

# Semantic Web

## RDF+OWL+Protege

### Introducere

Apariția webului în 1989 a constituit un mare pas înainte în felul în care se poate interacționa cu calculatorul. Webul a permis tuturor celor care nu aveau cunoștințe tehnice deosebite să acceseze informații într-un mod deosebit de simplu. Treptat, acesta a evoluat și de curând a apărut ceea ce astăzi numim Web 2.0 – webul în care primează colaborarea între utilizatori și crearea de conținut. Web 2.0 este caracterizat de aplicații și servicii colaborative – comunități, situri de socializare (social networking), sisteme de recomandare, wiki și folksonomii.

Pe de altă parte, webul semantic este, după Tim Berners-Lee, creatorul webului 1.0 - "o extensie a webului existent, în care informația conține un sens bine definit, permițând oamenilor și calculatoarelor să lucreze împreună" (Lee 2001). Ideea este ca webul să conțină date și nu documente, transformându-se astfel într-o bază de cunoștințe imensă, a cărei exploatare va fi făcută atât de oameni cât și de calculatoare. Web 2.0 poate fi văzut și ca un prim pas către webul semantic (deși există opinii că sunt incompatibile).

În acest capitol vom discuta îmbunătățirile fundamentale pe care le aduce webul semantic și principalele mecanisme pe care el le propune. Vom arăta care este structura lui, limbajele caracteristice și aplicațiile care pot fi folosite pentru a dezvolta instrumente semantice. Vom continua prin a arăta cum pot fi extinse serviciile web pentru a fi incluse în acest web semantic și vom arăta câteva aplicații tipice pentru webul semantic și foarte semnificative din punctul de vedere al interfeței om-calculator. Figura 1 prezintă structura webului semantic conform cu (Koivunen, 2001).

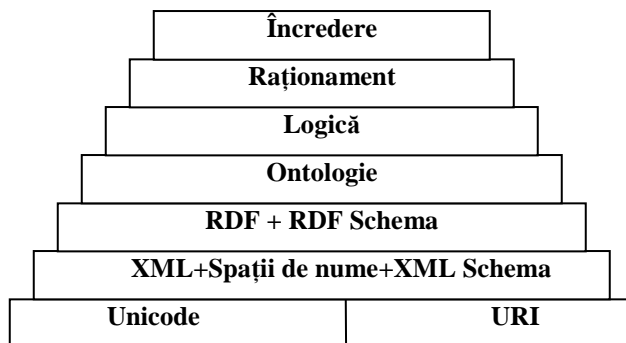


Figura 1 Structura webului semantic

# Ontologii, limbaje și instrumente pentru crearea ontologiilor

## Ontologii și folksonomii

Considerând cea mai citată definiție din domeniu, cea a lui Gruber, o ontologie este o specificație a conceptualizării unui domeniu (Gruber 1993). Ulterior, Guarino și Uschold au argumentat că o ontologie reprezintă o specificare explicită și parțială a unui domeniu, folosind un vocabular de termeni și o modalitate de specificare a definițiilor acestora (Guarino și Giaretta 1995) (Uschold 1996).

Definiții mai actuale afirmă că o ontologie reprezintă un catalog al conceptelor existente într-un domeniu. Ontologia conține predicate, semantica termenilor și a conceptelor, precum și relațiile dintre acestea (Sowa 2001). O ultimă definiție, ce conține și definiția termenului de „bază de cunoștințe” este dată de (Noy și McGuinness 2001). Acestea consideră că o ontologie este formată din clase (concepte) și sloturi (roluri sau proprietăți) și restricții pe aceste sloturi. O bază de cunoștințe este formată dintr-o ontologie împreună cu un set de instanțe ale claselor.

O caracteristică frecventă a ontologiilor este reprezentată de faptul că sunt formale. După cum vom vedea și în secțiunea destinată prezentării limbajelor, cele mai multe ontologii sunt exprimate în limbaje ce pot exprima un subset decidabil al logicii cu predicate de ordinul I.

Datorită acestei caracteristici, construcția unei ontologii este deosebit de complicată, deoarece trebuie să conțină un mare număr de relații logice corecte și mai ales care să nu fie în contradicție. În aceste condiții, au apărut mai multe metodologii pentru a facilita dezvoltarea rapidă și corectă a unei ontologii. Dintre acestea, cea mai practică mi se pare cea oferită de Noy și McGuinness, care constă în următorii pași:

1. definirea domeniului și obiectivelor ontologiei prin găsirea răspunsului la următoarele întrebări:
  - Care este domeniul pe care trebuie să îl acopere ontologia?
  - La ce o vom folosi?
  - La ce tip de întrebări trebuie să ofere răspuns ontologia?
  - Cine va trebui să o utilizeze și să o mențină?
2. găsirea unor ontologii similare cu scopul de a reutiliza conceptele sau chiar de a folosi ontologia existentă în caz că aceasta există
3. alcătuirea unei liste cu termenii ce vor apărea în ontologie
4. definirea claselor și a ierarhiei de clase folosind o abordare top-down (de la concepte generale spre concepte particulare), bottom-up (încadrarea în categorii a conceptelor simple) sau mixtă.
5. definirea proprietăților (sloturilor) unei clase
6. definirea tipurilor proprietăților (cardinalitate, domeniu și co-domeniu al valorilor)
7. crearea instanțelor claselor

Urmărirea pașilor acestei metodologii asigură o mai mare probabilitate ca ontologia să fie corectă și completă.

Ontologiile se împart în mai multe categorii în funcție de destinația lor. Ontologiile de nivel înalt sunt ontologii generale ce își propun să ierarhizeze termeni care vor apărea în orice domeniu în vârful ierarhiei. Cele mai cunoscute astfel de ontologii sunt Cyc (Lenat și Guha 1990), Dolce (Guarino) și SUMO (Niles și Pease 2001), care este dezvoltată de un grup de lucru al IEEE.

Ontologiile specifice unui domeniu conțin termeni și relații particulare domeniului descris. Exemple clasice de ontologii ale domeniului sunt ontologia vinurilor (<http://protege.cim3.net/file/pub/ontologies/wine/wine.owl>) și respectiv a tipurilor de pizza (<http://www.co-ode.org/ontologies/pizza/2007/02/12/pizza.owl>). Ontologii foarte practice și deosebit de folosite sunt Wordnet – în domeniul lingvistic, precum și un mare număr de ontologii din domeniul medical „Foundational Model of Anatomy” (<http://sig.biostr.washington.edu/projects/fm/AboutFM.html>), ontologia genelor (<http://www.geneontology.org/>), ontologia proteinelor (<http://proteinontology.info/>) etc.

Folksonomiile sunt o modalitate informală de reprezentare a cunoștințelor și sunt foarte folosite în comunitățile de utilizatori. Termenul de folksonomie provine din cuvintele „folk” – popular și „taxonomy” – taxonomie. Folksonomiile sunt create de utilizatori și, pentru a fi folosite ușor de aceștia trebuie să fie cât mai informale și să nu pretindă de la aceștia nici un fel de modelare formală. Din acest motiv au foarte mult succes dar nu sunt eficiente din punct de vedere al modelării formale a cunoștințelor (Hotho et al. 2006). Din punctul de vedere al metodologiei construcției sunt total diferite de ontologii fiind create prin adnotări libere lipsite de orice formalism.

## **Limbaje pentru crearea ontologiilor – OWL și RDF**

Cunoscând ce este o ontologie ne putem întreba cum putem să o creăm. În funcție de scopul construirii ontologiei există mai multe metode de construcție și respectiv limbaje. Ontologii, conform unora din definițiile precedente pot fi reprezentate și de un dicționar. Pentru a fi însă utilizate de webul semantic ontologiile trebuie să fie utilizate de calculatoare și astfel trebuie să fie exprimate în limbaje pe care să le poată înțelege ușor și calculatoarele și oamenii. Astfel, principalele limbaje de definire a ontologiilor sunt bazate pe XML – un limbaj care este foarte ușor interpretabil de calculatoare.

RDF (Resource Description Framework) este un limbaj bazat pe sintaxa XML ce utilizează un model de reprezentare a grafurilor pentru a exprima fapte despre resurse identificate prin URI-uri (Uniform Resource Identifier). URI-urile reprezintă pentru RDF corespondentele cheilor primare din modelele relaționale prin faptul că un URI va identifica în mod unic o resursă. Limbajul RDF are ca obiectiv să ofere metadate despre resurse web (autor, descriere, data creării și altele, în Dublin Core (DublinCore 2007), să ofere modele de reprezentare a informației și să permită diferitelor aplicații să colaboreze (de exemplu o aplicație ce utilizează o bază de date poate descrie o parte a modelului utilizat folosind RDF astfel încât o altă aplicație să poată folosi datele de la prima aplicație cu semantica lor inițială).

Elementul de bază al unui document RDF este tripletul. Un triplet este o propoziție ce are un subiect, predicat și un obiect (proprietate). Subiectul și predicatul sunt resurse identificate prin URI-uri, iar obiectul poate fi o resursă sau o valoare efectivă.

Cu ajutorul RDF pot fi descrise mai multe tipuri de resurse predefinite dar pot fi definite și tipuri noi de resurse cu ajutorul unei extensii numite RDF Schema. RDF Schema (RDFS) permite definirea unor clase, instanțe și proprietăți utilizând sintaxa RDF. De asemenea, permite definirea unor relații între resurse prin oferirea posibilității de a defini subclase, subproprietăți, precum și domenii și co-domenii pentru proprietăți.

OWL (Web Ontology Language - McGuinness 2004, Patel-Schneider 2004) este un limbaj conceput pentru a defini ontologii. Este un limbaj care extinde RDF, permițând folosirea unor

instrumente de inferență pe datele din ontologie. Are trei sub-limbaje distincte OWL Lite, OWL DL și OWL Full ce diferă prin ceea ce pot exprima.

OWL Lite suportă restricții de cardinalitate (0 sau 1) precum și restricții de tip „AllValuesFrom” și „SomeValuesFrom”. Restricțiile de cardinalitate specifică numărul de proprietăți de un anumit tip pe care le poate avea o clasă. Restricțiile de tip „SomeValuesFrom” și „AllValuesFrom” specifică faptul că unele sau toate valorile proprietății pe care se aplică restricția au tipul specificat. De asemenea, se poate specifica faptul că unele clase și proprietăți sunt echivalente cu alte clase și proprietăți, ceea ce poate fi foarte util pentru a arăta că unele concepte cu nume diferite din ontologii diferite reprezintă de fapt același lucru. Tot în OWL Lite se poate specifica faptul că o proprietate poate fi funcțională (ceea ce înseamnă că un element din domeniu poate fi asociat printr-o proprietate funcțională unui singur element din co-domeniu), simetrică, tranzitivă sau inversă.

OWL DL (Description Logic) este un limbaj mai avansat, bazat pe un subset decidabil al logicii cu predicate. OWL DL permite definirea unor relații de disjuncție între clase. El permite și definirea de clase prin reuniune, diferență și intersecție de clase. Permite și definirea unor restricții de cardinalitate mai avansate, astfel că orice număr nenegativ poate fi specificat în cadrul restricției. Pe o ontologie OWL DL, motoare de raționament ca de exemplu Racer (Racer 2007), Fact++ (Fact 2007), Kaon2 (Kaon 2007) sau Pellet (Pellet 2007) pot verifica consistența modelului sau poate să realizeze clasificare automată.

OWL Full permite definirea mai multor tipuri de constrângeri decât OWL DL dar fără a oferi garanții computaționale. În momentul actual, cele mai multe aplicații și instrumente de dezvoltare sunt construite pentru OWL DL.

Următorul fragment de cod OWL DL specifică descrierea unei clase „Justificare”, modelată conform modelului din teoria argumentării a lui Toulmin (Soukoup și Titsworth 2007). Clasei Justificare i se specifică faptul că elementele sale sunt disjuncte cu cele ale claselor „Date”, „Suport”, „Afirmație”, „Forță” și „Limitare”. De asemenea, sunt definite restricții de cardinalitate în sensul că o justificare trebuie să aibă cel puțin o dată pe care să se bazeze și restricții de co-domeniu pentru proprietatea „justificare\_pt\_date”.

```
<owl:Class rdf:about="#Justificare">
  <owl:disjointWith>
    <owl:Class rdf:about="#Date"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="justificare_pt_date"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Date"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Suport"/>
  <owl:disjointWith>
    <owl:Class rdf:ID="Limitare"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Afirmație"/>
  <rdfs:subClassOf rdf:resource="#Tip_argument"/>
  <owl:disjointWith>
    <owl:Class rdf:ID="Forța"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
```

```

<owl:Restriction>
  <owl:onProperty>
    <owl:ObjectProperty rdf:about="#justificare_pt_date"/>
  </owl:onProperty>
  <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
  >1</owl:minCardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

```

## Instrumente pentru crearea ontologiilor

Protege este un editor de ontologii și mai ales o platformă pentru dezvoltarea aplicațiilor bazate pe cunoștințe. Este dezvoltat în Java și este open-source. Un alt avantaj remarcabil al acestei aplicații este că are o structură la care se pot adăuga foarte ușor plugin-uri ceea ce permite să fie extins pentru a satisface orice necesitate.

Protege are suport pentru RDF și mai oferă un format propriu pentru stocarea informațiilor, interfața grafică de construcție a ontologiilor nefiind diferită pentru vreuna din cele două opțiuni. Această interfață grafică originală permite crearea de clase, proprietăți și instanțe. De asemenea, permite specificarea relațiilor de tip moștenire între clase. Se pot defini relații prin posibilitatea de a defini domeniul și co-domeniul proprietăților. O proprietate ce are ca și co-domeniu o altă clasă constituie o relație. Adăugând plugin-uri la Protege, capacitățile pe care le are devin mult mai puternice. Cel mai important plugin este probabil cel care permite accesul la mecanismele limbajului OWL. Acest plugin permite atât crearea, editarea, salvarea și importarea de ontologii în format OWL cât și lucrul cu baze de date prin intermediul unui driver JDBC în care să fie stocate ontologiile.

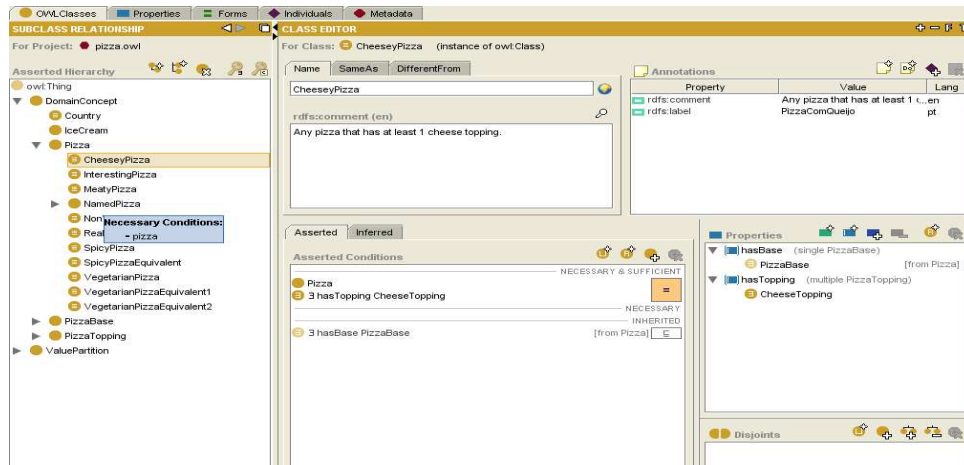


Figura 2 Reprezentarea ontologiilor în Protege

Acest plugin oferă numeroase alte facilități caracteristice OWL. Printre altele – se pot specifica constrângeri mult mai avansate decât în versiunea precedentă. Cele mai importante constrângeri ce se pot specifica sunt:

allValuesFrom – care specifică faptul că toate valorile unei proprietăți sunt într-un anumit domeniu.

someValuesFrom – care specifică faptul că o proprietate are valori într-un anumit domeniu

cardinalitate - specifică numărul minim (minCardinality), maxim (maxCardinality) sau exact (cardinality) de proprietăți distincte de tipul pe care se aplică restricția pe care le poate avea un individ din clasa pe care se aplică restricția.

operații cu mulțimi – se specifică faptul că o clasă poate fi formată ca uniune, intersecție sau diferență a altor clase (Knublauch et. al. 2004)

Aceste constrângeri pot fi specificate ca fiind necesare sau necesare și suficiente. Condiția de suficiență permite motorului de inferență realizarea de inferențe - clasificarea automată.

Folosind aceste constrângeri se pot aplica alte funcționalități ale OWL implementate în Protege. De exemplu, se pot aplica teste pe ontologie, se poate detecta ce sub-limbaj de OWL este folosit și se poate face conversie dintr-un sub-limbaj în altul (această facilitate este necesară deoarece nu există instrumente care să raționeze pe OWL Full și sunt unele care nu acceptă nici OWL DL).

Pentru a include în această platformă și raționamente pe baza ontologiilor, Protege acceptă comunicația cu instrumentele ce execută raționamente pe baza OWL folosind interfața DIG. DIG (Description Logic Implementation Group) este o interfață XML ce oferă posibilitatea de comunicare cu motoarele de inferență (DIG 2007). Prin intermediul acestei interfețe, Protege oferă opțiuni ca verificarea consistenței ontologiei și clasificare automată a claselor precum și calculul claselor obținute prin inferență.

Tot prin intermediul plugin-urilor, Protege oferă și modalități de vizualizare a informațiilor și de export în diferite formate.

Protege, fiind open-source, oferă pe lângă interfața vizuală și un API cuprinzător atât pentru tratarea ontologiilor RDF cât și pentru cele OWL. Acest API permite folosirea tuturor facilităților de mai sus în aplicații fără a implica și instrumentul de dezvoltare propriu-zis. Pe lângă acest API, mai sunt oferite și două facilități legate de exportul ontologiilor dezvoltate pe web. Prima posibilitate este de a exporta toate conceptele din baza de cunoștințe în pagini HTML statice – ceea ce devine destul de incomod când lucrăm cu baze foarte mari de cunoștințe. A doua metodă este reprezentată de o aplicație JSP care rulează pe un server Tomcat și care oferă și posibilități de editare și căutare pe lângă posibilitatea de a vizualiza conceptele din baza de cunoștințe care era oferită și în prima variantă [9].

Kaon este, de asemenea, o platformă de construcție a ontologiilor având însă marele dezavantaj că este bazat pe RDF/RDFS. Cu alte cuvinte – nu acceptă OWL. Acest dezavantaj este compensat de numeroasele instrumente pe care le oferă Kaon și pe care le vom descrie în rândurile următoare.

## **Raționamente bazate pe ontologii**

Există mai multe instrumente ce oferă efectuarea de raționamente pe baza ontologiilor. În paginile următoare ne propunem să descriem pe scurt aceste instrumente, precum și setul de expresii logice pe care le folosesc și să prezentăm câteva teste comparative între cele câteva soluții existente în acest moment.

Cum am văzut, atât în cazul limbajelor de definire a ontologiilor, cât și în cadrul platformelor de dezvoltare se face o diferență între clase și instanțe sau mai precis între ontologia propriu zisă ce conține conceptele și indivizii din baza de cunoștințe. Și în cadrul motoarelor de inferență apare această distincție, schema terminologică numindu-se TBox și totalitatea indivizilor numindu-se Abox (Baader et al. 2003). Problema este că algoritmi de raționament existenți în acest moment nu satisfac necesitățile, atât pentru TBox cât și pentru Abox, realizându-se un compromis între cele două.

În general, motoarele de inferență raționează asupra TBox-urilor, dar nu și asupra ABox-urilor. Aceasta poate cauza o problemă deoarece, în general, în lumea reală cea mai mare parte

dintr-o bază de cunoștințe este reprezentată de indivizi și interogările și raționamentele asupra indivizilor sunt mai mult folosite decât cele asupra conceptelor.

Tipurile de raționament depind direct de puterea subsetului logicii descriptive folosite în descrierea ontologiei. În funcție de ce pot exprima principalele subseturi ar fi următoarele:

AL – limbaj ce permite negarea expresiilor atomice, exprimarea restricțiilor, cuantificare existențială și exprimarea unor concepte noi ca fiind intersecția altor concepte.

S – oferă posibilitatea unor negări complexe, definirea de proprietăți tranzitive precum și toate facilitățile AL

H – oferă posibilitatea definirii de sub-proprietăți

O – oferă posibilitatea definirii de clase ca enumerare de concepte

I – oferă posibilitatea definirii de proprietăți inverse

N – oferă posibilitatea definirii de restricții de cardinalitate

(D) – oferă posibilitatea folosirii tipurilor de date (String, Boolean, Numeric, etc.) (Baader et al. 2003)

Aceste subseturi sunt importante deoarece, de exemplu, OWL-DL este bazat pe subsetul SHOIN(D) care reprezintă reuniunea subseturilor S,H,O,I,N și (D) iar editorul de ontologii Protege pe care îl vom prezenta mai jos acceptă același subset logic.

Cele mai importante motoare de inferență existente sunt Racer, Kaon2 și Pellet. Asupra acestor aplicații s-au realizat mai multe teste comparative și concluzia a fost că unele dintre bazele de cunoștințe pot fi clasificate și/sau verificate cu unele dintre motoarele de raționament, în timp ce altele pot fi verificate cu alt set. Nu toate motoarele de raționament funcționează corect și în timp finit pe toate ontologiile (chiar și dacă acestea din urmă sunt corect scrise). În general depinde de formalismul logic al ontologiei. Prin formalism logic înțelegem subsetul de operații logice din DL pe care îl folosește ontologia.

Cele mai importante tipuri de raționament care pot fi făcute sunt:

clasificarea bazată pe subsumare („subsumption”) – se identifică dacă un concept este inclus în altul

satisfacerea – verificarea dacă o expresie nu descrie un concept vid.

consistența – se verifică pentru ABox și se spune că un ABox este consistent dacă verifică modelul (orice instanță din ABox poate fi descrisă cu ajutorul modelului) (Baader 2000)

Aceste operații sunt și cele pe care, după cum se poate vedea și mai sus, le permite Protege-ul a fi operate asupra ontologiilor OWL.

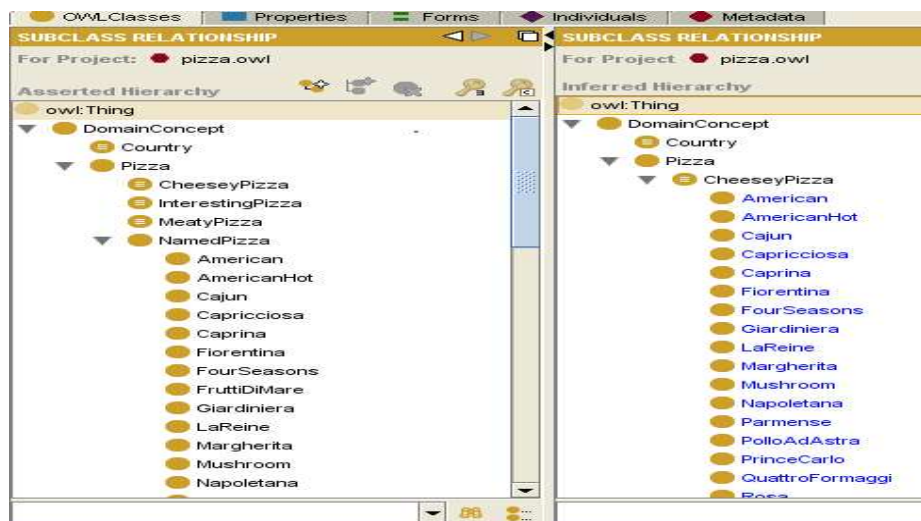


Figura 3 Clasificare automată realizată cu ajutorul unui motor de inferență în Protege

Pentru compararea instrumentelor existente au fost realizate mai multe teste comparative. (Motik 2006, Zhengxiang 2005, W3C 2003) oferă astfel de teste și în general sunt diferite între ele, astfel încât ne putem face o imagine de ansamblu. Primul test este și cel mai vechi și demonstrează doar cât de diferit se pot comporta aceste programe pe diverse date de test și mai precis că nici o astfel de soluție nu este perfectă. Al doilea test reflectă faptul că Racer este cel mai fiabil și că Fact++ este cel mai rapid, dar că se întâmplă să nu funcționeze deloc și că Pellet este cel mai lent în majoritatea testelor dar are cele mai complexe opțiuni. Testul realizat de cei de la Kaon nu este foarte elocvent deoarece, datorită faptului că ei nu au implementat mecanismul de cache al raționamentelor au realizat testele golind cache-ul celorlalte aplicații. Ori avantajul motoarelor de inferență mai noi constă tocmai în acest mecanism de cache care optimizează foarte mult procesul. Soluțiile optime în acest moment sunt reprezentate de Racer și de Pellet.

## Exemplu practic

Vom încerca să definim o ontologie a produselor hardware pentru PC-uri conform metodologiei de creare a ontologiilor propuse de Noy și McGuinness. Pașii pe care trebuie să-i urmăm pentru a construi o ontologie a calculatoarelor ar fi următorii.

la **primul pas** se vor defini domeniul și obiectivele ontologiei prin găsirea răspunsului la următoarele întrebări:

- *Care este domeniul pe care trebuie să îl acopere ontologia?*
- *La ce o vom folosi?*
- *La ce tip de întrebări trebuie să ofere răspuns ontologia?*
- *Cine va trebui să o utilizeze și să o mențină?*

Domeniul pe care vrem să-l definim este cel al calculatoarelor și mai precis al calculatoarelor personale. Vom defini noțiunea de computer și de parte a unui computer precum și legăturile între diferitele părți ale unui computer și ulterior constrângeri care apar la construcția unui computer. Utilizările unei astfel de ontologii pot fi multiple. Cele pe care vom insista în această carte sunt identificarea pretului cel mai bun pentru un anumit calculator sau o pentru o anumită componentă la firme diferite și respectiv realizarea unui configurator automat care să identifice erorile în construcția unui calculator și să recomande piese compatibile pentru realizarea unei configurații. Întrebările la care trebuie să răspundă ontologia sunt în funcție de cele 2 posibilități de utilizare, următoarele:

Pentru o memorie de 256 DDR produsă de firma X unde găsesc cel mai bun pret? Sau pur și simplu unde găsesc procesoare PIV la frecvență > 2.5 GHz?

Se da calculatorul compus dintr-o placă de bază socket 939 și un procesor PIV. Există vreă problemă de fabricație? Din procesoarele existente pe stoc care sunt compatibile cu placă de bază următoare?

Utilizatorii acestei ontologii vor fi vizitatorii unui site gen price.ro care vor vrea să vadă unde pot găsi mai ieftin produsele dorite și să-și creeze diferite configurații de calculatoare fără a avea foarte multe cunoștințe în domeniu.

**Pasul doi** presupune găsirea unor ontologii similare cu scopul de a reutiliza conceptele sau chiar de a folosi ontologia existentă în caz că aceasta există. Din păcate nu am găsit o ontologie în acest domeniu. Ceea ce ar putea să ne ajute însă sunt clasificările existente la marea majoritate a magazinelor online de piese electronice. Ceea ce vom putea observa este că aceste clasificări sunt



in cele mai multe cazuri destul de diferite.

Vom analiza 3 cazuri si anume definirea conceptului de placa de baza la 3 magazine importante de produse electronice din Romania, pe fiecare coloana fiind in fiecare caz atribuitele pe care le poate avea o placa de baza in cele 3 clasificari

<i>Magazin1</i>	<i>Magazin2</i>	<i>Magazin3</i>
<i>Producator</i>	<i>Producator</i>	<i>Producator</i>
<i>Model</i>	<i>Nume</i>	<i>Cod producator, Platforma</i>
<i>Socket</i>	<i>CPU</i>	<i>Socket</i>
<i>Chipset</i>	<i>Chipset</i>	<i>Chipset</i>
<i>I/O</i>	<i>FSB/ HTT</i>	<i>BUS</i>
<i>Sunet</i>	<i>Audio</i>	<i>Sunet</i>
<i>Sloturi</i>	<i>PCIe, AGP/PCI</i>	
<i>IDE/SATA</i>	<i>Conectori PATA, Conectori SATA</i>	<i>IDE/SATA</i>
<i>Memorie</i>	<i>Memorie</i>	<i>Memorie</i>
<i>LAN</i>	<i>Retea</i>	<i>LAN</i>
<i>Garantie</i>	<i>Garantie</i>	<i>Garantie</i>
<i>Format</i>	<i>Format</i>	<i>Form factor</i>
<i>Pret</i>	<i>Pret</i>	<i>Pret</i>

Aceasta grupare a fost destul de dificila deoarece dupa cum se poate vedea numele sunt destul de diferite si sunt unele caracteristici care nu au corespondenti directi in celelalte coloane. Astfel I/O nu apare la magazinele 2 si 3 iar la primul nu se specifica frecventa de functionare a magistralei. Remarcam numarul mare de termeni care sunt folositi pentru a defini termeni identici.

Aceste clasificari insa ne vor ajuta la pasii urmatiori pentru a identifica conceptele cheie ale ontologiei

**Pasul 3** presupune alcatuirea unei liste cu termenii ce vor aparea in ontologie. In alcatuirea acestei liste vom folosi o denumire care ni se pare relevanta urmand ca la pasii urmatiori sa gasim si sinonimele acestora.

**Boxa, Camera Web, Carcasa, Card Reader, Casca, Microfon, Cooler, Fax-Modem, Floppy Disk, Hard Disk, Imprimanta, Memorie, Monitor, Mouse, Placa de baza, Placa de captura, Placa de sunet, Placa video, Procesor, Element retea, Scaner, Software, Tableta grafice, Tastatura, Tuner TV, UPS, Unitate optica.**

Acestea ar fi principalele tipuri de componente interne unui calculator sau care pot fi atasate ca dispozitive periferice.

Pe langa aceste componente toate caracteristicile lor vor face parte din ontologie. Astfel putem sa mai adaugam si urmatoarele notiuni:

**Model, Putere, Numar boxe, Rezolutie, FPS, Senzor, Interfata, Format, Putere, Tip card, Alimentare, Lungime fir, Raspuns in frecventa (min, max), Diametru, Impedanta, RPM, Debit de aer, Nivel sonor, Chipset, Dimensiune, Capacitate, Memorie Buffer, Viteza tiparire alb-negru, Viteza tiparire color, Tehnologie tiparire, Tip imprimanta, Format, Viteza (Mhz), Viteza citire (MB/s), Viteza scriere (MB/s), Diagonala, Frecventa (min, max), Culoare, Dot pitch, Standard protectie, Socket, I/O, Sunet, Memorie, Slot, LAN, IDE/SATA, Conectori, CacheL2, CacheL1, FSB, Tehnologie, Adancime culoare, Numar taste, Telecomanda, Format TV, Format Captura, Facilitati, Software**

**Pasul 4** presupune definirea claselor si a ierarhiei de clase. La acest pas exista mai multe abordari posibile:

-abordarea top-down porneste cu crearea conceptelor cele mai generale si apoi particularizarea acestora. De exemplu o clasa din varful ierarhiei ar fi “Componenta” care apoi se va particulariza in “Placa de baza”, “Procesor”, etc. Clasa “Placa de Baza” va fi ulterior descompusa dupa principalul atribut in “Placa de baza socket 939”, “Placa de Baza socket 754” etc.

-abordarea bottom-up presupune procesul invers. Se pleaca de la cele mai simple concepte si apoi se incearca unirea lor in categorii. Astfel vor exista clasele “CPU Sempron 2200+ Socket A” si clasele “Sempron 2300+ Socket A” care impreuna cu altele vor forma clasa “Procesoare Socket A”. Aceasta clasa va fi grupata impreuna cu celelalte clase ce descriu procesoare si se va obtine clasa “Procesor”.

-abordarea mixta presupune gasirea elementelor cele mai generale si mai particulare si identificarea ulterioara a conceptelor intermediare.

**Pasul 5** presupune definirea proprietatilor claselor:

De exemplu pentru Procesor vom defini proprietatea frecventa si pentru placa de baza proprietatea tipSocket. Clasa componenta va avea o proprietate producator care va fi mostenita de toate clasele ce sunt de tipul componenta.

**Pasul 6** presupune definirea constrangerilor pentru fiecare proprietate. Aceste constrangeri pot fi de urmatoarele tipuri:

Cardinalitate – de cate ori poate sa apara proprietatea respectiva (de exemplu o componenta are un singur producator)

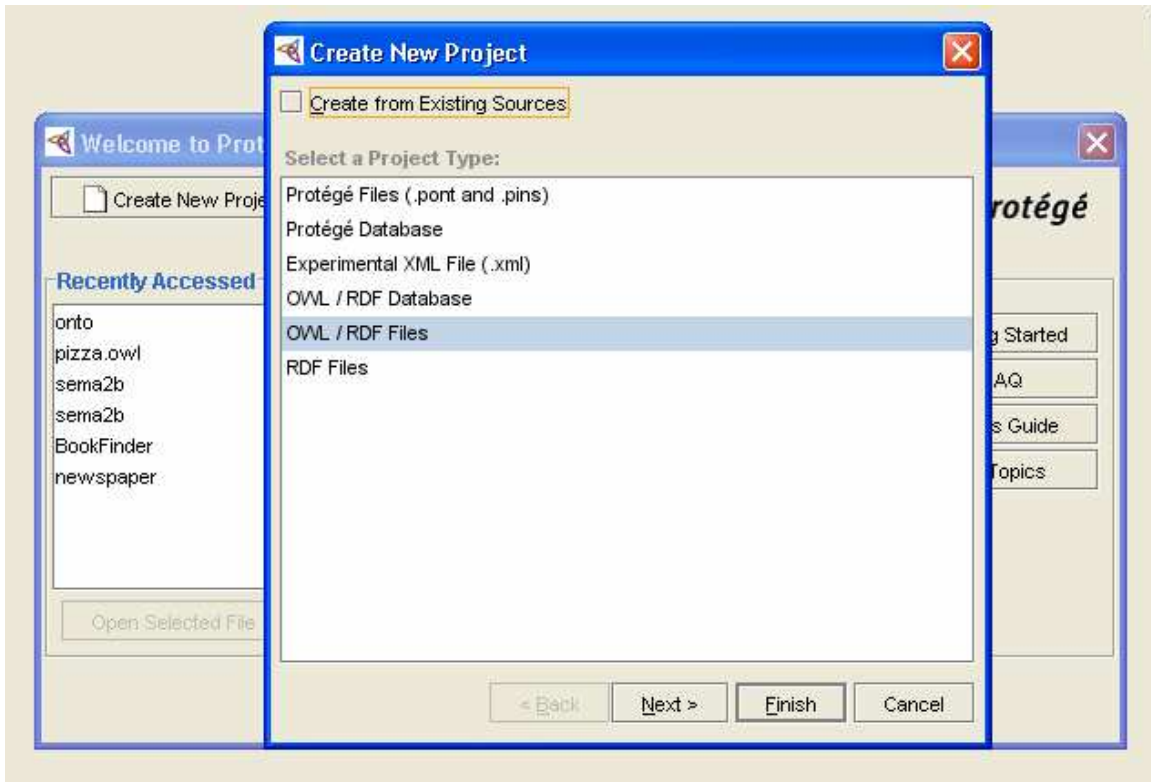
Domeniu (domain) – pe ce clase se aplica proprietatea

Codomeniu (range) – in ce clase sau in ce tipuri de date poate avea valori proprietatea

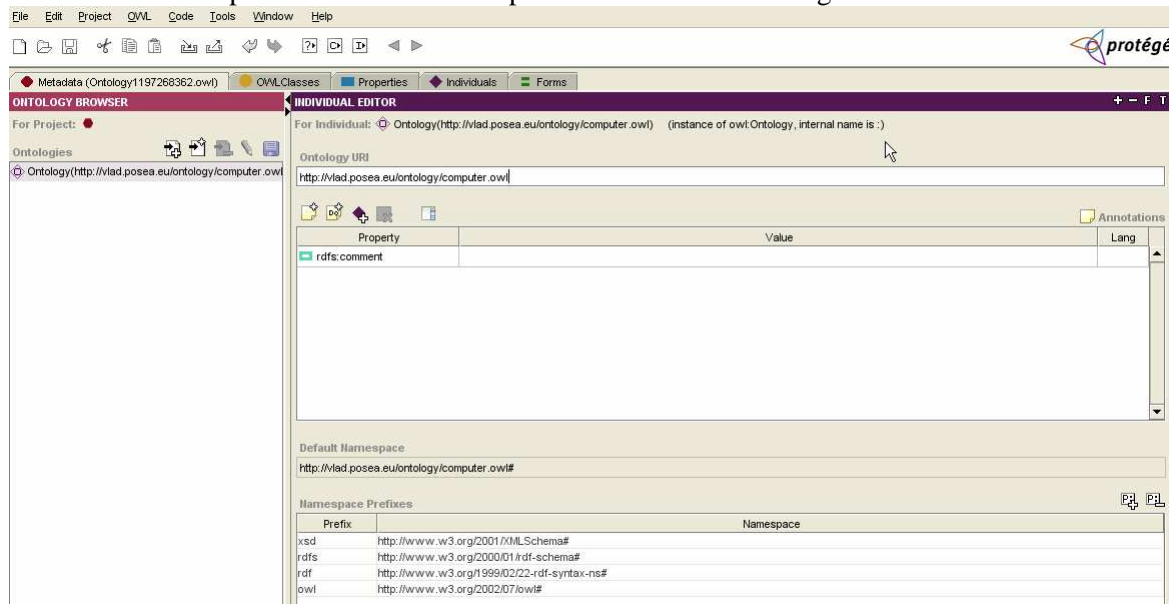
De exemplu proprietatea producator a clasei “Componenta” are domeniu “Componenta” si range clasa “Producator”

In imaginile urmatoare vom exemplifica derularea acestor pasi (4,5,6 deoarece primii 3 reprezinta colectarea termenilor) folosind Protege.

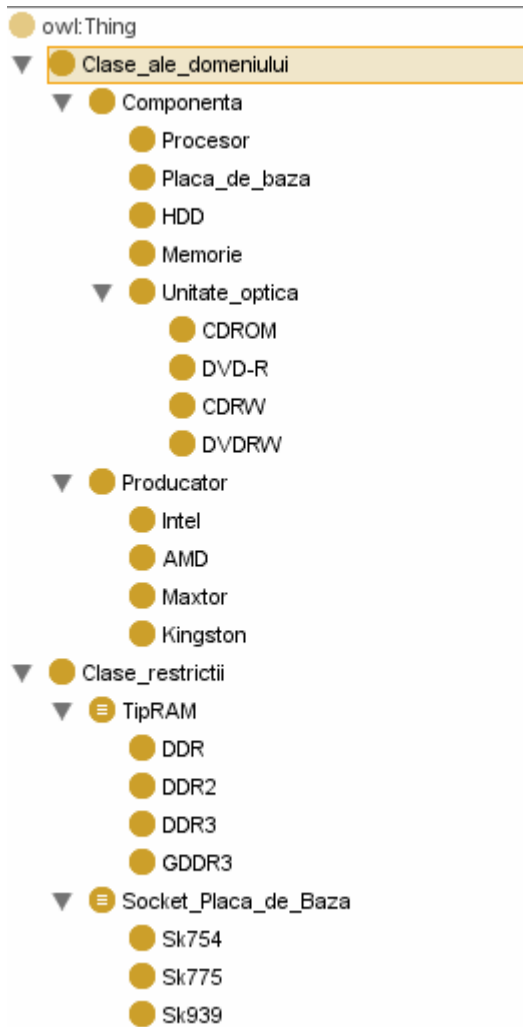
1. Crearea unui nou proiect de tip RDF/OWL



## 2. Fereastra de start a proiectului – definirea spatiului de nume al ontologiei



3. Constructia ierarhiei de clase (se observa ca am atasat unele clase pentru a specifica restrictii- vom vedea un pic mai incolo la ce ne folosesc)



#### 4. Definirea proprietatilor impreuna cu domeniu si co-domeniu

**PROPERTY BROWSER**

For Project: ● computer

Object   Datatype   Annotation   All

Object properties

- areProducator

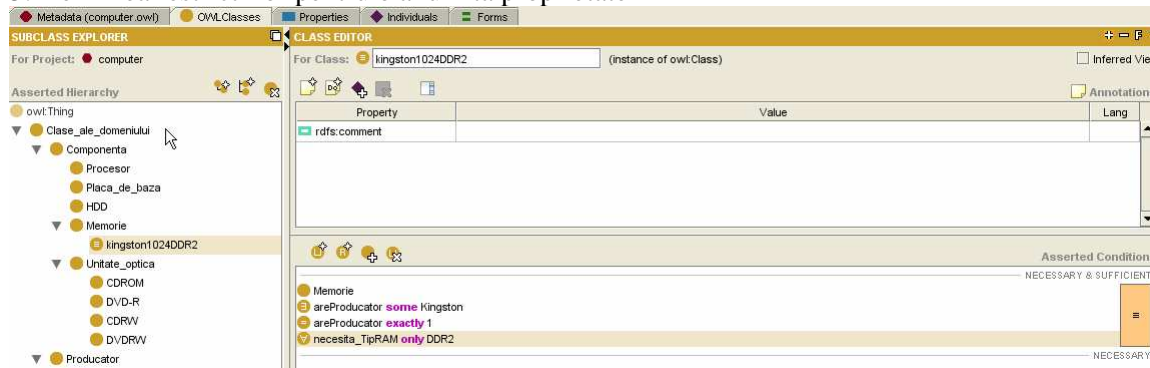
**PROPERTY EDITOR**

For Property: areProducator (instance of owl:OntProperty)

Property	
rdfs:comment	

Domain    Range

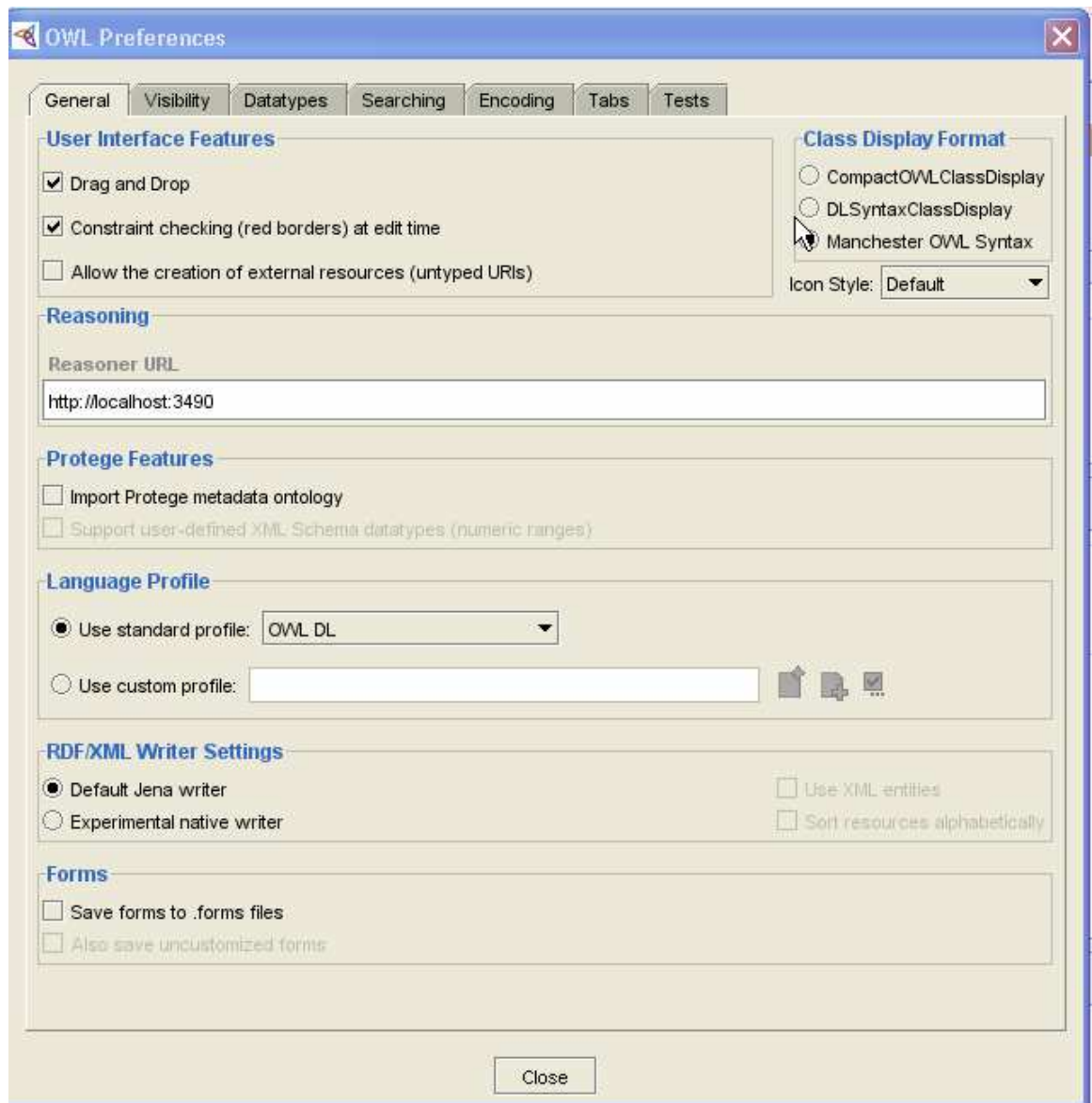
## 5. Definirea restrictiilor pentru o anumita proprietate



In figura de mai sus definim restrictii pentru o memorie kingston ddr2. Aceasta este o subclasa a “memorie”, are 1 singur producator (restrictie mostenita de la “Componenta”) si necesita\_TipRAM (o noua proprietate ce are domeniu Memorie si co-domeniu TipRAM) poate avea numai valoarea DDR2. Restrictiile definite sunt trecute in zona “necesare si suficiente” astfel incat pot fi prelucrate de un reasoner.

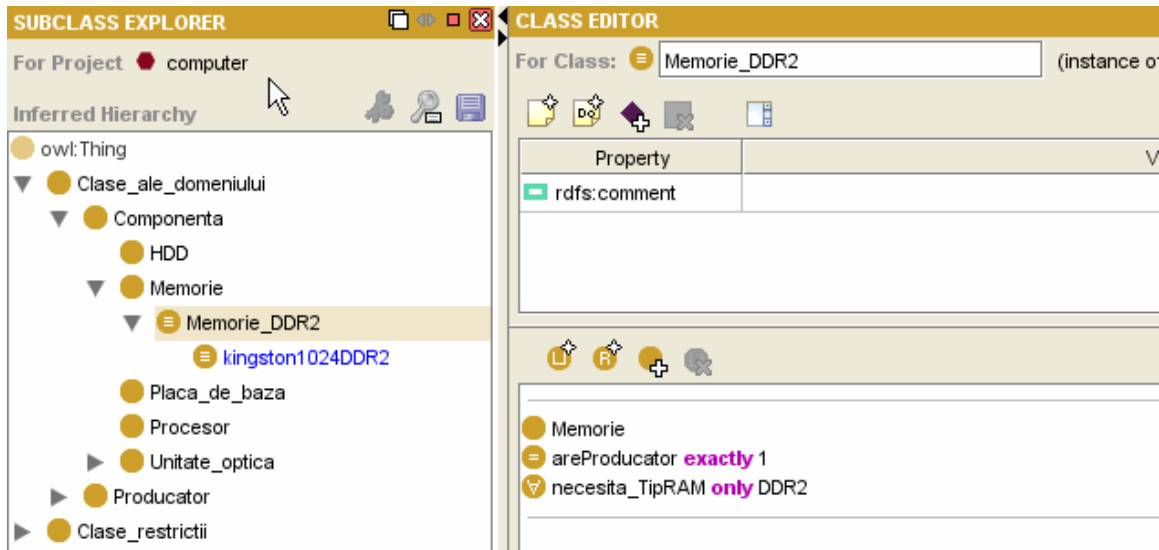
## 6. Configurare reasoner

Din meniul OWL se alege Preferences si se ajunge in fereastra din figura urmatoare. In cazul in care folositi ca reasoner Fact++ (<http://owl.man.ac.uk/factplusplus/>) acesta trebuie pornit in prealabil si va rula in mod default pe portul 3490. Acest port trebuie specificat si in fereastra de mai jos. Pentru alte reasonere trebuie consultata documentatia pentru a vedea care este portul default si/sau cum poate fi acesta schimbat.



Efectuarea unei clasificari

Din meniul OWL se alege optiunea Classify taxonomy care produce urmatorul output

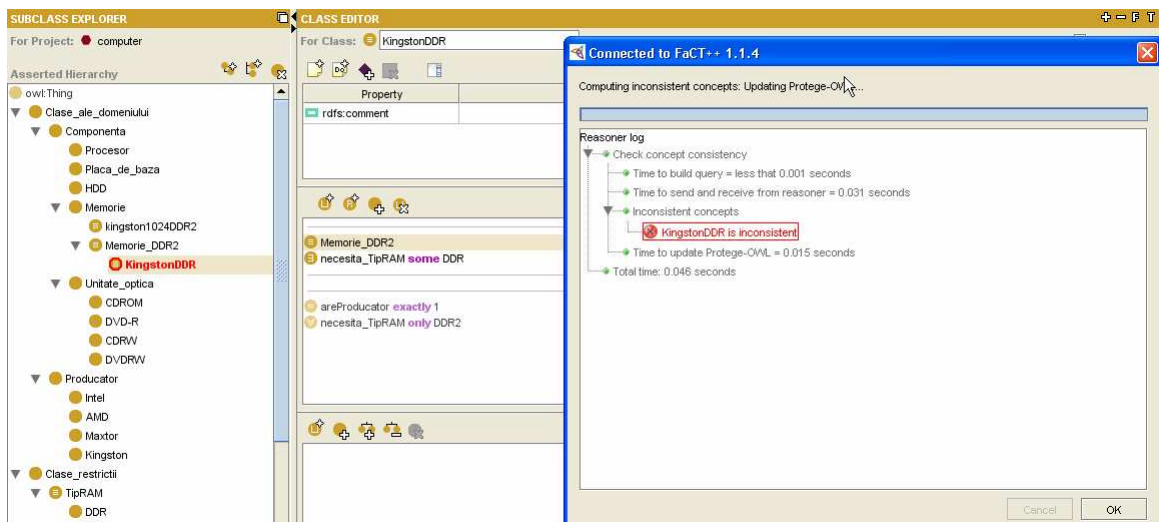


In prealabil am definit o clasa Memorie\_DDR2 care are constrangerile urmatoare:  
 Este subclasa a “Memorie”  
 Are un singur producator  
 Necesita\_TipRam ddr2  
 Reasonerul detecteaza ca memoria pe care noi am definit-o ca fiind kingston1024DDR2 este subclasa a acestui concept (avand in plus doar producatorul selectat)  
 Aceste tipuri de clasificari pot fi folosite ca si interogari ale ontologiei.

#### Detectia inconsistentelor

Reasonerele pot detecta o gama larga de inconsistente (greseli in definirea unor concepte). De exemplu definim clasa KingstonDDR careia ii atribuim urmatoarele restrictii:

- Subclasa a MemorieDDR2
- Necesita\_TipRAM DDR
- (mai trebuie sa specificam faptul ca DDR, DDR2, DDR3, GDDR3 sunt concepte disjuncte)



Pentru aceasta definitie reasonerul detecteaza inconsistenta, asa cum se poate vedea in imaginea de mai sus.

## Semantic Web(!?)

Cum afecteaza deci aceste informatii web-ul asa cum il stim acum?

Foarte multe ontologii, mai mult sau mai putin complicate sunt deja folosite pentru a descrie intr-un mod inteligibil de catre calculatoare continutul unor pagini web. Cele mai cunoscute ar fi (si poate v-ati intalnit cu ele deja fara sa va dati seama)

- friend of a friend (foaf) – permite descrierea profilului personal (v. tema 1 ☺ )  
<http://www.foaf-project.org/>
- dublin core (dc) - <http://dublincore.org/> - permite adaugarea de meta-date referitoare la un articol (autor, data la care a fost scris, etc.)
- description of a career (doac) - <http://ramonantonio.net/doac/0.1/> - set de concepte ce permit descrierea unei cariere
- XFN – XHTML Friends Network

Exemplu de includere al unui fisier foaf intr-o pagina web (in sectiunea de head):

```
<link rel="meta" type="application/rdf+xml" title="FOAF" href="http://pagina-me.ro/foaf.rdf"/>
```

Foarte putine ontologii sunt folosite pentru a descrie formal continutul efectiv al unei pagini web.

## Referințe

- Baader F., Calvanese D, McGuinness D.L., Nardi D., Patel-Schneider P.F., eds.(2003), The Description Logic Handbook: Theory, Implementation and Applications,. Cambridge University Press, 2003
- Baader F. (2000) Tableau Algorithms for Description Logics – <http://lat.inf.tu-dresden.de/~baader/Talks/Tableaux2000.pdf>
- Berners-Lee T., Hendler J., Lassila O. (2001) The Semantic Web. Scientific American, May 2001.
- DIG (2007) – <http://dig.sourceforge.net/> (accesat 2007)
- Dublin Core Metadata Initiative (DCMI) (2007)– <http://dublincore.org> (accesat 2007)
- Fact++ (2007) - site web - <http://owl.man.ac.uk/factplusplus/> (accesat 2007)
- FOAF (2007) – site web - <http://www.foaf-project.org/> (accesat 2007)
- Gabel T., Sure Y., Voelker J. (2001) KAON – An Overview - Karlsruhe Ontology Management Infrastructure - Institute AIFB, University of Karlsruhe
- Gangemi A., Guarino N., Masolo C., Oltramari A., Schneider L. (2002) Sweetening Ontologies with DOLCE. EKAW 2002: 166-181
- Gruber. T.R.(1993) A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220
- Guarino, N., Giaretta, P. (1995). Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (ed.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing 1995. IOS Press, Amsterdam: 25-32.
- Haystack project (2007) - <http://groups.csail.mit.edu/haystack/> (accesat 2007)
- Hotho A., Jascke R., Schmitz C., Stumme G. (2006) Information Retrieval in Folksonomies: Search and Ranking. ESWC 2006: 411-426
- Jena2 web site (2007) - <http://www.hpl.hp.com/semweb/jena.htm> (accesat 2007)
- Kaon2 site web (2007) – <http://kaon2.semanticweb.org> (accesat 2007)
- Knublauch H., Fergerson R., Noy N., Musen M. (2004) The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. [International Semantic Web Conference 2004](#): 229-243
- Koivunen, Miller(2001) W3C Semantic Web Activity. <http://www.w3.org/2001/12/semweb-fin/w3csw>
- Lenat D., Guha. R. V. (1990). Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project. Addison-Wesley. ISBN 0-201-51752-3.
- McGuinness D.L. , van Harmelen F. eds. (2004) OWL Web Ontology Language Overview - W3C Recommendation. <http://www.w3.org/TR/owl-features/>



- Motik B., Sattler U. (2006) Practical DL Reasoning over Large ABoxes with KAON2  
[http://www.fzi.de/KCMS/kcms\\_file.php?action=link&id=580](http://www.fzi.de/KCMS/kcms_file.php?action=link&id=580)
- Niles, I., Pease, A. 2001. Towards a Standard Upper Ontology. In Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), Chris Welty and Barry Smith, eds, Ogunquit, Maine, October 17-19, 2001.
- Noy N., McGuinness D. L. (2001) Ontology Development 101: A Guide to Creating Your First Ontology  
Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, Stanford Medical Informatics  
Technical Report SMI-2001-0880, March 2001.
- Patel-Schneider, Hayes, Horrocks, eds., (2004) OWL Web Ontology Language  
Semantics and Abstract Syntax - W3C Recommendation. Feb 2004. -  
<http://www.w3.org/TR/owl-semantics/>
- Pellet site web (2007) - <http://www.mindswap.org/2003/pellet/index.shtml> (accesat 2007)
- Protege site web (2007) <http://protege.stanford.edu> (accesat 2007)
- Piggy Bank (2007) - [http://simile.mit.edu/wiki/Piggy\\_Bank](http://simile.mit.edu/wiki/Piggy_Bank) (accesat 2007)
- Racer site web (2007) - <http://www.racer-systems.com/products/racerpro/index.phtml> (accesat 2007)
- RDF (2007) - site-ul web al W3C - [www.w3.org/RDF/](http://www.w3.org/RDF/) (accesat 2007)
- RDF Schema (2007) - site-ul web al W3C- <http://www.w3.org/TR/rdf-schema/> (accesat 2007)
- RDQL (2007) A Query Language for RDF - <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>
- Soukup C., Titsworth S. - A Description of Toulmin's Layout of Argumentation -  
<http://www.unl.edu/speech/comm109/Toulmin/layout.htm>
- Sowa J.F. (2001) <http://www.jfsowa.com/ontology/>.
- Uschold, M. (1996) Building Ontologies: Towards A Unified Methodology. Proc. Expert Systems 96, Cambridge, December 16-18th.
- W3C (2003) OWL Implementations Test - <http://www.w3.org/2003/08/owl-systems/test-results-out>
- Zhengxiang Pan (2005) - Benchmarking DL Reasoners Using Realistic Ontologies -  
<http://www.mindswap.org/2005/OWLWorkshop/sub6.pdf>