Języki modelowania ontologii: RDFs, OWL

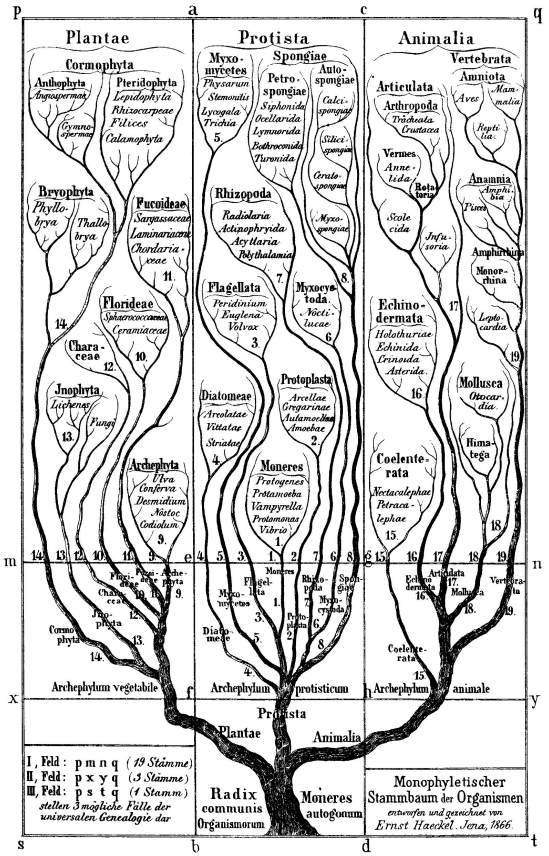
Agnieszka Ławrynowicz
Mikołaj Morzy

Instytut Informatyki
Poznań, rok akademicki 2013/2014

Ontologia z punktu widzenia filozofa

Ontologia (metafizyka)= *nauka o bycie* (Arystoteles, *Metafizyka*,ks. IV)

- ***co to jest byt? co charakteryzuje byt?***
- ***jak dokonywać klasyfikacji bytów?***



Ontologia z punktu widzenia informatyka

"engineering artefact [...]" **(Guarino 98)**

"An ontology is a

formal specification ⇨ maszynowa interpretacja

of a shared ⇨ grupa osób, konsensus

conceptualization ⇨ abstrakcyjny model zjawisk, pojęcia

of a domain of interest" ⇨ wiedza dziedzinowa

(Gruber 93)

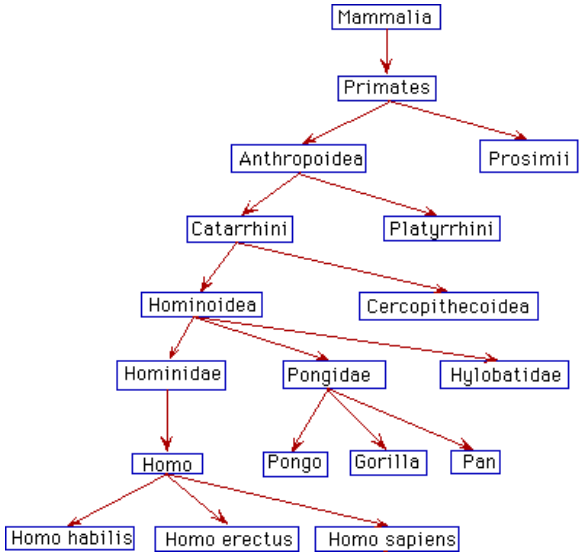
ontologia = formalna specyfikacja pojęć z danej dziedziny

Przykłady ontologii: proste taksonomie

Taksonomia (gr. *taksis* - układ, porządek + *nomos* - prawo)
nauka o zasadach i metodach klasyfikowania

- Klasyfikacja organizmów Linneusza

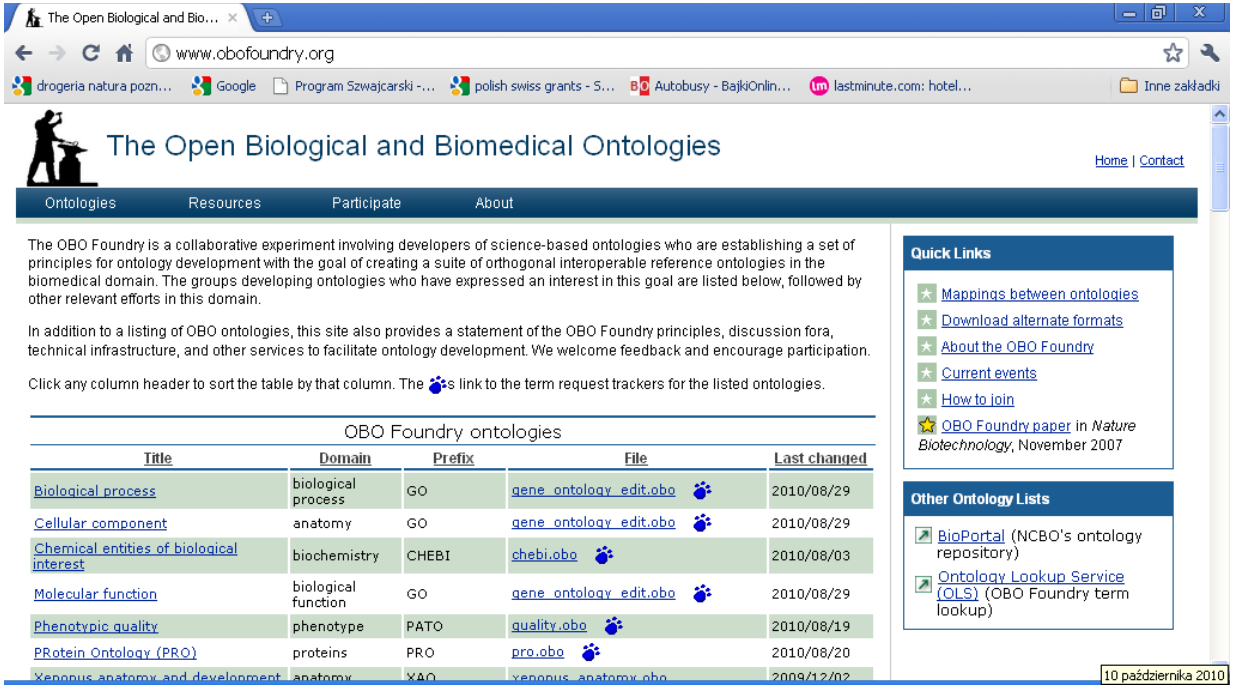
(**Karol Linneusz 1707-1778, „ojciec współczesnej taksonomii”**)



- kategorie wyszukiwarki Yahoo!
  (http://dir.yahoo.com/)
- Open Directory Project 590,000 kategorii
  dmoz.org/)
- katalog produktów Amazon

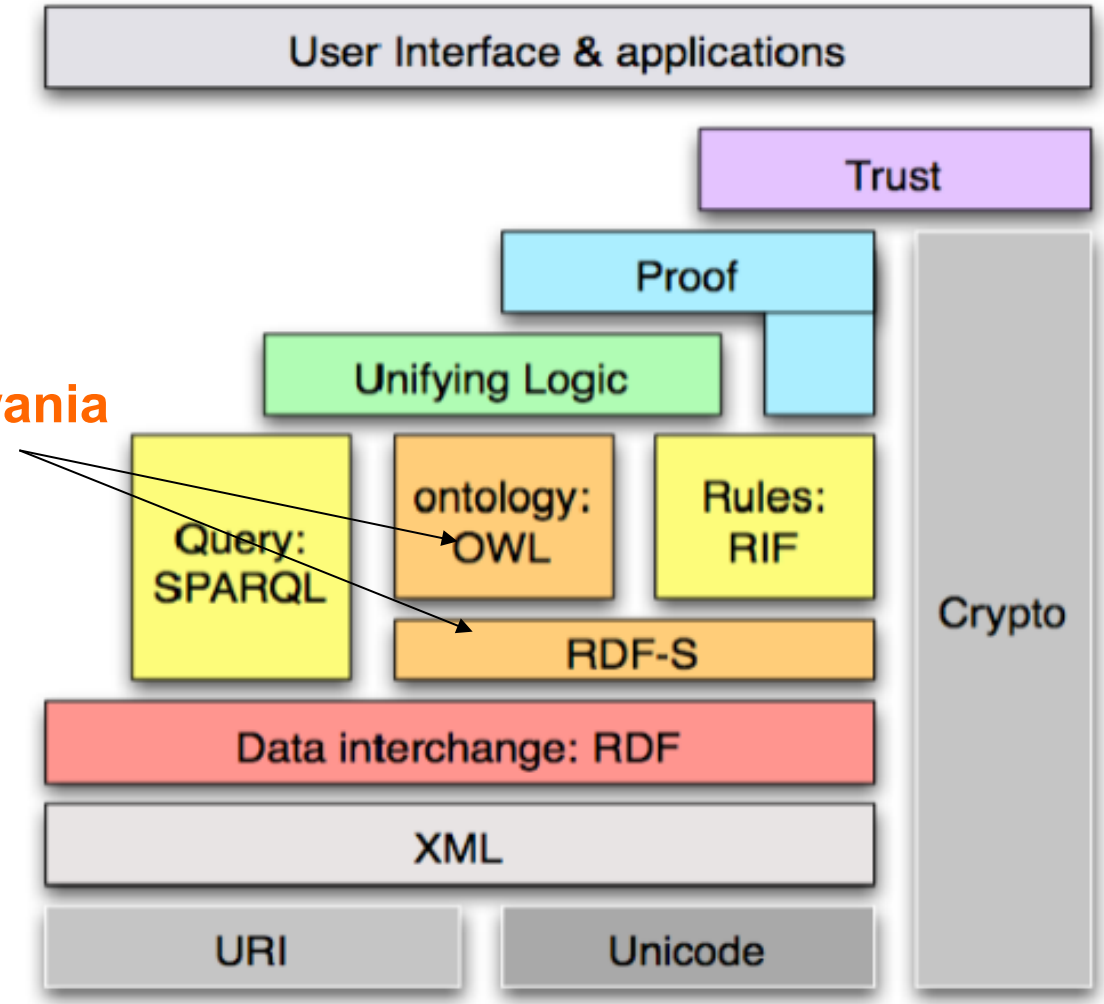
Przykłady ontologii: złożone

- Ontologie wyższe (‘upper ontologies’): Cyc, Sumo, DOLCE, BFO
- Ontologie biomedyczne: Obo Foundry (Open Biological and Biomedical Ontologies),w tym GO (Gene Ontology); Snomed CT, NCI, Galen



Stos języków Sieci Semantycznej

Języki modelowania ontologii



RDFS – RDF Schema

- **RDF definiuje tylko model danych**
- **potrzeba definicji słowników dla modelu danych - języka modelowania ontologii**

RDF Schema pozwala na definiowanie słowników pojęć wraz z relacjami między pojęciami

- pomaga wyrazić jak dane pojęcie powinno być interpretowane

RFDS – kluczowe klasy

rdfs:Resource – zasoby

rdfs:Class – klasy

rdfs:Literal – typy proste

odziedziczone z RDF:

rdf:type – określa klasę zasobu (której zasób jest instancją)

RDFS – kluczowe własności

- **rdfs:subClassOf** – określa nadklasę danej klasy

  wszystkie instancje klasy są także instancjami jej nadklasy
- **rdfs:subPropertyOf** – wiąże własność z jedną z jej podwłasności

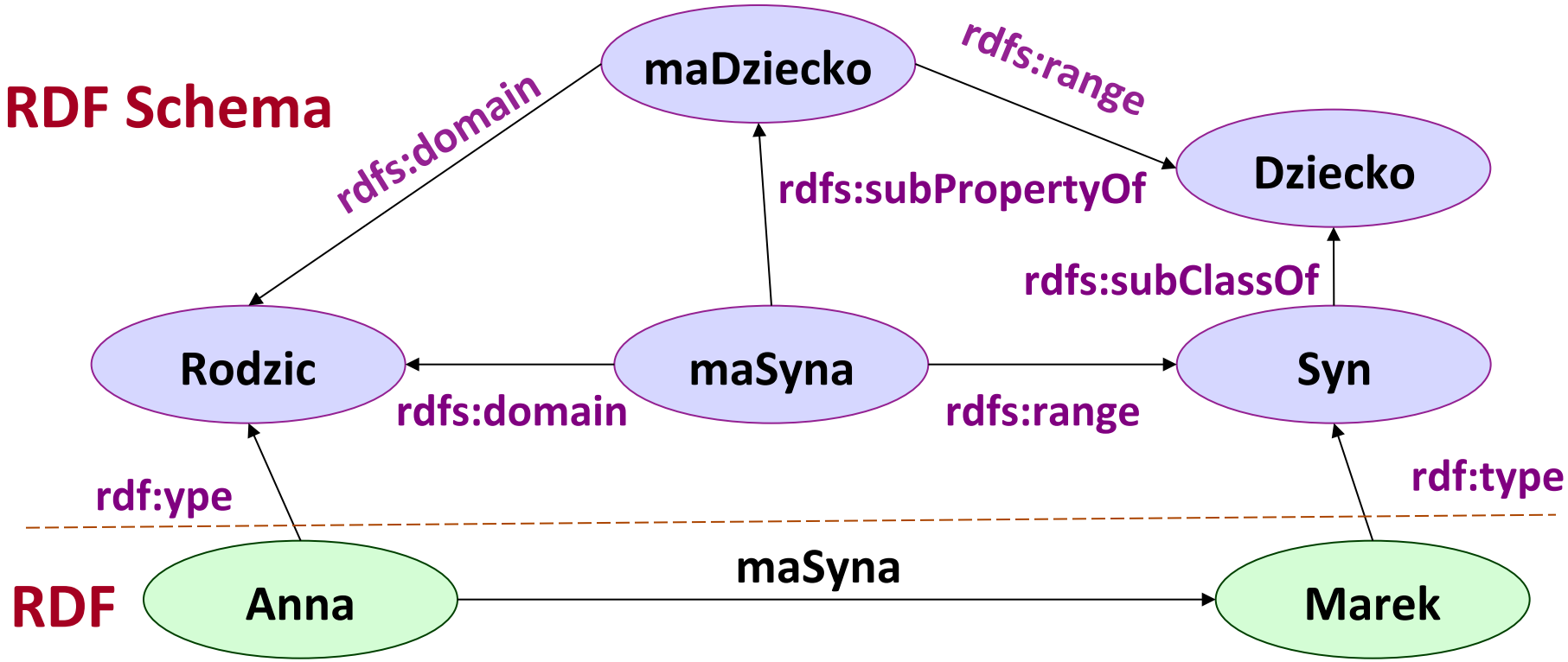
```
<rdfs:Class rdf:about="#Syn">
    <rdfs:subClassOf rdf:resource="#Dziecko"/>
</rdfs:Class>

<rdf:Property rdf:about="#maSyna">
    <rdfs:subPropertyOf rdf:resource="#maDziecko"/>
</rdf:Property>
```


RDFS – kluczowe własności c.d.

- **rdfs:domain** – specyfikuje dziedzinę własności P:
  - czyli klasę tych zasobów, które mogą się pojawiać jako podmiot (subject) w trójce z tym predykatem
  - Jeśli nie podano dziedziny, to w zdaniu może wystąpić dowolny zasób
- **rdfs:range** – określa zakres własności P:
  - czyli klasę tych zasobów, które mogą się pojawiać jako obiekt (object) w trójce z tym predykatem

```
<rdf:Property rdf:ID="maDziecko">
   <rdfs:domain rdf:resource="#Rodzic"/>
   <rdfs:range rdf:resource="#Dziecko"/>
</rdf:Property>
```


RDF osadzony w RDFS – przykład



Ograniczenia RDFS

Brak lokalnych ograniczeń na zakres i dziedzinę

Nie można wyrazić, że wartość w dziedzinie własności **maDziecko** należy do pojęcia **Człowiek**, gdy stosujemy ten predykat w stosunku do ludzi i że należy do pojęcia **Kot**, jeżeli stosujemy go dla kotów

Brak ograniczeń ∀, ∃ i ilościowych

Nie można wyrazić, że wszystkie instancje pojęcia **Człowiek** mają matkę i że **Matka** jest także człowiekiem lub że ludzie mają dokładnie dwóch rodziców

Brak przechodnich, odwrotnych lub symetrycznych relacji

Nie można wyrazić że **jestCzęścią** jest własnością przechodnią, i że **maCzęść** jest relacją odwrotną do **jestCzęścią**

OWL – Web Ontology Language

- **dostarcza bogatej kolekcji operatorów do konstrukcji złożonych pojęć**
- **semantyka języka korzysta z badań w ramach sztucznej inteligencji w zakresie reprezentacji wiedzy – logiki deskrypcyjne**

Skąd pochodzi akronim „OWL”?

Web Ontology Language ≠ WOL

Owl lived at The Chestnuts, an old-world residence of great charm, which was grander than anybody else’s, or seemed so to Bear, because it had both a knocker and a bell-pull. Underneath the knocker there was a notice which said:
PLES RING IF AN RNSER IS REQIRD.
Underneath the bell-pull there was a notice which said:
PLEZ CNOKE IF AN RNSR IS NOT REQID.
These notices had been written by Christopher Robin, who was the only one in the forest who could spell; for Owl, wise though he was in many ways, able to read and write and spell his own name **WOL**, yet somehow went all to pieces over delicate words like MEASLES and BUTTEREDTOAST.

(A.A. Milne, „Kubuś Puchatek”)

Warianty OWL 1.1

- **OWL full**
  - pełne słownictwo OWL
  - wykorzystanie możliwości RDFS (dowolność w opisywaniu klas i własności, np. klasa może być jednocześnie widziana jako zbiór jednostek i jako pojedyncza jednostka)
- **OWL DL**
  - bazuje na formalizmie logik deskrypcyjnych (możliwe wykorzystanie istniejących już mechanizmów i narzędzi do wnioskowania)
  - ograniczenia na używanie słownictwo RDFS (np. rozdzielenie klas, własności, indywiduów)
- **OWL Lite**
  - te same ograniczenia co OWL DL
  - dodatkowo operuje na prostym podzbiorze słownictwa, wystarczającym do modelowania prostych klasyfikacji

Profile OWL 2

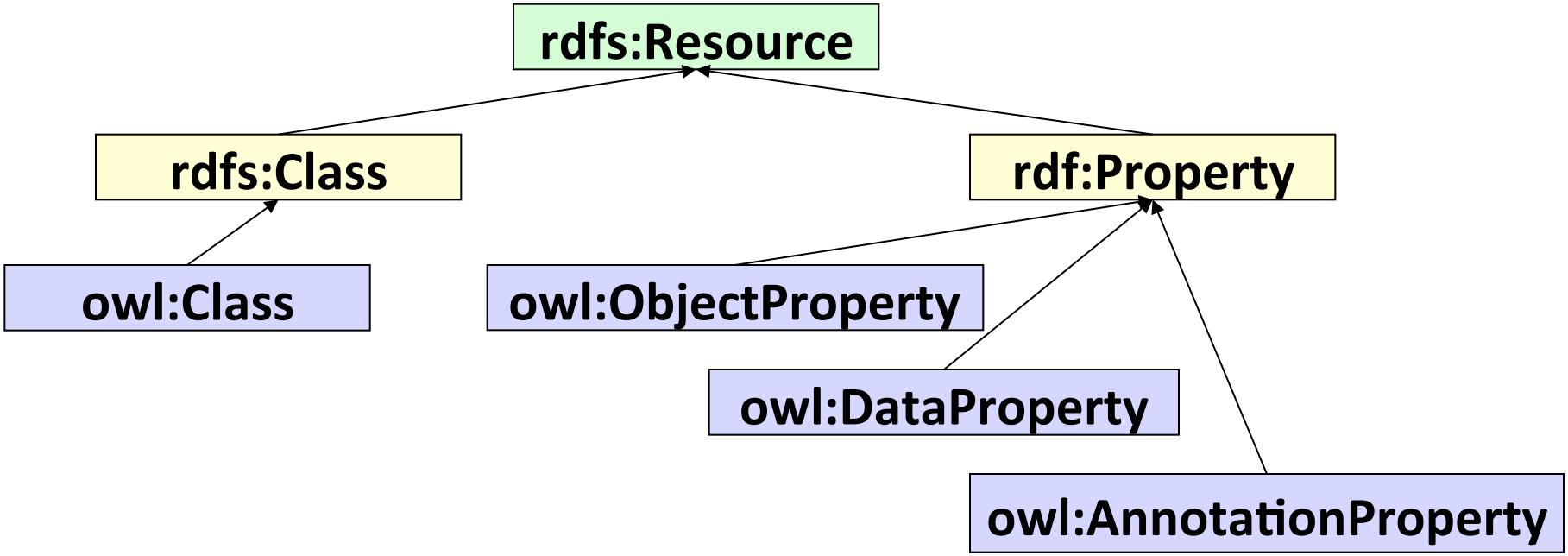
- **OWL EL:**
  - standardowe wnioskowanie w czasie wielomianowym, odpowiedni dla aplikacji korzystających z dużych ontologii (ontologie medyczne)
- **OWL QL:**
  - odpowiedni dla małych ontologii wykorzystujących dużą liczbę danych oraz w przypadkach, gdy dostęp do danych jest wymagany w postaci zapytań relacyjnych (np. SQL)
- **OWL RL:**
  - wykorzystuje algorytmy bazujące na technologiach baz danych rozszerzonych o reguły w czasie wielomianowym i manipulujące bezpośrednio na trójkach RDF; („forward chaining rules”)

Składnia

- RDF/XML (główna składnia dla OWL)
- abstrakcyjna składnia (wykorzystywana w dokumencie ze specyfikacją języka)
- składnia Protege-OWL (Manchester OWL Syntax)

Kompatybilność OWL z RDF Schema

- Wszystkie warianty OWL używają składni RDF
- Indywidua deklarowane tak jak w RDF – rdf:Description
- Konstruktory OWL – uszczegółowienie ich odpowiedników w RDF



OWL a logiki deskrypcyjne

Constructor	$\mathcal{DL}$ syntax
Thing	$\top$
Nothing	$\bot$
complementOf	$(\neg C)$
intersectionOf	$(C \sqcap D)$
unionOf	$(C \sqcup D)$
allValuesFrom	$(\forall R.C)$
someValuesFrom	$(\exists R.C)$
minCardinality	$\geq \mathrm{n} R$
maxCardinality	$\leq \mathrm{n} R$
oneOf	$\{a_1, ..., a_n\}$
Axiom	**$\mathcal{DL}$ syntax**
equivalentClass	$C \equiv D$
subClassOf	$C \sqsubseteq D$
equivalentProperty	$S \equiv R$
subPropertyOf	$S \sqsubseteq R$
inverseOf	$S \equiv R^-$
transitiveProperty	$R^+ \sqsubseteq R$
functionalProperty	$\top \sqsubseteq \leq 1R$
sameIndividualAs	$a = b$
differentFrom	$a \neq b$

owl:Ontology

```
<owl:Ontology rdf:about="">
<rdfs:comment>Przykladowa ontologia
</rdfs:comment>
<owl:priorVersion
    rdf:resource="http://przyklad.org/"/>
<owl:imports
   rdf:resource="http://przyklad.org/osoby"/>
<rdfs:label>Ontologia PP</rdfs:label>
</owl:Ontology>
```


OWL – elementy składowe

Encje – są to klasy, własności, indywidua i wszelkie inne elementy modelowanej domeny

Wyrażenia – złożone pojęcia na temat modelowanej domeny

Aksjomaty – twierdzenia, które są prawdziwe w modelowanej domenie

Klasy

- Zbiory instancji, definiowane za pomocą **owl:Class** (podklasa rdfs:Class)

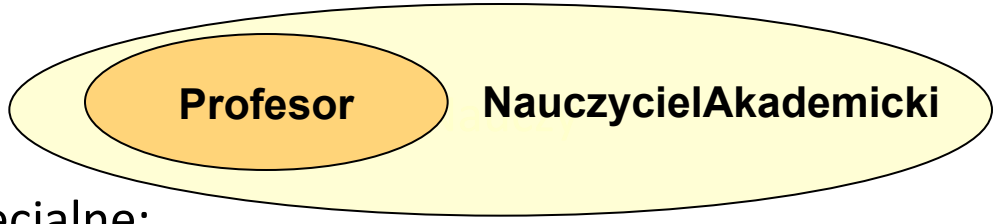
Abstrakcyjna składnia:

SubClassOf(*Profesor NauczycielAkademicki*)

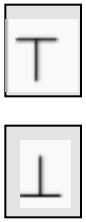
Serializacja:

```
<owl:Class rdf:ID="Profesor">
   <rdfs:subClassOf rdf:resource="#NauczycielAkademicki"/>
</owl:Class>
```

Przykład



- Klasy specjalne:
  - **owl:Thing** (klasa uniwersalna)
  - **owl:Nothing** (klasa pusta, „najniższa”)



Własności

- **własności obiektowe (ang. object properties)**
  - łączą obiekty z innymi obiektami
- **własności literałowe (ang. data properties)**
  - łączą obiekty z literałami (typy danych literałów np. z puli XML Schema)
- **własności adnotacyjne (ang. annotation properties)**
  - łączą obiekty z notatkami na ich temat (*rdfs:label, owl:versionInfo,…)*

```
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</owl:DatatypeProperty>
```

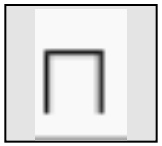

Klasy – wybrane wyrażenia

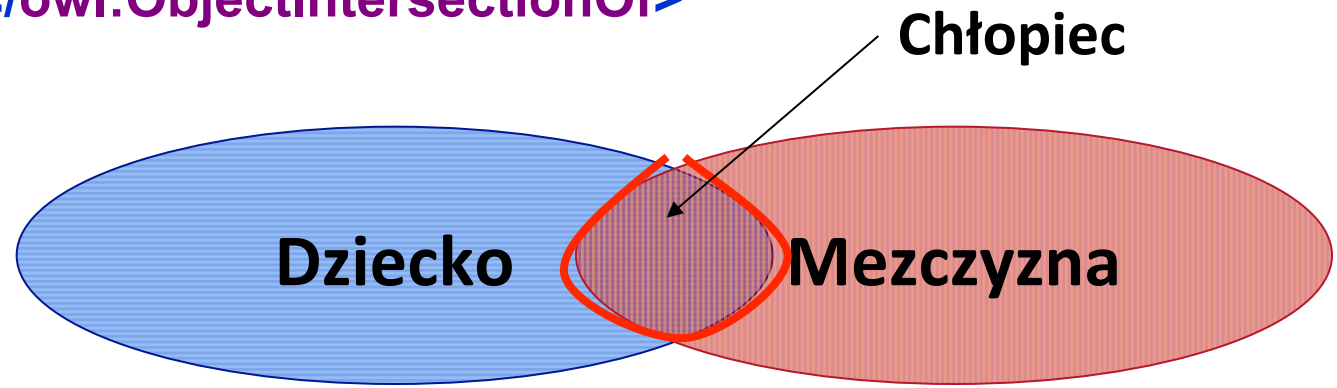
- iloczyn
- suma
- negacja
- kwantyfikator egzystencjalny
- kwantyfikator ogólny
- ograniczenie wartości indywiduum

Iloczyn

ObjectIntersectionOf(*Dziecko Mezczyzna*)

```
<owl:ObjectIntersectionOf>
    <owl:Class rdf:resource="Dziecko">
    <owl:Class rdf:resource="Mezczyzna">
</owl:ObjectIntersectionOf>
```

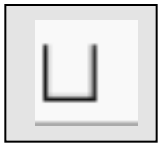


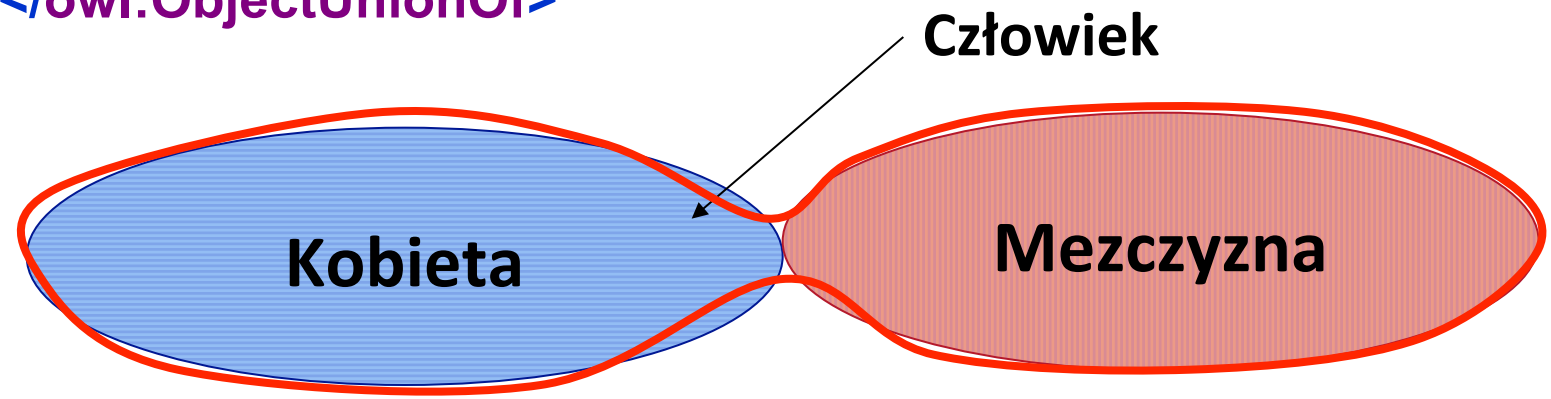


Suma

ObjectUnionOf(*Kobieta Mezczyzna*)

```
<owl:ObjectUnionOf>
    <owl:Class rdf:resource="Kobieta">
    <owl:Class rdf:resource="Mezczyzna">
</owl:ObjectUnionOf>
```



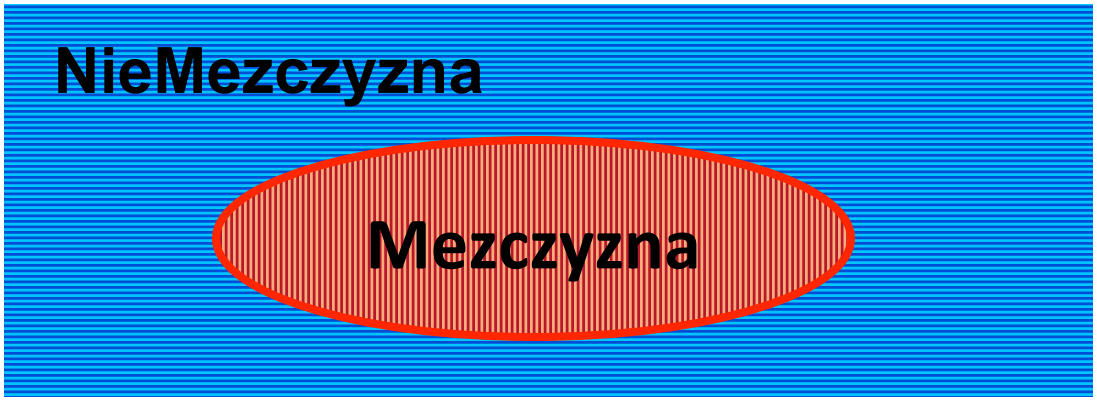


Negacja (dopełnienie)

ObjectComplementOf(*Mezczyzna*)

```
<owl:ObjectComplementOf>
      <owl:Class rdf:resource="Mezczyzna">
</owl:ObjectComplementOf>
```

¬



UWAGA na założenie otwartego świata! (omówione dalej)

Kwantyfikator egzystencjalny

“każdy profesor musi wykładać przynajmniej jeden przedmiot”

ObjectSomeValuesFrom(*wyklada Przedmiot*)


```
<owl:Class rdf:about=”#Profesor”>
    <owl:subClassOf>
     <owl:Restriction>
          <owl:onProperty rdf:resource=”#wyklada”/>
          <owl:ObjectSomeValuesFrom rdf:resource=”#Przedmiot”/>
     </owl:Restriction>
    </owl:subClassOf>
</owl:Class>
```


Kwantyfikator ogólny

“asystenci prowadzą tylko laboratoria”

ObjectAllValuesFrom(*prowadzi Laboratorium*)


```
<owl:Class rdf:about=”#Asystent”>
    <owl:subClassOf>
     <owl:Restriction>
          <owl:onProperty rdf:resource=”#prowadzi”/>
          <owl:ObjectAllValuesFrom rdf:resource=”#Laboratorium”/>
     </owl:Restriction>
    </owl:subClassOf>
</owl:Class>
```


Kwantyfikator ogólny c.d

Uwaga! Może to dotyczyć także asystentów, którzy nie prowadzą żadnych zajęć! Wynika to ze znaczenia kwantyfikatora ogólnego w logice pierwszego rzędu.

„wszystkie moje worki z pieniędzmi leżą na tym stole”

Ograniczenie wartości indywiduum

“banany mają kolor żółty”

ObjectHasValue(*maKolor żółty)*

```
<owl:Class rdf:ID=„Banan">
    <rdfs:subClassOf>
     <owl:Restriction>
          <owl:onProperty rdf:resource="#maKolor" />
          <owl:ObjectHasValue rdf:resource="#żółty" />
     </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```


Klasy – wybrane aksjomaty

-zawieranie się (subsumcja)

-równoważność

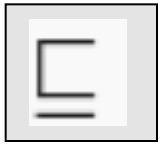
-rozłączność

Klasy: elementarne a zdefiniowane

opisy (deskrypcje)

elementarne pojęcia

SubClassOf(*Nazwa …*)



SubClassOf(*Rodzic*
ObjectSomeValuesFrom(*maDziecko*
ObjectUnionOf(*Chlopiec*
Dziewczynka))
)

Wszyscy rodzice posiadają między innymi dziecko będące chłopcem lub dziewczynką.

definicje

zdefiniowane pojęcia

EquivalentClasses(*Nazwa …*)

EquivalentClasses(Chlopiec
ObjectIntersectionOf(*Dziecko*
Mezczyzna)
)

Każdy kto między innymi jest dzieckiem i jednocześnie mężczyzną jest chłopcem.

Warun zające

„Wygląda jak ka

więc musi to by

Rozłączność

Dopóki nie są wprowadzone jawnie ograniczenia rozłącznościowe, klasy mogą mieć część wspólną

DisjointClasses(K*lasa_1, …, Klasa_n*)

DisjointClasses(*Chlopiec Dziewczynka*)

Własności obiektowe – wybrane aksjomaty

-własność odwrotna

-własność funkcyjna

-własność przechodnia

-łańcuchy własności obiektowych (OWL 2)

Własność odwrotna

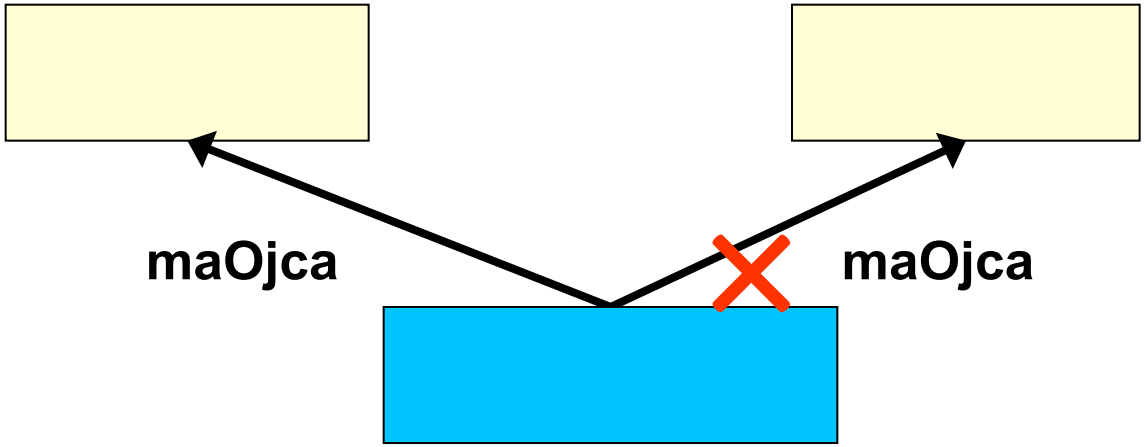
InverseObjectProperties(*maDziecko jestDzieckiem*)

```
<owl:ObjectProperty rdf:ID="maDziecko">
    <owl:inverseOf rdf:resource="#jestDzieckiem"/>
</owl:ObjectProperty>
```

Własność funkcyjna

FunctionalObjectProperty(*maOjca*)

<owl:FunctionalObjectProperty rdf:ID="maOjca"/>

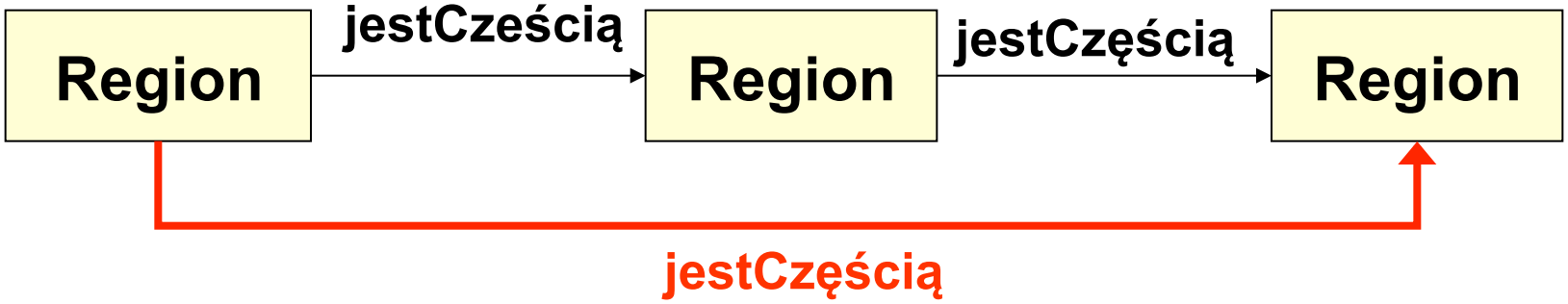


UWAGA na brak założenia o unikalności nazw! (omówione dalej)

Własność przechodnia

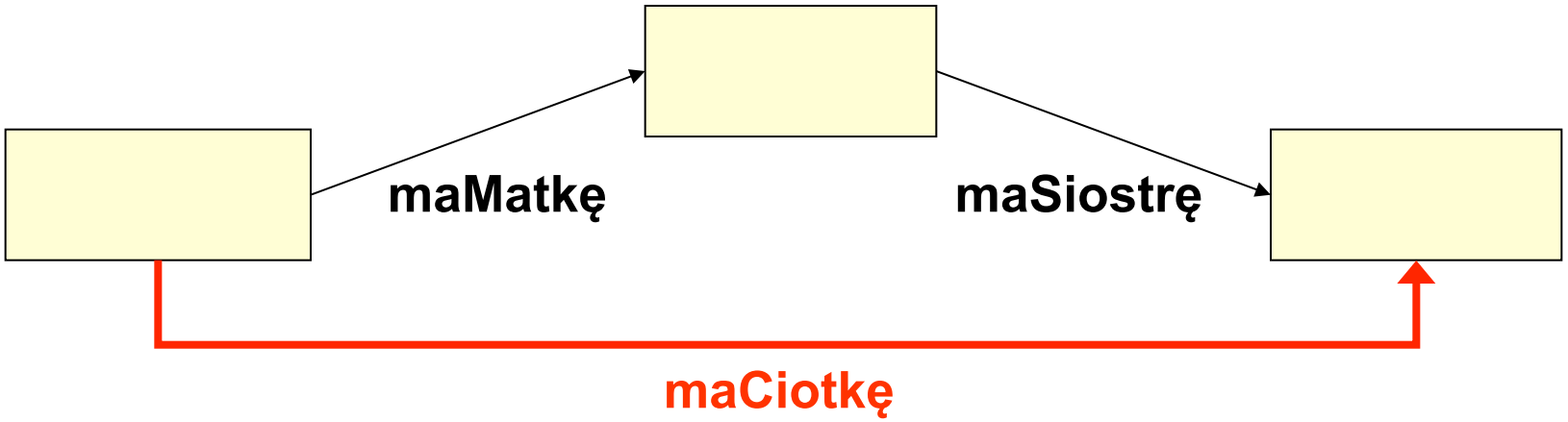
TransitiveObjectProperty(*jestCzęścią*)

```
<owl:TransitiveObjectProperty rdf:ID=„jestCzęścią">
    <rdfs:domain rdf:resource="#Region"/>
    <rdfs:range rdf:resource="#Region"/>
</owl:TransitiveObjectProperty>
```



Łańcuchy własności obiektowych (OWL 2)

SubObjectPropertyOf(ObjectPropertyChain(*maMatkę* *maSiostrę*) *maCiotkę*)



Indywidua

- **Asercje indywiduów do klas:**

  ClassAssertion(*Pies Azor*)

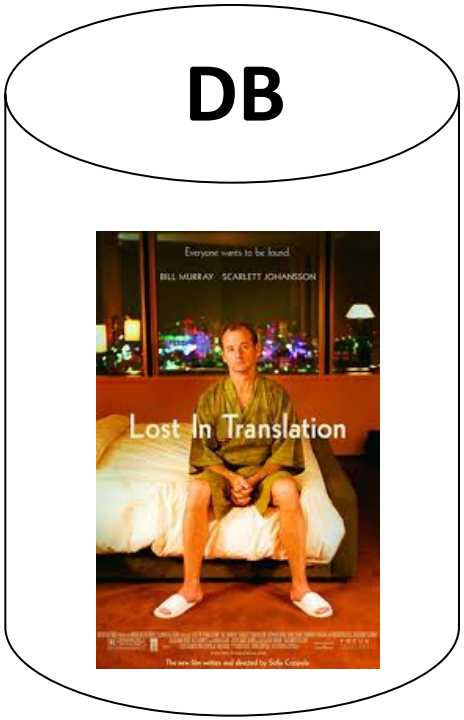
- **Asercje indywiduów do własności:**

  ObjectPropertyAssertion(*maPsa Adam Azor*)

”Świat zamknięty” kontra ”świat otwarty”

- **Zamknięty świat** (programowanie w logice, bazy danych)
  - ***kompletna* wiedza o indywiduach**
  - **brak informacji jest informacją negatywną (*negation-as-failure*)**
- **Otwarty świat** (logika deskrypcyjna, Sieć Semantyczna)
  - ***niekompletna* wiedza o indywiduach**
  - **negacja faktu musi być jawnie podana (*monotonic negation*)**

”Świat zamknięty” kontra ”świat otwarty” – przykład 1



Załóżmy, że w bazie mamy następujące dane:

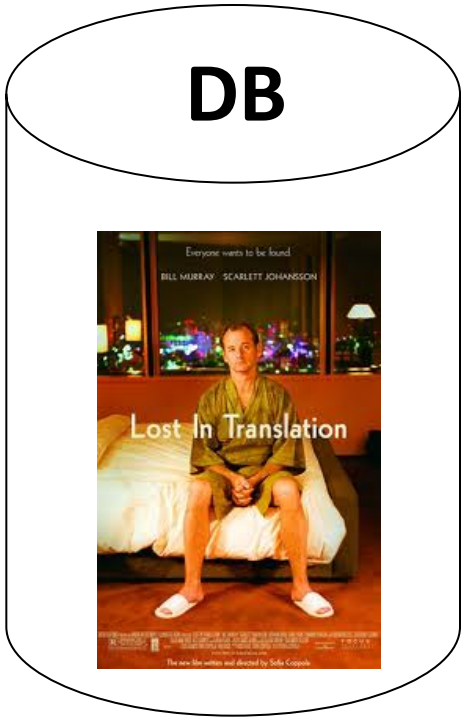
ClassAssertion (LostInTranslation OscarMovie)

ClassAssertion (Director SofiaCoppola)

ObjectPropertyAssertion (creates SofiaCoppola LostInTranslation)

Czy „**wszystkie filmy Sofii Coppoli są filmami oskarowymi?**”

”Świat zamknięty” kontra ”świat otwarty” – przykład 1



Załóżmy, że w bazie mamy następujące dane:

ClassAssertion (LostInTranslation OscarMovie)

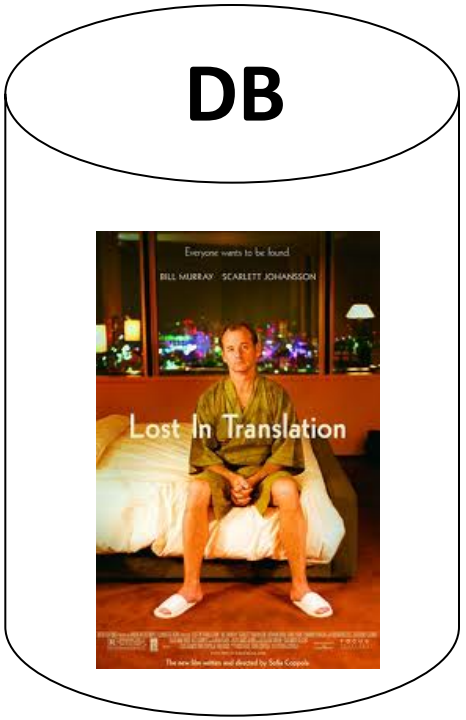
ClassAssertion (Director SofiaCoppola)

ObjectPropertyAssertion (creates SofiaCoppola LostInTranslation)

Czy „**wszystkie filmy Sofii Coppoli są filmami oskarowymi?**”

TAK – zamknięty świat

”Świat zamknięty” kontra ”świat otwarty” – przykład 1



Załóżmy, że w bazie mamy następujące dane:

ClassAssertion (LostInTranslation OscarMovie)

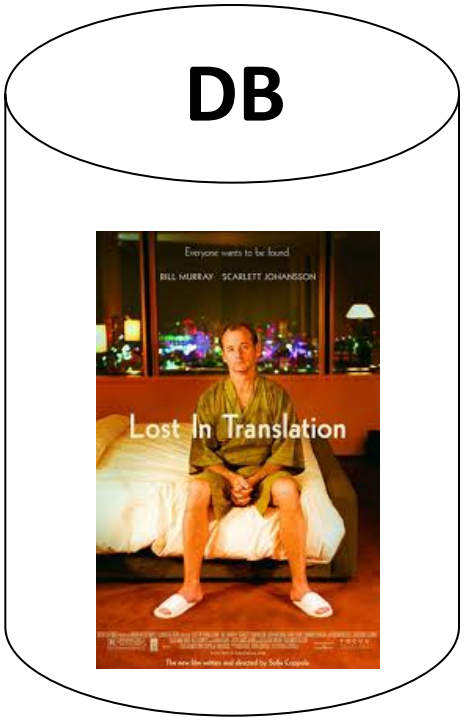
ClassAssertion (Director SofiaCoppola)

ObjectPropertyAssertion (creates SofiaCoppola LostInTranslation)

Czy „**wszystkie filmy Sofii Coppoli są filmami oskarowymi?**”

TAK – zamknięty świat **NIE WIEM – otwarty świat**

”Świat zamknięty” kontra ”świat otwarty” – przykład 1

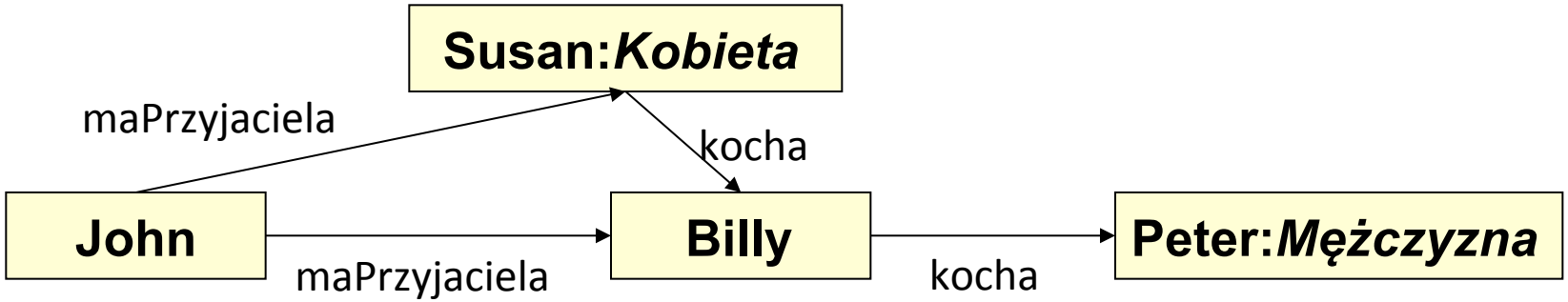


Czy „**wszystkie filmy Sofii Coppoli są filmami oskarowymi?**”

TAK – zamknięty świat

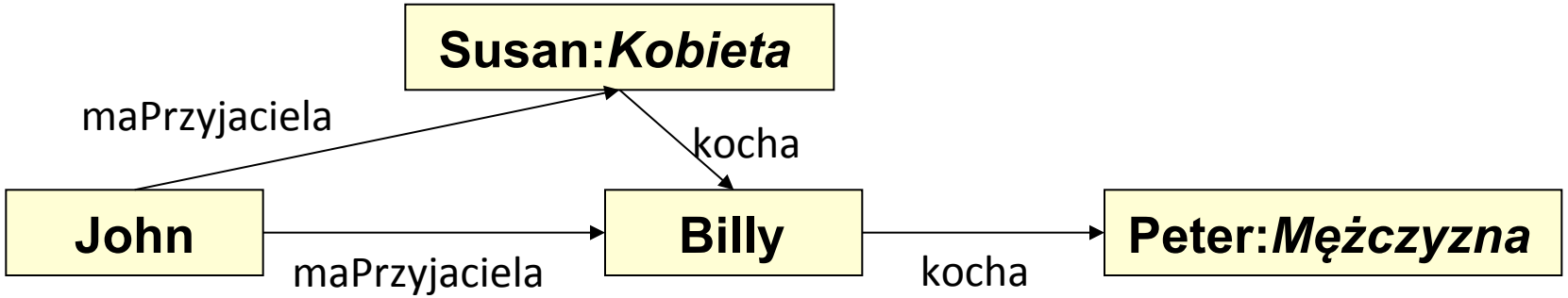
NIE WIEM – otwarty świat

”Świat zamknięty” kontra ”świat otwarty” – przykład 2



Czy John ma przyjaciółkę, która kocha mężczyznę?

”Świat zamknięty” kontra ”świat otwarty” – przykład 2



Kobieta

imię
Susan

Mężczyzna

imię
Peter

Przyjaciele

kto	kogo
John	**Susan**
John	**Billy**

Kochankowie

kto	kogo
Susan	**Billy**
Billy	**Peter**

”Świat zamknięty” kontra ”świat otwarty” – przykład 2

Kobiety

imię
Susan

Mężczyźni

imię
Peter

Przyjaciele

kto	kogo
John	**Susan**
John	**Billy**

Kochankowie

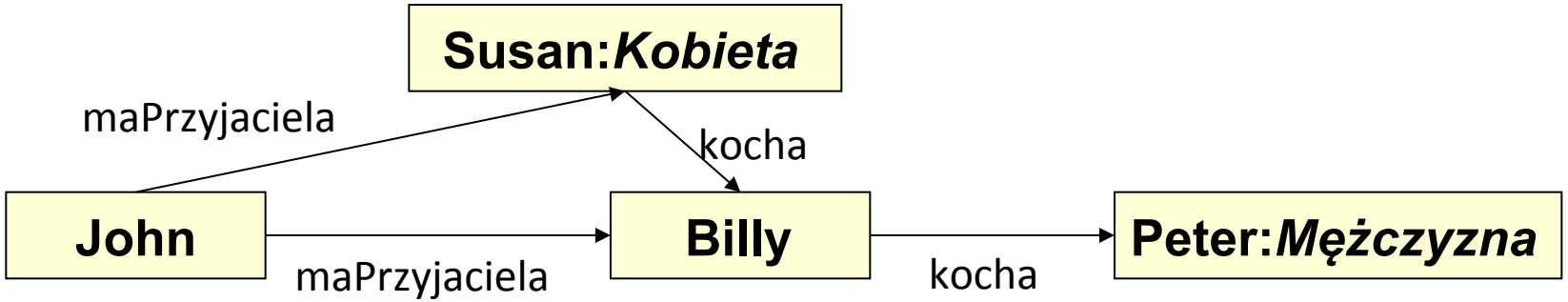
kto	kogo
Susan	**Billy**
Billy	**Peter**

SQL

```
SELECT p.kogo
FROM Kobiety k, Mężczyźni m, Przyjaciele p, Kochankowie c
WHERE p.kogo = k.imię AND k.imię = c.kto AND c.kogo = m.imię
AND p.kto = „John”
```

Odp. NIE (pusty wynik)

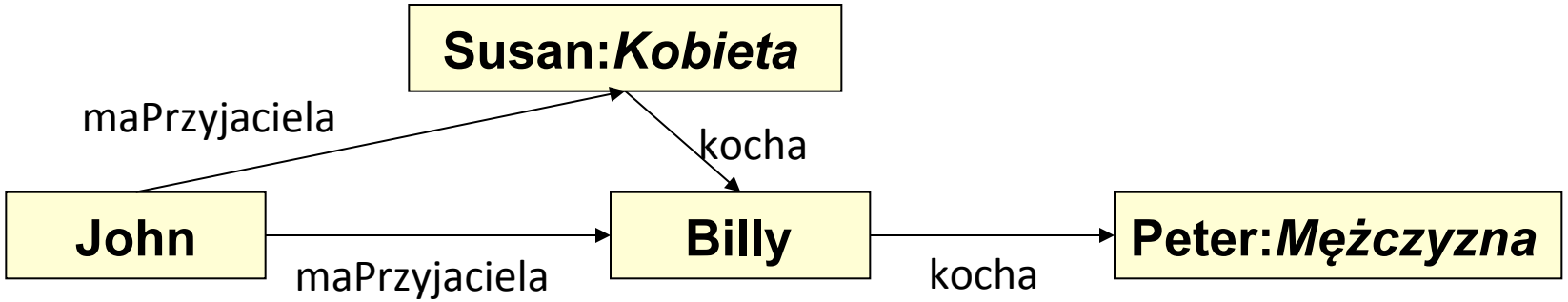
”Świat zamknięty” kontra ”świat otwarty” – przykład 2



Czy John ma przyjaciółkę, która kocha mężczyznę?

Czyli: czy istnieje taka kobieta, która kocha mężczyznę i jest przyjaciółką Johna?

”Świat zamknięty” kontra ”świat otwarty” – przykład 2



Czy John ma przyjaciółkę, która kocha mężczyznę?

Czyli: czy istnieje taka kobieta, która kocha mężczyznę i jest przyjaciółką Johna?

Wnioskowanie

Billy jest albo mężczyzną, albo kobietą.

Jeśli Billy jest mężczyzną: **TAK!** (ta przyjaciółka to Susan)

Jeśli Billy jest kobietą: **TAK!** (ta przyjaciółka to Billy)

Brak założenia o unikalności nazw

JFK

John F. Kennedy

John Fitzgerald Kennedy

wszystkie (różne) nazwy mogą oznaczać ten sam obiekt!

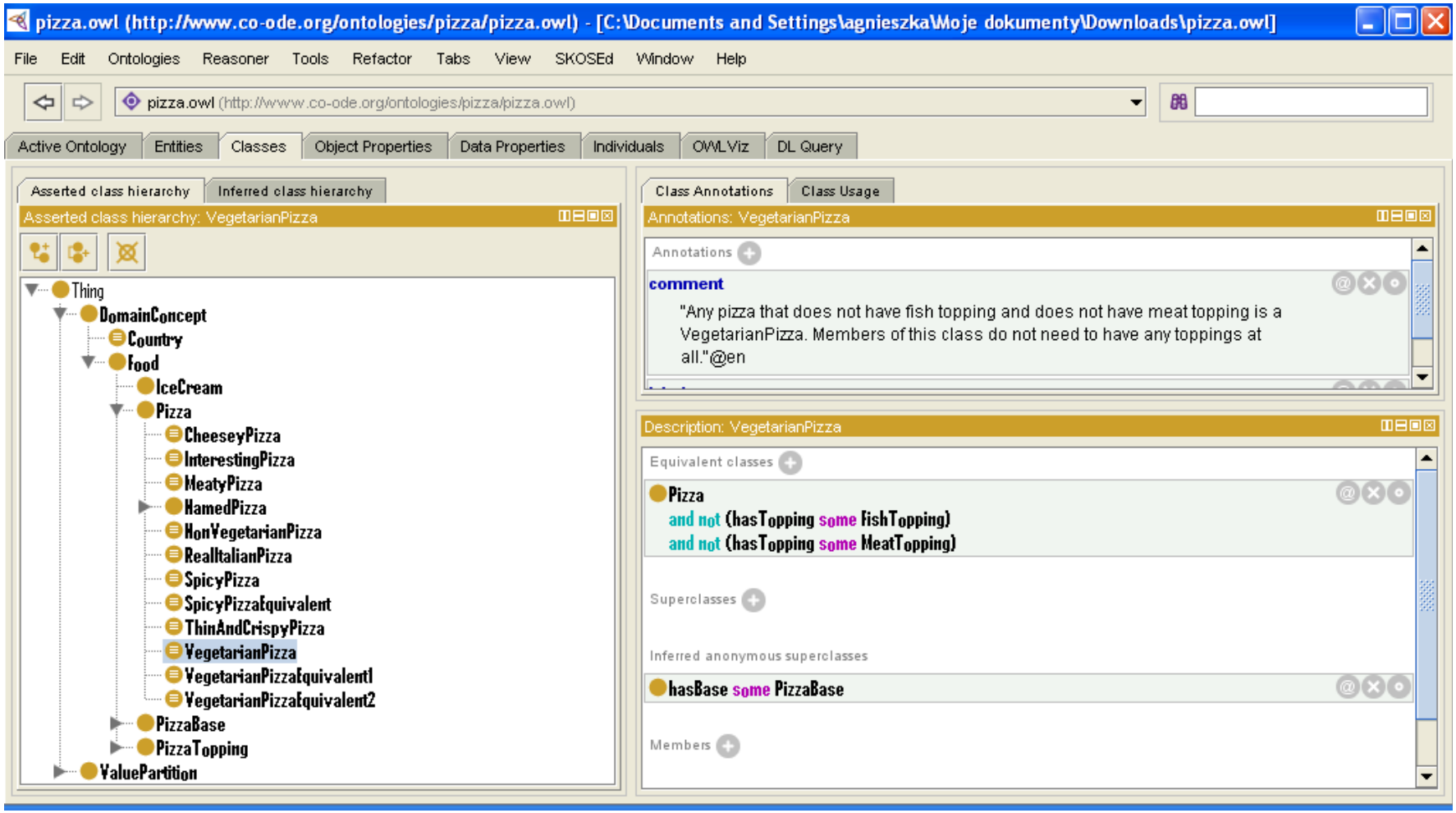
Aksjomaty o jawnej tożsamości lub rozłączności indywiduów:

- SameIndividual(*JFK John_F_Kennedy*)
- DifferentIndividuals(*JFK George_Bush Barack_Obama*)

Narzędzia

- Edytory
  - **Protégé, NeOn Toolkit, TopQuadrant Composer, Altova SemanticWorks…**
- Silniki wnioskujące
  - **Pellet, Racer, FaCT++, CEL, HermiT…**
- API, zestawy narzędzi
  - **OWL-API, Jena, KAON2, Protégé, OWLIM,…**

Protégé


pizza.owl (http://www.co-ode.org/ontologies/pizza/pizza.owl) - [C:\Documents and Settings\agnieszka\Moje dokumenty\Downloads\pizza.owl]
File Edit Ontologies Reasoner Tools Refactor Tabs View SKOSEd Window Help
pizza.owl (http://www.co-ode.org/ontologies/pizza/pizza.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query
Asserted class hierarchy Inferred class hierarchy
Asserted class hierarchy: VegetarianPizza
Thing
DomainConcept
Country
Food
IceCream
Pizza
CheeseyPizza
InterestingPizza
MeatyPizza
NamedPizza
NonVegetarianPizza
RealItalianPizza
SpicyPizza
SpicyPizzaEquivalent
ThinAndCrispyPizza
VegetarianPizza
VegetarianPizzaEquivalent1
VegetarianPizzaEquivalent2
PizzaBase
PizzaTopping
ValuePartition
Class Annotations Class Usage
Annotations: VegetarianPizza
Annotations
comment
"Any pizza that does not have fish topping and does not have meat topping is a VegetarianPizza. Members of this class do not need to have any toppings at all."@en
Description: VegetarianPizza
Equivalent classes
Pizza
and not (hasTopping some FishTopping)
and not (hasTopping some MeatTopping)
Superclasses
Inferred anonymous superclasses
hasBase some PizzaBase
Members

Lepsza ontologia…