UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI

Instrumente Structurale
2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Interacțiunea om-calculator

31. Proiectare şi implementare de interfețe bazate pe semantică (RDF, OWL)

Semantic Web
RDF+OWL+Protege

1. Introducere

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

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

În acest capitol vom discuta îmbunătăţirile fundamentale pe care le aduce webul semantic şi principalele mecanisme pe care el le propune. Vom arăta care este structura lui, limbajele caracteristice şi aplicaţiile 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 şi vom arăta câteva aplicaţii tipice pentru webul semantic şi foarte semnificative din punctul de vedere al interfeţei om-calculator. Figura 1 prezintă structura webului semantic conform cu (Koivunen, 2001).

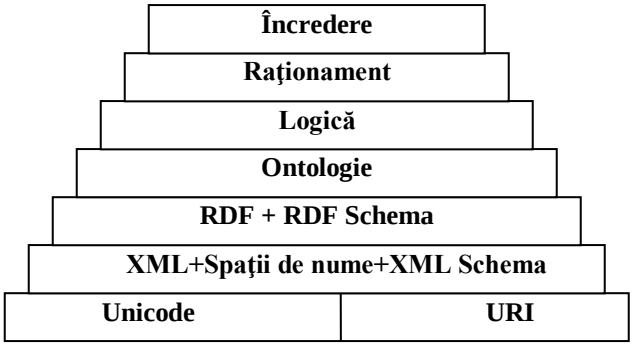


Figura 1 Structura webului semantic

2. Ontologii, limbaje şi instrumente pentru crearea ontologiilor

2.1. Ontologii şi folksonomii

Considerând cea mai citată definiţie din domeniu, cea a lui Gruber, o ontologie este o specificaţie a conceptualizării unui domeniu (Gruber 1993). Ulterior, Guarino şi Uschold au argumentat că o ontologie reprezintă o specificare explicită şi parţială a unui domeniu, folosind un vocabular de termeni şi o modalitate de specificare a definiţiilor acestora (Guarino şi Giaretta 1995) (Uschold 1996).

Definiţii mai actuale afirmă că o ontologie reprezintă un catalog al conceptelor existente într-un domeniu. Ontologia conţine predicate, semantica termenilor şi a conceptelor, precum şi relaţiile dintre acestea (Sowa 2001). O ultimă definiţie, ce conţine şi definiţia termenului de „bază de cunoştinţe" este dată de (Noy şi McGuinness 2001). Acestea consideră că o ontologie este formată din clase (concepte) şi sloturi (roluri sau proprietăţi) şi restricţii pe aceste sloturi. O bază de cunoştinţe este formată dintr-o ontologie împreună cu un set de instanţe ale claselor.

O caracteristică frecventă a ontologiilor este reprezentată de faptul că sunt formale. După cum vom vedea şi în secţiunea 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, construcţia unei ontologii este deosebit de complicată, deoarece trebuie să conţină un mare număr de relaţii logice corecte şi mai ales care să nu fie în contradicţie. În aceste condiţii, au apărut mai multe metodologii pentru a facilita dezvoltarea rapidă şi corectă a unei ontologii. Dintre acestea, cea mai practică mi se pare cea oferită de Noy şi McGuinness, care constă în următorii paşi:

1. definirea domeniului şi obiectivelor ontologiei prin găsirea raspunsului la următoarele întrebări:
   - Care este domeniul pe care trebuie să îl acopere ontologia?
   - La ce o vom folosi?
   - La ce tip de întrebări trebuie sa ofere răspuns ontologia?
   - Cine va trebui să o utilizeze şi să o menţină?

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 si 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 proprietăţilor (sloturilor) unei clase

6. definirea tipurilor proprietăţilor (cardinalitate, domeniu şi co-domeniu al valorilor)

7. crearea instanţelor claselor

Urmărirea paşilor acestei metodologii asigură o mai mare probabilitate ca ontologia să fie corectă şi completă.

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

Ontologiile specifice unui domeniu conţin termeni şi relaţii particulare domeniului descris. Exemple clasice de ontologii ale domeniului sunt ontologia vinurilor (http://protege.cim3.net/file/pub/ontologies/wine/wine.owl) şi respectiv a tipurilor de pizza (http://www.co-ode.org/ontologies/pizza/2007/02/12/pizza.owl). Ontologii foarte practice şi deosebit de folosite sunt Wordnet – în domeniul lingvistic, precum şi 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 cunoştinţelor şi sunt foarte folosite în comunităţile de utilizatori. Termenul de foksonomie provine din cuvintele „folk" – popular şi „taxonomy" – taxonomie. Folksonomiile sunt create de utilizatori şi, pentru a fi folosite uşor de aceştia trebuie să fie cât mai informale şi să nu pretindă de la aceştia 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 cunoştinţelor (Hotho et al. 2006). Din punctul de vedere al metodologiei construcţiei sunt total diferite de ontologii fiind create prin adnotări libere lipsite de orice formalism.

2.2. Limbaje pentru crearea ontologiilor – OWL şi RDF

Cunoscând ce este o ontologie ne putem întreba cum putem să o creăm. În funcţie de scopul construirii ontologiei există mai multe metode de construcţie şi respectiv limbaje. Ontologii, conform unora din definiţiile precedente pot fi reprezentate şi de un dicţionar. Pentru a fi însă utilizate de webul semantic ontologiile trebuie să fie utilizate de calculatoare şi astfel trebuie să fie exprimate în limbaje pe care să le poată înţelege uşor şi calculatoarele şi oamenii. Astfel, principalele limbaje de definire a ontologiilor sunt bazate pe XML – un limbaj care este foarte uşor 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 relaţionale 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 şi altele, în Dublin Core (DublinCore 2007), să ofere modele de reprezentare a informaţiei şi să permită diferitelor aplicaţii să

colaboreze (de exemplu o aplicaţie ce utilizează o bază de date poate descrie o parte a modelului utilizat folosind RDF astfel încât o altă aplicaţie să poată folosi datele de la prima aplicaţie cu semantica lor iniţială).

Elementul de bază al unui document RDF este tripletul. Un triplet este o propoziţie ce are un subiect, predicat şi un obiect (proprietate). Subiectul şi 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 şi tipuri noi de resurse cu ajutorul unei extensii numite RDF Schema. RDF Schema (RDFS) permite definirea unor clase, instanţe şi proprietăţi utilizând sintaxa RDF. De asemenea, permite definirea unor relaţii între resurse prin oferirea posibilităţii de a defini subclase, subproprietăţi, precum şi domeniu şi co-domeniu pentru proprietăţi.

OWL (Web Ontology Language - McGuinness 2004, Patel-Schneider 2004) este un limbaj conceput pentru a defini ontologii. Este un limbaj care extinde RDF, permiţând folosirea unor instrumente de inferenţă pe datele din ontologie. Are trei sub-limbaje distincte OWL Lite, OWL DL şi OWL Full ce diferă prin ceea ce pot exprima.

OWL Lite suportă restricţii de cardinalitate (0 sau 1) precum şi restricţii de tip „AllValuesFrom" şi „SomeValuesFrom". Restricţiile de cardinalitate specifică numărul de proprietăţi de un anumit tip pe care le poate avea o clasă. Restricţiile de tip „SomeValuesFrom" şi „AllValuesFrom" specifică faptul că unele sau toate valorile proprietăţii pe care se aplică restricţia au tipul specificat. De asemenea, se poate specifica faptul că unele clase şi proprietăţi sunt echivalente cu alte clase şi proprietăţi, ceea ce poate fi foarte util pentru a arăta că unele concepte cu nume diferite din ontologii diferite reprezintă de fapt acelaşi lucru. Tot în OWL Lite se poate specifica faptul că o proprietate poate fi funcţională (ceea ce înseamnă că un element din domeniu poate fi asociat printr-o proprietate funcţională 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 relaţii de disjuncţie între clase. El permite şi definirea de clase prin reuniune, diferenţă şi intersecţie de clase. Permite şi definirea unor restricţii de cardinalitate mai avansate, astfel că orice număr nenegativ poate fi specificat în cadrul restricţiei. Pe o ontologie OWL DL, motoare de raţionament ca de exemplu Racer (Racer 2007), Fact++ (Fact 2007), Kaon2 (Kaon 2007) sau Pellet (Pellet 2007) pot verifica consistenţa modelului sau poate să realizeze clasificare automată.

OWL Full permite definirea mai multor tipuri de constrângeri decât OWL DL dar fără a oferi garanţii computaţionale. În momentul actual, cele mai multe aplicaţii şi 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 şi Titsworth 2007). Clasei Justificare i se specifică faptul că elementele sale sunt disjuncte cu cele ale claselor „Date", „Suport", „Afirmaţie", „Forţă" şi „Limitare". De asemenea, sunt definite restricţii de cardinalitate în sensul că o justificare trebuie să aibă cel puţin o dată pe care să se bazeze şi restricţii 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="#Afirmaţie"/>
    <rdfs:subClassOf rdf:resource="#Tip_argument"/>
    <owl:disjointWith>
      <owl:Class rdf:ID="Forţa"/>
    </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>
```

2.3. Instrumente pentru crearea ontologiilor

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

Protege are suport pentru RDF şi mai oferă un format propriu pentru stocarea informaţiilor, interfaţa grafică de construcţie a ontologiilor nefiind diferită pentru vreuna din cele două opţiuni. Această interfaţă grafică originală permite crearea de clase, proprietăţi şi instanţe. De asemenea, permite specificarea relaţiilor de tip moştenire între clase. Se pot defini relaţii prin posibilitatea de a defini domeniul şi co-domeniul proprietăţilor. O proprietate ce are ca şi co-domeniu o altă clasă constituie o relaţie. Adăugând plugin-uri la Protege, capacităţile 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 şi importarea de ontologii în format OWL cât şi lucrul cu baze de date prin intermediul unui driver JDBC în care să fie stocate ontologiile.

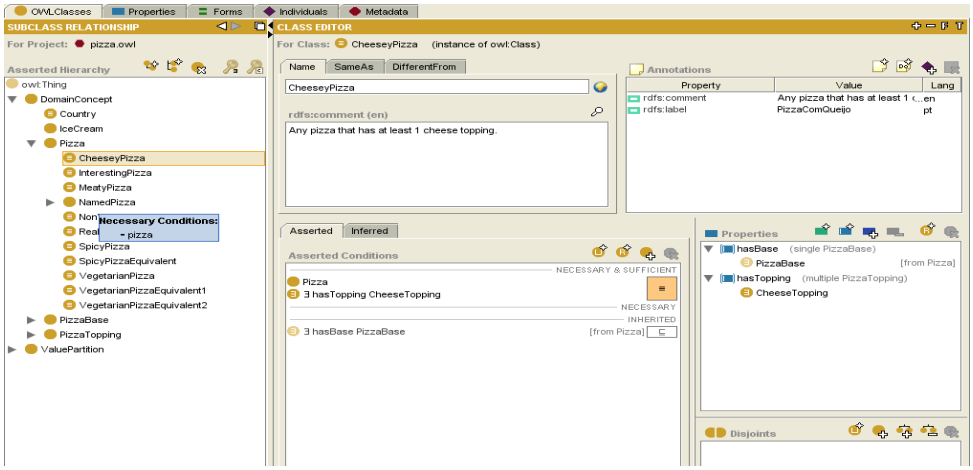


Figura 2 Reprezentarea ontologiilor în Protege

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

- allValuesFrom – care specifică faptul că toate valorile unei proprietăţi 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 proprietăţi distincte de tipul pe care se aplică restricţia pe care le poate avea un individ din clasa pe care se aplică restricţia.
- operaţii cu mulţimi – se specifică faptul că o clasă poate fi formată ca uniune, intersecţie sau diferenţă a altor clase(Knublauh et. al. 2004)

Aceste constrângeri pot fi specificate ca fiind necesare sau necesare şi suficiente. Condiţia de suficienţă permite motorului de inferenţă realizarea de inferenţe - clasificarea automată.
Folosind aceste constrângeri se pot aplica alte funcţionalităţi ale OWL implementate în Protege. De exemplu, se pot aplica teste pe ontologie, se poate detecta ce sub-limbaj de OWL este folosit şi se poate face conversie dintr-un sub-limbaj în altul (această facilitate este necesară deoarece nu există instrumente care să raţioneze pe OWL Full şi sunt unele care nu acceptă nici OWL DL).

Pentru a include în această platformă şi raţionamente pe baza ontologiilor, Protege acceptă comunicaţia cu instrumentele ce execută raţionamente pe baza OWL folosind interfaţa DIG. DIG (Description Logic Implementation Group) este o interfaţă XML ce oferă posibilitatea de comunicare cu motoarele de inferenţă (DIG 2007). Prin intermediul acestei interfeţe, Protege oferă opţiuni ca verificarea consistenţei ontologiei şi clasificare automată a claselor precum şi calculul claselor obţinute prin inferenţă.
Tot prin intermediul plugin-urilor, Protege oferă şi modalităţi de vizualizare a informaţiilor şi de export în diferite formate.

Protege, fiind open-source, oferă pe lângă interfaţa vizuală şi un API cuprinzător atât pentru tratarea ontologiilor RDF cât şi pentru cele OWL. Acest API permite folosirea tuturor facilităţilor de mai sus în aplicaţii fără a implica şi instrumentul de dezvoltare propriu-zis. Pe lângă acest API, mai sunt oferite şi două facilităţi legate de exportul ontologiilor dezvoltate pe web. Prima posibilitate este de a exporta toate conceptele din baza de cunoştinţe în pagini HTML statice – ceea ce devine destul de incomod când lucrăm cu baze foarte mari de cunoştinţe. A doua metodă este reprezentată de o aplicaţie JSP care rulează pe un server Tomcat şi care oferă şi posibilităţi de editare şi căutare

pe lângă posibilitatea de a vizualiza conceptele din baza de cunoştinte care era oferită şi în prima variantă [9].
Kaon este, de asemenea, o platformă de construcţie 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 şi pe care le vom descrie în rândurile următoare.

2.4. Raţionamente bazate pe ontologii

Există mai multe instrumente ce oferă efectuarea de raţionamente pe baza ontologiilor. În paginile următoare ne propunem să descriem pe scurt aceste instrumente, precum şi setul de expresii logice pe care le folosesc şi să prezentăm câteva teste comparative între cele câteva soluţii existente în acest moment.
Cum am văzut, atât în cazul limbajelor de definire a ontologiilor, cât şi în cadrul platformelor de dezvoltare se face o diferenţă între clase şi instanţe sau mai precis între ontologia propriu zisă ce conţine conceptele şi indivizii din baza de cunoştinţe. Şi în cadrul motoarelor de inferenţă apare această distincţie, schema terminologică numindu-se TBox şi totalitatea indivizilor numindu-se Abox (Baader et al. 2003). Problema este că algoritmii de raţionament existenţi în acest moment nu satisfac necesităţile, atât pentru TBox cât şi pentru Abox, realizându-se un compromis între cele două.
În general, motoarele de inferenţă raţionează asupra TBox-urilor, dar nu şi asupra ABox-urilor. Aceasta poate cauza o problemă deoarece, în general, în lumea reală cea mai mare parte dintr-o bază de cunoştinţe este reprezentată de indivizi şi interogările şi raţionamentele asupra indivizilor sunt mai mult folosite decât cele asupra conceptelor.
Tipurile de raţionament depind direct de puterea subsetului logicii descripţionale folosite in descrierea ontologiei. În funcţie de ce pot exprima principalele subseturi ar fi următoarele:

AL – limbaj ce permite negarea expresiilor atomice, exprimarea restricţiilor, cuantificare existenţială şi exprimarea unor concepte noi ca fiind intersecţia altor concepte.

S – oferă posibilitatea unor negări complexe, definirea de proprietăţi tranzitive precum şi toate facilităţile AL

H – oferă posibilitatea definirii de sub-proprietăţi

O – oferă posibilitatea definirii de clase ca enumerare de concepte

I – oferă posibilitatea definirii de proprietăţi inverse

N – oferă posibilitatea definirii de restricţii 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 şi (D) iar editorul de ontologii Protege pe care îl vom prezenta mai jos acceptă acelaşi subset logic.

Cele mai importante motoare de inferenţă existente sunt Racer, Kaon2 şi Pellet. Asupra acestor aplicaţii s-au realizat mai multe teste comparative şi concluzia a fost că unele dintre bazele de cunoştinţe pot fi clasificate şi/sau verificate cu unele dintre motoarele de raţionament, în timp ce altele pot fi verificate cu alt set. Nu toate motoarele de raţionament funcţionează corect şi în timp finit pe toate ontologiile (chiar şi dacă acestea din urmă sunt corect scrise). În general depinde de formalismul logic al ontologiei. Prin formalism logic înţelegem subsetul de operaţii logice din DL pe care îl foloseşte ontologia.

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

- clasificarea bazată pe subsumare („subsumption”) – se identifică dacă un concept este inclus în altul
- satisfacerea – verificarea daca o expresie nu descrie un concept vid.
- consistenţa – se verifică pentru ABox şi se spune că un ABox este consistent dacă verifică modelul (orice instanţă din ABox poate fi descrisă cu ajutorul modelului) (Baader 2000)

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

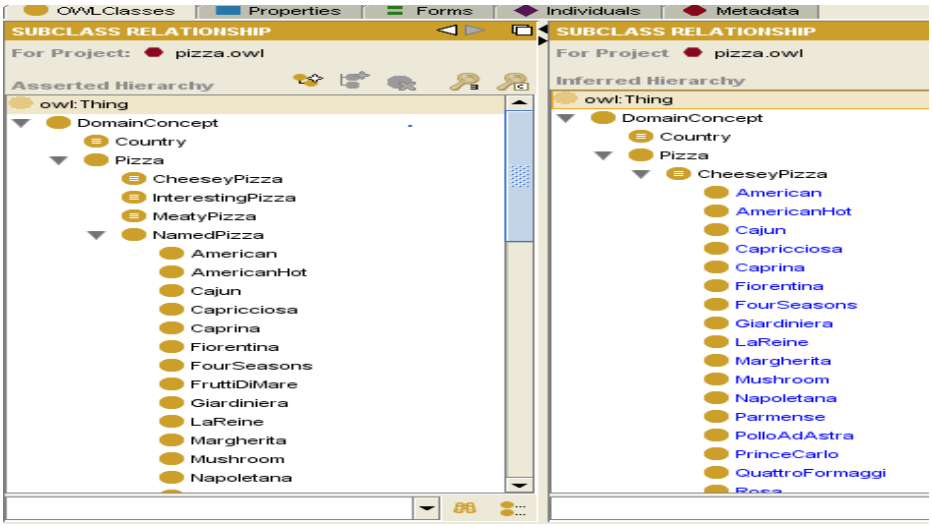


Figura 3 Clasificare automată realizată cu ajutorul unui motor de inferenţă în Protege

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

3. Exemplu practic

Vom incerca sa definim o ontologie a produselor hardware pentru PC-uri conform metodologiei de creare a ontologiilor propuse de Noy si McGuinness.
Pasii pe care trebuie sa-i urmam pentru a construi o ontologie a calculatoarelor ar fi urmatorii.

la **primul pas** se vor defini domeniul si obiectivele ontologiei prin gasirea raspunsului la urmatoarele intrebari:

- Care este domeniul pe care trebuie sa il acopere ontologia?
- La ce o vom folosi?
- La ce tip de intrebari trebuie sa ofere raspuns ontologia?

- Cine va trebui sa o utilizeze si sa o mentina?

Domeniul pe care vrem sa-l definim este cel al calculatoarelor si mai precis al calculatoarelor personale. Vom defini notiunea de computer si de parte a unui computer precum si legaturile intre diferitele parti ale unui computer si ulterior constrangeri care apar la constructia unui computer

Utilizarile unei astfel de ontologii pot fi multiple. Cele pe care vom insista in aceasta carte sunt identificarea pretului cel mai bun pentru un anumit calculator sau o pentru o anumita componenta la firme diferite si respectiv realizarea unui configurator automat care sa identifice erorile in constructia unui calculator si sa recomande piese compatibile pentru realizarea unei configuratii.

Intrebarile la care trebuie sa raspunda ontologia sunt in functie de cele 2 posibilitati de utilizare, urmatoarele:

Pentru o memorie de 256 DDR produsa de firma X unde gasesc cel mai bun pret? Sau pur si simplu unde gasesc procesoare PIV la frecventa > 2.5 GHz?

Se da calculatorul compus dintr-o placa de baza socket 939 si un procesor PIV. Exista vreo problema de fabricatie? Din procesoarele existente pe stoc care sunt compatibile cu placa de baza urmatoare?

Utilizatorii acestei ontologii vor fi vizitatorii unui site gen price.ro care vor vrea sa vada unde pot gasi mai ieftin produsele dorite si sa-si creeze diferite configuratii de calculatoare fara a avea foarte multe cunostinte in domeniu.

Pasul doi presupune gasirea unor ontologii similare cu scopul de a reutiliza conceptele sau chiar de a folosi ontologia existenta in caz ca aceasta exista. Din pacate nu am gasit o ontologie in acest domeniu. Ceea ce ar putea sa ne ajute insa sunt clasificarile existente la marea majoritate a magazinelor online de piese electronice. Ceea ce vom putea observa este ca aceste clasificari 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 atributele pe care le poate avea o placa de baza in cele 3 clasificari

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

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 urmatori pentru a identifica conceptele cheie ale ontologiei

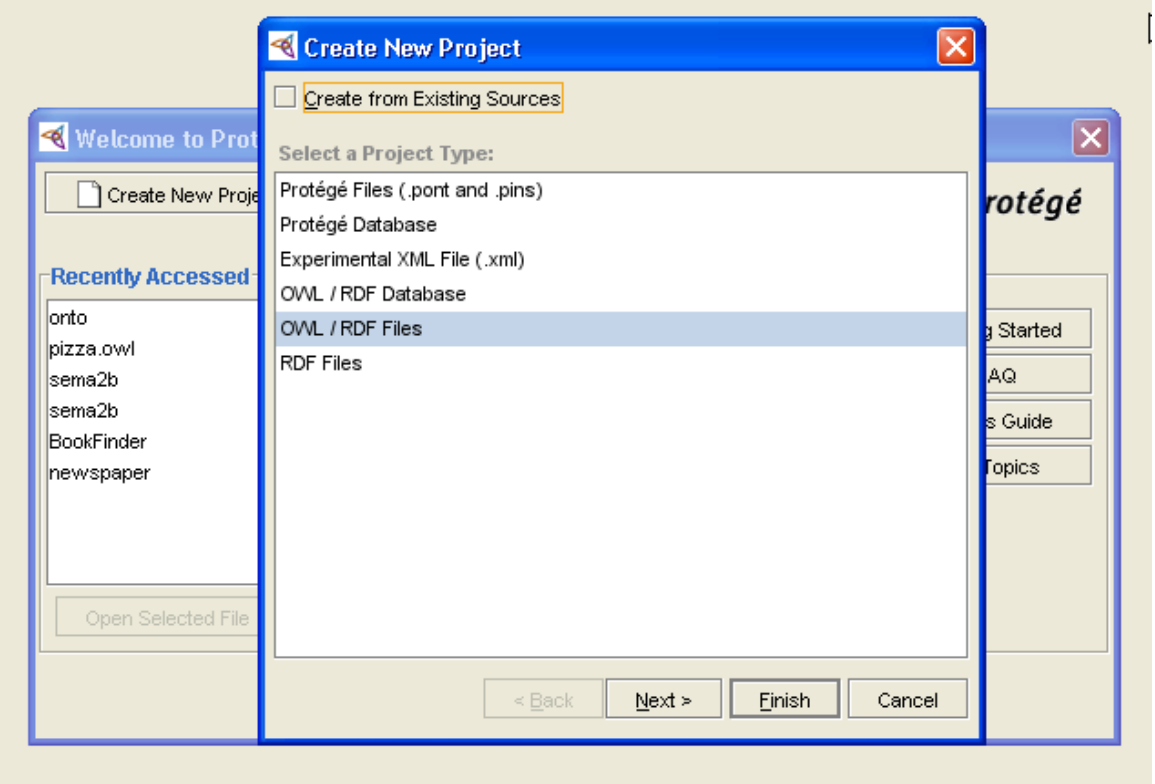
Pasul 3 presupune alcatuirea unei liste cu termenii ce vor aparea in ontologie. In alcatuirea acestei liste vom folosi o denumire care ni se pare relevanta urmand ca la pasii urmatori sa gasim si sinonimele acestora.
Boxa, Camera Web, Carcasa, Card Reader, Casca, Microfon, Cooler, Fax-Modem, Floppy Disk, 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.

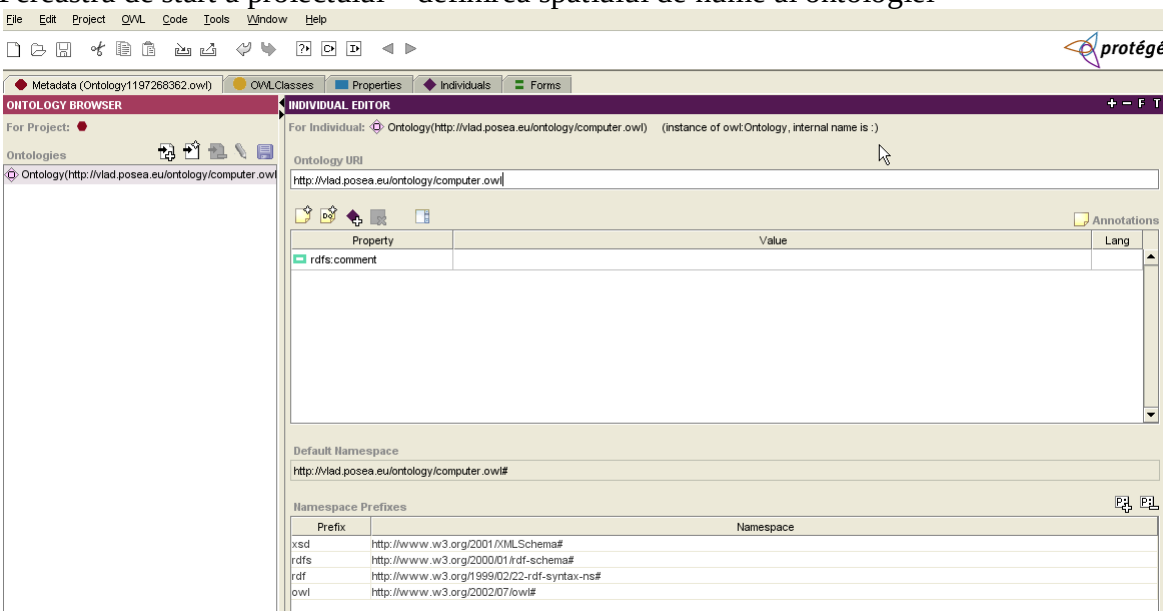
Pasul 6 presupune definirea constrangerilor pentru fiecare proprietate. Aceste constrangeri pot fi de urmatoarele tipuri:
Cardinalitate – de cate ori poate sa apara proprietatea respectiva (de exemplu o componenta are un

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”

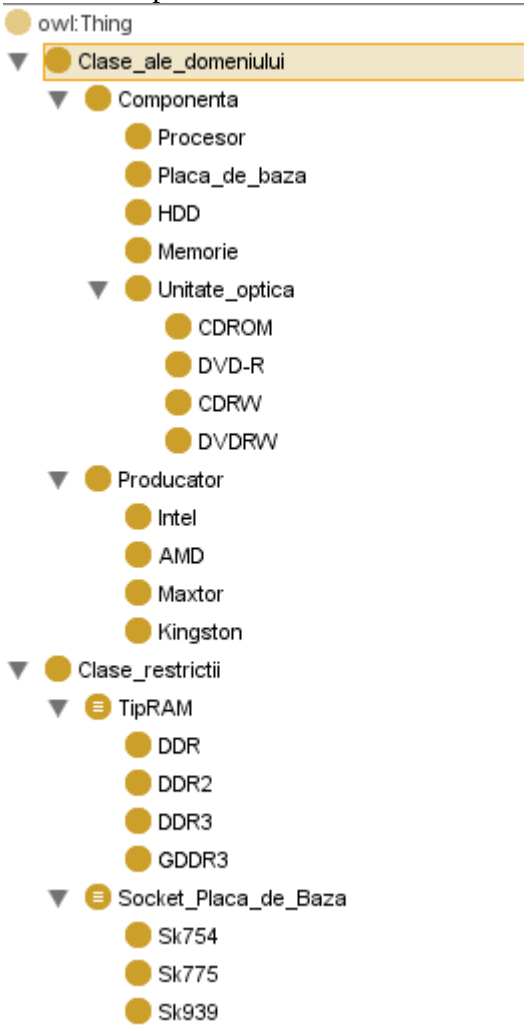
In imaginile urmatoare vom exemplifica derularea acestor pasi (4,5,6 deoarece primii 3 reprezinta colectarea termenilor) folosind Protege.
1. Creearea unui nou proiect de tip RDF/OWL



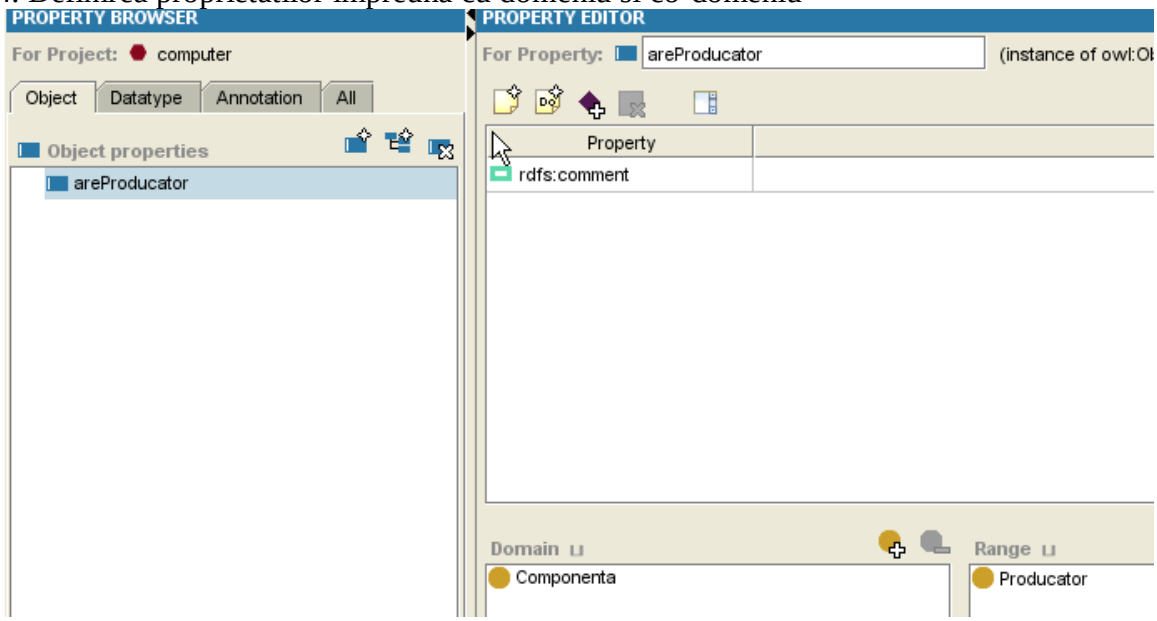
2. Fereastra de start a proiectului – definirea spatiului de nume al ontologiei



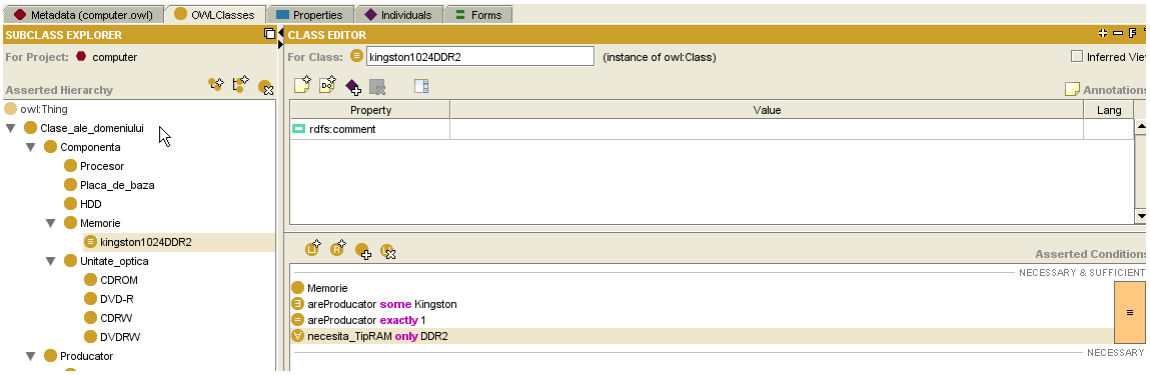
3. Constructia ierarhiei de clase (se observa ca am atasat unele clase pentru a specifica restrictii- vom vedea un pic mai incolo la ce ne folosesc)



4. Definirea proprietatilor impreuna cu domeniu si co-domeniu



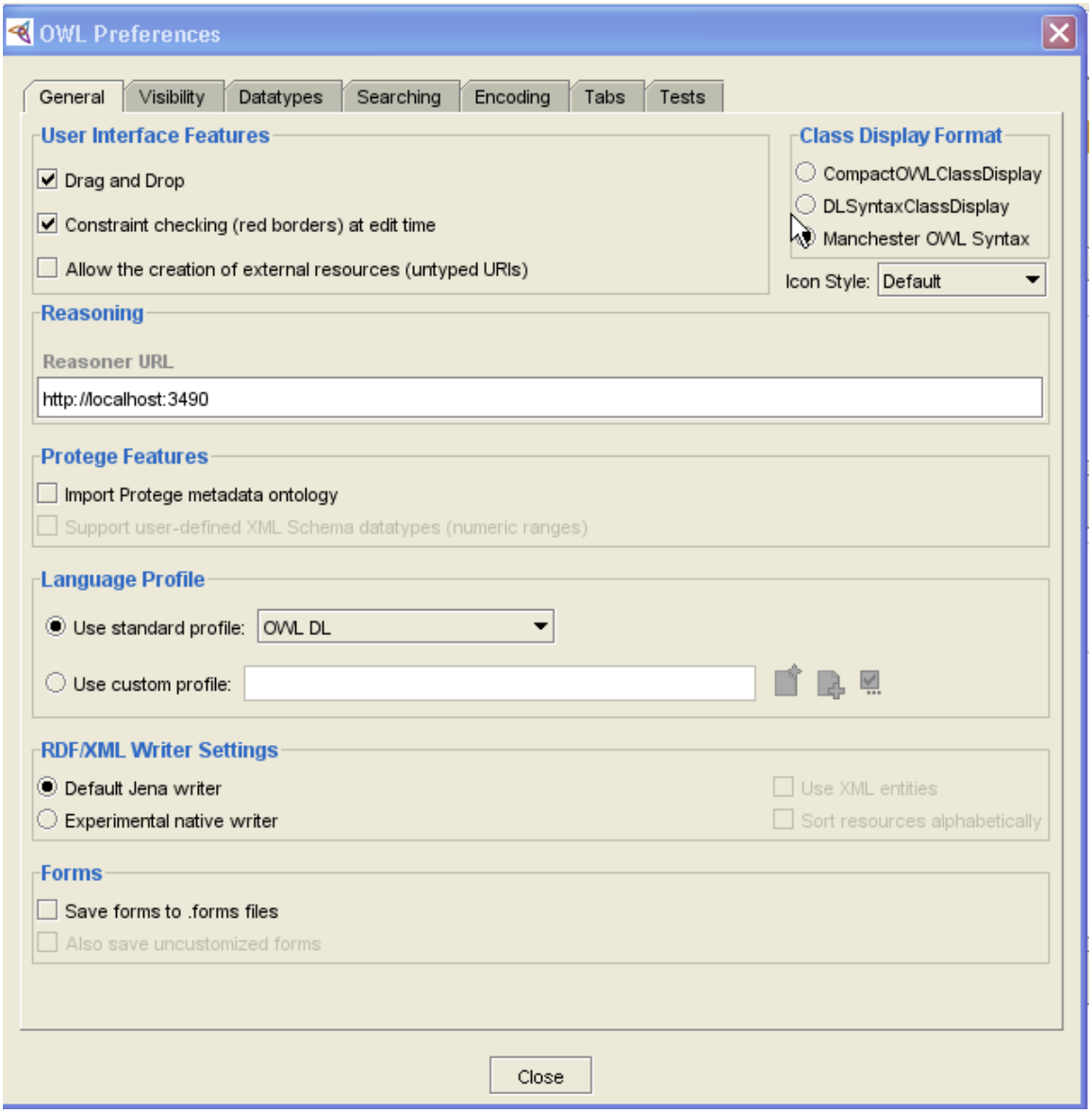
5. Definirea restrictiilor pentru o anumita proprietate



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.

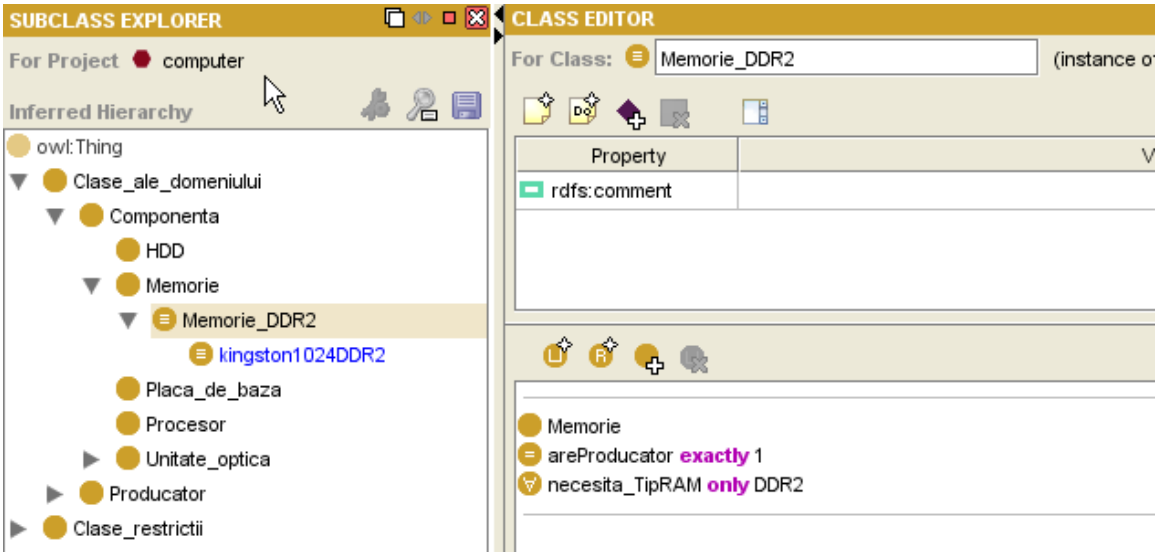
6. Configurare reasoner

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.



8. 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)

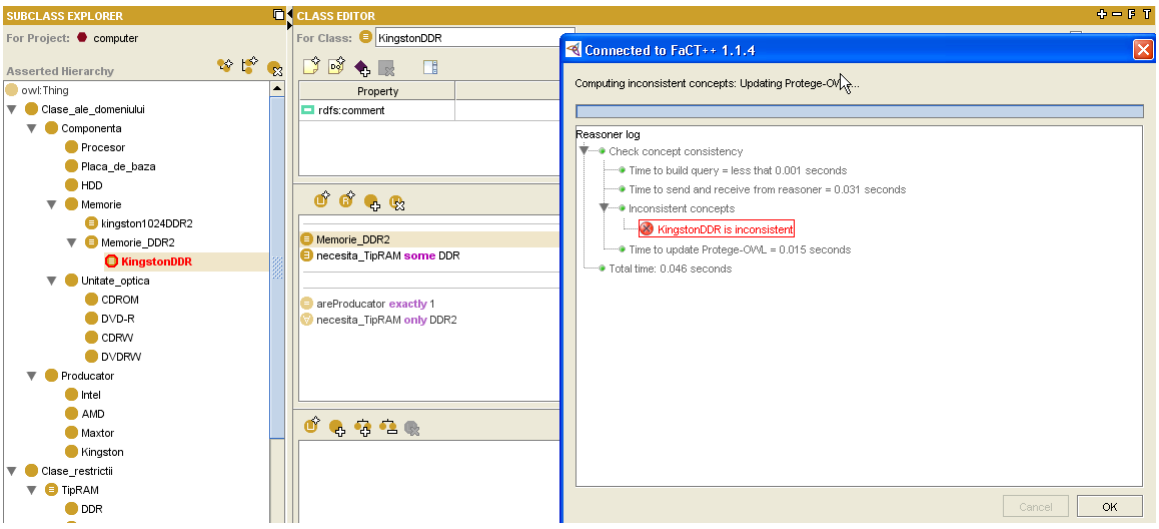
Aceste tipuri de clasificari pot fi folosite ca si interogari ale ontologiei.

9. 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.