Debreceni Egyetem

Informatika Kar

Szemantikus Web

Szakterületi Glosszárium

Témavezető:

Dr. Adamkó Attila

Egyetemi tanársegéd

Készítette:

Nagy Géza

Programtervező informatikus

Debrecen

2009

Tartalomjegyzék

Bevezetés

Szakdolgozatom témája a szemantikus web, amely bár hatalmas lehetőségeket rejt magában, ezek a lehetőségek a mai napig mind-mind kiaknázatlanok maradtak. Dolgozatomban szeretnék rávilágítani azokra a lehetőségekre, amelyek naggyá tehetik a szemantikus web technológiát, valamint a felmerülő kérdésekre és problémákra, ezek orvoslására, illetve elkerülésére próbálok kitérni.

Szeretnék elsősorban a szemantikus webre koncentrálni, mint fogalom, ismertetni picit magát az elképzelést, amely létrehozta, a működési koncepcióját, gyakorlati hasznát, valamint azt, hogy ezt egy alkalmazásban hogyan lehet hasznosítani.
Elsősorban rendkívül fontos, hogy megismerjük miért is jött létre maga a szemantikus web, illetve megalkotóikat milyen elvek vezérelték mikor lefektették az alapjait, és a meglevő koncepciót mennyire sikerült áttenni a gyakorlatba.
Meg kell továbbá ismerkednünk a módszerrel, amivel a fellépő plusz információkat tárolni tudjuk a számítógép által is ismert formában, úgy hogy az számára is egyértelműen érthető, könnyen beazonosítható, és feldolgozható legyen.

Elsősorban tehát egy átfogó, és alapvető ismertetést szeretnék adni a szemantikus webről, és annak lehetőségeiről. Természetesen ebben a kontextusban az alapvető szót nem a minimalista értelemben használtam, hiszen ha nem látjuk az összefüggéseket, amik e nyelvkörnyezetben, illetve a szabványban használatosak, akkor nem érthetjük meg teljes mértékben a fő célt, ennek folyományaként az alkalmazás mivoltát sem.

Tudásmegosztás, tudásmenedzsment:
2009-et írunk, ugrásszerűen megnőtt az otthonokban a számítógépek száma, 2005 óta szabadon nevezhetjük a végbement változást „internet-forradalom”-nak. Mára már több mint egymilliárdra rúg az otthoni internethasználók száma, és ez a szám egyre csak növekszik, mégis azt látjuk, hogy az internet felhasználók nincsenek teljesen tisztában a benne rejlő lehetőségekkel.
Szakdolgozatom szempontjából az egyik legfontosabb fogalom a tudásmegosztás. Az interneten minden tudás elérhető. Van, ami csak pár klikkre lakozik tőlünk és van, amit csak

hosszas keresgélés után sikerül felkutatnunk. Napjainkban, az informatika virágkorában a tudást, és az azt megjelenítő honlapokat számítógépek tárolják. A számítógépen tárolt információknak az a hátrányuk, hogy emberek számára tervezték. Ez a mondat felveti a következő kérdést:

„Hogyan lehet információt tervezni?”

vagy még inkább

„Hogyan lehet számítógépek számára információt tervezni?”

Tervezés alatt itt nem a konkrét megalkotást értem, hanem az információ strukturálását. Elsődleges probléma a strukturáláskor, hogy az emberi agy lehetőségei információ feldolgozás tekintetében közel határtalanok, míg egy számítógépes processzornak megvannak a korlátai, így az információ feldolgozásában sem lát semmi különöset, mint alfanumerikus sorozatokat. Ha esetleg egy olyan karaktersorozatba botlik, amelyet tud értelmezni (pl.: HTML Tag, utasítás, stb.…), tudja, mit kell tennie, de számára ez továbbra sem információ lesz, hanem egy felismert alfanumerikus karaktersorozat, levegőben lógó, egy szinten levő parancsok illetve fogalmak sorozata, amelyekről a számítógép nem tudja megállapítani sem a kontextusát, sem a hozzá kapcsolódó fogalmakat.

Pontosan úgy, mint ahogyan egy szótárban a szavak, amelyekhez éppen csak a pontos jelentésüket csatolják. Semmit nem tudunk meg sem a szóról, vagy kifejezésről, csak annyit, hogy szó szerinti fordításban mit jelent. Sem a kapcsolatait más szavakkal, illetve azt, hogy más szövegkörnyezetben használva esetleg milyen jelentéseket hordoz.
Ezek mind-mind olyan dolgok, amelyeket az emberi agy egy szó hallatán értelmez, felfog, és hozzárendel az adott szóhoz, hogy abból kifejezés, illetve fogalom legyen.
Információ tervezésekor struktúrák létrehozására kell törekednünk. Mint hogy a szótárnál az egyre kisebb elemekből (szavak) épülnek fel a bonyolultabb dolgok (kifejezés), úgy a számítógép számára is szeretnénk meghatározni, hogy azok a dolgok, amiket ő értelmez nem csak levegőben lógó szavak, hanem kapcsolódnak valamihez, egyéb kifejezésekkel kapcsolatban állnak, következtetni lehet rájuk.
A fenti módszert használva, létrehozhatjuk az információnak egy struktúráját. Az egyes szavak fölött így témakörök állnak, amelyek segítségével könnyen tudjuk csoportosítani az

egyes szavakat, így valamihez társítva őket, egy újabb tudással ruházhatjuk fel a számítógépünket (az egy csoportba tartozó szavak).

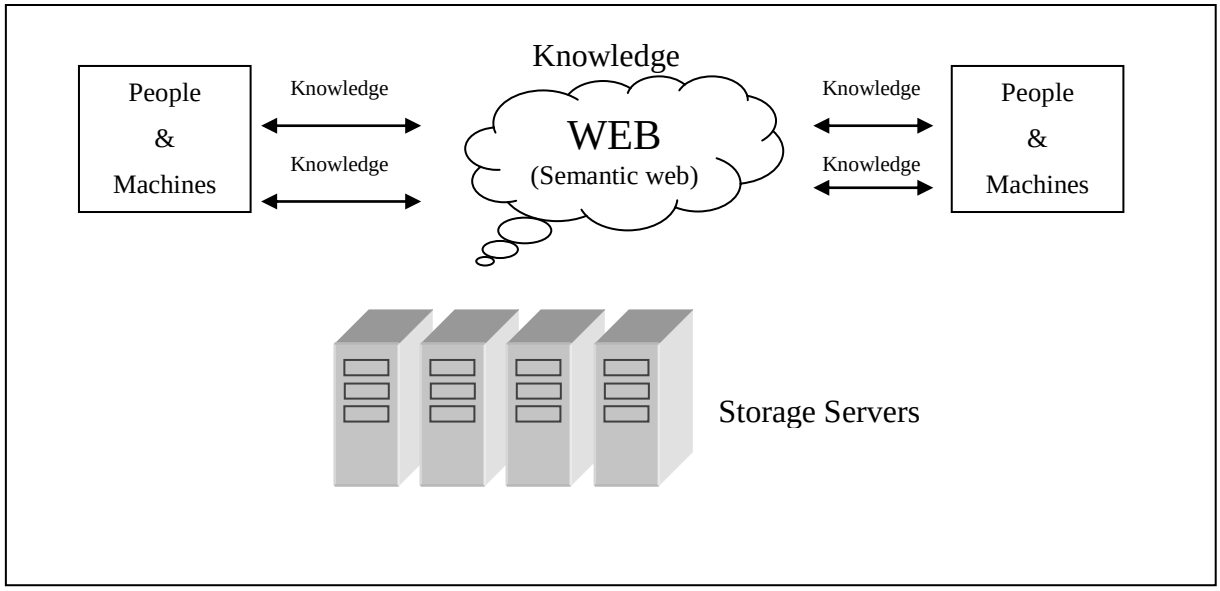
Ez az elv, hogy többet tároljuk mást is a környezet dolgairól, mint csak azokat amik közvetlenül hozzá kapcsolódnak, vezetett a szemantikus web kialakulásához.

A Szemantikus Web

A szemantikus webről általánosan

A szemantikus web kialakulása a WWW Consortium nevéhez kötődik. A **World Wide Web Consortium (W3C)** egy konzorcium, mely nyílt szoftver szabványokat – „ajánlásokat", ahogy ők hívják – alkot a világhálóra (World Wide Web). A konzorciumot Tim Berners-Lee vezeti, az URI (Uniform Resource Identifier, univerzális erőforrás azonosító), a HTTP (HyperText Transfer Protocol) és a HTML (HyperText Markup Language, hipertext leíró nyelv) eredeti megalkotója. Ezek a technológiák alkotják még ma is a Web alapjait.

A szemantikus web a világháló egyik legdinamikusabban fejlődő része, ahol az információ szemantikája és funkciója definiálva vannak, ezáltal lehetővé téve azt, hogy a számítógép megértse, és ki tudja elégíteni azokat az igényeket, amelyeket a felhasználók, esetleg más számítógépek böngészés közben támasztanak.

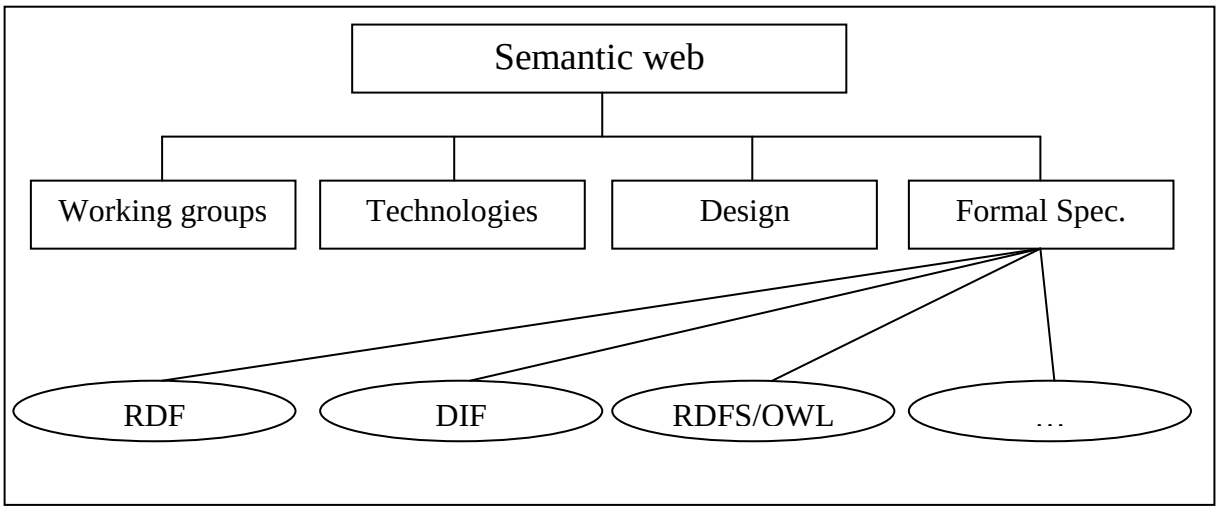


1. ábra **Sir Tim-Berners Lee elképzelése a szemantikus webről**

A Web ilyen módon történő felhasználása a WWW Consortium elnökétől Sir Tim Berners-Lee-től származik, aki a Webre úgy tekint, mint egy univerzális médiumra, amely képes adatokat és információkat úgy tárolni, hogy a tudásmegosztás legfontosabb bázisává váljon.

Alapjaiban véve a szemantikus web megjelenítési szabályok, közösen dolgozó munkacsoportok, valamint az őket segítő különféle technológiák együttese. Néhány eleme mind a mai napig csak előremutató lehetőségként van definiálva, annak megvalósítása, illetve kidolgozása, illetve implementálása még várat magára.

A szemantikus web további elemei formai specifikációkból állnak, mint például az RDF (Resource Description Framework), adatátviteli, adattárolási formátumok (RDF/XML, N3, …), és jelölési rendszerek, mint az RDF séma (RDF Schema, RDFS), és az OWL (Web Ontology Language).



2. ábra **A szemantikus web felépítése**

Mindezeket arra használja, hogy egy formális keretet adjon az elképzeléseknek, tényeknek, és a kapcsolatoknak egy tudásbázison belül.

Szemantikus web alkalmazása

Lássunk tehát néhány egyszerű alkalmazást, ahol tetten érhetőek a szemantikus web előnyei. A dokumentumkezelés, mint olyan, rengeteg merít a szemantikus web technológiájából.

Az üzleti és közigazgatási szervezetek, működésük folyamán, létezésük egyértelmű bizonyítékaként jelentős kisebb vagy nagyobb mennyiségű iratot, dokumentumot állítanak elő és hagynak maguk mögött. Ezeknek a dokumentumoknak (vagy másolataiknak) jelentős részét törvényi előírások miatt, más részét pedig a szervezet saját döntése és megfontolása alapján azonban hosszú éveken át meg kell őrizni. A hagyományos és széles körben elterjedt „dobozolós" technikát, nevezhetjük archívum-kialakításnak vagy archiválásnak.

Az archiválás a maga teljességében két világra terjed ki. Egyik fele - mint a dokumentumkezelés egésze is - a való világban létezik. Az archiválás ezen részével, tudnivalóival a hagyományos (tradicionális) vagy fizikai iratkezelés foglalkozik. A fizikai iratkezelő cégek varázsolnak rendet az irattárainkban, alkotják meg iratkezelési szabályzatainkat, hogy sorba rendezzék az irataink.
Az archiválás másik része a szép, új, digitális világban foglal helyet. Ennek a világnak a problémáira az elektronikus iratkezelés, elektronikus dokumentumkezelés kínál megoldásokat. A rendteremtés legalapvetőbb eszköze az iktatás elektronizálása, azaz egy jó iktatóprogram. Ennek bevezetése és működtetése után jöhet egy archiváló szoftver.

Ha megérett az elhatározás, hogy elbánunk a papírhegyekkel és a megfelelő hardver és tároló kapacitás is rendelkezésre áll, illetve beszereztük az archiváló szoftvert egy dolog van hátra. Az utolsó, de legalább olyan fontos dolog a megfelelő folyamatok kialakítása vagy a meglevők átalakítása.

A cégek vezetői hamar beláthatják, hogy a papír alapú tárolás helyigényes és igen drága. Az elektronikus archívum bevezetése viszont belátható időn belül megtérülhet. A jól kifejezhető irattári költségekkel ellentétben a biztonság például megfizethetetlen. Akinek azott már el irattára, vagy vált a lángok martalékává, az nagyon jól tudja, hogy mennyire megkönnyíti az szervezet munkájának folytatását, ha "kéznél" van a digitálisan archivált dokumentumtár.

Egy ilyen dokumentumtár szerkezeti struktúrájának megtervezésekor sokat meríthetünk a Szemantikus Web köré épülő technológiákból. Például nagyok könnyen csoportosíthatjuk a beérkezett dokumentumokat a témák segítségével, amelyek mind, mint erőforrás jelennek meg az alkalmazásban. Az egyes dokumentumokat hamar megtalálhatjuk mivel pontosan rendszerezve vannak, és az esetlegesen hozzá csatolt más elemeket (másolatok, hitelesítő adatlapok, kitöltendő kérvények, stb.…)

Ontológia, OWL

Az emberek a webet olyan dolgokra használják, mint megkeresni egy online szótárban egy szót, megrendelni egy vacsorát, vagy megkeresni egy olcsó DVD-t az interneten.

Azonban egy számítógép minderre nem képes emberi beavatkozás nélkül, mert a weboldalak úgy lettek tervezve, hogy emberek olvassák ne számítógépek. A szemantikus web az információk egy olyan halmazát jelenti az interneten, amelyek érthetőek a számítógép számára. Ezt az információt arra tudja felhasználni, hogy mélyrehatóbban tudjon vizsgálódni, és segítséget nyújtson a keresésben, megosztásban, valamint az információk kapcsolatának megjelenítésében.

> „Van egy álmom a Webről, ahol a számítógépek elemezni tudják az összes adatot az interneten – a tartalmat, a kapcsolatokat, és a számítógép illetve emberek közti kommunikációt. Ezt egy 'Szemantikus web' teheti elérhetővé, de mikor ez bekövetkezik, az olyan mindennapos dolgokat, mint a vásárlás vagy a kommunikáció, valamint a napjaink megszokott életét nagyban fogja vezérelni egy olyan rendszer, ahol számítógép kommunikál számítógéppel. Az 'Intelligens ügynökök' kora, amelyekben az emberek régóta kételkedtek, el fog érkezni"

Részlet *Sir Tim-Berners Lee, New York: „Az internet jövője" című konferencián mondott beszédéből 1999. október*

A Szemantikus publikáció rengeteget profitál a szemantikus webből. Tulajdonképpen a szemantikus web teljesen megreformálhatja a tudományos publikációkat, mint például a valós idejű publikálást, és a kísérletek eredményeinek azonnali közzétételét az interneten.

Ontológia

A Szemantikus Webnek szüksége van ontológiák felépítésére.
„Az ontológia kifejezéseket és összefüggéseket határoz meg egy adott tudásterület leírásához"

A cél: Egy Webontológia nyelv („Web Ontologies Language”), amely a következőkön alapszik:

- RDF és RDF Sémák
- korábbi munkák:
    - **DAML (DARPA projekt):** A DARPA Agent Markup Language egy amerikai program neve volt, amit a Védelmi Minisztérium (DARPA) indított 1999-ben. A program célja egy olyan Web megalkotása volt amely a Web-et számítógépek által is olvashatóvá kívánta tenni. Az programban dolgozó egyik személy maga Tim Berners-Lee volt, akinek hathatós közreműködése révén létrejöttek olyan technológiák és módszerek, amelyek ma a szemantikus web alapjait képezik.

      A legfontosabb eredménye a programnak a DAML nyelv létrejötte volt, amely RDF-en alapuló leíró nyelv.
    - **OIL (EU projekt):** Az OIL-ra (Ontológia Levezetési Réteg, vagy Ontológia Cserélő Nyelv) úgy tekinthetünk, mint egy ontológiai infrastruktúrára a szemantikus webhez. Az OIL leíró logika-beli következtetéseken, és keret alapú rendszereken alapszik. Egyik legfőbb erénye, hogy kompatibilis az RDFS-el.
      Az OIL-on elvégzett munka nagy részét már integrálták a DAML+OIL-ba, valamint az OWL-be.
    - **DAML+OIL (DAMN és OIL egyesítése):** A DAML+OIL egy formai szabályrendszer, ami az RDF-en és az XML-en alapszik, melyek segítségével könnyen szabályhalmazokat definiálhattak. Ezen szabályhalmazok összevonásával, és egyesítésével később egy ontológia született.
      A legfőbb innovációja a nyelvnek az volt, hogy az RDF-et használta az alapjául, és az RDF névterek segítségével tudta csoportosítani és egymáshoz illeszteni az egymástól nagyon eltérő ontológiákat is.
- a logika, tudásreprezentáció, stb., gyakorlati eredményei

A számítástechnikában és az informatikában egy ontológia a formális reprezentációja elképzeléseknek és a köztük levő kapcsolatoknak egy szakterületen belül. Arra használják,

hogy az adott szakterülethez tartozó tulajdonságokat, illetve kifejezéseket összefogja, illetve ritkább esetben az egész szakterületet definiálja.
Elméletben egy ontológia egy formailag pontosan definiált megosztott elképzelésvilág. Egy ontológia egy megosztott szótárat használ, amelynek segítségével modellezni tudjuk az adott szakterületet, vagyis olyan objektumokat biztosít számunkra, amelyek már ténylegesen léteznek, ezáltal a tulajdonságaik is és a köztük levő kapcsolatok is.

Az ontológiákat sok területen használják, a mesterséges intelligencia, Szemantikus Web, szoftverfejlesztés, bioinformatika, könyvtártudományok is profitálnak létezéséből.

Az ontológia szó a filozófiából származik, és sokféleképpen értelmezték eddig. Az informatikában úgy értelmezzük, mint egy modellt típusokkal, tulajdonságokkal, és kapcsolattípusokkal, amelyet a világ leírására használunk. Az hogy ezek pontosan hogyan néznek ki, vagy mit is jelentenek változó, de ezek mind lényegi elemei egy ontológiának. Tulajdonképpen elvárás is, hogy szoros kapcsolat legyen a valós világ és az ontológiai modell egyes részei között

Az ontológiában, mint számítástechnikai fogalomban, és az ontológiában, mint filozófiai fogalomban sok a közös vonás. Leginkább úgy lehetne megfogalmazni, hogy az entitások, ötletek, események megjelenítése ezek tulajdonságaival és egymás közti kapcsolataival együtt, egy kategorikus rendszer keretén belül. A két területen alkalmazott ontológia közti különbség nagy része szemléletbeli. A filozófusok sokkal kevésbé hisznek a megkötött, irányított szókészletekben, mint amennyire az informatikus kutatók, míg ez utóbbiakat kevésbé érdekli az elsődleges elvek kérdése.

Az OWL kialakulása

Hosszú története van az ontológiai fejlődésnek mind a filozófiai mind a számítástechnikai értelemben. Az 1990es évektől kezdve rengeteg kutatást végeztek, hogy a mesterséges intelligencia területén megjelenő tudásreprezentációt hogyan hasznosíthatnák az Interneten. Ennek folyományaként jelentek meg a HTML alapú nyelvek (SHOE), az XML (XOL, később OIL), és még rengeteg tudásreprezentáló nyelv.

Az OWL nyelv is egy ilyen kutatás alapján született meg, a DAML és az OIL (DAML+OIL) web ontológia nyelvek keresztezésével. A DAML+OIL –t egy angol-amerikai csoport alkotta meg, amelyet az Amerikai Védelmi Minisztérium kutatással foglalkozó részlege (DARPA) hívott életre Joint Working Group on Agent Markup Language néven.

A WWW Konzorcium létrehozta ennek hatására a „Web Ontológia Munkacsoportot”, amely 2001. November elején kezdte meg munkáját James Hendler, és Guus Schreiber vezérletével. Az absztrakt szintaxis első működő vázlatait, forrásait, és eredményeit 2002 júliusában publikálták. Újabb két év múlva a dokumentumok a W3C ajánlásává váltak, ezután nem sokkal feloszlatták az OWL-t megalkotó csoportot.

OWL ontológiák

Az információk, amiket egy OWL ontológiával írunk le úgy jelennek meg, mint egyedek és tulajdonságok halmaza, és a tulajdonságok egymáshoz rendelése, amelyek az egyedek közti kapcsolatokat reprezentálják. Egy OWL ontológia axiómák sorozatából áll, amik szabályokat fogalmaznak meg az egyedhalmazokra (osztályok) és a köztük megengedett kapcsolatokra vonatkozóan. Ezen axiómák segítségével egy szemantikát adunk meg, amit a többi rendszer arra használ fel, hogy extra információhoz jusson a meglévő adatokból való következtetéssel.

Vegyünk egy családokat leíró ontológiát:
Tartalmazhat olyan axiómát, ami megmondja, hogy egy „Édesanyja” tulajdonság csakis akkor van jelen, hogyha a „Szülők” tulajdonság is jelen van. Vagy a „0sVércsoportú” osztályba tartozó egyedek a „Szülő” tulajdonsággal sohasem lesznek az „ABsVércsoportú” osztályba tatozó egyedekhez kötve.

Tehát, ha veszünk két egyedet E1-et és E2-t, és ezeket összekapcsoljuk az „Édesanyja” tulajdonsággal, és E1 tagja a „0sVércsoportú” osztálynak, akkor következtethető, hogy E2 nem tagja az „ABsVércsoportú” osztálynak.

Leíró logika

A nagy ontológiák létrehozására irányuló kísérletek rendre kudarcba fulladtak az egyértelműen meghatározott definíciók hiánya miatt. Például létezett 18 féle értelmezése egy „IS-A” kapcsolatnak. Az OWL megpróbálja ezt elkerülni azzal, hogy meghatározza a nyelv logikai alapjait, amely a leíró logikán alapszik.

A leíró logika az elsőrendű logika eldönthető részkifejezéseinek családjába tartozik. Az OWL szemantikája egy megadott leíró logikára való fordítás. Ezáltal az OWL egy szintaktika is az ontológiák leírására és megosztására, és van egy formálisan meghatározott szemantikája, amely megadja a jelentését is.

Alnyelvek

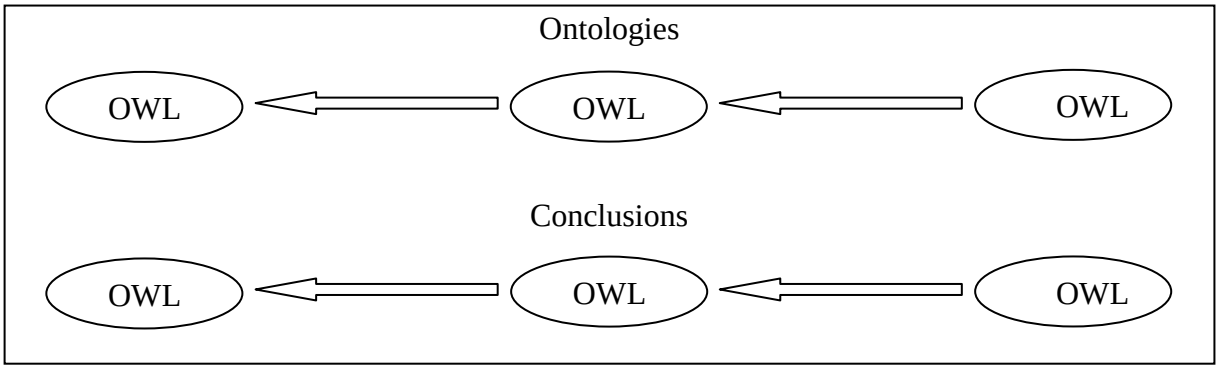
A W3C által patronált OWL specifikáció 3 féle OWL variánst definiál, mindegyik eltérő szintű kifejezésmóddal.

- OWL Lite: Eredetileg azokat számára lett kifejlesztve, akiknek elsősorban egy osztályhierarchia rendszerre volt szükségük egyszerű megszorításokkal. Például amíg csak számossági megszorításokat definiál, addig csak számossági értékeket engedélyez, például 0 vagy 1. Azt remélték, hogy az OWL Lite-hoz sokkal egyszerűbb lesz eszközrendszert fejleszteni, mint az OWL kifejezőbb részeihez, ezáltal könnyen és gyorsan alkalmazni lehet majd egy fogalomtárban például, vagy egy osztályozást nem használó rendszerbe. A gyakorlatban azonban a legtöbb kifejezésre irányuló megszorítás az OWL Lite-ban szintaktikai kellemetlenségekhez vezetett: A legtöbb OWL DL-ben megtalálható struktúra előállítható OWL Lite-beli elemeket kombinálva. Mindezek mellé az OWL Lite eszközök fejlesztése majdnem olyan nehéznek bizonyult, mint az OWL DL eszközöké, és emiatt az OWL Lite sohasem lett igazán elterjedt.
- OWL DL: A lehető legmagasabb szintű kifejezőeszközöket vonultatja fel, míg megtartja a teljességet (minden következtetés garantáltan levezethető), az eldönthetőséget (minden levezetés véges hosszú idő után befejeződik), és a

használatos következtetési algoritmusok alkalmazhatóságát. Az OWL DL tartalmazza az összes OWL nyelvi részt, de ezek csak bizonyos szabályok mellett használhatóak (például nem tehetünk megszorításokat olyan tulajdonságokra amelyeket tranzitívnak defniáltunk). Az OWL DL-t szándékosan úgy nevezték el, hogy utaljon a kapcsolatára a leíró logikával (Description Logic).

- Az OWL Full egy teljesen más szemantikára épül, és arra tervezték, hogy valamiféle kompatibilitást mutasson az RDF sémával. Például az OWL Full-ban egy osztály értelmezhető egyedek kollekciójaként, illetve mint egy egyed önmagában. Mindez nem megengedett az OWL DL-ben. Az OWL Full megengedi, hogy kiegészítsük a jelentését az egyes kifejezéseknek a szótárban legyen az RDF- vagy OWL-beli. Komplexitása miatt eléggé valószínűtlen, hogy fog készülni olyan alkalmazás, amely teljes mértékben támogatni fogja az OWL Full komplett eszközrendszerét.

Mindegyik alnyelv egy szintaktikai kiterjesztése az alapnyelvének. Az alábbi kapcsolatok értelmezhetőek az egyes nyelvek között:



3. ábra **Az egyes OWL alnyelvek közötti kapcsolatok**

- Minden érvényes OWL Lite ontológia egyben egy érvényes OWL DL ontológia
- Minden érvényes OWL DL ontológia egyben egy érvényes OWL Full ontológia
- Minden OWL Lite-ban levezetett következtetés egy érvényes OWL DL-beli következtetés
- Minden OWL DL-ben levezetett következtetés egy érvényes OWL Full-beli következtetés

Figyeljük meg, hogy míg ezek az állítások helytállóak, ezek megfordításai nem.

Az OWL 2-ben három alnyelvet definiáltak:

- OWL 2 EL
- OWL 2 QL: Az adatbázisokban tárolt adatokhoz való könnyebb hozzáférhetőséget segíti
- OWL 2 RL: Az OWL 2 szabályaival foglalkozó rész

Nyílt világszemlélet

Az OWL a nyílt világszemléletet vallja, az SQL-el és a Prolog-al ellentétben, amelyek a Zárt világszemléletet. Ezalatt azt értjük, hogy ha egy állításról nem tudjuk bebizonyítani, hogy az adott ismeretek mellett igaz, attól még nem vonhatjuk le automatikusan azt a következtetést, hogy az állítás hamis.

A Rövidítés

A logikusan következő betűszó a Web Ontológia Nyelvre a WOL lenne (Web Ontology Language), mégis OWL-nek rövidítjük. Ennek egyszerű okai vannak, mégpedig az, hogy egy könnyen kiejthető rövidítés, amelyhez könnyedén lehetett logókat tervezni, és nem utolsó sorban tisztelegni William A. Martin Világnyelv (One World Language) című kutatása iránt, amelyet az 1970-es években végzett. És hogy Guus Schreibert idézzük:

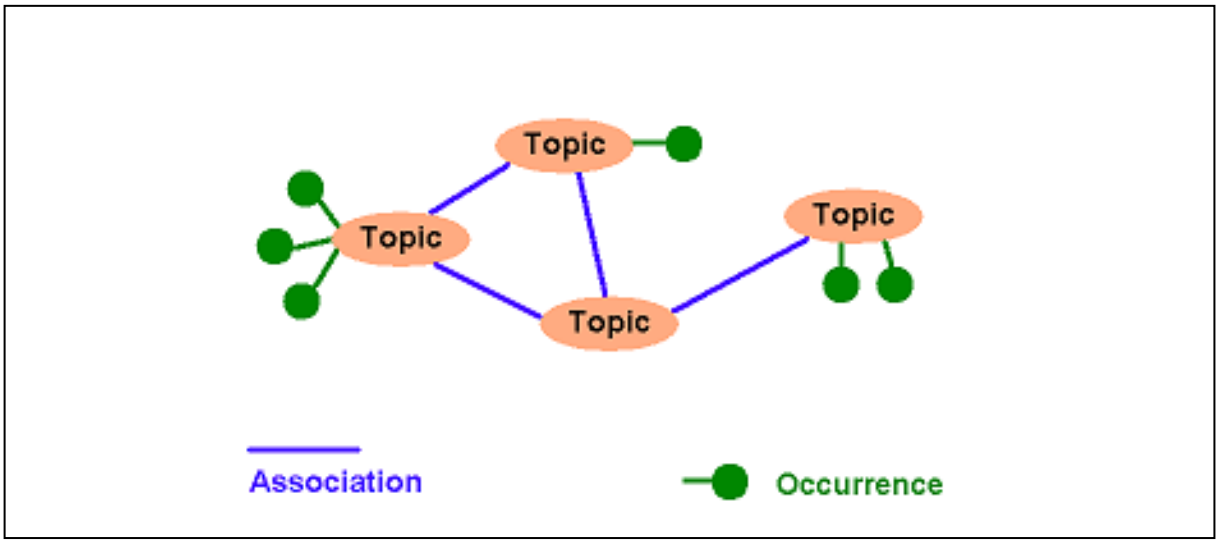
> „…Miért ne lehetnénk egy olyan nyelvben legalább egy szemléletből inkonzisztensek, ahol minden egyes apró részlet a konzisztenciáról szól…”

Guus Schreiber

Szemantikus web leíró nyelvei (Topic Maps, XTM, RDF)

Általános összefoglaló a Topic Maps-ről

A Topic Maps egy tudásreprezentációt illetve tudásmegosztást előtérbe helyező szabvány, amely a legfőbb hangsúlyt az információ megtalálhatóságára fekteti.



4. ábra **Topic Maps alapkoncepció**

Egy tématérkép (topic map) az információt témák (elképzelések, tervezetek, szervezetek, szoftver modulok, fájlok, események, stb.), asszociációk (a témák közötti kapcsolatok), és előfordulások (információk, amelyek egy adott témához fontosak) segítségével jeleníti meg.

A tématérkép tágabb értelemben a szemantikus web technológia egyik alkalmazása az weben, és egy kis munka árán akár az átjárhatóságot is biztosíthatjuk a W3C által megalkotott RDF/OWL szemantikus web szabványok és a Tématérképek között. A Tématérképek hasonlóak a koncepcionális térképekhez, és az agyi térképekhez (Mind maps) sok értelemben, legfőbb különbség köztük azonban az, hogy a Tématérképek szabványosítottak.

A legfrissebb kutatások a Topic Maps szabványosításában az ISO Topic Maps (ISO/IEC 13250) bizottság ellenőrzése alatt folynak.

XML Szerializációs formátumok

2000-ben a Topic Maps-hez definiáltak egy XML szintaxist, amelyet XTM-nek nevezünk. Ma ezt XTM 1.0-ként ismerjük, és még mindig eléggé elterjedt. Az ISO szabványokkal foglalkozó csoport 2006-ban publikált egy továbbfejlesztett XML szintaxist, amelyet elődje után XTM 2.0-nak neveztek el, és ma már egyre gyakrabban használják.

Megjelentek azonban egyéb szabványok vagy elképzelések, amelyek már vagy be lettek építve a szabványba, vagy még csak javasolva lettek:

- CXTM – Canonical XML Topic Maps formátum
- CTM – Compact Topic Maps Notation, amely nem is XML alapú
- GTM – Graphical Topic Maps Notation

Ezeken kívül léteznek még szerializációs formátumok, mint például az LTM, vagy az AsTMa, ám ezeket még nem szabványosították.
A Topic Maps referenciamodell (TMRM) és adatmodell (TMDM) szabványok úgy vannak definiálva, hogy függetlenek legyenek akármilyen szerializációtól, illetve szintaxistól.

Topic Maps API

Létezik a Topic Maps-nek egy de facto fejlesztői szabványa, a TMAPI (Common Topic Maps Application Programming Interface), amelyet 2004-ben publikáltak, és sok Topic Maps-el foglalkozó gyártó, vagy cég támogatja a saját implementációjában. Ennek is megjelentek további verziói, csakúgy mint az XTM-nek:

- TMAPI – Common Topic Maps Application Programming Interface
- TMAPI 2.0 – Topic Maps Application Programming Interface (v2.0)

Lekérdező nyelv

Normál használat esetén gyakran van szükségünk arra, hogy lekérdezzük egy adott Topic Map reprezentációban tárolt információt. Sok implementáció lehetőséget ad nekünk ennek elvégzésére (olyan ez mintha lenne egy SQL a Topic Maps-hez), a probléma ezzel viszont az, hogy a parancsok szintaxisa rendre más és más a különböző megvalósításokban. Ennek

kiküszöbölésére létrehoztak egy szabványosított lekérdező nyelvet a Topic Maps-hez: a TMQL-t (Topic Maps Query Language).

Megszorítási szabványok

Az is elvárható normális esetben, hogy megszorításokat definiálunk annak érdekében, hogy a szemantikus helyességet ellenőrizzük egy adott terület tématérképében (kicsit hasonlít ez is az adatbázisoknál használt megszorításokra). A megszorításokat arra használhatjuk, hogy megmondjunk olyan dolgokat például:

- Minden dokumentumnak kell, hogy legyen szerzője, vagy
- Minden menedzsernek embernek kell lennie, stb.…

Ezen dolgok betartását gyakran elérhetjük az egyedi implementációk által felkínált lehetőségekkel, de ezek megint csak egyedi megoldások. Így ezek kiküszöbölésére egy újabb szabvány született: a TMCL (Topic Maps Constraint Language).

Korábbi szabványok

A tématérképek elképzelése elég régóta létezik már. A HyTime szabvány például már 1992 óta jelen van (egyes források szerint régebb óta). Az ISO 13250-es szabványnak, amely jelenleg is érvényben van, szintén léteztek korábbi verziói.

Ontológia és az Egyesítés művelete

A témák, asszociációk, és előfordulások mind tipizálhatóak, ahol a típusokat kötelezően definiálnia kell a tématérkép(ek) létrehozójának. A használható típusok definícióját nevezzük egy tématérkép ontológiájának.

A Topic Maps szerkezeténél fogva támogatja az azonosság alapú összefésülés/egyesítés műveletét két téma vagy tématérkép között. Továbbá, mivel az ontológiák is tématérképek, ezeket is össze lehet fésülni, ezáltal lehetővé válik az információ integrálása külön forrásokból egyetlen új tématérképbe. Az olyan lehetőségeket, mint például a téma-azonosítók (URI-k), és a publikus téma-azonosítók (PSI-k), arra használják, hogy az összefésülést vezéreljék a

különböző osztályozás nélküli témák között. A nevekre koncentrálva lehetőségünk nyílik rendszeresíteni a egy témának adott különböző forrásokból származó különböző neveket.

Adatformátum

> „Ez a szabvány modellt, és nyelvtant biztosít az információ erőforrások struktúrájának megalkotásához, amelyek a témák és a témák közötti asszociációk (kapcsolatok) definiálásához vezetnek. A nevek, erőforrások, és a kapcsolatok egy absztrakt objektum jellemzői, amelyeket témának hívunk. A témák jellemzői csak a saját nézetükben (vagyis abban a környezetben, amelyben hivatkozhatunk rájuk a név, erőforrás, kapcsolat

4. ábra **Részlet a Topic Maps szabványból**

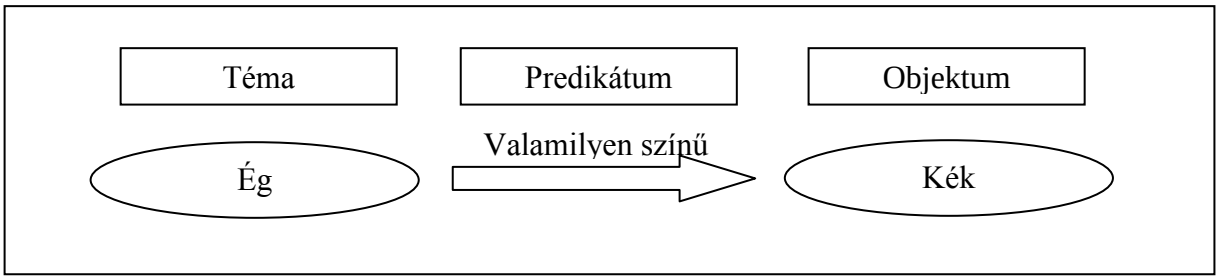
Egy Lineáris Tématérkép nevű formátum egyfajta írási szintaktikát biztosít nekünk, ha a tématérképeket hagyományos szövegszerkesztővel szeretnénk rögzíteni. Ez hasznos abban az esetben, ha rövid, személyes tématérképeket szeretnénk készíteni, vagy egy nagyobb tématérkép egy részét szeretnénk elküldeni e-mail-ben. A formátum további előnye, hogy könnyedén átkonvertálható XTM formátumúvá.

Egy másik tématérkép formátum az AsTMa, amely szintén hasonló célokat szolgál mint az LTM. Ha kézzel írjuk a tématérképeket, akkor sokkal előnyösebb, és természetesen ez is konvertálható XTM formátumúvá. Mindkét formátum mellett szól még egy előny, mégpedig az, hogy közvetlenül használható a Pearl TM moduljával.

RDF

Alapvetően az RDF adatmodell nem különbözik a klasszikus koncepcionális modellektől mint például az ER modell, vagy az osztálydiagramok, mivel azon az elven alapul, hogy kijelentéseket teszünk erőforrásokról, általában webes erőforrásokról téma-predikátum-objektum összefüggésben. Ezeket a kifejezéseket hármasoknak (triplets) nevezzük az RDF fogalomtárában. A téma jelöli az erőforrást, a predikátum pedig nézeteket, álláspontokat jelöl az erőforrásról, és ezáltal reprezentál számunkra egy kapcsolatot a téma és az objektum között.

Például azt a mondatot, hogy ’Az ég kék színű’ RDF hármasokkal az alábbi módon írhatjuk le:



5. ábra **RDF Triplet**

Az RDF egy absztrakt modell, rengeteg szerializációs formátummal (vagyis fájlformátummal) rendelkezik, és ezáltal a közvetlen mód, ahogyan ezeket a hármasokat tárolják változik minden egyes megvalósításnál.
Az erőforrások leírásának ezen módja meghatározó eleme a W3C Szemantikus Web projektjének: az internet korszakának egy új fázisa, amelyben egy automatizált szoftver tárolja, illetve ezeket az ismereteket felhasználva új, számítógép által is olvasható információkra tesz szert az interneten közzétett hatalmas információáradatból, ezáltal elősegítve azt, hogy a felhasználók sokkal hatékonyabban tudják használni a Web-et.
Az RDF egyszerű adatmodellje, és azon tulajdonsága, hogy absztrakt elképzeléseket is tud modellezni ahhoz vezetett, hogy egyre inkább használják a tudásmenedzsment területén is.

Az RDF állítások együttesen valójában egy címkézett irányított gráfot definiálnak. Egy RDF alapú adatmodell mint olyan sokkal jobban illeszkedik a tudásmegosztás és tudásmenedzsment elveihez, mint akár a relációs modell, akár más mai számítástechnikában használt ontológia modellek valamelyike. Mindazonáltal a gyakorlatban az RDF adatokat gyakran tárolják relációs adatbázisokban, vagy más helybeli megoldásokkal, úgynevezett hármas tárolókban (Triple Store), amely egy speciális adatbázis az RDF hármasok tárolására, vagy négyes tárolókban (Quad store), ha tárolni szeretnénk azt is, hogy melyik gráfban van megjelenítve az adott hármas.

Szerializációs formátumok

Két gyakori szerializációs formátumot használunk. Az első az XML formátum. Ezt gyakran nevezzük csak szimplán RDF formátumnak, mert amikor bemutatkozott az RDF mint W3C szabvány, ez már a része volt. Azonban fontos, hogy megkülönböztessük az XML file-t az absztrakt RDF modelltől. A file formátumának leírását az RFC 3870-es RFC dokumentumban találjuk, és azt javasolja, hogy az RDF dokumentumok kövessék az új, 2004-es specifikációt.
A RDF modell XML-be való tárolása mellett a W3C létrehozta a Notation3-at (N3), amely ugyan nem egy XML szerializáció, de ha kézzel szeretnénk leírni egy RDF modellt, akkor sokkal egyszerűbb dolgunk van vele, és néhány esetben az értelmezése is könnyebb. Mivel a táblázatos jelölésre épít, ezért az általuk jelölt hármas sokkal könnyebben értelmezhető, mint egy XML állományban. Az N3 a Turtle, és az N-Triples szerializációs formátumok része.

Erőforrás azonosítás

Egy RDF állítás tárgya vagy egy URI, vagy pedig egy úgynevezett üres csomópont (amely nem tartalmaz sem URI-t, sem literált), de mindkettő egy erőforrást jelöl. Azokat az erőforrásokat amelyekre üres csomópontok utalnak, anonymous erőforrásoknak nevezzük. Ez annyit jelent, hogy nem azonosíthatók egyértelműen az RDF állításból. A predikátum egy URI, amely egy erőforrást is jelöl, és egy kapcsolatot reprezentál. Az objektum vagy egy URI, vagy egy üres csomópont, vagy pedig egy Unicode sztring literál.
A Szemantikus Web alkalmazásokban, és az eléggé népszerű RDF alapú alkalmazásokban, mint az RSS, az erőforrások nagyon gyakran URI-k amelyek kifelé mutatnak az internet felé, és ezáltal hozzáférhetővé tesznek külső információkat. Az RDF nincs korlátozva csak az interneten található erőforrásokra. Sőt, valójában a URI neveknek egyáltalán nem kell feloldhatónak lenniük. Például egy URI, ami a „http:” karaktersorozattal kezdődik, és témaként van jelen az RDF-ben, nem szükségszerűen egy olyan erőforrást jelöl, amelyet HTTP protokollal tudunk elérni, vagy hálózaton keresztül, hanem gyakorlatilag egy ilyen URI akármit jelölhet. Azonban érvényben van egy megállapodás, miszerint egy normál URI (amely nem tartalmaz #-et) amely 300as szintű választ küld, ha a GET kérést alkalmazzuk rajta, akkor azt külső erőforrásként kell kezelnünk és arra az oldalra kell mutatnia, amelyet a GET kérésben szerepeltettünk.
Ezáltal szükségeltetik, hogy az RDF készítői, és felhasználói tisztában legyenek az erőforrás azonosítókkal, illetve megállapodjanak róluk. Az ilyen megállapodások nem jellemzőek az

RDF-ben, bár léteznek közös használatban levő szótárak, mint a Dublin Core Metadata, amelyet részlegesen egy URI-hez rendeltek az RDF-ben való használathoz.

Szakterületi Glosszárium

Glosszárium alapelve, létrehozásának szükségessége

Bár léteznek már olyan szoftverek, amelyek alkalmasak a Szakterületi Glosszárium problémáinak megoldására, ezek használata ennek megoldására nagyjából olyan lenne mintha ágyúval lőnénk verébre. Ezeket a szoftvereket nem ilyen egyszerű feladatok elvégzésére találták ki. Továbbá a program tervezésekor figyelembe kellett venni az egyes szakterületekhez tartozó szakkifejezések magas számát. Ez a magas szám sajnos eléggé megpecsételheti egy olyan szoftver sorsát, amely teljes egészében a szemantikus web által használt szabványokra épül. A már létező ingyenesen használható szoftverek használata egy laikus ember számára meglehetősen nehézkes lehet, nem is beszélve arról, hogy nem fogja tudni kihasználni a szemantikus web, illetve a program által nyújtott előnyöket. Ezért véleményem szerint sokkal inkább életképes egy olyan szoftver, amely bár valóban nem használja ki tökéletesen a szemantikus web összes előnyét, de használhatóságban inkább az egyszerűség és az egyértelműség felé hajaz, ezáltal sokkal felhasználóbarátabb, mint például egy TM4L. Bár sokkal kevesebb funkciót is tartalmaz, de arra alkalmas lehet, hogy felébressze az igényt, illetve láttassa a felhasználókkal az előnyöket, amelyek ebben a technológiában rejlenek.

Ennek fényében tehát szeretnénk egy olyan programot készíteni, amely a felhalmozott tudást rögzíti az elektronikus térben, ezáltal tárgyiasítva azt. Ezt a tárgyiasított tudást aztán szakterületekre bontva tudjuk csoportosítani, illetve elérni, a szabványt úgy alkalmazva, hogy egy szakterület egy ontológia legyen.
A tárgyiasított tudást természetesen rögzíteni kell a web szerveren, ezt pedig erőforrás fájlok formájában fogjuk megtenni.

Az információtechnológián belül napjainkban kialakuló, és fejlődő szemantikus web technológia ezen tudás tárolásában és reprezentálásában nagy segítségünkre lehet. A Glosszárium elkészítésekor a két legfőbb irányelvnek a Topic Maps ISO szabványt, és ennek leíró nyelvét, az OWL szabványt tekintettem.

Miért egy új alkalmazás, miért nem egy meglévő?

Bár már léteznek megoldások a szemantikus web elemeinek reprezentálására, ezek a technológiák, egy ilyen „egyszerű” alkalmazáshoz túl kiforrottak.

A TM4L tökéletesen és pontosan tudná tárolni az összes elemét a tématérképnek, sőt beépített vizuális moduljával a komplett fát is meg tudja rajzolni szemléltetés céljából. Pontosan definiálva van a fájlformátuma, saját helyi adatbázissal rendelkezik, ha letölthetővé tennénk a teljes tématérképet XTM dokumentumként, teljesen alkalmas lenne a feladat ellátására

Az egyedüli probléma ezzel az alkalmazással, hogy túlságosan robosztus és sokrétű egy ilyen célra. Egy olyan alkalmazás tökéletes erre a célra, amelyet bárki gyorsan használni tud, nem szükséges előzetes telepítés, és mindig naprakész.

Ennek a feladatnak az ellátására egy könnyed kis program szükségeltetik, amely nem foglal sok helyet a merevlemezen, és úgy lehet használni akár egy értelmező szótárt. Ezen elv fényében született meg az elhatározás a Szakterületi Glosszárium létrehozására, amely képes ellátni ezt a feladatot.

Hasonló alkalmazások már régebb óta léteznek, amelyek ilyen célokat szolgálnak, sajnálatosan csak angol nyelven elérhetőek, és nem kifejezetten egy szakterületre koncentrálva.

Megfeleltetés a szabvány és a Glosszárium egyes elemei között.

A szabvány három fontos fogalomról tesz említést, amit én is gyakran fogok alkalmazni. Témák, előfordulások, illetve kapcsolatok, vagy asszociációk más néven. Fontos megfeleltetni a szabványban felsorolt elemeket, és azt, hogy ezek a program adatmodelljében hogyan jelennek meg.

Témák

A glosszáriumban a szakkifejezések típusokba vannak sorolva, aszerint, hogy melyik ágazatban használatosak. Egyes szakkifejezések megjelenhetnek akár több típusban is, mindegyik helyen mást jelentve. Ha nem egy szakterületi kifejezést tekintünk, akkor is

előfordulhat, hogy más kontextusban használva az adott szót, mást jelent, így erre lehetőséget kell biztosítanunk, így a legkézenfekvőbb az volt, hogy témának az ontológiában egy típust tekintsünk. Téma lesz például az ontológiában a „Számviteli Standard” nevű típus, és olyan szakkifejezéseket fog tartalmazni, mint például a „könyvvizsgáló”.

A témák segítségével máris egy rendezettebb struktúrát kaptunk, mintha az összes szakkifejezést egy helyre sűrítettük volna, nem beszélve arról, hogy nem tudtuk volna megkülönböztetni, hogy melyik kontextusbeli jelentésére vagyunk kíváncsiak.

Előfordulások

Ha már típusok szerint csoportba foglaltuk a szakkifejezéseket, itt az ideje, hogy megnevezzük őket, így tehát egy előfordulás, egy szakkifejezést jelent pontosan. Minden szakkifejezéshez tárolva vannak bizonyos tulajdonságok is. Ilyen tulajdonságok maga a szakkifejezés neve, a szakkifejezés teljes neve, a szakkifejezés UML szabvány szerinti alakja, a szakkifejezés, típusa, mint utalás arra, hogy melyik témának az előfordulása, természetesen a jelentése, valamint egy leírás rész, amely egy összetett tulajdonság lesz, arra való tekintettel, hogy ebben a tulajdonságban lesznek tárolva a kapcsolataik a többi szakkifejezésekkel, illetve külső erőforrásokkal.

Asszociációk

Az egyes szakkifejezések között fellelhető kapcsolatot hivatottak szolgálni, és arra utalnak, hogy az egyes kifejezések milyen kapcsolatban vannak egymással. A különböző szakkifejezések, néhány előre meghatározott kapcsolatban állhatnak egymással, amelyek később lesznek ismertetve. A szakkifejezés mellé tárolt kapcsolatinformációk segítségével tehát az előfordulások egymásra hivatkoznak, mint belső erőforrásra, de lehetőségünk van külső erőforrások jegyzésére is, ezeket URL linkek formájában tehetjük meg.

Egy szakkifejezés általános alakja

Egy szakkifejezés mint erőforrás kétféle lehet, belső erőforrás és külső erőforrás. Belső erőforrásnak nevezzük az összes olyan szakkifejezést, amely megtalálható a glosszáriumban, külső erőforrásnak pedig egy URL linket, amely egy honlapra mutat valahol a Web-en.
Az ilyen erőforrások közötti kapcsolatokat linkeknek hívjuk. Egy szakkifejezés esetében rengeteg belső link, és egy külső link található meg. Az itt alkalmazott link fogalom nem

egyezik meg a HTML ismeretek alapján kialakult link fogalmával, bár a honlapon URL-ekre mutató HTML linkek fogják reprezentálni a kifejezések közötti kapcsolatot.
Egy szakkifejezés megjelenítésének két része van:

- Definíciós rész: kötelezően megadandó, önmagában is definiálja a szakkifejezést
- Leírás rész: opcionális kiegészítő rész, amelyben a szakkifejezés kapcsolatát jelezhetjük egyéb szavakkal. Bár opcionális, de érdemes megadni, mivel ezen rész elhagyásával elvesztjük a szemantikus web technológia által nyújtott előnyöket

Egy, az adatbázisban tárolt szakkifejezés az alábbi tulajdonságokkal jellemezhető:

Definíciós rész:

- **Helyes alak:** Egy szakkifejezés helyes alakjának nevezzük a szakkifejezés azon formáját, melyet a beszédben is leggyakrabban használunk. Ezek az alakok vannak listázva a Glosszárium szókészletében.
- **Teljes alak:** A helyes alaknak egy formája, ahol a kifejezésben az esetleges rövidítéseket kiírjuk. Általában ritkábban használt, mivel az emberek nagy része szeret rövidítésekben beszélni.
- **UML alak:** A szakkifejezés helyes alakjának UML szabvány szerint megfogalmazott alakja. Szabvány szerint a kifejezés első betűje kis betű, és ha a helyes alakban szerepelte szóközök, ezek eltűnnek, és a szavak kis kezdőbetűi nagyra módosulnak. **Angol név:** Ha az angol szakirodalomban létezik megfelelője az adott szakkifejezésnek, az itt szerepelhet.
- **Típus:** Annak a témának a neve, amelynek az adott szakkifejezés előfordulása, vagyis a szakkifejezés kontextusa.
- **Definíció:** A szakkifejezés magyarázata, értelmezése, alkalmazásának területei.

Leírás rész:

Összetett tulajdonság, itt találhatóak meg a szakkifejezés kapcsolatai a többi szakkifejezéssel, tehát a hivatkozások a belső erőforrásokra, és a külső erőforrásra. A leírásnak több része van:

- **Szinonimák:** Más szakkifejezések, amelyeket alkalmazhatunk az adott szakkifejezés helyett, mert jelentésük vagy azonos, vagy nagyon hasonló
- **Megkülönböztető fogalom**
- **Tágabb fogalom**
- **Szűkebb fogalom**
- **Kapcsolódó fogalom:** Más szóhasználatban: **Lásd még**
- **Lásd:** az adott szakkifejezéshez tartozó összes nemdeszkriptor fogalom
- **Ellentétes fogalom**
- **Megjegyzések, feljegyzések:** A szakkifejezés használatára vonatkozó megjegyzés (kialakulásának, változásának története)
- **Idézetek**
- **URL:** Külső erőforrás, megadhatjuk, hogy mely weblapon találhatunk még információt az adott kifejezéssel kapcsolatban

Például lássuk a „glosszárium” szakkifejezést, hogyan jelenik meg a programban:

Glosszárium

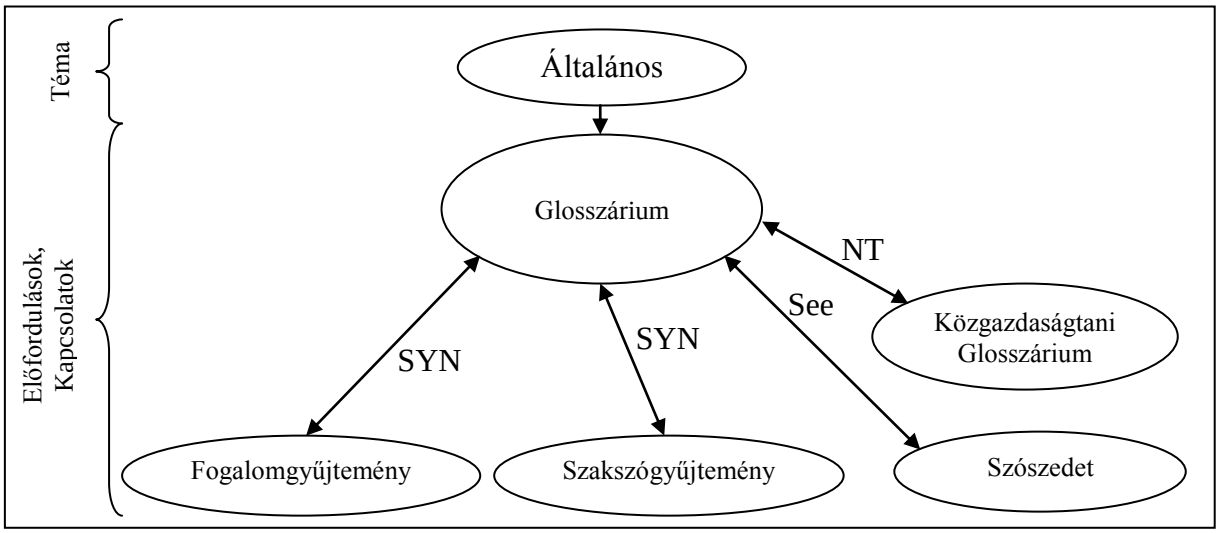
Helyes alak: Glosszárium
Teljes alak: Szakterületi Glosszárium
UML alak: szakterületiGlosszárium
Típus: Általános
Definíció: A szakterületi glosszárium fogalmakat, szakkifejezéseket, és magyarázatukat tartalmazó adattábla. Speciális tudásterületen használatos kifejezések listáját jelenti, ami a tudásterülettel kapcsolatos szavakat magyarázza.

Leírás

Szinonimák: Fogalomgyűjtemény, Szakszógyűjtemény
Megkülönböztető fogalom: -
Tágabb fogalom: -
Szűkebb fogalom: Közgazdaságtani Glosszárium
Kapcsolódó fogalom: -
Lásd: Szószedet
Ellentétes fogalom: -

Egy szó általános jellemzői a Glosszáriumban

Ez a szabvány szerint az alábbi módon néz ki:



6. ábra **A glosszárium szó megjelenése a szabványban**
Téma: Általános; Előfordulás: Glosszárium, Szószedet, stb.…; Kapcsolat: SYN, See, NT

Szakkifejezésre vonatkozó szabályok:

Meg kell fogalmaznunk néhány szabályt az ontológiában, hogy a szakkifejezések rengetege közt ne vesszünk el. Szabályokat és megkötéseket teszünk a Glosszárium tartalmára a könnyebb navigálhatóság, és az átláthatóbb megjelenés érdekében.

1. A szakkifejezések a Glosszáriumban abc sorrendben követik egymást
2. A szakkifejezések kezdőbetűje kisbetű, kivéve, ha tulajdonnév
3. Ha a szakkifejezés betűszó, akkor valamennyi betűje nagybetű
4. A szakkifejezéseket egyes számban írjuk kivéve, ha csak a többes számú alakját használjuk
5. A szakkifejezés értelmezéséhez kötelezően szerepelnie kell a definíciós résznek, a leírás rész teljes egészében opcionális.
6. Ha egy szakkifejezés szerepel a leírás részben, kötelezően szerepelnie kell a Glosszáriumban is kivéve, ha az egy külső erőforrás

Az Architektúra

A Glosszárium tervezésekor az első felmerülő kérdés az architektúra megválasztása. Legfontosabbnak kérdés természetesen az volt: elosztott rendszer legyen, vagy egyedülálló. Lássuk hát a Pro-t és a Kontra-t mindegyik megvalósítás esetén.
Először az egyedülálló rendszer:

Pro	Kontra
– Helyben tárolt adatbázis – gyors elérés: az elérési időnek gyakorlatilag csak a hardver gyorsasága szab határt – Internetkapcsolat nélküli működés – bár manapság eléggé ritka olyan hely ahol nincs internet elérés, de igény lehet rá	– Helyben tárolt adatbázis – ami az egyik oldalon előnyként szerepel, a másik oldalon hátrányként jelenik meg: nehézkes az adatbázis frissen tartása – A programmodulok frissen tartásának nehézsége, esetleges hibajavítások nehézsége – Programverziók kezelése: Az esetlegesen felmerülő programhibák

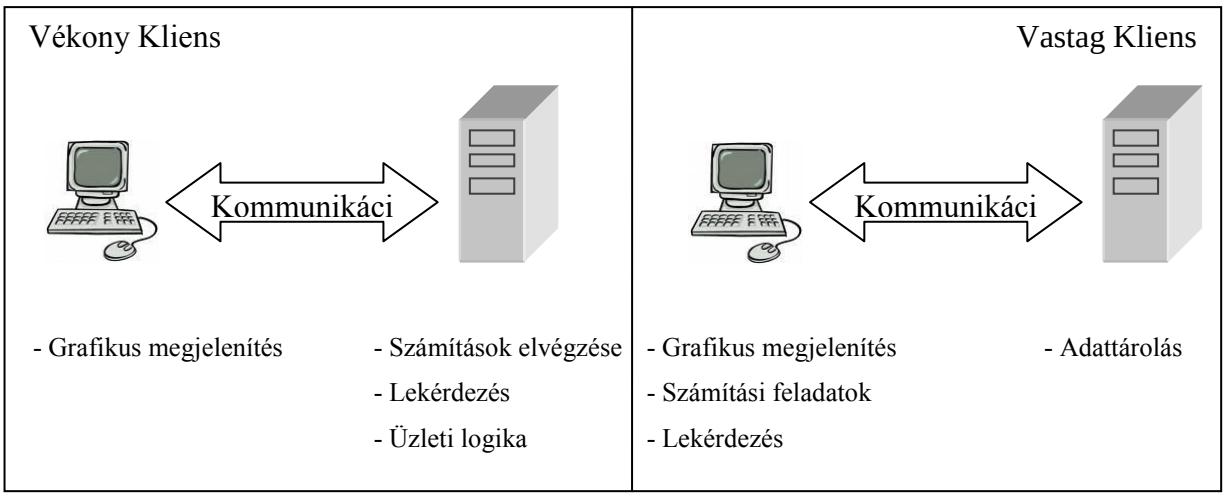
	kijavításával publikált új verziók használatára késztetés – Lemezterület igény: A szakkifejezések magas száma és tárolási módja miatt az adatbázis mérete eléggé megnőhet – Erőforrásigény: Egy kevesebb erőforrással rendelkező számítógép számára megterhelőbb feladat lehet futtatni egy ilyen alkalmazást – Platformfüggőség: Bár a Java Virtual Machine minden platformon elfut, mégis létezhetnek olyan kódolási nehézségek, illetve igénybe veendő rendszerszolgáltatások, amelyek platform specifikus megoldásokat igényelnek

És másodjára az elosztott rendszer, illetve a szerver-kliens architektúra:

Pro	Kontra
– A feladatokat elosztjuk a számítógépek között, így a kliensnek kisebb sokkal az erőforrásigénye, valamint egy esetleges hardverhiba bekövetkeztekor a szervert kicserélhetjük gond nélkül. – Az információk tárolása nem, vagy legalábbis minimális lemezterületet igényel. – Az adatbázis, valamint a Java Web Start technológia révén a kliens alkalmazás frissítése is könnyedén megoldható. – Jól elkülönülnek szerver-, és kliensfeladatok	– Internetkapcsolat megléte – Internetkapcsolatból fakadó késleltetés – Szerverközpontúság – Komplexitás

Mint azt láthatjuk az szerver-kliens architektúra inkább alkalmas ezen feladat ellátására, mint az egyedülálló architektúra, pusztán szerkezete miatt is alkalmasabb egy ilyen információcserére tervezett alkalmazás megvalósítására.

Így tehát eldőlt, hogy a szerver-kliens architektúrát alkalmazom a program megalkotásakor. Előnyben részesítettem a vékony kliens technológiát, hiszen teljesen feleslege lenne újabb feladatokkal halmozni a klienst, mint csak a grafikus megjelenítés.



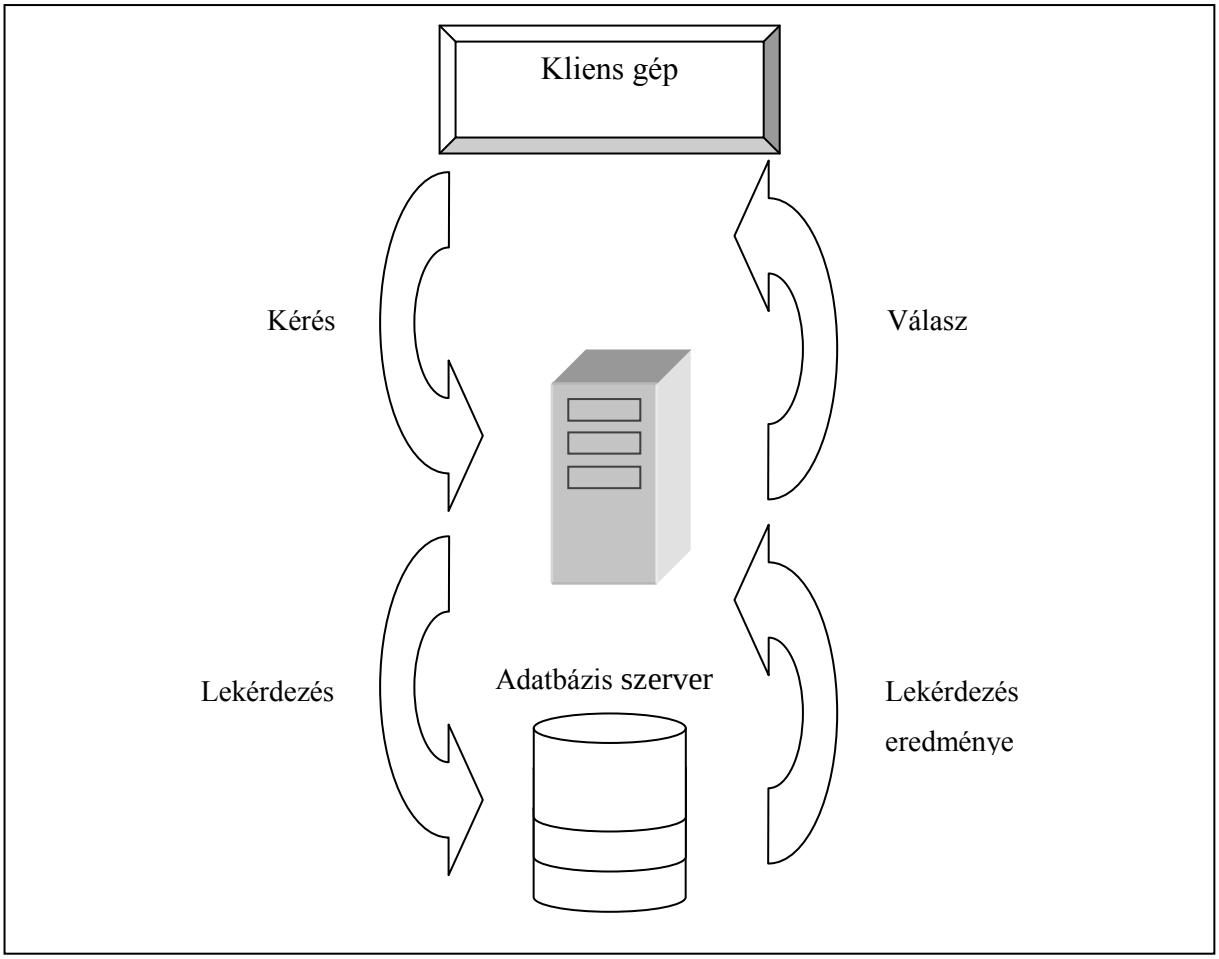
7. ábra **Vékony kliens és vastag kliens közti különbsége**

Első elképzelés szerint egy web szerver nyújtott volna szolgáltatást egy helyi gépen JVM alatt futó alkalmazás számára, ám ez ismét platformfüggetlenségi problémákat vetett volna fel, hiszen a kliensnek ismét több platformhoz kellett volna alkalmazkodnia.

Így úgy döntöttem, hogy a vékony kliens, annyira vékony kliens lesz, hogy nem egy önálló letöltendő alkalmazásként fog megjelenni, hanem weblapként.

Ezzel el lehet érni, hogy a lehető legminimálisabb erőforrást igényelje a program, és csak a grafikus megjelenítés legyen egy helyi gépen futó böngésző feladata.

Ez az alábbiak szerint működik:



8. ábra **Egy szakkifejezés lekérdezésének a menete**

A fenti rajzon szerepelnek olyan fogalmak, amelyek magyarázatra szorulnak:

- **Kliens gép:** A kliens gép, itt a kliens gépen futó böngészőt jelöli
- **Kérés:** A Kliens gépen futó böngészőben rákattintunk egy kifejezésre, amely egy HTTP GET kérést intéz a Web szerver felé
- **Válasz:** A Web szerver egy HTTP válasz formájában visszaküldi a kért adatot tartalmazó HTML lapot
- **Lekérdezés:** A Web szerveren futó Glosszárium alkalmazás lekérdezi az adatbázisból a szükséges adatokat

Szerializáció

A másik fontos eldöntendő kérdés az alkalmazásnál, az volt, hogy milyen szerializációt alkalmazzak az információ tárolására.
Topic Maps szabványt követve kézenfekvő lenne az XTM fájlformátum, amelyet már feljebb ismertettem. Ekkor egyetlen állományban tag-ekkel tagolva lettek volna elválasztva egymástól az egyes szakkifejezések, és a tulajdonságaik.
Ez a következőképpen nézett volna ki:

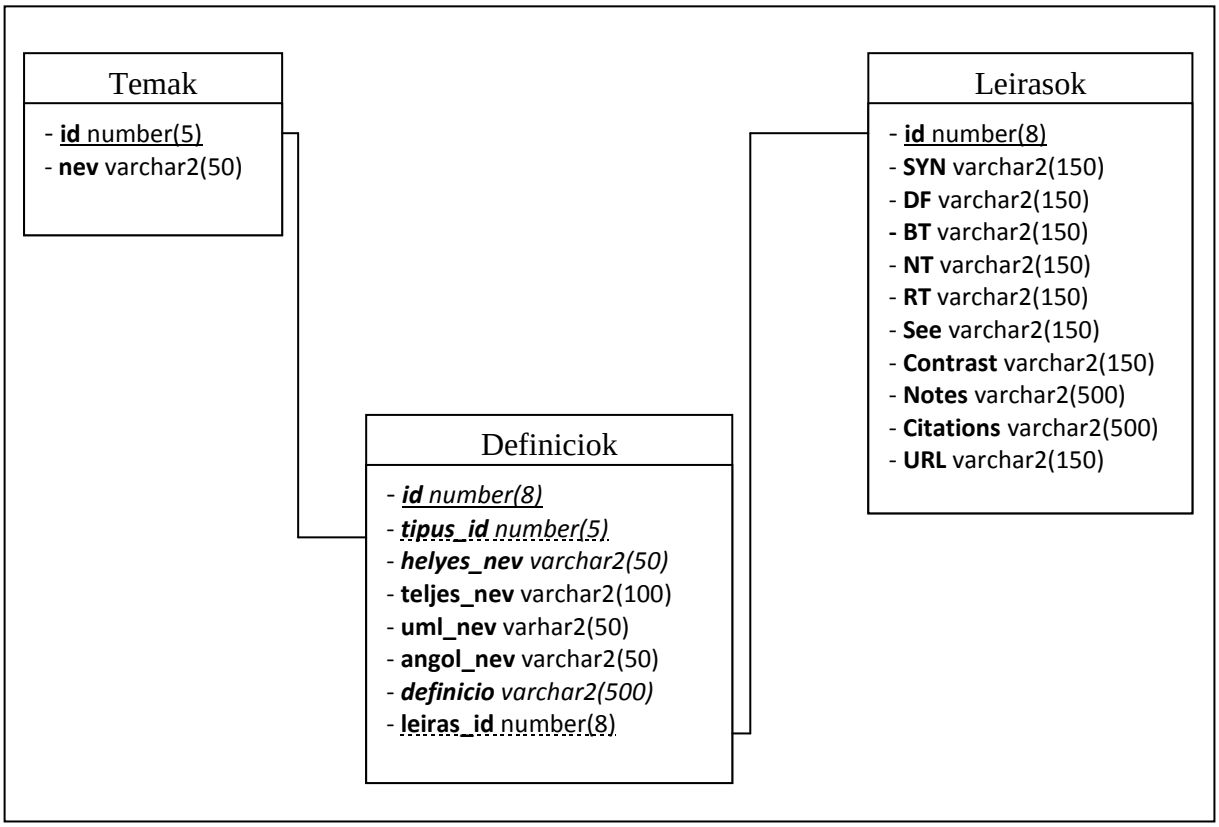
```
  <?xml version="1.0"?>
  <!DOCTYPE topicMap
            PUBLIC "-//TopicMaps.Org//DTD XML Topic Map (XTM) 1.0//EN"
                   "file://usr/local/home/gromit/xml/xtm/xtm1.dtd">

<topicMap>
      <topic id=”Általános”>

      </topic>

      …

      <topic id=”Glosszárium”>
        <instanceOf><topicRef xlink:href="#Általános"/></instanceOf>
      …
      </topic>
</topicMap>
```

9. ábra **Egy XTM dokumentum általános alakja**

Ám ez a szerializációs formátum csak néhány ezer kifejezésnyi méretig élhető. A fájlindexelés és a keresés problémája ezen felül már felüti a fejét. Ennyi kifejezés esetén ugyanis sokkal fejlettebb fájlolvasási, és kezelési technikákra lenne szükségünk, mint amit az operációs rendszer alapszinten nyújt. A keresés lenne a leghosszadalmasabb művelet, ugyanis semmilyen keresési algoritmust nem tudunk az állományon alkalmazni, csak úgy találhatunk meg egy kifejezést, ha végignézzük sorban az egész állományt. Ez a legrosszabb esetben azt jelentené, hogy az egész állományt végig kellene olvasni.
Ennek fényében az XTM állományt kiváltottam egy hasonló szerkezetű adatbázissal. A meglevő adatbázisrendszerek korszerű megoldásokat kínálnak ezekre a problémákra, csak el kell végeznünk néhány megfeleltetést az adatbázis elemei és az XTM állomány részei között.
Az egyes témákat például tárolhatjuk egy táblában, a szakkifejezés definíciós részét egy másik táblában, a leírás részt pedig egy harmadikban. Ezáltal jóval leegyszerűsödött a

rendezési és a keresési procedúra is, hiszen mindegyik egy SELECT segítségével könnyedén megoldható.

Az adatbázis tábláinak szerkezete tehát a következőképpen néz ki:



10. ábra **A tárolásra használt adatbázisséma**

Az adatbázisban az alábbi megszorítások érvényesek:

1. A „Temak” tábla elsődleges kulcsa az „id” mező
2. A „Definiciok” tábla elsődleges kulcsa az „id” mező
3. A „Definiciok” tábla külső kulcsa a „tipus_id” mező, amely a „Temak” tábla „id” mezőjére hivatkozik
4. A „Definiciok” tábla külső kulcsa a „leiras_id” oszlop, amely a Leirasok Tábla „id” oszlopára hivatkozik
5. A „Definiciok” tábla „tipus_id”, „helyes_nev”, „definicio” oszlopaira érvényes a NOT NULL megszorítás, azaz ezek egyike sem lehet kitöltetlen (vagyis NULL értékű).

Tehát a „Temak” tábla tartalmazza a típusokat, a „Definiciok” és a „Leirasok” pedig együttesen a szakkifejezést. A fenti ábra szemlélteti az egyes táblák közötti kapcsolatokat is. Ezzel a megoldással teljesen kiválthatjuk az XTM állományt. A Definíciók tábla helyes_nev

oszlopát és a leírások táblának a SYN, DF, BT, NT, RT, See oszlopait azért nem kapcsoltam össze, mert a leírás oszlopaiban több kifejezés is szerepelhet, és ez problémákat okozott volna.

A felhasznált technológiák

Most, hogy tisztában vagyunk a program elméleti szerkezetével, ismerjük meg kicsit azokat a technológiákat, amelyek a szoftvert működtetik. Ahhoz hogy futtatni tudjuk a programot, szükség van egy szerverre, amely tárolja magát a web alkalmazást, egy másik szerverre, amely az adatbázist tárolja, valamint magára a megjelenítendő HTML lapra, ahol az információ megtalálható.
Ezeket a feladatokat az alábbi alkalmazások látják el:

- A web alkalmazás: Apache Tomcat Servlet Container 6.0
- Az adatbázis: PostgeSQL 8.0

Ahhoz, hogy lássuk hogyan is jönnek létre azok a HTML lapok, amelyeket a böngészőben olvashatunk, meg kell ismernünk a Java Servleteket.

Java Servlet-ek

A Web gyors átalakuláson ment keresztül, egyszerű közlési eszközből kifinomult alkalmazási környezetté vált. A Web tartalmának egyre nagyobb része statikus HTML fájlok helyett dinamikusan generált lapokba került, melyek egyszeri megjelenés után eltűnnek. Sokan a webhelyek közül, különösen a nagyobb kiskereskedelmi helyek, melyeknek állandóan változó raktárakkal és árakkal kell dolgozniuk, teljes egészében dinamikusan generált lapokból állnak, egyetlen HTML fájlt sem tartalmaznak.

A dinamikusan generált tartalomnak sok előnye van: elmarad a HTML fájlok verzióellenőrzése, a lapok jobban elviselik a változó környezetet, a tartalmat minden olvasónál testre lehet szabni. Van azonban egy hátrány. A dinamikusan generált webhelyek sokkal több kifinomult programozási munkát kívánnak a tartalmuk megalkotásánál és átadásánál. Az új Java Servlet API és a Web szerverek segítséget nyújtnak azáltal, hogy könnyen kezelhető, nagyteljesítményű, rugalmas Java platformot adnak a weblapok röptében való összeállításához.

A szervletek Java nyelven készült olyan szerveroldali szoftverkomponensek, amelyek Java-futtató rendszerrel ellátott HTTP szerverekbe beágyazva kliensek részére különféle szolgáltatásokat nyújthatnak. Tehát egy szervlet egy speciális Java program, amely lehetővé teszi, hogy egy HTML oldal tartalmát dinamikusan változtassuk. Ennek fényében, ha egy kliens egy olyan HTML oldalt kér, amelyet dinamikusan kell létrehozni, akkor ezt a kérést delegálja a szerver a szervlet felé, az pedig legenerálja a kért oldalt.

Természetesen a szervletek nemcsak szerver szerepköröket tölthetnek be, például az említett három résztvevős kliens-szerver struktúrában a szervletek a középső, alkalmazásfüggő szerver komponensek szerepét is betölthetik úgy, hogy közben maguk is kliensei valamely háttéradatbázis elérését biztosító szervernek.

A glosszáriumban pontosan ez utóbbi szerepet töltik be. A szervlet JDBC-n keresztül kapcsolódik az adatbázisszerverhez, ahonnan egy SELECT utasítással lekérdezi a kért információkat, majd ezeket megfelelő formátumúvá alakítva és formázva visszaküldi válaszként a HTML lapot.

JDBC

A Java Database Connectivity, röviden JDBC egy API a Java programozási nyelvhez, amely az adatbázisokhoz való hozzáférést segíti. A JDBC definiálja az adatbázisok lekérdezéséhez és módosításához szükséges osztályokat és metódusokat. A JDBC célja a Java programokból történő SQL adatbázisok platform független elérésének megvalósítása. Ez annyit tesz, hogy elvileg lecserélhetünk egy adatbázis kezelő rendszert anélkül, hogy egy sor kódot is módosítanunk kellene a programunkban. Létezik tehát egy JDBC API melyen keresztül egységesesen kezelhetők a különböző típusú relációs adatbázis kezelő programok. Ezt a JDBC API-t a JDBC meghajtóprogramok valósítják meg, melyet általában az adott adatbázis kezelő programot előállító vállalat szállítja számunkra. Minden népszerűbb adatbázis kezelő programhoz létezik ilyen meghajtóprogram, ha mégsem, akkor rendelkezésre áll az ODBC-JDBC híd melyen keresztül már biztosan használható az adott adatbázis, feltéve ha rendelkezünk ODBC meghajtóprogrammal. A JDBC gyakorlatilag az ODBC (Open DataBase Connectivity) Java változata.

Apache Tomcat

A szervletek működtetéséhez szükségünk van egy kiszolgáló alkalmazásra, amely azokat képes kezelni, futtatni. Az Apache Tomcat pontosan ilyen célra kifejlesztett, úgynevezett szervlet konténer, amely lehetőséget ad szervletek, illetve JSP lapok kipróbálására. Teljes egészében java nyelven implementálták, forráskódja is elérhető. Minden különösebb beállítás nélkül a Tomcat-et elindítva, egy Coyote nevű http szerver lép működésbe, amely egy kliensről a http://localhost:8080/ hivatkozással, 8080 porton megszólítható. Web alkalmazások készítésekor az IDE becsomagolja az alkalmazás osztályait, és konfigurációs állományait egy „war" (Web Archive) archívumba, amely war file-t a Tomcat telepítési könyvtárának webapps mappájába kell elhelyeznünk. Ezek után a Tomcat auto-deploy funkciójának köszönhetően az alkalmazást kicsomagolja az archívumból, és a fenti címen a http://localhost:8080/Projektnév/Servletnév címet a böngészőbe beírva elérhetővé válik a szervlet. A Tomcat standalone módban, vagyis önálló szervlet konténerként fut.

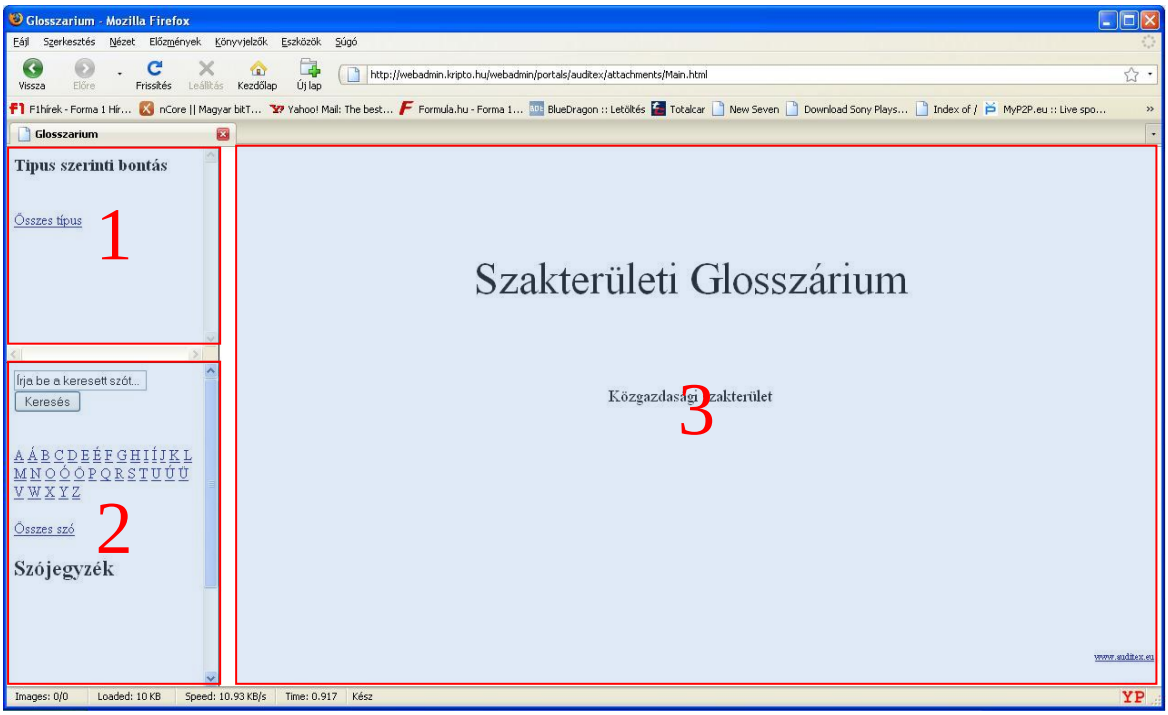
PostgreSQL

Az adatok tárolására a PostgreSQL adatbáziskezelőt használtam, amely egy nyílt forráskódú robosztus objektum-orientált adatbáziskezelő rendszer. Majdnem 15 éve fejlesztik folyamatosan, ennek eredményeképpen nagyon megbízható, és tárolási kapacitásban is több mint elegendő a 32 TB-nyi táblaméret.

A weblap szerkezete

A program tervezés során egyértelművé vált, hogy egy olyan kezelőfelületet kell alkotni, amelyet könnyen átláthatunk, nem bántó a szemnek a színvilága, egyszerű a kezelhetősége, és gyors a háttérben futó program működése, így az eredményt pillanatok alatt láthatjuk a képernyőn. Elsősorban akik használni fogják a programot, olyan emberek, akik nem szeretnének időt pazarolni az olyan dolgokra, mint a fájlbeolvasás, ezért is szerettem volna minél inkább a gyorsaság felé optimalizálni a programot.

A design ennek fényében eléggé a háttérbe lett szorítva, de ez a program használhatóságán semmit sem befolyásol. Így néz ki tehát a honlap:



11. ábra **A weblap szerkezeti felépítése**

Alapvetően 3 részre bontható a honlap. A bal oldalon találhatóak a kifejezések közötti böngészésre szolgáló frame-ek, a képernyő jobb oldalát és így nagyobb részét pedig maga a megjelenítési frame foglalja el.

Az 1-es számmal jelzett rész a típusjegyzék, amely az adatbázisban szereplő összes típust (azaz a „Temak” táblában szereplő sorokat) kilistázza abc sorrendben. Ennek segítségével szűkíteni tudjuk a keresendő kifejezést típus szerint. Az itt szereplő linkek kattintásra a szójegyzékben levő szavakon egy szűrést hajtanak végre. A háttérben működő adatbázis miatt ez a művelet egy egyszerű SELECT utasítás segítségével elvégezhető. Ehhez viszont el kell küldenünk a kattintott linkben a típusnak az azonosítóját. Az utasítás pedig az alábbi módon néz ki:

```
SELECT id, nev
FROM Tipusok
WHERE tipus_id = linkben_kozolt_id
```

Bemutató SELECT utasítás: típusok listázása

Megtalálható egy olyan link is, amely az összes típushoz tartozó kifejezést kilistázza, ehhez természetesen egy másik SELECT szükséges, amely ugyanígy néz ki csak WHERE feltétel nélkül.

A 2-es számmal jelölt rész a Szójegyzék rész, itt találhatóak meg a kifejezések az adatbázisban. A könnyebb navigálhatóság, és a keresések megkönnyítése érdekében itt is találunk néhány szűrési lehetőséget.
Ezek a kezdőbetű szerinti szűrések, és a keresés. A kezdőbetű szerinti szűréskor értelemszerűen csak az adott kezdőbetűvel kezdődő szakkifejezéseket fogja a program kilistázni, míg a keresésnél csak a beírt karaktersorozattal kezdődőket. Természetesen itt is jeleznünk kell a linkben, hogy milyen információra lesz szükségünk. Az ehhez tartozó SELECT utasítás:

```
SELECT id, helyes_nev
FROM Definiciok
WHERE helyes_nev LIKE lower(kapott_string)+'%' AND
tipus_id=kapott_id
ORDER BY helyes_nev;
```

Bemutató SELECT utasítás: szakkifejezések szűrése, keresése

A 3. rész pedig a megjelenítési rész, itt olvashatjuk a kifejezés definícióját, tulajdonságait, kapcsolatait a többi kifejezéssel, ezért érthetően nagyobb szerepet kapott a képernyő kitöltésében.
A Szójegyzékben található akármelyik szakkifejezésre kattintva a megjelenítési frame-ben fog megjelenni az összes információ, ami rendelkezésre áll. Egyedül itt érezhetünk egy pici hátrányt az adatbázisban való tárolásból, ugyanis az itt alkalmazandó SELECT utasítás kissé bonyolultabb, mint az előzőekben, ugyanis össze kell kapcsolni az összes táblát, mivel az összes rendelkezésre álló információt meg kell jelenítenünk.

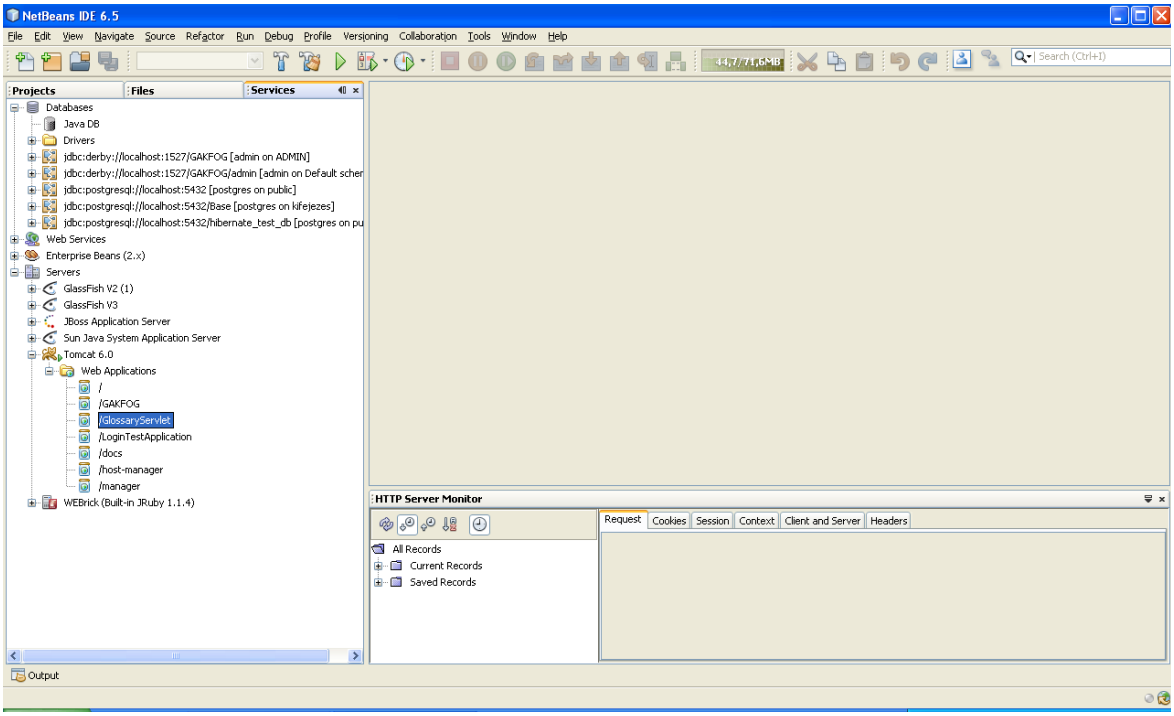
```
SELECT d.id, d.helyes_nev, t.nev, d.teljes_nev, d.uml_nev,
d.angol_nev, d.definicio, l.SYN, l.DF, l.BT, l.NT, l.RT,
l.See, l.Contrast, L.Notes, l.Citations, l.URL
FROM Temak t, Definiciok d, Leirasok l
WHERE t.id = d.tipus_id AND d.leiras_id = l.id AND d.id =
linkben_kapott_id;
```

Bemutató SELECT utasítás: kifejezés listázása

Ezzel a SELECT utasítással le tudjuk kérdezni az összes rendelkezésre álló információt. Ennek weblapra generálása a program feladata.

A program elkészítése

A program megalkotásához NetBeans 6.5-ös fejlesztői rendszert használtam, amely teljes mértékben támogatja mind a Java alapú alkalmazások fejlesztését, mind a tesztelésüket. Kezelőfelületének köszönhetően könnyen tudunk az IDE alól adatbáziskapcsolatot létrehozni a PostgreSQL adatbázisszerverrel, és kezelni a táblák tartalmát, valamint segítségével a Tomcat Servlet Container-t is vezérelhetjük, megnézhetjük, milyen Web alkalmazások vannak tárolva rajta, valamint akár újra is indíthatjuk a szervert.



12. ábra **NetBeans IDE**

Felül az adatbáziskapcsolatokat láthatjuk, alul pedig a Tomcat alá berakott Webalkalmazásokat

Futtatáskor a NetBeans elindítja a Tomcat szervert, ha még nem futna, majd bemásolja a projektből létrehozott war állományt a Tomcat web alkalmazásai közé, és az alapértelmezett honlapot megnyitja a böngészőben.

A Glosszárium alkalmazásban ugyan van egy alapértelmezett kezdőlap, de ezt nem fogjuk használni, hiszen a szervletek fogják azokat a szolgáltatásokat nyújtani, amikre szükségünk van.

Webalkalmazás létrehozása NetBeans alatt.

NetBeans-ben a Web alkalmazás projektet a Java Web kategória alatt találjuk meg. Az alkalmazás nevének és tárolási helyének beállítása után lehetőségünk nyílik annak a webszervernek a kiválasztására, ahol az alkalmazásunk futni fog majd, valamint a verziót is kiválaszthatjuk, ha esetleg régebbi rendszerben szeretnénk dolgozni.
A Context Path opciónál beállíthatjuk az alkalmazásunk elérési útját, vagyis az URL címét, amelyen majd a későbbiekben el kívánjuk érni.
A következő ablaknál kiválaszthatjuk az alkalmazni kívánt keretrendszert (Framework), amelyek komplex megoldásokat kínálnak a webprogramozás során felmerülő gyakori problémákra.

Szervletek létrehozása:

A Honlap minden egyes elemét Szervletek fogják generálni. A szervletek alapkoncepciója, hogy HTML kódot illeszthetünk a Java kódba.

Minden szervletnek két metódusa van alapértelmezetten, ezek a GET és a POST. Amikor egy HTML lapon meghívjuk a szervletet, és feltétlenül szeretnénk választ kapni a kérésünkre, akkor a GET metódusát hívjuk meg a szervletnek, ekkor automatikusan válaszban megkapunk egy HTML oldalt. A POST metódus akkor használjuk, ha csak el szeretnénk küldeni a szervletnek bizonyos információkat az URL-be ágyazva.

A fejlesztői IDE-ben a szervletet a Web kategóriában találjuk, majd kiválasztásakor a varázslóban megadhatjuk, hogy melyik csomagba szeretnénk a szervletet elhelyezni, majd ezek után az URL elérési útját is megadhatjuk.

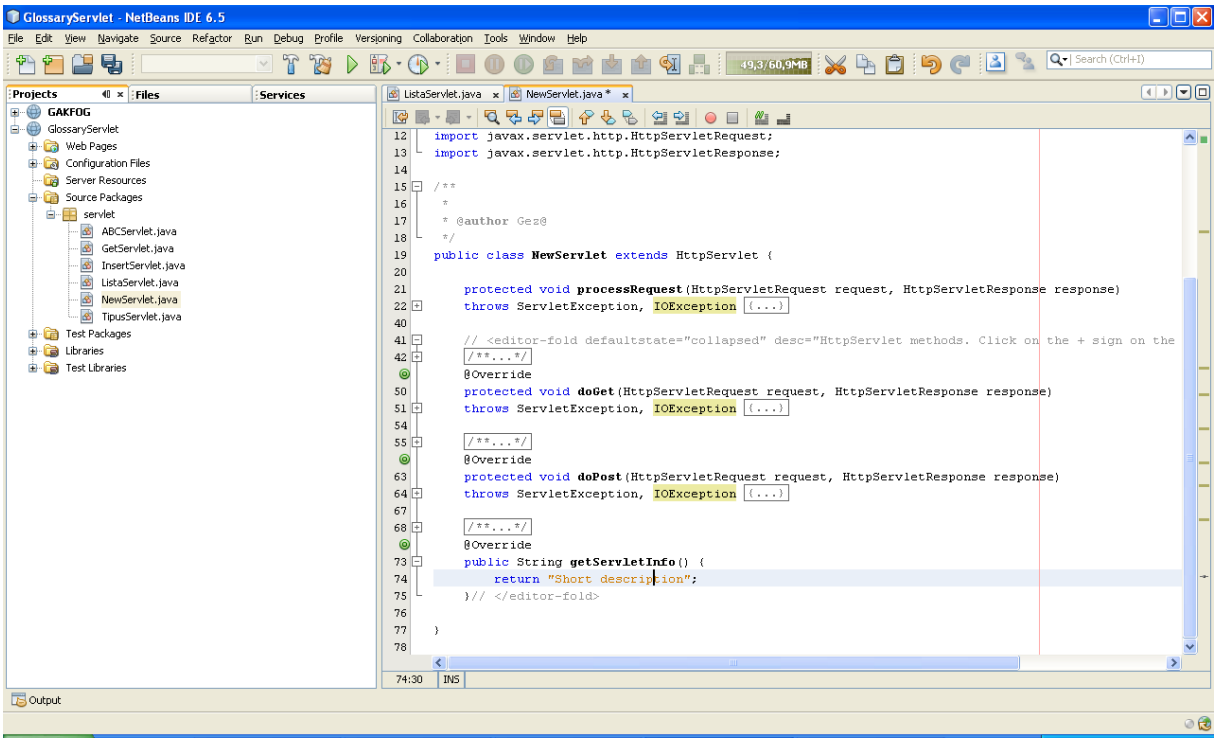
Ha a fentieket beállítottuk, akkor létrejön a kívánt file, és lehet implementálni.

Szervlet létrehozásakor a NetBeans létrehozza nekünk a doGet és a doPost metódusokat, amelyekhez alapértelmezetten hozzárendel még egy metódust, amit mindegyik helyen meghív, ez pedig a processRequest.

Ilyen módon, ha nincs szükségünk mindkét megvalósításra a szervletnél, mindegyik metódust implementálhatjuk egyszerre.

Mikor implementáljuk a metódusokat, a szervlet alapértelmezett kimenetére írunk, amely egy HTML lap lesz, így out.println() hívásokkal HTML tag-eket kell írnunk, úgy mintha egy Weblapot jegyzettömbben szerkesztenénk.

A processRequest metódusba beírt utasításoknak mindig try-catch blokkban kell lenniük, hiszen nem megengedhető egy alkalmazásban, hogy egy esetlegesen fellépő kivétel miatt leálljon a működésük.



13. ábra **Újonnan létrehozott szervlet NetBeans-be**

TipusServlet

A TipusServlet az adatbázisban tárolt típusokat listázza a kimenetére, amelynek megjelenését az alatta elhelyezkedő frame-ben fogjuk látni, implementációs része pedig a következőképpen néz ki:

```
Class.forName("org.postgresql.Driver");
String url = "jdbc:postgresql://localhost:5432/base";
Connection con = DriverManager.getConnection(url, "postgres", "");
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet srs = stmt.executeQuery("select nev, id from Tipusok;");


out.println("<html>");
out.println("<head>");
out.println("</head>");
out.println("<body>");
out.println("<h3>Tipus szerinti bontás</h3>");


out.println("<br><a href =
\"http://localhost:8081/GlossaryServlet/ListaServlet?tipus=\"\"
target=\"bottomFrame\">Összes típus</a>");
while (srs.next()) {
      out.println("<br><a href =
\"http://localhost:8081/GlossaryServlet/ListaServlet?tipus=" +
srs.getString("id") + "\" target=\"bottomFrame\">" + srs.getString("nev")
+ "</a>");
}


out.println("</body>");
out.println("</html>");
```

Négy részre tagolható a szervlet felépítése:

- **Adatgyűjtés:** Az adatgyűjtő részben létrehozzuk a JDBC kapcsolatot az adatbázisszerverrel.

  Ahhoz hogy a kapcsolat létrejöjjön kell nekünk egy Connection objektum. Az előtte levő Class.forName utasítással referenciát szerzünk a kapcsolathoz szükséges Driverre, amelyet hozzá kell adnunk a projektünk Lib könyvtárához az alkalmazott külső eszközök közé. NetBeans-ben ezt a projekt tulajdonságainál a Lib menüpont alatt tudjuk megtenni. A Driver tartalmazza a kapcsolódáshoz szükséges információkat, minden adatbázisszerverhez a gyártó általában biztosít saját JDBC Drivert.

  A Connection objektumnak három paramétere van, első az URL ahova csatlakozni szeretnénk, második a csatlakozáshoz szükséges felhasználónév, a harmadik pedig a jelszó.

  Ha létrejött a kapcsolat, akkor szükségünk van egy Statement objektumra a lekérdezés elvégzéséhez, valamint az eredmények tárolásához egy ResultSet-re.

  A Statement executeQuery metódusával tudjuk végrehajtani a lekérdezést.
- **HTML fejrész:** A kimenetre írjuk a HTML lap fejrészének kezdő tagjét (<HTML>) és a fejrészt tartalmazó két tag-et, a <HEAD>-et és a </HEAD>-et. Azt figyelembe véve, hogy ennek a HTML lapnak a fejrészét a böngésző nem fogja megjeleníteni, a fejlécbe nem írtam semmit. Ezt követően a <BODY> tag-et írjuk a lapra, amely a HTML lap törzsének a kezdését jelöli, és egy olyan címet amelyet már látni is fogunk.
- **Listázó rész:** A listázó részben a linkek generálását végzi el a szervlet, tehát feldolgozza a ResultSetben levő adatokat, és URL linkeket generál a forrásból.

  Mielőtt azonban végigmenne a ResultSet-en, ír egy olyan linket a kimenetére, amely az összes típust listázza. Ezek után egy while ciklussal végigiterál a ResultSet-en és minden egyes benne szereplő elemből létrehozza a linket. A linkben a target jelző mondja meg, hogy melyik frame-ben szeretnénk, hogy az eredmény megjelenjen. A generált linkben szereplő URL cím egy másik szervlet címe, amely számára beágyazunk egy azonosítót, annak a típusnak az azonosítóját amelyiket kilistáztunk.
- **HTML lezáró rész:** A HTML fejrészben szereplő még nem lezárt tag-ek záró tag-jei szerepelnek itt, tehát a </BODY> ami alap törzsét zárja le, valamint a </HTML> amely az egész lapot

A többi szervlet ugyanilyen alapon épül fel és működik, ugyanezeket a feladatokat hajtja végre hasonló módon, ezért a továbbiakban a feladataikat ismertetem.

ListaServlet

A ListaServlet fogja a TipusServlet által az URL-be ágyazott azonosítójú összes kifejezést kilistázni a kimenetén.

A keresés könnyítése érdekében tartalmaz egy HTML Form-ot a lap törzsének legelején, amely egy POST hívással a beírt kifejezést átadja az ABCServletnek. Ezen felül szintén a kapott találatok szűkítése érdekében a felsorolt kifejezéseket szűrhetjük kezdőbetű szerint.

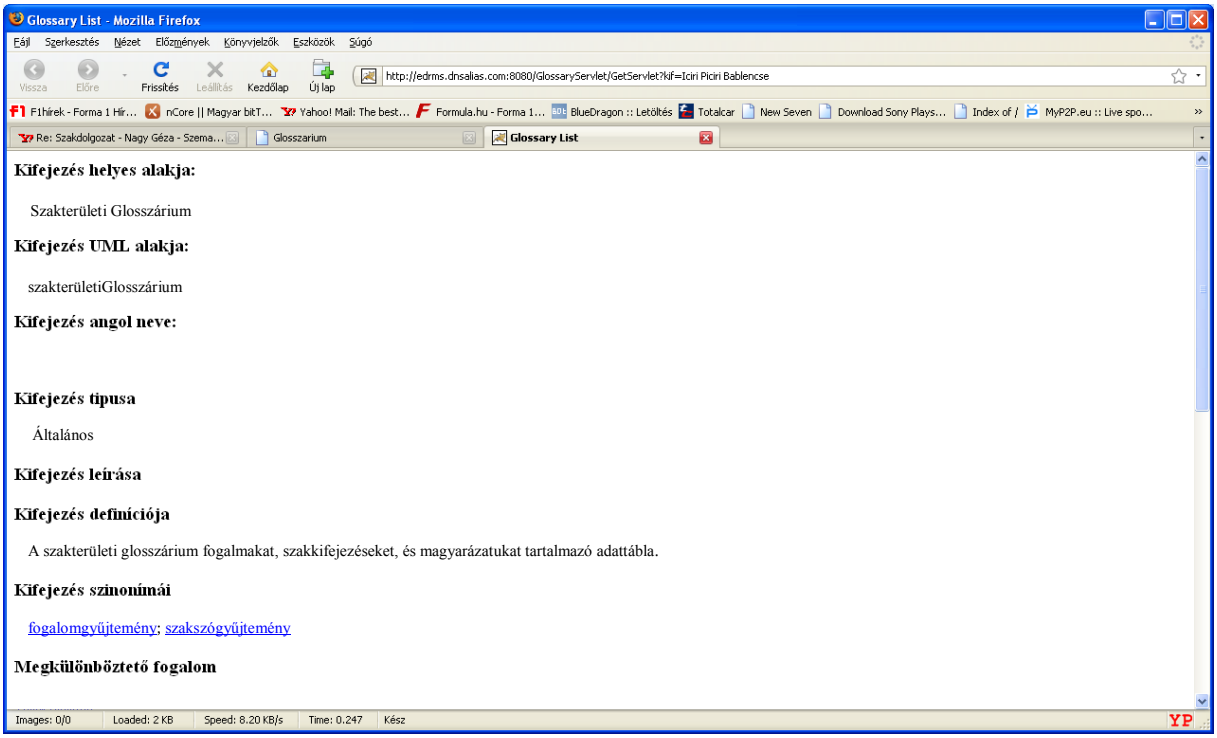
```
out.println("<form name=\"input\" action=\"ABCServlet\"
method=\"post\"><input name=\"betu\" type=\"text\" value=\"&Iacute;rja be
a keresett sz&oacute;t...\" /> <input name=\"Keres&eacute;s\"
type=\"submit\" value=\"Keres&eacute;s\" /></form><br/>");
      String abc = "AÁBCDEÉFGHIÍJKLMNOÓÖPQRSTUÚÜVWXYZ";
      for (int i = 0; i < abc.length(); i++){
            out.print("<a
href=\"localhost:8081/GlossaryServlet/ABCServlet?id=”+kapott_id+”betu="+a
bc.charAt(i)+"\" target=\"bottomFrame\" >"+abc.charAt(i)+"</a> ");
      }
```

A kereső form, és a kezdőbetű szerinti keresés kimenetre írása

A while ciklusban legenerált linkek egy másik szervletre mutatnak, amely az adott kifejezést fogja kilistázni, de ehhez szükség van a kattintott kifejezés azonosítójának beágyazására.

GetServlet

Ez a szervlet végzi el a kifejezések kilistázását a jobb oldalon levő fő frame-be (mainFrame). A linkben kapott azonosító által meghatározott szakkifejezést lekérdezi a Definiciok táblából, lekérdezi a hozzá tartozó leírást, valamint a Tipusok táblából a szakkifejezés típusát.



14. ábra **A GetServlet eredmény**

A leírásban szereplő információkból **a** weblapra generálja a kifejezések közötti kapcsolatokat reprezentáló linkeket is, amelyeket szintén ebben a frame-ben nyit meg.

ABCServlet

Az ABCServlet a keresésért, és a kezdőbetű szerinti szűkítésért felelős. Elsősorban meghatározza a lapra listázott szakkifejezéseket, majd ezeken végez el szűréseket.

A szűrést a SELECT utasításba ágyazva teszi meg, a lekérdező utasítás WHERE feltételébe illeszti a kulcsszót, ami alapján keresni szeretnénk. Így lehetséges, hogy egy szervlet képes ellátni a kezdőbetű szerinti szűrést és a keresést egyaránt.

A keret HTML oldal felépítése

Ahhoz hogy a kért információkat meg tudjuk jeleníteni, szükség van egy keret HTML oldalra, amely tartalmazza, hogy a képernyő melyik felén milyen méretarányban, melyik szervlet kimenete jelenjen meg.

Ezt HTML frame-ek, vagy más néven keretek segítségével érhetjük el. A kereteknek három fontos tulajdonsága van:

- meg tud jeleníteni egy önálló HTML dokumentumot
- rendelkezhet névvel, amely segítségével hivatkozhatunk rá
- automatikusan méretezi önmagát az ablak méretének változása során

A HTML frame-ekkel tagolni tudjuk a weboldalainkat sorok vagy akár oszlopok szerint is, mindegyik frame-be más weblapot megjelenítve.
A keretoldal kódja a következőképpen néz ki:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset = Windows-
1250" />
<title>Glosszarium</title>
</head>


<frameset rows="*" cols="233,*" framespacing="0" frameborder="yes"
border="1">
    <frameset rows="40%,*" frameborder="yes" border="1" framespacing="0">
      <frame
      src="http://edrms.dnsalias.com:8080/GlossaryServlet/TipusServlet"
      name="topFrame" scrolling="yes" noresize="noresize" id="topFrame"
      title="topFrame" />
      <frame
      src="http://edrms.dnsalias.com:8080/GlossaryServlet/ListaServlet?ti
      pus=" name="bottomFrame" id="bottomFrame" title="bottomFrame" />
    </frameset>


    <frame src="begin.html" name="mainFrame" id="mainFrame"
    title="mainFrame" />
</frameset>
<noframes><body>
</body>
</noframes></html>
```

Keret oldal HTML kódja

A lap feje általános információkat tartalmaz arról, hogy milyen szabványt használunk, milyen kódolást, és hogy konkrétan milyen típusú információ fog szerepelni a lapon.

Az első frameset tag-el két oszlopra osztottam a lapot, majd a benne szereplő másodikkal további két sorra. Ezáltal létre lett hozva az összes szükséges keret, a frame tagek src mezőjében pedig megmondtuk, hogy melyik szervletnek a kimenetét szeretnénk abban a frame-ben látni, illetve melyik előre létező HTML lapot, a name mező kitöltésével pedig elneveztem a frame-eket, ezáltal már tudok a linkek target mezőjében hivatkozni rájuk.

A HTML lap végén szereplő noframes opció segítségével megadhatjuk, hogy ha a böngésző amivel megnyitják az adott lapot, nem támogatja a frame-eket, akkor ebben az esetben mit jelenítsen meg a böngésző.

Összefoglalás

A mai napig nem létezik olyan szoftveres megvalósítása a szemantikus webnek, amely teljes mértékben lefedné annak mindenre kiterjedő részleteit. Valójában ez a program is csak töredékét mutatja annak, amire képesek lehetnénk egy komoly szemantikus web alkalmazással, és amiket modellezhetnénk vele.

A mai fejlődő világunkban az internet annyira nagy szerepet játszik, hogy szinte észrevétlenül belopta magát az emberek hétköznapjaiba, és lassan részévé válik a modern életvitelnek.

Azt a rengeteg felhalmozott tudást, amelyet besűríthetünk egyetlen weboldalba, képtelenség lenne rendszerezés nélkül kordában tartani. Ahogyan a technológia fejlődik, úgy halad előre a szemantikus web is a maga útján, sorra jelennek meg az olyan eszközök, amelyekkel könnyedén implementálhatunk egy RDF struktúrát, vagy egy Tématérképet. A szemantikus web folyamatosan szivárog be az internet felhasználók mindennapjaiba, sokszor úgy, hogy fogalmuk sincs arról, hogy amit használnak, az tulajdonképpen milyen elvek alapján működik.

Egy ilyen egyszerű kis alkalmazás is felkeltheti az érdeklődést ez iránt a technológia iránt, melyet folyamatosan csiszolnak, és amely egy nap egyeduralkodóvá válhat a tudás számítógépek általi reprezentálásában.

Irodalomjegyzék

1. Jack Park és Sam Hunting
   XML Topic Maps: Creating and Using Topic Maps for the Web
   Paperback - 2002

2. Richard Widhalm és Thomas Mück
   Topic Maps: Semantische Suche im Internet
   Kindle Edition - 2001

3. John Davies, Dieter Fensel & Frank van Harmelen
   Towards the Semantic Web, Ontology-driven knowlage management
   Wiley - 2002

4. Gottdank Tibor
   Szemantikus web - Bevezetés a tudásalapú Internet világába
   ComputerBooks Kiadó - 2005

Felhasznált internetes forrásanyag:

- *A W3C (World Wide Web Consortium) hivatalos honlapja*
  http://www.w3.org/
- *W3C Semantic Web Activity, a W3C és a Szemantikus Webbel*
  http://www.w3.org/2001/sw/
- *Dokumentumkezelési tanácsadás, iratkezelés*
  http://www.docuworld.hu/
- *Az OWL Web Ontológia Nyelv - Szemantika és absztrakt szintaxis*
  http://www.w3c.hu/forditasok/OWL/REC-owl-semantics-20040210/syntax.html
- *The DARPA Agent Markup Language honlap*
  http://www.daml.org/
- *Ontology Inference Layer (OIL) felépítése*
  http://www.ontoknowledge.org/oil/
- *topicmaps.org - An Independent Consortium Operating For Public Benefit*
  http://www.topicmaps.org/

- *Kovex KFT, Topic Maps kutatások*
  http://www.kovex.hu/kovex/topicmap.html
- *W3C RDF-el foglalkozó honlapja*
  http://www.w3.org/RDF/
- *Java Szervlet Technológiák*
  http://java.sun.com/products/servlet/