SZAKDOLGOZAT

Bujdosó Tamás

Debrecen
2009

Debreceni Egyetem
Informatika Kar

A SZEMANTIKUS WEB ARCHITEKTÚRÁJA

Témavezető:
Mecsei Zoltán
Egyetemi tanársegéd

Készítette:
Bujdosó Tamás
Programtervező informatikus hallgató

Debrecen
2009

Tartalomjegyzék:

Bevezetés

Az internet napjaink leggyorsabban fejlődő médiuma. A folyamatos fejlődés elengedhetetlen, hiszen naponta több tízmillió felhasználó olvasgat cikkeket, kutatja az ismerőseit, chatel, fórumozik, blogol, vagy keres az interneten. A világhálón szerverek és kliensek helyezkednek el. A kliensek a szerverek által nyújtott információkhoz férnek hozzá. Az egyre növekvő igények miatt a világhálón fellelhető adatmennyiség olyan hatalmas, hogy egyre nehezebbé válik a pontos keresés. Egy friss felmérés szerint egyre hosszabbá válnak a keresőbe beírt kifejezések is. A találatok száma viszont nem csökken, sőt ezzel bekerülnek olyan találatok is, amelyeknek nem sok köze van az általunk keresett témához. A keresők nem a fellelhető dokumentumok és a keresőbe beírt kifejezés jelentésével foglalkoznak, hanem kizárólag annak szöveges alakjával, vagyis hiányzik a szemantika. Az egyik legfőbb oka ennek, hogy az információk vagy természetes nyelven vannak megadva, vagy grafikus, multimédiás formátumban. Így a gépek nem ismerik az adott oldal tartalmát, nem látnak összefüggéseket közöttük. A keresési probléma egy lehetséges megoldása lenne a szemantikus web. Ez a technológia is a www „feltalálójától” Tim Berners-Lee-től származik, és lényege, hogy metaadatokkal lássák el az internetre kikerülő információkat, ami olvashatóvá teszi őket más gépek számára, ami a különböző interakciók automatizálását teszi lehetővé. A metaadatok olyan egységesen tárolt információk, amelyek az információkról és a közöttük fennálló kapcsolatokról, asszociációkról tartalmaznak bővebb ismereteket. Mivel ezek egységesen vannak leképezve, így a különböző rendszerek képesek ezeket egymás között megosztani és feldolgozni. A szemantikus web problémakörét nem csak az teszi bonyolulttá, hogy meg kellene mondanunk, hogy milyen metaadatokat rendeljünk milyen logika szerint az adott html-oldalakhoz, hanem a másik oldalról is meg kell vizsgálni a kérdést, vagyis ki kell tudni értékelni ezeket az adatokat más szolgáltatóknak, amik a metaadataink ismeretében releváns ajánlatot szeretnének nekünk tenni. A szemantikus web megalapozásához szükség van az egyes erőforrások egyértelmű elnevezésére; egy, az adatok leírására és összekapcsolására szolgáló általános modellre; az adatok ezen modell alapján történő elérésére; egy közös szóhasználat definíciójára és valamilyen következtetési rendszerre.

Szakdolgozatom célja, hogy bemutassa a jelenkori keresési problémákat, majd kifejtem a Web továbbfejlesztéséhez szükséges – fentebb említett – technológiákat.

2. Keresés

2.1 Mi a baj a jelenlegi kereséssel?

A mai keresők legtöbbje statisztikai algoritmusok használatán alapszik. A Google forradalmi ötlete az volt, hogy az oldalak közötti kapcsolatokat figyelembe vette annak megállapításában, hogy melyek azok az oldalak, amelyek egy-egy témában véleményformálóak, meghatározóak. A statisztikai módszerek mellett egy speciális algoritmust is megvalósít, amely az egyes oldalakhoz tisztán a linkstruktúrát alapul véve rendel fontosságot. Ebben azonban feltűnik két alapvető probléma, amelynek nem igazán látszik még a teljes megoldása a keresésben. Az egyik, ha új oldalak jelennek meg. Ekkor az a gond, hogy a statisztikai alapú algoritmusok természetükből adódóan csak visszafele tudnak nézni. Ha túlságosan új egy információ, még nem volt lehetősége kiépülni az idézési struktúrának, vagyis kevés hivatkozás található rá. Ez pedig alapfeltétele lenne annak, hogy jó helyre kerüljön a sorrendezésben. A másik probléma a keresőoptimalizálás (**SEO: *S***earch ***E***ngine ***O***ptimization). Ennek az a lényege, hogy egy weboldalt a webes keresők megtaláljanak, és a találati listában minél előrébb mutassanak. Ismert tény, hogy a keresés során az első 10-20 találatnál többet nem néznek meg az emberek, pedig lehet, hogy az általuk keresett információ pont a következő találati oldalon lenne, ahova pedig már nem mennek el. Így aztán egyre fontosabb, hogy egy adott weboldal minél előrébb legyen a találati sorban. Ez a SEO segítségével érhető el, azon belül azzal, hogy a megfelelő kulcsszavakat választjuk ki az adott oldal tartalmához. Azonban egyre több cég szakosodott az optimalizálásra, így aztán az is előfordulhat, hogy egy oldal csupán azért került előrébb egy listában, mert a tulajdonosa nem sajnálta a pénz arra, hogy egy jó céget bízzon meg a SEO-ra.

Ezek után felvetődhet a kérdés, hogy mi szolgáltathat megoldást a fent említett problémákra. Először is a találatok jobb megjelenítése. Vizualizálni lehetne a találatokat, vagy éppen tartalmi csoportokba, úgynevezett clusterekbe rendezni azokat. Ez megoldaná azt a problémát, hogy kevés találati oldalt nézünk át. Ez a lehetőség azonban eddig még nem terjedt el széles körben, véleményem szerint nem is fog, bár a napokban mutattak be egy ilyen keresőt, a Tianamo. Ez egy olyan kereső, amely 3D-s térképszerűen jeleníti meg a találatokat. A fent említett clusterekbe rendezi a találatokat, és kisebb-nagyobb hegyeket, dombokat jelenít meg, a találat számainak

megfelelően. Ez valóban forradalmi megoldás, azt viszont nehezen tudom elképzelni, hogy a Google-hoz vagy a Yahoo-hoz szokott nép képes lenne egy ilyen nagymértékű változtatásra a keresési szokásaiban. Egy másik lehetőség lenne, ha felhasználói interakciók segítségével osztályoznák az egyes lapokat, így azok a lapok előtérbe kerülnének, amelyek több szavazatot kapnak. Úgy gondolom, ez is elég nehezen kivitelezhető módszer, hiszen itt is azok az oldalak kerülnének előtérbe, amelyek esetleg ismertebbek, többen látogatnak, de nem pont azt az információt tartalmazzák, amelyre nekünk aktuálisan szükségünk lenne. A harmadik lehetőség a szemantikus világháló. Már megjelentek a kifejezetten erre megírt keresők, és a jelenlegi nagy keresők továbbfejlesztése is ez irányban történik.

A hagyományos keresés legfőbb problémája az, hogy nem a keresőbe beírt kifejezés és a világhálón fellelhető dokumentumok jelentésével, foglalkoznak, hanem csak azok szöveges alakjával, tehát hiányzik a szemantika. Az ebből a hiányból adódó legfőbb problémákat három kategóriába sorolhatjuk. Az első ezek közül a nyelvi problémák. Előfordulhat, hogy a számunkra fontos információt valamilyen idegen nyelvű oldalon találnánk meg. Így meg kellene oldani, hogy ne legyen lényeges, hogy milyen nyelven és milyen módon van letárolva az adott dokumentum. A másik probléma, ha képet, vagy valamilyen multimédiás tartalmat keresünk. Ilyenkor az jelent gondot, hogy az általuk tárolt információkat nem nagyon lehet automatikusan kinyerni. Képek esetében akkor van szerencsénk, ha maga a kép tartalmaz bizonyos szövegeket, feliratokat. A legtöbbször azonban a képen nincs semmilyen kézzelfogható, kinyerhető szöveg. Egy harmadik probléma a következtetés hiánya. Bizonyos esetekben szükségünk lehet arra, hogy a felhalmozott ismeretekből következtetni tudjunk. Tisztában kell lennünk az egyes fogalmak jelentésével és a fogalmak közti kapcsolatokkal, sőt valamilyen módon tudni kell azt is, hogy mit tartalmaznak a nem szöveges dokumentumok.

2.2 Mi lehet a megoldás?

A hatalmas mennyiségű adat és az egyes oldalak – például a hírportálok - folyamatos információváltozása miatt egy keresőrendszer nem tudja teljes mértékben lefedni a teljes tartalmat. Előfordulhat azonban, hogy amikor nem találunk meg valamit egy keresőrendszerben, akkor átnavigálunk egy másik keresőre, ahol beírva ugyanazt a kifejezést megtaláljuk azt, amire szükségünk van. Ezt a logikát követik az úgynevezett metakeresők is. Ezek más keresőgépekkel

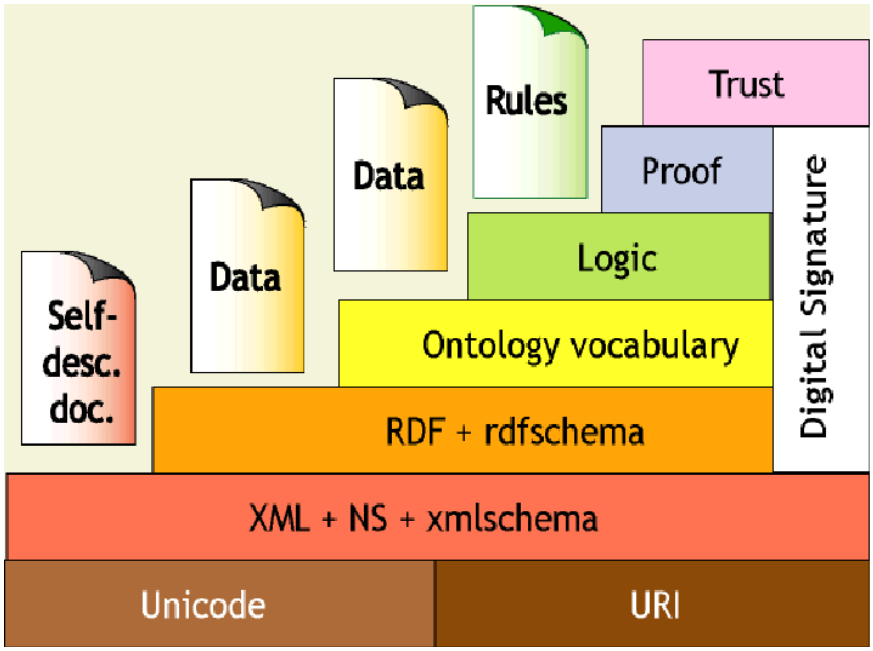
kereső keresőgépek. Működésük lényege, hogy a keresőbe beírt kifejezést elküldik több más keresőrendszereknek, és az ezekből visszakapott válaszokat megszűrve mutatja nekünk a találatokat.

Léteznek úgynevezett kérdés-átalakító keresők is. A kérdés-átalakítás célja, hogy jobban megértsük a felhasználó által feltett kérdést, és a teljes szöveges keresésen túlmutató keresést végezzünk a rendelkezésre álló dokumentumok között. A feltett keresőkérdést először nyelvileg elemezzük, majd ezen elemzés eredményét felhasználva készítünk egy másik, olyan kérdést, amely reményeink szerint jobb, hasznosabb találatokkal szolgál, mint az eredeti. A kérdés ilyetén átalakításához szükségünk van némi háttértudásra, ami általában valamilyen matematikai formalizmussal határozható meg.

A legtöbb figyelmet kapó megoldás jelenleg az, hogy az egyes oldalakat metainformációkkal lássuk el. Ez egy univerzális eszköz lenne arra, hogy jelentést társítsunk a webes tartalomhoz. Metainformáció lehet például egy oldal fontossága, az oldalban található kifejezések gyakorisága, vagy annak helye. Ezen információk használata nem új keletű dolog: a Word dokumentumok is tartalmazzák már többek között a készítő nevét, az utolsó módosítás dátumát,…stb. Ugyanez figyelhető meg, ha egy képfájlt nyitunk meg karakteres módban. Az első néhány sor itt is a képről tartalmaz bizonyos plusz információkat, például a fájl típusát, a készüléket, amivel a kép készült, az utolsó módosítás, vagy éppen a készítés dátumát. Sőt, már a weben is megjelentek ezek az adatok, a META nevű HTML címke is kifejezetten erre a célra szolgál. A szemantikus web elgondolása is az ilyen adatok használatán alapszik; egyrészt azok erőforrásokhoz való kapcsolásán, másrészt azon, hogy ezen metaadatok segítségével következtetni is kell tudni az adott dokumentum tartalmára. Ha egy-egy erőforrás jellemzőit meg tudjuk adni úgy, hogy az a keresők számára feltérképezhető legyen, óriási lépést tehetnénk az intelligensebb kereshetőség felé. Ehhez azonban arra lenne szükség, hogy ezeket a metaadatokat egységesen szolgáltassák magukról az információforrások.

3. A szemantikus web rétegei

Amikor Berners-Lee 2000-ben a világ elé tárta elképzeléseit a szemantikus webről, a következő ábrát hozta, amely azóta is a fejlesztési folyamat alapjának tekinthető(1. ábra). Ebben a fejezetben végigveszem az egyes rétegeket, valamint kifejtem, hogy azok miként szolgálják a szemantikus web célkitűzéseit.



1. ábra: A szemantikus web rétegei

(Tim Berners-Lee, 2000, XML 2000 konferencia)

3.1 Unicode

A Unicode és a vele párhuzamos ISO szabvány a legmodernebb karakterkódolást testesítik meg. Szakít az egy karakter-egy byte megoldással, ehelyett a kezelt karaktereket kódegységek sorozatának tekinti, amely kódegységek korlátozott hosszúságú számok. Ezt a számsorozatot végül byte-ok sorozataként tárolja, illetve ábrázolja. Így felépíthető egy univerzális karakterkészlet, amely több módon is kódolható, illetve a kódból visszaállítható. A Unicode

igyekszik magába foglalni a világ nyelveiben található összes előforduló karaktert, lehetővé téve a dokumentumok egész világon történő terjesztését.

3.2 URI

Az URI(***U***niform/***U***niversal ***R***esource ***I***dentifier) az egységes erőforrás azonosítót takarja. Az URI-t két részre lehet bontani: az URL-re(***U***niform/***U***niversal ***R***esource ***L***ocator), és az URN-re(***U***niform/***U***niversal ***R***esource ***N***ame). Előbbi az elérési móddal azonosítja az erőforrást, míg az utóbbi egy névvel látja el azt. Azért van szükség mindkettőre, mert az internet fejlődésével felmerült az igény, hogy ne csak a hálózati erőforrásoknak legyen azonosítója – erre szolgál az URL -, hanem más, absztrakt erőforrást is fel tudjunk ruházni valamilyen azonosítóval – ez lesz az URN -. Az URN bevezetésének gondolata azon alapszik, hogy egy URL nem állandó, sokszor változhat, ami néhány alapvető problémához vezet. Például, ha megváltozik egy URL, akkor a hivatkozásváltozások karbantartása nagy erőfeszítéseket igényel. Továbbá a változásokról mindenkit értesíteni is kell, aki hivatkozik arra a forrásra. Többek között ezekre jelent megoldást, ha elérési helytől független, egyedi azonosítót használunk. Ahhoz azonban, hogy a dokumentumot elérjük, természetesen annak lelőhelyére is szükség van, ám az adott azonosítóhoz tartozó tényleges lelőhelyet elegendő csupán egy központi helyen megadni és frissíteni, ahonnan aztán az azonosító alapján történő kereséskor ez kiolvasható. Az URI-kat hivatkozás szerint két csoportba lehet osztani: vannak abszolút és relatív hivatkozású azonosítók. Az abszolút hivatkozás lényege, hogy a saját környezetétől függetlenül hivatkozik egy erőforrásra. Ezzel szemben, ha a relatív változatot alkalmazzuk, akkor szükségünk lesz egy úgynevezett bázis URI-ra. Az, hogy mi ez a bázis URI, több dologtól is függhet, szabvány írja le ezeket. A bázis URI segítségével a relatív URI-kat egy algoritmussal feloldjuk, és abszolút azonosítókat készítünk belőlük. A relatív URI-k hasznosak, mert általuk egy összetettebb honlap különböző dokumentumai is hivatkozhatnak egymásra anélkül, hogy pontosan tudnák, hogy mely szerveren futnak éppen. Az URI-knak alapvető szerepük van a szemantikus web elképzelésben, hiszen segítségükkel egyértelmű állításokat fogalmazhatunk meg bármiről, mivel egyértelműen azonosítanak egy-egy erőforrást. Bárhol is legyen a világhálón két metaadat-leírás, ha ugyanazt az URI-t használja, akkor ugyanarról az erőforrásról jelent ki valamit az a leírás.

3.3 XML

Az XML (E***X***tensible ***M***arkup ***L***anguage) a kiterjeszthető jelölő nyelv rövidítése. Tehát ez egy leíró- vagy más néven meta-nyelv, melynek segítségével új nyelveket írhatunk le. Ha az XML segítségével megalkotunk egy új nyelvet, akkor azt az XML alkalmazásának nevezzük. Egy ilyen nyelv egy példányát XML dokumentumnak hívjuk. Egy ilyen dokumentum egy szöveges állománynak tekinthető, amely strukturált formában képes adatokat tárolni. Ebből is látszik, hogy egy XML dokumentum „nem csinál semmit", vagyis, mint bármely más szöveges állomány, adatokat tárol. Az azt feldolgozó alkalmazásnak kell eldöntenie, hogy mit fog majd vele kezdeni. Azonban alkalmas arra, hogy egy általános, az egymással összedolgozni szándékozó rendszerek közös adatcsere formátuma legyen, ezzel elősegítve a gépek közötti kommunikációt. Egy dokumentum legfőbb részei az elemek és az attribútumok. Egy elem három részből áll: egy nyitó címkéből (tag), magából az adatból, és a záró címkétől, ami ugyanaz, mint a nyitócímke azzal a különbséggel, hogy egy '/'-t írunk elé. Általában egy elemre annak nyitócímkéjével hivatkozunk. Az XML-ben nincsenek úgynevezett fenntartott szavak, egy elem neve – bizonyos elemi szabályokat figyelembe véve - tetszőleges karaktersorozat lehet. Az alábbiakban felírok egy nagyon egyszerű, CD nyilvántartó XML dokumentumot:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
 <nyilvan>
   <CD>
     <eloado>Bon Jovi</eloado>
     <cim>It's my life</cim>
     <ev>2001</ev>
   </CD>
   <CD>
     <eloado>Tina Turner</eloado>
     <cim>All the best</cim>
     <ev>2004</ev>
   </CD>
 </nyilvan>
```

A dokumentum egy fát határoz meg, így az elemeknek egymáshoz való hierarchikus viszonya is rögzítve van. Példám gyökéreleme a <nyilvan> címke, az <eloado>, <cim>, <ev> gyermekelemek, illetve egymásnak testvérei. Tartalma alapján egy elem lehet egyszerű, összetett vagy üres. Ez utóbbi csak nyitó-illetve záró címkéből áll. Az elemek tetszőleges számú

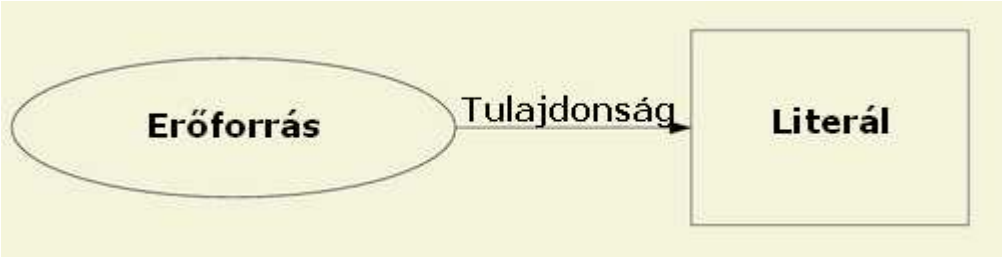
attribútummal rendelkezhetnek. Ha egy elemnek van attribútuma, akkor azt összetett XML elemnek nevezzük. Az attribútumok értékei csak egyszerű típusok lehetnek. Az XML dokumentumokban használt elem-és attribútum nevek egyediek az adott dokumentumra nézve, vagyis lokális nevek. A nevek használatából adódhat néhány probléma. Mi van akkor, ha több XML dokumentumból csinálunk egyet, és esetleg mindkettő tartalmazza ugyanazt a nevet? Vagy mi történik akkor, ha egy HTML-be ágyazott XML dokumentum is használ bizonyos HTML elemneveket? Ilyen esetekben bizony névütközés léphet fel. Ennek elkerülésére szolgálnak a névterek (***N***ame ***S***pace). A névterek alapötlete, hogy az elem-és attribútumneveket valamilyen prefixes alakban adja meg. Az így kiegészített nevet univerzális névnek hívjuk, amelyről már feltételezhetjük, hogy egyedi lesz, kiküszöbölve ezzel a névütközés problémáját. A névtereknek hatáskörük van, egy XML névtér az azt definiáló elemen belül látható. Új névtér definiálására két XML-attribútum szolgál: az `xmlns:prefix` illetve az `xmlns`. Az első alak egy *prefix* azonosítójú névtér definiálására való, míg a második alapértelmezett névtér megadását teszi lehetővé. A prefixszel nem rendelkező nevek automatikusan az utóbbi csoportba kerülnek. Egy dokumentumot jól formázott XML dokumentumnak nevezünk, ha betartja a formai követelményeket, vagyis, minden nyitó címkének megvan a maga záró címkéje; illetve az attribútumok kulcs-érték alakban vannak megadva. Az XML egyik legfőbb erénye, hogy ellenőrizni tudjuk, hogy egy dokumentum példánya-e egy adott nyelvnek. Erre szolgálnak az úgynevezett XML sémák (XMLShema). Egy ilyen séma leírja egy XML dokumentumban felhasználható elemeket és attribútumokat, azok tartalmát, valamint az elemek egymáshoz való viszonyát. Amennyiben adott egy séma és egy dokumentum, ezekből képesek leszünk eldönteni, hogy a dokumentum megfelel-e a sémának. Létezik az úgynevezett XSL (E***X***tensive ***S***tylesheet ***L***anguage Family), a kiterjeszthető stíluslap nyelvcsalád. Ezen elgondolás onnan ered, miszerint érdemes lenne különválasztani az XML-ben tárolt adatokat azok megjelenítésétől. Így a feldolgozó alkalmazásnak nem kell azzal törődnie, hogy a megjelenítés ténylegesen milyen eszközön és milyen módon történik majd. Amikor a megjelenítés megtörténik, egy transzformáció fut le, ami előállítja majd a megjelenítési alakot. Köztudott, hogy a web alapja a HTML. Felvetődik a kérdés, hogy XML-ben lehet-e HTML-t írni. A válasz: igen, és az így keletkezett alkalmazást XHTML-nek keresztelték el. Ez ötvözi a két nyelvet, így számos előnnyel bír. Mivel egy ilyen dokumentum XML dokumentum is egyben, így lehetőség van arra,

hogy más, XML alapú nyelvből átvegyünk bizonyos elemeket. Látható azonban, hogy az XML csak az adatok feldolgozását segíti elő, a szemantika még mindig hiányzik. Még mindig nem sikerült elérnünk azt, hogy alkalmazások más alkalmazásokkal kommunikáljanak anélkül, hogy előtte ne kelljen egyeztetni az átvitt információ jelentését. Ennek a kezelésére jött létre az RDF (***R***esource ***D***escription ***F***ramework), az erőforrás-leíró keretrendszer.

3.4 RDF

Láthattuk, hogy az XML-lel már elértük azt, hogy a gépek számára feldolgozható legyen az információ. Most szeretnénk elérni azt, hogy ez érthető is legyen. Erre szolgál az RDF. Az alapkoncepció az, hogy erőforrásokról adunk meg metainformációkat, majd a metainformációk kapcsolatrendszerét írjuk le, mindkettőt XML alakban. Az RDF egy nyelv, amely arra alkalmas, hogy az erőforrásokhoz metaadatokat kapcsoljon. Erőforrás minden, ami URI-val rendelkezik, így erőforrás lehet egy weblap, annak egy része, egy kép, vagy egy tetszőleges állomány. Azonban egy erőforrás nem feltétlenül kötődik a webhez, bárminek lehet URI-ja. A metaadatok leírhatósága érdekében az RDF definiál egy halmazelméleti alapokon nyugvó adatmodellt, amit nem meglepő módon RDF adatmodellnek neveztek el. A modellben több halmazt is definiáltak. Az első ilyen halmaz az erőforrások halmaza. Erőforrás minden olyan dolog, amelyet egy RDF kijelentés leírhat. A következő halmaz a tulajdonságok halmaza. Ezek maguk is RDF erőforrások, tehát URI-val azonosíthatók, valamint erőforrásokhoz kapcsolható jellemző karakterisztikákként definiálhatók. Harmadik halmaznak tekinthető a literálok halmaza, melynek elemei literálok, karaktersorozatok. Végül meg kell adni a kijelentéseket, amelynek elemei olyan elemhármasok, amiket kijelentéseknek tekintünk. Ezek alanyból, állítmányból és tárgyból épülnek fel, amelyben az alany egy tetszőleges erőforrás, az állítmány egy tetszőleges tulajdonság, a tárgy pedig szintén egy tetszőleges erőforrás, vagy egy literál. Az adatmodell jelentését pedig úgy definiálhatjuk, hogy a kijelentések halmazába tartozó elemhármasok igazak.

Így egy RDF kijelentés nem más, mint három összetartozó, URI-val azonosított erőforrás, vagy két erőforrás és egy literál. Ezeket megadhatjuk elemhármasokkal; címkézett, körmentes gráffal; vagy XML formátumban. Elméleti szempontból a gráf reprezentáció az elsődleges, a gépi feldolgozás miatt azonban az XML-ben történő leírás a legelterjedtebb.



2. ábra: Egy egyszerű RDF gráf felépítése

Létezik olyan megoldás, hogy bevezetünk egy köztes, úgynevezett névtelen erőforrást. Egy ilyen erőforráshoz nem tartozik URI, így nem azonosítható. A gráfban egy üres csomópontként jelenítik meg. Ezt olyankor vezetik be, ha a meglevő információ struktúráját növelni szeretnék. Például, ha egy összetett adatot szeretnénk tárolni – mint a teljes lakcímünk – akkor érdemes ilyen köztes csomópontokat létrehozni ahelyett, hogy egy helyen tárolnánk, hosszú literálként. Egy gráfban sosem lehet címke nélküli él. Felmerülhet azonban bennünk, hogy ezt mégis hogy tudnánk elemhármasokként, XML-alakban felírni. Erre szolgál a linearizálás. Ahhoz viszont, hogy ezt el tudjuk végezni, mégis csak el kellene neveznünk ezt a csomópontot úgy, hogy az egyértelműen megkülönböztethető legyen az URI-któl és a literáloktól. Ha az RDF-et XML alakban szeretnénk megadni, azt az RDF gráf linearizálásaként tehetjük meg. A fenti általános gráf XML-szintaxisa:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:s="http://www.example.org/terms/">
<rdf:Description rdf:about="Erőforrás">
<s:állítmány>Tárgy</s:állítmány>
</rdf:Description>
</rdf:RDF>
```

Látható, hogy ez egy valódi XML dokumentum, amely speciális nyitóelemeket, és névtereket tartalmaz. Nézzük végig sorról-sorra, mi mit jelent a fenti leírásban! Az első sor arról tájékoztat minket, hogy egy XML dokumentumot látunk, és hogy annak melyik verziójával készült. A

második sorból kiolvashatjuk, hogy ez az XML RDF leírást tartalmaz, ami a 8. sorban levő záró címkéig tart. A következő sor a névtér deklarációs rész. Ez meghatározza, hogy az összes címke a harmadik sorban megadott webcímen megadott névtér része. A negyedik sor is egy névtér deklaráció, amely az xmlns: után levő prefixet rendeli hozzá a megadott URI-val azonosított névtérhez. Mivel mindkét névteret a dokumentum gyökerében deklaráltuk, az XML specifikációnak megfelelően az egész dokumentumon belül érvényesek. Az eddig felsorolt dolgok mindig szükségesek, ha egy XML/RDF dokumentumot akarunk megadni. Az 5-7. sor a specifikus rész, lényegében ezek adják meg az RDF gráf által reprezentált kifejezést A Description elemmel írhatjuk le az RDF kijelentés alanyát. Ennek a gyermek elemei, a tulajdonság elemek határozzák meg a kijelentés predikátumát, és tárgyát. Az RDF XML szintaxisa az XML minősített neveit használja az RDF gráf élein található URI-k reprezentálására.

Lehetőségünk van arra is, hogy erőforrás-erőforrás kapcsolatot írjunk le. Ez akkor lehet hasznos, ha egy személynek, akinek az adatait már leírtuk RDF/XML alakban, szeretnénk megadni például az e-mail címét, ami ugyebár szintén egy URI-val azonosított erőforrás és nem egy literál. Ezt az rdf:resource attribútum bevezetésével valósíthatjuk meg. Példaként lássuk néhány adatomat:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:bt="http://www.utils.org/utils#">
  <rdf:Description rdf:about="http://hosted.filefront.com/bujdosot">
    <bt:nev>Bujdosó Tamás</bt:nev>
   </rdf:Description>
<rdf:Description rdf:about="http://hosted.filefront.com/bujdosot ">
     <bt:egyetem>Debreceni Egyetem</bt:egyetem>
   </rdf:Description>
<rdf:Description rdf:about="http://hosted.filefront.com/bujdosot ">
    <bt:emailcim rdf:resource=bujdosot@freemail.hu/>
   </rdf:Description>
```

```
</rdf:RDF>
```

RDF-ben lehetőség van konténerek vagy kollekciók megadására, így az erőforrásokat csoportokba foglalhatjuk, és ezekről a csoportokról állításokat fogalmazhatunk meg. A konténerek beépített RDF osztályok. A keretrendszer alábbi három konténertípust definiálja:

- Halmaz(Bag): Erőforrások és literálok olyan csoportja, ahol egy elem többször is előfordulhat, az elemek sorrendje nem lényeges. rdf:Bag-gel definiáljuk. Itt jegyezném meg, hogy az RDF nem a matematikai halmazfogalomnak megfelelően használja ezt a típust.
- Szekvencia(Sequence): Egy rendezett halmaznak fogható fel, vagyis egy az erőforrások és literálok olyan csoportjának, amelyben azonos elemek többször is előfordulhatnak, de azok sorrendje lényeges. rdf:Seq-kel deklaráljuk.
- Alternatíva(Alternative): Erőforrások vagy literálok nem rendezett csoportja, amelyben egy elem többször is előfordulhat, és az elemek esetlegesen felcserélhetőek egymással. rdf:Alt-tal adhatjuk meg. Lényeges különbség az előző két konténertípushoz képest, hogy ez a típus nem lehet üres, legalább egy elemet tartalmaznia kell, amely az alapértelmezett elem lesz.

Egy erőforrásról az rdf:type tulajdonsággal jelenthetjük ki azt, hogy ez eleme a fentebb említett konténerek valamelyikének. Egy konténer elemeit speciális tulajdonságokkal írhatjuk le. Ennek alakja rdf:_n, ahol n egy tízes számrendszerbeli természetes szám. Lehetőségünk van a nyelvben olyan csoportokat megadni, amelyek zártak, így csak az előre megadott elemeket tartalmazzák. Ezeket nevezzük RDF-kollekcióknak. Egy kollekció az rdf:List osztály egy példánya. A kollekció első elemét az rdf:first tulajdonsággal adhatjuk meg, míg a többi részére rdf:rest néven tudunk hivatkozni. A kollekció végét az rdf:nil erőforrás jelzi. Ezek után lássuk, milyen adattípusokat adhatunk meg az RDF-en belül! A literáloknak két típusát különböztethetjük meg: a közönséges literálokat, amelyek tartalomtól függetlenül mindig csak egy karaktersorozatot reprezentálnak; illetve a típusos literálokat, amik arra hivatottak, hogy például egy számot számként kezeljenek, ne pedig egymás mellé írt karakterekként. Sajnos a keretrendszer nem tartalmaz beépített típusokat, viszont hozzárendelhetünk az egyes literálokhoz egy-egy URI-t, amely azon típusát azonosítja. Ezt az rdf:datatype attribútum megadása segíti

elő. Ha a fent említett példámat ki szeretnénk egészíteni az életkorommal, akkor a következő sorokat kellene hozzáadnunk, amely már egész számként fogja értelmezni a megadott értéket:

```
<rdf:Description rdf:about=" http://hosted.filefront.com/bujdosot ">
  <bt:eletkor rdf:datatype="Int">23</bt:eletkor>
</rdf:Description>
```

Persze ehhez tudnunk kell azt is, hogy hol találunk olyan URI-t, amely a kívánt adattípust definiálja. Szerencsére az XML séma definiál ilyeneket. Egy XML sémával megköthetjük, hogy egy adott XML elem tartalma milyen típusú legyen. Tehát egy ilyen definiált adattípusra a http://www.w3.org/2001/XMLShema URI-val hivatkozhatunk úgy, hogy ennek a végére írjuk egy # után a kívánt adattípust.

Az RDF elgondolás még viszonylag új, így aztán felmerülnek még bizonyos aggályok a használatával kapcsolatban. Az egyik ezek közül, hogy semmi se ösztönzi az RDF leírást készítő személyeket, hogy valós metainformációkat szolgáltassanak a weblapok tartalmáról. Hiszen már régóta előfordul, hogy a keresőben elért előkelőbb helyezések miatt hazudnak az oldalak tartalmáról. Ezek kiküszöbölésére fejlesztettek ki néhány olyan technikát, amelyek automatikusan nyerik ki a metaadatokat, ezzel kiváltják a dokumentumok kézi bevitelét, és esetlegesen kizárják a szándékos pontatlanságokat. Mivel a szemantikus web egyik építőköve a következtetés, ezért ahhoz, hogy ezt helyesen el tudjuk végezni, szükséges az, hogy a megadott adatok pontosak legyenek. Ahhoz azonban, hogy következtetéseket tudjunk levonni, szükség van bizonyos háttértudásra is. Az eddig bemutatott technikák csak az információk leírására szolgáltak. Fontos lehet számunkra, hogy képesek legyünk saját osztály vagy tulajdonságot definiálni, esetleg ezek jellemzőit megadni. Ezt szorgalmazza az úgynevezett RDF-séma. Ez tulajdonképpen nem más, mint néhány további erőforrás meghatározása, amely beépül az RDF által eddig használt szótárába. Ezeket az erőforrásokat ugyanúgy felhasználhatjuk a kijelentéseink leírásában, mint az eddigiekben. Egy erőforrásról az rdf:type tulajdonsággal lehet kijelenteni azt, hogy ez egy osztály. Ennek értékeként az rdfs:Class értéket kell rendelni. Fontos megjegyezni, hogy az RDF a - programozási nyelvekkel ellentétben – nem hoz létre új osztályt. Mindössze annyi történik, hogy egy erőforrásról kijelentjük, hogy az egy osztály. Egy osztálynak létezhet

alosztálya is, ezt az rdfs:subClassOf attribútum megadásával jelezhetjük. Ez a tulajdonság tranzitív, tehát ha egy A osztály alosztálya B-nek, és B alosztálya C-nek, akkor A alosztálya lesz C-nek is. Lehetőségünk van egy osztályhoz, vagy valamely erőforráshoz rövid, szöveges magyarázatot fűzni. Erre a rdfs:comment szolgál. Egy sémában felhasználhatunk egy másik sémában definiált osztályt is, ezáltal kapcsolatot teremtünk a két séma közt.

Az RDF tulajdonság az rdf:Property osztály egy példánya, így egy erőforrásról példányosítással fejezhetjük ki, hogy az egy tulajdonság. Itt is lehetőség van egy bizonyos hierarchia megadására. Ezt az rdfs:subPropertyOf tulajdonsággal írhatjuk le, és természetesen ez is tranzitív. A definiált tulajdonságokat az RDF forrásokban kijelentések predikátumaként szerepeltethetjük. Lássunk egy újabb példát:

```
  <?xml version="1.0" encoding="ISO-8859-2"?>
<rdf:RDF
   xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-shema#”
   xmlns:s="http://www.utils.org/utils#">

   <rdf:Description rdf:ID="Jármű">
     <rdf:type rdf:resource= http://www.w3.org/2000/01/rdf-shema#Class/>
    </rdf:Description>
      </rdf:RDF>

  <?xml version="1.0" encoding="ISO-8859-2"?>
<rdf:RDF
   xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-shema#”
   xmlns:s="http://www.utils.org/util#">

   <rdfs:Class rdf:ID="Autó">
     <rdfs:subClassOf rdf:resource=”http://www.utils.org#Jármű/>
    </rdfs:Class>
    <rdfs:Class rdf:ID="Motor">
```

```
    <rdfs:subClassOf rdf:resource=”http://www.utils.org#Jármű/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="Hajó">
    <rdfs:subClassOf rdf:resource=”http://www.utils.org#Jármű/>
    <rdfs:comment>Vízi jármű</rdfs:comment>
  </rdfs:Class>
</rdf:RDF>
```

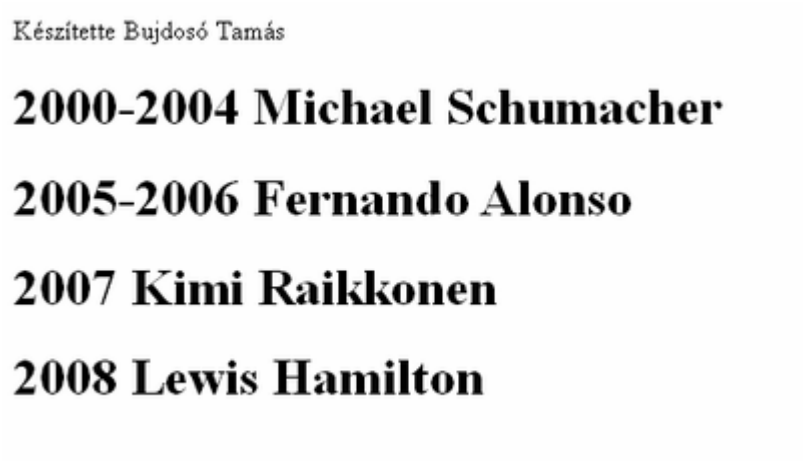
Az első kódban definiáltam a járművek osztályát, majd a második kódban felhasználtam a már meglévő osztályt, és létrehoztam három hozzá köthető alosztályt. A magyarázatok bemutatása céljából a harmadik alosztályba tettem egy comment-sort.

Ezek után nézzük meg, hogyan lehet ezeket az RDF-leírásokat feltenni a világhálóra annak érdekében, hogy a keresők is megtalálják. Az igazság az, hogy az RDF nem is kötődik minden esetben a webhez, hiszen használható független adatbázisokon elvégezhető keresésre, vagy következtetésre. Természetesen azonban a legnagyobb jelentősége az internet világában van, hiszen ez a szemantikus web egyik alapja. Nézzük tehát végig, hogy hogyan lehet egy weboldalhoz társítani az RDF-beli adatokat! Az adatokat egy HTML-dokumentum több helyére is beilleszthetjük. Az ideális mégis az oldalak feje, vagyis a <HEAD> és a </HEAD> közötti rész. Azért ez a legjobb megoldás az elhelyezésre, mert az oldalnak ezt a részét biztos, hogy beolvassák a keresők, mivel a HTML is tartalmaz alapszintű metainformációkat, amik szintén itt vannak elhelyezve. Azonban, most is szembesülnünk kell néhány problémával. Először is, ha megnyitjuk az így elkészített HTML-oldalt egy böngészővel, azt tapasztaljuk, hogy az RDF-ben definiált információk is kiíródnak. Ennek oka, hogy a legtöbb böngésző megjeleníti az XML-elemek tartalmát.

```
<html>
 <head>
  <title>A Forma-1 világbajnokai 2000-töl</title>
  <META http-equiv=Content-Language content=hu>
  <META http-equiv=Content-Type content="text/html; charset=iso-8859-2">
    <rdf:RDF
 xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
          xmlns:bt="http://www.utils.org/utils#”
        <rdf:Description rdf:about="">
         <bt:keszito>
```

```
            <rdf:li>Készítette Bujdosó Tamás</rdf:li>
        </bt:keszito>
    </rdf:RDF>
</head>
<body>
        <h1>2000-2004 Michael Schumacher</h1>
        <h2>2005-2006 Fernando Alonso</h2>
        <h3>2007 Kimi Raikkonen</h3>
        <h4>2008 Lewis Hamilton</h4>
</body>
</html>
```

A fenti kód egy egyszerű szöveges weboldal, amelyben a Formula-1 sportág XXI. századi világbajnokait sorolom fel. Látható, hogy metaadatként hozzáadtam a saját nevem. Most nyissuk meg ezt a kódot egy webböngészővel. Ekkor a következő eredményt kapjuk:



3. Ábra: Példa arra az esetre, amikor a metainformáció a HTML-fejben van megadva

(Mozilla Firefox által megjelenített)

Tehát megjelent az RDF-ben metaadatként hozzáadott információ is a képen. A problémára megoldást jelent, ha az RDF egyszerűsített írásmódját használjuk, amely segítségével az olyan predikátum-tárgy párokat, ahol a tárgy egy literál, ott XML-attribútumérték formában írjuk fel. Nem sok módosítást kell végrehajtani, a 9-12. sorban kell módosítást eszközölnünk:

```
<rdf:Description rdf:about=""
  bt:keszito="Készítette Bujdosó Tamás"/>
</rdf:RDF>
```

Ezáltal a kívánt eredmény érhető el. A másik probléma, hogy az ilyen módon készített dokumentum nem érvényes XHTML formátum, mivel nem teljesíti az XHTML sémának megfelelő előírásokat. Ennek ellenére a legtöbb RDF elemző alkalmazás támogatja a metaadatok XHTML-ből való kinyerését.

Az RDF adatokat elhelyezhetjük HTML-hez kapcsoltan is. Ekkor egy linket készítünk, ami a metaadatokra mutat. Ezt a linket kereséskor ugyanúgy követik a robotok, mint a többi linket, így végül eljutnak a valódi RDF leíráshoz. Ez egy nagyon jó megoldás, hiszen nem fordulnak elő a fenti problémák, a leírást a weboldaltól függetlenül lehet módosítani, és nem kell azokat a HTML-kódon belül még külön megkeresni. Ezt a megoldást úgy tudjuk megvalósítani, hogy a dokumentum fejrészében a <link> címke után megadjuk a külső erőforrás elérhetőségét. Nagy előnye a link alkalmazásának, hogy így az RDF-leírásokat többféle reprezentációjú állományban is csatolhatóak.

```
<head>
<link rel=”meta type=”application/rdf+xml”
href=”meta.rdf”/>
<link rel=”meta type=”application/rdf+xml”
href=”meta.nt”/>
</head>
```

Ezen rövid kódrészletben két forrásfájlt csatolunk a HTML-hez, egy rdf és egy nt kiterjesztésűt. A linkelésnek egy fontos hiba róható fel. Az XML taglalása során említettem, hogy több XML dokumentumból lehet egyet is csinálni. Sajnos ez a technika nem használja ki azt, hogy az RDF és a honlapok is XML formátumban vannak megírva, így ezt az egyszerűsítést nem alkalmazhatjuk. Most, hogy már ismerjük azt, hogy hogyan lehet az RDF-ben adatokat tárolni, nézzük meg, hogy miként tudjuk lekérdezni azokat. Erre több lehetőségünk is van. Az egyik, amely a legkézenfekvőbb lenne, az az, hogy XML lekérdezőket használunk. Két ilyen lekérdezőnyelvet ismerünk, az XQuery-t és az XPath-ot. Ezen nyelvek lényege, hogy az XML-fában végigmennek úgynevezett útkifejezések segítségével, amelyek egy-egy csomópontot

azonosítanak a fában. Ezen kifejezések felhasználásával tudunk komplex kérdéseket feltenni. Ha megkaptuk a keresett elemet, akkor ezt kiemeljük, majd köréjük HTML-kódot írunk, hiszen ez a nyelv HTML-alakban adja meg az eredményt. Az XML lekérdezők azonban csak korlátozott módon használhatóak RDF dokumentumokon. Ennek egyik oka az, hogy az XML csak az egyik lehetséges módja az RDF fizikai reprezentációjának. Szükségszerű volt tehát kifejleszteni olyan nyelvet, amely az RDF logikai modelljén operál. A nyelvek megalkotásánál kiemelt figyelmet kapott az SQL, mivel ez a nyelv lehetőséget ad arra, hogy a kérdés feltevője azt mondja meg, hogy mit szeretne tudni és nem azt, hogy azt az eredményt hogyan szeretné elérni. Így a tényleges adattárolásról semmit nem kell tudnunk, hiszen ennek ismerete a lekérdezésfeldolgozó feladata lesz. Ennek szellemében több RDF lekérdező is megtalálható az informatika világában. Egyik közös jellemzőjük, hogy támogatják az RDF-ben fellelhető különböző adattípusokat. Mint ahogy azt az elején említettem, a keretrendszer csak egy általános típusrendszerrel rendelkezik, amelyre az `rdf:datatype` attribútummal adjuk meg. Ezek az XML adattípusait hivatkozzák, az XML adattípusok pedig tetszőlegesen bővíthetők új típusokkal.

A lekérdezők egyik típusa úgy tekint az RDF dokumentumokra, mintha azok adatbázisok lennének. Ezeket nevezzük adatbázis-alapú lekérdezőnyelveknek. Ezek feltételezik a forrás szerkezetének pontos ismeretét. Egy ilyen nyelv az RQS. A nyelv alapötlete, hogy a feltehető kérdések az RDF konténerekből RDF konténereket állítanak elő. Az eredmény mindig részhalmaza a kiindulási konténernek, vagy üres konténer. A kérdések során különböző szűréseket lehet végezni, amelyek bizonyos elemeket választanak ki a megadott konténerből. Az eredménykonténereknek képezhetjük az unióját, metszetét, sorrendbe rakhatjuk az elemeit. Lehetőségünk van univerzális és egzisztenciális kvantorral megadott kérdések lefuttatására is. Láthatjuk, hogy az RQS egy egész komplex kis nyelv, széleskörű használatban azonban nem terjedt el. Ennek legfőbb oka talán az volt, hogy megköveteli a forrás szerkezetének pontos ismeretét. Ez azonban a metaadatok használata esetén igencsak nehézkes.

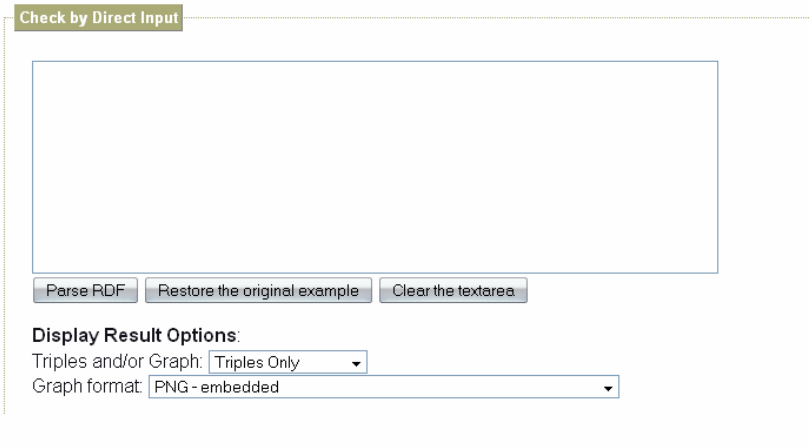
A lekérdezőnyelvek másik csoportjába tartoznak a modellalapú lekérdezők. Ebben a megközelítésben azt használjuk ki, hogy az RDF modell egy irányított gráf, amelynek élei URI-kkal vannak felcímkézve, csomópontjai üresek, vagy literálok. A nyelvek mindegyike a részgráfillesztés és változóbehelyettesítés technikáját alkalmazza a lekérdezéseknél. A kérdések maguk is irányított, címkézett gráfok, amelyek tartalmazhatnak változókat. A változók lehetnek

csomópontban – ekkor egy ismeretlen erőforrást jelöl -, illetve lehetnek élen, amikor is egy ismeretlen tulajdonságra hivatkoznak. A lekérdezésre adott válasz nem más, mint az RDF gráf egy olyan részgráfja, amely a feltett kérdéssel valamely változóbehelyettesítés mellett azonos. Mivel az eredmény is egy gráf, így ezen is lehet további lekérdezéseket végrehajtani. Ezzel biztosítva van a rekurzivitás. Véleményem szerint, ennek a lekérdezőnek a megalkotásakor a Prolog nyelv változókiértékelési algoritmusát vették alapul. Az RDQL egy tipikus modellalapú lekérdezőnyelv.

Érdemes még megemlíteni a SPARQL-t, amely szintén egy modellalapú lekérdező, azzal a sajátossággal, hogy a W3C az RDF-hez feljeleztette ki, és 2008. január 15-én lett hivatalos W3C ajánlás, tehát még egy friss dologról van szó. Ugyanazt a részgráfillesztéses változókiértékelési módot folytatja, mint amit fentebb bemutattam. Lehetőségünk van arra, hogy a keletkezett változóbehelyettesítések alkalmazásával új hármasokat, új RDF- gráfot hozzunk létre. Az is megengedett, hogy lekérdezzük egy erőforrásról annak jellemzőit, azaz hogy milyen élek milyen értékekkel kapcsolódnak hozzá. Egy igen/nem kérdéssel eldönthetjük, hogy a megadott kifejezés kielégíthető-e. Ez a nyelv különös figyelmet fordít a literálok kezelésére, ugyanis megadhatjuk annak nyelvét, típusát, valamint használhatunk reguláris kifejezéseket a keresések során.

Végül megemlíteném, hogy a W3C ezek után még magasabbra tette a mércét, hiszen hozzáfogott egy még újabb nyelv megkonstruálásához, amelyben már szinte természetes nyelven adhatjuk meg a lekérdezéseinket. Ez a nyelv a Metalog, amely a keresés emberi oldalának fejlesztését segíti elő. A hivatalos forrás szerint két dologra épül a Metalog: a logikai kifejezésekkel való következtetések elősegítésére, illetve a Szemantikus web leegyszerűsítésére. Ez azonban még kezdeti fázisban van, még a jövő zenéje.

Az RDF taglalásának végén bemutatnék egy weboldalt, ahol megnézhetjük az általunk írt kifejezések helyességét, megnézhetjük a leírt elemhármasokat, illetve megjeleníti nekünk a kifejezés RDF gráfját. Ez a *http://www.w3.org/RDF/Validator/* oldalon megtalálható online felület lenne. A főoldalon megjelenő szövegdobozba (4. ábra) beírhatjuk az RDF/XML kifejezésünket, majd beállítjuk a megjelenítés módját – elemhármasok, gráf, avagy mindkettő -, majd elküldjük. Ha esetleg hibás kifejezést írunk be, korrekt hibaüzenetet kapunk a hiba helyéről és okáról.



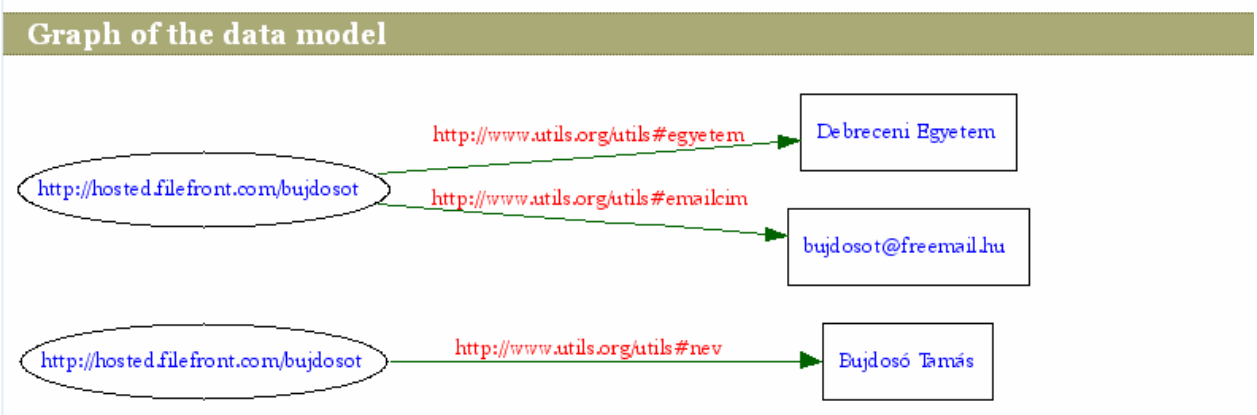
4. ábra: Az RDF Validator szövegbeviteli mezője

Ezt a Parse RDF gombbal tehetjük meg. Az elemző működésének bemutatása érdekében próbáljuk ki az alfejezet elején leírt RDF-kifejezést, amelyben az adataimat adtam meg. Lássuk milyen eredményt kapunk vissza:

Triples of the Data Model

Number	Subject	Predicate	Object
1	http://hosted.filefront.com/bujdosot	http://www.utils.org/utils#nev	"Bujdosó Tamás"
2	http://hosted.filefront.com/bujdosot	http://www.utils.org/utils#egyetem	"Debreceni Egyetem"
3	http://hosted.filefront.com/bujdosot	http://www.utils.org/utils#emailcim	"bujdosot@freemail.hu"

5. ábra: Egy beírt kifejezés eredménye elemhármasok nézetben



6. ábra: Egy beírt kifejezés eredménye RDF-gráf nézetben

Összességében elmondható az RDF-ről, hogy nagyon hasznos dolog, azonban nem old meg minden problémát. Az összetett alkalmazásoknak szükségük van arra, hogy következtetéseket tudjanak levonni. Meg tudja-e nekünk mondani egy alkalmazás azt, hogy ha „A” jobbra van „B”-től, és „B” jobbra van „C”-től, akkor „A” is jobbra van-e „C”-től? Az ember számára ez nyilvánvaló. Ahhoz, hogy a gép számára is az legyen, nekik ezt ki kell következtetniük és le kell vezetniük bizonyos állításokból. A szemantikus webnek ehhez úgynevezett ontológiákra, amelyek meghatározzák a tudásunk leírásához és kifejezéséhez szükséges fogalmakat és kapcsolatokat.

3.5 Ontológiák

Az ontológia szó jelentését sokan sokféleképpen leírták már. Eredetileg a filozófiából származik, ott a létezéstant, lételméletet jelenti és innen vette át az informatika. Thomas Gruber megfogalmazása szerint: „Az ontológia megegyezésen alapuló fogalmi rendszer formális, egyértelmű leírása”. Ez tehát egy tudományág, amely egymáshoz hasonló elméleteket tanulmányoz. A szemantikus web számára az ontológia egy dokumentumot vagy egy fájlt jelent, amelyben formálisan definiálva vannak egy szakterület fogalmai és kapcsolatai. Az ontológiához szorosan kapcsolódó fogalom a taxonómia. Taxonómia alatt a dolgok osztályozásának és kategorizálásának egy hierarchikus formáját értjük. A hierarchia fastruktúrával ábrázolható. Ezek arra jók, hogy osztályozzuk az információs egyedeket. Segítségükkel kifejezhetjük azt a minimális mértékű szemantikát, amely szükséges az objektumok megkülönböztetéséhez.

Az ontológiák és a taxonómiák többféleképpen növelhetik az internet hatékonyságát. Pontosítják a keresést, mivel ezek segítségével a keresők ki tudják szűrni azokat a lapokat, amelyek a pontos fogalomra hivatkoznak, így elkerülhetőek a kulcsszavas keresők által kiadott rossz találatok.

Az ontológiák alapelemei a fogalmak és a relációk. A fogalmak merev, állandó tulajdonságot leíró típusokat, vagy változó szerepeket jelölnek. A relációk a fogalmak összekapcsolását hivatottak elvégezni. Általában bináris relációt alkalmaznak, mivel azt könnyebb áttekinteni. Bevezeti még a példány fogalmát is, mely egy fogalom konkrét egyedének tekinthető. Az ontológiák bevezetését az informatikába az az igény inspirálta, hogy a tudásunkat megoszthassuk egy közös nyelv használatával, illetve, hogy elősegítsék az egyes alkalmazások közötti

kommunikációt. Ezek figyelembe vételével elkezdtek megjelenni az úgynevezett ontológianyelvek. Az ontológianyelvek az RDF sémából származnak, annak kibővítéseként alkották meg. A jelenlegi hivatalos W3C ontológianyelv az OWL(***W***eb ***O***ntology ***L***anguage). Ez előtt is készültek ilyen nyelvek, amelyek az OWL alapjaként tekinthetők. Ezek voltak az OIL(***O***ntology ***I***nterface ***L***ayer), és a DAML-ONT(***D***ARPA ***A***gent ***M***arkup ***L***anguage). Az OIL néhány egyetemi kutató közös munkájának tekinthető, amelyet aztán Manchesterben mutattak be. A leíró logikákat, valamint az XML és RDF nyelveket használták fel, hogy létrehozzák a szemantikus web egységes jelölőnyelvét. Ezzel szemben a DAML célja az volt, hogy a SGML és a HTML után egy újabb jelölőnyelvet hozzon létre, amely lehetővé teszi a weben található információ gépi feldolgozását, azaz a szemantikus web létrejöttét. 2001-ben egyesítették a két nyelv előnyös tulajdonságait, és létrejött a DAML+OIL nyelv. Ezt benyújtották a W3C bizottsághoz, hogy alapot adjon a készülő hivatalos ontológianyelvnek. Így jött létre az OWL nyelv. A W és az O betűk felcserélése nem véletlen, mivel az owl szó jelentése bagoly, amik ugyebár bölcs állatok, így a fejlesztők úgy gondolták, hogy az elkészült nyelvnek is jó név lesz ez. Az OWL 2004 óta hivatalos W3C ajánlás. Egyidőben vált azzá az RDF és az RDF séma is, ami nem véletlen, hiszen ezek egymásra épülő technológiák.

Az OWL-t, akárcsak az RDF-et tekinthetjük URI-k halmazának. Ebbe a halmazba olyan URI-k tartalmaznak, melyek segíthetnek nekünk egy terminológiai rendszer elkészítésében. Az erőforrások jelentése szabványos, segítségükkel leírhatjuk az osztályok diszjunktságát, ekvivalenciáját, tulajdonságok tranzitivitását, korlátosságuk megadását. Ezekre az RDF sémában nem volt lehetőségünk. Mint már említettem, az OWL az RDF szintaxisát használja, így egy RDF elemző alkalmas az olvasásukra. Az OWL él az úgynevezett „nyíltvilág-feltételezéssel”. Ez azt jelenti, hogy egy ontológiát készítő személy által definiált osztályhoz bárki más is rendelhet különböző tulajdonságokat. Ez a többletinformáció kiegészíti a készítő személy által mondottakat, így az hasznosabb lesz, vagy akár ellentmondásba is ütközhet. A nyelv maga monoton, vagyis ha egy konzisztens állapotban képesek voltunk levonni egy következtetést, akkor ezt meg kell tudnunk tenni akkor is, ha egy új információt adunk a rendszerhez, amellyel a konzisztens állapot megmarad. Ha ez nem így lenne, akkor az azt jelentené, hogy képesek vagyunk tetszőleges állításra következtetni. Így a nyelvben állítások törlésére nem lesz mód, még az olyanokra sem, amelyeket nem mi adtunk hozzá a tudásbázishoz, és esetlegesen

ellentmondásba ütköznénk vele. Az OWL megkülönbözteti az egyedeket és értékeket tartalmazó alaphalmazokat, bár ezek bizonyos esetekben egybeeshetnek. Ennek megfelelően beszélhetünk egyedtulajdonságokról, amelyek egyedeket egyedekkel kötnek össze, illetve vannak az adattípus-tulajdonságok, amelyek egyedekhez értékeket rendelnek hozzá. Az OWL hatfajta osztályt ismer, ezeket a létrehozás és a leírás módja közötti különbségek szerint csoportosíthatjuk. Ezek a csoportok a: megnevezett, felsorolásos, tulajdonságkorlátozással megadott, osztályok metszeteként megadott, osztályok uniójaként előállított és az egy osztály komplemenseként előállított osztályok. A megnevezett osztályok az egyedüliek, amelyekhez URI is társul. A többi névtelen osztály, amely megkötéseket tesz arra, hogy milyen objektumok lehetnek az elemei. A felsorolásos osztály konkrétan megadja az elemeit, az utolsó három osztálytípus már létező osztályokból hoz létre újakat, a megfelelő megszorítások alkalmazásával. Látható, hogy tetszőleges bonyolultságú osztályszerkezet hozható létre. A nyelvben van két előre definiált osztály: az owl:Thing, ami egy szuperosztály. Ez minden OWL osztály szülőosztálya, és ezzel képes megkülönböztetni az egyedeket az osztályoktól. A másik ilyen beépített osztály az owl:Nothing, ennek egyáltalán nincs példánya, így ez az összes OWL osztály alosztályának is tekinthető. Az összes többi osztályt a

```
<owl:Class>
 …
</owl:Class>
```

alakban adhatjuk meg, illetve az owl:Class helyett szerepelhet annak alosztálya is, az owl:Restriction, ha valamelyik korlátozással megadott osztályról van szó. Például a tulajdonságkorláttal megadott osztályok általános alakja a következő:

```
<owl:Restriction>
  <owl:onProperty rdf:resource=”P” />
</owl:Resrtiction>
```

ahol a P a megnevezett tulajdonság URI-ját jelenti. P egyaránt lehet egyed –és adattípus-tulajdonság is. Itt adhatunk meg számosságkorlátot is. Ezek azonban csak arra vannak hatással,

hogy bizonyos erőforrások melyik osztályba tartozzanak, arra nem, hogy egy erőforrásból hány darab és milyen típusú él indulhat ki. Ezt az owl:cardinality tulajdonsággal adhatjuk meg, amelynek jobb oldalán egy nemnegatív egész szám áll. Jelentése, hogy az adott osztályba azon példányok kerüljenek, amelyekhez pontosan a megadott számú olyan tulajdonságok kerüljenek, amelyeknek jobb oldala szemantikailag különbözik.

Térjünk át a következő osztályépítő műveletre, a metszetképzésre. Ez egy owl:Class osztálybeli példányt köt össze egy osztályleírásokat tartalmazó listával. Megvalósítására az owl:intersectionOf attribútum szolgál. Lássunk rá egy példát, amely megadja azokat a hallgatókat, akiknek angol és német nyelvvizsgájuk is van.

```
<owl:Class>
  <owl:intersectionOf rdf: parseType="Collection">
     <owl:Class rdf:about="#angol"/>
     <owl:Class rdf:about="#német"/>
 </owl:Class>
```

Ez két megnevezett osztály – az angol és a német – metszetét adja meg, de ez a művelet természetesen ugyanígy működik akkor is, ha felsorolásos osztályokkal dolgozunk.

A másik halmazművelet, amit használhatunk egy új osztály létrehozásához, az unióképzés. Az owl:unionOf egy owl:Class-beli osztályban levő példányt köt össze egy osztályleírásokat tartalmazó listával. Az így kapott osztály azon egyedeket tartalmazza, amelyek legalább az egyik, a listában szereplő osztályleírások által meghatározott osztály példányai. A fentebbi példát átírva megkaphatjuk azokat a hallgatókat, akiknek vagy angol vagy német nyelvvizsgájuk van.

```
<owl:Class>
  <owl:unionOf rdf: parseType="Collection">
     <owl:Class rdf:about="#angol"/>
     <owl:Class rdf:about="#német"/>
</owl:Class>
```

Az utolsó művelet, amellyel osztályt deifiniálhatunk, a komplemensképzés. Az owl:complementOf tulajdonság egy owl:Class osztály példányát egyetlen osztályleírással köt össze. Ezáltal olyan osztályokat hozhatunk létre, amelynek pontosan azok az elemek lesznek a példányai, amelyek nem példányai a megadott osztályleírásnak. Lássuk azokat az egyedeket, amelyeknek nincs angol nyelvvizsgájuk:

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="#angol"/>
  </owl:complementOf>
 </owl:Class>
```

Ez nem tűnik bonyolult megoldásnak, azonban van egy szépséghibája. Mégpedig az, hogy ebbe az osztályba bekerülhet bármi, ami nem példánya az *angol* osztálynak, például a könyv, vagy a számítógép. Erre megoldás lehet, ha létrehozunk egy Hallgatók osztályt, és a kódunkat az alábbi módon javítjuk:

```
<owl:Class>
   <owl:intersectionOf rdf:parseType:"Collection">
     <owl:Class rdf:about="Hallgató"/>
      <owl:Class>
         <owl:complementOf>
           <owl:Class rdf:about="#angol"/>
         </owl:complementOf>
      </owl:Class>
   </owl:intersectionOf>
</owl:Class>
```

Így tényleg csak hallgatók lehetnek a kapott osztályban.

Lehetőségünk van az OWL-ben úgynevezett OWL-axiómák létrehozására is, amelyek a különböző ontológiák közötti kapcsolatok megalapozására valók. Segítségükkel kijelenthetjük a saját ontológiánkról, hogy az abban szereplő osztályok ekvivalensek egy másik ontológiában

szereplő osztállyal, így azokat össze is kapcsolhatjuk. Az axiómákat háromféle attribútummal képezhetünk, ezek az rdf:subClassOf, az owl:equivalentClass és az owl:disjointWith. Az elsőt már ismerhetjük az RDF sémákból, ott arra használtuk, hogy két osztály tartalmazási viszonyát írjuk le vele. Jelen esetben pedig megmondhatjuk, hogy az alanyként megadott osztályleírás részhalmaza a tárgyként megadott osztályleírásnak. A második tulajdonsággal azt fejezhetjük ki, hogy az alanyként megadott osztály leírása pontosan megegyezik a tárgyként megadott osztályéval, tehát a két osztály példányhalmaza megegyezik. Ha a környezetből egyértelműen kiderül ez a tulajdonság, akkor az owl:equivalentClass megadása elhagyható. A harmadik attribútum segítségével az adható meg, hogy az alanyként és a tárgyként megadott osztályleírásoknak nincs közös eleme, vagyis diszjunktak. Mint ahogy azt már említettem, az OWl-ben egyed –és adattípus-tulajdonságokat különböztetünk meg. Az előbbit az owl:ObjectProperty, míg az utóbbit az owl:DatatypeProperty osztály példányaként tudjuk megadni. Sajnos a tulajdonságok definiálására már nincs olyan sok lehetőségünk, mint amennyi az osztályoknál volt. Nem tudunk tulajdonságokat más tulajdonságokkal definiálni, nem adhatunk meg úgy tulajdonságot, hogy felsoroljuk, hogy mely más tulajdonságok kompozíciója lenne. Az egyetlen lehetőségünk az, hogy kijelentjük egy tulajdonságról, hogy létezik. Léteznek tulajdonságaxiómák is a nyelvben. Ezeket négy nagy csoportra lehet osztani. Az első csoportba, az RDF séma konstrukciók közé tartozik a tulajdonság-altulajdonsági viszony kifejezésére szolgáló rdfs:subPropertyOf attribútum, illetve az értelmezési tartományra és értékkészletre vonatkozó megszorítások. A második csoportba tartozó konstrukciók segítségével kijelenthető egy tulajdonságról, hogy az ekvivalens-e egy másikkal, vagy inverze-e annak. A harmadik csoport, a globális számosság megkötések csoportja, amelyek egy tulajdonságra nézve kikötik, hogy azok jobb vagy bal oldalán állhat-e egynél több egyed, illetve érték. Az utolsó, tulajdonságkarakterisztikák nevű csoportban egy tulajdonságról kijelenthető, hogy az szimmetrikus vagy tranzitív.

Az OWL-ben az adattípusokat az RDF-ben bemutatott módon használhatjuk. Az OWL használja a legtöbb beépített XML Séma adattípust. Az ilyen adattípusokra való hivatkozás az adattípusnak megfelelő URI referenciával történik: http://www.w3.org/2001/XMLSchema. Minden OWL feldolgozó rendszertől elvárja a nyelv, hogy legalább az xsd:integer és az xsd:string

adattípusokat támogassa. Tegyük fel, hogy én a Hallgató osztály egy példánya vagyok, és ilyen formában szeretném megadni az életkoromat. Erre az alábbi lehetőség van:

```
<Hallgató rdf:ID="Bujdosó Tamás">
  <életkor rdf:datatype="&xsd;int">23</életkor>
</Hallgató>
```

Látható, hogy a második sorban hivatkozok az XML Séma egész szám típusára.

Ahhoz, hogy az ontológiák elérjék a maximális hatásukat, az kellene, hogy széles körben használják őket. Ehhez pedig elengedhetetlen, hogy a fejlesztésükhöz szükséges erőfeszítéseket minimalizáljuk. Ezért az ontológiáknak újrafelhasználhatóknak kell lenniük. Az lenne a legjobb, ha az ontológiákat nem is kellene fejleszteni, hanem csak meglévő ontológiákból összeállítani. Fontos azonban észrevenni azt, hogy az ontológiák fejlesztésében a legtöbb erőfeszítést arra kell fordítanunk, hogy úgy kapcsoljuk össze az osztályokat és a tulajdonságokat, hogy azok többféle információ kinyerését tegyék lehetővé. Azt szeretnénk, ha az osztályokról és példányaikról tett állításokból hasznos következtetéseket lehessen levonni. Az ontológiák fejlesztésének ez a legnehezebb része. Ha sikerül találnunk egy olyan ontológiát, amelyik sok finomításon esett már át, akkor érdemes azt átvenni a saját ontológiánkba. Azonban nem könnyű dolog a meglévő ontológiákból egy megfelelő újabbat összeállítani, mivel a konzisztencia majdnem biztosan sérülni fog. Ahhoz, hogy egy a meglévő ontológiát komponensként egy újabb ontológiába beépíthessünk, gyakran meg kell jelölnünk, hogy az egyik ontológiában egy adott osztály vagy tulajdonság egyenértékű egy másik ontológia valamelyik osztályával vagy tulajdonságával. Az owl:equivalentClass tulajdonságot használjuk annak jelölésére, hogy két osztálynak pontosan ugyanazok az egyedei. A tulajdonságok hasonló módon való összekapcsolása az owl:equivalentProperty használatával történhet. Egyedek azonosságát is vizsgálhatjuk. Ekkor, két ontológia egyesítése során, két egyedet, amelyet két különböző dokumentumban más néven definiáltak, azonosnak kiáltunk ki. Ezt az owl:sameAs attribútummal tehetjük meg. Láthatjuk, hogy az OWL nem feltételez egyedi neveket, azonban az, hogy a nevek különböznek, még nem jelenti azt, hogy ezek különböző egyedekre hivatkoznak. Ha már meg tudjuk vizsgálni két egyed azonosságát, vizsgálhatjuk annak ellentettjét is. A különbözőség vizsgálatára az

owl:differentFrom ad lehetőséget. Ez a sameAs tagadása, és azt deklarálja, hogy a két vizsgált URI különböző egyedekre mutat. Előfordulhat olyan eset is, amikor az ellentmondásosság kizárása végett garantálunk kell az egyedek kölcsönös különbözőségét. Ezt az owl:AllDifferent tulajdonsággal adhatjuk meg.

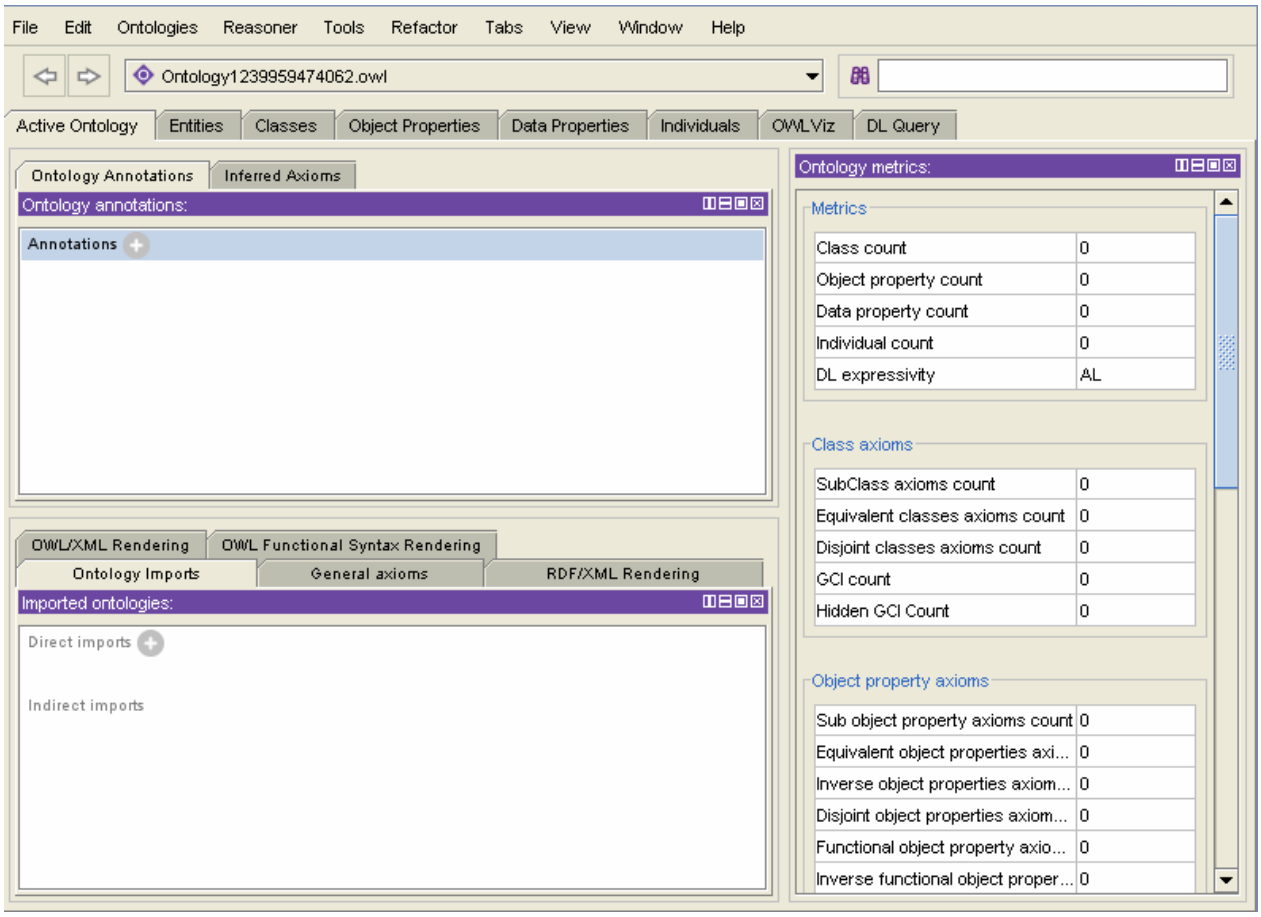
Mivel az ontológiák folyamatosan változhatnak, többféle verzió is létezhet egy adott ontológiából. Az OWL-ben lehetőségünk van a verziókezelésre is. Erre az owl:Ontology megadása után van lehetőségünk. Egy konkrét ontológia leírása mindig ezzel a jelölővel kezdődik. Itt adhatjuk meg a rá vonatkozó metaadatokat, a verzióját, az esetleges megjegyzéseket, vagy egy másik ontológia importálása is itt történik. Az owl:priorVersion segítségével lehetőségünk van az éppen definiált ontológiát az előző verziójához linkelni. Az ontológiák verziói azonban nem feltétlen kompatibilisek egymással. Előfordulhat, hogy a jelenlegi verzióban vannak olyan kijelentések, amelyek ellentmondásban állnak egy korábbi verzióéval. A kompatibilitást kétféleképpen fejezhetjük ki: az owl:backwardCompatibleWith használatával, amely azt mondja meg, hogy a hivatkozott ontológia a jelenleginek az előző verziója és visszamenőlegesen kompatibilis vele; illetve az owl:incompatibleWith attribútummal, amelyet akkor használhatunk, ha a hivatkozott ontológia a jelenleginek ugyan az előző verziója, viszont az visszamenőleg nem kompatibilis vele. Ha nem deklaráljuk a visszamenőleges kompatibilitást, akkor nem szabad a kompatibilitást feltételeznünk.

Fontos még említést tenni az OWL résznyelveiről. Három ilyen van, amelyek egyre erősebb kifejezőerővel rendelkeznek. Ezeket a fejlesztők és a felhasználók különböző csoportjainak fejlesztették ki. Így jött létre az OWL Lite, az OWL DL és az OWL FULL. Az OWL Lite olyan felhasználók támogatására készült, akik elsősorban osztályozási hierarchiákat és egyszerű korlátozásokat alkalmaznak. Így például támogatja a számosság korlátozást, de a számosság értékeként csak a 0 és 1 értéket engedi meg. Ennél a nyelvnél az volt a cél, hogy egy minimálisan használható készletet kapjunk, melyet könnyű implementálni. Könnyen felépíthetjük az osztályhierarchiákat és alkalmazhatjuk a tulajdonságmegkötéseket is. Az OWL DL azokat a felhasználókat támogatja, akik a maximális kifejezőképességet igénylik, vagyis minden konklúzió garantáltan kiszámítható, és minden számítás véges időn belül be is fejeződik. Az OWL DL tartalmazza az összes OWL nyelvi konstrukciót, de ezek csak bizonyos típus-

szétválasztási korlátozásokkal használhatók (például egy osztály több osztálynak is alosztálya lehet, egy osztály azonban nem lehet egyede egy másik osztálynak). Az OWL DL a leíró logika (***D***escription ***L***ogic) kifejezésre utal, mely az OWL formális megalapozásának logikai kérdéseit tárgyaló kutatási terület neve, és amellyel a következő fejezetben fogok foglalkozni.

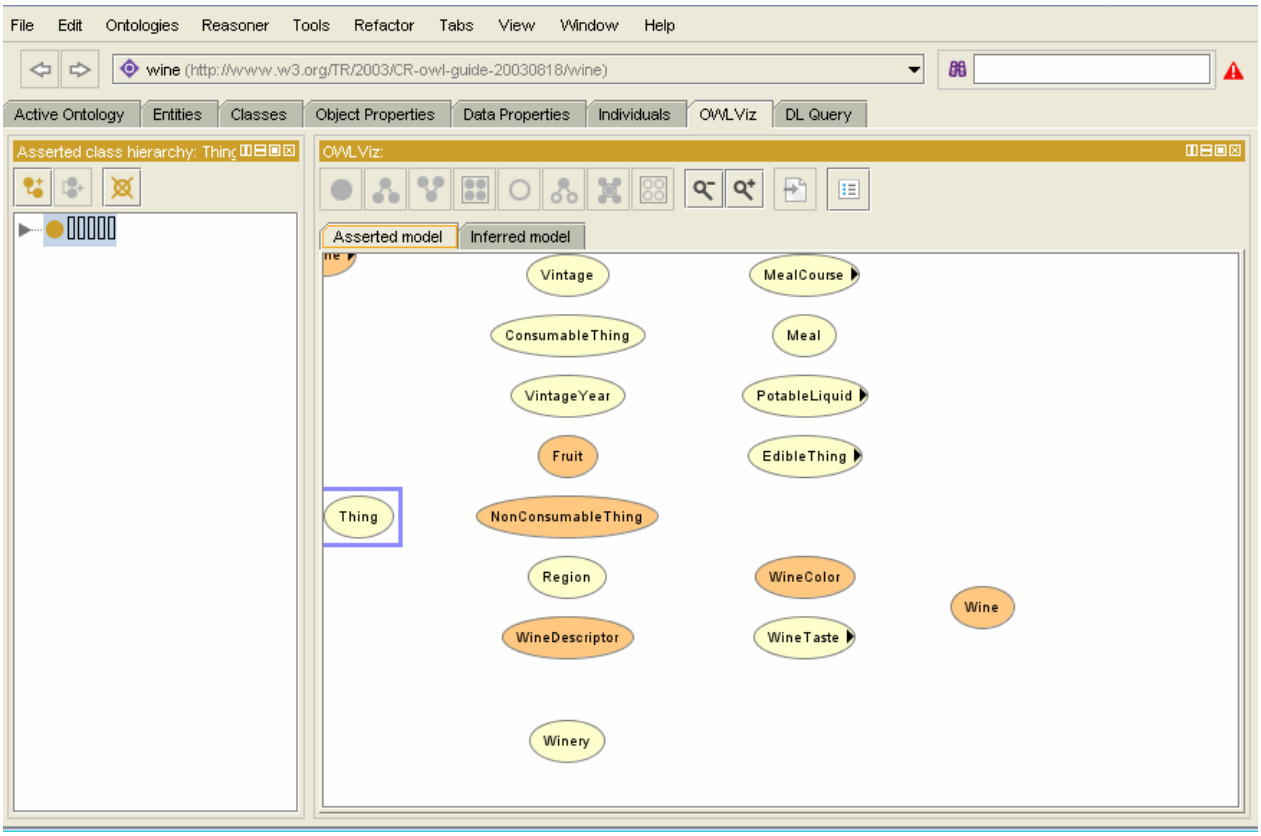
Az OWL Full alnyelvet olyan felhasználóknak szánták, akik a maximális kifejező erőt és az RDF szintaktikai szabadságát igénylik, de ennek fejében lemondanak a kiszámíthatósági garanciákról. Az OWL Full lehetővé teszi egy ontológiának, hogy kiterjessze az előre definiált (RDF vagy OWL) szókészlet jelentéstartományát. Ez a nyelv nem feleltethető meg egyetlen leíró logikának sem. Mielőtt hozzálátunk egy ontológia kifejlesztéséhez, el kell döntenünk, hogy számunkra melyik a legmegfelelőbb résznyelv. Ezek azért is hasznosak számunkra, mert egy egyszerűbb nyelvet egyszerűbb leprogramozni. Ha az ontológiánkban nincs szükség olyan bonyolult nyelvi eszközökre, amelyek az OWL Full-ban vannak, felesleges lenne ezt a nyelvet használni.

Végezetül ebben a fejezetben is szeretnék megmutatni egy konkrét eszközt, amellyel saját ontológiát fejleszthetünk. Az internetet és a szakkönyveket böngészve azt találtam, hogy erre a célra a Stanford Egyetem által kifejlesztett Protégé rendszer a legalkalmasabb. Ez egy ingyenes, nyílt forrású ontológiakészítő, amelyet bárki letölthet a http://protege.stanford.edu/ internetes oldalról. A program oldalán található egy kis bemutató is egy konkrét ontológia megírásáról. Ez különböző borokat ír le, azok ízét, cukortartalmát, termelési helyét…stb. Lássuk, mit hol lehet megadni a Protege-ben!



7. ábra: A Protégé kezelőfelülete

A Protégé indítása után az ábrán látható kezelőfelülettel találjuk szembe magunkat. Látható, hogy a menüsor alatt különböző fülek vannak. Ezeket váltogatva adhatjuk meg az ontológiánk különböző adatait. Az Entities fülre kattintva felsorolhatjuk az adott fogalmakat. Ezután a Classes fülre váltva az ontológia osztályait és osztályhierarchiáját alakíthatjuk ki. A következő fülön az objektumok különböző tulajdonságait állíthatjuk be, itt adható meg például a kompatibilitás (owl:compatibleWith). A Data Properties résznél fejezhetünk ki különböző dolgokat az osztályokról, például azt, hogy egy osztály mely másikkal ekvivalens(owl:equivalentWith) vagy éppen hogy mely osztályokkal diszjunkt(owl:disjointWith). Az Individuals fül alatt a fogalmakhoz tartozó konkrét egyedeket sorolhatjuk fel. Az elkészített ontológiánkat grafikusan is szemügyre vehetjük, ha a OWLViz lehetőséget választjuk. A fentebb említett gyakorló ontológia elkészülte után így néz ki ez a grafikus ábra:



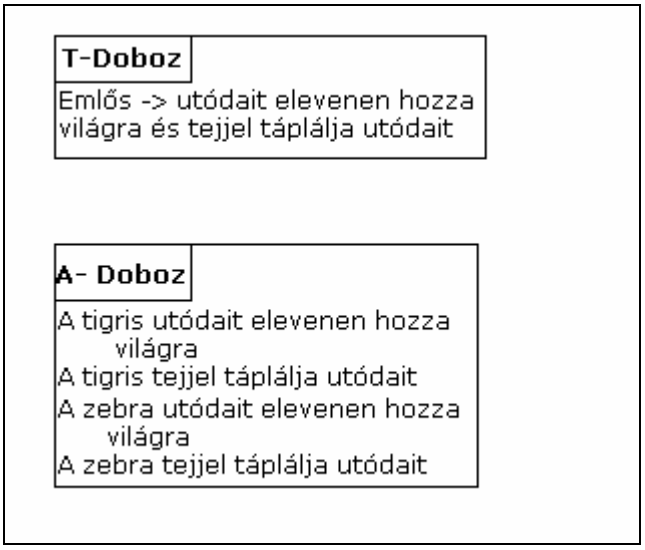
8. ábra: Egy ontológia grafikus vázlata a Protégé programban

A Stanfordi Egyetem és a Wikipedia összefogott, és létrehoztak egy internetes oldalt, ahova a felhasználók feltölthetik az általuk Protégé-ben elkészített ontológiájukat. Az oldal a http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library címen érhető el.

3.6 Logika

Berners-Lee diagramjának következő része a logika, azon belül is a leíró logikák. Ezek az elsőrendű logika résznyelvei, és ezek képezik az OWL hátterét. Amikor az ontológiákat ténylegesen működésbe akarjuk hozni, akkor az ontológiában lévő állításokat át kell alakítani leíró logikai állításokká. A leíró logikák segítségével le tudjuk írni valamely szakterület, vagy közismereti terület fogalmi rendszerét. A leírás során beszélhetünk fogalmakról, amik az egyedek halmazai, illetve az egyedek között fennálló kapcsolatokról, a szerepekről. A leíró logikákban az RDF-hez hasonló módon csak kétargumentumú kapcsolatokkal tudunk dolgozni. A fogalom

RDF-beli megfelelője az osztály, a szerepet pedig az RDF-beli tulajdonsággal rendelhetjük össze. A leíró logikák megalkotásának legfőbb célja az volt, hogy az összetett fogalmakat egyszerű, elemi fogalmak és szerepek segítségével tuják megadni. Erre azért van szükség, hogy a weblapok és egyéb webes információk a gép által értelmesen kereshetők legyenek. Ha a fogalmakat pontosan definiáljuk, akkor a gép automatikusan fog tudni következtetéseket végrehajtani. A következtetéshez kétféle állítást használhatunk: egy fogalom definícióját, amely általánosan igaz, és amit terminológiai állításnak nevezünk; illetve a konkrét tudást az egyedekről, amit adatállításnak hívunk. Ezeket egy közös, úgynevezett állításgyűjteménybe, tudásbázisba rendezzük. A tudásbázis két részre oszlik: a terminológiai állításokat tartalmazó T-dobozra, és az adatállításokat tartalmazó A-dobozra. A következtetésekhez szükségünk van egy következtetőrendszerre, amely feldolgozza az adott kérdést, és megpróbál rá válaszolni. Lekérdezhetjük például, hogy egy adott fogalomnak mik az egyedei, vagy azt hogy egy egyed melyik fogalomba sorolható. Előfordulhat olyan lehetőség is, amikor a következtetés elvégzéséhez nincs szükség A-dobozra, elegendő a T-doboz. Az egyik legfontosabb kérdés az, hogy kielégíthető-e egy adott fogalom. Ha számunkra nem elfogadható választ kapunk, mivel tudjuk a kérdésről, hogy az kielégíthető lenne, de a gép mégis azt adja vissza, hogy nem az, akkor valószínűleg a fogalmi rendszer felépítése során hibáztunk. Az elméleti áttekintés után lássunk egy példát:



9. ábra Példa egy leíró logika tudásbázisára

A fenti példában meghatároztam, hogy mit jelent az „emlős” fogalom, illetve két példát írtam rá az A-dobozban. Egy következtetőrendszernek a következő kérdéseket tehetjük fel ilyenkor:

- Kik az emlősök?
- A tigris emlős?
- A zebra emlős?

A rendszer a két dobozban levő állítások összevetésével meg tudja válaszolni a kérdéseket. Az állításokat elsőrendű logikai alakban formalizálhatjuk. Ezzel azonban van egy probléma: ez a fajta logika nem eldönthető, azaz létezhet olyan állítás, amelynek sem az igazsága sem a hamissága nem bizonyítható. Ezért kell a leíró logikát alkalmazni, mert ezek mind eldönthetőek. Formalizmusa hasonló az elsőrendű logikáéhoz. Ugyanúgy megtalálhatók a kvantorok, illetve alkalmazza a matematikai halmazműveleteket. A T-dobozbeli „emlős” fogalom például így néz ki: emlős≡utódait elevenen hozza világra ^ tejjel táplálja utódait. A konstruktorokat fogalomkonstruktorként használhatjuk, az alábbi alakban: kvantorSzerep.T, amivel egyedekhez rendelhetünk szerepet. Megtalálható a logikákban a negáció művelet is, illetve bevezeti még a ⊥ fenékjelet, ami a semmis fogalmat jelöli, amelynek nincs egyede. A halmazelméletből átveszi a metszet, az unió, az inverz- vagy más néven komplementerképzés illetve a részhalmaza fogalmakat. A metszet művelet megfelel az owl:intersectionOf műveletével, az unió az owl:unionOf-fal egyenértékű, míg owl:complementOf attribútum a leíró logika teljes negálásának felel meg. Ezek után nézzük meg, hogy a nyelvek milyen eszközöket alkalmaznak a következtetés végrehajtásához. Az egyszerűbb leíró logikai nyelvek – például az AL, ALC – a strukturális tartalmazási algoritmust használják. Ennek lényege, hogy a fogalmakat egy normálalakra hozzuk, így két fogalom tartalmazási viszonya a kifejezések szerkezetének vizsgálatával egyszerűen eldönthető. Az összetettebb nyelvek – *SHIQ, SH, SHIF* - az úgynevezett tabló-algoritmus szerint következtetnek. Ez a módszer egy fogalom kielégíthetőségét adja meg egy adott T-doboz felett. Először is megpróbáljuk felépíteni az adott fogalom modelljét. Ezt egy tabló-gráf segítségével oldjuk meg, amelynek csomópontjai az alaphalmaz elemeit, az élei pedig a szerepeket reprezentálják. Egy csúcs címkéje olyan fogalomkifejezésekből áll, amelyek mindegyikébe bele kell tartoznia az adott csúcsnak. Az éleket megcímkéző szerepeknek olyanoknak kell lenniük, amelyeknek az élhez tartozó két csúcs között fenn kell állniuk. A gráf mellé meg kell adnunk egy egyenlőtlenségrendszert, ez a két dolog együtt adja meg a tabló-

állapotot. Az algoritmusban transzformációs szabályokat adhatunk meg, amelyek a tabló-állapotot egy vagy több új állapottal bővíti ki. Az ilyen szabályok fontos tulajdonsága, hogy megőrzik a kielégíthetőséget. Egy állapotban ellentmondást kapunk, ha egy csúcs címkéjében levő fogalom együtt szerepel annak negáltjával. Ezt az esetet ütközésnek nevezzük. Az algoritmus véget ér, ha olyan állapotba jut, ahol már semmilyen transzformációs szabály nem alkalmazható és nem észlelt ütközést. Ebben az állapotban azt kapjuk, hogy a fogalom kielégíthető a T-doboz felett. Azonban úgy is véget érhet az algoritmus, hogy a keresés során minden ágon ütközést észlelt. Ilyenkor azt mondja, hogy a fogalom nem elégíthető ki a T felett. Ezután, ha már ismerjük a leíró logikák elméleti hátterét, nincs más dolgunk, mint hogy egy programozási nyelvben implementáljuk azt. A fogalmak, szerepek és a tabló-állapot leírásához valamilyen adatszerkezetet kell használnunk. A transzformációs szabályok gépi ábrázolásához a nyelv beépített függvényeit, illetve saját függvényeinket vehetjük igénybe. A végül megírjuk a következtető algoritmust.

3.7 Bizonyítás

Az OWL-ben, leíró logikai nyelven megadott állítások helyességének bizonyítására is szükségünk van. Ezek között lehetnek olyanok, amelyeknek az elvégzéséhez sok időre, vagy erőforrásra van szükségünk. Az elvégzett következtetések eredményét azonban meg szeretnénk osztani a weben, és az is jó lenne, ha az eredmények helyességéről bárki meggyőződhetne. Ahhoz, hogy a másik félnek ne kelljen még egyszer elvégezni az egész következtetést, csatolnunk kell a bizonyításokat, levezetéseket. Ha ezt megtesszük, akkor csak egy gyors ellenőrzést kell végrehajtania, hogy tényleg nincs-e benne hiba. Felvetődik a kérdés, hogy hogyan lehetne ezeket a bizonyításokat és levezetéseket csatolni egy állításhoz. Már ismerünk egy elég jól használható leíró technológiát, az RDF-et. A stanfordi egyetemen 2006 óta dolgoznak azon, hogy hivatalossá tegyék az új „találmányukat”, a PML-t. A PML (***P***roof ***M***arkup ***L***anguage), az RDF alapjain nyugszik, és egy bizonyítás leírására szolgál. Emellett arra is használható, hogy segítse a különböző következtetőrendszerek közötti automatikus információcserét. Ezzel lehetővé válik, hogy megjelöljék azokat a leírásokat, amelyek ugyanazt a fogalomrendszert írják le, csak éppen különböző felhasználók írták azokat, más-más technikákat, esetlegesen más nyelvet alkalmazva.

Fentebb említettem, hogy a PML az RDF-en alapszik, osztályszerkezetét viszont az OWL-ből vette át, ugyanis a PML osztályok az owl:Class attribútum alosztályai. A nyelv típusai megegyeznek az XML alaptípusaival. Mindemellett ez a technológia még csak a munkatervi fázisban van, így még elég messze áll a W3C ajánlástól, azonban ha hivatalosan is azzá válik majd, jelentősen megkönnyítheti a bizonyítások megosztásának folyamatát.

3.8 Bizalom és a digitális aláírás

Az összes eddig felsorolt technológia nem ér semmit, ha a felhasználó nem bízik meg a más forrásból szerzett információban. El kell érnünk, hogy bárki, aki használni akarja a szemantikus webet, biztos lehessen abban, hogy az információ, amit kap, bizonyítottan igaz. Ehhez úgynevezett digitális aláírásokkal kell ellátni azokat, amelyek olyan egyedi kódolt adatok, amelyek a címzettek számára egyértelműen azonosítják az információk küldőjét, és biztosítják számukra az adatok sértetlenségét. Így egy dokumentum nem csak magukból a leírt adatokból fog állni, hanem ahhoz hozzácsatolhatjuk azt is, hogy mely állításokat kik írtak alá, kik biztosítják azok helyességét. Ezáltal bárki felhasználhatja ezeket az adatokat például egy új ontológia vagy RDF-fájl írásához. A digitális aláírásokat könnyedén hozzácsatolhatjuk az RDF leíráshoz. Erre az rdf:this signature attribútum szolgál, ahol a signature kulcsszó után lehet megadni azokat, akik aláírták a dokumentumot, vagy megadható egy URI is, amely szintén ezekre hivatkozik.

Ezzel végigértünk a szemantikus web rétegein. A következőkben megpróbálom megmutatni, milyen irányban halad a kutatás jelenleg, és milyen jövője lehet a világhálónak.

4. Szemantikus keresők

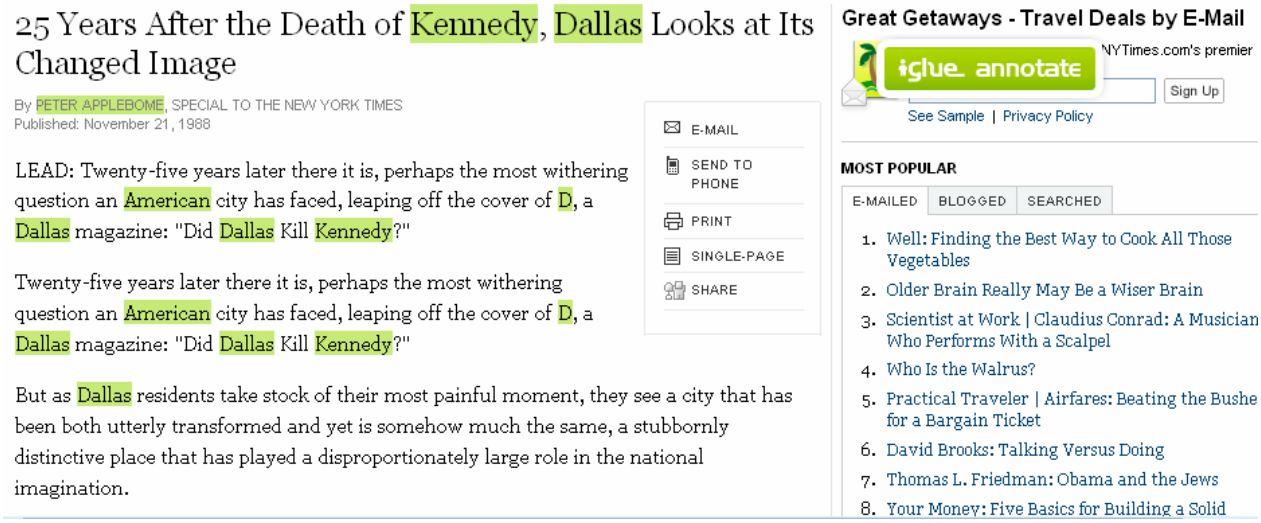
A keresők következő generációjának tekinthetjük az effajta keresőket. Megalkotásuknál az oldalak szövegének megértését vették alapul, és mivel ez az irány tűnik a jövő nagy feladatának, a mai nagy keresők is elkezdték a saját rendszerüket is ez irányban fejleszteni. Jelenleg két nagy ígéret van a szemantikus keresők terén, a Powerset, amelynek erejét mutatja az is, hogy a

Microsoft cég már meg is vette, illetve a hazai fejlesztésű iGlue. Mindkettő még elég kezdetleges állapotban van, azonban mindkettőt ki lehet próbálni.

A Powerset kereső jelenlegi állapotában még csak a Wikipediában és a Freebase adatbázisában fellelhető információkat tudja hasznosítani, azt viszont nagyon hatékonyan. Keresőjébe egy keresendő kifejezést mellett akár egy konkrét mondatot is beírhatunk. A fő kérdés az, hogy hogyan sikerül majd ezt a technológiát átültetni a teljes internetre, és hogy ehhez mekkora erőforrásokra lesz majd szükség. Viszont a Microsoft által szerzett anyagi háttér sokat lendíthet azon, hogy ez a kereső minél hamarabb elérje végleges alapját, és elkezdődhessen a szemantikán alapuló keresés.

10. ábra: A Powerset kezelőfelülete
(www.powerset.com)

Ejtsünk most szót a másik szemantikus keresőről, az iGlue-ról. A kereső lelke az úgynevezett hyperdata-modell, amely bármilyen szöveget, adatot, képet, multimédiás adatot vagy adatbázist egy konkrét adatcsomóponttá olvaszt össze. Az iGlue elolvassa és felismeri a webes tartalmak jelentéssel bíró elemeit. Ezután egy gombnyomásra helyben felépíti az adott elemhez tartozó adatkapcsolati hálót. Ez a technológia is elég kezdetleges állapotban van még, azonban egy kis bemutató segítségével képet kaphatunk arról, hogy mit is fog kínálni majd ez nekünk. A demo során a New York Times napilap egy cikkén keresztül ismerhetjük meg az iGlue-t.



11. ábra Az iGlue bemutatója a New York Times cikkén keresztül

A jobb felső sarokban levő zöld gombot – iGlue annotate - egyszer lenyomva, kiemelésre kerülnek a névként értékelhető szövegrészletek. Újabb kattintás után pedig feljön egy különálló ablak, ahol az egyes szavakról tudhatunk meg többet. Csoportokba szervezi az azonos szófajú szavakat, majd azok közül kiválasztva a nekünk szükségeset, kapunk egy új ablakot, ahol részletes ismertetőt kapunk az adott szóról. A legújabb fejlesztés, hogy hibajelentést adhatunk le, ha szerintünk valamelyik információ helytelenül van megadva.

Láthatjuk tehát, hogy a szemantikus keresők már az „ajtóinkban állnak”, azonban még kell néhány év, hogy használatuk széles körben elterjedjen. Az elképzelések nagyon jók, és a felhasználók tetszését is biztosan elnyerik majd.

Összefoglalás

A világháló folyamatosan változik. Az elérhető információk mennyisége hatalmas ütemben nő, emiatt mind nagyobb szükség van az adatforrások rendszerezésére, összekapcsolására. A web 1.0 az online megjelenésről, az információk elhelyezéséről, a web 2.0 a közös tartalom létrehozásáról, közösségépítésről szólt. Elérkeztünk arra a pontra, hogy szükség lesz a web 3.0 bevezetésére. A fejlesztők és informatikusok körében nagy vita, hogy pontosan mi jelentse majd a 3.0-át. Az egyik, és talán a legnagyobb támogatottsággal bíró lehetőség a szemantikus web. Ennek célja, hogy a világhálón található információkat a számítógépek ne csak olvasni, hanem

értelmezni is tudják. Ennek érdekében egyrészt az információkhoz megfelelő metaadatokat kell társítani, valamint a számítógépeket képessé kell tenni arra, hogy a metaadatokkal kapcsolatos következtetéseket el tudják végezni.

Szakdolgozatom célja az volt, hogy bemutassam a szemantikus web hátterében álló technológiákat, amelyek használatával elérhetővé válik ez a még mindig álomszerűnek tűnő dolog. Leírtam, milyen közös nyelven történik az adattárolás, milyen lehetőségünk van az adatokhoz metaadatokat kapcsolni. Ezután bemutattam az adatokon való következtetések végrehajtásához szükséges eszközöket. Az utolsó fejezetben pedig megpróbáltam képet adni arról, hogy hogyan is áll a szemantikus keresők fejlesztése a jelenben.

A szemantikus web elterjedésének legfőbb akadálya azonban nem az, hogy a szemantikus keresők igen korai állapotban vannak még, hanem az, hogy nem nagyon léteznek szemantikus weboldalak. Az egyes lapok tartalmát ugyanis le kell írni a gép által is értelmezhető módon, ami egyrészt időigényes, másrészt egy egyszerű felhasználó nagy valószínűséggel egyáltalán, de legalábbis olyan mértékben nem ismeri az RDF és az OWL nyelveket, hogy ezt meg is tudja tenni. Ahhoz azonban, hogy ez a technológia tökéletesen tudjon a jövőben működni az kell, hogy minden egyes oldal mellé le legyen tárolva az összes információ. Ezt úgy lehetne elérni, hogy a szemantikát automatikusan legeneráltassuk valamilyen programmal, ami segítene ezeknek a felhasználóknak. Ezekből is látszik, hogy hiába van meg a legtöbb eszköz már évek óta arra, hogy a szemantikus web „betörjön” a köztudatba, nagy valószínűséggel még egyszer ennyi időt várnunk kell erre. Ha azonban eljön ez az idő, akkor nekünk, felhasználóknak készen kell állnunk arra, hogy befogadjuk a világháló új, emberi tudás központú szemléletét.

Irodalomjegyzék:

[1] Szeredi Péter – Lukácsy Gergely – Benkő Tamás: A szemantikus világháló elmélete és gyakorlata, TYPOTEX, 2005

[2] Gottdank Tibor: Szemantikus web – Bevezetés a tudásalapú internet világába, ComputerBooks, 2005

[3] Wikipedia, a szabad enciklopédia: hu.wikipedia.org

[4] Zombori Zsolt – Lukácsy Gergely – Szeredi Péter: Hatékony következtetés ontológiákon: https://nws.niif.hu/ncd2008/docs/phu/101.pdf

[5] Tudományos és Műszaki Tájékoztatás – Könyvtár-és információtudományi szakfolyóirat Tudásalapú információkinyerés: http://tmt.omikk.bme.hu/show_news.html?id=3616&issue_id=450

[6] Kereső Világ: kereses.blog.hu

[7] Herendy Csilla: Web 3.0 - A szemantikus web?: http://damjanovich.hu/cikkek/web3.0-a-szemantikus-web.html

[8] iGlue: http://blog.iglue.hu/

[9] W3C Consortium - Resource Description Framework: www.w3c.org/RDF

[10] W3C Consortium – SPARQL Query Language for RDF : http://www.w3.org/TR/rdf-sparql-query/

[11] W3C Consortium - Metalog, the semantic web query/logical system: http://www.w3.org/RDF/Metalog/

[12] W3C Consortium - RDF Validation Service: http://www.w3.org/RDF/Validator

[13] W3C Consortium - OWL Web Ontology Language: http://www.w3.org/TR/2004/REC-owl-guide-20040210/

[14] PML – Proof Markup Language: http://iw.stanford.edu/2006/tami/TR-cwm-pml-translation.htm

[15] The Protégé Ontology Editor and Knowledge Acquisition System: http://protege.stanford.edu/