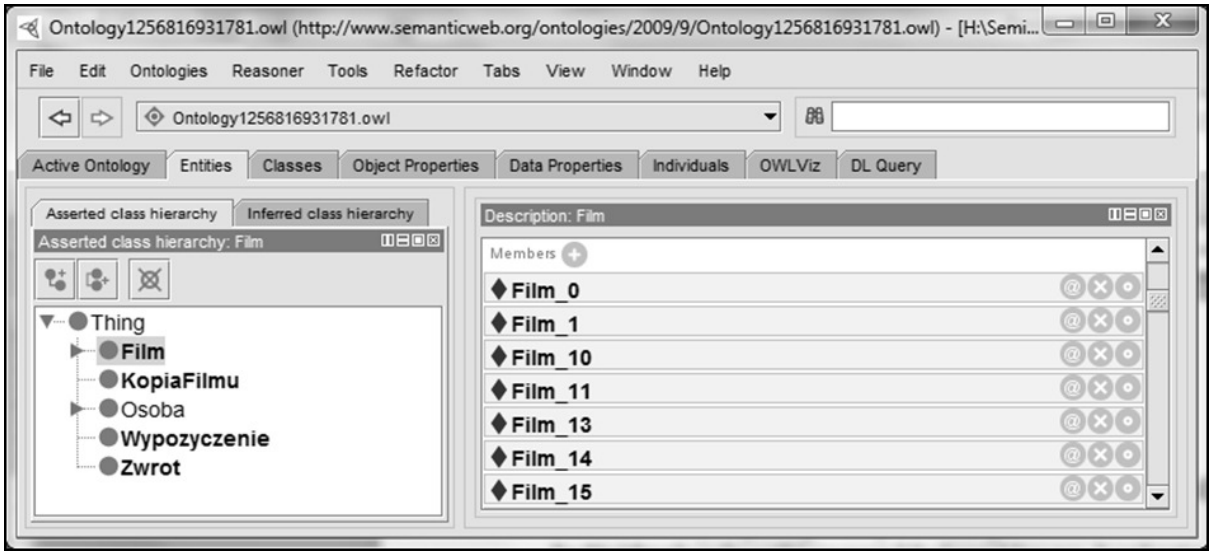
<owl:Class rdf:about="#Klient"/>

Klasy mogą być organizowane w hierarchie na przykład klasy: *Osoba, Obsada, Aktor, Reżyser*. W programowaniu obiektowym obiekt jest członkiem dokładnie jednej klasy. Przynależność ta jest z góry ustalona w czasie tworzenia obiektu i się nie zmienia. W RDF Schema i OWL jednostki mogą należeć do wielu klas, które są traktowane jak

zbiory jednostek a nie typy, jak w programowaniu obiektowym, i mogą zmieniać swoją przynależność ze względu na właściwość np. zakres, rangę. Języki programowania obiektowego nie pozwalają na ewolucję specyfikacji klas w czasie wykonywania programu. RDF Schema, OWL zostały stworzone, aby umożliwić integrację danych pobranych z różnych źródeł, co oznacza, że schemat jak i dane mogą ewoluować w czasie wykonywania programu. Klasy w programowaniu obiektowym definiują swoje znaczenie i funkcjonalność w większości za pomocą funkcji (metod). W ontologii nie możliwe jest zdefiniowanie wykonywalnych funkcji.

Individual

Individual (jednostki) są przykładami klas np. film „Ojciec chrzestny" jest przykładem klasy *Film*. W programowaniu obiektowym struktura obiektów musi być zgodna z definicją ich klasy. W RDF Schema klasy nie ogranicza struktura cech zasobów RDF (Individual) i tym samym zasoby mogą korzystać z dowolnych właściwości. Do hierarchii klas przyporządkowane są jednostki, np. filmy, które występują w bazie. Ilustracja poniżej (oraz kolejne) przedstawia widok ekranu programu Protege.



Ilustracja 4 Przykłady klasy Film

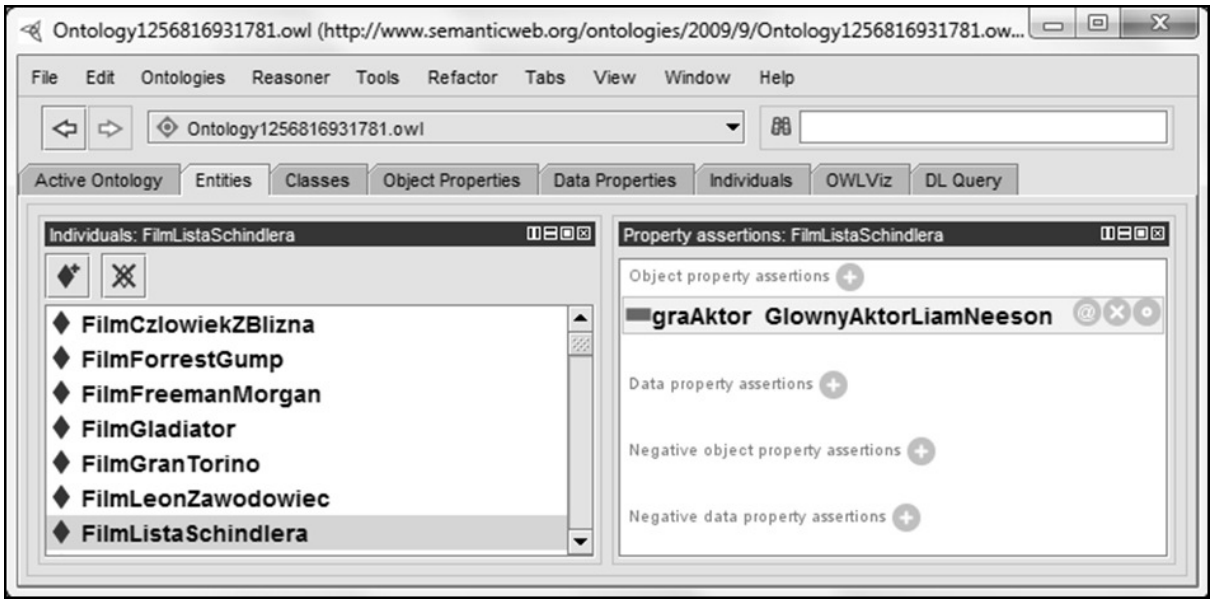
Przykładowy kod w języku OWL:

```
<owl:Thing rdf:about="#filmOjciecChrzestny">
        <rdf:type rdf:resource="#Film"/>
        </owl:Thing>
```

W pierwszej linijce definiujemy jednostkę *filmOjciecChrzestny*, która należy do klasy *Film.*

Properties

Properties (właściwości) są stosowane do określenia związków pomiędzy jednostkami lub jednostek powiązanych z typami danych, np. tekst, liczba całkowita. Właściwość *graAktor* ma wartość obiektową (Object Property), wiąże przypadki klas *Film* i *GlownyAktor.* Ilustracja poniżej przedstawia powyższy przykład w programie Protege.



Ilustracja 5 Zapis relacji między przykładami klasy Film i GlownyAktor

Przykładowy kod w języku OWL:

<owl:Thing rdf:about="#GlownyAktorLiamNeeson"/>

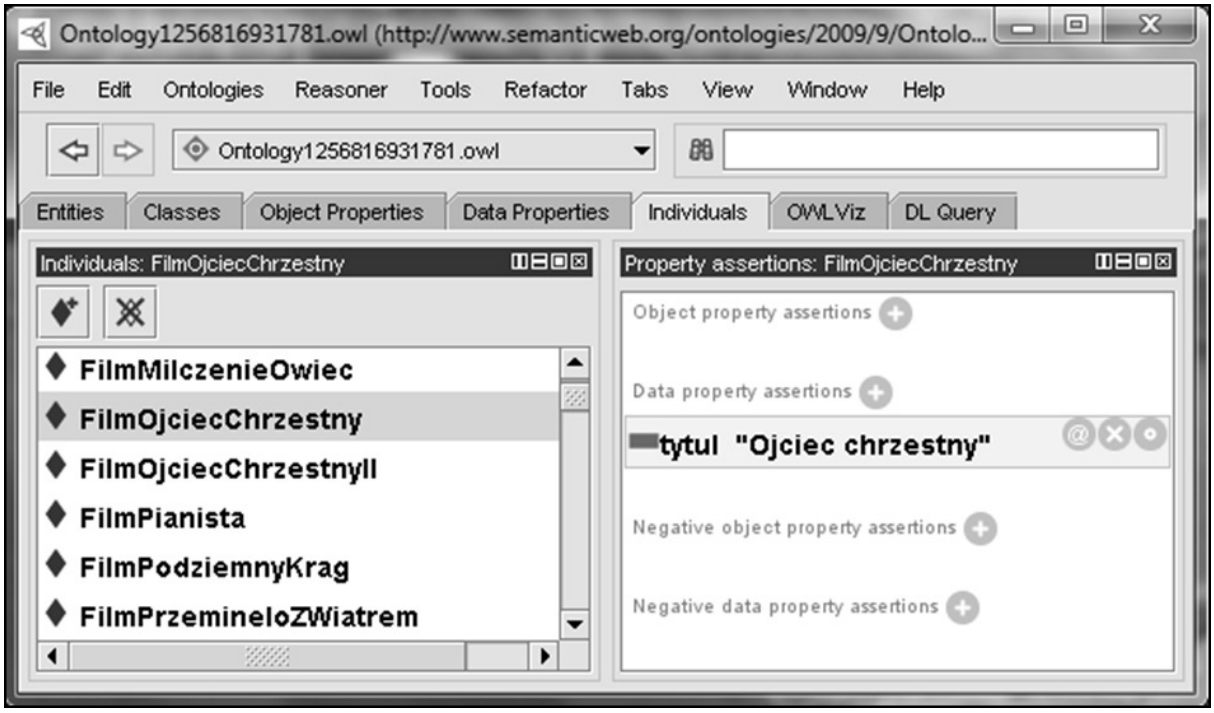
<owl:Thing rdf:about="#FilmListaSchindlera">

<graAktor rdf:resource="#GlownyAktorLiamNeeson"/>

</owl:Thing>

W filmie Lista Schindlera głównym aktorem jest Liam Neeson. Właściwość *graAktor* łączy jednostki *FilmListaSchindlera* i *GlownyAktorLiamNeeson.*

Właściwość *tytul jest* określana jako Data Property, wiąże ona przypadki klasy *Film* z typami danych. Ilustracja poniżej przedstawia powyższy przykład w programie Protege.



Ilustracja 6 Zapis relacji między przykładem klasy Film a typem danych

Przykładowy kod w języku OWL:

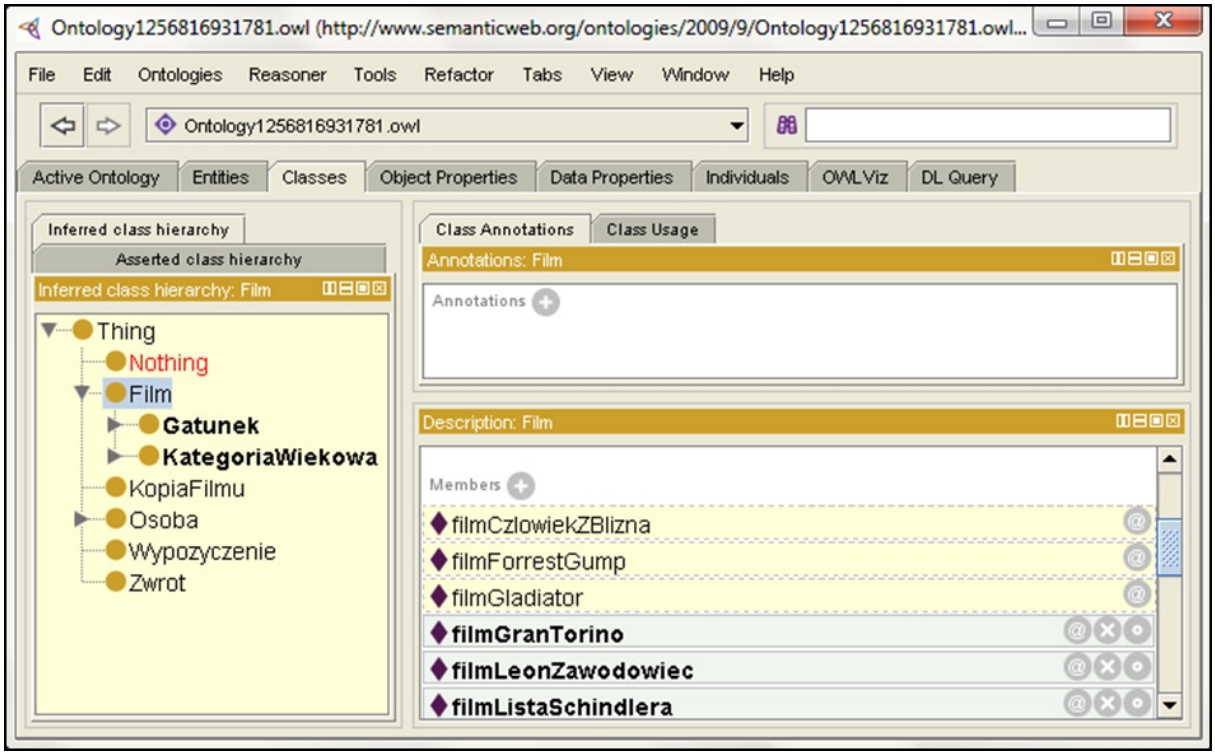
```
<owl:Thing rdf:about="#filmOjciecChrzestny">
        <rdf:type rdf:resource="#Film"/>
        <tytul rdf:datatype="&xsd;string">Ojciec chrzestny</tytul>
</owl:Thing>
```

Wartość *tytul* zawiera typ danych *string.*

W językach programowania atrybuty definiowane są lokalnie wewnątrz klasy. Mogą być używane przez obiekty tej klasy i zazwyczaj mają jeden typ przyjmowanych wartości. Właściwości definiowane w RDF Schema mogą być używane przez dowolne klasy i przyjmować dowolne wartości.

Superclasses

Superclasses, klasa nadrzędna (ogólniejsza) znajduje się wyżej w hierarchii i obejmuje klasy podrzędne, np. klasa *Film* jest klasą nadrzędną i zawiera podklasy *KategoriaWiekowa* i *Gatunek.* Program może wywnioskować, że jeżeli jednostki znajduje się w klasie *Gatunek* również należą do klasy *Film*. Ilustracja poniżej przedstawia powyższy przykład w programie Protege.



Ilustracja 7

Instancje *filmCzlowiekZBlizna*, *filmForretGump* i *filmGladiator* należą do klas *Gatunek* i *KategoriaWiekowa.* Zostały one dodane do klasy *Film* przez maszynę wnioskującą. Przykładowy kod w języku OWL:

```
<owl:Class rdf:about="#Film"/>
<owl:Class rdf:about="#Gatunek">
        <rdfs:subClassOf rdf:resource="#Film"/>
</owl:Class>
<owl:Class rdf:about="#KategoriaWiekowa">
        <rdfs:subClassOf rdf:resource="#Film"/>
</owl:Class>
```

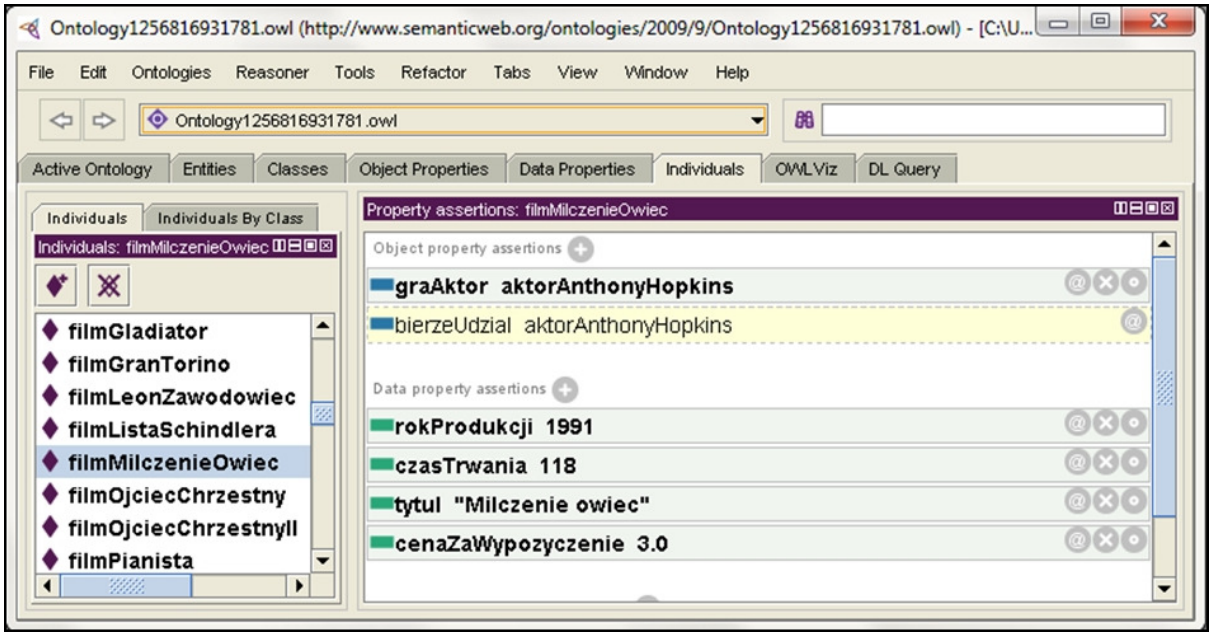
Superproperties

Superproperties, właściwości mogą zawierać podwłaściwości, np. *graAktor* i *kreciRezyser* są podwłaściwościami *bierzeUdzial*. Przykładowy kod w OWL:

```
<owl:ObjectProperty rdf:about="#bierzeUdzial"/>
<owl:ObjectProperty rdf:about="#graAktor">
        <rdfs:subPropertyOf rdf:resource="#bierzeUdzial"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#kreciRezyser">
        <rdfs:subPropertyOf rdf:resource="#bierzeUdzial"/>
```

</owl:ObjectProperty>

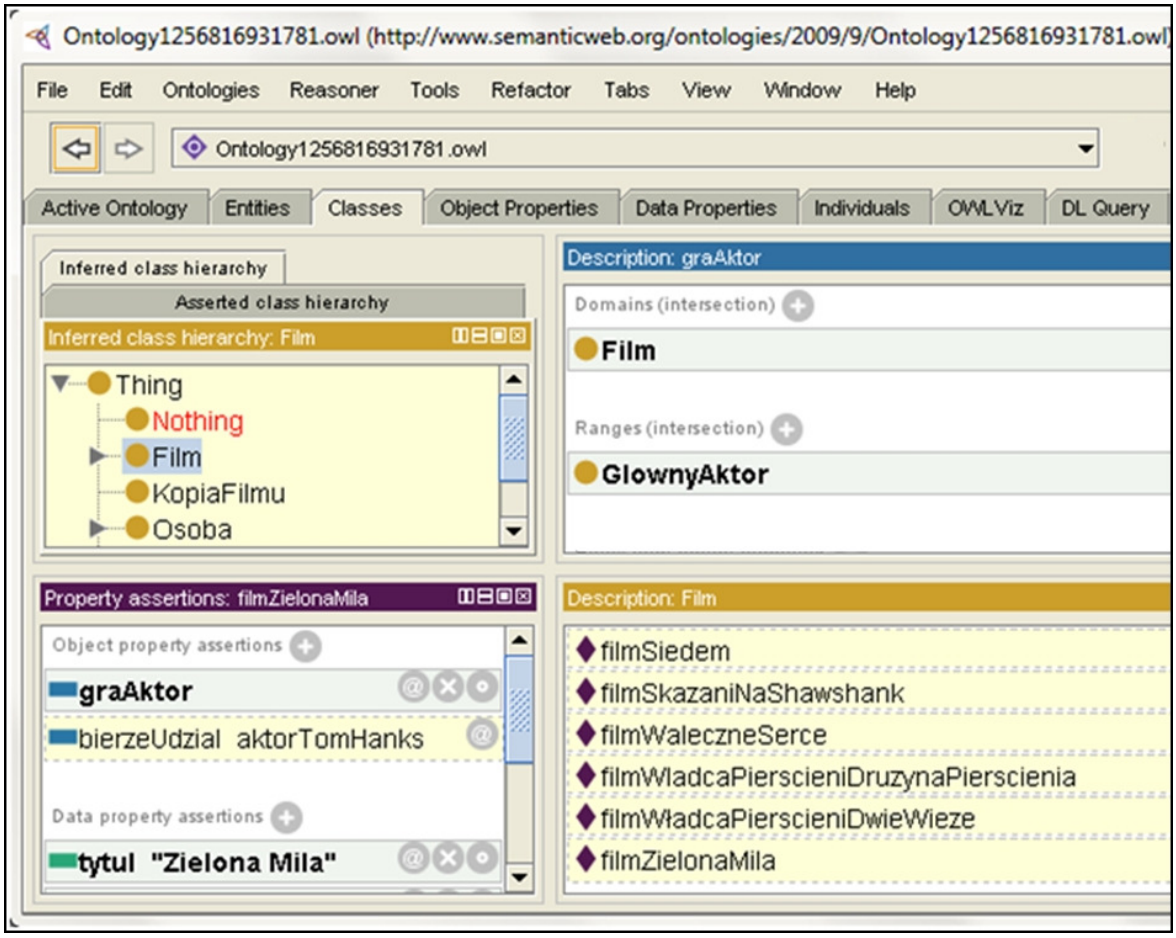
Program może wywnioskować, że jeżeli jednostka jest powiązana z inną jednostką poprzez właściwość *graAktor*, to jest również powiązana przez właściwość *bierzeUdzial*. Na zdjęciu poniżej Data properties *bierzeUdzial* jest podkreślona, ponieważ wartość *graAktor* jest jej nad właściwością. Ilustracja poniżej przedstawia powyższy przykład w programie Protege.



Ilustracja 8

Domains

Domains (domena), jeżeli właściwość posiada klasę jako swoją domenę i właściwość wiąże jednostkę z inną jednostką, to wtedy jednostka należy do klasy domeny, np. właściwość *graAktor* posiada domenę *Film*. Jeżeli jednostka *filmZielonaMila* jest powiązany za pomocą właściwości *garaAktor* z jednostką *aktorTomHanks*, program może wywnioskować, że jednostka *filmZielonaMila* jest filmem. Ilustracja poniżej przedstawia powyższy przykład w programie Protege.



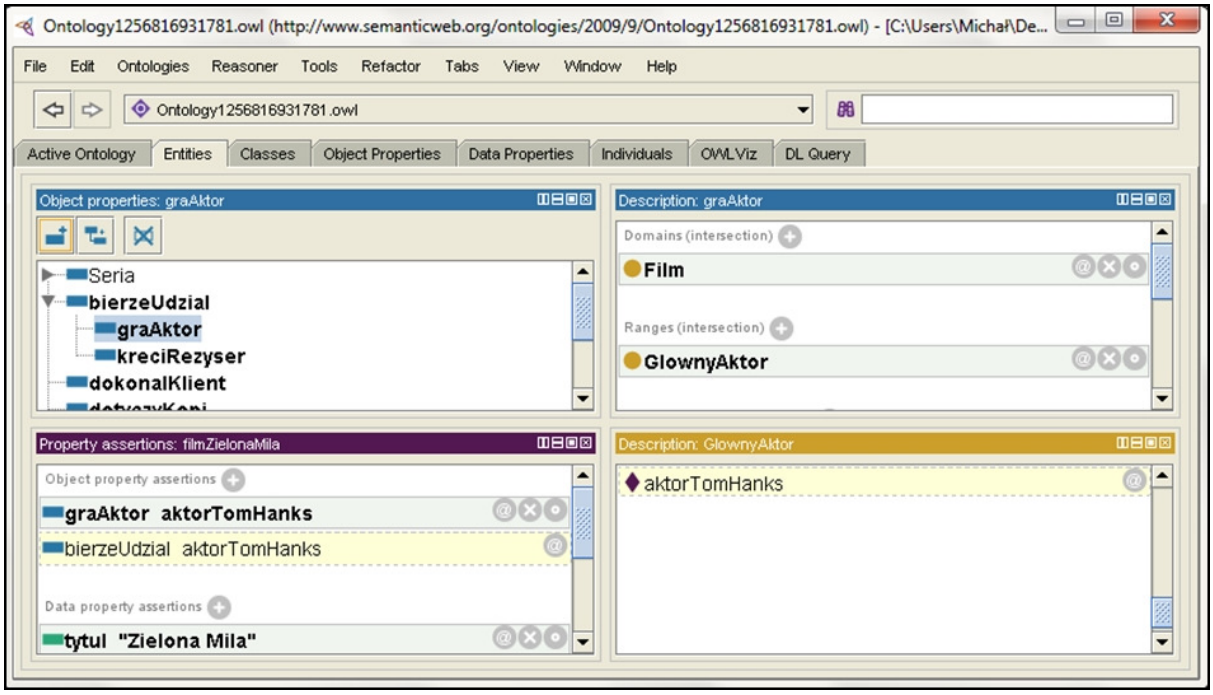
Ilustracja 9

Przykładowy kod w języku OWL:

```
<owl:ObjectProperty rdf:about="#graAktor">
        <rdfs:domain rdf:resource="#Film"/>
</owl:ObjectProperty>
```

Ranges

Ranges (zakres), jeżeli właściwość posiada klasę jako swój zakres i właściwość wiąże jednostkę z inną jednostką, wtedy ta inna jednostka należy do klasy zakresu, np. właściwość *graAktor* posiada *Ranges Film* i jeżeli jednostka *filmZielonaMila* jest powiązana za pomocą właściwości *garaAktor* z jednostką *aktorTomHanks*, to program może wywnioskować, że Tom Hanks jest aktorem. Ilustracja poniżej przedstawia powyższy przykład w programie Protege.



Ilustracja 10

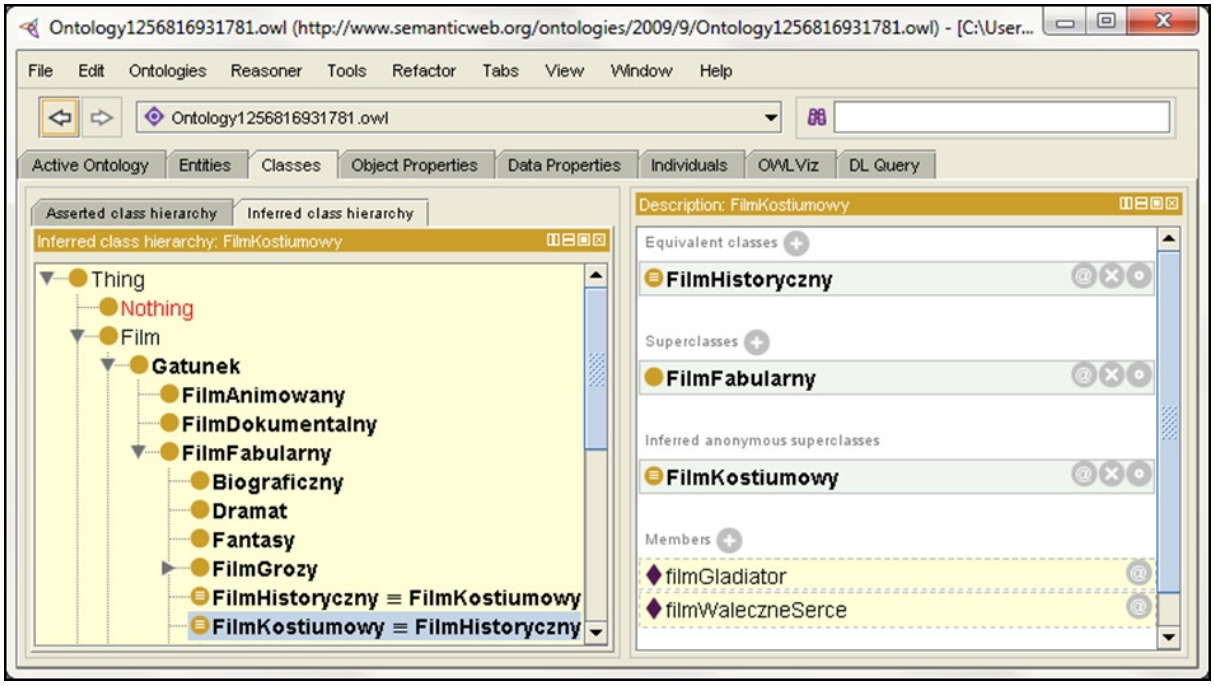
Przykładowy kod w języku OWL:

```
<owl:ObjectProperty rdf:about="#graAktor">
        <rdfs:range rdf:resource="#GlownyAktor"/>
</owl:ObjectProperty>
```

Cechy OWL Lite odnoszące się do równości i nie równości, nie występują w języku RDFS

Equivalent classes

Equivalent classes oznacza, że dwie klasy mają takie same elementy, np. klasa *FilmHistoryczny* jest ekwiwalentną klasą dla *FilmKostiumowy*. Program może wywnioskować, że jeżeli jednostka, która jest przykładem klasy *FilmHistoryczny* jest również przykładem klasy *FilmKostiumowy*. Ilustracja poniżej przedstawia tą zależność w programie Protege.



Ilustracja 11

Przykładowy kod w języku OWL:

<owl:Class rdf:about="#FilmHistoryczny">

<owl:equivalentClass rdf:resource="#FilmKostiumowy"/>

</owl:Class>

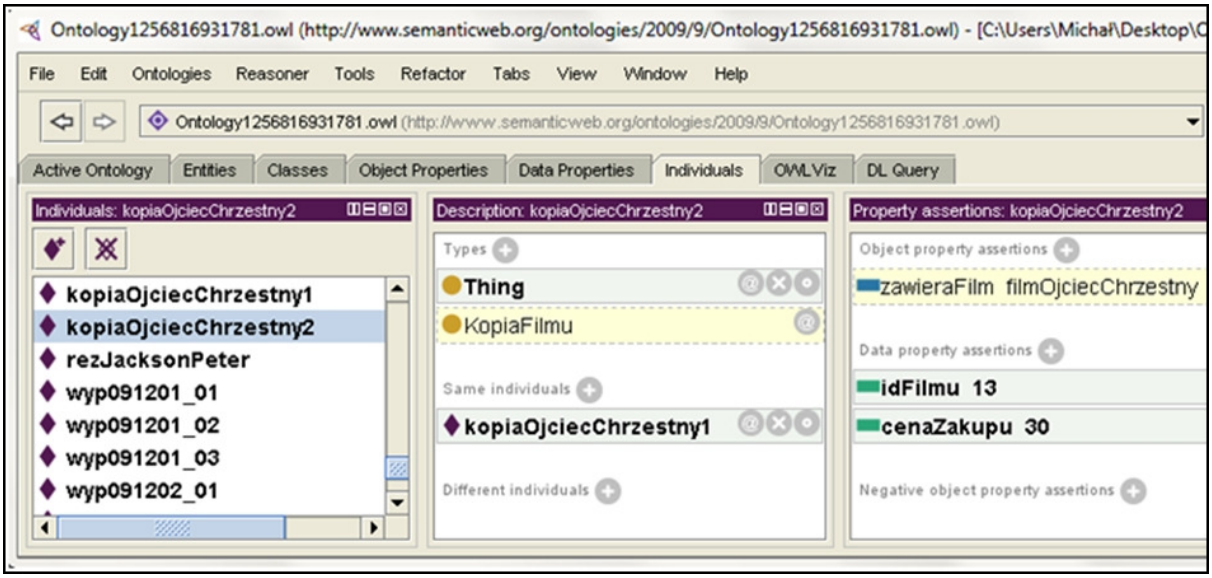
<owl:Class rdf:about="#FilmKostiumowy"> </owl:Class>

Equivalent properties

Equivalent properties (ekwiwalentna właściwość), stosuje się do tworzenia właściwości równoznacznych, np. właściwość tytuł filmu jest ekwiwalentną właściwością dla tytułu w języku angielskim, który jest jego podwłaściwością i odwrotnie.

Same individuals

Same individuals (taki sam), konstrukcja taka może być stosowane do tworzenia wielu różnych nazw, które odnoszą się do tej samej jednostki, np. jednostka *kopiaOjciec-Chrzestny1*, może być określana jako taka sama jednostka jak *kopiaOjciecChrzestny2*. Obie jednostki są przykładami kopi filmu „Ojciec Chrzestny". Wypożyczalnia może zawierać wiele kopi jednego filmu. Ilustracja przedstawia powyższy przykład w programie Protege.



Ilustracja 12

Maszyna wnioskująca dodała wartość *zawieraFilm filmOjciecChrzestny* do jednostki *kopiaOjciecChrzestny2*.

Przykładowy kod w języku OWL:

```
<owl:Thing rdf:about="#FilmOjciecChrzestny1">
        <owl:sameAs rdf:resource="#FilmOjciecChrzestny2"/>
</owl:Thing>
<owl:Thing rdf:about="#FilmOjciecChrzestny2"/>
```

Different individuals

Different indifiduals, jednostki, które należą do wspólnej klasy różnią się od siebie, np. film „Przeminęło z wiatrem” posiada dwie wersje kinowe.

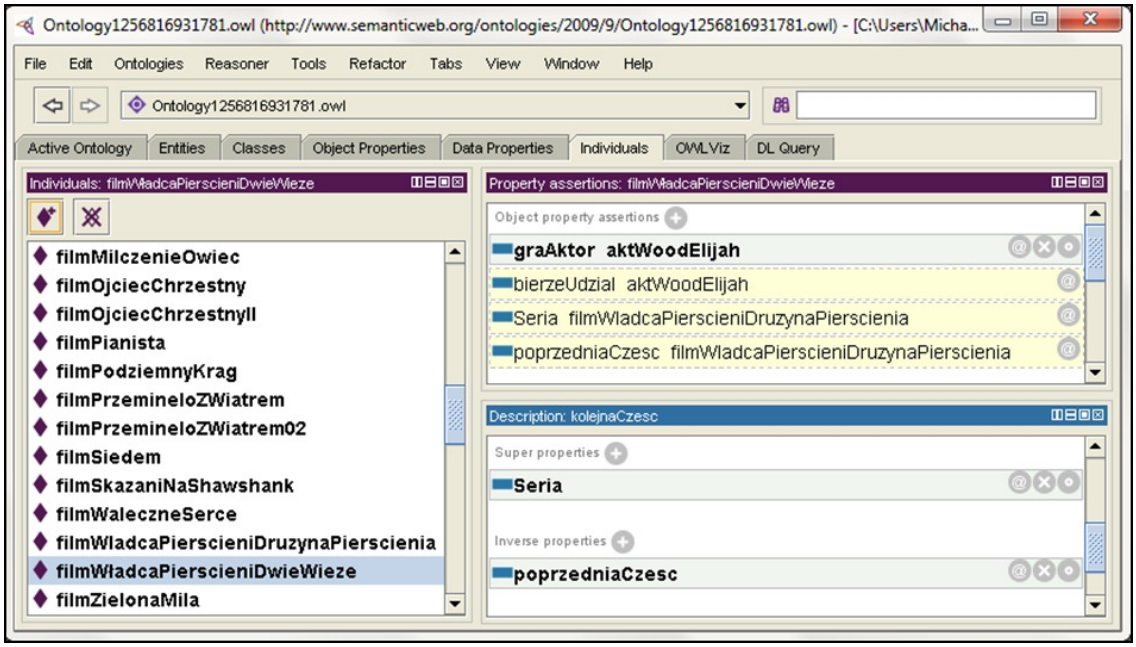
Przykładowy kod w języku OWL:

```
<rdf:Description>
        <rdf:type rdf:resource="&owl;AllDifferent"/>
        <owl:distinctMembers rdf:parseType="Collection">
        <rdf:Description rdf:about="#filmPrzemineloZWiatrem"/>
        <rdf:Description rdf:about="#filmPrzemineloZWiatrem2"/>
        </owl:distinctMembers>
</rdf:Description>
```

Poniżej są umieszczone cechy właściwości OWL DL i Full.

Inverse properties

Inverse prperties, właściwość jest odwrotnością innej właściwości. Object properties *kolejnaCzesc* jest odwrotnością *poprzedniaCzesc*. Jeżeli film„Drużyna pierścienia” jest powiązany wartością *kolejnaCzesc* z filmem „Dwie wieże”, program może wywnioskować, że film „Dwie wieże” jest poprzednią częścią filmu „Drużyna pierścienia”. Ilustracja przedstawia powyższy przykład. Maszyna wnioskująca dodała do jednostki *filmWładcaPierscieniDwieWieze*. Wartość *poprzeniaCzesc* jest podkreślona. Ilustracja poniżej przedstawia powyższy przykład w programie Protege.



Ilustracja 13

Przykładowy kod w języku OWL:

<owl:ObjectProperty rdf:about="#kolejanCzesc"/>

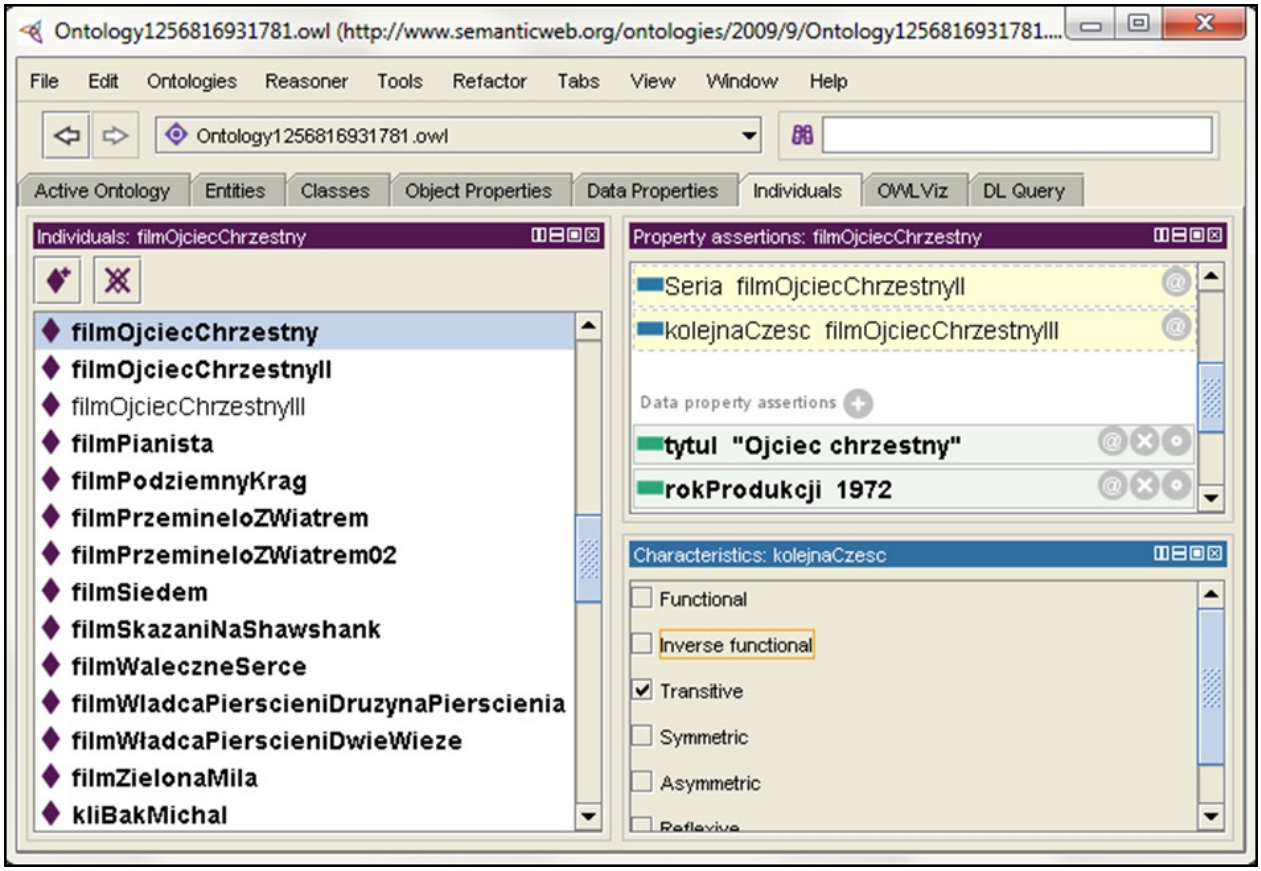
<owl:ObjectProperty rdf:about="#poprzedniaCzesc">

<owl:inverseOf rdf:resource="#kolejanCzesc"/>

</owl:ObjectProperty>

Transitive

Transitive, przechodnia właściwosść, np. film „Ojciec chrzestny III” jest kolejną częścią filmu „Ojciec chrzestny II”, który jest kolejną częścią filmu „Ojciec chrzestny I”, program może wywnioskować, że film „Ojciec chrzestny III” jest kolejną częścią filmu „Ojciec chrzestny I”. Ilustracja poniżej przedstawia powyższy przykład w programie Protege.



Ilustracja 14

Przykładowy kod w języku OWL:

```
<owl:ObjectProperty rdf:about="#kolejanCzesc">
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>
```

Symmetric

Symmetric, symetryczna właściwość. Wartość *podobny* jest symetryczną właściwością. Jeżeli film 1 jest powiązany przez właściwość *podobny* z filmem 2, to wtedy urządzenie może wywnioskować, że film 2 jest podobny do filmu 1.

Przykładowy kod w języku OWL:

```
<owl:ObjectProperty rdf:about="#podobny">
        <rdf:type rdf:resource="&owl;SymmetricProperty"/>
</owl:ObjectProperty>
```

Functional

Functinal, właściwość posiada jedną wartości albo nie zwiera żadnej. Na przykład wartość *tytulAng*, która przechowuje tytuł film w języku angielskim, może być określona jako *Functinal*. Film może zawierać tytuł w języku angielskim, ale nie musi.

Zapis w OWL:

<owl:DatatypeProperty rdf:about="#tytulAngielski">

<rdf:type rdf:resource="&owl;FunctionalProperty"/>

</owl:DatatypeProperty>

Inverse Functional

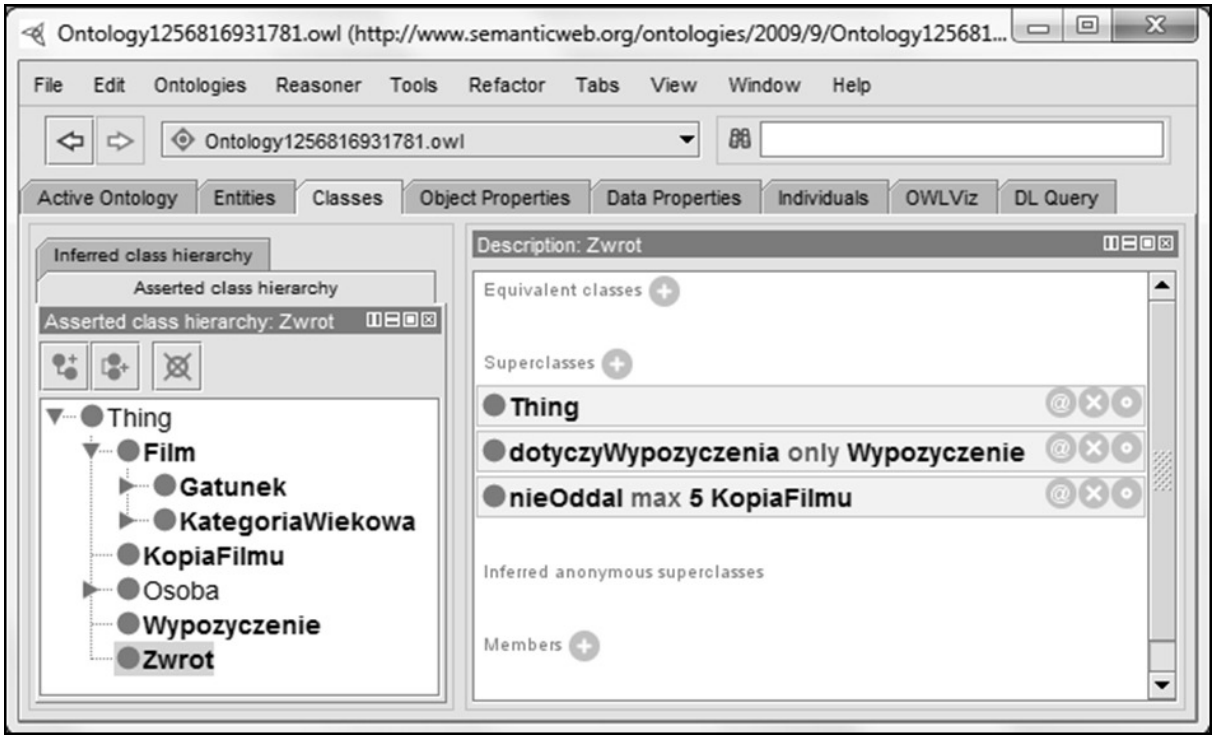
Inverse Functional odwrotnie funkcjonalna właściwość, która ma najwyżej jedną wartość dla jednostki.

Właściwość min Cardinality

Właściwość min Cardinality, określa minimalną moc zbioru.

Właściwość max Cardinality

Właściwość max Cardinality określa maksymalną moc zbioru, np. klasa *Zwrot* posiada ograniczenie *nieOddal max 5 kopiaFilmu* oznacza to, że klient może nie oddać maksymalnie pięciu płyt w terminie. Ilustracja poniżej przedstawia powyższy przykład w programie Protege.



Ilustracja 15 Ograniczenie klasy Zwrot

Zapis przykładu w języku OWL:

<owl:Class rdf:about="#Zwrot">

<rdfs:subClassOf>

```
        <owl:Restriction>
        <owl:onProperty rdf:resource="#nieOddal"/>
        <owl:onClass rdf:resource="#Zwrot"/>
        <owl:maxQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">5</owl:maxQualifiedCardinality>
        </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

Dziedziczenie

Dziedziczenie jest jednym z najważniejszych pojęć w obiektowości, które pozwala na utworzenie nowej klasy potomnej (podklasy) na podstawie klasy rodzica (nadklasy). Klasy potomne posiadają cechy nadklasy i swoje własne. Podklasa jest szczególnym przypadkiem nadklasy. Klasa Klient może dziedziczyć z klasy Osoba, ponieważ klient jest osobą. W programowaniu obiektowym obiekty dziedziczą metody i atrybuty zdefiniowane przez ich nadrzędne klasy. W RDF Schema, ponieważ właściwości nie należą do klas, nie są dziedziczone. W RDF Schema i OWL możemy definiować hierarchię dziedziczenia nie tylko klas, ale również i właściwości, np. Properties *graAktor* dziedziczy z właściwością *bierzeUdzial*. W RDF Schema właściwości mają takie same znaczenie jak klasy. W większości języków programowania obiektowego, np. Java możliwe jest dziedziczenie tylko z jednej klasy. W RDF Schema klasy mogą dziedziczyć po wielu klasach. Przykładowy kod w języku OWL:

```
<!-- Classes -->
<owl:Class rdf:about="#Film"/>
        <owl:Class rdf:about="#Aktor">
        <rdfs:subClassOf rdf:resource="#Obsada"/>
        </owl:Class>
<!--Object Properties  -->
<owl:ObjectProperty rdf:about="#bierzeUdzial">
        <rdfs:domain rdf:resource="#Film"/>
        <rdfs:range rdf:resource="#Obsada"/>
        </owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#graAktor">
        <rdfs:domain rdf:resource="#Film"/>
        <rdfs:range rdf:resource="#GlownyAktor"/>
```

<rdfs:subPropertyOf rdf:resource="#bierzeUdzial"/>

</owl:ObjectProperty>

Każdy zasób, który jest typu *Aktor* jest typu *Obsada*. Każda para zasobów spełniająca właściwość *graAktor* spełnia również właściwość *bierzeUdzial*. Jeżeli nie mielibyśmy zdefiniowanego dziedziczenia pomiędzy klasami, to właściwość *graAktor* nie byłby zdefiniowany dla klasy *Obsada*. Nie oznacza to, że nie możemy z niej skorzystać dla zasobu, który jest typu *Aktor*, stanie się on tylko jednocześnie zasobem typu *Obsada*. Jeżeli dane byłby tylko dziedziczone pomiędzy klasami, to właściwość *bierzeUdzial* byłaby dostępna dla zasobów typu *Obsada*, co nie jest prawdą.

2.4 Proces wnioskowania

Z danych opisanych w językach RDF, RDFS i OWL, można uzyskać nowe dane za pomocą wnioskowania. Powstało wiele narzędzi wnioskujących, np. Pellet, FaCt++. Narzędzie Pellet jest wbudowane w Protege 3, umożliwia wnioskowanie w warstwie RDFS, OWL i tworzenie zapytań w języku SPARQL. FaCt ++ znajduje się w Protege 4, służy do wnioskowania w warstwie OWL DL.

2.4.1 DL Query

Składnia OWL Syntax pozwala nam na tworzenie zapytań w zakładce DL Query w wersji Protege 4. Przed pisaniem zapytań należy włączyć maszynę wnioskującą FaCt++.

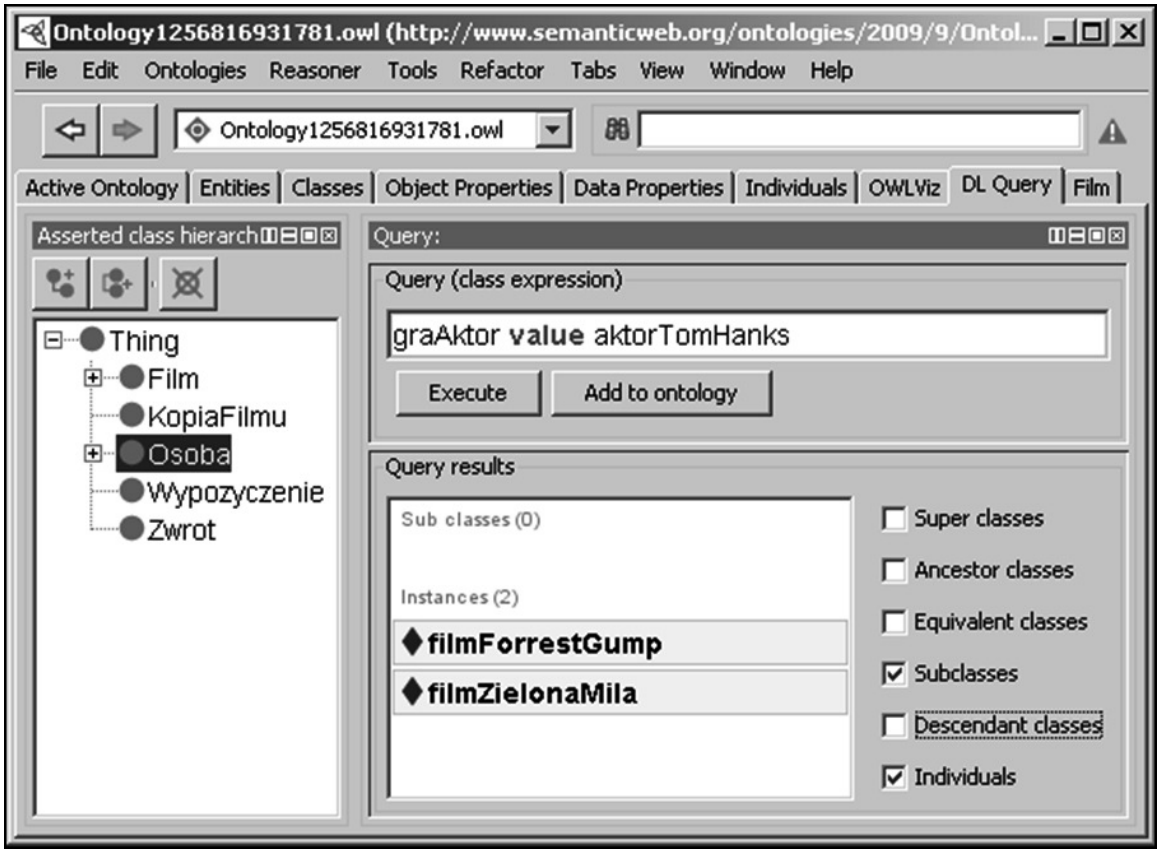
Przykładowe zapytania:

-jak wypisać wszystkie filmy w jakich grał Tomy Hanks?

Przykład zapytania w OWL Syntax:

graAktor value aktorTomHanks

Gdzie *graAktor* jest obiektem, *aktorTomHanks* jest instancją. Ilustracja poniżej przedstawia powyższy przykład w programie Protege.

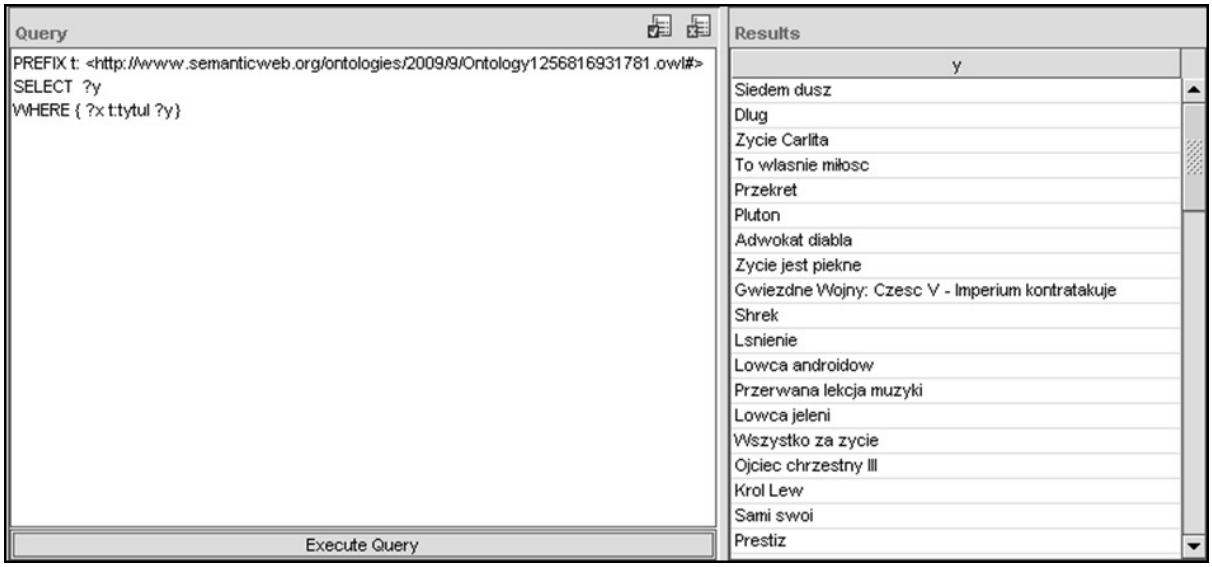


Ilustracja 15

2.4.2 SPARQL

SPARQL(Protocol And RDF Query Language) jest językiem zapytań opartym na dokumentach RDF. Przeznaczony jest do wydobywania informacji z semantycznej sieci, która jest oparta na meta języku XML do definiowania schematów znakowania oraz do przedstawienia danych przez RDF. Składnia zapytań SPARQL została zaczerpnięta z języka SQL. Występują klauzule PREFIX, SELECT, WEHER. PREFIX pozwala na nadanie zastępczej nazwy URI. SELECT odpowiada za wyselekcjonowanie zmiennych, które będą przedstawione w wyniku zapytań. Klauzula WHERE odnosi się do trójki RDF: Subject, Predicate, Object. Subject (temat lub podmiot) jest znacznikiem URI lub pustym węzłem. Predicate (predykat) lub Property (własność) jest znacznikiem URI. Object (obiekt) jest znacznikiem URI lub pustym węzłem. Język SPARQL w styczniu 2008 został uznany jako standard przez organizacje W3C.

Poniżej na zdjęciu znajduje się przykład użycia zapytania w języku SPARQUL zwraca ono bazę filmów dostępnych w wypożyczalni.



Ilustracja 16 Przykład zapytania w języku SPARQL

-jaki jest czas trwania filmu np. „Dług”?

Zapytanie w języku SPARQL:

```
PREFIX
t: <http://www.semanticweb.org/ontologies/2009/9/Ontology1256816931781.owl#>
SELECT ?tytul ?czasTrwania
WHERE {
        ?x t:tytul ?tytul.
        ?x t:czasTrwania ?czasTrwania
        FILTER regex(?tytul, "Dlug")
}
```

3. Model MVC, Framework CakePHP

3.1 Model-View-Controller

Model-View-Controller to architektoniczny wzorzec projektowy w informatyce do organizowania struktury aplikacji. Architektura ta dzieli aplikacje na trzy główne części. Model, Widok, Kontroler.

Model odpowiedzialny jest za logikę biznesową, czyli jak aplikacja przechowuje dane. Sposób przechowywania danych jest obojętny. Może to być baza danych, plik tekstowy. Ważne jest, aby szczegóły implementacyjne, w których model przechowuje dane, były ukryte przed resztą aplikacji. Model dostarcza innym komponentom spójnego interfejsu do przetwarzania tych danych, np. zapytanie SQL. Dobrym sposobem implementacji modelu jest utworzenie oddzielnej klasy dla każdego logicznego obiektu, np. użytkow-

nik. Najczęściej model przechowuje dane w bazie danych. Jedna klasa modelu odpowiada jednej tabeli w bazie danych lub grupie ściśle powiązanych tabeli. W czasie zmiany jednej tabeli, zmiana w kodzie PHP ogranicza się do jednej klasy modelu. Dobrze zaprojektowany model powinien być odporny na zewnętrzne zmiany. Zmiany w strukturze bazy danych nie powinny wpływać na interfejs, z którego korzystają pozostałe części aplikacji. Model powinien również być przenośny. Klasy modelu służące do obsługi użytkownika, np. rejestracja, logowanie, można łatwo przenieść do innej aplikacji, która wymaga również takiej, funkcjonalności.

Widok jest odpowiedzialny za prezentację. W aplikacjach internetowych używanym formatem wyjściowym jest HTML. Aplikacja może również prezentować wyniki w postaci XML, obrazów, plików PDF. Widok wykorzystuje model do pobrania danych, które będą wyświetlane. Nie powinien on modyfikować danych. Powinien np. wyświetlać użytkowników, ale nie powinien ich dodawać, usuwać. Widok również powinien być implementowany w postaci wielu klas, np. jedna strona na klasę.

Kontroler jest sercem aplikacji w architekturze MVC. Powinien on być częścią świadomą żądania HTTP. Na podstawie analizy żądania Kontroler powinien zdecydować jakiej akcji użyć i jaki widok wyświetlić. Musi on w jakiś sposób wiedzieć co składa się na widok i model aplikacji. Jeżeli aplikacja ma wyświetlić listę użytkowników kontroler tworzy widok, który z kolei pobiera z modelu odpowiednie dane i prezentuje je, np. w postaci listy. Jeżeli aplikacja ma dodać użytkownika kontroler musi sprawdzić dane POST i wysłać odpowiednią metodę modelu. W tym celu należy dodać akcję, która jest odpowiedzialna za pojedynczą czynność, np. dodanie użytkowników. Akcja może być implementowana, jako klasa tworzona przez kontroler wykonująca odpowiednie zadanie i przekazująca sterownikowi informacje jaki widok należy wyświetlić.

3.2 CakePHP

CakePHP działa na podstawie architektury MVC (Model, View, Controller). Framework pozwala na oddzielenie danych (model) od reguł ich przetwarzania (kontroler) oraz sposobu ich prezentacji (widok). Podejście to pozwala, np. zmianę bazy danych nie powodując zmiany pozostałych komponentów aplikacji, ponieważ są one od siebie oddzielone.

CakePHP jest to Framework wspierający szybkie programowanie w języku PHP, wzorowany na pakiecie Ruby On Rails. Jest on darmowy o otwartym kodzie. Pierwsza wersja powstała w 2005 roku. Obecnie ma on ugruntowaną pozycję i grono zainteresowanych jego dalszym rozwojem. Framework jest to zbiór wzorców, technik,

ułatwiających tworzenie aplikacji. Posiada on strukturę plików, która wymusza napisanie kodu, który jest przejrzysty i łatwy w zarządzaniu. CakePHP wiele rzeczy jest zautomatyzowanych, tworzenie skomplikowanych serwisów jest łatwiejsze. Programiści nie muszą ciągle tworzyć kodu od początku, mogą, się skupić tylko na logice aplikacji.

3.2.1 Cechy CakePHP

CakePHP (wersja 1.3) jest kompatybilna z wersją języka 4 i 5 PHP. Framework wykorzystuje programowanie obiektowe OOP (ang. Object Oriented Programming). Pozwala ona na tworzenie programów za pomocą obiektów. Z programowaniem obiektowym wiąże się pojęcie enkapsulacji (ukrywanie implementacji). Obiekt nie powinien zmieniać stanu innych obiektów. Tylko funkcje powinny modyfikować dane w nim zawarte. Innym ważnym pojęciem w programowaniu OOP jest dziedziczenie (definiowanie klas przy pomocy innych klas), abstrakcja (definiowanie obiektów bez implementacji wszystkich cech) oraz polimorfizm (możliwość określenia za pomocą jednej nazwy wielu metod, których właściwy dobór odbywa się w trakcie wykonywania programu). CakePHP wykorzystuje metodę z języka PHP5, która pozwala zmienić zachowanie obiektu. Funkcja __call() jest jedną z takich metod, która jest wywoływana, gdy aplikacja odwołuje się do nieistniejącej metody obiektu. Odpowiednio napisana funkcja umożliwia obsługę danych, np. pobranie danych z bazy za pomocą wirtualnych metod. Wywołana funkcja *$object -> findById(5)* będzie wywołaniem funkcji *__call()*, przeszuka ona bazę danych stosując zapytanie w SQL *SELECT * FROM object WHERE id = 5*.

CakePHP umożliwia łatwą obsługę do zapamiętania adresów WWW. Adres URL, który zawiera dużą liczbę parametrów może powodować błędy w indeksowaniu przez wyszukiwarki. Adres http://www.powtarzac.pl/users/login/ jest czytelny, prowadzi do strony z logowaniem. Na podstawie tych informacji możemy określić czy jesteśmy zainteresowani odwiedzeniem strony. Adres *http://profil.wp.pl/login.html?url=http%3A%2F%2Fpoczta.wp.pl%2Findex.html%3Fflg%3D1&serwis=nowa_poczta_wp/* jest nieczytelny, nie zawiera podstawowych informacji o stronie.

Framework pozwala na buforowanie stron WWW, redukuje opóźnienia w dostarczaniu danych do użytkownika oraz zmniejsza obciążenia serwera. Dane wygenerowane po raz pierwszy zostaną zachowane i udostępniane są innym użytkownikom do czasu ich modyfikacji lub usunięcia. Udostępnianie odbywa się bez konieczności pobierania

danych z bazy, wykonania obliczeń.

CakePHP posiada system szablonów, w którym udostępnione są dynamicznie wyświetlane fragmenty kodu.

Framework ma wbudowany mechanizm walidacji danych. Bezpieczeństwo i poprawność danych są jednymi z najważniejszych elementów programowania. CakePHP pozwala na zdefiniowanie reguł, które muszą być spełnione zanim wprowadzone dane do formularza zostaną uznane za poprawne i będzie możliwe ich dalsza modyfikacja. Reguły mają postać wyrażeń regularnych np. liczba dziesiętna */^[-+]?\b[0-9]*\.?[0-9]+\bS/.*

CakePHP umożliwia tworzenie serwisów w oparciu o technologię AJAX (ang. Asynchronous Java Script And XML). Za pomocą technologii AJAX tworzone są interaktywne strony, aktualizowane są tylko wymagane fragmenty stron. Dostęp do bazy danych w CakePHP jest zintegrowany. Utworzone aplikacje, współpracujące z różnymi bazami danych są skomplikowane. Różnice w API (ang . Application Programming Interface) baz danych powodują tworzenie kodu do obsługi każdej z nich oddzielnie. Różnice występują również w składni poleceń języka SQL. Framework ma dostęp do bazy danych dzięki warstwie obiektów pośredniczących, ukrywając przed programistą składnię związaną z konkretną bazą danych. Nie trzeba tworzyć kodu SQL, chociaż jest to możliwe.

Framework posiada mechanizm scaffolding, umożliwia on analizę tabel w bazie danych i automatyczne tworzenie formularzy pozwalające na edycję, usuwanie i przeglądanie danych. Scaffolding pozwala na szybkie sprawdzenie czy model i reguły walidacji danych działają zgodnie z oczekiwaniami.

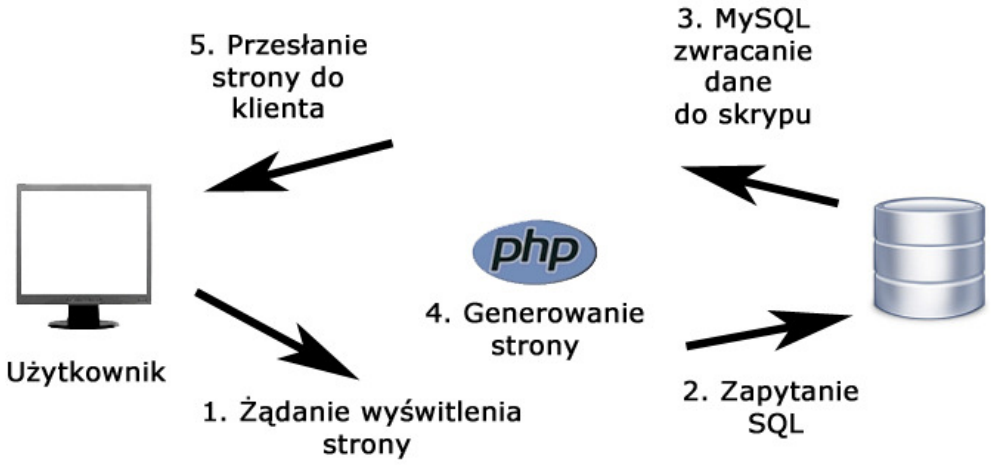
3.2.2 Model MVC w PHP

We wczesnych aplikacjach internetowych kod obsługujący model danych był przemieszany z kodem odpowiedzialnym z wyświetlanie danych w przeglądarce. Przykład aplikacji w której kod PHP i HTML są ze sobą przemieszane:

```
<?php
// połączenie z bazą danych
$connect_base = mysql_connect('localhost', 'login', 'hasło');
mysql_select_db('cakephp_db', $connect_base);
// zapytanie SQL do bazy danych
$sql = "SELECT * FROM table"
$result = @mysql_query($sql, $connect_base);
```

```
        ?>
        <html>
                <head>
                        <title>tytuł</title>
                </head>
        <body>
                        <$php
        // wyświetlenie wyniku zapytania
                        while($row = mysql_fetch_array($result)
                        {
                                echo $row['pole'];
                        }
        ?>
        </body>
        </html>
<$php
        // zmaknięcie połączenia do bazy
        mysql_close($connect_base);
?>
```

Rysunek poniżej przedstawia schemat aplikacji, w której kod PHP zwiera logikę biznesową z kodem, który jest odpowiedzialny za prezentację danych.



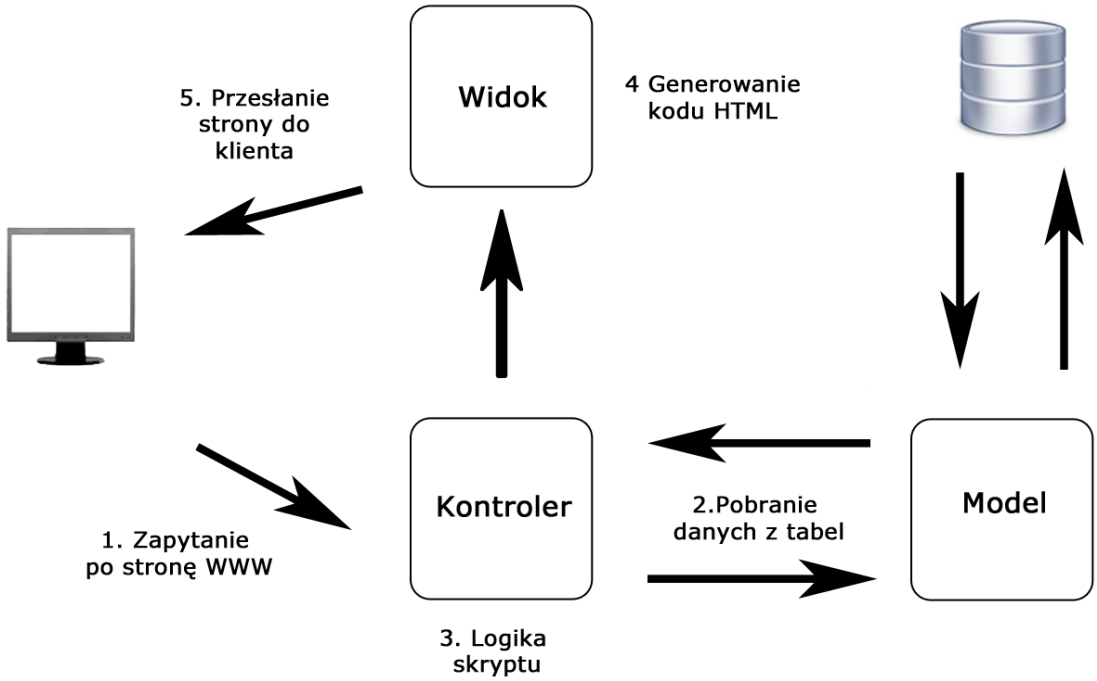
Rysunek 1 Architektura aplikacji, w której kod jest przemieszany z kodem odpowiedzialnym za wyświetleni danych.

Zaletą takiej techniki programowania jest szybkość tworzenia. Wad jest znacznie więcej. Każda zmiana modelu powoduje błędy w prezentacji danych. Program zawiera API do konkretnej bazy danych, np. MySQL. Kod HTML powiązany z PHP powoduje, że

nie można go wykorzystać w innych projektach. Często zachodzi zmiana sposobu wyświetlania danych. Czytelność kodu można poprawić wydzielając do oddzielnego pliku fragmenty kodu, które są odpowiedzialne za prezentacje danych. Takie oddzielnie kodu umożliwia łatwiejszą współpracę w zespole. Prezentację danych przygotowuje grafik bez obaw, że zmodyfikuje kod aplikacji, np.

```
<?php
// połączenie z bazą danych
$connect_base = mysql_connect('localhost', 'login', 'hasło');
mysql_select_db('cakephp_db', $connect_base);
// zapytanie SQL do bazy danych
$sql = "SELECT * FROM table"
$result = @mysql_query($sql, $connect_base);
        while($row = mysql_fetch_array($result)
        {
                $table[] = $row['pole'];
        }
// zamknięcie połączenia do bazy danych
mysql_close($connect_base);
require("widok.php");
?>
Plik widok.php
<htm>
        <head>
                <title>tytuł</title>
        </head>
<body>
                <$php
                        forech($table as $tab
                        {
                                echo['pole'];
}
?>
</body>
</html>
```

W warstwie prezentacji należy dążyć do tego, żeby było jak najmniej kodu PHP. Należy unikać generowania kodu HTML za pomocą PHP, np. echo *'witam na stronie
';*
Aby korzystać z innej bazy danych we wczesnych aplikacjach internetowych, wymagana jest przebudowa całej aplikacji, uwzględniając nowe API i upewnianiu się, że aplikacja wyświetla informacje we właściwy sposób. Problemu tego możemy uniknąć wydzielając do oddzielnego pliku fragmenty kodu, które są odpowiedzialne za komunikację z bazą danych. Zyskujemy interfejs łączący źródła danych z pozostałymi częściami aplikacji. Umożliwia to oddzielnie logiki biznesowej aplikacji. Rozbudowa, zmiana bazy danych nie zakłóci funkcjonowania aplikacji.
Podzielenie aplikacji na dostarczenie danych, wygląd i logikę odpowiada architekturze MVC. Umożliwia ona odseparowanie źródeł danych (model) od reguł ich przetwarzania (kontroler) oraz sposobu prezentacji. Podział taki pozwala na ukrycie szczegółów implementacji komponentów. Model pozwala na pobieranie informacji. Widok odpowiedzialny jest za wyświetlanie danych. Kontrole przetwarza informacje. Rysunek poniżej przedstawia architekturę MVC.



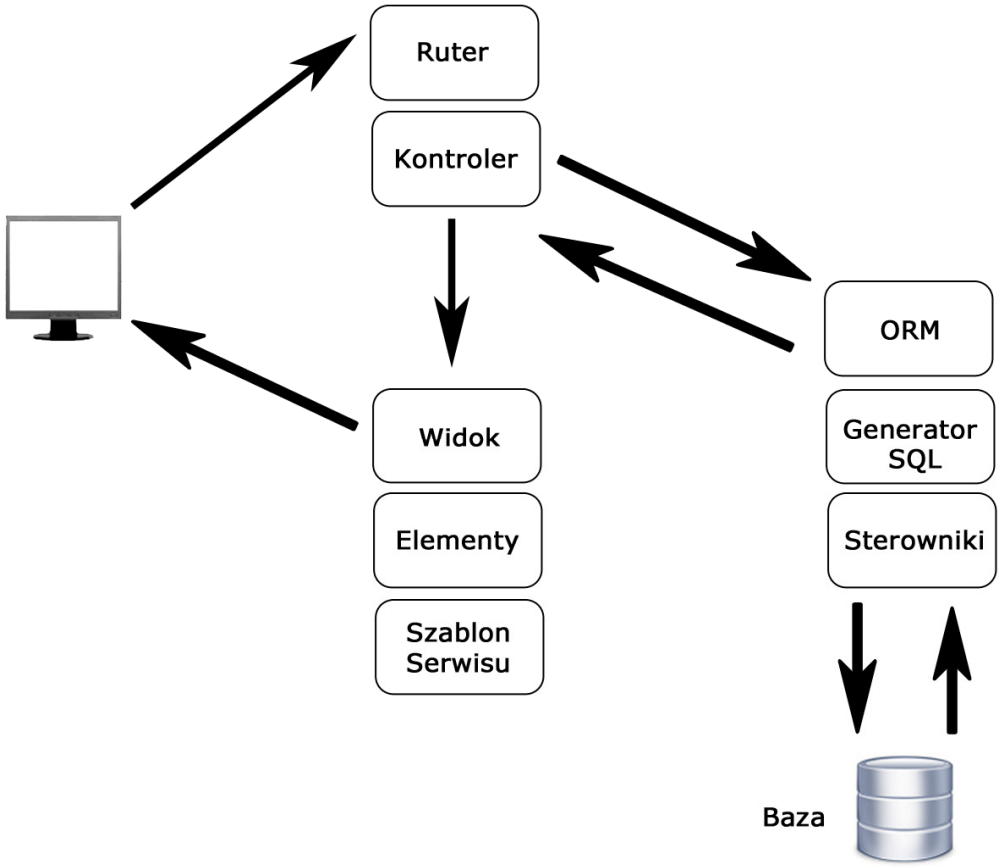
Rysunek 2 Model MVC

3.2.2 Implementacja architektury MVC w CakePHP

Architektura MVC w CakePHP wprowadza uproszczenia w zarządzaniu i tworzeniu aplikacji. Framework rozwiązuje problem różnic API i SQL miedzy bazami danych. Dostęp do bazy danych odbywa się przez obiekty pośredniczące, które ukrywają nisko poziomowy kod.

Widok w CakePHP ma dodatkowe funkcjonalności są to „elementy”oraz „layout”. Pozwalają one na uniknięciu powtarzania kodu na podstronach. W layout umieszczamy wspólne elementy dla całego serwisu, np. stopkę. Widoki są wyświetlane wewnątrz szablonu serwisu. Elementy to niewielkie komponenty, z których można tworzyć wygląd strony.

Użycie obiektów Ajax, cache, form, Html, Java Script, Number, Session, Text oraz Time muszą być poprzedzone deklaracją w kontrolerze. Kontroler zawiera kod odpowiedzialny za przetwarzanie danych. Aplikacja może mieć wiele kontrolerów. Są one podzielone na część odpowiedzialną za podjecie decyzji, które komponenty mają być użyte i część odpowiedzialna za obsługę funkcjonalność aplikacji. O tym, który kontroler ma zostać wywołany odpowiedzialny jest moduł Ruter. Rysunek poniżej przedstawia architekturę MVC w Frameworku CakePHP.



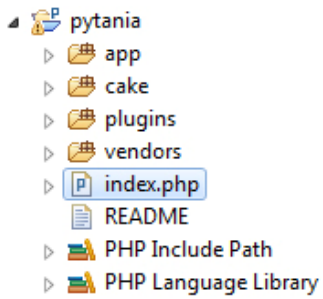
Rysunek 3 Architektura MVC w CakePHP

Budowa aplikacji sprawdza się do tworzenia klas pochodnych i ich rozbudowy. Do najważniejszych klas należą Ruter, AppController i AppModel. Ruter jak wcześniej wspomniano opowiedziany jest za podjęcie decyzji, który kontroler ma zostać uruchomiony. AppController klasa, która definiuje kontroler aplikacji. AppModel klasa definiuje model danych, która odpowiedzialna jest za dostęp do bazy. View jest klasą ob-

sługującą pliki HTML, zawiera layout i widoki poszczególnych stron. Klasa Helper grupuje funkcje pomocnicze, które mogą być wykorzystywane w widokach.

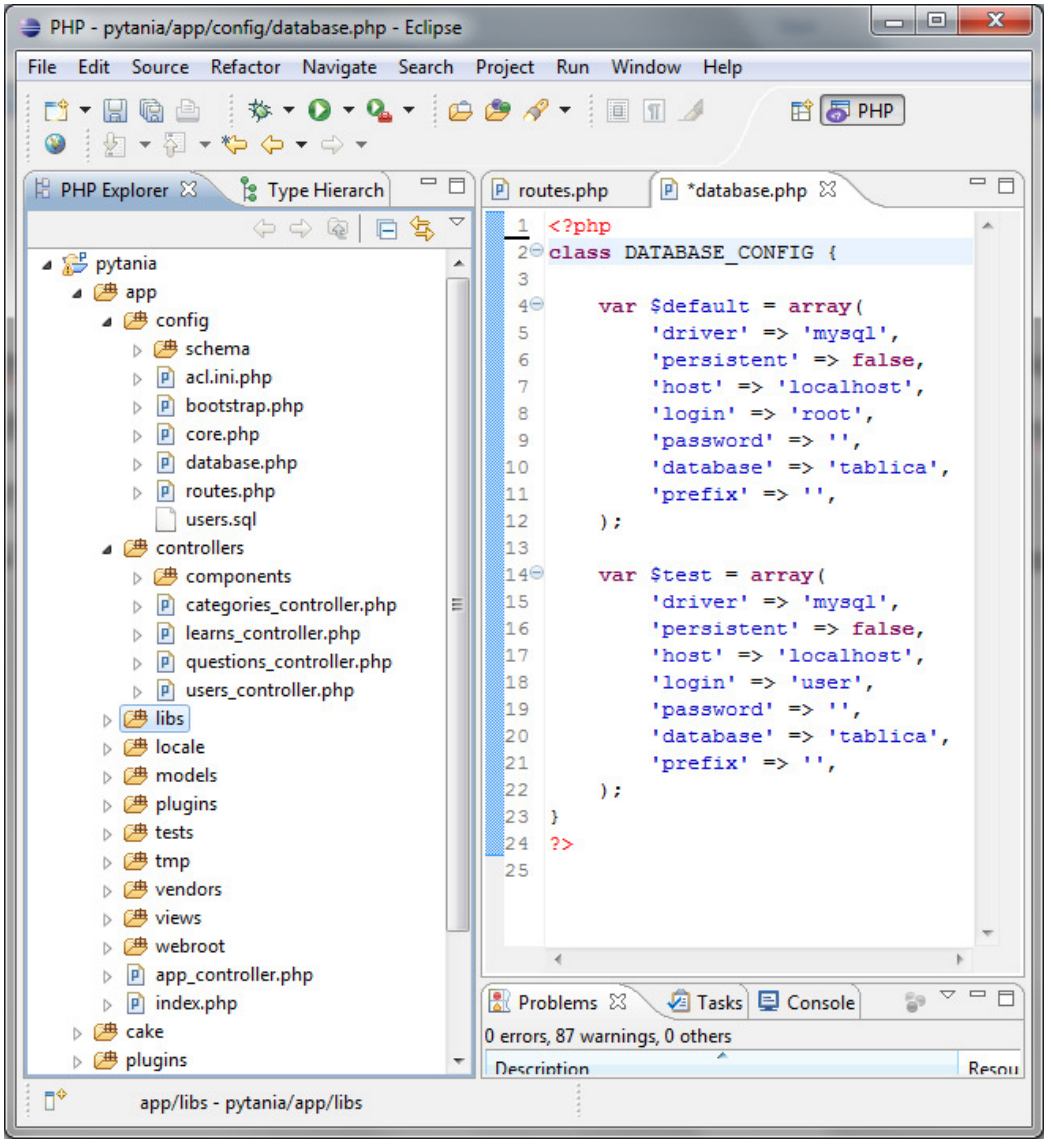
3.2.3 Organizacja kodu

W CakePHP wszystkie elementy aplikacji oddziałują na siebie wzajemnie. Framework wymusza na programistach dostosowanie się do prostych reguł, np. sposób nazywania komponentów aplikacji i położenie ich w strukturze katalogów programu. Ilustracja poniżej przedstawia najważniejsze katalogi w Frameworku CakePHP.



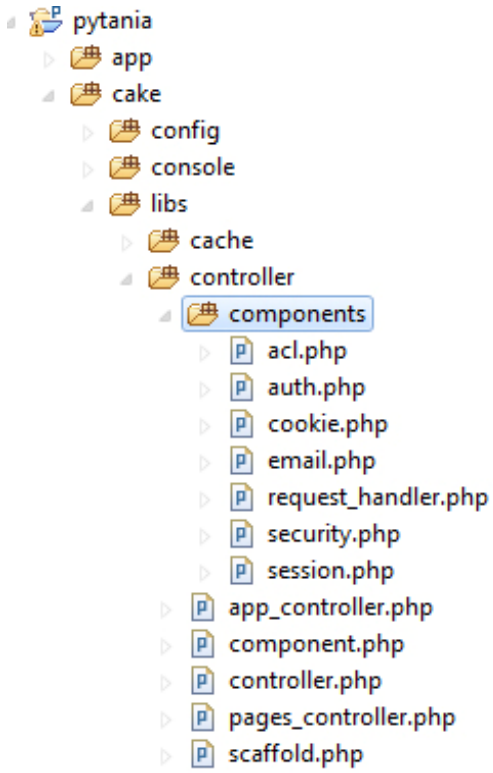
Ilustracja 1 Struktura katalogów w Frameworku CakePHP

W katalogu app/ znajduje się kod naszej aplikacji. Komponenty architektury MVC umieszczamy w podkatalogach controllers/, models/, vies/. Grafikę i style CSS zawiera katalogu webroot/. W katalogu vendors/ umieszczony jest kod niezwiązany z architekturą MVC. W katalogu app/config znajdują się pliki konfigurujące. Pozwalają one, np. na zdefiniowanie połączenia do bazy danych oraz konfigurację modułu rutera, który decyduje uruchomieniu komponentów CakePHP. Ilustracja poniżej przedstawia struktura katalogu app.



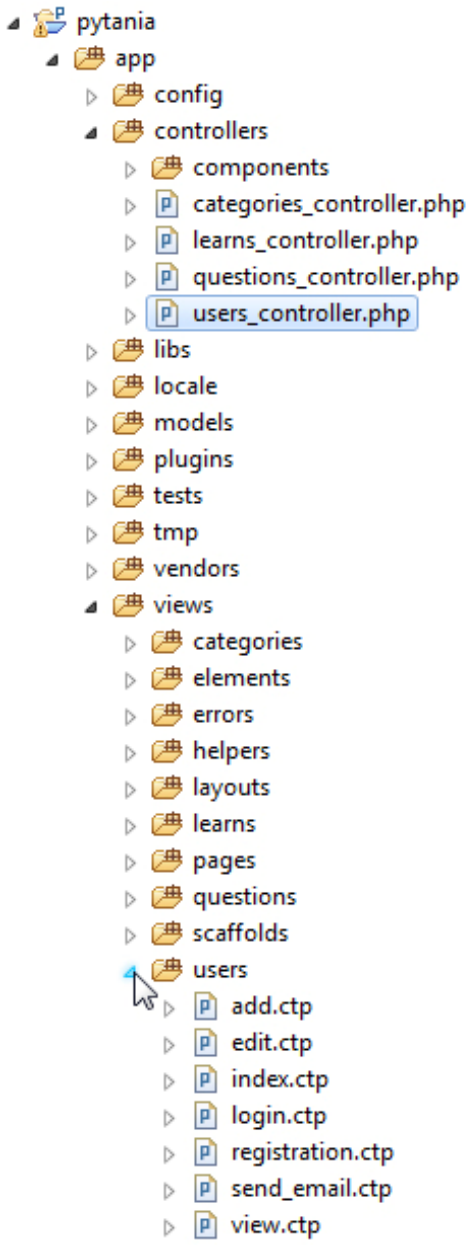
Ilustracja 2 Struktura katalogu app

W katalogu app/controllers/ umieszczone są pliki z rozszerzeniem .php, w których znajdują się definicje klas i metody sterujące logiką aplikacji. Podkatalog app/controllers/components/ zawiera kod dla wielu aplikacji. Takim komponentem może być moduł wysyłający e-mail. W katalogu cake/libs/controller/components/ znajdują się komponenty wbudowane. Ilustracja poniżej przedstawia strukturę katalogów.



Ilustracja 3 Struktura katalogu cake/libs/controller/components/

W katalogu app/views umieszczone są pliki z rozszerzeniem .ctp, które są powiązane z konkretną metodą kontrolera. Zawierają one wygląd strony. Ilustracja poniżej przedstawia pliki widoku.



Ilustracja 4 Pliki widoku są powiązane z konkretną metodą kontrolera.

W katalogu app/views/layouts/ umieszczone są elementy wspólne dla wygładu strony WWW, np. nagłówek, logo, stopka. Elementy te są niezmienne w całej aplikacji. Również w pliku layouts zawiera kod, który odpowiedzialny jest za wyświetlenie pozostałych widoków związane z poszczególnymi funkcjami aplikacji. Domyślny widok nazywa się default.ctp.

Katalogu app/views/elements/ znajdują się bloki informacji, które mogą być wyświetlane na wielu stronach np. formularze, przyciski. Dzięki temu, nie jest niepotrzebnie powielany kod. Są to mini widoki, które są umieszczane widokach.

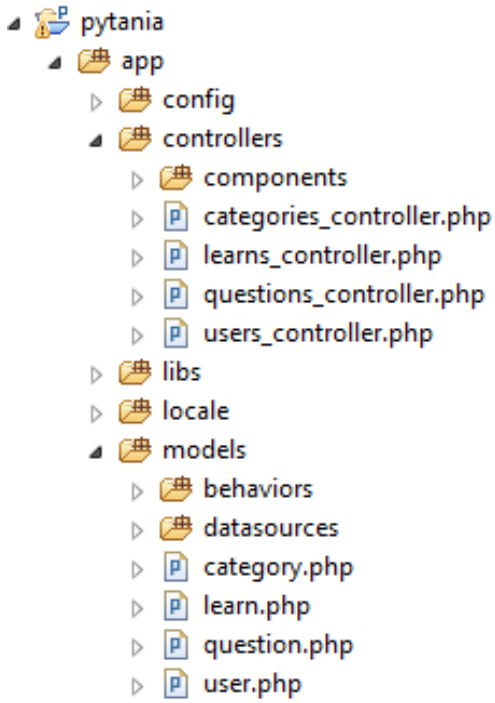
W katalogu app/webroot znajdują się pliki graficzne style CSS i Javascript.

Katalog cake/ zawiera biblioteki ułatwiające tworzenie aplikacji, domyślne szablony stron (katalog cake/libs/views/layouts/) i wbudowane pliki Helpers (Ajax, Cache, Form,

Html, Javascript, Number, Session, Text, Time), Componets (Acl, Reqest_handler, Security, Session).

3.3 Kontroler

Kontroler jest odpowiedzialny za logikę biznesową, interakcję z użytkownikiem, modelem i widokiem. Odpowiedzialny jest również za przepływ danych z modelu do widoku, walidację danych, zarządzanie wyświetlanych stron, tworzenie logu zdarzeń, zarządzanie modułami, które rozszerzają funkcjonalność aplikacji. Kontroler pozwala na dostęp do serwisu WWW dla użytkowników. Ruter mapuje wywołania WWW na odpowiedzialne funkcje. Metody zawarte w kontrolerze decydują o funkcjonalności aplikacji. Każda aplikacja może mieć wiele kontrolerów, które obsługują logikę biznesową związana z pojedynczymi modelem danych. Ilustracja poniżej przedstawia pliki kontroler i model.

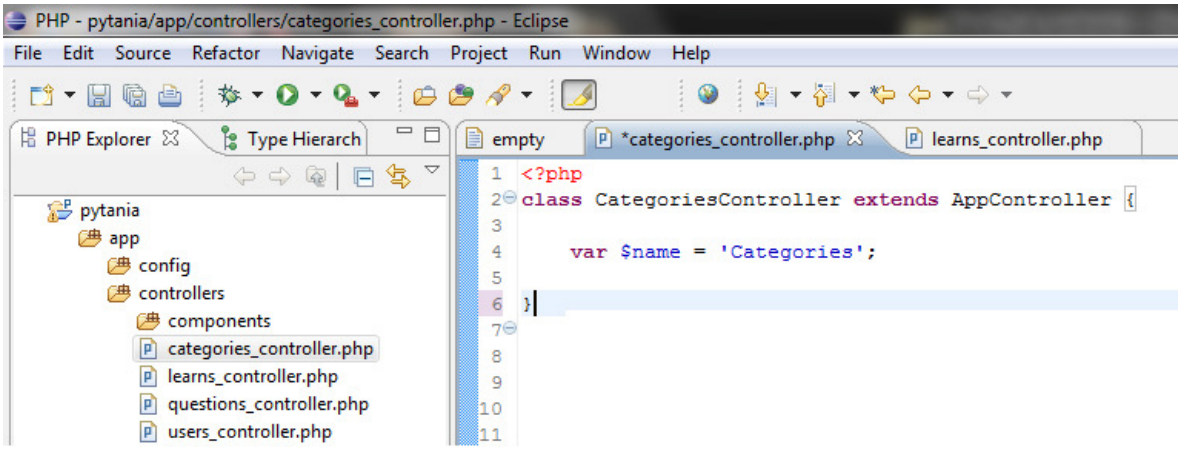


Ilustracja 5 Kontroler jest powiązane z konkretnym modelem.

Domyślny kontroler jest zdefiniowany w app/config/router.php.

Kontroler to obiekt zbudowany o klasę AppController, rozbudowany o funkcje, które pozwalają na pobranie danych i ich wyświetlenie. Każdy kontroler jest niezależny od pozostałych i zawiera własny zestaw funkcji. Jeżeli chcemy, żeby część funkcji była dostępna w innych kontrolerach, np. autoryzacja, korzystamy z dziedziczenia. Wszystkie metody wspólne dla więcej niż jednego kontrolera są zapisane w pliku cake/app.controller.php, w którym jest zdefiniowana klasa AppController. Klasa AppController rozbudowuje klasę Controller znajdująca się w pliku ca-

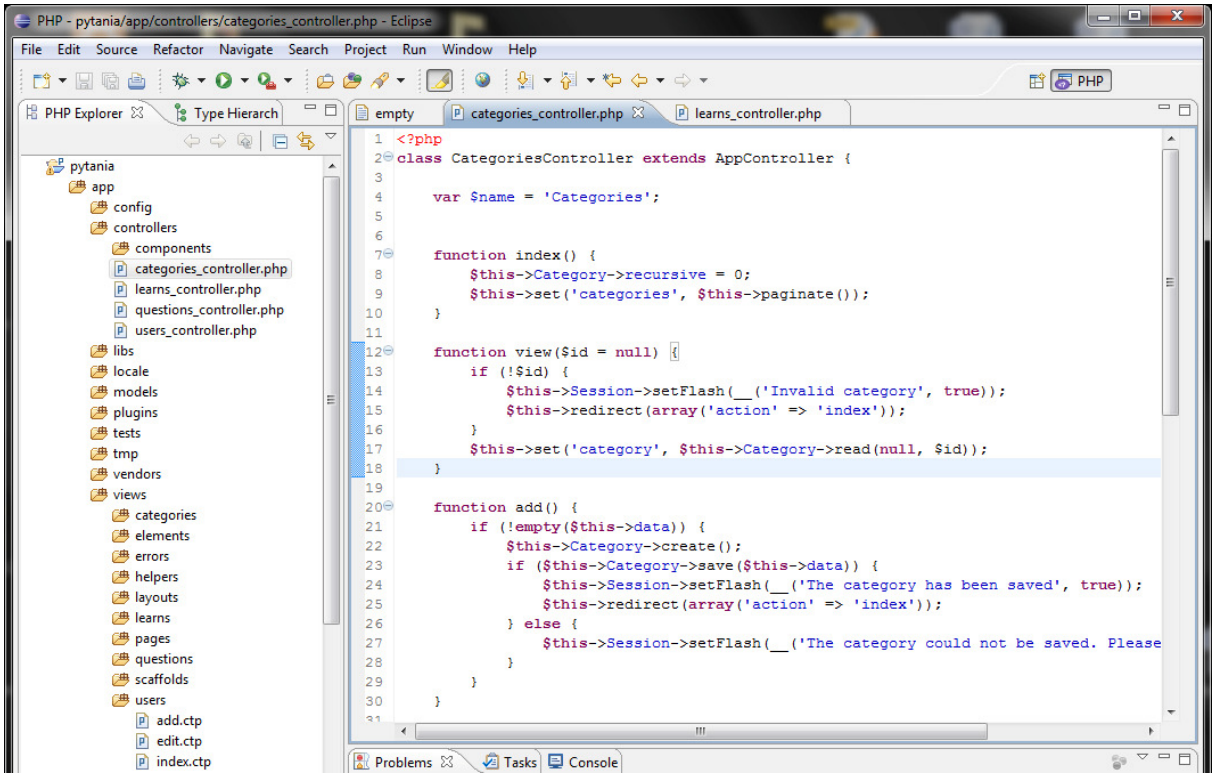
ke/libs/kontroler/controller.php. W jednym pliku znajduje się tylko jeden kontroler. Framework automatycznie odnajduje, wczytuje pliki na podstawie nazwy kontrolera. Nazwę klasy zapisujemy w liczbie mnogiej, bez spacji między wyrazami. Kolejny wyraz rozpoczynamy z dużej litery, nazwę kończymy słowem Controller. Przykład nazwy w kontrolerze: UserController. Nazwę pliku zapisujemy małymi literami, zamiast spacji używamy znak podkreślenia '_', całość kończymy słowem controller. Przykład nazwy pliku kontrolera: users_controller. Wszystkie pliki kontrole zapisujemy w katalogu app/controllers/. Jeżeli w kodzie znajduje się PHP4 należy zdefiniować zmienną name, która zawiera nazwę klas. Ilustracja przedstawia przykład klasy kontroler.



Ilustracja 6 Przykład kontrolera

3.3.1 Akcje

Akcja jest metodą obsługiwaną przez kontroler. Jest ona uruchamiana, kiedy ruter znajdzie adres WWW zgodny z regułami zdefiniowanymi w pliku konfiguracyjnym. Można zdefiniować dowolną liczbę akcji. Działanie z nich jest wyświetlane w widoku. Ilustracja poniżej przedstawia zawartość kontrolera *CategoriesContorller.*



Ilustracja 7 Przykład kontrolera CategoriesContorller, który zawiera akcje index(), add(), edit().

3.3.2 API kontrolera

Interakcja z widokiem

Interakcja kontrolera z widokiem odbywa się za pomocą funkcji AppController -> set(‘a’, ‘b’). Funkcja ta udostępnia dane widokowi. Po wywołaniu funkcji w widoku będzie dostępna zmienna ‘a’ zainicjowana wartością ‘b’. Metoda może przekazywać pojedyncze wartości jak również obiekty PHP. Funkcja AppController -> render($action = null, $layout = null, $file = null) pozwala na wyświetlenie strony zdefiniowanej przez layout i widok akcji. Funkcja jest wywoływana automatycznie, gdy akcja zakończy swoje działanie. Funkcji używamy jeżeli chcemy skorzystać z innego widoku niż domyślny, który znajduje się w app/view/nazwa_kontrolera/nazwa_akcji.ctp. Po wywołaniu funkcji należy użyć funkcji: exit(). Funkcja AppController -> requestAction($url, $extra = array) umożliwia wywołanie akcji z innego kontrolera i zwrócenie jej działania. Użycie requestAction() to jedyny sposób na interakcję miedzy różnymi widokami. Wynikiem działania funkcji może być widok wygenerowany przez akcję.

Interakcja z modelem danych

Funkcja *AppController-> postConditions($data)* pozwala na konwersję formatu danych przesłanych z formularza do formatu, który pozwala wykorzystać model danych. Wysyłając dane za pomocą formularza:

```
<$php
        echo $html -> input('name', array('size' => 30));
        echo $tml -> submit('Wyslij');
?>
```

Zmienne zostaną przekazane w zmiennej *$this -> data*. Jeżeli teraz zostaną przekazane dane do funkcji *postConditons()* to w wyniku otrzymamy strukturę, dzięki, której możemy użyć metodę do przeszukiwania danych *AppModel -> find()*.

Przekierowanie

Funkcja *AppController -> redirect($url, status = null)* umożliwia przekierowanie aplikację pod nowy adres URL. Przykłady użycia funkcji *redirect()*:

```
<?php
// przekierowanie do innego serwisu
$this -> redirect('http://www.kurnik.pl');
// przekierowanie do strony głównej
$this -> redirect('/');
// przekierowanie do innej akcji
$this -> redirect('/kontroler/akcja');
?>
```

Funkcja *AppContoller -> flash($message, $url, $pause = 1)* umożliwia wyświetlenie wiadomości ($message) przez okres czasu (*$pause*) a następnie przenosi użytkownika pod adres (*$url*). Jeżeli użyjemy funkcji *redirect()* lub *flash()* działanie aktywnej akcji nie zostanie zakończone. Bieżącą akcję należy zatrzymać funkcją *exit().*

Walidacja danych

Funkcja AppContoller -> validateError($model[$model…]) uruchamia proces walidacji danych znajdujący się w zmiennej $this -> data. W modelu danych wywołana jest metoda AppModel -> invalidFields(), która zwraca tablicę z nazwami pól niezgodne z regułami walidacji zdefiniowanej w zmiennej AppModel -> validate().

Funkcja callback

W kontrolerze możemy używać funkcji callback wywoływanych przed określonym zdarzeniem albo po nim. W ten sposób można budować, np. mechanizmy autentykacji, logowania. CakePHP znajdują się dwa typy filtrów: before() oraz beforeFilter(), afterFilter(). Przykład użycia funkcji beforeFilter(), która jest wywoływana przed logowaniem użytkownika do systemu:

```
public function beforeFilter(){
// dostęp użytkownika przed logowaniem
$this -> Auth -> allow('display', 'registration', 'sendEmail');
}
```

Funkcja alfterFilter() używana jest do modyfikacji danych wysyłanych do przeglądarki użytkownika. Funkcja może podmienić wybrane fragmenty tekstu i znaczniki HTML.

Logowanie zdarzeń

CakePHP pozwala na zapisanie informacji związanych z działaniem aplikacji, np. zapis aktywności użytkowników. Dane są zapisywane w katalogu app/tmp/. Funkcja *ApppController -> log($message, type = LOG_ERROR)* umożliwia zapisanie informacji do pliku logu. Przykład zapisu za pomocą funkcji log():

```
<$php
        $this -> log('Aplikacja nie działa');
        // 2012-04-01 12:07:30 Error: aplikacja przestała działać
        $this->log('formularz'.(empty($this -> data))? 'Nie zawiera danych': 'zawiera dane'), LOG_DEBUG);
        // 2012-04-01 13:07:30 Debug: formularz nie zawiera danych
        ?>
```

Funkcja Object -> log(), fizyczny zapis do piku realizowany jest za pomocą metody

CakeLog -> write() zdefiniowanej w pliku *cake/libs/cake_log.php*.

>

Zmienne

Jeżeli są używane w aplikacji interpreter PHP w wersji 4, należy zdefiniować zmienną *AppControlle -> $name*, która zawiera nazwę klasy.

Nazwa zmiennej *AppController -> uses* pozwala nam na korzystanie z modeli danych. Kod aplikacji powinien mieć postać *var $uses = array('Question', 'Learn').*

Helpers AppController -> helpers to obiekty grupujące funkcje wykorzystywane w widokach do zmiany formatu wyświetlanych danych. Kod aplikacji powinien mieć postać *var $helpers = array('Html').* Użyte pliki muszą znajdować się w katalogu *app/views/helpers/*.

W katalogu cake */libs/view/helpers/* znajdują się klasy należące do CakePHP.

Zmienna *AppController -> layout* zawiera nazwę szablonu aplikacji, do której jest dołączony kod generowany w poszczególnych widokach stron. Wartością domyślną jest *default* znajdujący się w katalogu *app/views/layouts/default.ctp*.

Każda akcja kończy się automatycznym wywołaniem odpowiedniego widoku. Jeżeli zmienna *$AppController -> autoRender* zawiera wartość *false* framework będzie oczekiwać wywołania metody *AppController -> render()*.

Zmienna *AppController -> componets* musi być przepisana tablica zawierająca nazwę komponentów wykorzystanych przez aplikację np. *$components = array('mail')*. Komponenty znajdują się w katalogu *app/controllers/components/*. Wbudowane komponenty znajdują się w katalogu *cake/libs/controller -> output* zawiera wynik działania akcji lub kod wygenerowany przez widok.

Zmienna *AppController -> pageTitle* pozwala zmienić tytuł wygenerowanej strony HTML.

Parametry

Parametry *AppController -> data* pozwala na pobranie w kontrolerze informacji wysłanych przez formularz. Zmienna data ma strukturę tablicy asocjacyjnej, w której kluczami są nazwy modeli danych. Poniższy kod przedstawia przekazanie danych z formularza do kontrolera

```
<?php
        echo $this->Form->create(array('action'=>'login'));
        echo $this->Form->inputs(array(
                'legend' => 'Logowanie',
                'username',
                'password'
                ));
        echo $this->Form->end('Logowanie');
?>
```

Dane przekazane do kontrolera *UsersController*

```
Array(
        [data] => Array(
                [User] => Array(
                        [username] => Admin
                        [password] => 123
                )
```

```
    )
)
```

Zmienna *$params* zawiera tablicę z informacjami głównie rutera, np. nazwa wywoływanego kontrolera, oraz parametr, które przekazane są w adresie URL.

Sesje

Sesje pozwalają na przechowywanie danych miedzy kolejnymi wywoływaniami serwisu. Pozwalają one na zapisywanie niewielkich paczek danych. Dzięki serializacji możliwe jest przechowanie obiektów. Sesje tworzone są na serwerze WWW. Zapis danych do sesji odbywa się za pomocą *$this -> Session -> write('zmienna', 'wartość')* odczyt $*this -> Session -> read('zmienna').* Funkcja *$Session -> check($name)* sprawdza czy w sesji pod kluczem o określonej nazwie została zdefiniowana jakaś wartość. Usuwanie sesji odbywa się za pomocą *Session -> del($name).* Na zwrócenie ostatniego komunikatu o błędzie pozwala funkcja *Session -> error().* Funkcja *Session -> setFlash($flashmessage, $layout = 'default', $params = array(), $key = 'flash')* pozwala zapisać do sesji wiadomość *$flashMessage*, która może zostać wyświetlona za pomocą funkcji *Session -> flash().* Parametr *$layout* decyduje w jaki sposób umieszczona jest wiadomość. Domyślna jest *<div class = „Messager"> $flashMessage *. Parametr *$layouts* pozwala na określenie szablonu wiadomości przed zapisem w sesji. Parametr *$params* definiuje wartości, które są przekazane widokowi. Parametr *$key* określa klucz do przechowania wiadomości. Przykład zapisu komunikatu za pomocą funkcji *setFlash()*:

```
<?php
    $session -> setFlash('Konto użytkownika zostało utworzone'.$username);
?>
```

Utworzenie samodzielnej wiadomości i zapis do sesji

```
<?php
    $key = 'flash';
    $flashMessage = 'Konto użytkownika zostało utworzone' . $username);
    $out = '<div id=" ' .$key . 'Message" class = "message">'. $flashMessage .
'</div>';
    $session -> write('Message. ' . $key, $out);
```

Funkcja *Session -> flash($key = 'flash')* pozwala na odczyt i zwrócenie komunikatu użytkownika zapisanej w sesji za pomocą funkcji *setFlash().*

Funkcja *Session -> read($name)* pozwala na odczyt z sesji.

Funkcja *Session -> write($name, $value)* zapisuje do sesji wartości *$value*.
Sprawdzenie czy sesja jest poprawna umożliwia nam funkcja *Session -> valid().* Funkcja testuje, czy nie upłynął termin ważności i czy żądanie dostępu do danych pochodzi z tego samego komputera. Funkcja jest automatycznie wywoływana przed odczytem i zapisem danych do sesji.
Funkcja *Session -> renew()* pozwala na zabezpieczenie danych przez ponowne stworzenie nowych i usuniecie starych. Dane przed usunięciem zostaną skopiowane do nowej sesji.

3.4 Widok

Widok odpowiada za prezentacje. Nie komunikuje się z modelem danych czy użytkownikiem. Za te czynności odpowiedzialny jest model danych i kontroler. Widok jest połączeniem szablonu wyglądu danych udostępnionych przez kontroler i kodu, który wyświetla dynamicznie zmieniające się dane. Za udostępnienie danych w kontrolerze jest odpowiedzialna metoda AppControler -> set(). W widoku kod PHP nie ma dostępu do innych zmiennych używanych w aplikacji. Dostęp jest możliwy do zmiennych globalnych, tablic zawierających dane przekazywane do formularzy, sesji itp. W widoku możemy posługiwać się obiektem helper, który jest deklarowany w kontrolerze. Strony zawierające fragmenty, które nie ulegają zmianie, np. nagłówek strony, stopka oraz logo serwisu można umieści w szablonie. Pozwala nam to na brak powielania kodu. W CakePHP wymagane są pewne zasady w nazywaniu plików widoków. Nazwa pliku jest taka sama jak nazwa metody kontrolera, którą obsługuje widok. Małymi literami zapisujemy nazwę widoku, ze znakiem podkreślenia zamiast spacji.

3.4.1 Layout

Elementy wspólne dla wyglądu aplikacji powinny być umieszczane w plikach szablonów. Szablony znajdują się w katalogu *app/views/layout/* lub *cake/libs/view/tmplates/layouts.* Domyślny szablon ma nazwę *default.ctp*. Przykład szablonu serwisu wyświetlającego dane:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title><?php echo $title_for_layout?></title>
<head>
<body>
```

```
        <!--Nagłówek strony -->
        <div id="header"> <!--kod HTML --> </div>

        <!—Wyswietlanie poszczególnych widoków aplikacji -->
        <div id="content">
                <?php echo $content_for_layout; ?> <!-- Treść strony -->
        </div>
        <!--Stopka strony-->
        <div id=”footer” <!--kod HTML --> </div>
</body>
</html>
```

Zmienne *$title_for_layout* oraz *$content_for_layout* są automatycznie inicjalizowane przez CakePHP. Zawiera on tytuł strony i zawartość widoku wygenerowanego po zakończeniu obsługi akcji. Tytuł strony można zmienić za pomocą atrybutu *AppController -> pageTitle*.

3.4.3 Elementy widoku

W aplikacjach znajdują się informacje, które musza być wyświetlane na wielu stronach, np. formularze, przyciski nawigacyjne itp. Mogą być wydzielane ze struktury widoku. Dzięki takiemu rozwiązaniu nie jest powielany kod aplikacji. Elementy są mini widokami, które są dołączane do widoków. Elementy mają rozszerzenie *ctp*. Znajdują się w katalogu *app/views/elements/*. Funkcja *View -> renderElement()* odpowiedzialna jest za wyświetlenie elementu przyjmującego dwa parametry: nazwę i tablicę, która zawiera dodatkowe zmienne.

3.4.4 Helpers

Często przed wyświetleniem dane muszą być przeformatowane. Modyfikacje przeprowadzane są w kontrolerze, jeżeli dotyczą zmiany struktury danych. Modyfikacje prostego formatu, np. sposób wyświetlenia, są przeprowadzane w widokach za pomocą funkcji zdefiniowanych w obiektach klasy Helper. Jej nazwa jest powiązana z nazwą pliku, zawiera przyrostek *Helper*, każdy wyraz jest zapisany wielką literą, bez spacji między wyrazami. Pliki helper znajdują się w katalogu *app/views/helpers/* lub systemowym katalogu *cake/libs/view/helpers/*.

Funkcja *HtmlHelper -> url($url = null, $return false)* odnajduje pełen adres URL. Zmienna *$url* bazuje na nazwie kontrolera i akcji. W przypadku braku parametru funkcja odszuka domyślny adres.

Funkcja *HtmlHelper -> charset($charset, $return = false)* tworzy-META znaczniki, które określają standard kodowania znaków.

Wywołanie funkcji *$html -> charset('utf-8')* odpowiada *<meta http-equiv="Content-Type" kontent="text/html; char set = utf-8" />*.

Funkcja, która tworzy link do arkusza CSS jest *HtmlHelper -> css($path, $rel = 'stylesheet', $htmlAttributes = null, $return=false).* Parametr *$path* określa położenie pliku. W nazwie nie należy podawać rozszerzenia .css. Wywołanie funkcji *$html -> css('cake.generic')* odpowiada *<link rel="stylesheet" type="text/css" href="/css/cake.generic.css" />*

Metoda *HtmlHelper -> tagErrorMsg($field, $text)* tworzy komunikat o błędzie *$text*, kiedy paramtr $field znajduje się w tablicy błedów funkcja odpowiada * $text *.

Funkcja HtmlHelper -> *tableHeaders($name, $trOptions = null, $thOptions = null, $return=false)* tworzy pojedynczy wiersz zawierający dane *<tr><td></td></tr>*. Parametr *$name* zawiera tablicę z nazwami nagłówków.

W kontrolerze deklarujemy z jakich klas widok może korzystać. Jeżeli zmienna *$helpers = array('Html')*, to w widoku można korzystać z zmiennej *$html*. W Framework CakePHP są gotowe do użycia klasy zapisane w plikach, które znajdują się w katalogu *cake/libs/view/hel pers/*. Funkcje, które w nich się znajdują ułatwiają formatowanie danych (Number, Text, Tmie), tworzenie interfejsu (Html, Form, Javascrip) i tworzenie serwisów zawansowanych (Ajax, Cache, Session).

Funkcja *HtmlHelper -> image($path, $htmlAttributes=null, $return=false*) umożliwia wyświetlenie znaczników *<img>,* które służą do wyświetlenia plików graficznych.

Funkcja *HtmlHelper -> link($title, $url = null, $htmlAttriutes = null, $confirmMessage = false, $escapeTitle = true, $return = false)* pozwala na utworzenie odnośnika *$title>*. Paramer *$confirmMessage* możemy wymusić wyświetlenie potwierdzenia zanim nastąpi przekierowanie pod nowy adres. Funkcja *Helper -> output($str, $return = false)* umożliwia zwrócenie tekstu *$str*.

3.4.4 HtmlHelper

Klasa *HtmlHelper* powstała, żeby maksymalnie uprościć budowę interfejsu graficznego serwisu. Wyniki działania są zgodne standardem XHTML. Funkcja *HtmlHeoper -> ta-*

bleCells($data, $oddTrOptions=null, VevenTrOptions = null, $return = false) tworzy pojedynczy wiersz, który zawiera dane tabeli odpowiadające *<tr><td> </td></tr>*. Parametr *$data* zawiera tablicę z danymi do umieszczenia w poszczególnych kolumnach.

Funkcje poniżej tworzą poszczególne elementy formularzy:

Html -> password($fielName, $htmlAttributes=null, $return=false);

HtmlHelper -> textarea($fieldName, $htmlAttributes=null, $return=false);

HtmlHelper -> input($fieldName, $htmlAttributes = null, $return = flase);

HtmlHelper -> radio($fieldname, $options, $inbetween = null, $htmlAttributes = array(), $return = false);

HtmlHelper -> checkbox($fieldname, $title=null, $htmlAttributes, $return=false);

HtmlHelper -> file($fieldName, $htmlAttributes = null, $return = false);

HtmlHelper -> hidden($fieldName, $htmlAttributes = null, $return=false);

Funkcje odpowiadają znaczikom HTML *<input>* i *<textarea>*. Parametr *$fieldName* zawiera nazwę modelu danych i nazwę pola formularza.

3.5 Model

W architekturze MVC model reprezentuje źródło danych. Jest on zbudowany o klasę AppModel, rozbudowaną o dodatkowe funkcje. Aplikacja, która wykorzystuje model, nazwa każdego z nich jest umieszczona w tablicy przepisanej do atrybutu AppController -> uses. Deklaracja *var $uses = array('name', 'surname')* spowoduje utworzenie zmiennych, które są dostępne w kontrolerze aplikacji. W katalogu *cake/app_model.php* umieszczany jest kod wspólny dla wszystkich modeli. ORM i warstwa abstrakcji dostępu do danych.

Język PHP pozwala na obsługę różnych baz danych. Powoduje to następujące problemy: różnice API wymusza tworzenie różnego kodu do obsługi każdej bazy. Składnia języka SQL jest odmienna w różnych bazach danych. Jeżeli zmieni się model danych, np. pojawienie się kolejnej kolumny w tabeli, trzeba poprawić całą aplikację we wszystkich miejscach, w których odwołaliśmy się do danych. Framework umożliwia obsługę baz danych różnymi API. W CakePHP dostęp do danych, odbywa się przez dodatkową warstwę obiektów, ukrywając niskopoziomowy kod związany z konkretną bazą danych. W Frameworku stosuje się technologię mapowania relacyjnych danych na obiekty ORM (ang. Object-Relational Mapping). Na przykład jeżeli znajduje się tabela messages w bazie danych, to program musi mieć zdefiniowaną klasę *Message*. Poszczególne wiersze z tabelki są automatycznie tworzone na obiekty klasy *Message*. Framework Cake-

PHP nie obsługuje technologi ORM. Zwrócone dane w wyniku zapytania nie są obiektami klasy modelu danych, ale strukturą tablic asocjacyjnych, w której kluczami są nazwy modelu danych i kolumny tabeli. CakePHP umożliwia tworzenie relacji między modelami. Zapytanie o dane jednego modelu pobiera automatycznie informacje z innych modeli. Wyniku mapowania relacyjnych danych na tablice asocjacyjne:

```
Array
(
  [0] => Array
  (
        [Learn] => Array
        (
              [id] => 23
              [user_id] => 25
              [question_id] => 1
              [container_number] => 1
              )

        [User] => Array
        (
              [id] => 25
              [username] => user1
              [password] => e4e84d280e578308a38de598b3d1c25cb0155594
              [active] => user
        )

        [Question] => Array
        (
              [id] => 1
              [categorie_id] => 1
              [question] => czy słowa Basic realm użyte przy uwierzytelnianiu HTTP są do czegoś przydatne?
              [answer] => tak. Przy ich użyciu można określić strefę bezpieczeństwa. Chronią konkretną parą nazwa-hasło. Kiedy użytkownik poda dane prawidłowe dla danej strefy, przeglądarka zapamięta to i nie będzie wyświetlać okna uwierzytelnienia związanego z tym samym obszarem. Oznacza to, że strefy pozwalają przegląd-
```

darce zapamiętać, iż dana osoba spełnia wymagania potrzebne do uzyskania dostępu do danego zbioru stron. Aby utworzyć taką grupę stron, należy podać tę samą strefę w ich nagłówkach uwierzytelnienia.

)

)

)

W CakePHP do opisu relacji używane są tablice asocjacyjne przepisywane do zmiennych o nazwach odpowiadających rodzajowi relacji. Framework umożliwia zdefiniowanie czterech rodzajów relacji: hasOne - relacja 1:1, hasMany - relacja 1:n, hasAndBelongsToMany - relacja n:m, belongsTo – relacja 1:1. Relacja n:m, belongsTo jest uzupełnieniem relacji hasMany i hasOne. Jeden plik może zawierać jeden model. Framework automatycznie odnajduje i wczytuje plik. Nazwa klasy zapisana jest w liczbie mnogiej, bez spacji między wyrazami. Każdy wyraz zapisany jest wielką literą np. *User*. Nazwa pliku zapisana jest małymi literami, ze znakiem podkreślenia '_' zamiast spacji. Przykład nazwy pliku: *user.php*. Pliki znajdują się w katalogu *app/models/*. Nazwa tabel w bazie zapisana jest w liczbie mnogiej, małymi literami ze znakiem podkreślenia '_' zamiast spacji. Podstawowy klucz tabeli musi nazywać się id. Nazwa klucza obcego powinna zawierać nazwę tabeli obcej w liczbie pojedynczej i _id np. *categorie_id*.

3.5.1 Relacje

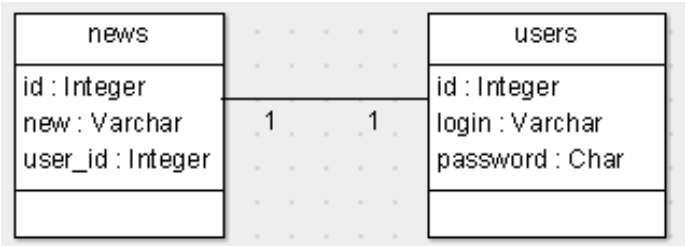
Relacja hasOne

Relacja hasOne tworzona jest przez przepisanie *zmiennej AppModel -> hasOne* tablicy asocjacyjnej . Przykład definicji relacji *hasOne*

```
<$php
        class News extends AppModel{
                var $name = 'New';
        var $hasOne = array(
                'Author' => array(
        'className' => 'User',
                        'conditions' => 'Author.active.active=1',
                        'order' => '',
                        'dependent' => false,
                        'foreignKey' => 'user_id',
```

```
        'fields' => 'Author.name'
        )
        );
        }
?>
```

Parametr *className* jest nazwą klasy modelu danych, w której jest w relacji. Nazwa *conditions* pozwala na tworzenie dodatkowego warunku na relacje, np. połączenie dotyczy tylko tych użytkowników, którzy mają aktywne konto (*Author.active = 1*). Parametr *order* odpowiedzialny jest za kolejność w jakiej mają być dołączone dane. Zmienna *dependent* pozwala na wymuszenie relacji. Jeżeli parametr zawiera wartości *true* powoduje usunięcie powiązanych danych. Parametr *foreignKey* jest nazwą klucza obcego. Pozwala na złączenie danych z modeli objętych relacją. Zmienna Fields pozwala na dołączenie kolumn, które są dołączone do wyniku zapytania. Domyślnie dołączane są wszystkie kolumny. Rysunek przedstawia relacje hasOne.



Rysunek 4 Relacja hasOne, wiadomość ma jednego autora

Relacja hasMany

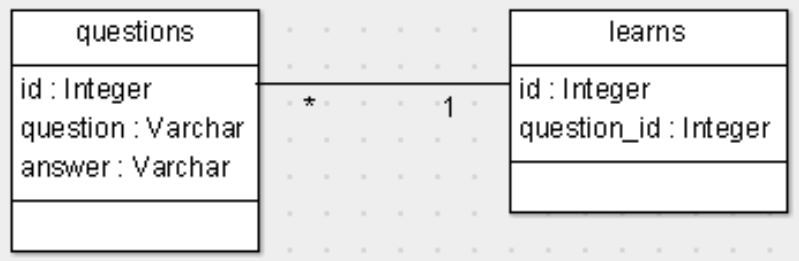
Relację *hasMany* tworzymy przez przepisanie zmiennej *AppModel -> hasMany* tablicy zawierającej parametry powiązania. Przykład relacji hasMany:

```
class Question extends AppModel {
        var $name = 'Question';
        var $hasMany = array(
                'Learn' => array(
                        'className' => 'Learn',
                        'foreignKey' => 'question_id',
                        'dependent' => false,
                        'conditions' => '',
                        'fields' => '',
                        'order' => '',
                        'limit' => '',
                        'offset' => '',
```

```
                'exclusive' => '',
                'finderQuery' => '',
                'counterQuery' => ''
            )
        );

}
```

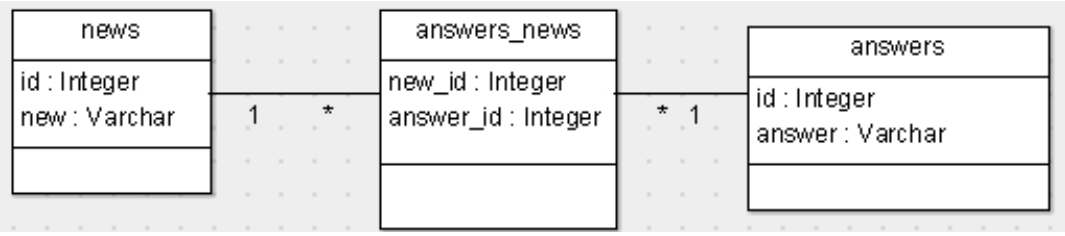
Parametr limit określa liczbę rekordów, które będą odczytane. Rysunek przedstawia relacje hasMany.



Rysunek 5 Relacja hasMany

Relacja hasAndBelongsToMany

Relacja hasAndBelongsToMany jest relacją wiele do wielu. Do jej utworzenia potrzebna jest tabela, która przechowuje informacje wierszy tabel tworzących relację. Framework wymaga, by nazwa tabeli pasowała do schematu [liczba_mnoga_model_1]_[liczba_mnoga_model_2]. Nazwy modeli danych muszą być w kolejności alfabetycznej. Klucze obce muszą pasować do wzorca [liczba_pojedyncza_model]_id. Rysunek przedstawia relacje hasAndBelongsToMany.



Rysunek 6 Relacja hasAndBelongsToMany, każda wiadomość może posiadać wiele odpowiedzi

Relacja wiele do wielu tworzona jest przez przepisanie zmiennej AppModel -> hasAndBelongsToMany tablicy, która zawiera parametry powiązania np.

```
<$php
    class New extends ApppModel
    {
    var $name = 'New';
```

```
        var $hasAndBelongsToMany = array(
                'Answer' => array(
        'className' => 'Answer'
        )
                );
        }
?>
<$php
        class Answer extends ApppModel
        {
        var $name = 'Answer';
        var $hasAndBelongsToMany = array(
                'New' => array(
        'className' => ‘New’
        )
                );
        }
?>
```

Relacja belongsTo

Relacja belongsTo tworzona jest przez przepisanie do zmiennej AppModel -> belongsTo tablicy asocjacyjnej

```
<?php
class Learn extends AppModel {
        var $name = 'Learn';
        //The Associations below have been created with all possible keys, those that are not needed can be removed

        var $belongsTo = array(
                'User' => array(
                        'className' => 'User',
                        'foreignKey' => 'user_id',
                        'conditions' => '',
                        'fields' => '',
                        'order' => ''
```

```
            ),
            'Question' => array(
                    'className' => 'Question',
                    'foreignKey' => 'question_id',
                    'conditions' => '',
                    'fields' => '',
                    'order' => ''
            )
    );
}
?>
```

Jest ona uzupełnieniem relacji hasOne i hasMany. Pozwala na zdefiniowanie połączenia w drugą stronę. Jeżeli zdefiniujemy za pomocą hasMany, że „pojemnik" zawiera wiele pytań, to za pomocą *belongsTo* uzupełniamy ją o powiązanie. Pytanie należy do „pojemnika".

Tworzenie relacji do tego samego modelu

Framework pozwala na dodawanie relacji do tego samego modelu. Przykładowo aplikacja przechowuje w jednej tabeli bazy danych pracowników i kierowników. Można stworzyć dwie relacje. Pracownik fizyczny ma jednego kierownika (hasOne), kierownik może mieć wielu podwładnych (hasMany). Zdefiniowane relacje umożliwiają zadanie zapytania o dane pracowników i informacji o jego kierowniku.

```
<?php
class Worker extends AppModel
{
    var $name = 'Worker';
    var $recursive = 1;
    // relacja podwładny ma tylko jednego przełożonego
    var $hasOne = array(
            'Manager' => array(
            'classname' => 'Worker',
            'foreignKey' => 'parent_id'
    )
);
// relacja szef może mieć wielu podwładnych
```

```
var $hasMany = array(
        'Employe' => array(
                'className' => ' Worker',
                'foreignKey' => 'parent_id'
        )
        );
}
?>
```

3.5.2 API modelu danych

Wyszukiwanie i pobieranie danych

CakePHP pozwala pobierać dane z bazy bez znajomości SQL. Dostęp do danych odbywa się za pomocą modelu danych, która reprezentuje jedną tabelę. Relacje między modelami pozwalają zwrócić informacje, dane pochodzące z powiązanych tabel. W wyniku zapytania zostają otrzymane tablice asocjacyjne.

Funkcja *find($conditions = null, Fields = null, order = null, $recursive = null)* umożliwia dostęp do jednego lub wszystkich wierszy, które spełniają warunki znajdujące się w zmiennej $conditions. Wiersze mogą być posortowane. Odpowiada za to parametr $order.

Funkcja *findAll($conditions = null, $fields = null, $order – null, $limit = null, $page = 1, $recursive = null)* ma podobne właściwości jak funkcja find(). Pozwala ona jendak określić, do której części wyników chcemy mieć wgląd. Odpowiedzialne są za to parametry $limit (liczba wierszy) oraz $page (strona zawierająca $limit wierszy danych). Parametr $recursive określa jak głęboko CakePHP ma poszukiwać powiązań między modelami.

Warunki zapytania możemy przekazywać, jako tablicę asocjacyjną lub ciąg znaków. Dane różniące się od określonej wartości można wyszukać *Message -> find('Message.id' => "<> $id").* Można też używać innych operatorów porównania i operatorów SQL np. *LIKE*, *BETWEEN*.

W zapytania CakePHP łączymy warunki zapytania za pomocą operatora logicznego AND. Można też budować warunki zawierające operatory OR, NOT i XOR. Operator IN pozwala na tworzenie warunków badających przynależność do określonego zbioru.

Funkcja AppModel -> read($field = null, $id = null) odczytuje pojedynczy wiersz z tabeli bez uwzględnienia relacji z innymi modelami danych. Wynik działania ograniczyć można do pożądanych kolumn za pomocą zmiennej $fields.
Funkcja AppModel -> getID($list = 0) zwraca identyfikator związany z aktualnym rekordem.
Funkcja appModel -> getNumRows() zwraca liczbę rekordów odczytanych za pomocą ostatniego zapytania.
Funkcja AppModel -> hasField($name) zwraca wartość true, gdy model, który obsługuje tablice zawiera kolumnę $name.
Funkcja AppModel -> exists($id) zwraca wartość true, gdy w bazie istnieje wiersz o identyfikatorze $id.

Zapisywanie

Model danych może zapisywać dane tylko do swojej tabeli. Zapisanie nowego rekordu wymaga przekazanie odpowiedniej struktury danych, zagnieżdżonych tablic w PHP. Funkcje AppModel -> red(), AppModel -> find() zwracają w odpowiednim formacie dane, które można zapisać. Również dane wysyłane przez formularz i przekazane zmiennej AppModel -> data są przygotowane do przekazania do funkcji AppModel -> save().
Funkcja AppModel -> save($data = ull, $validate = true, $fieldlist = array()) zapisuje dane do bazy danych. Przed utworzeniem nowego wiersza w tabeli przeprowadzana jest walidacja. Zmienna $filedlist przekazuje do funkcji listę kolumn, które można modyfikować.
Aktualizacja istniejących danych przeprowadzana jest w oparciu AppModel -> id. Zmienna zawierająca identyfikator znajdujący się w tabelce umożliwia aktualizację. Zapis danych spowoduje przepisanie do AppModel -> id nowej wartości.
Funkcja AppModel -> saveField($name, $value, $validate = false) modyfikuje dane znajdujące się w kolumnie $name, przepisując wartość $value (wiersz o identyfikatorze AppModel -> id jest modyfikowany). Przed zmianą danych sprawdzany jest rekord, gdy nie istnieje, zostanie utworzony wiersz z domyślnymi wartościami.

Usuwanie danych

Powiązania między modelami brane są podczas usuwania rekordów z tabel. Parametr *dependent* używany w relacji hasOne i hasMany, jeżeli ma wartość true, to podczas usuwania danych z jednego modelu usuwane są automatycznie dane z drugiego modelu.

Funkcja AppModel -> del($id=null, $cascade = true), AppModel -> delete($id = null, $cascade = true) usuwa z tabeli wiersze danych o identyfikatorze $id. Parametr $cascade pozwala na usunięcie powiązań danych z innych modeli, jeżeli taka relacja występuje. Funkcja zwraca wartość true, gdy operacja zakończy się sukcesem.

Zapytanie SQL

W Frameworku udostępnia API umożliwiające przesłanie zapytań w postaci kodu SQL. Funkcja AppModel -> query($sql) zwraca wynik zapytania SQL przekazanego w parametrze $sql.

Walidacja

Funkcja AppModel -> invalidFields() zwraca błędy, które nie są zgodne z regułami walidacji zdefiniowanej w zmiennej AppModel -> validate. Funkcja AppModel -> invalidate() pozwala na ręczne wymuszenie błędu walidacji pola przekazanego jako parametr. Obie powyższe funkcje współpracują z metodą HtmlHelper -> tagErrorMsg().

Dynamiczna zmiana relacji między modelami

Funkcje AppModel -> binModel($params), AppModel -> unbinModel($params) umożliwiają dynamiczne tworzenie lub usuwanie relacji z innymi modelami danych. W trakcie tworzenia powiązania musi zostać przekazana tablica zawierająca typ oraz zależność np. *AppModel -> unbinModel(array('typPowiązania' => array('nazwaPowiązanejKlasyModelu')))*.

Funkcja celbak

Funkcje AppModel -> beforeFind(&$conditions), AppModel -> afterFind($results) pozwalają na automatyczne wywołanie przed metodami, które wyszukują dane: find, findAll. Funkcja AppModel -> beforeSave() zostanie wywołana przed zapisem danych w tabeli. Funkcje AppModel -> afterDelete(),AppModel -> beforeDelete() zostaną automatycznie wywołane przed uruchomieniem i o uruchomieniu metody delete(), która usuwa dane z tabeli.

Zmienne

Zmienna AppModel -> data zawiera dane przekazane do formularza, zawiera te same dane co tablica AppController -> data. Zmienna AppModel -> primaryKey pozwala na przepisanie nowej kolumny z kluczem, jeżeli go nie ma.

Zmienna AppModel -> recursive umożliwia określenie jak „głęboko” ma działać poszukiwanie powiązań.

Zmienna AppModel -> useTable pozwala na wskazanie dowolnej tabeli, która będzie reprezentowała model danych (nie trzeba stosować konwencji nazw stosowanych w CakePHP). Można także przypisać zmiennej wartość false gdy nie korzysta się z dostępu do bazy danych, np. modelem danych są dane z pliku.

Zmienna AppModel -> validate umożliwia zdefiniowanie reguł, które muszą być spełnione zanim wprowadzone dane zostaną uznane za poprawne. Przykład zastosowania walidacji:

```
var $validat = array(
        'user' => VALID_NOT_EMPTY,
        'email' => VALID_EMAIL,
        'title' => VALID_NOT_EMPTY,
        'content' => VALID_NOT_EMPTY
);} ?>
```

CakePHP posiad wbudowane reguły. Są one zdefiniowane w pliku cake/libs/validators.php.

Biała lista AppModel -> whitelist jest mechanizmem filtrowania danych. Pozwala na zapis informacji tylko do wybranych kolumn tabeli. Aktywacja następuje przez przepisanie do zmiennej $whitelist tablicy, która zawiera nazwy kolumn.

3.5.3 Porównanie frameworków CakePHP, Zend

Tabelka poniżej przedstawia najważniejsze cechy frameworków CakePHP, Zend.

CakePHP	**Zend**
Informacje podstawowe	**Informacje podstawowe**
-Framework PHP. -Stworzony do realizacji wzorca projektowego Model View Component.	-Framework PHP -Stworzony do realizacji wzorca projektowego Model View Component.
Wymagania	**Wymagania**
-PHP 4 i 5 -Apache z mod_rewrite, niekoniecznie	-PHP 5 -Apache z mod_rewrite -Na licencji New BSD.

-Na licencji MIT. -Framework zajmuje około 1MB.	-Framework zajmuje około 20MB.
Bazy Danych	**Bazy Danych**
MySQL4+ PostgreSQL ADODB	MySQL(mysqli oraz PDO) PostgreSQL Oracle(oci8 oraz PDO) IIBM DB2 & Infomix Dynamic Server SQLite
Struktura	**Struktura**
/app /config /controllers /components /models /plugins /tmp /vendors /views /helpers /webroot /cake /vendors	/application /config /controllers /models /views /filters /helpers /scripts /library /public /images /scripts /styles
Mapowanie URLi	**Mapowanie URLi**
http://serwer/controller/action/ parameter1/ parameter2	http://serwer/controller/action/param name1/ parameter_value1 / parameter_name2/ parameter_value2
Kontrolery Akcji	**Kontrolery Akcji**
Umieszczone są w katalogu wapp/controllers/[name] controller.php.	Umieszczone są w katalogu application/controllers/[Name]Controller.ph

Deklaracja kontrolera: class [Name]Controll erex- tends AppController Deklaracja akcji: function [name]() Definicje użytych helperów, komponentów, modeli oraz filtrów w atrybutach kontrolera.	p. Deklaracja kontrolera: Class [Name]Controller extends Zend Controller Action Deklaracja akcji: public function [name]Action() Helpery, filtry i modele deklarowane są w momencie użycia
Modele	**Modele**
-Umieszczane w app/models/[model].php. -Deklaracja modelu: class [Name] extends AppModel -Nazewnictwo specyficzne atrybutów tabeli w bazie danych. -Asocjacje są definiowane w atrybutach: $hasOne $hasMany $belongsTo $hasAndBelongsTo	-Umieszczane w katalogu application/models/[Model].php. -Deklaracja modelu: class [Name] extends Zend_Db_Table_Abstract -Możliwość wykonania operacji kaskadowych UPDATE i DELETE na bazach, które nie sprawdzają integralności danych (np. MyISAM w MySQL lub SQLite). -Możliwość znalezienia rekordów, które są powiązane: findParent[Model]() findParent[Model]By[Relacja]() find[Model]() find[Model]By[Relacja]() find[Model]Via[Model_złączeń] find[Model]Via[Model_złączeń]By[Relacja] find[Model]Via[Model_złączeń]By[Relacja_1]And[Relacja_2] findDependentRowset() findParentRow() findManyToManyRowset() -Wyszukiwaniepoprzezmetody:

	find() fetchRow() fetchAll() -Danych są dodawane poprzez metodę insert(). -Dane są modyfikowane poprzez metodę update(). -Dane są usuwane poprzez metodę delete().
Widoki	**Widoki**
-Umieszczane wapp/views/[kontroler]/[akcja].ctp -przekazywanie parametru poprzez metodę set() kontrolera	-Umieszczane wapp/views/[kontroler]/[akcja].phtml. -Parametry są przekazywane poprzez ustawienie odpowiedniego atrybutu w widoku.
Praca z Bazą Danych	**Praca z Bazą Danych**
-Operacje na zagnieżdżonych tablicach asocjacyjnych, które są powiązane zależnościami zdefiniowanymi w modelu **-**Ustawienie parametru definiującego zagnieżdżenie znajdywania powiązań (recursive) -możliwość tworzenia zapytań SQL za pomocą metody query	-Operacje na obiektach: Zend_Db_Select Zend_Db_Table Zend_Db_Table_Row Zend_Db_Table_Rowset Zend_Db_Statement, **-**Zagnieżdżone wyszukiwanie poprzez wywołanie odpowiedniej metody -tworzenie zapytania SELECT przez wywoływanie metod na obiekcie Zend_Db_Select -Warunki przekazywane są jako obiekty Zend_Db_Expr i są automatycznie escape'owane
Walidacja Danych	**Walidacja Danych**
-Dostępna dla atrybutów	-Dostęp do danych poprzez obiekty

modeli. -Oparta na wyrażeniach regularnych	klasy Zend_Validate. -Jest możliwe zdefiniowania wielu kryteriów dla pojedynczych danych
Pluginy	**Pluginy**
-Znajduje się w katalogu /app/plugins -CakePHP posiada własne kontrolery, widoki i modele -nie może posiadać własnych komponentów -Nie może posiadać odrębnych baz danych -Ruter nie potrafi przekierowywać zdefiniowanych akcji do pluginów	-Pluginy umieszczane w katalogu /application/[moduł]. -Mogą posiadać własne niemal wszystkie elementy frameworka.
Zgodność ze standardami rozszerzalność	**Zgodność ze standardami rozszerzalność**

-CakePHP oferuje kompatybilność PHP do wersji 4.3.2 -pracuje z niską wartością raportowania błędów w PHP (E ERROR) - kontrolery, komponenty, widoki, modele mogą być przeciążone. Brak możliwości przeciążenia Rutera -cakePHP wymusza ustalone nazewnictwo i strukturę bazy danych oraz plików	-pracuje maksymalnym poziomem raportowania błędów (E ALLiE STRICT). -przeciążenie niemal każdego elementu Frameworka -Zend wymusza ustalone nazewnictwo i strukturę bazy danych oraz plików

Tabela 1

Framework Zend jest potężnym framworkiem. Wymaga większego nakładu pracy aby go opanować niż CakePHP. Zend został zaprojektowany z myślą o dużych projektach, które umożliwiają tworzenie skomplikowanej architektury systemu. Poprzez większy poziom skomplikowania, daje on również większe możliwości. Nadaje się dla programistów z większym doświadczeniem. Framwork jest dobrze udokumentowany i ma wsparcie w społeczności programistów. Framwork CakePHP jest dobrze udokumentowany, duży zbiór klas „Helperów" sprawia, że programowanie jest przyjemne.

4. Sieć semantyczna aplikacji

W pracy magisterskiej został stworzony sieć semantyczna języka PHP. Na podstawie relacyjnej bazy danych aplikacji służącej do przeprowadzania powtórek zostały wygenerowane instancje, które następnie uzupełniono o klasy. Pierwsze dwa poziomy klas powstały jedynie dzięki bazie danych. Natomiast cała wielopoziomowa, z wieloma powiązaniami sieć stworzona została na podstawie wiedzy o języku PHP nabytej dzięki użyciu aplikacji do przeprowadzania powtórek. Poniżej przedstawiona jest struktura sieci semantycznej języka PHP, będąca efektem pracy.

4.1 Sieć semantyczna

4.1.1 Diagram klas

Sieć semantyczna składa się z pięciu głównych klas: *Baza_danych, Kontroler, Model, Uzytkownik, Widok.* Ilustracja poniżej przedstawia hierarchie klas w sieci semantycznej.

Ilustracja 1 Hierarchia klas

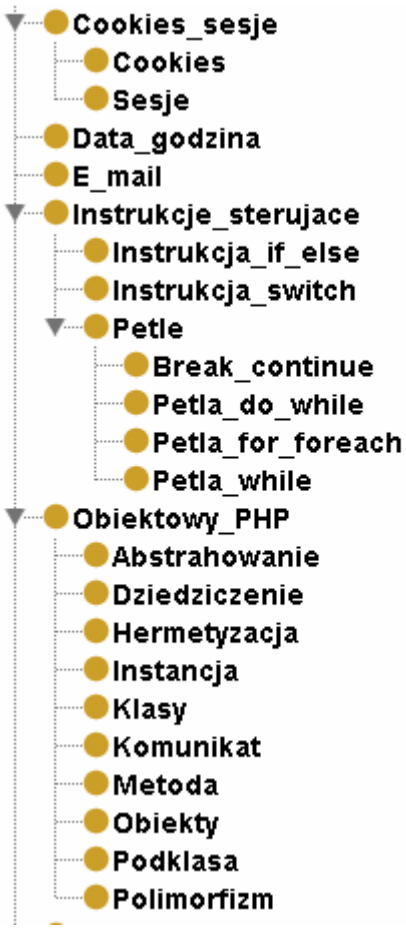
Klasa *Baza_danych* jest podzielona na podklasy. Rysunek poniżej.

Ilustracja 2 Baza_danych

Ilustracja poniżej przedstawia hierarchię klas *Tabela_Categories_Questions.*

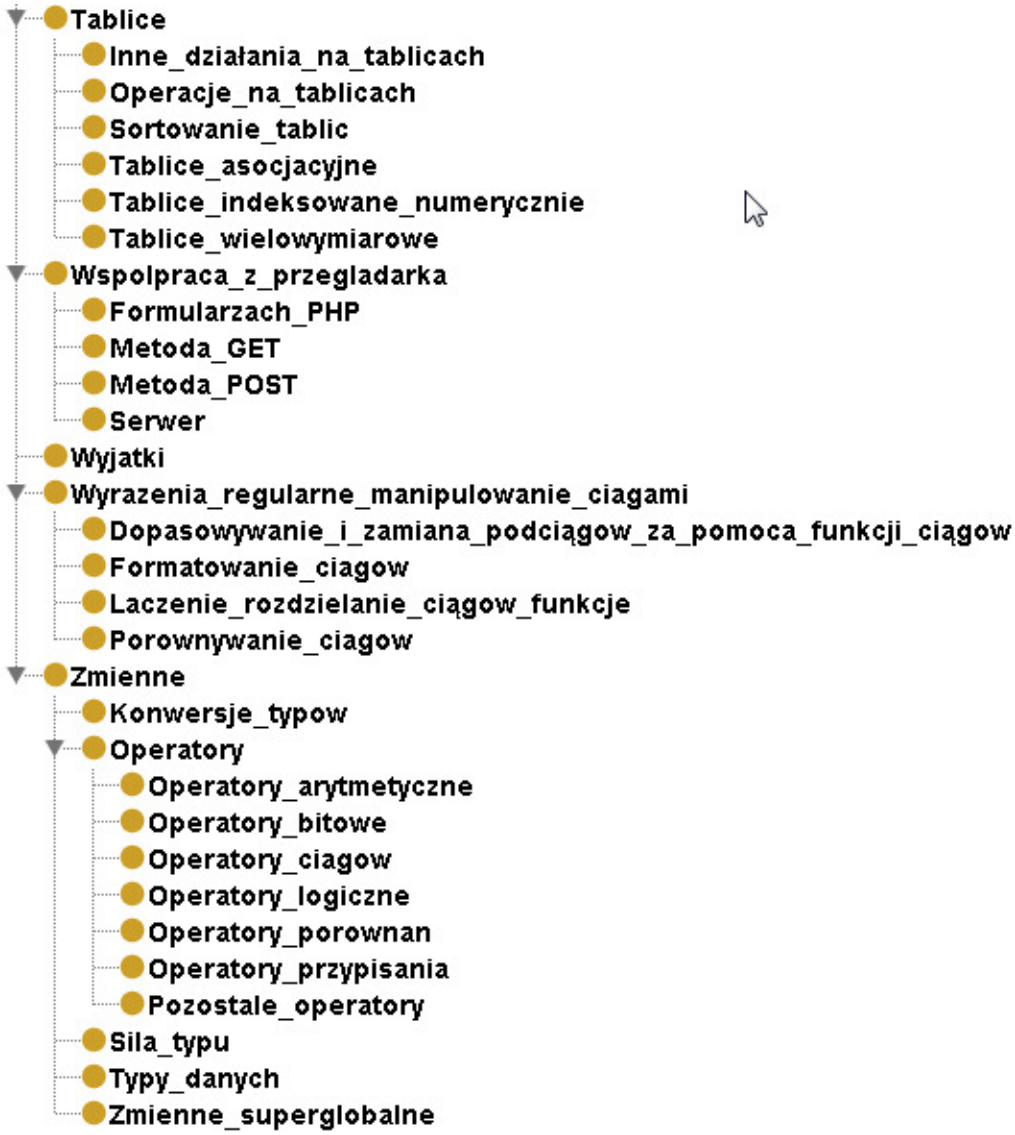
Ilustracja 3 Hierarchia klas Baza_danych_PHP_MySQL

Zdjęcia poniżej przedstawiają rozwinięcie klas: *Cookies_sesje, Petle, Obiektowy_PHP, Podstawy_SQL, Podstawy_PHP, Ponowne_wykorzystanie_kodu, Pliki, Tablice, Wspolpraca_z_przegladarka, Wyrazy_regularne_manipulowanie_ciagami, Operatory*.



Ilustracja 4 Rozwinięcie klas: Cookies_sesje, Petle, Obiektowy_PHP

Ilustracja 5 Rozwinięcie klas: Podstawy_SQL, Podstawy_PHP, Ponowne_wykorzystanie_kodu, Pliki

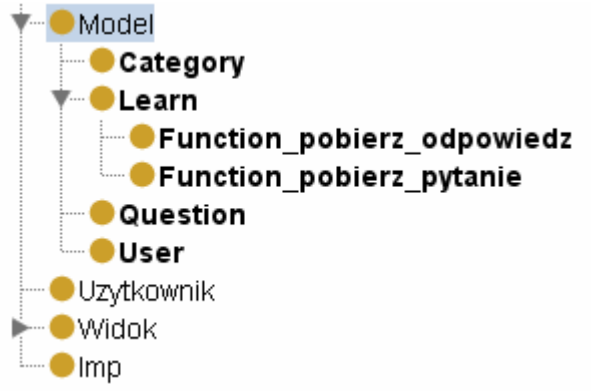


Ilustracja 6 Rozwinięcie klas: Tablice, Wspolpraca_z_przegladarka, Wyrazy_regularne_manipulowanie_ciagami, Operatory

Rozwinięcie klasy widok

Ilustracja 7 Rozwinięcie klasy Widok

Rozwinięcie klasy Model



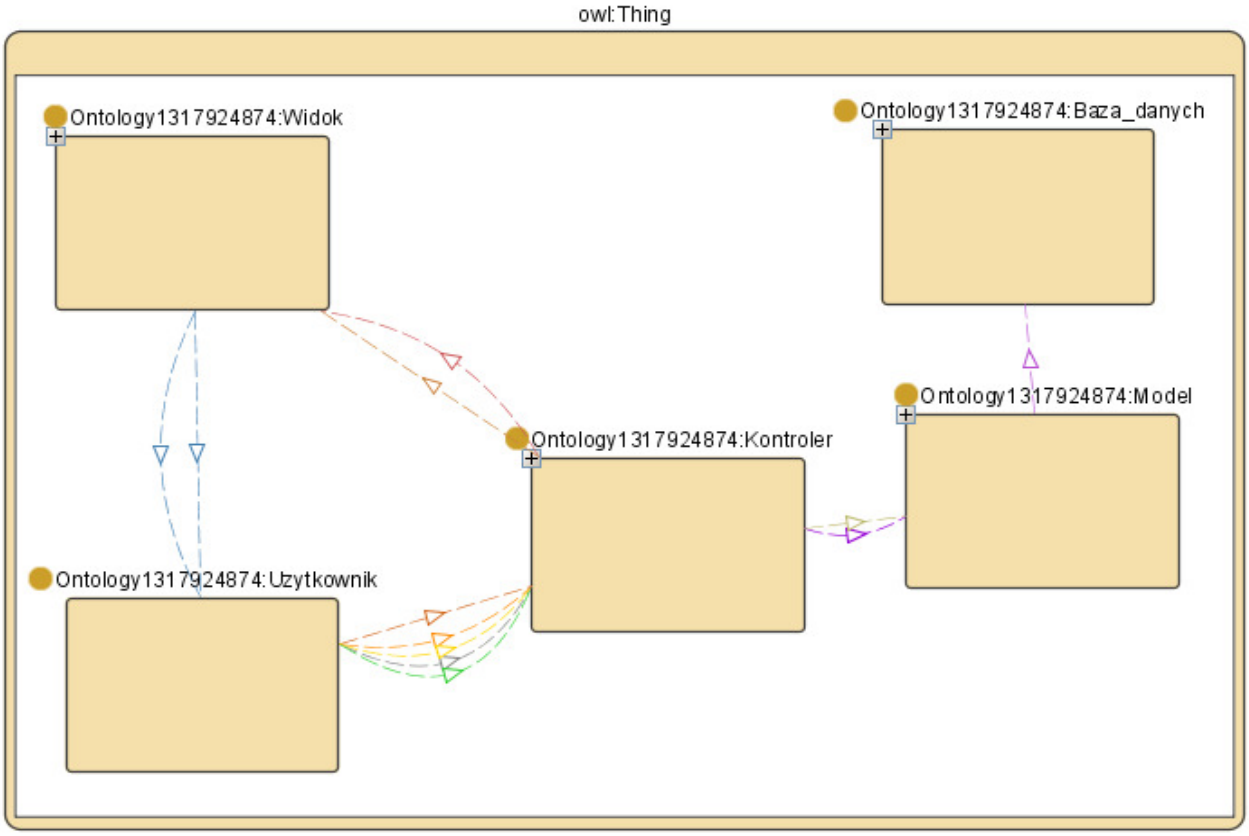
Ilustracja 8 Rozwinięcie klasy Model

4.1.2 Object Properties

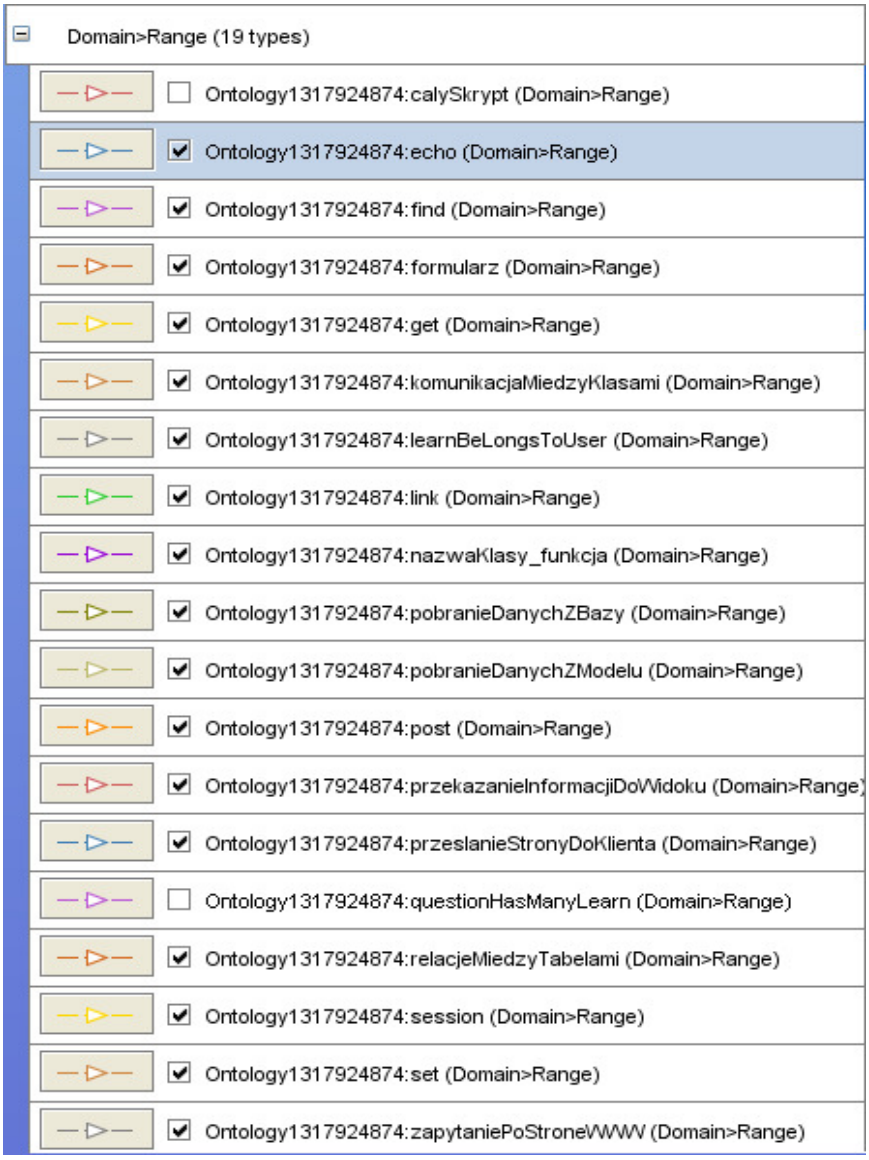
Object Properties pozwalają na tworzenie powiązań między klasami, obiektami. Ilustracja poniżej przedstawia ich przykłady.

Ilustracja 7 Object Properties

Rysunek poniżej został wygenerowany. Przedstawia powiązania między klasami i główną idę modelu MVC.



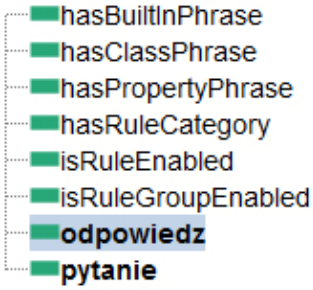
Ilustracja 8 Powiązania między klasami, model MVC



Ilustracja 8 Legenda, strzałki oznaczają relacje między klasami

4.1.3 Data Properties

Sieć semantyczna zawiera dwie Data Properties: *pytanie*, *odpowiedz.* Rysunek poniżej przedstawia ich przykłady.

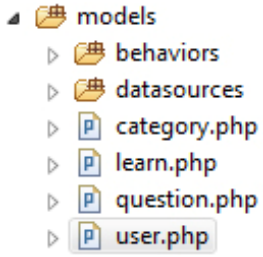


Ilustracja 9 Data Properties

4.2 Aplikacja

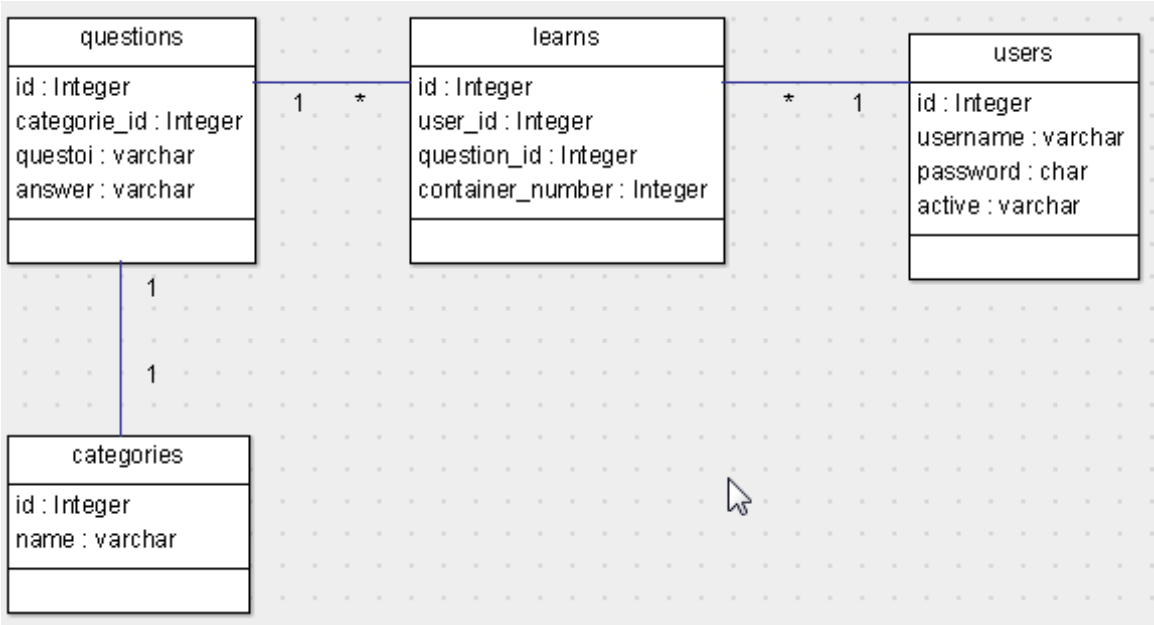
4.2.1 Powiązania między modelami

Aplikacja wspomagająca przeprowadzenie powtórek składa się z czterech modeli: *category, learn, qestion, user*. Ilustracja poniżej przedstawia modele aplikacji.



Ilustracja 1 Modele

Poniżej przedstawiam relacje między tabelkami w bazie danych.



Rysunek 1 Relacyjna baza danych

W relacyjnej bazie danych pytanie (question) może należeć do jednej kategorii (category). Relacji między klasami *Question* a *Category*

```
var $belongsTo = array(
        'Categorie' => array(
            'className' => 'Categorie',
            'foreignKey' => 'categorie_id',
            'conditions' => '',
            'fields' => '',
            'order' => ''
```

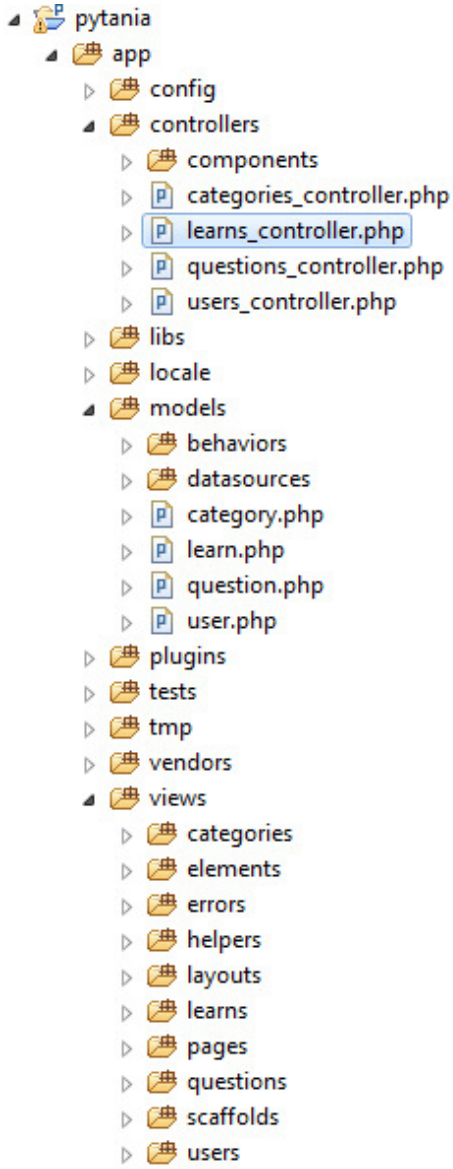
```
        )
    );
```

Tabelka *learns* pozwala na utworzenie relacji wiele do wielu między tabelami *question* a *users.* Użytkownik może posiadać wiele pytań i odwrotnie pytanie może mieć wielu użytkowników. Relacja między klasami *User* a *Question*:

```
var $belongsTo = array(
        'User' => array(
            'className' => 'User',
            'foreignKey' => 'user_id',
            'conditions' => '',
            'fields' => '',
            'order' => ''
        ),
        'Question' => array(
            'className' => 'Question',
            'foreignKey' => 'question_id',
            'conditions' => '',
            'fields' => '',
            'order' => ''
        )
    );
```

4.2.2 Kontrolery

Ilustracja poniżej przedstawia strukturę plików kontrolerów, modeli i widoków.



Ilustracja 2 Kontrolery dla poszczególnych modeli

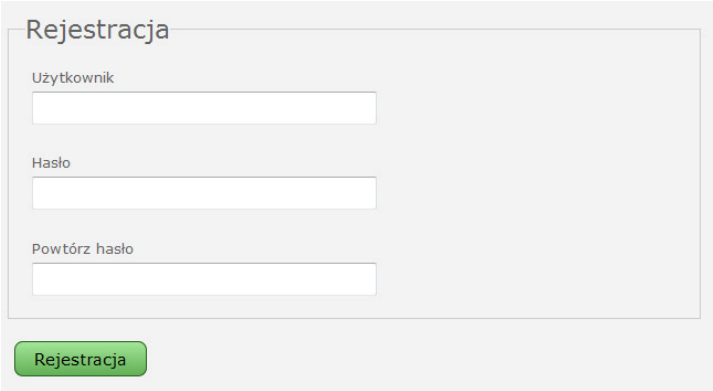
Kontroler jest odpowiedzialny za logikę biznesową, interakcję z użytkownikiem, modelem i widokiem. Odpowiedzialny jest również za przepływ danych z modelu do widoku. Kontroler learns_controller odpowiedzialny jest za przebieg procesu powtarzania. Kontroler questions_controller pozwala na wczytanie pytań z kategorii. Kontroler users_controller umożliwia autoryzacje użytkownika, i wysyłanie emaila.

4.2.3 Interfejs graficzny

Strona główna zawiera ogólne informacje o portalu i aplikacji do powtarzania wiedzy. Nawigacja pozwala na przejście do panelu rejestracji, konta użytkownika i formularza, który pozwala na wysłanie emaila. Ilustracja poniżej przedstawia stronę główną aplikacji.

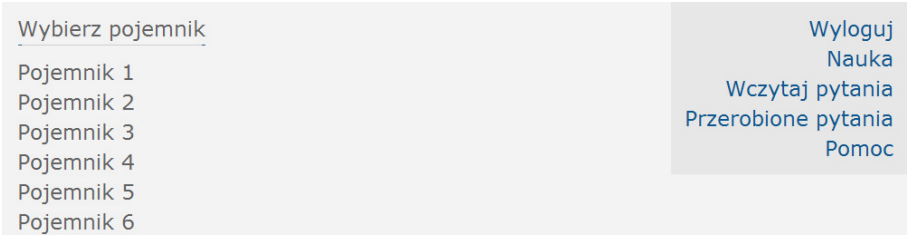
Ilustracja 3 Strona główna

Użytkownik w trakcie rejestracji musi podać login i dwa razy hasło. Ilustracja przedstawia panel rejestracji.



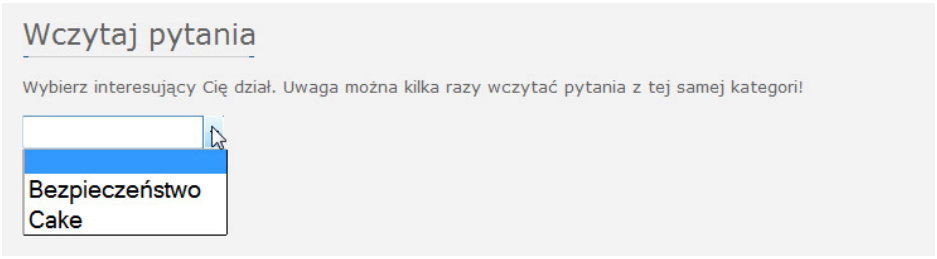
Ilustracja 4 Rejestracja

Ilustracja przedstawia interfejs po zalogowaniu.



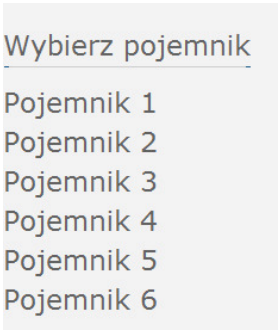
Ilustracja 5 Interfejs po zalogowaniu

Użytkownik po rejestracji i pierwszym zalogowaniu musi wybrać pytania, które go interesują. Ilustracja poniżej przedstawia interfejs „Wczytaj pytanie".



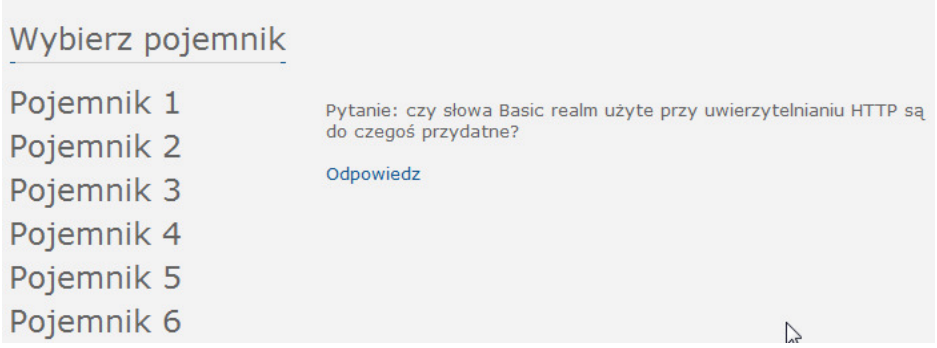
Ilustracja 6 Interfejs „Wczytaj pytania"

Po wczytaniu pytań znajdują się one w pierwszym pojemniku. Ilustracja przedstawia interfejs „Wybierz pojemnik".

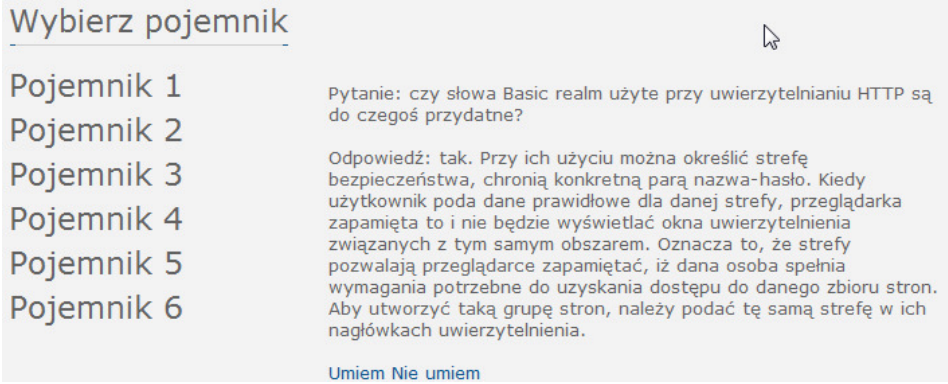


Ilustracja 7 Interfejs „Wybierz pojemnik"

Po wybraniu pojemnika na ekranie monitora wyświetla się pytanie. Zdjęcia poniżej przedstawiają proces wyświetlenia pytania i odpowiedzi.

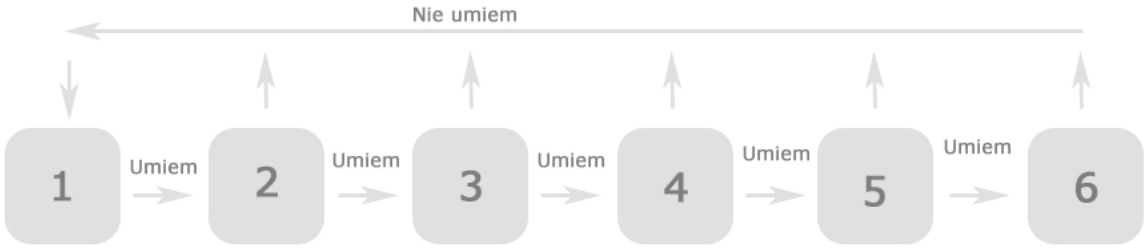


Ilustracja 8 Wyświetlone pytanie



Ilustracja 9 Wyświetlone pytanie i odpowiedź

Po wybraniu opcji „umiem" pytanie przechodzi do następnego pojemnika. Po wybraniu „nie umiem" pytanie umieszczane jest w pierwszym pojemniku. Rysunek poniżej przedstawia przebieg procesu nauki.



Rysunek 2 Proces nauki

W czasie nauki należy robić dłuższe przerwy między powtarzaniem informacji z różnych pojemników.

5. Edytor Protege

Sieć semantyczna została stworzona w dwóch wersjach programu Protege: Protege 3.4.2 i Protege 4. Edytory te są rozwijane w różnych ośrodkach. Protege został stworzony na uniwersytecie w Stanford w Stanach Zjednoczonych. Służy do tworzenia ontologii. Jest darmowym projektem typu Open Source. Program dokładnie został omówiony w dokumentacji [8].

6. Podsumowanie

W pracy magisterskiej zostały omówione zagadnienia ontologii i sieci semantycznej z uwzględnieniem języków do ich tworzenia. Poruszono również zagadnienia związane z architektura MVC i Frameworkiem CakePHP. Pod koniec pracy została przedstawiona sieci semantyczna języka PHP, która powstała na podstawie relacyjnej bazy danych

aplikacji wspomagającej przeprowadzenie powtórek w procesie uczenia się.

Przypisy

1. *A Basic Guide to Ontologies, 08.11.2010 [z:]* http://techwiki.openstructs.org/index.php/A_Basic_Guide_to_Ontologies z dnia 01.05.2012
2. Edward Krzywy - *Semantyczna Sieć: Boty uczą się kojarzyć,* Chip, 15.06.2009 [z:] http://www.chip.pl/artykuly/porady/2009/06/semantyczna-siec-boty-ucza-sie-kojarzyc z dnia 01.05.2012
3. Sebastian Ryszard Kruk - *Semantyczny tort: jak przełknąć wizję Web 3.0*, 25.09.2009 [z] http://www.semanticschool.com/2009/09/semantyczny-tort/ z dnia 01.05.2012
4. Krzysztof Goczyła – *Ontologie w systemach informatycznych,* Warszawa, EXIT, 2011, str. 25
5. Graham Klyne, Jeremy J. Carroll - *Resource Description Framework (RDF): Concepts and Abstract Syntax*, 10.02.2004 [z:] http://www.w3.org/TR/rdf-concepts z dnia 01.05.2012
6. Dan Brickley, R.V. Guha, Brian McBride - *RDF Vocabulary Description Language 1.0: RDF Schema*, 10.2.2004 [z:] http://www.w3.org/TR/rdf-schema z dnia 01.05.2012
7. Deborah L. McGuinness, Frank van Harmelen - *OWL Web Ontology Language Overview*, 10-2-2004 [z:] http://www.w3.org/TR/2004/REC-owl-features-20040210 z dnia 01.05.2012
8. http://protege.stanford.edu/doc/users.html z dnia 01.05.2012

Bibliografia

1. T.R. Gruber - *Toward Principles for the Design of Ontologies Used for Knowledge Sharing,* Padova, Technical Report KSL 93-04, Knowledge Systems Laboratory, Stanford University 23.04.1993 [z:] http://tomgruber.org/writing/onto-design.pdf z dnia 01.05.2012
2. Krzysztof Goczyła – *Ontologie w systemach informatycznych,* Warszawa, EXIT, 2011
3. W. Gliński - *Ontologie. Próba Uporządkowania terminologicznego chaosu,* [z:] http://bbc.uw.edu.pl/Content/20/13.pdf, z dnia 01.05.2012
4. Tim Bray - Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, 16.11.2008 [z:] http://www.w3.org/TR/REC-xml z dnia 01.05.2012
5. Graham Klyne, Jeremy J. Carroll - *Resource Description Framework (RDF): Concepts and Abstract Syntax*, 10.02.2004 [z:] http://www.w3.org/TR/rdf-concepts z dnia 01.05.2012
6. Dan Brickley, R.V. Guha, Brian McBride - *RDF Vocabulary Description Language 1.0: RDF Schema*, 10.2.2004 [z:] http://www.w3.org/TR/rdf-schema z dnia 01.05.2012
7. Deborah L. McGuinness, Frank van Harmelen - *OWL Web Ontology Language Overview*, 10-2-2004 [z:] http://www.w3.org/TR/2004/REC-owl-features-20040210 z dnia 01.05.2012
8. Mike Dean, Guus Schreiber, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein - OWL Web Ontology Language Reference, 10-02-2004 [z:] http://www.w3.org/TR/owl-ref z dnia 01.05.2012
9. http://protege.stanford.edu/doc/users.html z dnia 01.05.2012
10. http://protegewiki.stanford.edu/index.php/Main_Page z dnia 01.05.2012
11. http://www.co-ode.org/resources/reference/manchester_syntax z dnia 01.05.2012
12. Eric Prud'hommeaux, Andy Seaborne, Hewlett-Packard Laboratories, Bristol - SPARQL Query Language for RDF, 15-01-2008 [z:] http://www.w3.org/TR/rdf-sparql-query z dnia 01.05.2012
13. http://api13.cakephp.org/ z dnia 01.05.2012